Geometric Properties of Learned Representations

by

Tongzhou Wang

B.A. Computer Science and Statistics
University of California, Berkeley (2017)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2022

Author ....
Department of Electrical Engineering and Computer Science
August 26, 2022

Certified by..
Phillip Isola
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Certified by..
Antonio Torralba
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by ....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Geometric Properties of Learned Representations

by

Tongzhou Wang

Submitted to the Department of Electrical Engineering and Computer Science
on August 26, 2022, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

In machine learning, reprensentation learning refers to optimizing a mapping from data to some representation space (usually generic vectors in $\mathbb{R}^d$ for some pre-determined d much lower than data dimensions). While such training often uses no supervised labels, the learned representations have proved very useful for solving downstream tasks. Such successes sparkled an enormous amount of interests in representation learning methods among both academic researchers and practitioners. Despite the popularity, it is not always clear what the representation learning objectives are optimizing for, and how to design representation learning methods for new domains and tasks (such as reinforcement learning). In this thesis, we consider the structures captured by two geometric properties of learned representations: invariances and distances. From these two perspectives, we start by thoroughly analyzing the widely adopted contrastive representation learning, uncovering that it learns certain structures and relations among data. Then, we describe two new representation learning methods for reinforcement learning and control, where they respectively capture the optimal planning cost (distance) and the information invariant to environment noises.

Thesis Supervisor: Phillip Isola
Title: Associate Professor of Electrical Engineering and Computer Science

Thesis Supervisor: Antonio Torralba
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I would like to first thank my thesis advisors, Professor Phillip Isola and Professor Antonio Torralba, for their continued support, encouragements, and mentorship throughout my time here at MIT. I am extremely grateful for the opportunity of working with and learning from them.

Many thanks goes to my collaborators near and far, some of whom I have yet met in person due to the pandemic. It is truly an amazing journey working with each of you. Thank you to Jun-Yan Zhu, Alyosha Efros, George Cazenavette, Yuandong Tian, Simon S. Du and Amy Zhang for the wonderful collaborations and for always opening to disucssing new ideas and more. I also learned tremendously from and thoroughly enjoyed collaborations here at MIT with David Bau, Lucy Chai, Minyoung Huh, Manel Baradad and Jonas Wulf. I am greatly thankful to them. Special thanks goes to Steven Liu, Jingwei Ma and David Wu for trusting me with their valuable undergraduate research opportunities, teaching me many new knowledge, and helping me become a better researcher and mentor.

Among many other things, two have supported me the most during the most stressful hours of this journey—caffeine and friends. I appreciate Broadsheet for their fantastic coffee, Bluebottle for their fancy coffee, Flour cafe for their affordable and accessible coffee, Passenger for their tasty beans, Counter Culture for their reliable and consistent beans, and Vigilante for supplying caffeine during the darkest COVID-19 days. I am extremely lucky to have many great friends here at MIT accompanying me and giving me hope through this time—Wei-Chiu Ma, Ching-Yao Chuang, Yen-Chen Lin, Lucy Chai, Minyoung Huh, Caroline Chan, Hyojin Bahng, Yonglong Tian, Xavier Puig, Manel Baradad, Joanna Materzynska, Shuang Li,and many more.

Finally, my deepest appreciation goes to my parents, Wenjuan and Guangbin, my brother, Andrew, my girlfiend, Jiaxi Chen, and our fluffy family members, Mochi, Tabby and Kiwi, for their endless love, company, and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Representation learning methods have gained enormous attention in machine learning research [97, 119, 90, 71, 63]. In such methods, an encoder function is optimized to map each data samples to a high-dimensional vector, which is called the latent or the representation of that sample. By learning on (usually) large quantities of unlabeled data, such methods enjoy the benefits of avoiding laborious and expensive human labelling, and have proven to show strong empirical results on downstream tasks, where particular tasks on learned on top of the learned representations (e.g., a linear classifier trained on top features from a deep image encoder). Across computer vision (CV) [71], natural language processing (NLP) [90], and reinforcement learning (RL) [97, 63, 121], such methods are hugely successful and become increasingly popular.

These methods are often motivated as capturing some information / relation of data [149, 172, 7]. However, in some cases, it is not well understood what exact relation is captured in learned representations [158], and which structure we should capture for different tasks (e.g., reinforcement learning).

For example, contrastive representation learning [119] currently enjoys wide empirical usage and popularity, but the understanding and analysis of representation learning methods are rather lacking, and have only recently begin to catch up [6, 154]. CV and NLP representation learning ideas were directly adopted in reinforcement learning and control [119], but do not always find similar successes [97].

In this thesis, we propose to understand and develop representation learning algorithms by looking at the induced geometric properties on the learned representations. In particular, we focus on two properties: (1) the invariances captured by the many-to-one representation mappings and (2) the structures captured by the distance in the representation space.

From such perspectives, we are able to

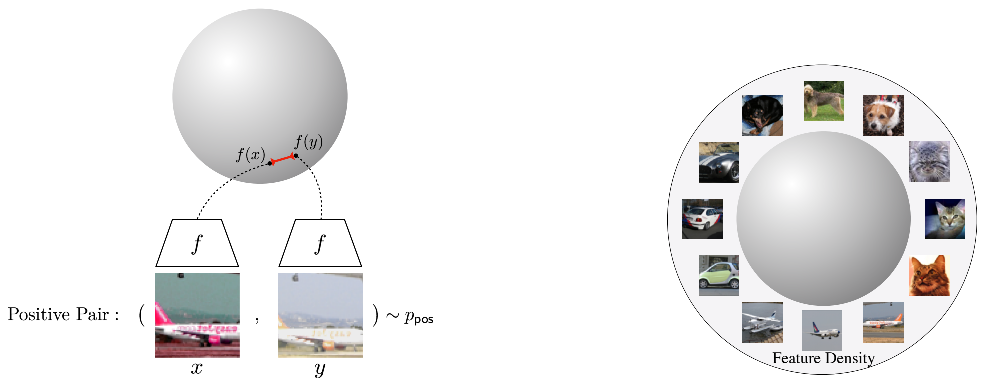
1. Analyze one of the most popular representation learning methods for vision and language, contrastive learning, via alignment (i.e., invariance) and uniformity on the hyperspherical representation space;
2. Propose a novel representation learning technique for RL and general quasimetric structures, where the quasimetric distances are preserved in embedding space;
3. Highlight and a common issue in RL representation learning with respect to irrelevant noises in the environment, and propose a method to address this shortcoming and learn representations invariant to such noises.

Each of these three parts is presented in a chapter.

Chapter 2

Understanding Contrastive Representation Learning through Alignment and Uniformity on the Hypersphere

Contrastive representation learning has been outstandingly successful in practice. In this chapter, we identify two key properties related to the contrastive loss: (1) *alignment* (closeness) of features from positive pairs, and (2) *uniformity* of the induced distribution of the (normalized) features on the hypersphere. We prove that, asymptotically, the contrastive loss optimizes these properties, and analyze their positive effects on downstream tasks. Empirically, we introduce an optimizable metric to quantify each property. Extensive experiments on standard vision and language datasets confirm the strong agreement between *both* metrics and downstream task performance. Directly optimizing for these two metrics leads to representations with comparable or better performance at downstream tasks than contrastive learning.



Alignment: Similar samples have similar features.
(Figure inspired by Tian et al. [153].)

Uniformity: Preserve maximal information.

Figure 2-1: Illustration of alignment and uniformity of feature distributions on the output unit hypersphere. STL-10 [31] images are used for demonstration.

2.1 Introduction

A vast number of recent empirical works learn representations with a unit ℓ_2 norm constraint, effectively restricting the output space to the unit hypersphere [123, 137, 104, 67, 165, 15, 114, 78, 34, 173], including many unsupervised contrastive representation learning methods [171, 7, 153, 70, 25].

Intuitively, having the features live on the unit hypersphere leads to several desirable traits. Fixed-norm vectors are known to improve training stability in modern machine learning where dot products are ubiquitous [173, 165]. Moreover, if features of a class are sufficiently well clustered, they are linearly separable with the rest of feature space (see Figure 2-2), a common criterion used to evaluate representation quality.

While the unit hypersphere is a popular choice of feature space, not all encoders that map onto it are created equal. Recent works argue that representations should additionally be invariant to unnecessary details, and preserve as much information as possible [119, 153, 76, 7]. Let us call these two properties *alignment* and *uniformity* (see Figure 2-1). *Alignment* favors encoders that assign similar features to similar samples. *Uniformity* prefers a feature distribution that preserves maximal information, i.e., the uniform distribution on the unit hypersphere.

In this work, we analyze the *alignment* and *uniformity* properties. We show

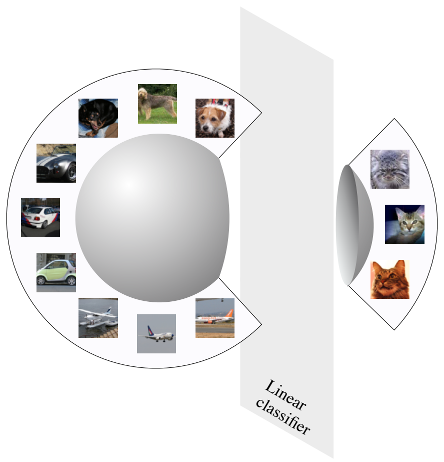


Figure 2-2: **Hypersphere:** When classes are well-clustered (forming spherical caps), they are linearly separable. The same does not hold for Euclidean spaces.

that a currently popular form of contrastive representation learning in fact directly optimizes for these two properties in the limit of infinite negative samples. We propose theoretically-motivated metrics for alignment and uniformity, and observe strong agreement between them and downstream task performance. Remarkably, directly optimizing for these two metrics leads to comparable or better performance than contrastive learning.

Our main contributions are:

- We propose quantifiable metrics for *alignment* and *uniformity* as two measures of representation quality, with theoretical motivations.

- We prove that the contrastive loss optimizes for alignment and uniformity asymptotically.

- Empirically, we find strong agreement between *both* metrics and downstream task performance.

- Despite being simple in form, our proposed metrics, when directly optimized with no other loss, empirically lead to comparable or better performance at downstream tasks than contrastive learning.

2.2 Related Work

Unsupervised Contrastive Representation Learning has seen remarkable success in learning representations for image and sequential data [106, 171, 119, 72, 153, 76, 7, 153, 70, 25]. The common motivation behind these work is the InfoMax principle [103], which we here instantiate as maximizing the mutual information (MI) between two views [153, 7, 170]. However, this interpretation is known to be inconsistent with the actual behavior in practice, e.g., optimizing a tighter bound on MI can lead to worse representations [158]. What the contrastive loss exactly does remains largely a mystery. Analysis based on the assumption of latent classes provides nice theoretical insights [134], but unfortunately has a rather large gap with empirical practices: the result that representation quality suffers with a large number of negatives is inconsistent with empirical observations [171, 153, 70, 25]. In this paper, we analyze and characterize the behavior of contrastive learning from the perspective of alignment and uniformity properties, and empirically verify our claims with standard representation learning tasks.

Representation learning on the unit hypersphere. Outside contrastive learning, many other representation learning approaches also normalize their features to be on the unit hypersphere. In variational autoencoders, the hyperspherical latent space has been shown to perform better than the Euclidean space [173, 34]. Directly matching uniformly sampled points on the unit hypersphere is known to provide good representations [15], agreeing with our intuition that uniformity is a desirable property. Mettes et al. [114] optimizes prototype representations on the unit hypersphere for classification. Hyperspherical face embeddings greatly outperform the unnormalized counterparts [123, 104, 165, 137]. Its empirical success suggests that the unit hypersphere is indeed a nice feature space. In this work, we formally investigate the interplay between the hypersphere geometry and the popular contrastive representation learning.

Distributing points on the unit hypersphere. The problem of uniformly distributing points on the unit hypersphere is a well-studied one. It is often defined as minimizing the total pairwise potential w.r.t. a certain kernel function [17, 96], e.g., the Thomson problem of finding the minimal electrostatic potential energy configuration of electrons [150], and minimization of the Riesz s-potential [52, 66, 105]. The uniformity metric we propose is based on the Gaussian potential, which can be used to represent a very general class of kernels and is closely related to the universally optimal point configurations [17, 32]. Additionally, the best-packing problem on hyperspheres (often called the Tammes problem) is also well studied [148].

2.3 Preliminaries on Unsupervised Contrastive Representation Learning

The popular unsupervised contrastive representation learning method (often referred to as *contrastive learning* in this paper) learns representations from unlabeled data. It assumes a way to sample *positive pairs*, representing similar samples that should have similar representations. Empirically, the positive pairs are often obtained by taking two independently randomly augmented versions of the same sample, e.g. two crops of the same image [171, 76, 7, 70, 25].

Let $p_{\mathsf{data}}(\cdot)$ be the data distribution over $\mathbb{R}^n$ and $p_{\mathsf{pos}}(\cdot, \cdot)$ the distribution of positive pairs over $\mathbb{R}^n \times \mathbb{R}^n$. Based on empirical practices, we assume the following property.

Assumption 2.3.1. Distributions p_{data} and p_{pos} should satisfy

- Symmetry: $\forall x, y,\ p_{\mathsf{pos}}(x, y) = p_{\mathsf{pos}}(y, x)$.
- Matching marginal: $\forall x,\ \int p_{\mathsf{pos}}(x, y)\, \mathrm{d}y = p_{\mathsf{data}}(x)$.

We consider the following specific and widely popular form of contrastive loss for training an encoder $f \colon \mathbb{R}^n \to \mathcal{S}^{m-1}$, mapping data to ℓ_2 normalized feature vectors of dimension m. This loss has been shown effective by many recent representation learning methods [106, 171, 153, 70, 76, 7, 25].

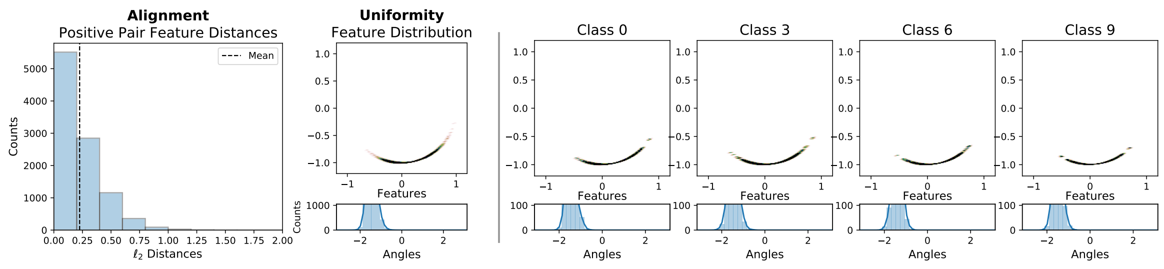
$$\mathcal{L}_{\textsf{contrastive}}(f;\tau,M) \triangleq \mathop{\mathbb{E}}_{\substack{(x,y)\sim p_{\textsf{pos}} \\ \{x_i^-\}_{i=1}^M \overset{\text{i.i.d.}}{\sim} p_{\textsf{data}}}} \left[-\log \frac{e^{f(x)^\mathsf{T} f(y)/\tau}}{e^{f(x)^\mathsf{T} f(y)/\tau} + \sum_i e^{f(x_i^-)^\mathsf{T} f(y)/\tau}} \right], \tag{2.1}$$

where $\tau > 0$ is a scalar temperature hyperparameter, and $M \in \mathbb{Z}_+$ is a fixed number of negative samples.

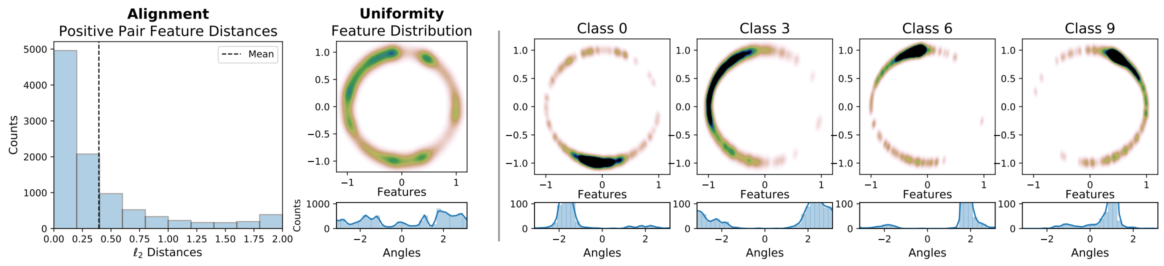
The term *contrastive loss* has also been generally used to refer to various objectives based on positive and negative samples, e.g., in Siamese networks [30, 62]. In this work, we focus on the specific form in Equation (2.1) that is widely used in modern unsupervised contrastive representation learning literature.

Necessity of normalization. Without the norm constraint, the `softmax` distribution can be made arbitrarily sharp by simply scaling all the features. Wang et al. [165] provided an analysis on this effect and argued for the necessity of normalization when using feature vector dot products in a cross entropy loss, as is in Eqn. (2.1). Experimentally, Chen et al. [25] also showed that normalizing outputs leads to superior representations.

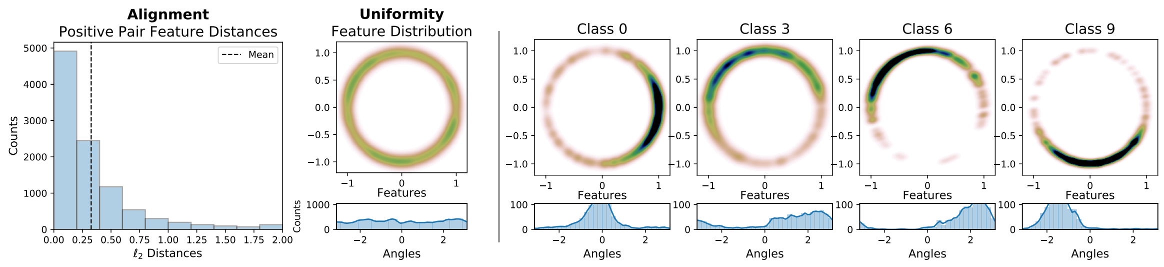
The InfoMax principle. Many empirical works are motivated by the InfoMax principle of maximizing $I(f(x); f(y))$ for $(x, y) \sim p_{\textsf{pos}}$ [153, 7, 170]. Usually they interpret $\mathcal{L}_{\textsf{contrastive}}$ in Eqn. (2.1) as a lower bound of $I(f(x); f(y))$ [119, 76, 7, 153]. However, this interpretation is known to have issues in practice, e.g., maximizing a tighter bound often leads to worse downstream task performance [158]. Therefore, instead of viewing it as a bound, we investigate the exact behavior of directly optimizing $\mathcal{L}_{\textsf{contrastive}}$ in the following sections.



(a) **Random Initialization.** Linear classification validation accuracy: 12.71%.



(b) **Supervised Predictive Learning.** Linear classification validation accuracy: 57.19%.



(c) **Unsupervised Contrastive Learning.** Linear classification validation accuracy: 28.60%.

Figure 2-3: Representations of CIFAR-10 validation set on $\mathcal{S}^1$. **Alignment analysis:** We show distribution of distance between features of positive pairs (two random augmentations). **Uniformity analysis:** We plot feature distributions with Gaussian kernel density estimation (KDE) in $\mathbb{R}^2$ and von Mises-Fisher (vMF) KDE on angles (i.e., $\operatorname{arctan2}(y, x)$ for each point $(x, y) \in \mathcal{S}^1$). **Four rightmost plots** visualize feature distributions of selected specific classes. Representation from contrastive learning is both *aligned* (having low positive pair feature distances) and *uniform* (evenly distributed on $\mathcal{S}^1$).

2.4 Feature Distribution on the Hypersphere

The contrastive loss encourages learned feature representation for positive pairs to be similar, while pushing features from the randomly sampled negative pairs apart. Conventional wisdom says that representations should extract the most shared information between positive pairs and remain invariant to other noise factors [103, 153, 170, 7]. Therefore, the loss should prefer two following properties:

- *Alignment*: two samples forming a positive pair should be mapped to nearby features, and thus be (mostly) invariant to unneeded noise factors.

- *Uniformity*: feature vectors should be roughly uniformly distributed on the unit hypersphere $\mathcal{S}^{m-1}$, preserving as much information of the data as possible.

To empirically verify this, we visualize CIFAR-10 [156, 94] representations on $\mathcal{S}^1$ ($m = 2$) obtained via three different methods:

- Random initialization.

- Supervised predictive learning: An encoder and a linear classifier are jointly trained from scratch with cross entropy loss on supervised labels.

- Unsupervised contrastive learning: An encoder is trained w.r.t. $\mathcal{L}_{\mathsf{contrastive}}$ with $\tau = 0.5$ and $M = 256$.

All three encoders share the same AlexNet based architecture [95], modified to map input images to 2-dimensional vectors in $\mathcal{S}^1$. Both predictive and contrastive learning use standard data augmentations to augment the dataset and sample positive pairs.

Figure 2-3 summarizes the resulting distributions of validation set features. Indeed, features from unsupervised contrastive learning (bottom in Figure 2-3) exhibit the most uniform distribution, and are closely clustered for positive pairs.

The form of the contrastive loss in Eqn. (2.1) also suggests this. We present informal arguments below, followed by more formal treatment in Section 2.4.2. From the symmetry of p, we can derive

$$\begin{aligned}\mathcal{L}_{\mathsf{contrastive}}(f; \tau, M) = & \mathop{\mathbb{E}}_{(x,y)\sim p_{\mathsf{pos}}}\left[-f(x)^{\mathsf{T}} f(y)/\tau\right] \\ & + \mathop{\mathbb{E}}_{\substack{(x,y)\sim p_{\mathsf{pos}} \\ \{x_i^-\}_{i=1}^M \overset{\text{i.i.d.}}{\sim} p_{\mathsf{data}}}}\left[\log\left(e^{f(x)^{\mathsf{T}} f(y)/\tau} + \sum_i e^{f(x_i^-)^{\mathsf{T}} f(x)/\tau}\right)\right].\end{aligned}$$

Because the $\sum_i e^{f(x_i^-)^{\mathsf{T}} f(x)/\tau}$ term is always positive and bounded below, the loss favors smaller $\mathbb{E}\left[-f(x)^{\mathsf{T}} f(y)/\tau\right]$, i.e., having more aligned positive pair features. Suppose the encoder is perfectly aligned, i.e., $\mathbb{P}\left[f(x) = f(y)\right] = 1$, then minimizing the loss is

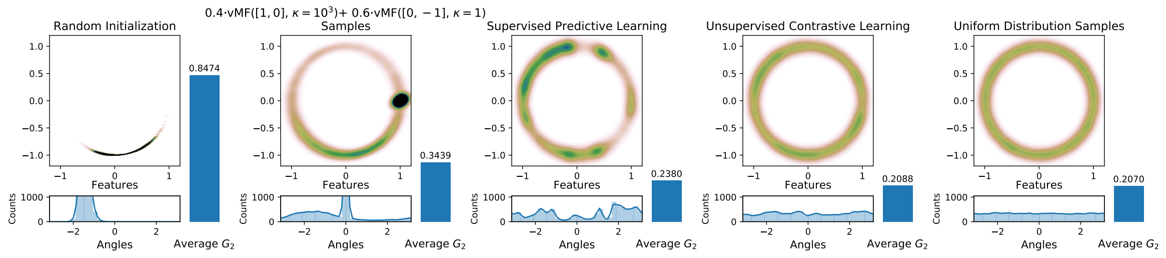


Figure 2-4: Average pairwise G_2 potential as a measure of uniformity. Each plot shows 10000 points distributed on $\mathcal{S}^1$, obtained via either applying an encoder on CIFAR-10 validation set (same as those in Figure 2-3) or sampling from a distribution on $\mathcal{S}^1$, as described in plot titles. We show the points with Gaussian KDE and the angles with vMF KDE.

equivalent to optimizing

$$\mathop{\mathbb{E}}_{\substack{x \sim p_{\mathsf{data}} \\ \{x_i^-\}_{i=1}^M \overset{\text{i.i.d.}}{\sim} p_{\mathsf{data}}}} \left[\log \left(e^{1/\tau} + \sum_i e^{f(x_i^-)^\intercal f(x)/\tau} \right) \right],$$

which is akin to maximizing pairwise distances with a `LogSumExp` transformation. Intuitively, pushing all features away from each other should indeed cause them to be roughly uniformly distributed.

2.4.1 Quantifying Alignment and Uniformity

For further analysis, we need a way to measure alignment and uniformity. We propose the following two metrics (losses).

Alignment

The alignment loss is straightforwardly defined with the expected distance between positive pairs:

$$\mathcal{L}_{\mathsf{align}}(f; \alpha) \triangleq \mathop{\mathbb{E}}_{(x,y) \sim p_{\mathsf{pos}}} \left[\| f(x) - f(y) \|_2^\alpha \right], \quad \alpha > 0.$$

Uniformity

We want the uniformity metric to be both asymptotically correct (i.e., the distribution optimizing this metric should converge to uniform distribution) and empirically reasonable with finite number of points. To this end, we consider the Gaussian potential kernel (also known as the Radial Basis Function (RBF) kernel) $G_t \colon \mathcal{S}^d \times \mathcal{S}^d \to \mathbb{R}_+$ [32, 17]:

$$G_t(u, v) \triangleq e^{-t\|u-v\|_2^2} = e^{2t \cdot u^\mathsf{T} v - 2t}, \quad t > 0,$$

and define the uniformity loss as the logarithm of the average pairwise Gaussian potential:

$$\begin{aligned}\mathcal{L}_{\mathsf{uniform}}(f; t) &\triangleq \log \mathop{\mathbb{E}}_{x,y \overset{\text{i.i.d.}}{\sim} p_{\mathsf{data}}} [G_t(u, v)] \\ &= \log \mathop{\mathbb{E}}_{x,y \overset{\text{i.i.d.}}{\sim} p_{\mathsf{data}}} \left[e^{-t\|f(x)-f(y)\|_2^2}\right], \quad t > 0.\end{aligned}$$

The average pairwise Gaussian potential is nicely tied with the uniform distribution on the unit hypersphere.

Definition 2.4.1 (Uniform distribution on $\mathcal{S}^d$)**.** σ_d denotes the normalized surface area measure on $\mathcal{S}^d$.

First, we show that the uniform distribution is the unique distribution that minimize the expected pairwise potential.

Proposition 2.4.2. For $\mathcal{M}(\mathcal{S}^d)$ the set of Borel probability measures on $\mathcal{S}^d$, σ_d is the unique solution of

$$\min_{\mu \in \mathcal{M}(\mathcal{S}^d)} \int_u \int_v G_t(u, v) \, \mathrm{d}\mu \, \mathrm{d}\mu.$$

Proof. See Appendix A.1.1. □

In addition, as number of points goes to infinity, distributions of points minimizing the average pairwise potential converge weak* to the uniform distribution. Recall the definition of the weak* convergence of measures.

Definition 2.4.3 (Weak* convergence of measures)**.** A sequence of Borel measures $\{\mu_n\}_{n=1}^\infty$ in $\mathbb{R}^p$ converges weak* to a Borel measure μ if for all continuous function $f \colon \mathbb{R}^p \to \mathbb{R}$, we have

$$\lim_{n \to \infty} \int f(x) \, \mathrm{d}\mu_n(x) = \int f(x) \, \mathrm{d}\mu(x).$$

Proposition 2.4.4. For each $N > 0$, the N point minimizer of the average pairwise potential is

$$\mathbf{u}_N^* = \underset{u_1,u_2,\ldots,u_N \in \mathcal{S}^d}{\arg\min} \sum_{1 \le i < j \le N} G_t(u_i, u_j).$$

The normalized counting measures associated with the $\{\mathbf{u}_N^*\}_{N=1}^{\infty}$ sequence converge weak* to σ_d.

Proof. See Appendix A.1.1. □

Designing an objective minimized by the uniform distribution is in fact nontrivial. For instance, average pairwise dot products or Euclidean distances is simply optimized by any distribution that has zero mean. Among kernels that achieve uniformity at optima, the Gaussian kernel is special in that it is closely related to the universally optimal point configurations and can also be used to represent a general class of other kernels, including the Riesz s-potentials. We refer readers to Borodachov et al. [17] and Cohn and Kumar [32] for in-depth discussions on these topics. Moreover, as we show below, $\mathcal{L}_{\mathsf{uniform}}$, defined with the Gaussian kernel, has close connections with $\mathcal{L}_{\mathsf{contrastive}}$.

Empirically, we evaluate the average pairwise potential of various finite point collections on $\mathcal{S}^1$ in Figure 2-4. The values nicely align with our intuitive understanding of uniformity.

We further discuss properties of $\mathcal{L}_{\mathsf{uniform}}$ and characterize its optimal value and range in Appendix A.1.1.

2.4.2 Limiting Behavior of Contrastive Learning

In this section, we formalize the intuition that contrastive learning optimizes alignment and uniformity, and characterize its asymptotic behavior. We consider optimization problems over all measurable encoder functions from the p_{data} measure in $\mathbb{R}^n$ to the Borel space $\mathcal{S}^{m-1}$.

We first define the notion of optimal encoders for each of these two metrics.

Definition 2.4.5 (Perfect Alignment)**.** We say an encoder f is *perfectly aligned* if $f(x) = f(y)$ a.s. over $(x, y) \sim p_{\textsf{pos}}$.

Definition 2.4.6 (Perfect Uniformity)**.** We say an encoder f is *perfectly uniform* if the distribution of $f(x)$ for $x \sim p_{\textsf{data}}$ is the uniform distribution σ_{m-1} on $\mathcal{S}^{m-1}$.

Realizability of perfect uniformity. We note that it is not always possible to achieve perfect uniformity, e.g., when the data manifold in $\mathbb{R}^n$ is lower dimensional than the feature space $\mathcal{S}^{m-1}$. Moreover, in the case that $p_{\textsf{data}}$ and $p_{\textsf{pos}}$ are formed from sampling augmented samples from a finite dataset, there cannot be an encoder that is *both* perfectly aligned and perfectly uniform, because perfect alignment implies that all augmentations from a single element have the same feature vector. Nonetheless, perfectly uniform encoder functions do exist under the conditions that $n \geq m - 1$ and $p_{\textsf{data}}$ has bounded density.

We analyze the asymptotics with infinite negative samples. Existing empirical work has established that larger number of negative samples consistently leads to better downstream task performances [171, 153, 70, 25], and often uses very large values (e.g., $M = 65536$ in He et al. [70]). The following theorem nicely confirms that optimizing w.r.t. the limiting loss indeed requires both alignment and uniformity.

Theorem 2.4.7 (Asymptotics of $\mathcal{L}_{\textsf{contrastive}}$). For fixed $\tau > 0$, as the number of negative samples $M \to \infty$, the (normalized) contrastive loss converges to

$$\begin{aligned}
&\lim_{M\to\infty} \mathcal{L}_{\textsf{contrastive}}(f; \tau, M) - \log M = \\
&\qquad -\frac{1}{\tau} \mathop{\mathbb{E}}_{(x,y)\sim p_{\textsf{pos}}}\left[f(x)^\intercal f(y)\right] + \mathop{\mathbb{E}}_{x\sim p_{\textsf{data}}}\left[\log \mathop{\mathbb{E}}_{x^-\sim p_{\textsf{data}}}\left[e^{f(x^-)^\intercal f(x)/\tau}\right]\right].
\end{aligned} \tag{2.2}$$

We have the following results:

1. The first term is minimized iff f is perfectly aligned.

2. If perfectly uniform encoders exist, they form the exact minimizers of the second term.

3. For the convergence in Equation (2.2), the absolute deviation from the limit decays in $\mathcal{O}(M^{-1/2})$.

Proof. See Appendix A.1.2. $\square$

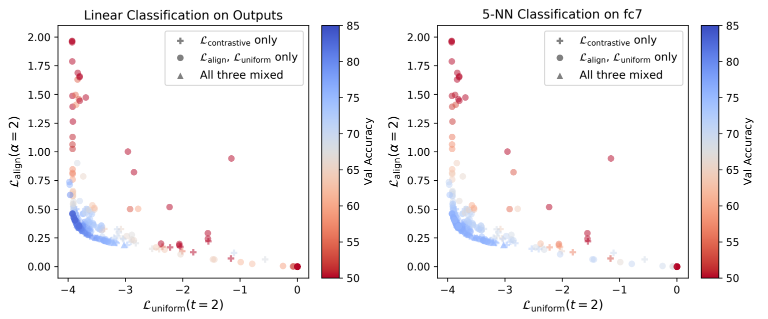
Relation with $\mathcal{L}_{\mathsf{uniform}}$. The proof of Theorem 2.4.7 in the Appendix A.1.2 connects the asymptotic $\mathcal{L}_{\mathsf{contrastive}}$ form with minimizing average pairwise Gaussian potential, i.e., minimizing $\mathcal{L}_{\mathsf{uniform}}$. Compared with the second term of Equation (2.2), $\mathcal{L}_{\mathsf{uniform}}$ essentially pushes the log outside the outer expectation, without changing the minimizer (perfectly uniform encoders). However, due to its pairwise nature, $\mathcal{L}_{\mathsf{uniform}}$ is much simpler in form and avoids the computationally expensive `softmax` operation in $\mathcal{L}_{\mathsf{contrastive}}$ [51, 10, 59, 54, 24].

Relation with feature distribution entropy estimation. When p_{data} is uniform over finite samples $\{x_1, x_2, \dots, x_N\}$ (e.g., a collected dataset), the second term in Equation (2.2) can be alternatively viewed as a resubstitution entropy estimator of $f(x)$ [2], where x follows the underlying distribution p_{nature} that generates $\{x_i\}_{i=1}^N$, via a von Mises-Fisher (vMF) kernel density estimation (KDE):

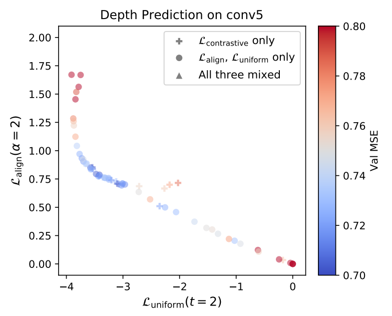
$$\begin{aligned}
\mathop{\mathbb{E}}_{x \sim p_{\mathsf{data}}}\left[\log \mathop{\mathbb{E}}_{x^- \sim p_{\mathsf{data}}}\left[e^{f(x^-)^\intercal f(x)/\tau}\right]\right] &= \frac{1}{N}\sum_{i=1}^N \log\left(\frac{1}{N}\sum_{j=1}^N e^{f(x_i)^\intercal f(x_j)/\tau}\right) \\
&= \frac{1}{N}\sum_{i=1}^N \log \hat{p}_{\mathsf{vMF\text{-}KDE}}(f(x_i)) + \log Z_{\mathsf{vMF}} \\
&\triangleq -\hat{H}(f(x)) + \log Z_{\mathsf{vMF}}, \qquad\qquad x \sim p_{\mathsf{nature}} \\
&\triangleq -\hat{I}(x; f(x)) + \log Z_{\mathsf{vMF}}, \qquad\qquad x \sim p_{\mathsf{nature}},
\end{aligned}$$

where

- $\hat{p}_{\mathsf{vMF\text{-}KDE}}$ is the KDE based on samples $\{f(x_j)\}_{j=1}^N$ using a vMF kernel with $\kappa = \tau^{-1}$,
- Z_{vMF} is the normalization constant for vMF distribution with $\kappa = \tau^{-1}$,
- $\hat{H}$ denotes the resubstitution entropy estimator,



(a) 304 STL-10 encoders are evaluated with linear classification on output features and 5-nearest neighbor (5-NN) on fc7 activations. Higher accuracy (blue color) is better.



(b) 64 NYU-Depth-V2 encoders are evaluated with CNN depth regressors on conv5 activations. Lower MSE (blue color) is better.

Figure 2-5: Metrics and performance of STL-10 and NYU-Depth-V2 experiments. Each point represents a trained encoder, with its x- and y-coordinates showing $\mathcal{L}_{\text{align}}$ and $\mathcal{L}_{\text{uniform}}$ metrics and color showing the performance on validation set. **Blue** is better for both tasks. Encoders with low $\mathcal{L}_{\text{align}}$ and $\mathcal{L}_{\text{uniform}}$ are consistently the better performing ones (lower left corners).

- $\hat{I}$ denotes the mutual information estimator based on $\hat{H}$, since f is a deterministic function.

Relation with the InfoMax principle. Many empirical works are motivated by the InfoMax principle, i.e., maximizing $I(f(x); f(y))$ for $(x, y) \sim p_{\text{pos}}$. However, the interpretation of $\mathcal{L}_{\text{contrastive}}$ as a lower bound of $I(f(x); f(y))$ is known to be inconsistent with its actual behavior in practice [158]. Our results instead analyze the properties of $\mathcal{L}_{\text{contrastive}}$ itself. Considering the identity $I(f(x); f(y)) = H(f(x)) - H(f(x) \mid f(y))$, we can see that while uniformity indeed favors large $H(f(x))$, alignment is stronger than merely desiring small $H(f(x) \mid f(y))$. In particular, both Theorem 2.4.7 and the above connection with maximizing an entropy estimator provide alternative interpretations and motivations that $\mathcal{L}_{\text{contrastive}}$ optimizes for *aligned* and *information-preserving* encoders.

Finally, even for the case where only a single negative sample is used (i.e., $M = 1$), we can still prove a weaker result, which we describe in details in the Appendix A.1.2.

```
# bsz : batch size (number of positive pairs)
# d   : latent dim
# x   : Tensor, shape=[bsz, d]
#       latents for one side of positive pairs
# y   : Tensor, shape=[bsz, d]
#       latents for the other side of positive pairs
# lam : hyperparameter balancing the two losses

def lalign(x, y, alpha=2):
    return (x - y).norm(dim=1).pow(alpha).mean()

def lunif(x, t=2):
    sq_pdist = torch.pdist(x, p=2).pow(2)
    return sq_pdist.mul(-t).exp().mean().log()

loss = lalign(x, y) + lam * (lunif(x) + lunif(y)) / 2
```

Figure 2-6: PyTorch implementation of $\mathcal{L}_{\mathsf{align}}$ and $\mathcal{L}_{\mathsf{uniform}}$.

2.5 Experiments

In this section, we empirically verify the hypothesis that alignment and uniformity are desired properties for representations. Recall that our two metrics are

$$\mathcal{L}_{\mathsf{align}}(f;\alpha) \triangleq \mathbb{E}_{(x,y)\sim p_{\mathsf{pos}}}\left[\|f(x)-f(y)\|_2^\alpha\right]$$
$$\mathcal{L}_{\mathsf{uniform}}(f;t) \triangleq \log\ \mathbb{E}_{x,y \overset{\text{i.i.d.}}{\sim} p_{\mathsf{data}}}\left[e^{-t\|f(x)-f(y)\|_2^2}\right].$$

We conduct extensive experiments with convolutional neural network (CNN) and recurrent neural network (RNN) based encoders on four popular representation learning benchmarks with distinct types of downstream tasks:

- STL-10 [31] classification on AlexNet-based encoder outputs or intermediate activations with a linear or k-nearest neighbor (k-NN) classifier.
- NYU-Depth-V2 [118] depth prediction on CNN encoder intermediate activations after convolution layers.
- ImageNet and ImageNet-100 (random 100-class subset of ImageNet) classification on CNN encoder penultimate layer activations with a linear classifier.
- BookCorpus [177] RNN sentence encoder outputs used for Moview Review Sentence Polarity (MR) [122] and Customer Product Review Sentiment (CR)

	Loss Formula	Validation Set Accuracy ↑			
		Output + Linear	Output + 5-NN	fc7 + Linear	fc7 + 5-NN
Best $\mathcal{L}_{\mathsf{contrastive}}$ only	$\mathcal{L}_{\mathsf{contrastive}}(\tau{=}0.19)$	80.46%	78.75%	83.89%	76.33%
Best $\mathcal{L}_{\mathsf{align}}$ and $\mathcal{L}_{\mathsf{uniform}}$ only	$0.98 \cdot \mathcal{L}_{\mathsf{align}}(\alpha{=}2) + 0.96 \cdot \mathcal{L}_{\mathsf{uniform}}(t{=}2)$	**81.15%**	78.89%	**84.43%**	**76.78%**
Best among all encoders	$\mathcal{L}_{\mathsf{contrastive}}(\tau{=}0.5) + \mathcal{L}_{\mathsf{uniform}}(t{=}2)$	81.06%	**79.05%**	84.14%	76.48%

Table 2.1: STL-10 encoder evaluations. Numbers show linear and 5-nearest neighbor (5-NN) classification accuracies on the validation set. The best result is picked by encoder outputs linear classifier accuracy from a 5-fold training set cross validation, among all 150 encoders trained from scratch with 128-dimensional output and 768 batch size.

	Loss Formula	Validation Set MSE ↓	
		conv5	conv4
Best $\mathcal{L}_{\mathsf{contrastive}}$ only	$0.5 \cdot \mathcal{L}_{\mathsf{contrastive}}(\tau{=}0.1)$	0.7024	**0.7575**
Best $\mathcal{L}_{\mathsf{align}}$ and $\mathcal{L}_{\mathsf{uniform}}$ only	$0.75 \cdot \mathcal{L}_{\mathsf{align}}(\alpha{=}2) + 0.5 \cdot \mathcal{L}_{\mathsf{uniform}}(t{=}2)$	**0.7014**	0.7592
Best among all encoders	$0.75 \cdot \mathcal{L}_{\mathsf{align}}(\alpha{=}2) + 0.5 \cdot \mathcal{L}_{\mathsf{uniform}}(t{=}2)$	**0.7014**	0.7592

Table 2.2: NYU-Depth-V2 encoder evaluations. Numbers show depth prediction mean squared error (MSE) on the validation set. The best result is picked based on conv5 layer MSE from a 5-fold training set cross validation, among all 64 encoders trained from scratch with 128-dimensional output and 128 batch size.

[167] binary classification tasks with logisitc classifiers.

For image datasets, we follow the standard practice and choose positive pairs as two independent augmentations of the same image. For BookCorpus, positive pairs are chosen as neighboring sentences, following Quick-Thought Vectors [106].

We perform majority of our analysis on STL-10 and NYU-Depth-V2 encoders, where we calculate $\mathcal{L}_{\mathsf{contrastive}}$ with negatives being other samples within the minibatch following the standard practice [76, 7, 153, 25], and $\mathcal{L}_{\mathsf{uniform}}$ as the logarithm of average pairwise feature potentials also within the minibatch. Due to their simple forms, these two losses can be implemented in PyTorch [124] with less than 10 lines of code, as shown in Figure 2-6.

To investigate *alignment* and *uniformity* properties on recent contrastive learning methods and larger datasets, we also analyze ImageNet and ImageNet-100 encoders trained with Momentum Contrast (MoCo) [70, 26], and BookCorpus encoders trained with Quick-Thought Vectors [106], with these methods modified to also allow $\mathcal{L}_{\mathsf{align}}$ and $\mathcal{L}_{\mathsf{uniform}}$.

We optimize a total of 304 STL-10 encoders, 64 NYU-Depth-V2 encoders, 45 ImageNet-100 encoders, and 108 BookCorpus encoders without supervision.

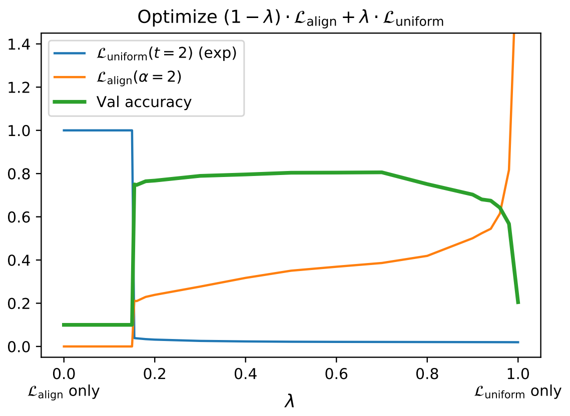


Figure 2-7: Effect of optimizing different weighted combinations of $\mathcal{L}_{\text{align}}(\alpha{=}2)$ and $\mathcal{L}_{\text{uniform}}(t{=}2)$ for STL-10. For each encoder, we show the $\mathcal{L}_{\text{align}}$ and $\mathcal{L}_{\text{uniform}}$ metrics, and validation accuracy of a linear classifier trained on encoder outputs. $\mathcal{L}_{\text{uniform}}$ is exponentiated for plotting purposes.

The encoders are optimized w.r.t. weighted combinations of $\mathcal{L}_{\text{contrastive}}$, $\mathcal{L}_{\text{align}}$, and/or $\mathcal{L}_{\text{uniform}}$, with varying

- (possibly zero) weights on the three losses,
- temperature τ for $\mathcal{L}_{\text{contrastive}}$,
- $\alpha \in \{1, 2\}$ for $\mathcal{L}_{\text{align}}$,
- $t \in \{1, 2, \dots, 8\}$ for $\mathcal{L}_{\text{uniform}}$,
- batch size (affecting the number of (negative) pairs for $\mathcal{L}_{\text{contrastive}}$ and $\mathcal{L}_{\text{uniform}}$),
- embedding dimension,
- number of training epochs and learning rate,
- initialization (from scratch vs. a pretrained encoder).

See Appendix A.2 for more experiment details and the exact configurations used.

$\mathcal{L}_{\text{align}}$ and $\mathcal{L}_{\text{uniform}}$ strongly agree with downstream task performance. For each encoder, we measure the downstream task performance, and the $\mathcal{L}_{\text{align}}$, $\mathcal{L}_{\text{uniform}}$ metrics on the validation set. Figure 2-5 visualizes the trends between both metrics and representation quality. We observe that the two metrics strongly agrees the

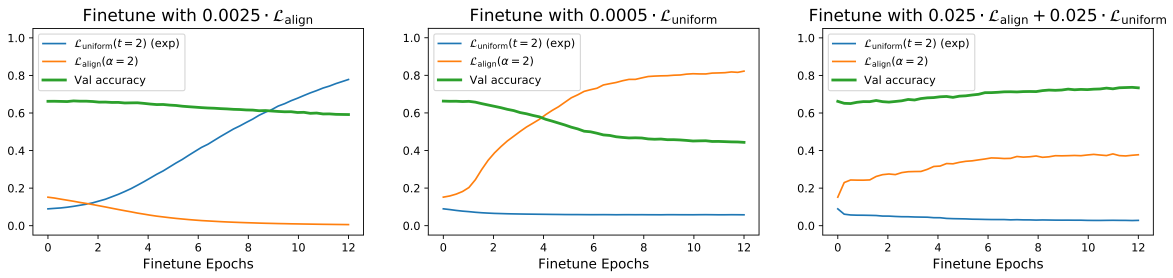
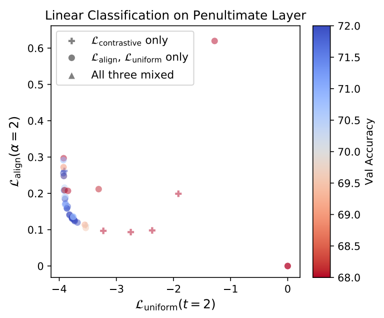


Figure 2-8: Finetuning trajectories from a STL-10 encoder trained with $\mathcal{L}_{\text{contrastive}}$ using a suboptimal temperature $\tau = 2.5$. Finetuning objectives are weighted combinations of $\mathcal{L}_{\text{align}}(\alpha{=}2)$ and $\mathcal{L}_{\text{uniform}}(t{=}2)$. For each intermediate checkpoint, we measure $\mathcal{L}_{\text{align}}$ and $\mathcal{L}_{\text{uniform}}$ metrics, as well as validation accuracy of a linear classifier trained from scratch on the encoder outputs. $\mathcal{L}_{\text{uniform}}$ is exponentiated for plotting purpose. **Left and middle:** Performance degrades if only one of alignment and uniformity is optimized. **Right:** Performance improves when both are optimized.

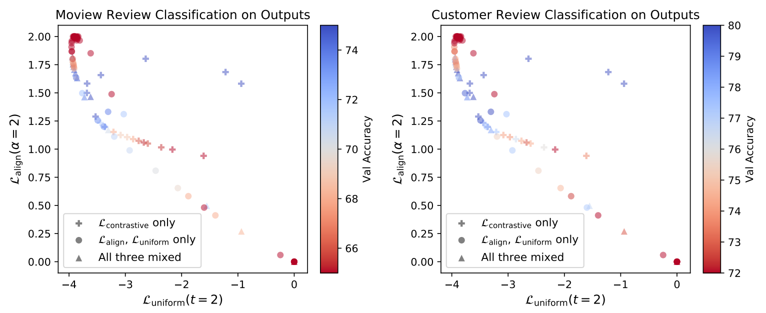
representation quality overall. In particular, the best performing encoders are exactly the ones with low $\mathcal{L}_{\text{align}}$ and $\mathcal{L}_{\text{uniform}}$, i.e., the lower left corners in Figure 2-5.

Directly optimizing only $\mathcal{L}_{\text{align}}$ and $\mathcal{L}_{\text{uniform}}$ can lead to better representations. As shown in Tables 2.1 and 2.2, encoders trained with only $\mathcal{L}_{\text{align}}$ and $\mathcal{L}_{\text{uniform}}$ consistently outperform their $\mathcal{L}_{\text{contrastive}}$-trained counterparts, for both tasks. Theoretically, Theorem 2.4.7 showed that $\mathcal{L}_{\text{contrastive}}$ optimizes alignment and uniformity asymptotically with infinite negative samples. This empirical performance gap suggests that directly optimizing these properties can be superior in practice, when we can only have finite negatives.

Both alignment and uniformity are necessary for a good representation. Figure 2-7 shows how the final encoder changes in response to optimizing differently weighted combinations of $\mathcal{L}_{\text{align}}$ and $\mathcal{L}_{\text{uniform}}$ on STL-10. The trade-off between the $\mathcal{L}_{\text{align}}$ and $\mathcal{L}_{\text{uniform}}$ indicates that perfect alignment and perfect uniformity are likely hard to simultaneously achieve in practice. However, the inverted-U-shaped accuracy curve confirms that both properties are indeed necessary for a good encoder. When $\mathcal{L}_{\text{align}}$ is weighted much higher than $\mathcal{L}_{\text{uniform}}$, degenerate solution occurs and all inputs are mapped to the same feature vector ($\exp \mathcal{L}_{\text{uniform}} = 1$). However, as long as the ratio between two weights is not too large (e.g., < 4), we observe that the representation



(a) 45 ImageNet-100 encoders are trained with MoCo-based methods, and evaluated with linear classification.



(b) 108 BookCorpus encoders are trained with Quick-Thought-Vectors-based methods, and evaluated with logistic binary classification on Movie Review Sentence Polarity and Customer Product Review Sentiment tasks.

Figure 2-9: Metrics and performance of ImageNet-100 and BookCorpus experiments. Each point represents a trained encoder, with its x- and y-coordinates showing $\mathcal{L}_{\text{align}}$ and $\mathcal{L}_{\text{uniform}}$ metrics and color showing the validation accuracy. **Blue** is better. Encoders with low $\mathcal{L}_{\text{align}}$ and $\mathcal{L}_{\text{uniform}}$ consistently perform well (lower left corners), even though the training methods (based on MoCo and Quick-Thought Vectors) are different from directly optimizing the contrastive loss in Equation (2.1).

quality remains relatively good and insensitive to the exact weight choices.

$\mathcal{L}_{\text{align}}$ and $\mathcal{L}_{\text{uniform}}$ causally affect downstream task performance. We take an encoder trained with $\mathcal{L}_{\text{contrastive}}$ using a suboptimal temperature $\tau = 2.5$, and finetune it according to $\mathcal{L}_{\text{align}}$ and/or $\mathcal{L}_{\text{uniform}}$. Figure 2-8 visualizes the finetuning trajectories. When only one of alignment and uniformity is optimized, the corresponding metric improves, but both the other metric and performance degrade. However, when both properties are optimized, the representation quality steadily increases. These trends confirm the causal effect of alignment and uniformity on the representation quality, and suggest that directly optimizing them can be a reasonable choice.

Alignment and uniformity also matter in other contrastive representation learning variants. MoCo [70] and Quick-Thought Vectors [106] are contrastive representation learning variants that have nontrivial differences with directly optimizing $\mathcal{L}_{\text{contrastive}}$ in Equation (2.1). MoCo introduces a memory queue and a momentum encoder. Quick-Thought Vectors uses two different encoders to encode each sentence in a positive pair, only normalizes encoder outputs during evaluation, and does not use random sampling to obtain minibatches. After modifying them to also allow

	Loss Formula	Validation Set Accuracy ↑	
		top1	top5
Best $\mathcal{L}_{\mathsf{contrastive}}$ only	$\mathcal{L}_{\mathsf{contrastive}}(\tau{=}0.07)$	72.80%	91.64%
Best $\mathcal{L}_{\mathsf{align}}$ and $\mathcal{L}_{\mathsf{uniform}}$ only	$3 \cdot \mathcal{L}_{\mathsf{align}}(\alpha{=}2) + \mathcal{L}_{\mathsf{uniform}}(t{=}3)$	**74.60%**	**92.74%**
Best among all encoders	$3 \cdot \mathcal{L}_{\mathsf{align}}(\alpha{=}2) + \mathcal{L}_{\mathsf{uniform}}(t{=}3)$	**74.60%**	**92.74%**

Table 2.3: ImageNet-100 encoder evaluations. Numbers show validation set accuracies of linear classifiers trained on encoder penultimate layer activations. The encoders are trained using MoCo-based methods. The best result is picked based on top1 accuracy from a 3-fold training set cross validation, among all 45 encoders trained from scratch with 128-dimensional output and 128 batch size.

	MR Classification		CR Classification	
	Loss Formula	Val. Set Accuracy ↑	Loss Formula	Val. Set Accuracy ↑
Best $\mathcal{L}_{\mathsf{contrastive}}$ only	$\mathcal{L}_{\mathsf{contrastive}}(\tau{=}0.075)$	**77.51%**	$\mathcal{L}_{\mathsf{contrastive}}(\tau{=}0.05)$	**83.86%**
Best $\mathcal{L}_{\mathsf{align}}$ and $\mathcal{L}_{\mathsf{uniform}}$ only	$0.9 \cdot \mathcal{L}_{\mathsf{align}}(\alpha{=}2) + 0.1 \cdot \mathcal{L}_{\mathsf{uniform}}(t{=}5)$	73.76%	$0.9 \cdot \mathcal{L}_{\mathsf{align}}(\alpha{=}2) + 0.1 \cdot \mathcal{L}_{\mathsf{uniform}}(t{=}5)$	80.95%
Best among all encoders	$\mathcal{L}_{\mathsf{contrastive}}(\tau{=}0.075)$	**77.51%**	$\mathcal{L}_{\mathsf{contrastive}}(\tau{=}0.05)$	**83.86%**

Table 2.4: BookCorpus encoder evaluations. Numbers show Movie Review Sentence Polarity (MR) and Customer Product Sentiment (CR) validation set classification accuracies of logistic classifiers fit on encoder outputs. The encoders are trained using Quick-Thought-Vectors-based methods. The best result is picked based on accuracy from a 5-fold training set cross validation, individually for MR and CR, among all 108 encoders trained from scratch with 1200-dimensional output and 400 batch size.

$\mathcal{L}_{\mathsf{align}}$ and $\mathcal{L}_{\mathsf{uniform}}$, we train these methods on ImageNet-100 and BookCorpus, respectively. Figure 2-9 shows that $\mathcal{L}_{\mathsf{align}}$ and $\mathcal{L}_{\mathsf{uniform}}$ metrics are still correlated with the downstream task performances. Tables 2.3 and 2.4 show that directly optimizing them also leads to comparable or better representation quality. Table 2.5 also shows improvements on full ImageNet when we use $\mathcal{L}_{\mathsf{align}}$ and $\mathcal{L}_{\mathsf{uniform}}$ to train MoCo v2 [26] (an improved version of MoCo). These results suggest that alignment and uniformity are indeed desirable properties for representations, for *both* image and text modalities, and are likely connected with general contrastive representation learning methods.

Loss Formula	Validation Set top1 Accuracy ↑
$\mathcal{L}_{\mathsf{contrastive}}(\tau{=}0.2)$ (MoCo v2 Chen et al. [26])	$67.5\% \pm 0.1\%$
$3 \cdot \mathcal{L}_{\mathsf{align}}(\alpha{=}2) + \mathcal{L}_{\mathsf{uniform}}(t{=}3)$	**67.69%**

Table 2.5: ImageNet encoder evaluations with MoCo v2, and its variant with $\mathcal{L}_{\mathsf{align}}$ and $\mathcal{L}_{\mathsf{uniform}}$. MoCo v2 results are from the MoCo v2 official implementation [27], with mean and standard deviation across 5 runs. Both settings use 200 epochs of unsupervised training.

2.6 Discussion

Alignment and *uniformity* are often alluded to as motivations for representation learning methods (see Figure 2-1). However, a thorough understanding of these properties is lacking in the literature.

Are they in fact related to the representation learning methods? Do they actually agree with the representation quality (measured by downstream task performance)?

In this work, we have presented a detailed investigation on the relation between these properties and the popular paradigm of contrastive representation learning. Through theoretical analysis and extensive experiments, we are able to relate the contrastive loss with the alignment and uniformity properties, and confirm their strong connection with downstream task performances. Remarkably, we have revealed that directly optimizing our proposed metrics often leads to representations of better quality.

Below we summarize several suggestions for future work.

Niceness of the unit hypersphere. Our analysis was based on the empirical observation that representations are often ℓ_2 normalized. Existing works have motivated this choice from a manifold mapping perspective [104, 34] and computation stability [173, 165]. However, to our best knowledge, the question of why the unit hypersphere is a nice feature space is not yet rigorously answered. One possible direction is to formalize the intuition that connected sets with smooth boundaries are nearly linearly separable in the hyperspherical geometry (see Figure 2-2), since linear separability is one of the most widely used criteria for representation quality and is related to the notion of disentanglement [74].

Beyond contrastive learning. Our analysis focused on the relationship between contrastive learning and the alignment and uniformity properties on the unit hypersphere. However, the ubiquitous presence of ℓ_2 normalization in the representation learning literature suggests that the connection may be more general. In fact, several existing empirical methods are directly related to uniformity on the hypersphere

[15, 34, 173]. We believe that relating a broader class of representations to uniformity and/or alignment on the hypersphere will provide novel insights and lead to better empirical algorithms.

Chapter 3

On the Learning and Learnability of Quasimetrics

Our world is full of asymmetries. Gravity and wind can make reaching a place easier than coming back. Social artifacts such as genealogy charts and citation graphs are inherently directed. In reinforcement learning and control, optimal goal-reaching strategies are rarely reversible (symmetrical). Distance functions supported on these asymmetrical structures are called *quasimetrics*. Despite their common appearance, little research has been done on the learning of quasimetrics.

Our theoretical analysis reveals that a common class of learning algorithms, including unconstrained multilayer perceptrons (MLPs), provably fails to learn a quasimetric consistent with training data. In contrast, our proposed Poisson Quasimetric Embedding (PQE) is the first quasimetric learning formulation that both is learnable with gradient-based optimization and enjoys strong performance guarantees. Experiments on random graphs, social graphs, and offline Q-learning demonstrate its effectiveness over many common baselines.

3.1 Introduction

Learned *symmetrical* metrics have been proven useful for innumerable tasks including dimensionality reduction [149], clustering [172], classification [169, 77], and information retrieval [166]. However, the real world is largely *asymmetrical*, and *symmetrical* metrics can only capture a small fraction of it.

Generalizing metrics, *quasimetrics* (Definition 3.2.1) allow for *asymmetrical* distances and can be found in a wide range of domains (see Figure 3-1). Ubiquitous physical forces, such as gravity and wind, as well as human-defined rules, such as one-way roads, make the traveling time between places a quasimetric. Furthermore, many of our social artifacts are directed graphs— genealogy charts, follow-relation on Twitter [101], citation graphs [129], hyperlinks over the Internet, etc. Shortest paths on these graphs naturally induce quasimetric spaces. In fact, we can generalize to Markov Decision Processes (MDPs) and observe that optimal goal-reaching plan costs (i.e., universal value/Q-functions [136, 146]) always form a quasimetric [12, 151]. Moving onto more abstract structures, quasimetrics can also be found as expected hitting times in Markov chains, and as conditional Shannon entropy $H(\cdot \mid \cdot)$ in information theory. (See the appendix for proofs and discussions of these quasimetrics.)

In this work, we study the task of *quasimetric learning*. Given a sampled training set of pairs and their quasimetric distances, we ask: <u>how well can we learn a quasimetric that fits the training data</u>? We define *quasimetric learning* in analogy to metric learning: whereas metric learning is the problem of learning a metric function, quasimetric learning is the problem of learning a quasimetric function. This may involve searching over a hypothesis space constrained to only include quasimetric functions (which is what our method does) or it could involve searching for approximately quasimetric functions (we compare to and analyze such approaches). Successful formulations have many potential applications, such as structural priors in reinforcement learning [136, 151], graph learning [132] and causal relation learning [8].

Towards this goal, our contributions are

- We study the quasimetric learning task with two goals: (1) fitting training data well and (2) respecting quasimetric constraints (Section 3.3);
- We prove that a large family of algorithms, including unconstrained networks trained in the Neural Tangent Kernel (NTK) regime [83], fail at this task, while a learned embedding into a latent quasimetric space can potentially succeed

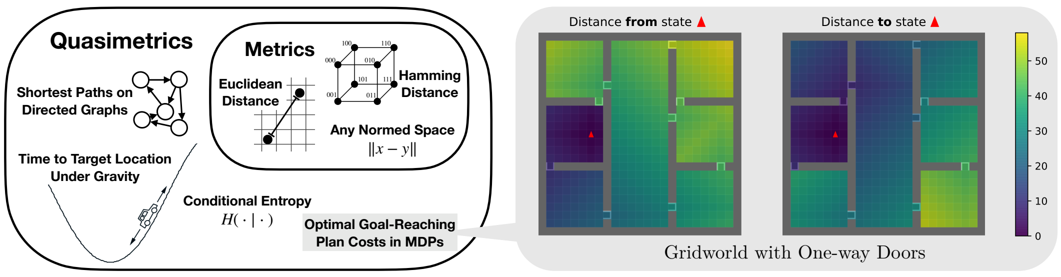


Figure 3-1: Examples of quasimetric spaces. The car drawing is borrowed from Sutton and Barto [145].

(Section 3.4);

- We propose Poisson Quasimetric Embeddings (PQEs), the first quasimetric embedding formulation learnable with gradient-based optimization that also enjoys strong theoretical guarantees on approximating arbitrary quasimetrics (Section 3.5);
- Our experiments complement the theory and demonstrate the benefits of PQEs on random graphs, social graphs and offline Q-learning (Section 4.5).

3.2 Preliminaries on Quasimetrics and Poisson Processes

Quasimetric space is a generalization of metric space where all requirements of metrics are satisfied, except that the distances can be asymmetrical.

Definition 3.2.1 (Quasimetric Space)**.** A *quasimetric space* is a pair $(\mathcal{X}, d)$, where $\mathcal{X}$ is a set of points and $d\colon \mathcal{X} \times \mathcal{X} \to [0, \infty]$ is the quasimetric, satisfying the following conditions:

$$\forall x, y \in \mathcal{X}, \qquad x = y \iff d(x, y) = 0, \quad \text{(Identity of Indiscernibles)}$$
$$\forall x, y, z \in \mathcal{X}, \qquad d(x, y) + d(y, z) \geq d(x, z). \qquad \text{(Triangle Inequality)}$$

Being asymmetric, quasimetrics are often thought of as (shortest-path) distances of some (possibly infinite) weighted directed graph. A natural way to quantify the complexity of a quasimetric is to consider that of its underlying graph. *Quasimetric treewidth* is an instantiation of this idea.

Definition 3.2.2 (Treewidth of Quasimetric Spaces [113]). Consider a quasimetric space M as shortest-path distances on a positively-weighted directed graph. *Treewidth* of M is the minimum over all such graphs' treewidths.

Poisson processes are commonly used to model events (or points) randomly occurring across a set A [89] , e.g., raindrops hitting a windshield, photons captured by a camera. The number of such events within a subset of A is modeled as a Poisson distribution, whose mean is given by a measure μ of A that determines how "frequently the events happen at each location".

Definition 3.2.3 (Poisson Process). For nonatomic measure μ on set A, a *Poisson process* on A with *mean measure* μ is a random countable subset $P \subset A$ (i.e., the random events / points) such that

- for any disjoint measurable subsets $A_1, \dots, A_n$ of A, the random variables $N(A_1), \dots, N(A_n)$ are independent, where $N(B) \triangleq \#\{P \cap B\}$ is the number of points of P in B, and
- $N(B)$ has the Poisson distribution with mean $\mu(B)$, denoted as $\text{Pois}(\mu(B))$.

Fact 3.2.4 (Differentiability of $\mathbb{P}[N(A_1) \leq N(A_2)]$). For two measurable subsets A_1, A_2,

$$\mathbb{P}[N(A_1) \leq N(A_2)] = \mathbb{P}\big[\underbrace{\text{Pois}(\mu(A_1 \setminus A_2)) \leq \text{Pois}(\mu(A_2 \setminus A_1))}_{\text{two } \textit{independent} \text{ Poissons}}\big]. \tag{3.1}$$

Furthermore, for independent $X \sim \text{Pois}(\mu_1)$, $Y \sim \text{Pois}(\mu_2)$, the probability $\mathbb{P}[X \leq Y]$ is *differentiable w.r.t.* μ_1 *and* μ_2. In the special case where μ_1 or μ_2 is zero, we can simply compute

$$\begin{aligned}\mathbb{P}[X \leq Y] &= \begin{cases} \mathbb{P}[0 \leq Y] = 1 & \text{if } \mu_1 = 0 \\ \mathbb{P}[X \leq 0] = \mathbb{P}[X = 0] = e^{-\mu_1} & \text{if } \mu_2 = 0 \end{cases} \qquad (\text{Pois}(0) \text{ is always } 0) \\ &= \exp\left(-(\mu_1 - \mu_2)^+\right), \end{aligned} \tag{3.2}$$

where $x^+ \triangleq \max(0, x)$. For general μ_1, μ_2, this probability and its gradients can be obtained via a connection to noncentral χ^2 distribution [84]. We derive the formulas in the appendix.

Therefore, if A_1 and A_2 are parametrized by some θ such that $\mu(A_1 \setminus A_2)$ and $\mu(A_2 \setminus A_1)$ are differentiable w.r.t. θ, so is $\mathbb{P}\left[N(A_1) \leq N(A_2)\right]$.

3.3 Quasimetric Learning

Consider a quasimetric space $(\mathcal{X}, d)$. The *quasimetric learning* task aims to infer a quasimetric from observing a training set $\{(x_i, y_i, d(x_i, y_i))\}_i \subset \mathcal{X} \times \mathcal{X} \times [0, \infty]$. Naturally, our goals for a learned predictor $\hat{d} \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ are: respecting the quasimetric constraints and fitting training distances.

Crucially, we are not simply aiming for the usual sense of *generalization*, i.e., low population error. Knowing that true distances have a quasimetric structure, we can better evaluate predictors and desire ones that fit the training data and are (approximately) quasimetrics. These objectives also indirectly capture generalization because a predictor failing either requirement must have large error on some pairs, whose true distances follow quasimetric constraints. We formalize this relation in Theorem 3.4.3.

3.3.1 Learning Algorithms and Hypothesis Spaces

Ideally, quasimetric learning should scale well with data, potentially generalize to unseen samples, and support integration with other deep learning systems (e.g., via differentiation).

Relaxed hypothesis spaces. One can simply learn a generic function approximator that maps the (concatenated) input pair to a scalar as the prediction of the pair's distance, or its transformed version (e.g., log distance). This approach has been adopted in learning graph distances [132] and plan costs in MDPs [151]. When the function approximator is a deep neural network, we refer to such methods as *unconstrained networks*. While they are known to fit training data well [83], in this paper we also investigate whether they learn to be (approximately) quasimetrics.

Restricted hypothesis spaces. Alternatively, we can encode each input to a latent space $\mathcal{Z}$, where a latent quasimetric d_z gives the distance prediction. This guarantees learning a quasimetric over data space $\mathcal{X}$. Often d_z is restricted to a subset unable to approximate all quasimetrics, i.e., an **overly restricted hypothesis space**, such as metric embeddings and the recently proposed DeepNorm and WideNorm [127]. While our proposed Poisson Quasimetric Embedding (PQE) (specified in Section 3.5) is also a latent quasimetric, it can approximate arbitrary quasimetrics (and is differentiable). PQE thus searches in **a space that approximates all quasimetrics and only quasimetrics**.

3.3.2 A Toy Example

To build up intuition on how various algorithms perform according to our two goals, we consider a toy quasimetric space with only 3 elements in Figure 3-2. The space has a total of 9 pairs, 8 of which form the training set. Due to quasimetric requirements (esp. triangle inequality), knowing distances of these 8 pairs restricts valid values for the heldout pair to a particular range (which is $[28, 31]$ in this case). If a model approximates 8 training pairs well *and* respects quasimetric constraints well, its prediction on that heldout pair should fall into this range.

We train three models w.r.t. mean squared error (MSE) over the training set using

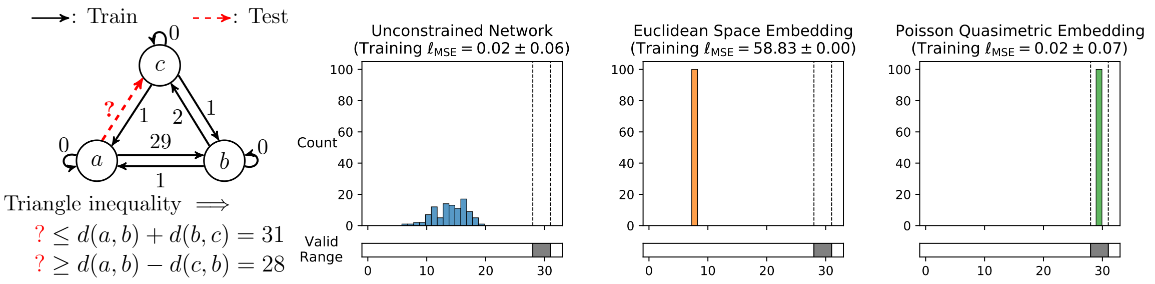


Figure 3-2: Quasimetric learning on a 3-element space. **Leftmost:** Training set contains all pairs except for (a, c). Arrow labels show quasimetric distances (rather than edge weights). A quasimetric $\hat{d}$ should predict $\hat{d}(a, c) \in [28, 30]$. **Right three:** Different formulations are trained to fit training pairs distances, and then predict on the test pair. Plots show distribution of the prediction over 100 runs.

gradient descent:

- Unconstrained deep network that predicts distance,
- Metric embedding into a latent Euclidean space with a deep encoder,
- Quasimetric embedding into a latent PQE space with a deep encoder (our method from Section 3.5).

The three approaches exhibit interesting qualitative differences. Euclidean embedding, unable to model asymmetries in training data, fails to attain a low training error. While both other methods approximate training distances well, unconstrained networks greatly violate quasimetric constraints; only PQEs respect the constraints and consistently predicts within the valid range.

Here, the structural prior of embedding into a quasimetric latent space appears important to successful learning. Without any such prior, unconstrained networks fail badly. In the next section, we present a rigorous theoretical study of the quasimetric learning task, which confirms this intuition.

3.4 Theoretical Analysis of Various Learning Algorithms

In this section, we define concrete metrics for the two quasimetric learning objectives stated above, and present positive and negative theoretical findings for various learning algorithms.

Overview. Our analysis focuses on data-agnostic bounds, which are of great interests in machine learning (e.g., VC-dimension [161]). We prove a strong negative result for a general family of learning algorithms (including unconstrained MLPs trained in NTK regime, k-nearest neighbor, and min-norm linear regression): they may arbitrarily badly fail to fit training data or respect quasimetric constraints (Theorem 3.4.6). Our informative construction reveals the core reason of their failure. Quasimetric embeddings, however, enjoy nice properties as long as they can approximate arbitrary quasimetrics, which motivates searching for "universal quasimetrics". The next section presents PQEs as such universal approximators and states their theoretical guarantees.

Assumptions. We consider quasimetric spaces $(\mathcal{X}, d)$ with $\mathcal{X} \subset \mathbb{R}^d$, finite size $n = |X| < \infty$, and finite distances (i.e., d has range $[0, \infty)$). It allows discussing deep networks which can't handle infinities well. This mild assumption can be satisfied by simply capping max distances in quasimetrics. For training, $m < n^2$ pairs are uniformly sampled as training pairs $S \subset \mathcal{X} \times \mathcal{X}$ without replacement.

In the appendix, we provide all full proofs, further discussions of our assumptions and presented results, as well as additional results concerning specific learning algorithms and settings.

3.4.1 Distortion and Violation Metrics for Quasimetric Learning

We use *distortion* as a measure of how well the distance is preserved, as is standard in embedding analyses (e.g., Bourgain [18]). In this work, we especially consider *distortion over a subset of pairs*, to quantify how well a predictor $\hat{d}$ approximates distances over the training subset S.

Definition 3.4.1 (Distortion)**.** Distortion of $\hat{d}$ over a subset of pairs $S \subset \mathcal{X} \times \mathcal{X}$ is $\mathsf{dis}_S(\hat{d}) \triangleq \big(\max_{(x,y)\in S, x\neq y} \frac{\hat{d}(x,y)}{d(x,y)}\big)\big(\max_{(x,y)\in S, x\neq y} \frac{d(x,y)}{\hat{d}(x,y)}\big)$, and its overall distortion is $\mathsf{dis}(\hat{d}) \triangleq \mathsf{dis}_{\mathcal{X}\times\mathcal{X}}(\hat{d})$.

For measuring consistency w.r.t. quasimetric constraints, we define the *(quasimetric) violation* metric. Violation focuses on *triangle inequality*, which can often be more complex (e.g., in Figure 3-2), compared to the relatively simple *non-negativity* and *Identity of Indiscernibles*.

Definition 3.4.2 (Quasimetric Violation)**.** *Quasimetric violation* (*violation* for short) of $\hat{d}$ is $\mathsf{vio}(\hat{d}) \triangleq \max_{A_1, A_2, A_3 \in \mathcal{X}} \frac{\hat{d}(A_1, A_3)}{\hat{d}(A_1, A_2) + \hat{d}(A_2, A_3)}$, where we define $\frac{0}{0} = 1$ for notation simplicity.

Both distortion and violation are nicely agnostic to scaling. Furthermore, assuming *non-negativity* and *Identity of Indiscernibles*, $\mathsf{vio}(\hat{d}) \geq 1$ always, with equality iff $\hat{d}$ is a quasimetric.

Distortion and violation also capture generalization. Because the true distance d has optimal training distortion (on S) and violation, a predictor $\hat{d}$ that does badly on either must also be far from truth.

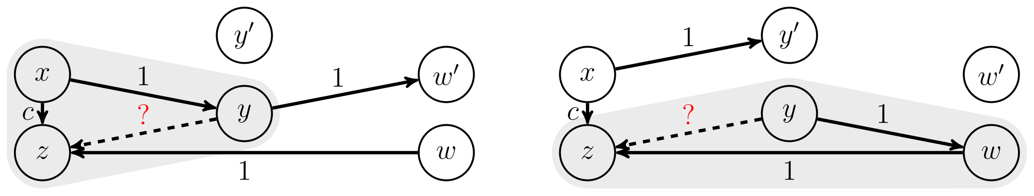
Theorem 3.4.3 (Distortion and Violation Lower-Bound Generalization Error). For non-negative $\hat{d}$, $\mathsf{dis}(\hat{d}) \geq \max(\mathsf{dis}_S(\hat{d}), \sqrt{\mathsf{vio}(\hat{d})})$, where $\mathsf{dis}(\hat{d})$ captures generalization over the entire $\mathcal{X}$ space.

3.4.2 Learning Algorithms Equivariant to Orthogonal Transforms

For quasimetric space $(\mathcal{X}, d)$, $\mathcal{X} \subset \mathbb{R}^d$, we consider applying general learning algorithms by concatenating pairs to form inputs $\in \mathbb{R}^{2d}$ (e.g., unconstrained networks). While straightforward, this approach means that algorithms are generally *unable to relate the same element appearing as 1st or 2nd input.* As we will show, this is sufficient for a wide family of learning algorithms to fail badly– ones **equivariant to orthogonal transforms** ($\mathsf{OrthEquiv}$ algorithms; Definition 3.4.4).

For an $\mathsf{OrthEquiv}$ algorithm, training on orthogonally transformed data does not affect its prediction, as long as test data is identically transformed. In fact, many standard learning algorithms are $\mathsf{OrthEquiv}$, including unconstrained MLP trained in NTK regime (Lemma 3.4.5).

Definition 3.4.4 (Equivariant Learning Algorithms)**.** Given training set $\mathcal{D} = \{(z_i, y_i)\}_i$, where $z_i \in \mathcal{Z}$ are inputs and $y_i \in \mathcal{Y}$ are targets, a learning algorithm Alg produces a function $\mathsf{Alg}(\mathcal{D})\colon \mathcal{Z} \to Y$ such that $\mathsf{Alg}(\mathcal{D})(z')$ is the function's prediction on sample z'. Consider $\mathcal{T}$ a set of transformations $\mathcal{Z} \to \mathcal{Z}$. Alg is equivariant to $\mathcal{T}$ iff for all transform $T \in \mathcal{T}$, training set $\mathcal{D}$, $\mathsf{Alg}(\mathcal{D}) = \mathsf{Alg}(T\mathcal{D}) \circ T$, where $T\mathcal{D} = \{(Tz, y)\colon (z, y) \in \mathcal{D}\}$ is the training set with transformed inputs.



$$\mathsf{vio}(\hat{d}) \geq \frac{\hat{d}(x,z)}{\hat{d}(x,y)+\hat{d}(y,z)} \geq \frac{c}{\mathsf{dis}_S(\hat{d})(\mathsf{dis}_S(\hat{d})+\hat{d}(y,z))} \qquad \mathsf{vio}(\hat{d}) \geq \frac{\hat{d}(y,z)}{\hat{d}(y,w)+\hat{d}(w,z)} \geq \frac{\hat{d}(y,z)}{2\cdot\mathsf{dis}_S(\hat{d})}$$

Training ($\longrightarrow$) : $d(x,z)=c,\quad d(w,z)=1,$ $d(x,y)=1,\quad d(y,w')=1.$ Training ($\longrightarrow$) : $d(x,z)=c,\quad d(w,z)=1,$ $d(x,y')=1,\quad d(y,w)=1.$

Test ($\dashrightarrow$) : $\hat{d}(y,z)=?$ Test ($\dashrightarrow$) : $\hat{d}(y,z)=?$

Figure 3-3: Two training sets pose incompatible constraints (●) for the test pair distance $d(y,z)$. With one-hot features, an orthogonal transform can exchange $(*,y) \leftrightarrow (*,y')$ and $(*,w) \leftrightarrow (*,w')$, leaving the test pair (y,z) unchanged, but transforming the training set from one scenario to the other. Given either set, an OrthEquiv algorithm must attain same training distortion and predict identically on (y,z). For appropriate c, this implies large distortion (not fitting training set) or violation (not approximately a quasimetric) in one of these cases.

Lemma 3.4.5 (Examples of OrthEquiv Algorithms). k-nearest-neighbor with Euclidean distance, dot-product kernel ridge regression (including min-norm linear regression and MLP trained with squared loss in NTK regime) are OrthEquiv.

Failure case. These algorithms treat the concatenated inputs as generic vectors. If a transform fundamentally changes the quasimetric structure but is not fully reflected in the learned function (e.g., due to equivariance), learning must fail. The two training sets in Figure 3-3 are sampled from two different quasimetrics over the same 6 elements An orthogonal transform links both training sets *without affecting the test pair*, which is constrained differently in two quasimetrics. An OrthEquiv algorithm, necessarily predicting the test pair identically seeing either training set, must thus fail on one. In the appendix, we empirically verify that unconstrained MLPs indeed *do fail* on this construction.

Extending to larger quasimetric spaces, we consider graphs containing many copies of *both* patterns in Figure 3-3. With high probability, our sampled training set fails in the same way—the learning algorithm can not distinguish it from another training set with different quasimetric constraints.

Theorem 3.4.6 (Failure of OrthEquiv Algorithms). Let $(f_n)_n$ be an *arbitrary* sequence of large values. There is an infinite sequence of quasimetric spaces $((\mathcal{X}_n, d_n))_n$ with $|\mathcal{X}_n| = n$, $\mathcal{X}_n \subset \mathbb{R}^n$ such that, over a random training set S of size m, any $\mathsf{OrthEquiv}$ algorithm outputs a predictor $\hat{d}$ that

- $\hat{d}$ fails *non-negativity*, or
- $\max(\mathsf{dis}_S(\hat{d}), \mathsf{vio}(\hat{d})) \geq f_n$ (i.e., $\hat{d}$ approximates training S badly or is far from a quasimetric),

with probability $1/2 - o(1)$, as long as S does not contain almost all of the pairs $1 - m/n^2 = \omega(n^{-1/3})$, and does not only include few pairs $m/n^2 = \omega(n^{-1/2})$.

Furthermore, standard NTK results show that unconstrained MLPs trained in NTK regime converge to a function with zero training loss. By the above theorem, the limiting function is not a quasimetric with nontrivial probability. In the appendix, we formally state this result. Despite their empirical usages, these results suggest that unconstrained networks are likely not suited for quasimetric learning.

3.4.3 Quasimetric Embeddings

A quasimetric embedding consists of a mapping f from data space $\mathcal{X}$ to a latent quasimetric space $(\mathcal{Z}, d_z)$, and predicts $\hat{d}(x, y) \triangleq d_z(f(x), f(y))$. Therefore, they always respect all quasimetric constraints and attain optimal violation of value 1, *regardless of training data.*

However, unlike deep networks, their distortion (approximation) properties depend on the specific latent quasimetrics. If the latent quasimetric is not overly restrictive and can approximate *any* quasimetric (with flexible learned encoders), we have nice guarantees for both distortion and violation.

In the section below, we present Poisson Quasimetric Embedding (PQE) as such a latent quasimetric, along with its theoretical distortion and violation guarantees.

3.5 Poisson Quasimetric Embeddings (PQEs)

Motivated by above theoretical findings, we aim to find a latent quasimetric space $(\mathbb{R}^d, d_z)$ with a deep network encoder $f\colon \mathcal{X} \to \mathbb{R}^d$, and a quasimetric d_z that is both *universal* and *differentiable*:

- (universality) for any data quasimetric $(\mathcal{X}, d)$, there exists an encoder f such that $d_z(f(x), f(y)) \approx d(x, y)$;
- (differentiability) d_z is differentiable (for optimizing f and possible integration with other gradient-based systems).

Notation 3.5.1. We use x, y for elements of the data space $\mathcal{X}$, u, v for elements of the latent space $\mathbb{R}^d$, upper-case letters for random variables, and $(\cdot)_z$ for indicating functions in latent space (e.g., d_z).

An existing line of machine learning research learns *quasipartitions*, or *partial orders*, via Order Embeddings [162]. Quasipartitions are in fact special cases of quasimetrics whose distances are restricted to be binary, denoted as π. An Order Embedding is a representation of a quasipartition, where $\pi^{\mathsf{OE}}(x, y) = 0$ (i.e., x is related to y) iff $f(x) \leq f(y)$ coordinate-wise:

$$\pi^{\mathsf{OE}}(x, y) \triangleq \pi_z^{\mathsf{OE}}(f(x), f(y)) \triangleq 1 - \prod_j \mathbf{1}_{f(x)_j - f(y)_j \leq 0}. \tag{3.3}$$

Order Embedding is *universal* and can model *any quasipartition* (see appendix and Hiraguchi [75]).

Can we extend this discrete idea to general continuous quasimetrics? Quite naïvely, one may attempt a straightforward soft modification of Order Embedding:

$$\pi_z^{\mathsf{SoftOE}}(u, v) \triangleq 1 - \prod_j \exp\big(-(u_j - v_j)^+\big) = 1 - \exp\Big(-\sum_j (u_j - v_j)^+\Big), \tag{3.4}$$

which equals 0 if $u \leq v$ coordinate-wise, and increases to 1 as some coordinates violate this condition more. However, it is unclear whether this gives a quasimetric.

A more principled way is to parametrize a (scaled) *distribution of latent quasipartitions* Π_z, whose expectation naturally gives a continuous-valued quasimetric:

$$d_z(u, v; \Pi_z, \alpha) \triangleq \alpha \cdot \mathbb{E}_{\pi_z \sim \Pi_z}\left[\pi_z(u, v)\right], \qquad \alpha \geq 0. \tag{3.5}$$

Poisson Quasimetric Embedding (PQE) gives a general recipe for constructing such Π_z distributions so that d_z is *universal* and *differentiable*. Within this framework, we will see that π_z^{SoftOE} is actually a quasimetric based on such a distribution and is (almost) sufficient for our needs.

3.5.1 Distributions of Latent Quasipartitions

A random latent quasipartition $\pi_z \colon \mathbb{R}^d \times \mathbb{R}^d \to \{0, 1\}$ is a difficult object to model, due to complicated quasipartition constraints. Fortunately, the Order Embedding representation (Equation (3.3)) is without such constraints. If, instead of fixed latents u, v, we have *random latents* $R(u), R(v)$, we can compute:

$$\mathbb{E}_{\pi_z}\left[\pi_z(u,v)\right] = \mathbb{E}_{R(u),R(v)}\left[\pi_z^{\mathsf{OE}}(R(u), R(v))\right] = 1 - \mathbb{P}\left[R(u) \leq R(v) \text{ coordinate-wise}\right]. \tag{3.6}$$

In this view, we represent a random π_z via a joint distribution of random vectors[1] $\{R(u)\}_{u \in \mathbb{R}^d}$, i.e., a *stochastic process*. To easily compute the probability of this coordinate-wise event, we assume that each dimension of random vectors is from an independent process, and obtain

$$\mathbb{E}_{\pi_z}\left[\pi_z(u,v)\right] = 1 - \prod_j \mathbb{P}\left[R_j(u) \leq R_j(v)\right]. \tag{3.7}$$

The choice of stochastic process is flexible. Using *Poisson processes* (with Lebesgue mean measure; Definition 3.2.3) that count random points on half-lines[2] $(-\infty, a]$, we can have $R_j(u) = N_j((\infty, u_j])$, the (random) count of events in $(\infty, u_j]$ from j-th

[1]In general, these random vectors $R(u)$ do not have to be of the same dimension as $u \in \mathbb{R}^d$, although the dimensions do match in the PQE variants we experiment with.

[2]Half-lines has Lebesgue measure ∞. More rigorously, consider using a small value as the lower bounds of these intervals, which leads to same result.

Poisson process:

$$\mathbb{E}_{\pi_z \sim \Pi_z}\left[\pi_z(u,v)\right] = 1 - \prod_j \mathbb{P}\big[N_j((-\infty, u_j]) \leq N_j((-\infty, v_j])\big] \tag{3.8}$$

$$= 1 - \prod_j \exp\big(-(u_j - v_j)^+\big) = \pi_z^{\mathsf{SoftOE}}(u,v), \tag{3.9}$$

where we used Fact 3.2.4 and the observation that one half-line is either subset or superset of another. Indeed, π_z^{SoftOE} is an expected quasipartition (and thus a quasimetric), and is *differentiable.*

Considering a mixture of such distributions for expressiveness, the full latent quasimetric formula is

(3.10)

where we slightly abuse notation and consider latents u and v as (reshaped to) 2-dimensional. We will see that this is a special PQE case with **L**ebesgue measure and **h**alf-lines, and thus denoted PQE-LH.

3.5.2 General PQE Formulation

We can easily generalize the above idea to independent Poisson processes of general mean measures μ_j and (sub)set parametrizations $u \to A_j(u)$, and obtain an expected quasipartition as:

$$\mathbb{E}_{\pi_z \sim \Pi_z^{\mathsf{PQE}}(\mu, A)}[\pi_z(u,v)] \tag{3.11}$$

$$\triangleq 1 - \prod_j \mathbb{P}\left[N_j(A_j(u)) \leq N_j(A_j(v))\right] \tag{3.12}$$

$$= 1 - \prod_j \mathbb{P}\Big[\mathrm{Pois}(\underbrace{\mu_j(A_j(u) \setminus A_j(v))}_{\text{Poisson rate of points landing only in } A_j(u)}) \leq \mathrm{Pois}(\mu_j(A_j(v) \setminus A_j(u)))\Big], \tag{3.13}$$

which is *differentiable* as long as the measures and set parametrizations are (after set differences). Similarly, considering a mixture gives us an expressive latent quasimetric.

A general PQE latent quasimetric is defined with $\{(\mu_{i,j}, A_{i,j})\}_{i,j}$ and weights $\alpha_i \geq 0$

as:

$$\begin{aligned}
& d_z^{\mathsf{PQE}}(u, v; \mu, A, \alpha) \\
& \triangleq \sum_i \alpha_i \cdot \mathbb{E}_{\pi_z \sim \Pi_z^{\mathsf{PQE}}(\mu_i, A_i)} \left[\pi_z(u, v) \right] \qquad (3.14) \\
& = \sum_i \alpha_i \Big(1 - \prod_j \mathbb{P}\Big[\mathrm{Pois}(\mu_{i,j}(A_{i,j}(u) \setminus A_{i,j}(v))) \leq \mathrm{Pois}(\mu_{i,j}(A_{i,j}(v) \setminus A_{i,j}(u))) \Big] \Big),
\end{aligned}$$

whose optimizable parameters include $\{\alpha_i\}_i$, possible ones from $\{(\mu_{i,j}, A_{i,j})\}_{i,j}$ (and encoder f).

This general recipe can be instantiated in many ways. Setting $A_{i,j}(u) \rightarrow (-\infty, u_{i,j}]$ and Lebesgue $\mu_{i,j}$, recovers PQE-LH. In the appendix, we consider a form with **G**aussian-based measures and **G**aussian-shapes, denoted as PQE-GG. Unlike PQE-LH, PQE-GG always gives nonzero gradients.

The appendix also includes several implementation techniques that empirically improve stability, including learning α_i's with deep linear networks, a formulation that outputs discounted distance, etc.

3.5.3 Continuous-valued Stochastic Processes

But why Poisson processes over more common choices such as Gaussian processes? It turns out that common continuous-value processes fail to give a *differentiable* formula.

Consider a non-degenerate process $\{R(u)\}_u$, where $(R(u), R(v))$ has bounded density if $u \neq v$. Perturbing $u \rightarrow u + \delta$ leaves $\mathbb{P}[R(u) = R(u + \delta)] = 0$. Then one of $\mathbb{P}\big[R(u) \leq R(u + \delta)\big]$ and $\mathbb{P}\big[R(u + \delta) \leq R(u)\big]$ must be far away from 1 (as they sum to 1), breaking differentiability at $\mathbb{P}[R(u) \leq R(u)] = 1$. (This argument is formalized in the appendix.) Discrete-valued processes, however, can leave most probability mass on $R(u) = R(u + \delta)$ and thus remain differentiable.

3.5.4 Theoretical Guarantees

Our PQEs bear similarity with the algorithmic quasimetric embedding construction in Mémoli et al. [113]. Extending their analysis to PQEs, we obtain the following distortion and violation guarantees.

Theorem 3.5.2 (Distortion and violation of PQEs). Under the assumptions of Section 3.4, *any* quasimetric space with size n and treewidth t admits a PQE-LH and a PQE-GG with distortion $\mathcal{O}(t \log^2 n)$ and violation 1, with an expressive encoder (e.g., a ReLU network with ≥ 3 layers and polynomial width).

In fact, these guarantees apply to any PQE formulation that satisfies a mild condition. Informally, any PQE with $h \times k$ Poisson processes (i.e., h mixtures) enjoys the above guarantees if it can approximate the discrete counterpart: mixtures of h Order Embeddings, each specified with k dimensions. In the appendix, we make this condition precise and provide a full proof of the above theorem.

3.6 Experiments

Our experiments are designed to (1) confirm our theoretical findings and (2) compare PQEs against a wider range of baselines, across different types of tasks. In all experiments, we optimize γ-discounted distances (with $\gamma \in \{0.9, 0.95\}$), and compare the following five families of methods:

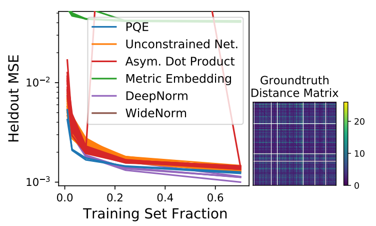
- **PQEs (2 formulations):** PQE-LH and PQE-GG with techniques mentioned in Section 3.5.2.
- **Unconstrained networks (20 formulations):** Predict raw distance (directly, with exp transform, and with $(\cdot)^2$ transform) or γ-discounted distance (directly, and with a sigmoid-transform). Each variant is run with a possible triangle inequality regularizer $\mathbb{E}_{x,y,z}\big[\max(0, \gamma^{\hat{d}(x,y)+\hat{d}(y,z)} - \gamma^{\hat{d}(x,z)})^2\big]$ for each of 4 weights $\in \{0, 0.3, 1, 3\}$.
- **Asymmetrical dot products (20 formulations):** On input pair (x, y), encode each into a feature vector with a *different* network, and take the dot product. Identical to unconstrained networks, the output is used in the same 5 ways, with the same 4 triangle inequality regularizer options.

- **Metric encoders (4 formulations):** Embed into Euclidean space, ℓ_1 space, hypersphere with (scaled) spherical distance, or a mixture of all three.
- **DeepNorm (2 formulations) and WideNorm (3 formulations):** Quasimetric embedding methods that often require significantly more parameters than PQEs (often on the order of $10^6 \sim 10^7$ more effective parameters; see the appendix for detailed comparisons) but can only approximate a subset of all possible quasimetrics [127].

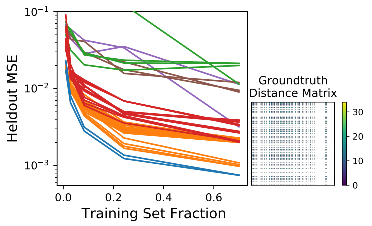
We show average results from 5 runs. The appendix provides experimental details, full results (including standard deviations), additional experiments, and ablation studies.

Random directed graphs. We start with randomly generated directed graphs of 300 nodes, with 64-dimensional node features given by randomly initialized neural networks. After training with MSE on discounted distances, we test the models' prediction error on the unseen pairs (i.e., generalization), measured also by MSE on discounted distances. On three graphs with distinct structures, PQEs significantly outperform baselines across almost all training set sizes (see Figure 3-4). Notably, while DeepNorm and WideNorm do well on the dense graph quasimetric, they struggle on the other two, attaining both high test MSE (Figure 3-4) and train MSE (not shown). This is consistent with the fact that they can only approximate a subset of all quasimetrics, while PQEs can approximate all quasimetrics.

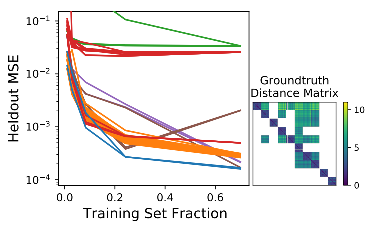
Large-scale social graph. We choose the Berkeley-Stanford Web Graph [101] as the real-wold social graph for evaluation. This graph consists of 685,230 pages as nodes, and 7,600,595 hyperlinks as directed edges. We use 128-dimensional node2vec features [55] and the landmark method [132] to construct a training set of 2,500,000 pairs, and a test set of 150,000 pairs. PQEs generally perform better than other methods, accurately predicting finite distances while predicting high values for infinite distances (see Table 3.1). DeepNorms and WideNorms learn finite distances less accurately here, and also do much worse than PQEs on learning the (quasi)metric of an *undirected*



(a) A dense graph.



(b) A sparse graph.



(c) A sparse graph with block structure.

Figure 3-4: Comparison of PQE and baselines on quasimetric learning in random directed graphs.

	Triangle inequality regularizer	MSE w.r.t. γ-discounted distances ($\times 10^{-3}$) $\downarrow$	L1 Error when true $d < \infty \downarrow$	Prediction $\hat{d}$ when true $d = \infty \uparrow$
PQE-LH	✗	3.043	1.626	69.942
PQE-GG	✗	3.909	1.895	101.824
Best Unconstrained Net.	✗	3.086	2.115	59.524
	✓	2.813	2.211	61.371
Best Asym. Dot Product	✗	48.106	2.520×10^{11}	2.679×10^{11}
	✓	48.102	2.299×10^{11}	2.500×10^{11}
Best Metric Embedding	✗	17.595	7.540	53.850
Best DeepNorm	✗	5.071	2.085	120.045
Best WideNorm	✗	3.533	1.769	124.658

Table 3.1: Quasimetric learning on large-scale web graph. "Best" is selected by *test* MSE w.r.t. γ-discounted distances.

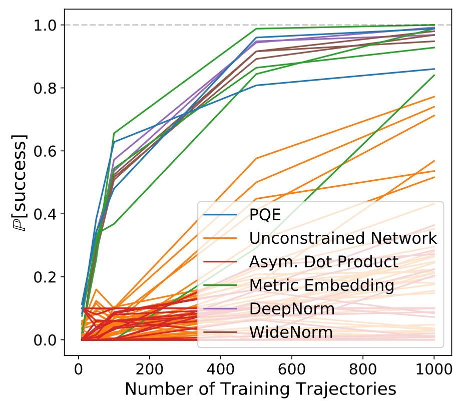


Figure 3-5: Offline Q-learning results.

social graph (shown in the appendix).

Offline Q-learning. Optimal goal-reaching plan costs in MDPs are quasimetrics [12, 151] (see also the appendix). In practice, optimizing deep Q-functions often suffers from stability and sample efficiency issues [73, 46]. As a proof of concept, we use PQEs as goal-conditional Q-functions in offline Q-learning, on the grid-world environment with one-way doors built upon `gym-minigrid` [28] (see Figure 3-1 right), following the algorithm and data sampling procedure described in Tian et al. [151]. Adding strong quasimetric structures greatly improves sample efficiency and greedy planning success rates over popular existing approaches such as unconstrained networks used in Tian et al. [151] and asymmetrical dot products used in Schaul et al. [136] (see Figure 3-5). As an interesting observation, some metric embedding formulations work comparably well.

3.7 Related Work

Metric learning. Metric learning aims to approximate a target metric/similarity function, often via a learned embedding into a metric space. This idea has successful applications in dimensionality reduction [149], information retrieval [166], clustering [172], classification [169, 77], etc. While asymmetrical formulations have been explored, they either ignore quasimetric constraints [119, 106, 136], or are not general enough to approximate arbitrary quasimetric [8], which is the focus of the present paper.

Isometric embeddings. Isometric (distance-preserving) embeddings is a highly influential and well-studied topic in mathematics and statistics. Fundamental results, such as Bourgain's random embedding theorem [18], laid important ground work in understanding and constructing (approximately) isometric embeddings. While most such researches concern metric spaces, Mémoli et al. [113] study an algorithmic construction of a quasimetric embedding via basic blocks called *quasipartitions*. Their approach requires knowledge of quasimetric distances between all pairs and thus is not suitable for learning. Our formulation takes inspiration from the form of their embedding, but is fully learnable with gradient-based optimization over a training subset.

Quasimetrics and partial orders. Partial orders (quasipartitions) are special cases of quasimetrics (see Section 3.5). A line of machine learning research studies embedding partial order structures into latent spaces for tasks such as relation discovery and information retrieval [162, 147, 68, 47]. Unfortunately, unlike PQEs, such formulations do not straightforwardly generalize to arbitrary quasimetrics, which are more than binary relations. Similar to PQEs, DeepNorm and WideNorm are quasimetric embedding approaches learnable with gradient-based optimization [127]. Theoreically, they universally approximates a subset of quasimetrics (ones induced by asymmetrical norms). Despite often using many more parameters, they are restricted to this subset and unable to approximate general quasimetrics like PQEs do (Figure 3-4).

3.8 Implications

In this work, we study quasimetric learning via both theoretical analysis and empirical evaluations.

Theoretically, we show strong negative results for a common family of learning algorithms, and positive guarantees for our proposed Poisson Quasimetric Embedding (PQE). Our results introduce the novel concept of equivariant learning algorithms, which may potentially be used for other learnability analyses with algorithms such as deep neural networks. Additionally, a thorough average-case or data-dependent analysis would nicely complement our results, and may shed light on conditions where algorithms like deep networks can learn decent approximations to quasimetrics in practice.

PQEs are the first quasimetric embedding formulation that can be learned via gradient-based optimization. Empirically, PQEs show promising performance in various tasks. Furthermore, PQEs are fully differentiable, and (implicitly) enforce a quasimetric structure in any latent space. They are particularly suited for integration in large deep learning systems, as we explore in the Q-learning experiments. This can potentially open the gate to many practical applications such as better embedding for planning with MDPs, efficient shortest path finding via learned quasimetric heuristics, representation learning with quasimetric similarities, causal relation learning, etc.

Chapter 4

Denoised MDPs: Learning World Models Better Than the World Itself

The ability to separate signal from noise, and reason with clean abstractions, is critical to intelligence. With this ability, humans can efficiently perform real world tasks without considering all possible nuisance factors. How can artificial agents do the same? What kind of information can agents safely discard as noises? In this chapter, we categorize information out in the wild into four types based on controllability and relation with reward, and formulate useful information as that which is both *controllable* and *reward-relevant*. This framework clarifies the kinds information removed by various prior work on representation learning in reinforcement learning (RL), and leads to our proposed approach of learning a *Denoised MDP* that explicitly factors out certain noise distractors. Extensive experiments on variants of DeepMind Control Suite and `RoboDesk` demonstrate superior performance of our denoised world model over using raw observations alone, and over prior works, across policy optimization control tasks as well as the non-control task of joint position regression.

4.1 Introduction

The real world provides us a plethora of information, from microscopic physical interactions to abstracted semantic signals such as the latest COVID-19 news. Fortunately, processing each and every signal is unnecessary (and also impossible). In fact, any particular reasoning or decision often only relies on a small portion of information.

Imagine waking up and wanting to embrace some sunlight. As you open the curtain, a nearby resting bird is scared away and you are pleasantly met with a beautiful sunny day. Far away, a jet plane is slowly flying across the sky.

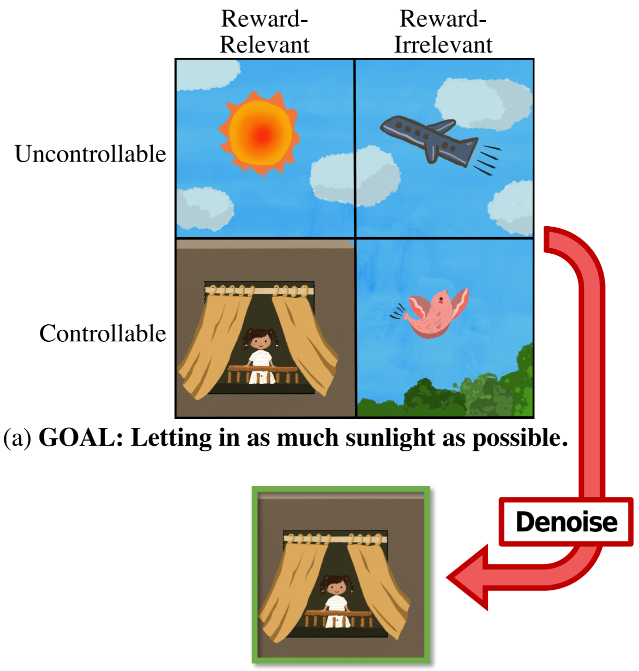
This may seem a simple activity, but in fact highlights four distinct types of information (see Figure 4-1), with respect to the goal of letting in as much sunlight as possible:

- **Controllable and reward-relevant**: curtain, influenced by actions and affecting incoming sunlight;
- **Controllable and reward-irrelevant**: bird, influenced by actions but not affecting sunlight;
- **Uncontrollable and reward-relevant**: weather, independent with actions but affecting sunlight;
- **Uncontrollable and reward-irrelevant**: plane, independent with both actions and the sunlight.

Our optimal actions towards the goal, however, only in fact depend on information that is **controllable and reward-relevant**, and the three other kinds of information are merely *noise distractors*. Indeed, no matter how much natural sunlight there is outside, or how the plane and the bird move, the best plan is always to open up the curtain.

When performing a particular task, we humans barely think about the other three types of information, and usually only plan on how our actions affect information that is **controllable and reward-relevant**. Our mental model is an abstract and condensed version of the real world that is actually *better* suited for the task.

The notion of better model/data is ubiquitous in data science and machine learning.



(a) **GOAL: Letting in as much sunlight as possible.**

(b) Optimal control only relies on information that is **both controllable and reward-relevant**. Good world models should ignore other factors as noisy distractors.

Figure 4-1: **Illustrative example: (a)** Four distinct kinds of information in the scenario described in Section 4.1, where the person desires to increase the amount of sunlight let into the room. Their opening of the curtain scares away the bird. **(b)** A denoised world model only includes a small subset of all information.

Algorithms rarely perform well on raw noisy real data. The common approach is to perform data cleaning and feature engineering, where we manually select the useful signals based on prior knowledge and/or heuristics. Years of research have identified ways to extract good features for computer vision [108, 37], natural language processing [40, 115], reinforcement learning (RL) [109, 9], etc. Similarly, system identification aligns real observation with a predefined set of abstract signals/states. Yet for tasks in the wild (in the general form of (partially observable) Markov Decision Processes), there can be very little prior knowledge of the optimal set of signals. In this work, we ask: can we infer and extract these signals automatically, in the form of a learned world model?

The general idea of a mental world model have long been under active research in philosophy and social science [33, 36], cognitive science, where an intuitive physics model is hypothesized to be core in our planning capabilities [141], and in reinforcement learning, where various methods investigate state abstractions for faster and better learning [144, 143].

In this work, we explore this idea within the context of machine learning and reinforcement learning, where we aim to make concrete the different types of information in the wild, and automatically learn a world model that removes noise distractors and is beneficial for both control (i.e., policy optimization) and non-control tasks. Toward this goal, our contributions are

- We categorize information into four distinct kinds as in Figure 4-1, and review prior approaches under this framework (Section 4.2).
- Based on the above framework, we propose Denoised MDPs, a method for learning world models with certain distractors removed (Section 4.3).
- Through experiments in DeepMind Control Suite and `RoboDesk` environments, we demonstrate superior performance of policies learned our method, across many distinct types of noise distractors (Sections 4.5.1 and 4.5.2).
- We show that Denoised MDP is also beneficial beyond control objectives, improving the supervised task of robot joint position regression (Section 4.5.1).

4.2 Different Types of Information in the Wild

In Section 4.1, we illustrated the four types of information available in the wild w.r.t. a task. Here we make these notions more concrete, and relate them to existing works.

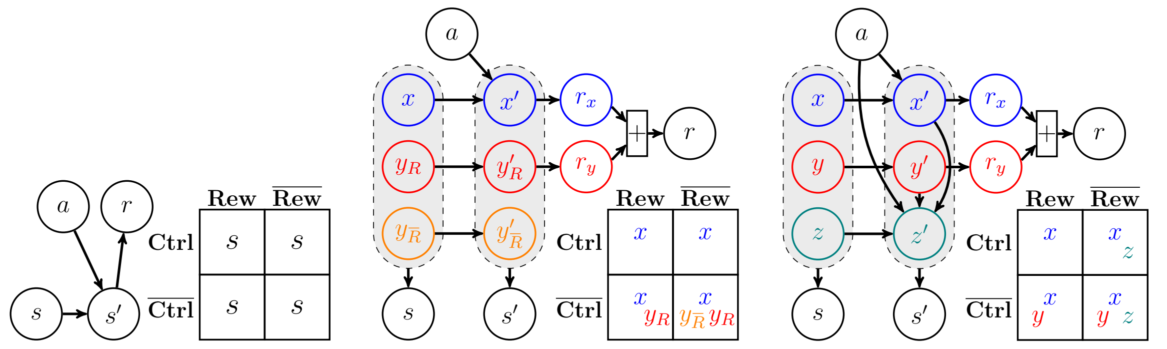
For generality, we consider tasks in the form of Markov Decision Processes (MDPs), described in the usual manner: $\mathcal{M} \triangleq (\mathcal{S}, \mathcal{A}, R, P, p_{s_0})$ [130], where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $R \colon \mathcal{S} \rightarrow \Delta([0, r_{\mathsf{max}}])$ defines the reward random variable $R(s')$ received for arriving at state $s' \in \mathcal{S}$, $P \colon \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ is the transition dynamics, and $p_{s_0} \in \Delta(\mathcal{S})$ defines the distribution of initial state. We use $\Delta(A)$ to denote the set of all distributions over A. P and R define the most important components of a MDP: the transition dynamics $\mathbb{P}[s' \mid s, a]$ and the reward function $\mathbb{P}[r \mid s']$. Usually, the objective is to find a policy $\pi \colon \mathcal{S} \rightarrow \Delta(\mathcal{A})$ acting based on current state, that maximizes the expected cumulative (discounted) reward.

Indeed, MDPs provide a general formulation that encompasses many tasks. In fact, the entire real world may be viewed as an MDP with a rich state/observation space $\mathcal{S}$ that contains all possible information/signal. For an artificial agent to successfully perform real world tasks, it must be able to process observations that are incredibly rich and high-dimensional, such as visual or audio signals.

We characterize different types of information in such observations by considering two intuitive notions of "noisy and irrelevant" signals: (1) uncontrollable information and (2) reward-irrelevant information. Such factors can often be ignored without affecting optimal control, and are referred to as *noise distractors.*

To understand their roles in MDPs, we study different formulations of the transition dynamics and reward functions, and show how different structures naturally leads to decompositions that may help identify such distractors. Removing these distractors can thus *transform the original noisy MDP to a clean denoised one*, to be used in downstream tasks.

For starters, the most generic transition model in Figure 4-2a has little to no structure. The state s can contain both the useful signals and noise distractors. Therefore, it is not directly useful for extracting important information.



(a) Transition without useful structure. s may contain any type of information.

(b) Transition that factorizes out uncontrollable information in $y_{\bar{R}}$ and y_R.

(c) Transition that factorizes out uncontrollable y and reward-irrelevant z.

Figure 4-2: MDP transition structures consisting of dynamics and reward functions. Unlike the regular structure of **(a)**, **(b, c)** factorized (yet still general) structures inherently separate information into controllable (**Ctrl**) versus uncontrollable ($\overline{\textbf{Ctrl}}$), and reward-relevant (**Rew**) versus reward-irrelevant ($\overline{\textbf{Rew}}$). Presence of a variable in a cell means *possible* containing of respective information. E.g., in **(c)**, z can only contain reward-irrelevant information. In **(b, c)**, the x dynamics form an MDP with less noise and sufficient for optimal planning. Our Denoised MDP (see Section 4.3) is based on these two factorizations.

4.2.1 Controllability

Intuitively, if something is not controllable, an agent might be able to do well without considering it. Yet it is not enough to only require some variable to be unaffected by actions (e.g., wind directions should not be ignored while sailing). Instead, we focus on factors that simply evolve on their own, without influencing or being influenced by others.

Not all such information can be safely ignored, as they still may affect reward (e.g., traffic lights when driving). Fortunately, in the usual objective of maximizing expected return, we can ignore ones that only additively affect reward.

Concretely, if an MDP transition can be represented in the form of Figure 4-2b, we say variables $y_{\bar{R}}$ and y_R are *uncontrollable* information, as they evolve independently of actions and do not affect *controllable* x. Here y_R (additively) affects reward, but can be ignored. One can safely discard both $y_{\bar{R}}$ and y_R as *noise distractors*. Operating with the compressed MDP of only x is sufficient for optimal control.

4.2.2 Reward-Relevance

Among controllable information, there can still be some that is completely unrelated to reward. In Figure 4-1, the bird is affected by the opening curtain, but is irrelevant to the task of letting in sunlight. In such cases, the information can be safely discarded, as it does not affect the objective.

If an MDP transition can be represented in the form of Figure 4-2c, we say z is *reward-irrelevant* because it evolves by potentially using everything (i.e., all latent variables and actions), but crucially *does not affect anything but itself.*

Similar to uncontrollable information, z (and y) is a *noise distractor* that can be discarded. The compressed MDP of only x contains all signals needed for optimal control.

4.2.3 Which Information Do Existing Methods Learn?

In RL, many prior work have explored state abstractions in some form. Here we cast several representative ones under the framework described above, and show which kinds of information they learn to remove, summarized in Figure 4-3, together with our proposed method (explained in Section 4.3). Below we discuss each prior work in detail.

Reconstruction-Based Model-Based RL. Many model-based RL methods learn via reconstruction from a single latent code, often as a result of a variational formulation [63, 64, 99]. The latent code must try to compress all information present in the observation, and necessarily contains all types of information.

Bisimulation. Bisimulation defines a state abstraction where states aggregated together must have the same expected return and transition dynamics up to the abstraction [50], and is known to optimally ignore reward-irrelevant information [44]. While its continuous version, bisimilation metric, is gaining popularity, learning them is computationally difficult [117]. Even with many additional assumptions, it is generally only possible to learn an on-policy variant that loses the above guarantee [22, 176].

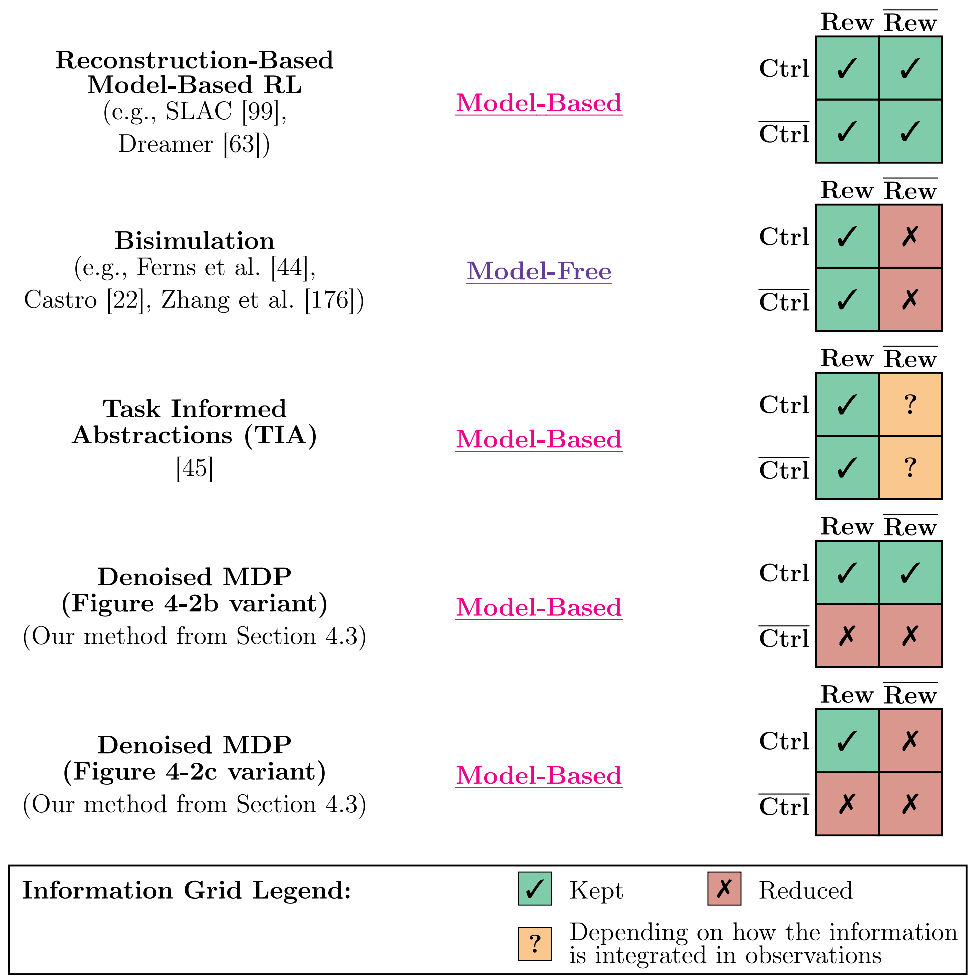

Reconstruction-Based Model-Based RL (e.g., SLAC [99], Dreamer [63])
Model-Based
Rew $\overline{\text{Rew}}$
Ctrl ✓ ✓
$\overline{\text{Ctrl}}$ ✓ ✓
Bisimulation (e.g., Ferns et al. [44], Castro [22], Zhang et al. [176])
Model-Free
Rew $\overline{\text{Rew}}$
Ctrl ✓ ✗
$\overline{\text{Ctrl}}$ ✓ ✗
Task Informed Abstractions (TIA) [45]
Model-Based
Rew $\overline{\text{Rew}}$
Ctrl ✓ ?
$\overline{\text{Ctrl}}$ ✓ ?
Denoised MDP (Figure 4-2b variant) (Our method from Section 4.3)
Model-Based
Rew $\overline{\text{Rew}}$
Ctrl ✓ ✓
$\overline{\text{Ctrl}}$ ✗ ✗
Denoised MDP (Figure 4-2c variant) (Our method from Section 4.3)
Model-Based
Rew $\overline{\text{Rew}}$
Ctrl ✓ ✗
$\overline{\text{Ctrl}}$ ✗ ✗
Information Grid Legend:
✓ Kept
✗ Reduced
? Depending on how the information is integrated in observations

Figure 4-3: Categorization of information learned and removed by various methods with distinct formulations.

Task Informed Abstractions (TIA). TIA [45] extends Dreamer by modelling two independent latent MDPs, representing signal and noise. The noise latent is enforced to be independent with reward and reconstruct the observation as well as possible. Reconstructions from each latent are composed together using an inferred mask in pixel-space, to form the full reconstruction for the reconstruction loss. Because of its special structure, TIA can remove *reward-irrelevant* noise distractors that are present via pixel-wise composing two images from *independent* processes (e.g., agent moving on a noisy background), but not general ones (e.g., a shaky camera affecting both the agent and the noisy background).

Predictive Information, Data Augmentation, etc. Another set of researches learn state representation that only contains information useful for predicting future states (e.g., CPC [119] and PI-SAC [100]) or augmented views of the current state (e.g., CURL [97]). These methods *do not guarantee* removal of any of the three redundant piece of information identified above. Non-i.i.d. noises (e.g., people moving in background) are predictive of future and may be kept by CPC and PI-SAC. The performance of augmentation-based methods can critically rely on specific types of augmentation used and relevance to the tasks. As we show in experiments (see Section 4.5), indeed they struggle to handle certain noise types.

4.2.4 Possible Extensions to Further Factorizations

The above framework is sufficient for characterizing most prior work and related tasks, and can also be readily extended with further factorized transition structures. E.g., if an independent process confounds a signal process and a noise process, fitting the Figure 4-2c structure must group all three processes into x (to properly model the dependencies). However, a further factorization shows that only considering the signal and the confounding processes is theoretically sufficient for control. We leave such extensions as future work.

4.3 Denoised MDPs

Figures 4-2b and 4-2c show two special MDP structures that automatically identify certain information that can be ignored, leaving x as the useful information (which also forms an MDP). This suggests a naïve approach: directly fitting such structures to collected trajectories, and then extract x.

However, the same MDP dynamics and rewards can be decomposed as Figures 4-2b and 4-2c in many different ways. In the extreme case, x may even contain all information in the raw state s, and such extraction may not help at all. Instead, we desire a fit with the *minimal* x, defined as being least informative of s (so that removal of the other latent variables discards the most information possible). Concretely, we aim for a fit with least $I(\{x_t\}_{t=1}^T; \{s_t\}_{t=1}^T \mid \{a_t\}_{t=1}^T)$, the mutual information x contains about s over T steps. Then from this fit, we can extract a minimal *Denoised MDP* of only x. For notation simplicity, we use bold symbols to denote variable sequences, and thus write, e.g., $I(\boldsymbol{x}; \boldsymbol{s} \mid \boldsymbol{a})$.

Practically, we consider regularizing model-fitting with $I(\boldsymbol{x}; \boldsymbol{s} \mid \boldsymbol{a})$. As we show below, this amounts to a modification to the well-established variational objective [63]. The resulting method is easy-to-implement yet effective, enabling clean removal of various noise distractors the original formulation cannot handle (see Section 4.5).

We instantiate this idea with the structure in Figure 4-2c. The Figure 4-2b formulation can be obtained by simply removing the z components and viewing y as combined y_R and $y_{\overline{R}}$.

The transition structure is modeled with components:

$$
\begin{aligned}
p_\theta^{(x_t)} &\triangleq p_\theta(x_t \mid x_{t-1}, a) && (x \text{ dynamics}) \\
& \quad\; p_\theta(r_x \mid x_t) && (x \text{ reward}) \\
p_\theta^{(y_t)} &\triangleq p_\theta(y_{t-1} \mid y_{t-1}) && (y \text{ dynamics}) \\
& \quad\; p_\theta(r_y \mid y_t) && (y \text{ reward}) \\
p_\theta^{(z_t)} &\triangleq p_\theta(z_t \mid x_t, y_t, z_{t-1}, a) && (z \text{ dynamics}) \\
& \quad\; p_\theta(s_t \mid x_t, y_t, z_t). && (\text{obs. emission})
\end{aligned}
$$

Consider training data in the form of trajectory segments $\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r}$ sampled from some data distribution p_{data} (e.g., stored agent experiences from a replay buffer). We perform model learning by minimizing the negative log likelihood:

$$
\mathcal{L}_{\mathsf{MLE}}(\theta) \triangleq -\mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{r} \sim p_{\mathsf{data}}} \big[\log p_\theta \left(\boldsymbol{s}, \boldsymbol{r} \mid \boldsymbol{a} \right) \big].
$$

To obtain a tractable form, we jointly learn three variational posterior components (i.e., encoders):

$$q_\psi^{(x_t)} \triangleq q_\psi(x_t \mid x_{t-1}, y_{t-1}, z_{t-1}, s_t, a_t) \qquad (x \text{ posterior})$$

$$q_\psi^{(y_t)} \triangleq q_\psi(y_t \mid x_{t-1}, y_{t-1}, z_{t-1}, s_t, a_t) \qquad (y \text{ posterior})$$

$$q_\psi^{(z_t)} \triangleq q_\psi(z_t \mid x_t, y_t, s_t, a_t), \qquad (z \text{ posterior})$$

whose product defines the posterior $q_\psi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{s}, \boldsymbol{a})$[1]. We choose this factorized form based on the forward (prior) model structure of Figure 4-2c.

Then, the model can be optimized w.r.t. the standard variational bound on log likelihood:

$$\begin{aligned}\mathcal{L}_{\mathsf{MLE}}(\theta) = \min_\psi \mathop{\mathbb{E}}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{r}} \mathop{\mathbb{E}}_{\substack{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}\sim \\ q_\psi(\cdot\mid \boldsymbol{s},\boldsymbol{a},\boldsymbol{r})}} \Bigg[& \underbrace{-\log p_\theta(\boldsymbol{s}, \boldsymbol{r} \mid \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, \boldsymbol{a})}_{\triangleq \mathcal{L}_{\mathsf{recon}}(\theta,\psi)} \\ & + \underbrace{\sum_{t=1}^T D_{\mathsf{KL}}\big(q_\psi^{(x_t)} \parallel p_\theta^{(x_t)}\big)}_{\triangleq \mathcal{L}_{\mathsf{KL}\text{-}x}(\theta,\psi)} + \underbrace{\sum_{t=1}^T D_{\mathsf{KL}}\big(q_\psi^{(y_t)} \parallel p_\theta^{(y_t)}\big)}_{\triangleq \mathcal{L}_{\mathsf{KL}\text{-}y}(\theta,\psi)} \\ & + \underbrace{\sum_{t=1}^T D_{\mathsf{KL}}\big(q_\psi^{(z_t)} \parallel p_\theta^{(z_t)}\big)}_{\triangleq \mathcal{L}_{\mathsf{KL}\text{-}z}(\theta,\psi)} \Bigg],\end{aligned} \tag{4.1}$$

where equality is attained by optimal q_ψ that is compatible with p_θ, i.e., q_ψ is the exact posterior of p_θ.

The mutual information regularizer $I(\boldsymbol{x}; \boldsymbol{s} \mid \boldsymbol{a})$, using a variational formulation, can be written as

$$I(\boldsymbol{x}; \boldsymbol{s} \mid \boldsymbol{a}) = \min_\theta \mathcal{L}_{\mathsf{KL}\text{-}x}(\theta, \psi), \tag{4.2}$$

with equality attained when q_ψ and p_θ are compatible. The appendix describes this derivation in detail.

Therefore, for a regularizer weight of $c \geq 0$, we can optimize Equations (4.1)

[1]Following Dreamer [63], we define posterior of first-step latents $q_\psi(x_1, y_1, z_1 \mid s_1) \triangleq q_\psi(\cdot, \cdot, \cdot \mid \mathbf{0}, \mathbf{0}, \mathbf{0}, s_1, \mathbf{0})$, where $\mathbf{0}$ is the all zeros vector of appropriate size.

and (4.2) together as

$$\begin{aligned}
&\min_{\theta} \mathcal{L}_{\mathsf{MLE}}(\theta) + c \cdot I(\boldsymbol{x}; \boldsymbol{s} \mid \boldsymbol{a}) \\
&= \min_{\theta,\psi} \mathcal{L}_{\mathsf{recon}}(\theta,\psi) + (1+c) \cdot \mathcal{L}_{\mathsf{KL}\text{-}x}(\theta,\psi) \\
&\qquad + \mathcal{L}_{\mathsf{KL}\text{-}y}(\theta,\psi) + \mathcal{L}_{\mathsf{KL}\text{-}z}(\theta,\psi).
\end{aligned} \tag{4.3}$$

Recall that we fit to the true MDP with the structure of Figure 4-2c, which inherently guarantees all useful information in the x latent variable. As the regularizer ensures learning the *minimal* x latents, the learned model extracts an MDP of condensed useful information with $\mathcal{X}$ as the *denoised* state space, $p_\theta(x' \mid x, a)$ as the transition dynamics, $p_\theta(r_x \mid x')$ as the reward function. This MDP is called the *Denoised MDP*, as it discards the noise distractors contained in y and z. Additionally, we also obtain $q_\psi(\boldsymbol{x} \mid \boldsymbol{s}, \boldsymbol{a})$ as the encoder mapping from raw noisy observation s to the denoised x.

A loss variant for improved stability. When using a large $c \geq 0$ (e.g. when the environment is expected to be very noisy), Equation (4.3) contains to a term with a large weight. Thus Equation (4.3) often requires learning rates to be tuned for different c. To avoid this, we use the following loss form that empirically has better training stability and does not require tuning learning rates w.r.t. other hyperparameters:

$$\min_{\theta,\psi} \mathcal{L}_{\mathsf{recon}} + \alpha \cdot \left(\mathcal{L}_{\mathsf{KL}\text{-}x} + \beta\mathcal{L}_{\mathsf{KL}\text{-}y} + \beta\mathcal{L}_{\mathsf{KL}\text{-}z}\right), \tag{4.4}$$

where θ, ψ in arguments are omitted, and the hyperparameters are $\alpha > 0$ and $0 < \beta \leq 1$. Here β is bounded, where $\beta = 1$ represents no regularization. α is also generally small and simply chosen according to the state-space dimensionality (see the appendix; $\alpha \in \{1, 2\}$ in our experiments). This form is justified from the observation that in practice we use isotropic Gaussians with fixed variance to parameterize the distributions of observation $p_\theta(s \mid \dots)$ and reward $p_\theta(r \mid \dots)$, where scaling log likelihoods is essentially changing the variance hyperparameter. Thus, Equation (4.4) is effectively a scaled Equation (4.3) with different variance hyperparameters.

Algorithm 1 Denoised MDP

```
Input: Model $p_\theta$. Posterior encoder $q_\psi$. Policy $\pi \colon \mathcal{X} \to \Delta(\mathcal{A})$.
       Policy optimization algorithm Pi-Opt.
Output: Denoised MDP of $x$ in $p_\theta$. Encoder $q_\psi$. Policy $\pi$.
 1: while training do
 2:     // Exploration
 3:     Collect trajectories with $\pi$ acting on $q_\psi$ encoded outputs
 4:     // Model learning
 5:     Sample a batch of $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r})$ segments from reply buffer
 6:     Train $p_\theta$ and $q_\psi$ with Equation (4.4) on $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r})$
 7:     // Policy optimization
 8:     Sample $\boldsymbol{x} \sim q_\psi(\boldsymbol{x} \mid \boldsymbol{s}, \boldsymbol{a})$
 9:     Compute $\overline{\boldsymbol{r}_{\boldsymbol{x}}} = \mathbb{E}\left[p_\theta(\boldsymbol{r}_{\boldsymbol{x}} \mid \boldsymbol{x})\right]$
10:     Train $\pi$ by running Pi-Opt on $(\boldsymbol{x}, \boldsymbol{a}, \overline{\boldsymbol{r}_{\boldsymbol{x}}})$
11: end while
```

Online algorithm with policy optimization. The model fitting objective of Equation (4.4) can be used in various settings, e.g., offline over a collected trajectory dataset. Without assuming existing data, we explore an online setting, where the training process iteratively performs (1) exploration, (2) model-fitting, and (3) policy optimization, as shown in Algorithm 1. The policy $\pi \colon \mathcal{X} \to \Delta(\mathcal{A})$ soley operates on the Denoised MDP of x, which has all information sufficient for control. For policy optimization, the learned posterior encoder $q_\psi(\boldsymbol{x} \mid \boldsymbol{s}, \boldsymbol{a})$ is used to extract $\boldsymbol{x}$ information from the raw trajectory $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r})$, obtaining transition sequences in $\mathcal{X}$ space. Paired with the $p_\theta(\boldsymbol{r}_{\boldsymbol{x}} \mid \boldsymbol{x})$ rewards, we obtain $(\boldsymbol{x}, \boldsymbol{a}, \boldsymbol{r}_{\boldsymbol{x}})$ as trajectories collected from the Denoised MDP on x. Any general-purpose MDP policy optimization algorithm may be employed on these data, such as Stochastic Actor-Critic (SAC) [61]. We can also utilize the learned *differentiable* Denoised MDP, e.g., optimizing policy by backpropagating through additional roll-outs from the model, as is done in Dreamer.

While presented in the fully observable setting, Denoised MDP readily handles partial observability without extra changes. In the appendix, we discuss this point in details, and provide a guideline for choosing hyperparameters α, β.

4.4 Related Work

Model-Based Learning for Control jointly learns a world model and a policy. Such methods often enjoy good sample efficiency on RL tasks with rich observations. Some formulations rely on strong assumptions, e.g., deterministic transition in DeepMDP [49] and bilinear transition in FLAMBE [1]. Most general-setting methods use a reconstruction-based objective [64, 87, 60, 99]. Among them, Dreamer [63] trains world models with a variational formulation and optimizes policies by backpropagating through latent-space rollouts. It has proven effective across a variety of environments with image observations. However, such reconstruction-based approaches can struggle with the presence of noise distractors. TIA [45] partially addresses this limitation (see Section 4.2.3) but can not handle general distractors, unlike our method.

Representation Learning and Reinforcement Learning. Our work automates selecting useful signals from noisy MDPs by learning denoised world models, and can be viewed as an approach for learning general representations [37, 115, 70, 79]. In model-free RL, various methods learn state embeddings that are related to value functions [136, 9], transition dynamics [109, 100], recent action [125], bisimulation structure [44, 22, 176], data augmentations [97] etc. Recently, Eysenbach et al. [42] proposes a regularizer similar to ours but for the different purpose of robust compressed policies. The theoretical work by Efroni et al. [39] is closest to our setting but concerns a more restricted set of distractors (ones both uncontrollable and reward-irrelevant). Unlike Denoised MDP, their proposed algorithm is largely impractical and does not produce a generative model of observations (i.e., no decoder).

System Identification. Our work is related to system identification, where an algorithm infers from real world an abstract state among a predefined limited state space, e.g., pose estimation [131, 175] and material estimation [65]. Such results are useful for robotic manipulation [110], image generation [57], etc. Our setting is not limited to a predefined abstract state space, but instead focuses on automatic discovery of such valuable states.

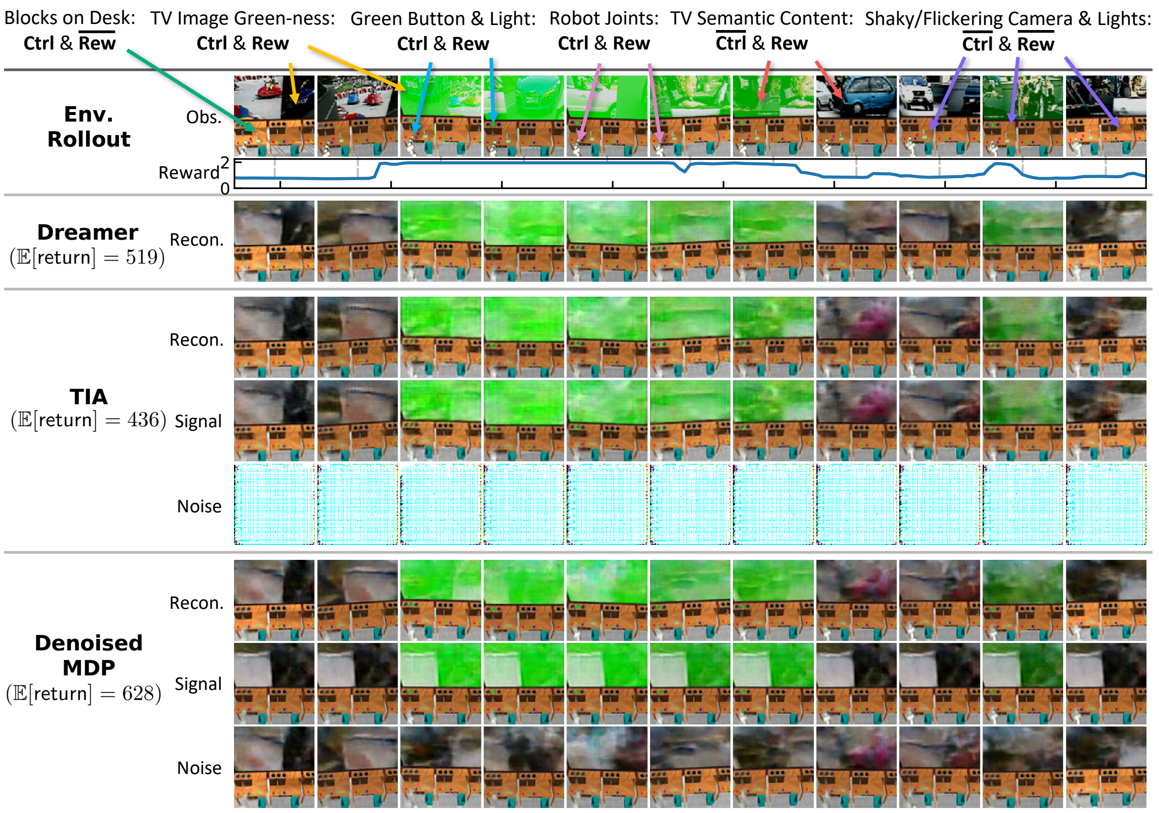

Blocks on Desk: Ctrl & $\overline{\text{Rew}}$
TV Image Green-ness: Ctrl & Rew
Green Button & Light: Ctrl & Rew
Robot Joints: Ctrl & Rew
TV Semantic Content: $\overline{\text{Ctrl}}$ & Rew
Shaky/Flickering Camera & Lights: $\overline{\text{Ctrl}}$ & $\overline{\text{Rew}}$
Env. Rollout
Obs.
Reward
2
0
Dreamer ($\mathbb{E}$[return] = 519)
Recon.
TIA ($\mathbb{E}$[return] = 436)
Recon.
Signal
Noise
Denoised MDP ($\mathbb{E}$[return] = 628)
Recon.
Signal
Noise

Figure 4-4: Visualization of learned models for RoboDesk by using decoders to reconstruct from encoded latents. For TIA and Denoised MDP, we visualize how they separate information as signal versus noise. In each row, *what changes over frames is the information modeled by the corresponding latent component.* E.g., in the bottom row, only the TV content, camera pose and lighting condition change, so Denoised MDP considers these factors as noises, while modelling the TV hue as signal. See our website for clearer video visualizations.

4.5 Experiments

In this section, we contrast our method with existing approaches on environments with image observations and many distinct types of noise distractors. Our experiments are designed to include a variety of noise distractors and to confirm our analysis on various methods in Section 4.2.3.

Environments. We choose DeepMind Control (DMC) Suite [159] (Section 4.5.2) and RoboDesk [86] (Section 4.5.1) with image observations, where we explore adding various noise distractors. Information types in all evaluated environments are categorized in Table C.1 of the appendix. Tasks include control (policy optimization) and a non-control task of regressing robot joint position from RoboDesk image observations.

Methods. We compare not only model-based RL methods, but also model-free algorithms and general representation learning approaches, when the task is suited:

- **Model Learning**: Denoised MDP (our method), Dreamer [63], and TIA [45];
- **Model-Free**: DBC [176], CURL [97], PI-SAC [**?**], and SAC on true state-space [61] (instead of using image observations, this is roughly an "upper bound");
- **General Image Representation Learning for Non-Control Tasks**: Contrastive learning with the Alignment+Uniformity loss [168] (a form of contrastive loss theoretically and empirically comparable to the popular InfoNCE loss [119]).

Model-learning methods can be used in combination with any policy optimization algorithm. For a complete comparison for general control, we compare the models trained with these two policy learning choices: (1) backpropagating via the learned dynamics and (2) SAC on the learned latent space (which roughly recovers SLAC [99] when used with an unfactorized model such as Dreamer).

Most compared methods do not apply data augmentations, which is known to strongly boost performance [174, 98]. Therefore, for a fair comparison, we run PI-SAC *without augmentation* to highlight its main contribution—representation of only predictive information.

All results are aggregated from 5 runs, showing mean and standard deviations. The appendix contains more details, hyperparameter studies, and additional results. Our website presents videos showing clearer video visualizations.

For Denoised MDP, we use the Figure 4-2b variant. Empirically, the Figure 4-2c variant leads to longer training time and sometimes inferior performance (perhaps due to having to optimize extra components and fit a more complex model). The appendix provides a comparison between them.

4.5.1 `RoboDesk` with Various Noise Distractors

We augment `RoboDesk` environment with many noise distractors that models realistic noises (e.g., flickering lights and shaky camera). Most importantly, we place a large

TV in the scene, which plays natural RGB videos. A green button on the desk controls the TV's hue (and a light on the desk). The agent is tasked with using this button to shift the TV to a green hue. Its reward is directly affected by how green the TV image is. The first row of Figure 4-4 shows a trajectory with various distractors annotated. All four types of information exist (see Table C.1), with the controllable and reward-relevant information being the robot arm, the green button, the light on the desk, and the TV screen green-ness.

Only Denoised MDP learns a clean denoised model. Using learned decoders, Figure 4-4 visualizes how the models captures various information. As expected, Dreamer model captures all information. TIA also fails to separate any noise distractors out (the Noise row fails to capture anything), likely due to its limited ability to model different noises. In contrast, Denoised MDP cleanly extracts all controllable and reward-relevant information as signals—the Signal row only *tracks changes* in robot arms, green button and light, and the TV screen green-ness. All other information is modeled as noises (see the Noise row). We recommend viewing video visualizations on our website.

Denoised models improve policy learning. Figure 4-4 also shows the total episode return achieved by policies learned with each of the three models, where the cleanest model from Denoised MDP achieves the best performance. Aggregating over 5 runs, the complete comparison in Figure 4-5 shows that Denoised MDP (with backpropagating via dynamics) generally outperforms all baselines, suggesting that its clean models are helpful for control.

Denoised models benefit non-control tasks. We evaluate the learned representations on a *supervised non-control* task—regressing the robot arm joint position from observed images. Using various pretrained encoders, we finetune on a labeled training set, and measure mean squared error (MSE) on a heldout test set. In addition to RL methods, we compare encoders learned via general contrastive learning on the same amount of data. In Figure 4-6, Denoised MDP representations lead to best converged

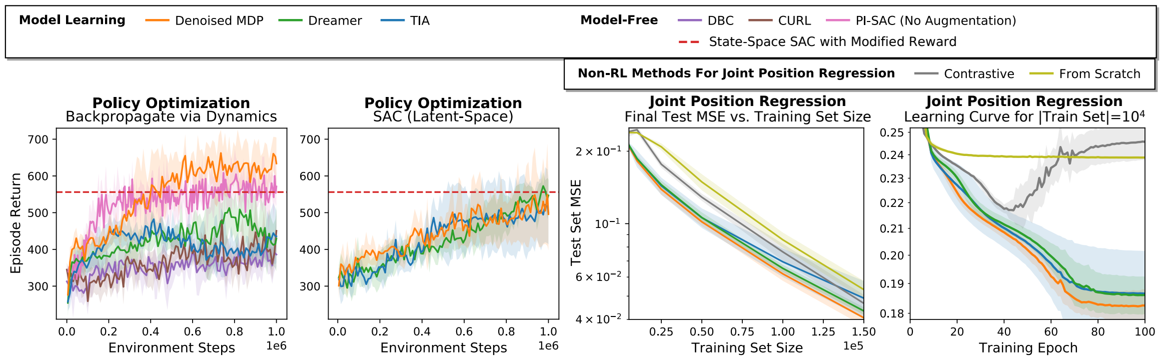


Figure 4-5: Policy optimization on `RoboDesk`. We give state-space SAC a less noisy reward so it can learn (see appendix).

Figure 4-6: Performance of finetuning various encoders to infer joint position from `RoboDesk` image observation.

	Policy Learning: Backprop via Dynamics			**Policy Learning:** SAC (Latent-Space)			DBC	PI-SAC (No Aug.)	CURL (Use Aug.)	State-Space SAC (Upper Bound)
	Denoised MDP	TIA	Dreamer	Denoised MDP	TIA	Dreamer				
Noiseless	801.4 ± 96.6	769.7 ± 97.1	**848.6 ± 137.1**	**587.1 ± 98.7**	480.2 ± 125.5	575.4 ± 146.2	297.4 ± 72.5	246.4 ± 56.6	417.3 ± 183.2	910.3 ± 28.2
Video Background	**597.7 ± 117.8**	407.1 ± 225.4	227.8 ± 102.7	309.8 ± 153.0	**318.1 ± 123.7**	188.7 ± 78.2	188.0 ± 67.4	131.7 ± 20.1	478.0 ± 113.5	910.3 ± 28.2
Video Background + Noisy Sensor	**563.1 ± 143.0**	261.2 ± 200.4	212.4 ± 89.7	**288.2 ± 123.4**	197.3 ± 124.2	218.2 ± 58.1	79.9 ± 36.0	152.5 ± 12.6	354.3 ± 119.9	919.8 ± 100.7
Video Background + Camera Jittering	**254.1 ± 114.2**	151.7 ± 160.5	98.6 ± 27.7	**186.8 ± 47.7**	126.5 ± 125.6	105.2 ± 33.8	68.0 ± 38.4	91.6 ± 7.6	**390.4 ± 64.9**	910.3 ± 28.2

Table 4.1: DMC policy optimization results. For each variant, we aggregate performance across three tasks (Cheetah Run, Walker Walk, Reacher Easy) by averaging. Denoised MDP performs well across all four variants with distinct noise types. **Bold numbers** show the best model-learning result for specific policy learning choices, or the best overall result. On **Camera Jittering**, Denoised MDP greatly outperforms all other methods except for CURL, which potentially benefits from its specific data augmentation choice (random crop) on this task, and can be seen as using extra information (i.e., knowing the noise distractor form). In fact, Denoised MDP is the only method that consistently performs well across all tasks and noise variants, which can be seen from the full results in the appendix.

solutions across a wide range of training set sizes, achieve faster training, and avoid overfitting when the training set is small. DBC, CURL and PI-SAC encoders, which take in stacked frames, are not directly comparable and thus absent from Figure 4-6. In the appendix, we compare them with running Denoised MDP encoder on each frame and concatenating the output features, where Denoised MDP handily outperforms both DBC and CURL by a large margin.

4.5.2 DeepMind Control Suite (DMC)

To evaluate a diverse set of noise distractors, we consider four variants for each DMC task (see Figure 4-7 top row):

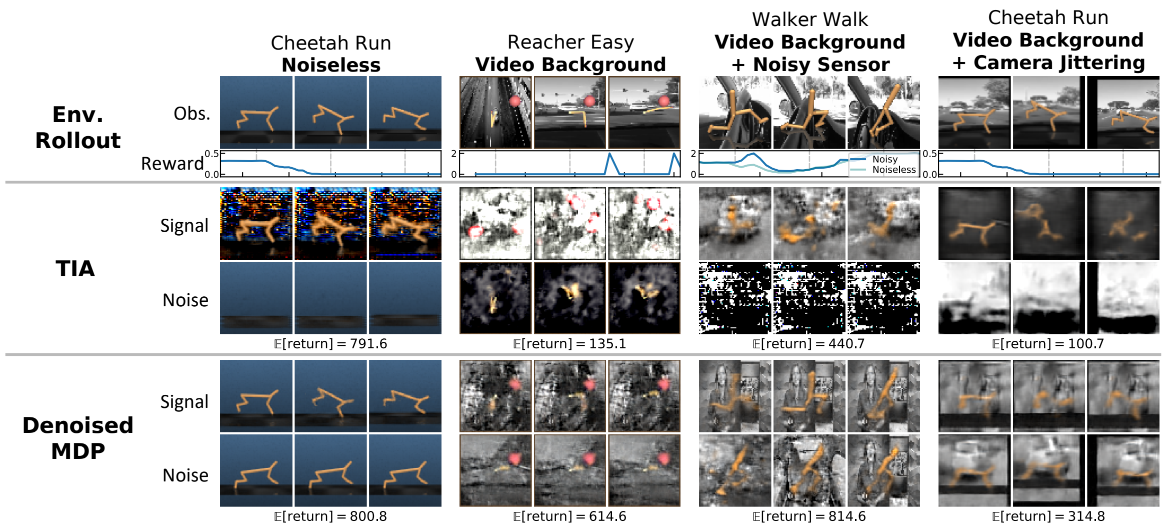


Figure 4-7: Visualization of the different DMC variants and factorizations learned by TIA and Denoised MDP. E.g., bottom Noise row often shows a static agent but varying background, indicating that only the background is modeled as noises in Denoised MDP. Visualizations of full reconstructions are in appendix. See our website for clearer video visualizations.

- **Noiseless**: Original environment without distractors.
- **Video Background**: Replacing noiseless background with natural videos [176] ($\overline{\textbf{Ctrl}} + \overline{\textbf{Rew}}$).
- **Video Background + Sensor Noise**: Imperfect sensors sensitive to intensity of a background patch ($\overline{\textbf{Ctrl}} + \textbf{Rew}$).
- **Video Background + Camera Jittering**: Shifting the observation by a smooth random walk ($\overline{\textbf{Ctrl}} + \overline{\textbf{Rew}}$).

Denoised MDP consistently removes noise distractors. In Figure 4-7, TIA struggles to learn clean separations in many settings. Consistent with analysis in Section 4.2.3, it cannot handle **Sensor Noise** or **Camera Jittering**, as the former is reward-relevant noise that it cannot model, and the latter (although reward-irrelevant) cannot be represented by masking. Furthermore, it fails on Reacher Easy with **Video Background**, where the reward is given by the distance between the agent and a randomly-located ball. TIA encourages its noise latent to be independent of reward, but does not prevent it from capturing the controllable agent. These failures lead to either TIA trying to model everything as useful signals, or a badly-fit model (e.g., wrong agent pose in the last column). In contrast, Denoised MDP separates out noise in all cases, obtaining a clean and accurate MDP (its Signal rows only have the agent moving).

Denoised models consistently improve policy learning. We evaluate the learned policies in Table 4.1, where results are aggregated by the noise distractor variant. Other methods, while sometimes handling certain noise types well, struggle to deal with all four distinct variants. TIA, as expected, greatly underperforms Denoised MDP under **Noisy Sensor** or **Camera Jittering**. CURL, whose augmentation choice potentially helps handling **Camera Jittering**, underperforms in other three variants. In contrast, Denoised MDP policies *consistently* perform well for all noisy variants and also the noiseless setting, regardless of the policy optimizer.

Model-based approaches have a significant lead over the model-free ones, as seen from the DBC results in Table 4.1 and the well-known fact that direct model-free learning on raw image observations usually fails [98, 93, 174]. These results show that learning in a world model is useful, and that learning in a denoised world model is even better.

4.6 Implications

In this work we explore learning denoised and compressed world models in the presence of environment noises.

As a step towards better understanding of such noises, we categorize of information in the wild into four types (Section 4.2). This provides a framework to contrast and understand various methods, highlighting where they may be successful and where they will suffer (Section 4.2.3). Insights gained this way empirically agrees with findings from extensive experiments (Section 4.5). It can potentially assist better algorithm design and analysis of new MDP representation methods, as we have done in designing Denoised MDP (Section 4.3). We believe that this categorization will be a useful framework for investigation on learning under noises, revealing not just the (conceptual) success scenarios, but also the failure scenarios at the same time. Additionally, the framework can be readily extended with more sophisticated factorizations (Section 4.2.4), which can lead to corresponding Denoised MDP variants and/or new algorithms.

Based on the framework, our proposed Denoised MDP novelly can remove *all* noise distractors that are *uncontrollable or reward-irrelevant*, in distinction to prior works. Empirically, it effectively identifies and removes a diverse set of noise types, obtaining clean denoised world models (Section 4.5). It may serve as an important step towards efficient learning of general tasks in the noisy real world. Our experiments also highlight benefits of cleanly denoised world models on both standard control tasks as well as non-control tasks. The success in both cases highlights the general usefulness of such models. Given the generality of MDPs, this opens up the possibility of casting non-RL tasks as MDPs and automatically learn representations from denoised world models, as an alternative to manual feature engineering.

Appendix A

Proofs, Details, and Additional Discussions for Chapter 2

A.1 Proofs and Additional Theoretical Analysis

In this section, we present proofs for propositions and theorems in main paper Sections 2.4.1 and 2.4.2.

The propositions in Section 2.4.1 illustrate the deep relations between the Gaussian kernel $G_t \colon \mathcal{S}^d \times \mathcal{S}^d \to \mathbb{R}$ and the uniform distribution on the unit hypersphere $\mathcal{S}^d$. As we will show below in Appendix A.1.1, these properties directly follow well-known results on strictly positive definite kernels.

In Appendix A.1.2, we present a proof for Theorem 2.4.7. Theorem 2.4.7 describes the asymptotic behavior of $\mathcal{L}_{\mathsf{contrastive}}$ as the number of negative samples M approaches infinity. The theorem is strongly related to empirical contrastive learning, given an error term (deviation from the limit) decaying in $\mathcal{O}(M^{-1/2})$ and that empirical practices often use a large number of negatives (e.g., $M = 65536$ in He et al. [70]) based on the observation that using more negatives consistently leads to better representation quality [171, 153, 70]. Our proof further reveals connections between $\mathcal{L}_{\mathsf{contrastive}}$ and $\mathcal{L}_{\mathsf{uniform}}$ which is defined via the Gaussian kernel.

Finally, also in Appendix A.1.2, we present a weaker result on the setting where only a single negative is used in $\mathcal{L}_{\mathsf{contrastive}}$ (i.e., $M = 1$).

A.1.1 Proofs for Section 2.4.1 and Properties of $\mathcal{L}_{\mathsf{uniform}}$

To prove Proposition 2.4.2 and 2.4.4, we utilize the *strict positive definiteness* [13, 142] of the Gaussian kernel G_t:

$$G_t(u, v) \triangleq e^{-t\|u-v\|_2^2} = e^{2t\cdot u^\mathsf{T} v - 2t}, \quad t > 0.$$

From there, we apply a known result about such kernels, from which the two propositions directly follow.

Definition A.1.1 (Strict positive definiteness [13, 142])**.** A symmetric and lower semi-continuous kernel K on $A \times A$ (where A is infinite and compact) is called strictly positive definite if for every finite signed Borel measure μ supported on A whose energy

$$I_K[\mu] \triangleq \int_{\mathcal{S}^d} \int_{\mathcal{S}^d} K(u, v) \, \mathrm{d}\mu(v) \, \mathrm{d}\mu(u)$$

is well defined, we have $I_K[\mu] \geq 0$, where equality holds only if $\mu \equiv 0$ on the σ-algebra of Borel subsets of A.

Definition A.1.2. Let $\mathcal{M}(\mathcal{S}^d)$ be the set of Borel probability measures on $\mathcal{S}^d$.

We are now in the place to apply the following two well-known results, which we present by restating Proposition 4.4.1, Theorem 6.2.1 and Corollary 6.2.2 of Borodachov et al. [17] in weaker forms. We refer readers to Borodachov et al. [17] for their proofs.

Lemma A.1.3 (Strict positive definiteness of G_t). For $t > 0$, the Gaussian kernel $G_t(u, v) \triangleq e^{-t\|u-v\|_2^2} = e^{2t\cdot u^\mathsf{T} v - 2t}$ is strictly positive definite on $\mathcal{S}^d \times \mathcal{S}^d$.

Lemma A.1.4 (Strictly positive definite kernels on $\mathcal{S}^d$). Consider kernel $K_f \colon \mathcal{S}^d \times \mathcal{S}^d \to (-\infty, +\infty]$ of the form,

$$K_f(u, v) \triangleq f(\|u - v\|_2^2). \tag{A.1}$$

If K_f is strictly positive definite on $\mathcal{S}^d \times \mathcal{S}^d$ and $I_{K_f}[\sigma_d]$ is finite, then σ_d is the unique measure (on Borel subsets of $\mathcal{S}^d$) in the solution of $\min_{\mu \in \mathcal{M}(\mathcal{S}^d)} I_{K_f}[\mu]$, and the normalized counting measures associated with any K_f-energy minimizing sequence of N-point configurations on $\mathcal{S}^d$ converges weak* to σ_d.

In particular, this conclusion holds whenever f has the property that $-f'(t)$ is strictly completely monotone on $(0, 4]$ and $I_{K_f}[\sigma_d]$ is finite.

We now recall Propositions 2.4.2 and 2.4.4.

Proposition 2.4.2. σ_d is the unique solution (on Borel subsets of $\mathcal{S}^d$) of

$$\min_{\mu \in \mathcal{M}(\mathcal{S}^d)} I_{G_t}[\mu] = \min_{\mu \in \mathcal{M}(\mathcal{S}^d)} \int_{\mathcal{S}^d} \int_{\mathcal{S}^d} G_t(u, v) \,\mathrm{d}\mu(v) \,\mathrm{d}\mu(u). \tag{A.2}$$

Proof of Proposition 2.4.2. This is a direct consequence of Lemmas A.1.3 and A.1.4. □

Proposition 2.4.4. For each $N > 0$, the N point minimizer of the average pairwise potential is

$$\mathbf{u}_N^* = \operatorname*{arg\,min}_{u_1, u_2, \dots, u_N \in \mathcal{S}^d} \sum_{1 \le i < j \le N} G_t(u_i, u_j).$$

The normalized counting measures associated with the $\{\mathbf{u}_N^*\}_{N=1}^{\infty}$ sequence converge weak* to σ_d.

Proof of Proposition 2.4.4. This is a direct consequence of Lemmas A.1.3 and A.1.4. □

More Properties of $\mathcal{L}_{\mathsf{uniform}}$

Range of $\mathcal{L}_{\mathsf{uniform}}$. It's not obvious what the optimal value of $\mathcal{L}_{\mathsf{uniform}}$ is. In the following proposition, we characterize the exact range of the expected Gaussian potential and how it evolves as dimensionality increases. The situation for $\mathcal{L}_{\mathsf{uniform}}$ directly follows as a corollary.

Proposition A.1.5 (Range of the expected pairwise Gaussian potential G_t). For $t > 0$, the expected pairwise Gaussian potential w.r.t. Borel probability measure $\mu \in \mathcal{M}(\mathcal{S}^d)$

$$I_{G_t}[\mu] = \int_{\mathcal{S}^d} \int_{\mathcal{S}^d} G_t(u, v) \,\mathrm{d}\mu(v) \,\mathrm{d}\mu(u)$$

has range $[e^{-2t}\,{}_0F_1(;\frac{d+1}{2};t^2), 1]$, where ${}_0F_1$ is the confluent hypergeometric limit function defined as

$$ {}_0F_1(;\alpha;z) \triangleq \sum_{n=0}^{\infty} \frac{z^n}{(\alpha)_n n!}, \tag{A.3}$$

where we have used the Pochhammer symbol

$$(a)_n = \begin{cases} 1 & \text{if } n = 0 \\ a(a+1)(n+2)\dots(a+n-1) & \text{if } n \geq 1. \end{cases}$$

We have

- The minimum $e^{-2t}\,{}_0F_1(;\frac{d+1}{2};t^2)$ is achieved iff $\mu = \sigma_d$ (on Borel subsets of $\mathcal{S}^d$). Furthermore, this value strictly decreases as d increases, converging to e^{-2t} in the limit of $d \to \infty$.

- The maximum is achieved iff μ is a Dirac delta distribution, i.e., $\mu = \delta_u$ (on Borel subsets of $\mathcal{S}^d$), for some $u \in \mathcal{S}^d$.

Proof of Proposition A.1.5.

- **Minimum.**

  We know from Proposition 2.4.2 that σ_d *uniquely* achieves the minimum, given by the following integral ratio

$$\begin{aligned} I_{G_t}[\sigma_d] &= \frac{\int_0^\pi e^{-t(2\sin\frac{\theta}{2})^2} \sin^{d-1}\theta \,\mathrm{d}\theta}{\int_0^\pi \sin^{d-1}\theta \,\mathrm{d}\theta} \\ &= \frac{\int_0^\pi e^{-2t(1-\cos\theta)} \sin^{d-1}\theta \,\mathrm{d}\theta}{\int_0^\pi \sin^{d-1}\theta \,\mathrm{d}\theta} \\ &= e^{-2t}\frac{\int_0^\pi e^{2t\cos\theta} \sin^{d-1}\theta \,\mathrm{d}\theta}{\int_0^\pi \sin^{d-1}\theta \,\mathrm{d}\theta}. \end{aligned}$$

  The denominator, with some trigonometric identities, can be more straightforwardly evaluated as

$$\int_0^\pi \sin^{d-1}\theta \,\mathrm{d}\theta = \sqrt{\pi}\frac{\Gamma(\frac{d}{2})}{\Gamma(\frac{d+1}{2})}.$$

The numerator is

$$\begin{aligned}
\int_0^\pi e^{2t\cos\theta} \sin^{d-1}\theta \, \mathrm{d}\theta &= -\int_0^\pi e^{2t\cos\theta} \sin^{d-2}\theta \cos'\theta \, \mathrm{d}\theta \\
&= \int_{-1}^{1} e^{2ts}(1-s^2)^{d/2-1} \, \mathrm{d}s \\
&= \frac{\Gamma(\frac{d-1}{2}+\frac{1}{2})\sqrt{\pi}}{\Gamma(\frac{d-1}{2}+1)} \, {}_0F_1(;\frac{d-1}{2}+1; -\frac{1}{4}(-2it)^2) \\
&= \frac{\Gamma(\frac{d}{2})\sqrt{\pi}}{\Gamma(\frac{d+1}{2})} \, {}_0F_1(;\frac{d+1}{2}; t^2),
\end{aligned}$$

where we have used the following identity based on the Poisson formula for Bessel functions and the relationship between ${}_0F_1$ and Bessel functions:

$$\int_{-1}^{1} e^{izs}(1-s^2)^{\nu-\frac{1}{2}} \, \mathrm{d}s = \frac{\Gamma(\nu+\frac{1}{2})\sqrt{\pi}}{(\frac{z}{2})^\nu} J_\nu(z) = \frac{\Gamma(\nu+\frac{1}{2})\sqrt{\pi}}{\Gamma(\nu+1)} \, {}_0F_1(;\nu+1; -\frac{1}{4}z^2).$$

Putting both together, we have

$$\begin{aligned}
I_{G_t}[\sigma_d] &= e^{-2t} \frac{\int_0^\pi e^{2t\cos\theta} \sin^{d-1}\theta \, \mathrm{d}\theta}{\int_0^\pi \sin^{d-1}\theta \, \mathrm{d}\theta} \\
&= e^{-2t} \frac{\frac{\Gamma(\frac{d}{2})\sqrt{\pi}}{\Gamma(\frac{d+1}{2})} \, {}_0F_1(;\frac{d+1}{2}; t^2)}{\sqrt{\pi}\frac{\Gamma(\frac{d}{2})}{\Gamma(\frac{d+1}{2})}} \\
&= e^{-2t} \, {}_0F_1(;\frac{d+1}{2}; t^2) \\
&= e^{-2t} \sum_{n=0}^{\infty} \frac{t^{2n}}{(\frac{d+1}{2})_n n!},
\end{aligned}$$

where we have used the definition of ${}_0F_1$ in Equation (A.3) to expand the formula.

Notice that each summand strictly decreases as $d \to \infty$. So must the total sum.

For the asymptotic behavior at $d \to \infty$, it only remains to show that

$$\lim_{d\to\infty} \sum_{n=0}^{\infty} \frac{t^{2n}}{(\frac{d+1}{2})_n n!} = 1. \tag{A.4}$$

For the purpose of applying the Dominated Convergence Theorem (DCT) (on the counting measure). We consider the following summable series

$$\sum_{n=0}^{\infty} \frac{t^{2n}}{n!} = e^{t^2},$$

with each term bounding the corresponding one in Equation (A.4):

$$\frac{t^{2n}}{n!} \geq \frac{t^{2n}}{(\frac{d+1}{2})_n n!}, \qquad \forall n \geq 0, d > 0.$$

Thus,

$$\lim_{d\to\infty} \sum_{n=0}^{\infty} \frac{t^{2n}}{(\frac{d+1}{2})_n n!} = \sum_{n=0}^{\infty} \lim_{d\to\infty} \frac{t^{2n}}{(\frac{d+1}{2})_n n!} = 1 + 0 + 0 + \cdots = 1.$$

Hence, the asymptotic lower range is e^{-2t}.

- **Maximum.**

Obviously, Dirac delta distributions δ_u, $u \in \mathcal{S}^d$ would achieve a maximum of 1. We will now show that all Borel probability measures μ s.t. $I_{G_t}[\mu] = 1$ are delta distributions.

Suppose that such a μ is not a Dirac delta distribution. Then, we can take distinct $x, y \in \operatorname{supp}(\mu) \subseteq \mathcal{S}^d$, and open neighborhoods around x and v, $N_x, N_y \in \mathcal{S}^d$ such that they are small enough and disjoint:

$$N_x \triangleq \{u \in \mathcal{S}^d \colon \|u - x\|_2 < \frac{1}{3}\|x - y\|_2\}$$
$$N_y \triangleq \{u \in \mathcal{S}^d \colon \|u - y\|_2 < \frac{1}{3}\|x - y\|_2\}.$$

Then,

$$\begin{aligned}
I_{G_t}[\mu] &= \int_{\mathcal{S}^d} \int_{\mathcal{S}^d} G_t(u, v) \, \mathrm{d}\mu(v) \, \mathrm{d}\mu(u) \\
&= \int_{\mathcal{S}^d} \int_{\mathcal{S}^d} e^{-t\|u-v\|_2^2} \, \mathrm{d}\mu(v) \, \mathrm{d}\mu(u) \\
&\leq (1 - 2\mu(N_x)\mu(N_y))e^{-t \cdot 0} + 2 \int_{N_x} \int_{N_y} e^{-t\|u-v\|_2^2} \, \mathrm{d}\mu(v) \, \mathrm{d}\mu(u) \\
&< 1 - 2\mu(N_x)\mu(N_y) + 2\mu(N_x)\mu(N_y) e^{-t(\|x-y\|_2/3)^2} \\
&= 1 - 2\mu(N_x)\mu(N_y)(1 - e^{-\frac{t}{9}\|x-y\|_2^2}) \\
&< 1.
\end{aligned}$$

Hence, only Dirac delta distributions attain the maximum.

□

Corollary A.1.6 (Range of $\mathcal{L}_{\mathsf{uniform}}$). For encoder $f \colon \mathbb{R}^n \to \mathcal{S}^{m-1}$, $\mathcal{L}_{\mathsf{uniform}}(f; t) \in [-2t + \log {}_0F_1(; \frac{m}{2}; t^2), 0]$, where the lower bound $-2t + \log {}_0F_1(; \frac{m}{2}; t^2)$ is achieved only by perfectly uniform encoders f, and the upper bound 0 is achieved only by degenerate encoders that output a fixed feature vector almost surely.

Furthermore, the lower bound strictly decreases as the output dimension m increases, attaining the following asymptotic value

$$\lim_{m \to \infty} -2t + \log {}_0F_1(; \frac{m}{2}; t^2) = -2t. \tag{A.5}$$

Intuition for the optimal $\mathcal{L}_{\mathsf{uniform}}$ value in high dimensions. If we ignore the $\log {}_0F_1(; \frac{m}{2}; t^2)$ term, informally, the optimal value of $-2t$ roughly says that any pair of feature vectors on $\mathcal{S}^d$ has distance about $\sqrt{2}$, i.e., are nearly orthogonal to each other. Indeed, vectors of high dimensions are usually nearly orthogonal, which is also consistent with the asymptotic result in Equation (A.5).

Figures A-1 and A-2 visualize how ${}_0F_1$ and the optimal $\mathcal{L}_{\mathsf{uniform}}$ (given by perfectly uniform encoders) evolve.

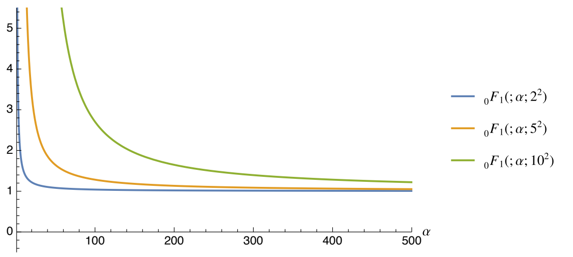


Figure A-1: Asymptotic behavior of ${}_0F_1(;\alpha;z)$. For $z > 0$, as α grows larger, the function converges to 1.

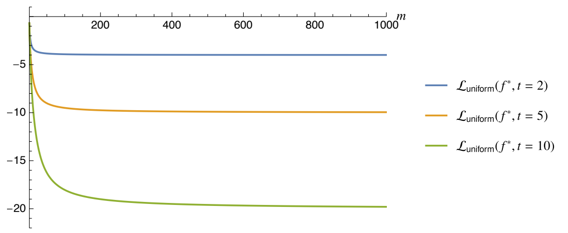


Figure A-2: Asymptotic behavior of optimal $\mathcal{L}_{\text{uniform}}(f, t)$, attained by a perfectly uniform encoder f^*. As the feature dimension m grows larger, the value converges to $-2t$.

Lower bound of $\mathcal{L}_{\text{uniform}}$ estimates. In practice, when $\mathcal{L}_{\text{uniform}}$ calculated using expectation over (a batch of) empirical samples $\{x_i\}_{i=1}^{B}$, $B > 1$, the range in Corollary A.1.6 is indeed valid, since it bounds over all distributions:

$$\hat{\mathcal{L}}_{\text{uniform}}^{(1)} \triangleq \log \frac{1}{B^2} \sum_{i=1}^{B} \sum_{j=1}^{B} e^{-t\|f(x_i)-f(x_j)\|^2} > -2t + \log {}_0F_1(; \frac{m}{2}; t^2). \tag{A.6}$$

However, often $\mathcal{L}_{\text{uniform}}$ is empirically estimated without considering distances between a vector and itself (e.g., in Figure 2-6 and in our experiment settings as described in Appendix A.2):

$$\hat{\mathcal{L}}_{\text{uniform}}^{(2)} \triangleq \log \frac{1}{B(B-1)} \sum_{i=1}^{B} \sum_{j\in\{1,\dots,B\}\setminus\{i\}} e^{-t\|f(x_i)-f(x_j)\|^2}. \tag{A.7}$$

While both quantities converge to the correct value in the limit, the lower bound is not always true for this one, because it is *not* the expected pairwise Gaussian kernel based on some distribution. Note the following relation:

$$\hat{\mathcal{L}}_{\text{uniform}}^{(2)} = \log \left(\frac{B \cdot \exp(\hat{\mathcal{L}}_{\text{uniform}}^{(1)}) - 1}{B-1} \right).$$

We can derive a valid lower bound using Equation (A.6): for ${}_0F_1(; \frac{m}{2}; t^2) > \frac{e^{2t}}{B}$,

$$\hat{\mathcal{L}}_{\text{uniform}}^{(2)} > \log \left(\frac{B \cdot \exp(-2t + \log {}_0F_1(; \frac{m}{2}; t^2)) - 1}{B-1} \right) = \log \left(\frac{Be^{-2t}\, {}_0F_1(; \frac{m}{2}; t^2) - 1}{B-1} \right).$$

Since this approaches fails for cases that ${}_0F_1(;\frac{m}{2};t^2) \leq \frac{e^{2t}}{B}$, we can combine it with the naive lower bound $-4t$, and have

$$\hat{\mathcal{L}}^{(2)}_{\mathsf{uniform}} > \begin{cases} \max(-4t, \log\left(\frac{Be^{-2t}\,{}_0F_1(;\frac{m}{2};t^2)-1}{B-1}\right)) & \text{if } {}_0F_1(;\frac{m}{2};t^2) > \frac{e^{2t}}{B} \\ -4t & \text{otherwise.} \end{cases}$$

Non-negative versions of $\mathcal{L}_{\mathsf{uniform}}$ for practical uses. By definition, $\mathcal{L}_{\mathsf{uniform}}$ always non-positive. As shown above, different $\mathcal{L}_{\mathsf{uniform}}$ empirical estimates may admit different lower bounds. However, in our experience, for reasonably large batch sizes, adding an offset of $2t$ often ensures a non-negative loss that is near zero at optimum. When output dimensionality m is low, it might be useful to add an additional offset of $-\log {}_0F_1(;\frac{m}{2};t^2)$, which can be computed with the help of the SciPy package function `scipy.special.hyp0f1(m/2, t**2)` [164].

A.1.2 Proofs and Additional Results for Section 2.4.2

The following lemma directly follows Theorem 3.3 and Remarks 3.4 (b)(i) of Serfozo [138]. We refer readers to Serfozo [138] for its proof.

Lemma A.1.7. Let A be a compact second countable Hausdorff space. Suppose

1. $\{\mu_n\}_{n=1}^{\infty}$ is a sequence of finite and positive Borel measures supported on A that converges weak* to some finite and positive Borel measure μ (which is same as vague convergence since A is compact);

2. $\{f_n\}_{n=1}^{\infty}$ is a sequence of Borel measurable functions that converges continuously to a Borel measurable f;

3. $\{f_n\}_n$ are uniformly bounded over A.

Then, we have the following convergence:

$$\lim_{n\to\infty} \int_{x\in A} f_n(x)\,\mathrm{d}\mu_n(x) = \int_{x\in A} f(x)\,\mathrm{d}\mu(x).$$

We now recall Theorem 2.4.7.

Theorem 2.4.7 (Asymptotics of $\mathcal{L}_{\mathsf{contrastive}}$). For fixed $\tau > 0$, as the number of negative samples $M \to \infty$, the (normalized) contrastive loss converges to

$$\begin{aligned}
&\lim_{M\to\infty} \mathcal{L}_{\mathsf{contrastive}}(f;\tau,M) - \log M \\
&\qquad = \lim_{M\to\infty} \mathop{\mathbb{E}}_{\substack{(x,y)\sim p_{\mathsf{pos}} \\ \{x_i^-\}_{i=1}^M \overset{\text{i.i.d.}}{\sim} p_{\mathsf{data}}}} \left[-\log \frac{e^{f(x)^\intercal f(y)/\tau}}{e^{f(x)^\intercal f(y)/\tau} + \sum_i e^{f(x_i^-)^\intercal f(y)/\tau}} \right] - \log M \\
&\qquad = -\frac{1}{\tau} \mathop{\mathbb{E}}_{(x,y)\sim p_{\mathsf{pos}}} \left[f(x)^\intercal f(y) \right] + \mathop{\mathbb{E}}_{x\sim p_{\mathsf{data}}} \left[\log \mathop{\mathbb{E}}_{x^-\sim p_{\mathsf{data}}} \left[e^{f(x^-)^\intercal f(x)/\tau} \right] \right]. \qquad (2.2)
\end{aligned}$$

We have the following results:

1. The first term is minimized iff f is perfectly aligned.

2. If perfectly uniform encoders exist, they form the exact minimizers of the second term.

3. For the convergence in Equation (2.2), the absolute deviation from the limit (i.e., the error term) decays in $\mathcal{O}(M^{-1/2})$.

Proof of Theorem 2.4.7. We first show the convergence stated in Equation (2.2) along with its speed (result 3), and then the relations between the two limiting terms and the alignment and uniformity properties (results 1 and 2).

- **Proof of the convergence in Equation (2.2) and the $\mathcal{O}(M^{-1/2})$ decay rate of its error term (result 3).**

  Note that for any $x, y \in \mathbb{R}^n$ and $\{x_i^-\}_{i=1}^M \overset{\text{i.i.d.}}{\sim} p_{\mathsf{data}}$, we have, almost surely,

$$\lim_{M\to\infty} \log \left(\frac{1}{M} e^{f(x)^\intercal f(y)/\tau} + \frac{1}{M} \sum_{i=1}^M e^{f(x_i^-)^\intercal f(x)/\tau} \right) = \log \mathop{\mathbb{E}}_{x^-\sim p_{\mathsf{data}}} \left[e^{f(x^-)^\intercal f(x)/\tau} \right], \qquad \text{(A.8)}$$

  by the strong law of large numbers (SLLN) and the Continuous Mapping Theorem.

Then, we can derive

$$
\begin{aligned}
&\lim_{M\to\infty} \mathcal{L}_{\mathsf{contrastive}}(f;\tau,M) - \log M \\
&\qquad = \mathop{\mathbb{E}}_{(x,y)\sim p_{\mathsf{pos}}}\left[-f(x)^\intercal f(y)/\tau\right] \\
&\qquad\qquad + \lim_{M\to\infty} \mathop{\mathbb{E}}_{\substack{(x,y)\sim p_{\mathsf{pos}} \\ \{x_i^-\}_{i=1}^M \overset{\text{i.i.d.}}{\sim} p_{\mathsf{data}}}} \left[\log\left(\frac{1}{M}e^{f(x)^\intercal f(y)/\tau} + \frac{1}{M}\sum_{i=1}^M e^{f(x_i^-)^\intercal f(x)/\tau}\right)\right] \\
&\qquad = \mathop{\mathbb{E}}_{(x,y)\sim p_{\mathsf{pos}}}\left[-f(x)^\intercal f(y)/\tau\right] \\
&\qquad\qquad + \mathbb{E}\left[\lim_{M\to\infty}\log\left(\frac{1}{M}e^{f(x)^\intercal f(y)/\tau} + \frac{1}{M}\sum_{i=1}^M e^{f(x_i^-)^\intercal f(x)/\tau}\right)\right] \\
&\qquad = -\frac{1}{\tau}\mathop{\mathbb{E}}_{(x,y)\sim p_{\mathsf{pos}}}\left[f(x)^\intercal f(y)\right] + \mathop{\mathbb{E}}_{x\sim p_{\mathsf{data}}}\left[\log \mathop{\mathbb{E}}_{x^-\sim p_{\mathsf{data}}}\left[e^{f(x^-)^\intercal f(x)/\tau}\right]\right],
\end{aligned}
$$

where we justify the switching of expectation and limit by the convergence stated in Equation (A.8), the boundedness of $e^{u^\intercal v/\tau}$ (where $u, v \in \mathcal{S}^d, \tau > 0$), and the Dominated Convergence Theorem (DCT).

For convergence speed, we have

$$
\begin{aligned}
&\left|\left(\lim_{M\to\infty} \mathcal{L}_{\mathsf{contrastive}}(f;\tau,M) - \log M\right) - \left(\mathcal{L}_{\mathsf{contrastive}}(f;\tau,M) - \log M\right)\right| \\
&\qquad = \left|\mathop{\mathbb{E}}_{\substack{(x,y)\sim p_{\mathsf{pos}} \\ \{x_i^-\}_{i=1}^M \overset{\text{i.i.d.}}{\sim} p_{\mathsf{data}}}}\left[\log \mathop{\mathbb{E}}_{x^-\sim p_{\mathsf{data}}}\left[e^{f(x^-)^\intercal f(x)/\tau}\right] - \log\left(\frac{1}{M}e^{f(x)^\intercal f(y)/\tau} + \frac{1}{M}\sum_{i=1}^M e^{f(x_i^-)^\intercal f(x)/\tau}\right)\right]\right| \\
&\qquad \le \mathop{\mathbb{E}}_{\substack{(x,y)\sim p_{\mathsf{pos}} \\ \{x_i^-\}_{i=1}^M \overset{\text{i.i.d.}}{\sim} p_{\mathsf{data}}}}\left[\left|\log \mathop{\mathbb{E}}_{x^-\sim p_{\mathsf{data}}}\left[e^{f(x^-)^\intercal f(x)/\tau}\right] - \log\left(\frac{1}{M}e^{f(x)^\intercal f(y)/\tau} + \frac{1}{M}\sum_{i=1}^M e^{f(x_i^-)^\intercal f(x)/\tau}\right)\right|\right] \\
&\qquad \le e^{1/\tau}\mathop{\mathbb{E}}_{\substack{(x,y)\sim p_{\mathsf{pos}} \\ \{x_i^-\}_{i=1}^M \overset{\text{i.i.d.}}{\sim} p_{\mathsf{data}}}}\left[\left|\mathop{\mathbb{E}}_{x^-\sim p_{\mathsf{data}}}\left[e^{f(x^-)^\intercal f(x)/\tau}\right] - \left(\frac{1}{M}e^{f(x)^\intercal f(y)/\tau} + \frac{1}{M}\sum_{i=1}^M e^{f(x_i^-)^\intercal f(x)/\tau}\right)\right|\right] \\
&\qquad \le \frac{1}{M}e^{2/\tau} + e^{1/\tau}\mathop{\mathbb{E}}_{x,\{x_i^-\}_{i=1}^M \overset{\text{i.i.d.}}{\sim} p_{\mathsf{data}}}\left[\left|\mathop{\mathbb{E}}_{x^-\sim p_{\mathsf{data}}}\left[e^{f(x^-)^\intercal f(x)/\tau}\right] - \frac{1}{M}\sum_{i=1}^M e^{f(x_i^-)^\intercal f(x)/\tau}\right|\right] \\
&\qquad = \frac{1}{M}e^{2/\tau} + \mathcal{O}(M^{-1/2}), \qquad\qquad \text{(A.9)}
\end{aligned}
$$

where the first inequality follows the Intermediate Value Theorem and the $e^{1/\tau}$ upper bound on the absolute derivative of log between the two points, and the last equality follows the Berry-Esseen Theorem given the bounded support of $e^{f(x_i^-)^\mathsf{T} f(x)/\tau}$ as following: for i.i.d. random variables Y_i with bounded support $\subset [-a, a]$, zero mean and $\sigma_Y^2 \leq a^2$ variance, we have

$$\begin{aligned}
\mathbb{E}\left[\left|\frac{1}{M}\sum_{i=1}^M Y_i\right|\right] &= \frac{\sigma_Y}{\sqrt{M}}\mathbb{E}\left[\left|\frac{1}{\sqrt{M}\sigma_Y}\sum_{i=1}^M Y_i\right|\right] \\
&= \frac{\sigma_Y}{\sqrt{M}}\int_0^{\frac{a\sqrt{M}}{\sigma_Y}} \mathbb{P}\left[\left|\frac{1}{\sqrt{M}\sigma_Y}\sum_{i=1}^M Y_i\right| > x\right] \mathrm{d}x \\
&\leq \frac{\sigma_Y}{\sqrt{M}}\int_0^{\frac{a\sqrt{M}}{\sigma_Y}} \mathbb{P}\left[|\mathcal{N}(0,1)| > x\right] + \frac{C_a}{\sqrt{M}} \mathrm{d}x \qquad \text{(Berry-Esseen)} \\
&\leq \frac{\sigma_Y}{\sqrt{M}}\left(\frac{aC_a}{\sigma_Y} + \int_0^\infty \mathbb{P}\left[|\mathcal{N}(0,1)| > x\right]\mathrm{d}x\right) \\
&= \frac{\sigma_Y}{\sqrt{M}}\left(\frac{aC_a}{\sigma_Y} + \mathbb{E}\left[|\mathcal{N}(0,1)|\right]\right) \\
&\leq \frac{C_a}{\sqrt{M}} + \frac{a}{\sqrt{M}}\mathbb{E}\left[|\mathcal{N}(0,1)|\right] \\
&= \mathcal{O}(M^{-1/2}),
\end{aligned}$$

where the constant C_a only depends on a (which controls both the second and the third moment).

- **Proof of result 1: The first term is minimized iff f is perfectly aligned.**

  Note that for $u, v \in \mathcal{S}^d$,

  $$\|u - v\|_2^2 = 2 - 2 \cdot u^T v.$$

  Then the result follows directly the definition of perfect alignment, and the existence of perfectly aligned encoders (e.g., an encoder that maps every input to the same output vector).

- **Proof of result 2: If perfectly uniform encoders exist, they form the exact minimizers of the second term.**

For simplicity, we define the following notation:

Definition A.1.8. $\forall \mu \in \mathcal{M}(\mathcal{S}^d)$, $u \in \mathcal{S}^d$, we define the continuous and Borel measurable function

$$U_\mu(u) \triangleq \int_{\mathcal{S}^d} e^{u^\intercal v/\tau} \, \mathrm{d}\mu(v). \tag{A.10}$$

with its range bounded in $[e^{-1/\tau}, e^{1/\tau}]$.

Then the second term can be equivalently written as

$$\mathop{\mathbb{E}}_{x \sim p_{\mathsf{data}}} \left[\log \mathop{\mathbb{E}}_{x^- \sim p_{\mathsf{data}}} \left[e^{f(x^-)^\intercal f(x)/\tau} \right] \right] = \mathop{\mathbb{E}}_{x \sim p_{\mathsf{data}}} \left[\log U_{p_{\mathsf{data}} \circ f^{-1}}(f(x)) \right],$$

where $p_{\mathsf{data}} \circ f^{-1} \in \mathcal{M}(\mathcal{S}^d)$ is the probability measure of features, i.e., the pushforward measure of p_{data} via f.

We now consider the following relaxed problem, where the minimization is taken over $\mathcal{M}(\mathcal{S}^d)$, all possible Borel probability measures on the hypersphere $\mathcal{S}^d$:

$$\min_{\mu \in \mathcal{M}(\mathcal{S}^d)} \int_{\mathcal{S}^d} \log U_\mu(u) \, \mathrm{d}\mu(u). \tag{A.11}$$

Our strategy is to show that the unique minimizer of Equation (A.11) is σ_d, from which the result 2 directly follows. The rest of the proof is structured in three parts.

1. **We show that minimizers of Equation (A.11) exist, i.e., the above infimum is attained for some $\mu \in \mathcal{M}(\mathcal{S}^d)$.**

   Let $\{\mu_m\}_{m=1}^\infty$ be a sequence in $\mathcal{M}(\mathcal{S}^d)$ such that the infimum of Equation (A.11) is reached in the limit:

   $$\lim_{m \to \infty} \int_{\mathcal{S}^d} \log U_{\mu_m}(u) \, \mathrm{d}\mu_m(u) = \inf_{\mu \in \mathcal{M}(\mathcal{S}^d)} \int_{\mathcal{S}^d} \log U_\mu(u) \, \mathrm{d}\mu(u).$$

   From the Helly's Selection Theorem, let μ^* denote some weak* cluster point of this sequence. Then μ_m converges weak* to μ^* along a subsequence

$m \in \mathcal{N} \in \mathbb{N}$. For simplicity and with a slight abuse of notation, we denote this convergent (sub)sequence of measures by $\{\mu_n\}_{n=1}^{\infty}$.

We want to show that μ^* attains the limit (and thus the infimum), i.e.,

$$\int_{\mathcal{S}^d} \log U_{\mu^*}(u) \, \mathrm{d}\mu^*(u) = \lim_{n\to\infty} \int_{\mathcal{S}^d} \log U_{\mu_n}(u) \, \mathrm{d}\mu_n(u). \tag{A.12}$$

In view of Lemma A.1.7, since $\mathcal{S}^d$ is a compact second countable Hausdorff space and $\{\log U_{\mu_n}\}_n$ is uniformly bounded over $\mathcal{S}^d$, it remains to prove that $\{\log U_{\mu_n}\}_n$ is continuously convergent to $\log U_{\mu^*}$.

Consider any convergent sequence of points $\{x_n\}_{n=1}^{\infty} \in \mathbb{R}^{d+1}$ s.t. $x_n \to x$ where $x \in \mathcal{S}^d$.

Let $\delta_n = x_n - x$. By simply expanding U_{μ_n} and μ_{μ^*}, we have

$$e^{-\|\delta_n\|/\tau} U_{\mu_n}(x) \leq U_{\mu_n}(x_n) \leq e^{\|\delta_n\|/\tau} U_{\mu_n}(x).$$

Since both the upper and the lower bound converge to $U_{\mu^*}(x)$ (by the weak * convergence of $\{\mu_n\}_n$ to μ^*), $U_{\mu_n}(x_n)$ must as well. We have proved the continuous convergence of $\{\log U_{\mu_n}\}_n$ to $\log U_{\mu^*}$.

Therefore, the limit in Equation (A.12) holds. The infimum is thus attained at μ^*:

$$\lim_{n\to\infty} \int_u \log U_{\mu_n}(u) \, \mathrm{d}\mu_n = \int_u \log U_{\mu^*}(u) \, \mathrm{d}\mu^*.$$

2. **We show that U_{μ^*} is constant μ^*-almost surely for any minimizer μ^* of Equation (A.11).**

   Let μ^* be any solution of Equation (A.11):

$$\mu^* \in \operatorname*{arg\,min}_{\mu \in \mathcal{M}(\mathcal{S}^d)} \int_u \log U_\mu(u) \, \mathrm{d}\mu.$$

   Consider the Borel sets where μ^* has positive measure: $\mathcal{T} \triangleq \{T \in \mathcal{B}(\mathcal{S}^d) \colon \mu^*(T) > 0\}$. For any $T \in \mathcal{T}$, let μ^*_T denote the conditional distribution of μ^* on T, i.e., $\forall A \in \mathcal{B}(\mathcal{S}^d)$,

$$\mu_T^*(A) = \frac{\mu^*(A \cap T)}{\mu^*(T)}.$$

Note that for any such $T \in \mathcal{T}$, the mixture $(1-\alpha)\mu^* + \alpha\mu_T^*$ is a valid probability distribution (i.e., in $\mathcal{M}(\mathcal{S}^d)$) for $\alpha \in (-\mu^*(T), 1)$, an open interval containing 0.

By the first variation, we must have

$$\begin{aligned}
0 &= \frac{\partial}{\partial\alpha} \int_{\mathcal{S}^d} \log U_{(1-\alpha)\mu^*+\alpha\mu_T^*}(u) \,\mathrm{d}((1-\alpha)\mu^* + \alpha\mu_T^*)(u)\bigg|_{\alpha=0} \\
&= \frac{\partial}{\partial\alpha}(1-\alpha) \int_{\mathcal{S}^d} \log U_{(1-\alpha)\mu^*+\alpha\mu_T^*}(u) \,\mathrm{d}\mu^*(u)\bigg|_{\alpha=0} + \frac{\partial}{\partial\alpha}\alpha \int_{\mathcal{S}^d} \log U_{(1-\alpha)\mu^*+\alpha\mu_T^*}(u) \,\mathrm{d}\mu_T^*(u)\bigg|_{\alpha=0} \\
&= -\int_{\mathcal{S}^d} \log U_{(1-\alpha)\mu^*+\alpha\mu_T^*}(u) \,\mathrm{d}\mu^*(u)\bigg|_{\alpha=0} + \frac{\partial}{\partial\alpha} \int_{\mathcal{S}^d} \log U_{(1-\alpha)\mu^*+\alpha\mu_T^*}(u) \,\mathrm{d}\mu^*(u)\bigg|_{\alpha=0} \\
&\qquad + \int_{\mathcal{S}^d} \log U_{(1-\alpha)\mu^*+\alpha\mu_T^*}(u) \,\mathrm{d}\mu_T^*(u)\bigg|_{\alpha=0} + 0 \cdot \frac{\partial}{\partial\alpha} \int_{\mathcal{S}^d} \log U_{(1-\alpha)\mu^*+\alpha\mu_T^*}(u) \,\mathrm{d}\mu_T^*(u)\bigg|_{\alpha=0} \\
&= -\int_{\mathcal{S}^d} \log U_{\mu^*}(u) \,\mathrm{d}\mu^*(u) + \int_{\mathcal{S}^d} \frac{U_{\mu_T^*}(u) - U_{\mu^*}(u)}{U_{\mu^*}(u)} \,\mathrm{d}\mu^*(u) \\
&\qquad + \int_{\mathcal{S}^d} \log U_{\mu^*}(u) \,\mathrm{d}\mu_T^*(u) + 0 \cdot \int_{\mathcal{S}^d} \frac{U_{\mu_T^*}(u) - U_{\mu^*}(u)}{U_{\mu^*}(u)} \,\mathrm{d}\mu_T^*(u) \\
&= \int_{\mathcal{S}^d} \frac{U_{\mu_T^*}(u)}{U_{\mu^*}(u)} \,\mathrm{d}\mu^*(u) + \int_{\mathcal{S}^d} \log U_{\mu^*}(u) \,\mathrm{d}(\mu_T^* - \mu^*)(u) - 1, \qquad \text{(A.13)}
\end{aligned}$$

where the Leibniz rule along with the boundedness of U_{μ^*} and $U_{\mu_{T_n}^*}$ together justify the exchanges of integration and differentiation.

Let $\{T_n\}_{n=1}^\infty$ be a sequence of sets in $\mathcal{T}$ such that

$$\lim_{n\to\infty} \int_{\mathcal{S}^d} U_{\mu^*}(u) \,\mathrm{d}\mu_{T_n}^*(u) = \sup_{T \in \mathcal{T}} \int_{\mathcal{S}^d} U_{\mu^*}(u) \,\mathrm{d}\mu_T^*(u) \triangleq U^*,$$

where the supremum must exist since U_{μ^*} is bounded above.

Because U_{μ^*} is a continuous and Borel measurable function, we have

$\{u\colon U_{\mu^*}(u) > U^*\} \in \mathcal{B}(\mathcal{S}^d)$ and thus

$$\begin{aligned}
&\mu^*(\{u\colon U_{\mu^*}(u) > U^*\}) = 0, \\
&\mu^*_{T_n}(\{u\colon U_{\mu^*}(u) > U^*\}) = 0, \qquad\qquad \forall n = 1, 2, \dots,
\end{aligned}$$

otherwise $\{u\colon U_{\mu^*}(u) > U^*\} \in \mathcal{T}$, contradicting the definition of U^* as the supremum.

Asymptotically, U_{μ^*} is constant $\mu^*_{T_n}$-almost surely:

$$\begin{aligned}
&\int_{\mathcal{S}^d} \left| U_{\mu^*}(u) - \int_{\mathcal{S}^d} U_{\mu^*}(u') \,\mathrm{d}\mu^*_{T_n}(u') \right| \mathrm{d}\mu^*_{T_n}(u) \\
&\qquad = 2 \int_{\mathcal{S}^d} \max\left(0,\ U_{\mu^*}(u) - \int_{\mathcal{S}^d} U_{\mu^*}(u') \,\mathrm{d}\mu^*_{T_n}(u')\right) \mathrm{d}\mu^*_{T_n}(u) \\
&\qquad \leq 2(U^* - \int_{\mathcal{S}^d} U_{\mu^*}(u) \,\mathrm{d}\mu^*_{T_n}(u)) \\
&\qquad \to 0, \qquad\qquad \text{as } n \to \infty,
\end{aligned}$$

where the inequality follows the boundedness of U_{μ^*} and that $\mu^*_{T_n}(\{u\colon U_{\mu^*}(u) > U^*\}) = 0$.

Therefore, given the continuity of log and the boundedness of U_{μ^*}, we have

$$\lim_{n\to\infty} \int_{\mathcal{S}^d} \log U_{\mu^*}(u) \,\mathrm{d}\mu^*_{T_n} = \log U^*.$$

Equation (A.13) gives that $\forall n = 1, 2, \dots,$

$$\begin{aligned}
1 &= \int_{\mathcal{S}^d} \frac{U_{\mu^*_{T_n}}(u)}{U_{\mu^*}(u)} \,\mathrm{d}\mu^* + \int_{\mathcal{S}^d} \log U_{\mu^*}(u) \,\mathrm{d}(\mu^*_{T_n} - \mu^*) \\
&\geq \frac{1}{U^*} \int_{\mathcal{S}^d} U_{\mu^*_{T_n}}(u) \,\mathrm{d}\mu^*(u) + \int_{\mathcal{S}^d} \log U_{\mu^*}(u) \,\mathrm{d}\mu^*_{T_n} - \int_{\mathcal{S}^d} \log U_{\mu^*}(u) \,\mathrm{d}\mu^* \\
&= \frac{1}{U^*} \int_{\mathcal{S}^d} U_{\mu^*}(u) \,\mathrm{d}\mu^*_{T_n}(u) + \int_{\mathcal{S}^d} \log U_{\mu^*}(u) \,\mathrm{d}\mu^*_{T_n} - \int_{\mathcal{S}^d} \log U_{\mu^*}(u) \,\mathrm{d}\mu^*,
\end{aligned}$$

where the inequality follows the boundedness of $\frac{U_{\mu^*_{T_n}}}{U_{\mu^*}}$ and that $\mu^*(\{u\colon U_{\mu^*}(u) > U^*\}) = 0$.

Taking the limit of $n \to \infty$ on both sides, we have

$$\begin{aligned}
1 = \lim_{n\to\infty} 1 \geq \frac{1}{U^*} \lim_{n\to\infty} \int_{\mathcal{S}^d} U_{\mu^*}(u)\,\mathrm{d}\mu^*_{T_n}(u) + \lim_{n\to\infty} \int_{\mathcal{S}^d} \log U_{\mu^*}(u)\,\mathrm{d}\mu^*_{T_n}(u) \\
- \int_{\mathcal{S}^d} \log U_{\mu^*}(u)\,\mathrm{d}\mu^*(u) \\
&= 1 + \log U^* - \int_{\mathcal{S}^d} \log U_{\mu^*}(u)\,\mathrm{d}\mu^*(u) \\
&\geq 1 + \log U^* - \log \int_{\mathcal{S}^d} U_{\mu^*}(u)\,\mathrm{d}\mu^*(u) \\
&\geq 1,
\end{aligned}$$

where the last inequality holds because the supremum taken over $\mathcal{T} \supset \{\mathcal{S}^d\}$.

Since $1 = 1$, all inequalities must be equalities. In particular,

$$\int_{\mathcal{S}^d} \log U_{\mu^*}(u)\,\mathrm{d}\mu^*(u) = \log \int_{\mathcal{S}^d} U_{\mu^*}(u)\,\mathrm{d}\mu^*(u).$$

That is, for any solution μ^* of Equation (A.11), U_{μ^*} must be constant μ^*-almost surely.

3. **We show that σ_d is the unique minimizer of the relaxed problem in Equation (A.11).**

Let $S \subset \mathcal{M}(\mathcal{S}^d)$ be the set of measures where the above property holds:

$$S \triangleq \left\{\mu \in \mathcal{M}(\mathcal{S}^d) \colon U_\mu \text{ is constant } \mu\text{-almost surely}\right\}.$$

The problem in Equation (A.11) is thus equivalent to minimizing over S:

$$\begin{aligned}
\arg\min_{\mu \in \mathcal{M}(\mathcal{S}^d)} \int_{\mathcal{S}^d} \log U_\mu(u) \,\mathrm{d}\mu(u) &= \arg\min_{\mu \in S} \int_{\mathcal{S}^d} \log U_\mu(u) \,\mathrm{d}\mu(u) \\
&= \arg\min_{\mu \in S} \log \int_{\mathcal{S}^d} U_\mu(u) \,\mathrm{d}\mu(u) \\
&= \arg\min_{\mu \in S} \log \int_{\mathcal{S}^d} \int_{\mathcal{S}^d} e^{u^\mathsf{T} v/\tau} \,\mathrm{d}\mu(v) \,\mathrm{d}\mu(u) \\
&= \arg\min_{\mu \in S} \left(\frac{1}{\tau} + \log \int_{\mathcal{S}^d} \int_{\mathcal{S}^d} e^{-\frac{1}{2\tau}\|u-v\|^2} \,\mathrm{d}\mu(v) \,\mathrm{d}\mu(u) \right) \\
&= \arg\min_{\mu \in S} \int_{\mathcal{S}^d} \int_{\mathcal{S}^d} G_{\frac{1}{2\tau}}(u, v) \,\mathrm{d}\mu(v) \,\mathrm{d}\mu(u).
\end{aligned}$$

By Proposition 2.4.2 and $\tau > 0$, we know that the uniform distribution σ_d is the unique solution to

$$\arg\min_{\mu \in \mathcal{M}(\mathcal{S}^d)} \int_{\mathcal{S}^d} \int_{\mathcal{S}^d} G_{\frac{1}{2\tau}}(u, v) \,\mathrm{d}\mu(v) \,\mathrm{d}\mu(u). \tag{A.14}$$

Since $\sigma_d \in S$, it must also be the unique solution to Equation (A.11).

Finally, if perfectly uniform encoders exist, σ_d is realizable, and they are the exact encoders that realize it. Hence, in such cases, they are the exact minimizers of

$$\min_f \mathop{\mathbb{E}}_{x \sim p_\mathsf{data}} \left[\log \mathop{\mathbb{E}}_{x^- \sim p_\mathsf{data}} \left[e^{f(x^-)^\mathsf{T} f(x)/\tau} \right] \right].$$

$\square$

Relation between Theorem 2.4.7, $\mathcal{L}_\mathsf{align}$ and $\mathcal{L}_\mathsf{uniform}$. The first term of Equation (2.2) is equivalent with $\mathcal{L}_\mathsf{align}$ when $\alpha = 2$, up to a constant and a scaling. In the above proof, we showed that the second term favors uniformity, via the feature distribution that minimizes the pairwise Gaussian kernel (see Equation (A.14)):

$$\arg\min_{\mu \in \mathcal{M}(\mathcal{S}^d)} \int_{\mathcal{S}^d} \int_{\mathcal{S}^d} G_{\frac{1}{2\tau}}(u, v) \,\mathrm{d}\mu(v) \,\mathrm{d}\mu(u), \tag{A.15}$$

which can be alternatively viewed as the relaxed problem of optimizing for the uniformity loss $\mathcal{L}_{\mathsf{uniform}}$:

$$\arg\min_f \mathcal{L}_{\mathsf{uniform}}(f; \frac{1}{2\tau}) = \arg\min_f \mathbb{E}_{x,y \overset{\text{i.i.d.}}{\sim} p_{\mathsf{data}}} \left[G_{\frac{1}{2\tau}}(f(x), f(y)) \right]. \tag{A.16}$$

The relaxation comes from the observation that Equation (A.15) minimizes over all feature distributions on $\mathcal{S}^d$, while Equation (A.16) only considers the realizable ones.

Relation between Equation (A.11) and minimizing average pairwise Gaussian potential (i.e., minimizing $\mathcal{L}_{\mathsf{uniform}}$). In view of the Proposition 2.4.2 and the proof of Theorem 2.4.7, we know that the uniform distribution σ_d is the unique minimizer of both of the following problems:

$$\begin{aligned}
\{\sigma_d\} &= \min_{\mu \in \mathcal{M}(\mathcal{S}^d)} \log \int_{\mathcal{S}^d} \int_{\mathcal{S}^d} e^{u^\mathsf{T} v/\tau} \,\mathrm{d}\mu(v) \,\mathrm{d}\mu(u), \\
\{\sigma_d\} &= \min_{\mu \in \mathcal{M}(\mathcal{S}^d)} \int_{\mathcal{S}^d} \log \int_{\mathcal{S}^d} e^{u^\mathsf{T} v/\tau} \,\mathrm{d}\mu(v) \,\mathrm{d}\mu(u).
\end{aligned}$$

So pushing the log inside the outer integral doesn't change the solution. However, if we push the log all the way inside the inner integral, the problem becomes equivalent with minimizing the norm of the mean, i.e.,

$$\min_{\mu \in \mathcal{M}(\mathcal{S}^d)} \mathbb{E}_{U \sim \mu} [U]^\mathsf{T} \, \mathbb{E}_{U \sim \mu} [U],$$

which is minimized for any distribution with mean being the all-zeros vector 0, e.g., $\frac{1}{2}\delta_u + \frac{1}{2}\delta_{-u}$ for any $u \in \mathcal{S}^d$ (where δ_u is the Dirac delta distribution at u s.t. $\delta_u(S) = \mathbb{1}_S(u)$, $\forall S \in \mathcal{B}(\mathcal{S}^d)$). Therefore, the location of the log is important.

Theorem A.1.9 (Single negative sample). If perfectly aligned and uniform encoders exist, they form the exact minimizers of the contrastive loss $\mathcal{L}_{\mathsf{contrastive}}(f; \tau, M)$ for fixed $\tau > 0$ and $M = 1$.

Proof of Theorem A.1.9. Since $M = 1$, we have

$$
\begin{aligned}
\mathcal{L}_{\mathsf{contrastive}}(f;\tau,1) &= \mathop{\mathbb{E}}_{\substack{(x,y)\sim p_{\mathsf{pos}} \\ x^-\sim p_{\mathsf{data}}}}\left[-\frac{1}{\tau}f(x)^\intercal f(y) + \log\left(e^{f(x)^\intercal f(y)/\tau} + e^{f(x^-)^\intercal f(x)/\tau}\right)\right] \\
&\geq \mathop{\mathbb{E}}_{\substack{x\sim p_{\mathsf{data}} \\ x^-\sim p_{\mathsf{data}}}}\left[-\frac{1}{\tau} + \log\left(e^{1/\tau} + e^{f(x^-)^\intercal f(x)/\tau}\right)\right] \qquad \text{(A.17)} \\
&\geq -\frac{1}{\tau} + \min_{\mu\in\mathcal{M}(\mathcal{S}^d)} \int_{\mathcal{S}^d}\int_{\mathcal{S}^d} \log\left(e^{1/\tau} + e^{u^\intercal v/\tau}\right) \mathrm{d}\mu(u)\,\mathrm{d}\mu(v) \qquad \text{(A.18)} \\
&= -\frac{1}{\tau} + \min_{\mu\in\mathcal{M}(\mathcal{S}^d)} \int_{\mathcal{S}^d}\int_{\mathcal{S}^d} \log\left(e^{1/\tau} + e^{(2-\|u-v\|_2^2)/(2\tau)}\right) \mathrm{d}\mu(u)\,\mathrm{d}\mu(v).
\end{aligned}
$$

By the definition of perfect alignment, the equality in Equation (A.17) is satisfied iff f is perfectly aligned.

Consider the function $f\colon (0,4] \to \mathbb{R}_+$ defined as

$$f(t) = \log(e^{\frac{1}{\tau}} + e^{\frac{2-t}{2\tau}}).$$

It has the following properties:

- $-f'(t) = \frac{1}{2\tau}\frac{e^{-\frac{t}{2\tau}}}{1+e^{-\frac{t}{2\tau}}} = \frac{1}{2\tau}(1-(1+e^{-\frac{t}{2\tau}})^{-1})$ is strictly completely monotone on $(0,+\infty)$:

  $\forall t \in (0,+\infty)$,

$$
\begin{aligned}
\frac{1}{2\tau}(1-(1+e^{-\frac{t}{2\tau}})^{-1}) &> 0 \\
(-1)^n\frac{\mathrm{d}^n}{\mathrm{d}t^n}\frac{1}{2\tau}(1-(1+e^{-\frac{t}{2\tau}})^{-1}) &= \frac{n!}{(2\tau)^{n+1}}(1+e^{-\frac{t}{2\tau}})^{-(n+1)} > 0, \qquad n = 1,2,\dots.
\end{aligned}
$$

- f is bounded on $(0,4]$.

In view of Lemma A.1.4, we have that the equality in Equation (A.18) is satisfied iff the feature distribution induced by f (i.e., the pushforward measure $p_{\mathsf{data}} \circ f^{-1}$) is σ_d, that is, in other words, f is perfectly uniform.

Therefore,

$$\begin{aligned}\mathcal{L}_{\mathsf{contrastive}}(f;\tau,1) &\geq -\frac{1}{\tau} + \int_{\mathcal{S}^d}\int_{\mathcal{S}^d} \log\left(e^{1/\tau} + e^{u^\intercal v/\tau}\right) \mathrm{d}\sigma_d(u)\,\mathrm{d}\sigma_d(v) \\ &= \text{constant independent of } f,\end{aligned}$$

where equality is satisfied iff f is perfectly aligned and uniform. This concludes the proof. □

Difference between conditions of Theorems 2.4.7 and A.1.9. We remark that the statement in Theorem A.1.9 is weaker than the previous Theorem 2.4.7. Theorem A.1.9 is conditioned on the existence perfectly aligned and uniform encoders. It only shows that $\mathcal{L}_{\mathsf{contrastive}}(f;\tau,M=1)$ favors alignment under the condition that perfect uniformity is realizable, and vice versa. In Theorem 2.4.7, $\mathcal{L}_{\mathsf{contrastive}}$ decomposes into two terms, each favoring alignment and uniformity. Therefore, the decomposition in Theorem 2.4.7 is exempof t from this constraint.

A.2 Experiment Details

All experiments are performed on 1-4 NVIDIA Titan Xp, Titan X PASCAL, Titan RTX, or 2080 Ti GPUs.

A.2.1 CIFAR-10, STL-10 and NYU-Depth-V2 Experiments

For CIFAR-10, STL-10 and NYU-Depth-V2 experiments, we use the following settings, unless otherwise stated in Tables A.3 and A.4 below:

- Standard data augmentation procedures are used for generating positive pairs, including resizing, cropping, horizontal flipping, color jittering, and random grayscale conversion. This follows prior empirical work in contrastive representation learning [171, 153, 76, 7].
- Neural network architectures follow the corresponding experiments on these

datasets in Tian et al. [153]. For NYU-DEPTH-V2 evaluation, the architecture of the depth prediction CNN is described in Table A.1.

- We use minibatch stochastic gradient descent (SGD) with 0.9 momentum and 0.0001 weight decay.
- We use linearly scaled learning rate (0.12 per 256 batch size) [53].
  - CIFAR-10 and STL-10: Optimization is done over 200 epochs, with learning rate decayed by a factor of 0.1 at epochs 155, 170, and 185.
  - NYU-DEPTH-V2: Optimization is done over 400 epochs, with learning rate decayed by a factor of 0.1 at epochs 310, 340, and 370.
- Encoders are optimized over the training split. For evaluation, we freeze the encoder, and train classifiers / depth predictors on the training set samples, and test on the validation split.
  - CIFAR-10 and STL-10: We use standard train-val split. Linear classifiers are trained with Adam [88] over 100 epochs, with $\beta_1 = 0.5, \beta_2 = 0.999, \epsilon = 10^{-8}$, 128 batch size, and an initial learning rate of 0.001, decayed by a factor of 0.2 at epochs 60 and 80.
  - NYU-DEPTH-V2: We use the train-val split on the 1449 labeled images from Nathan Silberman and Fergus [118]. Depth predictors are trained with Adam [88] over 120 epochs, with $\beta_1 = 0.5, \beta_2 = 0.999, \epsilon = 10^{-8}$, 128 batch size, and an initial learning rate of 0.003, decayed by a factor of 0.2 at epochs 70, 90, 100, and 110.

At each SGD iteration, a minibatch of K positive pairs is sampled $\{(x_i, y_i)\}_{i=1}^{K}$, and the three losses for this minibatch are calculated as following:

- $\mathcal{L}_{\mathsf{contrastive}}$: For each x_i, the sample contrastive loss is taken with the positive being y_i, and the negatives being $\{y_j\}_{j \neq i}$. For each y_i, the sample loss is computed

Operator	Input Spatial Shape	Input #Channel	Kernel Size	Stride	Padding	Output Spatial Shape	Output #Channel
Input	$[h_{\mathsf{in}}, w_{\mathsf{in}}]$	c_{in}	—	—	—	$[h_{\mathsf{in}}, w_{\mathsf{in}}]$	c_{in}
Conv. Transpose + BN + ReLU	$[h_{\mathsf{in}}, w_{\mathsf{in}}]$	c_{in}	3	2	1	$[2h_{\mathsf{in}}, 2w_{\mathsf{in}}]$	$\lfloor c_{\mathsf{in}}/2 \rfloor$
Conv. Transpose + BN + ReLU	$[2h_{\mathsf{in}}, 2w_{\mathsf{in}}]$	$\lfloor c_{\mathsf{in}}/2 \rfloor$	3	2	1	$[4h_{\mathsf{in}}, 4w_{\mathsf{in}}]$	$\lfloor c_{\mathsf{in}}/4 \rfloor$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Conv. Transpose + BN + ReLU	$[h_{\mathsf{out}}/2, w_{\mathsf{out}}/2]$	$\lfloor c_{\mathsf{in}}/2^{n-1} \rfloor$	3	2	1	$[h_{\mathsf{out}}, w_{\mathsf{out}}]$	$\lfloor c_{\mathsf{in}}/2^{n} \rfloor$
Conv.	$[h_{\mathsf{out}}, w_{\mathsf{out}}]$	$\lfloor c_{\mathsf{in}}/2^{n} \rfloor$	3	1	1	$[h_{\mathsf{out}}, w_{\mathsf{out}}]$	1

Table A.1: NYU-Depth-V2 CNN depth predictor architecture. Each Conv. Transpose+BN+ReLU block increases the spatial shape by a factor of 2, where BN denotes Batch Normalization [82]. A sequence of such blocks computes a tensor of the correct spatial shape, from an input containing intermediate activations of a CNN encoder (which downsamples the input RGB image by a power of 2). A final convolution at the end computes the single-channel depth prediction.

similarly. The minibatch loss is calculated by aggregating these $2K$ terms:

$$\frac{1}{2K}\sum_{i=1}^{K}\log\frac{e^{f(x_i)^\mathsf{T} f(y_i)/\tau}}{\sum_{j=1}^{K} e^{f(x_i)^\mathsf{T} f(y_j)/\tau}} + \frac{1}{2K}\sum_{i=1}^{K}\log\frac{e^{f(x_i)^\mathsf{T} f(y_i)/\tau}}{\sum_{j=1}^{K} e^{f(x_j)^\mathsf{T} f(y_i)/\tau}}.$$

This calculation follows empirical practices and is similar to Oord et al. [119], Hénaff et al. [72], and *end-to-end* in He et al. [70].

- $\mathcal{L}_{\mathsf{align}}$: The minibatch alignment loss is straightforwardly computed as

$$\frac{1}{K}\sum_{i=1}^{K}\|f(x_i) - f(y_i)\|_2^{\alpha}.$$

- $\mathcal{L}_{\mathsf{uniform}}$: The minibatch uniform loss is calculated by considering each pair of $\{x_i\}_i$ and $\{y_i\}_i$:

$$\frac{1}{2}\log\left(\frac{2}{K(K-1)}\sum_{i\neq j} e^{-t\|f(x_i)-f(x_j)\|_2^2}\right) + \frac{1}{2}\log\left(\frac{2}{K(K-1)}\sum_{i\neq j} e^{-t\|f(y_i)-f(y_j)\|_2^2}\right).$$

Tables A.3 and A.4 below describe the full specifications of all 304 STL-10 and 64 NYU-Depth-V2 encoders. These experiment results are visualized in main paper Figure 2-5, showing a clear connection between representation quality and $\mathcal{L}_{\mathsf{align}}$ & $\mathcal{L}_{\mathsf{uniform}}$ metrics.

ImageNet-100 Classes									
n02869837	n01749939	n02488291	n02107142	n13037406	n02091831	n04517823	n04589890	n03062245	n01773797
n01735189	n07831146	n07753275	n03085013	n04485082	n02105505	n01983481	n02788148	n03530642	n04435653
n02086910	n02859443	n13040303	n03594734	n02085620	n02099849	n01558993	n04493381	n02109047	n04111531
n02877765	n04429376	n02009229	n01978455	n02106550	n01820546	n01692333	n07714571	n02974003	n02114855
n03785016	n03764736	n03775546	n02087046	n07836838	n04099969	n04592741	n03891251	n02701002	n03379051
n02259212	n07715103	n03947888	n04026417	n02326432	n03637318	n01980166	n02113799	n02086240	n03903868
n02483362	n04127249	n02089973	n03017168	n02093428	n02804414	n02396427	n04418357	n02172182	n01729322
n02113978	n03787032	n02089867	n02119022	n03777754	n04238763	n02231487	n03032252	n02138441	n02104029
n03837869	n03494278	n04136333	n03794056	n03492542	n02018207	n04067472	n03930630	n03584829	n02123045
n04229816	n02100583	n03642806	n04336792	n03259280	n02116738	n02108089	n03424325	n01855672	n02090622

Table A.2: 100 randomly selected ImageNet classes forming the ImageNet-100 subset. These classes are the same as the ones used by Tian et al. [153].

A.2.2 ImageNet and ImageNet-100 with Momentum Contrast (MoCo) Variants

MoCo and MoCo v2 with $\mathcal{L}_{\mathsf{align}}$ and $\mathcal{L}_{\mathsf{uniform}}$. At each SGD iteration, let

- K be the minibatch size,
- $\{f(x_i)_i\}_{i=1}^{K}$ be the batched query features encoded by the current up-to-date encoder f (i.e., `q` in Algorithm 1 of He et al. [70]),
- $\{f_{\mathsf{EMA}}(y_i)\}_{i=1}^{K}$ be the batched key features encoded by the exponential moving average encoder f_{EMA} (i.e., `k` in Algorithm 1 of He et al. [70]),
- $\{\mathtt{queue}_j\}_{j=1}^{N}$ be the feature queue, where N is the queue size.

$\mathcal{L}_{\mathsf{align}}$ and $\mathcal{L}_{\mathsf{uniform}}$ for this minibatch are calculated as following:

- $\mathcal{L}_{\mathsf{align}}$: The minibatch alignment loss is computed as disparity between features from the two encoders:

$$\frac{1}{K}\sum_{i=1}^{K}\|f(x_i) - f_{\mathsf{EMA}}(y_i)\|_2^{\alpha}.$$

- $\mathcal{L}_{\mathsf{uniform}}$: We experiment with two forms of $\mathcal{L}_{\mathsf{uniform}}$:

1. Only computing pairwise distance between $\{f(x_i)\}_i$ and $\{\texttt{queue}_j\}_j$:

$$\log\left(\frac{1}{NK}\sum_{i=1}^{K}\sum_{j=1}^{N} e^{-t\left\|f(x_i)-\texttt{queue}_j\right\|_2^2}\right). \tag{A.19}$$

2. Also computing pairwise distance inside $\{f(x_i)\}_i$:

$$\begin{aligned}\log\Bigg(&\frac{2}{2NK+K(K-1)}\sum_{i=1}^{K}\sum_{j=1}^{N} e^{-t\left\|f(x_i)-\texttt{queue}_j\right\|_2^2} \\ &+\frac{2}{2NK+K(K-1)}\sum_{i\neq j} e^{-t\|f(x_i)-f(x_j)\|_2^2}\Bigg).\end{aligned} \tag{A.20}$$

ImageNet-100 with MoCo

ImageNet-100 details. We use the same ImageNet-100 sampled by Tian et al. [153], containing the 100 randomly selected classes listed in Table A.2.

MoCo settings. Our MoCo experiment settings below mostly follow He et al. [70] and the unofficial implementation by Tian [152], because the official implementation was not released at the time of performing these analyses:

- Standard data augmentation procedures are used for generating positive pairs, including resizing, cropping, horizontal flipping, color jittering, and random grayscale conversion, following Tian [152].
- Encoder architecture is ResNet50 [69].
- We use minibatch stochastic gradient descent (SGD) with 128 batch size, 0.03 initial learning rate, 0.9 momentum and 0.0001 weight decay.
- Optimization is done over 240 epochs, with learning rate decayed by a factor of 0.1 at epochs 120, 160, and 200.
- We use 0.999 exponential moving average factor, following He et al. [70].
- For evaluation, we freeze the encoder, and train a linear classifier on the training set samples, and test on the validation split. Linear classifiers are trained with

minibatch SGD over 60 epochs, with 256 batch size, and an initial learning rate of 10, decayed by a factor of 0.2 at epochs 30, 40, and 50.

Table A.5 below describes the full specifications of all 45 ImageNet-100 encoders. These experiment results are visualized in main paper Figure 2-9a, showing a clear connection between representation quality and $\mathcal{L}_{\mathsf{align}}$ & $\mathcal{L}_{\mathsf{uniform}}$ metrics.

ImageNet with MoCo v2

MoCo v2 settings. Our MoCo v2 experiment settings directly follow Chen et al. [26] and the official implementation [27]:

- Standard data augmentation procedures are used for generating positive pairs, including resizing, cropping, horizontal flipping, color jittering, random grayscale conversion, and random Gaussian blurring, following Chen et al. [27].
- Encoder architecture is ResNet50 [69].
- We use minibatch stochastic gradient descent (SGD) with 256 batch size, 0.03 initial learning rate, 0.9 momentum and 0.0001 weight decay.
- Optimization is done over 200 epochs, with learning rate decayed by a factor of 0.1 at epochs 120 and 160.
- We use 0.999 exponential moving average factor, 65536 queue size, 128 feature dimensions.
- For evaluation, we freeze the encoder, and train a linear classifier on the training set samples, and test on the validation split. Linear classifiers are trained with minibatch SGD over 100 epochs, with 256 batch size, and an initial learning rate of 30, decayed by a factor of 0.1 at epochs 60 and 80.

Unlike the MoCo experiments on ImageNet-100, which were based on unofficial implementations for reasons stated in Sec. A.2.2, the MoCo v2 experiments on full ImageNet were based on the official implementation by Chen et al. [27]. We provide a reference implementation that can fully reproduce the results in Table 2.5 at `https://github.com/SsnL/moco_align_uniform`, where we also provide a model checkpoint (trained using $\mathcal{L}_{\mathsf{align}}$ and $\mathcal{L}_{\mathsf{uniform}}$) of 67.694% validation top1 accuracy.

A.2.3 BookCorpus with Quick-Thought Vectors Variants

BookCorpus details. Since the original BookCorpus dataset [177] is not distributed anymore, we use the unofficial code by Kobayashi [91] to recreate our copy. Our copy ended up containing 52,799,513 training sentences and 50,000 validation sentences, compared to the original copy used by Quick-Thought Vectors [106], which contains 45,786,400 training sentences and 50,000 validation sentences.

Quick-Thought Vectors with $\mathcal{L}_{\mathsf{align}}$ and $\mathcal{L}_{\mathsf{uniform}}$. With Quick-Thought Vectors, the positive pairs are the neighboring sentences. At each optimization iteration, let

- $\{x_i\}_{i=1}^K$ be the K *consecutive* sentences forming this minibatch, where K be the minibatch size,
- f and g be the two RNN sentence encoders.

The original Quick-Thought Vectors [106] does not $l2$-normalize on encoder outputs during training the encoder. Here we describe the calculation of $\mathcal{L}_{\mathsf{contrastive}}$, $\mathcal{L}_{\mathsf{align}}$, and $\mathcal{L}_{\mathsf{uniform}}$ for $l2$-normalized encoders, in our modified Quick-Thought Vectors method. Note that this does not affect evaluation since features are $l2$-normalized before using in downstream tasks, following the original Quick-Thought Vectors [106]. For a minibatch, these losses are calculated as following:

- $\mathcal{L}_{\mathsf{contrastive}}$ with temperature:

$$\begin{aligned}
&\frac{1}{K}\,\texttt{ce}(\texttt{softmax}(\{f(x_1)^\mathsf{T} g(x_j)\}_j), \{0,1,0,\dots,0\}) \\
&\quad + \frac{1}{K}\sum_{i=2}^{K-1}\texttt{ce}(\texttt{softmax}(\{f(x_i)^\mathsf{T} g(x_j)\}_j), \{\underbrace{0,\dots,0}_{(i-2)\text{ 0's}}, \frac{1}{2}, 0, \frac{1}{2}, \underbrace{0,\dots,0}_{(K-i-1)\text{ 0's}}\}) \\
&\quad + \frac{1}{K}\,\texttt{ce}(\texttt{softmax}(\{f(x_K)^\mathsf{T} g(x_j)\}_j), \{0,\dots,1,0\}),
\end{aligned}$$

where $\mathsf{ce}(p, q)$ is the cross entropy between prediction p and target q.

This is almost identical with the original contrastive loss used by Quick-Thought Vectors, except that this does not additionally manually masks out the entries $f(x_i)^\mathsf{T} g(x_i)$ with zeros, which is unnecessary with $l2$-normalization.

- $\mathcal{L}_{\mathsf{align}}$: The minibatch alignment loss is computed as disparity between features from the two encoders encoding neighboring sentences (assuming $K >= 2$):

$$\frac{1}{K}\|f(x_1) - g(x_2)\|_2^\alpha + \frac{1}{2K}\sum_{i=2}^{K-2} \left(\|f(x_{i-1}) - g(x_i)\|_2^\alpha + \|f(x_i) - g(x_{i+1})\|_2^\alpha\right) \\ + \frac{1}{K}\|f(x_{K-1}) - g(x_K)\|_2^\alpha.$$

- $\mathcal{L}_{\mathsf{uniform}}$: We combine the uniformity losses for each of f and g by summing them (instead of averaging since f and g are two different encoders):

$$\frac{2}{K(K-1)}\sum_{i \neq j} e^{-t\|f(x_i) - f(x_j)\|_2^2} + \frac{2}{K(K-1)}\sum_{i \neq j} e^{-t\|g(x_i) - g(x_j)\|_2^2}.$$

Our experiment settings below mostly follow the official implementation by Logeswaran and Lee [106]:

- Sentence encoder architecture is bi-directional Gated Recurrent Unit (GRU) [29] with inputs from a 620-dimensional word embedding trained jointly from scratch.

- We use Adam [88] with $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$, 400 batch size, 0.0005 constant learning rate, and 0.5 gradient norm clipping.

- Optimization is done during 1 epoch over the training data.

- For evaluation on a binary classification task, we freeze the encoder, and fit a logistic classifier with $l2$ regularization on the encoder outputs. A 10-fold cross validation is performed to determine the regularization strength among

$\{1, 2^{-1}, \ldots, 2^{-8}\}$, following Kiros et al. [90] and Logeswaran and Lee [106]. The classifier is finally tested on the validation split.

Table A.6 below describes the full specifications of all 108 BookCorpus encoders along with 6 settings that lead to training instability (i.e., `NaN` occurring). These experiment results are visualized in main paper Figure 2-9b, showing a clear connection between representation quality and $\mathcal{L}_{\mathsf{align}}$ & $\mathcal{L}_{\mathsf{uniform}}$ metrics. For the unnormalized encoders, the features are normalized before calculated $\mathcal{L}_{\mathsf{align}}$ and $\mathcal{L}_{\mathsf{uniform}}$ metrics, since they are nonetheless still normalized before being used in downstream tasks [106].

Table A.3: Experiment specifications for all 304 STL-10 encoders. We report the encoder representation quality measured by accuracy of linear and k-nearest neighbor (k-NN) with $k = 5$ classifiers on either encoder outputs or fc7 activations, via both a 5-fold cross validation of the training set and the held out validation set.
For encoder initialization, "`rand`" refers to standard network initialization, and symbols denote finetuning from a pretrained encoder, obtained via the experiment row marked with the same symbol. Initial learning rates (LRs) are usually either fixed as 0.12 or computed via a linear scaling (0.12 per 256 batch size). Dimensionality (abbreviated as "Dim.") shows the ambient dimension of the output features, i.e., they live on the unit hypersphere of one less dimension. The last three rows show encoders that are used to initialize finetuning, but are not part of the 285 encoders plotted in main paper Figure 3, due to their unusual batch size of 786. Their accuracy and $\mathcal{L}_{\mathsf{align}}$ & $\mathcal{L}_{\mathsf{uniform}}$ metrics follow the same trend shown in Figure 2-5a.

Losses			Init.	Epochs	Batch Size	Initial LR	Dim.	Training Set 5-Fold Cross Val. Accuracy ↑				Validation Set Accuracy ↑			
$\mathcal{L}_{\mathsf{contrastive}}$	$\mathcal{L}_{\mathsf{align}}$	$\mathcal{L}_{\mathsf{uniform}}$						Output + Linear	Output + 5-NN	fc7 + Linear	fc7 + 5-NN	Output + Linear	Output + 5-NN	fc7 + Linear	fc7 + 5-NN
—	$1.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	2	0.0009375	128	—	—	—	—	19.31%	22.56%	47.58%	35.30%
—	$1.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	3	0.00140625	128	—	—	—	—	43.97%	42.89%	56.89%	47.63%
—	$1.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	4	0.001875	128	—	—	—	—	53.96%	52.89%	62.86%	55.06%
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.07)$	—	—	`rand`	200	16	0.0075	128	—	—	—	—	70.46%	70.54%	75.54%	69.63%
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	`rand`	200	16	0.0075	128	—	—	—	—	69.59%	70.04%	76.23%	68.38%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	16	0.0075	128	—	—	—	—	74.68%	74.34%	79.06%	73.68%
—	$1.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	16	0.0075	128	—	—	—	—	74.75%	73.00%	77.84%	71.70%
—	$1.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	16	0.0075	128	—	—	—	—	73.93%	74.09%	79.25%	73.38%
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	`rand`	200	16	0.12	128	—	—	—	—	67.30%	66.36%	71.53%	66.38%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	16	0.12	128	—	—	—	—	71.93%	71.24%	75.49%	69.89%
—	$1.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	16	0.12	128	—	—	—	—	71.85%	70.21%	74.65%	69.88%
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.07)$	—	—	`rand`	200	32	0.015	128	—	—	—	—	71.80%	72.04%	77.29%	70.74%
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	`rand`	200	32	0.015	128	—	—	—	—	73.39%	73.39%	79.43%	73.85%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	32	0.015	128	—	—	—	—	78.04%	76.60%	82.23%	76.04%
—	$1.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	32	0.015	128	—	—	—	—	78.71%	76.45%	81.66%	76.25%

	$\mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	`rand`	200	32	0.12	128	—	—	—	—	70.43%	69.66%	74.95%	69.69%
	—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	32	0.12	128	—	—	—	—	75.40%	73.70%	78.56%	73.21%
	—	$1.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	32	0.12	128	—	—	—	—	75.83%	73.95%	78.48%	73.55%
	$\mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	`rand`	200	64	0.03	128	—	—	—	—	74.59%	74.48%	80.64%	75.52%
	—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	64	0.03	128	—	—	—	—	79.25%	77.84%	82.84%	76.53%
	—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	64	0.12	128	—	—	—	—	77.80%	75.75%	81.45%	75.49%
	—	$1.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	64	0.12	128	—	—	—	—	78.66%	76.19%	81.40%	75.30%
	—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	64	0.03	512	—	—	—	—	80.44%	78.05%	83.04%	77.29%
	—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	64	0.03	1024	—	—	—	—	81.48%	78.49%	82.88%	77.11%
	—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	64	0.03	1024	—	—	—	—	80.81%	77.80%	83.18%	77.15%
	$\mathcal{L}_{\mathsf{c}}(\tau{=}0.07)$	—	—	`rand`	200	128	0.06	128	—	—	—	—	73.14%	73.73%	79.90%	72.58%
	$\mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	`rand`	200	128	0.06	128	—	—	—	—	75.26%	74.88%	80.98%	75.36%
	—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	128	0.06	128	—	—	—	—	79.55%	78.09%	83.39%	76.96%
	$\mathcal{L}_{\mathsf{c}}(\tau{=}0.07)$	—	—	`rand`	200	128	0.12	128	—	—	—	—	73.11%	73.84%	78.44%	72.11%
	$\mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	`rand`	200	128	0.12	128	—	—	—	—	75.65%	74.80%	80.74%	74.58%
	$\mathcal{L}_{\mathsf{c}}(\tau{=}0.687)$	—	—	`rand`	200	128	0.12	128	—	—	—	—	74.13%	73.14%	79.81%	74.10%
	—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	128	0.12	128	—	—	—	—	79.74%	77.78%	82.70%	75.23%
	—	$1.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	128	0.12	128	—	—	—	—	80.19%	77.91%	82.75%	75.91%
	—	$0.75 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	256	0.12	64	—	—	—	—	78.40%	78.26%	83.46%	76.25%
	$\mathcal{L}_{\mathsf{c}}(\tau{=}0.07)$	—	—	`rand`	200	256	0.12	128	—	—	—	—	75.23%	75.86%	80.64%	73.56%
♡	$\mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	`rand`	200	256	0.12	128	—	—	—	—	76.09%	75.81%	81.49%	75.52%
	$\mathcal{L}_{\mathsf{c}}(\tau{=}0.6)$	—	—	`rand`	200	256	0.12	128	—	—	—	—	75.61%	74.56%	81.09%	75.36%
	—	$0.75 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	256	0.12	128	—	—	—	—	80.54%	78.55%	83.54%	76.81%
	—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	256	0.12	128	—	—	—	—	80.76%	78.57%	84.24%	76.60%
△	—	$1.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	256	0.12	128	—	—	—	—	81.29%	78.49%	83.55%	74.08%
	—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	256	0.12	256	—	—	—	—	81.79%	79.13%	84.11%	76.60%

—	$0.75 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	256	0.12	256	—	—	—	—	81.48%	79.61%	83.86%	76.79%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	256	0.12	256	—	—	—	—	80.95%	78.74%	83.69%	77.11%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	256	0.12	512	—	—	—	—	81.33%	78.76%	83.81%	76.88%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	360	0.16875	8192	—	—	—	—	82.49%	78.96%	83.86%	76.68%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	512	0.24	4096	—	—	—	—	82.34%	78.84%	84.06%	75.74%
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.07)$	—	—	rand	200	768	0.36	2	—	—	—	—	29.46%	25.50%	59.95%	52.83%
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	rand	200	768	0.36	2	—	—	—	—	30.66%	25.39%	48.61%	42.49%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	2	—	—	—	—	27.85%	26.04%	49.29%	43.10%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	2	—	—	—	—	29.05%	23.94%	45.39%	38.48%
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.07)$	—	—	rand	200	768	0.36	3	—	—	—	—	39.59%	39.66%	63.24%	56.64%
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	rand	200	768	0.36	3	—	—	—	—	42.29%	39.70%	68.35%	59.82%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	3	—	—	—	—	41.10%	39.63%	65.64%	56.04%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	3	—	—	—	—	41.40%	41.45%	67.88%	58.78%
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.07)$	—	—	rand	200	768	0.36	4	—	—	—	—	46.94%	47.08%	64.35%	58.10%
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	rand	200	768	0.36	4	—	—	—	—	53.39%	55.41%	73.93%	67.89%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	4	—	—	—	—	47.19%	51.69%	70.00%	62.36%
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.07)$	—	—	rand	200	768	0.36	16	—	—	—	—	64.20%	68.73%	75.66%	69.55%
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	rand	200	768	0.36	16	—	—	—	—	71.93%	73.54%	80.53%	74.66%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	16	—	—	—	—	65.41%	70.41%	77.18%	70.55%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	16	—	—	—	—	70.25%	74.99%	81.59%	74.52%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	32	—	—	—	—	70.30%	73.50%	79.63%	72.21%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	32	—	—	—	—	73.65%	76.93%	82.81%	75.19%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2.5)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	32	—	—	—	—	73.71%	77.40%	82.93%	75.86%
—	$0.75 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	64	—	—	—	—	77.33%	78.35%	84.00%	76.63%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	64	—	—	—	—	77.94%	78.23%	83.51%	76.59%
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.005)$	—	—	rand	200	768	0.36	128	67.88%	70.15%	74.64%	68.19%	68.14%	71.13%	75.14%	68.88%

$\mathcal{L}_c(\tau{=}0.01)$	—	—	rand	200	768	0.36	128	69.63%	70.62%	75.68%	68.99%	69.86%	70.98%	76.13%	69.65%
$\mathcal{L}_c(\tau{=}0.07)$	—	—	rand	200	768	0.36	128	75.01%	75.11%	80.93%	73.20%	75.46%	75.58%	81.34%	73.93%
$\mathcal{L}_c(\tau{=}0.08)$	—	—	rand	200	768	0.36	128	76.12%	76.06%	81.72%	73.95%	76.58%	76.79%	81.81%	74.43%
$\mathcal{L}_c(\tau{=}0.09)$	—	—	rand	200	768	0.36	128	77.15%	77.15%	82.52%	73.96%	77.74%	77.46%	83.23%	74.81%
$\mathcal{L}_c(\tau{=}0.1)$	—	—	rand	200	768	0.36	128	77.55%	77.40%	82.93%	74.29%	77.83%	77.81%	83.39%	75.19%
$\mathcal{L}_c(\tau{=}0.11)$	—	—	rand	200	768	0.36	128	78.48%	78.20%	83.29%	74.99%	79.01%	78.73%	83.73%	75.60%
$\mathcal{L}_c(\tau{=}0.125)$	—	—	rand	200	768	0.36	128	79.05%	78.06%	83.30%	74.53%	79.59%	78.55%	84.09%	75.55%
$\mathcal{L}_c(\tau{=}0.13)$	—	—	rand	200	768	0.36	128	79.46%	78.55%	**83.98%**	75.16%	79.80%	78.60%	**84.45%**	75.98%
$\mathcal{L}_c(\tau{=}0.15)$	—	—	rand	200	768	0.36	128	79.81%	78.47%	83.62%	74.64%	80.16%	78.99%	84.19%	75.20%
$\mathcal{L}_c(\tau{=}0.16)$	—	—	rand	200	768	0.36	128	79.54%	78.38%	83.35%	74.42%	80.04%	78.68%	83.88%	75.06%
$\mathcal{L}_c(\tau{=}0.175)$	—	—	rand	200	768	0.36	128	79.74%	78.20%	83.56%	74.80%	80.29%	78.49%	83.96%	75.81%
$\mathcal{L}_c(\tau{=}0.19)$	—	—	rand	200	768	0.36	128	80.14%	78.30%	83.52%	75.39%	80.46%	78.75%	83.89%	76.33%
$\mathcal{L}_c(\tau{=}0.2)$	—	—	rand	200	768	0.36	128	79.64%	77.80%	83.37%	75.07%	79.99%	77.96%	83.73%	75.98%
$\mathcal{L}_c(\tau{=}0.25)$	—	—	rand	200	768	0.36	128	79.27%	77.24%	82.70%	75.33%	79.50%	77.49%	83.10%	76.31%
$\mathcal{L}_c(\tau{=}0.3)$	—	—	rand	200	768	0.36	128	78.79%	77.01%	82.58%	75.16%	78.98%	77.18%	82.84%	75.74%
$\mathcal{L}_c(\tau{=}0.5)$	—	—	rand	200	768	0.36	128	76.57%	75.30%	81.18%	75.30%	76.66%	75.61%	81.61%	75.71%
$\mathcal{L}_c(\tau{=}0.75)$	—	—	rand	200	768	0.36	128	74.59%	73.41%	79.72%	74.27%	74.63%	73.52%	80.18%	75.01%
$\mathcal{L}_c(\tau{=}1)$	—	—	rand	200	768	0.36	128	72.88%	72.14%	79.16%	74.08%	73.00%	72.31%	79.54%	74.61%
$\mathcal{L}_c(\tau{=}2)$	—	—	rand	200	768	0.36	128	67.79%	67.15%	77.04%	71.65%	67.13%	66.77%	77.35%	71.84%
★ $\mathcal{L}_c(\tau{=}2.5)$	—	—	rand	200	768	0.36	128	66.11%	65.30%	75.80%	70.59%	65.33%	65.30%	76.31%	70.93%
$\mathcal{L}_c(\tau{=}5)$	—	—	rand	200	768	0.36	128	55.56%	55.74%	70.29%	65.25%	55.75%	55.83%	70.75%	65.58%
$\mathcal{L}_c(\tau{=}0.07)$	$0.5 \cdot \mathcal{L}_a(\alpha{=}2)$	—	rand	200	768	0.36	128	75.13%	75.59%	81.52%	73.55%	75.59%	76.26%	82.10%	74.33%
$\mathcal{L}_c(\tau{=}0.1)$	$0.5 \cdot \mathcal{L}_a(\alpha{=}2)$	—	rand	200	768	0.36	128	77.76%	78.02%	83.28%	74.56%	78.04%	78.44%	83.73%	75.33%
$\mathcal{L}_c(\tau{=}0.5)$	$0.5 \cdot \mathcal{L}_a(\alpha{=}2)$	—	rand	200	768	0.36	128	74.86%	73.92%	80.16%	74.55%	74.96%	73.93%	80.63%	75.13%
$\mathcal{L}_c(\tau{=}0.5)$	$0.5 \cdot \mathcal{L}_a(\alpha{=}2)$	—	rand	200	768	0.36	128	74.69%	74.10%	80.53%	74.77%	74.80%	74.28%	80.91%	75.31%
$\mathcal{L}_c(\tau{=}0.5)$	$\mathcal{L}_a(\alpha{=}2)$	—	rand	200	768	0.36	128	73.31%	72.84%	79.82%	73.73%	73.54%	72.94%	80.26%	74.58%

$\mathcal{L}_\mathsf{c}(\tau{=}0.07)$	$0.4\cdot\mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.2\cdot\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	75.77%	75.98%	81.50%	73.48%	76.11%	76.45%	82.08%	74.00%
$\mathcal{L}_\mathsf{c}(\tau{=}0.1)$	$0.4\cdot\mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.2\cdot\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	78.17%	77.61%	83.04%	74.54%	78.64%	78.10%	83.26%	75.45%
$\mathcal{L}_\mathsf{c}(\tau{=}0.5)$	$0.4\cdot\mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.2\cdot\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	77.73%	76.23%	81.96%	75.10%	77.98%	76.60%	82.38%	75.45%
$\mathcal{L}_\mathsf{c}(\tau{=}0.07)$	$0.3\cdot\mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.4\cdot\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	75.93%	75.55%	81.45%	73.18%	76.13%	76.00%	81.95%	74.11%
$\mathcal{L}_\mathsf{c}(\tau{=}0.1)$	$0.3\cdot\mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.4\cdot\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	77.98%	77.18%	82.77%	74.12%	78.38%	77.79%	83.51%	74.99%
$\mathcal{L}_\mathsf{c}(\tau{=}0.5)$	$0.3\cdot\mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.4\cdot\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	78.69%	76.99%	82.57%	75.12%	79.03%	77.38%	82.93%	75.46%
$\mathcal{L}_\mathsf{c}(\tau{=}0.07)$	$0.2\cdot\mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.6\cdot\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	75.71%	75.22%	80.94%	72.80%	76.05%	75.60%	81.56%	73.46%
$\mathcal{L}_\mathsf{c}(\tau{=}0.1)$	$0.2\cdot\mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.6\cdot\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	78.38%	77.85%	82.87%	74.36%	78.84%	78.54%	83.10%	74.73%
$\mathcal{L}_\mathsf{c}(\tau{=}0.5)$	$0.2\cdot\mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.6\cdot\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	79.72%	77.94%	83.03%	75.32%	80.04%	78.24%	83.28%	75.66%
$\mathcal{L}_\mathsf{c}(\tau{=}0.07)$	$0.1\cdot\mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.8\cdot\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	76.19%	75.62%	81.15%	73.09%	76.90%	76.21%	81.61%	74.48%
$\mathcal{L}_\mathsf{c}(\tau{=}0.1)$	$0.1\cdot\mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.8\cdot\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	78.59%	78.02%	83.18%	74.63%	78.68%	78.48%	83.76%	75.49%
$\mathcal{L}_\mathsf{c}(\tau{=}0.5)$	$0.1\cdot\mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.8\cdot\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	80.25%	78.32%	83.35%	74.26%	80.43%	78.71%	83.76%	75.44%
$\mathcal{L}_\mathsf{c}(\tau{=}0.07)$	—	$\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	76.31%	75.78%	81.59%	72.79%	76.69%	76.33%	82.23%	73.63%
$\mathcal{L}_\mathsf{c}(\tau{=}0.1)$	—	$\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	78.55%	77.94%	83.21%	74.67%	79.03%	78.45%	83.75%	75.71%
$\mathcal{L}_\mathsf{c}(\tau{=}0.5)$	—	$\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	79.93%	78.25%	82.92%	75.22%	80.30%	78.54%	83.34%	76.04%
$\mathcal{L}_\mathsf{c}(\tau{=}0.5)$	—	$\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	**80.84%**	78.87%	83.72%	75.56%	81.06%	79.05%	84.14%	76.48%
$\mathcal{L}_\mathsf{c}(\tau{=}0.5)$	—	$2\cdot\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	77.49%	76.15%	80.99%	74.41%	78.09%	76.83%	81.63%	75.11%
$0.5\cdot\mathcal{L}_\mathsf{c}(\tau{=}0.07)$	$0.5\cdot\mathcal{L}_\mathsf{a}(\alpha{=}2)$	—	`rand`	200	768	0.36	128	75.40%	75.53%	81.53%	73.91%	75.74%	76.19%	82.00%	74.63%
$0.5\cdot\mathcal{L}_\mathsf{c}(\tau{=}0.1)$	$0.5\cdot\mathcal{L}_\mathsf{a}(\alpha{=}2)$	—	`rand`	200	768	0.36	128	77.70%	77.70%	83.39%	75.27%	78.06%	78.26%	83.93%	76.21%
$0.5\cdot\mathcal{L}_\mathsf{c}(\tau{=}0.5)$	$0.5\cdot\mathcal{L}_\mathsf{a}(\alpha{=}2)$	—	`rand`	200	768	0.36	128	73.86%	73.12%	80.08%	74.54%	74.05%	73.18%	80.53%	75.14%
$0.5\cdot\mathcal{L}_\mathsf{c}(\tau{=}0.07)$	$0.4\cdot\mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.2\cdot\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	76.12%	76.22%	81.75%	73.68%	76.46%	76.75%	82.36%	74.44%
$0.5\cdot\mathcal{L}_\mathsf{c}(\tau{=}0.1)$	$0.4\cdot\mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.2\cdot\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	78.40%	78.01%	83.39%	75.21%	78.83%	78.30%	83.74%	75.84%
$0.5\cdot\mathcal{L}_\mathsf{c}(\tau{=}0.5)$	$0.4\cdot\mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.2\cdot\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	78.35%	76.49%	82.02%	75.60%	78.60%	77.18%	82.65%	76.19%
$0.5\cdot\mathcal{L}_\mathsf{c}(\tau{=}0.07)$	$0.3\cdot\mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.4\cdot\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	76.59%	75.74%	81.48%	73.59%	77.20%	76.43%	82.03%	74.36%
$0.5\cdot\mathcal{L}_\mathsf{c}(\tau{=}0.1)$	$0.3\cdot\mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.4\cdot\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	78.85%	77.43%	82.98%	74.87%	79.20%	77.95%	83.29%	75.60%
$0.5\cdot\mathcal{L}_\mathsf{c}(\tau{=}0.5)$	$0.3\cdot\mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.4\cdot\mathcal{L}_\mathsf{u}(t{=}2)$	`rand`	200	768	0.36	128	79.53%	77.56%	82.84%	75.19%	79.71%	77.95%	83.19%	76.08%

$0.5 \cdot \mathcal{L}_{\mathsf{c}}(\tau{=}0.07)$	$0.2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.6 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	77.07%	76.49%	81.78%	73.10%	77.44%	76.98%	82.33%	73.85%
$0.5 \cdot \mathcal{L}_{\mathsf{c}}(\tau{=}0.1)$	$0.2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.6 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	78.55%	78.04%	83.20%	74.30%	78.91%	78.38%	83.81%	75.18%
$0.5 \cdot \mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	$0.2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.6 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	80.47%	78.36%	83.42%	75.82%	80.88%	78.51%	83.83%	76.65%
$0.5 \cdot \mathcal{L}_{\mathsf{c}}(\tau{=}0.07)$	$0.1 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.8 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	76.30%	76.43%	81.72%	73.35%	76.56%	77.11%	82.11%	74.00%
$0.5 \cdot \mathcal{L}_{\mathsf{c}}(\tau{=}0.1)$	$0.1 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.8 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	78.71%	78.00%	83.35%	74.46%	79.29%	78.44%	83.81%	75.45%
$0.5 \cdot \mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	$0.1 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.8 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	80.51%	78.99%	83.57%	75.47%	80.95%	79.44%	83.98%	76.45%
$0.5 \cdot \mathcal{L}_{\mathsf{c}}(\tau{=}0.07)$	—	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	75.48%	76.10%	81.47%	72.97%	75.80%	76.86%	82.06%	73.81%
$0.5 \cdot \mathcal{L}_{\mathsf{c}}(\tau{=}0.1)$	—	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	77.78%	78.07%	83.23%	74.51%	78.38%	78.46%	83.89%	75.49%
$0.5 \cdot \mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	78.04%	76.18%	81.89%	73.67%	78.43%	76.44%	82.33%	74.44%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	—	`rand`	200	768	0.36	128	10.00%	10.36%	11.07%	14.20%	10.00%	9.40%	12.53%	14.27%
—	$0.9875 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.025 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	10.00%	9.90%	11.04%	13.72%	10.00%	10.94%	13.03%	13.64%
—	$0.975 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.05 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	10.00%	9.98%	10.65%	14.29%	10.00%	9.75%	12.11%	14.77%
—	$0.9 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.1 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	10.00%	10.08%	10.10%	13.62%	10.00%	9.95%	10.00%	13.49%
—	$0.95 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.1 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	10.00%	10.51%	10.15%	13.27%	10.00%	9.85%	10.00%	11.99%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.1 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	10.00%	9.93%	10.39%	14.38%	10.00%	10.26%	10.00%	14.03%
—	$0.56 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.12 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	75.93%	75.10%	80.88%	74.87%	75.99%	75.41%	81.40%	75.66%
—	$0.88 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.12 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	10.00%	10.13%	10.00%	12.89%	10.00%	11.18%	10.03%	12.43%
—	$0.9375 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.125 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	10.00%	10.52%	10.42%	13.71%	10.00%	9.14%	10.05%	14.26%
—	$0.57 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.14 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	76.35%	75.51%	81.07%	75.27%	76.55%	75.86%	81.69%	75.70%
—	$0.86 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.14 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	10.00%	9.07%	10.33%	14.24%	10.00%	9.91%	10.73%	15.08%
—	$0.855 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.145 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	10.00%	10.67%	10.30%	14.11%	10.00%	9.35%	11.70%	13.30%
—	$0.85 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.15 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	10.00%	10.17%	10.00%	12.97%	10.00%	10.05%	10.00%	13.16%
—	$0.925 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.15 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	10.00%	9.79%	10.10%	13.11%	10.00%	9.73%	10.11%	12.91%
—	$0.845 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.155 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	74.56%	74.06%	80.10%	74.93%	74.99%	74.39%	80.44%	75.83%
—	$0.58 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.16 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	77.03%	76.34%	81.25%	75.26%	77.33%	76.76%	81.80%	75.89%
—	$0.84 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.16 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	74.49%	74.03%	80.30%	74.72%	74.73%	74.10%	80.70%	75.13%

—	$0.9125 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.175 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	10.00%	9.41%	10.39%	13.64%	10.00%	10.14%	10.10%	14.14%
—	$0.59 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.18 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	77.25%	76.38%	81.39%	75.41%	77.65%	77.06%	81.68%	76.19%
—	$0.82 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.18 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	76.09%	75.10%	80.99%	75.63%	76.45%	75.48%	81.45%	76.48%
—	$0.91 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.18 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	75.11%	74.63%	80.50%	75.28%	75.40%	75.04%	80.85%	75.83%
—	$0.9075 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.185 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	75.29%	74.83%	80.64%	75.04%	75.69%	75.41%	80.93%	75.65%
—	$0.905 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.19 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	75.69%	74.61%	80.80%	74.98%	75.99%	74.95%	81.21%	75.59%
—	$0.9025 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.195 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	75.81%	74.93%	80.75%	74.66%	76.06%	75.29%	81.16%	75.14%
—	$0.8 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.2 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	76.52%	75.96%	81.05%	75.38%	76.75%	76.24%	81.29%	75.83%
—	$0.9 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.2 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	75.92%	75.02%	80.85%	75.36%	76.15%	75.29%	81.15%	76.24%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.2 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	75.14%	74.29%	80.39%	74.76%	75.46%	74.44%	80.64%	75.34%
—	$0.7 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.3 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	78.61%	77.00%	82.14%	75.73%	78.94%	77.50%	82.26%	76.34%
—	$0.6 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.4 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	79.36%	77.80%	82.63%	75.55%	79.60%	77.93%	82.86%	76.63%
—	$0.8 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.4 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	79.24%	77.52%	82.44%	75.23%	79.65%	77.89%	82.69%	75.71%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.4 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	78.45%	77.09%	82.30%	75.38%	78.85%	77.53%	82.86%	76.02%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	80.03%	78.47%	83.12%	75.14%	80.39%	78.70%	83.56%	75.70%
—	$0.75 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	79.72%	77.30%	82.69%	75.44%	79.96%	77.55%	83.35%	76.14%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	79.09%	77.50%	82.80%	75.46%	79.27%	77.96%	83.10%	76.45%
—	$0.4 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.6 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	80.23%	78.67%	83.49%	75.61%	80.45%	78.83%	84.01%	76.61%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.6 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	80.37%	78.82%	83.05%	75.54%	80.48%	79.11%	83.33%	76.50%
—	$0.7 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.6 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	80.29%	78.16%	83.40%	75.59%	80.59%	78.66%	83.83%	76.24%
—	$0.3 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.7 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	80.16%	78.91%	83.39%	**76.21%**	80.58%	79.51%	83.78%	77.03%
—	$0.2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.8 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	74.67%	78.15%	82.53%	75.83%	75.13%	78.63%	83.03%	76.45%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.8 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	80.59%	78.73%	83.73%	76.05%	81.08%	79.10%	84.04%	76.88%
—	$0.6 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.8 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	80.29%	78.74%	83.53%	75.75%	80.65%	78.89%	83.89%	76.86%
—	$0.1 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.9 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	69.77%	75.72%	80.55%	73.38%	70.29%	76.13%	80.88%	74.14%
—	$0.08 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.92 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	rand	200	768	0.36	128	67.65%	73.97%	79.35%	71.86%	68.04%	74.90%	79.84%	72.50%

—	$0.96 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.92 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	80.74%	78.71%	83.49%	76.14%	81.08%	79.26%	83.95%	77.26%
—	$0.06 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.94 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	66.88%	73.81%	79.21%	72.32%	67.46%	74.68%	79.56%	73.09%
—	$0.97 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.94 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	80.28%	78.45%	83.51%	75.68%	80.63%	78.63%	83.83%	76.33%
—	$0.04 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.96 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	63.89%	70.80%	76.33%	69.55%	64.21%	71.49%	77.10%	70.38%
—	$0.98 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.96 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	80.76%	78.69%	83.97%	75.63%	81.15%	78.89%	84.43%	76.78%
—	$\mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.975 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	79.94%	78.45%	83.34%	75.23%	80.44%	78.86%	83.65%	75.83%
—	$0.02 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.98 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	56.39%	63.06%	69.48%	62.85%	56.78%	63.90%	69.80%	63.82%
—	$0.99 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.98 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	80.24%	78.90%	83.34%	74.89%	80.45%	79.40%	83.76%	75.55%
—	$\mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.98 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	80.29%	78.64%	83.46%	75.23%	80.77%	78.84%	83.96%	75.90%
—	—	$\mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	20.62%	15.96%	24.52%	16.13%	20.50%	16.14%	24.64%	16.24%
—	$0.0025 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$\mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	36.14%	33.19%	46.82%	35.22%	36.28%	33.76%	47.04%	36.05%
—	$0.005 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$\mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	48.38%	49.74%	59.67%	49.55%	48.69%	50.41%	59.81%	50.40%
—	$0.0125 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$\mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	51.31%	57.94%	64.95%	57.49%	51.80%	58.75%	65.40%	58.01%
—	$0.025 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}1)$	$\mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	46.13%	51.81%	58.51%	51.30%	46.61%	52.65%	59.03%	51.99%
—	$0.025 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$\mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	57.34%	62.50%	69.09%	61.76%	57.89%	63.43%	69.58%	62.51%
—	$0.25 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}1)$	$\mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	70.80%	75.24%	80.59%	72.59%	71.40%	75.54%	81.20%	73.36%
—	$0.25 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$\mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	76.14%	78.45%	82.97%	75.90%	76.83%	78.88%	83.51%	76.74%
—	$0.3 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$\mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	76.72%	78.01%	83.26%	75.61%	77.30%	78.43%	83.79%	76.25%
—	$0.4 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$\mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	78.71%	77.76%	83.13%	75.42%	79.36%	78.01%	83.64%	76.24%
—	$0.5 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$\mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	80.41%	**79.18%**	83.85%	75.54%	80.03%	79.35%	84.20%	76.84%
□ —	$0.75 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$\mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	80.54%	78.84%	83.61%	75.26%	80.89%	79.29%	84.23%	76.28%
—	$\mathcal{L}_\mathsf{a}(\alpha{=}2)$	$\mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	80.32%	78.90%	83.48%	74.97%	80.76%	79.23%	83.75%	76.15%
—	$1.025 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$\mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	80.37%	78.69%	83.48%	75.78%	80.74%	79.06%	84.00%	76.56%
—	$1.25 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$\mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	80.50%	78.41%	83.54%	75.89%	80.84%	78.65%	83.95%	76.56%
—	$0.4 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$1.2 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	75.37%	73.62%	78.88%	71.55%	75.78%	73.83%	79.15%	72.35%
—	$0.3 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$1.4 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	rand	200	768	0.36	128	72.69%	75.62%	80.67%	73.49%	73.14%	75.99%	81.49%	74.20%

—	$0.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$1.5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	70.61%	73.50%	78.53%	71.85%	71.03%	74.10%	79.13%	72.50%
—	$0.2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$1.6 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	67.35%	70.98%	76.84%	69.13%	67.69%	71.64%	77.40%	69.91%
—	$0.1 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$1.8 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	64.43%	68.89%	74.24%	68.15%	65.01%	69.34%	74.70%	68.80%
—	$0.0875 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$1.825 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	63.38%	68.83%	73.56%	67.33%	64.05%	69.76%	73.91%	68.14%
—	$0.075 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$1.85 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	63.02%	69.32%	74.49%	68.22%	63.44%	69.91%	75.05%	69.06%
—	$0.0625 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$1.875 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	58.73%	64.37%	70.93%	63.74%	59.23%	65.14%	71.54%	64.69%
—	$0.05 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$1.9 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	57.61%	64.13%	69.13%	63.09%	58.03%	65.09%	69.43%	64.09%
—	$0.025 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$1.95 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	50.89%	57.70%	63.93%	57.83%	51.46%	58.39%	64.45%	58.34%
—	$0.0125 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$1.975 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	44.71%	50.89%	57.75%	51.21%	45.14%	51.99%	57.98%	52.11%
—	—	$2 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	21.99%	19.46%	28.94%	20.10%	21.91%	19.75%	29.65%	20.76%
—	$0.1 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$2 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	63.63%	70.70%	75.85%	69.41%	64.14%	71.43%	76.50%	69.99%
—	$0.2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$2 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	66.52%	72.89%	77.66%	70.98%	67.16%	73.52%	78.19%	71.79%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$2 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	10.00%	10.00%	10.00%	10.00%	10.00%	10.00%	10.00%	10.00%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$2.5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	10.00%	10.00%	10.00%	10.00%	10.00%	10.00%	10.00%	10.00%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$3 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	10.00%	10.00%	10.00%	10.00%	10.00%	10.00%	10.00%	10.00%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$4 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	10.00%	10.00%	10.00%	10.00%	10.00%	10.00%	10.00%	10.00%
—	—	$5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	19.61%	14.29%	21.70%	14.97%	19.64%	14.19%	21.61%	15.58%
—	$0.05 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	50.49%	55.71%	61.45%	55.15%	50.91%	56.71%	61.58%	56.19%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	128	10.00%	10.00%	10.00%	10.01%	10.00%	10.00%	10.00%	10.00%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	256	—	—	—	—	82.10%	79.45%	84.15%	77.10%
—	$0.75 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	256	—	—	—	—	81.53%	79.03%	83.54%	76.35%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	256	—	—	—	—	81.33%	79.06%	84.03%	75.89%
—	$0.025 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	512	—	—	—	—	75.76%	72.75%	78.29%	71.04%
—	$0.375 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	512	—	—	—	—	82.33%	79.18%	83.91%	76.44%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	512	—	—	—	—	82.55%	79.64%	84.29%	75.74%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	512	—	—	—	—	82.04%	78.79%	83.98%	76.50%

—	$0.025 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	1024	—	—	—	—	76.39%	72.45%	78.23%	70.59%
—	$0.05 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	1024	—	—	—	—	79.68%	75.43%	80.81%	73.45%
—	$0.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	1024	—	—	—	—	**83.03%**	79.63%	84.15%	76.10%
—	$0.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	1024	—	—	—	—	82.85%	79.44%	83.91%	75.35%
—	$0.375 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	1024	—	—	—	—	82.63%	79.33%	83.69%	76.09%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	1024	—	—	—	—	82.85%	**79.75%**	83.85%	76.81%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	1024	—	—	—	—	81.89%	79.09%	84.03%	75.51%
—	$0.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	768	0.36	1536	—	—	—	—	82.93%	79.55%	84.00%	75.81%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	1024	0.48	512	—	—	—	—	82.20%	79.36%	83.69%	75.73%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	1024	0.48	512	—	—	—	—	81.66%	79.03%	83.88%	75.49%
—	$0.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	1024	0.48	1024	—	—	—	—	82.40%	78.98%	83.34%	75.85%
—	$0.375 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	1024	0.48	1024	—	—	—	—	82.74%	79.48%	83.70%	76.59%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	200	1024	0.48	1024	—	—	—	—	82.51%	79.11%	83.46%	74.94%
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	△	12	256	0.12	128	—	—	—	—	79.31%	77.45%	83.34%	76.60%
—	$5e-05 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	—	♣	12	256	0.12	128	—	—	—	—	64.11%	62.45%	77.96%	68.56%
—	$0.0005 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	—	♣	12	256	0.12	128	—	—	—	—	63.90%	62.40%	77.81%	68.55%
—	$0.005 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	—	♣	12	256	0.12	128	—	—	—	—	61.53%	61.66%	76.83%	66.68%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	—	♣	12	256	0.12	128	—	—	—	—	10.36%	23.01%	49.19%	39.39%
—	—	$0.01 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	♣	12	256	0.12	128	—	—	—	—	44.75%	41.79%	55.59%	38.59%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.01 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	♣	12	256	0.12	128	—	—	—	—	10.03%	32.81%	57.95%	41.53%
—	—	$0.1 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	♣	12	256	0.12	128	—	—	—	—	54.78%	54.05%	65.77%	50.13%
—	—	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	♣	12	256	0.12	128	—	—	—	—	55.74%	52.03%	63.90%	50.59%
—	$0.005 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	♣	12	256	0.12	128	—	—	—	—	57.85%	55.18%	65.64%	53.33%
—	$0.05 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	♣	12	256	0.12	128	—	—	—	—	68.46%	66.07%	72.88%	64.65%
—	$0.4 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	♡	12	256	0.12	128	—	—	—	—	77.63%	76.65%	81.75%	75.95%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	♠	12	256	0.12	128	—	—	—	—	70.00%	68.21%	74.15%	66.77%

—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	♡	12	256	0.12	128	—	—	—	—	77.73%	76.33%	81.61%	76.00%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	◊	12	256	0.12	128	—	—	—	—	74.23%	72.89%	79.01%	71.46%
—	$0.625 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	◊	12	256	0.12	128	—	—	—	—	74.40%	72.84%	79.29%	71.41%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	♡	12	256	0.12	128	—	—	—	—	76.48%	75.86%	81.04%	75.43%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	◊	12	256	0.12	128	—	—	—	—	73.13%	72.24%	78.33%	71.15%
—	$1.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	♡	12	256	0.12	128	—	—	—	—	76.80%	75.75%	81.00%	75.11%
—	$1.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	◊	12	256	0.12	128	—	—	—	—	73.11%	71.73%	78.23%	71.79%
—	$1.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	♠	12	256	0.12	128	—	—	—	—	69.10%	67.21%	74.19%	66.25%
—	$1.875 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	◊	12	256	0.12	128	—	—	—	—	72.63%	71.08%	77.79%	70.98%
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	□	12	768	0.36	128	—	—	—	—	75.34%	74.00%	81.09%	73.23%
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	★	12	768	0.36	128	—	—	—	—	65.60%	64.25%	70.73%	64.79%
$0.5 \cdot \mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	★	12	768	0.36	128	—	—	—	—	69.64%	67.70%	74.89%	68.74%
$0.25 \cdot \mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	★	12	768	0.36	128	—	—	—	—	69.11%	68.34%	74.30%	69.30%
$0.05 \cdot \mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	★	12	768	0.36	128	—	—	—	—	70.43%	69.70%	76.08%	71.31%
$0.025 \cdot \mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	□	12	768	0.36	128	—	—	—	—	80.27%	78.65%	83.93%	77.00%
$0.025 \cdot \mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	★	12	768	0.36	128	—	—	—	—	70.00%	68.74%	76.24%	71.86%
$0.01 \cdot \mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	□	12	768	0.36	128	—	—	—	—	80.46%	78.88%	83.64%	**77.38%**
$0.01 \cdot \mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	★	12	768	0.36	128	—	—	—	—	68.13%	67.38%	75.63%	71.28%
—	$0.00025 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	—	★	12	768	0.36	128	—	—	—	—	65.94%	64.33%	75.14%	70.90%
—	$0.0005 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	—	★	12	768	0.36	128	—	—	—	—	64.88%	63.18%	74.78%	70.88%
—	$0.0005 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	—	★	12	768	0.36	128	—	—	—	—	64.89%	63.53%	74.76%	70.89%
—	$0.001 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	—	★	12	768	0.36	128	—	—	—	—	62.65%	61.93%	74.31%	70.36%
—	$0.0025 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	—	★	12	768	0.36	128	—	—	—	—	59.18%	60.09%	72.98%	69.41%
—	$0.005 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	—	★	12	768	0.36	128	—	—	—	—	52.18%	55.06%	71.40%	67.10%
—	$0.005 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	—	★	12	768	0.36	128	—	—	—	—	52.86%	55.95%	71.63%	67.76%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	—	★	12	768	0.36	128	—	—	—	—	10.00%	17.42%	36.69%	34.94%

—	—	$0.0001 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	60.32%	59.49%	70.65%	64.70%
—	—	$0.0005 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	44.34%	43.41%	61.06%	53.97%
—	$0.0005 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.0005 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	66.14%	66.13%	75.29%	70.20%
—	—	$0.001 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	41.61%	40.73%	56.91%	48.24%
—	$0.001 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.001 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	66.23%	66.55%	75.16%	70.25%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.001 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	10.00%	17.79%	35.06%	34.11%
—	$0.002 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.002 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	66.35%	67.07%	74.50%	70.33%
—	—	$0.01 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	44.64%	41.55%	50.75%	42.90%
—	$0.01 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.01 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	71.54%	70.71%	75.45%	70.43%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.01 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	10.00%	18.05%	32.93%	31.53%
—	$0.03 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.02 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	72.13%	71.86%	76.33%	71.78%
—	$0.025 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.025 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	73.40%	72.58%	76.44%	72.09%
—	$0.0375 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.025 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	72.54%	71.56%	76.14%	71.89%
—	$0.05 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.05 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	73.94%	72.63%	77.05%	72.36%
—	—	$0.1 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	54.51%	48.40%	60.60%	49.00%
—	$0.1 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.1 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	73.30%	72.21%	76.54%	72.13%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.1 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	67.45%	67.03%	74.04%	68.73%
—	$0.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.25 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	73.09%	71.66%	76.80%	71.16%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	72.18%	71.56%	76.38%	70.93%
—	—	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	39.45%	35.56%	47.18%	35.60%
—	$0.0005 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	43.58%	38.19%	49.38%	38.64%
—	$0.005 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	50.10%	47.36%	56.66%	48.73%
—	$0.05 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	65.65%	66.15%	71.48%	66.10%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	70.34%	70.04%	74.88%	68.76%
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	70.84%	69.88%	75.61%	69.34%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	66.83%	65.59%	72.09%	65.30%

	—	$1.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$1.5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	65.18%	62.32%	69.77%	62.31%
	—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$2 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	63.21%	61.86%	68.66%	60.80%
	—	$2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$2 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	★	12	768	0.36	128	—	—	—	—	61.93%	60.78%	68.54%	60.18%
$\Diamond$	$\mathcal{L}_{\mathsf{c}}(\tau{=}1)$	—	—	`rand`	200	786	0.12	128	—	—	—	—	70.35%	70.11%	80.41%	73.15%
$\clubsuit$	$\mathcal{L}_{\mathsf{c}}(\tau{=}2)$	—	—	`rand`	200	786	0.12	128	—	—	—	—	64.19%	62.38%	78.11%	68.77%
$\spadesuit$	$\mathcal{L}_{\mathsf{c}}(\tau{=}3)$	—	—	`rand`	200	786	0.12	128	—	—	—	—	55.04%	53.94%	74.95%	64.04%

Table A.4: Experiment specifications for all 64 NYU-Depth-V2 encoders. We report the encoder representation quality measured by mean squared error (MSE) of a CNN depth predictor trained on conv5 or conv4 activations, via both a 5-fold cross validation of the training set and the held out validation set.
All encoders in this table use standard network initialization (denoted as "`rand`"). Dimensionality (abbreviated as "Dim.") shows the ambient dimension of the output features, i.e., they live on the unit hypersphere of one less dimension.

Losses			Init.	Epochs	Batch Size	Initial LR	Dim.	Training Set 5-Fold Cross Val. MSE ↓		Validation Set MSE ↓	
$\mathcal{L}_{\mathsf{contrastive}}$	$\mathcal{L}_{\mathsf{align}}$	$\mathcal{L}_{\mathsf{uniform}}$						conv5	conv4	conv5	conv4
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7405	0.7979	0.7378	0.7969
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.25)$	—	—	`rand`	400	128	0.06	128	0.7188	0.7747	0.7259	0.7761
—	$4.375 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.8039	0.8297	0.8032	0.8281
—	$3.625 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7290	0.7775	0.7303	0.7749
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7121	0.7689	0.7191	0.7725
—	$3.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7270	0.7741	0.7260	0.7772
$\mathcal{L}_{\mathsf{c}}(\tau{=}4)$	—	—	`rand`	400	128	0.06	128	0.7592	0.8195	0.7598	0.8175
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.3333 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7165	0.7697	0.7215	0.7693
—	$2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7300	0.7669	0.7226	0.7699
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.05)$	—	—	`rand`	400	128	0.06	128	0.7170	0.7672	0.7206	0.7637
$\mathcal{L}_{\mathsf{c}}(\tau{=}1)$	—	—	`rand`	400	128	0.06	128	0.7505	0.7958	0.7560	0.7965
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$7.5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.8188	0.8556	0.8302	0.8590
—	$1.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7237	0.7788	0.7224	0.7806
—	$4.625 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.8692	0.8820	0.8724	0.8840
—	$3.375 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7663	0.7935	0.7691	0.7938
—	$0.75 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	**0.7008**	0.7621	**0.7014**	0.7592
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.25 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7293	0.7997	0.7313	0.8013
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.07)$	—	—	`rand`	400	128	0.06	128	0.7079	**0.7468**	0.7105	**0.7460**
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.005)$	—	—	`rand`	400	128	0.06	128	0.7608	0.8109	0.7633	0.8149

—	$4 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7721	0.8195	0.7737	0.8190
—	$1.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7231	0.7810	0.7193	0.7889
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7044	0.7714	0.7047	0.7718
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7329	0.7751	0.7454	0.7786
—	$2.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7295	0.7747	0.7304	0.7785
—	$4.125 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7497	0.8129	0.7478	0.8128
—	$0.125 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$2.5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.8109	0.8535	0.8092	0.8523
—	$1.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7509	0.7892	0.7324	0.7926
—	$3.75 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7514	0.8005	0.7531	0.8003
—	$2.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7360	0.7706	0.7413	0.7747
—	$4.875 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.8699	0.8882	0.8717	0.8918
—	$3.125 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7203	0.7713	0.7138	0.7682
—	$1.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7261	0.7744	0.7259	0.7715
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—	`rand`	400	128	0.06	128	0.7334	0.7743	0.7293	0.7701
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.2857 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7456	0.8070	0.7423	0.8030
—	$2.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7289	0.7591	0.7250	0.7597
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$3 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7819	0.8352	0.7808	0.8314
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$10 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.8422	0.8896	0.8430	0.8857
—	$3 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7203	0.7642	0.7160	0.7643
—	$3.875 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7477	0.7980	0.7476	0.7960
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.4)$	—	—	`rand`	400	128	0.06	128	0.7181	0.7628	0.7163	0.7651
—	$0.75 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7670	0.8225	0.7700	0.8224
—	$1.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7311	0.7922	0.7265	0.7942
—	$1.75 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7323	0.7900	0.7297	0.7884
—	$4.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7592	0.8350	0.7585	0.8297
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7909	0.8517	0.7891	0.8526

$0.5 \cdot \mathcal{L}_{\mathsf{c}}(\tau{=}0.07)$	—	—	`rand`	400	128	0.06	128	0.7068	0.7594	0.7028	0.7624
—	$3.75 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7352	0.7853	0.7294	0.7817
—	$3.125 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7152	0.7661	0.7060	0.7667
—	$3.625 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7420	0.7925	0.7505	0.7970
—	$5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.8072	0.8631	0.8084	0.8617
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.1)$	—	—	`rand`	400	128	0.06	128	0.7074	0.7539	0.7124	0.7491
—	$1.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7255	0.7793	0.7199	0.7765
—	$7.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.8160	0.8512	0.8131	0.8505
—	$4.75 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.8102	0.8633	0.8084	0.8721
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$2.5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7696	0.8208	0.7669	0.8141
—	$2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7209	0.7839	0.7370	0.7867
$0.5 \cdot \mathcal{L}_{\mathsf{c}}(\tau{=}0.1)$	—	—	`rand`	400	128	0.06	128	0.7062	0.7586	0.7024	0.7575
$\mathcal{L}_{\mathsf{c}}(\tau{=}10)$	—	—	`rand`	400	128	0.06	128	0.7860	0.8375	0.7850	0.8335
—	$3.375 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7236	0.7703	0.7230	0.7728
—	$0.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7596	0.8122	0.7574	0.8107
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.3)$	—	—	`rand`	400	128	0.06	128	0.7337	0.7653	0.7361	0.7640
$\mathcal{L}_{\mathsf{c}}(\tau{=}5)$	—	—	`rand`	400	128	0.06	128	0.7801	0.8278	0.7715	0.8355
—	$3.25 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.7495	0.7903	0.7503	0.7941
—	$0.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$4 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	`rand`	400	128	0.06	128	0.8062	0.8597	0.8042	0.8608

Table A.5: Experiment specifications for all 45 ImageNet-100 ResNet50 encoders trained using methods based on Momentum Contrast (MoCo) [70]. We report the encoder representation quality measured by accuracy of a linear classifier on penultimate layer activations, via both a 3-fold cross validation of the training set and the held out validation set.
All encoders in this table use standard network initialization (denoted as "`rand`"). Dimensionality (abbreviated as "Dim.") shows the ambient dimension of the output features, i.e., they live on the unit hypersphere of one less dimension.
For $\mathcal{L}_{\mathsf{uniform}}$, the "Intra-batch" column denotes whether $\mathcal{L}_{\mathsf{uniform}}$ calculation includes pairwise distances within batch in addition to distances w.r.t. to the queue (i.e., Equation (A.20) vs. Equation (A.19)).

Losses				Init.	Epochs	Batch Size	Queue Size	Initial LR	Dim.	Training Set 3-Fold Cross Val. Accuracy ↑		Validation Set Accuracy ↑	
$\mathcal{L}_{\mathsf{contrastive}}$	$\mathcal{L}_{\mathsf{align}}$	$\mathcal{L}_{\mathsf{uniform}}$								top1	top5	top1	top5
		Form	Intra-batch										
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.01)$	—	—		`rand`	240	128	16384	0.03	128	62.45%	85.64%	64.14%	86.12%
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.07)$	—	—		`rand`	240	128	16384	0.03	128	71.68%	91.00%	72.80%	91.64%
$\mathcal{L}_{\mathsf{c}}(\tau{=}0.5)$	—	—		`rand`	240	128	16384	0.03	128	68.56%	91.21%	69.98%	91.80%
$\mathcal{L}_{\mathsf{c}}(\tau{=}1)$	—	—		`rand`	240	128	16384	0.03	128	62.19%	87.73%	64.06%	88.32%
$\mathcal{L}_{\mathsf{c}}(\tau{=}2)$	—	—		`rand`	240	128	16384	0.03	128	53.62%	83.03%	55.46%	84.18%
$\mathcal{L}_{\mathsf{c}}(\tau{=}5)$	—	—		`rand`	240	128	16384	0.03	128	37.52%	68.93%	39.00%	70.86%
—	$2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	—		`rand`	240	128	16384	0.03	128	1.03%	5.12%	1.22%	5.42%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.125 \cdot \mathcal{L}_{\mathsf{u}}(t{=}8)$	✓	`rand`	240	128	16384	0.03	128	65.89%	88.28%	67.42%	88.96%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.15 \cdot \mathcal{L}_{\mathsf{u}}(t{=}7)$	✓	`rand`	240	128	16384	0.03	128	67.51%	88.95%	68.90%	89.68%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.17 \cdot \mathcal{L}_{\mathsf{u}}(t{=}6)$	✓	`rand`	240	128	16384	0.03	128	67.90%	89.83%	69.18%	90.76%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.2 \cdot \mathcal{L}_{\mathsf{u}}(t{=}5)$	✓	`rand`	240	128	16384	0.03	128	69.27%	90.08%	70.46%	90.86%
—	$1.8 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.2 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	240	128	16384	0.03	128	1.00%	4.94%	1.00%	5.00%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.25 \cdot \mathcal{L}_{\mathsf{u}}(t{=}4)$	✓	`rand`	240	128	16384	0.03	128	69.77%	90.57%	70.70%	91.14%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.33 \cdot \mathcal{L}_{\mathsf{u}}(t{=}3)$	✓	`rand`	240	128	16384	0.03	128	70.67%	91.14%	71.86%	91.58%
—	$1.6 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.4 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	240	128	16384	0.03	128	67.34%	90.27%	69.16%	91.00%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✗	`rand`	240	128	16384	0.03	128	70.91%	91.38%	72.34%	91.86%

—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.5 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	240	128	16384	0.03	128	71.03%	91.61%	71.90%	92.06%
—	$1.4 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.6 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	240	128	16384	0.03	128	71.11%	91.69%	72.06%	92.28%
—	$1.2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.8 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	240	128	16384	0.03	128	71.76%	91.51%	72.78%	91.90%
—	$0.75 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	240	128	16384	0.03	128	70.23%	91.01%	71.40%	91.36%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}1)$	✓	`rand`	240	128	16384	0.03	128	68.07%	90.66%	69.54%	91.14%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	✗	`rand`	240	128	16384	0.03	128	69.59%	90.67%	70.64%	91.28%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	240	128	16384	0.03	128	70.45%	91.25%	71.48%	91.72%
—	$1.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	240	128	16384	0.03	128	72.39%	91.71%	73.80%	92.22%
—	$2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	✗	`rand`	240	128	16384	0.03	128	72.19%	**92.35%**	73.30%	92.74%
—	$2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	✗	`rand`	240	128	32768	0.03	128	72.41%	92.08%	73.54%	92.74%
—	$2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	240	128	16384	0.03	128	72.69%	92.21%	73.74%	**92.80%**
—	$2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	240	128	32768	0.03	128	72.65%	92.09%	73.68%	92.46%
—	$2.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	✗	`rand`	240	128	16384	0.03	128	71.77%	91.99%	73.00%	92.14%
—	$2.5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	240	128	16384	0.03	128	72.31%	91.99%	73.50%	92.38%
—	$3 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	240	128	16384	0.03	128	72.03%	92.09%	73.48%	92.56%
—	$3 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}3)$	✓	`rand`	240	128	16384	0.03	128	**73.49%**	92.24%	**74.60%**	92.74%
—	$4 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}4)$	✓	`rand`	240	128	16384	0.03	128	72.93%	92.03%	74.30%	92.54%
—	$5 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}5)$	✓	`rand`	240	128	16384	0.03	128	71.96%	91.67%	73.04%	92.28%
—	$6 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}6)$	✓	`rand`	240	128	16384	0.03	128	70.49%	90.63%	72.02%	91.24%
—	$7 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}7)$	✓	`rand`	240	128	16384	0.03	128	70.66%	90.83%	72.32%	91.86%
—	$8 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}8)$	✓	`rand`	240	128	16384	0.03	128	69.47%	90.33%	70.86%	91.26%
—	$0.8 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$1.2 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	240	128	16384	0.03	128	70.45%	90.72%	71.22%	91.06%
—	$0.6 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$1.4 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	240	128	16384	0.03	128	69.03%	90.53%	70.44%	90.92%
—	$0.4 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$1.6 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	240	128	16384	0.03	128	67.04%	89.24%	68.32%	89.76%
—	$0.2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$1.8 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	240	128	16384	0.03	128	66.71%	88.93%	68.10%	89.48%
—	—	$2 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	240	128	16384	0.03	128	2.43%	9.97%	2.92%	10.56%

—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	240	128	16384	0.03	128	58.43%	84.67%	60.36%	85.02%
—	$2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	✗	`rand`	240	128	32768	0.03	128	69.68%	91.13%	70.80%	91.80%
—	$2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	240	128	16384	0.03	128	69.62%	90.77%	70.92%	91.42%

Table A.6: Experiment specifications for all 108 BookCorpus recurrent encoders trained using methods based on Quick-Thought Vectors [106]. We report the encoder representation quality measured by accuracy of logistic classifiers on encoder outputs for the Movie Review Sentence Polarity (MR) and Customer Product Sentiment (CR) binary classification tasks, via both a 5-fold cross validation of the training set (of the downstream task) and the held out validation set (of the downstream task).
All encoders in this table use standard network initialization (denoted as "`rand`"). Dimensionality (abbreviated as "Dim.") shows the ambient dimension of the output features, i.e., features from $l2$-normalized encoders live on the unit hypersphere of one less dimension. Regardless of whether the encoder is $l2$-normalized (indicated in "Normalization" column), the features are always normalized before being used for downstream tasks, following Logeswaran and Lee [106].
The only unnormalized encoder is obtained using the unmodified Quick-Thought Vectors algorithm. 6 configurations that suffer from training instability (i.e., `NaN` occurring) are also reported.

Losses			Normalization	Init.	Epochs	Batch Size	Initial LR	Dim.	Training Set 5-Fold Cross Val. Accuracy ↑		Validation Set Accuracy ↑	
$\mathcal{L}_{\mathsf{contrastive}}$	$\mathcal{L}_{\mathsf{align}}$	$\mathcal{L}_{\mathsf{uniform}}$							MR	CR	MR	CR
$\mathcal{L}_{\mathsf{c}}(\tau=1)$	—	—	✗	rand	1	400	0.0005	1200	76.33%	81.90%	77.23%	83.07%
$\mathcal{L}_{\mathsf{c}}(\tau=0.005)$	—	—	✓	rand	1	400	0.0005	1200	74.97%	82.94%	76.85%	82.54%
$\mathcal{L}_{\mathsf{c}}(\tau=0.01)$	—	—	✓	rand	1	400	0.0005	1200	75.02%	82.20%	75.54%	82.28%
$\mathcal{L}_{\mathsf{c}}(\tau=0.05)$	—	—	✓	rand	1	400	0.0005	1200	75.48%	**83.64%**	**77.69%**	83.86%
$\mathcal{L}_{\mathsf{c}}(\tau=0.075)$	—	—	✓	rand	1	400	0.0005	1200	**76.37%**	83.32%	77.51%	82.28%
$\mathcal{L}_{\mathsf{c}}(\tau=0.1)$	—	—	✓	rand	1	400	0.0005	1200	75.82%	81.90%	74.79%	83.86%
$\mathcal{L}_{\mathsf{c}}(\tau=0.2)$	—	—	✓	rand	1	400	0.0005	1200	74.33%	81.08%	75.63%	80.16%
$\mathcal{L}_{\mathsf{c}}(\tau=0.25)$	—	—	✓	rand	1	400	0.0005	1200	72.33%	79.49%	71.51%	78.84%
$\mathcal{L}_{\mathsf{c}}(\tau=0.3)$	—	—	✓	rand	1	400	0.0005	1200	72.85%	78.54%	73.57%	79.10%
$\mathcal{L}_{\mathsf{c}}(\tau=0.4)$	—	—	✓	rand	1	400	0.0005	1200	69.72%	77.28%	67.85%	77.51%
$\mathcal{L}_{\mathsf{c}}(\tau=0.5)$	—	—	✓	rand	1	400	0.0005	1200	68.97%	76.27%	68.98%	74.07%
$\mathcal{L}_{\mathsf{c}}(\tau=0.6)$	—	—	✓	rand	1	400	0.0005	1200	68.61%	75.48%	68.88%	73.81%
$\mathcal{L}_{\mathsf{c}}(\tau=0.7)$	—	—	✓	rand	1	400	0.0005	1200	67.89%	74.01%	67.76%	76.46%
$\mathcal{L}_{\mathsf{c}}(\tau=0.8)$	—	—	✓	rand	1	400	0.0005	1200	67.02%	74.77%	66.07%	74.34%
$\mathcal{L}_{\mathsf{c}}(\tau=0.9)$	—	—	✓	rand	1	400	0.0005	1200	66.78%	74.01%	65.32%	72.75%

$\mathcal{L}_\mathsf{c}(\tau{=}1)$	—	—	✓	`rand`	1	400	0.0005	1200	66.67%	74.12%	65.79%	74.34%
$\mathcal{L}_\mathsf{c}(\tau{=}1.5)$	—	—	✓	`rand`	1	400	0.0005	1200	63.92%	70.47%	65.42%	75.93%
$\mathcal{L}_\mathsf{c}(\tau{=}2)$	—	—	✓	`rand`	1	400	0.0005	1200	63.97%	72.06%	62.79%	71.69%
$\mathcal{L}_\mathsf{c}(\tau{=}5)$	—	—	✓	`rand`	1	400	0.0005	1200	62.21%	69.50%	62.98%	73.54%
$\mathcal{L}_\mathsf{c}(\tau{=}0.075)$	$\mathcal{L}_\mathsf{a}(\alpha{=}2)$	—	✓	`rand`	1	400	0.0005	1200	69.16%	73.39%	68.13%	72.75%
$\mathcal{L}_\mathsf{c}(\tau{=}1)$	$\mathcal{L}_\mathsf{a}(\alpha{=}2)$	—	✓	`rand`	1	400	0.0005	1200	49.68%	63.81%	49.77%	63.49%
$\mathcal{L}_\mathsf{c}(\tau{=}0.075)$	$0.9 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.1 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	71.26%	77.90%	71.42%	76.72%
$\mathcal{L}_\mathsf{c}(\tau{=}1)$	$0.9 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.1 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	51.26%	63.78%	52.01%	63.49%
$\mathcal{L}_\mathsf{c}(\tau{=}0.075)$	$0.8 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.2 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	76.25%	83.05%	76.48%	83.33%
$\mathcal{L}_\mathsf{c}(\tau{=}1)$	$0.8 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.2 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	71.33%	79.31%	70.48%	78.31%
$\mathcal{L}_\mathsf{c}(\tau{=}0.075)$	$0.7 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.3 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	75.67%	81.20%	74.60%	81.48%
$\mathcal{L}_\mathsf{c}(\tau{=}1)$	$0.7 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.3 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	71.59%	78.72%	73.66%	78.84%
$\mathcal{L}_\mathsf{c}(\tau{=}0.075)$	$0.6 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.4 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	75.06%	82.23%	74.41%	81.48%
$\mathcal{L}_\mathsf{c}(\tau{=}1)$	$0.6 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.4 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	70.53%	78.43%	68.88%	75.93%
$\mathcal{L}_\mathsf{c}(\tau{=}0.075)$	$0.5 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.5 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	74.45%	81.61%	74.51%	**84.66%**
$\mathcal{L}_\mathsf{c}(\tau{=}1)$	$0.5 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.5 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	66.06%	72.97%	63.64%	73.02%
$\mathcal{L}_\mathsf{c}(\tau{=}0.075)$	$0.4 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.6 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	73.23%	80.61%	74.32%	82.54%
$\mathcal{L}_\mathsf{c}(\tau{=}1)$	$0.4 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.6 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	57.75%	67.55%	57.92%	69.84%
$\mathcal{L}_\mathsf{c}(\tau{=}0.075)$	$0.3 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.7 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	72.99%	79.46%	74.88%	77.25%
$\mathcal{L}_\mathsf{c}(\tau{=}1)$	$0.3 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.7 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	56.96%	64.31%	55.30%	65.34%
$\mathcal{L}_\mathsf{c}(\tau{=}0.075)$	$0.2 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.8 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	71.94%	79.43%	70.95%	78.04%
$\mathcal{L}_\mathsf{c}(\tau{=}1)$	$0.2 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.8 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	54.90%	64.22%	55.11%	63.76%
$\mathcal{L}_\mathsf{c}(\tau{=}0.075)$	$0.1 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.9 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	70.53%	78.25%	69.82%	78.57%
$\mathcal{L}_\mathsf{c}(\tau{=}1)$	$0.1 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.9 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	55.56%	64.90%	53.98%	65.08%
$\mathcal{L}_\mathsf{c}(\tau{=}0.075)$	—	$\mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	70.13%	77.66%	70.67%	77.25%
$\mathcal{L}_\mathsf{c}(\tau{=}1)$	—	$\mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	54.76%	63.45%	53.98%	64.81%

—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	—	✓	`rand`	1	400	0.0005	1200	49.85%	63.81%	50.05%	63.49%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	—	✓	`rand`	1	400	0.0005	1200	50.02%	63.81%	49.30%	63.49%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	—	✓	`rand`	1	400	0.0005	1200	50.04%	63.81%	49.95%	63.49%
—	$0.9091 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.0909 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	49.67%	63.81%	49.86%	63.49%
—	$0.9 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.1 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	49.71%	63.81%	49.77%	63.49%
—	$0.9 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.1 \cdot \mathcal{L}_{\mathsf{u}}(t{=}5)$	✓	`rand`	1	400	0.0005	1200	73.42%	81.23%	73.76%	80.95%
—	$0.9 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.1 \cdot \mathcal{L}_{\mathsf{u}}(t{=}7)$	✓	`rand`	1	400	0.0005	1200	70.59%	78.57%	71.60%	77.51%
—	$0.8889 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.1111 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	50.14%	63.81%	49.86%	63.49%
—	$0.875 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.125 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	50.33%	63.98%	49.86%	63.49%
—	$0.875 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.125 \cdot \mathcal{L}_{\mathsf{u}}(t{=}7)$	✓	`rand`	1	400	0.0005	1200	64.70%	72.71%	64.10%	71.69%
—	$0.8571 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.1429 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	59.80%	66.52%	59.51%	67.72%
—	$0.8333 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$0.1667 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	68.42%	76.07%	68.60%	75.13%
—	$0.8333 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.1667 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	66.69%	73.09%	67.95%	71.69%
—	$0.833 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.167 \cdot \mathcal{L}_{\mathsf{u}}(t{=}5)$	✓	`rand`	1	400	0.0005	1200	54.35%	64.49%	56.33%	63.49%
—	$0.8298 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$0.1702 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	67.38%	74.68%	67.29%	73.81%
—	$0.8298 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.1702 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	66.24%	73.33%	64.76%	77.25%
—	$0.8261 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$0.1739 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	65.91%	75.27%	66.82%	74.07%
—	$0.8261 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.1739 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	67.65%	73.56%	67.95%	72.49%
—	$0.8222 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$0.1778 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	66.73%	75.13%	67.85%	73.54%
—	$0.8222 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.1778 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	69.33%	73.42%	69.54%	74.60%
—	$0.8182 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$0.1818 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	66.17%	74.36%	65.70%	74.34%
—	$0.8182 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.1818 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	69.61%	75.51%	70.10%	75.40%
—	$0.814 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$0.186 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	63.43%	72.74%	63.82%	73.28%
—	$0.814 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.186 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	71.32%	77.72%	70.85%	77.25%
—	$0.8095 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$0.1905 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	63.47%	72.33%	63.82%	73.28%
—	$0.8095 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.1905 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	71.33%	77.19%	71.13%	75.40%

—	$0.8049 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}1)$	$0.1951 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	61.17%	70.79%	61.01%	73.54%
—	$0.8049 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.1951 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	72.04%	77.93%	73.38%	77.51%
—	$0.8 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}1)$	$0.2 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	60.91%	69.41%	59.14%	70.37%
—	$0.8 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.2 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	72.60%	80.34%	73.48%	79.89%
—	$0.8 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.2 \cdot \mathcal{L}_\mathsf{u}(t{=}5)$	✓	`rand`	1	400	0.0005	1200	54.82%	63.19%	51.64%	64.02%
—	$0.8 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.2 \cdot \mathcal{L}_\mathsf{u}(t{=}7)$	✓	`rand`	1	400	0.0005	1200	53.67%	63.90%	57.92%	65.61%
—	$0.75 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}1)$	$0.25 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	55.29%	63.63%	55.11%	70.11%
—	$0.75 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.25 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	72.60%	80.72%	71.88%	79.63%
—	$0.7 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}1)$	$0.3 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	54.24%	63.87%	55.01%	68.52%
—	$0.7 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.3 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	71.80%	78.93%	73.76%	77.78%
—	$0.7 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.3 \cdot \mathcal{L}_\mathsf{u}(t{=}5)$	✓	`rand`	1	400	0.0005	1200	55.34%	62.07%	53.51%	63.23%
—	$0.7 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.3 \cdot \mathcal{L}_\mathsf{u}(t{=}7)$	✓	`rand`	1	400	0.0005	1200	54.22%	64.28%	55.20%	60.85%
—	$0.6667 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}1)$	$0.3333 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	55.42%	63.25%	54.83%	68.78%
—	$0.6667 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.3333 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	68.49%	76.48%	67.20%	74.60%
—	$0.6 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}1)$	$0.4 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	54.86%	63.63%	55.30%	67.46%
—	$0.6 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.4 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	60.60%	69.35%	61.29%	68.25%
—	$0.6 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.4 \cdot \mathcal{L}_\mathsf{u}(t{=}5)$	✓	`rand`	1	400	0.0005	1200	54.64%	63.96%	56.61%	62.43%
—	$0.6 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.4 \cdot \mathcal{L}_\mathsf{u}(t{=}7)$	✓	`rand`	1	400	0.0005	1200	55.28%	63.63%	55.20%	63.76%
—	$0.5 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}1)$	$0.5 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	53.61%	64.40%	52.86%	66.14%
—	$0.5 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.5 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	55.42%	64.75%	55.76%	66.40%
—	$0.5 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.5 \cdot \mathcal{L}_\mathsf{u}(t{=}5)$	✓	`rand`	1	400	0.0005	1200	55.49%	63.16%	55.39%	64.29%
—	$0.5 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.5 \cdot \mathcal{L}_\mathsf{u}(t{=}7)$	✓	`rand`	1	400	0.0005	1200	56.06%	63.90%	57.73%	64.81%
—	$0.4 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}1)$	$0.6 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	54.27%	64.37%	54.45%	63.49%
—	$0.4 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.6 \cdot \mathcal{L}_\mathsf{u}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	55.22%	63.69%	57.73%	67.72%
—	$0.4 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.6 \cdot \mathcal{L}_\mathsf{u}(t{=}5)$	✓	`rand`	1	400	0.0005	1200	53.26%	63.57%	53.70%	65.87%
—	$0.4 \cdot \mathcal{L}_\mathsf{a}(\alpha{=}2)$	$0.6 \cdot \mathcal{L}_\mathsf{u}(t{=}7)$	✓	`rand`	1	400	0.0005	1200	54.53%	63.66%	53.14%	64.55%

—	$0.3 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$0.7 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	54.75%	63.43%	53.42%	64.02%
—	$0.3 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.7 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	53.64%	63.84%	54.64%	62.70%
—	$0.3 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.7 \cdot \mathcal{L}_{\mathsf{u}}(t{=}5)$	✓	`rand`	1	400	0.0005	1200	55.13%	63.81%	55.39%	64.81%
—	$0.3 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.7 \cdot \mathcal{L}_{\mathsf{u}}(t{=}7)$	✓	`rand`	1	400	0.0005	1200	56.56%	63.87%	56.04%	66.67%
—	$0.2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$0.8 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	53.86%	64.04%	54.83%	69.31%
—	$0.2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.8 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	53.73%	65.34%	53.98%	64.55%
—	$0.2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.8 \cdot \mathcal{L}_{\mathsf{u}}(t{=}5)$	✓	`rand`	1	400	0.0005	1200	54.76%	64.37%	55.76%	65.87%
—	$0.2 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.8 \cdot \mathcal{L}_{\mathsf{u}}(t{=}7)$	✓	`rand`	1	400	0.0005	1200	54.86%	63.51%	53.89%	66.40%
—	$0.1 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$0.9 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	54.60%	65.72%	56.42%	68.52%
—	$0.1 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.9 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	54.60%	64.90%	57.26%	60.85%
—	$0.1 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.9 \cdot \mathcal{L}_{\mathsf{u}}(t{=}5)$	✓	`rand`	1	400	0.0005	1200	56.23%	63.66%	55.48%	66.14%
—	$0.1 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}2)$	$0.9 \cdot \mathcal{L}_{\mathsf{u}}(t{=}7)$	✓	`rand`	1	400	0.0005	1200	54.65%	65.22%	55.95%	64.02%
—	—	$\mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	55.02%	62.69%	57.36%	67.72%
—	—	$\mathcal{L}_{\mathsf{u}}(t{=}5)$	✓	`rand`	1	400	0.0005	1200	54.95%	64.04%	56.04%	64.02%
—	—	$\mathcal{L}_{\mathsf{u}}(t{=}7)$	✓	`rand`	1	400	0.0005	1200	54.55%	63.48%	56.33%	63.49%
—	$\mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	—	✓	`rand`	1	400	0.0005	1200	`NaN` occurred			
—	$0.9091 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$0.0909 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	`NaN` occurred			
—	$0.9 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$0.1 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	`NaN` occurred			
—	$0.8889 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$0.1111 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	`NaN` occurred			
—	$0.875 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$0.125 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	`NaN` occurred			
—	$0.8571 \cdot \mathcal{L}_{\mathsf{a}}(\alpha{=}1)$	$0.1429 \cdot \mathcal{L}_{\mathsf{u}}(t{=}2)$	✓	`rand`	1	400	0.0005	1200	`NaN` occurred			

Appendix B

Proofs, Details, and Additional Discussions for Chapter 3

B.1 Discussions for Section 3.2: Preliminaries on Quasimetrics and Poisson Processes

B.1.1 Quasimetric Spaces

Definition 3.2.1 (Quasimetric Space)**.** A *quasimetric space* is a pair $(\mathcal{X}, d)$, where $\mathcal{X}$ is a set of points and $d\colon \mathcal{X} \times \mathcal{X} \to [0, \infty]$ is the quasimetric, satisfying the following conditions:

$$\forall x, y \in \mathcal{X}, \qquad x = y \iff d(x, y) = 0, \quad \text{(Identity of Indiscernibles)}$$
$$\forall x, y, z \in \mathcal{X}, \qquad d(x, y) + d(y, z) \geq d(x, z). \qquad \text{(Triangle Inequality)}$$

Definition B.1.1 (Quasipseudometric Space)**.** As a further generalization, we say $(\mathcal{X}, d)$ is a *quasipseudometric space* if the *Identity of Indiscernibles* requirement is only satisfied in one direction:

$$\forall x, y \in \mathcal{X}, \qquad x = y \implies d(x, y) = 0, \quad \text{(Identity of Indiscernibles)}$$
$$\forall x, y, z \in \mathcal{X}, \qquad d(x, y) + d(y, z) \geq d(x, z). \qquad \text{(Triangle Inequality)}$$

Examples of Quasimetric Spaces

Proposition B.1.2 (Expected Hitting Time of a Markov Chain). Let random variables $(X_t)_t$ be a Markov Chain with support $\mathcal{X}$. Then $(\mathcal{X}, d_{\mathsf{hitting}})$ is a quasimetric space, where

$$d_{\mathsf{hitting}}(s, t) \triangleq \mathbb{E}\left[\text{time to hit } t \mid \text{start from } s\right], \tag{B.1}$$

where we define the hitting time of s starting from s to be 0.

Proof of Proposition B.1.2. Obviously d_{hitting} is non-negative. We then verify the following quasimetric space properties:

- **Identity of Indiscernibles.** By definition, we have, $\forall x, y \in \mathcal{X}$, $x \neq y$,

$$d_{\mathsf{hitting}}(x, x) = 0 \tag{B.2}$$
$$d_{\mathsf{hitting}}(x, y) \geq 1. \tag{B.3}$$

- **Triangle Inequality.** For any $x, y, z \in \mathcal{X}$, we have

$$\begin{aligned} d_{\mathsf{hitting}}(x, y) + d_{\mathsf{hitting}}(y, z) &= \mathbb{E}\left[\text{time to hit } y \text{ then hit } z \mid \text{start from } x\right] && \text{(B.4)} \\ &\geq \mathbb{E}\left[\text{time to hit } z \mid \text{start from } x\right] && \text{(B.5)} \\ &= d_{\mathsf{hitting}}(x, z). && \text{(B.6)} \end{aligned}$$

Hence, $(\mathcal{X}, d_{\mathsf{hitting}})$ is a quasimetric space. □

Proposition B.1.3 (Conditional Shannon Entropy). Let $\mathcal{X}$ be the set of random variables (of some probability space). Then $(\mathcal{X}, d_H)$ is a quasipseudometric space, where

$$d_H(X, Y) \triangleq H(Y \mid X). \tag{B.7}$$

If for all distinct $(X, Y) \in \mathcal{X} \times \mathcal{X}$, X can not be written as (almost surely) a deterministic function of Y, then $(\mathcal{X}, d_H)$ is a quasimetric space.

Proof of Proposition B.1.3. Obviously d_H is non-negative. We then verify the following quasipseudometric space properties:

- **Identity of Indiscernibles.** By definition, we have, $\forall X, Y \in \mathcal{X}$,

$$d_H(X, X) = H(X \mid X) = 0 \tag{B.8}$$
$$d_H(Y, X) = H(Y \mid X) \geq 0, \tag{B.9}$$

  where $\leq$ is $=$ iff Y is a (almost surely) deterministic function of X.

- **Triangle Inequality.** For any $X, Y, Z \in \mathcal{X}$, we have

$$\begin{aligned} d_H(X, Y) + d_H(Y, Z) &= H(Y \mid X) + H(Z \mid Y) && \text{(B.10)} \\ &\geq H(Y \mid X) + H(Z \mid XY) && \text{(B.11)} \\ &= H(YZ \mid X) && \text{(B.12)} \\ &\geq H(Z \mid X) && \text{(B.13)} \\ &= d_H(X, Z). && \text{(B.14)} \end{aligned}$$

Hence, $(\mathcal{X}, d_H)$ is a quasipseudometric space, and a quasimetric space when the last condition is satisfied. □

Conditional Kolmogorov Complexity. From algorithmic information theory, the conditional Kolmogorov complexity $K(y \mid x)$ also similarly measures "the bits needed to create y given x as input" [92]. It is also almost a quasimetric, but the exact definition affects some constant/log terms that may make the quasimetric constraints non-exact. For instance, when defined with the prefix-free version, conditional Kolmogorov complexity is always strictly positive, even for $K(x \mid x) > 0$ [102]. One may remedy this with a definition using a universal Turing machine (UTM) that simply outputs the input on empty program. But to make triangle inequality work, one needs to reason about how the input and output parts work on the tape(s) of the UTM. Nonetheless, regardless of the definition details, conditional Kolmogorov complexity do satisfy a triangle inequality up to log terms [56]. So intuitively, it behaves roughly like a quasimetric defined on the space of binary strings.

Optimal Goal-Reaching Plan Costs in Markov Decision Processes (MDPs) We define MDPs in the standard manner: $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$ [130], where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{R}\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\mathcal{P}\colon \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition function (where $\Delta(\mathcal{S})$ is the set of all distributions over $\mathcal{S}$), and $\gamma \in (0, 1)$ is the discount factor.

We define Π as the collection of all stationary policies $\pi\colon \mathcal{S} \to \Delta(\mathcal{A})$ on $\mathcal{M}$. For a particular policy $\pi \in \Pi$, it induces random *trajectories*:

- *Trajectory* starting from state $s \in \mathcal{S}$ is the random variable

$$\xi_\pi(s) = (s_1, a_1, r_1, s_2, a_2, r_2, \dots), \tag{B.15}$$

distributed as

$$\begin{aligned} s_1 &= s && \text{(B.16)} \\ a_i &\sim \pi(s_i), \qquad \forall i \geq 1 && \text{(B.17)} \\ s_{i+1} &\sim \mathcal{P}(s_i, a_i), \qquad \forall i \geq 1. && \text{(B.18)} \end{aligned}$$

- *Trajectory* starting from state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ is the random variable

$$\xi_\pi(s, a) = (s_1, a_1, r_1, s_2, a_2, r_2, \dots), \tag{B.19}$$

distributed as

$$s_1 = s \tag{B.20}$$

$$a_1 = a \tag{B.21}$$

$$a_i \sim \pi(s_i), \qquad \forall i \geq 2 \tag{B.22}$$

$$s_{i+1} \sim \mathcal{P}(s_i, a_i), \qquad \forall i \geq 1. \tag{B.23}$$

Proposition B.1.4 (Optimal Goal-Reaching Plan Costs in MDPs). Consider an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$. WLOG, assume that $\mathcal{R} \colon \mathcal{S} \times \mathcal{A} \to (-\infty, 0]$ has only non-positive rewards (i.e., negated costs). Let $\mathcal{X} = \mathcal{S} \cup (\mathcal{S} \times \mathcal{A})$. Then $(\mathcal{X}, d_{\mathsf{sum}})$ and $(\mathcal{X}, d_\gamma)$ are quasipseudometric spaces, where

$$\begin{aligned} & d_{\mathsf{sum}}(x, y) \\ & \triangleq \min_{\pi \in \Pi} \mathbb{E}\left[\text{total costs from } x \text{ to } y \text{ under } \pi\right] \qquad \text{(B.24)} \\ & = \begin{cases} \min_{\pi \in \Pi} \mathbb{E}_{(s_1, a_1, r_1, \dots) = \xi_\pi(x)} \left[-\sum_t r_t \underbrace{\mathbf{1}_{s' \notin \{s_i\}_{i \in [t]}}}_{\text{not reached } s' \text{ yet}} \right] & \text{if } \underbrace{y = s' \in \mathcal{S}}_{\text{goal is a state}}, \\ \min_{\pi \in \Pi} \mathbb{E}_{(s_1, a_1, r_1, \dots) = \xi_\pi(x)} \left[-\sum_t r_t \underbrace{\mathbf{1}_{(s', a') \notin \{(s_i, a_i)\}_{i \in [t-1]}}}_{\text{not reached } s' \text{ \underline{and} performed } a' \text{ yet}} \right] & \text{if } \underbrace{y = (s', a') \in \mathcal{S} \times \mathcal{A}}_{\text{goal is a state-action pair}}, \end{cases} \end{aligned}$$

$$\text{(B.25)}$$

and

$$d_\gamma(x, y) \triangleq \log_\gamma \max_{\pi \in \Pi} \mathbb{E}\left[\gamma^{\text{total costs from } x \text{ to } y \text{ under } \pi}\right] \tag{B.26}$$

is defined similarly.

If the reward function is always *negative*, $(\mathcal{X}, d_{\mathsf{sum}})$ and $(\mathcal{X}, d_\gamma)$ are *quasimetric* spaces.

Proof of Proposition B.1.4. Obviously both d_{sum} and d_γ are non-negative, and satisfy *Identity of Indiscernibles* (for quasipseudometric spaces). For triangle inequality, note that for each y, we can instead consider alternative MDPs:

- If $y = s' \in \mathcal{S}$, modify the original MDP to make s' a sink state, where performing any action yields 0 reward (i.e., 0 cost);

- If $y = (s', a') \in \mathcal{S} \times \mathcal{A}$, modify the original MDP such that performing action a' in state s' surely transitions to a new sink state, where performing any action yields 0 reward (i.e., 0 cost).

Obviously, both are Markovian. Furthermore, they are Stochastic Shortest Path problems with no negative costs [58], implying that there are Markovian (i.e., stationary) optimal policies (respectively w.r.t. either minimizing expected total cost or maximizing expected $\gamma^{\text{total cost}}$). Thus optimizing over the set of stationary policies, Π, gives the optimal quantity over all possible policies, including concatenation of two stationary policies. Thus the triangle inequality is satisfied by both.

Hence, $(\mathcal{X}, d_{\mathsf{sum}})$ and $(\mathcal{X}, d_\gamma)$ are quasipseudometric spaces.

Finally, if the reward function is always *negative*, $x \neq y \implies d_{\mathsf{sum}}(x, y) > 0$ and $d_\gamma(x, y) > 0$, so $(\mathcal{X}, d_{\mathsf{sum}})$ and $(\mathcal{X}, d_\gamma)$ are quasimetric spaces. $\square$

Remark B.1.5. We make a couple remarks:

- Any MDP with a bounded reward function can be modified to have only non-positive rewards by subtracting the maximum reward (or larger);

- We have
$$d_{\mathsf{sum}}(s, (s, a)) = d_\gamma(s, (s, a)) = -\mathcal{R}(s, a). \tag{B.27}$$

- When the dynamics is deterministic, $d_{\mathsf{sum}} \equiv d_\gamma$, $\forall \gamma \in (0, 1)$.

- Unless y is reachable from x with probability 1 under some policy, $d_{\mathsf{sum}}(x, y) = \infty$.

- Unless y is *unreachable* from x with probability 1 under *all* policies, $d_{\mathsf{sum}}(x, y) < \infty$. Therefore, it is often favorable to consider d_γ types.

- In certain MDP formulations, the reward is stochastic and/or dependent on the reached next state. The above definitions readily extend to those cases.

- $\gamma^{d_\gamma((s,a),y)}$ is very similar to Q-functions except that Q-function applies discount based on time, and $\gamma^{d_\gamma((s,a),y)}$ applies discount based on costs. We note that a Q-learning-like recurrence can also be found for $\gamma^{d_\gamma((s,a),y)}$.

  If the cost is constant in the sense for some fixed $c < 0$, $\mathcal{R}(s,a) = c$, $\forall (s,a) \in \mathcal{S} \times \mathcal{A}$, then time and cost are equivalent up to a scale. Therefore, $\gamma^{d_\gamma((s,a),y)}$ coincides with the optimal Q-functions for the MDPs described in proof, and $\gamma^{d_\gamma(s,y)}$ coincides with the optimal value functions for the respective MDPs.

Quasimetric Treewidth and Graph Treewidth

Definition 3.2.2 (Treewidth of Quasimetric Spaces [113])**.** Consider representations of a quasimetric space M as shortest-path distances on a positively-weighted directed graph. *Treewidth* of M is the minimum over all such graphs' treewidths. (Recall that the treewidth of a graph (after replacing directed edges with undirected ones) is a measure of its complexity.)

Graph treewidth is a standard complexity measure of how "similar" a graph is to a tree [133]. Informally speaking, if a graph has low treewidth, we can represent it as a tree, preserving all connected paths between vertices, except that in each tree node, we store a small number of vertices (from the original graph) rather than just 1.

Graph treewidth is widely used by the Theoretical Computer Science and Graph Theory communities, since many NP problems are solvable in polynomial time for graphs with bounded treewidth [11].

B.1.2 Poisson Processes

Definition 3.2.3 (Poisson Process)**.** For nonatomic measure μ on set A, a *Poisson process* on A with *mean measure* μ is a random countable subset $P \subset A$ (i.e., the random events / points) such that

- for any disjoint measurable subsets $A_1, \dots, A_n$ of A, the random variables $N(A_1), \dots, N(A_n)$ are independent, where $N(B) \triangleq \#\{P \cap B\}$ is the number of points of P in B, and

- $N(B)$ has the Poisson distribution with mean $\mu(B)$, denoted as $\text{Pois}(\mu(B))$.

Poisson processes are usually used to model events that randomly happens "with no clear pattern", e.g., visible stars in a patch of the sky, arrival times of Internet packages to a data center. These events may randomly happen all over the sky / time. To an extent, we can say that their characteristic feature is a property of statistical independence [89].

To understand this, imagine raindrops hitting the windshield of a car. Suppose that we already know that the rain is heavy, knowing the exact pattern of the raindrops hitting on the left side of the windshield tells you little about the hitting pattern on the right side. Then, we may assume that, as long as we look at regions that are disjoint on the windshield, the number of raindrops in each region are independent.

This is the fundamental motivation of Poisson processes. In a sense, from this characterization, Poisson processes are inevitable (see Sec. 1.4 of [89]).

Poisson Race Probability $\mathbb{P}[\text{Pois}(\mu_1) \leq \text{Pois}(\mu_2)]$ and Its Gradient Formulas

In Fact 3.2.4 we made several remarks on the Poisson race probability, i.e., for *independent* $X \sim \text{Pois}(\mu_1)$, $Y \sim \text{Pois}(\mu_2)$, the quantity $\mathbb{P}[X \leq Y]$. In this section, we detailedly describe how we arrived at those conclusions, and provide the exact gradient formulas for differentiating $\mathbb{P}[X \leq Y]$ w.r.t. μ_1 and μ_2.

From Skellam distribution CDF to Non-Central χ^2 distribution CDF. Distribution of the difference of two independent Poisson random variables is called the *Skellam* distribution [139], with its parameter being the rate of the two Poissons. That is, $X - Y \sim \text{Skellam}(\mu_1, \mu_2)$. Therefore, $\mathbb{P}[X \leq Y]$ is essentially the cumulative distribution function (CDF) of this Skellam at 0. In Eq. (4) of [84], a connection is made between the CDF of $\text{Skellam}(\mu_1, \mu_2)$ distribution, and the CDF of a non-central χ^2 distribution (which is a non-centered generalization of χ^2 distribution) with two parameters $k > 0$ degree(s) of freedom and non-centrality parameter $\lambda \geq 0$): for integer $n > 0$,

$$\mathbb{P}\left[\text{Skellam}(\mu_1, \mu_2) \geq n\right] = \mathbb{P}\big[\text{NonCentral}\chi^2(\underbrace{2n}_{\text{degree(s) of freedom}}, \underbrace{2\mu_2}_{\text{non-centrality parameter}}) < 2\mu_1\big], \tag{B.28}$$

which can be evaluated using statistical computing packages such as `SciPy` [163] and `CDFLIB` [21, 19].

Marcum-Q-Function and gradient formulas. To differentiate through Equation (B.28), we consider representing the non-central χ^2 CDF as a Marcum-Q-function [111]. One definition of the Marcum-Q-function $Q_M \colon \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ in statistics is

$$Q_M(a, b) \triangleq \int_b^\infty x \left(\frac{x}{a}\right)^{M-1} \exp\left(-\frac{x^2 + a^2}{2}\right) I_{M-1}(ax)\,\mathrm{d}x, \tag{B.29}$$

where I_{M-1} is the modified Bessel function of order $M - 1$. (When M is non-integer, we refer readers to [20, 111] for definitions, which are not relevant to the discussion below.) When used in CDF of non-central χ^2, we have

$$\mathbb{P}\left[\text{NonCentral}\chi^2(k, \lambda) < x\right] = 1 - Q_{\frac{k}{2}}(\sqrt{\lambda}, \sqrt{x}). \tag{B.30}$$

Combining with Equation (B.28), and using the symmetry $\text{Skellam}(\mu_1, \mu_2) \overset{d}{=} -\text{Skellam}(\mu_2, \mu_1)$, we have, for integer n,

$$\begin{aligned}
\mathbb{P}\left[X \leq Y + n\right] &= \mathbb{P}\left[\text{Skellam}(\mu_1, \mu_2) \leq n\right] && \text{(B.31)}\\
&= \begin{cases} \mathbb{P}\left[\text{NonCentral}\chi^2(-2n, 2\mu_1) < 2\mu_2\right] & \text{if } n < 0 \\ 1 - \mathbb{P}\left[\text{NonCentral}\chi^2(2(n+1), 2\mu_2) < 2\mu_1\right] & \text{if } n \geq 0 \end{cases} && \text{(B.32)}\\
&= \begin{cases} 1 - Q_{-n}(\sqrt{2\mu_1}, \sqrt{2\mu_2}) & \text{if } n < 0 \\ Q_{n+1}(\sqrt{2\mu_2}, \sqrt{2\mu_1}) & \text{if } n \geq 0. \end{cases} && \text{(B.33)}
\end{aligned}$$

Prior work [20] provides several derivative formula for the Marcum-Q-Function:

- For $n < 0$, we have

$$\frac{\partial}{\partial \mu_1}\mathbb{P}\left[X \le Y + n\right] = \frac{\partial}{\partial \mu_1}\left(1 - Q_{-n}(\sqrt{2\mu_1}, \sqrt{2\mu_2})\right) \tag{B.34}$$

$$= Q_{-n}(\sqrt{2\mu_1}, \sqrt{2\mu_2}) - Q_{-n+1}(\sqrt{2\mu_1}, \sqrt{2\mu_2}) \qquad \text{(Eq. (16) of [20])}$$

$$= -\left(\frac{\mu_2}{\mu_1}\right)^{-\frac{n}{2}} e^{-(\mu_1+\mu_2)} I_{-n}(2\sqrt{\mu_1\mu_2}) \qquad \text{(Eq. (2) of [20])}$$

$$= -\left(\frac{\mu_2}{\mu_1}\right)^{-\frac{n}{2}} e^{-(\sqrt{\mu_1}-\sqrt{\mu_2})^2} I_{-n}^{(e)}(2\sqrt{\mu_1\mu_2}), \tag{B.35}$$

where $I_v^{(e)}(x) \triangleq e^{-|x|}I_v(x)$ is the exponentially-scaled version of I_v that computing libraries often provide due to its superior numerical precision (e.g., `SciPy` [163]),

$$\frac{\partial}{\partial \mu_2}\mathbb{P}\left[X \le Y + n\right] = \frac{\partial}{\partial \mu_2}\left(1 - Q_{-n}(\sqrt{2\mu_1}, \sqrt{2\mu_2})\right) \tag{B.36}$$

$$= \left(\frac{\mu_2}{\mu_1}\right)^{-\frac{n+1}{2}} e^{-(\mu_1+\mu_2)} I_{-n-1}(2\sqrt{\mu_1\mu_2}) \qquad \text{(Eq. (19) of [20])}$$

$$= \left(\frac{\mu_2}{\mu_1}\right)^{-\frac{n+1}{2}} e^{-(\sqrt{\mu_1}-\sqrt{\mu_2})^2} I_{-n-1}^{(e)}(2\sqrt{\mu_1\mu_2}), \tag{B.37}$$

- For $n \ge 0$, we have

$$\frac{\partial}{\partial \mu_1}\mathbb{P}\left[X \le Y + n\right] = \frac{\partial}{\partial \mu_1}Q_{n+1}(\sqrt{2\mu_2}, \sqrt{2\mu_1}) \tag{B.38}$$

$$= -\left(\frac{\mu_1}{\mu_2}\right)^{n} e^{-(\mu_1+\mu_2)} I_{n}(2\sqrt{\mu_1\mu_2}) \qquad \text{(Eq. (19) of [20])}$$

$$= -\left(\frac{\mu_1}{\mu_2}\right)^{n} e^{-(\sqrt{\mu_1}-\sqrt{\mu_2})^2} I_{n}^{(e)}(2\sqrt{\mu_1\mu_2}), \tag{B.39}$$

and,

$$\begin{aligned}
\frac{\partial}{\partial \mu_2}\mathbb{P}\left[X \leq Y + n\right] &= \frac{\partial}{\partial \mu_2} Q_{n+1}(\sqrt{2\mu_2}, \sqrt{2\mu_1}) && \text{(B.40)} \\
&= Q_{n+2}(\sqrt{2\mu_2}, \sqrt{2\mu_1}) - Q_{n+1}(\sqrt{2\mu_2}, \sqrt{2\mu_1}) \\
& && \text{(Eq. (16) of [20])} \\
&= \left(\frac{\mu_1}{\mu_2}\right)^{\frac{n+1}{2}} e^{-(\mu_1+\mu_2)} I_{n+1}(2\sqrt{\mu_1\mu_2}) && \text{(Eq. (2) of [20])} \\
&= \left(\frac{\mu_1}{\mu_2}\right)^{\frac{n+1}{2}} e^{-(\sqrt{\mu_1}-\sqrt{\mu_2})^2} I^{(e)}_{n+1}(2\sqrt{\mu_1\mu_2}). && \text{(B.41)}
\end{aligned}$$

Setting $n = 0$ gives the proper forward and backward formulas for $\mathbb{P}\left[X \leq Y\right]$.

B.2 Proofs, Discussions and Additional Results for Section 3.4: Theoretical Analysis of Various Learning Algorithms

Assumptions. Recall that we assumed a quasimetric space, which is stronger than a quasipseudometric space (Definition B.1.1), with finite distances. These are rather mild assumptions, since any quasipseudometric with infinities can always be modified to obey these assumptions by (1) adding a small metric (e.g., $d_\epsilon(x, y) \triangleq \epsilon \mathbf{1}_{x \neq y}$ with small $\epsilon > 0$) and (2) capping the infinite distances to a large value higher than any finite distance.

Worst-case analysis. In this work we focus on the *worst-case* scenario, as is common in standard (quasi)metric embedding analyses [18, 85, 81, 113]. Such results are important because embeddings are often used as heuristics in downstream tasks (e.g., planning) which are sensitive to any error. While our negative result readily extends to the average-case scenario (since the error (distortion or violation) is arbitrary), we leave a thorough average-case analysis as future work.

Data-independent bounds. We analyze possible *data-independent* bounds for various algorithms. In this sense, the positive result for PQEs (Theorem B.3.4) is really strong, showing good guarantees *regardless data quasimetric*. The negative result (Theorem 3.4.6) is also revealing, indicating that a family of algorithms should probably not be used, unless we know something more about data. *Data-independent* bounds are often of great interest in machine learning (e.g., concepts of VC-dimension [161] and PAC learning [160]). An important future work is to explore data-dependent results, possibly via defining a quasimetric complexity metric that is both friendly for machine learning analysis, and connects well with combinatorics measures such as quasimetric treewidth.

Violation and distortion metrics. The optimal violation has value 1. Specifically, it is 1 iff $\hat{d}$ is a quasimetric on $\mathcal{X}$ (assuming *non-negativity*). Distortion (over training set) and violation together quantify how well $\hat{d}$ learns a quasimetric consistent with the training data. A predictor can fit training data well (low distortion), but ignores basic quasimetric constraints on heldout data (high violation). Conversely, a predictor can perfectly obey the training data constraints (low violation), but doesn't actually fit training data well (high distortion). Indeed, (assuming *non-negativity* and *Identity of Indiscernibles*), perfect distortion (value 1) and violation (value 1) imply that $\hat{d}$ is a quasimetric consistent with training data.

Relation with classical in-distribution generalization studies. Classical generalization studies the prediction error over the underlying data distribution, and often involves complexity of the hypothesis class and/or training data [161, 112]. Our focus on quasimetric constraints violation is, in fact, not an orthogonal problem, but potentially a core part of in-distribution generalization for this setting. Here, the underlying distribution is supported on all pairs of $\mathcal{X} \times \mathcal{X}$. Indeed, if a learning algorithm has large distortion, it must attain large prediction error on $S \subset \mathcal{X} \times \mathcal{X}$; if it has large violation, it must violates the quasimetric constraints and necessarily admits bad prediction error on some pairs (whose true distances obey the quasimetric constraints). Theorem 3.4.3 (proved below) formalizes this idea, where we characterize generalization with the distortion *over all possible pairs in $\mathcal{X} \times \mathcal{X}$*.

B.2.1 Theorem 3.4.3: Distortion and Violation Lower-Bound Generalization Error

Theorem 3.4.3 (Distortion and Violation Lower-Bound Generalization Error). For non-negative $\hat{d}$, $\mathsf{dis}(\hat{d}) \geq \max(\mathsf{dis}_S(\hat{d}), \sqrt{\mathsf{vio}(\hat{d})})$, where $\mathsf{dis}(\hat{d})$ captures generalization over the entire $\mathcal{X}$ space.

Proof

Proof of Theorem 3.4.3. It is obvious that

$$\mathsf{dis}(\hat{d}) \geq \mathsf{dis}_S(\hat{d}). \tag{B.42}$$

Therefore, it remains to show that $\mathsf{dis}(\hat{d}) \geq \sqrt{\mathsf{vio}(\hat{d})}$.

WLOG, say $\mathsf{vio}(\hat{d}) > 1$. Otherwise, the statement is trivially true.

By the definition of violation (see Definition 3.4.2), we have, for some $x, y, z \in \mathcal{X}$, with $\hat{d}(x, z) > 0$,

$$\frac{\hat{d}(x, z)}{\hat{d}(x, y) + \hat{d}(y, z)} = \mathsf{vio}(\hat{d}). \tag{B.43}$$

If $\hat{d}(x, y) + \hat{d}(y, z) = 0$, then we must have one of the following two cases:

- If $d(x, y) > 0$ or $d(y, z) > 0$, the statement is true because $\mathsf{dis}(\hat{d}) = \infty$.
- If $d(x, y) = d(y, z) = 0$, then $d(x, z) = 0$ and the statement is true since $\mathsf{dis}(\hat{d}) \geq \frac{\hat{d}(x,z)}{d(x,z)} = \infty$.

It is sufficient to prove the case that $\hat{d}(x, y) + \hat{d}(y, z) > 0$. We can derive

$$\hat{d}(x, z) = \mathsf{vio}(\hat{d}) \left(\hat{d}(x, y) + \hat{d}(y, z)\right) \tag{B.44}$$

$$\geq \frac{\mathsf{vio}(\hat{d})}{\mathsf{dis}(\hat{d})} \left(d(x, y) + d(y, z)\right) \tag{B.45}$$

$$\geq \frac{\mathsf{vio}(\hat{d})}{\mathsf{dis}(\hat{d})}\, d(x, z). \tag{B.46}$$

If $d(x, z) = 0$, then $\mathsf{dis}(\hat{d}) = \infty$ and the statement is trivially true.

If $d(x, z) > 0$, above Equation (B.46) implies

$$\mathsf{dis}(\hat{d}) \geq \frac{\hat{d}(x, z)}{d(x, z)} \geq \frac{\mathsf{vio}(\hat{d})}{\mathsf{dis}(\hat{d})} \implies \mathsf{dis}(\hat{d}) \geq \sqrt{\mathsf{vio}(\hat{d})}. \tag{B.47}$$

Combining Equations (B.42) and (B.47) gives the desired statement.

□

B.2.2 Lemma 3.4.5: Examples of OrthEquiv Algorithms

Lemma 3.4.5 (Examples of OrthEquiv Algorithms). *k-nearest-neighbor with Euclidean distance, dot-product kernel ridge regression (including min-norm linear regression and MLP trained with squared loss in NTK regime) are* OrthEquiv.

Recall the definition of Equivariant Learning Transforms.

Definition 3.4.4 (Equivariant Learning Algorithms)**.** Given training set $\mathcal{D} = \{(z_i, y_i)\}_i$, where $z_i \in \mathcal{Z}$ are inputs and $y_i \in \mathcal{Y}$ are targets, a learning algorithm Alg produces a function $\mathsf{Alg}(\mathcal{D})\colon \mathcal{Z} \to Y$ such that $\mathsf{Alg}(\mathcal{D})(z')$ is the function's prediction on sample z'. Consider $\mathcal{T}$ a set of transformations $\mathcal{Z} \to \mathcal{Z}$. Alg is equivariant to $\mathcal{T}$ iff for all transform $T \in \mathcal{T}$, training set $\mathcal{D}$, $\mathsf{Alg}(\mathcal{D}) = \mathsf{Alg}(T\mathcal{D}) \circ T$, where $T\mathcal{D} = \{(Tz, y)\colon (z, y) \in \mathcal{D}\}$ is the training set with transformed inputs.

Proof

Proof of Lemma 3.4.5. We consider the three algorithms individually:

- ***k*-nearest neighbor with Euclidean distance.**

  It is evident that if a learning algorithm only depend on pairwise dot products (or distances), it is equivariant to orthogonal transforms, which preserve dot products (and distances). k-nearest-neighbor with Euclidean distance only depends on pairwise distances, which can be written in terms of dot products:

$$\|x - y\|_2^2 = x^\intercal x + y^\intercal y - 2x^\intercal y. \tag{B.48}$$

  Therefore, it is equivariant to orthogonal transforms.

- **Dot-product kernel ridge regression.**

  Since orthogonal transforms preservers dot-products, dot-product kernel ridge regression is equivariant to them.

  As two specific examples, let's look at linear regression and NTK for fully-connected MLPs.

  - **Min-norm least-squares linear regression.**

    Recall that the solution to min-norm least-squares linear regression $Ax = b$ is given by Moore–Penrose pseudo-inverse $x = A^+ b$. For any matrix $A \in \mathbb{R}^{m \times n}$ with SVD $U\Sigma V^* = A$, and $T \in O(n)$ (where $O(n)$ is the orthogonal group in dimension n), we have

$$(AT^\intercal)^+ = (U\Sigma V^* T^\intercal)^+ = TV\Sigma^+ U^* = TA^+, \tag{B.49}$$

    where we used $T^* = T^\intercal$ for $T \in O(n)$. The solution for the transformed data $AT^\intercal$ and b is thus

$$(AT^\intercal)^+ b = TA^+ b. \tag{B.50}$$

Thus, for any new data point $\tilde{x} \in \mathbb{R}^n$ and its transformed version $T\tilde{x} \in \mathbb{R}^n$,

$$\underbrace{(T\tilde{x})^\mathsf{T}(AT^\mathsf{T})^+b}_{\text{transformed problem prediction}} = \tilde{x}^\mathsf{T}T^\mathsf{T}TA^+ = \underbrace{\tilde{x}A^+}_{\text{original problem prediction}}. \tag{B.51}$$

Hence, min-norm least-squares linear regression is equivariant to orthogonal transforms.

– **MLP trained with squared loss in NTK regime.**

We first recall the NTK recursive formula from [83].

Denote the NTK for a MLP with L layers with the scalar kernel $\Theta^{(L)} \colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. Let $\beta > 0$ be the (fixed) parameter for the bias strength in the network model, and σ be the activation function. Given $x, z \in \mathbb{R}^d$, it can be recursively defined as following. For $h \in [L]$,

$$\Theta^{(h)}(x,z) \triangleq \Theta^{(h-1)}(x,z)\dot{\Sigma}^{(h)}(x,z) + \Sigma^{(h)}(x,z), \tag{B.52}$$

where

$$\Sigma^{(0)}(x,z) = \frac{1}{d}x^\mathsf{T}z + \beta^2, \tag{B.53}$$

$$\Lambda^{(h-1)}(x,z) = \begin{pmatrix} \Sigma^{(h-1)}(x,x) & \Sigma^{(h-1)}(x,z) \\ \Sigma^{(h-1)}(z,x) & \Sigma^{(h-1)}(z,z) \end{pmatrix}, \tag{B.54}$$

$$\Sigma^{(h)}(x,z) = c \cdot \mathbb{E}_{(u,v)\sim\mathcal{N}(0,\Lambda^{(h-1)})}\left[\sigma(u)\sigma(v)\right] + \beta^2, \tag{B.55}$$

$$\dot{\Sigma}^{(h)}(x,z) = c \cdot \mathbb{E}_{(u,v)\sim\mathcal{N}(0,\Lambda^{(h-1)})}\left[\dot{\sigma}(u)\dot{\sigma}(v)\right], \tag{B.56}$$

for some constant c.

It is evident from the recursive formula, that $\Theta^{(h)}(x,z)$ only depends on $x^\mathsf{T}x$, $z^\mathsf{T}z$ and $x^\mathsf{T}z$. Therefore, the NTK is *invariant* to orthogonal transforms.

Furthermore, training an MLP in NTK regime is the same as kernel regression with the NTK [83], which has a unique solution only depending on the kernel matrix on training set, denoted as $K_{\mathsf{train}} \in \mathbb{R}^{n\times n}$, where n is the training set size. Specifically, for training data $\{(x_i, y_i)\}_{i\in[n]}$, the solution $f^*_{\mathsf{NTK}} \colon \mathbb{R} \to \mathbb{R}$ can be written as

$$f^*_{\mathsf{NTK}}(x) = \begin{pmatrix} \Theta^{(L)}(x, x_1) & \Theta^{(L)}(x, x_2) & \cdots & \Theta^{(L)}(x, x_n) \end{pmatrix} K^{-1}_{\mathsf{train}} y, \tag{B.57}$$

where $y = \begin{pmatrix} y_1 & y_2 & \dots & y_n \end{pmatrix}$ is the vector of training labels.

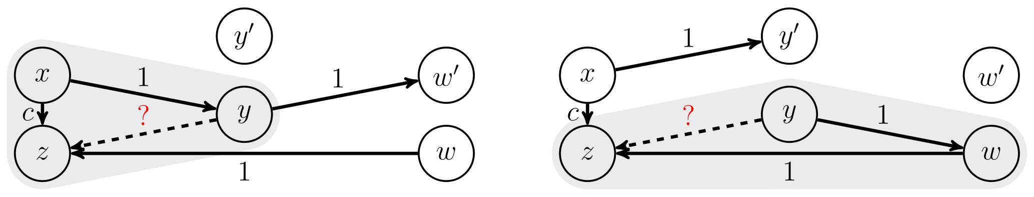
Consider any orthogonal transform $T \in O(d)$, and the NTK regression trained on the transformed data $\{(Tx_i, y_i)\}_{i \in [n]}$. Denote the solution as $f^*_{\mathsf{NTK},T} \colon \mathbb{R} \to \mathbb{R}$. As we have shown, K^{-1}_{train} is invariant to such transforms, and remains the same. Therefore,

$$\begin{aligned} f^*_{\mathsf{NTK},T}(Tx) &= \begin{pmatrix} \Theta^{(L)}(Tx, Tx_1) & \Theta^{(L)}(Tx, Tx_2) & \cdots & \Theta^{(L)}(Tx, Tx_n) \end{pmatrix} K^{-1}_{\mathsf{train}} y && \text{(B.58)} \\ &= \begin{pmatrix} \Theta^{(L)}(x, x_1) & \Theta^{(L)}(x, x_2) & \cdots & \Theta^{(L)}(x, x_n) \end{pmatrix} K^{-1}_{\mathsf{train}} y && \text{(B.59)} \\ &= f^*_{\mathsf{NTK}}(x). && \text{(B.60)} \end{aligned}$$

Hence, MLPs trained (with squared loss) in NTK regime is equivariant to orthogonal transforms.

Furthermore, we note that there are many variants of MLP NTK formulas depending on details such as the particular initialization scheme and bias settings. However, they usually only lead to slight changes that do not affect our results. For example, while the above recursive NTK formula are derived assuming that the bias terms are initialized with a normal distribution [83], the formulas for initializing bias as zeros [48] does not affect the dependency only on dot product, and thus our results still hold true.

These cases conclude the proof. $\square$



$$\mathsf{vio}(\hat d) \geq \frac{\hat d(x,z)}{\hat d(x,y) + \hat d(y,z)} \geq \frac{c}{\mathsf{dis}_S(\hat d)(\mathsf{dis}_S(\hat d) + \hat d(y,z))} \qquad \mathsf{vio}(\hat d) \geq \frac{\hat d(y,z)}{\hat d(y,w) + \hat d(w,z)} \geq \frac{\hat d(y,z)}{2 \cdot \mathsf{dis}_S(\hat d)}$$

Training ($\longrightarrow$) : $d(x,z) = c$, $d(w,z) = 1$, $d(x,y) = 1$, $d(y,w') = 1$.
Test ($\dashrightarrow$) : $\hat d(y,z) = ?$

Training ($\longrightarrow$) : $d(x,z) = c$, $d(w,z) = 1$, $d(x,y') = 1$, $d(y,w) = 1$.
Test ($\dashrightarrow$) : $\hat d(y,z) = ?$

Figure B-1: Two training sets pose incompatible constraints (●) for the test pair distance $d(y,z)$. With one-hot features, an orthogonal transform can exchange $(*,y) \leftrightarrow (*,y')$ and $(*,w) \leftrightarrow (*,w')$, leaving the test pair (y,z) unchanged, but transforming the training set from one scenario to the other. Given either set, an OrthEquiv algorithm must attain same training distortion and predict identically on (y,z). For appropriate c, this implies large distortion (not fitting training set) or violation (not approximately a quasimetric) in one of these cases.

B.2.3 Theorem 3.4.6: Failure of OrthEquiv Algorithms

Theorem 3.4.6 (Failure of OrthEquiv Algorithms). Let $(f_n)_n$ be an *arbitrary* sequence of large values. There is an infinite sequence of quasimetric spaces $((\mathcal{X}_n, d_n))_n$ with $|\mathcal{X}_n| = n$, $\mathcal{X}_n \subset \mathbb{R}^n$ such that, over a random training set S of size m, any OrthEquiv algorithm outputs a predictor $\hat d$ that

- $\hat d$ fails *non-negativity*, or
- $\max(\mathsf{dis}_S(\hat d), \mathsf{vio}(\hat d)) \geq f_n$ (i.e., $\hat d$ approximates training S badly or is far from a quasimetric),

with probability $1/2 - o(1)$, as long as S does not contain almost all of the pairs $1 - m/n^2 = \omega(n^{-1/3})$, and does not only include few pairs $m/n^2 = \omega(n^{-1/2})$.

Recall that the little-Omega notation means $f = \omega(g) \iff g = o(f)$.

Proof

Proof strategy. In our proof below, we will extend the construction discussed in Section 3.4.2 to large quasimetric spaces (reproduced here as Figure B-1). To do so, we

1. Construct large quasimetric spaces containing many copies of the (potentially failing) structure in Figure B-1, where we can consider training sets of certain properties such that

    - we can pair up such training sets,
    - an algorithm equivariant to orthogonal transforms must fail on one of them,
    - for each pair, the two training sets has equal probability of being sampled;

    Then, it remains to show that with probability $1 - o(1)$ we end up with a training set of such properties.

2. Consider sampling training set as independently collecting each pair with a certain probability p, and carefully analyze the conditions to sample a training set with the special properties with high probability $1 - o(1)$.

3. Extend to fixed-size training sets and show that, under similar conditions, we sample a training set with the special properties with high probability $1 - o(1)$.

In the discussion below and the proof, we will freely speak of infinite distances between two elements of $\mathcal{X}$, but really mean a very large value (possibly finite). This allows us to make the argument clearer and less verbose. Therefore, we are not restricting the applicable settings of Theorem 3.4.6 to quasimetrics with (or without) infinite distances.

In Section 3.4.2, we showed how orthogonal-transform-equivariant algorithms can not predict $\hat{d}(y, z)$ differently for the two particular quasimetric spaces and their training sets shown in Figure B-1.

But are these the only bad training sets? Before the proof, let us consider what kinds of training sets are bad for these two quasimetric spaces. Consider the quasimetrics $d_{\textsf{left}}$ and $d_{\textsf{right}}$ over $\mathcal{X} \triangleq \{x, y, y', z, w, w'\}$, with distances as shown in the left and right parts of Figure B-1, where we assume that the unlabeled pairs have infinite distances except in the <u>left pattern</u> $d(x, w') \leq 2$, and in the <u>both patterns</u> $d(y, z)$ has some appropriate value consistent with the respective triangle inequality.

Specifically, we ask:

- For what training sets $S_{\mathsf{left}} \subset \mathcal{X} \times \mathcal{X}$ can we interchange $y \leftrightarrow y'$ and $w \leftrightarrow w'$ on 2nd input to obtain a valid training set for d_{right}, regardless of c?
- For what training sets $S_{\mathsf{right}} \subset \mathcal{X} \times \mathcal{X}$ can we interchange $y \leftrightarrow y'$ and $w \leftrightarrow w'$ on 2nd input to obtain a valid training set for d_{left}, regardless of c?

Note that if S_{left} (or S_{right}) satisfies its condition, the predictor $\hat{d}$ from an algorithm equivariant to orthogonal transforms must (1) predict $\hat{d}(y, z)$ identically and (2) attain the same training set distortion on it and its transformed training set. As we will see in the proof for Theorem 3.4.6, this implies large distortion or violation for appropriate c.

Intuitively, all we need is that the transformed data do not break quasimetric constraints. However, its conditions are actually nontrivial as we want to set c to arbitrary:

- We can't have $(x, w) \in S_{\mathsf{right}}$ because it would be transformed into (x, w') which has $d_{\mathsf{left}}(x, w') \leq 2$. Then $d_{\mathsf{right}}(x, w) \leq 2$ and then restricts the possible values of c due to triangle inequality with $d_{\mathsf{right}}(w, z) = 1$. For similar reasons, we can't have $(x, w') \in S_{\mathsf{left}}$. In fact, we can't have a path of finite total distance from x to w (or w') in S_{right} (or S_{left}).
- We can not have $(y', y') \in S_{(\cdot)}$ (which has distance 0), which would get transformed into (y', y) with distance 0, which (on the other pattern) would restrict the possible values of c due to triangle inequality. For similar reasons (w', w'), and cycles containing y' or w' with finite total distance, should be avoided.
- For the theoretical analysis, we assumed that the truth d is a quasimetric rather than just being a quasipseudometric. The difference is that quasipseudometric additionally allows two distinct elements to have 0 distance. This assumptions allows us to freely talk about distance ratios for defining distortion and violation.

  For this particular reason, we can't allow (y, y'), (y', y), (w, w'), (w', w), (y, y) or (w, w), as they break this assumption. However, with metrics more friendly to zero distances (than distortion and violation, which are based on distance ratios), it might be possible to allow them and obtain better bounds in the second-moment argument below in the proof for Theorem 3.4.6.

With these understandings of the pattern shown in Figure B-1, we are ready to discuss the constructed quasimetric space and training sets.

Proof of Theorem 3.4.6. Our proof follows the outline listed above.

1. **Construct large quasimetric spaces containing many copies of the (potentially failing) structure in Figure B-1.**

   For any $n > 0$, consider the following quasimetric space $(\mathcal{X}_n, d_n)$ of size n, with one-hot features. WLOG, assume $n = 12k$ is a multiple of 12. If it is not, set at most 11 elements to have infinite distance with every other node. This won't affect the asymptotics. Let the $n = 12k$ elements of the space be

$$\begin{aligned}\mathcal{X}_n = \{ & x_1^{\text{left}}, \dots, x_k^{\text{left}},\ x_1^{\text{right}}, \dots, x_k^{\text{right}}, w_1^{\text{left}}, \dots, w_k^{\text{left}},\ w_1^{\text{right}}, \dots, w_k^{\text{right}}, \\ & y_1^{\text{left}}, \dots, y_k^{\text{left}},\ y_1^{\text{right}}, \dots, y_k^{\text{right}}, w_1^{\prime\text{left}}, \dots, w_k^{\prime\text{left}}, w_{k+1}^{\prime\text{right}}, \dots, w_{2k}^{\prime\text{right}}, \\ & y_1^{\prime\text{left}}, \dots, y_k^{\prime\text{left}}, y_{k+1}^{\prime\text{right}}, \dots y_{2k}^{\prime\text{right}}, z_1, \dots, z_k, \qquad z_{k+1}, \dots, z_{2k}\},\end{aligned} \tag{B.61}$$

   with quasimetric distances, $\forall i, j$,

$$d_n(x_i^{\text{left}}, z_j) = d_n(x_i^{\text{right}}, z_j) \quad = c \tag{B.62}$$

$$d_n(w_i^{\text{left}}, z_j) = d_n(w_i^{\text{right}}, z_j) \quad = 1 \tag{B.63}$$

$$d_n(x_i^{\text{left}}, y_i^{\text{left}}) = d_n(x_i^{\text{right}}, y_i^{\prime\text{right}}) = 1 \tag{B.64}$$

$$d_n(y_i^{\text{left}}, w_i^{\prime\text{left}}) = d_n(y_i^{\text{right}}, w_i^{\text{right}}) = 1 \tag{B.65}$$

$$d_n(x_i^{\text{left}}, w_i^{\prime\text{left}}) = 2 \tag{B.66}$$

$$d_n(y_i^{\text{left}}, z_j) = c \tag{B.67}$$

$$d_n(y_i^{\text{right}}, z_j) = 2, \tag{B.68}$$

   where subscripts are colored to better show when they are the same (or different), unlisted distances are infinite (except that $d_n(u, u) = 0, \forall u \in \mathcal{X}$). Essentially, we equally divide the $12k$ nodes into 6 "types", $\{x, y, w, z, w', y'\}$, corresponding to the 6 nodes from Figure B-1, where each type has half of its nodes corresponding to the <u>left pattern</u> (of Figure B-1), and the other half corresponding to the <u>right pattern</u>, except for the z types.

Furthermore,

- Among the <u>left-pattern</u> nodes, each set with the same subscript are bundled together in the sense that x_i^{left} only has finite distance to y_i^{left} which only has finite distance to $w_i'^{\mathsf{left}}$ (instead of other y_j^{left}'s or $w_k'^{\mathsf{left}}$'s). However, since distance to/from y_i^{left} and w_i^{left} are infinite anyways, we can pair

$$(x_i^{\mathsf{left}}, y_i^{\mathsf{left}}, w_i'^{\mathsf{left}}, y_j'^{\mathsf{left}}, w_l^{\mathsf{left}}, z_h) \tag{B.69}$$

  for any i, j, l, h, to obtain a <u>left pattern</u>.

- Among the <u>right-pattern</u> nodes, each set with the same subscript are bundled together in the sense that x_i^{right} only has finite distance to $y_i'^{\mathsf{right}}$, and y_j^{right} which only has finite distance to w_j^{right} (instead of other $y_j'^{\mathsf{right}}$'s or w_k^{right}'s). However, since are distances are infinite anyways, we can pair

$$(x_i^{\mathsf{right}}, y_i'^{\mathsf{right}}, y_j^{\mathsf{right}}, w_j^{\mathsf{right}}, w_l'^{\mathsf{right}}, z_h) \tag{B.70}$$

  for any i, j, l, h, to obtain a <u>right pattern</u>.

We can see that $(\mathcal{X}, d)$ indeed satisfies all quasimetric space requirements (Definition 3.2.1), including triangle inequalities (e.g., by, for each (a, b) with finite distance $d_n(a, b) < \infty$, enumerating finite-length paths from a to b).

Now consider the sampled training set S.

- We say S is *bad* on a <u>left pattern</u> specified by $i_{\mathsf{left}}, j_{\mathsf{left}}, l_{\mathsf{left}}, h_{\mathsf{left}}$, if

$$S \supset \{(x^{\mathsf{left}}_{i_{\mathsf{left}}}, z_{h_{\mathsf{left}}}), (x^{\mathsf{left}}_{i_{\mathsf{left}}}, y^{\mathsf{left}}_{i_{\mathsf{left}}}), (y^{\mathsf{left}}_{i_{\mathsf{left}}}, w'^{\mathsf{left}}_{i_{\mathsf{left}}}), (w^{\mathsf{left}}_{l_{\mathsf{left}}}, z_{h_{\mathsf{left}}})\} \tag{B.71}$$

$$\begin{aligned}\emptyset = S \cap \{&(y^{\mathsf{left}}_{i_{\mathsf{left}}}, z_{h_{\mathsf{left}}}), (y'^{\mathsf{left}}_{i_{\mathsf{left}}}, y^{\mathsf{left}}_{i_{\mathsf{left}}}), (w^{\mathsf{left}}_{l_{\mathsf{left}}}, w^{\mathsf{left}}_{l_{\mathsf{left}}}), (y'^{\mathsf{left}}_{j_{\mathsf{left}}}, y'^{\mathsf{left}}_{j_{\mathsf{left}}}), (w'^{\mathsf{left}}_{i_{\mathsf{left}}}, w'^{\mathsf{left}}_{i_{\mathsf{left}}}),\\ &(x^{\mathsf{left}}_{i_{\mathsf{left}}}, w'^{\mathsf{left}}_{i_{\mathsf{left}}}), (y^{\mathsf{left}}_{i_{\mathsf{left}}}, y'^{\mathsf{left}}_{j_{\mathsf{left}}}), (w^{\mathsf{left}}_{l_{\mathsf{left}}}, w'^{\mathsf{left}}_{i_{\mathsf{left}}}), (y'^{\mathsf{left}}_{j_{\mathsf{left}}}, y^{\mathsf{left}}_{i_{\mathsf{left}}}), (w'^{\mathsf{left}}_{i_{\mathsf{left}}}, w^{\mathsf{left}}_{l_{\mathsf{left}}})\}\end{aligned} \tag{B.72}$$

- We say S is *bad* on a <u>right pattern</u> specified by $i_{\mathsf{right}}, j_{\mathsf{right}}, l_{\mathsf{right}}, h_{\mathsf{right}}$, if

$$S \supset \{(x^{\mathsf{right}}_{i_{\mathsf{right}}}, z_{h_{\mathsf{right}}}), (x^{\mathsf{right}}_{i_{\mathsf{right}}}, y'^{\mathsf{right}}_{i_{\mathsf{right}}}), (y'^{\mathsf{right}}_{j_{\mathsf{right}}}, w^{\mathsf{right}}_{j_{\mathsf{right}}}), (w^{\mathsf{right}}_{j_{\mathsf{right}}}, z_{h_{\mathsf{right}}})\} \tag{B.73}$$

$$\begin{aligned}\emptyset = S \cap \{&(y^{\mathsf{right}}_{j_{\mathsf{right}}}, z_{h_{\mathsf{right}}}), (y^{\mathsf{right}}_{j_{\mathsf{right}}}, y^{\mathsf{right}}_{j_{\mathsf{right}}}), (w^{\mathsf{right}}_{j_{\mathsf{right}}}, w^{\mathsf{right}}_{j_{\mathsf{right}}}), (y'^{\mathsf{right}}_{i_{\mathsf{right}}}, y'^{\mathsf{right}}_{i_{\mathsf{right}}}),\\ &(w'^{\mathsf{right}}_{l_{\mathsf{right}}}, w'^{\mathsf{right}}_{l_{\mathsf{right}}}), (x^{\mathsf{right}}_{i_{\mathsf{right}}}, w'^{\mathsf{right}}_{j_{\mathsf{right}}}), (y^{\mathsf{right}}_{j_{\mathsf{right}}}, y'^{\mathsf{right}}_{i_{\mathsf{right}}}), (w^{\mathsf{right}}_{j_{\mathsf{right}}}, w'^{\mathsf{right}}_{l_{\mathsf{right}}}),\\ &(y'^{\mathsf{right}}_{i_{\mathsf{right}}}, y^{\mathsf{right}}_{j_{\mathsf{right}}}), (w'^{\mathsf{right}}_{l_{\mathsf{right}}}, w^{\mathsf{right}}_{j_{\mathsf{right}}})\}\end{aligned} \tag{B.74}$$

Most importantly,

- If S is bad on a <u>left pattern</u> specified by $i_{\mathsf{left}}, j_{\mathsf{left}}, l_{\mathsf{left}}, h_{\mathsf{left}}$, consider the orthogonal transform that interchanges $y^{\mathsf{left}}_{i_{\mathsf{left}}} \leftrightarrow y'^{\mathsf{left}}_{j_{\mathsf{left}}}$ and $w^{\mathsf{left}}_{l_{\mathsf{left}}} \leftrightarrow w'^{\mathsf{left}}_{i_{\mathsf{left}}}$ on 2nd input. In S, the possible transformed pairs are

$$\begin{aligned}
d(x^{\mathsf{left}}_{i_{\mathsf{left}}}, y^{\mathsf{left}}_{i_{\mathsf{left}}}) = 1 &\quad\longrightarrow\quad d(x^{\mathsf{left}}_{i_{\mathsf{left}}}, y'^{\mathsf{left}}_{j_{\mathsf{left}}}) = 1, && \text{(known in } S\text{)}\\
d(y^{\mathsf{left}}_{i_{\mathsf{left}}}, w'^{\mathsf{left}}_{i_{\mathsf{left}}}) = 1 &\quad\longrightarrow\quad d(y^{\mathsf{left}}_{i_{\mathsf{left}}}, w^{\mathsf{left}}_{l_{\mathsf{left}}}) = 1, && \text{(known in } S\text{)}\\
d(u, y^{\mathsf{left}}_{i_{\mathsf{left}}}) = \infty &\quad\longrightarrow\quad d(u, y'^{\mathsf{left}}_{j_{\mathsf{left}}}) = \infty,\\
& && \text{(poissble in } S \text{ for some } u \neq x^{\mathsf{left}}_{i_{\mathsf{left}}})\\
d(u, y'^{\mathsf{left}}_{j_{\mathsf{left}}}) = \infty &\quad\longrightarrow\quad d(u, y^{\mathsf{left}}_{i_{\mathsf{left}}}) = \infty, && \text{(poissble in } S \text{ for some } u)\\
d(u, w'^{\mathsf{left}}_{i_{\mathsf{left}}}) = \infty &\quad\longrightarrow\quad d(u, w^{\mathsf{left}}_{l_{\mathsf{left}}}) = \infty,\\
& && \text{(poissble in } S \text{ for some } u \notin \{x^{\mathsf{left}}_{i_{\mathsf{left}}}, y^{\mathsf{left}}_{i_{\mathsf{left}}}\})\\
d(u, w^{\mathsf{left}}_{l_{\mathsf{left}}}) = \infty &\quad\longrightarrow\quad d(u, w'^{\mathsf{left}}_{i_{\mathsf{left}}}) = \infty. && \text{(poissble in } S \text{ for some } u)
\end{aligned}$$

The crucial observation is that the transformed training set just look like one sampled from a quasimetric space where

– the quasimetric space has one less set of left-pattern elements,

– the quasimetric space has one more set of right-pattern elements, and

– transformed training set is *bad* on that extra right pattern (given by the extra set of right-pattern elements),

which can be easily verified by comparing the transformed training set with the requirements in Equations (B.73) and (B.74).

- Similarly, if S is bad on a right pattern specified by $i_{\mathsf{right}}, j_{\mathsf{right}}, l_{\mathsf{right}}, h_{\mathsf{right}}$, consider the orthogonal transform that interchanges $y^{\mathsf{right}}_{j_{\mathsf{right}}} \leftrightarrow y'^{\mathsf{right}}_{i_{\mathsf{right}}}$ and $w^{\mathsf{right}}_{j_{\mathsf{right}}} \leftrightarrow w'^{\mathsf{right}}_{l_{\mathsf{right}}}$ on 2nd input. In S the possible transformed pairs are

$$
\begin{aligned}
d(x^{\mathsf{right}}_{i_{\mathsf{right}}}, y'^{\mathsf{right}}_{i_{\mathsf{right}}}) = 1 \quad &\longrightarrow \quad d(x^{\mathsf{right}}_{i_{\mathsf{right}}}, y^{\mathsf{right}}_{j_{\mathsf{right}}}) = 1, && \text{(known in } S\text{)} \\
d(y^{\mathsf{right}}_{j_{\mathsf{right}}}, w^{\mathsf{right}}_{j_{\mathsf{right}}}) = 1 \quad &\longrightarrow \quad d(y^{\mathsf{right}}_{j_{\mathsf{right}}}, w'^{\mathsf{right}}_{l_{\mathsf{right}}}) = 1, && \text{(known in } S\text{)} \\
d(u, y^{\mathsf{right}}_{j_{\mathsf{right}}}) = \infty \quad &\longrightarrow \quad d(u, y'^{\mathsf{right}}_{i_{\mathsf{right}}}) = \infty, \\
& && \text{(poissble in } S \text{ for some } u\text{)} \\
d(u, y'^{\mathsf{right}}_{i_{\mathsf{right}}}) = \infty \quad &\longrightarrow \quad d(u, y^{\mathsf{right}}_{j_{\mathsf{right}}}) = \infty, \\
& && \text{(poissble in } S \text{ for some } u \neq x^{\mathsf{right}}_{i_{\mathsf{right}}}\text{)} \\
d(u, w'^{\mathsf{right}}_{l_{\mathsf{right}}}) = \infty \quad &\longrightarrow \quad d(u, w^{\mathsf{right}}_{j_{\mathsf{right}}}) = \infty, \\
& && \text{(poissble in } S \text{ for some } u\text{)} \\
d(u, w^{\mathsf{right}}_{j_{\mathsf{right}}}) = \infty \quad &\longrightarrow \quad d(u, w'^{\mathsf{right}}_{l_{\mathsf{right}}}) = \infty. \\
& && \text{(poissble in } S \text{ for some } u \notin \{x^{\mathsf{right}}_{i_{\mathsf{right}}}, y^{\mathsf{right}}_{j_{\mathsf{right}}}\}\text{)}
\end{aligned}
$$

Again, the crucial observation is that the transformed training set just look like one sampled from a quasimetric space where

– the quasimetric space has one less set of right-pattern elements,

– the quasimetric space has one more set of left-pattern elements, and

– transformed training set is *bad* on that extra left pattern (given by the extra set of left-pattern elements), which can be easily verified by comparing the transformed training set with the requirements in Equations (B.71) and (B.72).

Therefore, when S is bad on *both a left pattern and a right pattern* (necessarily on disjoint sets of pairs), we consider the following orthogonal transform composed of:

(a) both transforms specified above (which only transforms 2nd inputs),

(so that after this we obtain *another possible training set of same size from the quasimetric space that is only different up to some permutation of* $\mathcal{X}$)

(b) a permutation of $\mathcal{X}$ (on both inputs) so that the bad left-pattern nodes and the bad right-pattern nodes exchange features,

This transforms gives *another possible training set of same size from the same quasimetric space, also is bad on a left pattern and a right pattern*. Moreover, with a particular way of select bad patterns (e.g., by the order of the subscripts), this process is *reversible*. Therefore, we have defined a way to pair up all such bad training sets.

Consider the predictors $\hat{d}_{\mathsf{before}}$ and $\hat{d}_{\mathsf{after}}$ trained on these two training sets (before and after transform) with an learning algorithm equivariant to orthogonal transforms. Assuming that they satisfy non-negativity and Identity of Indiscernibles, we have,

- The predictors have the same distortion over respective training sets.

  Therefore we denote this distortion as $\mathsf{dis}_S(\hat{d})$ without specifying the predictor $\hat{d}$ or training set S.

- the predictors must predict the same on heldout pairs in the sense that

$$\hat{d}_{\mathsf{before}}(y^{\mathsf{left}}_{i_{\mathsf{left}}}, z_{h_{\mathsf{left}}}) = \hat{d}_{\mathsf{after}}(y^{\mathsf{right}}_{j_{\mathsf{right}}}, z_{h_{\mathsf{right}}}) \tag{B.75}$$

$$\hat{d}_{\mathsf{before}}(y^{\mathsf{right}}_{j_{\mathsf{right}}}, z_{h_{\mathsf{right}}}) = \hat{d}_{\mathsf{after}}(y^{\mathsf{left}}_{i_{\mathsf{left}}}, z_{h_{\mathsf{left}}}). \tag{B.76}$$

Focusing on the first, we denote

$$\hat{d}(y, z) \triangleq \hat{d}_{\mathsf{before}}(y^{\mathsf{left}}_{i_{\mathsf{left}}}, z_{h_{\mathsf{left}}}) = \hat{d}_{\mathsf{after}}(y^{\mathsf{right}}_{j_{\mathsf{right}}}, z_{h_{\mathsf{right}}}) \tag{B.77}$$

without specifying the predictor $\hat{d}$ or the specific y and z.

However, the quasimetric constraints on heldout pairs $(y^{\mathsf{left}}_{i_{\mathsf{left}}}, z_{h_{\mathsf{left}}})$ and $(y^{\mathsf{right}}_{j_{\mathsf{right}}}, z_{h_{\mathsf{right}}})$ are completely different (see the left vs. right part of Figure B-1). Therefore, as shown in Figure B-1, assuming *non-negativity*, *one of the two predictors* must have total violation at least

$$\mathsf{vio}(\hat{d}) \geq \max\left(\frac{c}{\mathsf{dis}_S(\hat{d})(\mathsf{dis}_S(\hat{d}) + \hat{d}(y, z))}, \frac{\hat{d}(y, z)}{2 \cdot \mathsf{dis}_S(\hat{d})}\right). \tag{B.78}$$

Fixing a large enough c, two terms in the max of Equation (B.78) can equal for some $\hat{d}(y, z)$, and are respectively decreasing and increasing in $\hat{d}(y, z)$. In that case, we have

$$\mathsf{vio}(\hat{d}) \geq \frac{\delta}{2 \cdot \mathsf{dis}_S(\hat{d})}, \tag{B.79}$$

for $\delta > 0$ such that

$$\frac{c}{\mathsf{dis}_S(\hat{d})(\mathsf{dis}_S(\hat{d}) + \delta)} = \frac{\delta}{2 \cdot \mathsf{dis}_S(\hat{d})}. \tag{B.80}$$

Solving the above quadratic equation gives

$$\delta = \frac{-\mathsf{dis}_S(\hat{d}) + \sqrt{\mathsf{dis}_S(\hat{d})^2 + 8c}}{2}, \tag{B.81}$$

leading to

$$\mathsf{vio}(\hat{d}) \geq \frac{-1 + \sqrt{1 + 8c/\mathsf{dis}_S(\hat{d})^2}}{4}. \tag{B.82}$$

Therefore, choosing $c \geq f_n^2(4f_n+1)^2$ gives

$$\begin{aligned}
& \mathsf{dis}_S(\hat{d}) \leq f_n && \text{(B.83)} \\
\implies & \mathsf{vio}(\hat{d}) \geq \frac{-1+\sqrt{1+8c/\mathsf{dis}_S(\hat{d})^2}}{4} && \text{(B.84)} \\
& \geq \frac{-1+\sqrt{1+8f_n^2(4f_n+1)^2/f_n^2}}{4} && \text{(B.85)} \\
& = \frac{-1+\sqrt{1+8(4f_n+1)^2}}{4} && \text{(B.86)} \\
& \geq \frac{-1+4f_n+1}{4} && \text{(B.87)} \\
& = f_n. && \text{(B.88)}
\end{aligned}$$

Hence, for training sets that are *bad* on *both a <u>left pattern</u> and a <u>right pattern</u>*, we have shown a way to pair them up such that

- each pair of training sets have the same size, and
- the algorithm fail on one of each pair by producing a distance predictor that
  - has either distortion over training set $\geq f_n$, or violation $\geq f_n$, and
  - has test MSE $\geq f_n$.

Remark B.2.1. Note that all training sets of size m has equal probability of being sampled. Therefore, to prove the theorem, it suffices to show that with probability $1-o(1)$, we can sample a training set of size m that is *bad* on *both a <u>left pattern</u> and a <u>right pattern</u>*.

2. **Consider sampling training set as individually collecting each pair with a certain probability p, and carefully analyze the conditions to sample a training set with the special properties with high probability $1-o(1)$.**

   In probabilistic methods, it is often much easier to work with independent random variables. Therefore, instead of considering uniform sampling a training set S of fixed size m, we consider including each pair in S with probability p, chosen independently. We will first show result based on this sampling procedure via a second moment argument, and later extend to the case with a fixed-size training set.

First, let's define some notations that ignore constants:

$$f \sim g \iff f = (1 + o(1))g \tag{B.89}$$

$$f \ll g \iff f = o(g). \tag{B.90}$$

We start with stating a standard result from the second moment method [3].

Corollary B.2.2 (Corollary 4.3.5 of [3]). Consider random variable $X = X_1 + X_2 + \cdots + X_n$, where X_i is the indicator random variable for event A_i. Write $i \sim j$ if $i \neq j$ and the pair of events (A_i, A_j) are not independent. Suppose the following quantity does not depend on i:

$$\Delta^* \triangleq \sum_{j \sim i} \mathbb{P}\left[A_j \mid A_i\right]. \tag{B.91}$$

If $\mathbb{E}[X] \to \infty$ and $\Delta^* \ll \mathbb{E}[X]$, then $X \sim \mathbb{E}[X]$ with probability $1 - o(1)$.

We will apply this corollary to obtain conditions on p such that S with probability $1 - o(1)$ is *bad* on some left pattern, and conditions such that S with probability $1 - o(1)$ is *bad* on some right pattern. A union bound would then give the desired result.

- **S is *bad* on some left pattern.**

  Recall that a left pattern is specified by $i_{\mathsf{left}}, j_{\mathsf{left}}, l_{\mathsf{left}}, h_{\mathsf{left}}$ all $\in [k]$:

$$(x^{\mathsf{left}}_{i_{\mathsf{left}}}, y^{\mathsf{left}}_{i_{\mathsf{left}}}, w'^{\mathsf{left}}_{i_{\mathsf{left}}}, y'^{\mathsf{left}}_{j_{\mathsf{left}}}, w^{\mathsf{left}}_{l_{\mathsf{left}}}, z_{h_{\mathsf{left}}}) \tag{B.92}$$

Therefore, we consider $k^4 = (\frac{n}{12})^4$ events of the form

$$A_{i_{\text{left}}, j_{\text{left}}, l_{\text{left}}, h_{\text{left}}} \triangleq \{S \text{ is bad on the } \underline{\text{left pattern}} \text{ at } i_{\text{left}}, j_{\text{left}}, l_{\text{left}}, h_{\text{left}}\}. \tag{B.93}$$

Obviously, these events are symmetrical, and the Δ^* in Equation (B.91) does not depend on i.

By the quasimetric space construction and the requirement for S to be bad on a <u>left pattern</u> in Equations (B.71) and (B.72), we can see that $(i_{\text{left}}, j_{\text{left}}, l_{\text{left}}, h_{\text{left}}) \sim (i'_{\text{left}}, j'_{\text{left}}, l'_{\text{left}}, h'_{\text{left}})$ only if $i_{\text{left}} = i'_{\text{left}}$ or $j_{\text{left}} = j'_{\text{left}}$ or $l_{\text{left}} = l'_{\text{left}}$ or $h_{\text{left}} = h'_{\text{left}}$.

Therefore, we have

$$\begin{aligned}
\mathbb{E}\left[X\right] &\sim n^4p^4(1-p)^{10} && \text{(include 4 pairs \& exclude 10 pairs)}\\
\Delta^* &\ll n^3p^4(1-p)^9 && \text{(share } j_{\mathsf{left}}\text{)}\\
&+ n^3p^2(1-p)^7 && \text{(share } i_{\mathsf{left}}\text{)}\\
&+ n^3p^4(1-p)^9 && \text{(share } l_{\mathsf{left}}\text{)}\\
&+ n^3p^4(1-p)^{10} && \text{(share } h_{\mathsf{left}}\text{)}\\
&+ n^2p^2(1-p)^4 && \text{(share } j_{\mathsf{left}}, i_{\mathsf{left}}\text{)}\\
&+ n^2p^4(1-p)^8 && \text{(share } j_{\mathsf{left}}, l_{\mathsf{left}}\text{)}\\
&+ n^2p^4(1-p)^9 && \text{(share } j_{\mathsf{left}}, h_{\mathsf{left}}\text{)}\\
&+ n^2p^2(1-p)^4 && \text{(share } i_{\mathsf{left}}, l_{\mathsf{left}}\text{)}\\
&+ n^2p(1-p)^6 && \text{(share } i_{\mathsf{left}}, h_{\mathsf{left}}\text{)}\\
&+ n^2p^3(1-p)^9 && \text{(share } l_{\mathsf{left}}, h_{\mathsf{left}}\text{)}\\
&+ n(1-p)^3 && \text{(share } i_{\mathsf{left}}, l_{\mathsf{left}}, h_{\mathsf{left}}\text{)}\\
&+ np^3(1-p)^8 && \text{(share } j_{\mathsf{left}}, l_{\mathsf{left}}, h_{\mathsf{left}}\text{)}\\
&+ np(1-p)^3 && \text{(share } j_{\mathsf{left}}, i_{\mathsf{left}}, h_{\mathsf{left}}\text{)}\\
&+ np^2(1-p) && \text{(share } j_{\mathsf{left}}, i_{\mathsf{left}}, l_{\mathsf{left}}\text{)}\\
&\sim n^3p^2(1-p)^7 + n^2(p^2(1-p)^4 + p(1-p)^6) && \text{(B.94)}\\
&+ n((1-p)^3 + p^2(1-p)). && \text{(B.95)}
\end{aligned}$$

Therefore, to apply Corollary B.2.2, we need to have

$$n^4p^4(1-p)^{10} \to \infty \tag{B.96}$$

$$n^3p^2(1-p)^7 \ll n^4p^4(1-p)^{10} \tag{B.97}$$

$$n^2(p^2(1-p)^4 + p(1-p)^6) \ll n^4p^4(1-p)^{10} \tag{B.98}$$

$$n((1-p)^3 + p^2(1-p)) \ll n^4p^4(1-p)^{10}, \tag{B.99}$$

which gives

$$p \gg n^{-1/2} \tag{B.100}$$

$$1 - p \gg n^{-1/3} \tag{B.101}$$

as a sufficient condition to for S to be bad on some left pattern with probability $1 - o(1)$.

- **S is *bad* on some right pattern.**

Recall that a right pattern is specified by $i_{\mathsf{right}}, j_{\mathsf{right}}, l_{\mathsf{right}}, h_{\mathsf{right}}$ all $\in [k]$:

$$(x^{\mathsf{right}}_{i_{\mathsf{right}}}, y'^{\mathsf{right}}_{i_{\mathsf{right}}}, y^{\mathsf{right}}_{j_{\mathsf{right}}}, w^{\mathsf{right}}_{j_{\mathsf{right}}}, w'^{\mathsf{right}}_{l_{\mathsf{right}}}, z_{h_{\mathsf{right}}}) \tag{B.102}$$

Similarly, we consider $k^4 = (\frac{n}{12})^4$ events of the form

$$A_{i_{\mathsf{right}}, j_{\mathsf{right}}, l_{\mathsf{right}}, h_{\mathsf{right}}} \triangleq \{S \text{ is bad on the left pattern at } i_{\mathsf{right}}, j_{\mathsf{right}}, l_{\mathsf{right}}, h_{\mathsf{right}}\}. \tag{B.103}$$

Again, these events are symmetrical, and Δ^* in Equation (B.91) does not depend on i.

Similarly, we have

$$\begin{aligned}
\mathbb{E}\left[X\right] &\sim n^4p^4(1-p)^{10} && \text{(include 4 pairs \& exclude 10 pairs)} \\
\Delta^* &\ll n^3p^3(1-p)^9 && \text{(share } i_{\mathsf{right}}\text{)} \\
&\quad + n^3p^3(1-p)^8 && \text{(share } j_{\mathsf{right}}\text{)} \\
&\quad + n^3p^4(1-p)^{10} && \text{(share } h_{\mathsf{right}}\text{)} \\
&\quad + n^3p^4(1-p)^9 && \text{(share } l_{\mathsf{right}}\text{)} \\
&\quad + n^2p^2(1-p)^4 && \text{(share } i_{\mathsf{right}}, j_{\mathsf{right}}\text{)} \\
&\quad + n^2p^2(1-p)^9 && \text{(share } i_{\mathsf{right}}, h_{\mathsf{right}}\text{)} \\
&\quad + n^2p^3(1-p)^8 && \text{(share } i_{\mathsf{right}}, l_{\mathsf{right}}\text{)} \\
&\quad + n^2p^2(1-p)^7 && \text{(share } j_{\mathsf{right}}, h_{\mathsf{right}}\text{)} \\
&\quad + n^2p^3(1-p)^5 && \text{(share } j_{\mathsf{right}}, l_{\mathsf{right}}\text{)} \\
&\quad + n^2p^4(1-p)^9 && \text{(share } h_{\mathsf{right}}, l_{\mathsf{right}}\text{)} \\
&\quad + np^2(1-p)^4 && \text{(share } j_{\mathsf{right}}, h_{\mathsf{right}}, l_{\mathsf{right}}\text{)} \\
&\quad + np^2(1-p)^8 && \text{(share } i_{\mathsf{right}}, h_{\mathsf{right}}, l_{\mathsf{right}}\text{)} \\
&\quad + np^2(1-p) && \text{(share } i_{\mathsf{right}}, j_{\mathsf{right}}, l_{\mathsf{right}}\text{)} \\
&\quad + n(1-p) && \text{(share } i_{\mathsf{right}}, j_{\mathsf{right}}, h_{\mathsf{right}}\text{)} \\
&\sim n^3p^3(1-p)^8 + n^2p^2(1-p)^4 && \text{(B.104)} \\
&\quad + n(1-p). && \text{(B.105)}
\end{aligned}$$

Therefore, to apply Corollary B.2.2, we need to have

$$\begin{aligned}
n^4p^4(1-p)^{10} &\to \infty && \text{(B.106)} \\
n^3p^3(1-p)^8 &\ll n^4p^4(1-p)^{10} && \text{(B.107)} \\
n^2p^2(1-p)^4 &\ll n^4p^4(1-p)^{10} && \text{(B.108)} \\
n(1-p) &\ll n^4p^4(1-p)^{10}, && \text{(B.109)}
\end{aligned}$$

which gives

$$p \gg n^{-3/4} \tag{B.110}$$

$$1 - p \gg n^{-1/3} \tag{B.111}$$

as a sufficient condition to for S to be bad on some right pattern with probability $1 - o(1)$.

So, by union bound, as long as

$$p \gg n^{-1/2} \tag{B.112}$$

$$1 - p \gg n^{-1/3}, \tag{B.113}$$

S is bad on some left pattern *and* some right pattern with probability $1 - o(1)$.

3. **Extend to fixed-size training sets and show that, under similar conditions, we sample a training set with the special properties with high probability $1 - o(1)$.**

   To extend to fixed-size training sets, we consider the following alteration procedure:

   (a) Sample training set S by independently include each pair with probability $p \triangleq \frac{m+\delta}{n^2}$, for some $\delta > 0$.

   (b) Show that with high probability $1 - o(1)$, we end up with $[m, m + 2\delta]$ pairs in S.

   (c) Make sure that p satisfy Equation (B.112) and Equation (B.113) so that S is bad on some left pattern *and* some right pattern with high probability $1 - o(1)$.

   (d) Randomly discard the additional pairs, and show that with high probability $1 - o(1)$ this won't affect that S is bad on some left pattern *and* some right pattern.

We now consider each step in details:

(a) **Sample training set S by independently include each pair with probability $p \triangleq \frac{m+\delta}{n^2}$, for some $\delta > 0$.**

For $p \triangleq \frac{m+\delta}{n^2}$, the number of pairs in the training set is distributed as

$$\text{Binomial}(n^2, \frac{m+\delta}{n^2}). \tag{B.114}$$

(b) **Show that with high probability $1-o(1)$, we end up with $[m, m+2\delta]$ pairs in S.**

Standard Binomial concentration tells us that,

$$\delta \gg n\sqrt{p(1-p)} \implies \mathbb{P}\left[\text{Binomial}(n^2, \frac{m+\delta}{n^2}) \notin [m, m+2\delta]\right] \to 0, \tag{B.115}$$

which can be satisfied if

$$\delta \gg n. \tag{B.116}$$

(c) **Make sure that p satisfy Equation (B.112) and Equation (B.113) so that S is bad on some <u>left pattern</u> *and* some <u>right pattern</u> with high probability $1-o(1)$.**

Therefore, we want

$$\frac{m+\delta}{n^2} \gg n^{-1/2} \tag{B.117}$$

$$1 - \frac{m+\delta}{n^2} \gg n^{-1/3}. \tag{B.118}$$

(d) **Randomly discard the additional pairs, and show that with high probability $1-o(1)$ this won't affect that S is bad on some <u>left pattern</u> *and* some <u>right pattern</u>.**

Consider any specific bad <u>left pattern</u> *and a* <u>right pattern</u> in S. It is sufficient that we don't break these two patterns during discarding.

Since we only discard pairs, it suffices to only consider the pairs we want to preserve, which are a total of 8 pairs across two patterns.

Each such pair is discarded the probability $\leq \frac{2\delta}{m}$, since we remove at most 2δ pairs. By union bound,

$$\mathbb{P}\left[\text{all 8 pairs are preserved}\right] \geq 1 - \frac{16\delta}{m}. \tag{B.119}$$

Hence, it suffices to make sure that

$$\delta \ll m. \tag{B.120}$$

Collecting all requirements, we have

$$\delta \gg n \tag{B.121}$$

$$\frac{m+\delta}{n^2} \gg n^{-1/2} \tag{B.122}$$

$$1 - \frac{m+\delta}{n^2} \gg n^{-1/3} \tag{B.123}$$

$$\delta \ll m. \tag{B.124}$$

Assume that

$$\frac{m}{n^2} \gg n^{-1/2} \tag{B.125}$$

$$1 - \frac{m}{n^2} \gg n^{-1/3}. \tag{B.126}$$

It can be easily verified that using $\delta \triangleq n^{1.1}$ satisfies all conditions.

Hence, for a uniformly randomly sampled training set S with size m, S is bad on some <u>left pattern</u> *and* some <u>right pattern</u> with high probability $1 - o(1)$, as long as

$$\frac{m}{n^2} \gg n^{-1/2} \tag{B.127}$$

$$1 - \frac{m}{n^2} \gg n^{-1/3}. \tag{B.128}$$

This is exactly the condition we need to prove the theorem (see Remark B.2.1). This concludes the proof. □

Discussions

Training set size dependency. Intuitively, when the training set has almost all pairs, violation can be lowered by simply fitting training set well; when it is small and sparse, the learning algorithm may have an easier job finding some consistent quasimetric. Theorem 3.4.6 shows that, outside these two cases, algorithms equivariant to orthogonal transforms can fail. Note that for the latter case, Theorem 3.4.6 requires the training fraction to decrease slower than $n^{-1/2}$, which rules out training sizes that is linear in n. We leave improving this result as future work. Nonetheless, Theorem 3.4.6 still covers common scenarios such as a fixed fraction of all pairs, and highlights that a training-data-agnostic result (such as the ones for PQEs) is not possible for these algorithms.

Proof techniques. In embedding theory, it is quite standard to analyze quasimetrics as directed graphs due to their lack of nice metric structure. In the proof for Theorem 3.4.6, we used abundant techniques from the probabilistic method, which are commonly used for analyzing graph properties in the asymptotic case, including Corollary B.2.2 from the second moment technique, and the alteration technique to extend to fixed-size training sets. While such techniques may be new in learning theory, they are standard for characterizing asymptotic probabilities on graphs, which quasimetrics are often analyzed as [23, 113].

To provide more intuition on why these techniques are useful here, we note that the construction of a training set of pairs is essentially like constructing an Erdős-Rényi random graph on n^2 vertices. Erdős-Rényi (undirected) random graphs come in two kinds:

- Uniformly sampling a fixed number of m edges;
- Adding an edge between each pair with probability p, decided independently.

The latter, due to its independent decisions, is often much easy to analyze and preferred by many. The alteration technique (that we used in the proof) is also a standard way to transfer a result on a random graph of the latter type, to a random graph of the former type [16]. Readers can refer to [3, 16, 41] for more in-depth treatment of these topics.

Generalization to other transforms. The core of this construction only relies on the ability to swap (concatenated) inputs between $(x, y) \leftrightarrow (x, y')$ and between $(y, w) \leftrightarrow (y, w')$ via a transform. For instance, here the orthogonal transforms satisfy this requirement on one-hot features. Therefore, the result can also be generalized to other transforms and features with the same property. Our stated theorem focuses on orthogonal transforms because they correspond to several common learning algorithms (see Lemma 3.4.5). If a learning algorithm is equivariant to some other transform family, it would be meaningful to generalize this result to that transform family, and obtain a similar negative result. We leave such extensions as future work.

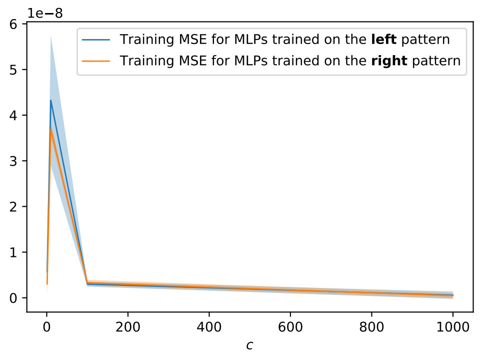
Corollary of Distortion and Violation for Unconstrained MLPs

Corollary B.2.3 (Distortion and Violation of Unconstrained MLPs). Let $(f_n)_n$ be an arbitrary sequence of desired violation values. There is an infinite collection of quasimetric spaces $((\mathcal{X}_n, d_n))_{n=1,2,\dots}$ with $|\mathcal{X}_n| = n$, $\mathcal{X}_n \subset \mathbb{R}^n$ such that MLP trained with squared loss in NTK regime converges to a function $\hat{d}$ that either

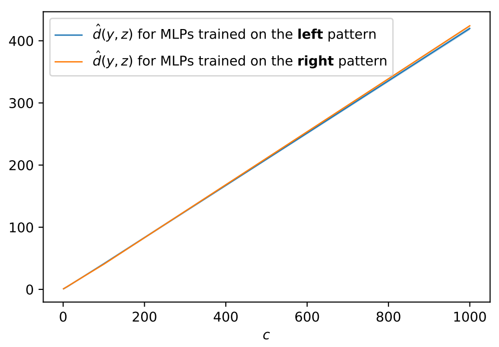
- fails non-negativity, or
- $\mathsf{vio}(\hat{d}) \geq f_n$,

with probability $1/2 - o(1)$ over the random training set S of size m, as long as S does not contain almost all pairs $1 - m/n^2 = \omega(n^{-1/3})$, and does not only include few pairs $m/n^2 = \omega(n^{-1/2})$.

Proof of Corollary B.2.3. This follows directly from Theorem 3.4.6 and standard NTK convergence results obtained from the kernel regression optimality and the positive-definiteness of the NTK. In particular, Proposition 2 of [83] claims that the NTK is positive-definite when restricted to a hypersphere. Since the construction in proof of



(a) Training losses for varying c. Note the scale of the vertical axis.



(b) Prediction on heldout pair $\hat{d}(y, z)$ for varying c.

Figure B-2: Training unconstrained MLPs on the toy failure construction discussed in Section 3.4.2 (reproduced as Figure B-1). Two patterns in the construction have different constraints on distance of the heldout pair (y, z). Plots show mean and standard deviations over 5 runs. **Left:** All training conclude with small training error. **Right:** Trained MLPs predict identically for both patterns. Here standard deviation is small compared to mean and thus not very visible.

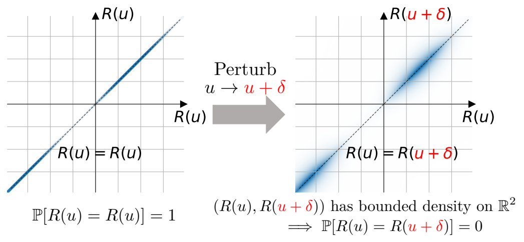
Theorem 3.4.6 uses one-hot features, the input (concatenation of two features) lie on the hypersphere with radius $\sqrt{2}$. Hence, the NTK is guaranteed positive definite. $\square$

Empirical Verification of the Failure Construction

We train unconstrained MLPs on the toy failure construction discussed in Section 3.4.2 (reproduced as Figure B-1). The MLP uses 12-1024-1 architecture with ReLU activations, takes in the concatenated one-hot features, and directly outputs predicted distances. Varying $c \in \{1, 10, 100, 1000\}$, we train the above MLP 5 times on each of the two patterns in Figure B-1, by regressing towards the training distances via MSE loss.

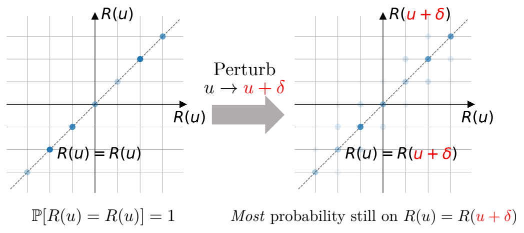
In Figure B-2, we can see that all training runs conclude with small training error, and indeed the trained MLPs predict very similarly on the heldout pair, regardless whether it is trained on the left or right pattern of Figure B-1, which restricts the heldout pair distance differently.

This verifies our theory (Theorem 3.4.6 and Corollary B.2.3) that algorithms equivariant to orthogonal transforms (including MLPs in NTK regime) cannot distinguish these two cases and thus must fail on one of them.



R(u)
R(u)
R(u) = R(u)
Perturb
$u \to u + \delta$
R(u + δ)
R(u)
R(u) = R(u + δ)
$\mathbb{P}[R(u) = R(u)] = 1$
$(R(u), R(u+\delta))$ has bounded density on $\mathbb{R}^2$
$\implies \mathbb{P}[R(u) = R(u+\delta)] = 0$

(a) Continuous-valued stochastic process.



R(u)
R(u)
R(u) = R(u)
Perturb
$u \to u + \delta$
R(u + δ)
R(u)
R(u) = R(u + δ)
$\mathbb{P}[R(u) = R(u)] = 1$
Most probability still on $R(u) = R(u+\delta)$

(b) Discrete-valued stochastic process.

Figure B-3: Bivariate distributions from different stochastic processes. **Left:** In a continuous-valued process (where $(N_\theta, N_{\theta'})$ has bounded density if $\theta \neq \theta'$), perturbing one $\theta \to \theta + \epsilon$ leaves $\mathbb{P}\left[N_\theta = N_{\theta+\epsilon}\right] = 0$. Then one of $\mathbb{P}\big[N_\theta \leq N_{\theta+\epsilon}\big]$ and $\mathbb{P}\big[N_{\theta+\epsilon} \leq N_\theta\big]$ must be far away from 1 (as they sum to 1), breaking differentiability at either $\mathbb{P}\left[N_\theta \leq N_\theta\right] = 1$ or $\mathbb{P}\left[N_{\theta+\epsilon} \leq N_{\theta+\epsilon}\right] = 1$. **Right:** For discrete-valued processes, most probability can still be left on $N_\theta = N_{\theta+\epsilon}$ and thus do not break differentiability.

B.3 Proofs and Discussions for Section 3.5: Poisson Quasimetric Embeddings (PQEs)

B.3.1 Non-differentiability of Continuous-Valued Stochastic Processes

In this section we formalize the argument presented in Section 3.5.3 to show why continuous-valued stochastic processes lead to non-differentiability. Figure B-3 also provides a graphical illustration of the general idea.

Proposition B.3.1 (Quasimetric Embeddings with Continuous-Valued Stochastic Processes are not Differentiable). Consider any $\mathbb{R}^k$-valued stochastic process $\{R(u)\}_{u \in \mathbb{R}^d}$ such that $u \neq u' \implies \mathbb{P}\left[R(u) = R(u')\right] < c$ for some universal constant $c < 1$. Then $\mathbb{P}\left[R(u) \leq R(u')\right]$ is not differentiable at any $u = u'$.

Proof of Proposition B.3.1. Assume that the quantity is differentiable. Then it must be continuous in u and v.

We will use the (ϵ, δ)-definition of continuity.

At any $u \in \mathbb{R}^d$, consider small $\epsilon \in (0, \frac{1-c}{3})$. By continuity, since

$$\mathbb{P}\left[R(u) \leq R(u)\right] = \mathbb{P}\left[R(u + \delta) \leq R(u + \delta)\right] = 1 \tag{B.129}$$

we can find $\epsilon \in \mathbb{R}^d$ such that

$$\mathbb{P}\left[R(u) \leq R(u+\delta)\right] \geq 1-\epsilon \tag{B.130}$$

$$\mathbb{P}\left[R(u+\delta) \leq R(u)\right] \geq 1-\epsilon. \tag{B.131}$$

However, by assumption, $\mathbb{P}\left[R(u) = R(u+\delta)\right] < c$. Therefore,

$$\mathbb{P}\left[R(u) \leq R(u+\delta)\right] \geq 1-\epsilon \tag{B.132}$$

$$\mathbb{P}\left[R(u+\delta) < R(u)\right] \geq 1-\epsilon-c, \tag{B.133}$$

which implies

$$1 = \mathbb{P}\left[R(u) \leq R(u+\delta)\right] + \mathbb{P}\left[R(u+\delta) < R(u)\right] \geq 2-2\epsilon-c \geq \frac{5}{3} - \frac{2}{3}c > 1. \tag{B.134}$$

By contradiction, the quantity must not be differentiable at any $u = u'$. □

B.3.2 PQE-GG: Gaussian-based Measure and Gaussian Shapes

In Section 3.5.1, we presented the following PQE-LH formulation for Lebesgue measures and half-lines:

$$d_z^{\textsf{PQE-LH}}(u, v) \triangleq \sum_i \alpha_i \cdot \Big(1 - \exp\big(-\sum_j (u_{i,j} - v_{i,j})^+\big)\Big). \tag{??}$$

Here, $u_{i,j}$ and $v_{i,j}$ receive zero gradient when $u_{i,j} \leq v_{i,j}$.

Gaussian shapes parametrization. We therefore consider a set parametrization where no one set is entirely contained in a different set— the regions regions $\subset \mathbb{R}^2$ between an axis and a 1D Gaussian density function of fixed variance $\sigma^2_{\textsf{shape}} = 1$. That is, for each given $u \in R$, we consider sets

$$A_{\mathcal{N}}(\mu) \triangleq \{(a, b) \colon b \in [0, f_{\mathcal{N}}(a; \mu, 1)]\}, \tag{B.135}$$

where $f_\mathcal{N}(b; \mu, \sigma^2)$ denotes the density of 1D Gaussian $\mathcal{N}(\mu, \sigma^2)$ with mean μ and variance σ^2 evaluated at b. Since the Gaussian density function have unbounded support, these sets, which are translated versions of each other, never have one set fully contained in another. For latent $u \in \mathbb{R}^{h \times k}$ reshaped as 2D, our set parametrizations are,

$$u \to A_{i,j}(u) \triangleq A_\mathcal{N}(u_{i,j}), \qquad i \in [h], j \in [k]. \quad \text{(B.136)}$$

A Gaussian-based measure. These subsets of $\mathbb{R}^2$ always have Lebesgue measure 1, which would make PQE symmetrical (if used with a (scaled) Lebesgue measure). Thus, we use an alternative $\mathbb{R}^2$ measure given by the product of a $\mathbb{R}$ Lebesgue measure on the b-dimension (i.e., dimension of the function value of the Gaussian density) and a $\mathbb{R}$ Gaussian measure on the a-dimension (i.e., dimension on the input of the Gaussian density) centered at 0 with *learnable* variances $(\sigma^2_{\mathsf{measure}})_{i,j}$. To avoid being constrained by the bounded total measure of 1, we also optimize learnable positive scales $c_{i,j} > 0$. Hence, the each Poisson process has a mean measure as the product of a $\mathbb{R}$ Lebesgue measure and a $\mathbb{R}$ Gaussian with learnable standard deviation, then scaled with a learnable scale.

Note that the Gaussian measure should not be confused with the Gaussian shape. Their parameters also are fully independent with one another.

Computing measures of Gaussian shapes and their intersections. The intersection of two such Gaussian shapes is formed by two Gaussian tail shapes, reflected around the middle point of the two Gaussian means (since they have the same standard deviation $\sigma^{\mathsf{shape}} = 1$). Hence, it is sufficient to describe how to integrate a Gaussian density on a Gaussian measure over an interval. Applying this with different intervals would give the measure of the intersection, and the measures of the two Gaussian shapes. Omit indices i, j for clarity. Formally, we integrate the Gaussian density $f_\mathcal{N}(a; u, \sigma^2_{\mathsf{shape}})$ over the centered Gaussian measure with variance $\sigma^2_{\mathsf{measure}}$, which has density $f_{\mathcal{N}}(a; 0, \sigma^2_{\mathsf{measure}})$:

$$\int c \cdot f_{\mathcal{N}}(a; u, \sigma^2_{\mathsf{shape}}) f_{\mathcal{N}}(a; 0, \sigma^2_{\mathsf{measure}})\, \mathrm{d}a, \tag{B.137}$$

which is also another Gaussian integral (e.g., considered as integrating the product measure along the a line of the form $y = x+u$). After standard algebraic manipulations (omitted here), we obtain

$$\int c \cdot f_{\mathcal{N}}(a; u, \sigma^2_{\mathsf{shape}}) f_{\mathcal{N}}(a; 0, \sigma^2_{\mathsf{measure}})\, \mathrm{d}a \tag{B.138}$$

$$= \frac{c \cdot \exp\left(-u^2/\sigma^2_{\mathsf{total}}\right)}{\sqrt{2\pi\sigma^2_{\mathsf{total}}}} \int f_{\mathcal{N}}\left(a; u\frac{\sigma^2_{\mathsf{measure}}}{\sigma^2_{\mathsf{total}}}, \frac{\sigma^2_{\mathsf{shape}}\sigma^2_{\mathsf{measure}}}{\sigma^2_{\mathsf{total}}}\right) \mathrm{d}a, \tag{B.139}$$

for

$$\sigma^2_{\mathsf{total}} \triangleq \sigma^2_{\mathsf{shape}} + \sigma^2_{\mathsf{measure}}. \tag{B.140}$$

This can be easily evaluated using statistical computing packages that supports computing the error function and/or Gaussian CDF. Moreover, this final form is also readily differentiable with standard gradient formulas. To summarize,

- each set $A(u)$ has total measure

$$\frac{c}{\sqrt{2\pi\sigma^2_{\mathsf{total}}}} \exp\left(-u^2/\sigma^2_{\mathsf{total}}\right); \tag{B.141}$$

- the intersection of $A(v)$ and $A(u_2)$, for $v \le u_2$ has measure

$$\frac{c \cdot \exp\left(-u_2^2/\sigma^2_{\mathsf{total}}\right)}{\sqrt{2\pi\sigma^2_{\mathsf{total}}}} \int_{-\infty}^{\frac{v+u_2}{2}} f_{\mathcal{N}}\left(a; u_2\frac{\sigma^2_{\mathsf{measure}}}{\sigma^2_{\mathsf{total}}}, \frac{\sigma^2_{\mathsf{shape}}\sigma^2_{\mathsf{measure}}}{\sigma^2_{\mathsf{total}}}\right) \mathrm{d}a \tag{B.142}$$

$$+ \frac{c \cdot \exp\left(-v^2/\sigma^2_{\mathsf{total}}\right)}{\sqrt{2\pi\sigma^2_{\mathsf{total}}}} \int_{\frac{v+u_2}{2}}^{+\infty} f_{\mathcal{N}}\left(a; v\frac{\sigma^2_{\mathsf{measure}}}{\sigma^2_{\mathsf{total}}}, \frac{\sigma^2_{\mathsf{shape}}\sigma^2_{\mathsf{measure}}}{\sigma^2_{\mathsf{total}}}\right) \mathrm{d}a. \tag{B.143}$$

Interpretation and representing any total order. Consider two Gaussian shapes $A(v)$ and $A(u_2)$. Note that the Gaussian-based measure μ_{Gaussian} is symmetric around and centered at 0. Therefore,

$$|v| < |u_2| \implies \mu_{\mathsf{Gaussian}}(A(v)) > \mu_{\mathsf{Gaussian}}(A(u_2)) \tag{B.144}$$

$$\implies \mu_{\mathsf{Gaussian}}(A(v) \setminus A(u_2)) > \mu_{\mathsf{Gaussian}}(A(u_2) \setminus A(v)). \tag{B.145}$$

Moreover, scaling the rates of a Poisson makes it more concentrated (as a Poisson's mean grows as the square of its standard deviation) so that $\lim_{c\to\infty} \mathbb{P}\left[\mathrm{Pois}(c\mu_1) \leq \mathrm{Pois}(c\mu_2)\right] = \mathbf{1}_{\mu_1<\mu_2}$ for $\mu_1 \neq \mu_2$. Then any total order can be represented as the limit of a Poisson process with Gaussian shapes, with the shapes' having their means arranged according to the total order, as the scale on the Gaussian-based measure grows to infinity.

B.3.3 Theoretical Guarantees for PQEs

Theorem 3.5.2 (Distortion and violation of PQEs). Under the assumptions of Section 3.4, *any* quasimetric space with size n and treewidth t admits a PQE-LH and a PQE-GG with distortion $\mathcal{O}(t \log^2 n)$ and violation 1, with an expressive encoder (e.g., a ReLU network with ≥ 3 layers and polynomial width).

In Section 3.5.4, we presented the above theoretical distortion and violation guarantees for PQE-LH and PQE-GG. Furthermore, we commented that the same guarantees apply to more generally to PQEs satisfying a mild condition. Here, we first precisely describe this condition, show that PQE-LH and PQE-GG do satisfy it, state and prove the general result, and then show the above as a straightforward corollary.

The Concentration Property

Recall that PQEs are generally defined with measures μ and set parametrizations A as

$$d_z^{\mathsf{PQE}}(u, v; \mu, A, \alpha) \triangleq \sum_i \alpha_i \cdot \mathbb{E}_{\pi_z \sim \Pi_z^{\mathsf{PQE}}(\mu_i, A_i)}\left[\pi_z(u, v)\right], \tag{3.14}$$

where

$$\mathbb{E}_{\pi_z \sim \Pi_z^{\mathsf{PQE}}(\mu, A)}[\pi_z(u, v)] \triangleq 1 - \prod_j \mathbb{P}\left[N_j(A_j(u)) \leq N_j(A_j(v))\right]. \tag{3.13}$$

Because the measures μ and set parametrizations A themselves may have parameters (e.g., as in PQE-GG), we consider them as classes of PQEs. E.g., PQE-GG is a class of PQEs such that the μ is the specific Gaussian-based form, and A is the specific Guassian-shape.

Definition B.3.2 (Concetration Property of PQEs)**.** Consider a PQE class with h mixtures of quasipartition distributions, each from k Poisson processes. We say that it has concentration property if it satisfies the following. Consider any finite subset of $\mathcal{X}' \subset \mathcal{X}$, and arbitrary function $g \colon \mathcal{X} \to \mathbb{R}^{h \times k}$. There exists a sequence of $((f^{(n)}, \mu^{(n)}, A^{(n)})_n$ such that

- $f^{(n)} \colon \mathcal{X}' \to \mathbb{R}^d$,
- $\mu^{(n)}, A^{(n)}$ are valid members of this PQE,
- $\mathbb{E}_{\pi_z \sim \Pi_z^{\mathsf{PQE}}(\mu_i, A_i)} \left[\pi_z(f^{(n)}(x'), f^{(n)}(y')) \right]$ uniformly converges to $1 - \prod_j \mathbf{1}_{g(x)_{i,j} \leq g(y)_{i,j}}$, over all mixtures i and pairs $x, y \in \mathcal{X}'$.

A sufficient condition. It suffices to make the probabilities

$$(x, y, i, j) \to \mathbb{P}\left[N_j(A_j(u)) \leq N_j(A_j(v)) \right], \tag{B.146}$$

along some PQE sequence uniformly converge to the indicators

$$(x, y, i, j) \to \mathbf{1}_{g(x')_{i,j} \leq g(y')_{i,j}}. \tag{B.147}$$

This is sufficient since product of bounded functions is uniformly convergent, if each function is. Both statements below together form **a sufficient condition** for Equation (B.146) to uniformly converge to Equation (B.147):

1. For any g, there exists a specific PQE of this class satisfying
    - Measures (of set differences) are consistent with g with some margin $\epsilon > 0$:

$\forall i \in [h], j \in [k], x \in \mathcal{X}', y \in \mathcal{X}'$,

$$\begin{aligned}
& g(x)_{i,j} < g(y)_{i,j} \\
\iff & \mu_{i,j}(A_{i,j}(f(x)) \setminus A_{i,j}(f(y))) + \epsilon < \mu_{i,j}(A_{i,j}(f(y)) \setminus A_{i,j}(f(x))) \\
& g(x)_{i,j} = g(y)_{i,j} \\
\iff & \mu_{i,j}(A_{i,j}(f(x)) \setminus A_{i,j}(f(y))) = \mu_{i,j}(A_{i,j}(f(y)) \setminus A_{i,j}(f(x))) = 0.
\end{aligned}$$

- Either of the following:
  - One side must be zero: $\forall i \in [h], j \in [k], x \in \mathcal{X}, y \in \mathcal{X}$,

$$(\mu_{i,j}(A_{i,j}(f(x)) \setminus A_{i,j}(f(y)))) \, (\mu_{i,j}(A_{i,j}(f(y)) \setminus A_{i,j}(f(x)))) = 0, \tag{B.148}$$

  - Max measure is bounded by some constant $c > 0$:

$$\max_{x,y,i,j} \mu_{i,j}(A_{i,j}(f(x)) \setminus A_{i,j}(f(y))) \leq c. \tag{B.149}$$

2. For any given specific PQE of this class, for any positive scale $d > 0$, there is another PQE (with same formulation) whose measures (of set differences) equal exactly those of the given PQE scaled by d.

We now show that this is a sufficient condition. Note that a Poisson distribution has standard deviation equal to square root of its mean. This means that as we scale the rate of a Poisson, it becomes more concentrated. Applying to Poisson race probability, we have, for $0 \leq \mu_1 + \epsilon < \mu_2$,

- one direction of Poisson race probability:

$$\begin{aligned}
&\mathbb{P}\left[\mathrm{Pois}(d\cdot\mu_1) \le \mathrm{Pois}(d\cdot\mu_2)\right] && \text{(B.150)}\\
\ge\ & \mathbb{P}\left[|\mathrm{Pois}(d\cdot\mu_2) - \mathrm{Pois}(d\cdot\mu_1) - d(\mu_2-\mu_1)| \le d(\mu_2-\mu_1)\right] && \text{(B.151)}\\
\ge\ & 1 - \frac{\mu_1+\mu_2}{d(\mu_2-\mu_1)^2} && \text{(B.152)}\\
\ge\ & \begin{cases} 1-\frac{2}{d\epsilon} & \text{if } \mu_1 = 0\\ 1-\frac{2c}{d\epsilon^2} & \text{if } \mu_2 < c; \end{cases} && \text{(B.153)}
\end{aligned}$$

- the other direction of Poisson race probability:

$$\begin{aligned}
&\mathbb{P}\left[\mathrm{Pois}(d\cdot\mu_2) \le \mathrm{Pois}(d\cdot\mu_1)\right] && \text{(B.154)}\\
\le\ & \mathbb{P}\left[|\mathrm{Pois}(d\cdot\mu_2) - \mathrm{Pois}(d\cdot\mu_1) - d(\mu_2-\mu_1)| \ge d(\mu_2-\mu_1)\right] && \text{(B.155)}\\
\le\ & \frac{\mu_1+\mu_2}{d(\mu_2-\mu_1)^2} && \text{(B.156)}\\
\le\ & \begin{cases} \frac{2}{d\epsilon} & \text{if } \mu_1 = 0\\ \frac{2c}{d\epsilon^2} & \text{if } \mu_2 < c. \end{cases} && \text{(B.157)}
\end{aligned}$$

Therefore, applying to scaled versions of the PQE from Item 1 above, we have thus obtained the desired sequence, where Equation (B.146) uniformly converges to Equation (B.147) with rate $\mathcal{O}(1/d)$.

Lemma B.3.3. PQE-LH and PQE-GG both have the concentration property.

Proof of Lemma B.3.3. We show that both classes satisfy the above sufficient condition.

- PQE-LH: Lebesgue measure λ and half-lines.

  WLOG, since $\mathcal{X}$ is countable, we assume that g satisfies

$$g(x)_{i,j} \ne g(y)_{i,j} \implies |g(x)_{i,j} - g(y)_{i,j}| > 1, \qquad \forall i \in [h], j \in [k], x \in \mathcal{X}', y \in \mathcal{X}'. \tag{B.158}$$

The encoder in Item 1 above $f\colon \mathcal{X} \to \mathbb{R}^{h\times k}$ can simply be g. We then have

$$\mu_{i,j}(A_{i,j}(f(y))\backslash A_{i,j}(f(x))) = \mathrm{Leb}((-\infty, g(y)]\backslash(-\infty, g(x)]) = (g(y)_{i,j} - g(x)_{i,j})^+. \tag{B.159}$$

This ensures that one side is always zero. Furthermore, scaling can be done by simply scaling the encoder f. Hence, PQE-LH satisfies this constraint.

- PQE-GG: Gaussian-based measure and Gaussian shapes (see Appendix B.3.2).

  Because $\mathcal{X}'$ is finit, we can have positive constant margin for the PQE requirements in Item 1. (Infinite $\mathcal{X}'$ does not work because the total measure is finite (for a specific PQE-GG with specific values of the scaling).) Concretely, we satisfy both requirements via

  - in descending order of $g(\cdot)_{i,j}$ we assign Gaussian shapes increasingly further from the origin;
  - scaling comes from that we allow scaling the Gaussian-based measure.

  Hence, PQE-GG satisfies this constraint for finite $\mathcal{X}$.

□

A General Statement

We now state the general theorem for PQEs with the above concentration property.

Theorem B.3.4 (Distortion and violation of PQEs (General)). Consider any PQE class with the concentration property. Under the assumptions of Section 3.4, *any* quasimetric space with size n and treewidth t admits such a PQE with distortion $\mathcal{O}(t\log^2 n)$ and violation 1, with an expressive encoder (e.g., a ReLU network with ≥ 3 hidden layers, $\mathcal{O}(n)$ hidden width, and $\mathcal{O}(n^2)$ quasipartition distributions, each with $\mathcal{O}(n)$ Poisson processes.).

Before proving this more general theorem, let us extend a result from Mémoli et al. [113].

Lemma B.3.5 (Quasimetric Embeddings with Low Distortion; Adapted from Corollary 2 in Mémoli et al. [113]). Let $M = (X, d)$ be a quasipseudometric space with treewidth t, and $n = |X|$. Then M admits an embedding into a convex combination (i.e., scaled mixture) of $\mathcal{O}(n^2)$ quasipartitions with distortion $\mathcal{O}(t \log^2 n)$.

Proof of Lemma B.3.5. The distortion bound is proved in Corollary 2 in [113], which states that any quasipseudometric space with n elements and t treewidth admits an embedding into a convex combination of quasipartitions with distortion $\mathcal{O}(t \log^2 n)$.

To see that n^2 quasipartitions suffice, we scrutinize their construction of quasipartitions in Algorithm 2 of [113], reproduced below as Algorithm 2.

Algorithm 2 Random quasipartition of a graph with bounded treewidth. Algorithm 2 of [113].

```
Input: A digraph $G$ of treewidth $t$, a hierarchical tree of separators of $G$ $(H, f)$ with width $t$, and $r > 0$.
Output: A random $r$-bounded quasipartition $R$.
  Initialization: Set $G^* = G$, $H^* = H$ and $R = E(G)$. Perform the following recursive algorithm on $G^*$ and $H^*$.
  Step 1. Pick $z \in [0, r/2]$ uniformly at random.
  Step 2. If $|V(G^*)| \leq 1$,terminate the current recursive call. Otherwise pick the set of vertices $K = G^*$. Let $H_1, \ldots, H_m$ be the sub-trees of $H^*$ below $\mathsf{root}(H^*)$ that are hierarchical trees of separators of $C_1, \ldots, C_m$ respectively.
  Step 3. For all $(u, v) \in E(G^*)$ remove $(u, v)$ from $R$ if one of the following holds:
      (a) $d_G(u, x) > z$ and $d_G(v, x) \leq z$ for some vertex $x \in K$.
      (b) $d_G(x, v) > z$ and $d_G(x, u) \leq z$ for some vertex $x \in K$.
  Step 4. For all $i \in \{1, \ldots, m\}$ perform a recursive call of Steps 2-4 setting $G^* = G^*[C_i]$ and $H^* = H_i$.
  Step 5. Once all branches of the recursive terminate, enforce transitivity on $R$: For all $u, v, w \in V(G)$ if $(u, v) \in R$ and $(v, w) \in R$, add $(u, w)$ to $R$.
```

Many concepts used in Algorithm 2 are not relevant for our purpose (e.g., r-bounded quasipartition). Importantly, we observe that for a given quasimetric space, the produced quasipartition is entirely determined by the random choice of z in Step 1, which is only used to compare with distance values between node pairs. Note that there are n^2 node pairs, whose minimum distance is exactly 0 (i.e., distance from a node to itself). Since $z \geq 0$, there are at most n^2 choices of z that lead to at most n^2 different quasipartitions, for all possible values of r.

The construction used to prove Corollary 2 of [113] uses exactly quasipartitions given by this algorithm. Therefore, the lemma is proved. $\square$

Lemma B.3.5 essentially proves the first half of Theorem B.3.4. Before proving the full Theorem B.3.4, we restate the following result from [75], which gives us a bound on how many total orders are needed to represent a general partial order (i.e., quasipartition).

Theorem B.3.6 (Hiraguchi's Theorem [75, 14]). Let (X, P) be a partially ordered set such that $|X| \geq 4$. Then there exists a mapping $f\colon X \to \mathbb{R}^{\lfloor |X|/2 \rfloor}$ such that

$$\forall x, y \in X, \qquad xPy \iff f(x) \leq f(y) \text{ coordinate-wise}. \tag{B.160}$$

Proof of Theorem B.3.4. It immediately follows from Lemma B.3.5 and Theorem B.3.6 that any quasimetric space with n elements and treewidth t admits an embedding with distortion $\mathcal{O}(t \log^2 n)$ into a convex combination of n^2 quasipartitions, each represented with an intersection of $\mathcal{O}(n)$ total orders.

Because the PQE class has concentration property, for any finite quasimetric space, we can simply select a PQE that is close enough to the desired convex combination of n^2 quasipartitions, to obtain distortion $\mathcal{O}(t \log^2 n)$. Since each Poisson process in PQE takes a constant number of latent dimensions, we can have such a PQE with $\mathcal{O}(n^3)$-dimensional latents and n^2 quasipartition distributions.

It remains only to prove that we can compute such required latents using the described architecture.

Consider any $x \in \mathcal{X} \subset \mathbb{R}^d$. Since $\mathcal{X}$ is finite, we can always find direction $u_x \in \mathbb{R}^d$ such that $\forall y \in \mathcal{X} \setminus \{x\}$, $y^\mathsf{T} u_x \neq x^\mathsf{T} u_x$. That is, x has a unique projection onto u_x.

Therefore, we can have $c, b_+, b_- \in \mathbb{R}$ such that

$$c \cdot u_x^\mathsf{T} x + b_+ = 1 \tag{B.161}$$

$$-c \cdot u_x^\mathsf{T} x + b_- = 1, \tag{B.162}$$

but for $y \in \mathcal{X} \setminus \{x\}$, we have, for some $a > 0$, either

$$c \cdot u_x^\mathsf{T} y + b_+ = -a \tag{B.163}$$

$$-c \cdot u_x^\mathsf{T} y + b_- = a + 2, \tag{B.164}$$

or

$$c \cdot u_x^\mathsf{T} y + b_+ = a + 2 \tag{B.165}$$

$$-c \cdot u_x^\mathsf{T} y + b_- = -a. \tag{B.166}$$

Then, consider computing two of the first layer features as, on input z,

$$[\mathrm{ReLU}(c \cdot u_x^\mathsf{T} z + b_+) \quad \mathrm{ReLU}(-c \cdot u_x^\mathsf{T} z + b_-)], \tag{B.167}$$

which, if $z = x$, is $[1, 1]$; if $z \neq x$, is either $[0, 2 + a]$ or $[2 + a, 0]$, for some $a > 0$.

Then, one of the second layer features may sum these two features and threshold it properly would single out x, i.e., activate only when input is x.

After doing this for all $x \in \mathcal{X}$, we obtain an n-dimensional second layer feature space that is just one-hot features.

The third layer can then just be a simple embedding look up, able to represent any embedding, including the one allowing a PQE to have distortion $\mathcal{O}(t \log n)$, as described above.

Because quasimetric embeddings naturally have violation 1, this concludes the proof. $\square$

Proof of Theorem 3.5.2: Distortion and violation of PQEs

Proof of Theorem 3.5.2. Lemma B.3.3 and Theorem B.3.4 imply the result. To see that polynomial width is sufficient, note that the hidden width are polynomial by Theorem B.3.4, and that the embedding dimensions needed to represent each of the $\mathcal{O}(n^3)$ Poisson processes is constant 1 in both PQE-LH and PQE-GG. Hence the latent space is also polynomial. This concludes the result. □

Discussions

Dependency on $\log n$. $\log n$ dependency frequently occurs in distortion results. Perhaps the most well-known ones are Bourgain's Embedding Theorem [18] and the Johnson-Lindenstrauss Lemma [85], which concern *metric* embeddings into Euclidean spaces.

Dependency on treewidth t. Treewidth t here works as a complexity measure of the quasimetric. We will use a simple example to illustrate why low-treewidth is easy. Consider the extreme case where the quasimetric is the shortest-path distance on a tree, whose each edge is converted into two opposing directed ones and assigned arbitrary non-negative weights. Such a quasimetric space has treewidth 1 (see Definition 3.2.2). On a tree,

1. the shortest path between two points is fixed, regardless of the weights assigned,
2. for each internal node u and one of its child c, the followings are quasipartitions:

$$d'_{01}(x, y) \triangleq \mathbf{1}_{\text{shortest path from } x \text{ to } y \text{ passes } (u, c)}$$
$$d''_{01}(x, y) \triangleq \mathbf{1}_{\text{shortest path from } x \text{ to } y \text{ passes } (c, u)}.$$

Hence it can be *exactly* represented as a convex combination of quasipartitions. However, both of observations becomes false when the graph structure becomes more complex (higher treewidth) and the shortest paths can are less well represented as tree paths of the tree composition.

Comparison with unconstrained MLPs. Theorem B.3.4 requires a poly-width encoder to achieve low distortion. This is comparable with deep unconstrained MLPs trained in NTK regime, which can reach 0 training error (distortion 1 on training set) in the limit but also requires polynomial width [5].

Quasipseudometrics and infinite distances. Theorem B.3.4 relies on our assumptions that $(\mathcal{X}, d)$ is not a quasipseudometric space and has all finite distances. In fact, if we allow a PQE to have infinite convex combination weights, it can readily represent quasipseudometric spaces with infinite distances. Additionally, PQE can still well approximate the quasimetric space with infinities replaced with any sufficiently large finite value (e.g., larger than the maximum finite distance). Thus, this limit is generally not important in practice (e.g., learning γ-discounted distances), where a large value and infinity are usually not treated much differently.

Optimizing quasimetric embeddings. From Theorem B.3.4, we know that optimizing PQEs over the training set S w.r.t. distortion achieves low distortion (and optimal violation by definition). While directly optimizing distortion (or error on log distance or distance ratios, equivalently) seems a valid choice, such objectives do not always train stably in practice, with possible infinities and zeros. Often more stable losses are used, such as MSE over raw distances or γ-discounted distances γ^d, for $\gamma \in (0, 1)$. These objectives do not directly relate to distortion, except for some elementary loose bounds. To better theoretically characterize their behavior, an alternative approach with an average-case analysis might be necessary.

B.3.4 Implementing Poisson Quasimetric Embeddings (PQEs)

Section 3.5.2 mentioned a couple implementation techniques for PQEs. In this section, we present them in full details.

Normalized Measures

Consider a PQE whose each of j expected quasipartitions is defined via k Poisson processes, with set parametrizations $u \to A_{i,j}(u), i \in [h], j \in [k]$. To be robust to the choice of k, we instead use the normalized set parametrizations $A'_{i,j}$'s:

$$A'_{i,j}(u) \triangleq A_{i,j}(u)/k, \qquad\qquad i \in [h], j \in [k]. \tag{B.168}$$

This does not change the PQE's concentration property (Definition B.3.2) or its theoretical guarantees (e.g., Theorems 3.5.2 and B.3.4).

Outputting γ-Discounted Distances

Recall the PQE quasimetric formulation in Equation (3.14), for $\alpha_i \geq 0$, and encoder $f \colon \mathcal{X} \to \mathbb{R}^d$ (with set parametrizations $A_{i,j}$'s and measures $\mu_{i,j}$'s):

$$\hat{d}(x, y) \triangleq \sum_i \alpha_i \left(1 - \prod_j \mathbb{P}\left[\mathrm{Pois}(\mu_{i,j}(A^f_{i,j}(x) \setminus A^f_{i,j}(y)) \leq \mathrm{Pois}(\mu_{i,j}(A^f_{i,j}(y) \setminus A^f_{i,j}(x)) \right] \right), \tag{3.14}$$

where we used shorthands $A^f_{i,j}(x) \triangleq A_{i,j}(f(x))$.

With discount factor $\gamma \in (0, 1)$, we can write the γ-discounted PQE distance as

$$\gamma^{\hat{d}(x,y)} = \prod_i (\underbrace{\gamma^{\alpha_i}}_{\text{a scalar that can take value in any } (0,1)})^{1 - \prod_j \mathbb{P}\left[\mathrm{Pois}(\mu_{i,j}(A^f_{i,j}(x) \setminus A^f_{i,j}(y)) \leq \mathrm{Pois}(\mu_{i,j}(A^f_{i,j}(y) \setminus A^f_{i,j}(x))\right]}. \tag{B.169}$$

Therefore, instead of learning $\alpha_i \in [0, \infty)$, we can learn bases $\beta_i \in (0, 1)$ such and define the γ-discounted PQE distance as

$$\gamma^{\hat{d}(x,y)} \triangleq \prod_i \beta_i^{1 - \prod_j \mathbb{P}\left[\mathrm{Pois}(\mu_{i,j}(A^f_{i,j}(x) \setminus A^f_{i,j}(y)) \leq \mathrm{Pois}(\mu_{i,j}(A^f_{i,j}(y) \setminus A^f_{i,j}(x))\right]}. \tag{B.170}$$

These bases $\beta_i \in (0, 1)$ can be parametrized via a sigmoid transform. Consider quasimetric learning w.r.t. errors on γ-discounted distances (e.g., MSE). Unlike the parametrization with directly learning the convex combination weights α_i's, such a parametrization (that learns the bases β_i's) does not explicitly include γ and thus can potentially be more stable for a wider range of γ choices.

Initialization. Consider learning bases β_i's via a sigmoid transform: learning b_i and defining $\beta_i \triangleq \sigma(b_i)$. We must take care in initializing these b_i's so that $\sigma(b_i)$'s are not too close to 0 or 1, since we take a product of powers with these bases. To be robust to different h numbers of quasipartition distributions, we initialize the each b_i to be from the uniform distribution

$$\mathcal{U}[\sigma^{-1}(0.5^{2/h}), \sigma^{-1}(0.75^{2/h})], \tag{B.171}$$

which means that, at initialization,

$$\prod_{i \in [h]} \beta_i^{0.5} = \prod_{i \in [h]} \sigma(b_i)^{0.5} \in [0.5, 0.75], \tag{B.172}$$

providing a good range of initial outputs, assuming that the exponents (expected outputs of quasipartition distributions) are close to 0.5. Alternatively, b_i's maybe parametrized by a deep linear network, a similar initialization is employed. See Appendix B.3.4 below for details.

Learning Linear/Convex Combinations with Deep Linear Networks

Deep linear networks have the same expressive power as regular linear models, but enjoy many empirical and theoretical benefits in optimization [135, 126, 80]. Specifically, instead of directly learning a matrix $\in \mathbb{R}^{m \times n}$, a deep linear network (with bias) of l

layers learns a sequence of matrices

$$M_1 \in \mathbb{R}^{m_1 \times n} \tag{B.173}$$

$$M_2 \in \mathbb{R}^{m_2 \times m_1} \tag{B.174}$$

$$\vdots \quad \vdots \tag{B.175}$$

$$M_{l-1} \in \mathbb{R}^{m_{l-1} \times m_{l-2}} \tag{B.176}$$

$$M_l \in \mathbb{R}^{m \times m_{l-1}} \tag{B.177}$$

$$B \in \mathbb{R}^{m \times n}, \tag{B.178}$$

where the linear matrix can be obtained with

$$M_l \; M_{l-1} \dots M_2 \; M_1 + B, \tag{B.179}$$

and we require

$$\min(m_1, m_2, \dots, m_{l-1}) \geq \min(m, n). \tag{B.180}$$

In our case, the convex combination weights for the quasipartition distributions often need to be large, in order to represent large quasimetric distances; in Poisson process mean measures with learnable scales (e.g., the Gaussian-based measure described in Appendix B.3.2), the scales may also need to be large to approximate particular quasipartitions (see Appendix B.3.3).

Therefore, we choose to use deep linear networks to optimize these parameters. In particular,

- **For the convex combination weights for h quasipartition distributions,**
  - When learning the convex combination weights $\{\alpha_i\}_{i \in [h]}$, we use a deep linear network to parametrize a matrix $\in \mathbb{R}^{1 \times h}$ (i.e., a linear map from $\mathbb{R}^h$ to $\mathbb{R}$), which is then viewed as a vector $\in \mathbb{R}^h$ and applied an element-wise square transform $a \to a^2$ to obtain non-negative weights $\alpha \in [0, \infty)^h$;
  - When learning the bases for discounted quasimetric distances β_i's (see Appendix B.3.4), we use a deep linear network to parametrize a matrix

$\in \mathbb{R}^{h \times 1}$, which is then viewed as a vector $\in \mathbb{R}^h$ and applied an element-wise sigmoid transform $a \to \sigma(a)$ to obtain bases $\beta \in (0,1)^h$.

Note that here we parametrize a matrix $\in \mathbb{R}^{h \times 1}$ rather than $\mathbb{R}^{1 \times h}$ as above for α_i's. The reason for this choice is entirely specific to the initialization scheme we use (i.e., (fully-connected layer weight matrix initialization, as discussed below). Here the interpretation of a linear map is no longer true. If we use $\mathbb{R}^{1 \times h}$, the initialization method would lead to the entries distributed with variance roughly $1/n$, which only makes sense if they are then added together. Therefore, we use $\mathbb{R}^{h \times 1}$, which would lead to constant variance.

- **For scales of the Poisson process mean measure, such as PQE-GG,** we consider a slightly different strategy.

  Consider a PQE formulation with $h \times k$ independent Poisson processes, from which we form h quasipartition distributions, each from k total orders parametrized by k Poisson processes. The Poisson processes are defined on sets

$$\{A_{i,j}\}_{i \in [h], j \in [k]}, \tag{B.181}$$

use mean measures

$$\{\mu_{i,j}\}_{i \in [h], j \in [k]}, \tag{B.182}$$

and set parametrizations

$$\{u \to A_{i,j}(u)\}_{i \in [h], j \in [k]}, \tag{B.183}$$

to compute quantities

$$\mu_{i,j}(A_{i,j}(u) \setminus A_{i,j}(v)) \qquad \text{for } u \in \mathbb{R}^d, v \in \mathbb{R}^d, i \in [h], j \in [k]. \tag{B.184}$$

Scaling each mean measure independently. Essentially, adding learnable scales (of mean measures) $w \in [0, \infty)^{h \times k}$ (or, equivalently, $\{w_{i,j} \in [0,\infty)\}_{i,j}$)

gives a scaled set of measures

$$\{w_{i,j} \cdot \mu_{i,j}\}_{i\in[h],j\in[k]}. \tag{B.185}$$

This means that the quantities in Equation (B.184) becomes respectively scaled as

$$w_{i,j} \cdot \mu_{i,j}(A_{i,j}(u) \setminus A_{i,j}(v)) \qquad \text{for } u \in \mathbb{R}^d, v \in \mathbb{R}^d, i \in [h], j \in [k]. \tag{B.186}$$

Convex combinations of *all* measures. However, we can be more flexible here, and allow not just scaling each measure independently, but also *convex combinations of all measures*. Instead of having w as a collection of $h \times k$ scalar numbers $\in [0, \infty)$, we have a collection of $(h \times k)$ vectors each having length $(h \times k)$ (or $h \times k$-shape tensors)

$$\{w_{i,j} \in [0, \infty)^{h\times k}\}_{i\in[h],j\in[k]}, \tag{B.187}$$

and have the quantities in Equation (B.184) respectively scaled and combined as

$$\sum_{i',j'} w_{i,j,i',j'} \cdot \mu_{i',j'}(Ai', j'(u) \setminus A_{i',j'}(v)) \qquad \text{for } u \in \mathbb{R}^d, v \in \mathbb{R}^d, i \in [h], j \in [k]. \tag{B.188}$$

Note that these still are valid Poisson processes for a PQE. Specifically, the new Poisson processes now all use the same set parametrization (as the collection of original ones), with different measures (as different weighted combinations of the original measures). This generalizes the case where each mean measure is scaled independently (as w can be diagonal).

Therefore, we will apply this more general strategy using **convex combinations of *all* measures**.

Similarly to learning the convex combination weights of quasipartition distributions, we collapse a deep linear network into a tensor $\in \mathbb{R}^{h \times k \times h \times k}$, and apply an element-wise square $a \to a^2$, result of which is used as the convex combination weights w to ensure non-negativity.

Initialization. For initializing the matrices $(M_1, M_2, \dots, M_l)$ of a deep linear network (Equation (B.177)), we use the standard weight matrix initialization of fully-connected layers in PyTorch [124]. The bias matrix B (Equation (B.178)) is initialized to all zeros.

When used for learning the bases for discounted quasimetric distances β_i's (as described in Appendix B.3.4), we have a deep linear network parametrizing a matrix $\in \mathbb{R}^{h \times 1}$, initialized in the same way as above (including initializing B as all zeros). Consider the matrix up to before the last one:

$$M^* \triangleq M_{l-1} \cdot M_2 \ M_1 \in \mathbb{R}^{m_{l-1} \times 1}. \tag{B.189}$$

M^* is essentially a projection to be applied on each row of the last matrix $M_l \in \mathbb{R}^{h \times m_{l-1}}$, to obtain b_i (which is then used to obtain bases $\beta_i \triangleq \sigma(b_i)$). Therefore, we simply rescale the M^* subspace for each row of M_l and keep the orthogonal space intact, such that the projections would be distributed according to the distribution specified in Equation (B.171):

$$\mathcal{U}[\sigma^{-1}(0.5^{2/h}), \sigma^{-1}(0.75^{2/h})], , \tag{B.171}$$

which has good initial value properties, as shown in Appendix B.3.4.

Choosing h the Number of Quasipartition Distributions and k the Number of Poisson Processes for Each Quasipartition Distribution

A PQE (class) is defined with $h \times k$ independent Poisson processes with means $\{\mu_{i,j}\}_{i \in [h], j \in [k]}$ along with $h \times k$ set parametrizations $\{A_{i,j}\}_{i \in [h], j \in [k]}$. For k pairs of means and set parametrizations, we obtain a random quasipartition. A mixture (convex combination) of the resulting h random quasipartitions gives the quasimetric. The choices of μ and A are flexible. In this work we explore PQE-LH and PQE-GG as two options, both using essentially the same measure and parametrization across all i, j (up to individual learnable scales). These two instantiations both perform well empirically. In this section we aim to provide some intuition on choosing these two hyperparameters h and k.

h the Number of Quasipartition Distributions Theoretical result Theorem B.3.4 suggest thats, for a quasimetric space with n elements, n^2 quasipartition distributions suffice to learn a low distortion embedding. Since this is a worst-case result, the practical scenario may require much fewer quasipartitions. For instance, Appendix B.3.3 shows that $\mathcal{O}(n)$ quasipartitions is sufficient for any quasimetric space with a tree structure. In our experiments, $h \in [8, 128]$ quasipartition distributions are used.

k the Number of Poisson Processes for Each Quasipartition Distribution (Random Partial Order) It is well-known that such intersection of sufficiently many total orders can represent *any* partial order [157, 75]. This idea is equivalent with the dominance drawing dimension of directed graphs [120], which concerns an order embedding of the vertices to preserve the poset specified by the reachability relation. In this graph theoretical view, several results are known. [43] prove that planar graphs have at most 8 dimension. [120] show that the dimension of any graph with n vertices is at most $\min(w_P, \frac{n}{2})$, where w_P the maximum size of a set of incomparable vertices. A simpler and more fundamental result can be traced to Hiraguchi from 1951:

Theorem B.3.6 (Hiraguchi's Theorem [75, 14]). Let (X, P) be a partially ordered set such that $|X| \geq 4$. Then there exists a mapping $f \colon X \to \mathbb{R}^{\lfloor |X|/2 \rfloor}$ such that

$$\forall x, y \in X, \qquad xPy \iff f(x) \leq f(y) \text{ coordinate-wise}. \tag{B.160}$$

Theorem B.3.6 states that $\frac{n}{2}$ dimensions generally suffice for any poset of size $n \geq 4$.

In our formulation, this means that using $k = \frac{n}{2}$ Poisson processes (giving $\frac{n}{2}$ random total orders) will be maximally expressive. In practice, this is likely unnecessary and sometimes impractical. In our experiments, we choose a small fixed number $k = 4$.

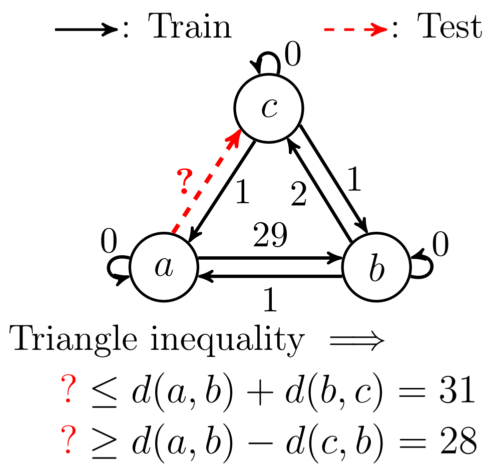

: Train
: Test
0
c
?
1
2
1
0
29
0
a
b
1
Triangle inequality $\Longrightarrow$
$? \leq d(a,b) + d(b,c) = 31$
$? \geq d(a,b) - d(c,b) = 28$

Figure B-4: The 3-element quasimetric space, and the training pairs.Training set contains all pairs except for (a, c). Arrows show quasimetric distances (rather than edge weights of some graph).

B.4 Experiment Settings and Additional Results

Computation power. All our experiments run on a single GPU and finish within 3 hours. GPUs we used include NVIDIA 1080, NVIDIA 2080 Ti, NVIDIA 3080 Ti, NVIDIA Titan Xp, NVIDIA Titan RTX, and NVIDIA Titan V.

B.4.1 Experiments from Section 3.3.2: A Toy Example

In Section 3.3.2 and Figure 3-2, we show experiment results on a simple 3-element quasimetric space.

Quasimetric space. The quasimetric space has 3 elements with one-hot features $\in \mathbb{R}^3$. Thequasimetric and training pairs are shown in Figure B-4.

Unconstrained network. The unconstrained network has architecture 6-128-128-32-1, with ReLU activations.

Metric embedding. The embedding space is 32-dimensional, upon which corresponding metric is applied. The encoder network has architecture 6-128-128-32, with ReLU activations.

Asymmetric dot products. The embedding space is 32-dimensional. The two inputs are encoded with a *different* encoder of architecture 6-128-128-32, with ReLU activations. Then the dot product of the two 32-dimensional vector is taken, which parametrizes a distance estimate

Poisson Quasimetric Embeddings. The embedding space is 32-dimensional, which parametrizes 8 quasimetric distributions, each from 4 independent Poisson processes using (scaled) Lebesgue measure and half-lines. We use deep linear networks, as described in Appendix B.3.4. A deep linear network (without bias) of architecture 8-32-32-1 parametrizes the convex combination weights $\{\alpha_i\}_{i\in[8]}$. Another deep linear network (without bias) of architecture 32-64-64-32 parametrizes convex combination weights of the mean measures $d \in [0, \infty)^{32\times 32}$. Note that these *do not* give many more effective parameters to PQEs as they are equivalent with simple linear transforms.

Optimization. All models are trained w.r.t. MSE on distances with the Adam optimizer [88] with learning rate 0.0003 for 1000 iterations (without mini-batching since the training set has size 8).

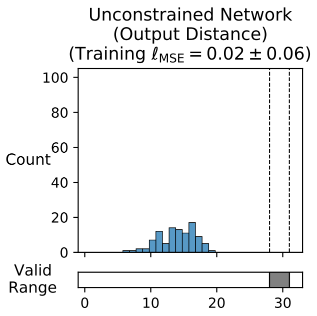
Additional results. Results with additional formulations (together with the ones presented in Figure 3-2) are shown in Figure B-5.

B.4.2 Experiments from Section 4.5: Experiments

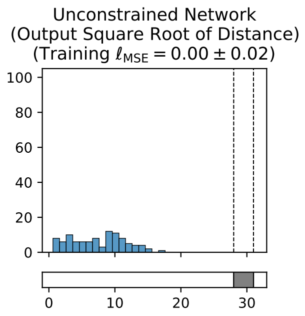
Triangle inequality regularizer. For methods that do not inherently respect triangle inequalities (e.g., unconstrained networks and asymmetrical dot products), we explore training with a regularizer that encourages following these inequalities. By sampling random triplets uniformly over the training set, the regularizer is formulated as,

$$\mathbb{E}_{x,y,z}\big[\max(0, \gamma^{\hat{d}(x,y)+\hat{d}(y,z)} - \gamma^{\hat{d}(x,z)})^2\big], \tag{B.190}$$

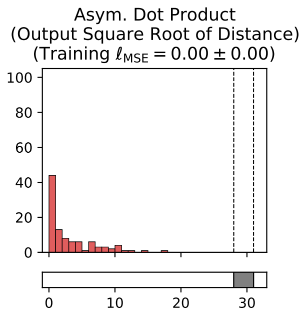
where the γ-discounted terms and the squared form allows easier balancing with the training loss, which, across all experiments, are MSEs on some γ-discounted distances.



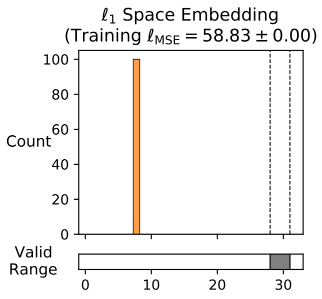
(a) Unconstrained network that directly predicts distance.



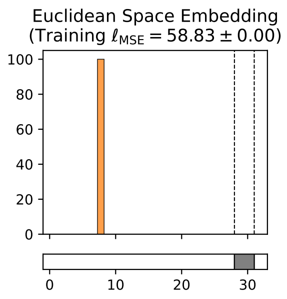
(b) Unconstrained network that predicts distance with a square $a \to a^2$ transform.



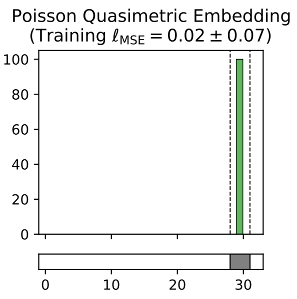
(c) Asymmetrical dot product that predicts distance with a square $a \to a^2$ transform.



(d) Metric embedding into an ℓ_1 space.



(e) Metric embedding into an Euclidean space.



(f) Poisson Quasimetric Embedding specified in Appendix B.4.1.

Figure B-5: Training different formulations to fit training pairs distances via MSE, and using them to predict on the test pair. Plots show distribution of the prediction over 100 runs. Standard deviations of the training error are shown.

PQE settings. Across all experiments of this section, when given an encoder architecture mapping input to an $\mathbb{R}^d$ latent space, we construct PQEs according to the following general recipe, to obtain the two PQEs settings used across all experiments: PQE-LH (PQE with Lebesgue measure and half-lines) and PQE-GG (PQE with Gaussian-based measure and Gaussian shapes, see see Appendix B.3.2):

- (Assuming d is a multiple of 4,) We use $h \triangleq= d/4$ quasipartition distributions, each given by $k \triangleq 4$ Poisson processes;

- A deep linear network (see Appendix B.3.4), is used for parametrizing the convex combination weights $\alpha \in \mathbb{R}^{d/4}$ or the bases $\beta \in \mathbb{R}^{d/4}$ (see Appendix B.3.4), we follow the initialization and parametrization described in Appendix B.3.4, with hidden sizes $[n_{\mathsf{hidden}}, n_{\mathsf{hidden}}, n_{\mathsf{hidden}}]$ (i.e., 4 matrices/layers), where $n_{\mathsf{hidden}} \triangleq \max(64, 2^{1+\lceil \log_2 (d/4) \rceil})$.

- For PQE-GG,

  - The learnable $\sigma^2_{\mathsf{measure}} \in (0, \infty)^d$ (one for each Poisson Process) is achieved by optimizing the log variance, which is initialized as all zeros.
  - The Gaussian-based measures need learnable scales. We use a deep linear network to parametrize the $[0, \infty)^{d \times d}$ weights for the convex combinations of measures, as described in Appendix B.3.4. Similarly, it has hidden sizes $[n_{\mathsf{hidden}}, n_{\mathsf{hidden}}, n_{\mathsf{hidden}}]$ (i.e., 4 matrices/layers), where $n_{\mathsf{hidden}} \triangleq \max(64, 2^{1+\lceil \log_2 d \rceil})$.

Note that the PQEs add only a few extra effective parameters on top of the encoder (d for PQE-LH, and $d + d^2$ for PQE-GG), as the deep linear networks do not add extra effective parameters.

Mixed space metric embedding settings. Across all experiments of this section, when given an encoder architecture mapping input to an $\mathbb{R}^d$ latent space, we construct the metric embedding into mixed space as follows:

- (Assuming d is a multiple of 4,) We use (1) a $(d/2)$-dimensional Euclidean space (2) a $(d/4)$-dimensional ℓ_1 space, and a $(d/4)$-dimensional spherical distance space (without scale).

- Additionally, we optimize three scalar values representing the log weights of the convex combination to mix these spaces.

DeepNorm and WideNorm method overview and parameter count comparison with PQEs. Both DeepNorm and WideNorm parametrize asymmetrical norms. When used to approximate quasimetrics, they are applied as $\hat{d}(x, y) \triangleq f_{\mathsf{AsymNorm}}(f_{\mathsf{Enc}}(x) - f_{\mathsf{Enc}}(y))$, where f_{Enc} is the encoder mapping from data space to an $\mathbb{R}^d$ latent space and f_{AsymNorm} is either the DeepNorm or the WideNorm predictor on that latent space [127].

- **DeepNorm** is a modification from Input Convex Neural Network (ICNN; Amos et al. [4]), with restricted weight matrices and activation functions for positive homogeneity (a requirement of asymmetrical norms), and additional concave function for expressivity.

  For an input latent space of $\mathbb{R}^d$, consider an n-layer DeepNorm with width w (i.e., ICNN output size) and the suggested intermediate MaxReLU activation and MaxMean final aggregation (see [127] for details of these functions). This

DeepNorm predictor $f_{\textsf{DeepNorm}}$ (on latent space) has

$$\begin{aligned}\#\text{parmaters of } f_{\textsf{DeepNorm}} = & \underbrace{n \times (d \times w)}_{U \text{ matrices from input to each layer}} \\ + & \underbrace{(n-1) \times w^2}_{W \text{ matrices between neighboring layer activations}} \\ + & \underbrace{n \times w}_{\text{intermediate MaxReLU activations}} \\ + & \underbrace{w \times (4+5)}_{\text{concave function (with 5 components) parameters}} \\ + & \underbrace{1}_{\text{final MaxMean aggregation}} ,\end{aligned}$$

which is on the order of $\mathcal{O}(nw \max(d, w))$. In the common case where the hidden size w is chosen to be on the same magnitude as d, this becomes $\mathcal{O}(nd^2)$.

- **WideNorm** is based on the observation that

$$x \to \|W \text{ ReLU}(x :: -x)\|_2 \tag{B.191}$$

is an asymmetric norm when W is non-negative, where :: denotes vector concatenation. WideNorm then learns many such norms each with a different W matrix parameter, before (again) feeding the norm values into a concave function and aggregating them together with MaxMean.

For an input latent space of $\mathbb{R}^d$, consider a WideNorm with c such learned norms with W matrices of shape $\mathbb{R}^{(2d)\times w}_{\geq>0}$. This WideNorm predictor $f_{\textsf{WideNorm}}$

(on latent space), has

$$\begin{aligned}
\#\text{parmaters of } f_{\mathsf{WideNorm}} = & \underbrace{c \times (2d \times w)}_{W \text{ matrices}} \\
+ & \underbrace{c \times (4 + 5)}_{\text{concave function (with 5 components) parameters}} \\
+ & \underbrace{1}_{\text{final MaxMean aggregation}},
\end{aligned}$$

which is on the order of $\mathcal{O}(cdw)$. In the common case where both the number of components c and the output size of each component (before applying the $l2$-norm) are chosen to be on the same magnitude as d, this becomes $\mathcal{O}(d^3)$.

For both DeepNorm and WideNorm, their parameter counts are much larger than the number of effective parameters of PQEs (d for PQE-LH and $d + d^2$ for PQE-GG). For a 256-dimensional latent space, this difference can be on the order of $10^6 \sim 10^7$.

DeepNorm and WideNorm settings. Across all experiments of this section, we evaluate 2 DeepNorm settings and 3 WideNorm settings, all derived from the experiment setting of the original paper [127]. For both DeepNorm and WideNorm, we use MaxReLU activations, MaxMean aggregation, and concave function of 5 components. For DeepNorm, we use 3-layer networks with 2 different hidden sizes: 48 and 128 for the 48-dimensional latent space in random directed graphs experiments, 512 and 128 for the 512-dimensional latent space in the large-scale social graph experiments, 128 and 64 for the 128-dimensional latent space in offline Q-learning experiments. For WideNorm, we components of size 32 and experiment with 3 different numbers of components: 32, 48, and 128.

Error range. Results are gathered across 5 random seeds, showing both averages and population standard deviations.

Random Directed Graphs Quasimetric Learning

Graph generation. The random graph generation is controlled by three parameters d, ρ_{un} and ρ_{di}. d is the dimension of the vertex features. ρ_{un} specifies the fraction of pairs that should have at least one (directed) edge between them. ρ_{di} specifies the fraction of *such* pairs that should *only* have one (directed) edge between them. Therefore, if $\rho_{\mathsf{un}} = 1, \rho_{\mathsf{di}} = 0$, we have a fully connected graph; if $\rho_{\mathsf{un}} = 0.5, \rho_{\mathsf{di}} = 1$, we have a graph where half of the vertex pairs have exactly one (directed) edge between them, and the other half are not connected. For completeness, the exact generation procedure for a graph of n vertices is the following:

1. randomly add $\rho_{\mathsf{un}} \cdot n^2$ undirected edges, each represented as two opposite directed edges;

2. optimize $\mathbb{R}^{n \times d}$ vertex feature matrix using Adam [88] w.r.t. $\mathcal{L}_{\mathsf{align}}(\alpha = 2) + 0.3 \cdot \mathcal{L}_{\mathsf{uniform}}(t = 3)$ from [168], where each two node is considered a positive pair if they are connected;

3. randomly initialize a network f of architecture d-4096-4096-4096-4096-1 with tanh activations;

4. for each connected vertex pair (u, v), obtain $d_{u \to v} \triangleq f(\mathsf{feature}(u)) - f(\mathsf{feature}(v))$ and $d_{v \to u} = -d_{u \to v}$;

5. for each (u, v) such that $d_{u \to v}$ is among the top $1 - \rho_{\mathsf{di}}/2$ of such values (which is guaranteed to not include both directions of the same pair due to symmetry of $d_{u \to v}$), make $v \to u$ the only directed edge between u and v.

We experiment with three graphs of 300 vertices and 64-dimensional vertex features:

- **Figure B-6:** A graph generated with $\rho_{\mathsf{un}} = 0.15, \rho_{\mathsf{di}} = 0.85$;

- **Figure B-7:** A sparser graph generated with $\rho_{\mathsf{un}} = 0.05, \rho_{\mathsf{di}} = 0.85$;

- **Figure B-8:** A sparse graph with block structure by

1. generating 10 small dense graphs of 30 vertices and 32-dimensional vertex features, using $\rho_{\mathsf{un}} = 0.18, \rho_{\mathsf{di}} = 0.15$,
2. generating a sparse 10-vertex "supergraph" with 32-dimensional vertex features, using $\rho_{\mathsf{un}} = 0.22, \rho_{\mathsf{di}} = 0.925$,
3. for each supergraph vertex
   (a) associating it with a different small graph,
   (b) for all vertices of the small graph, concatenate the supergraph vertex's feature to the existing feature, forming 64-dimensional vertex features for the small graph vertices,
   (c) picking a random representative vertex from the small graph,
4. connecting all 10 representative vertices in the same way as their respective supergraph vertices are connected in the supergraph.

Architecture. All encoder based methods (PQEs, metric embeddings, dot products) use 64-128-128-128-48 network with ReLU activations, mapping 64-dimensional inputs to a 48-dimensional latent space. Unconstrained networks use a similar 128-128-128-128-48-1 network, mapping concatenated the 128-dimensional input to a scalar output.

Data. For each graph, we solve the groundtruth distance matrix and obtain 300^2 pairs, from which we randomly sample the training set, and use the rest as the test set. We run on 5 training fractions evenly spaced on the logarithm scale, from 0.01 to 0.7.

Training. We use 2048 batch size with the Adam optimizer [88], with learning rate decaying according to the cosine schedule without restarting [107] starting from 10^{-4} to 0 over 3000 epochs. All models are optimized w.r.t. MSE on the γ-discounted distances, with $\gamma = 0.9$. When running with the triangle inequality regularizer, $683 \approx 2048/3$ triplets are uniformly sampled at each iteration.

Full results and ablation studies. Figures B-6 to B-8 show full results of all methods running on all three graphs. In Figure B-9, we perform ablation studies on the implementation techniques for PQEs mentioned in Appendix B.3.4: outputting discounted distance and deep linear networks. On the simple directed graphs such as the dense graph, the basic PQE-LH without theses techniques works really well, even surpassing the results with both techniques. However, on graphs with more complex structures (e.g., the sparse graph and the sparse graph with block structure), basic versions of PQE-LH and PQE-GG starts to perform badly and show large variance, while the versions with both techniques stably trains to the best results. Therefore, for robustness, we use both techniques in other experiments.

Large-Scale Social Graphs Quasimetric Learning

Data source. We choose the Berkeley-StanfordWebGraph [101] as the large-scale *directed* social graph, which consists of 685,230 pages as nodes, and 7,600,595 hyperlinks as directed edges. Additionally, we also use the Youtube social network [101, 116] as a *undirected* social graph, which consists of 1,134,890 users as nodes, and 2,987,624 friendship relations as undirected edges. Both datasets are available from the SNAP website [101] under the BSD license.

Data processing. For each graph, we use node2vec to obtain 128-dimensional node features [55]. Since the graph is large, we use the landmark method [132] to construct training and test sets. Specifically, we randomly choose 150 nodes, called landmarks, and compute the distances between these landmarks and all nodes. For directed graph, this means computing distances of both directions. From the obtained pairs and distances, we randomly sample 2,500,000 pairs to form the training set. Similarly, we form a test set of 150,000 from a disjoint set of 50 landmarks. For the undirected graph, we double the size of each set by reversing the pairs, since the distance is symmetrical.

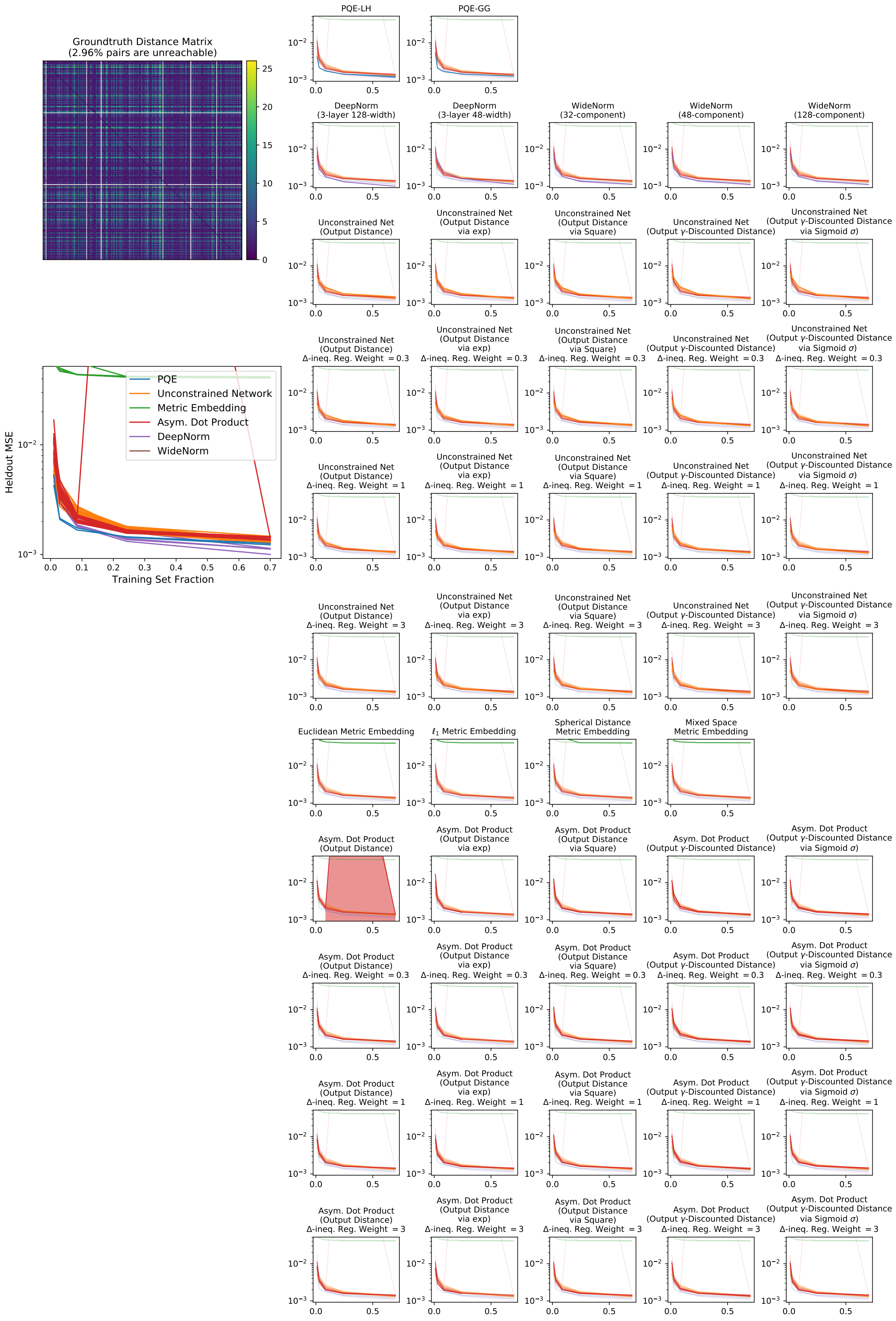


Figure B-6: A dense graph. Individual plots on the right show standard deviations.

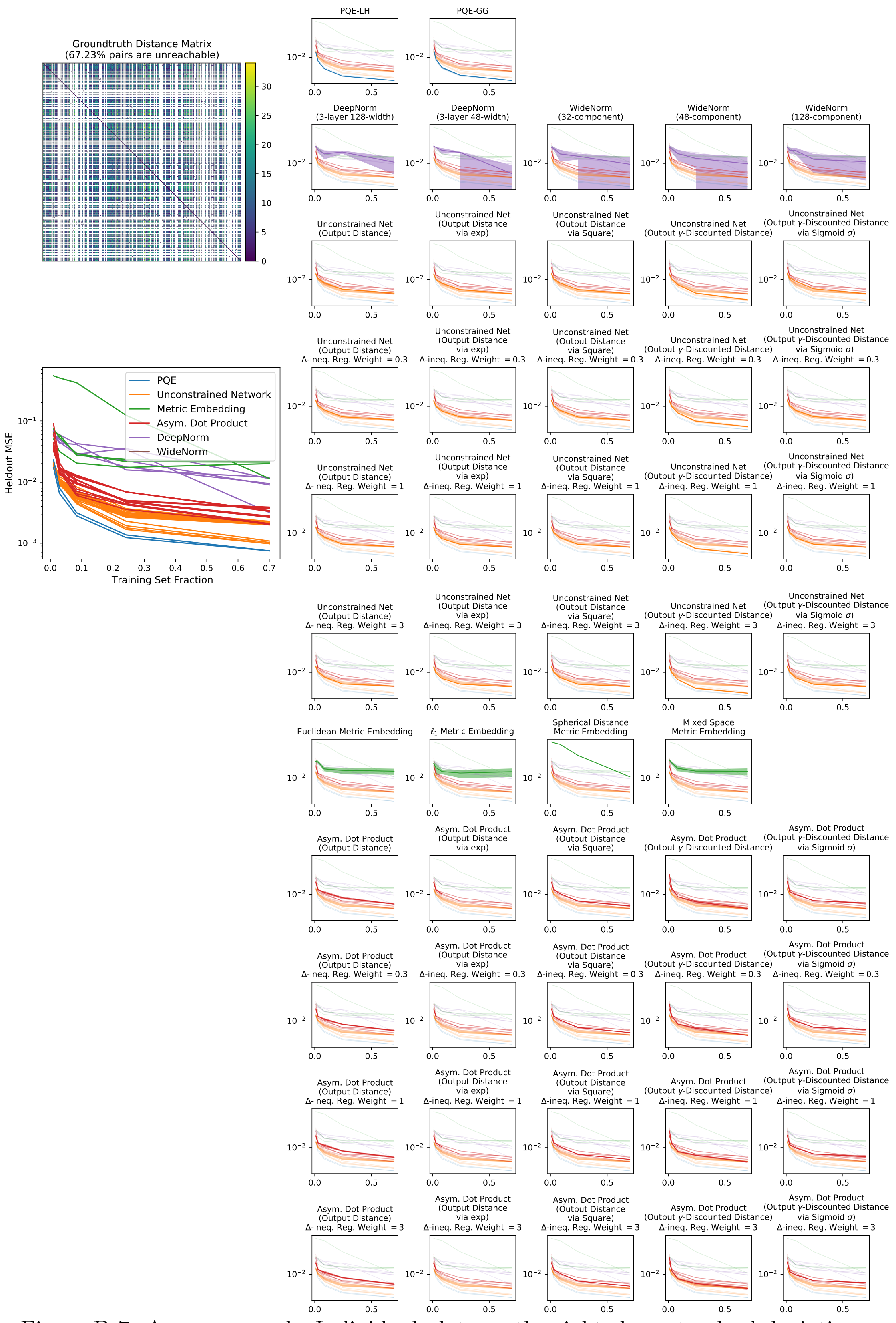


Figure B-7: A sparse graph. Individual plots on the right show standard deviations.

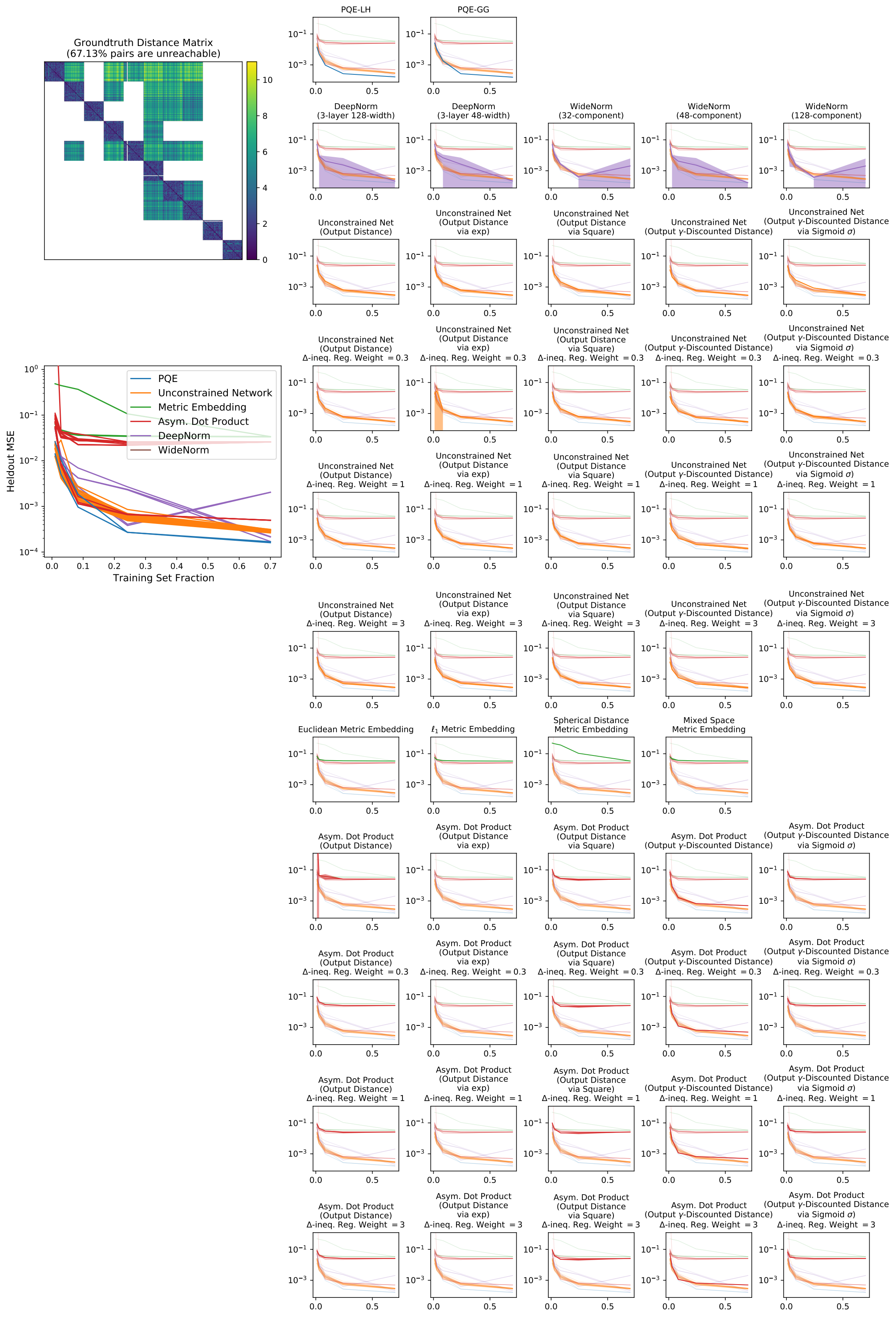


Figure B-8: A sparse graph with block structure. Individual plots on the right show standard deviations.

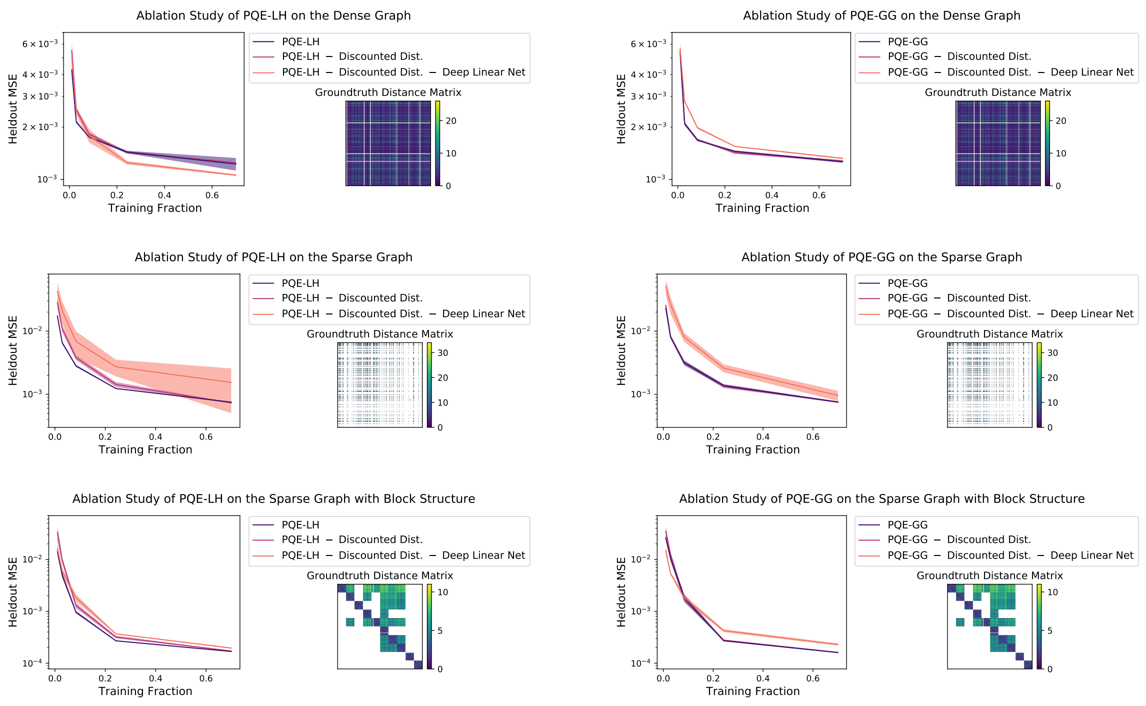


Figure B-9: Ablation studies of PQE-LH and PQE-GG on three random graphs.

Architecture. All encoder based methods (PQEs, metric embeddings, dot products) use 128-2048-2048-2048-512 network with ReLU activations and Batch Normalization [82] after each activation, mapping 128-dimensional inputs to a 512-dimensional latent space. Unconstrained networks use a similar 256-2048-2048-2048-512-1 network, mapping concatenated the 256-dimensional input to a scalar output.

Training. We use 1024 batch size with the Adam optimizer [88], with learning rate decaying according to the cosine schedule without restarting [107] starting from 10^{-4} to 0 over 80 epochs. All models are optimized w.r.t. MSE on the γ-discounted distances, with $\gamma = 0.9$. When running with the triangle inequality regularizer, $342 \approx 1024/3$ triplets are uniformly sampled at each iteration.

Full results. Tables B.1 and B.2 show full results of distance learning on these two graphs. On the *directed* Berkeley-StanfordWebGraph, PQE-LH performs the best (w.r.t. discounted distance MSE). While PQE-GG has larger discounted distance MSE than some other baselines, it accurately predicts finite distances and outputs large

Method Family	Formulation		MSE w.r.t. γ-discounted distances ($\times 10^{-3}$) $\downarrow$	L1 Error when true $d < \infty$ $\downarrow$	Prediction $\hat{d}$ when true $d = \infty$ $\uparrow$
PQEs	PQE-LH		3.0427 ± 0.1527	1.6263 ± 0.0550	69.9424 ± 0.4930
	PQE-GG		3.9085 ± 0.1258	1.8951 ± 0.0336	101.8240 ± 10.3970
Unconstrained Nets (without Triangle Inequality Regularizer)	Output distance	$\hat{d}(x,y) \triangleq f(x,y)$	3.0862 ± 0.0392	2.1151 ± 0.0241	59.5243 ± 0.3700
	Output distance via $\exp(\cdot)$	$\hat{d}(x,y) \triangleq \exp(f(x,y))$	3.3541 ± 0.1759	$1.0090 \times 10^{23} \pm 2.0179 \times 10^{23}$	$5.3583 \times 10^{5} \pm 1.0582 \times 10^{6}$
	Output distance via squaring $a \to a^2$	$\hat{d}(x,y) \triangleq (f(x,y))^2$	4.5663 ± 0.2294	3.3459 ± 0.2494	68.2613 ± 11.6061
	Output γ-discounted distance	$\gamma^{\hat{d}(x,y)} \triangleq f(x,y)$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output γ-discounted distance via sigmoid $\sigma(\cdot)$	$\gamma^{\hat{d}(x,y)} \triangleq \sigma(f(x,y))$	3.1823 ± 0.1133	$\infty \pm$ NaN	65.8630 ± 0.4287
Unconstrained Nets (Triangle Inequality Regularizer Weight = 0.3)	Output distance	$\hat{d}(x,y) \triangleq f(x,y)$	2.8128 ± 0.0625	2.2109 ± 0.0341	61.3709 ± 0.3936
	Output distance via $\exp(\cdot)$	$\hat{d}(x,y) \triangleq \exp(f(x,y))$	2.9344 ± 0.0455	$\infty \pm$ NaN	$\infty \pm$ NaN
	Output distance via squaring $a \to a^2$	$\hat{d}(x,y) \triangleq (f(x,y))^2$	4.9947 ± 0.4198	16.5445 ± 29.3175	58.9205 ± 6.4216
	Output γ-discounted distance	$\gamma^{\hat{d}(x,y)} \triangleq f(x,y)$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output γ-discounted distance via sigmoid $\sigma(\cdot)$	$\gamma^{\hat{d}(x,y)} \triangleq \sigma(f(x,y))$	2.9178 ± 0.1351	$\infty \pm$ NaN	$\infty \pm$ NaN
Unconstrained Nets (Triangle Inequality Regularizer Weight = 1)	Output distance	$\hat{d}(x,y) \triangleq f(x,y)$	3.0481 ± 0.1272	2.3729 ± 0.1378	60.4040 ± 0.1890
	Output distance via $\exp(\cdot)$	$\hat{d}(x,y) \triangleq \exp(f(x,y))$	3.0161 ± 0.0718	$\infty \pm$ NaN	$3.1289 \times 10^{16} \pm 6.2579 \times 10^{16}$
	Output distance via squaring $a \to a^2$	$\hat{d}(x,y) \triangleq (f(x,y))^2$	4.4921 ± 0.3534	3.6930 ± 0.4896	90.6206 ± 66.5704
	Output γ-discounted distance	$\gamma^{\hat{d}(x,y)} \triangleq f(x,y)$	4.4046 ± 0.5167	2.7873 ± 0.0770	31.3195 ± 0.9929
	Output γ-discounted distance via sigmoid $\sigma(\cdot)$	$\gamma^{\hat{d}(x,y)} \triangleq \sigma(f(x,y))$	2.9314 ± 0.1022	2.2634 ± 0.1147	$\infty \pm$ NaN
Unconstrained Nets (Triangle Inequality Regularizer Weight = 3)	Output distance	$\hat{d}(x,y) \triangleq f(x,y)$	5.2955 ± 0.5279	3.8060 ± 0.2908	58.1193 ± 0.4383
	Output distance via $\exp(\cdot)$	$\hat{d}(x,y) \triangleq \exp(f(x,y))$	3.5713 ± 0.2002	212.5421 ± 416.9256	$\infty \pm$ NaN
	Output distance via squaring $a \to a^2$	$\hat{d}(x,y) \triangleq (f(x,y))^2$	4.3745 ± 0.3709	2.9491 ± 0.2228	53.1119 ± 5.5452
	Output γ-discounted distance	$\gamma^{\hat{d}(x,y)} \triangleq f(x,y)$	7.3416 ± 0.6486	3.5232 ± 0.1352	26.9200 ± 0.4697
	Output γ-discounted distance via sigmoid $\sigma(\cdot)$	$\gamma^{\hat{d}(x,y)} \triangleq \sigma(f(x,y))$	3.5818 ± 0.3565	$\infty \pm$ NaN	65.7709 ± 0.8646
Asym. Dot Products (without Triangle Inequality Regularizer)	Output distance	$\hat{d}(x,y) \triangleq f(x)^\mathsf{T} g(y)$	$3.1622 \times 10^{19} \pm$ NaN	$23.4270 \pm$ NaN	$0.1529 \pm$ NaN
	Output distance via $\exp(\cdot)$	$\hat{d}(x,y) \triangleq \exp(f(x)^\mathsf{T} g(y))$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output distance via squaring $a \to a^2$	$\hat{d}(x,y) \triangleq (f(x)^\mathsf{T} g(y))^2$	48.1056 ± 0.0056	$2.5195 \times 10^{11} \pm 2.1751 \times 10^{11}$	$2.6794 \times 10^{11} \pm 2.5398 \times 10^{11}$
	Output γ-discounted distance	$\gamma^{\hat{d}(x,y)} \triangleq f(x)^\mathsf{T} g(y)$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output γ-discounted distance via sigmoid $\sigma(\cdot)$	$\gamma^{\hat{d}(x,y)} \triangleq \sigma(f(x)^\mathsf{T} g(y))$	48.1073 ± 0.0112	$\infty \pm$ NaN	$\infty \pm$ NaN
Asym. Dot Products (Triangle Inequality Regularizer Weight = 0.3)	Output distance	$\hat{d}(x,y) \triangleq f(x)^\mathsf{T} g(y)$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output distance via $\exp(\cdot)$	$\hat{d}(x,y) \triangleq \exp(f(x)^\mathsf{T} g(y))$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output distance via squaring $a \to a^2$	$\hat{d}(x,y) \triangleq (f(x)^\mathsf{T} g(y))^2$	48.1041 ± 0.0035	$1.9498 \times 10^{11} \pm 7.9641 \times 10^{10}$	$1.6049 \times 10^{11} \pm 3.7099 \times 10^{10}$
	Output γ-discounted distance	$\gamma^{\hat{d}(x,y)} \triangleq f(x)^\mathsf{T} g(y)$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output γ-discounted distance via sigmoid $\sigma(\cdot)$	$\gamma^{\hat{d}(x,y)} \triangleq \sigma(f(x)^\mathsf{T} g(y))$	48.1103 ± 0.0110	$\infty \pm$ NaN	$\infty \pm$ NaN
Asym. Dot Products (Triangle Inequality Regularizer Weight = 1)	Output distance	$\hat{d}(x,y) \triangleq f(x)^\mathsf{T} g(y)$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output distance via $\exp(\cdot)$	$\hat{d}(x,y) \triangleq \exp(f(x)^\mathsf{T} g(y))$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output distance via squaring $a \to a^2$	$\hat{d}(x,y) \triangleq (f(x)^\mathsf{T} g(y))^2$	48.1021 ± 0.0002	$2.2986 \times 10^{11} \pm 9.1970 \times 10^{10}$	$2.5002 \times 10^{11} \pm 1.4464 \times 10^{11}$
	Output γ-discounted distance	$\gamma^{\hat{d}(x,y)} \triangleq f(x)^\mathsf{T} g(y)$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output γ-discounted distance via sigmoid $\sigma(\cdot)$	$\gamma^{\hat{d}(x,y)} \triangleq \sigma(f(x)^\mathsf{T} g(y))$	58.4894 ± 23.2224	$\infty \pm$ NaN	$\infty \pm$ NaN
Asym. Dot Products (Triangle Inequality Regularizer Weight = 3)	Output distance	$\hat{d}(x,y) \triangleq f(x)^\mathsf{T} g(y)$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output distance via $\exp(\cdot)$	$\hat{d}(x,y) \triangleq \exp(f(x)^\mathsf{T} g(y))$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output distance via squaring $a \to a^2$	$\hat{d}(x,y) \triangleq (f(x)^\mathsf{T} g(y))^2$	48.1031 ± 0.0020	$2.3522 \times 10^{11} \pm 2.6429 \times 10^{11}$	$1.7025 \times 10^{11} \pm 1.0700 \times 10^{11}$
	Output γ-discounted distance	$\gamma^{\hat{d}(x,y)} \triangleq f(x)^\mathsf{T} g(y)$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output γ-discounted distance via sigmoid $\sigma(\cdot)$	$\gamma^{\hat{d}(x,y)} \triangleq \sigma(f(x)^\mathsf{T} g(y))$	48.3034 ± 0.4485	$\infty \pm$ NaN	$\infty \pm$ NaN
Metric Embeddings	Euclidean space	$\hat{d}(x,y) \triangleq \lVert f(x) - f(y) \rVert_2$	17.5952 ± 0.2667	7.5399 ± 0.0742	53.8500 ± 3.8430
	ℓ_1 space	$\hat{d}(x,y) \triangleq \lVert f(x) - f(y) \rVert_1$	18.0521 ± 0.3546	7.1154 ± 0.1835	66.2507 ± 3.3308
	Spherical distance space w/ learnable scale α	$\hat{d}(x,y) \triangleq \alpha \cdot \arccos(\frac{f(x)^\mathsf{T} f(y)}{\lVert f(x) \rVert_2 \lVert f(y) \rVert_2})$	19.2990 ± 0.2032	6.9545 ± 0.0887	32.1458 ± 0.4562
	Mixing above three spaces w/ learnable weights		17.8312 ± 0.3099	7.3493 ± 0.1086	51.7481 ± 3.6248
DeepNorms	3-layer 128-width		7.0862 ± 0.3170	2.4498 ± 0.0617	111.2209 ± 2.5045
	3-layer 512-width		5.0715 ± 0.1348	2.0853 ± 0.0633	120.0452 ± 4.3525
WideNorms	32-component (each of size 32)		3.5328 ± 0.2120	1.7694 ± 0.0213	124.6580 ± 2.8678
	48-component (each of size 32)		3.6842 ± 0.2385	1.8081 ± 0.0680	122.6833 ± 5.5026
	128-component (each of size 32)		3.8125 ± 0.2331	1.8096 ± 0.0765	128.5427 ± 5.1412

Table B.1: Quasimetric learning on the large-scale *directed* Berkeley-StanfordWebGraph.

Method Family	Formulation		MSE w.r.t. γ-discounted distances ($\times 10^{-3}$) $\downarrow$	L1 Error when true $d < \infty$ $\downarrow$	Prediction $\hat{d}$ when true $d = \infty$ $\uparrow$
PQEs	PQE-LH		2.4400 $\pm$ 0.0695	0.6480 $\pm$ 0.0119	NaN $\pm$ NaN
	PQE-GG		2.5895 $\pm$ 0.0318	0.6697 $\pm$ 0.0042	NaN $\pm$ NaN
Unconstrained Nets (without Triangle Inequality Regularizer)	Output distance	$\hat{d}(x,y) \triangleq f(x,y)$	1.4883 $\pm$ 0.0168	0.5084 $\pm$ 0.0029	NaN $\pm$ NaN
	Output distance via $\exp(\cdot)$	$\hat{d}(x,y) \triangleq \exp(f(x,y))$	1.5223 $\pm$ 0.0160	0.4910 $\pm$ 0.0151	NaN $\pm$ NaN
	Output distance via squaring $a \rightarrow a^2$	$\hat{d}(x,y) \triangleq (f(x,y))^2$	2.2955 $\pm$ 1.1674	0.6185 $\pm$ 0.1409	NaN $\pm$ NaN
	Output γ-discounted distance	$\gamma^{\hat{d}(x,y)} \triangleq f(x,y)$	1.5069 $\pm$ 0.0228	0.4975 $\pm$ 0.0211	NaN $\pm$ NaN
	Output γ-discounted distance via sigmoid $\sigma(\cdot)$	$\gamma^{\hat{d}(x,y)} \triangleq \sigma(f(x,y))$	1.4802 $\pm$ 0.0197	0.5082 $\pm$ 0.0036	NaN $\pm$ NaN
Unconstrained Nets (Triangle Inequality Regularizer Weight = 0.3)	Output distance	$\hat{d}(x,y) \triangleq f(x,y)$	1.5009 $\pm$ 0.0208	0.5107 $\pm$ 0.0032	NaN $\pm$ NaN
	Output distance via $\exp(\cdot)$	$\hat{d}(x,y) \triangleq \exp(f(x,y))$	1.5206 $\pm$ 0.0444	0.4935 $\pm$ 0.0098	NaN $\pm$ NaN
	Output distance via squaring $a \rightarrow a^2$	$\hat{d}(x,y) \triangleq (f(x,y))^2$	1.7398 $\pm$ 0.3896	0.5488 $\pm$ 0.0600	NaN $\pm$ NaN
	Output γ-discounted distance	$\gamma^{\hat{d}(x,y)} \triangleq f(x,y)$	1.5005 $\pm$ 0.0148	0.4986 $\pm$ 0.0121	NaN $\pm$ NaN
	Output γ-discounted distance via sigmoid $\sigma(\cdot)$	$\gamma^{\hat{d}(x,y)} \triangleq \sigma(f(x,y))$	1.4851 $\pm$ 0.0168	0.5089 $\pm$ 0.0026	NaN $\pm$ NaN
Unconstrained Nets (Triangle Inequality Regularizer Weight = 1)	Output distance	$\hat{d}(x,y) \triangleq f(x,y)$	1.4999 $\pm$ 0.0243	0.5107 $\pm$ 0.0046	NaN $\pm$ NaN
	Output distance via $\exp(\cdot)$	$\hat{d}(x,y) \triangleq \exp(f(x,y))$	1.5224 $\pm$ 0.0376	0.4948 $\pm$ 0.0169	NaN $\pm$ NaN
	Output distance via squaring $a \rightarrow a^2$	$\hat{d}(x,y) \triangleq (f(x,y))^2$	1.8875 $\pm$ 0.5078	0.5692 $\pm$ 0.0683	NaN $\pm$ NaN
	Output γ-discounted distance	$\gamma^{\hat{d}(x,y)} \triangleq f(x,y)$	1.4769 $\pm$ 0.0176	0.4919 $\pm$ 0.0128	NaN $\pm$ NaN
	Output γ-discounted distance via sigmoid $\sigma(\cdot)$	$\gamma^{\hat{d}(x,y)} \triangleq \sigma(f(x,y))$	1.4846 $\pm$ 0.0115	0.5088 $\pm$ 0.0021	NaN $\pm$ NaN
Unconstrained Nets (Triangle Inequality Regularizer Weight = 3)	Output distance	$\hat{d}(x,y) \triangleq f(x,y)$	1.4939 $\pm$ 0.0110	0.5099 $\pm$ 0.0018	NaN $\pm$ NaN
	Output distance via $\exp(\cdot)$	$\hat{d}(x,y) \triangleq \exp(f(x,y))$	1.5154 $\pm$ 0.0389	0.4871 $\pm$ 0.0174	NaN $\pm$ NaN
	Output distance via squaring $a \rightarrow a^2$	$\hat{d}(x,y) \triangleq (f(x,y))^2$	2.4747 $\pm$ 1.0850	0.6505 $\pm$ 0.1357	NaN $\pm$ NaN
	Output γ-discounted distance	$\gamma^{\hat{d}(x,y)} \triangleq f(x,y)$	1.4915 $\pm$ 0.0127	0.4983 $\pm$ 0.0160	NaN $\pm$ NaN
	Output γ-discounted distance via sigmoid $\sigma(\cdot)$	$\gamma^{\hat{d}(x,y)} \triangleq \sigma(f(x,y))$	1.4829 $\pm$ 0.0153	0.5084 $\pm$ 0.0029	NaN $\pm$ NaN
Asym. Dot Products (without Triangle Inequality Regularizer)	Output distance	$\hat{d}(x,y) \triangleq f(x)^\mathsf{T} g(y)$	2633.7907 $\pm$ NaN	11.3879 $\pm$ NaN	NaN $\pm$ NaN
	Output distance via $\exp(\cdot)$	$\hat{d}(x,y) \triangleq \exp(f(x)^\mathsf{T} g(y))$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output distance via squaring $a \rightarrow a^2$	$\hat{d}(x,y) \triangleq (f(x)^\mathsf{T} g(y))^2$	339.1550 $\pm$ 0.0022	7.8948×10^{11} $\pm$ 7.4010×10^{11}	NaN $\pm$ NaN
	Output γ-discounted distance	$\gamma^{\hat{d}(x,y)} \triangleq f(x)^\mathsf{T} g(y)$	2.6920 $\pm$ 1.2655	0.7062 $\pm$ 0.2156	NaN $\pm$ NaN
	Output γ-discounted distance via sigmoid $\sigma(\cdot)$	$\gamma^{\hat{d}(x,y)} \triangleq \sigma(f(x)^\mathsf{T} g(y))$	182.2068 $\pm$ 1.2382	∞ $\pm$ NaN	NaN $\pm$ NaN
Asym. Dot Products (Triangle Inequality Regularizer Weight = 0.3)	Output distance	$\hat{d}(x,y) \triangleq f(x)^\mathsf{T} g(y)$	9.9748×10^5 $\pm$ NaN	8.1867 $\pm$ NaN	NaN $\pm$ NaN
	Output distance via $\exp(\cdot)$	$\hat{d}(x,y) \triangleq \exp(f(x)^\mathsf{T} g(y))$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output distance via squaring $a \rightarrow a^2$	$\hat{d}(x,y) \triangleq (f(x)^\mathsf{T} g(y))^2$	339.1560 $\pm$ 0.0010	6.8658×10^{11} $\pm$ 3.4985×10^{11}	NaN $\pm$ NaN
	Output γ-discounted distance	$\gamma^{\hat{d}(x,y)} \triangleq f(x)^\mathsf{T} g(y)$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output γ-discounted distance via sigmoid $\sigma(\cdot)$	$\gamma^{\hat{d}(x,y)} \triangleq \sigma(f(x)^\mathsf{T} g(y))$	183.3337 $\pm$ 1.0384	∞ $\pm$ NaN	NaN $\pm$ NaN
Asym. Dot Products (Triangle Inequality Regularizer Weight = 1)	Output distance	$\hat{d}(x,y) \triangleq f(x)^\mathsf{T} g(y)$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output distance via $\exp(\cdot)$	$\hat{d}(x,y) \triangleq \exp(f(x)^\mathsf{T} g(y))$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output distance via squaring $a \rightarrow a^2$	$\hat{d}(x,y) \triangleq (f(x)^\mathsf{T} g(y))^2$	339.1552 $\pm$ 0.0021	7.4588×10^{11} $\pm$ 3.7277×10^{11}	NaN $\pm$ NaN
	Output γ-discounted distance	$\gamma^{\hat{d}(x,y)} \triangleq f(x)^\mathsf{T} g(y)$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output γ-discounted distance via sigmoid $\sigma(\cdot)$	$\gamma^{\hat{d}(x,y)} \triangleq \sigma(f(x)^\mathsf{T} g(y))$	191.0928 $\pm$ 9.7137	∞ $\pm$ NaN	NaN $\pm$ NaN
Asym. Dot Products (Triangle Inequality Regularizer Weight = 3)	Output distance	$\hat{d}(x,y) \triangleq f(x)^\mathsf{T} g(y)$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output distance via $\exp(\cdot)$	$\hat{d}(x,y) \triangleq \exp(f(x)^\mathsf{T} g(y))$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output distance via squaring $a \rightarrow a^2$	$\hat{d}(x,y) \triangleq (f(x)^\mathsf{T} g(y))^2$	339.1556 $\pm$ 0.0020	9.0283×10^{11} $\pm$ 6.0203×10^{11}	NaN $\pm$ NaN
	Output γ-discounted distance	$\gamma^{\hat{d}(x,y)} \triangleq f(x)^\mathsf{T} g(y)$	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
	Output γ-discounted distance via sigmoid $\sigma(\cdot)$	$\gamma^{\hat{d}(x,y)} \triangleq \sigma(f(x)^\mathsf{T} g(y))$	228.0300 $\pm$ 37.0632	∞ $\pm$ NaN	NaN $\pm$ NaN
Metric Embeddings	Euclidean space	$\hat{d}(x,y) \triangleq \lVert f(x) - f(y) \rVert_2$	1.3131 $\pm$ 0.0671	0.4833 $\pm$ 0.0128	NaN $\pm$ NaN
	ℓ_1 space	$\hat{d}(x,y) \triangleq \lVert f(x) - f(y) \rVert_1$	3.5993 $\pm$ 1.5986	0.7787 $\pm$ 0.1842	NaN $\pm$ NaN
	Spherical distance space w/ learnable scale α	$\hat{d}(x,y) \triangleq \alpha \cdot \arccos(\frac{f(x)^\mathsf{T} f(y)}{\lVert f(x) \rVert_2 \lVert f(y) \rVert_2})$	6.7731 $\pm$ 0.1915	1.0829 $\pm$ 0.0177	NaN $\pm$ NaN
	Mixing above three spaces w/ learnable weights		2.1014 $\pm$ 0.0685	0.5923 $\pm$ 0.0109	NaN $\pm$ NaN
DeepNorms	3-layer 128-width		8.0192 $\pm$ 0.2476	1.1834 $\pm$ 0.0213	NaN $\pm$ NaN
	3-layer 512-width		5.4366 $\pm$ 0.0855	0.9666 $\pm$ 0.0072	NaN $\pm$ NaN
WideNorms	32-component (each of size 32)		3.0841 $\pm$ 0.0667	0.7272 $\pm$ 0.0068	NaN $\pm$ NaN
	48-component (each of size 32)		3.0438 $\pm$ 0.1322	0.7247 $\pm$ 0.0173	NaN $\pm$ NaN
	128-component (each of size 32)		2.9964 $\pm$ 0.1363	0.7173 $\pm$ 0.0166	NaN $\pm$ NaN

Table B.2: Metric learning on the large-scale *undirected* `Youtube` graph. This graph does not have unreachable pairs so the last column is always `NaN`.

values for unreachable pairs. On the *undirected* Youtube graph, perhaps as expected, metric embedding methods have an upper hand, with the best performing method being an Euclidean space embedding. Notably, DeepNorms and WideNorms do much worse than PQEs on this symmetric graph.

Offline Q-Learning

As shown in Proposition B.1.4 and Remark B.1.5, we know that a quasimetric is formed with the optimal goal-reaching plan costs in a MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$ where each action has *unit cost* (i.e., negated reward). The quasimetric is defined on $\mathcal{X} \triangleq \mathcal{S} \cup (\mathcal{S} \times \mathcal{A})$.

Similarly, Tian et al. [151] also make this observation and propose to optimize a distance function by Q-learning on a collected set of trajectories. The optimized distance function (i.e., Q-function) is then used with standard planning algorithms such as the Cross Entropy Method (CEM) [35]. The specific model they used is an unconstrained network $f \colon (s, a, s') \to \mathbb{R}$, outputting discounted distances (Q-values).

Due to the existing quasimetric structure, we explore using PQEs as the distance function formulation. We mostly follow the algorithm in Tian et al. [151] except for the following minor differences:

- Tian et al. [151] propose to sample half of the goal from future steps of the same trajectory, and half of the goal from similar states across the entire dataset, defined by a nearest neighbor search. For simplicity, in the latter case, we instead sample a random state across the entire dataset.

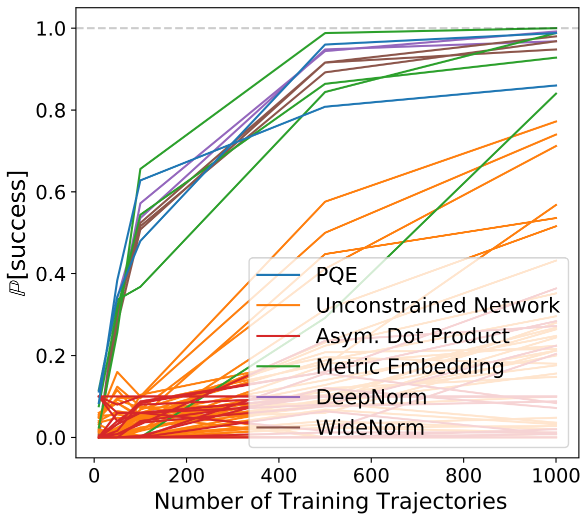
- In Tian et al. [151], target goals are defined as single states, and the Q-learning formulation only uses quantities distances from state-action pairs $(s, a) \in \mathcal{S} \times \mathcal{A}$ to states s': $\hat{d}((s, a), s')$.

  However, if we only train on $\hat{d}((s, a), s')$, quasimetric embeddings might not learn much about the distance to state-action pairs, or from states, because it may simply only assign finite distances to $\hat{d}((s, a), s')$, and set everything else to infinite. To prevent such issues, we choose to use state-action pairs as target goals, by adding a random action. Then, the embedding methods only need to embed state-action pairs.

In planning when the target is actually a single goal $s' \in \mathcal{S}$, we use the following distance/Q-function

$$\hat{d}((s,a),s') \triangleq -1 + \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} \hat{d}((s,a),(s',a')). \tag{B.192}$$

Such a modification is used for all embedding methods (PQEs, metric embeddings, asymmetrical dot products). For unconstrained networks, we test both the original formulation (of using single state as goals) and this modification.



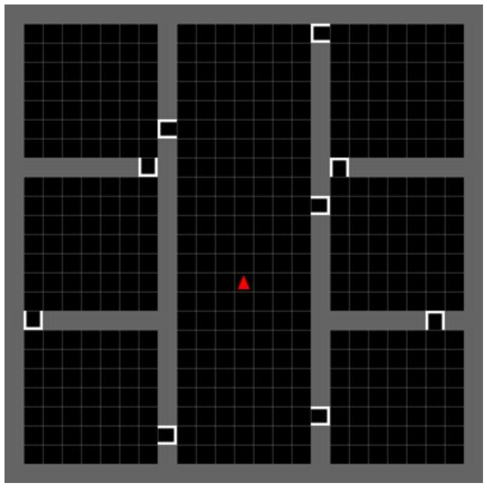


Figure B-10: Grid-world offline Q-learning average planning success rates. Right shows the environment.

Environment. The environment is a grid-world with one-way doors, as shown in of Figure B-10, which is built upon `gym-minigrid` [28] (a project under Apache 2.0 License). The agent has 4 actions corresponding to moving towards 4 directions. When it moves toward a direction that is blocked by a wall or an one-way door, it does not move. States are represented as 18-dimensional vectors, containing the 2D location of the agent (normalized to be within $[-1,1]^2$). The other dimensions are always constant in our enviroment as they refer to information that can not be changed in this particular environment (e.g., the state of the doors). The agent always starts at a random location in the center room (e.g., the initial position of the red triangle in Figure B-10). The environment also defines a goal sampling distribution as a random location in one of the rooms on the left or right side. Note that this goal distribution is only used for data collection and evaluation. In training, we train goal-conditional policies using the goal sampling mechanism adapted from Tian et al. [151], as described above.

Training trajectories. To collect the training trajectories, we use an ϵ-greedy planner with groundtruth distance toward the environment goal, with a large $\epsilon = 0.6$. Each trajectory is capped to have at most 200 steps.

Architecture. All encoder based methods (PQEs, metric embeddings, dot products) use 18-2048-2048-2048-1024 network with ReLU activations and Batch Normalization [82] after each activation, mapping a 18-dimensional state to four 256-dimensional latent vectors, corresponding to the embeddings for all four state-action pairs. Unconstrained networks use a similar architecture and take in concatenated 36-dimensional inputs. With the original formulation with states as goals, we use a 36-2048-2048-2048-256-4 network to obtain a $\mathbb{R}^{|\mathcal{A}|}$ output, representing the distance/Q-values from each state-action pair to the goal; with the modified formulation with state-action pairs as goals, we use a 36-2048-2048-2048-256-16 network to obtain a $\mathbb{R}^{|\mathcal{A}|\times|\mathcal{A}|}$ output.

Training. We use 1024 batch size with the Adam optimizer [88], with learning rate decaying according to the cosine schedule without restarting [107] starting from 10^{-4} to 0 over 1000 epochs. Since we are running Q-learning, all models are optimized w.r.t. MSE on the γ-discounted distances, with $\gamma = 0.95$. When running with the triangle inequality regularizer, $341 \approx 1024/3$ triplets are uniformly sampled at each iteration.

Planning details. To use the learned distance/Q-function for planning towards a given goal, we perform greedy 1-step planning, where we always select the best action in $\mathcal{A}$ according to the learned model, without any lookahead. In each of 50 runs, the planner is asked to reach a goal given by the environment within 300 steps. The set of 50 initial location and goal states is entirely decided by the seed used, regardless of the model. We run each method 5 times using the same set of 5 seeds.

Full results. Average results across 5 runs are shown in Figure B-10, with full results (with standard deviations) shown in Figure B-11. Planning performance across the formulations vary a lot, with PQEs and the Euclidean metric embedding being the best and most data-efficient ones. Using either formulation (states vs. state-action pairs as goals) does not seem to affect the performance of unconstrained networks. We note that the the asymmetrical dot product formulation outputting discounted distance is similar to Universal Value Function Approximators (UVFA) formulation [136]; the unconstrained network outputting discounted distance with states as goals is the same formulation as the method from Tian et al. [151].

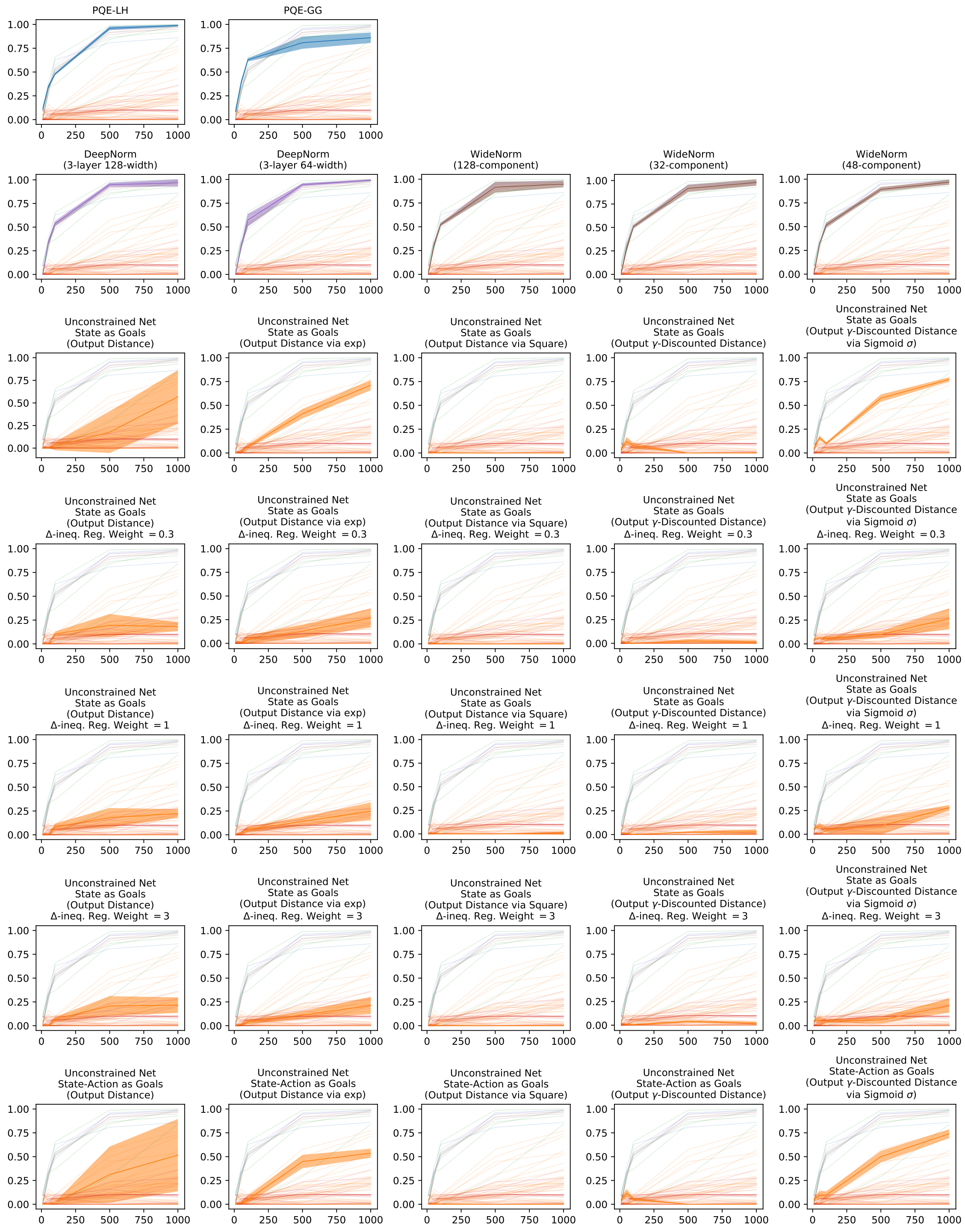


Figure B-11: Grid-world offline Q-learning full results. Individual plots on show standard deviations.

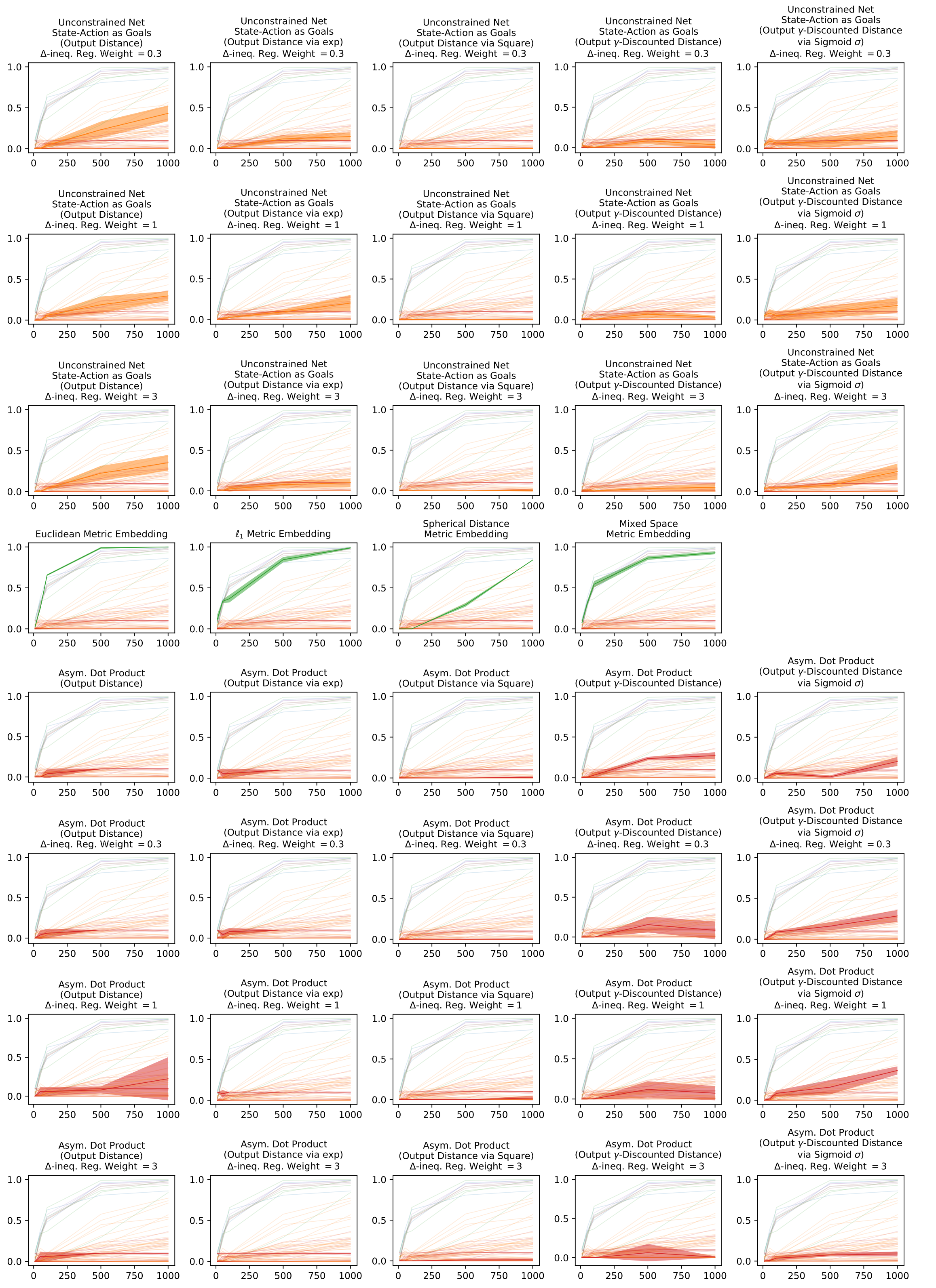


Figure B-11: Grid-world offline Q-learning full results (cont.). Individual plots on show standard deviations.

Appendix C

Details and Additional Discussions for Chapter 4

C.1 Denoised MDP Discussions

C.1.1 Loss Derivation

To apply our mutual information regularizer $I(\boldsymbol{x}; \boldsymbol{s} \mid \boldsymbol{a})$, we can consider a form using another variational distribution ρ (see, e.g., Poole et al. [128]),

$$
\begin{aligned}
I(\boldsymbol{x}; \boldsymbol{s} \mid \boldsymbol{a}) &= \min_{\rho} \mathbb{E}_{\boldsymbol{a}} \mathbb{E}_{p_\theta(\boldsymbol{s} \mid \boldsymbol{a})} \left[D_{\mathsf{KL}}(p_\theta(\boldsymbol{x} \mid \boldsymbol{s}, \boldsymbol{a}) \,\|\, \rho(\boldsymbol{x} \mid \boldsymbol{a})) \right] \\
&\approx \min_{\rho} \mathbb{E}_{\boldsymbol{a}} \mathbb{E}_{q_\psi(\boldsymbol{s} \mid \boldsymbol{a})} \left[D_{\mathsf{KL}}(q_\psi(\boldsymbol{x} \mid \boldsymbol{s}, \boldsymbol{a}) \,\|\, \rho(\boldsymbol{x} \mid \boldsymbol{a})) \right] \\
&\qquad\qquad\qquad\qquad (\text{assume } q_\psi \text{ is roughly the posterior of } p_\theta) \\
&= \min_{\theta'} \mathcal{L}_{\mathsf{KL}\text{-}x}(\psi, \theta'). \qquad (\text{C.1})
\end{aligned}
$$

The assumption that q_ψ is roughly the posterior of p_θ is acceptable because it is the natural consequence of optimizing the variational MLE objective in Equation (4.1) over θ, ψ.

Alternatively, we can consider the MI defined by a joint conditional distribution $P(\boldsymbol{x}, \boldsymbol{s} \mid a)$ not from the forward model p_θ, but from the data distribution and posterior model $q_\psi(\boldsymbol{x} \mid \boldsymbol{s}, \boldsymbol{a})$. This is also sensible because the variational MLE objective in

Equation (4.1) optimizes for compatible p_θ and q_ψ that both fit data and consistently describe (conditionals of) the same underlying distribution. Thus regularizing either can encourage a low MI. This approach leads to exactly Equation (C.1), without approximation.

Then, the total loss in Equation (4.3) from combining Equations (4.1) and (C.1) is given by

$$\begin{aligned}
&\min_\theta \mathcal{L}_{\mathsf{MLE}}(\theta) + c \cdot I(\boldsymbol{x}; \boldsymbol{s} \mid \boldsymbol{a}) \\
=&\min_{\theta,\theta',\psi} \mathcal{L}_{\mathsf{recon}}(\theta,\psi) + \mathcal{L}_{\mathsf{KL}\text{-}x}(\theta,\psi) + \mathcal{L}_{\mathsf{KL}\text{-}y}(\theta,\psi) + \mathcal{L}_{\mathsf{KL}\text{-}z} + c \cdot + \mathcal{L}_{\mathsf{KL}\text{-}x}(\theta',\psi) \\
=&\min_{\theta,\psi} \mathcal{L}_{\mathsf{recon}}(\theta,\psi) + (1+c) \cdot \mathcal{L}_{\mathsf{KL}\text{-}x}(\theta,\psi) + \mathcal{L}_{\mathsf{KL}\text{-}y}(\theta,\psi) + \mathcal{L}_{\mathsf{KL}\text{-}z}(\theta,\psi).
\end{aligned}$$

C.1.2 Discussions

We discuss some algorithmic choices of Denoised MDP below. Specific implementation details (e.g., architectures) can be found at Appendix C.2.1.

Posterior distributions of r_x and r_y. The p_θ reward distributions $p_\theta(r_x \mid x_t)$ and $p_\theta(r_y \mid y_t)$ are modelled via Gaussians (as is done usually in world models, such as Dreamer [63]). By the transition structure of Denoised MDPs, these distributions are inherently independent. Recall that $r = r_x + r_y$. Therefore, we can easily compute the distribution of $p_\theta(r \mid x_t, y_t)$ and its log likelihoods. This enables easy optimization of the variational MLE objective, without requiring the posterior model to also infer r_x and r_y from observed r subject to the addition relation.

Partial observability. Sections 4.2 and 4.3 discussions are mostly based in the fully observable setting. Yet most benchmarks and real-world tasks are partially observable, e.g., robot joint speeds that can not be inferred from a single frame. Fortunately, the transition models used in Denoised MDP are fully capable of handle such cases, as long as the encoder q_ψ is not deterministic and the observation model $p_\theta(s \mid \dots)$ does not have the block structure [38] (which would make x, y, z fully determined from s). In practice, we let both components to be generic conditional distributions

		$\textbf{Ctrl} + \textbf{Rew}$	$\textbf{Ctrl} + \overline{\textbf{Rew}}$	$\overline{\textbf{Ctrl}} + \textbf{Rew}$	$\overline{\textbf{Ctrl}} + \overline{\textbf{Rew}}$
DMC	**Noiseless**	Agent	—	—	—
	Video Background	Agent	—	—	Background
	Video Background + Noisy Sensor	Agent	—	Background	—
	Video Background + Camera Jittering	Agent	—	—	Background, Jittering camera
`RoboDesk`		Agent, Button, Light on desk, Green hue of TV	Blocks on desk, Handle on desk, Other movable objects	TV content, Button sensor noise	Jittering and flickering environment lighting, Jittering camera

Table C.1: Categorization of various information in the environments we evaluated with.

(parameterized by regular deep neural networks). Therefore, Denoised MDP does not require full observability.

Hyperparameter choice. The loss in Equation (4.4) has two hyperparameters: $\alpha \in (0, \infty)$ and $\beta \in (0, 1)$. To maintain relative ratio with the observation reconstruction loss, we recommend scaling α roughly proportionally with dimensionality of the observation space, as is done in our experiments presented in this paper. A smaller β means stronger regularization. Therefore, β can be chosen based on training stability and the level of noise distractors in the task.

C.2 Experiment Details

All code (including code for our environment variants and code for our Denoised MDP method) will be released upon publication.

C.2.1 Implementation Details

Environments and Tasks

In all environments, trajectories are capped at 1000 timesteps. Table C.1 shows a summary of what kinds of information exist in each environment.

DeepMind Control Suite (DMC). Our **Video Background** implementation follows Deep Bisimulation for Control [176] on most environments, using `Kinetics-400` grayscale videos [140], and replacing pixels where blue channel is strictly the greatest of three. This method, however, does not cleanly remove most of background in the Walker Walk environment, where we use an improved mask that replaces all pixels where the blue channel is *among the greatest* of three. For **Camera Jittering**, we shift the observation image according to a smooth random walk, implemented as, at each step, Gaussian-perturbing acceleration, decaying velocity, and adding a pulling force if the position is too far away from origin. For **Sensor Noise**, we select one sensor, and perturb it according to intensity of a patch of the natural video background (i.e., adding average patch value -0.5). We perturb the `speed` sensor for Cheetah Run, the `torso_height` sensor for Walker Walk, and the normalized `finger_to_target_dist` sensor for Reacher Easy. These sensor values undergo non-linear (mostly piece-wise linear) transforms to compute rewards. While they can not be perfectly modelled by additive reward noise, such a model is usually sufficient in most cases when the sensor values are not too extreme and stay in one linear region.

`RoboDesk`. We modify the original `RoboDesk` environment by adding a TV screen and two neighboring desks. The TV screen places (continuously horizontally shifting) natural RGB videos from the `Kinetics-400` dataset [140]. The environment has three light sources from the above, to which we added random jittering and flickering. The viewing camera is placed further to allow better view of the noise distractors. Resolution is increased from 64×64 to 96×96 to compensate this change. Camera jittering is implemented by a 3D smooth random walk. Finally, the button sensor (i.e., detected value of how much the button is pressed) is also offset by a random walk. Each of the three button affects the corresponding light on the desk. Additionally, pressing the green button also shifts the TV screen content to a green hue. Following `RoboDesk` reward design, we reward the agent for (1) placing arm close to the button, (2) pressing the button, and (3) how green the TV screen content is.

Operator	Input Shape	Kernel Size	Stride	Padding
Input	$[3, 96, 96]$	—	—	—
Conv. + ReLU	$[k, 47, 47]$	4	2	0
Conv. + ReLU	$[2k, 22, 22]$	4	2	0
Conv. + ReLU	$[4k, 10, 10]$	4	2	0
Conv. + ReLU	$[8k, 4, 4]$	4	2	0
Conv. + ReLU	$[8k, 2, 2]$	3	1	0
Reshape + FC	$[m]$	—	—	—

Table C.2: Encoder architecture for (96×96)-resolution observation. The output of this encoder is then fed to other network for inferring posteriors. m and k are two architectural hyperparameters. m controls the output size (unrelated to the actual latent variable sizes). k controls the network width.

Operator	Input Shape	Kernel Size	Stride	Padding
Input	[`input_size`]	—	—	—
FC + ReLU + Reshape	$[m, 1, 1]$	—	—	—
Conv. Transpose + ReLU	$[4k, 3, 3]$	5	2	0
Conv. Transpose + ReLU	$[4k, 9, 9]$	5	2	0
Conv. Transpose + ReLU	$[2k, 21, 21]$	5	2	0
Conv. Transpose + ReLU	$[k, 46, 46]$	6	2	0
Conv. Transpose + ReLU	$[3, 96, 96]$	6	2	0

Table C.3: Decoder architecture for (96×96)-resolution observation. m and k are two architectural hyperparameters. m controls width the fully connected part. k controls width of the convolutional part. They are the same values as in Table C.2.

`RoboDesk` Joint Position Regression Datasets. To generate training and test set, we use four policies trained by state-space SAC at different stages of training (which is not related to any of the compared methods) and a uniform random actor, to obtain five policies of different qualities. For each policy, we sample 100 trajectories, each containing 1001 pairs (from 1000 interactions) of image observation and groundtruth joint position (of dimension 9). This leads to a total of 500.5×10^3 samples from each policy. From these, 100×10^3 samples are randomly selected as test set. Training sets of sizes $5 \times 10^3, 10 \times 10^3, 25 \times 10^3, 50 \times 10^3, 100 \times 10^3, 150 \times 10^3$ are sampled from the rest. For all test sets and training sets, we enforce each policy to strictly contribute an equal amount.

Model Learning Methods

For all experiments, we let the algorithms use 10^6 environment steps. For PI-SAC and CURL, we follow the original implementations [97, 100] and use an action repeat of 4 for Cheetah Run and Reacher Easy, and an action repeat of 2 for Walker Walk. For Denoised MDP, Dreamer, TIA and DBC, we always use an action repeat of 2, following prior works [63, 45, 176].

Denoised MDP, Dreamer, and TIA. Both Dreamer and TIA use the same training schedule and the Recurrent State-Space Model (RSSM) as the base architecture [64]. Following them, Denoised MDP also uses these components, and follow the same prefilling and training schedule (see Dreamer [64] for details). These three model learning methods take in 64×64 RGB observations for DMC, and 96×96 RGB observations for `RoboDesk`. Dreamer only implements encoder and decoder for the former resolution. To handle the increased resolution, we modify the 64×64 architectures and obtain convolutional encoder and decoder shown in Tables C.2 and C.3. For fair comparison, we ensure that each method has roughly equal number of parameters by using different latent variable sizes, encoder output sizes (m of Table C.2) and convolutional net widths (k of Table C.3). Details are shown in Table C.4.

KL clipping (free nats). For Denoised MDP, we follow Dreamer [64, 63] and TIA [45], and allow 3 free nats for the $\mathcal{L}_{\mathsf{KL}\text{-}x}$ term. In other words, for each element of a batch, we do not optimize the KL term if it is less than 3 (e.g., implemented via clipping). However, we do not allow this for the $\mathcal{L}_{\mathsf{KL}\text{-}y}$ and $\mathcal{L}_{\mathsf{KL}\text{-}z}$ terms, as these variables are to be discarded and information is not allowed to hide in them unless permitted by the structure. An alternative strategy, which we find also empirically effective, is to consider $\mathcal{L}_{\mathsf{KL}\text{-}x} = \underbrace{\beta \cdot \mathcal{L}_{\mathsf{KL}\text{-}x}}_{\text{VAE KL term}} + \underbrace{(1-\beta) \cdot \mathcal{L}_{\mathsf{KL}\text{-}x}}_{\text{MI regularizer term}}$, and to allow free nats only for the first term that is a part of the variational model fitting objective. All results presented in this paper use the first strategy. Both strategies are implemented in our open source code repository: `github.com/facebookresearch/denoised_mdp`.

Policy Optimization Algorithms Used with Model Learning

Backpropagate via Dynamics. We use the same setting as Dreamer [63], optimizing a λ-return over 15-step-long rollouts with $\lambda = 0.95$, clipping gradients with norm greater than 100. TIA uses the same strategy, except that it groups different models together for gradient clipping. We strictly follow the official TIA implementation.

	DMC				RoboDesk			
	Latent Sizes	m	k	Total Number of Parameters	Latent Sizes	m	k	Total Number of Parameters
Dreamer	$(220+33)$	1024	32	7,479,789	$(220+33)$	1024	32	6,385,511
TIA	$(120+20)+(120+20)$	490	24	7,475,567	$(120+20)+(120+20)$	490	24	6,384,477
Denoised MDP	$(120+20)+(120+20)$	1024	32	7,478,826	$(120+20)+(120+20)$	1024	32	6,384,248

Table C.4: The specific architecture parameters for model learning methods. Since RSSM uses a deterministic part and a stochastic part to represent each latent variable, we use (`deterministic_size` + `stochastic_size`) to indicate size of a latent variable. TIA and Denoised MDP have more than one latent variable. Note that while TIA has lower m and k, it has multiple encoder and decoders, whereas Dreamer and Denoised MDP only have one encoder and one decoder. The total number of parameters is measured with the actor model, but without any additional components from policy optimization algorithm (e.g., critics in SAC). Total number of parameters is lower for `RoboDesk` as the encoder and decoder architecture is narrower than those of DMC for the purpose of reducing memory usage, despite with a higher resolution.

Latent-Space SAC. We use the regular SAC with automatic entropy tuning, without gradient clipping. This works well for almost all settings, except for Walker Walk variant of DMC, where training often collapses after obtaining good return, regardless of the model learning algorithm. To address instability in this case, we reduce learning rates from 3×10^{-4} to 1×10^{-4} and clip gradients with norm greater than 100 for all latent-space SAC run on these variants.

Model-Free Methods

DBC. For DMC, we used 84×84-resolution observation following original work (even though other methods train on 64×64-resolution observations). For `RoboDesk`, DBC uses the encoder in Table C.2 for 96×96-resolution observation, for fair comparison with other methods. Following the original work, we stack 3 consecutive frames to approximate the required full observability. In the robot arm joint position regression experiment Section 4.5.1, DBC encoders also see stacked observations. For DMC evaluations, we use the data provided by Zhang et al. wherever possible, and run the official repository for other cases.

State-Space SAC. The state space usually contains robot joint states, including position, velocity, etc. For DMC, when **Sensor Noise** is present, this is not the true optimal state space, as we do not supply it with the noisy background that affects the noisy reward. However, it still works well in practice. For `RoboDesk`, the TV's effect on reward is likely stronger and direct state-space SAC fails to learn. Since this evaluation is to obtain a rough "upper bound", we train state-space SAC with a modified reward with less noise— the agent is rewarded by pressing the button, independent of the TV content. This still encourages the optimal strategy of the task allows achieving good policies.

Non-RL methods

Contrastive Learning. We used the Alignment+Uniformity contrastive learning loss from Wang and Isola [168]. The hyperparameters and data augmentations strictly follow their experiments on STL-10 [31], which also is of resolution 96×96. The exact loss form is $\mathcal{L}_{\mathsf{align}}(\alpha = 2) + \mathcal{L}_{\mathsf{uniform}}(t = 2)$, a high-performance setting for STL-10.

C.2.2 Compute Resources

All our experiments are run on a single GPU, requiring 8GB memory for DMC tasks, and 16GB memory for `RoboDesk` tasks. We use NVIDIA GPUs of the following types: 1080 Ti, 2080 Ti, 3080 Ti, P100, V100, Titan XP, Titan RTX. For `MuJoCo` [155], we use the `EGL` rendering engine. Training time required for each run heavily depends on the CPU specification and availability. In general, a Denoised MDP run needs $12 \sim 36$ hours on DMC and $24 \sim 50$ hours on `RoboDesk`. TIA uses about $1.5\times$ of these times, due to the adversarial losses. For a comparison between the two Denoised MDP variants, running the same DMC task on the same machine, the Figure 4-2b variant used 23 hours while the Figure 4-2c variant used 26 hours.

C.2.3 Visualization Details

Visualizations of components in learned models. We use different methods to visualize signal and noise information learned by TIA and Denoised MDP in Figures 4-4 and 4-7. For TIA, we used the reconstructions from the two latent (before mask-composing them together as the full reconstruction). For Denoised MDP, we only have one decoder (instead of three for TIA), and thus we decode $(x_t, \texttt{const})$ and $(\texttt{const}, y_t)$ to visualize information contained in each variable, with `const` chosen by visual clarity (usually as value of the other variable at a fixed timestep). Due to the fundamental different ways to obtain these visualizations, in DMC, TIA can prevent the agent from showing up in noise visualizations, while Denoised MDP cannot. However, as stated in Section 4.5.2, our focus should be on what evolves/changes in these images, rather than what is visually present, as static components are essentially not modelled by the corresponding transition dynamics. Visualizations in Figures 4-4 and 4-7 use trajectories generated by a policy trained with state-space SAC. To obtain diverse behaviors, policy outputs are randomly perturbed before being used as actions. From the same trajectory, we use the above described procedure to obtain visualizations. The specific used trajectory segments are chosen to showcase both the modified environment and representative behavior of each method. Please see the supplementary video for clearer visualizations.

C.2.4 RoboDesk Result Details

Environment modifications. The agent controls a robotic arm placed in front of a desk and a TV, and is tasked to push down the green button on the desk, which turns on a small green light and makes the TV display have a green hue. The intensity of the TV image's green channel is given to the agent as part of their reward, in addition to distance between the arm to the button, and how much the button is pressed. Additionally, the environment contains other noise distractors, including moveable blocks on the desk ($\textbf{Ctrl} + \overline{\textbf{Rew}}$), flickering environment light and camera jittering ($\overline{\textbf{Ctrl}} + \overline{\textbf{Rew}}$), TV screen hue ($\textbf{Ctrl} + \textbf{Rew}$), TV content ($\overline{\textbf{Ctrl}} + \textbf{Rew}$), and noisy button sensors ($\overline{\textbf{Ctrl}} + \textbf{Rew}$).

Denoised MDP hyperparameters. RoboDesk has roughly twice as many pixels as DMC has. For Denoised MDP, we scale α with the observation space dimensionality (see Section 4.3) and use $\alpha = 2$, with a fixed $\beta = 0.125$. When using the alternative

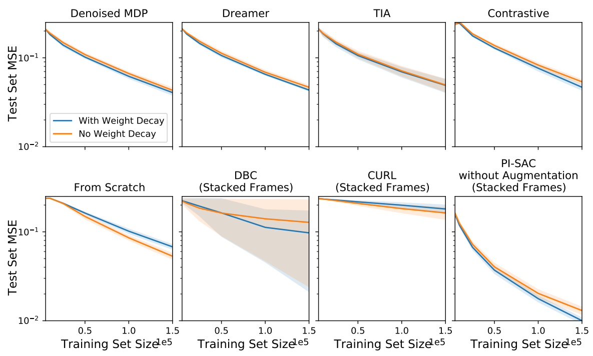


Figure C-1: Effect of weight decay on `RoboDesk` joint position regression. The curves show final test MSE for various training set sizes. Weight decay generally helps when finetuning from a pretrained encoder, but hurts when training from scratch.

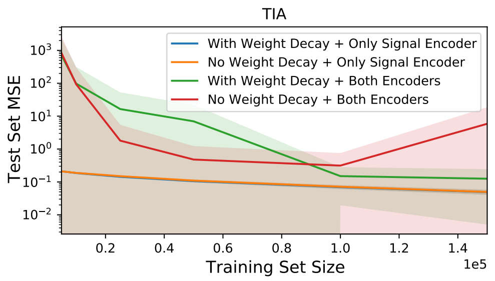


Figure C-2: Performance of all TIA settings on `RoboDesk` joint position regression. Only using the signal encoder is necessary for good performance.

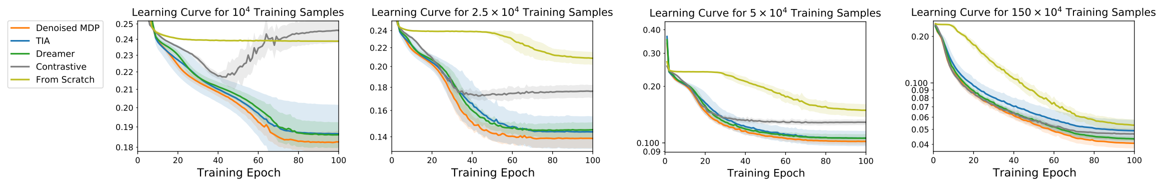


Figure C-3: Training curve comparisons for the `RoboDesk` joint position regression task across many training set sizes.

KL free nats strategy discussed in Appendix C.2.1 (results not shown in paper), we find $\alpha = 1$ and $\beta = 0.25$ also effective.

TIA hyperparameters. We follow recommendations in the TIA paper, setting $\lambda_{\text{Radv}} = 25{,}000$ to match reconstruction loss in magnitude, and setting $\lambda_{O_s} = 2$ where training is stable.

Robot Arm Joint Position Regression.

Training details. For this task, we jointly train the pre-trained backbone and a three-layer MLP head that has 256 hidden units at each layer, with a learning rate of 8×10^{-5}. For finetuning from pretrained encoders, we follow common finetuning practice and apply a weight decay of 3×10^{-5} whenever it is helpful (all cases except CURL and training from scratch). See Figure C-1 for comparisons for weight decay options over all methods.

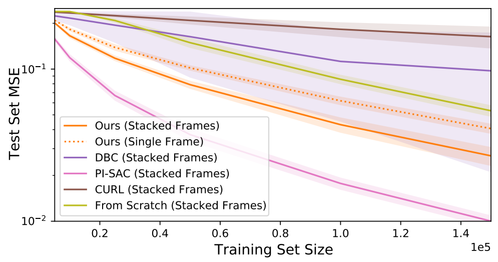


Figure C-4: Performance comparison of finetuning from Denoised MDP encoders and frame-stacked encoders that take in 3 consecutive frames. For Denoised MDP and training from scratch, the encoders *take in only a single frame* and are applied for each of the frame, with output concatenated together before feeding to the prediction head.

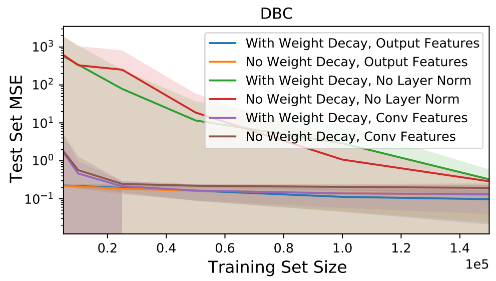


Figure C-5: Performance of all DBC settings on `RoboDesk` joint position regression. Using the output features (after layer normalization) is necessary for good performance.

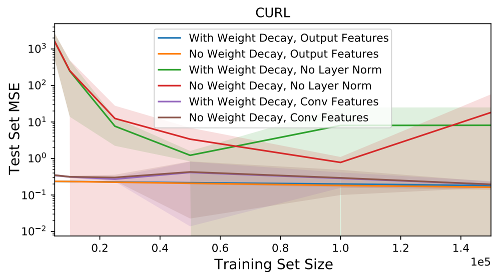


Figure C-6: Performance of all CURL settings on `RoboDesk` joint position regression. Using the output features (after layer normalization) is necessary for good performance.

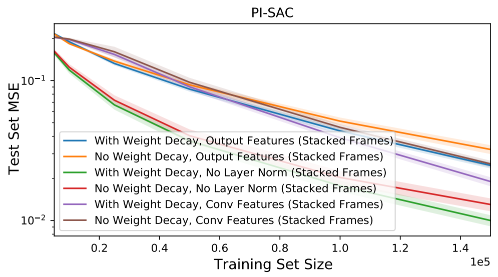


Figure C-7: Performance of all PI-SAC settings on `RoboDesk` joint position regression. Using the activations *before layer normalization* gives best performance.

- For model-based RL, we take encoders trained with backpropagating via dynamics as the policy optimization algorithm.

- In training the contrastive encoder, for a (more) fair comparison with RL-trained encoders that are optimized over 10^6 environment steps, we train contrastive encoders on 10^6 samples, obtained in the exact same method of the training sets of this task. In a sense, these contrastive encoders have the advantage of training on the exact same distribution, and seeing more samples (since RL-trained encoders use action repeat of 2 and thus only ever see 0.5×10^6 samples).

- TIA has two sets of encoders. Using concatenated latents from both unfortunately hurts performance greatly (see Figure C-2). So we use only the encoder for the signal latent.

We also compare training speeds over a wide range of training set sizes in Figure C-3. Denoised MDP encoders lead to faster and better training in all settings.

Additional comparison with frame-stacking encoders. Other pretrained encoders (DBC, CURL and PI-SAC) take in stacked 3 consecutive frames, and are not directly comparable with the other methods. To compare, we also try running Denoised MDP encoders on the 3 consecutive frames, whose feature vector is concatenated before feeding into the head. The result in Figure C-4 shows that our encoder outperforms all but PI-SAC encoders. Finally, for DBC, CURL and PI-SAC, we attempted evaluating intermediate features, features before the final layer normalization, and the output space, and find the last option best-performing for DBC and CURL, and the second option best-performing for PI-SAC (see Figures C-5 to C-7). Therefore, we use these respective spaces, which arguably gives a further edge to these methods, as we essentially tune this additional option on test results. Notably, these respective choices are often the only one achieving relatively good performance, highlighting the necessity of tuning for these methods.

C.2.5 DeepMind Control Suite (DMC) Result Details

Full policy optimization results. Figure C-8 presents the full results on each DMC environment (task + variant). For environment, a comparison plot is made based on which policy learning algorithm is used with the model learning method (with model-free baselines duplicated in both). Such separation is aimed to highlight the performance difference caused by model structure (rather than policy learning algorithm). Across most noisy environments, Denoised MDP performs the best. It also achieves high return on noiseless environments.

Visualization of learned models. Figure C-9 is the extended version of Figure 4-7 in main text, with full reconstructions from all three models. Please see the supplementary video for clearer visualizations.

Comparison between Denoised MDP variants. We compare the two Denoised MDP variants based Figures 4-2b and 4-2c on Cheetah Run environments with policy trained by packpropagating via learned dynamics. The comparison is shown in the top row of Figure C-8, where we see the Figure 4-2b variant often performing a bit better. We hypothesize that this may due to the more complex prior and posterior structure of Figure 4-2c, which may not learn as efficiently. This also makes Figure 4-2c variant needing longer (wall-clock) time to optimize, as mentioned above in Appendix C.2.2.

TIA hyperparameters and instability. We strictly follow recommendations of the original paper, and use their suggested value for each DMC task. We also note that TIA runs sometimes collapse during training, leading to sharp drops in rewards. After closely inspecting the models before and after collapses, we note that in many cases, such collapses co-occur with sudden spikes in TIA's reward disassociation loss, which is implemented as an adversarial minimax loss, and the noise latent space instantly becomes degenerate (i.e., not used in reconstruction). We hypothesize that this adversarial nature can cause training instability. However, a few collapses do not co-occur with such loss spikes, which maybe alternatively due to that TIA model structure cannot model the respective noise types and that better fitting the model naturally means a degenerate noise latent space.

PI-SAC hyperparameters. For each task, we use the hyperparameters detailed in the original paper [100]. PI-SAC is usually run with augmentations. However, unlike CURL, augmentation is not an integral part of the PI-SAC algorithm and is completely optional. For a fair comparisons with other methods and to highlight the effect of the predictive information regularizer, the main mechanism proposed by PI-SAC, we do not use augmentations for PI-SAC.

Denoised MDP hyperparameters. For DMC, we always use fixed $\alpha = 1$. β can be tune according to amount of noises in environment, and to training stability. In Figure C-10, we compare effects of choosing different β's. On noiseless environments, larger β (i.e., less regularization) performs often better. Whereas on noisy environments,

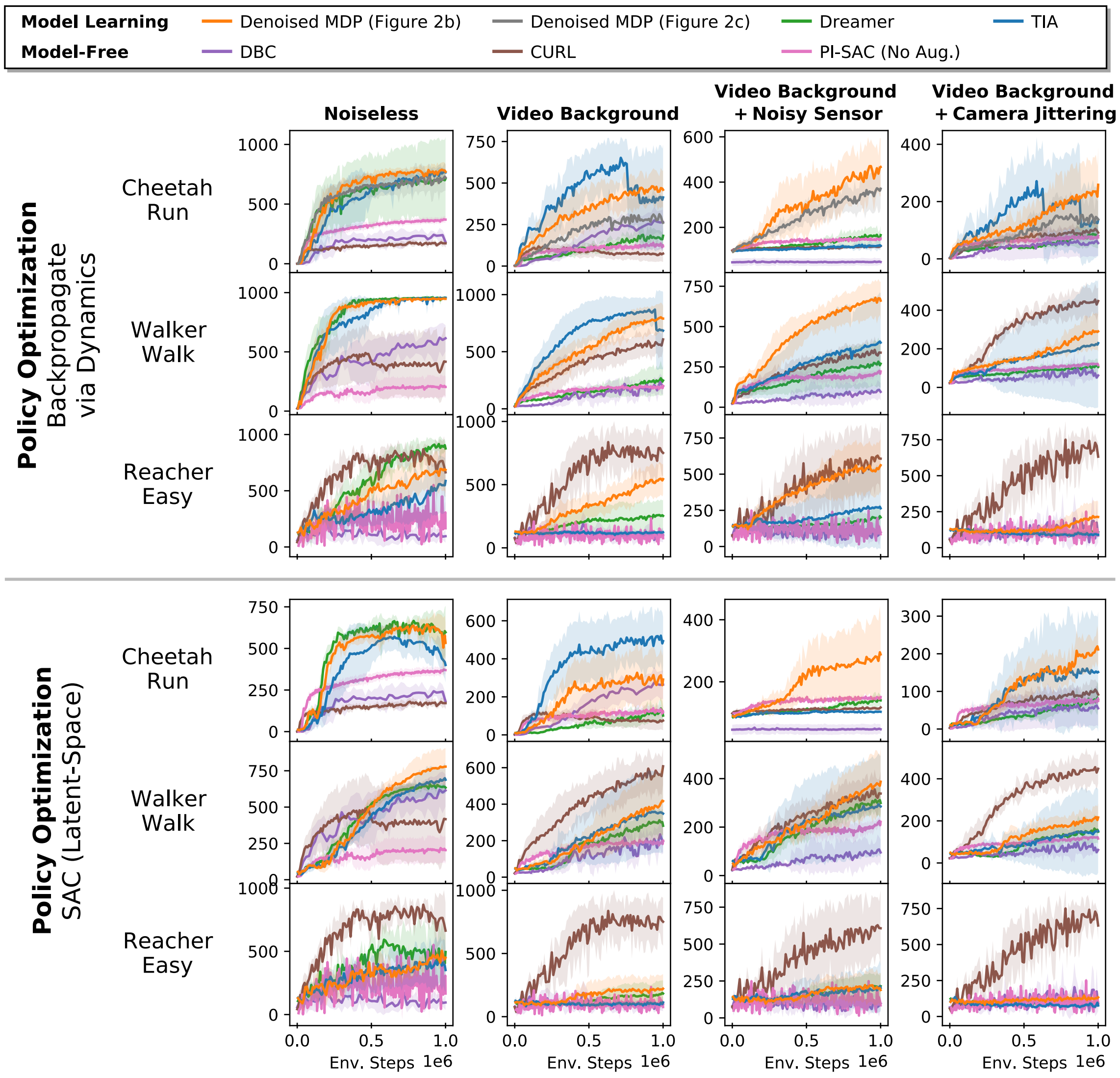


Figure C-8: Policy optimization results on DMC. Each plot focuses on a single task variant, showing total episode return versus environment steps taken. For three model-based approaches, we use two policy optimization choices to train on the learned model: **(top half)** backpropagate via learned dynamics and **(bottom half)** SAC on the learned MDP. We also compare with DBC, a model-free baseline. For an "upper bound" (not plotted due to presentation clarity), SAC on true state-space (i.e., optimal representation) in 10^6 environment steps reaches episode return ≈ 800 on Cheetah Run variants, ≈ 980 on Walker Walk variants, and ≈ 960 on Reacher Easy variants. CURL's specific augmentation choice (random crop) potentially helps significantly for Reacher Easy (where the reacher and the target appear in random spatial locations) and **Camera Jittering**. However, unlike Denoised MDP, it does not generally perform well across all environments and noise variants.

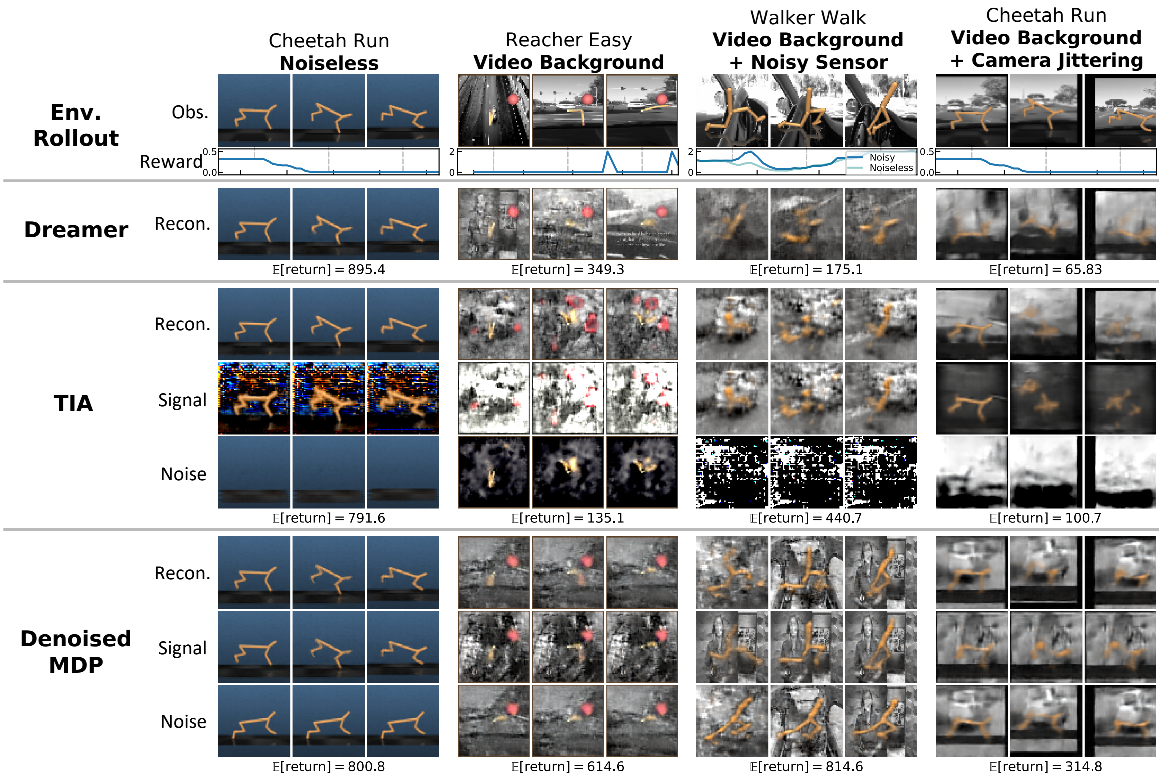

Cheetah Run
Noiseless
Reacher Easy
Video Background
Walker Walk
Video Background
+ Noisy Sensor
Cheetah Run
Video Background
+ Camera Jittering
Env.
Rollout
Obs.
Reward
0.5
0.0
2
0
Noisy
Noiseless
Dreamer
Recon.
$\mathbb{E}$[return] = 895.4
$\mathbb{E}$[return] = 349.3
$\mathbb{E}$[return] = 175.1
$\mathbb{E}$[return] = 65.83
TIA
Recon.
Signal
Noise
$\mathbb{E}$[return] = 791.6
$\mathbb{E}$[return] = 135.1
$\mathbb{E}$[return] = 440.7
$\mathbb{E}$[return] = 100.7
Denoised
MDP
Recon.
Signal
Noise
$\mathbb{E}$[return] = 800.8
$\mathbb{E}$[return] = 614.6
$\mathbb{E}$[return] = 814.6
$\mathbb{E}$[return] = 314.8

Figure C-9: Complete visualization of the different DMC variants and factorizations learned by TIA and Denoised MDP. In addition to visualizations of Figure 4-7, we also visualize full reconstructions from Dreamer, TIA, and Denoised MDP.

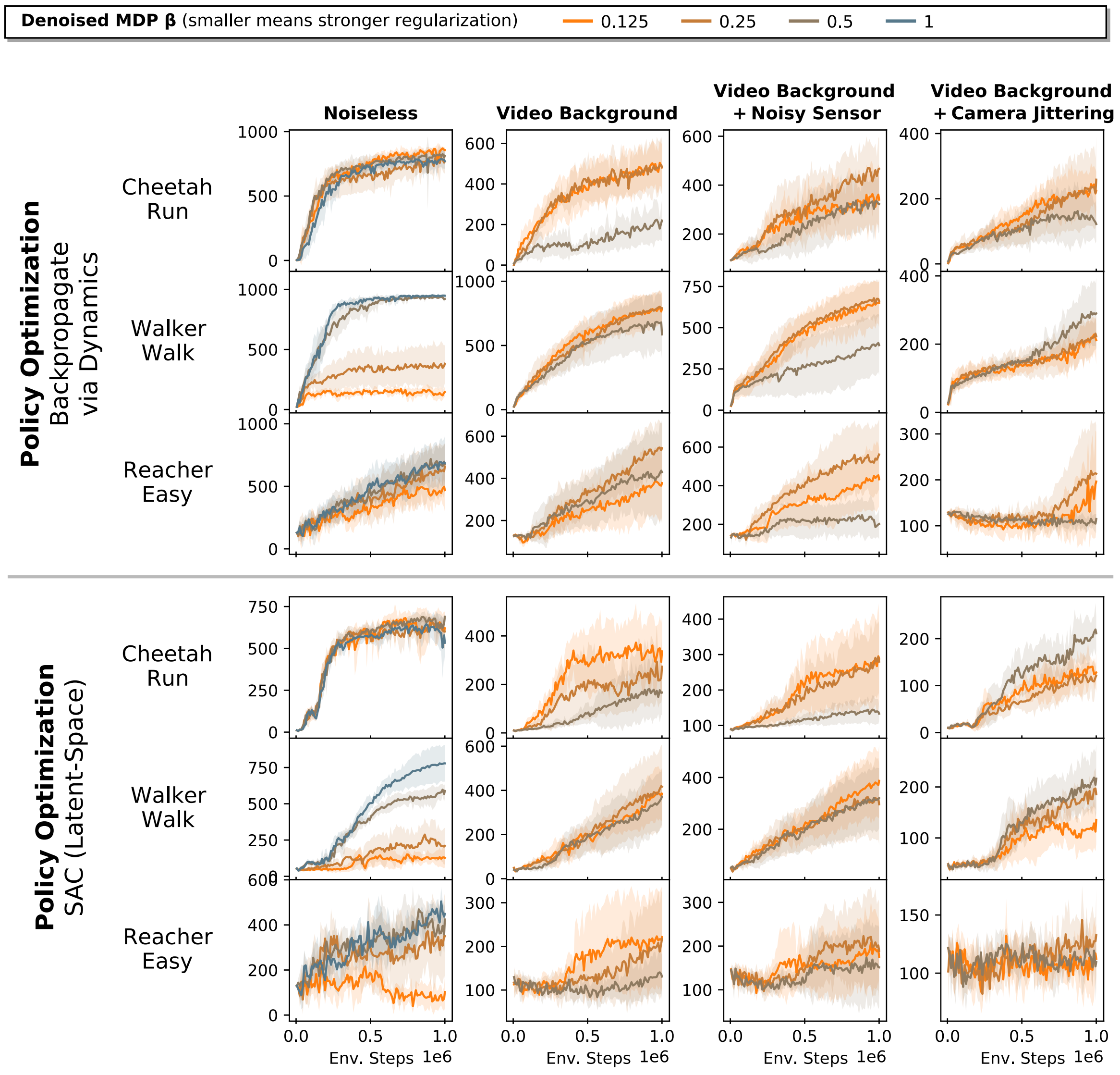


Figure C-10: Effect of choosing β in Denoised MDP on DMC policy optimization results. Setting $\beta = 1$ disables regularization and is *only run on noiseless variants.*

		Noiseless	Video Background	Video Background + Noisy Sensor	Video Background + Camera Jittering
Policy Learning: Backprop via Dynamics	Cheetah Run	1	0.125	0.25	0.25
	Walker Walk	1	0.25	0.25	0.5
	Reacher Easy	1	0.25	0.25	0.25
Policy Learning: SAC (Latent-Space)	Cheetah Run	1	0.125	0.125	0.25
	Walker Walk	1	0.25	0.125	0.5
	Reacher Easy	1	0.125	0.25	0.25

Table C.5: β choices for Denoised MDP results shown in Table 4.1 and Figure C-8. We choose $\beta = 1$ (i.e., disabling regularization) for all noiseless environments, and tuned others. However, as seen in Figure C-10, the results often are not too sensitive to small β changes.

sometimes stronger regularization can boost performance. However, overall good performance can be obtained by usually several β values. In Table C.5, we summarize our β choices for each environment in Table C.5.

Bibliography

[1] Alekh Agarwal, Sham Kakade, Akshay Krishnamurthy, and Wen Sun. FLAMBE: Structural complexity and representation learning of low rank mdps. *arXiv preprint arXiv:2006.10814*, 2020.

[2] Ibrahim Ahmad and Pi-Erh Lin. A nonparametric estimation of the entropy for absolutely continuous distributions (corresp.). *IEEE Transactions on Information Theory*, 22(3):372–375, 1976.

[3] Noga Alon and Joel H Spencer. *The probabilistic method.* John Wiley & Sons, 2004.

[4] Brandon Amos, Lei Xu, and J Zico Kolter. Input convex neural networks. In *International Conference on Machine Learning*, pages 146–155. PMLR, 2017.

[5] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Ruslan Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. *arXiv preprint arXiv:1904.11955*, 2019.

[6] Sanjeev Arora, Hrishikesh Khandeparkar, Mikhail Khodak, Orestis Plevrakis, and Nikunj Saunshi. A theoretical analysis of contrastive unsupervised representation learning. *arXiv preprint arXiv:1902.09229*, 2019.

[7] Philip Bachman, R Devon Hjelm, and William Buchwalter. Learning representations by maximizing mutual information across views. In *Advances in Neural Information Processing Systems*, pages 15509–15519, 2019.

[8] Ananth Balashankar and Lakshminarayanan Subramanian. Learning faithful representations of causal graphs. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 839–850, 2021.

[9] Marc Bellemare, Will Dabney, Robert Dadashi, Adrien Ali Taiga, Pablo Samuel Castro, Nicolas Le Roux, Dale Schuurmans, Tor Lattimore, and Clare Lyle. A geometric perspective on optimal representations for reinforcement learning. *Advances in neural information processing systems*, 32:4358–4369, 2019.

[10] Yoshua Bengio et al. Quick training of probabilistic neural nets by importance sampling.

[11] Umberto Bertele and Francesco Brioschi. On non-serial dynamic programming. *J. Comb. Theory, Ser. A*, 14(2):137–148, 1973.

[12] Dimitri P Bertsekas and John N Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991.

[13] S Bochner. Monotone funktionen, stieltjessche integrale und harmonische analyse. *Collected Papers of Salomon Bochner*, 2:87, 1992.

[14] Kenneth P Bogart. Maximal dimensional partially ordered sets i. hiraguchi's theorem. *Discrete Mathematics*, 5(1):21–31, 1973.

[15] Piotr Bojanowski and Armand Joulin. Unsupervised learning by predicting noise. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 517–526. JMLR. org, 2017.

[16] Béla Bollobás and Bollobás Béla. *Random graphs.* Number 73. Cambridge university press, 2001.

[17] Sergiy V Borodachov, Douglas P Hardin, and Edward B Saff. *Discrete energy on rectifiable sets.* Springer, 2019.

[18] Jean Bourgain. On lipschitz embedding of finite metric spaces in hilbert space. *Israel Journal of Mathematics*, 52(1-2):46–52, 1985.

[19] Barry Brown, James Lovato, and Kathy Russell. CDFLIB: library of fortran routines for cumulative distribution functions, inverses, and other parameters, 1994.

[20] Yury Brychkov. On some properties of the marcum q function. *Integral Transforms and Special Functions*, 23:177–182, 03 2012. doi: 10.1080/10652469.2011.573184.

[21] John Burkardt. C++ source code for CDFLIB. `https://people.sc.fsu.edu/~jburkardt/cpp_src/cdflib/cdflib.html`, 2021.

[22] Pablo Samuel Castro. Scalable methods for computing state similarity in deterministic markov decision processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 10069–10076, 2020.

[23] Moses Charikar, Konstantin Makarychev, and Yury Makarychev. Directed metrics and directed graph partitioning problems. In *SODA*, volume 6, pages 51–60. Citeseer, 2006.

[24] Patrick H Chen, Si Si, Sanjiv Kumar, Yang Li, and Cho-Jui Hsieh. Learning to screen for fast softmax inference on large vocabulary neural networks. 2018.

[25] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020.

[26] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020.

[27] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. GitHub repository `https://github.com/facebookresearch/moco/tree/78b69cafae80bc74cd1a89ac3fb365dc20d157d3`, 2020.

[28] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

[29] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179.

[30] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546. IEEE, 2005.

[31] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223, 2011.

[32] Henry Cohn and Abhinav Kumar. Universally optimal distribution of points on spheres. *Journal of the American Mathematical Society*, 20(1):99–148, 2007.

[33] Kenneth James Williams Craik. *The nature of explanation*, volume 445. CUP Archive, 1952.

[34] Tim R. Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M. Tomczak. Hyperspherical variational auto-encoders. *34th Conference on Uncertainty in Artificial Intelligence (UAI-18)*, 2018.

[35] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

[36] Daniel C Dennett. Why the law of effect will not go away. *Journal for the Theory of Social Behaviour*, 1975.

[37] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655. PMLR, 2014.

[38] Simon Du, Akshay Krishnamurthy, Nan Jiang, Alekh Agarwal, Miroslav Dudik, and John Langford. Provably efficient rl with rich observations via latent state decoding. In *International Conference on Machine Learning*, pages 1665–1674. PMLR, 2019.

[39] Yonathan Efroni, Dipendra Misra, Akshay Krishnamurthy, Alekh Agarwal, and John Langford. Provable rl with exogenous distractors via multistep inverse dynamics. *arXiv preprint arXiv:2110.08847*, 2021.

[40] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[41] Paul Erdős and Alfréd Rényi. On random graphs. i. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[42] Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. Robust predictable control. *arXiv preprint arXiv:2109.03214*, 2021.

[43] Stefan Felsner, Ching Man Li, and William T. Trotter. Adjacency posets of planar graphs. *Discrete Mathematics*, 310(5):1097–1104, 2010. ISSN 0012-365X.

[44] Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite markov decision processes. In *UAI*, volume 4, pages 162–169, 2004.

[45] Xiang Fu, Ge Yang, Pulkit Agrawal, and Tommi Jaakkola. Learning task informed abstractions. In *International Conference on Machine Learning*, pages 3480–3491. PMLR, 2021.

[46] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[47] Octavian Ganea, Gary Bécigneul, and Thomas Hofmann. Hyperbolic entailment cones for learning hierarchical embeddings. In *International Conference on Machine Learning*, pages 1646–1655. PMLR, 2018.

[48] Amnon Geifman, Abhay Yadav, Yoni Kasten, Meirav Galun, David Jacobs, and Ronen Basri. On the similarity between the laplace and neural tangent kernels. *arXiv preprint arXiv:2007.01580*, 2020.

[49] Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G Bellemare. Deepmdp: Learning continuous latent space models for representation learning. In *International Conference on Machine Learning*, pages 2170–2179. PMLR, 2019.

[50] Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence*, 147 (1-2):163–223, 2003.

[51] Joshua Goodman. Classes for fast maximum entropy training. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 01CH37221)*, volume 1, pages 561–564. IEEE, 2001.

[52] Mario Götz and Edward B Saff. Note on d—extremal configurations for the sphere in r d+1. In *Recent Progress in Multivariate Approximation*, pages 159–162. Springer, 2001.

[53] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[54] Edouard Grave, Armand Joulin, Moustapha Cissé, Hervé Jégou, et al. Efficient softmax approximation for gpus. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1302–1310. JMLR. org, 2017.

[55] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[56] Peter Grunwald and Paul Vitányi. Shannon information and kolmogorov complexity. *arXiv preprint cs/0410002*, 2004.

[57] Shuyang Gu, Jianmin Bao, Hao Yang, Dong Chen, Fang Wen, and Lu Yuan. Mask-guided portrait editing with conditional gans. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3436–3445, 2019.

[58] Matthieu Guillot and Gautier Stauffer. The stochastic shortest path problem: a polyhedral combinatorics perspective. *European Journal of Operational Research*, 285(1):148–158, 2020.

[59] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304, 2010.

[60] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[61] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[62] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference*

on Computer Vision and Pattern Recognition (CVPR'06), volume 2, pages 1735–1742. IEEE, 2006.

[63] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[64] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. PMLR, 2019.

[65] David Hahn, Pol Banzet, James M Bern, and Stelian Coros. Real2sim: Visco-elastic parameter estimation from dynamic motion. *ACM Transactions on Graphics (TOG)*, 38(6):1–13, 2019.

[66] DP Hardin and EB Saff. Minimal riesz energy point configurations for rectifiable d-dimensional manifolds. *Advances in Mathematics*, 193(1):174–204, 2005.

[67] Md Hasnat, Julien Bohné, Jonathan Milgram, Stéphane Gentric, Liming Chen, et al. von mises-fisher mixture model-based deep learning: Application to face verification. *arXiv preprint arXiv:1706.04264*, 2017.

[68] Nozomi Hata, Shizuo Kaji, Akihiro Yoshida, and Katsuki Fujisawa. Nested subspace arrangement for representation of relational data. In *International Conference on Machine Learning*, pages 4127–4137. PMLR, 2020.

[69] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[70] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722*, 2019.

[71] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

[72] Olivier J Hénaff, Ali Razavi, Carl Doersch, SM Eslami, and Aaron van den Oord. Data-efficient image recognition with contrastive predictive coding. *arXiv preprint arXiv:1905.09272*, 2019.

[73] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[74] Irina Higgins, David Amos, David Pfau, Sebastien Racaniere, Loic Matthey, Danilo Rezende, and Alexander Lerchner. Towards a definition of disentangled representations. *arXiv preprint arXiv:1812.02230*, 2018.

[75] Toshio Hiraguchi. On the dimension of partially ordered sets. *The science reports of the Kanazawa University*, 1(2):77–94, 1951.

[76] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*, 2018.

[77] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *International workshop on similarity-based pattern recognition*, pages 84–92. Springer, 2015.

[78] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a unified classifier incrementally via rebalancing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 831–839, 2019.

[79] Minyoung Huh, Pulkit Agrawal, and Alexei A Efros. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016.

[80] Minyoung Huh, Hossein Mobahi, Richard Zhang, Brian Cheung, Pulkit Agrawal, and Phillip Isola. The low-rank simplicity bias in deep networks. *arXiv preprint arXiv:2103.10427*, 2021.

[81] Piotr Indyk. Algorithmic applications of low-distortion geometric embeddings. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 10–33. IEEE, 2001.

[82] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[83] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *arXiv preprint arXiv:1806.07572*, 2018.

[84] N. L. Johnson. On an extension of the connexion between poisson and χ^2 distributions. *Biometrika*, 46(3/4):352–363, 1959. ISSN 00063444.

[85] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

[86] Harini Kannan, Danijar Hafner, Chelsea Finn, and Dumitru Erhan. RoboDesk: A multi-task reinforcement learning benchmark. `https://github.com/google-research/robodesk`, 2021.

[87] Seung Wook Kim, Yuhao Zhou, Jonah Philion, Antonio Torralba, and Sanja Fidler. Learning to Simulate Dynamic Environments with GameGAN. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020.

[88] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[89] John Frank Charles Kingman. Poisson processes. *Encyclopedia of biostatistics*, 6, 2005.

[90] Ryan Kiros, Yukun Zhu, Russ R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302, 2015.

[91] Sosuke Kobayashi. Homemade bookcorpus. GitHub repository `https://github.com/soskek/bookcorpus/tree/5fe0cec8d7fd83940e48c799739496dc68ab2798`, 2019.

[92] Andrei N Kolmogorov. On tables of random numbers. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 369–376, 1963.

[93] Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

[94] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[95] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[96] Naum Samoĭlovich Landkof. *Foundations of modern potential theory*, volume 180. Springer, 1972.

[97] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. CURL: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*, pages 5639–5650. PMLR, 2020.

[98] Misha Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *Advances in Neural Information Processing Systems*, 33:19884–19895, 2020.

[99] Alex X Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *arXiv preprint arXiv:1907.00953*, 2019.

[100] Kuang-Huei Lee, Ian Fischer, Anthony Liu, Yijie Guo, Honglak Lee, John Canny, and Sergio Guadarrama. Predictive information accelerates learning in rl. *Advances in Neural Information Processing Systems*, 33:11890–11901, 2020.

[101] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[102] Ming Li, Paul Vitányi, et al. *An introduction to Kolmogorov complexity and its applications*, volume 3. Springer, 2008.

[103] Ralph Linsker. Self-organization in a perceptual network. *Computer*, 21(3): 105–117, 1988.

[104] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 212–220, 2017.

[105] Weiyang Liu, Rongmei Lin, Zhen Liu, Lixin Liu, Zhiding Yu, Bo Dai, and Le Song. Learning towards minimum hyperspherical energy. In *Advances in Neural Information Processing Systems*, pages 6222–6233. 2018.

[106] Lajanugen Logeswaran and Honglak Lee. An efficient framework for learning sentence representations. In *International Conference on Learning Representations*, 2018.

[107] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[108] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[109] Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 8(10), 2007.

[110] Lucas Manuelli, Wei Gao, Peter Florence, and Russ Tedrake. kpam: Keypoint affordances for category-level robotic manipulation. *arXiv preprint arXiv:1903.06684*, 2019.

[111] J. I. Marcum. *Table of Q Functions.* RAND Corporation, Santa Monica, CA, 1950.

[112] David A McAllester. Some pac-bayesian theorems. *Machine Learning*, 37(3): 355–363, 1999.

[113] Facundo Mémoli, Anastasios Sidiropoulos, and Vijay Sridhar. Quasimetric embeddings and their applications. *Algorithmica*, 80(12):3803–3824, 2018.

[114] Pascal Mettes, Elise van der Pol, and Cees Snoek. Hyperspherical prototype networks. In *Advances in Neural Information Processing Systems*, pages 1485–1495, 2019.

[115] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[116] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and Analysis of Online Social Networks. In *Proceedings of the 5th ACM/Usenix Internet Measurement Conference (IMC'07)*, San Diego, CA, October 2007.

[117] Aditya Modi, Nan Jiang, Ambuj Tewari, and Satinder Singh. Sample complexity of reinforcement learning using linearly combined model ensembles. In *International Conference on Artificial Intelligence and Statistics*, pages 2010–2020. PMLR, 2020.

[118] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[119] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[120] Giacomo Ortali and Ioannis G Tollis. Multidimensional dominance drawings. *arXiv preprint arXiv:1906.09224*, 2019.

[121] Sherjil Ozair, Yazhe Li, Ali Razavi, Ioannis Antonoglou, Aäron van den Oord, and Oriol Vinyals. Vector quantized models for planning. *arXiv preprint arXiv:2106.04615*, 2021.

[122] Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 115–124. Association for Computational Linguistics, 2005.

[123] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. 2015.

[124] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8026–8037. 2019.

[125] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

[126] Jeffrey Pennington, Samuel Schoenholz, and Surya Ganguli. The emergence of spectral universality in deep networks. In *International Conference on Artificial Intelligence and Statistics*, pages 1924–1932. PMLR, 2018.

[127] Silviu Pitis, Harris Chan, Kiarash Jamali, and Jimmy Ba. An inductive bias for distances: Neural nets that respect the triangle inequality. *arXiv preprint arXiv:2002.05825*, 2020.

[128] Ben Poole, Sherjil Ozair, Aaron Van Den Oord, Alex Alemi, and George Tucker. On variational bounds of mutual information. In *International Conference on Machine Learning*, pages 5171–5180. PMLR, 2019.

[129] DJ de S Price. *Networks of scientific papers*. Princeton University Press, 2011.

[130] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., USA, 1st edition, 1994. ISBN 0471619779.

[131] Iasonas Kokkinos Rıza Alp Güler, Natalia Neverova. Densepose: Dense human pose estimation in the wild. 2018.

[132] Fatemeh Salehi Rizi, Joerg Schloetterer, and Michael Granitzer. Shortest path distance approximation using deep learning techniques. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 1007–1014. IEEE, 2018.

[133] Neil Robertson and Paul D Seymour. Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984.

[134] Nikunj Saunshi, Orestis Plevrakis, Sanjeev Arora, Mikhail Khodak, and Hrishikesh Khandeparkar. A theoretical analysis of contrastive unsupervised representation learning. In *International Conference on Machine Learning*, pages 5628–5637, 2019.

[135] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[136] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pages 1312–1320. PMLR, 2015.

[137] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[138] Richard Serfozo. Convergence of lebesgue integrals with varying measures. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 380–402, 1982.

[139] J. G. Skellam. The frequency distribution of the difference between two poisson variates belonging to different populations. *Journal of the Royal Statistical Society. Series A (General)*, 109(Pt 3):296–296, 1946.

[140] Lucas Smaira, João Carreira, Eric Noland, Ellen Clancy, Amy Wu, and Andrew Zisserman. A short note on the kinetics-700-2020 human action dataset. *arXiv preprint arXiv:2010.10864*, 2020.

[141] Elizabeth S Spelke and Katherine D Kinzler. Core knowledge. *Developmental science*, 10(1):89–96, 2007.

[142] James Stewart. Positive definite functions and generalizations, an historical survey. *The Rocky Mountain Journal of Mathematics*, 6(3):409–434, 1976.

[143] Richard S Sutton. An adaptive network that constructs and uses and internal model of its world. *Cognition and Brain Theory*, 4(3):217–246, 1981.

[144] Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

[145] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

[146] Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768, 2011.

[147] Ryota Suzuki, Ryusuke Takahama, and Shun Onoda. Hyperbolic disk embeddings for directed acyclic graphs. In *International Conference on Machine Learning*, pages 6066–6075. PMLR, 2019.

[148] Pieter Merkus Lambertus Tammes. On the origin of number and arrangement of the places of exit on the surface of pollen-grains. *Recueil des travaux botaniques néerlandais*, 27(1):1–84, 1930.

[149] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[150] Joseph John Thomson. Xxiv. on the structure of the atom: an investigation of the stability and periods of oscillation of a number of corpuscles arranged at equal intervals around the circumference of a circle; with application of the results to the theory of atomic structure. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 7(39):237–265, 1904.

[151] Stephen Tian, Suraj Nair, Frederik Ebert, Sudeep Dasari, Benjamin Eysenbach, Chelsea Finn, and Sergey Levine. Model-based visual planning with self-supervised functional distances. *arXiv preprint arXiv:2012.15373*, 2020.

[152] Yonglong Tian. Contrastive multiview coding. GitHub repository `https://github.com/HobbitLong/CMC/tree/58d06e9a82f7fea2e4af0a251726e9c6bf67c7c9`, 2019.

[153] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. *arXiv preprint arXiv:1906.05849*, 2019.

[154] Yonglong Tian, Chen Sun, Ben Poole, Dilip Krishnan, Cordelia Schmid, and Phillip Isola. What makes for good views for contrastive learning? *arXiv preprint arXiv:2005.10243*, 2020.

[155] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[156] Antonio Torralba, Rob Fergus, and William T Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970, 2008.

[157] William T Trotter. Partially ordered sets. *Handbook of combinatorics*, 1:433–480, 1995.

[158] Michael Tschannen, Josip Djolonga, Paul K Rubenstein, Sylvain Gelly, and Mario Lucic. On mutual information maximization for representation learning. *arXiv preprint arXiv:1907.13625*, 2019.

[159] Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6: 100022, 2020.

[160] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27 (11):1134–1142, 1984.

[161] Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pages 11–30. Springer, 2015.

[162] Ivan Vendrov, Ryan Kiros, Sanja Fidler, and Raquel Urtasun. Order-embeddings of images and language. *arXiv preprint arXiv:1511.06361*, 2015.

[163] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa,

Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[164] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020.

[165] Feng Wang, Xiang Xiang, Jian Cheng, and Alan Loddon Yuille. Normface: L2 hypersphere embedding for face verification. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 1041–1049, 2017.

[166] Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. Learning fine-grained image similarity with deep ranking. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1386–1393, 2014.

[167] Sida Wang and Christopher D Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th annual meeting of the association for computational linguistics: Short papers-volume 2*, pages 90–94. Association for Computational Linguistics, 2012.

[168] Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *International Conference on Machine Learning*, pages 9929–9939. PMLR, 2020.

[169] Kilian Q Weinberger, John Blitzer, and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. In *Advances in neural information processing systems*, pages 1473–1480, 2006.

[170] Mike Wu, Chengxu Zhuang, Daniel Yamins, and Noah Goodman. On the importance of views in unsupervised representation learning. 2020.

[171] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3733–3742, 2018.

[172] Eric P Xing, Andrew Y Ng, Michael I Jordan, and Stuart Russell. Distance metric learning with application to clustering with side-information. In *NIPS*, volume 15, page 12. Citeseer, 2002.

[173] Jiacheng Xu and Greg Durrett. Spherical latent spaces for stable variational autoencoders. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4503–4513, 2018.

[174] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021.

[175] Lin Yen-Chen, Pete Florence, Jonathan T. Barron, Alberto Rodriguez, Phillip Isola, and Tsung-Yi Lin. iNeRF: Inverting neural radiance fields for pose estimation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[176] Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*, 2020.

[177] Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *arXiv preprint arXiv:1506.06724*, 2015.