Determinantal Point Processes: Theory and Simulations

by
Advay Koranne
Class of 2024

Advised by

Professor Sungwoo Jeong

Mathematics Department
Cornell University

May 3rd, 2024

Contents

1 Abstract

The focus of this senior thesis will be on Determinantal Point Processes (DPPs), encompassing the exploration of background information, applications, and simulations to validate findings within the field. In the introductory section, I aim to provide a high-level overview of DPPs, their properties, and present relevant theorems. A specific emphasis will be placed on the Descents in Random Sequence DPP. We will show how to construct the kernel for this problem and verify that the kernel representation is accurate when compared to empirical simulations. There has been no formal verification to see if sampling from the descents in random sequence kernel results in similar distributions as empirical implementations. Because there are no formal methods to verify if the two methods are identical, we will provide some metrics that can be used to show approximate equivalence.

We will also provide a brief introduction to the Charlier ensemble DPP which is related to the problem of the longest weakly increasing subsequence in a random word. Lastly, we will explore the applications of DPPs, particularly within the realm of machine learning, a domain that has recently attracted increased attention. We will show that using a modified determinantal point process for gradient descent leads to more stable convergence. We will also highlight some recent papers in the space of deep learning that have been using DPPs to solve some of the existing problems in deep learning.

2 Acknowledgments

First of all, I would like to thank my parents Sandeep Koranne and Jyoti Aneja. They have both always encouraged me to explore my interests from a young age and have supported me in all my endeavors. I would also like to thank the friends I have made during my four years here at Cornell University who have encouraged me to explore new fields and opportunities outside of my coursework. Lastly, I would like to thank Professor Sungwoo Jeong as without his guidance this thesis would not have been possible.

3 Overview

Let us first provide a high-level overview of what Determinantal Point Processes (DPP) are and some of their important theorems and properties. DPPs are probabilistic models that have negative correlation and can allow for computationally efficient algorithms for sampling, marginalization, conditioning, and other tasks. DPPs are also stochastic point processes with the characteristic that the probability distribution is characterized as a determinant of some matrix.

Let us first provide an overview of point processes. A **point process** on a

ground set Y is a probability measure on the power set of Y which is 2^Y. This section delves into point processes as that will provide some important background for DPPs.

Let us suppose we have a set of elements $\{\text{apple}, \text{banana}, \text{orange}, \text{mandarins}\}$, a realization of this set could be: $\{\text{apple}, \text{banana}\}$ (example from [Bar18]). We know that apples and bananas are different but oranges and mandarins are very similar. A point process has attraction if two elements tend to be together in a subset – this would mean that oranges and mandarins appear in a subset together. We have repulsion if this is not true – i.e. we have that mandarins and oranges do not appear together. In a broad sense DPPs are a kind of point process with repulsion where points are selected to encourage diversity.

Definition 3.1. Determinantal Point Processes: A point process P is called a determinantal point process, if Y is a random subset drawn according to P, then we have for every $S \subseteq Y$: $P(S \subseteq Y) = \det(K_s)$ for K some similarity matrix $K \in \mathbb{R}^{n \times n}$.

K_s is a submatrix of K which is obtained by restricting the rows and and columns indexed by S. Because P is a probability measure all the principal minors (refers to the determinants of submatrices that are derived by deleting any combination of rows and columns from the original matrix) $\det(K_s)$ of K must be non-negative and therefore K must be positive and semidefinite. Because of the property that K is positive and semidefinite, we can show that the eigenvalues of K must be in between 0 and 1. K is called the **marginal kernel**.

We call K the marginal kernel because it has all the information needed to compute the probability of having any subset A being in Y. Let us assume that our subset A has a single element which we will denote $A = \{i\}$. Then we get that $P(i \in Y) = K_{ii}$. This tells us that the marginal probability of including individual elements of Y are given by the diagonal elements of K. If we are given two elements i.e. $A = \{i, j\}$ then:

$$\begin{aligned} P(i, j \in Y) &= \begin{vmatrix} K_{ii} & Kij \\ K_{ji} & K_{jj} \end{vmatrix} \\ &= K_{ii}K_{jj} - K_{ji}K_{ji} \\ &= P(i \in Y)P(j \in Y) - K_{ij}^2 \end{aligned}$$

Looking at this equation we can see that the element K_{ij} determines the negative correlation between two elements. Remember that K is a similarity matrix — therefore, if the value of K_{ij} is large then there is a low probability that i and j will appear together. In the case where $K_{ij} = \sqrt{K_{ii}K_{jj}}$ then i and j are identical and will not appear together at all.

Let us now provide a simple example that shows how the kernel K can encourage diversity of points. Let us start by randomly plotting $n = 50$ points

on a plane [Bar18]. Let us now use the traditional Gaussian Kernel which is defined as the following:

$$L_{ij} = \exp(-\frac{1}{2l^2}||x_i - x_j||^2)$$

Points that are close together are similar and points further apart are dissimilar. Now using the matrix let us pick $k = 3$ points that are far apart using the Kernel matrix L (see Figure 1). The points in grey are the randomly sampled $n = 50$ points and the points in red are the points chosen using the Kernel L. In Figure 2 we can see the sample we did for $k = 20$.

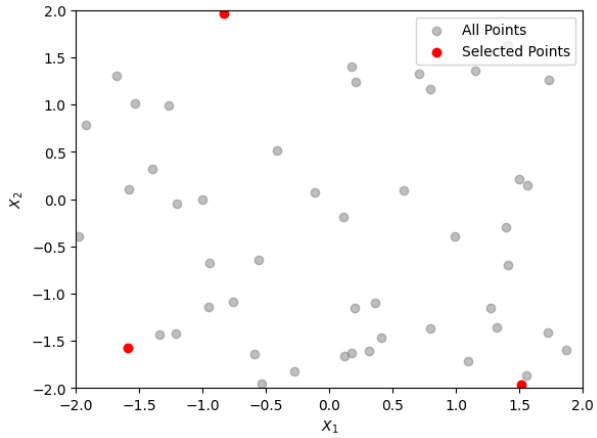

Figure 1: Gaussian kernel Point Process

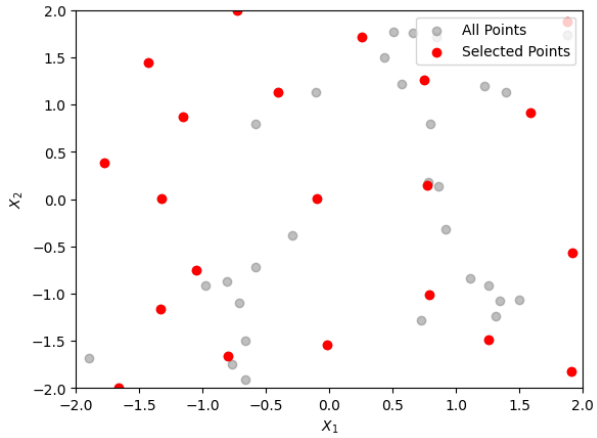

Figure 2: Gaussian kernel Point Process

We can clearly see in Figure 2 that by selecting points based on the Gaussian Kernel we were able to select points that are dissimilar.

3.1 L-ensembles

For data modeling it is helpful to look at a restricted version of DPPs called L-ensembles. A L-ensemble is a DPP, but rather than the marginal kernel K there is a symmetric matrix L that is indexed by the elements of Y. In the equation for DPPs above we were given the marginal probabilities but we will now look at the direct possibility of every instance of X: $P_L(X = Y) \propto \det(L_y)$. The statement given above is just for proportionality and does not require that the eigenvalues of L be less than one. In order to normalize we can use the following closed-form equation:

Theorem 1. For any $A \subseteq \mathcal{Y}$:

$$\sum_{A \subseteq Y \subseteq \mathcal{Y}} \det(L_y) = \det(L + I_{\bar{A}})$$

where we define $I_{\bar{A}}$ as a diagonal matrix with ones in the diagonal positions which are the elements from $\bar{A} = \mathcal{Y} - A$ and zeroes everywhere else.

Theorem 2. An L-ensemble is a DPP, and its marginal kernel is:

$$K = L(L + I)^{-1} = I - (L + I)^{-1}$$

3.2 Properties

Throughout this paper, we have used three terms that are not universal in DPPs. The first term we have used in this paper is **restriction** which is defined as $P(A \subseteq Y)$. We have called this restriction because in Definition 3.1, we restrict the kernel to a submatrix. The second term we have used is **complement**. This is because we want $P(A \cap \mathbf{Y} = \emptyset)$. The third term we have used is **conditioning** which we define as $P(\mathbf{Y} = B | A \cap \mathbf{Y} = \emptyset)$. This is because we are conditioning on $A \cap \mathbf{Y} = \emptyset$.

Lemma 3. Restriction: If Y is distributed as a DPP with marginal kernel K, then $Y \cap A$, where $A \subseteq \mathcal{Y}$, is also distributed as a DPP with marginal kernel K_A. Let us prove this statement as we will use this property frequently in this paper.

Proof [Kul12]:

$$\begin{aligned} P_L(A \subseteq Y) &= \frac{\sum_{Y \subseteq A} \det(L_Y)}{\sum_Y \det(L_Y)} \\ &= \frac{\det(L + I_{\bar{A}})}{\det(L + I)} \\ &= \det((L + I_{\bar{A}})(L + I)^{-1}) \end{aligned}$$

We can use the following property: $L(L+I)^{-1} = I - (L+I)^{-1}$ to get:

$$\begin{aligned} &= \det(I - (L+I)^{-1} + I_{\bar{A}}(L+I)^{-1}) \\ &= \det(I - I_A(L+I)^{-1}) \end{aligned}$$

Let us let $K = I - (L+I)^{-1}$

$$= \det(I_{\bar{A}} - I_A(K))$$

Multiplying by I_A just keeps all the rows in A and makes everything else 0. Letting $Y = \bar{A} \cup A$ we get:

$$\det(I_A + I_A K) = \begin{vmatrix} I_{|A|\times|A|} & 0 \\ K_{AA} & K_A \end{vmatrix} = \det(K_A) \blacksquare$$

Lemma 4. Complement: If $\mathbf{Y}$ is distributed as a DPP with a marginal kernel K, then $\mathcal{Y} - \mathbf{Y}$ is also distributed as a DPP, with marginal kernel $\bar{K} = I - K$. We get:

$$P(A \cap \mathbf{Y} = \emptyset) = \det(\bar{K_A}) = \det(I - K_A)$$

Lemma 5. Conditioning: The distribution by conditioning on a DPP that has none of the elements in A can be seen as the following. For $B \subseteq \mathcal{Y}$ not intersecting with A we have:

$$P_L(\mathbf{Y} = B | A \cap \mathbf{Y} = \emptyset) = \frac{P_L(\mathbf{Y} = B)}{P_L(A \cap \mathbf{Y}) = \emptyset} = \frac{\det(L_B)}{\det(L_{\bar{A}} + I)}$$

3.3 DPP Samplers

The high-level idea of using a DPP sampler is to select diverse subsets of items from a larger set where the diversity is mathematically quantified by the properties of determinants. One of the most popular sampling algorithms was published in 2006 but relies on the eigendecomposition of the DPP kernel L. The most commonly used algorithms for eigendecomposition are based on the QR algorithm and have a complexity of $O(n^3)$. Let $L = \sum_{n=1}^{N} \lambda_n v_n v_n^T$ be an orthonormal eigendecomposition of a positive semidefinite matrix L then the following algorithm samples $\mathbf{Y} \sim P_L$. We have provided the pseudo-code in Algorithm 1. [Kul12]

Algorithm 1 QR Based DPP Sampler

Require: eigendecomposition $\{(v_n, \lambda_n)\}_{n=1}^{N}$ of L

$J \leftarrow \emptyset$
for $n = 1$ to N **do**
    $J \leftarrow J \cup \{n\}$ with probability $\frac{\lambda_n}{\lambda_n + 1}$
end for
$V \leftarrow \{v_n\}_{n \in J}$
$Y \leftarrow \emptyset$
while $|V| > 0$ **do**
    Select i from Y with $\Pr(i) = \frac{1}{|V|} \sum_{v \in V} (v^T e_i)^2$
    $Y \leftarrow Y \cup \{i\}$
    $V \leftarrow V_\perp$, an orthonormal basis for the subspace of V orthogonal to e_i
end while

Ensure: Y

We have included Algorithm 1 as it is the most widely used DPP sampling algorithm.

For most of our experiments in this paper, we have used the much simpler DPP sampler from the paper published in 2019 titled "High-performance sampling of generic Determinantal Point Processes" which has efficient direct sampling schemes for non-Hermitian and Hermitian DPP kernels. [Pou20] We have provided the pseudo-code in Algorithm 2.

Algorithm 2 Simple DPP Sampler

Require: K and an integer n

$sample \leftarrow []$
$A \leftarrow K$
for $j = 0$ **to** $n - 1$ **do**
  if $Bernoulli(A_j)$ **then**
    $sample.append(j)$
  else
    $A_j \leftarrow A_j - 1$
  end if
  $A_{j+1:n,j} \leftarrow A_{j+1:n,j}/A_j$
  $A_{j+1:n} \leftarrow A_{j+1:n} - A_{j+1:n,j}A_{j,j+1:n}$
end for

Ensure: $sample, A$

3.4 Check DPP

Let us now check if our implementation of Algorithm 2 is correct. One method to check if our DPP sampler is correct is that if Y is a random subset drawn to P, then we know from Lemma 3:

$$P(A \subseteq Y) = \det(K_A)$$

where K is defined as some real symmetric $N \times N$ matrix, where we define $K_\emptyset = 1$ and $K_A = [K_{ij}]_{i,j \in A}$. Because P is a probability measure $\det(K_A)$ must be non-negative and therefore K must be positive semi-definite.

To test if our DPP sampler is correct according to the theoretical values, let us first construct a positive semi-definite matrix K. We will do this by first constructing a random $A \in \mathbb{R}^{3\times 3}$ matrix and letting $K = A \cdot A^T$. For our experiments, we have set the seed in NumPy for reproducibility (see Section 8 for GitHub repository of code). We will then divide K by two times the maximum eigenvalue to normalize all the values.

We will calculate the determinant for every sub-matrix using the power set of $X = \{1, 2, 3\}$. The power set of X is $\{\{\}, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$. For example, we will compute the determinant for $K_{1,1}$ which will just be the value at $(1, 1)$. We will then compute the determinant for $\{2, 3\}$ which will be the bottom right 2 by 2 matrix (see figure 3) and do the same for the rest of the elements in the power set.

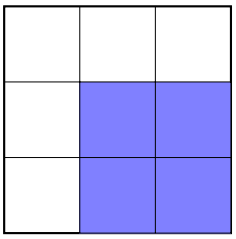

Figure 3: DPP Submatrix $\{2, 3\}$.

The determinant of the submatrices gives us the theoretical probability of the subset being in our sample. For all subsets of the power set we have listed the determinant of the corresponding submatrix in Table 1.

Subset	**Determinant of K**
()	1.0
(1)	0.0641
(2)	0.2272
(3)	0.3089
(1, 2)	0.0110
(1, 3)	0.0186
(2, 3)	0.0225
(1, 2, 3)	0.00098

Table 1: Determinant of K for Various Subsets

Now to compute the exact value: $P_L(\mathbf{Y} = Y)$ we can use the following formula:

$$P_L(\mathbf{Y} = Y) = \frac{\det(L_Y)}{\det(L + I)}$$

Let us list all these theoretical probabilities using this equation in Table 2.

Because Table 2 gives us: $P_L(\mathbf{Y} = Y)$ and in Table 1 we calculated $P(A \subseteq Y)$ we can compare to check the values to make sure that everything is correct. For example, $P(\{2, 3\} \subseteq Y)$ is given in Table 1 and that should be equal to $P_L(\mathbf{Y} = \{2, 3\}) + P_L(\mathbf{Y} = \{1, 2, 3\})$. According to Table 1 $P(\{2, 3\} \in Y =$

Subset	Probability
()	0.4509
(1)	0.0355
(2)	0.1947
(3)	0.2689
(1, 2)	0.0100
(1, 3)	0.0176
(2, 3)	0.0215
(1, 2, 3)	0.00098

Table 2: Probabilities using L Matrix for Various Subsets

0.0225. Now using Table 2 we get $P_L(\mathbf{Y} = \{2, 3\}) + P_L(\mathbf{Y} = \{1, 2, 3\}) = 0.0215 + 0.00098 = 0.02248$. This value is very similar to the value we got above using our determinant.

If we now use our sampler that we implemented using Algorithm 2 we can compute $P_L(\mathbf{Y} = Y)$. Let us sample $10,000,000$ samples, and compute the probabilities of seeing a specific sub-set and compare that to the theoretical values. We have provided the comparison in Table 3.

Subset	L Matrix	Poulson's Sampling	Absolute Difference
()	0.450859	0.450906	0.000047
(1)	0.035486	0.035495	0.000009
(2)	0.194681	0.194575	0.000106
(3)	0.268857	0.268916	0.000059
(1, 2)	0.010034	0.010004	0.000030
(1, 3)	0.017600	0.017610	0.000010
(2, 3)	0.021507	0.021530	0.000023
(1, 2, 3)	0.000977	0.000963	0.000014

Table 3: Comparison of Probabilities: Poulson's Sampling Algorithm vs. L Matrix for Various Subsets

We can see in Table 3 that the values of using Algorithm 2 are very similar to the theoretical values. Therefore, we are able to validate the implementation of our DPP sampler.

4 Descents in Random Sequences

4.1 Overview

Let us now look at an example of a well studied DPP called Descents in Random Sequences. [BDF09] Assume we have an alphabet $\mathbb{B} = \{0, 1, ...b-1\}$, and let $B_1, B_2, ..., B_n$ be a sequence or randomly choose elements of $\mathbb{B}$. We say there is a descent at index i if $B_i > B_{i+1}$ where $1 \leq i \leq n-1$. For example, given the sequence: [4, 6, 3, 8, 7, 7, 2] the descents happen at index: [2, 4, 6] – we are using 1 indexing.

If we are given a sequence of N random numbers drawn uniformly and independent from some finite set, it has been shown that this point process of descents P_n is determinantal with correlation kernel $K(i,j) = k(j-i)$ where:

$$\sum_{m \in \mathbb{Z}} k(m)t^m = \frac{1}{1-(1-t)^b}$$

Let us construct this matrix for $b = 3$. We get $\frac{1}{1-(1-t)^3}$. If we expand this using either a library in Python such as SymPy or use something like Wolframalpha we get the following values as the coefficients:

m	-1	0	1	2	3	4	5	6	7	8	9	10	11	12
$k(m)$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{2}{9}$	$\frac{1}{9}$	$\frac{1}{27}$	0	$-\frac{1}{3^4}$	$-\frac{1}{3^4}$	$-\frac{2}{3^5}$	$-\frac{1}{3^5}$	$-\frac{1}{3^6}$	0	$\frac{1}{3^7}$	$\frac{1}{3^7}$

Let us now construct the correlation matrix given that $K(i,j) = k(j-i)$. Let us compute some values to show how one can fill in the matrix. Let $K(0,0) = k(0) = 1/3$. $K(3,1) = k(-2) = 0$. $K(3,5) = k(2) = 1/9$. Therefore, we get the following correlation matrix:

$$(K(x,y))_{i,j=1}^{6} = \begin{bmatrix} \frac{1}{3} & \frac{2}{9} & \frac{1}{9} & \frac{1}{27} & 0 & -\frac{1}{81} \\ \frac{1}{3} & \frac{1}{3} & \frac{2}{9} & \frac{1}{9} & \frac{1}{27} & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & \frac{2}{9} & \frac{1}{9} & \frac{1}{27} \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{2}{9} & \frac{1}{9} \\ 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{2}{9} \\ 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$$

We now use the method above to construct the kernel matrix K. We expand the polynomial using sympy and then construct the matrix following the method above.

4.2 Empirical Implementation

In-order to check that the kernel is correct we wrote a empirical implementation of Descents in random sequences (see Algorithm 3).

Algorithm 3 Find Descents in a Sequence

Require: A sequence of numbers stored in `sequence`
Ensure: List of descents' positions

```
descents ← []
for i = 1 to length(sequence) − 1 do
    if sequence[i − 1] > sequence[i] then
        descents.append(i)
    end if
end for
return descents
```

Sampling from our DPP sampler will give us the distribution of the descent index. Our main goal is to show that sampling using the DPP kernel leads to a similar distribution of samples as compared to using our empirical implementation. There is no theoretical way to compare the two methods so we will check common characteristics of the distribution among the two sampling methods.

4.3 Empirical vs. Theoretical Experiments

4.3.1 Descent Index Distribution

The kernel provided in section 4.1 gives the distribution of the descent index (i.e. the location of numbers where the current number is less than the previous number). Sampling from the DPP using Algorithm 2 will give us the distribution of the descent index – this will be our theoretical value. I want to highlight that when we pass our kernel to the sampler we will get a sequence – this sequence is the distribution of the descent index. To compare if this matches the empirical implementation we generated $1,000,000$ sequences of numbers from $[1,3]$ of length 7. Here is an example of such sequence: $[3,3,1,1,2,2,1]$ and the corresponding descent index: $[2,6]$. We then computed the descent index for all $1,000,000$ samples and plotted the frequency of the index on the histogram of the two methods (see Figure 4).

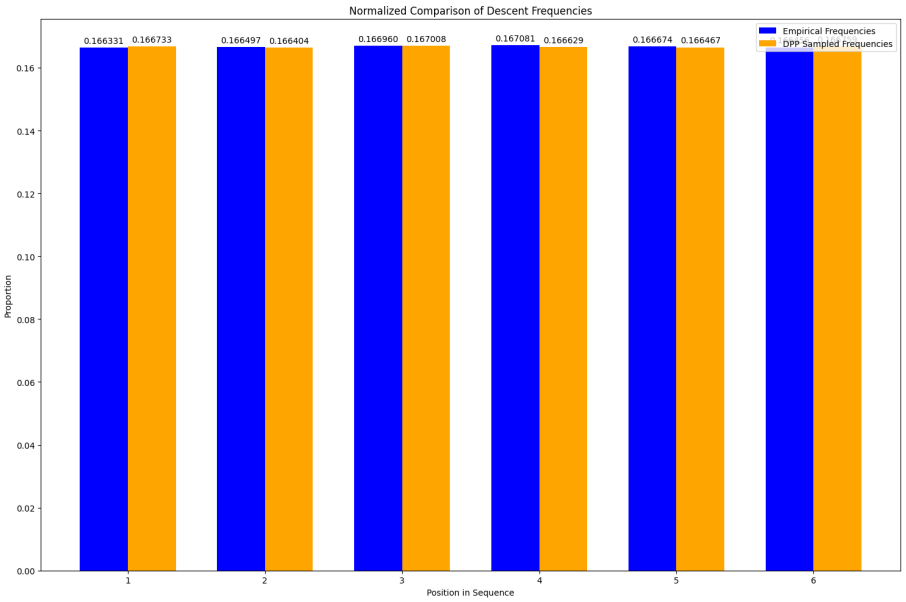

Figure 4: Comparison of Descent Frequencies

We can clearly see in Figure 4 that the distribution of descent index of the theoretical value by sampling from the DPP kernel is the same as generating $1,000,000$ sequences and emperically finding the index where we observe descents.

We will now look at some other metrics to confirm that the distribution of the indices of the empirical method is the same as if we were to sample from the DPP. The next metric we have looked at is the distribution of the maxi-

mum index in the sequence. For example, if the sequence sampled is $[4, 2, 6]$ the maximum index is 6. In Figure 5 we can see the distribution of the empirical vs. the DPP kernel for the largest index.

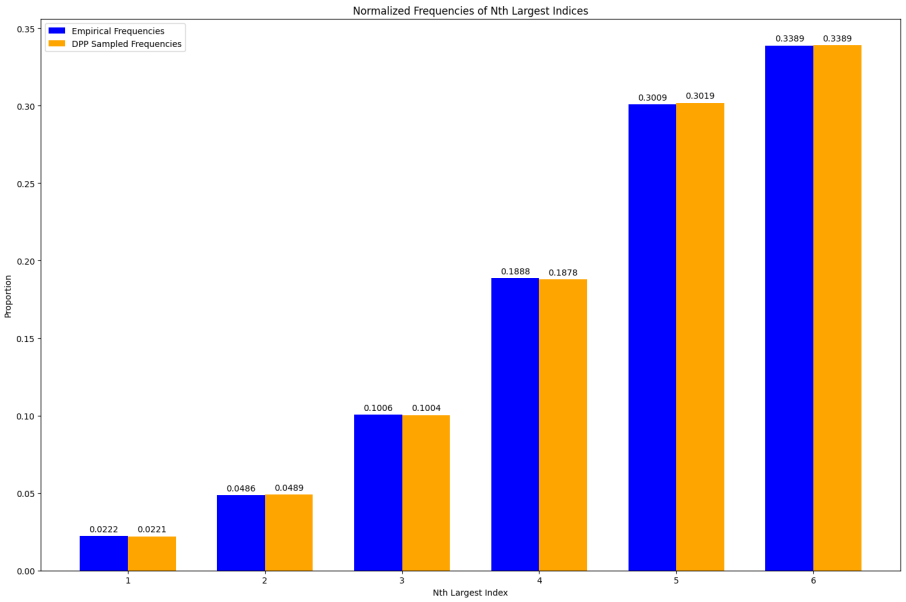

Figure 5: Comparison of Largest Descent Index

Now we can extrapolate from using the largest index and check for the n-th large index. Figure 6 has the distribution for the 2nd largest index, and Figure 7 has the result for the 3rd largest index. We can see that the distributions are very similar indicating that using the kernel to sample is almost identical to using empirical sampling.

We can see that both sampling using the DPP kernel and using the the empirical method result in similar distributions of normalized indices for different metrics.

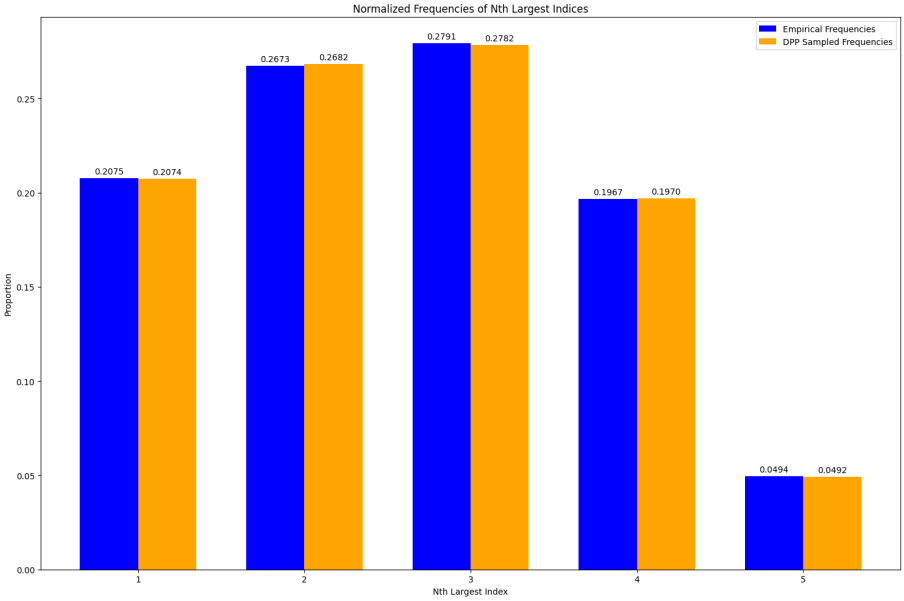

Figure 6: 2nd Largest Descent Index

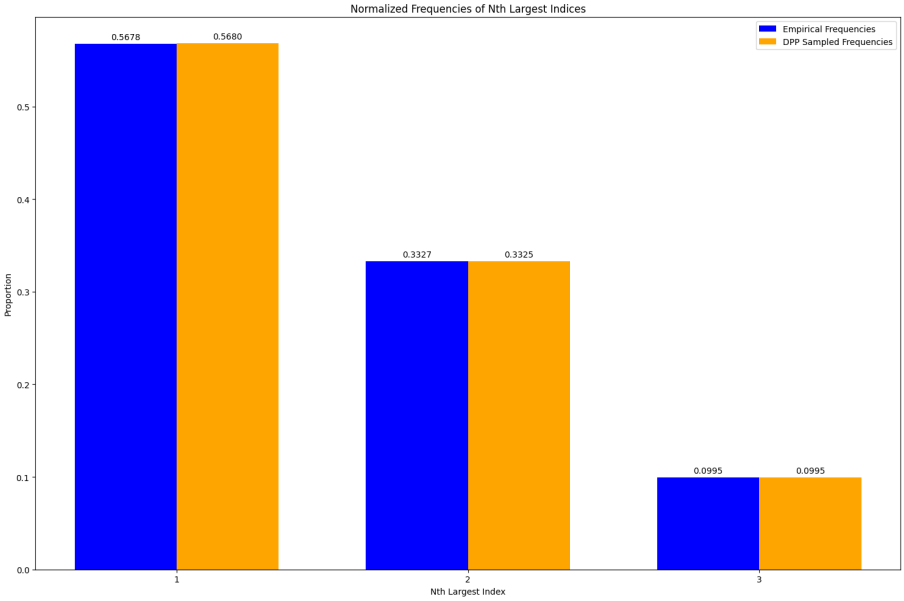

Figure 7: 3rd Largest Descent Index

4.3.2 Complement Probabilities

We know that DPPs offer very efficient ways to compute conditional and certain probabilities of subsets. One example was with the complement of a set as seen in Lemma 4. Let us suppose we want to calculate the probability that: $\mathbb{P}(\text{Max Index} = 6)$. We can calculate this with the following equation: $\mathbb{P}(\text{Max Index} = 6) = \mathbb{P}(\text{Max Index} < 6) - \mathbb{P}(\text{Max Index} < 5)$.

We first implemented this in code by using our empirical descent function above and sampling $1,000,000$ sequences. The complement formula is given as $P(A \cap \mathbf{Y} = \emptyset) = \det(\bar{K_A}) = \det(I - K_A)$. To find $\mathbb{P}(\text{Max Index} < 6)$ we want $A = \{6\}$, and to find $\mathbb{P}(\text{Max Index} < 5)$ we want $A = \{5, 6\}$ – i.e. we want the sample to not contain 5 or 6. Here are the results we get from the empirical experiment:

Index	Cumulative Probability	Exact Probability
$P(X < 7)$	1.000000	-
$P(X < 6)$	0.650161	0.295614
$P(X < 5)$	0.354547	0.185659
$P(X < 4)$	0.168888	0.098707
$P(X < 3)$	0.070181	0.047992
$P(X < 2)$	0.022189	0.005627
$P(X < 1)$	0.016562	0.016562

Table 4: Empirical Experiment

Let us now generate the same values by using the theoretical value from Lemma 4.

Index	Cumulative Probability	Exact Probability
$P(X < 7)$	1.000000	-
$P(X < 6)$	0.666667	0.296296
$P(X < 5)$	0.370370	0.185185
$P(X < 4)$	0.185185	0.098765
$P(X < 3)$	0.086420	0.048011
$P(X < 2)$	0.038409	0.021948
$P(X < 1)$	0.016461	0.016461

Table 5: Theoretical Probabilities

Now to find the exact values i.e $\mathbb{P}(X = N)$ we can compute:

$$\mathbb{P}(X = N) = \mathbb{P}(X < N) - \mathbb{P}(X < (N - 1))$$

.

Index	Empirical Data	Theoretical	Absolute Difference
P(X=6)	0.3015	0.2963	0.0052
P(X=5)	0.1883	0.1852	0.0031
P(X=4)	0.1002	0.0988	0.0014
P(X=3)	0.0485	0.0480	0.0005
P(X=2)	0.0224	0.0219	0.0005
P(X=1)	0.0166	0.0165	0.0001

Table 6: Comparison of Probability Values with Absolute Differences

We can see from Table 6 that the empirical data follows the distribution of the theoretical values as evident with the small difference in the absolute error.

4.3.3 Restricted Probabilities

Now let us look at some of the unique areas where DPPs are particularly unique for sampling. Let us suppose that our goal now is to sample from the DPP but to ignore indices that are outside s. For example, if $s = 5$ and our sample sequence is: $sequence = [4, 6, 3, 8, 7, 7, 2]$ then the filtered sequence is $[2, 4]$. Similar to before we have implemented this empirically where we generate sequences and remove the indices greater than or equal to s. But, to use our sampler we now restrict our kernel matrix K to be $K' = K[s, s]$. We have essentially just restricted our matrix to be the first k rows and k columns and then we sample using our same DPP sampler shown above. In Figure 8 we we see the descent frequency for $s = 3$ and in Figure 9 we can see the descent frequency for $s = 4$.

As one can see in Figure 8 and Figure 9, the distributions of the indices using the empirical implementation are the same as using the restricted approach with sampling.

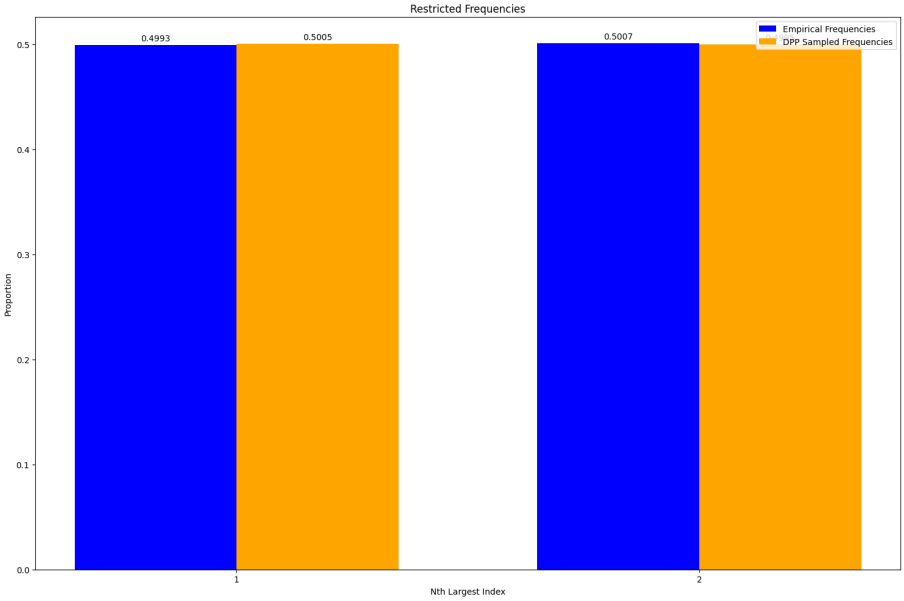

Figure 8: Comparison of Descent Frequencies Restricted to $s = 3$

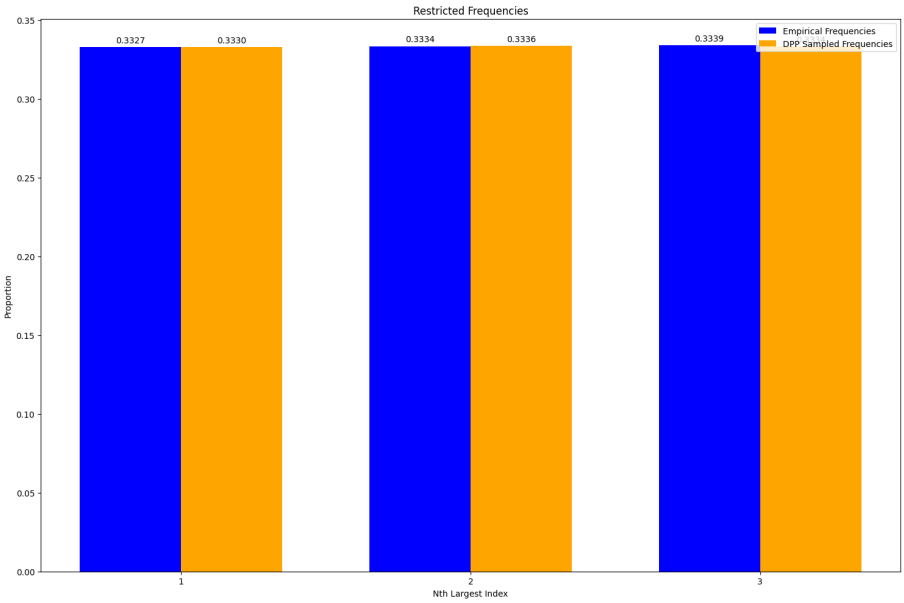

Figure 9: Comparison of Descent Frequencies Restricted to $s = 4$

4.3.4 Conditional Probabilities

Another unique way to use DPPs is in the ability to condition in an efficient way. Let us suppose we want to do the following sampling method: we want to sample from a kernel K and throw away any sample that contains anything from S. This is identical to creating a new matrix K' using L-ensembles:

$$\begin{aligned} L &= K(I-K)^{-1} \\ K' &= \frac{L[s,s]}{(I+L[s,s])} \end{aligned}$$

Once again we have implemented an empirical function and used our DPP sampler to compute the distribution of values. The empirical function we wrote throws away any descent index sequence if it contains anything from S. In Figure 10 we have the comparison of using conditioning vs. using the empirical sampler for $s = 3$ and in Figure 11 we have the result for $s = 4$.

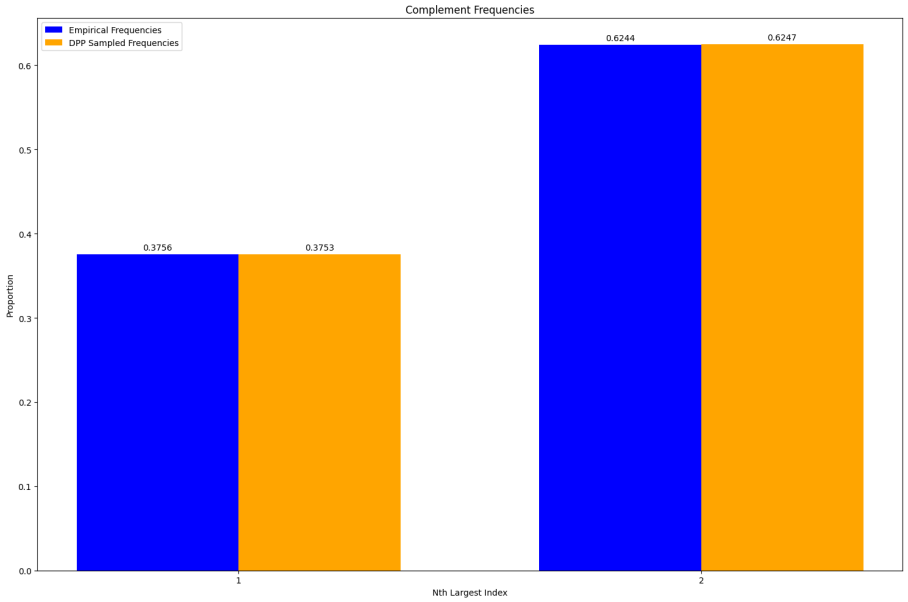

Figure 10: Conditioned Sampling restricted to values outside of $s = 3$

It is very important to note that we have done two different ways above. The first one was to sample and then just ignore the indices that are greater than or equal to s, and the second method was to throw away the entire sample if it contains anything greater than or equal to s. These two methods are very different and lead to different distributions.

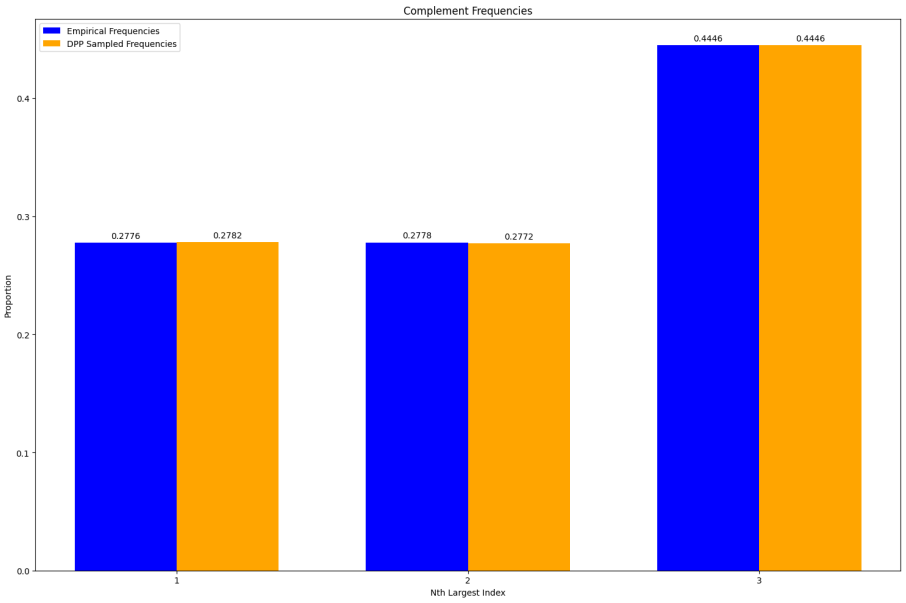

Figure 11: Conditioned Sampling restricted to values outside of $s = 4$

4.4 Examples of Determinantal Point Processes

Now that we have looked at the Descents in Random Sequences and the corresponding correlation matrix let's now provide a brief overview of some of the other popular DPPs.

4.4.1 Aztec Diamond Tilings

One of the most famous DPPs is the Aztec diamond tilings. The Aztec diamond is a diamond-shaped object made out of smaller squares that are either white or grey (think of a checkerboard). If we have 2×1 domino tiles that cover the diamond then it has been proved that the subset of squares that are colored grey that are on the left half of a horizontal tile or on the bottom of a vertical tile are distributed as a DPP (see Figure 12). [Gup15]

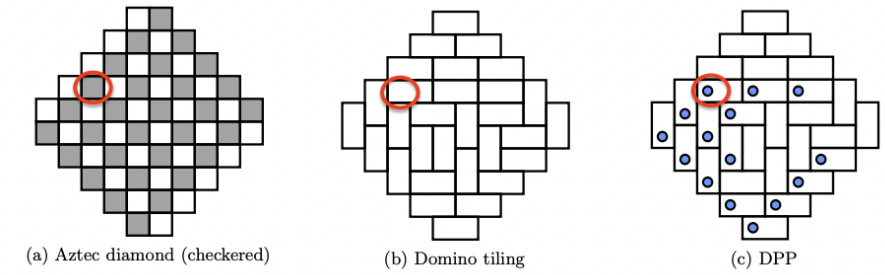

Figure 12: Aztec Diamond Tilings (image from [Gup15])

4.4.2 Random Matrices

If we construct a random matrix M and sample its entries independently from the complex normal distribution, then the eigenvalues of M are distributed as a DPP.

4.4.3 Non-intersecting Random Walks

Let us assume we are given k independent random walks. Each walk can be represented as: $x_1, x_2, ...x_T$. For each of the k walks let the starting position be $x_1^1, x_1^2, ..., x_1^k$ and the ending position $x_T^1, x_T^2, ..., x_T^k$. None of the k trajectories intersect (see Figure 13). It has been proved that at any time t the positions of each of the k trajectories $x_t^1, x_t^2, ..., x_t^k$ are a subset of $\mathbb{Z}$ and are distributed according to a DPP. [Kul12]

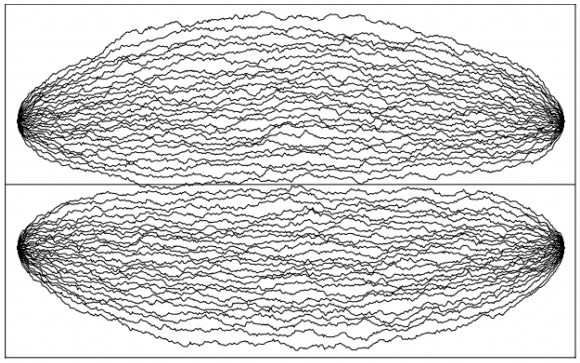

Figure 13: Example of Non-intersecting Random Walks from (image from [AFM13])

4.5 Summary

In this section we first provided an overview of the Descents in Random Sequences problem. We also showed how one can construct the correlation kernel. Because there have been no formal ways to verify the validity of a kernel using simulations we constructed an empirical implementation. We then evaluated different metrics such as the distribution of the descent indexes, the distribution of the n-th largest index. We also looked at restricted, complemented, and conditional probabilities. All of these metrics support the claim that the correlation kernel of the Descents in Random Sequences problem does in fact encode information about the distribution of descents in random sequence.

5 Machine Learning Applications

5.1 Overview

In this section, we will look at multiple applications of DPPs in machine learning. Although DPPs have been around since 1965, their applications in the field of machine learning have been much more recent – i.e. starting mostly since 2000. One of the main benefits that DPPs offer is sampling and conditioning very efficiently. DPPs are very appealing for machine learning applications because they can capture "negative interactions" between modeling variables. Let us first give some background of some of the machine learning theories we will be presenting in the rest of this paper.

In machine learning, there are two main categories of learning: the first one is supervised learning and the second one is unsupervised learning. Supervised learning involves training a model based on a labeled dataset where we are given the correct output and the goal of the model is to predict the output. One of the most common examples of supervised learning is spam detection. The model is trained on many emails that have labels of whether the email is spam or not and then it learns to classify emails. Unsupervised learning is data that does not have labels and the goal of the model then becomes to find patterns or groupings within the data. One example of unsupervised learning is that we can group users based on their song play histories forming clusters of users with similar musical tastes. Once this is trained we can use this to recommend new songs to users.

In this section we will first look at how we can use the the fact that DPPs can be used to select diverse points in the context of gradient descent. We will then look at how this encouragement of diversity can also be used in clustering algorithms to select centroids in a much more efficient manner. Lastly, we will look at how DPPs are being used in deep learning to address common problems of mode collapse and misalignment.

5.2 Determinantal Point Processes for Gradient Descent

If we recall the Gaussian Kernel Point process Figure 2, we saw that we can use the Gaussian Kernel to select points that are far apart. In 2017 there was a paper published titled "Determinantal Point Processes for Mini-Batch Diversification." [ZKM17]

Gradient descent plays a critical role in most machine learning algorithms. In machine learning our main goal is to reduce the error between the predicted output of our model and the actual outputs. Our goal in gradient descent is to move towards the minimum value of our loss function and to find the parameters that minimize our loss function.

Let us assume we have some multi-variable function $F(x)$ that is well defined and differentiable in the neighborhood of some point a. Then the direction of the negative gradient of a is given as: $-\nabla F(a)$. Therefore, for some small step-size or learning $\gamma \in \mathbb{R}_+$ we get that the next iteration of a is given as: $a_{n+1} = a_n - \gamma \nabla F(a_n)$. Let us quickly provide an overview of the most common gradient descent algorithms:

1. **Batch Gradient Descent:** for this all the training data is taken into consideration for a single step. We then take the average of all the gradients to update our parameters.

2. **Mini-batch Gradient Descent:** we use a small subset of the data which is known as a mini-batch to compute the gradient of the loss function. The benefit to this is that it is more efficient but has some more stability as we are using a batch of gradients.

3. **Stochastic-Gradient-Descent SGD:** in this case we use an individual data point drawn at random. This can be less stable than mini-batch gradient descent but can be more efficient as we are just using a single data point.

We will now look at a novel method published that uses a DPP for gradient descent. We saw above that at a high-level DPPs allow for the ability to choose diverse data and by using Diversified Mini-Batch SGD (DM-SGD) the authors of the paper were able to show that it could lead to better classification accuracies in supervised set-ups.

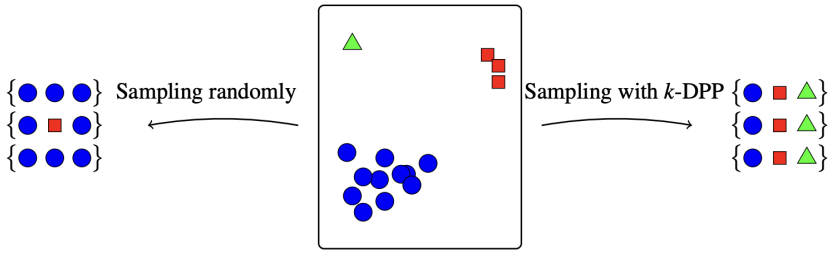

Figure 14: Sampling mini-batches using the k-DPP from Determinantal Point Processes for Mini-Batch Diversification paper (image from [ZKM17])

Because mini-batches have a fixed size we can use a k-DPP for sampling. By having diverse samples we are able to have variance reduction as compared to stochastic gradient descent. The paper proposes the following gradient update that is based on the diversified mini-batches of a fixed size k:

$$\theta_{t+1} = \theta_t - \rho_t \frac{1}{k} \sum_{i \in B} \nabla \ell(\theta, x_i), \quad B \sim \text{k-DPP}.$$

The algorithm in the paper tells us to sample from the kernel using an eigendecomposition. Let us first look at a simple way of understanding why using this k-DPP approach might be more beneficial as compared to using mini batch or stochastic gradient descent. Let us do a similar thing as we did above for the Gaussian-Kernel and create this similarity matrix for all our training data X.

Algorithm 4 Modified Diversified Mini-Batch Selection

Require: similarity matrix K, batch size b, total samples N
Ensure: A diversified mini-batch selection Y

$Y \leftarrow \emptyset$
Select an initial index i at random from $\{1, 2, \ldots, N\}$
$Y \leftarrow Y \cup \{i\}$
while $|Y| < b$ **do**
    Calculate the sum of similarities for each index not in Y with respect to Y
    Find the index j not in Y with the minimum sum of similarities
    $Y \leftarrow Y \cup \{j\}$
end while

5.2.1 Experiments and Results

For simplicity we have decided to look at logistic regression with two classes first. We created 1000 synthetic data points and used a $80 - 20$ training-testing split. To train the model we did three types of gradient descent:

1. Stochastic-Gradient Descent with one example.
2. Batch gradient descendent with a batch size of 32.
3. Diversified mini-batch selection with a batch size of 32.

In Figures 15, 16, 17 we can see the learned hyper-planed and the data we used to train and test. We can see that for all of the examples that we get a hyper-plane that clearly separates the two classes and that they are all almost identical.

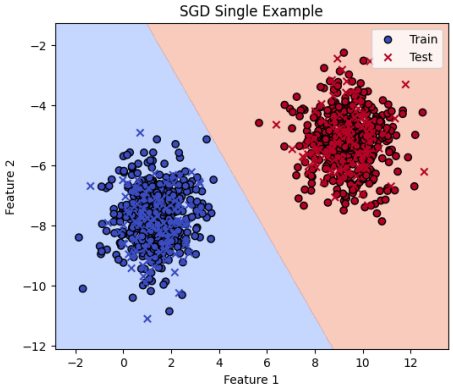

Figure 15: SGD Single Example

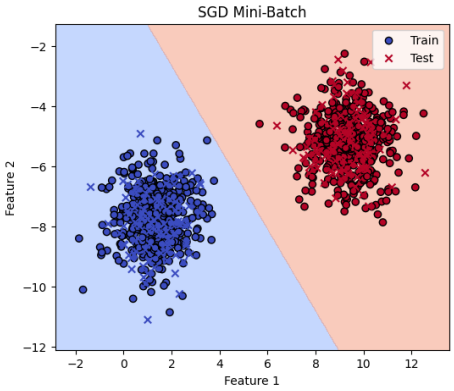

Figure 16: SGD Mini-batch

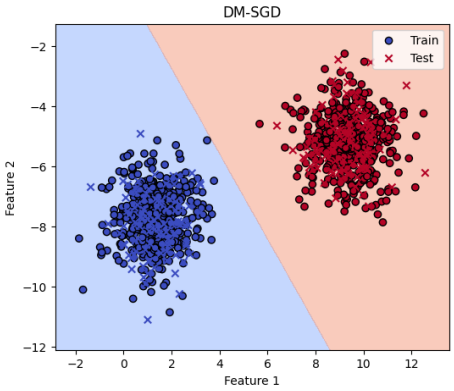

Figure 17: DM-SGD

One of the main appeals of using the Diversified Mini-Batch SGD was that it should experience less variance during gradient descent as we are picking samples that are diverse. In order to test this we plotted the loss over training epochs for all three SGD algorithms (see Figure 19).

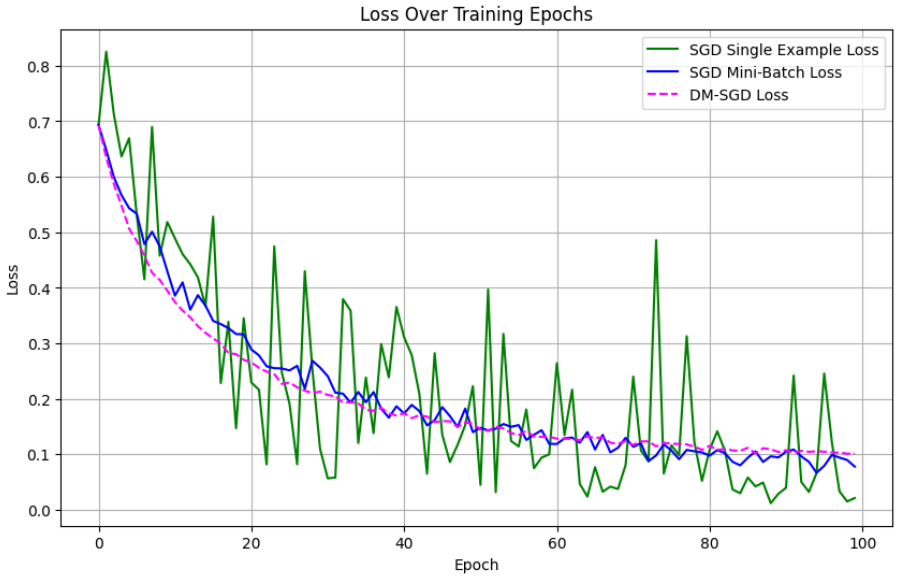

Figure 18: Logistic Regression Loss Over Training Epochs

It is well known that stochastic gradient descent with one example is very noisy, and we can observe this in the variance in the graph above. We can also see that SGD Mini-Batch loss is much more stable but still exhibits variance. It is very interesting to see that DM-SGD is much more stable than the other two gradient descent algorithms. Furthermore, we can see that in the early iterations that DM-SGD performs slightly better than SGD Mini-batch. Overtime, both SGD Mini-batch loss and DM-SGD loss converge. It seems from this synthetic data set up that it does seem to be the case that DM-SGD is more stable and has less variance.

We have now provided another example other than using logistic regression. For the next test we have used the MNIST Data Set but have restricted our self to only 1000 samples for the first result and only binary classification of digits 0 and 1. We have created a simple Neural Net Architecture which is a simple feed-forward neural network with an input layer taking vectors of size 20, a hidden layer with 10 neurons using ReLU activation, and an output layer with a single neuron using sigmoid activation for binary classification. Here is the result in the same format as above:

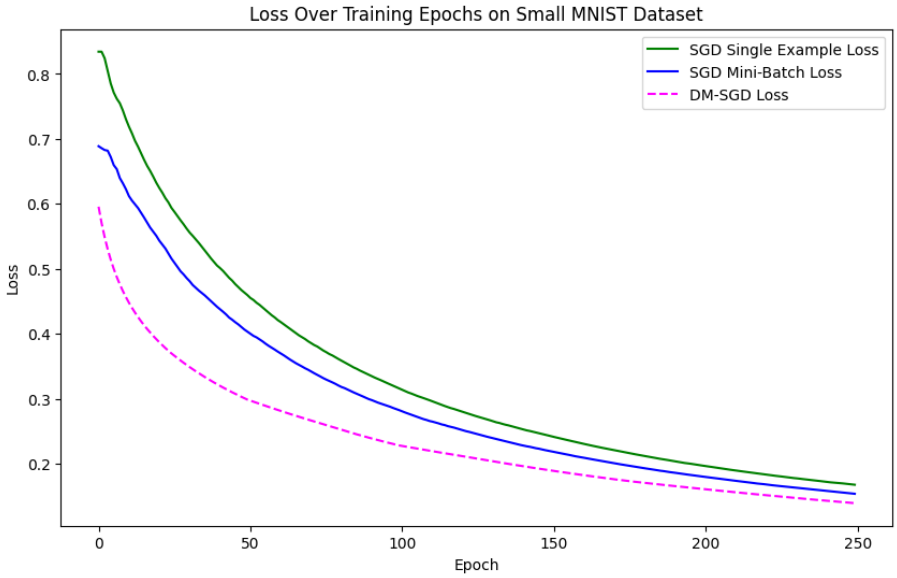

Figure 19: Neural Network Loss Over Training Epochs

We can clearly see in the example above that the DM-SGD loss clearly has less variance and is far better at learning than the other two gradient descent algorithms. In 2019 TensorFlow, an open-source machine learning framework developed by Google, added determinantal point processes (DPPs) to their package. The algorithm implemented was based of the eigendecomposition of the DPP kernel L shown in Algorithm 1.

We have decided to use the sampling algorithm from TensorFlow as well for our gradient descent and have compared the results. Because the sampling algorithm implemented in TensorFlow is not efficient we have decided to train on 500 synthetic samples, with same 20 features, and two classes. Here are the results for the training time of the different methods (see Table 7). We know that Algorithm 1 is $O(n^3)$ and is probably the reason why dpp-sgd using TensorFlow has a higher training time compared to dm-sgd.

Method	Training Time (seconds)
sgd-single	6.75
sgd	9.10
dm-sgd	6.84
dpp-sgd (TensorFlow)	1342.13

Table 7: Training times for different methods.

We have plotted the loss over the training epochs (see Figure 20).

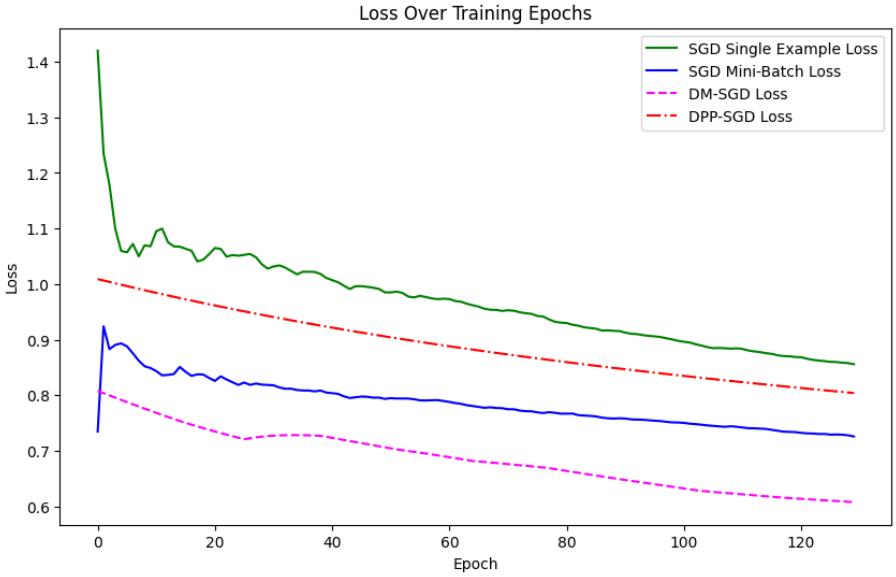

Figure 20: Neural Network Loss Over Training Epochs

For much more complicated models where the data lies in a higher dimensional space, computing the similarity matrix K using the Gaussian-Kernel will be too computationally expensive to justify the trade-off for more stability during training. There are numerous ways to construct a kernel based of a feature function (see [Kan13] for details).

5.3 Clustering

Let us first provide a high level overview of one of the most common clustering algorithms called K-means clustering. K-means clustering is an unsupervised learning algorithm where we want to partition the data into K clusters. Let us suppose we have a bunch of data points in some space (let us stick to $\mathbb{R}^2$ for now). We will initialize K centroids randomly based on the data. We will then assign every data point to the nearest cluster – this will form K clusters. We will then update the centroids based on the mean of the new data points.

One application of K-means clustering is in content recommendation. Once the clusters are established we can recommend content to users based on the preference of other users in the clusters. Here is the loss function we aim to minimize in K-means clustering:

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - c_j \right\|^2$$

The K-means clustering algorithm minimizes the objective function J, which is given by $J = \sum_{j=1}^{k} \sum_{i=1}^{n} \|x_i^{(j)} - c_j\|^2$, where n is the number of samples, k is the number of clusters, $x_i^{(j)}$ is the i-th sample in the j-th cluster, and c_j is the centroid of the j-th cluster. We have provided the psudeo-code for K-means Clustering in Algorithm 5. [457]

Algorithm 5 K-Means Clustering

Require: Data points $\{x_i\}_{i=1}^n$, number of clusters k
Ensure: Cluster assignments $C_1, \ldots, C_k$
Initialize cluster centroids $C_1, \ldots, C_k$ randomly
repeat
    for $\ell = 1$ to k **do**
        Compute centroid $\mu_\ell = \frac{1}{|C_\ell|} \sum_{i \in C_\ell} x_i$
    end for
    for each data point x_i **do**
        Assign x_i to the cluster with the closest centroid
    end for
    Check if cluster assignments have changed
until cluster assignments do not change **return** $C_1, \ldots, C_k$

It is an easy proof to see that the K-means algorithm converges to a local optimum. In K-means clustering K is a hyper-parameter. One of the most common ways of selecting K is to find the knee as seen in Figure 21.

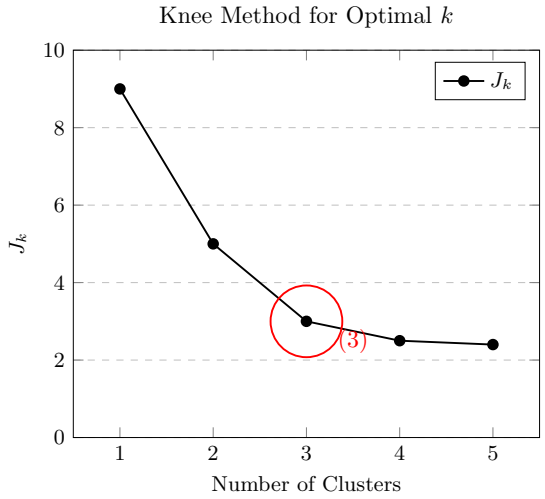

Figure 21: Number of clusters based on "knee"

Already we can start to see how DPPs might play a role in helping us select the initial clusters in Algorithm. Because DPPs can help us sample diverse points, sampling from a DPP may lead to better initialization of clusters. [Kan13] A simple approach would be to follow a similar method as done in section 5.2. However, because DPPs sample diverse points one of the main problems is that if our training data has outliers we may select those and misfit our model. Therefore, to prevent this we add a penalizer term that ensures that we don't select outliers (see the paper for more details).

5.4 GDPP

Let us now look at a paper published in 2019 titled "GDPP: Learning Diverse Generations using Determinantal Point Processes." [Elf+19] We will outline the high level overview of the paper and how DPPs are being used to prevent mode collapse in Generative adversarial networks (GANs).

Let us first give a high-level overview of what GANs are. GANs are a way to create new data that resemble training data – in the context of image generation we can imagine that the training data is many facial images and the GAN will create a realistic looking face that is not in the training set. GANs have two main components: a generator and a discriminator. The goal of the generator is to fool the discriminator, and the goal of the discriminator is to differentiate between data from the data set and generated data. Through this adversarial procedure the generator becomes better and better at generating more realistic images. However, one of the main problems with GANs is that sometimes the generator focuses on a few points that fool the discriminator instead of fully learning the underlying distribution of the training data – this is called mode collapse.

The paper proposes using a DPP to model the diversity of the data samples. The fundamental idea of the paper was to include a penalty term denoted the Generative Determinantal Point Processes (GDPP) loss – the goal of this loss is to generate data is is similar to the diversity of the real data. They do this by generating a DPP kernel for both the real data and the generated data and to compare the two kernels. The eigenvalues and the eigenvectors should capture the overall structure of both the kernels. The end goal then becomes to learn a fake diversity kernel L_{S_B} that is close to the real diversity kernel L_{D_B}. The loss is then defined as the following:

$$\mathcal{L}_{DPP} = \mathcal{L}_m + \mathcal{L}_s = \sum_i \left\| \lambda^i_{real} - \lambda^i_{fake} \right\|^2 - \sum_i \hat{\lambda}^i_{real} \cos(\mathbf{v}^i_{real}, \mathbf{v}^i_{fake})$$

where λ^i_{fake} and λ^i_{real} are the i^{th} eigenvalues of L_{D_B} and L_{S_B} respectively.

Facebook research's GAN toolbox for researchers and developers has the ability to apply the GDPP loss to models.

5.5 DPPMask

We saw above that the main attraction of DPPs in machine learning is in their ability to draw samples in a diversified way. We will look at a recent paper published in 2023 titled "DPPMask: Masked Image Modeling with Determinantal Point Processes." [Xu+23].

The application of DPPMask is in image generation for an unsupervised task – i.e. let's suppose we are just given images without labels, our goal is to create "new" unseen images. Let us suppose we have many images – one idea can be to mask certain parts of the image and then to try to recover the original image. Once a model is learned to recreate the image given the masked image it can be used for various other tasks such as image generation. One of the biggest problems with masking is that it can drop important aspects of an image – this is called the misalignment problem (see Figure 22).

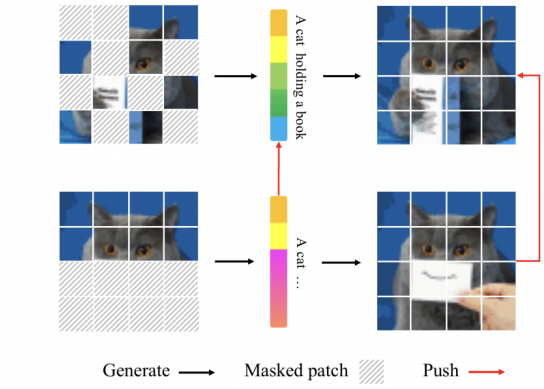

Figure 1: Illustration of the misalignment problem in MIM. The model generates predictions that differ plausibly from the original image, while the original image has still been imposed as supervision, leading to an unreasonable high loss.

Figure 22: Misalignment Problem from DPPMask paper (image from [Xu+23])

The paper proposes using DPPs because "DPPs will compute the distance of each patch, and select patches that are dissimilar from the selected subset. This process makes the network focus on the patches with more representative information." [Xu+23] The paper shows that using a DPP can capture the important semantic information in the image as compared to using random sampling. Here is an image that shows the areas that the DPP sampling picked that results in a more accurate reconstruction of the image (see Figure 23).

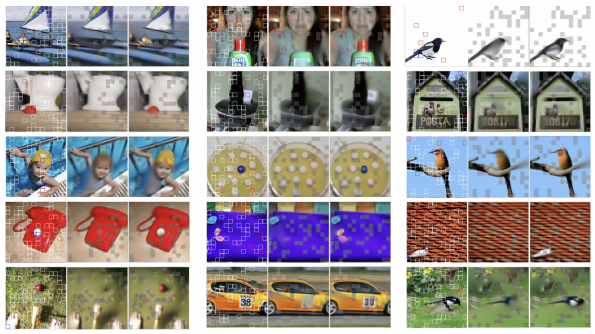

Figure 6: Comparison of DPPs sampling and random sampling, each triplet indicates the original image (right), reconstruction result with random sampling (middle), and DPPs sampling (left). The white boxes represent patches selected by both random and DPPs, while red boxes are for random sampling and blue boxes are for DPPs sampling. The threshold τ is set to 0.8, best views with zoom-in.

Figure 23: DPP vs. Random Sampling DPPMask paper (image from [Xu+23])

6 Charlier ensemble

6.1 Overview

Now that we have looked at a simple example above and showed how to construct a kernel, sample from the kernel, and create an empirical sampler to compare the distributions, we will now look at another well known DPP called the Charlier ensemble. [Joh04]

The Charlie ensemble arises from the problem of the longest weakly increasing subsequence in a random word. The random word problem is the following: we are given a *word* of length N on M letters where $M, N \geq 1$. We have a map $w : \{1, ..., N\} \to \{1, ..., M\}$. We let $\mathbb{P}_{W,M,N}$ be the uniform probability distribution on $W_{M,N}$ where all M^N words have the same probability. We call a weekly increasing subsequence of w a subsequence $w(i_1), ...w(i_m)$ such that $i_1 < ... < i_m$ and $w(i_1) \leq ... \leq w(i_m)$ and we denote $L(w)$ to be the length of the largest weakly increasing subsequence in w.

Here is an example to illustrate what the largest weakly increasing subsequence would be: given a word example word $= [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]$ the sequence returned would be $[3, 4, 5, 5, 5]$. The length of the largest weakly increasing subsequence would be 5.

At a high-level, Johansson proved that the following two proprieties are equivalent. Let us suppose we are given a kernel – we will call this the Charlier Kernel (we will explain how to construct this kernel in section 6.2.2). We can then use our DPP sampler from above and the distribution of largest index of the sample from the DPP will be the same as the distribution of the largest weakly increasing subsequence. The second method involves using Poissonization – so let us provide that definition.

Definition 6.1. Poissonization: If the goal is to solve a problem that has something of a fixed length or fixed size n and if the system is not independent it can be very hard to solve such a problem. But if we use Poisson distribution we can approximate n and get properties that are easier to use.

Here is the poissonization of the longest increasing sequence. First we will select the length of the random word N from Poisson(α). We then sample a word of length N where the letters can be from $\{1...M\}$, We then compute the largest weakly increasing subsequence length.

Johansson proved the following:

$$\mathbb{P}_{W,M,N}[L(w) \leq t] = \mathbb{P}_{Ch,M,N}[\lambda_1 \leq t]$$

6.2 Empirical Implementation

Let us now provide the distribution of the poissonized process. We have set the following parameters $\alpha = 5, M = 10, samples = 1000000$.

Algorithm 6 Generate and Analyze Subsequences

Require: α, M, number of trials
Ensure: List of lengths of the longest weakly increasing subsequences

```
Initialize results as an empty list
for i = 1 to trials do
    N ← Poisson(α)                          ▷ Generate a random integer
    if N > 0 then
        Generate a random word of length N with each element from 1 to M
        Compute the length of the longest weakly increasing subsequence (llis)
        Append llis to results
    end if
end for
return results
```

We have plotted the distribution of the length of the largest weakly increasing subsequence in Figure 24.

Let us now give an overview of the Charlier Kernel and how one can construct the kernel to sample from. But to understand this first let us take a little dive into orthogonal polynomials.

6.2.1 Orthogonal Polynomials

To understand the Kernel for the Charlier Ensemble we must first discuss Orthogonal Polynomials. In the context of vectors $\{x, y\}$ we say that the set of vectors is orthogonal if $x \cdot y = 0$. we say that two functions $f_i(x)$ and $f_j(x)$ are orthogonal on the interval $[a, b]$ if: $\int_a^b f_i(x) f_j(x) dx = 0$.

Definition 6.2. Orthogonal Polynomials: A set of polynomials is orthonormal if:

$$\int_a^b w(x) p_m(x) f_n(x) dx = c_n \delta_{mn}.$$

The interval $[a, b]$ where $\delta_{mn} = 1$ if $m = n$ and 0 if $m \neq n$ and $w(x)$ is a weighting function.

Definition 6.3. Discrete Orthogonal Polynomials: A set of polynomials is orthonormal if:

$$\sum_{x \in S} w(x) p_m(x) f_n(x) dx = c_n \delta_{mn}.$$

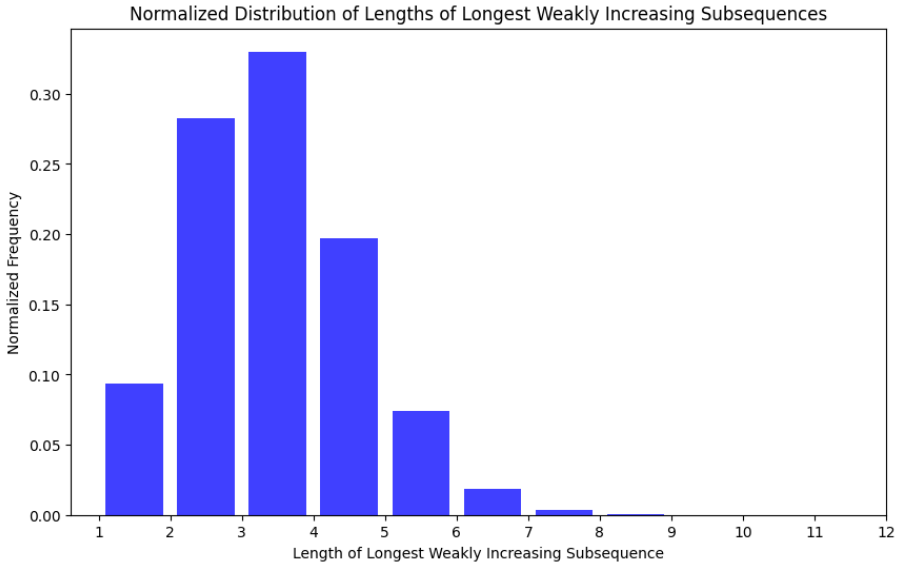

Figure 24: Distribution of the Length of the Largest Weakly Increasing Subsequence

The interval $[a, b]$ where $\delta_{mn} = 1$ if $m = n$ and 0 if $m \neq n$ and $w(x)$ is a weighting function.

Another key component of the Charlier Kernel is Charlier polynomials which we define as the following:

Charlier polynomials: Are a family of discrete orthogonal polynomials and are defined as:

$$C_n(x_i; \mu) = 2F_0(-n, -x_i; -1/\mu) = (-1)^n n! L_n^{(-1-x)}\left(\frac{-1}{\mu}\right)$$

Where L is the generalized Laguerre polynomials. Charlier Polynomials satisfy the following orthogonality relation:

$$\sum_{x=0}^{\infty} \frac{\mu^x}{x!} C_n(x; \mu) C_m(x; \mu) = \mu^{-n} e^{\mu} n! \delta_{nm}, \quad \mu > 0.$$

6.2.2 Charlier Kernel

Now that we have an understanding of Charlier polynomial. We first need to define the Bessel functions as the following:

$$w_a(x) = e^{-a} \frac{a^x}{x!}, \quad x \in \mathbb{N}, \quad a > 0.$$

We define the kernel as the following for $x \neq y$ [Joh04]:

$$K^{\alpha}_{Ch,M}(x,y) = \sqrt{\alpha}\left(\frac{C_M\left(x;\frac{\alpha}{M}\right)C_{M-1}\left(y;\frac{\alpha}{M}\right)-C_{M-1}\left(x;\frac{\alpha}{M}\right)C_M\left(y;\frac{\alpha}{M}\right)}{x-y}\right) \times\sqrt{w_{\alpha/M}(x)w_{\alpha/M}(y)}$$

and for $x = y$ we have:

$$K^{\alpha}_{\mathrm{Ch,M}}(x,x) = \sqrt{\alpha}w_{\alpha/M}(x)\Big[C'_M\left(x;\tfrac{\alpha}{M}\right)C_{M-1}\left(x;\tfrac{\alpha}{M}\right) -C_{M-1}\left(x;\tfrac{\alpha}{M}\right)C'_M\left(x;\tfrac{\alpha}{M}\right)\Big]$$

A future goal of this paper would be to construct this kernel. We can then sample from this kernel using Algorithm 2. Once we have samples can compare similar metrics as we did in the descents in random sequence example.

7 Summary

In this paper we provided an overview of DPPs and highlighted the important properties that they contain. Some of these properties included the ability to compute restrictions, complements, and conditional probabilities in a very efficient manner using properties of determinants of kernels. We then looked at popular sampling algorithms and showed how one can verify a DPP sampler implementation using properties of DPPs.

We were able to verify the similarity kernel of the Descents in Random Sequence by comparing metrics such as the distribution of indexes, n-th largest index etc... to an empirical implementation. According to our literature review there have been no formal verifications of checking the Descents in Random Sequence similarity kernel through simulation.

We then showed how DPPs can be beneficial in machine learning applications. We started by showing how Diversified Mini-Batch SGD (DM-SGD) can lead to more stable convergence as compared to other standard gradient descent methods. We then looked at recent applications of DPPs in deep learning applications such as in GANs and image modeling.

Lastly, we provided a brief introduction to Charlier enembles and the problem of longest weakly increasing subsequence. We showed how one could construct a kernel and also provided some preliminary results from an empirical implementation.

8 Code

The code used for the experiments above can be found in the following GitHub repo.

References

[Joh04] Kurt Johansson. *Discrete orthogonal polynomial ensembles and the Plancherel measure*. 2004. arXiv: math/9906120 [math.CO].

[BDF09] Alexei Borodin, Persi Diaconis, and Jason Fulman. *On adding a list of numbers (and other one-dependent determinantal processes)*. 2009. arXiv: 0904.3740 [math.PR].

[Kul12] Alex Kulesza. "Determinantal Point Processes for Machine Learning". In: *Foundations and Trends in Machine Learning* 5.2–3 (2012), pp. 123–286. ISSN: 1935-8245. DOI: 10.1561/2200000044. URL: http://dx.doi.org/10.1561/2200000044.

[AFM13] Mark Adler, Patrik L. Ferrari, and Pierre van Moerbeke. "Nonintersecting random walks in the neighborhood of a symmetric tacnode". In: *The Annals of Probability* 41.4 (July 2013). ISSN: 0091-1798. DOI: 10.1214/11-aop726. URL: http://dx.doi.org/10.1214/11-AOP726.

[Kan13] Byungkon Kang. "Fast determinantal point process sampling with application to clustering". In: *Advances in Neural Information Processing Systems* 26 (2013).

[Gup15] Swati Gupta. *1 Determinantal Point Processes*. Sept. 2015. URL: https://people.csail.mit.edu/stefje/fall15/notes_lecture21.pdf.

[ZKM17] Cheng Zhang, Hedvig Kjellstrom, and Stephan Mandt. *Determinantal Point Processes for Mini-Batch Diversification*. 2017. arXiv: 1705.00607 [cs.LG].

[Bar18] Simon Barthelmé. *Determinantal Point Processes: a quick introduction*. Nov. 2018. URL: https://barthesi.gricad-pages.univ-grenoble-alpes.fr/personal-website/dpps/2018-26-11-dpps_intro/.

[Elf+19] Mohamed Elfeki et al. *GDPP: Learning Diverse Generations Using Determinantal Point Process*. 2019. arXiv: 1812.00068 [cs.LG].

[Pou20] Jack Poulson. "High-performance sampling of generic determinantal point processes". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 378.2166 (2020). ISSN: 1471-2962. DOI: 10.1098/rsta.2019.0059. URL: http://dx.doi.org/10.1098/rsta.2019.0059.

[Xu+23] Junde Xu et al. *DPPMask: Masked Image Modeling with Determinantal Point Processes*. 2023. arXiv: 2303.12736 [cs.CV].

[457] Cornell CS 4/5780. *K-means*. URL: https://www.cs.cornell.edu/courses/cs4780/2022sp/notes/LectureNotes04.html.