Learning and Enforcing Diversity with Determinantal Point Processes

by

Zelda Elaine Mariet

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

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2016

Author .....
Department of Electrical Engineering and Computer Science
January 29, 2016

Certified by .....
Suvrit Sra
Principal Research Scientist
Thesis Supervisor

Certified by .....
Leslie Pack Kaelbling
Professor
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Chairman, Department Committee on Graduate Theses

Learning and Enforcing Diversity with Determinantal Point Processes

by

Zelda Elaine Mariet

Submitted to the Department of Electrical Engineering and Computer Science
on January 29, 2016, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

As machine-learning techniques continue to require more data and become increasingly memory-heavy, being able to choose a subset of relevant, high-quality and diverse elements among large amounts of redundant or noisy data and parameters has become an important concern.

Here, we approach this problem using Determinantal Point Processes (DPPs), probabilistic models that provide an intuitive and powerful way of balancing quality and diversity in sets of items. We introduce a novel, fixed-point algorithm for estimating the maximum likelihood parameters of a DPP, provide proof of convergence and discuss generalizations of this technique.

We then apply DPPs to the difficult problem of detecting and eliminating redundancy in fully-connected layers of neural networks. By placing a DPP over a layer, we are able to sample a subset of neurons that perform non-overlapping computations and merge all other neurons of the layer into the previous diverse subset. This allows us to significantly reduce the size of the neural network while simultaneously maintaining a good performance.

Thesis Supervisor: Suvrit Sra
Title: Principal Research Scientist

Thesis Supervisor: Leslie Pack Kaelbling
Title: Professor

Acknowledgments

Before anything else, I would like to thank my advisors: Suvrit Sra, for introducing me to DPPs, guiding me throughout this experience and asking many thought-provoking questions that helped me better understand my research; and Leslie Pack Kaelbling, for sharing her invaluable insight on all things machine-learning, as well as for her advice and support. I have learned a lot as their student, and their advice and help were invaluable throughout my time at MIT.

I would also like to thank my labmates Zi Wang and Beomjoon Kim, as well as all my graduate and undergraduate colleagues in LIS and CSAIL, for their support and for making my graduate experience such an enjoyable one.

Finally, I would like to thank my family for encouraging and supporting me unconditionally in all my scientific endeavors throughout the years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Selecting a representative sample from large amounts of data is a recurring concern in many machine-learning domains: in document summarization, a summary is created from the sentences in the document that are representative of the various topics covered by the text; for search engines, the retrieved results should cover a broad range of the initial query's possible meanings.

In order to approach the problem of selecting a diverse subset of given items, we must model negative interactions: when selecting one item, the probability of simultaneously choosing other, similar items must decrease.

1.1 Determinantal Point Processes

Determinantal Point Processes (DPPs) are powerful probabilistic models over subsets of a ground set that allow to model negative correlations; they provide polynomial time algorithms for most usual tasks such as sampling, marginalizing and conditioning. DPPs arose in statistical mechanics as "fermion processes" [37] to model the repulsive interactions of fermion systems in thermal equilibrium. Recently, they have witnessed substantial interest in a variety of machine learning applications [26, 29].

DPPs provide the ability to model the notion of diversity while respecting quality, a concern that underlies the broader task of subset selection where balancing quality with diversity is a well-known issue—see e.g., document summarization [35], [12],

object retrieval [3], recommender systems [47], and sensor placement [24]. They have also been recently applied to modeling inter-neuron inhibitions in neural spiking behavior in the rat hippocampus [42].

DPPs are also interesting in their own right: they have various combinatorial, probabilistic, and analytic properties, and involve a fascinating set of open problems [36, 21, 26]. Within machine learning, DPPs have found good use—see for instance [14]; [28]; [27]; [3]; [4]; [2]; [13]. We provide below some basic definitions and theorems in DPP theory. For a more in-depth description, as well as additional references and material, we refer the reader to the extensive work by Kulesza and Taskar [29].

1.1.1 Formal definition

Without loss of generality, we assume that the ground set of N items is $\{1, 2, \ldots, N\}$, which we denote by $\mathcal{Y}$. A (discrete) DPP on $\mathcal{Y}$ is a probability measure $\mathcal{P}$ on $2^{\mathcal{Y}}$ (the set of all subsets of $\mathcal{Y}$) such that for any subset $Y \subseteq \mathcal{Y}$ drawn randomly from $\mathcal{P}$ and any subset $A \subseteq \mathcal{Y}$,

$$\mathcal{P}(A \subseteq Y) = \det(K_A).$$

where K is a positive semi-definite matrix in $\mathbb{R}^{N \times N}$ with all of its eigenvalues bounded by 1. K is called the marginal kernel, and K_A indicates the principal submatrix of K indexed by the elements in A, with the convention that $\det(K_\emptyset) = 1$.

1.1.2 L-ensembles

In this thesis, we will slightly restrict our analysis of DPPs by considering the class of L-ensembles [7], which define the probability of a subset $Y \subseteq \mathcal{Y}$ in the following way:

$$\mathcal{P}(Y) \propto \det(L_Y), \tag{1.1.1}$$

where L is also a positive semi-definite matrix in $\mathbb{R}^{N \times N}$.

The normalization constant for $\mathcal{P}$ follows upon observing [29, Theorem 2.1] that

$$\sum_{Y \subseteq \mathcal{Y}} \det(L_Y) = \det(L + I).$$

Thus,

$$\mathcal{P}(Y) = \frac{\det(L_Y)}{\det(L + I)}, \qquad Y \subseteq \mathcal{Y}.$$

One can easily show [29, Theorem 2.2] that an L-ensemble is also a DPP[1], and that the relationship between L and the marginal kernel K is given by the following equation:

$$K = L(L + I)^{-1}. \tag{1.1.2}$$

This also implies that K and L have the same eigenvectors and differ only in their eigenvalues. It can also be shown [26] that $\mathcal{P}(Y) = |\det(K - I_{Y^c})|$, where I_{Y^c} is a partial $N \times N$ identity matrix with diagonal entries in Y zeroed out.

Intuitively, the diagonal entry L_{ii} of the kernel matrix L captures some notion of the importance of item i, whereas an off-diagonal entry $L_{ij} = L_{ji}$ measures the similarity between items i and j. This provides further motivation for seeking DPPs with non-diagonal kernels when there is implicit interaction between the observed items.

1.1.3 Quality and diversity

A more intuitive understanding of how a DPP balances the quality and diversity of the subsets of $\mathcal{Y}$ is provided by the decomposition by Kulesza and Taskar [29, §3.1]: as any positive semi-definite matrix, the DPP kernel L can be decomposed as a Gramian matrix: $L = B^\top B$, where each column of B represents one of the N items of the ground set. This, in turn, provides a decomposition of L into quality terms

[1] The converse is not necessarily true: if an eigenvalue of the marginal kernel K is equal to 1, the DPP is not an L-ensemble.

($q_i \in \mathbb{R}^+$) and normalized diversity features ($\phi_i \in \mathbb{R}^p$, $\|\phi_i\| = 1$):

$$L_{ij} = q_i \phi_i^\top \phi_j q_j.$$

The Gram matrix S for vectors ϕ_i ($S_{ij} = \phi_i^\top \phi_j$) is called the diversity model; q is called the quality model.

This allows us to rewrite 1.1.1 as

$$\mathcal{P}(Y) \propto \left(\prod_{i \in Y} q_i \right) \det(S_Y),$$

providing a decomposition of the probability of a subset Y as the product of the quality of its elements and their diversity. The probability of Y is thus equal to the squared volume spanned by the vectors $q_i \phi_i$: the probability of a subset increases with the quality of its items, and decreases as two items become more similar.

1.1.4 DPPs algorithms

A brief overview of some standard operations over DPPs is given in the following section. DPPs allow for polynomial-time normalization, marginalization, conditioning, and sampling.

Due to the complexity of estimating a matrix determinant, the complexities are bounded by $\mathcal{O}(N^3)$ in the general case. However, in situations where the eigendecomposition of the DPP kernel is available, some algorithms have lower complexities.

Operation		Complexity
Normalization	$\sum_{Y \in \mathcal{Y}} \mathcal{P}(Y)$	$\mathcal{O}(N^3)$
Marginalization	$\mathcal{P}(A \subseteq Y)$	$\mathcal{O}(N^3)$
Conditioning	$\mathcal{P}(A \subseteq Y, B \cap Y = \emptyset)$	$\mathcal{O}(N^3)$
Sampling	$Y \sim \mathcal{P}$	$\mathcal{O}(N^3)$
Mode estimation	$\arg\max_Y \mathcal{P}(Y)$	$\mathcal{O}(2^N)$

Table 1.1: Overview of some standard operations over DPPs and their complexity

Mode estimation, however, is an NP-hard problem [23]; specifically, approximating $\max_Y \det(L_Y)$ to a factor $\frac{8}{9} + \varepsilon$ is NP-hard [29, Theorem 2.9]. Approximate so-

lutions include greedy algorithms [48] and submodular function maximization-based approaches [13].

Finally, the non-convex problem of learning a full DPP kernel non-parametrically from data, i.e. finding the L that maximizes the probability of observing n subsets

$$\arg\max_{L} \mathcal{P}_L(Y_1, \ldots, Y_n)$$

is also conjectured to be NP-hard [26, Conjecture 4.1]. Previous approaches include projected gradient ascent and Expectation-Maximization [14], and are based on learning the marginal kernel K. We present a new approach to this problem, using fixed-point analysis to learn instead the kernel L, in Chapter 2.

1.1.5 k-DPPs

In some cases, one may wish to have a probability distribution over subsets of the same size k. For example, it may be preferable to return exactly 5 answers in certain recommender systems, or to model the negative interactions of a fixed number of elements.

This can be done using a variant of DPPs called k-DPPs [27]. Although the algorithms described in Table 1.1 must be modified in the k-DPP setting, their complexities are similar to those of simple DPPs. Once again, when an eigendecomposition of the k-DPP's kernel is provided, the time complexity can be somewhat reduced.

k-DPPs have been used in various settings, such as document analysis [12] and image search engines [29, §5.3]. In Chapter 3 we apply k-DPPs to restructuring the architecture of neural networks.

1.2 Positive definite matrices

For clarity purposes, we provide here some of the key definitions and properties of positive definite matrices that are used in the following chapters.

Definition 1.2.1 (Positive definite matrices). *A matrix $M \in \mathbb{R}^{n\times n}$ is positive definite if it verifies one of the following properties, which are equivalent:*

(i) *M is symmetric and each eigenvalue λ of M verifies $\lambda > 0$*

(ii) *M is symmetric and M's leading principal minors are all strictly positive*

(iii) *M is symmetric and for $x \neq 0 \in \mathbb{R}^n$, $x^\top Mx > 0$.*

A matrix $M \in \mathbb{R}^{n\times n}$ is *positive semi-definite* if it satisfies the previous definition with non-strict inequalities.

It easily follows from definition 1.2.1 (*iii*) that the open set of positive definite matrices is convex (for $t \in [0,1]$, if $x^\top Ax > 0$ and $x^\top Bx > 0$, then it follows that $(1-t)\,x^\top Ax + t\,x^\top Bx > 0$).

Proposition 1.2.2. *Let $M = PDP$ be the spectral decomposition of a positive semi-definite matrix M. As all coefficients of the diagonal matrix D are non-negative, we can define $X = PD^{1/2}P$. X is positive semi-definite and verifies $X^2 = M$; we note this matrix $M^{1/2}$.*

Proposition 1.2.3. *For $A \in \mathbb{R}^{n\times n}$ positive semi-definite and $X \in \mathbb{R}^{n\times p}$, $X^\top AX$ is positive semi-definite. Moreover, if A is positive definite and X is of full rank, $X^\top AX$ is positive definite.*

Definition 1.2.4 (Partial order on positive definite matrices). *For $A, B \in \mathbb{R}^{n\times n}$ positive semi-definite, we write $A \succ B$ if $A - B$ is positive definite, and $A \succeq B$ if $A - B$ is positive semi-definite. This defines a partial order on the set of positive semi-definite matrices.*

Theorem 1.2.5. *This order verifies $A \succ B \succ 0 \implies A^{-1} \prec B^{-1}$.*

Proof. Equivalently, we prove that for $A, B \in \mathbb{R}^{n\times n}$ positive definite, $(A+B)^{-1} \prec A^{-1}$, using Def 1.2.1 (*iii*).

Let $x \in \mathbb{R}^n \neq 0$ and $y = (A+B)^{-1}x$.

$$\begin{aligned} x^\top A^{-1} x &= y^\top (A+B) A^{-1} (A+B) y^\top \\ &= y^\top (A + 2B + BA^{-1}B) y \\ &= y^\top (A+B) y + y^\top B (B^{-1} + A^{-1}) B y \\ &= x^\top (A+B)^{-1} x + (By)^\top (B^{-1} + A^{-1}) By \end{aligned}$$

As both A^{-1} and B^{-1} are positive definite, $(By)^\top (B^{-1} + A^{-1}) By > 0$, and hence

$$x^\top A^{-1} x > x^\top (A+B)^{-1} x,$$

i.e. $(A+B)^{-1} \prec A^{-1}$. □

Theorem 1.2.6. $\log\det$ *is defined and concave on the open set of positive definite matrices, and verifies* $\nabla(\log\det)(A) = A^{-1}$.

Proof. For $A \in \mathbb{R}^{n\times n}$ positive definite, $\det(A) > 0$: $\log\det$ is defined on the set of positive definite matrices.

By using the expansion of $\det A$ according to its cofactors C_{ij}, we have that

$$\begin{aligned} \frac{\partial \log\det A}{\partial a_{ij}} &= \frac{1}{\det A} \frac{\partial \det A}{\partial a_{ij}} \\ &= \frac{1}{\det A} \frac{\partial}{\partial a_{ij}} \left(\sum_{k=1}^{n} a_{ik} C_{ik} \right) \\ &= \frac{1}{\det A} C_{ij} \end{aligned}$$

As A is symmetric, its cofactor matrix C is as well; thus, as $A^{-1} = \frac{1}{\det A} C^\top$, it follows that $\frac{\partial \log\det A}{\partial a_{ij}} = A^{-1}_{ij}$, i.e.

$$\nabla(\log\det)(A) = A^{-1}.$$

Finally, we prove the concavity of $\log\det$ by proving its concavity when restricted

to an arbitrary line. For $t \geq 0$ and $A, B \in \mathbb{R}^{n \times n}$ positive definite,

$$\begin{aligned}\log\det(A + tB) &= \log\det(A) + \log\det(I + tA^{-1/2}BA^{-1/2}) \\ &= \log\det(A) + \sum_{i=1}^{n} \log(1 + t\lambda_i)\end{aligned}$$

where the $\lambda_i > 0$ are the eigenvalues of $A^{-1/2}BA^{-1/2}$. Thus, $\log\det$ is concave on the set of positive definite matrices. □

1.3 Contributions

The key contributions of this thesis are the following:

- Learning the DPP kernel (this work was previously published in [39])
  - **Picard iteration**: we introduce a simple, fast and flexible fixed-point algorithm for learning the kernel of a DPP from observed data. The Picard iteration guarantees positive definiteness of the kernel estimate L at each iteration and does not require any projection step. Furthermore, it also guarantees to increase the log-likelihood of the model at each iteration.
  - **Generalized Picard iteration**: we additionally describe a more powerful, generalized version of the Picard iteration which does not guarantee a systematic increase in log-likelihood; however, it experimentally provides significantly faster convergence.
  - **Proof of convergence**: we provide a proof of convergence for the Picard iteration based on non-convex optimization theory. For the generalized version of the Picard algorithm, we present bounds on the admissible step-size, and conjecture that these bounds may be exact.
- Diversity networks (also available in [40])
  - **Diversity model for neural networks**: we introduce the use of Determinantal Point Processes as a flexible, powerful tool for modeling layer-wise

neuronal diversity. Specifically, we present a practical method for creating DPPs over a set of neurons by analyzing each neuron's activation vector over training input, thus allowing diversity-promoting sampling.

- **Fusing procedure**: we present a simple but powerful "fusing" procedure that minimizes the adverse effects of removing neurons from a layer in a neural network by transferring the contributions of the pruned neurons to the ones that remain.
- **DIVNET**: by using the two previous algorithms in succession, redundant neurons in a layer can be removed without significantly impacting the network's overall calculation. This procedure, DIVNET, provides a simple approach to reducing the memory footprint of neural networks without overly penalizing their performance, and does not require further training of the network after the pruning. Additionally, DIVNET is independent from most hyperparameters of a network (number of layers, activation functions, etc.) and as such can be combined with other approaches to size reduction in neural networks.

Both contributions are validated on multiple datasets. We evaluate the Picard iteration on synthetic and real-world data, compare its performance to that of Expectation-Maximization, and show that the Picard iteration reaches the same log-likelihood value at a much faster rate. As for DIVNET, we evaluate it on fully-connected neural networks of varying sizes, trained on several common image recognition datasets: `MNIST`, `MNIST_ROT` and `CIFAR-10`, and show that DIVNET greatly outperforms a previous neuron-pruning approach for neural networks.

1.4 Thesis organization

We introduce the Picard iteration in Chapter 2, and present theoretical results regarding its convergence in §2.2. Experimental results on synthetic and real-world data are described in §2.3.

We apply Determinantal Point Processes to reorganizing the structure of neural networks in order to reduce their memory footprint in Chapter 3. We describe how to place a DPP over layers of a network and how to balance the network after pruning in §3.3, and discuss theoretical results in §3.4.

Chapter 2

The Picard iteration

2.1 Introduction

The work presented in this chapter was motivated by the recent work of Gillenwater et al. [14], who made notable progress on the task – conjectured to be NP-Hard [26, Conjecture 4.1] – of learning a DPP kernel from data. Gillenwater et al. [14] presented a carefully designed EM-style procedure, which, unlike several previous approaches (e.g., [28, 27, 3]) learns a full DPP kernel nonparameterically.

One main observation of Gillenwater et al. [14] is that applying projected gradient ascent to the DPP log-likelihood usually results in degenerate estimates (because it involves projection onto the set $\{X : 0 \preceq X \preceq I\}$). Hence, one may wonder if instead more sophisticated manifold optimization techniques [1, 8] could be applied. While this idea is attractive, and indeed applicable, e.g., via the MANOPT toolbox [8], empirically it turns out to be computationally too demanding; the EM strategy of Gillenwater et al. [14] is more practical.

Here, we depart from both EM and manifold optimization to develop the Picard iteration, a new learning algorithm that, compared to EM,

- is significantly simpler
- yields essentially the same log-likelihood values
- runs significantly faster. In particular, our algorithm runs an order of magnitude

faster than EM on larger problems.

The key innovation of our approach is a derivation via a fixed-point view, which by construction ensures positive definiteness at every iteration. Its convergence analysis involves an implicit bound-optimization iteration to ensure monotonic ascent. A pleasant byproduct of the fixed-point approach is that it avoids any eigenvalue/vector computations, enabling further savings in running time.

2.1.1 Problem setup: learning the DPP Kernel

Parameterizations (2.1.1) and (2.1.2) of the DPP probability are both useful in this context: Gillenwater et al. [14] used a formulation in terms of K; we prefer (2.1.1) as it aligns better with our algorithmic approach.

$$\mathcal{P}(Y) = \frac{\det(L_Y)}{\det(L+I)}, \qquad Y \subseteq \mathcal{Y}. \tag{2.1.1}$$

$$\mathcal{P}(A \subseteq Y) = \det(K_A). \tag{2.1.2}$$

The learning task aims to fit a DPP kernel (either L or equivalently the marginal kernel K) consistent with a collection of observed subsets: suppose we obtain as training data n subsets $(Y_1, \ldots, Y_n)$ of the ground set $\mathcal{Y}$; the task is then to maximize the likelihood of these observations. Two equivalent formulations of this maximization task may be considered:

$$\max_{L \succeq 0} \sum_{i=1}^{n} \log\det(L_{Y_i}) - n \log\det(I+L), \tag{2.1.3}$$

$$\max_{0 \preceq K \preceq I} \sum_{i=1}^{n} \log\big(|\det(K - I_{Y_i^c})|\big). \tag{2.1.4}$$

We will use formulation (2.1.3) in this paper. Gillenwater et al. [14] used formulation (2.1.4) and exploited its structure to derive a somewhat intricate EM-style method for its optimization.

Both (2.1.3) and (2.1.4) are nonconvex and difficult to optimize. For instance, using projected gradient on (2.1.4) may seem tempting, but projection ends up yielding

degenerate (diagonal and rank-deficient) solutions, which is undesirable when trying to capture interactions between items – indeed, this criticism motivated Gillenwater et al. [14] to derive the EM algorithm.

We approach problem (2.1.3) from a different viewpoint (which also avoids projection) and as a result obtain a new optimization algorithm for estimating L. This algorithm, its analysis, and empirical performance are the subject of the remainder of this chapter.

2.2 The Picard iteration

The method that we derive has two key components:

- a fixed-point view that helps obtain an iteration that satisfies the crucial positive definiteness constraint $L \succeq 0$ by construction
- an implicit bound optimization-based analysis that ensures monotonic ascent

If $|Y| = k$, then for a suitable $N \times k$ indicator matrix U we can write $L_Y = U^* L U$, which is also known as a *compression* (U^* denotes the Hermitian transpose[1]). We write $U_i^* L U_i$ interchangeably with L_{Y_i}, implicitly assuming suitable indicator matrices U_i such that $U_i^* U_i = I_{|Y_i|}$. We will drop the subscript on the identity matrix, its dimension being clear from context.

Denote by $\phi(L)$ the objective function in (2.1.3). Assume for simplicity that the constraint set is open, i.e., $L \succ 0$. Then any critical point of the log-likelihood must satisfy the following condition:

$$\begin{aligned} &\nabla\phi(L) = 0, \quad \text{or equivalently} \\ &\sum\nolimits_{i=1}^{n} U_i \left(U_i^* L U_i\right)^{-1} U_i^* - n\left(I + L\right)^{-1} = 0. \end{aligned} \tag{2.2.1}$$

Any (strictly) positive definite solution to the nonlinear matrix equation (2.2.1) is a candidate locally optimal solution.

[1] Although we present our theoretical analysis in the broader setting of complex matrices, in practice their coefficients are in $\mathbb{R}$.

We solve this matrix equation by developing a fixed-point iteration. In particular, define

$$\Delta := \tfrac{1}{n}\sum\nolimits_{i=1}^{n} U_i \left(U_i^* L U_i\right)^{-1} U_i^* - (I+L)^{-1},$$

with which we may equivalently write (2.2.1) as

$$\Delta + L^{-1} = L^{-1}. \tag{2.2.2}$$

Equation (2.2.2) suggests the following iteration

$$L_{k+1}^{-1} \leftarrow L_k^{-1} + \Delta_k, \qquad k = 0, 1, \ldots. \tag{2.2.3}$$

A priori there is no reason for iteration (2.2.3) to be valid (i.e., to converge to a stationary point). But we write it in this form to highlight its crucial feature: starting from an initial $L_0 \succ 0$, it generates positive definite iterates (Prop. 2.2.1).

Proposition 2.2.1. *Let $L_0 \succ 0$. Then, the sequence $\{L_k\}_{k\geq 1}$ generated by (2.2.3) remains positive definite.*

Proof. The proof is by induction. It suffices to show that

$$L \succ 0 \implies L^{-1} + \Delta \succ 0.$$

Since $I + L \succ L$, from the order inversion property of the matrix inverse map it follows that $L^{-1} \succ (I+L)^{-1}$.

Thus, since $\frac{1}{n}\sum_{i=1} U_i \left(U_i^* L U_i\right)^{-1} U_i^* \succeq 0$, we have

$$L^{-1} + \tfrac{1}{n}\sum_{i=1} U_i \left(U_i^* L U_i\right)^{-1} U_i^* \succ (I+L)^{-1} + \tfrac{1}{n}\sum_{i=1} U_i \left(U_i^* L U_i\right)^{-1} U_i^*$$

$$L^{-1} + \Delta \succ \tfrac{1}{n}\sum_{i=1} U_i \left(U_i^* L U_i\right)^{-1} U_i^* \succeq 0$$

□

A quick experiment reveals that iteration (2.2.3) *does not converge* to a local maximizer of $\phi(L)$. To fix this defect, we rewrite the key equation (2.2.2) in a different manner:

$$L = L + L\Delta L. \tag{2.2.4}$$

This equation is obtained by multiplying (2.2.2) on the left and right by L. Therefore, we now consider the iteration

$$L_{k+1} \leftarrow L_k + L_k \Delta_k L_k, \quad k = 0, 1, \ldots. \tag{2.2.5}$$

Prop. 2.2.1, in combination with the fact that congruence preserves positive definiteness (i.e., if $X \succeq 0$, then $Z^* X Z \succeq 0$ for any matrix Z), implies that if $L_0 \succ 0$, then the sequence $\{L_k\}_{k \geq 1}$ obtained from iteration (2.2.5) is also positive definite. What is more remarkable is that, contrary to iteration (2.2.3), the sequence generated by (2.2.5) monotonically increases the log-likelihood.

While monotonicity is not evident from our derivation above, it becomes apparent once we recognize an implicit change of variables that seems to underlie our method.

2.2.1 Convergence Analysis

Lemma 2.2.2. *Let $U \in \mathbb{C}^{N \times k}$ $(k \leq N)$ such that $U^*U = I$. The map $g(S) := \log\det(U^* S^{-1} U)$ is convex on the set of positive definite matrices.*

Proof. Since g is continuous it suffices to establish midpoint convexity. Consider therefore, $X, Y \succ 0$ and let

$$X \# Y := X^{1/2}(X^{-1/2} Y X^{-1/2})^{1/2} X^{1/2} \tag{2.2.6}$$

be their geometric mean. It's easy to see from Eq. 2.2.6 that

$$\det(X \# Y) = \sqrt{\det X}\sqrt{\det Y} \tag{2.2.7}$$

and

$$(X\#Y)^{-1} = X^{-1}\#Y^{-1}. \tag{2.2.8}$$

The operator inequality $X\#Y \preceq \frac{X+Y}{2}$ is well-known [6, Theorem 4.1.3].

Hence,

$$\left(\tfrac{X+Y}{2}\right)^{-1} \preceq (X\#Y)^{-1} = X^{-1}\#Y^{-1} \tag{2.2.9}$$

$$U^*\left(\tfrac{X+Y}{2}\right)^{-1} U \preceq U^*(X^{-1}\#Y^{-1})U \tag{2.2.10}$$

$$\preceq (U^*X^{-1}U)\#(U^*Y^{-1}U), \tag{2.2.11}$$

where the final inequality follows from [6, Theorem 4.1.5] (see [43, Theorem 8] for an explicit proof).

Since $\log\det$ is monotonic on positive definite matrices, it then follows from 2.2.8 and 2.2.11 that

$$\begin{aligned}\log\det\Big(U^*\left(\tfrac{X+Y}{2}\right)^{-1}U\Big) &\leq \log\det\left((U^*X^{-1}U)\#(U^*Y^{-1}U)\right)\\ &\leq \tfrac{1}{2}\log\det(U^*X^{-1}U) + \tfrac{1}{2}\log\det(U^*Y^{-1}U)\end{aligned}$$

which proves the lemma. □

Theorem 2.2.3. *Let L_k be generated via (2.2.5). Then, the sequence $\{\phi(L_k)\}_{k\geq 0}$ is monotonically increasing.*

Proof. The key insight is to consider $S = L^{-1}$ instead of L; this change is only for the analysis—the actual iteration that we implement is still (2.2.5).

Writing $\psi(S) := \phi(L)$, we have

$$\begin{aligned}\psi(S) &= \tfrac{1}{n}\sum\nolimits_i \log\det(U_i^*S^{-1}U_i) - \log\det(S^{-1}+I)\\ &= \tfrac{1}{n}\sum\nolimits_i \log\det(U_i^*S^{-1}U_i) + \log\det(S) - \log\det(I+S).\end{aligned}$$

We define

$$h(S) = \tfrac{1}{n}\sum_i \log\det(U_i^* S^{-1} U_i) - \log\det(I+S)$$
$$f(S) = \log\det(S).$$

Clearly, f is concave in S, while h is convex is S; the latter from Lemma 2.2.2 and the fact that $-\log\det(I+S)$ is convex. This observation allows us to invoke iterative bound-optimization (an idea that underlies EM, CCCP [46], and other related algorithms).

We construct an *auxiliary function* ξ so that

$$\psi(S) \geq \xi(S,R), \qquad \forall S, R \succ 0,$$
$$\psi(S) = \xi(S,S), \qquad \forall S \succ 0.$$

As in [46], we select ξ by exploiting the convexity of h: as $h(S) \geq h(R) + \langle \nabla h(R),\, S-R\rangle$, we simply set

$$\xi(S,R) := f(S) + h(R) + \langle \nabla h(R) \,|\, S-R\rangle\,.$$

Given an iterate S_k, we then obtain S_{k+1} by solving

$$S_{k+1} := \operatorname{argmax}_{S\succ 0}\ \xi(S, S_k), \tag{2.2.12}$$

which clearly ensures monotonicity: $\psi(S_{k+1}) \geq \psi(S_k)$.

Since (2.2.12) has an open set as a constraint and $\xi(S,\cdot)$ is strictly concave, to solve (2.2.12) it suffices to solve the necessary condition $\nabla_S \xi(S, S_k) = 0$. This amounts to

$$S^{-1} = \ (I+S_k)^{-1} + \tfrac{1}{n}\sum\nolimits_i S_k^{-1} U_i (U_i^* S_k^{-1} U_i)^{-1} U_i^* S_k^{-1}.$$

Rewriting in terms of L we immediately see that by setting

$$L_{k+1} = L_k + L_k \Delta_k L_k,$$

we have

$$\phi(L_{k+1}) \geq \phi(L_k)$$

(the inequality is strict unless $L_{k+1} = L_k$). □

Theorem 2.2.3 shows that iteration (2.2.5) is well-defined (positive definiteness was established by Prop. 2.2.1).

2.2.2 Generalized Picard iteration

The fixed-point formulation (2.2.5) actually suggests a broader iteration, with an additional step-size a:

$$L_{k+1} = L_k + aL_k\Delta_k L_k. \tag{2.2.13}$$

Above we showed that for $a = 1$ ascent is guaranteed. Empirically, $a > 1$ often also provides good results and faster convergence.

Prop. 2.2.4 presents an easily computable upper bound on feasible a:

Proposition 2.2.4. *Let L, U_i, and Δ be as defined above. Let*

$$Z = \tfrac{1}{n}\sum_i U_i \left(U_i^* L U_i\right)^{-1} U_i^*.$$

Define the constant

$$\gamma := \max\{\lambda_{\min}(LZ), 1/\lambda_{\max}(I + L)\}. \tag{2.2.14}$$

Then, $0 \leq \gamma \leq 1$ and for $a \leq (1-\gamma)^{-1}$ the update

$$L' \leftarrow L + aL\Delta L$$

ensures that L' is also positive definite.

Proof. Let $Z = \frac{1}{n}\sum_{i=1}^n U_i \left(U_i^* L U_i\right)^{-1} U_i^*$.

To ensure $L + aL\Delta L \succ 0$ we equivalently show

$$\begin{aligned}
&L^{-1} + a\Big(\tfrac{1}{n}\sum_{i=1}^{n} U_i \left(U_i^* L U_i\right)^{-1} U_i^* - (L+I)^{-1}\Big) \succ 0 \\
&\implies L^{-1} + aZ \succ a\,(L+I)^{-1} \\
&\implies I + aL^{1/2} Z L^{1/2} \succ aL\,(L+I)^{-1} \\
&\implies I + aL^{1/2} Z L^{1/2} \succ a(I - (I+L)^{-1}) \\
&\implies (1-a)I + a(I+L)^{-1} + aL^{1/2} Z L^{1/2} \succ 0 \\
&\implies (1-a) + a\lambda_{\min}\left((I+L)^{-1} + L^{1/2} Z L^{1/2}\right) > 0.
\end{aligned}$$

This inequality can be numerically optimized to find the largest feasible value of a. The simpler bound in question can be obtained by noting that

$$\lambda_{\min}((I+L)^{-1} + L^{1/2} Z L^{1/2}) \geq \max\{\lambda_{\min}(LZ), 1/\lambda_{\max}(I+L)\} = \gamma.$$

Thus, we have the easily computable bound for feasible a:

$$a \leq \frac{1}{1-\gamma}.$$

Clearly, by construction $\gamma \geq 0$. To see why $\gamma \leq 1$, observe that $(I+L) \prec I$, so that $\lambda_{\min}((I+L)^{-1}) < 1$. Further, block-matrix calculations show that $Z \preceq L^{-1}$, whereby $\lambda_{\min}(L^{1/2} Z L^{1/2}) \leq \lambda_{\min}(I) = 1$. □

Conjecture 2.2.5. *We conjecture that for all feasible values $a \geq 1$, iteration (2.2.5) is guaranteed to increase the log-likelihood.*

2.2.3 Picard iteration for feature matrices

The Picard iteration can be adapted to the case where $L = F^* W F$, where $F \in \mathbb{R}^{N \times d}$ is a fixed feature matrix and $W \in \mathbb{R}^{d \times d}$ is the weight matrix that we wish to learn.

In this case, the $L_{k+1} \leftarrow L_k + L_k \Delta_k L_k$ update is rewritten as

$$F^* W_{k+1} F \leftarrow F^* W_k F + F^* W_k F \Delta F^* W_k F. \tag{2.2.15}$$

By applying the following update on W:

$$W_{k+1} \leftarrow W_k + W_k F \Delta_k F^* W_k \tag{2.2.16}$$

the update from 2.2.15 is verified. Thus, the Picard iteration also generalizes to the case of feature-based DPP learning via the update from 2.2.16, with similar guarantees to those of the previous section.

2.2.4 Pseudocode

Pseudocode of our resulting learning method is presented in Algorithms 1 and 2.

Algorithm 1 Picard Iteration

Input: Matrix L, training set T, step-size $a > 0$.
for $i = 1$ **to** maxIter **do**
  $L \longleftarrow$ FixedPointMap(L, T, a)
  if stop(L, T, i) **then**
    break
  end if
end for
return L

Algorithm 2 FixedPointMap

Input: Matrix L, training set T, step-size $a > 0$
$Z \longleftarrow 0$
for Y **in** T **do**
  $Z_Y = Z_Y + L_Y^{-1}$
end for
return $L + aL(Z/|T| - (L + I)^{-1})L$

2.2.5 Iteration cost and convergence speed

The cost of each iteration of our algorithm is dominated by the computation of Δ, which costs a total of $O(\sum_{i=1}^{n} |Y_i|^3 + N^3) = O(n\kappa^3 + N^3)$ arithmetic operations, where $\kappa = \max_i |Y_i|$. The $O(|Y_i|^3)$ cost comes from computing the inverse $L_{Y_i}^{-1}$, while the N^3 cost stems from computing $(I + L)^{-1}$. Moreover, additional N^3 costs arise when computing the $L\Delta L$ product.

In comparison, each iteration of the method of Gillenwater et al. [14] has a complexity of $O(nN\kappa^2 + N^3)$, which is comparable to, though slightly greater than, $O(n\kappa^3 + N^3)$, as $N \geq \kappa$. In applications where the sizes of the sampled subsets satisfy $\kappa \ll N$, the difference can be more substantial. Moreover, we do not need any eigenvalue/vector computations to implement our algorithm.

Finally, the Picard iteration also runs slightly faster than a K-Ascent iteration, which costs $O(nN^3)$. Additionally, similarly to EM, our algorithm avoids the projection step necessary in the K-Ascent algorithm (in order to insure that K is a valid marginal kernel, i.e. $K \in \{X : 0 \preceq X \preceq I\}$). As shown in [14], avoiding this step is beneficial, as it helps learn non-diagonal matrices.

We note in passing that similarly to EM, assuming a non-singular local maximum, we can also obtain a local linear rate of convergence. This follows by relating iteration (2.2.5) to scaled-gradient methods [5, §1.3] (taking into account the fact that we have an implicit positive semi-definite constraint).

2.3 Experimental results

We compare the performance of our algorithm, referred to as *Picard iteration*[2], against the EM algorithm presented in Gillenwater et al. [14]. We experiment on both synthetic and real-world data.

[2]Our nomenclature stems from the usual name for such iterations in fixed-point theory [15].

2.3.1 Baby registries dataset

For real-world data, we use the baby registry dataset on which results are reported in [14]. This dataset consists in 111,006 sub-registries describing items across 13 different categories; this dataset was obtained by collecting baby registries from `amazon.com`, all containing between 5 and 100 products, and then splitting each registry into subregistries according to which of the 13 categories (such as "feeding", "diapers", "toys", etc.) each product in the registry belongs to. [14] provides a more in-depth description of this dataset.

These sub-registries are used to learn a DPP capable of providing recommendations for these products: indeed, a DPP is well-suited for this task as it provides sets of products in a category that are popular yet diverse enough to all be of interest to a potential customer.

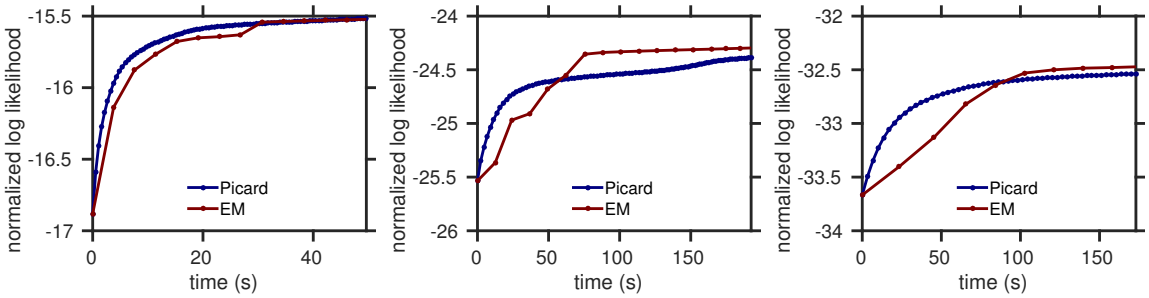
2.3.2 Implementation details

We measure convergence by testing the relative change $\frac{|\phi(L_{k+1})-\phi(L_k)|}{|\phi(L_k)|} \leq \varepsilon$ in the log-likelihood. We used a tighter convergence criterion for our algorithm ($\varepsilon_{\text{pic}} = 0.5 \cdot \varepsilon_{\text{em}}$) to account for the fact that the distance between two subsequent log-likelihoods tends to be smaller for the Picard iteration than for EM.

The parameter a for Picard was set at the beginning of each experiment and never modified as it remained valid throughout each test. However, although it was not necessary in our experiments, if the parameter a becomes invalid, it can be halved until it reaches 1.

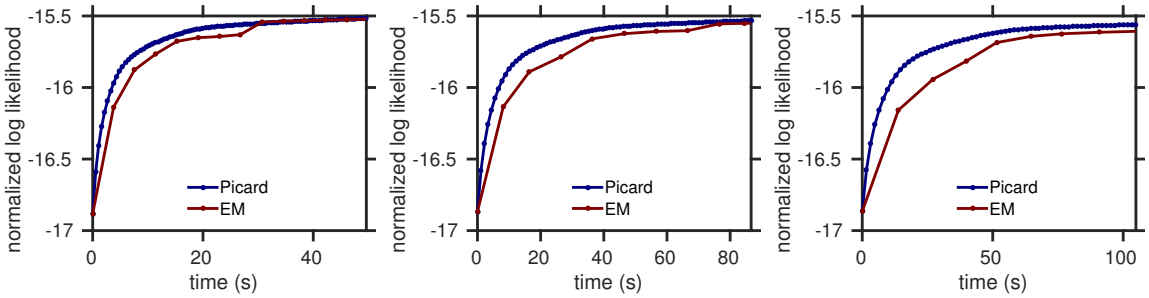
In EM, the step size was initially set to 1 and halved when necessary, as per the algorithm described in [14]; we used the code of Gillenwater et al. [14] for our EM implementation[3].

[3]These experiments were run with MATLAB, on a Linux Mint system, using 16GB of RAM and an i7-4710HQ CPU @ 2.50GHz.



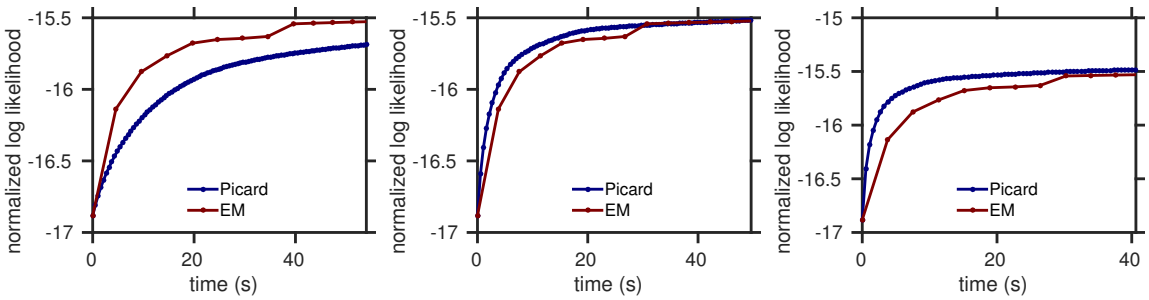
(a) $N = 50$ (b) $N = 100$ (c) $N = 150$

Figure 2-1: Normalized log-likelihood as a function of time for various set sizes N, with $n = 5000$ and $a = 5$ using the `BASIC` random distribution.



(a) $n = 5000$ (b) $n = 10000$ (c) $n = 15000$

Figure 2-2: Normalized log likelihood as a function of time for various numbers of training sets, with $N = 50$ and $a = 5$ using the `BASIC` random distribution.



(a) $a = 1$ (b) $a = 5$ (c) $a = 10$

Figure 2-3: Normalized log likelihood as a function of time for different values of a, with $N = 50$ and $n = 5000$ using the `BASIC` random distribution.

2.3.3 Synthetic tests

In each experiment, we sample n training sets from a base DPP of size N, then learn the DPP using EM and the Picard iteration. We initialize the learning process with a random positive definite matrix L_0 (or K_0 for EM) drawn from the same distribution as the true DPP kernel.

Specifically, we used two matrix distributions to draw the true kernel and the initial matrix values from:

- `BASIC`: We draw the coefficients of a matrix M from the uniform distribution over $\left[0, \sqrt{2}\right]$, then return $L = MM^\top$ conditioned on its positive definiteness.
- `WISHART`: We draw L from a Wishart distribution with N degrees of freedom and an identity covariance matrix, and rescale it with a factor $\frac{1}{N}$.

Figures 2-1, 2-2 and 2-3 show the log-likelihood as a function of time for different parameter values when both the true DPP kernel and the initial matrix L_0 were drawn from the `BASIC` distribution. Tables 2.1 and 2.2 show the final log-likelihood and the time necessary for each method to reach 99% of the optimal log likelihood for both distributions and parameters $n = 5000$, $a = 5$.

As shown in Figure 2-1, the difference in time necessary for both methods to reach a good approximation of the final likelihood (as defined by best final likelihood) grows drastically as the size N of the set of all elements $\{1, 2, \ldots, N\}$ increases. Figure 2-2 illustrates the same phenomenon when N is kept constant and n increases.

Finally, the influence of the parameter a on convergence speed is illustrated in Figure 2-3[4]. Larger a values noticeably increase Picard's convergence speed, as long as the matrices remain positive definite during the Picard iteration.

The greatest strength of the Picard iteration lies in its initial rapid convergence: the log-likelihood increases significantly faster for the Picard iteration than for EM. Although for small datasets EM sometimes performs better, our algorithm provides substantially better results in shorter timeframes, especially when dealing with larger

[4]In the cases where $a > 1$, a safeguard was added to check that the matrices returned by our algorithm were positive definite.

Table 2.1: Final log-likelihoods and time necessary for an iteration to reach 99% of the optimal log likelihood for both algorithms when using `BASIC` distribution for true and initialization matrices (training set size of 5,000, $a = 5$).

	Log-Likelihood		Runtime to 99%	
	Picard	EM	Picard	EM
$N = 50$	-15.5	-15.5	17.3s	30.7s
$N = 100$	-24.4	-24.2	143s	75.5s
$N = 150$	-32.5	-32.5	40.7s	84.0s
$N = 200$	-40.8	-41.2	51.1s	1,730s
$N = 250$	-45.7	-46.0	99.1s	2,850s

Table 2.2: Final log-likelihoods and time necessary for an iteration to reach 99% of the optimal log likelihood for both algorithms when using `WISHART` distribution for true and initialization matrices (training set size of 5,000, $a = 5$).

	Log-Likelihood		Runtime to 99%	
	Picard	EM	Picard	EM
$N = 50$	-33.0	-33.1	0.2s	2.0s
$N = 100$	-66.2	-66.2	0.5s	3.6s
$N = 150$	-99.2	-99.3	0.8s	5.2s
$N = 200$	-132.1	-132.4	1.2s	8.9s
$N = 250$	-165.1	-165.7	2.5s	11s

datasets, due amongst others to the smaller complexity (and thus shorter runtime) of one Picard iteration compared to one EM iteration.

Overall, our algorithm converges to 99% of the optimal log-likelihood (defined as the maximum of the log-likelihoods returned by each algorithm) significantly faster than the EM algorithm for both distributions, particularly when dealing with large values of N.

Thus, the Picard iteration is preferable when dealing with large ground sets; it is also very well-suited to cases where larger amounts of training data are available.

2.3.4 Baby registries experiment

We tested our implementation on all 13 product categories in the baby registry dataset, using two different initializations:

- the aforementioned Wishart distribution
- the data-dependent moment matching initialization (MM) described in [14]

In each case, 70% of the baby registries in the product category were used for training; 30% served as test. The results presented in Figures 2-4 and 2-5 are averaged over 5 learning trials, each with different initial matrices.

The parameter a was set equal to 1.3 for all iterations.

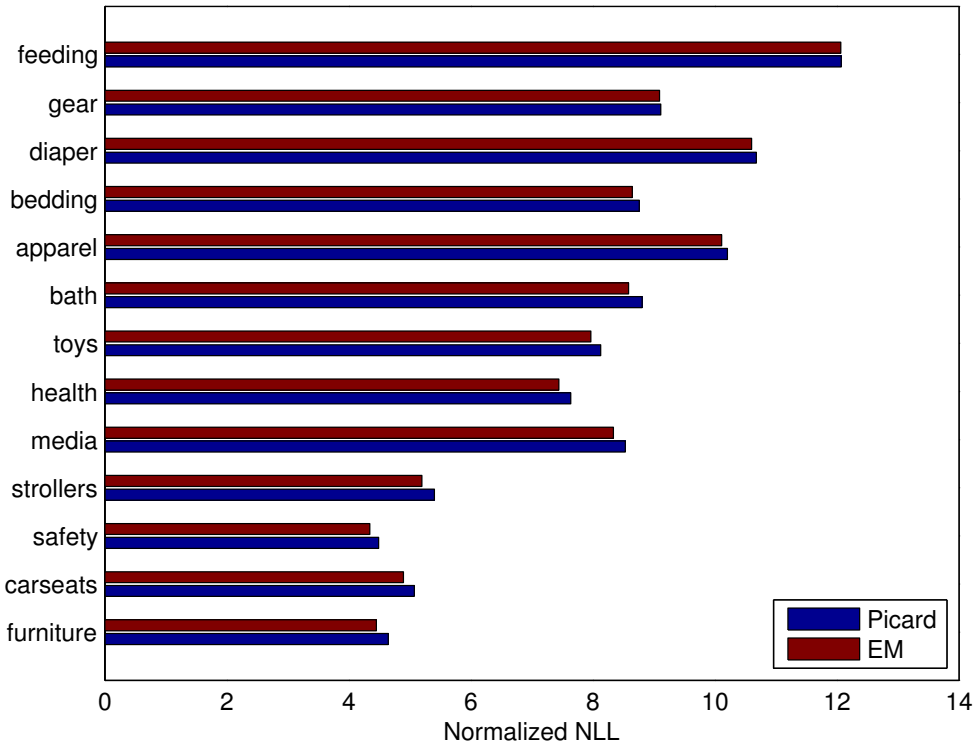
Similarly to its behavior on synthetic datasets, the Picard iteration provides overall significantly shorter runtimes when dealing with large matrices and training sets. As shown in Table 2.3, the final log-likelihoods are very close (on the order of 10^{-2} to 10^{-4}) to those attained by the EM algorithm.

Using a moments-matching initialization leaves Picard's runtimes overall unchanged (a notable exception being the 'gear' category). However, EM's runtime decreases drastically with this initialization, although it remains significantly longer than Picard's in most categories.

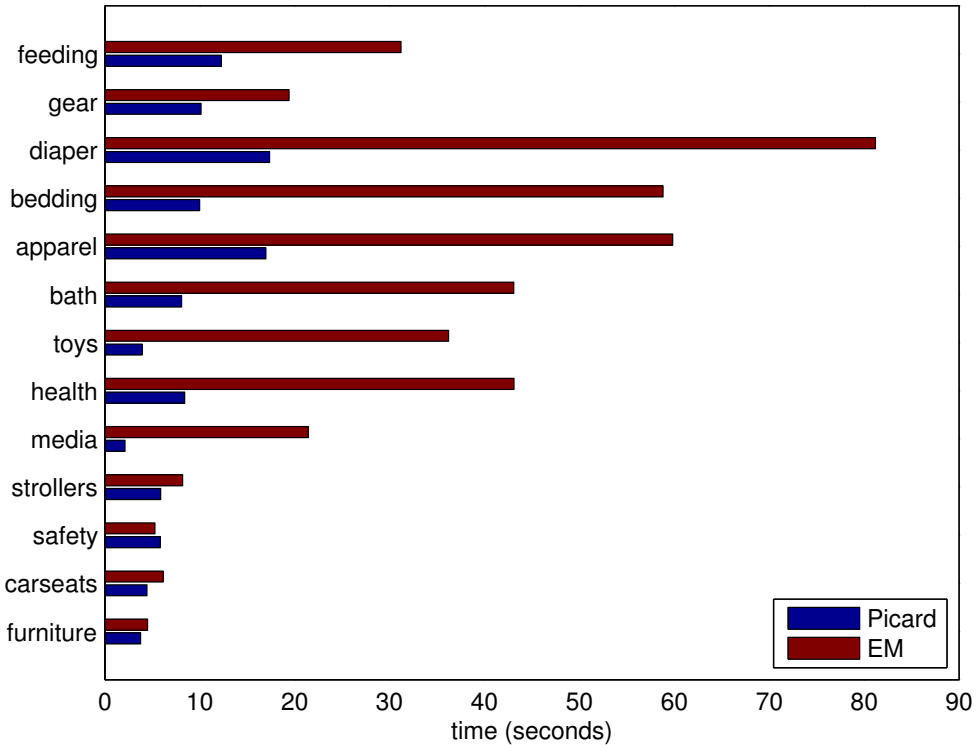
The final log-likelihoods are also closer when using moments-matching initialization (Table 2.3).

Table 2.3: Comparison of final log-likelihoods on all product categories for both algorithms. δ is the relative closeness between Picard and EM: $\delta = |\phi_{\text{em}} - \phi_{\text{pic}}|/\phi_{\text{em}}$.

Category	δ (Wishart)	δ (MM)
furniture	4.4e-02	1.2e-03
carseats	3.7e-02	7.6e-04
safety	3.3e-02	8.0e-04
strollers	3.9e-02	3.0e-03
media	2.3e-02	2.4e-03
health	2.6e-02	7.4e-03
toys	2.0e-02	5.9e-03
bath	2.6e-02	2.9e-03
apparel	9.2e-03	4.3e-03
bedding	1.3e-02	7.6e-03
diaper	7.2e-03	5.3e-03
gear	2.3e-03	9.0e-03
feeding	4.9e-04	2.1e-03

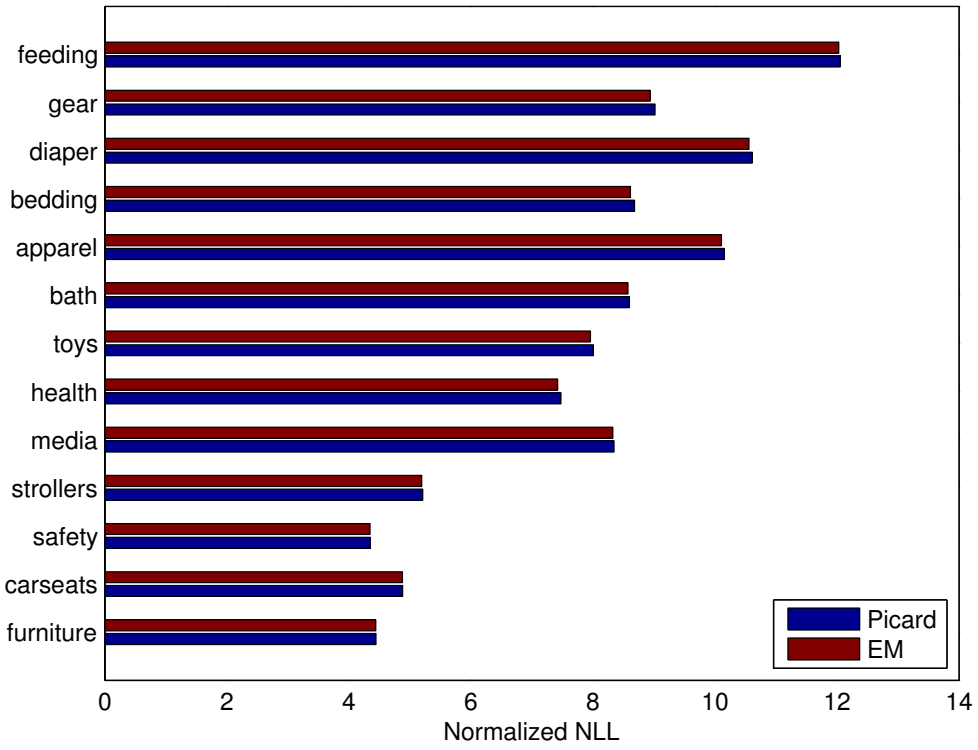


(a) Final negative log-likelihood

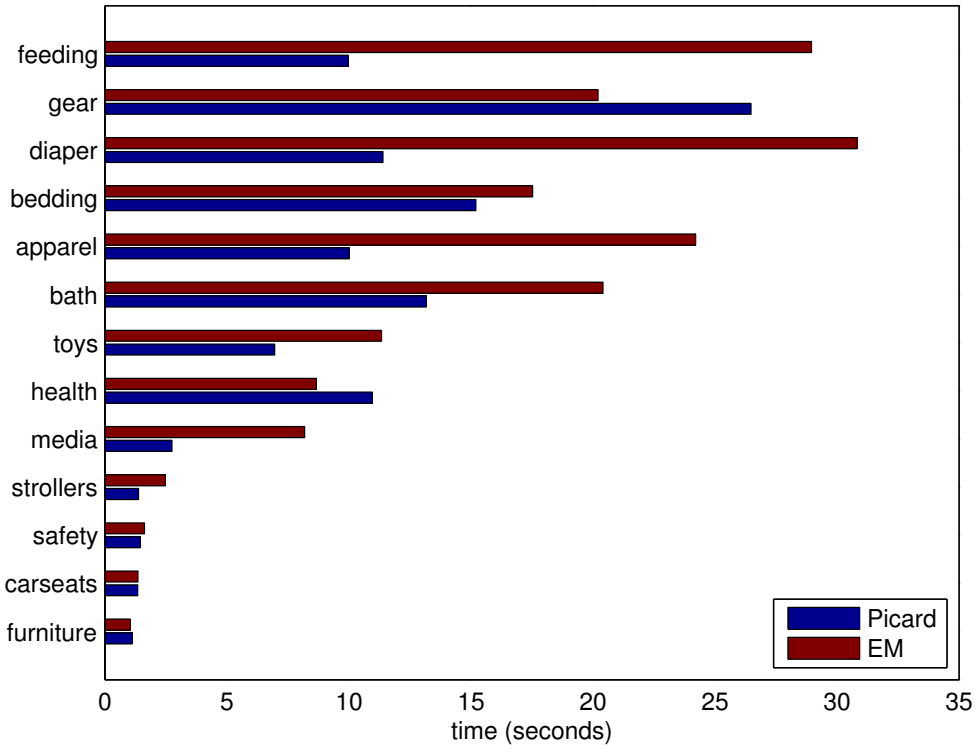


(b) Runtime

Figure 2-4: Evaluation of EM and the Picard iteration on the baby registries dataset using Wishart initialization.



(a) Final negative log-likelihood



(b) Runtime

Figure 2-5: Evaluation of EM and the Picard iteration on the baby registries dataset using moments-matching initialization.

Chapter 3

Diversity networks

3.1 Introduction

Training neural networks requires setting several hyper-parameters to adequate values: number of hidden layers, number of neurons per hidden layer, learning rate, momentum, dropout rate, etc. Although tuning such hyper-parameters via parameter search has been recently investigated by Maclaurin et al. [38], doing so remains extremely costly, which makes it imperative to develop more efficient techniques.

Of the many hyper-parameters, those that determine the network's architecture are among the hardest to tune, especially because changing them during training is more difficult than adjusting more dynamic parameters such as the learning rate or momentum. Typically, the architecture parameters are set once and for all before training begins. Thus, assigning them correctly is paramount: if the network is too small, it will not learn well; if it is too large, it may take significantly longer to train while running the risk of overfitting. Networks are therefore typically trained with more parameters than necessary, and pruned once the training is complete.

This chapter introduces DIVNET, a technique for reducing the size of a network. DIVNET decreases the amount of redundancy in a neural network (and hence reduces its size as well) in two steps: first, it samples a diverse subset of neurons; then, it merges the remaining neurons with the ones previously selected.

Specifically, DIVNET models neuronal diversity by placing a Determinantal Point

Process [21] over neurons in a layer, which is then used to select a subset of diverse neurons. Subsequently, DIVNET "fuses" information from the dropped neurons into the selected ones through a reweighting procedure. Together, these steps reduce network size (and act as implicit regularization), without requiring any further training or significantly hurting performance. DIVNET is fast and runs in time negligible compared to the network's prior training time. Moreover, it is agnostic to other network parameters such as activation functions, number of hidden layers, and learning rates.

For simplicity, we describe and analyze DIVNET for feed-forward neural networks, however DIVNET is not limited to this setting. Indeed, as DIVNET operates on a layer fully connected to the following one in the network's hierarchy, it applies equally well to other architectures with fully connected layers. For example, it can be applied without any further modification to Deep Belief Nets and to the fully-connected layers in Convolutional Neural Networks. As these layers are typically responsible for the large majority of the CNNs' memory footprint [45], DIVNET is particularly well suited to such types of networks.

3.2 Related work

Due to their large number of parameters, deep neural networks typically have a heavy memory footprint. Moreover, in many deep neural network models, parameters show a significant amount of redundancy [11]. There has consequently been significant interest in developing techniques for reducing a network's size without penalizing its performance.

A common approach to reducing the number of parameters is to remove connections between layers. In [32, 18], connections are deleted using information drawn from the Hessian of the network's error function. Sainath et al. [41] reduce the number of parameters by analyzing the weight matrices, and applying low-rank factorization to the final weight layer. Han et al. [17] remove connections with weights smaller than a given threshold before retraining the network. These methods focus on deleting parameters whose removal influences the network the least, while DIVNET seeks

diversity and merges similar neurons; these methods can thus be used in conjunction with ours for further size reduction.

Although methods such as [32] that remove connections between layers may also delete neurons from the network by removing all of their outgoing or incoming connections, it is likely that the overall impact on the size of the network will be lesser than approaches such as DivNet that remove entire neurons: indeed, removing a neuron decreases the number of rows or columns of the weight matrices connecting the neuron's layer to the previous and following layers.

Convolutional Neural Networks [33] replace fully-connected layers with convolution and subsampling layers, which significantly decreases the number of parameters. However, as CNNs still maintain fully-connected layers, they may also benefit from using DivNet.

The procedure in [19] is closer to our own approach of reducing the network's memory footprint by directly removing hidden neurons: in a given layer, each neuron's importance is evaluated according to a predefined importance function, and neurons with the smaller importance scores are deleted from the network.

In [44], a neuron is pruned when its weights are similar to those of another neuron in the same layer. This leads the authors to formulate a reweighting procedure that is somewhat similar in idea (albeit significantly simpler) to the reweighting step we describe in section 3.3.2: where they consider removing neurons with equal or very similar weights, we consider the more complicated task of merging neurons that as a group perform redundant calculations based on their activation vectors.

Other recent approaches consider network compression without pruning: in [20], a new, smaller network is trained on the outputs of the large network; Chen et al. [9] use hashing to reduce the size of the weight matrices by forcing all connections within the same hash bucket to have the same weight. Courbariaux et al. [10] and Gupta et al. [16] show that a network can be trained and run using limited precision values to store its parameters, thus reducing the overall memory footprint.

We emphasize that DivNet's focus on neuronal diversity is orthogonal and complementary to prior network compression techniques. Consequently, DivNet can be

combined, in most cases trivially, with previous approaches to reduce the memory footprint of neural networks.

3.3 Diversity and redundancy reduction

In this section we introduce our technique for modeling neuronal diversity more formally.

Let $\mathcal{T}$ denote the training data, ℓ a layer of n_ℓ neurons, a_{ij} the activation of the i-th neuron on input t_j, and $v_i = (a_{i1}, \ldots, a_{in_\ell})^\top$ the activation vector of the i-th neuron obtained by feeding the training data through the network. To enforce diversity in layer ℓ, we must determine which neurons are computing redundant information and remove them. Doing so requires finding a maximal subset of (linearly) independent activation vectors in a layer and retaining only the corresponding neurons.

In practice, however, the number of items in the training set (or the number of batches) can be much larger than the number of neurons in a layer, so the activation vectors $v_1, \ldots, v_{n_\ell}$ are likely linearly independent. Merely selecting by the maximal subset may thus lead to a trivial solution that selects all neurons.

Reducing redundancy therefore requires a more careful approach to sampling. We propose to select a subset of neurons whose activation patterns are diverse while contributing to the network's overall computation. We achieve this diverse selection by formulating the neuron selection task as sampling from a DPP. We describe the details of this approach below.

3.3.1 Neuronal diversity via Determinantal Point Processes

There are numerous potential choices for the DPP kernel. We found that experimentally a well-tuned Gaussian RBF kernel provides a good balance between simplicity and quality: for instance, it provides much better results that simple linear kernels (obtained via the outer product of the activation vectors) and is easier to use than more complex Gaussian RBF kernels with additional parameters.

Recall that layer ℓ has activations $v_1, \ldots, v_{n_\ell}$. Using these, we first create an $n_\ell \times n_\ell$ kernel L' with bandwidth parameter β by setting

$$L'_{ij} = \exp(-\beta \|v_i - v_j\|^2) \qquad 1 \leq i, j \leq n_\ell. \tag{3.3.1}$$

To ensure strict positive definiteness of the kernel matrix L', we add a small diagonal perturbation εI to L' ($\varepsilon = 0.01$). The choice of the bandwidth parameter could be done by cross-validation, but that would greatly increase the training cost. Therefore, we use the fixed choice $\beta = 10/|\mathcal{T}|$, which was experimentally seen to work well.

Finally, in order to limit rounding errors, we introduce a final scaling operation: suppose we wish to obtain a desired size, say k, of sampled subsets (using a k-DPP). To that end, we can scale the kernel $L' + \varepsilon I$ by a factor γ, so that its *expected* sample size becomes k. For a DPP with kernel L, the expected sample size is given by [29, Eq. 34]:

$$\mathbb{E}[|Y|] = \mathrm{Tr}(L(I + L)^{-1}).$$

Therefore, we scale the kernel to $\gamma(L' + \varepsilon I)$ with γ such that

$$\gamma = \frac{k}{n_\ell - k} \cdot \frac{n_\ell - k'}{k'},$$

where k' is the expected sample size for the kernel $L' + \varepsilon I$.

Finally, generating and then sampling from $L = \gamma(L' + \varepsilon I)$ has $\mathcal{O}(n_\ell^3 + n_\ell^2|\mathcal{T}|)$ cost. In our experiments, this sampling cost was negligible compared to the cost of training the network. For networks with very large hidden layers, one can avoiding the n_ℓ^3 cost by using more scalable sampling techniques [34, 22].

3.3.2 Fusing redundant neurons

Simply excising the neurons that are not sampled by the DPP drastically alters the neuron inputs to the next layer. Intuitively, since activations of neurons marked redundant are not arbitrary, throwing them away is wasteful. Ideally we should preserve the total information of a given layer, which suggests that we should "fuse"

the information from unselected neurons into the selected ones. We achieve this via a reweighting procedure as outlined below.

Without loss of generality, let neurons 1 through k be the ones sampled by the DPP and $v_1, \ldots, v_k$ be their corresponding activation vectors. Let w_{ij} be the weights connecting the i-th neuron ($1 \leq i \leq k$) in the current layer to the j-th neuron in the next layer; let $\widetilde{w}_{ij} = \delta_{ij} + w_{ij}$ denote the updated weights after merging the contributions from the removed neurons.

We seek to minimize the impact of removing $n_\ell - k$ neurons from layer ℓ. To that end, we minimize the difference in inputs to neurons in the subsequent layer before ($\sum_{i \leq n_\ell} w_{ij} v_i$) and after ($\sum_{i=1 \leq k} \widetilde{w}_{ij} v_i$) DPP pruning.

That is, we wish to solve for all neurons in the next layer (indexed by j, with $1 \leq j \leq n_{\ell+1}$):

$$\min_{\widetilde{w}_{ij} \in \{R\}} \left\| \sum_{i=1}^{k} \widetilde{w}_{ij} v_i - \sum_{i=1}^{n_\ell} w_{ij} v_i \right\|_2 = \min_{\delta_{ij} \in \{R\}} \left\| \sum_{i=1}^{k} \delta_{ij} v_i - \sum_{i=k+1}^{n_\ell} w_{ij} v_i \right\|_2 \tag{3.3.2}$$

Eq. 3.3.2 is minimized when $\sum_{i \leq k} \delta_{ij} v_i$ is the projection of $\sum_{i>k} w_{ij} v_i$ onto the linear space generated by $\{v_1, \ldots, v_k\}$. Thus, to minimize Eq. 3.3.2, we obtain the coefficients α_{ij} that for $j > k$ minimize

$$\left\| v_j - \sum_{i=1}^{k} \alpha_{ij} v_i \right\|_2$$

and then update the weights by setting

$$\forall i,\, 1 \leq i \leq k,\, \widetilde{w}_{ij} = w_{ij} + \sum_{r=k+1}^{n_\ell} \alpha_{ir} w_{rj} \tag{3.3.3}$$

Using ordinary least squares to obtain α, the reweighting procedure has a complexity of $\mathcal{O}(|\mathcal{T}| n_\ell^2 + n_\ell^3)$.

3.4 Experimental results

To quantify the performance of our algorithm, we present below the results of experiments[1] on common datasets for neural network evaluation: `MNIST` [31], `MNIST_ROT` [30] and `CIFAR-10` [25].

All networks were trained up until a certain training error threshold, using softmax activation on the output layer and sigmoids on other layers; see Table 3.1 for more details. In all following plots, error bars represent standard deviations.

Table 3.1: Overview of the sets of networks used in the experiments. We train each class of networks until the first iteration of backprop for which the training error reaches a predefined threshold.

Dataset	Instances	Trained up until	Architecture
`MNIST`	5	$< 1\%$ error	784 - 500 - 500 - 10
`MNIST_ROT`	5	$< 1\%$ error	784 - 500 - 500 - 10
`CIFAR-10`	5	$< 50\%$ error	3072 - 1000 - 1000 - 1000 - 10

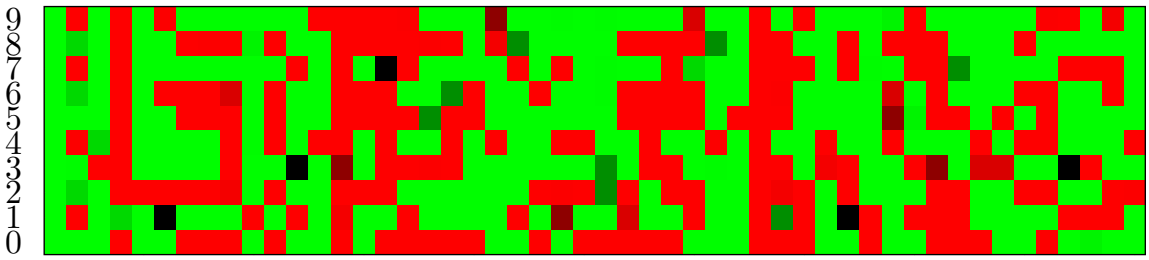
3.4.1 Pruning and reweighting analysis

To validate our claims on the benefits of using DPPs and fusing neurons, we compare these steps separately and also simultaneously against random pruning, where a fixed number of neurons are chosen uniformly at random from a layer and then removed, with and without our fusing step. We present performance results on the test data; of course, both DPP selection and reweighting are based solely on information drawn from the training data.

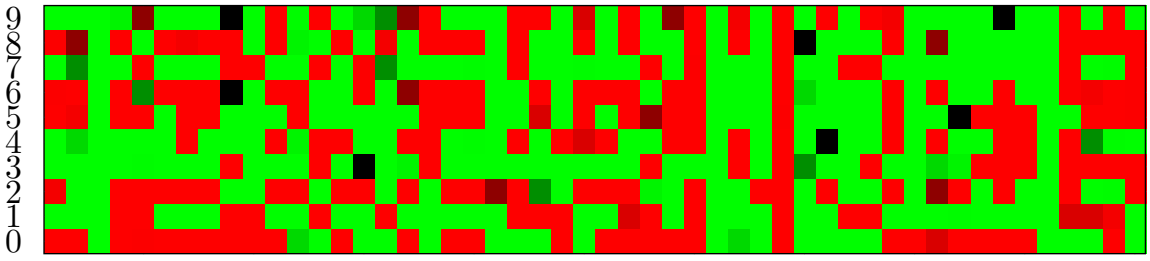
Figure 3-1 visualizes neuron activations in the first hidden layer of a network trained on the `MNIST` dataset. Each column in the plotted heat maps represents the activation of a neuron on instances of digits 0 through 9. Figure 3-1a shows the activations of the 50 neurons sampled using a k-DPP ($k = 50$) defined over the first hidden layer, whereas Figure 3-1b shows the activations of the first 50 neurons of the same layer. Figure 3-1b contains multiple similar columns: for example, there are 3

[1] Run in MATLAB, based on the code from DeepLearnToolBox (`https://github.com/rasmusbergpalm/DeepLearnToolbox`) and Alex Kulesza's code for DPPs (`http://web.eecs.umich.edu/~kulesza/`), on a Linux Mint system with 16GB of RAM and an i7-4710HQ CPU @ 2.50GHz.

entirely green columns, corresponding to three neurons that saturate to 1 on each of the 10 instances. In contrast, the DPP samples neurons with diverse activations, and Figure 3-1a shows no similar redundancy.



(a) 50 neurons sampled via DPP from the first hidden layer



(b) First 50 neurons of the first hidden layer

Figure 3-1: Heat map of the activation of subsets of 50 neurons for one instance of each class of the `MNIST` dataset. The rows correspond to digits 0 through 9. Each column corresponds to the activation values of one neuron in the network's first layer on images of digits 0 through 9. Green values are activations close to 1, red values are activations close to 0.

Figures 3-2 through 3-7 illustrate the impact of each step of DivNet separately (Figures 3-2 through 3-4 show pruning on the first hidden layer, Figures 3-5 through 3-7 on the second hidden layer). Figures 3-2 and 3-5 show the impact of pruning on test error using DPP pruning and random pruning (in which a fixed number of neurons are selected uniformly at random and removed from the network). DPP-pruned networks have consistently better training and test errors than networks pruned at random for the same final size. As expected, the more neurons are maintained, the less the error suffers from the pruning procedure; however, the pruning is in both cases destructive, and is seen to significantly increase the error rate.

This phenomenon can be mitigated by our reweighting procedure, as shown in

Figures 3-3 and 3-6. By fusing and reweighting neurons after pruning, the training and test errors are considerably reduced, even when 90% of the layer's neurons are removed. Pruning also reduces the variability of the results: the standard deviation for the results of the reweighted networks is significantly smaller than for the non-reweighted networks, and may be thus seen as a way to regularize neural networks.

Finally, Figures 3-4 and 3-7 illustrate the performance of DIVNET (DPP pruning and reweighting) compared to random pruning with reweighting. Although DIVNET's performance is ultimately better, the reweighting procedure also dramatically benefits the networks that were pruned randomly.

Notably, we found that the gap between DIVNET and random pruning's performances was much wider when pruning the last layer. We believe this is due to the connections to the output layer being learned much faster, thus letting a small, diverse subset of neurons (hence well suited to DPP sampling) in the last hidden layer take over the majority of the computational effort.

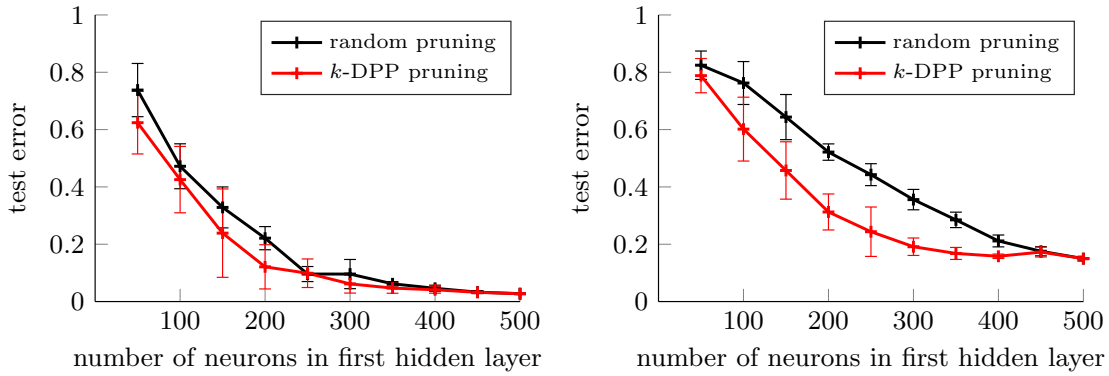
3.4.2 Performance analysis

Much attention has been given to reducing the size of neural networks in order to reduce memory consumption. When using neural nets locally on devices with limited memory, it is crucial that their memory footprint be as small as possible.

Node importance-based pruning (henceforth "importance pruning") is one of the most intuitive ways to cut down on network size. Introduced to deep networks by [19], this method removes the neurons whose calculations impact the network the least. Among the three solutions to estimating a neuron's importance discussed in [19], the sum the output weights of each neuron (the 'onorm' function) provided the best results:

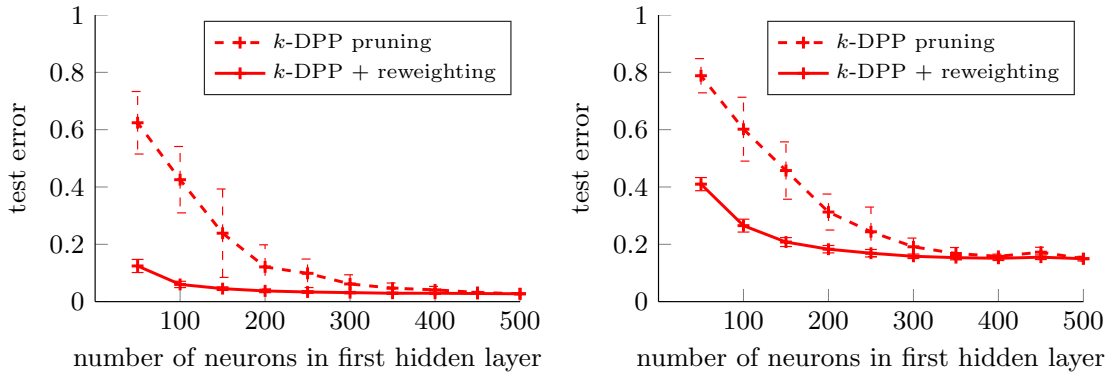
$$\text{onorm}(n_i) := \frac{1}{n_{\ell+1}} \sum\nolimits_{j=1}^{n_\ell} |w_{ij}^{\ell+1}|.$$

Figures 3-8 and 3-9 compare the test data error of the networks after being pruned using importance pruning that uses onorm as a measure of relevance against DIVNET.



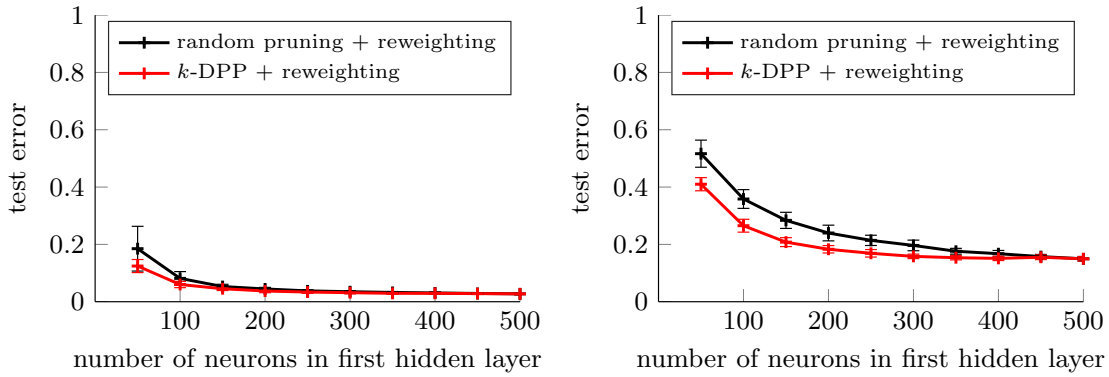
(a) `MNIST` dataset (b) `MNIST_ROT` dataset

Figure 3-2: Comparison of random and k-DPP pruning procedures.



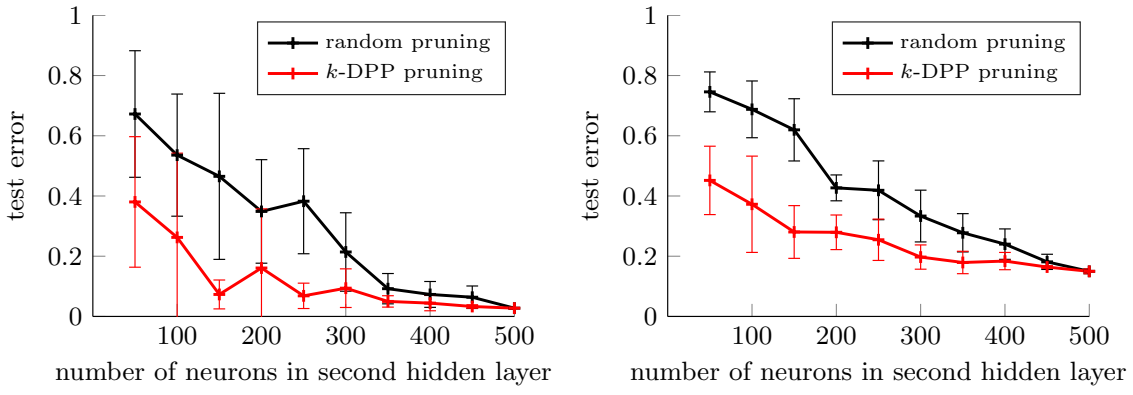
(a) `MNIST` dataset (b) `MNIST_ROT` dataset

Figure 3-3: Comparison of DivNet (k-DPP + reweighting) to simple k-DPP pruning.



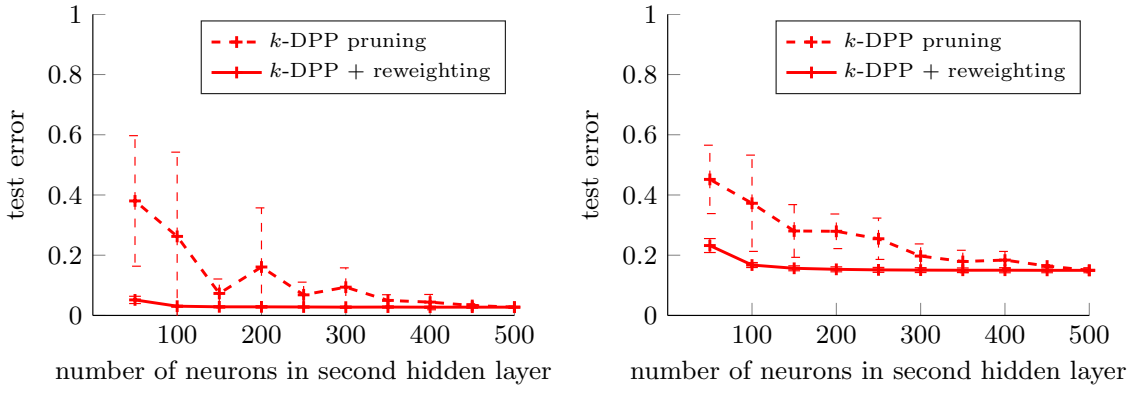
(a) `MNIST` dataset (b) `MNIST_ROT` dataset

Figure 3-4: Comparison of random and k-DPP pruning of the first hidden layer when both are followed by reweighting.



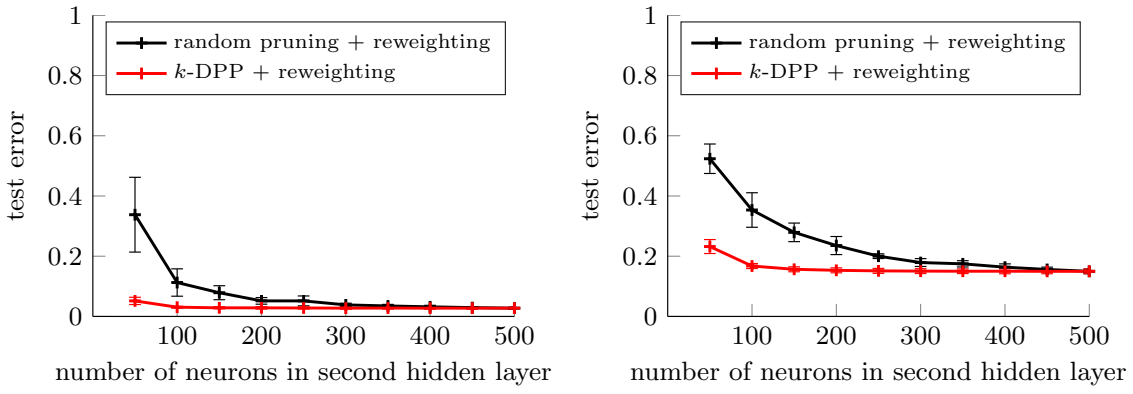
(a) `MNIST` dataset (b) `MNIST_ROT` dataset

Figure 3-5: Comparison of random and k-DPP pruning procedures.



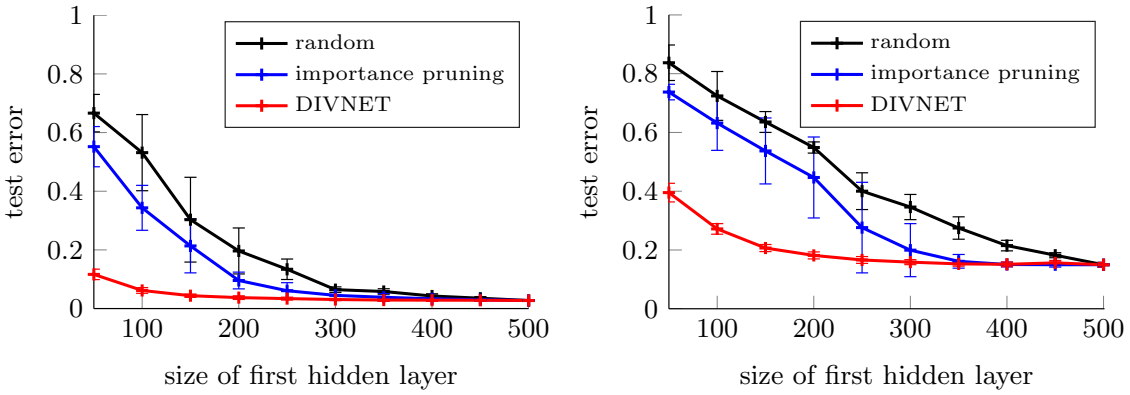
(a) `MNIST` dataset (b) `MNIST_ROT` dataset

Figure 3-6: Comparison of DIVNET to simple k-DPP pruning.



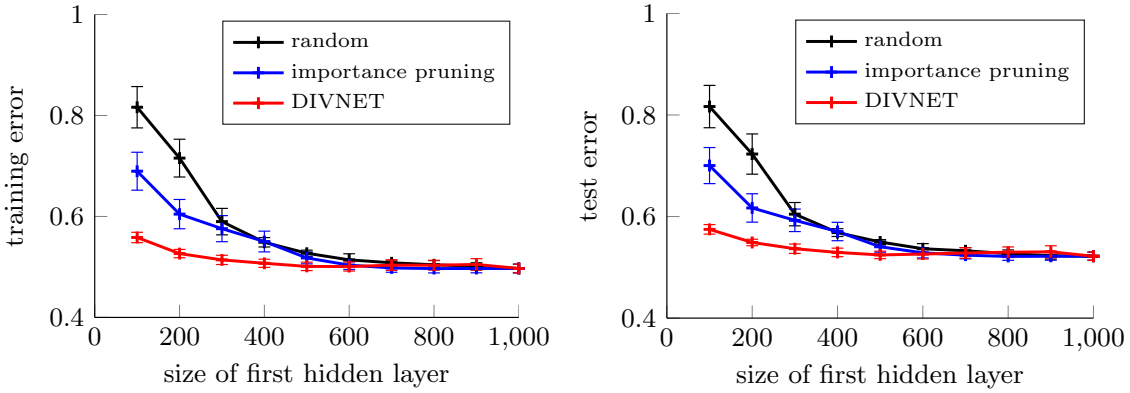
(a) `MNIST` dataset (b) `MNIST_ROT` dataset

Figure 3-7: Comparison of random and k-DPP pruning when both are followed by reweighting.



(a) `MNIST` dataset (b) `MNIST_ROT` dataset

Figure 3-8: Comparison of random pruning, importance pruning, and DIVNET's impact on the network's performance after decreasing the number of neurons in the first hidden layer.



(a) Training error on `CIFAR-10` dataset (b) Test error on `CIFAR-10` dataset

Figure 3-9: Comparison of random pruning, importance pruning, and DIVNET's impact on the network's performance after decreasing the number of parameters in the network.

Since importance pruning deletes the neurons that contribute the least to the next layer's computations, it performs well up to a certain point; however, when pruning a significant amount of neurons, this pruning procedure even removes neurons performing essential calculations, hurting the network's performance significantly.

However, since DIVNET fuses redundant neurons instead of merely deleting them, its resulting network delivers a much better performance even when used with large amounts of pruning.

Table 3.2 shows network training and test errors under various compression rates,

without additional retraining (that is, the pruned network is not retrained to further optimize its weights).

Table 3.2: Training and test error for different percentages of remaining neurons (mean ± standard deviation). Initially, `MNIST` and `MNIST_ROT` nets have 1000 hidden neurons, and `CIFAR-10` have 3000.

Remaining hidden neurons		10%	25%	50%	75%
`MNIST`	training error	0.76 ± 0.06	0.28 ± 0.12	0.15 ± 0.04	0.06 ± 0.04
	test error	0.76 ± 0.07	0.29 ± 0.12	0.17 ± 0.05	0.07 ± 0.03
`MNIST_ROT`	training error	0.74 ± 0.08	0.54 ± 0.09	0.34 ± 0.06	0.20 ± 0.03
	test error	0.73 ± 0.09	0.49 ± 0.11	0.25 ± 0.07	0.06 ± 0.03
`CIFAR-10`	training error	0.84 ± 0.05	0.61 ± 0.01	0.52 ± 0.01	0.50 ± 0.01
	test error	0.85 ± 0.05	0.62 ± 0.02	0.54 ± 0.01	0.52 ± 0.01

3.4.3 Influence of the bandwidth on the pruning procedure

Figure 3-10 shows how varying the RBF kernel's bandwidth parameter β influences the training error of a network after pruning; Figure 3-11 illustrates how β influences the sampled size of a subset when the size of the final subset isn't controlled by using a k-DPP. When $\beta >> 1$, the kernel L approaches the identity matrix, in which case in expectation $n/2$ neurons are chosen, uniformly at random.

3.4.4 Discussion and Remarks

- In all experiments, sampling and reweighting ran several orders of magnitude faster than training; reweighting required significantly more time than sampling. If DIVNET must be further sped up, a fraction of the training set can be used instead of the entire set, at the possible cost of subsequent network performance.
- When using DPPs with a Gaussian kernel, sampled neurons need not have linearly independent activation vectors: not only is the DPP sampling probabilistic, the Gaussian kernel itself is not scale invariant. Indeed, for two collinear but unequal activation vectors, the corresponding coefficient in the kernel will not be 1 (or γ with the $L \leftarrow \gamma L$ update).

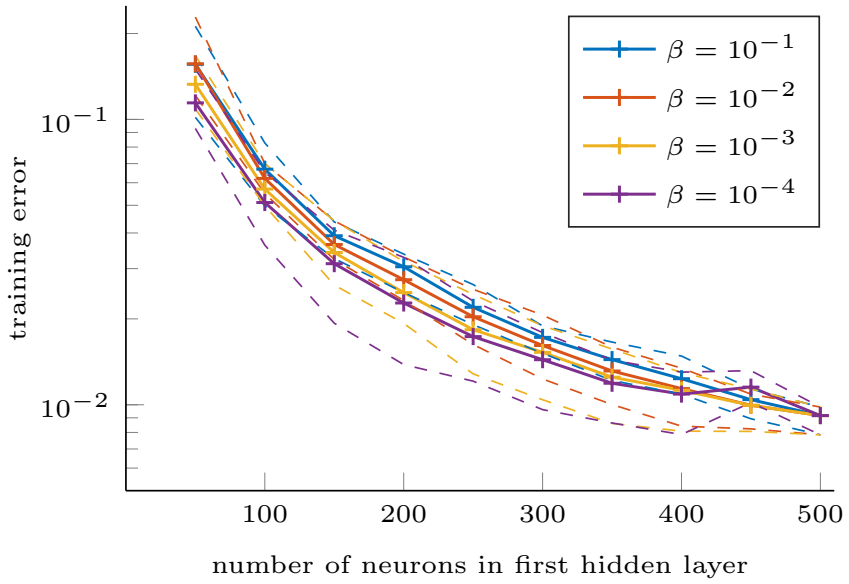


Figure 3-10: Influence of β on training error (using the networks trained on `MNIST`). The dotted lines show min and max errors.

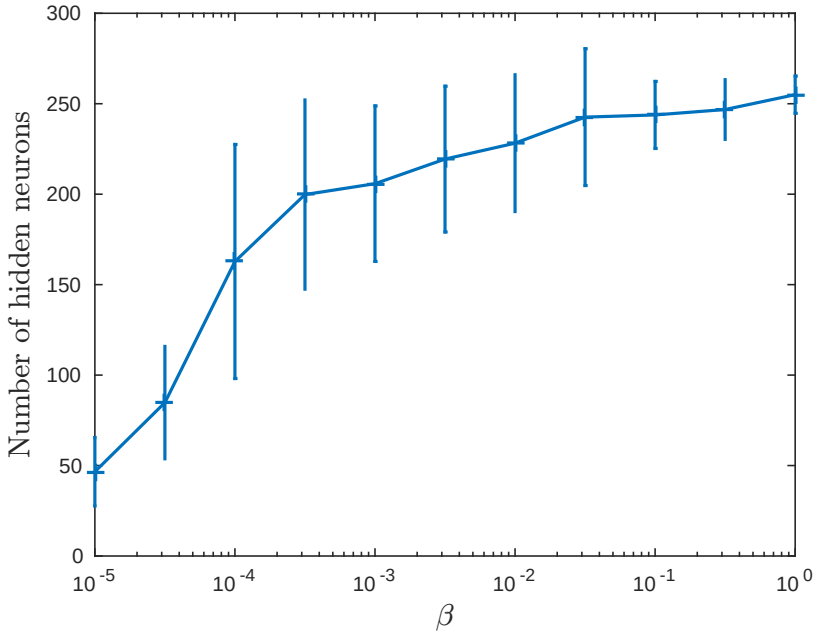


Figure 3-11: Influence of β on the number of neurons that remain after pruning networks trained on `MNIST` (when pruning non-parametrically, using a DPP instead of a k-DPP.)

– In our work, we selected a subset of neurons by sampling once from the DPP. Alternatively, one could sample a fixed amount of times, using the subset with the highest likelihood (i.e., largest $\det(L_Y)$), or greedily approximate the mode of the

DPP distribution.

- Our reweighting procedure benefits competing pruning methods as well (see Figure 3-4).
- We also investigated DPP sampling for pruning concurrently with training iterations, hoping that this might allow us to detect superfluous neurons before convergence, and thus reduce training time. However, we observed that in this case DPP pruning, with or without reweighting, did not offer a significant advantage over random pruning.
- Consistently over all datasets and networks, the expected sample size from the kernel L' was much smaller for the last hidden layer than for other layers. We hypothesize that this is caused by the connections to the output layer being learned faster than the others, allowing a small subset of neurons to take over the majority of the computational effort.

Chapter 4

Conclusion

In a world where large amounts of data are available and where machine-learning tools are increasingly memory-heavy, leveraging methods for enforcing diversity – both in the data and in the machine-learning structures themselves – is a necessary step to obtaining efficient and precise approaches to machine-learning.

By allowing a careful balancing of quality and diversity considerations, Determinantal Point Processes provide a powerful way to approach these issues, both theoretically and on real-world data.

In Chapter 2, we approached the problem of maximum-likelihood estimation of a DPP kernel from a new angle: we analyzed the stationarity properties of the cost function and used them to obtain a novel fixed-point Picard iteration. Experiments on both simulated and real data showed that for a range of ground set sizes and number of samples, the Picard iteration runs remarkably faster than the previous best approach, while being extremely simple to implement. In particular, for large ground set sizes our experiments show that our algorithm cuts down runtime to a fraction of the previously optimal EM runtimes.

We presented a theoretical analysis of the convergence properties of the Picard iteration, and found sufficient conditions for its convergence. However, our experiments reveal that the Picard iteration converges for a wider range of step-sizes (parameter a in the iteration and plots) than currently accessible to our theoretical analysis. It is a part of our future work to develop a more complete convergence theory, especially

because of its strong empirical performance. In light of our results, another line of future work is to apply fixed-point analysis to other DPP learning tasks.

Chapter 3 introduces DIVNET, a DPP-based algorithm which leverages similarities between the behaviors of neurons in a layer to detect redundant parameters and merge them, thereby enforcing neuronal diversity within each hidden layer. Using DIVNET, large, redundant networks can be shrunk to much smaller structures without impacting their performance and without requiring further training.

Many hyper-parameters can be tuned by a user as per need: the number of remaining neurons per layer can be fixed manually; the precision of the reweighting and the sampling procedure can be tuned by choosing how many training instances are used to generate the DPP kernel and the reweighting coefficients, creating a trade-off between accuracy, memory management, and computational time.

Although DIVNET requires the user to select the size of the final network, we believe that a method where no parameter explicitly needs to be tuned is worth investigating. The fact that DPPs can be augmented to also reflect different distributions over the sampled set sizes [29, §5.1.1] might be leveraged to remove the burden of choosing the layer's size from the user.

Importantly, DIVNET is agnostic to most parameters of the network, as it only requires knowledge of the activation vectors. Consequently, DIVNET can be easily used jointly with other pruning and memory management methods to reduce size. Moreover, the reweighting procedure is agnostic to how the pruning is done, as shown in our experiments.

Furthermore, the principles behind DIVNET can theoretically also be leveraged non fully-connected settings. For example, the same diversifying approach may also be applicable to filters in CNNs: if a layer of the CNN is connected to a simple, feed-forward layer – such as the S4 layer in [33] – by viewing each filter's activation values as a vector and applying DIVNET on the resulting activation matrix, one may be able to remove entire filters from the network, thus significantly reducing a CNN's memory footprint.

Finally, we believe that investigating DPP pruning with different kernels, includ-

ing learning the kernel from data, may provide insight into which interactions between neurons of a layer contain the information necessary for obtaining good representations and accurate classification. This marks an interesting line of future investigation, both for training and representation learning.

Bibliography

[1] P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009.

[2] R. Affandi, A. Kulesza, E. Fox, and B. Taskar. Nyström approximation for large-scale Determinantal Point Processes. In *Artificial Intelligence and Statistics (AISTATS)*, 2013.

[3] R. Affandi, E. Fox, R. Adams, and B. Taskar. Learning the parameters of Determinantal Point Process kernels. In *International Conference on Machine Learning*, 2014.

[4] R. Affandi, E. Fox, and B. Taskar. Approximate inference in continuous Determinantal Point Processes. In *Uncertainty in Artificial Intelligence (UAI)*, 2103.

[5] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, second edition, 1999.

[6] R. Bhatia. *Positive Definite Matrices*. Princeton University Press, 2007.

[7] A. Borodin and E. M. Rains. Eynard Mehta Theorem, Schur Process, and their Pfaffian Analogs. *Journal of Statistical Physics*, 121:291–317, Nov. 2005. doi: 10.1007/s10955-005-7583-z.

[8] N. Boumal, B. Mishra, P.-A. Absil, and R. Sepulchre. Manopt, a Matlab toolbox for optimization on manifolds. *Journal of Machine Learning Research*, 15:1455–1459, 2014. URL http://www.manopt.org.

[9] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. *CoRR*, abs/1504.04788, 2015.

[10] M. Courbariaux, Y. Bengio, and J. David. Low precision arithmetic for deep learning. *CoRR*, abs/1412.7024, 2014.

[11] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas. Predicting parameters in deep learning. *CoRR*, abs/1306.0543, 2013.

[12] J. Gillenwater, A. Kulesza, and B. Taskar. Discovering diverse and salient threads in document collections. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 710–720, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[13] J. Gillenwater, A. Kulesza, and B. Taskar. Near-optimal MAP inference for Determinantal Point Processes. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[14] J. Gillenwater, A. Kulesza, E. Fox, and B. Taskar. Expectation-Maximization for learning Determinantal Point Processes. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[15] A. Granas and J. Dugundji. *Fixed-point theory*. Springer, 2003.

[16] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. *CoRR*, abs/1502.02551, 2015.

[17] S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626, 2015.

[18] B. Hassibi, D. G. Stork, and S. C. R. Com. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems 5*, pages 164–171. Morgan Kaufmann, 1993.

[19] T. He, Y. Fan, Y. Qian, T. Tan, and K. Yu. Reshaping deep neural network for fast decoding by node-pruning. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 245–249, May 2014.

[20] G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[21] J. B. Hough, M. Krishnapur, Y. Peres, and B. Virág. Determinantal processes and independence. *Probability Surveys*, 3(206–229):9, 2006.

[22] B. Kang. Fast determinantal point process sampling with application to clustering. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2319–2327. Curran Associates, Inc., 2013.

[23] C.-W. Ko, J. Lee, and M. Queyranne. An exact algorithm for maximum entropy sampling. CORE Discussion Papers 1993006, Université catholique de Louvain, Center for Operations Research and Econometrics (CORE), 1993.

[24] A. Krause, A. Singh, and C. Guestrin. Near-optimal sensor placements in Gaussian processes: theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research (JMLR)*, 9:235–284, 2008.

[25] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[26] A. Kulesza. *Learning with Determinantal Point Processes*. PhD thesis, University of Pennsylvania, 2013.

[27] A. Kulesza and B. Taskar. k-DPPs: Fixed-size Determinantal Point Processes. In *International Conference on Maachine Learning (ICML)*, 2011.

[28] A. Kulesza and B. Taskar. Learning Determinantal Point Processes. In *Uncertainty in Artificial Intelligence (UAI)*, 2011.

[29] A. Kulesza and B. Taskar. *Determinantal Point Processes for machine learning*, volume 5. Foundations and Trends in Machine Learning, 2012.

[30] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 473–480, 2007.

[31] Y. LeCun and C. Cortes. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, 2010. URL http://yann.lecun.com/exdb/mnist/.

[32] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, pages 598–605. Morgan Kaufmann, 1990.

[33] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[34] C. Li, S. Jegelka, and S. Sra. Efficient sampling for k-determinantal point processes. *preprint*, 2015.

[35] H. Lin and J. Bilmes. Learning mixtures of submodular shells with application to document summarization. In *Uncertainty in Artificial Intelligence (UAI)*, 2012.

[36] R. Lyons. Determinantal probability measures. *Publications Mathématiques de l'Institut des Hautes Études Scientifiques*, 98(1):167–212, 2003.

[37] O. Macchi. The coincidence approach to stochastic point processes. *Advances in Applied Probability*, 7(1), 1975.

[38] D. Maclaurin, D. Duvenaud, and R. P. Adams. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the 32nd International Conference on Machine Learning*, July 2015.

[39] Z. Mariet and S. Sra. Fixed-point algorithms for learning determinantal point processes. In F. R. Bach and D. M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Proceedings*, pages 2389–2397. JMLR.org, 2015.

[40] Z. Mariet and S. Sra. Diversity networks. *CoRR*, abs/1511.05077, 2015. URL http://arxiv.org/abs/1511.05077.

[41] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 6655–6659. IEEE, 2013.

[42] J. Snoek, R. Zemel, and R. P. Adams. A determinantal point process latent variable model for inhibition in neural spiking data. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 1932–1940. Curran Associates, Inc., 2013.

[43] S. Sra and R. Hosseini. Conic geometric optimization on the manifold of positive definite matrices. *SIAM Journal on Optimization*, 25(1):713–739, 2015.

[44] S. Srinivas and R. V. Babu. Data-free parameter pruning for deep neural networks. *CoRR*, abs/1507.06149, 2015. URL http://arxiv.org/abs/1507.06149.

[45] Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. J. Smola, L. Song, and Z. Wang. Deep fried convnets. *CoRR*, abs/1412.7149, 2014.

[46] A. L. Yuille and A. Rangarajan. The concave-convex procedure. *Neural Comput.*, 15(4):915–936, Apr. 2003.

[47] T. Zhou, Z. Kuscsik, J.-G. Liu, M. Medo, J. R. Wakeling, and Y.-C. Zhang. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515, 2010.

[48] A. Çivril and M. Magdon-Ismail. On selecting a maximum volume sub-matrix of a matrix and related problems. *Theoretical Computer Science*, 410(4749):4801 – 4811, 2009.