The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Dynamic Determinantal Point Processes

Takayuki Osogami, Rudy Raymond
IBM Research AI

Akshay Goel
Graduate School of Mathematics
Kyushu University

Tomoyuki Shirai
Institute of Mathematics for Industry
Kyushu University

Takanori Maehara
RIKEN Center for
Advanced Intelligence Project

Abstract

The determinantal point process (DPP) has been receiving increasing attention in machine learning as a generative model of subsets consisting of relevant and diverse items. Recently, there has been a significant progress in developing efficient algorithms for learning the kernel matrix that characterizes a DPP. Here, we propose a dynamic DPP, which is a DPP whose kernel can change over time, and develop efficient learning algorithms for the dynamic DPP. In the dynamic DPP, the kernel depends on the subsets selected in the past, but we assume a particular structure in the dependency to allow efficient learning. We also assume that the kernel has a low rank and exploit a recently proposed learning algorithm for the DPP with low-rank factorization, but also show that its bottleneck computation can be reduced from $O(M^2\,K)$ time to $O(M\,K^2)$ time, where M is the number of items under consideration, and K is the rank of the kernel, which can be set smaller than M by orders of magnitude.

Introduction

A determinantal point process (DPP) defines a probability distribution over the subsets of items in a given ground set in a way that the subsets consisting of relevant and diverse items have high probabilities (Macchi 1975; Kulesza and Taskar 2012; Soshnikov 2000; Shirai and Takahashi 2003a; 2003b). The DPP is receiving increasing attention in machine learning because of its broad applicability to recommendation of products (Gillenwater et al. 2014; Gartrell, Paquet, and Koenigstein 2017), summarization of documents or videos (Gong et al. 2014), modeling neural spiking (Snoek, Zemel, and Adams 2013), and other areas where we want to select a subset of relevant and diverse items.

In some of these applications, it is important to take into account the dependency across time (Affandi, Kulesza, and Fox 2012; Gong et al. 2014; Qiao et al. 2016; Snoek, Zemel, and Adams 2013). For example, a customer who purchased a particular subset of products last week might want to purchase another particular subset of products that are otherwise unneeded this week. In this paper, we study a DPP whose distribution can change over time depending on the subsets that have been selected.

A key challenge in the applications of a DPP is in the high computational complexity that is needed for learning the probability distribution from data. Because the log-likelihood of a DPP is generally non-concave with respect to its parameters, and maximum likelihood estimation of a DPP is conjectured to be NP-hard (Kulesza 2012), the standard approach in the literature is to find a local maxima via gradient-based methods. However, even one iteration of gradient ascent can be computationally prohibitive. Specifically, a DPP can be specified by an $M \times M$ kernel matrix, where M is the size of the ground set. An iteration of gradient ascent involves basic matrix-operations such as inversion, multiplication, and determinant, and a naive approach would require $O(M^3)$ time.

Recently, Gartrell et al. (2017) have proposed an efficient learning algorithm for DPPs, where they assume that the kernel has a low rank and leverage its low-rank factorization. The per iteration complexity of their gradient-based approach is $O(T\,\kappa^3 + K\,M^2)$, where K is the rank of the kernel, T is the number of the subsets in training data, and κ is the maximum size of those subset. This significantly improves upon learning algorithms for full-rank kernels, which require $O(T\,\kappa^3 + M^3)$ (Mariet and Sra 2015) or more (Gillenwater et al. 2014).

We extend the algorithm by Gartrell et al. to the DPP whose kernel can change over time, which we refer to as a Dynamic DPP. We design the Dynamic DPP in a way that the dynamic kernel not only allows low-rank factorization but also has a particular parametrization that enables efficient learning through (stochastic) gradient-based approaches. Specifically, our dynamic kernel can be represented by a matrix that is a linear function of the statistics of previously selected subsets. The definition of the Dynamic DPP is our first contribution.

In developing a learning algorithm for the Dynamic DPP, we make three improvements to the algorithm by Gartrell et al. First, we use a dual representation of the kernel to reduce the computational complexity of a bottleneck from $O(K\,M^2)$ to $O(K^2\,M)$. Second, we replace an expression involving the inverse of a matrix that might not be convertible with a slightly simpler expression with pseudo-inverse. Third, we represent our algorithm in vector-matrix operations. When our algorithm for the Dynamic DPP is specialized for the (standard) DPP, its per step complex-

ity is $O(T\,\kappa^2\,K + K^2\,M)$. This can be further reduced to $O(T\,\kappa^3 + K^2\,M)$ by using inverse instead of pseudo-inverse or setting $K = O(\kappa)$. Our algorithm thus has smaller per step complexity than the best known algorithms (Gartrell, Paquet, and Koenigstein 2017; Mariet and Sra 2016), as we discuss further in the following. The per iteration complexity of our algorithm for the Dynamic DPP is $O(T\,(M\,D\,K + K^2\,M))$, where D a hyperparameter of the Dynamic DPP and represents the number of statistics of previously selected subsets that the dynamic kernel takes into account. However, we also propose an algorithm whose per iteration complexity is $O(T\,(M\,D\,K + D^2\,K^2) + K^2\,M)$, where we iteratively use rank-one updates to recursively compute matrix inversions. These efficient learning algorithms for the Dynamic DPP and the DPP constitute our second contribution.

Related Work

Our proposed learning algorithm extends and improves upon the learning algorithm for the low-rank DPP by Gartrell et al. (Gartrell, Paquet, and Koenigstein 2017). The low-rank DPP has been enhanced into a Bayesian low-rank DPP with a Hamiltonian Monte Carlo (HMC) approach in (Gartrell, Paquet, and Koenigstein 2016). However, the per iteration complexity of the Bayesian low-rank DPP remains $O(T\,\kappa^3 + K\,M^2)$ time, because it relies on the form of the gradient derived for the low-rank DPP. Our techniques can also be used with an HMC approach.

Another approach complementary to the low-rank DPP is a Kronecker DPP, which assumes that the kernel is a Kronecker product (Mariet and Sra 2016). The learning algorithm for the Kronecker DPP in Mariet and Sra (2016) has per iteration complexity of $O(T\,\kappa^3 + M^2)$ time. Although low-rank DPP and Kronecker DPP assume different structures, and their learning algorithms (for finding local maxima) are not directly comparable, this paper shows that the per iteration complexity of the low-rank DPP can be made smaller than the Kronecker DPP.

The prior work has studied various forms of DPPs that change over time (Affandi, Kulesza, and Fox 2012; Gong et al. 2014; Qiao et al. 2016; Snoek, Zemel, and Adams 2013). However, the prior work does not fully learn the dynamic kernel as we discuss in the following.

Affandi et al. (2012) study a Markov DPP, where the pair of the subsets selected at two consecutive steps forms a DPP. They use a heuristic method (that generally does not maximize an objective function) to learn only the quality measures of the DPP kernel, assuming that the similarity metric is fixed and known. Notice that the kernel of a DPP can be decomposed into a quality vector of size M and a similarity matrix of size $M \times M$ (Kulesza and Taskar 2012).

Gong et al. (2014) study a sequential DPP, which is similar to the Markov DPP but assumes that the ground set changes over time. The proposed learning algorithm is similar to the one for the low-rank DPP but considers a mapping from features of items to the low-rank matrix. They learn this static mapping, while the ground set is dynamic. On the other hand, we assume that the ground set is static and learn a dynamic kernel.

Qiao et al. (2016) study a time-varying DPP, where the kernel changes over time but only slightly (*i.e.*, updated by a low-rank matrix) at a time. They propose an efficient sampling method for the time-varying DPP. However, they assume that their time-varying kernel is given and do not study any learning algorithm. Our Dynamic DPP has a low-rank update and can benefit from their sampling algorithm.

Snoek et al. (2013) apply a DPP with time-varying kernel to neural spiking data. They assume that the quality measure is time-varying depending on external stimuli. They learn the time-varying kernel with a relatively naïve approach. Due to the computational complexity, their application is limited to $M = 31$ neurons.

The Dynamic DPP is auto-regressive, and its particular parametrization is motivated by the dynamic Boltzmann machine (DyBM) (Osogami and Otsuka 2015b; 2015a; Osogami 2017), which makes the parameters of the Boltzmann machine depend on historical patterns. Similar to the DyBM and vector auto regression (Lütkepohl 2005), our dynamic kernel is linear in its parameters and allows effective optimization via gradient-based approaches.

Dynamic Determinantal Point Processes

We study a determinantal point process (DPP) whose kernel can vary over time. Let $\mathbf{L}(t)$ be the kernel of the DPP at time t. We assume that the ground set stays unchanged over time and let M be the size of the ground set. Then, for each t, $\mathbf{L}(t)$ is an $M \times M$ positive semi-definite matrix indexed by the items in the ground set. The probability that a subset $\mathcal{X}$ is selected from a ground set at time t is given by

$$\mathcal{P}_t(\mathcal{X}) = \frac{\det(\mathbf{L}(t)_{\mathcal{X}})}{\det(\mathbf{L}(t) + \mathbf{I})}, \tag{1}$$

where $\mathbf{L}(t)_{\mathcal{X}}$ denotes the principal submatrix of $\mathbf{L}(t)$ indexed by the items in $\mathcal{X}$.

Similar to Gartrell et al. (2017), we use a low-rank factorization of $\mathbf{L}(t)$:

$$\mathbf{L}(t) = \mathbf{V}(t)\,\mathbf{V}(t)^\top, \tag{2}$$

where $\mathbf{V}(t)$ is an $M \times K$ matrix for $K \leq M$. Then, by Sylvester's determinant identity, we can work with the dual representation:

$$\mathcal{P}_t(\mathcal{X}) = \frac{\det(\mathbf{L}(t)_{\mathcal{X}})}{\det(\mathbf{C}(t) + \mathbf{I})}, \tag{3}$$

where $\mathbf{C}(t)$ is a $K \times K$ matrix and can be computed from $\mathbf{V}(t)$ in $O(K^2\,M)$ time:

$$\mathbf{C}(t) \equiv \mathbf{V}(t)^\top\,\mathbf{V}(t). \tag{4}$$

We define the Dynamic DPP in a way that $\mathbf{V}(t)$ can depend on the subset selected by time $t-1$ in the following specific manner. Let $\mathbf{x}(s)$ denote an indicator column-vector of length M that represents the subset selected (sampled) at time s. Specifically, the i-th element of $\mathbf{x}(s)$ is 1 iff the i-th item is selected (is in the selected subset) at time s. For each t, we then let

$$\mathbf{V}(t) = \mathbf{B} + \sum_{d=1}^{D} \mathbf{x}(t-d)\,\mathbf{w}(d)^\top, \tag{5}$$

where $\mathbf{B}$ is an $M \times K$ matrix, and $\mathbf{w}(d)$ is a column-vector of length K. The Dynamic DPP thus has parameters $\Theta \equiv (\mathbf{B}, \mathbf{w}(1), \ldots, \mathbf{w}(D))$. If no items are selected during the period from $t-D$ to $t-1$, we have $\mathbf{L}(t) = \mathbf{B}\,\mathbf{B}^\top$, which may be understood as the baseline kernel. When $D = 0$, we have $\mathbf{V}(t) = \mathbf{B}$ for any t, and the Dynamic DPP is reduced to the (static) DPP.

With our parametrization, the kernel can be represented as follows:

$$\begin{aligned}\mathbf{L}(t) = \mathbf{B}\,\mathbf{B}^\top &+ \sum_{d=1}^{D} \mathbf{B}\,\mathbf{w}(d)\,\mathbf{x}(t-d)^\top \\ &+ \sum_{d=1}^{D} \mathbf{x}(t-d)\,\mathbf{w}(d)^\top \mathbf{B}^\top \\ &+ \sum_{d=1}^{D}\sum_{d'=1}^{D} \mathbf{x}(t-d)\,\mathbf{w}(d)^\top \mathbf{w}(d')\,\mathbf{x}(t-d')^\top. \quad (6)\end{aligned}$$

To shed light on this parametrization, let $\mathbf{B}_{m,:}$ be the m-th row of $\mathbf{B}$ (*i.e.*, a row-vector of length K) for each m. Observe that the (i,j)-th element of $\mathbf{L}(t)$ is given by

$$\begin{aligned}L_{i,j}(t) = \mathbf{B}_{i,:}\,(\mathbf{B}_{j,:})^\top &+ \sum_{d=1}^{D} \mathbf{B}_{i,:}\,\mathbf{w}(d)\,x_j(t-d) \\ &+ \sum_{d=1}^{D} x_i(t-d)\,\mathbf{w}(d)^\top (\mathbf{B}_{j,:})^\top \\ &+ \sum_{d=1}^{D}\sum_{d'=1}^{D} x_i(t-d)\,\mathbf{w}(d)^\top\,\mathbf{w}(d')\,x_j(t-d'), \quad (7)\end{aligned}$$

where $x_i(t)$ denotes the i-th element of $\mathbf{x}(t)$. Notice that $L_{i,j}(t)$ represents the product of the relevance of item i, the relevance of j, and the similarity between the two items. Recall that $\mathbf{x}(t)$ is binary, and "1" represents that the corresponding item is selected at time t.

If neither i nor j are selected during the period from $t-D$ to $t-1$, we have $L_{i,j}(t) = \mathbf{B}_{i,:}\,(\mathbf{B}_{j,:})^\top$. If j is selected at $t-d$, then $L_{i,j}(t)$ is increased by $\mathbf{B}_{i,:}\,\mathbf{w}(d)$. If i is selected at $t-d$, then $L_{i,j}(t)$ is increased by $\mathbf{w}(d)^\top\,(\mathbf{B}_{j,:})^\top$. If i is selected at $t-d$ and j is selected at $t-d'$, then $L_{i,j}(t)$ is increased by $\mathbf{w}(d)^\top \mathbf{w}(d')$. Hence, i and j become more or less likely to be selected individually or together, when one or both of these items have been selected in the last D steps. The precise impact is given by the parameters $\mathbf{w}(1), \ldots, \mathbf{w}(D)$, and $\mathbf{B}$. We will learn those parameters in a way that it best explains a given sequence of subsets.

Learning Dynamic DPPs

We seek to learn Θ in a way that it maximizes the log-likelihood of given series of subsets. The training dataset can consist of a set of sequences of subsets. For notational simplicity, we assume a single sequence of subsets in this section, but extension to multiple sequences is trivial. Let $(\mathcal{X}_1, \ldots, \mathcal{X}_T)$ be the training dataset, where $\mathcal{X}_t$ is the subset selected at time t.

The log-likelihood of the training dataset is

$$f(\Theta) \equiv \log \mathcal{P}((\mathcal{X}_1, \ldots, \mathcal{X}_T)) = \sum_{t=1}^{T} \log \mathcal{P}_t(\mathcal{X}_t \mid \mathcal{X}_{\leq t-1}), \quad (8)$$

where $\mathcal{P}_t(\mathcal{X}_t \mid \mathcal{X}_{\leq t-1})$ denotes the conditional probability of selecting $\mathcal{X}_t$ at time t given that $\mathcal{X}_{\leq t-1} \equiv (\mathcal{X}_1, \ldots, \mathcal{X}_{t-1})$ are selected by time $t-1$. We define

$$f_t(\Theta) \equiv \log \mathcal{P}_t(\mathcal{X}_t \mid \mathcal{X}_{\leq t-1}), \quad (9)$$

so that $f(\Theta) = \sum_t f_t(\Theta)$ and $\nabla f(\Theta) = \sum_t \nabla f_t(\Theta)$, where the gradient is with respect to Θ, and the summation is from $t=1$ to $t=T$.

Similar to Equation (5) of Gartrell et al. (2017), we can represent $\nabla f_t(\Theta)$ as follows:

$$\begin{aligned}&\nabla f_t(\Theta) \\ &= \nabla \log\det(\mathbf{L}(t)_{\mathcal{X}_t}) - \nabla \log\det(\mathbf{C}(t) + \mathbf{I}) \quad (10) \\ &= \mathrm{tr}\Big((\mathbf{L}(t)_{\mathcal{X}_t})^{-1}\,\nabla \mathbf{L}(t)_{\mathcal{X}_t}\Big) - \mathrm{tr}\Big((\mathbf{C}(t) + \mathbf{I})^{-1}\,\nabla \mathbf{C}(t)\Big), \quad (11)\end{aligned}$$

where, for matrices $\mathbf{Y}$ and $\mathbf{Z}$, one should understand $\mathrm{tr}(\mathbf{Y}\,\nabla \mathbf{Z})$ as the matrix, indexed by Θ, whose entry is $\mathrm{tr}(\mathbf{Y}\,\frac{\partial \mathbf{Z}}{\partial \theta})$ for $\theta \in \Theta$. We will also use $\mathrm{tr}(\mathbf{Y}\,\nabla_{\mathbf{S}} \mathbf{Z})$ to denote the matrix whose entry is $\mathrm{tr}(\mathbf{Y}\,\frac{\partial \mathbf{Z}}{\partial \theta})$ for $\theta \in \mathbf{S}$ for a matrix $\mathbf{S}$ consisting of a subset of the parameters in Θ.

Unfortunately, the inverse in the first term of (11) might not exist, which was not addressed in Gartrell et al. (2017). Here, we derive an alternative expression using a pseudo-inverse. For a matrix $\mathbf{Y}$, the partial derivative of $\log\det(\mathbf{Y}\,\mathbf{Y}^\top)$ with respect to θ can be represented as

$$\begin{aligned}\frac{\partial \log\det(\mathbf{Y}\,\mathbf{Y}^\top)}{\partial \theta} &= \sum_{i,j} \frac{\partial (\mathbf{Y}^\top)_{i,j}}{\partial \theta} \frac{\partial \log\det(\mathbf{Y}\,\mathbf{Y}^\top)}{\partial (\mathbf{Y}^\top)_{i,j}} \\ &= \sum_{i,j} \frac{\partial \mathbf{Y}_{j,i}}{\partial \theta}\, 2\,(\mathbf{Y}^+)_{i,j} \quad (12) \\ &= 2\,\mathrm{tr}\Big(\frac{\partial \mathbf{Y}}{\partial \theta}\,\mathbf{Y}^+\Big), \quad (13)\end{aligned}$$

where $\mathbf{Y}^+$ denotes the pseudo-inverse of $\mathbf{Y}$. Because $\mathbf{L}(t)_{\mathcal{X}_t} = \mathbf{V}(t)_{\mathcal{X}_t,:}\,(\mathbf{V}(t)_{\mathcal{X}_t,:})^\top$, we can write (10) as

$$\begin{aligned}\nabla f_t(\Theta) = 2\,\mathrm{tr}&\Big(\nabla \mathbf{V}(t)_{\mathcal{X}_t,:}\,\big(\mathbf{V}(t)_{\mathcal{X}_t,:}\big)^+\Big) \\ &- \mathrm{tr}\Big(\big(\mathbf{C}(t) + \mathbf{I}\big)^{-1}\,\nabla \mathbf{C}(t)\Big). \quad (14)\end{aligned}$$

The following theorem shows specific forms of the gradients of $f_t(\Theta)$ with respect to each parameter of $\Theta = (\mathbf{B}, \mathbf{w}(1), \ldots, \mathbf{w}(D))$. The proof of the theorem is postponed to the appendix.

Theorem 1. *Consider a determinantal point process with a dynamic kernel* $\mathbf{L}(t)$*:*

$$\mathcal{P}_t(\mathcal{X}_t \mid \mathcal{X}_{\leq t-1}) = \frac{\det(\mathbf{L}(t)_{\mathcal{X}_t})}{\det(\mathbf{C}(t) + \mathbf{I})}, \quad (15)$$

Algorithm 1 A learning algorithm for Dynamic DPPs.

1: **input:** $\mathcal{X}_1, \ldots, \mathcal{X}_T$
2: Initialize $\mathbf{B}, \mathbf{W}$
3: **repeat**
4: **for** $t = 1$ to T **do**
5: $\mathcal{J} \leftarrow [T - t + 1, T - t + D]$
6: $\mathbf{V} \leftarrow \mathbf{B} + \mathbf{X}[:, \mathcal{J}]\, \mathbf{W}^\top$
7: $\mathbf{R} \leftarrow \mathbf{V}\, (\mathbf{V}^\top \mathbf{V} + \mathbf{I})^{-1}$
8: $\tilde{\mathbf{A}} \leftarrow \mathbf{V}[\mathcal{X}_t, :]^+$
9: $\Delta\mathbf{B}(t) \leftarrow -\mathbf{R}$
10: $\Delta\mathbf{B}(t)[\mathcal{X}_t, :] \leftarrow \Delta\mathbf{B}[\mathcal{X}_t, :] + \tilde{\mathbf{A}}^\top$
11: $\Delta\mathbf{W}(t) \leftarrow \tilde{\mathbf{A}}\, \mathbf{X}[\mathcal{X}_t, \mathcal{J}] - \mathbf{R}^\top\, \mathbf{X}[:, \mathcal{J}]$
12: **end for**
13: $\mathbf{B} \leftarrow \mathbf{B} + \eta \sum_{t=1}^{T} \Delta\mathbf{B}(t)$
14: $\mathbf{W} \leftarrow \mathbf{W} + \eta \sum_{t=1}^{T} \Delta\mathbf{W}(t)$
15: **until** a stopping condition is met
16: **return:** $\mathbf{B}, \mathbf{W}$

where $\mathbf{L}(t) = \mathbf{V}(t)\, \mathbf{V}(t)^\top$, $\mathbf{C}(t) = \mathbf{V}(t)^\top\, \mathbf{V}(t)$,

$$\mathbf{V}(t) \equiv \mathbf{B} + \sum_{d=1}^{D} \mathbf{x}(t-d)\, \mathbf{w}(d)^\top \tag{16}$$

and $\mathbf{L}(t)_{\mathcal{X}_t}$ *denotes the principal submatrix of* $\mathbf{L}(t)$ *indexed by the items in* $\mathcal{X}_t$*. Then the gradient of*

$$f_t(\Theta) \equiv \log \mathcal{P}_t(\mathcal{X}_t \mid \mathcal{X}_{\leq t-1}) \tag{17}$$

is given by

$$\nabla_{\mathbf{B}_{\mathcal{X}_t,:}} f_t(\Theta) = 2\tilde{\mathbf{A}}^\top - 2\left(\mathbf{V}(t)\, \mathbf{A}\right)_{\mathcal{X}_t} \tag{18}$$

$$\nabla_{\mathbf{B}_{\bar{\mathcal{X}}_t,:}} f_t(\Theta) = -2\left(\mathbf{V}(t)\, \mathbf{A}\right)_{\bar{\mathcal{X}}_t} \tag{19}$$

$$\nabla_{\mathbf{w}(\delta)} f_t(\Theta) = 2\, \tilde{\mathbf{A}}\, \mathbf{x}(t-\delta)_{\mathcal{X}_t} - 2\, \mathbf{A}\, \mathbf{V}(t)^\top\, \mathbf{x}(t-\delta) \tag{20}$$

for $\delta \in [1, D]$*, where* $\mathbf{B}_{\mathcal{X}_t,:}$ *denotes the submatrix consisting of rows of* $\mathbf{B}$ *indexed by the items in* $\mathcal{X}_t$*,* $\mathbf{V}(t)_{\mathcal{X}_t,:}$ *is defined analogously,* $\bar{\mathcal{X}}_t$ *denote the items not in* $\mathcal{X}_t$*,*

$$\tilde{\mathbf{A}} \equiv \left(\mathbf{V}(t)_{\mathcal{X}_t,:}\right)^+, \text{ and } \mathbf{A} \equiv \left(\mathbf{C}(t) + \mathbf{I}\right)^{-1}. \tag{21}$$

Theorem 1 leads to a gradient-based learning algorithm shown in Algorithm 1. Here, we define $\mathbf{W}$ to be the $K \times D$ matrix consisting of $\mathbf{w}(\cdot)$ and $\mathbf{X}$ to be the $M \times (T + D - 1)$ matrix consisting of $\mathbf{x}(\cdot)$, except $\mathbf{x}(T)$, in the reverse order:

$$\mathbf{W} \equiv (\mathbf{w}(1), \ldots, \mathbf{w}(D)) \tag{22}$$

$$\mathbf{X} \equiv (\mathbf{x}(T-1), \ldots, \mathbf{x}(1), \mathbf{0}, \ldots, \mathbf{0}). \tag{23}$$

The whole $\mathbf{X}$ may be prepared in the beginning, or its submatrices may be constructed when they are needed. For clarity, we use the notation such as $\mathbf{B}[\mathcal{X}_t, :]$ to denote the submatrix consisting of rows of $\mathbf{B}$ indexed by the items in $\mathcal{X}_t$.

Algorithm 1 computes the gradients given in Theorem 1 in the for-loop and applies them in Steps 13-14 to update $\mathbf{B}$ and $\mathbf{W}$. For simplicity, we omit the factor of 2, which is common in (18)-(20). Hence, η in Steps 13-14 may be understood as the learning rate multiplied by 2.

In each repetition of the for-loop, $\mathcal{J}$ defines the set of columns of $\mathbf{X}$ to be used in that repetition. Notice that

$$\mathbf{X}[:, \mathcal{J}] = (\mathbf{x}(t-1), \ldots, \mathbf{x}(t-D)). \tag{24}$$

Then $\mathbf{V}$ in Step 6 corresponds to $\mathbf{V}(t)$, and $\mathbf{R}$ in Step 7 corresponds to $\mathbf{V}(t)\, \mathbf{A}$.

The Dynamic DPP has hyperparameters K and D that needs to be set appropriately with validation. When the size of a subset $\mathcal{X}$ is greater than K, the probability $\mathcal{P}_t(\mathcal{X})$ must be zero. We therefore need to set K at least as large as the maximum size κ of the subsets in the training data. We thus assume $K \geq \kappa$. Notice that one could also work with the probability of the complement of the selected subset. Then we can set $K \geq M - \bar{\kappa}$, where $\bar{\kappa}$ is the minimum size of the selected subset (*i.e.*. $M - \bar{\kappa}$ is the maximum size of the complement of the selected subset).

Algorithm 1 has the following computational complexity per iteration of the repeat-loop:

Theorem 2. *Each iteration of Algorithm 1 takes* $O(T\, (M\, D\, K + K^2\, M))$ *time.*

Proof. Step 6 involves a multiplication of an $M \times D$ matrix and a $D \times K$ matrix, taking $O(M\, D\, K)$ time. Step 7 involves a multiplication of a $K \times M$ matrix and an $M \times K$ matrix, an inversion of a $K \times K$ matrix, and a multiplication of an $M \times K$ matrix and a $K \times K$ matrix, taking $O(K^2\, M)$ time. Step 8 involves a pseudo-inverse of a $\kappa \times K$ matrix, where κ is the size of $\mathcal{X}_t$. Step 8 thus takes $O(\kappa^2\, K)$ time. Step 11 involves a multiplication of a $K \times \kappa$ matrix and a $\kappa \times D$ matrix as well as a multiplication of a $K \times M$ matrix and an $M \times D$ matrix, taking $O(K\, M\, D)$ time. The theorem now follows because we need to set $K \in [\kappa, M]$. □

One can also consider a learning algorithm based on stochastic gradients for Dynamic DPPs. Such an algorithm has the following computational complexity per step:

Corollary 1. *A stochastic update for the Dynamic DPP takes* $O(M\, D\, K + K^2\, M)$ *time.*

When $D = 0$ in Algorithm 1, we have a learning algorithm for the (static) DPP (see Algorithm 2). Because the kernel is static, we can place Step 4 of Algorithm 2 out of the for-loop, unlike the $\mathbf{R}$ in Step 7 of Algorithm 1 that needs to be computed for each t. As a result, Algorithm 2 has the following computational complexity per iteration.

Theorem 3. *Each iteration of Algorithm 2 takes* $O(T\, \kappa^2\, K + K^2\, M)$ *time, where* κ *is the maximum size of* $\mathcal{X}_t$ *for* $t \in [1, T]$*.*

Proof. Step 4 involves a multiplication of a $K \times M$ matrix and an $M \times K$ matrix, inversion of a $K \times K$ matrix, and a multiplication of an $M \times K$ matrix and a $K \times K$ matrix, taking $O(K^2\, M)$ time. Step 6 involves a pseudo-inverse of an at most $\kappa \times K$ matrix, taking $O(\kappa^2 \times K)$ time. □

Algorithm 2 thus has a smaller per-iteration computational complexity than the $O(T\, \kappa^3 + K\, M^2)$ time algorithm for the low-rank DPP in Gartrell et al. (2017) as long as $K = O(\kappa)$. However, notice that the $O(\kappa^2\, K)$ cost stems from pseudo-inverse. The use of inverse as in Gartrell et

Algorithm 2 A learning algorithm for DPPs.

input: $\mathcal{X}_1, \ldots, \mathcal{X}_T$
Initialize $\mathbf{B}$
repeat
  $\Delta\mathbf{B} \leftarrow -T\,\mathbf{B}\,(\mathbf{B}^\top\,\mathbf{B} + \mathbf{I})^{-1}$
  for $t = 1$ to T **do**
    $\Delta\mathbf{B}[\mathcal{X}_t, :] \leftarrow \Delta\mathbf{B}[\mathcal{X}_t, :] + (\mathbf{B}[\mathcal{X}_t, :]^+)^\top$
  end for
  $\mathbf{B} \leftarrow \mathbf{B} + \eta\,\Delta\mathbf{B}$
until a stopping condition is met
return: $\mathbf{B}$

al. (2017) would reduce the computational complexity to $O(T\,\kappa^3 + K^2\,M)$. Also, observe that Algorithm 2 is fully expressed in basic matrix operations and simpler than the one in Gartrell et al. (2017).

Rank-one Updates for Faster Computation

When $D = o(K)$, the bottleneck of Algorithm 1 is on the computation of $\mathbf{A}(t) \equiv (\mathbf{I} + \mathbf{C}(t))^{-1}$, which needs to be computed for every t. This is in contrast to Algorithm 2 for DPPs, where the kernel is independent of t, and the corresponding quantity needs to be computed only once (Step 4). Recall that computation of $\mathbf{A}(t)$ requires $O(M\,K^2)$ time for computing $\mathbf{C}(t)$ and $O(K^3)$ time for inverting $(\mathbf{I} + \mathbf{C}(t))$.

In this section, we will show that we can reduce the computational cost of this bottleneck when $D = o(\sqrt{M})$ by exploiting the fact that $\mathbf{V}(t+1)$ differs from $\mathbf{V}(t)$ only by a small amount. Specifically, observe from (5) that $\mathbf{V}(t+1)$ can be computed from $\mathbf{V}(t)$ recursively:

$$\mathbf{V}(t+1) = \mathbf{V}(t) + \sum_{d=1}^{D} \Big(\mathbf{x}(t+1-d) - \mathbf{x}(t-d)\Big)\,\mathbf{w}(d)^\top. \tag{25}$$

Letting $\boldsymbol{\delta}(t,d) \equiv \mathbf{x}(t+1-d) - \mathbf{x}(t-d)$, we can then write $\mathbf{I} + \mathbf{C}(t+1)$ recursively:

$$\begin{aligned}\mathbf{I} + \mathbf{C}(t+1) = \mathbf{I} + \mathbf{C}(t) &+ \sum_{d=1}^{D} \mathbf{V}(t)^\top \boldsymbol{\delta}(t,d)\,\mathbf{w}(d)^\top \\ &+ \sum_{d=1}^{D} \mathbf{w}(d)\,\boldsymbol{\delta}(t,d)^\top\,\mathbf{V}(t) \\ &+ \sum_{d=1}^{D}\sum_{d'=1}^{D} \mathbf{w}(d)\,\boldsymbol{\delta}(t,d)^\top\,\boldsymbol{\delta}(t,d')\,\mathbf{w}(d)^\top.\end{aligned} \tag{26}$$

Namely, once we have the value of $\mathbf{A}(t)$, the value of $\mathbf{A}(t+1)$ can be obtained in $O(D^2\,K^2)$ time, avoiding computation from scratch, by combining $\mathbf{A}(t)$ with rank-one updates suggested by the following Sherman-Morrison lemma:

Lemma 1 (Sherman-Morrison)**.** *Let $\mathbf{Y}$ be an invertible $K \times K$ matrix. Let $\mathbf{u}$ and $\mathbf{v}$ be column vectors of dimension K. Then,*

$$\left(\mathbf{Y} + \mathbf{u}\,\mathbf{v}^\top\right)^{-1} = \mathbf{Y}^{-1} - \frac{\mathbf{Y}^{-1}\,\mathbf{u}\,\mathbf{v}^\top\,\mathbf{Y}^{-1}}{1 + \mathbf{v}^\top\,\mathbf{Y}^{-1}\,\mathbf{u}}.$$

Algorithm 3 Step 7 of Algorithm 1 with rank-one updates.

if $t = 1$ **then**
  $\mathbf{A}(t) \leftarrow (\mathbf{V}^\top\,\mathbf{V} + \mathbf{I})^{-1}$
else
  Compute $\mathbf{A}(t+1)$ from $\mathbf{A}(t)$ by applying the rank-one update for $D^2 + 2\,D$ times
end if
$\mathbf{R} \leftarrow \mathbf{V}\,\mathbf{A}(t)$

This lemma implies that $\mathbf{Y} + \mathbf{u}\,\mathbf{v}^\top$ can be inverted in $O(K^2)$ time if $\mathbf{Y}^{-1}$ is known.

Step 7 of Algorithm 1 can thus be modified into Algorithm 3, which leads to the following computational complexity:

Theorem 4. *Each iteration of Algorithm 1 takes $O(T\,(M\,D\,K + K^2\,D^2) + K^2\,M)$ time when Step 7 is replaced with Algorithm 3.*

A particularly attractive case from computational perspectives is when $D = 1$. Then we only need three rank-one updates to compute $\mathbf{A}(t+1)$ from $\mathbf{A}(t)$. There are, however, other cases where one can enjoy this reduced computational cost. For example, when $\mathbf{w} \equiv \mathbf{w}(1) = \mathbf{w}(2) = \ldots = \mathbf{w}(D)$, we can write

$$\mathbf{V}(t+1) = \mathbf{V}(t) + (\mathbf{x}(t) - \mathbf{x}(t-D))\,\mathbf{w}^\top. \tag{27}$$

In this case, $\mathbf{V}(t)$ depends on $\mathbf{x}(s)$ for $s < t$ only through $\sum_{d=1}^{D} \mathbf{x}(t-d)$, which represents the number of selection during the last D steps for each item. Another example is when $\mathbf{w}(d) = \lambda\,\mathbf{w}(d-1)$ for each d for some $0 < \lambda < 1$. With $\mathbf{w} = \mathbf{w}(1)$, we then have

$$\mathbf{V}(t+1) = \mathbf{V}(t) + \mathbf{y}\,\mathbf{w}^\top, \tag{28}$$

where

$$\mathbf{y} \equiv \mathbf{x}(t) - (1-\lambda)\sum_{d=1}^{D-1} \lambda^{d-1}\mathbf{x}(t-d) - \lambda^{D-1}\mathbf{x}(t-D). \tag{29}$$

In this case, the effect of the selection in previous steps diminishes by a factor of λ at every time step.

After we learn the parameters of a Dynamic DPP, we would want to compute a sequence of the probabilities given by (3) for a given sequence of subsets. This would involve computing a sequence of the determinants. In particular, the denominator of (3) is the determinant of $\mathbf{C}(t)$, which needs to be computed from the $M \times K$ matrix $\mathbf{V}(t)$:

$$\det\Big(\mathbf{I} + \mathbf{C}(t)\Big) = \det\Big(\mathbf{I} + \mathbf{V}(t)^\top\,\mathbf{V}(t)\Big). \tag{30}$$

Thus, the computational time for each step of t is $O(K^2\,M)$.

The following matrix determinant lemma, however, shows that this determinant can be updated in $O(D^2\,K^2)$ time, similar to the computation of $\mathbf{A}(t+1)$ from $\mathbf{A}(t)$.

Lemma 2 (Matrix determinant)**.** *Let $\mathbf{Y}$ be an invertible $K \times K$ matrix. Let $\mathbf{u}$ and $\mathbf{v}$ be column vectors of dimension K. Then we have*

$$\det\left(\mathbf{Y} + \mathbf{u}\mathbf{v}^\top\right) = \left(1 + \mathbf{v}^\top\mathbf{Y}^{-1}\mathbf{u}\right)\det\left(\mathbf{Y}\right). \tag{31}$$

Numerical Experiments

Here, we apply the proposed learning algorithms to four music datasets (JSB Chorales, Nottingham, Piano-midi, and MuseData), which have been used in Boulanger-Lewandowski et al. (2012). Each dataset consists of a set of sequences of chords. A chord is a subset of notes (items) that are selected from the ground set of $M = 88$ notes[1], corresponding to the keys of the piano. A sequence of the chords forms a tune from a particular category of music associated with each dataset. The dataset is divided into training, validation, and test data.

In the experiment, we first apply Algorithm 2 to learn the $\mathbf{B}$ of a DPP, which will be used as the baseline. We then apply Algorithm 1 to learn the $\mathbf{B}$ and $\mathbf{W}$ of a Dynamic DPP, where we use the $\mathbf{B}$ trained for the DPP as the initial values of $\mathbf{B}$. In Algorithm 1 and Algorithm 2, we choose the step size η via a relatively simple approach of backtracking line search, loosely following Armijo (1966)[2]. We stop the iteration when no step size significantly increases the objective function or when parameters are updated 100 times. Because the training data consists of multiple sequences, the for-loop in each of the algorithms is now over all of the steps in all of the sequences in the training data.

Figure 1 shows the log-likelihood given by the trained models on test data. The horizontal axis represents the lag D in Equation (5), and $D = 0$ corresponds to the DPP (baseline). Each curve in the figure shows the results with a particular value of rank K as indicated in the legend, where K_0 is the maximum number of notes that constitute a chord (*i.e.*, maximum size of selected subsets) in training or validation data for each dataset. For JSB Chorales, the choice of $K = K_0 = 4$ results in low log-likelihood of below -10.0 for any D under consideration and is not shown in the figure ($K = 4\,K_0$ is shown instead). Note that JSB Chorales has a particularly small value of $K_0 = 4$, compared to the other datasets.

We can observe that the Dynamic DPP ($D > 0$) consistently outperforms the baseline for any K and D, which suggests the soundness of our learning algorithm for the Dynamic DPP. Particularly significant improvement is observed when we increase $D = 0$ to $D = 1$. Increasing K from $K = K_0$ does not significantly increase log-likelihood. This means that the low-rank assumption can speed up learning without significantly sacrificing the quality of learning. In fact, we find that estimated kernels tend to have K_0 dominating singular values.

We implemented our learning algorithms with Python and measured the computational time on a machine running Ubuntu 16.04 with 4.0 GHz Intel Core i7-6700K CPU and 48 GB Memory. For example, for the JSB Chorales dataset, the average time per iteration for learning $\mathbf{B}$ and $\mathbf{W}$ of a Dynamic DPP with $D = 3$ (from Line 4 to Line 14 of Algorithm 1) varies from 2.3 seconds for $K = K_0 = 4$ to 2.8 seconds for $K = 3\,K_0 = 12$. The corresponding average time per iteration for learning $\mathbf{B}$ of a DPP (from Line 4 to Line 8 of Algorithm 2) varies from 1.3 seconds for $K = K_0$ to 1.6 seconds for $K = 3\,K_0$.

[1] We ignore the notes that appear neither in training data nor in validation data. The effective size of the ground set is thus reduced from $M = 88$ to $M = 51$ for JSB Chorales, $M = 57$ for Nottingham, and $M = 78$ for MuseData (Piano-midi uses all $M = 88$ notes). This preprocess reduces the computational complexity, but we find qualitatively similar results without the preprocess.

[2] Specifically, we search the largest step size η that increases the objective value from the candidate step sizes that can be represented as $\eta = 10^{-n}$ for a nonnegative integer $n \in [n_{\min}, 9]$, where $n_{\min} = 3$. When the largest step size is used to update parameters, we decrease $n_{\min}$ by one to allow larger step sizes.

Conclusion

We have proposed the Dynamic DPP, where the kernel of a DPP can change over time in an auto-regressive manner. The particular structure of the Dynamic DPP allows learning via (stochastic) gradient-based methods. A potential bottleneck of such a learning algorithm is in inverting the dynamic kernel at every step. We have shown, however, that the use of dual representation and rank-one updates can substantially reduce the computational complexity.

In this paper, we have primarily provided theoretical support on the proposed approach. Although we have also demonstrated soundness and validity of the proposed algorithms through numerical experiments, we have not yet investigated their full capabilities. Important future work is in advancing practical aspects of the proposed approach and in customizing it for particular applications.

Appendix: Proof of Theorem 1

We prove the theorem by establishing the following two lemmas, which respectively derive computationally convenient form of the two terms of (14). These two lemmas immediately imply Theorem 1.

Lemma 3. *Let*

$$\mathbf{V}(t)_{\mathcal{X}_t,:} \equiv \mathbf{B}_{\mathcal{X}_t,:} + \sum_{d=1}^{D} \mathbf{x}(t-d)_{\mathcal{X}_t}\, \mathbf{w}(d)^\top, \tag{32}$$

and let $\bar{\mathcal{X}}_t$ denote the items not in $\mathcal{X}_t$. Then we have

$$\mathrm{tr}\Big(\nabla_{\mathbf{B}_{\mathcal{X}_t,:}} \mathbf{V}(t)_{\mathcal{X}_t,:}\, (\mathbf{V}(t)_{\mathcal{X}_t,:})^+\Big) = ((\mathbf{V}(t)_{\mathcal{X}_t,:})^+)^\top \tag{33}$$

$$\mathrm{tr}\Big(\nabla_{\mathbf{B}_{\bar{\mathcal{X}}_t,:}} \mathbf{V}(t)_{\mathcal{X}_t,:}\, (\mathbf{V}(t)_{\mathcal{X}_t,:})^+\Big) = \mathbf{0} \tag{34}$$

$$\mathrm{tr}\Big(\nabla_{\mathbf{w}(\delta)} \mathbf{V}(t)_{\mathcal{X}_t,:}\, (\mathbf{V}(t)_{\mathcal{X}_t,:})^+\Big) = (\mathbf{V}(t)_{\mathcal{X}_t,:})^+\, \mathbf{x}(t-\delta)_{\mathcal{X}_t} \tag{35}$$

for $\delta \in [1, D]$.

Lemma 4. *Let $\mathbf{C}(t) = \mathbf{V}(t)^\top \mathbf{V}(t)$, where*

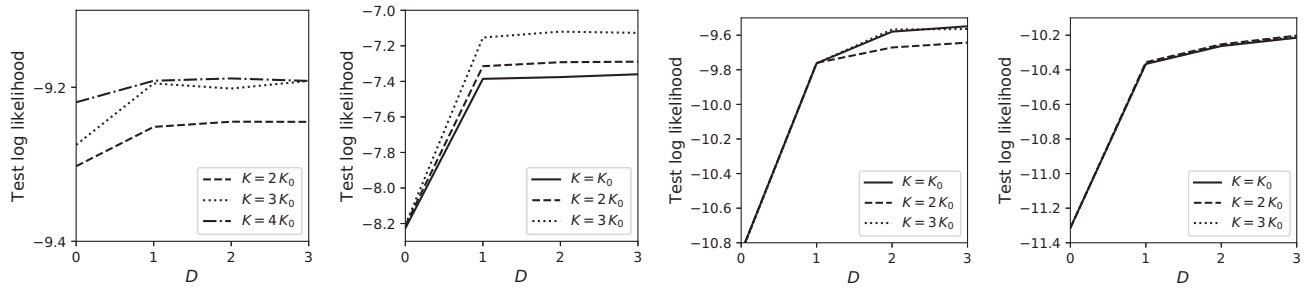
$$\mathbf{V}(t) \equiv \mathbf{B} + \sum_{d=1}^{D} \mathbf{x}(t-d)\, \mathbf{w}(d)^\top, \tag{36}$$

and let $\mathbf{A} \equiv \big(\mathbf{C}(t) + \mathbf{I}\big)^{-1}$, where $\mathbf{I}$ is the $K \times K$ identity matrix. Then we have

$$\mathrm{tr}\Big(\big(\mathbf{C}(t) + \mathbf{I}\big)^{-1} \nabla_{\mathbf{B}} \mathbf{C}(t)\Big) = 2\, \mathbf{V}(t)\, \mathbf{A} \tag{37}$$

$$\mathrm{tr}\Big(\big(\mathbf{C}(t) + \mathbf{I}\big)^{-1} \nabla_{\mathbf{w}(\delta)} \mathbf{C}(t)\Big) = 2\, \mathbf{A}\, \mathbf{V}(t)^\top \mathbf{x}(t-\delta) \tag{38}$$

for $\delta \in [1, D]$.



(a) JSB Chorales ($K_0 = 4$) (b) Nottingham ($K_0 = 9$) (c) Piano-midi ($K_0 = 12$) (d) MuseData ($K_0 = 15$)

Figure 1: Log-likelihood given by trained Dynamic DPPs on test data for the four music datasets. The rank K is varied across panels. The hyper-parameter D is varied along the horizontal axis, and $D = 0$ corresponds to the DPPs (baseline). The rank K is set as indicated in the legend, where K_0 is the maximum number of notes that constitute a chord in training or validation data for each dataset.

Proof of Lemma 3

Let $\tilde{\mathbf{B}}$ denote the $|\mathcal{X}_t| \times K$ submatrix of $\mathbf{B}$, where each row of $\tilde{\mathbf{B}}$ corresponds to the row of $\mathbf{B}$ indexed by an item in $\mathcal{X}_t$. Let $\tilde{\mathbf{x}}(s)$ denote the subvector of $\mathbf{x}(s)$, where each element of $\tilde{\mathbf{x}}(s)$ corresponds to the element of $\mathbf{x}(s)$ indexed by an item in $\mathcal{X}_t$. Namely,

$$\tilde{\mathbf{B}} \equiv \mathbf{B}_{\mathcal{X}_t,:} \tag{39}$$

$$\tilde{\mathbf{x}}(s) \equiv \mathbf{x}(s)_{\mathcal{X}_t}. \tag{40}$$

Then we have

$$\mathbf{V}(t)_{\mathcal{X}_t,:} = \tilde{\mathbf{B}} + \sum_d \tilde{\mathbf{x}}(t-d)\,\mathbf{w}(d)^\top. \tag{41}$$

It is easy to see that

$$\nabla_{\tilde{\mathbf{B}}_{i,j}} \mathbf{V}(t)_{\mathcal{X}_t,:} = \mathbf{e}_{i,j} \tag{42}$$

where $\mathbf{e}_{i,j}$ is matrix of order $|\mathcal{X}_t| \times K$ whose entries are zero except the (i,j)-th entry, which is one. Therefore, letting $\tilde{\mathbf{A}} \equiv (\mathbf{V}(t)_{\mathcal{X}_t,:})^+$, we have

$$\operatorname{tr}\Big(\nabla_{\tilde{\mathbf{B}}_{i,j}} \mathbf{V}(t)_{\mathcal{X}_t,:}\,\tilde{\mathbf{A}}\Big) = \operatorname{tr}\Big(\mathbf{e}_{i,j}\,\tilde{\mathbf{A}}\Big) = \tilde{\mathbf{A}}_{j,i} = (\tilde{\mathbf{A}}^\top)_{i,j}. \tag{43}$$

Hence we have

$$\operatorname{tr}\Big(\nabla_{\tilde{\mathbf{B}}} \mathbf{V}(t)_{\mathcal{X}_t,:}\,\tilde{\mathbf{A}}\Big) = \tilde{\mathbf{A}}^\top. \tag{44}$$

Likewise, we can obtain the gradient with respect to $\mathbf{w}(\delta)$ by observing that

$$\nabla_{w(\delta)_j} \mathbf{V}(t)_{\mathcal{X}_t,:} = \tilde{\mathbf{x}}(t-\delta)\,\mathbf{e}_j^\top, \tag{45}$$

where $\mathbf{e}_j$ is the indicator column vector of length K with only j-th entry being one. Hence, we have

$$\operatorname{tr}\Big(\nabla_{\mathbf{w}(\delta)_j} \mathbf{V}(t)_{\mathcal{X}_t,:}\,\tilde{\mathbf{A}}\Big) = \operatorname{tr}\Big(\tilde{\mathbf{x}}(t-\delta)\,\mathbf{e}_j^\top\,\tilde{\mathbf{A}}\Big) \tag{46}$$

$$= \operatorname{tr}\Big(\mathbf{e}_j^\top\,\tilde{\mathbf{A}}\,\tilde{\mathbf{x}}(t-\delta)\Big) \tag{47}$$

$$= \Big(\tilde{\mathbf{A}}\,\tilde{\mathbf{x}}(t-\delta)\Big)_j \tag{48}$$

Therefore, we establish

$$\operatorname{tr}\Big(\nabla_{\mathbf{w}(\delta)} \mathbf{V}(t)_{\mathcal{X}_t,:}\,\tilde{\mathbf{A}}\Big) = \tilde{\mathbf{A}}\,\tilde{\mathbf{x}}(t-\delta). \tag{49}$$

Proof of Lemma 4

To prove lemma 4, we will use the following lemma:

Lemma 5. *Let $\mathbf{Y}$ be a $K \times K$ symmetric matrix, and let $\mathbf{Z}$ be a $M \times K$ matrix. Then*

$$\operatorname{tr}(\mathbf{Y}\nabla(\mathbf{Z}^\top\mathbf{Z})) = 2\operatorname{tr}((\mathbf{Z}\mathbf{Y})^\top\nabla\mathbf{Z}). \tag{50}$$

Proof.

$$\operatorname{tr}(\mathbf{Y}\nabla(\mathbf{Z}^\top\mathbf{Z})) = \operatorname{tr}(\mathbf{Y}\,(\nabla\mathbf{Z})^\top\mathbf{Z} + \mathbf{Y}\,\mathbf{Z}^\top\nabla\mathbf{Z}) \tag{51}$$

$$= \operatorname{tr}((\nabla\mathbf{Z})^\top\mathbf{Z}\,\mathbf{Y} + (\mathbf{Z}\,\mathbf{Y}^\top)^\top\nabla\mathbf{Z}) \tag{52}$$

$$= \operatorname{tr}((\mathbf{Z}\,\mathbf{Y})^\top\nabla\mathbf{Z} + (\mathbf{Z}\,\mathbf{Y}^\top)^\top\nabla\mathbf{Z}) \tag{53}$$

$$= 2\operatorname{tr}((\mathbf{Z}\,\mathbf{Y})^\top\nabla\mathbf{Z}). \tag{54}$$

□

First observe that

$$\nabla_{\mathbf{B}_{i,j}} \mathbf{V}(t) = \mathbf{e}_{i,j}, \tag{55}$$

where $\mathbf{e}_{i,j}$ is matrix of order $M \times K$ whose entries are zero except the (i,j)-th entry, which is one. Because $\mathbf{A}$ is symmetric, we can use Lemma 5 to show

$$\operatorname{tr}(\mathbf{A}\,\nabla_{\mathbf{B}_{i,j}}(\mathbf{V}(\mathbf{t})^\top\mathbf{V}(\mathbf{t}))) = 2\operatorname{tr}((\mathbf{V}(t)\,\mathbf{A})^\top\nabla_{\mathbf{B}_{i,j}}\mathbf{V}(t)) \tag{56}$$

$$= 2\operatorname{tr}((\mathbf{V}(t)\,\mathbf{A})^\top\mathbf{e}_{i,j}) \tag{57}$$

$$= 2\,((\mathbf{V}(t)\,\mathbf{A})^\top)_{j,i} \tag{58}$$

$$= 2\,(\mathbf{V}(t)\,\mathbf{A})_{i,j}, \tag{59}$$

where the last equality holds because $\mathbf{A}$ is symmetric. Therefore, we have

$$\operatorname{tr}(\mathbf{A}\,\nabla_{\mathbf{B}}(\mathbf{V}(\mathbf{t})^\top\mathbf{V})) = 2\,\mathbf{V}(t)\,\mathbf{A} \tag{60}$$

Likewise, we have

$$\nabla_{w(\delta)_j} \mathbf{V}(t) = \mathbf{x}(t-\delta)\,\mathbf{e}_j^\top, \tag{61}$$

where $\mathbf{e}_j$ is the indicator column vector of length K whose only j^{th} entry is 1. Again, because $\mathbf{A}$ is symmetric, we can use Lemma 5 to show

$$\begin{aligned}
\mathrm{tr}(\mathbf{A}\,\nabla_{w(\delta)_j}(\mathbf{V}^\top\mathbf{V})) &= 2\,\mathrm{tr}((\mathbf{V}(t)\,\mathbf{A})^\top\nabla_{w(\delta)_j}\mathbf{V}(t)) && (62)\\
&= 2\,\mathrm{tr}((\mathbf{V}(t)\,\mathbf{A})^\top\mathbf{x}(t-\delta)\mathbf{e}_j^\top) && (63)\\
&= 2\,((\mathbf{V}(t)\,\mathbf{A})^\top\mathbf{x}(t-\delta))_j && (64)\\
&= 2\,(\mathbf{A}\,\mathbf{V}(t)^\top\mathbf{x}(t-\delta))_j, && (65)
\end{aligned}$$

where the last equality holds because $\mathbf{A}$ is symmetric. Therefore, we establish

$$\mathrm{tr}(\mathbf{A}\,\nabla_{w(\delta)}(\mathbf{V}(\mathbf{t})^\top\mathbf{V})) = 2\,\mathbf{A}\,\mathbf{V}(t)^\top\mathbf{x}(t-\delta). \quad (66)$$

Acknowledgments

T. O. and R. R. are supported by JST CREST Grant Number JPMJCR1304, Japan. A. G. is fully supported by JICA-Friendship Scholarship. T. S. is partially supported by JSPS Grant-in-Aid (26287019, 16H06338).

References

Affandi, R. H.; Kulesza, A.; and Fox, E. 2012. Markov determinantal point processes. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence*, 26–35.

Armijo, L. 1966. Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics* 16(1):1–3.

Boulanger-Lewandowski, N.; Bengio, Y.; and Vincent, P. 2012. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, 1159–1166.

Gartrell, M.; Paquet, U.; and Koenigstein, N. 2016. Bayesian low-rank determinantal point processes. In *Proceedings of the 10th ACM Conference on Recommender Systems*, 349–356.

Gartrell, M.; Paquet, U.; and Koenigstein, N. 2017. Low-rank factorization of determinantal point processes. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, 1912–1918.

Gillenwater, J. A.; Kulesza, A.; Fox, E.; and Taskar, B. 2014. Expectation-maximization for learning determinantal point processes. In Ghahramani, Z.; Welling, M.; Cortes, C.; Lawrence, N. D.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc. 3149–3157.

Gong, B.; Chao, W.-L.; Grauman, K.; and Sha, F. 2014. Diverse sequential subset selection for supervised video summarization. In Ghahramani, Z.; Welling, M.; Cortes, C.; Lawrence, N. D.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc. 2069–2077.

Kulesza, A., and Taskar, B. 2012. *Determinantal Point Processes for Machine Learning*. Hanover, MA, USA: Now Publishers Inc.

Kulesza, J. A. 2012. *Learning with determinantal point processes*. Ph.D. Dissertation, University of Pennsylvania.

Lütkepohl, H. 2005. *New Introduction to Multiple Time Series Analysis*. Springer-Verlag Berlin Heidelberg.

Macchi, O. 1975. The coincidence approach to stochastic point processes. *Advances in Applied Probability* 7(1):83–122.

Mariet, Z., and Sra, S. 2015. Fixed-point algorithms for learning determinantal point process. In *Proceedings of the 32nd International Conference on Machine Learning*, 2389–2397.

Mariet, Z., and Sra, S. 2016. Kronecker determinantal point processes. In Lee, D. D.; Sugiyama, M.; Luxburg, U. V.; Guyon, I.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc. 2694–2702.

Osogami, T., and Otsuka, M. 2015a. Learning dynamic Boltzmann machines with spike-timing dependent plasticity. Technical Report RT0967, IBM Research.

Osogami, T., and Otsuka, M. 2015b. Seven neurons memorizing sequences of alphabetical images via spike-timing dependent plasticity. *Scientific Reports* 5:14149.

Osogami, T. 2017. Boltzmann machines for time-series. Technical Report RT0980, IBM Research - Tokyo.

Qiao, M.; Xu, R. Y. D.; Bian, W.; and Tao, D. 2016. Fast sampling for time-varying determinantal point process. *ACM Transactions on Knowledge Discovery from Data* 11(Article No. 8).

Shirai, T., and Takahashi, Y. 2003a. Random point fields associated with certain Fredholm determinants. I. Fermion, Poisson and boson point processes. *Journal of Functional Analysis* 205(2):414–463.

Shirai, T., and Takahashi, Y. 2003b. Random point fields associated with certain Fredholm determinants. II. Fermion shifts and their ergodic and Gibbs properties. *The Annals of Probability* 31(3):1533–1564.

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

Soshnikov, A. 2000. Determinantal random point fields. *Rossiĭskaya Akademiya Nauk. Moskovskoe Matematicheskoe Obshchestvo. Uspekhi Matematicheskikh Nauk* 55(5(335)):107–160.