Understanding Vision-based Dynamics Models

by

Cynthia Liu

SB, Computer Science and Engineering, Massachusetts Institute of Technology, 2020

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

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2021

Author .....
Department of Electrical Engineering and Computer Science
May 20, 2021

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

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Understanding Vision-based Dynamics Models

by

Cynthia Liu

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2021, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Recent developments in vision-based dynamics models have helped researchers achieve state-of-the-art results in a number of fields. For instance, in model-based reinforcement learning, vision-based methods perform extremely well on a variety of games and control tasks while using orders of magnitudes less data than model-free methods. One example is GameGAN, which learns to simulate the dynamics of observed games solely from visual and action inputs. However, there is very little understanding of these models and how they work. To address this lack of understanding, we apply the Network Dissection framework to analyze vision-based dynamics prediction models. We inspect individual trained neurons in convolutional layers of these models and modify the output of neurons to understand their effect on the representation. We also theoretically extend the Network Dissection framework by generalizing it to fully connected layers instead of only convolutional layers. Overall, we provide insight into the node-level workings of dynamics models.

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

Acknowledgments

I would like to acknowledge Prof. Torralba and the entire Torralba lab for being incredibly approachable and insightful. I appreciate all the feedback everyone gave on the project and everyone's kindness.

I would like to acknowledge my direct supervisor, Yunzhu Li, for clear guidance during both my undergraduate work and my Master's thesis work, as well as your ability to be flexible and adapt to my needs in the current moment.

I also want to acknowledge the support systems at MIT for helping me through some issues. I had a difficult time first adapting to graduate life, but my support network was able to give all the information I needed and the motivation to complete this work.

Finally, I want to acknowledge my classmates and friends. Although we were physically distanced this entire year, technologies constantly kept us in touch. All of you kept me relaxed and happy.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the past decade, machine learning models have achieved groundbreaking results and have significantly shaped our day-to-day lives. Machine learning techniques have solved the protein folding problem [21], introduced intelligent helpers such as Siri, and generated completely new art and music, among many other applications.

Despite the ubiquity of machine learning, researchers' understanding of how these models work at a qualitative level lags behind. This poses a number of problems. For instance, a lack of understanding of machine learning algorithms hinders research efforts at addressing racial or gender bias found in many such algorithms [5]. Poor understanding of models also means that users cannot fully trust the decisions of the model, an issue that becomes especially pronounced in applications such as medicine.

Some work has been done to understand models. This work includes but is not limited to gaining a better understanding of model behavior near certain inputs [19], or producing "saliency maps" on an input that reflect where a network's output values are higher on said input [28] [20]. However, the scope of this work is usually limited to feedforward computer vision models, in limited classification or object detection problems. There is much less work on interpreting models used for other machine learning problems.

One such other problem is that of learning models that can predict future actions or dynamics. We will call such models *dynamics models*. The motivation for training such models is strong: neuroscience suggests that humans predict future dynamics

and interactions within a system prior to acting [17]. Some researchers even believe that prediction is crucial to intelligence in general [4]. Not only is dynamics prediction an interesting problem in and of itself, but it also has plentiful applications. Models with the ability to predict dynamics could be used to simulate environments without explicitly knowing how they work, better help humans by anticipating their actions, or generate massive amounts of realistic video by predicting many frames from just a few images, among others.

The dynamics models that have appeared in recent years achieved incredible results across a variety of tasks. For instance, in reinforcement learning, dynamics models learn to simulate environment physics in low dimensional space, and then the reinforcement learning agent can train directly on the simulation instead of the environment [9] [25] [10] [11]. These models have achieved state-of-the-art results on a number of benchmarks.

Despite the explosion of models, there is very little work in understanding how these dynamics models work and what they learn. In this work, we hope to start bridging this gap of understanding by interpreting dynamics models.

1.1 Our Contribution

Our main contribution is to provide insight into the inner workings of dynamics models. To get this insight, we use *Network Dissection* [1]. Network dissection is a technique that allows researchers to visualize the behavior of individual nodes in a model given the observation input. Furthermore, network dissection provides a set of tools with which to manipulate individual neurons in a model for more fine-grained analysis. These tools allow us to conduct a finegrained analysis of the model at the neuron level.

Using dissection, we start to understand where neurons in dynamics models are focusing within their environment. We observe that, when models learn features of their environment before being exposed to the downstream task of dynamics prediction, they seem to learn information that is not pertinent to environment dynamics.

We also observe that models do not necessarily need to focus or activate on individual limbs of actors in their environment to successfully predict dynamics. Finally, we use our results to pose new questions and extensions.

Chapter 2

Preliminaries

In order to understand network dissection on various dynamics models, we must first understand dynamics models as well as the network dissection framework. We cover these topics in this section.

2.1 Dynamics Prediction

Predicting the future, whether frames or actions, is a very important problem in artificial intelligence with many applications. As a result, there are many works on predicting dynamics. There are also a lot of works that incorporate dynamics prediction into a broader task.

There is an extensive body of work in dynamics prediction on video or image inputs. This prediction may take the form of frame prediction [15] [18] [13], or action anticipation [8] [22]. Frame prediction is the problem of computing possible future frames, given some number of past frames, in an image sequence. This computation may be of the frame directly, may be selecting the most appropriate frame from a number of choices, or other possibilities. By contrast, action anticipation does not compute future frames but instead tries to extract information from past frames about actions that will happen in later frames.

There are also many works that incorporate dynamics prediction into solutions to separate problems. As mentioned in the introduction, in reinforcement learning,

dynamics prediction is used to simulate environments or games to enable learning agents to train in simulation [9] [10] [11] [25]. Dynamics prediction is also present in robotics applications [7] [6], and is used to create more semantically meaningful low-dimensional representations of video than regular image feature extraction [15] [12] [16] [26].

Despite the many works involving dynamics prediction, there is very little work into analyzing such models and gaining a better understanding of why they work. Our work starts to remedy this gap, by applying network dissection to dynamics models.

2.2 Network Dissection

Network dissection, as applied to both convolutional classifiers and GANS, is a framework to rigorously analyze individual neurons in computer vision models [1]. Network dissection as an interpretation technique is especially powerful because, as a white box technique, it does not require training an auxiliary model as is used in methods such as LIME [19]. Furthermore, it allows for direct analysis of individual neurons, instead of being limited to checking the behavior of the overall network such as with CAM, GradCAM, or similar methods [28] [20].

The basis of network dissection is "hooking" a model to visualize and potentially modify the output activations of individual neurons in the model. Given a unit u in a convolutional layer, let $a_u(x, p)$ be its activation on input x at location p in the feature map. To visualize these activations, we find $\{p|a_u(x, p) > t_u\}$, where t_u is a unit-specific activation threshold usually at the top 1 or top 5% of activations, and then upsample the feature map back to the original image via reversing the convolutions. The images made as a result of this procedure are called *activation maps*.

Network dissection also provides the tools to modify $a_u(x, p)$ values when running an image through the model. The main modifications used by the creators of net dissection are *node zeroing* and *node activation*, in which $a_u(x, p)$ is forcibly set to 0 or t_u respectively. By intervening on individual neurons in this way, users can figure

out causal relationships within their models by seeing how outputs change with these forced activations.

In addition to providing visualizations of activations and modifying activations, network dissection lets users match regions of the visualization with semantically meaningful concepts such as "leg" or "grass." Because our work is less focused on the semantics of the environment images and more so on the behavior of the dynamics model itself, we do not make use of this capability for the scope of this project.

2.2.1 Fully Connected Layer Dissection

One limitation of producing activation maps via Network Dissection is that, as given, it is only applicable to convolutional layers. However, these are not the only types of layers present in dynamics prediction pipelines. We partially address this gap here by theoretically extending network dissection to fully connected linear layers.

The key to this is simply to observe that fully connected layers are a special case of convolutional layers. In a fully connected layer, the convolutional filter size is the same size as the input size. By extension, the output feature map size of a linear layer is 1 by 1 – the number of coordinates in the output is just the number of output channels. Finally, the stride of the linear layer is just 1.

As a result, we can directly map any fully connected layer to a corresponding convolutional layer, and apply the network dissection framework as is.

Chapter 3

Datasets

All visual observation data used with dynamics prediction models are either artificially *simulated* or from *real* videos. As a result, we use both types of environments. Our goal is to determine what dynamics models focus on for both types, and whether there are any observable differences.

3.1 Simulated Environments - MuJoCo and DeepMind Control

We first consider simulated environments, such as those used in reinforcement learning. For this project, we consider environments powered by the MuJoCo [24] physics engine. Some are OpenAI Gym Environments [3], while others are from the DeepMind Control (DMC) Suite [23]. All of these environments feature relatively simple agents on either a plain or checkered floor, against a plain background. As a result, compared to real video, these environments are visually very simple.

In order to generate observations in these environments, we must provide actions. We choose to randomly sample actions from each environment's action space and use those as the actions.

A segment of an agent trajectory with random actions from an OpenAI Gym environment, HalfCheetah-v2, is displayed in Figure 3-1, while a segment of a DMC envi-

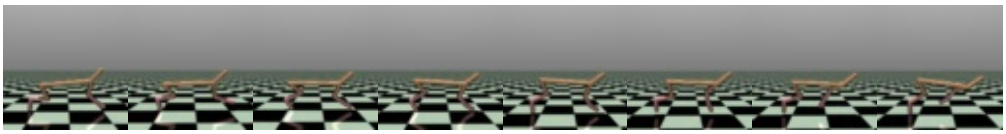


Figure 3-1: Example segment of HalfCheetah with random actions

Figure 3-2: Example segment of Finger with random actions

ronment, Finger, is displayed in Figure 3-2. We try a total of 10 agents: HalfCheetah, Swimmer, Finger, Humanoid, Hopper, Walker, Cup, Acrobot, CartPole, and Manipulator. The first two agents are from OpenAI MuJoCo, while the remaining eight are from DeepMind Control. However, a majority of experiments in this environment are conducted solely on the HalfCheetah environment.

3.2 Real Video Environment - Penn Action

While simulated environments are very relevant in computer vision and reinforcement learning, they are not the only type of environment used with dynamics prediction. Many works use real video datasets instead. As a result, we also consider a real video dataset for this work.

We chose the Penn Action dataset [27]. This dataset features videos of people performing 15 different, easily identifiable actions. The 15 different actions are presented in Figure 3-3. Three example episodes in this dataset are presented in Figure 3-4 with no preprocessing.

This dataset contains a total of 2326 videos, which are split into 1258 training and 1068 validation. These videos have highly varying length–from 18 frames to 663 frames. As a result, to train on this dataset, we randomly sample short clips from each video between training epochs. In addition, we address the fact that the videos have varying aspect ratios by resizing and cropping each frame to 240x320 pixels.

Figure 3-3: The 15 action types in the Penn Action dataset

Figure 3-4: Example segments of trajectories in the Penn Action dataset

Chapter 4

Models

Recall that in Section 2, we covered a number of unique dynamics prediction models. Because we cannot reasonably analyze all of them, we must choose or design one or two models that could give insight into the broader class of models.

4.1 Commonalities in Dynamics Prediction Models

Across all dynamics prediction models, we see that they almost always have a *vision component* and a *dynamics component*. The vision component extracts features from high-dimensional image input and creates a lower dimensional latent representation. The dynamics component is a recurrent network that then trains on these latent representations, taking some number of historical representations and outputting a predicted future representation. An example of vision and dynamics components in a real paper, World Models, is displayed in Figure 4-1 [9].

4.2 Our Model

With this in mind, we define a vision and dynamics component on which to apply network dissection.

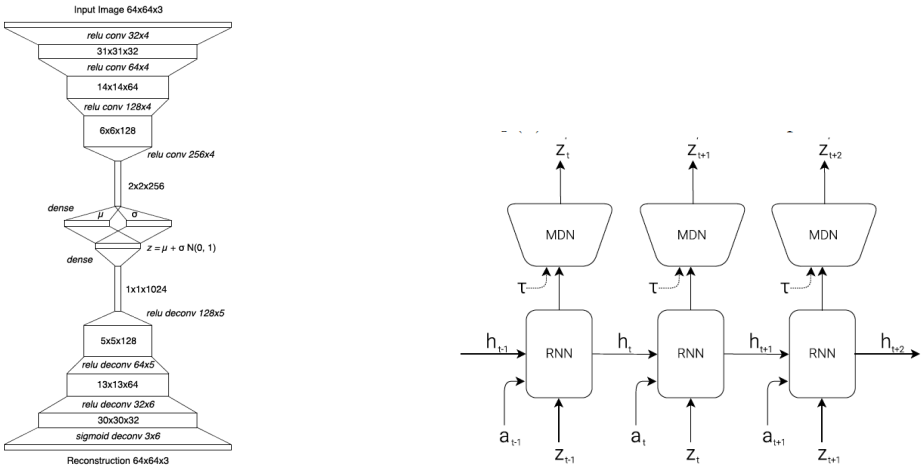


Figure 4-1: Example vision and dynamics components for World Models

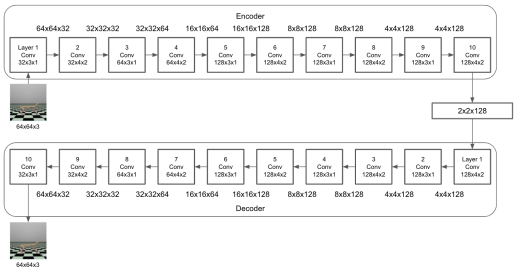
4.2.1 Deep Autoencoder

In order to work with varying resolutions and types of visual data, we decide to train a deep convolutional autoencoder as the vision component. Autoencoders $A(x) \rightarrow x'$ are networks designed to learn low-dimensional encodings of data x in an unsupervised manner [14] by compressing x to a low-dimensional embedding h, and learning to reconstruct x into x' using h. Our autoencoder consists of nine or ten layers in each of the encoder $E(x) \rightarrow h$ and decoder $D(h) \rightarrow x'$ components, with no residual or attention blocks. We choose this structure because it is relatively easy to interpret and adapt to varying datasets.

The autoencoder architecture we use for simulated and real video data are displayed in Figure 4-2. We call the simulated autoencoder $E_s(x)$ and $D_s(x)$ *small* while we call the real video autoencoder $E_l(x)$ and $D_l(x)$ *large*. If we refer to an encoder or decoder in general absent a dataset specification, we will use $E(x)$ and $D(x)$. The main difference between the autoencoders is different latent representation dimensions, kernel size and stride of layers.

The low-dimensional representations used for dynamics prediction are the outputs of the last layer of the encoder. For simulated data, we flatten the encoding to a single length-512 vector. For real video data, we keep the shape of the encoder output.

The autoencoders are trained using l_2 reconstruction loss:



(a) Autoencoder used for 64x64 simulated data

	Encoder - kernel, output ch, stride, pad || output dim
1	8x8, 32, 2, 3 || o: 120x160
2	3x3, 64, 1, 1 || o: 120x160
3	4x4, 64, 2, 1 || o: 60x80
4	3x3, 128, 1, 1|| o: 60x80
5	4x4, 128, 1, 1 || o: 30x40
6	3x3, 128, 2, 1 || o: 30x40
7	4x4, 128, 1, 1 || o: 15x20
8	3x3, 128, 2, 1 || o: 15x20
9	3x3, 128, 1, 1 || o: 8x10

	Decoder - kernel, output ch, stride, pad || output dim
1	Transpose 3x3, 128, 2, 1 || o: 15x19
2	3x3, 128, 1, 1 || o: 15x19
3	Transpose 4x4, 128, 2, (1, 0) || o: 30x40
4	3x3, 128, 1, 1 || o: 30x40
5	Transpose 4x4, 128, 2, 1 || o: 60x80
6	3x3, 64, 1, 1 || o: 60x80
7	Transpose 4x4, 64, 2, 1 || o: 120x160
8	3x3, 32 1, 1 || o: 120x160
9	Transpose 8x8, 32, 2, 3 || o: 240x320

(b) Autoencoder used for 240x320 real video data

Figure 4-2: Autoencoder architectures used for this work

$$\mathcal{L} = \|x - x'\|_2^2 = \|x - D(E(x))\|_2^2. \tag{4.1}$$

4.2.2 Dynamics RNN

The dynamics component in pre-existing works is more varied. However, all models use either a recurrent network or transformer-like architecture. For initial experiments, we pick the simplest option and use an RNN without memory components $R(f(h_{t-k+1}, \ldots, h_t)) \rightarrow \Delta h_t$.

For simulated data, the recurrent network is a multilayer perceptron. For real data, the network is a small convolutional neural network. In both cases, the network takes some combination of k historical low-dimensional embeddings $f(h_{t-k+1}, \ldots, h_t)$, and predicts a residual on the previous encoding Δh_t. The true predicted encoding h_{t+1} is the residual added to the previous encoding: $h'_{t+1} = h_t + \Delta h_t$.

The architecture of the convolutional RNN is displayed in Figure 4-3. The MLP-based RNN works identically, with the only difference being that the CNN in the figure is replaced with an MLP.

Similar to the autoencoder, the RNN is trained using l_2 reconstruction loss. If the vision component is fixed (the parameters are not changing), we apply reconstruction loss directly in the embedding space:

$$\mathcal{L} = \|h_{t+1} - h'_{t+1}\|_2^2. \tag{4.2}$$

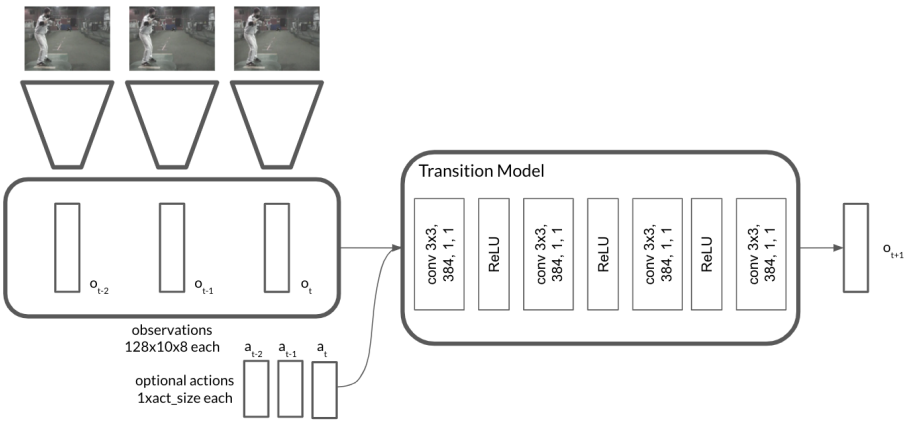


Figure 4-3: Architecture of the simple convolutional RNN

If the vision component is not fixed, such as during the finetuning process of training (see Section 5), then the RNN is trained with l_2 reconstruction loss in the vision space:

$$\mathcal{L} = \|x_{t+1} - D(h'_{t+1})\|_2^2. \tag{4.3}$$

Chapter 5

Results

In this section, we outline our experiments and describe their empirical results.

5.1 Training Procedure

In order to train our models, we first train the vision component autoencoder alone by optimizing equation 4.1 (Step 1). Afterwards, we freeze the parameters in the autoencoder and train the dynamics prediction component alone by optimizing equation 4.2 (Step 2). Finally, we may finetune the autoencoder and dynamics model together with 4.3 (Step 3). This procedure mirrors that of many existing works: usually, existing works first learn a new representation or use a pretrained feature extractor, then they learn to predict future frames, either with the low-dimensional representation or separately.

The initial learning rate for autoencoders and RNNs for all datasets is 6e-4, and this learning rate is adjusted using a stepwise scheduler. For all models, we using Adam optimization. When finetuning, we use an initial learning rate of 1e-2 times the initial dynamics model training rate.

All models are trained on Nvidia Titan V or Titan X in the CSAIL Vision Cluster. Unless otherwise stated, all autoencoders are trained on 60 epochs. Dynamics models are trained on 20 epochs then, if applicable, finetuned for another 20 epochs. For MuJoCo and DeepMind Control environments, dynamics models are trained to

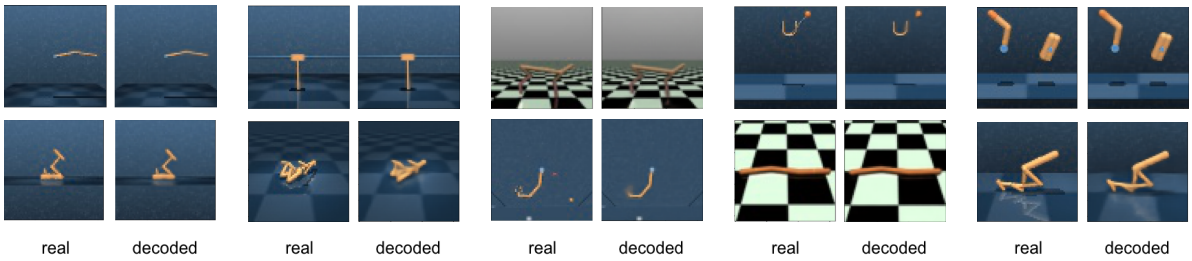


Figure 5-1: MuJoCo and DeepMind Control decoded images

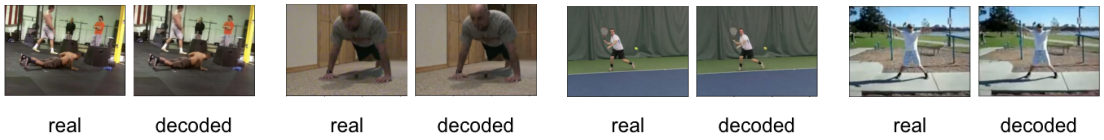


Figure 5-2: Penn Action decoded images

predict 3 steps into the future, while for Penn Action, dynamics models are trained to predict 5 steps into the future. At evaluation time, models will predict 30 frames into the future for simulated environments and 10 frames for Penn Action.

5.2 Autoencoder and Dynamics Training

We first present our results for steps 1 and 2 of the training process. Without decent models at this step, network dissection and further interpretation of the model would not provide any insight into functional models.

For simulated environments, we train individual models for just HalfCheetah and Swimmer, as well as an additional model on all 10 environments listed in Section 3 simultaneously. Example autoencoder reconstructions for the model trained on all 10 simulated environments, for all 10 simulated environments, can be found in Figure 5-1, while example autoencoder decodings for Penn Action are in Figure 5-2. We see that the reconstructed images are nearly identical to the originals for both simulated and real environments. We do not display the autoencoding results for the models trained on HalfCheetah and Swimmer individually, because they are visibly identical to those results from the autoencoder trained on all 10 simulated environments.

We then train the dynamics models for both datasets. Example predicted trajecto-

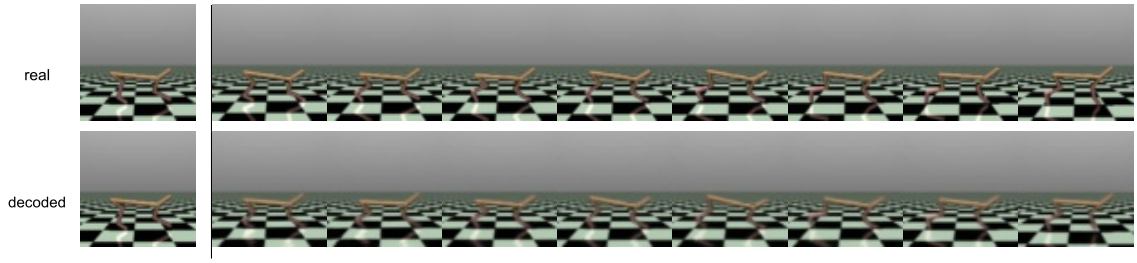


Figure 5-3: HalfCheetah dynamics prediction example

Figure 5-4: Penn Action dynamics prediction example

ries for the HalfCheetah subtask within MuJoCo are in Figure 5-3. We see that, while the trajectory may diverge, the overall behavior is realistic and generally correct.

Dynamics prediction on the Penn Action dataset was much less successful than the MuJoCo dataset, and the model's solution is not too different than the degenerate solution where images stay the same regardless of input. An example representative video trajectory alongside its predicted counterpart is visible in Figure 5-4.

5.3 Finetuning

Given the results of steps 1 and 2 from the previous section, we can now conduct step 3: finetuning both the vision and dynamics components of the model jointly. We focus on the HalfCheetah task for Step 3.

We note that finetuning HalfCheetah does not seem to improve performance. A graph of the $l2$ loss between the true and predicted image at different steps of the predicted trajectory is provided in Figure 5-5. Given the high level of performance prior to finetuning, this suggests that the found solution is already near or at optimal, and therefore finetuning would not improve it.

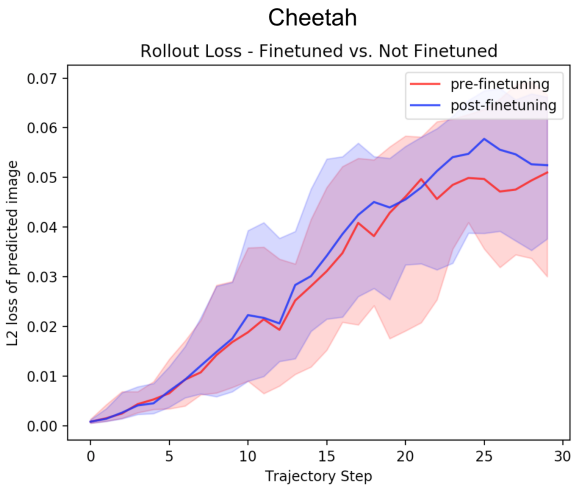


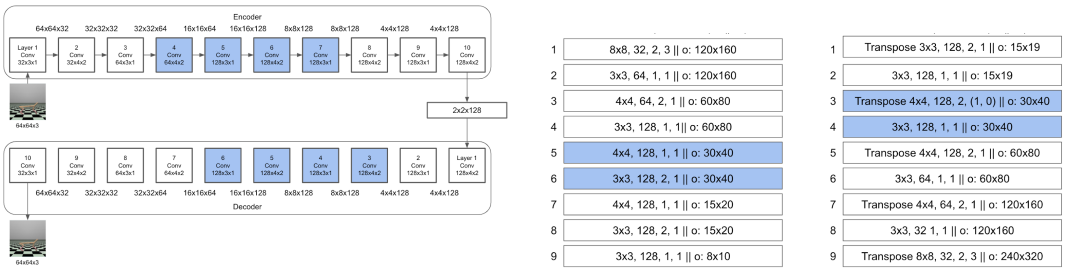
Figure 5-5: Reconstruction loss between true image and predicted image at different steps of trajectory prediction

5.4 Dissections - Activation Maps

With working autoencoders from Steps 1 or 3, we can dissect them to produce activation maps (defined in section 2). Recall that these maps extract the visual information these models find salient. For simulated environments, we can compare activation maps before and after finetuning as well. In this section, we present activation map dissections for the vision component autoencoder for both simulated and real datasets. For simulated data, we compare finetuned versus pre-finetuned models.

In traditional dissections, researchers focus on convolutional layers near the network output. This is because later layers in a convolutional network usually detect higher-level semantic concepts, such as doors and trees, than the earlier layers, which would capture concepts such as lines and textures [1] [2]. Part of the reason for this is that feature maps in later layers are much smaller than in earlier layers, so any single element in a later feature map corresponds to very many pixels in the original image. The smaller a feature map, the larger the region in the original image corresponding to a single entry in the feature map.

In an autoencoder, the feature maps associated with the low-dimensional embedding are very small: 2x2 for simulated environments or 8x10 in real video. As a result, in order to get sufficiently fine-grained features, we choose to dissect the "mid-



(a) Autoencoder used for 64x64 simulated data

(b) Autoencoder used for 240x320 real video data

Figure 5-6: Autoencoder architectures, with their dissection layers highlighted in blue

dle" layers of the autoencoder instead of the layers closest to the low-dimensional encoding. These "middle" layers are layers 4 through 7 of the encoder and layers 3 through 6 of the decoder for the small autoencoder, while they are layers 5 and 6 of the encoder and 3 and 4 of the decoder for the large autoencoder. These middle layers are highlighted in Figure 5-6.

In order to best understand the focus of a neuron, we visualize the images that *maximize* the output activations of each neuron. This is the same strategy as used by the original authors of network dissection [1]. By detecting commonalities between images that maximize a single node's activation, we can derive conclusions about what parts of images are most "important" to that node. In order to get a higher diversity of images, we manually filter the activation maximizing images such that no two reported images are within 5 frames of each other in the same trajectory.

5.4.1 MuJoCo and DeepMind Control

For simulated environments, a nontrivial number of neurons in the encoder do not "fire," or have positive output. Table 5.1 displays the number of neurons with maximum activation 0 over all 10 MuJoCo and DMC environments before finetuning across the middle layers displayed in Figure 5-6. The large number of non-activating nodes suggest that our model is heavily overparameterized for the environment.

Among the neurons that do fire before finetuning, a majority of the neurons appear to focus on the floor or background. This can be seen by their activation

Encoder 4	51/64	Decoder 3	2/128
Encoder 5	111/128	Decoder 4	6/128
Encoder 6	108/128	Decoder 5	1/128
Encoder 7	93/128	Decoder 6	7/128

Table 5.1: Number of neurons in each layer that do not activate across all environments

maps highlighting parts of the floor or background. To highlight this, we display the top 3 HalfCheetah activation maps of the first 5 nodes of each layer with nonzero activations in Figure 5-7.

Surprisingly, for the HalfCheetah environment, this is true before and after finetuning. We see this in Figure 5-8, where we compare the top 3 activation maps before and after finetuning in the first 10 nodes with nonzero activations from decoder layer 4. In fact, we notice that with the exception of the first node on the third row, the activation maps before and after finetuning look nearly or exactly identical. This fact, alongside the graph in Figure 5-5, suggests that dynamics prediction prior to finetuning may have already reached a near-optimal or optimal solution. As a result, finetuning would not significantly alter weights because alteration would not significantly improve performance.

Another observation is that, in the autoencoder trained on all 10 simulated environments, nodes tend to activate on an environment-specific "pose" across most or all environments. The top 3 activation maximizing images for 5 selected nodes in all environments across various layers are displayed in Figure 5-9. Some of these "poses," such as the ones displayed for encoder layer 5 node 124, seem to share common visual traits across all environments. For encoder layer 5 node 124, the common visual trait in most environments is that the agent body is relatively horizontal. For those bodies that are not horizontal, the unit seems to activate under horizontal components of those agents, such as in HalfCheetah.

One implication of this observation is that the model does not generally treat the 10 environments separately. If the model treated the environments separately, we would observe many nodes that only activate on one or few environments, whereas

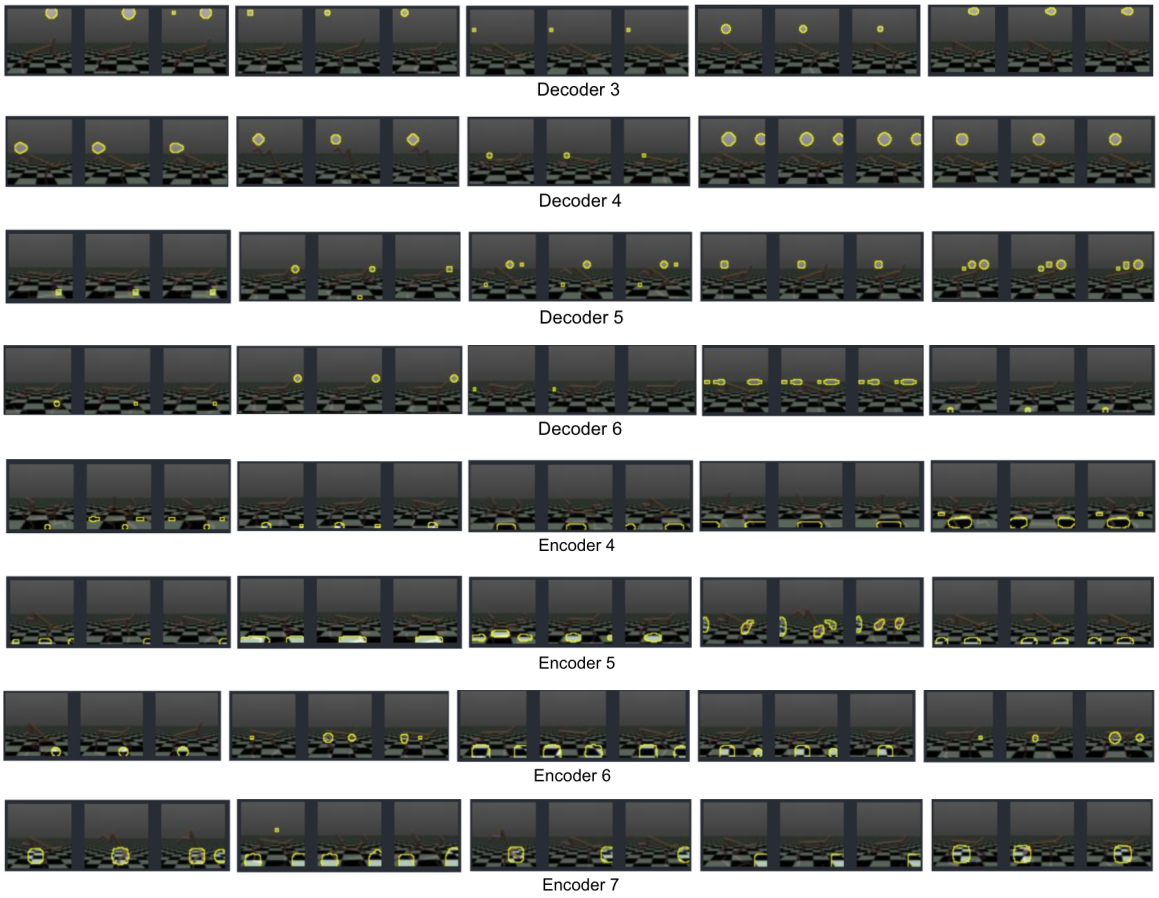


Figure 5-7: Activation maps for decoder layer 4 on HalfCheetah before and after finetuning

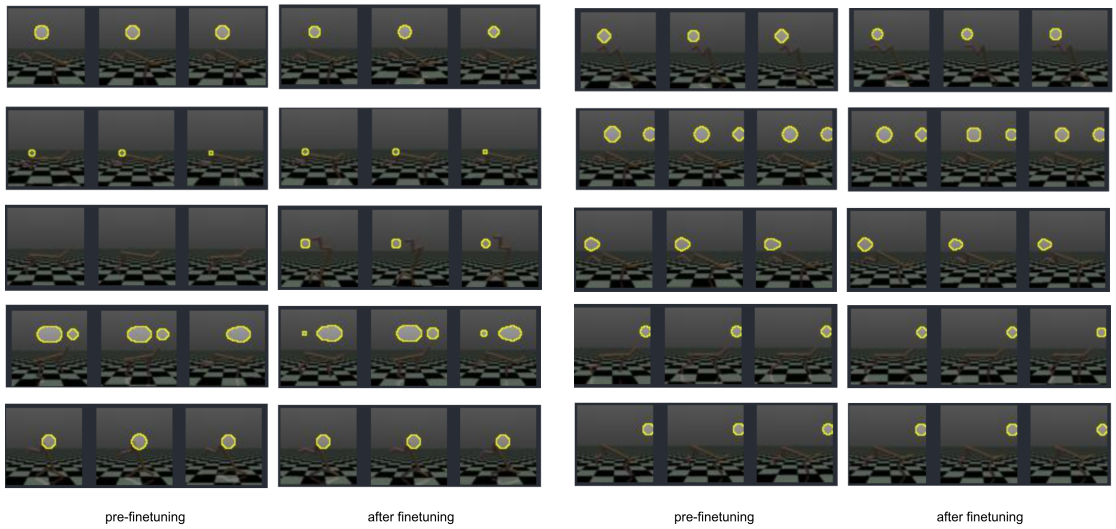


Figure 5-8: Activation maps for decoder layer 4 on HalfCheetah before and after finetuning

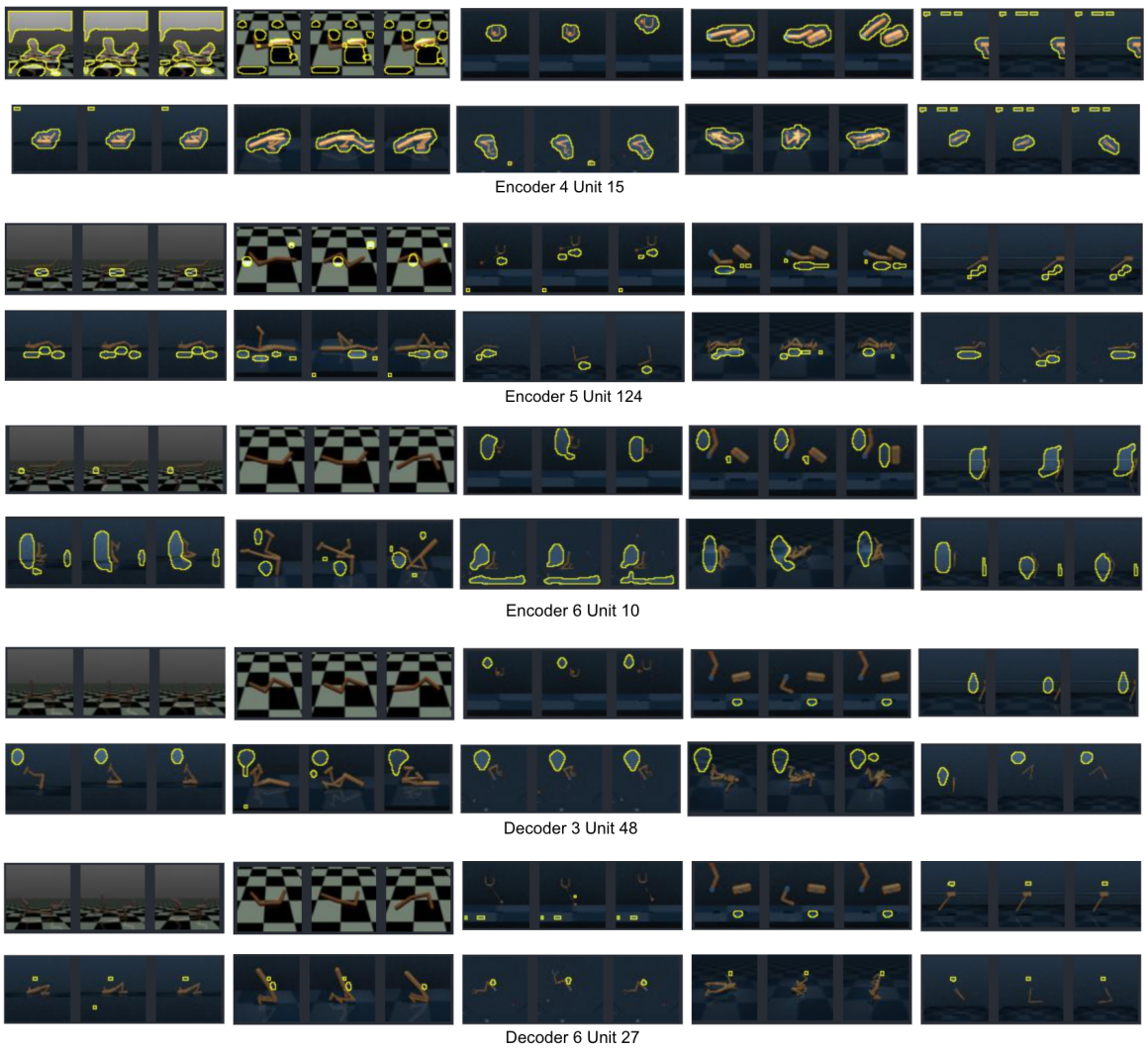


Figure 5-9: Pose matching across different environments on an autoencoder trained on all 10 simulated environments

this is not observed.

Prior to finetuning, this makes sense because the autoencoder currently is only trained on the end task of replicating the original image as accurately as possible. In other words, it does not need to learn anything about the actual dynamics within an environment, which differs greatly across the 10 subtasks. Instead, the model focuses on purely visual features of agent position.

While we attempted to train dynamics simultaneously for all 10 environments, our implementation was not sufficiently efficient to produce tangible results with the given compute resources.

Figure 5-10: Penn Action activation maps. E5 U14 stands for encoder layer 5, unit 14. The other units are named accordingly

5.4.2 Penn Action

Similar to the 10 simulated environment case, nodes in the Penn Action autoencoder prior to finetuning focus on visual details shared across many types of actions. As we can see in Figure 5-10, nodes often maximize their activations on images from multiple action types. They may also maximize their activations on details, such as the words "One Result" in some of the images, that are independent of the action. This makes sense because, once again, the autoencoder has only been trained with the end task of reproducing the original image, instead of predicting future dynamics.

We hope to observe the difference between activation maps before and after finetuning in future work.

5.5 Zeroing Out Nodes

In this section, we go a step further and modify neurons in the Vision component of our models. This allows us to determine casual relationships within the network. We focus on the node zeroing strategy listed in Section 2.2. We then describe the results of this modification on HalfCheetah-v2 before and after finetuning.

Once again, we work with the middle layers 5-6 of the vision component. For each node, we choose to zero out its activation on the images on which it originally activated the most. This way, we can observe the most pronounced differences in

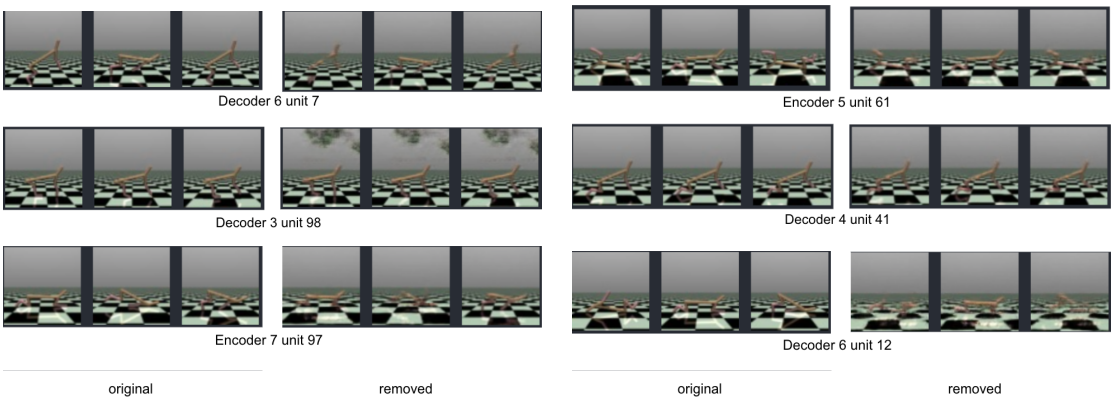


Figure 5-11: Environment observations, and decoded versions of those observations after the specified node is removed in the pre-finetuned model

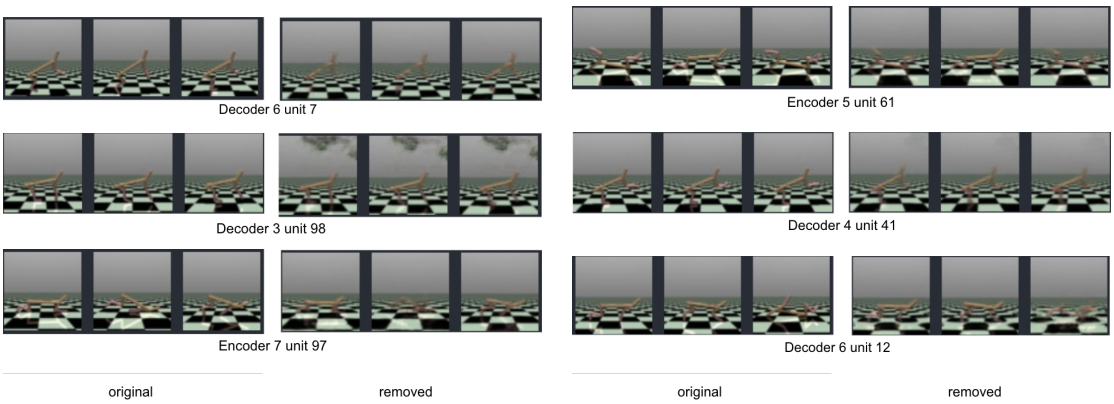


Figure 5-12: Environment observations, and decoded versions of those observations after the specified node is removed in the post-finetuned model

decoded images before and after the node was removed.

In Figure 5-11, we observe selected original images, and those same images decoded after a specified node in the vision component is removed prior to finetuning. In Figure 5-12, we replicate the same figure but on the finetuned model. We pick the first 10 nodes for each of the middle layers in both cases.

Overall, we see that the model largely treats the cheetah body as a single unit. This is because, if node removal blurs out part of the agent's body, it almost certainly blurs out the entire agent. This behavior exists for both the pre-finetuned and post-finetuned models.

The fact that most removed units affecting the agent's body influence the entire

body is very interesting. Firstly, this contradicts removed neuron behavior in GANs [2] [1], in which zeroing the activation of a "door" neuron usually removes just the door instead of an entire building or region around the door. Secondly, we see that whole-body-removal behavior appears even after exposing the vision component to the end task of dynamics prediction via finetuning. This means that, even to predict dynamics, the model still treats the body as one unit instead of separating the joints.

Beyond this, we note that, despite the fact that a majority of nodes activate on parts of the background based on their activation maps, fewer nodes cause part of the background to be blacked out when removed. This is likely due to redundancy within the network, in which multiple nodes tend to cover the same regions of the background. That way, removing a singular node affecting the background would not actually remove any part of the background, and the model can interpolate the background at the affected area.

We also note that, as seen in the figures, not all HalfCheetah positions are treated identically in node removal. Even nodes that activate on similar cheetah body poses may not behave the same way. In particular, decoder layer 6 unit 7 and decoder layer 4 unit 41 activate on mostly similar HalfCheetah poses, but removing the activation at decoder layer 6 unit 7 affects the resulting image more.

Just like for the activation maps, node removal results for the model before and after finetuning are strikingly similar. Once again, this bolsters the notion that the model did not significantly change despite finetuning.

Chapter 6

Discussion and Conclusion

In this work, we gain a better understanding of the inner workings of dynamics prediction model. We determine components shared across all dynamics models, design simple but representative and relatively interpretable versions of said components, and analyze these components with network dissection. We see that, for simulated 2D environments such as those in reinforcement learning, the model can still achieve success in dynamics prediction even if the model's nodes primarily focus on visual details of the environment, such as the background. For HalfCheetah, we demonstrate that this is true even after exposing the vision component to the end task of dynamics prediction via finetuning.

We see that in general, prior to introducing the end task of dynamics prediction, no autoencoder seems to focus on individual limbs or joints on any agent or human. In fact, node removal experiments on HalfCheetah suggest that autoencoders might learn about 2D actor bodies as a single unit. This is interesting because joints are crucial to the human understanding of dynamics. This indicates that either joint information is incorporated during finetuning and exposing the vision component to the dynamics prediction task, or, as in the case of Cheetah, that joint information is not necessary for adequate dynamics prediction. We hope that in future work, researchers can reach a more definitive answer about the role of joints in a dynamics model's representation of its environment.

Based on these results, we believe that dynamics model behavior may be very

unintuitive. However, we do note that this is based solely on our observations using network dissection. One very interesting question would be to compare the results of network dissection with other network interpretation methods. We hope that this work motivates more researchers to try different techniques to understand dynamics models. We also encourage practitioners to be more aware of the limits of their understanding of dynamics models, and to not assume that a model works in a way that is similar to humans.

6.1 Future Work

We believe that this work is merely a start to potentially much more work in understanding dynamics models. Here, we list a few extensions considered for the project.

6.1.1 Implementation of Section 2.2.1

To bolster our understanding of dynamics prediction in simple simulated environments, we can formally implement the technique presented in Section 2.2.1 that extends the computation of activation maps to linear layers. This way, we can determine the focus of the network while it is predicting future frames, instead of determining the focus of the network in creating the low dimensional representations used for said prediction.

6.1.2 Predicting Optical Flow

Due to the difficulty of predicting dynamics on real, higher-resolution video, as well as the fact that optical flow (the movement of pixels) is not explicitly visible in image sequences, we considered using an additional model R' for real video. This model, influenced by deep visual foresight [6], explicitly predicts the *movement* of existing pixels into the next frame.

Instead of predicting a residual to the next embedding of dimension $C \times H \times W$, we predict an output of dimension $2 \times H \times W$, where the values (m_i, m_j) at location

(i, j) in both filter maps correspond to the movement of the values at (i, j) in the input embedding. This is an example of *bilinear sampling*. Due to time constraints, we ultimately did not implement this model. However, implementing such a model or other models that predict future frames of real video input is of interest.

Bibliography

[1] David Bau, Jun-Yan Zhu, Hendrik Strobelt, Agata Lapedriza, Bolei Zhou, and Antonio Torralba. Understanding the role of individual units in a deep neural network. *PNAS*, September 2020.

[2] David Bau, Jun-Yan Zhu, Hendrik Strobelt, Agata Lapedriza, Bolei Zhou, and Antonio Torralba. Understanding the role of individual units in a deep neural network. *Proceedings of the National Academy of Sciences*, 2020.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[4] Andreja Bubic, D. Yves Von Cramon, and Ricarda Schubotz. Prediction, cognition and the brain. *Frontiers in Human Neuroscience*, 4:25, 2010.

[5] Aylin Caliskan, Joanna J. Bryson, and Arvind Narayanan. Semantics derived automatically from language corpora contain human-like biases. *Science*, 356(6334):183–186, 2017.

[6] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex X. Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *CoRR*, abs/1812.00568, 2018.

[7] Chelsea Finn, Ian J. Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. *CoRR*, abs/1605.07157, 2016.

[8] Harshala Gammulle, Simon Denman, Sridha Sridharan, and Clinton Fookes. Predicting the future: A jointly learnt model for action anticipation. *CoRR*, abs/1912.07148, 2019.

[9] David Ha and Jürgen Schmidhuber. World models. *CoRR*, abs/1803.10122, 2018.

[10] Danijar Hafner, Timothy P. Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *CoRR*, abs/1912.01603, 2019.

[11] Danijar Hafner, Timothy P. Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *CoRR*, abs/2010.02193, 2020.

[12] Tengda Han, Weidi Xie, and Andrew Zisserman. Video representation learning by dense predictive coding. *CoRR*, abs/1909.04656, 2019.

[13] Seung Wook Kim, Yuhao Zhou, Jonah Philion, Antonio Torralba, and Sanja Fidler. Learning to simulate dynamic environments with gamegan, 2020.

[14] M. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *Aiche Journal*, 37:233–243, 1991.

[15] William Lotter, Gabriel Kreiman, and David D. Cox. Deep predictive coding networks for video prediction and unsupervised learning. *CoRR*, abs/1605.08104, 2016.

[16] Antoine Miech, Jean-Baptiste Alayrac, Lucas Smaira, Ivan Laptev, Josef Sivic, and Andrew Zisserman. End-to-end learning of visual representations from uncurated instructional videos. *CoRR*, abs/1912.06430, 2019.

[17] Yael Niv. Reinforcement learning in the brain. *Journal of Mathematical Psychology*, 53(3):139–154, 2009.

[18] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L. Lewis, and Satinder P. Singh. Action-conditional video prediction using deep networks in atari games. *CoRR*, abs/1507.08750, 2015.

[19] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. *CoRR*, abs/1602.04938, 2016.

[20] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016.

[21] Andrew W. Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander W. R. Nelson, Alex Bridgland, Hugo Penedones, Stig Petersen, Karen Simonyan, Steve Crossan, Pushmeet Kohli, David T. Jones, David Silver, Koray Kavukcuoglu, and Demis Hassabis. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792), 2020.

[22] Dídac Surís, Ruoshi Liu, and Carl Vondrick. Learning the predictability of the future. *CoRR*, abs/2101.01600, 2021.

[23] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy P. Lillicrap, and Martin A. Riedmiller. Deepmind control suite. *CoRR*, abs/1801.00690, 2018.

[24] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. pages 5026–5033, 10 2012.

[25] Tobias Weyand, Ilya Kostrikov, and James Philbin. Planet - photo geolocation with convolutional neural networks. In *European Conference on Computer Vision (ECCV)*, 2016.

[26] Dejing Xu, Jun Xiao, Zhou Zhao, Jian Shao, Di Xie, and Yueting Zhuang. Self-supervised spatiotemporal learning via video clip order prediction. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10326–10335, 2019.

[27] Weiyu Zhang, Menglong Zhu, and Konstantinos G. Derpanis. From actemes to action: A strongly-supervised representation for detailed action understanding. In *2013 IEEE International Conference on Computer Vision*, pages 2248–2255, 2013.

[28] Bolei Zhou, Aditya Khosla, Àgata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. *CoRR*, abs/1512.04150, 2015.