Kronecker Determinantal Point Processes

Zelda Mariet
Massachusetts Institute of Technology
Cambridge, MA 02139
zelda@csail.mit.edu

Suvrit Sra
Massachusetts Institute of Technology
Cambridge, MA 02139
suvrit@mit.edu

Abstract

Determinantal Point Processes (DPPs) are probabilistic models over all subsets a ground set of N items. They have recently gained prominence in several applications that rely on "diverse" subsets. However, their applicability to large problems is still limited due to $\mathcal{O}(N^3)$ complexity of core tasks such as sampling and learning. We enable efficient sampling and learning for DPPs by introducing KRONDPP, a DPP model whose kernel matrix decomposes as a tensor product of multiple smaller kernel matrices. This decomposition immediately enables fast *exact* sampling. But contrary to what one may expect, leveraging the Kronecker product structure for speeding up DPP learning turns out to be more difficult. We overcome this challenge, and derive batch and stochastic optimization algorithms for efficiently learning the parameters of a KRONDPP.

1 Introduction

Determinantal Point Processes (DPPs) are discrete probability models over the subsets of a ground set of N items. They provide an elegant model to assign probabilities to an exponentially large sample, while permitting tractable (polynomial time) sampling and marginalization. They are often used to provide models that balance "diversity" and quality, characteristics valuable to numerous problems in machine learning and related areas [17].

The antecedents of DPPs lie in statistical mechanics [24], but since the seminal work of [15] they have made inroads into machine learning. By now they have been applied to a variety of problems such as document and video summarization [6, 21], sensor placement [14], recommender systems [31], and object retrieval [2]. More recently, they have been used to compress fully-connected layers in neural networks [26] and to provide optimal sampling procedures for the Nyström method [20]. The more general study of DPP properties has also garnered a significant amount of interest, see e.g., [1, 5, 7, 12, 16–18, 23].

However, despite their elegance and tractability, widespread adoption of DPPs is impeded by the $\mathcal{O}(N^3)$ cost of basic tasks such as (exact) sampling [12, 17] and learning [10, 12, 17, 25]. This cost has motivated a string of recent works on approximate sampling methods such as MCMC samplers [13, 20] or core-set based samplers [19]. The task of learning a DPP from data has received less attention; the methods of [10, 25] cost $\mathcal{O}(N^3)$ per iteration, which is clearly unacceptable for realistic settings. This burden is partially ameliorated in [9], who restrict to learning low-rank DPPs, though at the expense of being unable to sample subsets larger than the chosen rank.

These considerations motivate us to introduce KRONDPP, a DPP model that uses Kronecker (tensor) product kernels. As a result, KRONDPP enables us to learn large sized DPP kernels, while also permitting efficient (exact and approximate) sampling. The use of Kronecker products to scale matrix models is a popular and effective idea in several machine-learning settings [8, 27, 28, 30]. But as we will see, its efficient execution for DPPs turns out to be surprisingly challenging.

To make our discussion more concrete, we recall some basic facts now. Suppose we have a ground set of N items $\mathcal{Y} = \{1, \dots, N\}$. A discrete DPP over $\mathcal{Y}$ is a probability measure $\mathcal{P}$ on $2^{\mathcal{Y}}$

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

parametrized by a positive definite matrix K (the *marginal kernel*) such that $0 \preceq K \preceq I$, so that for any $Y \in 2^{\mathcal{Y}}$ drawn from $\mathcal{P}$, the measure satisfies

$$\forall A \subseteq \mathcal{Y}, \qquad \mathcal{P}(A \subseteq Y) = \det(K_A), \tag{1}$$

where K_A is the submatrix of K indexed by elements in A (i.e., $K_A = [K_{ij}]_{i,j \in A}$). If a DPP with marginal kernel K assigns nonzero probability to the empty set, the DPP can alternatively be parametrized by a positive definite matrix L (the *DPP kernel*) so that

$$\mathcal{P}(Y) \propto \det(L_Y) \quad \Longrightarrow \quad \mathcal{P}(Y) = \frac{\det(L_Y)}{\det(L+I)}. \tag{2}$$

A brief manipulation (see e.g., [17, Eq. 15]) shows that when the inverse exists, $L = K(I-K)^{-1}$. The determinants, such as in the normalization constant in (2), make operations over DPPs typically cost $\mathcal{O}(N^3)$, which is a key impediment to their scalability.

Therefore, if we consider a class of DPP kernels whose structure makes it easy to compute determinants, we should be able to scale up DPPs. An alternative approach towards scalability is to restrict the size of the subsets, as done in k-DPP [16] or when using rank-k DPP kernels [9] (where $k \ll N$). Without further assumptions, both approaches still require $\mathcal{O}(N^3)$ preprocessing for exact sampling; another caveat is that they limit the DPP model by assigning zero probabilities to sets of cardinality greater than k.

In contrast, KRONDPP uses a kernel matrix of the form $L = L_1 \otimes \ldots \otimes L_m$, where each *sub-kernel* L_i is a smaller positive definite matrix. This decomposition has two key advantages: (i) it significantly lowers the number of parameters required to specify the DPP from N^2 to $\mathcal{O}(N^{2/m})$ (assuming the sub-kernels are roughly the same size); and (ii) it enables fast sampling and learning.

For ease of exposition, we describe specific details of KRONDPP for $m = 2$; as will become clear from the analysis, typically the special cases $m = 2$ and $m = 3$ should suffice to obtain low-complexity sampling and learning algorithms.

Contributions. Our main contribution is the KRONDPP model along with efficient algorithms for sampling from it and learning a Kronecker factored kernel. Specifically, inspired by the algorithm of [25], we develop KRK-PICARD (**Kr**onecker-**K**ernel Picard), a block-coordinate ascent procedure that generates a sequence of Kronecker factored estimates of the DPP kernel while ensuring monotonic progress on its (difficult, nonconvex) objective function. More importantly, we show how to implement KRK-PICARD to run in $\mathcal{O}(N^2)$ time when implemented as a batch method, and in $\mathcal{O}(N^{3/2})$ time and $\mathcal{O}(N)$ space, when implemented as a stochastic method. As alluded to above, unlike many other uses of Kronecker models, KRONDPP does not admit trivial scaling up, largely due to extensive dependence of DPPs on arbitrary submatrices of the DPP kernel. An interesting theoretical nugget that arises from our analysis is the combinatorial problem that we call *subset clustering*, a problem whose (even approximate) solution can lead to further speedups of our algorithms.

2 Preliminaries

We begin by recalling basic properties of Kronecker products needed in our analysis; we omit proofs of these well-known results for brevity. The Kronecker (tensor) product of $A \in \mathbb{R}^{p \times q}$ with $B \in \mathbb{R}^{r \times s}$ two matrices is defined as the $pr \times qs$ block matrix $A \otimes B = [a_{ij}B]_{i,j=1}^{p,q}$.

We denote the block $a_{ij}B$ in $A \otimes B$ by $(A \otimes B)_{(ij)}$ for any valid pair (i, j), and extend the notation to non-Kronecker product matrices to indicate the submatrix of size $r \times s$ at position (i, j).

Proposition 2.1. *Let A, B, C, D be matrices of sizes so that AC and BD are well-defined. Then,*

(i) If $A, B \succeq 0$, then, $A \otimes B \succeq 0$;

(ii) If A and B are invertible then so is $A \otimes B$, with $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$;

(iii) $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$.

An important consequence of Prop. 2.1(iii) is the following corollary.

Corollary 2.2. *Let $A = P_A D_A P_A^\top$ and $B = P_B D_B P_B^\top$ be the eigenvector decompositions of A and B. Then, $A \otimes B$ diagonalizes as $(P_A \otimes P_B)(D_A \otimes D_B)(P_A \otimes P_B)^\top$.*

We will also need the notion of partial trace operators, which are perhaps less well-known:

Definition 2.3. Let $A \in \mathbb{R}^{N_1N_2 \times N_1N_2}$. The *partial traces* $\mathrm{Tr}_1(A)$ and $\mathrm{Tr}_2(A)$ are defined as follows:

$$\mathrm{Tr}_1(A) := \left[\mathrm{Tr}(A_{(ij)})\right]_{1 \le i,j \le N_1} \in \mathbb{R}^{N_1 \times N_1}, \qquad \mathrm{Tr}_2(A) := \sum\nolimits_{i=1}^{N_1} A_{(ii)} \in \mathbb{R}^{N_2 \times N_2}.$$

The action of partial traces is easy to visualize: indeed, $\mathrm{Tr}_1(A \otimes B) = \mathrm{Tr}(B)A$ and $\mathrm{Tr}_2(A \otimes B) = \mathrm{Tr}(A)B$. For us, the most important property of partial trace operators is their positivity.

Proposition 2.4. Tr_1 *and* Tr_2 *are positive operators, i.e., for* $A \succ 0$, $\mathrm{Tr}_1(A) \succ 0$ *and* $\mathrm{Tr}_2(A) \succ 0$.

Proof. Please refer to [4, Chap. 4]. □

3 Learning the kernel matrix for KRONDPP

In this section, we consider the key difficult task for KRONDPPs: learning a Kronecker product kernel matrix from n observed subsets $Y_1, \ldots, Y_n$. Using the definition (2) of $\mathcal{P}(Y_i)$, maximum-likelihood learning of a DPP with kernel L results in the optimization problem:

$$\arg\max_{L \succ 0} \quad \phi(L), \qquad \phi(L) = \frac{1}{n}\sum_{i=1}^{n} \left(\log\det(L_{Y_i}) - \log\det(L+I)\right). \tag{3}$$

This problem is nonconvex and conjectured to be NP-hard [15, Conjecture 4.1]. Moreover the constraint $L \succ 0$ is nontrivial to handle. Writing U_i as the indicator matrix for Y_i of size $N \times |Y_i|$ so that $L_{Y_i} = U_i^\top L U_i$, the gradient of ϕ is easily seen to be

$$\Delta := \nabla\phi(L) = \frac{1}{n}\sum\nolimits_{i=1}^{n} U_i L_{Y_i}^{-1} U_i^\top - (L+I)^{-1}. \tag{4}$$

In [25], the authors derived an iterative method ("the Picard iteration") for computing an L that solves $\Delta = 0$ by running the simple iteration

$$L \leftarrow L + L\Delta L. \tag{5}$$

Moreover, iteration (5) is guaranteed to monotonically increase the log-likelihood ϕ [25]. But these benefits accrue at a cost of $\mathcal{O}(N^3)$ per iteration, and furthermore a direct application of (5) cannot guarantee the Kronecker structure required by KRONDPP.

3.1 Optimization algorithm

Our aim is to obtain an efficient algorithm to (locally) optimize (3). Beyond its nonconvexity, the Kronecker structure $L = L_1 \otimes L_2$ imposes another constraint. As in [25] we first rewrite ϕ as a function of $S = L^{-1}$, and re-arrange terms to write it as

$$\phi(S) = \underbrace{\log\det(S)}_{f(S)} + \underbrace{\frac{1}{n}\sum\nolimits_{i=1}^{n} \log\det\left(U_i^\top S^{-1} U_i\right) - \log\det(I+S)}_{g(S)}. \tag{6}$$

It is easy to see that f is concave, while a short argument shows that g is convex [25]. An appeal to the convex-concave procedure [29] then shows that updating S by solving $\nabla f(S^{(k+1)}) + \nabla g(S^{(k)}) = 0$, which is what (5) does [25, Thm. 2.2], is guaranteed to monotonically increase ϕ.

But for KRONDPP this idea does not apply so easily: due the constraint $L = L_1 \otimes L_2$ the function

$$g_\otimes : (S_1, S_2) \to \tfrac{1}{n}\sum\nolimits_{i=1}^{n} \log\det\left(U_i^\top (S_1 \otimes S_2)^{-1} U_i\right) - \log\det(I + S_1 \otimes S_2),$$

fails to be convex, precluding an easy generalization. Nevertheless, for fixed S_1 or S_2 the functions

$$\begin{cases} f_1 : S_1 \mapsto f(S_1 \otimes S_2) \\ g_1 : S_1 \mapsto g(S_1 \otimes S_2) \end{cases}, \qquad \begin{cases} f_2 : S_2 \to f(S_1 \otimes S_2) \\ g_2 : S_2 \to g(S_1 \otimes S_2) \end{cases}$$

are once again concave or convex. Indeed, the map $\otimes : S_1 \to S_1 \otimes S_2$ is linear and f is concave, and $f_1 = f \circ \otimes$ is also concave; similarly, f_2 is seen to be concave and g_1 and g_2 are convex. Hence, by generalizing the arguments of [29, Thm. 2] to our "block-coordinate" setting, updating via

$$\nabla f_i\left(S_i^{(k+1)}\right) = -\nabla g_i\left(S_i^{(k)}\right), \quad \text{for } i = 1, 2, \tag{7}$$

should increase the log-likelihood ϕ at each iteration. We prove below that this is indeed the case, and that updating as per (7) ensure positive definiteness of the iterates as well as monotonic ascent.

3.1.1 Positive definite iterates and ascent

In order to show the positive definiteness of the solutions to (7), we first derive their closed form.

Proposition 3.1 (Positive definite iterates)**.** *For $S_1 \succ 0$, $S_2 \succ 0$, the solutions to (7) are given by the following expressions:*

$$\nabla f_1(X) = -\nabla g_1(S_1) \iff X^{-1} = \operatorname{Tr}_1((I \otimes S_2)(L + L\Delta L))/N_2$$
$$\nabla f_2(X) = -\nabla g_2(S_2) \iff X^{-1} = \operatorname{Tr}_2((S_1 \otimes I)(L + L\Delta L))/N_1.$$

Moreover, these solutions are positive definite.

Proof. The details are somewhat technical, and are hence given in Appendix A. We know that $L \succ 0 \implies L + L\Delta L \geq 0$, because $L - L(I+L)^{-1}L \succ 0$. Since the partial trace operators are positive (Prop. 2.4), it follows that the solutions to (7) are also positive definite. □

We are now ready to establish that these updates ensure monotonic ascent in the log-likelihood.

Theorem 3.2 (Ascent)**.** *Starting with $L_1^{(0)} \succ 0$, $L_2^{(0)} \succ 0$, updating according to (7) generates positive definite iterates $L_1^{(k)}$ and $L_2^{(k)}$, and the sequence $\left\{\phi\big(L_1^{(k)} \otimes L_2^{(k)}\big)\right\}_{k\geq 0}$ is non-decreasing.*

Proof. Updating according to (7) generates positive definite matrices S_i, and hence positive definite subkernels $L_i = S_i$. Moreover, due to the convexity of g_1 and concavity of f_1, for matrices $A, B \succ 0$

$$f_1(B) \leq f_1(A) + \nabla f_1(A)^\top (B - A),$$
$$g_1(A) \geq g_1(B) + \nabla g_1(B)^\top (A - B).$$

Hence, $f_1(A) + g_1(A) \geq f_1(B) + g_1(B) + (\nabla f_1(A) + \nabla g_1(B))^\top (A - B)$.

Thus, if $S_1^{(k)}, S_1^{(k+1)}$ verify (7), by setting $A = S_1^{(k+1)}$ and $B = S_1^{(k)}$ we have

$$\phi\big(L_1^{(k+1)} \otimes L_2^{(k)}\big) = f_1\big(S_1^{(k+1)}\big) + g_1\big(S_1^{(k+1)}\big) \geq f_1\big(S_1^{(k)}\big) + g_1\big(S_1^{(k)}\big) = \phi\big(L_1^{(k)} \otimes L_2^{(k)}\big).$$

The same reasoning holds for L_2, which proves the theorem. □

As $\operatorname{Tr}_1((I \otimes S_2)L) = N_2 L_1$ (and similarly for L_2), updating as in (7) is equivalent to updating

$$L_1 \leftarrow L_1 + \operatorname{Tr}_1\big((I \otimes L_2^{-1})(L\Delta L)\big)/N_2, \qquad L_2 \leftarrow L_2 + \operatorname{Tr}_2\big((L_1^{-1} \otimes I)(L\Delta L)\big)/N_1.$$

Generalization. We can generalize the updates to take an additional step-size parameter a:

$$L_1 \leftarrow L_1 + a\operatorname{Tr}_1\big((I \otimes L_2^{-1})(L\Delta L)\big)/N_2, \qquad L_2 \leftarrow L_2 + a\operatorname{Tr}_2\big((L_1^{-1} \otimes I)(L\Delta L)\big)/N_1.$$

Experimentally, $a > 1$ (as long as the updates remain positive definite) can provide faster convergence, although the monotonicity of the log-likelihood is no longer guaranteed. We found experimentally that the range of admissible a is larger than for Picard, but decreases as N grows larger.

The arguments above easily generalize to the multiblock case. Thus, when learning $L = L_1 \otimes \cdots \otimes L_m$, by writing E_{ij} the matrix with a 1 in position (i, j) and zeros elsewhere, we update L_k as

$$(L_k)_{ij} \leftarrow (L_k)_{ij} + N_k/(N_1 \dots N_m)\operatorname{Tr}\left[(L_1 \otimes \dots \otimes L_{k-1} \otimes E_{ij} \otimes L_{k+1} \otimes \dots \otimes L_m)(L\Delta L)\right].$$

From the above updates it is not transparent whether the Kronecker product saves us any computation. In particular, it is not clear whether the updates can be implemented to run faster than $\mathcal{O}(N^3)$. We show below in the next section how to implement these updates efficiently.

3.1.2 Algorithm and complexity analysis

From Theorem 3.2, we obtain Algorithm 1 (which is different from the Picard iteration in [25], because it operates alternatingly on each subkernel). It is important to note that a further speedup to Algorithm 1 can be obtained by performing stochastic updates, i.e., instead of computing the full gradient of the log-likelihood, we perform our updates using only one (or a small minibatch) subset Y_i at each step instead of iterating over the entire training set; this uses the stochastic gradient $\Delta = U_i L_{Y_i}^{-1} U_i^\top - (I + L)^{-1}$. The crucial strength of Algorithm 1 lies in the following result:

Algorithm 1 KRK-PICARD iteration

Input: Matrices L_1, L_2, training set T, parameter a.
for $i = 1$ **to** maxIter **do**
  $L_1 \leftarrow L_1 + a \operatorname{Tr}_1\big((I \otimes L_2^{-1})(L\Delta L)\big)/N_2$ // or update stochastically
  $L_2 \leftarrow L_2 + a \operatorname{Tr}_2\big((L_1^{-1} \otimes I)(L\Delta L)\big)/N_1$ // or update stochastically
end for
return (L_1, L_2)

Theorem 3.3 (Complexity). *For $N_1 \approx N_2 \approx \sqrt{N}$, the updates in Algorithm 1 can be computed in $\mathcal{O}(n\kappa^3 + N^2)$ time and $\mathcal{O}(N^2)$ space, where κ is the size of the largest training subset. Furthermore, stochastic updates can be computed in $\mathcal{O}(N\kappa^2 + N^{3/2})$ time and $\mathcal{O}(N + \kappa^2)$ space.*

Indeed, by leveraging the properties of the Kronecker product, the updates can be obtained without computing $L\Delta L$. This result is non-trivial: the components of Δ, $\frac{1}{n}\sum_i U_i L_{Y_i}^{-1} U_i^\top$ and $(I + L)^{-1}$, must be considered separately for computational efficiency. The proof is provided in App. B. However, it seems that considering more than 2 subkernels does not lead to further speed-ups.

This is a marked improvement over [25], which runs in $\mathcal{O}(N^2)$ space and $\mathcal{O}(n\kappa^3 + N^3)$ time (non-stochastic) or $\mathcal{O}(N^3)$ time (stochastic); Algorithm 1 also provides faster stochastic updates than [9][1]. However, one may wonder if by learning the sub-kernels by alternating updates the log-likelihood converges to a sub-optimal limit. The next section discusses how to jointly update L_1 and L_2.

3.2 Joint updates

We also analyzed the possibility of updating L_1 and L_2 jointly: we update $L \leftarrow L + L\Delta L$ and then recover the Kronecker structure of the kernel by defining the updates L_1' and L_2' such that:

$$\begin{cases} (L_1', L_2') \text{ minimizes } \|L + L\Delta L - L_1' \otimes L_2'\|_F^2 \\ L_1' \succ 0, L_2' \succ 0, \|L_1'\| = \|L_2'\| \end{cases} \tag{8}$$

We show in appendix C that such solutions exist and can be computed from the first singular value and vectors of the matrix $R = \left[\operatorname{vec}((L^{-1} + \Delta)_{(ij)})^\top\right]_{i,j=1}^{N_1}$. Note however that in this case, there is no guaranteed increase in log-likelihood. The pseudocode for the related algorithm (JOINT-PICARD) is given in appendix C.1. An analysis similar to the proof of Thm. 3.3 shows that the updates can be obtained $\mathcal{O}(n\kappa^3 + \max(N_1, N_2)^4)$.

3.3 Memory-time trade-off

Although KRONDPPS have tractable learning algorithms, the memory requirements remain high for non-stochastic updates, as the matrix $\Theta = \frac{1}{n}\sum_i U_i L_{Y_i}^{-1} U_i^\top$ needs to be stored, requiring $\mathcal{O}(N^2)$ memory. However, if the training set can be subdivided such that

$$\{Y_1, \dots, Y_n\} = \cup_{k=1}^m S_k \quad s.t. \quad \forall k, |\cup_{Y \in S_k} Y| < z, \tag{9}$$

Θ can be decomposed as $\frac{1}{n}\sum_{k=1}^m \Theta_k$ with $\Theta_k = \sum_{Y_i \in S_k} U_i L_{Y_i}^{-1} U_i^\top$. Due to the bound in Eq. 9, each Θ_k will be sparse, with only z^2 non-zero coefficients. We can then store each Θ_k with minimal storage and update L_1 and L_2 in $\mathcal{O}(n\kappa^3 + mz^2 + N^{3/2})$ time and $\mathcal{O}(mz^2 + N)$ space.

Determining the existence of such a partition of size m is a variant of the NP-Hard Subset-Union Knapsack Problem (SUKP) [11] with m knapsacks and where the value of each item (i.e. each Y_i) is equal to 1: a solution to SUKP of value n with m knapsacks is equivalent to a solution to Eq. 9. However, an approximate partition can also be simply constructed via a greedy algorithm.

4 Sampling

Sampling exactly (see Alg. 2 and [17]) from a full DPP kernel costs $\mathcal{O}(N^3 + Nk^3)$ where k is the size of the sampled subset. The bulk of the computation lies in the initial eigendecomposition of L;

[1] For example, computing matrix B in [9] (defined after Eq. 7), which is a necessary step for (stochastic) gradient ascent, costs $\mathcal{O}(N^2)$ due to matrix multiplications.

the k orthonormalizations cost $\mathcal{O}(Nk^3)$. Although the eigendecomposition need only happen once for many iterations of sampling, exact sampling is nonetheless intractable in practice for large N.

Algorithm 2 Sampling from a DPP kernel L

Input: Matrix L.
Eigendecompose L as $\{(\lambda_i, v_i)\}_{1 \leq i \leq N}$.
$J \leftarrow \emptyset$
for $i = 1$ **to** N **do**
  $J \rightarrow J \cup \{i\}$ with probability $\lambda_i/(\lambda_i + 1)$.
end for
$V \leftarrow \{v_i\}_{i \in J}$, $Y \leftarrow \emptyset$
while $|V| > 0$ **do**
  Sample i from $\{1 \dots N\}$ with probability $\frac{1}{|V|} \sum_{v \in V} v_i^2$
  $Y \leftarrow Y \cup \{i\}$, $V \leftarrow V_\perp$, where $V_\perp$ is an orthonormal basis of the subspace of V orthonormal to e_i
end while
return Y

It follows from Prop. 2.2 that for KRONDPPS, the eigenvalues λ_i can be obtained in $\mathcal{O}(N_1^3 + N_2^3)$, and the k eigenvectors in $\mathcal{O}(kN)$ operations. For $N_1 \approx N_2 \approx \sqrt{N}$, exact sampling thus only costs $\mathcal{O}(N^{3/2} + Nk^3)$. If $L = L_1 \otimes L_2 \otimes L_3$, the same reasoning shows that exact sampling becomes linear in N, only requiring $\mathcal{O}(Nk^3)$ operations.

One can also resort to MCMC sampling; for instance such a sampler was considered in [13] (though with an incorrect mixing time analysis). The results of [20] hold only for k-DPPs, but suggest their MCMC sampler may possibly take $\mathcal{O}(N^2 \log(N/\epsilon))$ time for full DPPs, which is impractical. Nevertheless if one develops faster MCMC samplers, they should also be able to profit from the Kronecker product structure offered by KRONDPP.

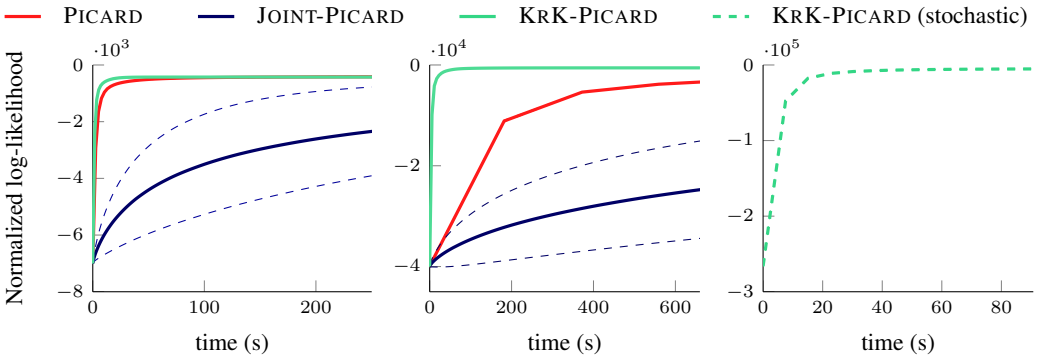
5 Experimental results

In order to validate our learning algorithm, we compared KRK-PICARD to JOINT-PICARD and to the Picard iteration (PICARD) on multiple real and synthetic datasets.[2]

5.1 Synthetic tests

To enable a fair comparison between algorithms, we test them on synthetic data drawn from a full (non-Kronecker) ground-truth DPP kernel. The sub-kernels were initialized by $L_i = X^\top X$, with X's coefficients drawn uniformly from $[0, \sqrt{2}]$; for PICARD, L was initialized with $L_1 \otimes L_2$.

For Figures 1a and 1b, training data was generated by sampling 100 subsets from the true kernel with sizes uniformly distributed between 10 and 190.



(a) $N_1 = N_2 = 50$ (b) $N_1 = N_2 = 100$ (c) $N_1 = 100, N_2 = 500$

Figure 1: $a = 1$; the thin dotted lines indicated the standard deviation from the mean.

[2] All experiments were repeated 5 times and averaged, using MATLAB on a Linux Mint system with 16GB of RAM and an i7-4710HQ CPU @ 2.50GHz.

To evaluate KRK-PICARD on matrices too large to fit in memory and with large κ, we drew samples from a $50 \cdot 10^3 \times 50 \cdot 10^3$ kernel of rank $1,000$ (on average $|Y_i| \approx 1,000$), and learned the kernel stochastically (only KRK-PICARD could be run due to the memory requirements of other methods); the likelihood drastically improves in only two steps (Fig.1c).

As shown in Figures 1a and 1b, KRK-PICARD converges significantly faster than PICARD, especially for large values of N. However, although JOINT-PICARD also increases the log-likelihood at each iteration, it converges much slower and has a high standard deviation, whereas the standard deviations for PICARD and KRK-PICARD are barely noticeable. For these reasons, we drop the comparison to JOINT-PICARD in the subsequent experiments.

5.2 Small-scale real data: baby registries

We compared KRK-PICARD to PICARD and EM [10] on the baby registry dataset (described in-depth in [10]), which has also been used to evaluate other DPP learning algorithms [9, 10, 25]. The dataset contains 17 categories of baby-related products obtained from Amazon. We learned kernels for the 6 largest categories ($N = 100$); in this case, PICARD is sufficiently efficient to be prefered to KRK-PICARD; this comparison serves only to evaluate the quality of the final kernel estimates.

The initial marginal kernel K for EM was sampled from a Wishart distribution with N degrees of freedom and an identity covariance matrix, then scaled by $1/N$; for PICARD, L was set to $K(I - K)^{-1}$; for KRK-PICARD, L_1 and L_2 were chosen (as in JOINT-PICARD) by minimizing $\|L - L_1 \otimes L_2\|$. Convergence was determined when the objective change dipped below a threshold δ. As one EM iteration takes longer than one Picard iteration but increases the likelihood more, we set $\delta_{\text{PIC}} = \delta_{\text{KRK}} = 10^{-4}$ and $\delta_{\text{EM}} = 10^{-5}$.

The final log-likelihoods are shown in Table 1; we set the step-sizes to their largest possible values, i.e. $a_{\text{PIC}} = 1.3$ and $a_{\text{KRK}} = 1.8$. Table 1 shows that KRK-PICARD obtains comparable, albeit slightly worse log-likelihoods than PICARD and EM, which confirms that for tractable N, the better modeling capability of full kernels make them preferable to KRONDPPS.

Table 1: Final log-likelihoods for each large category of the baby registries dataset

(a) Training set

Category	EM	PICARD	KRK-PICARD
apparel	-10.1	-10.2	-10.7
bath	-8.6	-8.8	-9.1
bedding	-8.7	-8.8	-9.3
diaper	-10.5	-10.7	-11.1
feeding	-12.1	-12.1	-12.5
gear	-9.3	-9.3	-9.6

(b) Test set

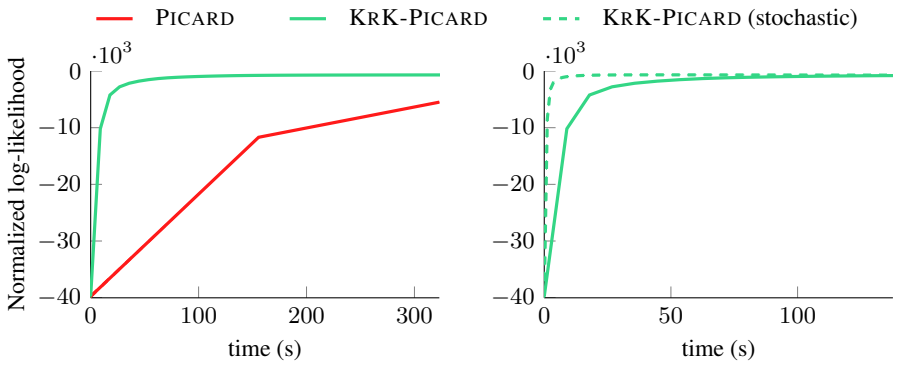
Category	EM	PICARD	KRK-PICARD
apparel	-10.1	-10.2	-10.7
bath	-8.6	-8.8	-9.1
bedding	-8.7	-8.8	-9.3
diaper	-10.6	-10.7	-11.2
feeding	-12.2	-12.2	-12.6
gear	-9.2	-9.2	-9.5

5.3 Large-scale real dataset: GENES

Finally, to evaluate KRK-PICARD on large matrices of real-world data, we train it on data from the GENES [3] dataset (which has also been used to evaluate DPPs in [3, 19]). This dataset consists in 10,000 genes, each represented by 331 features corresponding to the distance of a gene to hubs in the BioGRID gene interaction network.

We construct a ground truth Gaussian DPP kernel on the GENES dataset and use it to obtain 100 training samples with sizes uniformly distributed between 50 and 200 items. Similarly to the synthetic experiments, we initialized KRK-PICARD's kernel by setting $L_i = X_i^\top X_i$ where X_i is a random matrix of size $N_1 \times N_1$; for PICARD, we set the initial kernel $L = L_1 \otimes L_2$.

Figure 2 shows the performance of both algorithms. As with the synthetic experiments, KRK-PICARD converges much faster; stochastic updates increase its performance even more, as shown in Fig. 2b. Average runtimes and speed-up are given in Table 2: KRK-PICARD runs almost an order of magnitude faster than PICARD, and stochastic updates are more than two orders of magnitude faster, while providing slightly larger initial increases to the log-likelihood.



(a) Non-stochastic learning

(b) Stochastic vs. non-stochastic

Figure 2: $n = 150$, $a = 1$.

Table 2: Average runtime and performance on the GENES dataset for $N_1 = N_2 = 100$

	PICARD	KRK-PICARD	KRK-PICARD (stochastic)
Average runtime	161.5 ± 17.7 s	8.9 ± 0.2 s	1.2 ± 0.02 s
NLL increase (1st iter.)	$(2.81 \pm 0.03) \cdot 10^4$	$(2.96 \pm 0.02) \cdot 10^4$	$(3.13 \pm 0.04) \cdot 10^4$

6 Conclusion and future work

We introduced KRONDPPS, a variant of DPPs with kernels structured as the Kronecker product of m smaller matrices, and showed that typical operations over DPPs such as sampling and learning the kernel from data can be made efficient for KRONDPPS on previously untractable ground set sizes.

By carefully leveraging the properties of the Kronecker product, we derived for $m = 2$ a low-complexity algorithm to learn the kernel from data which guarantees positive iterates and a monotonic increase of the log-likelihood, and runs in $\mathcal{O}(n\kappa^3 + N^2)$ time. This algorithm provides even more significant speed-ups and memory gains in the stochastic case, requiring only $\mathcal{O}(N^{3/2} + N\kappa^2)$ time and $\mathcal{O}(N + \kappa^2)$ space. Experiments on synthetic and real data showed that KRONDPPS can be learned efficiently on sets large enough that L does not fit in memory.

Our experiments showed that KRONDPP's reduced number of parameters (compared to full kernels) did not impact its performance noticeably. However, a more in-depth investigation of its expressivity may be valuable for future study. Similarly, a deeper study of initialization procedures for DPP learning algorithms, including KRK-PICARD, is an important question.

While discussing learning the kernel, we showed that L_1 and L_2 cannot be updated simultaneously in a CCCP-style iteration since g is not convex over (S_1, S_2). However, it can be shown that g is geodesically convex over the Riemannian manifold of positive definite matrices, which suggests that deriving an iteration which would take advantage of the intrinsic geometry of the problem may be a viable line of future work.

KRONDPPS also enable fast sampling, in $\mathcal{O}(N^{3/2} + Nk^3)$ operations when using two sub-kernels, and in $\mathcal{O}(Nk^3)$ when using three sub-kernels. This speedup allows for exact sampling at comparable or even better costs than previous algorithms for approximate sampling. However, the subset size k is still limiting, due to the $\mathcal{O}(Nk^3)$ cost of sampling and learning. A key aspect of future work on obtaining truly scalable DPPs is to overcome this computational bottleneck.

Acknowledgements

SS acknowledges partial support from NSF grant IIS-1409802.

References

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

[2] R. Affandi, E. Fox, R. Adams, and B. Taskar. Learning the parameters of Determinantal Point Process kernels. In *ICML*, 2014.

[3] N. K. Batmanghelich, G. Quon, A. Kulesza, M. Kellis, P. Golland, and L. Bornn. Diversifying sparsity using variational determinantal point processes. *arXiv:1411.6307*, 2014.

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

[5] A. Borodin. Determinantal point processes. *arXiv:0911.1153*, 2009.

[6] W. Chao, B. Gong, K. Grauman, and F. Sha. Large-margin determinantal point processes. In *Uncertainty in Artificial Intelligence (UAI)*, 2015.

[7] L. Decreusefond, I. Flint, N. Privault, and G. L. Torrisi. Determinantal point processes, 2015.

[8] S. Flaxman, A. Wilson, D. Neill, H. Nickisch, and A. Smola. Fast Kronecker inference in Gaussian processes with non-Gaussian likelihoods. In *ICML*, pages 607–616, 2015.

[9] M. Gartrell, U. Paquet, and N. Koenigstein. Low-rank factorization of determinantal point processes for recommendation. *arXiv:1602.05436*, 2016.

[10] J. Gillenwater, A. Kulesza, E. Fox, and B. Taskar. Expectation-Maximization for learning Determinantal Point Processes. In *NIPS*, 2014.

[11] O. Goldschmidt, D. Nehme, and G. Yu. Note: On the set-union knapsack problem. *Naval Research Logistics*, 41:833–842, 1994.

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

[13] B. Kang. Fast determinantal point process sampling with application to clustering. In *Advances in Neural Information Processing Systems 26*, pages 2319–2327. Curran Associates, Inc., 2013.

[14] A. Krause, A. Singh, and C. Guestrin. Near-optimal sensor placements in Gaussian processes: theory, efficient algorithms and empirical studies. *JMLR*, 9:235–284, 2008.

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

[16] A. Kulesza and B. Taskar. k-DPPs: Fixed-size Determinantal Point Processes. In *ICML*, 2011.

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

[18] F. Lavancier, J. Møller, and E. Rubak. Determinantal point process models and statistical inference. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 77(4):853–877, 2015.

[19] C. Li, S. Jegelka, and S. Sra. Efficient sampling for k-determinantal point processes. *arXiv:1509.01618*, 2015.

[20] C. Li, S. Jegelka, and S. Sra. Fast DPP sampling for Nyström with application to kernel methods. *arXiv:1603.06052*, 2016.

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

[22] C. V. Loan and N. Pitsianis. Approximation with kronecker products. In *Linear Algebra for Large Scale and Real Time Applications*, pages 293–314. Kluwer Publications, 1993.

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

[24] O. Macchi. The coincidence approach to stochastic point processes. *Adv. Appl. Prob.*, 7(1), 1975.

[25] Z. Mariet and S. Sra. Fixed-point algorithms for learning determinantal point processes. In *ICML*, 2015.

[26] Z. Mariet and S. Sra. Diversity networks. *Int. Conf. on Learning Representations (ICLR)*, 2016. URL arXiv:1511.05077.

[27] J. Martens and R. B. Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *ICML*, 2015.

[28] G. Wu, Z. Zhang, and E. Y. Chang. Kronecker factorization for speeding up kernel machines. In *SIAM Data Mining (SDM)*, pages 611–615, 2005.

[29] A. L. Yuille and A. Rangarajan. The concave-convex procedure (cccp). In *Advances in Neural Information Processing Systems 14*, pages 1033–1040. MIT Press, 2002.

[30] X. Zhang, F. X. Yu, R. Guo, S. Kumar, S. Wang, and S.-F. Chang. Fast orthogonal projection based on kronecker product. In *ICCV*, 2015.

[31] 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. *PNAS*, 107(10):4511–4515, 2010.