UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
ESCUELA DE POSTGRADO Y EDUCACIÓN CONTINUA
DEPARTAMENTO DE INGENIERÍA MATEMÁTICA

ESTRATEGIAS DE SELECCIÓN DE MINI BATCHES UTILIZANDO PROCESOS PUNTUALES DETERMINANTALES PARA EL ENTRENAMIENTO DE REDES NEURONALES MEDIANTE DESCENSO DE GRADIENTE ESTOCÁSTICO

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIA DE DATOS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL MATEMÁTICO

NELSON BRUNO ANDRÉS MORENO CABAÑAS

PROFESOR GUÍA:
FELIPE TOBAR HENRÍQUEZ

MIEMBROS DE LA COMISIÓN:
DANIEL REMENIK ZISIS
JOAQUÍN FONTBONA TORRES

Este trabajo ha sido parcialmente financiado por:
Fondecyt Regular Nº 1210606

SANTIAGO DE CHILE
2023

RESUMEN DE LA TESIS PARA OPTAR Al GRADO DE
MAGÍSTER EN CIENCIA DE DATOS Y AL TÍTULO PROFESIONAL DE INGENIERO CIVIL MATEMÁTICO
POR: NELSON BRUNO ANDRÉS MORENO CABAÑAS
FECHA: 2023
PROF. GUÍA: FELIPE TOBAR HENRÍQUEZ

ESTRATEGIAS DE SELECCIÓN DE MINI BATCHES UTILIZANDO PROCESOS PUNTUALES DETERMINANTALES PARA EL ENTRENAMIENTO DE REDES NEURONALES MEDIANTE DESCENSO DE GRADIENTE ESTOCÁSTICO

El aprendizaje profundo o *Deep Learning* es una de las ramas del aprendizaje de máquinas que más desarrollo y mejoras ha tenido en las últimas décadas. Los constantes esfuerzos de la comunidad de *Machine Learning* por mejorar las técnicas de entrenamiento han abierto al paso a la utilización de herramientas cada vez más complejas y sofisticadas.

En linea con lo anterior, esta tesis propone la utilización de tres metodologías para el entrenamiento de redes neuronales utilizando un proceso repulsivo conocido como Proceso Puntual Determinantal (DPP) cuya función es la de permitir que los *mini batch* sampleados en cada iteración del entrenamiento tengan la mayor diversidad posible según alguna métrica de distancia. Esta métrica es definida como una distancia euclidiana aplicada sobre una representación de baja dimensionalidad de los datos obtenida a partir del entrenamiento previo de un *Autoencoder* o una red *Oneshot*.

La primera arquitectura denominada *Fast DPP* resuelve el problema del alto costo computacional que requiere utilizar un DPP filtrando aquellos ejemplos poco relevantes. La segunda arquitectura, *Mixed DPP*, combina el entrenamiento estándar (sampleo uniforme) con una inicialización mediante *Fast DPP*. Finalmente, la tercera arquitectura plantea el entrenamiento en paralelo de un *Autoencoder* necesario para la definición de la métrica y la red encargada de resolver el problema principal.

Se prueban las arquitecturas en un problema de clasificación binaria con un *dataset* artificial en dos dimensiones y uno de clasificación multiclase con *Fashion MNIST*. Los resultados muestran que la red *Fast DPP* tiene un mejor rendimiento en los primeros 30 segundos de entrenamiento que un método estándar (*baseline*) en el problema de clasificación multiclase y la red *Mixed DPP* alcanza un rendimiento similar al método estándar pero en menor tiempo.

La importancia de este trabajo radica en la apertura de nuevas posibilidades de entrenamiento con sampleo activo mediante un proceso repulsivo tan estudiado como los DPP y además, aporta en el estudio de métricas de distancia en problemas de alta dimensionalidad con un enfoque en el entrenamiento de redes neuronales.

Para mi madre, mi hermana y mi padre, que en paz descanse.

Agradecimientos

En primer lugar, agradezco a mi familia, mi hermana Nicole que siempre me ha brindado su apoyo incondicional y ha sido parte de todas las decisiones de mi vida, estoy tremendamente agradecido de ser tu hermano. A mi sobrino Santino por su cariño y alegría, espero que este trabajo te motive a seguir una linda carrera de Ingeniería. A mi madre Gloria, que ha sido y seguirá siendo el pilar fundamental de mi vida, sin ti, no sería ni la mitad de la persona que soy el día de hoy y es por eso que te dedico mis logros, mis sueños y metas, muchísimas gracias. Finalmente, agradezco eternamente a mi padre que lamentablemente no podrá ver la culminación de mi carrera, pero de la que sé, estaría tremendamente orgulloso. Viejito, donde quiera que te encuentres, esto es para ti.

Me gustaría agradecer al profesor Felipe Tobar por su guía durante el magíster y todas las instancias en que nos encontramos durante mi estadía en la universidad. Definitivamente, una persona comprometida con la formación de los estudiantes y que no dudó en ayudarme ni una sola vez, mil gracias. Agradezco también a todos los tesistas a cargo de Felipe, sin su aporte, el desarrollo de esta tesis nunca pudo haber sido llevado a cabo. Agradezco además al proyecto Fondecyt regular 1210606 por haber financiado el costo del magíster durante el año 2022.

Agradezco a mis amigos de las U por hacer más ameno el paso por la carrera y por todos los buenos momentos compartidos. Agradezco también a mis amigos del colegio por estar siempre presentes y apoyándome en lo que necesitara, ¡son lo más grande!.

Finalmente, me gustaría agradecer a la persona que más cerca ha visto este proceso, mi pareja Karin. Sin su amor incondicional, su paciencia y la motivación que me entregó, jamás podría haber completado esta tesis. Estoy tremendamente agradecido por todas la veces que me levantaste cuando no me vi capaz, marcaste una época en mi vida que jamás olvidaré y es por esto que comparto este logro contigo, te amo.

Tabla de Contenido

1. Introducción **1**
- 1.1. Motivación . . . 1
- 1.2. Hipótesis . . . 2
- 1.3. Objetivos Generales . . . 2
- 1.4. Objetivos Particulares . . . 2
- 1.5. Contribuciones . . . 2
- 1.6. Estructura de la Tesis . . . 3

2. Marco Teórico **4**
- 2.1. Proceso Puntual . . . 4
  - 2.1.1. Procesos Puntuales Determinantales . . . 4
    - 2.1.1.1. Definición . . . 4
    - 2.1.1.2. Aplicaciones . . . 5
    - 2.1.1.3. Sampling . . . 5
    - 2.1.1.4. k-DPP . . . 8
  - 2.1.2. Poisson Disk Sampling . . . 8
- 2.2. Redes Neuronales . . . 10
  - 2.2.1. Descenso del Gradiente Estocástico . . . 11
    - 2.2.1.1. Introducción . . . 11
    - 2.2.1.2. Descenso del Gradiente Estocástico: Mini-Batch . . . 12
    - 2.2.1.3. Épocas de Entrenamiento . . . 12
  - 2.2.2. Redes Neuronales Convolucionales . . . 12
  - 2.2.3. Autoencoders . . . 13
  - 2.2.4. One Shot Learning . . . 13

3. Estado del Arte **15**
- 3.1. Introducción . . . 15
- 3.2. Active Mini Batch Sampling . . . 16
- 3.3. Oportunidad de Mejora . . . 17

4. Metodología **18**
- 4.1. Métodos Propuestos . . . 18
  - 4.1.1. Fast DPP . . . 18
  - 4.1.2. Mixed DPP . . . 19
  - 4.1.3. DPP NET . . . 20
- 4.2. Definición de Métricas . . . 21
  - 4.2.1. Kernel RBF . . . 21
  - 4.2.2. Minkowski de orden p . . . 22

4.2.3. Distancia Coseno . . . 23
4.2.4. Autoencoder: Representación Latente . . . 23
4.2.5. Learned Metric: One Shot Learning . . . 23
4.3. Métrica de Desempeño . . . 24
4.4. Conjuntos de Datos . . . 24
4.4.1. Clasificación Binaria - Baja Dimensionalidad . . . 24
4.4.2. Clasificación Multiclase - Alta Dimensionalidad . . . 25
4.5. Experimentos . . . 25
4.5.1. Sampling de la Clase Minoritaria . . . 26
4.5.2. Fast DPP . . . 26
4.5.3. Mixed DPP . . . 26
4.5.4. DPP NET . . . 26
4.6. Arquitecturas . . . 27
4.6.1. Baseline . . . 27
4.6.2. Autoencoder . . . 27
4.6.3. Oneshot . . . 28
4.6.4. DPP NET . . . 29

5. Resultados y Análisis **30**
5.1. Muestreo Clase Minoritaria . . . 30
5.2. Fast DPP . . . 33
5.3. Mixed DPP . . . 37
5.4. DPP NET . . . 40

6. Conclusiones **42**
6.1. Comentarios . . . 42
6.2. Trabajo a Futuro . . . 43

Bibliografía **44**

Anexos **46**
A. Acrónimos . . . 46
B. Teoremas y Demostraciones . . . 47
C. Parámetros de Arquitecturas . . . 51
D. Especificaciones de Hardware . . . 52

Índice de Tablas

4.1. Ejemplos por Cluster - Clasificación Binaria. Total = 1800. 25
4.2. Conjunto de Entrenamiento - Clasificación Multiclase. Total = 16000. 26
5.1. Conjunto de Validación - Clasificación Multiclase. Total = 1600. 31
C.1. Parámetros Arquitectura Baseline Binario por Experimento. 51
C.2. Parámetros Arquitectura Baseline Multiclase por Experimento. 51
C.3. Parámetros Arquitectura Autoencoder por Experimento. 51
C.4. Parámetros Oneshot por Experimento. 52
C.5. Parámetros DPP NET por experimento. 52

Índice de Ilustraciones

2.1. Muestreo de un DPP en una y dos dimensiones. Cada ejemplo sampleado disminuye la probabilidad de muestrear otro en una vecindad cercana (Kulesza 2012) . . . 7
2.2. Sampling de un PDS en dos dimensiones. Cada elemento muestreado crea un área de rechazo para los siguientes ejemplos (Zhang et al. 2018). . . . 9
2.3. Representación del Perceptrón para un input $x \in \mathbb{R}^4$ y pesos $w \in \mathbb{R}^4$. . . . 10
2.4. MLP con 2 capas ocultas de pesos $W^{(i)}$ y funciones de activación $f^{(i)}$. Además, incluye una capa de output con función de activación g y pesos U. No se agrega un vector de *bias*. . . . 10
2.5. Representación de distintos tipos de descensos del gradiente según el tamaño del *Batch* (Wash 2022) [9]. . . . 11
2.6. Autoencoder, el cuello de botella en la mitad de la estructura codifica una representación de menor dimensionalidad del *input* (Zhang 2018). . . . 13
2.7. Arquitectura One Shot Learning, los *batches* de entrenamiento se componen de tuplas de ejemplos que son entregados en cada sección de la red. . . . 14
2.8. Triplet Loss: La imagen *Anchor* se compara con una imagen de misma etiqueta (*Positive*) y una de distinta etiqueta (*Negative*) (Ghandi 2018) [15]. . . . 14
4.1. Fast DPP: El diagrama muestra el proceso de extracción de un *Mini Batch* desde el conjunto de entrenamiento y su posterior muestreo con DPP. . . . 18
4.2. DPP NET: El diagrama muestra una arquitectura que incluye el muestreo mediante un DPP, el *Autoencoder* encargado de obtener la representación de baja dimensionalidad de los datos y la subred encargada de la predicción de la etiqueta del problema de clasificación. . . . 21
4.3. Bola Unitaria para distintos valores de p. . . . 23
4.4. Dataset de Clasificación Binaria con 4 clusters artificiales y desbalanceados en forma de luna. . . . 24
4.5. Ejemplos de las 10 clases presentes en el conjunto de datos Fashion-MNIST. . 25
4.6. Ejemplos de la Clase 3 del conjunto de datos Fashion - MNIST. . . . 25
4.7. Baseline Clasificación Binaria: Esta arquitectura se utiliza como base para realizar comparaciones de las metodologías propuestas en los problemas de clasificación binaria. . . . 27
4.8. Baseline Clasificación Multiclase: Esta arquitectura se utiliza como base para realizar comparaciones de las metodologías propuestas en los problemas de clasificación multiclase (Creado con NN-SVG de Alex Lenail). . . . 27
4.9. Autoencoder: Esta arquitectura se utiliza para obtener la representación de baja dimensionalidad de los datos. . . . 28
4.10. Oneshot: Esta arquitectura se utiliza para el aprendizaje de la métrica de distancia necesaria para un DPP. . . . 28

4.11. Baseline y DPP NET: Esta figura muestra la arquitectura del baseline con el que se compararán los resultados de DPP NET. 29
5.1. Comparación de un muestreo de tamaño 32 sobre el dataset de clasificación binaria en 2 dimensiones utilizando una estrategia uniforme y una con DPP. . 30
5.2. Razón del total promedio ($n = 100$) de ejemplos sampleados en el dataset de 2 dimensiones. Las barras azules representan las proporciones para un sampling uniforme y las barras naranjas para el sampling con DPP, las barras negras verticales corresponden a la varianza de los datos. 31
5.3. Visualización en 2 dimensiones de Fashion-MNIST mediante la técnica de reducción dimensionalidad UMAP utilizada sobre la representación latente entregada por el *Autoencoder* y la red *Oneshot* luego de ser entrenados. 32
5.4. Razón del total promedio ($n = 50$) de ejemplos muestreados en el dataset Fashion-MNIST. La barra azul corresponde a un sampleo uniforme y el resto de barras corresponden a distintas métricas utilizadas para el sampleo con un DPP. 32
5.5. Razón del total normalizada promedio (n = 50) de ejemplos muestreados en el dataset Fashion-MNIST. La normalización se realizó en función del desbalance del dataset. 33
5.6. Primeras iteraciones SGD en el problema de clasificación binaria. Las zonas de color celeste y anaranjado delimitan la región de decisión del problema. 34
5.7. Primeras iteraciones Fast DPP en el problema de clasificación binaria. Las zonas de color celeste y anaranjado delimitan la región de decisión del problema. . . 34
5.8. Última iteración de las metodologías SGD y DPP si no se descartan ejemplos como lo propone Fast DPP. 34
5.9. Comparación de resultados de las metodologías Baseline y DPP según la métrica de *loss* en el dataset de clasificación binaria promediando 30 iteraciones del experimento y $M = 100$. 35
5.10. Comparación de resultados de las metodologías Baseline y DPP según la métrica de *loss* en el dataset de clasificación multiclase promediando 5 iteraciones del experimento y $M = 400$. 36
5.11. Comparación de resultados de las metodologías Baseline, Fast DPP, Mixed DPP y Reversed Mixed DPP según la métrica de *loss* en el dataset de clasificación binario promediando 10 iteraciones del experimento y $M = 100$. 37
5.12. Comparación de los resultados de las metodologías Baseline y Mixed DPP según la métrica de *loss* en el dataset de clasificación multiclase promediando 5 iteraciones del experimento y $M = 400$. 38
5.13. Comparación de los resultados de las metodologías Baseline y Reversed Mixed DPP según la métrica de *loss* en el dataset de clasificación multiclase promediando 5 iteraciones del experimento y $M = 400$. 39
5.14. Tiempo de entrenamiento de las redes Oneshot y Autoencoder en el dataset de lasificación multiclase. 39
5.15. Comparación de los resultados de las metodologías Baseline y DPP NET según la métrica de *loss* (en la subred encargada de clasificar) en el dataset de clasificación multiclase promediando 5 iteraciones del experimento y $M = 400$. 40
5.16. Resultados del entrenamiento del *Autoencoder* en la metodología *DPP NET* según la métrica de *loss* para el problema de clasificación multiclase promediando 5 iteraciones del experimento y $M = 400$. 41

Capítulo 1

Introducción

1.1. Motivación

El aprendizaje profundo o *Deep Learning* es una de las ramas del aprendizaje de máquinas que más desarrollo y mejoras ha tenido en las últimas décadas y no solo por los avances en *hardware* que permite el entrenamiento de redes más profundas y complejas, sino que también por los constantes esfuerzos de la comunidad de *Machine Learning* en probar técnicas más sofisticadas en el entrenamiento y construcción de estas redes.

Dada la complejidad de las redes neuronales y la amplia variedad de elementos que la conforman, no es sorprendente que los recientes avances en el área sean debidos a modificaciones marginales de sus componentes que en conjunto suponen grandes cambios en el rendimiento de la red. Es por esto que surge la necesidad de continuar añadiendo mejoras a estas componentes haciendo uso de la intuición y las herramientas que la matemática proporciona.

En esta tesis, nos centraremos en la selección de los *batches* que se utilizan para entrenar redes neuronales mediante el descenso del gradiente estocástico. Actualmente y dado su ligero costo computacional, la elección del *batch* se realiza de manera aleatoria pero uniforme entre todos los datos y por tanto, al ser un sampleo de tamaño reducido (*mini batch*), no se logra capturar correctamente la distribución de los datos (añade mucho ruido) y tampoco da espacio a la selección activa y adaptativa.

Entre los esfuerzos por introducir métodos inteligentes de sampleo de *batches*, se encuentra el trabajo de *Zhang, C. et al.* [1] en el que se propone utilizar procesos repulsivos que permiten añadir el concepto de "diversidad" a las muestras de datos capturadas en cada iteración del entrenamiento de una red neuronal. Si bien el artículo muestra una mejora considerable en el rendimiento de la red mediante una reducción de la varianza del estimador del gradiente de la función de pérdida (*loss*) en el algoritmo de descenso de gradiente estocástico, se queda limitado al uso de un proceso repulsivo de bajo costo computacional pero altamente restrictivo (*Poisson Disk Sampling*) en el sentido de la repulsión entre los elementos.

Se propone entonces, utilizar un proceso repulsivo profundamente estudiado conocido como *Determinantal Point Process* que si bien es de mayor coste computacional, no es restrictivo en la repulsión de los elementos pues su enfoque es puramente probabilístico. Para ello, se buscarán formas de soslayar el problema del alto coste computacional mediante propuestas de arquitecturas que se entrenan únicamente con datos relevantes y no redundantes según el proceso repulsivo, compitiendo así con redes que miran todos los datos pero de forma ingenua.

Otro desafío importante de esta propuesta de entrenamiento radica en la diversidad de tipologías de datos con la que se trabajan, tales como imágenes, series de tiempo, videos, entre otras. Para establecer un proceso repulsivo, es necesario contar con una métrica de distancia (o similitud) que permita medir la repulsión entre pares de elementos y esto se vuelve complejo al tratar con ciertos tipos de datos. La búsqueda de una métrica transversal a los problemas y de fácil implementación es entonces otro de los objetivos que se buscará resolver dentro de esta tesis.

1.2. Hipótesis

Las hipótesis de este trabajo son:

1. La selección de *batches* mediante un proceso repulsivo como *Determinantal Point Process* en una arquitectura que no considera todos los datos, sino que entrena únicamente con los más relevantes, logra acelerar significativamente el entrenamiento inicial de una red neuronal. Una vez ajustada, esta red supera el rendimiento obtenido por una red entrenada mediante una estrategia de muestreo uniforme.

2. La utilización de redes que aprenden representaciones de baja dimensionalidad de los datos, o bien, que aprendan directamente una métrica de distancia, permite que en problemas de alta dimensión, un *Determinantal Point Process* muestree con mayor frecuencia los ejemplos de la clase minoritaria en los primeros *batch* del entrenamiento. Esto permite lograr una aproximación más precisa del problema de clasificación en las primeras iteraciones.

1.3. Objetivos Generales

El objetivo general de esta tesis es desarrollar un algoritmo de selección de *batches* que utilice un proceso puntual determinantal para el entrenamiento de una red neuronal mientras que se resuelve el problema del alto costo computacional y la selección de la métrica de distancia adecuada al tipo de problema.

1.4. Objetivos Particulares

- Crear arquitecturas que realicen el entrenamiento de una red neuronal sin utilizar todos los datos y que implementen un proceso puntual determinantal para la selección de *batches*.

- Comparar distintas métricas de distancia definidas sobre el conjunto de entrenamiento y su efecto como medida diversidad para el proceso puntual determinantal.

- Analizar cuantitativa y cualitativamente el efecto de utilizar un proceso repulsivo para el muestreo de *batches* en problemas de clasificación desbalanceados en baja y alta dimensionalidad.

1.5. Contribuciones

Las contribuciones de este trabajo se pueden resumir como:

- Se proponen 3 arquitecturas alternativas a las clásicas que admiten el uso de un proceso puntual determinantal para la selección de batches representativos y tal que su entrenamiento se realiza sin utilizar todos los datos. Se muestra experimentalmente que una de ellas, *Mixed DPP*, supera el rendimiento de una red convencional (con SGD sin selección activa de *batches*) en un dataset de alta dimensionalidad y alto coste de procesamiento (*Fashion MNIST*).

- Se proponen y discuten distintas alternativas de métricas de distancia que pueden ser utilizadas para samplear con un proceso repulsivo y cómo algunas de ellas son transversales a problemas con distintos tipos de datos.

- Se establece una guía para continuar el desarrollo de esta tesis mediante espacios de mejora, ya sea en optimización de los algoritmos presentados, propuestas de métricas de distancia y esquemas de entrenamiento.

1.6. Estructura de la Tesis

La tesis se divide en 6 capítulos siendo el primer capítulo la introducción presentada. A continuación, se presenta un capítulo con el marco teórico que permite dar una visión general de los algoritmos y estructuras que se utilizarán a lo largo de la tesis y más importante aún, la teoría que fundamenta los procesos puntuales determinantales que serán el foco de los experimentos posteriores. En el Capítulo 3, se hará una revisión de las principales iniciativas en la selección activa de *batches* y en particular, de una que estudia el uso de los procesos repulsivos para ello. Esta última da pie a los modelos propuestos y descritos en el Capítulo 4 junto a las métricas de distancia que se analizarán. El Capítulo 5 recopila los resultados de estos modelos en problemas de clasificación con datos de distinto tipo y dimensionalidad. Finalmente, el Capítulo 6 hará una síntesis de los resultados, experimentos y explicaciones que permitirán corroborar o refutar las hipótesis propuestas en el primer capítulo, además del trabajo a futuro y los espacios de mejora.

Capítulo 2

Marco Teórico

2.1. Proceso Puntual

Un proceso puntual es un proceso estocástico que es utilizado para modelar eventos que ocurren en intervalos aleatorios y relativos al eje del tiempo o del espacio. Estos, describen un subconjunto aleatorio de puntos en algún espacio $\mathcal{X}$ o la ocurrencia a lo largo del tiempo de eventos aleatorios secuenciales.

Si bien existe una definición más general de un proceso puntual, nos centraremos en el caso particular discreto y real. Sea $\mathcal{Y}$ un conjunto de puntos de la forma $\{1, \ldots, N\}$ y denotemos $N(\mathcal{Y})$ como el conjunto de todas las medidas puntuales sobre $\mathcal{Y}$. Se dirá que $\mathcal{P}$ es un proceso puntual sobre $\mathcal{Y}$ si es una medida de probabilidad sobre $N(\mathcal{Y})$ y lo denotaremos como $Y \sim PP(\mathcal{Y})$.

Existen múltiples procesos puntuales entre los que podemos destacar los procesos de *Poisson*, *Markov Random Fields*, procesos de *Cox*, entre otros.

2.1.1. Procesos Puntuales Determinantales

Los Procesos Puntuales Determinantales, desde ahora DPP, son modelos probabilísticos de repulsión (procesos repulsivos) que surgen naturalmente en mecánica cuántica y en teoría de matrices aleatorias y que, a diferencia de otros modelos utilizados como *Markov Random Fields*, estos proveen algoritmos eficientes y exactos para samplear, marginalizar, condicionar, entre otras tareas de inferencia [2].

2.1.1.1. Definición

De manera formal, un DPP es un proceso puntual estocástico cuya distribución de probabilidad queda completamente definida por el determinante de alguna función. En particular si $Y \sim PP(\mathcal{Y})$ acorde a $\mathcal{P}$ tal que

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

entonces diremos que $Y \sim DPP(\mathcal{Y})$. Aquí, K_A es la restricción a las columnas definidas por el conjunto A de una matriz $K \in \mathcal{M}_{N\times N}(\mathbb{R})$ definida positiva, simétrica y a valores propios acotados por 1 cuya indexación está definida por $\mathcal{Y}$ y se le conoce como kernel marginal. Una observación sencilla es que si $A = \{i\}$ un singleton, entonces

$$\mathcal{P}(i \in Y) = K_{ii},$$

y por tanto, las diagonales con números cercanos a 1 corresponden a elementos que pueden ser seleccionados con alta probabilidad. Por otro lado, si $A = \{i, j\}$, entonces

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

es decir, los elementos fuera de la diagonal representan correlaciones entre pares de elementos, entre mayor sea el valor de K_{ij}^2, menor será la probabilidad de que i, j aparezcan juntos. Finalmente, dado que K debe ser una matriz definida positiva, es posible probar que $K_{ij}^2 \leq K_{ii}K_{jj}$ y por tanto, lo anterior logra definir una probabilidad.

2.1.1.2. Aplicaciones

Existen numerosos casos de uso de los DPP, se mencionan algunos de ellos a continuación:

- **Valores propios de Matrices Aleatorias** [3][4]

  Sea M una matriz aleatoria tomando cada entrada independientemente de una distribución normal compleja.

  Los valores propios de M que forman un subconjunto del plano complejo, están distribuidos acorde a un DPP.

- **Random Walks No-Intersectantes** [5]

  Consideremos un conjunto de k caminatas aleatorias simples, independientes, enteras, de largo T. Si comienzan de $x_1^1, x_1^2, \ldots, x_1^k$, condicionadas a que no se intersecten y terminen en $x_T^1, x_T^2, \ldots, x_T^k$. Entonces para cada instante t, $x_t^1, x_t^2, \ldots, x_t^k$ distribuye como un DPP.

- **Aristas de Spanning Trees Aleatorios.** [6]

  Sea G un grafo arbitrario finito con N aristas y sea T un árbol generador escogido aleatoriamente del conjunto de todos los árboles generadores de G. Las aristas de T que forman un subconjunto de T distribuyen como un DPP.

- **Descensos en Secuencias Aleatorias.** [7]

  Dada una secuencia de N números escogidos de forma independiente y uniforme de un conjunto finito de números (del 1 al 9 por ejemplo), la posición en la secuencia donde tal número es menor que el escogido previamente forma un subconjunto de $\{2\ , \ldots, \mathrm{N}\}$ y distribuye como un DPP.

2.1.1.3. Sampling

Antes de definir el algoritmo de sampleo, es útil definir los DPP no a través de su kernel marginal K, y más bien por una matriz L que llamamos L-ensamblaje. Sea Y un conjunto dado e $\mathbf{Y}$ un conjunto aleatorio, la matriz de L-ensamblaje es tal que

$$\mathcal{P}_L(\mathbf{Y} = Y) \propto \det(L_Y),$$

con L_Y la restricción de L a los índices de Y. Lo anterior define de forma atómica la probabilidad de observar el conjunto Y. En este caso, es evidente que L debe ser definida semi-positiva (para que pueda definir una probabilidad) pero gracias a que sólo es una relación de proporcionalidad, no es necesario pedir que la matriz tenga valores propios acotados por 1.

Es posible probar que la constante de proporcionalidad viene dada por $\frac{1}{\det(L+I)}$ con I la matriz identidad con las mismas dimensiones de L (Teorema B.1). Por otro lado, obtener la matriz K a través de L es posible mediante la ecuación (Teorema B.2)

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

y de forma inversa,

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

la observación fundamental es que es posible computar K mediante la descomposición en valores y vectores propios de $L = \sum_{n=1}^{N} \lambda_n v_n v_n^\top$ simplemente reescalando

$$K = \sum_{n=1}^{N} \frac{\lambda_n}{\lambda_n + 1} v_n v_n^\top.$$

Notar que no todo kernel marginal define un L-ensamblaje, en particular cuando alguno de sus valores propios alcanza el valor de 1 pues la matriz $(I-K)$ deja de ser invertible. Estamos en condiciones de definir el algoritmo de sampleo para un DPP.

Teorema 2.1.1 *Sea $L = \sum_{n=1}^{N} \lambda_n v_n v_n^\top$ una descomposición ortonormal de una matriz semi-definida positiva L. El siguiente algoritmo samplea $\mathbf{Y} \sim \mathcal{P}_L$.*

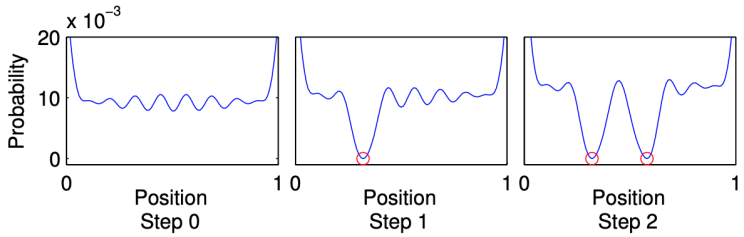
Algoritmo 1 Muestreo de un DPP $O(Nk^3)$, $k = |V|$ (Kulesza 2012)

Require: Descomposición en valores y vectores propios $\{(v_n, \lambda_n)\}_n$ de L
  $J \leftarrow \emptyset$
  for n = 1 , . . . , N **do**
    $J \leftarrow J \cup \{n\}$ con probabilidad $\frac{\lambda_n}{\lambda_n+1}$
  end for
  $V \leftarrow \{v_n\}_{n \in J}$
  $Y \leftarrow \emptyset$
  while $|V| > 0$ **do**
    Seleccionar i de $\mathcal{Y}$ con probabilidad $Pr(i) = \frac{1}{|V|} \sum_{v \in V} (v^\top e_i)^2$ con e_i vector canónico de dimensión N
    $Y \leftarrow Y \cup i$
    $V \leftarrow V_\perp$, una base ortonormal del subespacio de V ortogonal a e_i
  end while
  Output: Y

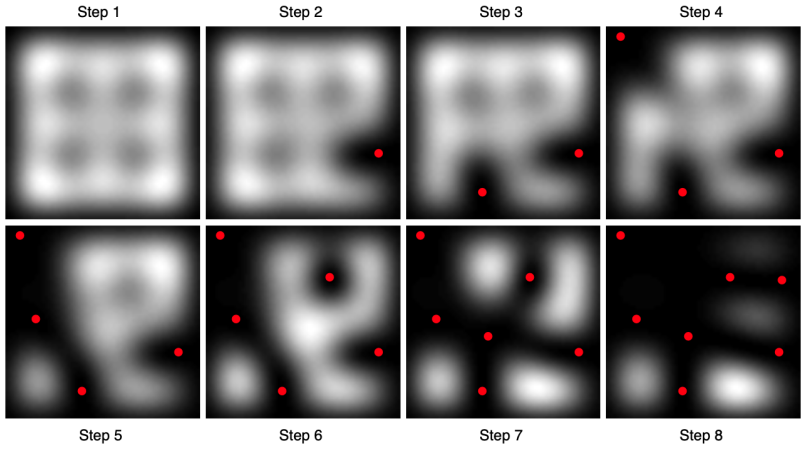
La demostración de este teorema es extensa y por tanto no incluida en el marco teórico, la complejidad del algoritmo es principalmente causada por la ortonormalización *Gram-Schmidt* $O(Nk^2)$ necesaria para calcular $V_\perp$. Por otro lado, existe un algoritmo de sampleo alternativo y más eficiente en situaciones en las que N es grande.

Sea L un L-ensamblaje y consideremos la descomposición de *Cholesky* $L = B^\top B$ con $B \in \mathcal{M}_{D \times N}(\mathbb{R})$ donde $D \leq N$, se define

$$C = BB^\top,$$



(a) Sampling points on an interval



(b) Sampling points in the plane

Figura 2.1: Muestreo de un DPP en una y dos dimensiones. Cada ejemplo sampleado disminuye la probabilidad de muestrear otro en una vecindad cercana (Kulesza 2012)

por lo que $C \in \mathcal{M}_{D\times D}(\mathbb{R})$, es idealmente, una matriz de menor dimensión que L. El algoritmo de muestreo (Algoritmo 2) a continuación, se basa en el siguiente resultado:

$$C = \sum_{i=1}^{D} \lambda_n \hat{v}_n \hat{v}_n^\top.$$

es una descomposición en valores y vectores propios de C si y solamente si

$$L = \sum_{i=1}^{D} \lambda_n \left(\frac{1}{\sqrt{\lambda_n}} B^\top \hat{v}_n\right)\left(\frac{1}{\sqrt{\lambda_n}} B^\top \hat{v}_n\right)^\top,$$

es una descomposición en valores y vectores propios de L (Teorema B.3). La idea es representar V, el conjunto de vectores en $\mathbb{R}^N$ como el conjunto $\hat{V}$ de vectores en $\mathbb{R}^D$ utilizando el mapeo

$$V = \left\{B^\top \hat{v} \mid \hat{v} \in \hat{V}\right\},$$

que cumple que si $v_1 = B^\top \hat{v}_1$ y $v_2 = B^\top \hat{v}_2$, entonces

$$
\begin{aligned}
v_1^\top v_2 &= (B^\top \hat{v}_1)^\top (B^\top \hat{v}_2) \\
&= \hat{v}_1^\top C \hat{v}_2.
\end{aligned}
$$

Lo anterior, permite calcular el producto entre v_1 y v_2 en términos de vectores y matrices con dimensión en D, reduciendo la complejidad a $O(D^2)$, por ejemplo, cuando se realiza la normalización en el algoritmo de *Gram-Schmidt* realizando la actualización de $v = B^\top \hat{v}$ implícitamente con la actualización de $\hat{v} \leftarrow \frac{\hat{v}}{\sqrt{\hat{v}^\top C \hat{v}}}$.

Algoritmo 2 Muestreo de un DPP $O(NDk^2 + D^2k^3)$, $k = |\hat{V}|$ (Kulesza 2012)

Require: Descomposición en valores y vectores propios $\{(\hat{v}_n, \lambda_n)\}_n$ de C

$J \leftarrow \emptyset$

for n = 1 , . . . , N **do**

$J \leftarrow J \cup \{n\}$ con probabilidad $\frac{\lambda_n}{\lambda_n + 1}$

end for

$V \leftarrow \{\frac{\hat{v}_n}{\sqrt{\hat{v}_n^\top C \hat{v}_n}}\}_{n \in J}$

$Y \leftarrow \emptyset$

while $|V| > 0$ **do**

Seleccionar i de $\mathcal{Y}$ con probabilidad $Pr(i) = \frac{1}{|\hat{V}|} \sum_{\hat{v} \in \hat{V}} (\hat{v}^\top B_i)^2$

$Y \leftarrow Y \cup i$

Sea $\hat{v}_0$ un vector en $\hat{V}$ tal que $B_i^\top \hat{v}_0 \neq 0$

$\hat{V} \leftarrow \left\{ \hat{v} - \frac{\hat{v}^\top B_i}{\hat{v}_0^\top B_i} \hat{v}_0 \mid \hat{v} \in \hat{V} \backslash \{\hat{v}_0\} \right\}$

Ortonormalizar $\hat{V}$ con respecto al producto punto definido como $\langle \hat{v}_1, \hat{v}_2 \rangle = \hat{v}_1^\top C \hat{v}_2$

end while

Output: Y

2.1.1.4. k-DPP

Un proceso puntual determinantal (DPP) asigna una probabilidad a cada subconjunto de $\mathcal{Y}$, es decir, eventualmente puede que un sampleo de este proceso sea un conjunto vacío o incluso todo $\mathcal{Y}$. Con el objetivo de fijar la cardinalidad k del sampling, se puede modificar el Algoritmo 2 para samplear exactamente k vectores propios forzando la cardinalidad de J. El algoritmo que realiza este proceso es asintóticamente tan rápido conforme crece el N como lo es samplear de un DPP común, sin embargo, no será detallado en esta tesis.

2.1.2. Poisson Disk Sampling

Este tipo de procesos repulsivos se caracteriza por mostrar una repulsión mucho mayor a la de un DPP pues no permite que los ejemplos se encuentren a una distancia menor a un r definido. En concreto, sea $\rho(x, y)$ la cantidad de puntos esperados alrededor de x, y (densidad producto de segundo orden) y $\lambda(x)$ la cantidad de puntos esperados alrededor de x (densidad producto de primer orden), entonces

$$
\left[\frac{\rho(x,y)}{\lambda(x)\lambda(y)} \right] = \begin{cases} 0 & \text{Si los puntos están cerca } (d(x,y) \leq r) \\ 1 & \text{Si los puntos están lejos } (d(x,y) >> r) \end{cases},
$$

es decir, no es posible samplear un par de ejemplos a una distancia menor de r según alguna métrica de distancia $d(\cdot, \cdot)$ a diferencia de los DPP que asignan una pequeña (eventualmente nula) probabilidad a que esto suceda.

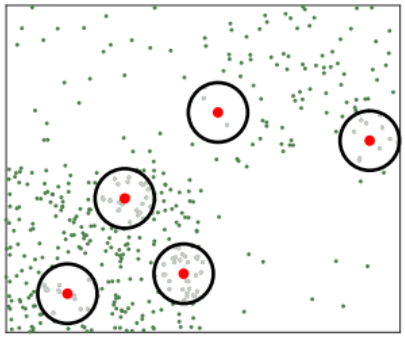


Figura 2.2: Sampling de un PDS en dos dimensiones. Cada elemento muestreado crea un área de rechazo para los siguientes ejemplos (Zhang et al. 2018).

El algoritmo que permite samplear de un *Poisson Disk Sampling* (PDS) es el siguiente

Algoritmo 3 Muestreo de un PDS $O(k^2)$, k tamaño de la muestra (Zhang et al. 2018)

Require: Distancia mínima r, conjunto de ejemplos $\{x_i\}_{i=0}^{N}$
  $Y \leftarrow \{x_0\}$ Uniformemente seleccionado de $\{x_i\}_{i=0}^{N}$
  $L \leftarrow \{0\}$
  while $|L| > 0$ **do**
    Escoger i de L uniformemente
    $J \leftarrow$ Hasta $(k - |Y|)$ puntos sampleados uniformemente en el anillo de radio $[r, 2r]$ alrededor de x_i
    for $x_j \in J$ **do**
      if x_j está a distancia mayor a r de todos los elementos en Y **then**
        $Y \leftarrow Y \cup \{x_j\}$
        $L \leftarrow L \cup \{j\}$
      end if
    end for
    if Ningún elemento se agregó a L **then**
      $L \leftarrow L - i$
    end if
  end while
  Output: Y

La idea es samplear "hacia afuera" partiendo desde un punto inicial x_0 e iterando el proceso con cada punto nuevo agregado hasta alcanzar la cantidad requerida de sampleos k (notar la importancia de la inicialización en este caso).

2.2. Redes Neuronales

Una red neuronal artificial es un modelo de aprendizaje de máquinas inspirado en el funcionamiento de las neuronas biológicas que constituyen los cerebros de los animales. [8]

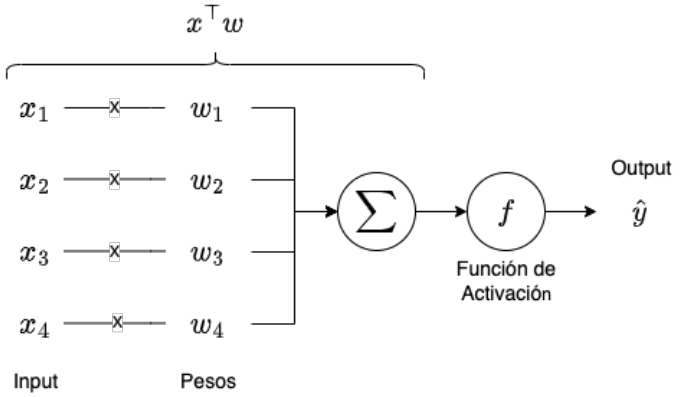


Figura 2.3: Representación del Perceptrón para un input $x \in \mathbb{R}^4$ y pesos $w \in \mathbb{R}^4$.

La unidad fundamental de una red neuronal es el Perceptrón (Figura 2.3), que es simplemente una función matemática que se encarga de tomar un *input* $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ y multiplicarlo con un vector de pesos $w = (w_1, \ldots, w_n) \in \mathbb{R}^n$ mediante

$$\hat{y} = f(x^\top w),$$

donde $f : \mathbb{R} \to \mathbb{R}$ es una función usualmente no lineal conocida como función de activación e $\hat{y} \in \mathbb{R}$ es la salida (*output*).

Los modelos más comunes de redes neuronales se conocen como *feedforward neural networks* (FFNN) o *multilayer perceptrons* (MLPs) y están compuestos por múltiples perceptrones interconectados en una estructura de capas como lo muestra la Figura 2.4.

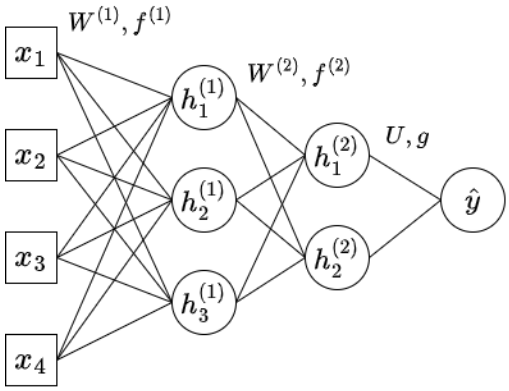


Figura 2.4: MLP con 2 capas ocultas de pesos $W^{(i)}$ y funciones de activación $f^{(i)}$. Además, incluye una capa de output con función de activación g y pesos U. No se agrega un vector de *bias*.

El *output* de cada capa $h^{(k)} = (h_1^{(k)}, h_2^{(k)}, \ldots, h_{w(k)}^{(k)})$ con $w(k)$ el ancho (cantidad de neuro-

nas) de la capa k, se puede describir mediante

$$h^{(k)} = f^{(k)}(h^{(k-1)}W^{(k)} + b^{(k)}), \quad \forall k \in \{1, \ldots, l\}, \quad h^{(0)} = x,$$

donde l es la profundidad de la red, W la matriz de pesos y b el vector de *bias*. La última capa, encargada de producir el *output* de la red, se obtiene de

$$\hat{y} = g(h^{(l)}U + c),$$

donde c es un vector de bias, U una matriz de pesos y g es la función aplicada en la capa de *output* que cambia según el tipo de problema que se quiera resolver, ya sea de clasificación o regresión.

2.2.1. Descenso del Gradiente Estocástico

2.2.1.1. Introducción

El aprendizaje de una red neuronal se realiza en base a la actualización de las matrices de peso $W^{(k)}$ y los *bias* $b^{(k)}$ con el objetivo de minimizar el valor de la función de error (*loss*) $l(\hat{y}, y)$ que depende de la predicción de la red neuronal $\hat{y}$ y la etiqueta (o valor) real del dato y.

Entre los algoritmos utilizados para la minimización de la función de error $l(\hat{y}, y)$ se pueden distinguir

- Métodos Batch o Determinísticos: Son aquellos que utilizan el conjunto de entrenamiento completo en cada iteración y tienden a quedar atrapados en óptimos locales.
- Métodos Estocásticos (*Online*): Aquellos que utilizan un solo ejemplo a la vez para el entrenamiento y que resultan ineficientes para grandes volúmenes de datos.
- Métodos *Minibatch* Estocásticos: Aquellos que utilizan un subconjunto reducido de ejemplos y promedian los gradientes para obtener el valor esperado.

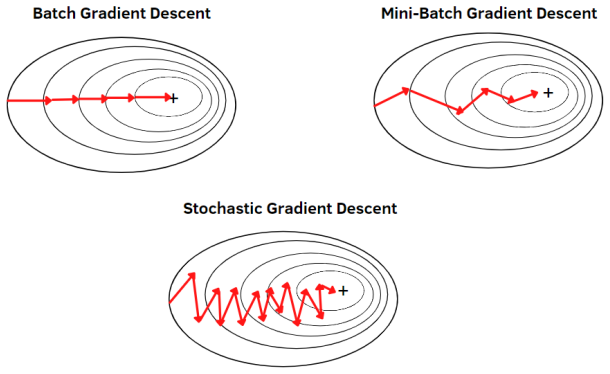


Figura 2.5: Representación de distintos tipos de descensos del gradiente según el tamaño del *Batch* (Wash 2022) [9].

Nos centraremos en este último pues es uno de los más utilizados para el entrenamiento de redes neuronales debido a su rapidez y capacidad de convergencia superior a otros métodos [10], [11].

De manera formal, sea θ el parámetro que se busca encontrar y $J^*(\theta)$ la función objetivo que depende de la medida de desempeño $l(\hat{y}, y) = l(f(x, \theta), y)$ según:

$$J^*(\theta) = \mathbb{E}_{(x,y)\sim p_{\text{data}}} l(f(x, \theta), y).$$

El objetivo es minimizar el error esperado sobre la distribución generada de los datos p_{data}. Para realizar esta tarea, reemplazaremos la expresión J^* por la aproximación

$$J(\theta) = \mathbb{E}_{(x,y)\sim \hat{p}_{\text{data}}} l(f(x, \theta), y) = \frac{1}{k} \sum_{i=1}^{k} l(f(x_i, \theta), y_i),$$

es decir, el promedio de los errores para un *batch* de tamaño k. Notar que en esta segunda expresión, los datos fueron generados por una distribución $\hat{p}_{\text{data}}$ a priori distinta de p_{data}.

2.2.1.2. Descenso del Gradiente Estocástico: Mini-Batch

Con la función a minimizar $J(\theta)$ definida, la actualización de los parámetros θ se realiza mediante

$$\theta_{t+1} = \theta_t - \rho_t \frac{1}{|B|} \sum_{i \in B} \boldsymbol{\nabla} l(f(x_i, \theta_t), y_i), \quad B \sim \mathcal{P},$$

donde $B \subset \{1, \ldots, N\}$ es el conjunto de índices del *mini-batch* sampleado según $\mathcal{P}$, normalmente de manera uniforme entre todo el conjunto de entrenamiento que no ha sido visto previamente y ρ_t la tasa de aprendizaje (*learning rate*) del algoritmo en el instante t.

2.2.1.3. Épocas de Entrenamiento

En el contexto del *Deep Learning*, las épocas se definen como ciclos de entrenamiento en el que se recorre completamente un conjunto de datos iterando a través de *batchs*. Una mayor cantidad de épocas permite que los modelos aprendan de manera gradual el conjunto sobre el cual se esta entrenando, pero también deben estar acotadas a un número máximo para prevenir el *overfitting* (el sobreajuste no permite generalizar a datos no vistos anteriormente). La elección correcta de las épocas va a depender de la complejidad de los datos, la arquitectura de la red y las características del problema a resolver.

2.2.2. Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales (CNN) son un tipo de red neuronal utilizadas principalmente en problemas cuyo *input* son del tipo matricial como series de tiempo, imágenes, videos, entre otros [12]. La principal característica de este tipo de redes es que en algunas de sus capas, se utiliza la operación matemática de convolución, definida como

$$s(t) = (x * w)(t) = \int x(a) w(t - a) da,$$

donde x es el *input* a convolucionar, w es el kernel con parámetros que se busca aprender y el *output* $s(t)$ se conoce como *feature map*. La ventaja de esta arquitectura y de la operación que utiliza en sus capas, es que la red aprende características locales que no dependen de su posición (invariante a traslación) y que además requieren de una menor cantidad de parámetros entrenables y conexiones, por lo que resultan altamente eficientes y robustas al *overfitting*.

2.2.3. Autoencoders

Un *Autoencoder* es un tipo de red neuronal que busca replicar en el *output* el *input* entregado. El desafío es que esto se pueda realizar a partir de una representación de menor dimensionalidad que el *input* a través de un cuello de botella en la arquitectura de la red [13].

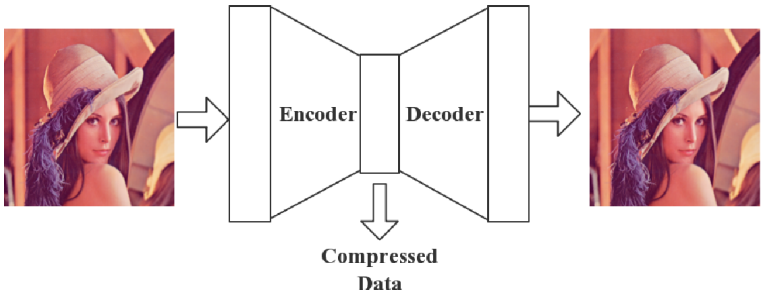


Figura 2.6: Autoencoder, el cuello de botella en la mitad de la estructura codifica una representación de menor dimensionalidad del *input* (Zhang 2018).

El *Autoencoder* en la Figura 2.6) se compone de 2 partes, la primera denominada *encoder*, es la que se encarga de extraer la información más representativa del *input*. La segunda llamada *decoder*, es la que se encarga de recrear la imagen en base a la representación provista.

Este tipo de red, que si bien su estructura podría ser del tipo FFNN o CNN, no requiere conocer la etiqueta del *input* (y), es decir, se enmarca dentro de los métodos de aprendizaje de máquinas no supervisados.

2.2.4. One Shot Learning

One Shot Learning o (*Few Shot Learning*) es un método de aprendizaje de máquinas utilizado comúnmente en el área de visión computacional y que busca reducir drásticamente la cantidad de ejemplos necesarios para clasificar objetos [14].

Para lograr esto, se implementa una red del tipo siamés representada en la Figura 2.7 que consta de 2 CNN entrenadas en paralelo y que comparten los mismos pesos. Además, 2 FFNN también entrenadas en paralelo pero con pesos independientes se encargan de interpretar las características entregadas y su relación. Finalmente, ambas representaciones son comparadas mediante alguna medida de distancia, usualmente euclidiana, para obtener el *score* de similaridad.

El entrenamiento de este tipo de redes puede ser realizado de múltiples formas entre las que se encuentran:

- *Contrastive Loss*: Esta función de pérdida calculada entre pares de ejemplos busca que datos similares sean mapeados en regiones cercanas del espacio.

$$l(x_i, x_j) = -\log \frac{\exp(d(z_i, z_j)/\tau)}{\sum_{k=1, k\neq i}^{N} \exp(d(z_i, z_k)/\tau)},$$

  Donde z_i y z_j son la representación de los ejemplos x_i y x_j respectivamente. Notar la similitud con una función *Softmax* a la que además se le agrega un factor de normalización (temperatura) τ. La función $d(\cdot, \cdot)$ es alguna métrica de distancia previamente

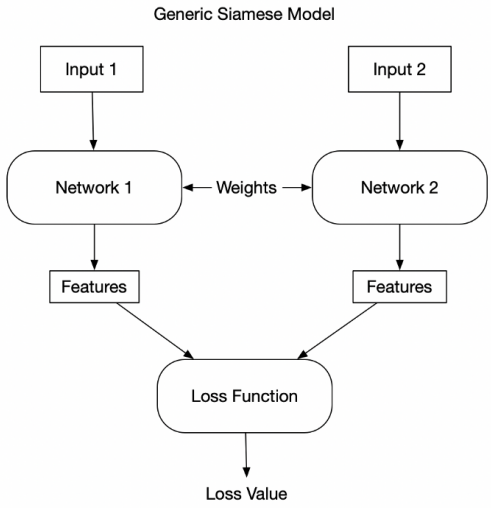


Figura 2.7: Arquitectura One Shot Learning, los *batches* de entrenamiento se componen de tuplas de ejemplos que son entregados en cada sección de la red.

definida como una distancia coseno o una distancia euclidiana.

- *Triplet Loss*: Esta función de pérdida se calcula entre tripletas de ejemplos lo que permite que el entrenamiento se pueda hacer con muy pocos datos. Para cada imagen base, denominada *Anchor*, se escoge una imagen de la misma clase (*Positive*) y una de distinta clase (*Negative*).

Figura 2.8: Triplet Loss: La imagen *Anchor* se compara con una imagen de misma etiqueta (*Positive*) y una de distinta etiqueta (*Negative*) (Ghandi 2018) [15].

La función de *triplet loss* tomará en cuenta estas 2 comparaciones según

$$l(x_a, x_p, x_n) = \max(d(z_a, z_p) - d(z_a, z_n) + \text{margin}, 0),$$

es decir, considerara la capacidad de la red de reconocer imágenes de igual y distinta clase en simultaneo.

Ambos métodos tienen la ventaja de que se necesita una pequeña cantidad de datos para aprender a diferenciar, pues, el conjunto de entrenamiento se crea en base a todos los posibles pares (o tríos) de ejemplos que se pueden formar.

Capítulo 3

Estado del Arte

3.1. Introducción

En la búsqueda de nuevas técnicas y herramientas que permitan mejorar el rendimiento de las redes neuronales, están aquellas que se centran en los algoritmos de optimización, en la selección de *batches*, las épocas de entrenamiento, entre muchos otros elementos que componen la estructura de la red.

En este contexto, uno de los avances más importantes del área se ha obtenido aplicando cambios en el algoritmo de optimización. Desde la adición de *Momentum* al algoritmo de descenso del gradiente estocástico, hasta modificaciones dinámicas del *learning rate* e incluso, combinaciones de ambas con el algoritmo ADAM (*Adaptative Moment Optimization*) [16].

Por otro lado, en la selección de *batches*, el artículo *Online Batch Selection for Faster Training of Neural Networks* [17] propone visitar con más frecuencia aquellos ejemplos que contribuyan en mayor magnitud a la función de *loss*.

Específicamente, el algoritmo propuesto en [17], ordena los N elementos del conjunto de entrenamiento según su último gradiente computado $\psi_i(x)$ en sentido descendiente. En la próxima iteración, cada elemento del *batch* se selecciona según una probabilidad p_i definida como

$$p_i = \frac{1/\exp(\log(s_e)/N)^i}{\sum_{k=1}^{N} 1/\exp(\log(s_e)/N)^k},$$

donde s_e es un parámetro del modelo denominado *selection pressure*. Notar que p_i sigue un decaimiento exponencial y que los elementos que más aportan al gradiente serán muestreados con una mayor probabilidad en la siguiente iteración.

El artículo *Submodular Batch Selection for Training Deep Neural Networks* publicado en 2019 [18], propone que el *batch* seleccionado tiene que ser lo más informativo posible. Se utiliza el *Uncertainty Score* $U(x_i)$ que corresponde a la entropía del modelo actual w^t en la iteración t para el elemento x_i

$$U(x_i) = -\sum_{y \in C} P(y|x_i, w^t) \log P(y|x_i, w^y),$$

donde C es el conjunto de todas las clases. Notar que si bien podríamos seleccionar un *batch* que únicamente maximice esta cantidad, se podría llevar a samplear elementos muy similares con alta entropía. Para evitar esto, se complementa con el *Redundacy Score* $R(x_i)$ definido

como

$$R(x_i) = \min_{x_j \in S: i \neq j} \phi(x_i, x_j),$$

donde $\phi(\cdot, \cdot)$ es alguna métrica de distancia. Entre mayor sea $R(x_i)$, el elemento x_i añadido al *batch* es más diverso con respecto a sus pares y por tanto un mejor candidato.

Finalmente, el artículo *Active Mini Batch Sampling using Repulsive Point Processes* [1] publicado en 2018, propone utilizar procesos repulsivos para el selección de los *mini batches* de entrenamiento y prueba que efectivamente, esto reduce la varianza del estimador del gradiente en el algoritmo de SGD. Es en este último trabajo en el que profundizaremos.

3.2. Active Mini Batch Sampling

La modificación al algoritmo de descenso de gradiente estocástico (SGD) que motiva el artículo [1] es la siguiente

$$\theta_{t+1} = \theta_t - \rho_t \hat{G}(\theta_t) = \theta_t - \rho_t \frac{1}{|B|} \sum_{i \in B} \boldsymbol{\nabla} l(f(x_i, \theta_t), y_i), \quad B \sim \mathcal{P},$$

donde $B \sim \mathcal{P}$ es el *mini-batch* sampleado a través de algún proceso puntual $\mathcal{P}$ (a diferencia de un SGD estándar donde B es sampleado de forma aleatoria). El proceso puntual seleccionado corresponde a uno del tipo repulsivo, es decir, que busca maximizar la diversidad de la muestra según alguna métrica. El resultado que fundamenta la utilización de este proceso es

Teorema 3.2.1 *Sea $\hat{G}$ el estimador del gradiente objetivo definido como*

$$\hat{G}(\theta) = \frac{1}{|B|} \sum_{i \in B} \boldsymbol{\nabla} l(f(x_i, \theta), y_i),$$

con B sampleado a través de un proceso repulsivo $\mathcal{P}$. Entonces

$$\mathbb{V}ar(\hat{G}) = \frac{1}{k^2} \int_{\nu \times \nu} \lambda(x)\lambda(y) g(x,\theta)^\top g(y,\theta) \left[\frac{\rho(x,y)}{\lambda(x)\lambda(y)} - 1 \right] dxdy + \frac{1}{k^2} \int_{\nu} ||g(x,\theta)||^2 \lambda(x) dx,$$

donde $g(x, \theta) = \boldsymbol{\nabla} l(f(x, \theta))$, $\lambda(x)$ la cantidad de puntos esperada alrededor de x, $\rho(x, y)$ la cantidad de puntos esperada alrededor de x e y, $\nu \times \nu \subset \mathbb{R}^n \times \mathbb{R}^n$ la grilla considerada y $k = |B|$ el tamaño del mini-batch.

La demostración se incluye en el anexo (Teorema B.4). Los 2 puntos importantes en este resultado son que:

1. El primer término de $\mathbb{V}ar(\hat{G})$ es negativo para cualquier proceso repulsivo.
2. El primer término desaparece cuando se utiliza un método de sampleo uniforme (como en el SGD estándar) pues $\rho(x, y) = \lambda(x)\lambda(y)$.

Con ambas propiedades en mente (Corolario B.1), se puede concluir que un proceso repulsivo disminuye la varianza del estimador del gradiente objetivo y por tanto la convergencia al óptimo se puede realizar de forma mucho más rápida (por ejemplo, aumentando el valor del *learning rate*).

3.3. Oportunidad de Mejora

El problema con utilizar un proceso repulsivo en la selección del *mini batch* es que añade un costo extra a cada iteración del algoritmo de SGD y que resulta en un cuello de botella para el entrenamiento de la red neuronal. Para soslayar este asunto, el artículo [1] propone utilizar un proceso repulsivo *Poisson Disk Sampling* (PSD) cuya complejidad es considerablemente menor ($O(k^2)$) a la de un k-DPP ($O(Nk^3)$) pero que muestra un repulsión local mucho mayor [19] (lo que no siempre es conveniente en problemas de clasificación desbalanceados).

En este contexto, el objetivo de esta tesis será encontrar estructuras alternativas que permitan utilizar un proceso repulsivo tan estudiado como un DPP pero que no se vean limitadas por el tiempo de entrenamiento de la red. En la siguiente sección, se detallan posibles arquitecturas que utilizan un DPP para la selección de sus *batches*, pero que a diferencia de lo presentado en el artículo [1] no entrenan con todos los datos.

Capítulo 4

Metodología

4.1. Métodos Propuestos

4.1.1. Fast DPP

El primer algoritmo de esta sección, que denominaremos *Fast DPP*, es la primera modificación al propuesto en [1] y descrito en el capítulo anterior. Este, sienta las bases de los algoritmos propuestos más adelante y busca realizar el entrenamiento de la red neuronal sin la necesidad de mirar todos los datos en cada época, pero intentando siempre maximizar la diversidad de los *batches* sampleados mediante un DPP.

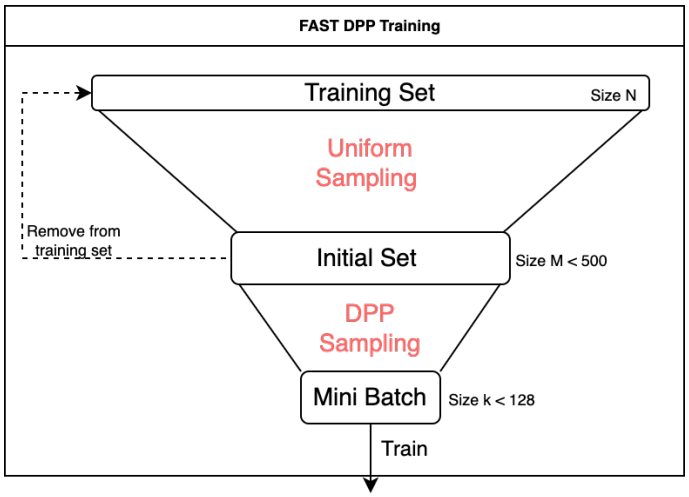


Figura 4.1: Fast DPP: El diagrama muestra el proceso de extracción de un *Mini Batch* desde el conjunto de entrenamiento y su posterior muestreo con DPP.

Como lo muestra la Figura 4.1, en cada iteración de una época, se extrae de manera uniforme, un conjunto inicial de cardinalidad $M < 500$ del cual se samplea posteriormente el *mini-batch* de tamaño $k < 128$ haciendo uso del DPP, lo elementos no utilizados se eliminan del conjunto de entrenamiento. La razón por la cual se realiza esta primera selección, es la de reducir el coste de sampling de un DPP y según el parámetro M, controlar la cantidad de iteraciones que el algoritmo realiza en la época (a mayor M menos iteraciones por época pero menor será la cantidad de ejemplos visitados y mayor será el tiempo de sampling del

DPP).

Notar que si bien podrían eliminarse del conjunto de entrenamiento únicamente los ejemplos utilizados en el *mini-batch* (y no los M muestreados), nuestro objetivo es el de acelerar el entrenamiento de la red mediante la omisión de ejemplos "redundantes" según el DPP, vale decir, asumir que el *mini-batch* "resume" la información contenida en el conjunto muestreado inicialmente.

Algoritmo 4 Fast DPP Training por época

Require: Conjunto de entrenamiento $\{(x_i, y_i)\}_{i=1}^N$, M tamaño del primer sampling y k tamaño del mini-batch, $k < M$.
  $Z \leftarrow \{x_i\}_{i=1}^N$
  while $|Z| > 0$ **do**
    if $|Z| > M$ **then**
      $D \leftarrow$ Sampling uniforme de Z, $|D| = M$
      $B \leftarrow$ Sampling de un DPP de D, $|B| = k$
    else
      $D \leftarrow Z$
      $B \leftarrow$ Sampling de un DPP de D, $|B| = \min(k, |D|)$
    end if
    Entrenar la red con el mini-batch B
    $Z \leftarrow Z - D$
  end while

Es directo ver que el Algoritmo 4 siempre termina pues $D \neq \emptyset$ y la actualización $Z \leftarrow Z - D$ reduce el tamaño de Z hasta que el conjunto es eventualmente vacío. La cantidad de iteraciones por época está dado por $\lfloor N/M \rfloor$ si M es divisor de N y $\lfloor N/M \rfloor + 1$ si no lo es, por lo que comparado a un algoritmo clásico de descenso de gradiente estocástico que realiza $\lfloor N/k \rfloor$ iteraciones o bien $\lfloor N/k \rfloor + 1$, *Fast DPP* realiza aproximadamente $\lfloor M/k \rfloor$ veces menos iteraciones.

4.1.2. Mixed DPP

La estrategia mencionada anteriormente puede ser de gran utilidad para el entrenamiento de una red neuronal cuyo costo por iteración es elevado. Lamentablemente, el rendimiento de la red *Fast DPP* a lo largo de las épocas no mejora tanto como lo haría un algoritmo que pasa por todos los datos.

Con el objetivo de acelerar el entrenamiento y no sacrificar desempeño, es posible realizar un esquema mixto de entrenamiento que denominamos *Mixed DPP*. Este esquema consta de estrategias de sampleo condicionadas a la época en que se encuentre el entrenamiento, vale decir, es posible utilizar una estrategia *Fast DPP* en las primeras épocas ("inicialización") del entrenamiento y terminar con un algoritmo de SGD estándar que visite todos los datos o viceversa (*Reversed Mixed DPP*).

La idea de *Mixed DPP* es que permite "mapear" de manera acelerada las regiones de decisión del problema de clasificación en las primeras épocas y terminar el entrenamiento realizando ligeros "ajustes" utilizando un SGD clásico. Se añade además *Reversed Mixed DPP*, que es la idea contraria a *Mixed DPP* pues es en las últimas épocas de entrenamiento

Algoritmo 5 Mixed DPP Training

Require: Conjunto de entrenamiento $\{(x_i, y_i)\}_{i=1}^N$, M tamaño del primer sampling y k tamaño del mini-batch, $k < M$, l cantidad de épocas de entrenamiento y l_m época de cambio de estrategia , $l_m < l$.

```
for e in {1, ... l} do
    if e < l_m then
        Entrenar la red con ALG_4({(x_i, y_i)}_{i=1}^N, M, k)
    else
        Entrenar la red con SGD({(x_i, y_i)}_{i=1}^N, k)
    end if
end for
```

Algoritmo 6 Reversed Mixed DPP Training

Require: Conjunto de entrenamiento $\{(x_i, y_i)\}_{i=1}^N$, M tamaño del primer sampling y k tamaño del mini-batch, $k < M$, l cantidad de épocas de entrenamiento y l_m época de cambio de estrategia , $l_m < l$.

```
for e in {1, ... l} do
    if e > l_m then
        Entrenar la red con ALG_4({(x_i, y_i)}_{i=1}^N, M, k)
    else
        Entrenar la red con SGD({(x_i, y_i)}_{i=1}^N, k)
    end if
end for
```

(cuando la función de *loss* ya no varía mucho), en la que se aplica *Fast DPP* con la intención de superar el *plateau* al cambiar la distribución de los *batches*.

Ambas estrategias Algoritmo 5 y Algoritmo 6 son más rápidas que entrenar con un algoritmo *SGD* clásico pues optimizan el tiempo de entrenamiento en las épocas con *Fast DPP*.

4.1.3. DPP NET

Este modelo propuesto intenta solucionar el problema de la definición de la métrica de similitud necesaria para samplear un DPP mientras se entrena en simultáneo la red que se encarga del problema de clasificación.

La red se compone principalmente de un *Autoencoder* y una *fully connected* (FC) o una red convolucional (CNN). La sección del *Encoder* junto a la FC (o CNN), se encargan de realizar la predicción de la etiqueta del problema de clasificación. Por otro lado, el *Autoencoder* en simultáneo, estima una representación de menor dimensionalidad del *input* que servirá para los futuros muestreos del DPP con el uso de una distancia euclidiana. Para entrenar la red, se utiliza la siguiente función de *loss*

$$\mathcal{L} = \alpha \mathcal{L}_A(\hat{x}, x) + (1 - \alpha)\mathcal{L}_{CNN}(\hat{y}, y),$$

es decir, una suma ponderada de las *losses* del *Autoencoder* y de la subred encargada del problema de clasificación como lo muestra el Diagrama 4.2.

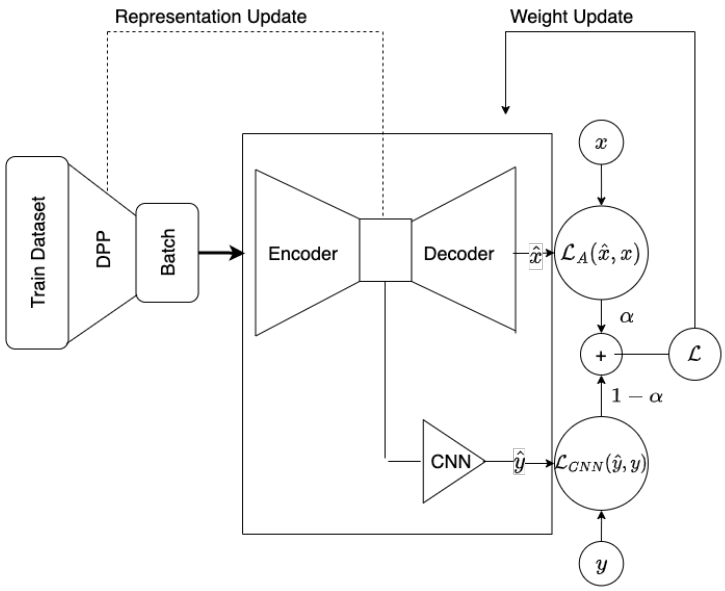


Figura 4.2: DPP NET: El diagrama muestra una arquitectura que incluye el muestreo mediante un DPP, el *Autoencoder* encargado de obtener la representación de baja dimensionalidad de los datos y la subred encargada de la predicción de la etiqueta del problema de clasificación.

La ventaja de este enfoque es que no requiere del entrenamiento previo del *Autoencoder* o del aprendizaje de la métrica con *Oneshot Learning*. Por otro lado, con el parámetro α, es posible controlar qué tanto se busca actualizar los pesos de ambas redes por separado y de ser necesario, desconectar una de otra.

El desafío de este modelo es que el entrenamiento en comparación con un *baseline* de arquitectura equivalente pero que no entrene un *Autoencoder*, es el tiempo por iteración necesario para su ajuste. Por tanto, será importante ajustar el valor de M en *Fast DPP* con tal de no perder en rendimiento ni en tiempo de entrenamiento.

4.2. Definición de Métricas

4.2.1. Kernel RBF

Recordemos que para samplear de un DPP es necesario un kernel K (o un L-ensamblaje) que cumpla con ser semidefinido positivo y de valores propios acotados por 1. Por otro lado, si se asume que todos los elementos de un conjunto son equiprobables de samplear, la diagonal del kernel debe tener valores iguales, es decir $K_{i,i} = c > 0, \forall i$.

Con lo anterior, un kernel que cumple con tales características es el kernel RBF (Radial Basis Function) [20] definido como

$$K(x_i, x_j) = \exp\left(-\frac{d(x_i, x_j)^2}{2l^2}\right),$$

donde $d(\cdot, \cdot)$ es alguna métrica de distancia y $l > 0$ el *length scale*. Notar que $d(x_i, x_i) = 0 \Rightarrow K(x_i, x_i) = 1$ y si $d(x_i, x_j) \to \infty \Rightarrow K(x_i, x_j) \to 0$ lo que hace sentido pensando en que en

Algoritmo 7 DPP NET por época

Require: Conjunto de entrenamiento $\{(x_i, y_i)\}_{i=1}^N$, M tamaño del primer sampling y k tamaño del mini-batch, $k < M$, $\alpha \in [0, 1]$
  $Z \leftarrow \{x_i\}_{i=1}^N$
  while $|Z| > 0$ **do**
    if $|Z| > M$ **then**
      $D \leftarrow$ Sampling uniforme de Z, $|D| = M$
      $\hat{D} \leftarrow$ Encoder(D)
      $B \leftarrow$ Sampling de un DPP de $\hat{D}$, $|B| = k$
    else
      $D \leftarrow Z$
      $\hat{D} \leftarrow$ Encoder(D)
      $B \leftarrow$ Sampling de un DPP de $\hat{D}$, $|B| = \min(k, |\hat{D}|)$
    end if
    Entrenar la red con el mini-batch B y *loss* $\mathcal{L} = \alpha\mathcal{L}_A(\hat{x}, x) + (1 - \alpha)\mathcal{L}_{FC}(\hat{y}, y)$
    $Z \leftarrow Z - D$
  end while

los procesos repulsivos $\mathcal{P}$ samplean con mayor probabilidad elementos alejados unos de otros ya que

$$\mathcal{P}(i, j \in Y) = \mathcal{P}(i \in Y)\mathcal{P}(j \in Y) - K_{ij}^2.$$

4.2.2. Minkowski de orden p

La principal dificultad de trabajar con un proceso repulsivo como un DPP es definir correctamente la métrica de similitud (o distancia) $d(\cdot, \cdot)$. Esto pues, dependiendo del tipo de dato ya sean imágenes, series de tiempo, textos, entre otros, la métrica más adecuada puede variar entre los problemas.

Para abordar el problema, se puede suponer que el tipo de dato es un vector $x \in \mathbb{R}^d$ donde generalmente $d \in \mathbb{N}$ es grande ($d \geq 9$). Es posible utilizar $d(x, y) = ||x - y||_2$ la distancia euclidiana como medida de similaridad pero no es tan efectiva en altas dimensiones como lo muestra [21] por lo que se propone alternativamente utilizar la generalización

$$d_p(x, y) = ||x - y||_p = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{1/p},$$

llamada distancia de *Minkowski* de orden p. Si $p = 2$ se obtiene la distancia euclidiana, si $p = 1$ se obtiene la distancia de *Manhattan* y si $p < 1$ se obtiene la "distancia" fraccional que si bien no cumple exactamente la desigualdad triangular, son menos propensas a sufrir el efecto de "concentración" que tiene la distancia euclidiana en altas dimensiones y por tanto son útiles para este tipo de problemas [22].

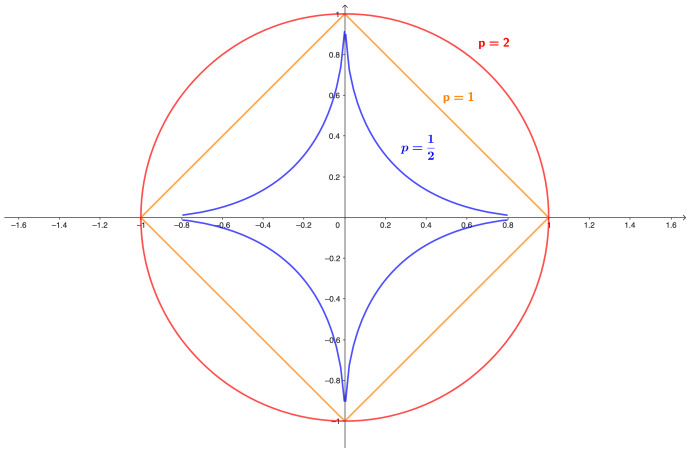


Figura 4.3: Bola Unitaria para distintos valores de p.

4.2.3. Distancia Coseno

Otra métrica relevante a considerar es la "distancia" coseno definida como

$$d_{\cos}(x, y) = 1 - \text{sim}_{\cos}(x, y) = 1 - \frac{\langle x, y \rangle}{||x||_2 ||y||_2},$$

notar que si bien no cumple con la desigualdad triangular, esta métrica es de utilidad para diferenciar elementos según su dirección con respecto al origen y no según la magnitud como la distancia euclidiana. Lamentablemente, esta métrica se ve afectada por la maldición de la dimensionalidad aunque en menor magnitud.

4.2.4. Autoencoder: Representación Latente

La utilización de un *Autoencoder* resulta de gran utilidad para definir una métrica de distancia que no dependa en gran medida del tipo de datos con los que se trabaja. Al obtener la representación de baja dimensionalidad de los datos, la distancia euclidiana y la distancia coseno representan de manera fidedigna la similitud de los datos y por tanto pueden ser utilizados por el DPP. De manera formal

$$d_{auto} = ||\text{encoder}(x) - \text{encoder}(y)||_2.$$

Lamentablemente, es necesario ajustar un *Autoencoder* previo al entrenamiento de la red encargada de resolver el problema de clasificación y esto podría traer aumentos de tiempo importantes a considerar. Las posibles soluciones son la combinación de entrenamientos o el uso de redes pre-entrenadas para este propósito.

4.2.5. Learned Metric: One Shot Learning

Este tipo de métrica es resultado directo del entrenamiento de una red siamesa descrita en el capítulo 2 y que, al igual que un *Autoencoder*, utiliza una representación latente de los ejemplos para determinar su similitud con una distancia euclidiana. La ventaja de este enfoque es que el *output* de la red es efectivamente una métrica de distancia (*Learned Metric*) y que además requiere de unos pocos ejemplos para ser entrenada.

4.3. Métrica de Desempeño

El tipo de problema en que nos centraremos corresponden a problemas de clasificación (binarios y multiclase), es decir, donde el *output* Y es una variable cualitativa discreta. Se define el *accuracy* como

$$\text{accuracy}(y_{pred}, y_{real}) = \frac{TP + TN}{TP + TN + FP + FN},$$

donde TP = Verdaderos positivos, TN = Verdaderos Negativos, FP = Falsos Positivos y FN = Falsos Negativos. Si bien podríamos utilizar esta métrica para medir el desempeño de la red, lo anterior no toma en consideración los ajustes sobre la región de decisión que no cambian la clasificación de los ejemplos pero que si los diferencia con mayor certeza. Dado lo anterior, se decide utilizar directamente la métrica que calcula la función de *loss* en problemas de clasificación

$$\mathcal{L} = -\sum_{i=1}^{C} t_i \log(p_i),$$

donde C es el total de clases, t_i es la etiqueta real del ejemplo y p_i es la probabilidad de pertenecer a la clase i (calculada con una función *sigmoid* en el caso binario y con una función *softmax* en el caso multiclase). Esta métrica permite notar diferencias en el desempeño de una red incluso cuando todos los ejemplos del conjunto son clasificados con las mismas etiquetas que en una iteración anterior.

4.4. Conjuntos de Datos

Se presenta a continuación los datasets que serán utilizados para los experimentos de la sección posterior.

4.4.1. Clasificación Binaria - Baja Dimensionalidad

El dataset de clasificación binaria *dummy* busca recrear una situación en que los datos de 2 grupos se encuentran en forma de "clusters" donde uno de ellos posee considerablemente menos ejemplos.

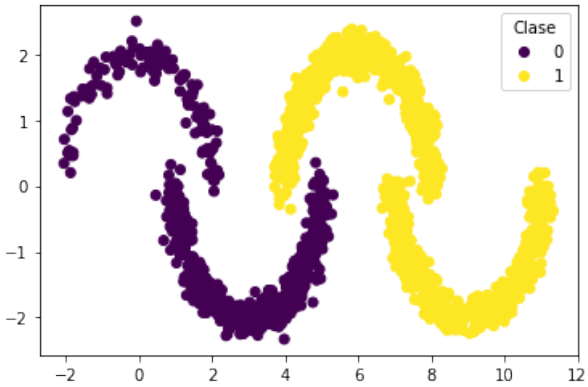


Figura 4.4: Dataset de Clasificación Binaria con 4 clusters artificiales y desbalanceados en forma de luna.

Tabla 4.1: Ejemplos por Cluster - Clasificación Binaria. Total = 1800.

Clase / Cluster	Cluster Superior	Cluster Inferior	Total
Clase 0	120 (6.66 %)	480 (26.66 %)	600 (33.33 %)
Clase 1	600 (33.33 %)	600 (33.33 %)	1200 (66.66 %)

El objetivo de esta configuración es el de permitir que el DPP samplee con mayor frecuencia los ejemplos del cluster que se encuentra en minoría frente al resto.

4.4.2. Clasificación Multiclase - Alta Dimensionalidad

El dataset de clasificación multiclase *Fashion MNIST* [23] en Figura 4.5 abarca el problema de clasificación multiclase (10 clases) de distintas prendas de ropa.

Figura 4.5: Ejemplos de las 10 clases presentes en el conjunto de datos Fashion-MNIST.

Notar que al igual que en el dataset anterior, dentro de una misma clase es posible encontrar distintos "clusters" correspondiente a la misma prenda de ropa pero con diferencias sustanciales en Figura 4.6.

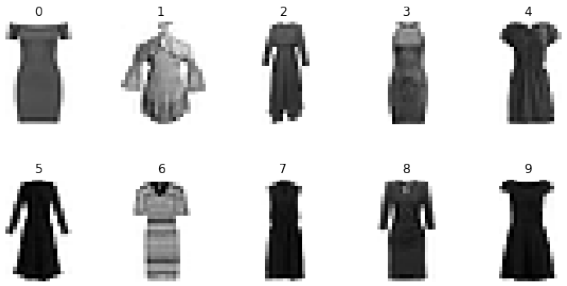


Figura 4.6: Ejemplos de la Clase 3 del conjunto de datos Fashion - MNIST.

Con el fin de agregar complejidad al problema, se utilizará una versión desbalanceada del dataset como la muestra la Tabla 4.2.

4.5. Experimentos

Tabla 4.2: Conjunto de Entrenamiento - Clasificación Multiclase. Total = 16000.

Clase	0	1	2	3	4	5	6	7	8	9
Cantidad	2500	2500	2500	2500	1500	1500	1500	500	500	500
(%)	15.62	15.62	15.62	15.62	9.37	9.37	9.37	3.13	3.13	3.13

4.5.1. Sampling de la Clase Minoritaria

El primer experimento consiste en mostrar la implementación del Proceso Puntual Determinantal (DPP) y ejemplificar su acción al momento de samplear sobre conjuntos de datos desbalanceados. Para ello, consideraremos el conjunto de clasificación binaria en 2 dimensiones utilizando la distancia euclidiana junto a un kernel RBF como medida de similitud y se realizarán múltiples sampleos con reposición con el objetivo de comparar la distribución resultante. A continuación, se utilizará el conjunto de clasificación multiclase desbalanceado para mostrar los mismos resultados pero comparando la distancia euclidana con las métricas propuestas anteriormente *Coseno*, *Autoencoder*, *Oneshot*. Se excluye del análisis la distancia *Minkoswki* $p < 1$ pero se propone como trabajo a futuro.

4.5.2. Fast DPP

En esta sección, se compara el rendimiento de la red implementada *Fast DPP* contra un *baseline* de igual arquitectura pero distinta estrategia de sampleo (SGD) en los conjuntos de datos presentados anteriormente. Las arquitecturas utilizadas son descritas en la siguiente sección, las métricas de distancia se utilizarán según los resultados del experimento anterior. El objetivo del experimento es comparar el *performance* de ambas estrategias tanto numérica como visualmente con el fin de justificar las metodologías propuestas en esta tesis.

4.5.3. Mixed DPP

El experimento en esta sección es análogo en estructura al realizado en *Fast DPP* pues busca comparar el rendimiento de la red propuesta contra un *baseline* de igual arquitectura pero estrategia de sampleo estándar (SGD) en los datasets presentados. A diferencia del experimento anterior, ahora se entrena por una mayor cantidad de épocas con el fin de determinar si samplear con alguna estrategia distinta en el principio (*Mixed DPP*) o al final (*Reversed Mixed DPP*) resulta en alguna mejora al modelo.

4.5.4. DPP NET

En este último experimento, se busca expandir la metodología descrita en las secciones anteriores mediante el entrenamiento en paralelo del *Autoencoder* necesario para obtener la representación de baja dimensionalidad de los ejemplos del dataset *Fashion MNIST*. Si bien se compara esta nueva red con un *baseline* de arquitectura "equivalente", el objetivo es medir el ajuste de la propia red y si es viable para su utilización con un *DPP*. Esto permitiría poder trazar líneas de desarrollo posteriores a esta tesis que serán detalladas en la última sección de conclusiones. La arquitectura a utilizar se describe en la siguiente sección, al igual que los parámetros utilizados.

4.6. Arquitecturas

Se describen a continuación, el detalle de cada una de las arquitecturas de las redes neuronales utilizadas para los experimentos descritos previamente.

4.6.1. Baseline

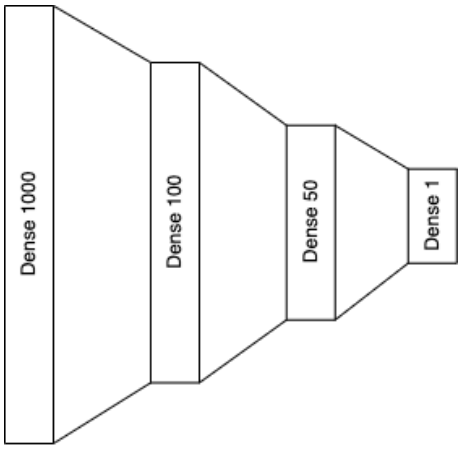


Figura 4.7: Baseline Clasificación Binaria: Esta arquitectura se utiliza como base para realizar comparaciones de las metodologías propuestas en los problemas de clasificación binaria.

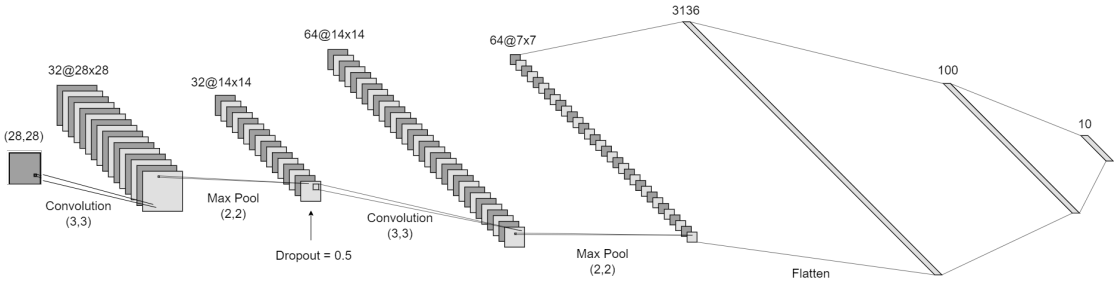


Figura 4.8: Baseline Clasificación Multiclase: Esta arquitectura se utiliza como base para realizar comparaciones de las metodologías propuestas en los problemas de clasificación multiclase (Creado con NN-SVG de Alex Lenail).

Las Figuras 4.7 y 4.8 muestran la arquitectura sobre la cual se comparan los métodos propuestos para los problemas de clasificación binaria y multiclase con *Fashion MNIST*. Los parámetros de entrenamiento de ambas se detallan en Tabla C.1 y Tabla C.2 respectivamente.

4.6.2. Autoencoder

La Figura 4.9 muestra la arquitectura que se utilizará para el entrenamiento del *Autoencoder* utilizado para la reducción de dimensionalidad de las imágenes de *Fashion MNIST*. Los parámetros en los distintos experimentos se detallan en Tabla C.3.

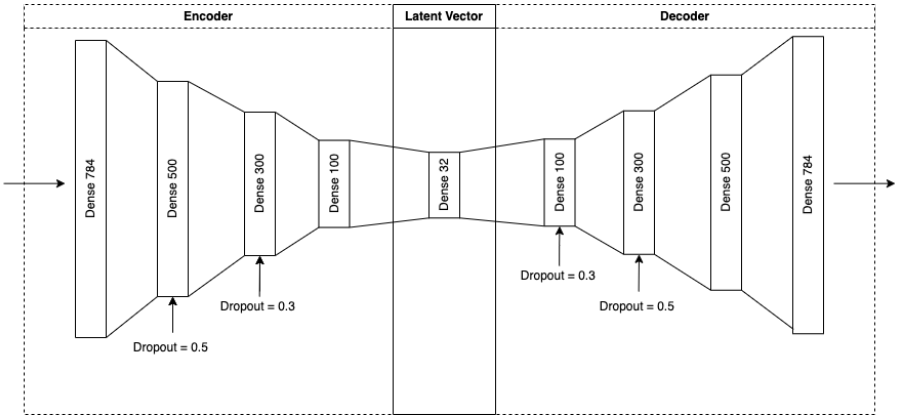


Figura 4.9: Autoencoder: Esta arquitectura se utiliza para obtener la representación de baja dimensionalidad de los datos.

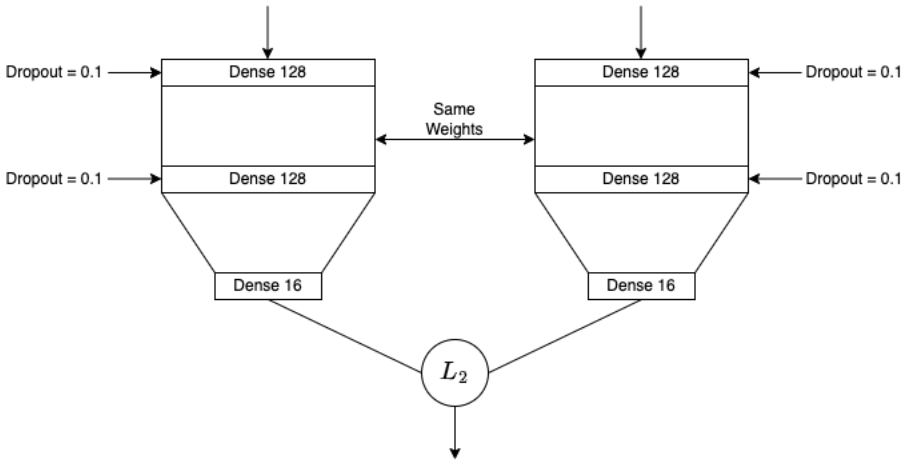


Figura 4.10: Oneshot: Esta arquitectura se utiliza para el aprendizaje de la métrica de distancia necesaria para un DPP.

4.6.3. Oneshot

La Figura 4.10 muestra la arquitectura que se utilizará para el entrenamiento de la red siamesa *Oneshot* que define una métrica de distancia entre pares de ejemplos de *Fashion MNIST*. El entrenamiento se realiza utilizando *contrastive loss* y los parámetros utilizados en los distintos experimentos se detallan en Tabla C.4.

4.6.4. DPP NET

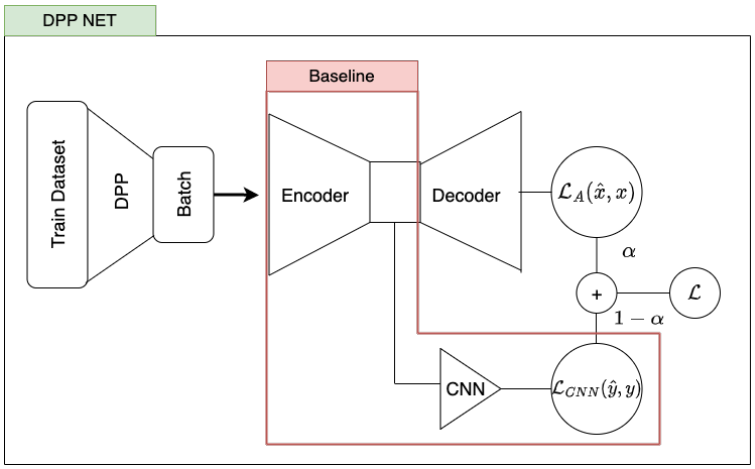


Figura 4.11: Baseline y DPP NET: Esta figura muestra la arquitectura del baseline con el que se compararán los resultados de DPP NET.

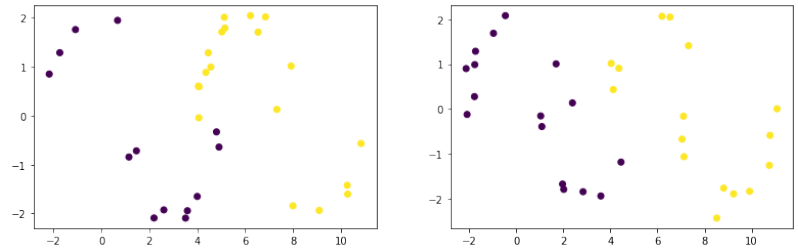
La Figura 4.2 detalla el esqueleto utilizado para el entrenamiento de la red DPP NET. La arquitectura del *Autoencoder* es la misma presentada en Figura 4.9 y la red encargada de clasificar los ejemplos es la presentada en Figura 4.8. Para ser comparables ambas metodologías, el *baseline* se re-define como la Figura 4.11. Los parámetros utilizados en los experimentos se detallan en Tabla C.5.

Capítulo 5

Resultados y Análisis

5.1. Muestreo Clase Minoritaria

Los procesos puntuales determinantales permiten muestrear subconjuntos de datos de tal forma que exista una alta repulsión entre sus elementos. Esta repulsión depende de una métrica de distancia que es complicada de construir en datos con dimensiones altas y que puede variar según la tipología del dato.



(a) Sampling Uniforme (b) Sampling DPP

Figura 5.1: Comparación de un muestreo de tamaño 32 sobre el dataset de clasificación binaria en 2 dimensiones utilizando una estrategia uniforme y una con DPP.

Cuando se trabaja en un dataset de 2 dimensiones, la métrica euclidiana rescata perfectamente la distancia entre pares de ejemplos. La primera parte del experimento tiene como objetivo mostrar la diferencia entre un sampling uniforme y un sampling mediante un DPP en 2 dimensiones. Como se puede ver en la Figura 5.1 (a), un método uniforme samplea en función del desbalance del dataset $\frac{C_0}{C_1} = 1/2$, en este caso 13 elementos de la clase 0 y 19 de la clase 1. Por otro lado, al samplear mediante un DPP en Figura 5.1 (b), se asigna una menor probabilidad a muestrear puntos que se encuentren espacialmente cercanos, por lo que no se espera una alta densidad en C_1 como en el caso uniforme. El experimento efectivamente comprueba este resultado, pues se obtienen 16 puntos para la clase 0 y 16 puntos en la clase 1.

Iterando el ejercicio una mayor cantidad de veces ($n = 100$) y promediando sus resultados se obtiene la Figura 5.2. Como se puede ver en esta figura, el método uniforme en promedio samplea en función del desbalance del dataset (barras azules) y con una alta varianza. En cambio, el sampleo con un DPP (barras naranjas) muestrea de manera más homogénea, cercano a la proporción 1:1 entre clases y con una menor varianza.

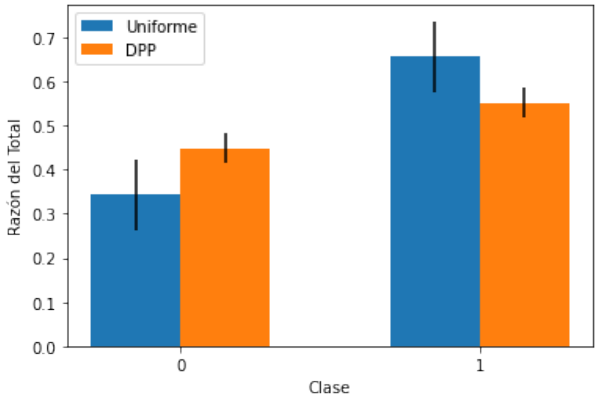


Figura 5.2: Razón del total promedio ($n = 100$) de ejemplos sampleados en el dataset de 2 dimensiones. Las barras azules representan las proporciones para un sampling uniforme y las barras naranjas para el sampling con DPP, las barras negras verticales corresponden a la varianza de los datos.

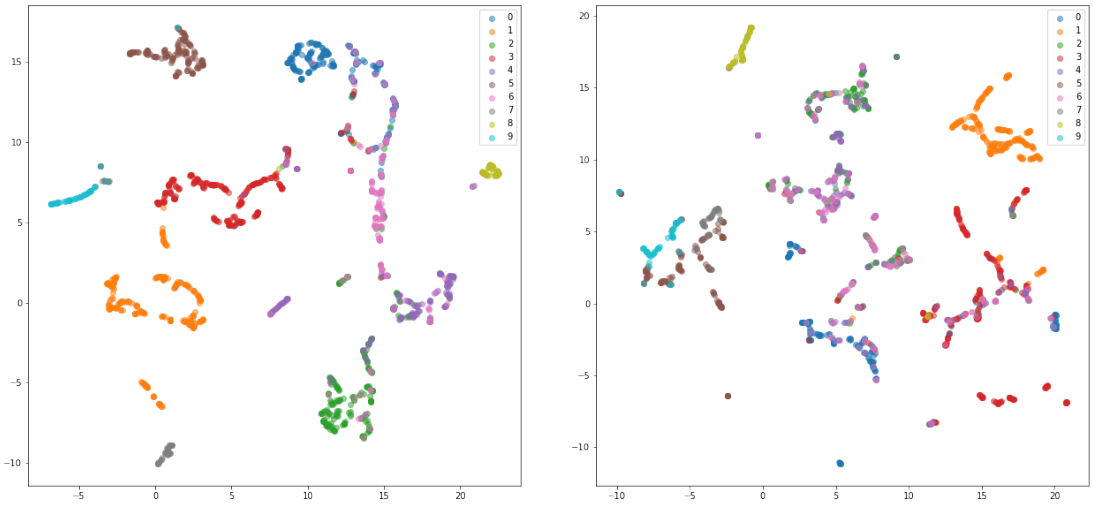
Lo anterior permite concluir que la metodología de sampleo con un DPP permite crear consistentemente *batches* con áreas de menor densidad de puntos y mayor presencia de elementos de las clases minoritarias.

En esta segunda parte del primer experimento, se repite el mismo ejercicio con el dataset *Fashion MNIST* pero ahora comparando las distintas métricas de distancia (que utiliza el DPP) presentadas en la metodología. Con el fin de disminuir el costo de muestreo, se considera el siguiente conjunto reducido de ejemplos (Tabla 5.1) pero que sigue la misma distribución que el presentado en la Tabla 4.2.

Tabla 5.1: Conjunto de Validación - Clasificación Multiclase. Total = 1600.

Clase	0	1	2	3	4	5	6	7	8	9
Cantidad	250	250	250	250	150	150	150	50	50	50
(%)	15.62	15.62	15.62	15.62	9.37	9.37	9.37	3.13	3.13	3.13

Para el aprendizaje de la métrica de distancia basados en el *Autoencoder* o en la red *Oneshot*, se mantuvo la arquitectura presentada en la sección anterior pero cambiando el tamaño del espacio latente a uno de largo 16 y se entrenan con el mismo conjunto de entrenamiento (Tabla 4.2). Con el único fin de visualizar las representaciones, se aplicó una técnica de reducción de dimensionalidad *UMAP* [24]



(a) Representación Oneshot (b) Representación Autoencoder

Figura 5.3: Visualización en 2 dimensiones de Fashion-MNIST mediante la técnica de reducción dimensionalidad UMAP utilizada sobre la representación latente entregada por el *Autoencoder* y la red *Oneshot* luego de ser entrenados.

Como se aprecia en Figura 5.3 la técnica UMAP calculada sobre la representación entregada por la red *Oneshot* (Figura 5.3 (a)) y *Autoencoder* (Figura 5.3 (b)) permite verificar que las clases se encuentran correctamente separadas (diferencias entre-clases, visualmente mejor en Figura 5.3 (a)) y que por sobre todo, existen "clusters" dentro de una misma clase (diferencias intra-clase) que el DPP permitiría representar. No se utilizará esta representación *UMAP* para los experimentos pero está dentro de los trabajos a realizar en el futuro.

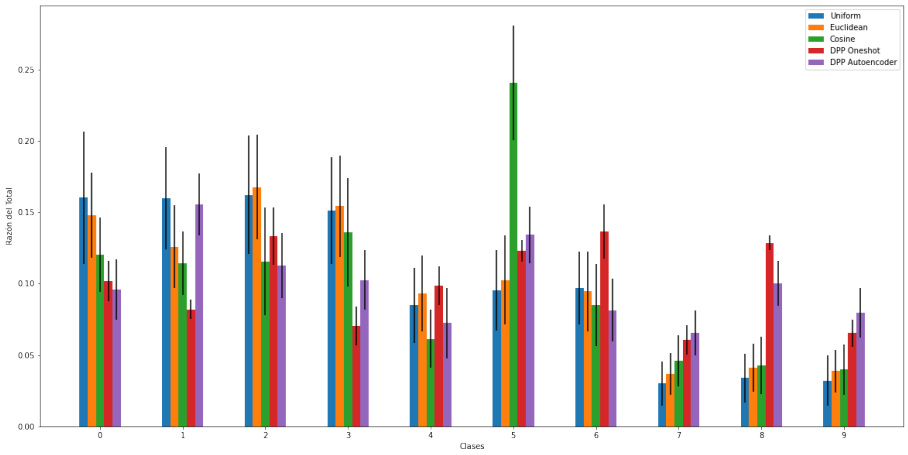


Figura 5.4: Razón del total promedio ($n = 50$) de ejemplos muestreados en el dataset Fashion-MNIST. La barra azul corresponde a un sampleo uniforme y el resto de barras corresponden a distintas métricas utilizadas para el sampleo con un DPP.

La Figura 5.4 corresponde a la proporción de ejemplos sampleados en $n = 50$ iteraciones entre las distintas clases y para cada una de las métricas propuestas. Notar que la métrica euclidiana junto a un kernel RBF (barra naranja) tiene un comportamiento similar al de un sampleo uniforme (barra azul) pues se ve afectada por la maldición de la dimensionalidad,

es decir, la métrica euclidiana no permite diferenciar clases debido a la alta dimensión de los vectores. Por otro lado, si bien la magnitud de los vectores no afecta a la distancia coseno (barra verde) y debiese ser más robusta a la dimensión, esta tampoco muestra un comportamiento considerablemente distinto al de los 2 anteriores.

Ahora bien, la distancia euclidana aplicada sobre una representación de baja dimensionalidad (*Oneshot* y *Autoencoder*, barra roja y morada respectivamente) si muestran un comportamiento considerablemente distinto pues samplean con mayor frecuencia las clases minoritarias 7, 8 y 9.

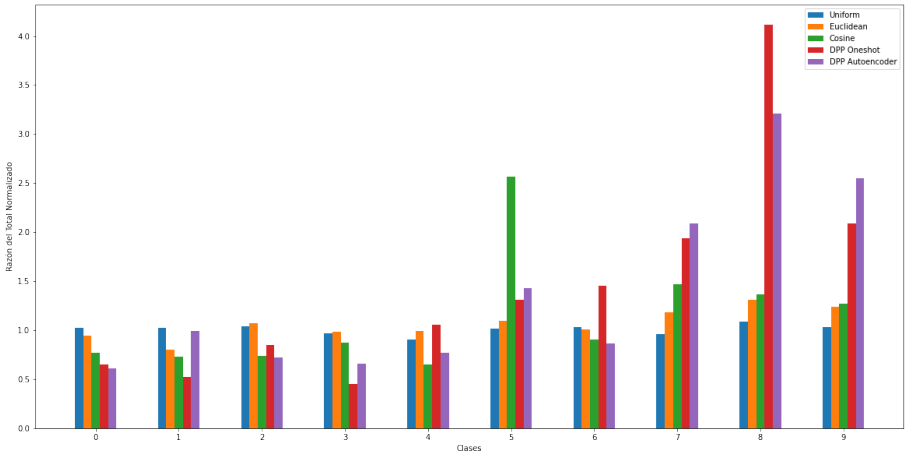


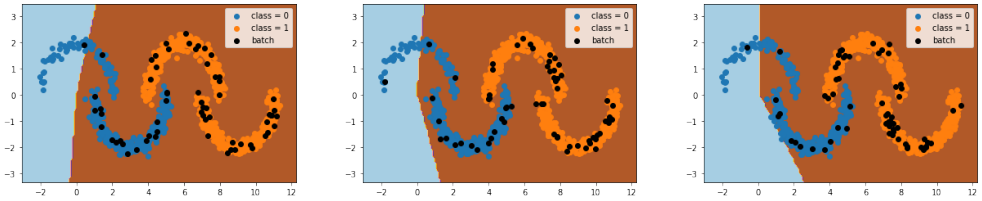
Figura 5.5: Razón del total normalizada promedio (n = 50) de ejemplos muestreados en el dataset Fashion-MNIST. La normalización se realizó en función del desbalance del dataset.

Normalizando por la proporción de ejemplos en cada clase con respecto al total, se obtiene la Figura 5.5. Un comportamiento uniforme debería reflejarse como un valor 1 en la gráfica (samplear en función del desbalance del dataset). Notar que los métodos *Oneshot* y *Autoencoder* samplean de 3 a 4 veces más de lo esperado a las clases minoritarias y con una menor varianza, un comportamiento muy similar a lo que ocurría en el caso binario.

Con lo anterior y dado que nuestro objetivo es utilizar un DPP centrado en una métrica que distinga las clases y samplee con mayor probabilidad a las clase minoritarias, se valida el uso de *Oneshot* y *Autoencoder* como las métricas que se usarán en las siguientes secciones. Se propone como trabajo futuro la búsqueda de métricas que no dependan del entrenamiento de un red para realizar esta tarea y que tengan efectos similares a los presentados (por ejemplo, la distancia fraccional).

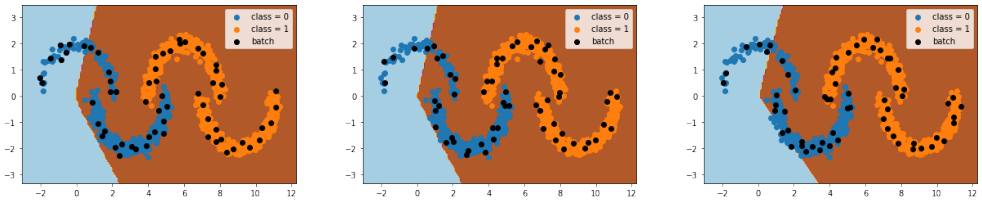
5.2. Fast DPP

El algoritmo propuesto en Algoritmo 4 se fundamenta en las primeras iteraciones del entrenamiento de una red neuronal. Para problemas de clasificación de baja dimensionalidad y una cantidad acotada de clases, la región de decisión se logra aproximar en unas cuantas iteraciones, mientras que para otros problemas más complejos, esto ocurrirá después.



(a) Iteración 1. (b) Iteración 2. (c) Iteración 3.

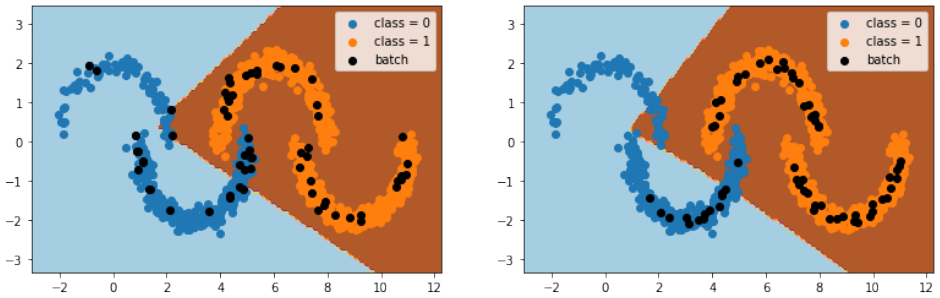
Figura 5.6: Primeras iteraciones SGD en el problema de clasificación binaria. Las zonas de color celeste y anaranjado delimitan la región de decisión del problema.



(a) Iteración 1. (b) Iteración 2. (c) Iteración 3.

Figura 5.7: Primeras iteraciones Fast DPP en el problema de clasificación binaria. Las zonas de color celeste y anaranjado delimitan la región de decisión del problema.

Las Figuras 5.6 y 5.7 muestran las primeras 3 iteraciones del entrenamiento de una red neuronal que resuelve el problema de clasificación binaria en 2 dimensiones. Al igual que en la sección anterior, es posible notar que la selección de un *batch* mediante el uso de un DPP (Figura 5.7) tiene una mayor proporción de la clase minoritaria en comparación a la selección uniforme (Figura 5.6). En consecuencia, la región de decisión luego de 3 iteraciones es mejor en Figura 5.7 (c) pues clasifica correctamente más ejemplos de la clase minoritaria (clase 0) y mantiene los correctamente clasificados de la clase mayoritaria (clase 1).



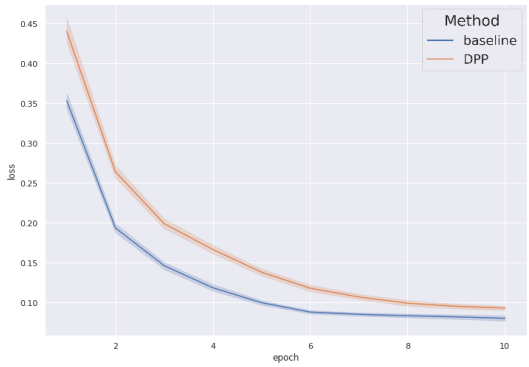
(a) Última iteración SGD. (b) Última iteración DPP.

Figura 5.8: Última iteración de las metodologías SGD y DPP si no se descartan ejemplos como lo propone Fast DPP.

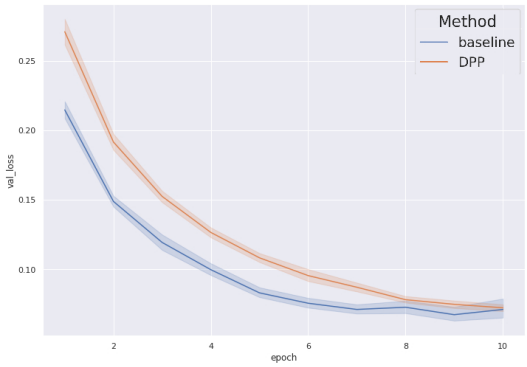
Si no descartamos ejemplos como lo propone *Fast DPP*, es decir, descartar únicamente los ejemplos muestreados en cada *mini batch* y no los M que se proponen, las últimas iteraciones del algoritmo serían como lo muestra la Figura 5.8 (b) pues el conjunto del cual samplear es reducido y los ejemplos de la clase minoritaria ya fueron muestreados en su totalidad.

Por otro lado, los rendimientos de ambos modelos en Figura 5.8 son bastante parecidos pues ambos observarían todos los ejemplos del dataset y no habría mayor diferencia (salvo en el tiempo de entrenamiento).

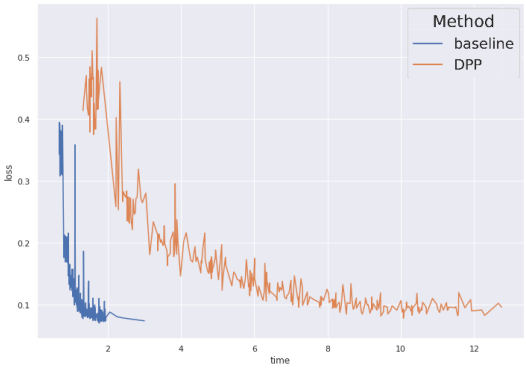
Lo anterior justifica la utilización del algoritmo *Fast DPP* que hace uso de la capacidad del DPP para samplear ejemplos diversos durante las primeras iteraciones de cada época (alcanzando así una "buena" región de decisión) sin la necesidad de mirar el resto de ejemplos, acelerando así el entrenamiento.



(a) Train loss según época de entrenamiento.



(b) Validation loss según época de entrenamiento.



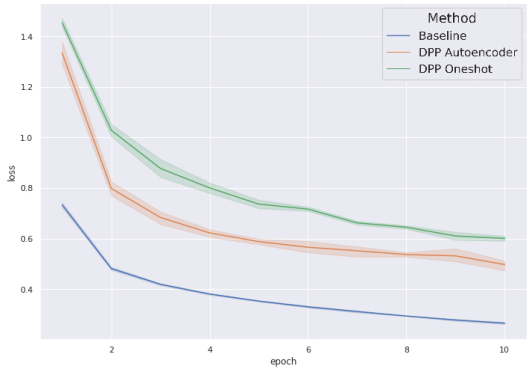
(c) Validation loss según tiempo de entrenamiento.

Figura 5.9: Comparación de resultados de las metodologías Baseline y DPP según la métrica de *loss* en el dataset de clasificación binaria promediando 30 iteraciones del experimento y $M = 100$.

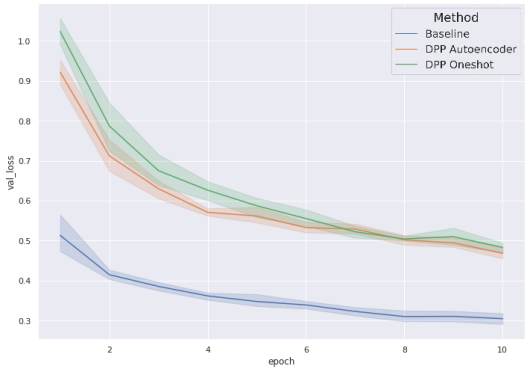
Con respecto al entrenamiento en 10 épocas con un tamaño de sampling inicial $M = 100$ (es decir, descartando $M - k = 100 - 64 = 36$ ejemplos en cada iteración con k el tamaño del *batch*), según la Figura 5.9 (a) y (b), es claro que el método *baseline* es superior durante todo el entrenamiento al método *Fast DPP* según la métrica de *loss*. Lo anterior es un resultado esperable de un algoritmo que no mira todos los ejemplos pues su función es la de acelerar el entrenamiento, lamentablemente, esto tampoco es logrado como lo muestra Figura 5.9 (c) pues la arquitectura de la red es poco profunda y los datos con los que trabaja de baja dimensionalidad. En consecuencia, el costo base de utilizar un DPP es superior a la de usar todos los datos para el entrenamiento.

Cuando esto es aplicado sobre un conjunto de mayor complejidad como *Fashion MNIST*

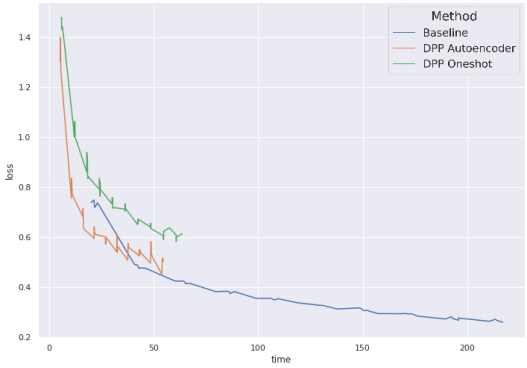
donde el costo por iteración es más elevado (por la profundidad de la red y la dimensión de los datos), es cuando se puede apreciar la intención detrás de *Fast DPP*.



(a) Train loss según época de entrenamiento.



(b) Validation loss según época de entrenamiento.



(c) Validation loss según tiempo de entrenamiento.

Figura 5.10: Comparación de resultados de las metodologías Baseline y DPP según la métrica de *loss* en el dataset de clasificación multiclase promediando 5 iteraciones del experimento y $M = 400$.

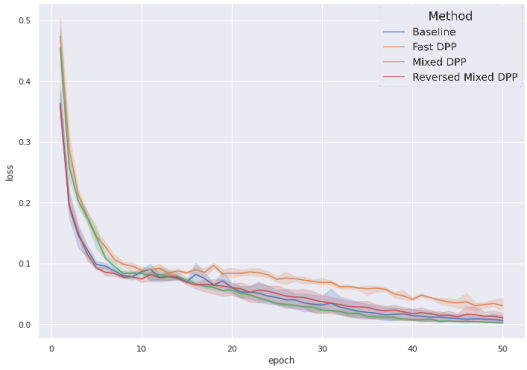
Como se puede notar en Figura 5.10 (a) y (b), el rendimiento del modelo *baseline* es superior durante todas las épocas de entrenamiento lo que es de esperar dada la cantidad de ejemplos con los que entrena. Además, el rendimiento en *Train Loss* es superior para la métrica *Autoencoder* (curva naranja) en comparación a la métrica *Oneshot* (curva verde) pero similares en *Val Loss*.

Lo que resulta interesante de este resultado es que la metodología utilizando *Fast DPP* alcanza durante las primeros 30 segundos aproximadamente, un mejor rendimiento en *Train Loss* que el modelo *baseline* (para la métrica DPP *Autoencoder*). Esto da a lugar a su utilización como un "iniciador" pues existe un intervalo de tiempo en el que su uso es mejor que el modelo *baseline* y esto es lo que motiva la propuesta de la Sección 4.1.2 donde se prueban esquemas mixtos de entrenamiento utilizando estos resultados.

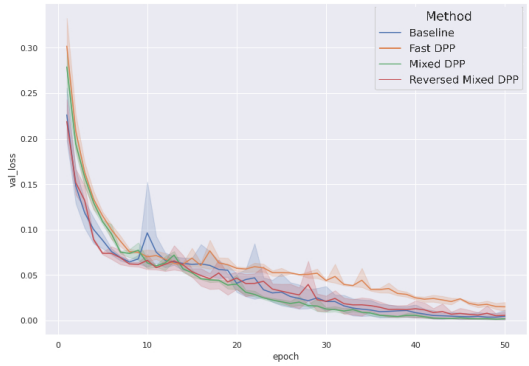
5.3. Mixed DPP

En la Sección 5.2, se muestra que durante los primeros 30 segundos del entrenamiento (5 épocas aproximadamente) de *Fast DPP* para el problema de clasificación multiclase, la metodología *DPP Autoencoder* superó en rendimiento al *baseline* definido. Como nuestro objetivo es mejorar el tiempo y rendimiento de la red, se propone utilizar un esquema mixto de entrenamiento llamado *Mixed DPP* (detallado en Algoritmo 5) donde vamos a considerar la época de cambio de estrategia $l_m = 5$.

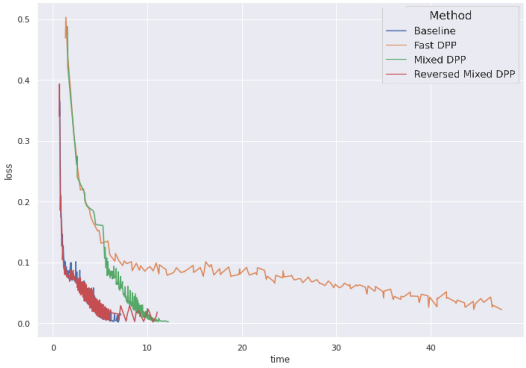
Sobre el dataset Tabla 4.1 del problema de clasificación binario y entrenando durante 50 épocas, se obtienen los siguientes resultados:



(a) Train loss según época de entrenamiento.



(b) Validation loss según época de entrenamiento.



(c) Validation loss según tiempo de entrenamiento.

Figura 5.11: Comparación de resultados de las metodologías Baseline, Fast DPP, Mixed DPP y Reversed Mixed DPP según la métrica de *loss* en el dataset de clasificación binario promediando 10 iteraciones del experimento y $M = 100$.

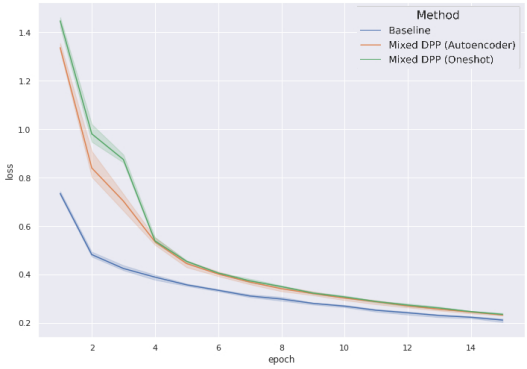
En este problema, la metodología *Fast DPP* no era de gran utilidad para disminuir el tiempo de entrenamiento pues los datos son de baja dimensionalidad y por tanto de tiempo de procesamiento reducido. Ahora bien, como es posible ver en Figura 5.11 (a) y (b), el rendimiento de la red *Mixed DPP* en *Train Loss* y *Val Loss* es superior a partir de las 12 épocas con respecto al *baseline* definido.

Este resultado es muy importante pues permitiría validar una de las hipótesis planteadas

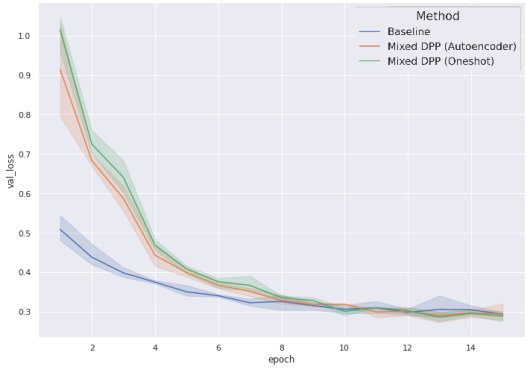
en el Capítulo 1 donde se menciona que las estrategias de entrenamiento inicial "acelerado" resultan en mejoras al rendimiento de la red, no obstante, el resultado puede ser engañoso al notar que la metodología *baseline* debería ser comparada en condiciones de tiempos de entrenamiento iguales y no en épocas como muestra la gráfica.

Según Figura 5.11 (c) el entrenamiento mediante *Mixed DPP* (curva verde) toma un mayor tiempo en completar 50 épocas (5 segundos más aproximadamente) que el entrenamiento *baseline* (curva azul) y por tanto, si buscamos un problema donde se pueda mostrar el valor que aporta *Fast DPP*, es necesario trabajar con un problema de alta dimensionalidad.

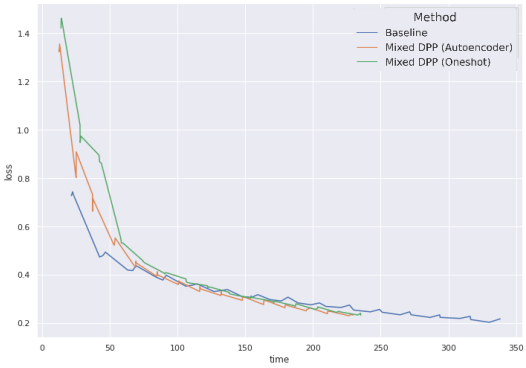
Por otro lado, no se destacan resultados importantes en la red *Reversed Mixed DPP*. Se esperaba que tuviera algún efecto en las últimas iteraciones al ajustar los ejemplos de la clase minoritaria (mejorando eventualmente alguna métrica de precisión sobre esa clase).



(a) Train loss según época de entrenamiento.



(b) Validation loss según época de entrenamiento.



(c) Validation loss según tiempo de entrenamiento.

Figura 5.12: Comparación de los resultados de las metodologías Baseline y Mixed DPP según la métrica de *loss* en el dataset de clasificación multiclase promediando 5 iteraciones del experimento y $M = 400$.

Al trabajar con el problema de clasificación multiclase *Fashion MNIST*, se obtienen los resultados mostrados en Figura 5.12. En este caso, el método *baseline* es superior a los métodos *Mixed DPP (Autoencoder)* y *Mixed DPP (Oneshot)* en *Train Loss* (Figura 5.12 (a)), ahora bien, con respecto al conjunto de validación, los resultados muestran que los 3 modelos tienen un rendimiento similar en *Val Loss* (Figura 5.12 (b)) e incluso las metodologías propuestas son ligeramente mejores (quizás eventualmente reduciendo el *overfitting*).

Un resultado relevante se da al analizar el tiempo de entrenamiento en Figura 5.12 (c) de las distintas metodologías. Es claro que el rendimiento de ambas redes propuestas es superior al del *baseline* alcanzando un resultado similar con una diferencia de hasta 100 segundos producto de la inicialización mediante *Fast DPP*. El hecho de que ambas redes propuestas hayan podido tener este resultado valida la segunda hipótesis del Capítulo 1 que propone la utilización de representaciones de baja dimensionalidad (obtenidas a partir del entrenamiento de una red) para el aprendizaje de la métrica de distancia utilizada por el DPP. A su vez, este resultado indica que justamente un esquema mixto de entrenamiento bien ajustado supera el rendimiento de una metodología clásica SGD con sampling uniforme, tal y como conjeturamos en la primera hipótesis de la Sección 1.

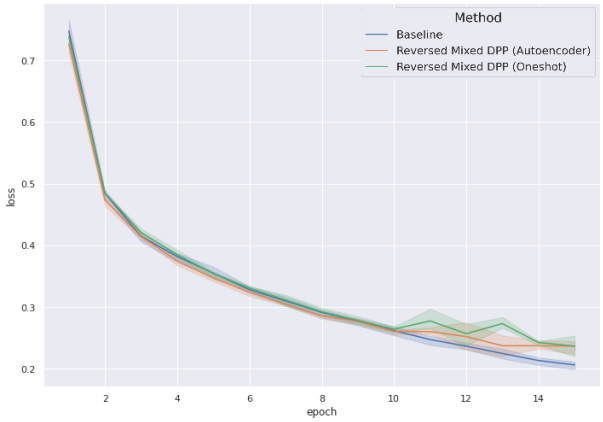
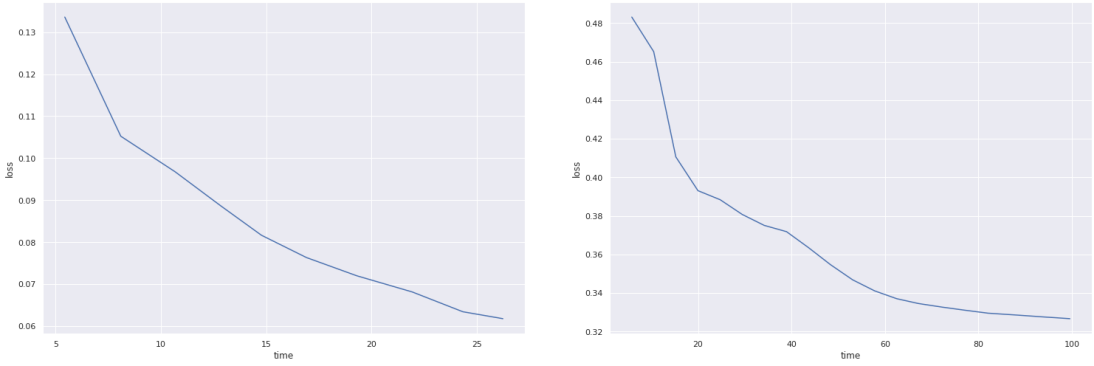


Figura 5.13: Comparación de los resultados de las metodologías Baseline y Reversed Mixed DPP según la métrica de *loss* en el dataset de clasificación multiclase promediando 5 iteraciones del experimento y $M = 400$.

Por otro lado, los resultados de *Reversed Mixed DPP* en la Figura 5.13 tampoco fueron exitosos, pues, al entrenar con menos ejemplos en las últimas épocas donde la reducción de la función de *loss* es de menor magnitud en comparación al inicio, resultó en una pérdida de *performance* general. Se analiza la posibilidad de trabajo a futuro con respecto a una metodología que cambie la estrategia de sampleo de *batches* cuando la disminución de *loss* sea despreciable.



(a) Tiempo Entrenamiento Oneshot.

(b) Tiempo Entrenamiento Autoencoder.

Figura 5.14: Tiempo de entrenamiento de las redes Oneshot y Autoencoder en el dataset de lasificación multiclase.

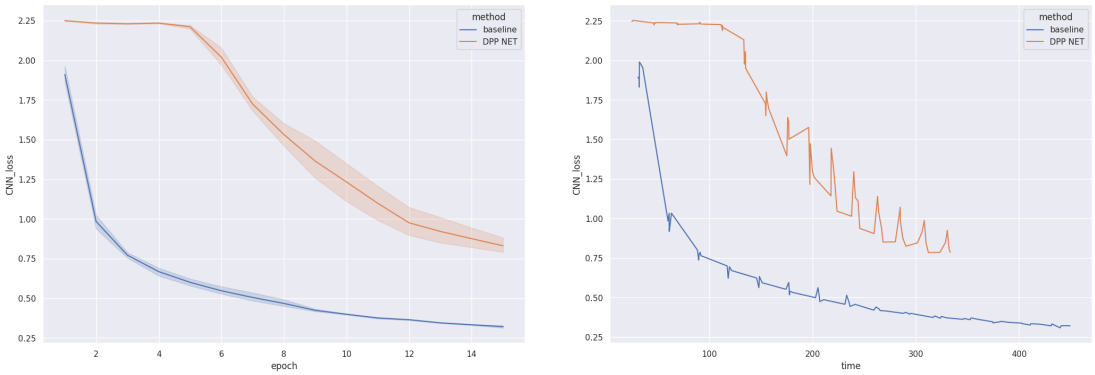
Los resultados que hemos obtenido hasta ahora son dependientes del aprendizaje de la métrica mediante *Oneshot* o aprendiendo una representación de baja dimensionalidad mediante un *Autoencoder*. El tiempo de entrenamiento se reporta en Figura 5.14. Si bien este debería ser sumado al tiempo de entrenamiento en Figura 5.12 (c), para el caso de la métrica con *Oneshot* (Figura 5.12 (a)) el resultado no cambia pero para el *Autoencoder* (Figura 5.12 (b)) el resultado resulta equivalente a la metodología *baseline*.

Este detalle puede ser tratado de múltiples formas, desde el uso de *Autoencoders* o redes *Oneshot* menos profundas (con el fin de reducir el tiempo de entrenamiento), la búsqueda de métricas que no dependan del entrenamiento de una red (como las distancias de *Minkowski* discutidas en la Sección 4.2.2) o la que se propone en el Algoritmo 7 que se denomina DPP NET cuyo objetivo es el de entrenar en paralelo la red encargada de obtener la representación mientras se resuelve el problema de clasificación utilizando un sampleo con DPP. La siguiente sección detalla los resultados obtenidos por esta red.

5.4. DPP NET

Las propuestas de métricas de distancia que hemos desarrollado durante la tesis incluyen el entrenamiento de una red *Autoencoder* o *Oneshot* que podría eventualmente tomar más tiempo que el entrenamiento de la red encarga de clasificar. Si bien esto depende del problema en específico y la profundidad con la que se trabaje cada red, se hace necesario la búsqueda de alternativas con las que realizar este proceso.

La propuesta en este escenario es la de entrenar en paralelo la red encargada de definir la métrica y la red encargada de la clasificación, en este ejemplo, entrenar un *Autoencoder* encargado de obtener la representación de baja dimensionalidad para utilizarla en el DPP y a su vez, utilizarla como *input* para la red encargada de la clasificación. Por otro lado, con el fin de hacer la red lo más comparable posible a un *baseline*, se hace uso de *Fast DPP* para la reducción en los tiempos de entrenamiento.



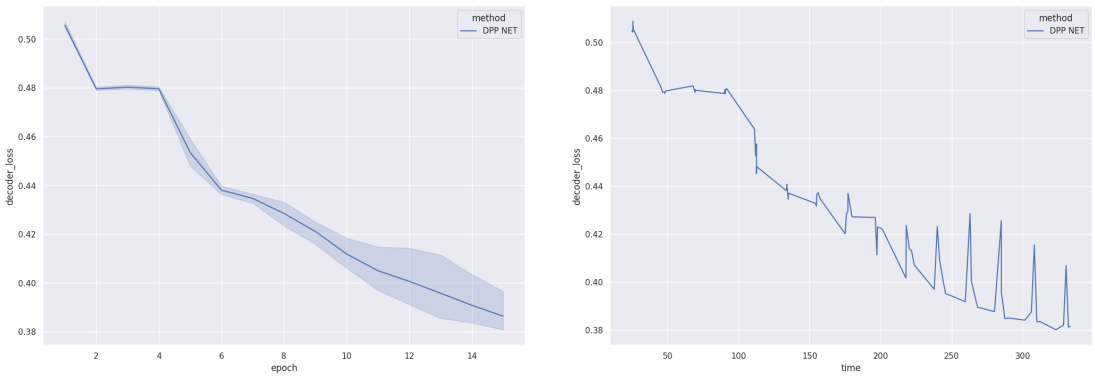
(a) Train loss según época de entrenamiento. (b) Train loss según tiempo de entrenamiento.

Figura 5.15: Comparación de los resultados de las metodologías Baseline y DPP NET según la métrica de *loss* (en la subred encargada de clasificar) en el dataset de clasificación multiclase promediando 5 iteraciones del experimento y $M = 400$.

La Figura 5.15 muestra los resultados obtenidos por la metodología DPP NET que hace uso de la reducción de tiempo de entrenamiento gracias a *Fast DPP*. Como se puede apreciar,

el resultado alcanzado es desfavorable para nuestra metodología pues se ve ampliamente superada por el *baseline*.

Un resultado interesante del entrenamiento se puede ver alrededor de la época 4, pues el *CNN loss* comienza a disminuir de manera considerable en la metodología DPP NET. Lo anterior puede ser explicado por el entrenamiento del *Autoencoder* cuya mejora (aunque pequeña) en rendimiento hace que la métrica de distancia consumida por el DPP sea más precisa.



(a) Train loss según época de entrenamiento. (b) Train loss según tiempo de entrenamiento.

Figura 5.16: Resultados del entrenamiento del *Autoencoder* en la metodología *DPP NET* según la métrica de *loss* para el problema de clasificación multiclase promediando 5 iteraciones del experimento y $M = 400$.

De los parámetros del experimento descritos en Tabla C.5, uno de gran importancia en el entrenamiento de la red DPP NET es el $\alpha = 1/3$ que controla en qué proporción impacta en *loss* general el error de cada red por separado (como lo describe la arquitectura en Figura 4.2). En este caso, notar que en la Figura 5.16 (b) alcanza un *loss* de 0.38 a los 320 segundos aproximadamente, a diferencia del entrenamiento del *Autoencoder* en Figura 5.14 (b) que alcanza un 0.32 en 100 segundos aproximadamente. En otras palabras, en DPP NET el *Autoencoder* aún no alcanzaba un *performance* aceptable que permitiera obtener una representación de baja dimensionalidad precisa para usar en el DPP.

Distintas estrategias pueden tomarse a partir de este punto, como controlar el valor de α de manera dinámica con el fin de priorizar el entrenamiento del *Autoencoder* en un principio y luego que este ya tenga un rendimiento aceptable, priorizar el entrenamiento de la red que clasifica (por ejemplo, desconectando o disminuyendo el entrenamiento del *Autoencoder*). También se puede trabajar en la creación de una nueva estructura que utilice un aprendizaje de métrica como *Oneshot Learning* que podría disminuir los tiempos de entrenamiento. Finalmente, hacer uso de un esquema mixto de entrenamiento similar al propuesto en *Mixed DPP* también podría resultar en mejoras sustanciales al modelo.

Capítulo 6

Conclusiones

6.1. Comentarios

En esta tesis se ha mostrado que las metodologías *Fast DPP* y su modificación posterior, conocida como *Mixed DPP*, permiten acelerar el proceso de entrenamiento de una red neuronal sin comprometer su rendimiento. Mediante el uso de *Fast DPP*, fue posible aplicar un Proceso Puntual Determinantal (DPP) incluso con su tiempo de muestreo elevado, y los resultados mostraron que en conjuntos de alta dimensión, la metodología *Fast DPP* supera en rendimiento a una red de referencia con muestreo uniforme durante los primeros 30 segundos de entrenamiento. Este resultado impulsó el desarrollo de *Mixed DPP*, que haciendo uso de esta herramienta en las primeras épocas de entrenamiento, logra alcanzar el rendimiento final esperado de la red en un tiempo considerablemente menor como lo muestra la Figura 5.12.

Por otro lado, para el conjunto de clasificación binario, el resultado anterior no era cierto pues la dimensión del conjunto de entrenamiento era baja y la red poco profunda, resultando en que el uso de un DPP aumentara el tiempo de entrenamiento. Si se comparan los resultados en términos de épocas, para este conjunto de datos, la metodología *Mixed DPP* mostró mejores resultados que una metodología *baseline*.

Con respecto a la modificación *Reversed Mixed DPP*, esta no mostró resultados favorables en ninguno de los problemas de clasificación pero no se descarta una investigación posterior en este asunto pues el cambio de estrategia de sampleo puede resultar en mejoras (aunque marginales) del *performance* de la red.

Para la red propuesta *DPP NET* tampoco se tuvieron resultados destacables, principalmente pues el entrenamiento del *Autoencoder* requiere de una mayor cantidad de iteraciones en comparación a la red encargada de la clasificación. De todas formas, se proponen distintas líneas de investigación con respecto a esta metodología que podrían resultar en arquitecturas de entrenamiento en paralelo más eficientes.

Finalmente, el uso de redes que aprenden representaciones de baja dimensionalidad de los datos y que a posteriori permiten definir métricas de distancia (o similitud), fueron cruciales para que el muestreo de los DPP fuera preciso y no se viera afectado por la maldición de la dimensionalidad, como ocurre con la distancia euclidiana. En particular, se mostró que la representación de un *Autoencoder* y el aprendizaje de la métrica con *Oneshot* utilizada sobre los DPP, permiten obtener muestreos más representativos de las diferencias intra-clase y entre-clases en comparación a un muestreo uniforme.

Con lo anterior mencionado, se corroboran las hipótesis del trabajo y se cumplen los objetivos generales y particulares propuestos para esta tesis.

6.2. Trabajo a Futuro

Con el objetivo de complementar el trabajo presentado con ideas que no fueron probadas pero que podrían ser perfectamente objeto de trabajo futuro, se detalla una lista de posibles rutas y métodos que podrían resultar en mejoras a los resultados presentados:

1. Parámetros: validar los parámetros sobre los cuales los métodos aún continúan funcionando, por ejemplo, probar hasta qué punto el aumento del *batch size* mantiene los resultados mostrados sin perjudicar el tiempo de sampling del DPP. También se puede estudiar cuántas épocas de inicialización con *Mixed DPP* son óptimas para el rendimiento de la red.

2. Optimización DPP: si bien la complejidad de los DPP es de $O(Nk^3)$, la implementación utilizada en esta tesis pudo haber sido más óptima al programar los métodos en algún lenguaje de programación con un compilado más rápido como C++. Por otro lado, existen implementaciones en la literatura que aproximan los DPP mediante el uso de MCMC como lo detalla [25] y que tienen una complejidad considerablemente menor.

3. Reducción de Dimensionalidad: en esta tesis se utilizaron 2 métodos para reducir la dimensionalidad de los datos, las redes *Autoencoder* y *Oneshot*. Existen técnicas clásicas para reducir la dimensión de los datos, PCA, TSNE, UMAP, entre otras. La propuesta es implementar las redes propuestas en esta tesis pero utilizando la distancia euclidiana (coseno o una *Minkowski* para algún p) aplicada sobre estas representaciones como métrica.

4. DPP NET: esta red no tuvo los resultados esperados pues el *Autoencoder* requiere de múltiples iteraciones y épocas para su entrenamiento. La construcción de una arquitectura que implemente una red *Oneshot* (que se entrene con *contrastive loss* o incluso *triplet loss*) y su entrenamiento en paralelo podría reducir este problema y mostrar resultados favorables.

5. Nuevos Datasets: los resultados fueron mostrados en una cantidad aún acotada de ejemplos y con una tipología de datos en específico (imágenes). Se propone iterar este trabajo con nuevos datasets y problemas, por ejemplo, en series de tiempo, se pueden aplicar las metodologías de esta tesis y comparar con otras derivadas de la métrica *Dynamic Time Warping* (DTW) que funciona bien con este tipo de datos.

6. Cambios de Estrategia de Sampleo: la idea fundamental detrás de *Reversed Mixed DPP* era la de cambiar la estrategia de sampleo (al usar un DPP) cuando la *loss* ya no estaba teniendo variaciones de valor. Esta idea tiene mucho potencial pero se vio perjudicada pues *Fast DPP* no entrena con todos los datos y los cambios incluso empeoraban el *performance*. La propuesta es investigar acerca de estrategias de sampleo alternativas que se puedan implementar cuando un método estándar (SGD con sampleo uniforme) ya no disminuya considerablemente la *loss* en las iteraciones.

Bibliografía

[1] Zhang, C., Öztireli, C., Mandt, S., y Salvi, G., "Active mini-batch sampling using repulsive point processes," 2018, doi:10.48550/ARXIV.1804.02772.

[2] Kulesza, A., "Determinantal point processes for machine learning," Foundations and Trends® in Machine Learning, vol. 5, no. 2-3, pp. 123–286, 2012, doi:10.1561/2200000044.

[3] Mehta, M. y Gaudin, M., "On the density of eigenvalues of a random matrix," Nuclear Physics, vol. 18, pp. 420–427, 1960, doi:https://doi.org/10.1016/0029-5582(60)90414-4.

[4] Ginibre, J., "Statistical ensembles of complex, quaternion, and real matrices," Journal of Mathematical Physics, vol. 6, no. 3, pp. 440–449, 1965, doi:10.1063/1.1704292.

[5] Johansson, K., "Determinantal processes with number variance saturation," Communications in Mathematical Physics, vol. 252, pp. 111–148, 2004, doi:10.1007/s00220-004-1186-4.

[6] Burton, R. y Pemantle, R., "Local characteristics, entropy and limit theorems for spanning trees and domino tilings via transfer-impedances," 2004, doi:10.48550/ARXIV.MATH/0404048.

[7] Borodin, A., Diaconis, P., y Fulman, J., "On adding a list of numbers (and other one-dependent determinantal processes)," 2009, doi:10.48550/ARXIV.0904.3740.

[8] Fitch, F. B., "Warren s. mcculloch and walter pitts. a logical calculus of the ideas immanent in nervous activity. bulletin of mathematical biophysics, vol. 5 (1943), pp. 115–133.," The Journal of Symbolic Logic, vol. 9, no. 2, pp. 49–50, 1944.

[9] Wash, A., "Gradient descent and its types," 2022, https://www.analyticsvidhya.com/blog/2022/07/gradient-descent-and-its-types/.

[10] Wilson, D. y Martinez, T. R., "The general inefficiency of batch training for gradient descent learning," Neural Networks, vol. 16, no. 10, pp. 1429–1451, 2003, doi:https://doi.org/10.1016/S0893-6080(03)00138-2.

[11] Radiuk, P., "Impact of training set batch size on the performance of convolutional neural networks for diverse datasets," Information Technology and Management Science, vol. 20, pp. 20–24, 2017, doi:10.1515/itms-2017-0003.

[12] Indolia, S., Goswami, A. K., Mishra, S., y Asopa, P., "Conceptual understanding of convolutional neural network- a deep learning approach," Procedia Computer Science, vol. 132, pp. 679–688, 2018, doi:https://doi.org/10.1016/j.procs.2018.05.069. International Conference on Computational Intelligence and Data Science.

[13] Zhang, Y., "A better autoencoder for image: Convolutional autoencoder," 2018.

[14] O' Mahony, N., Campbell, S., Carvalho, A., Krpalkova, L., Hernandez, G. V., Hara-

panahalli, S., Riordan, D., y Walsh, J., "One-shot learning for custom identification tasks; a review," Procedia Manufacturing, vol. 38, pp. 186–193, 2019, doi:https://doi.org/10.1016/j.promfg.2020.01.025. 29th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM 2019), June 24-28, 2019, Limerick, Ireland, Beyond Industry 4.0: Industrial Advances, Engineering Education and Intelligent Manufacturing.

[15] Ghandi, R., "Siamese network & triplet loss," 2018, https://towardsdatascience.com/siamese-network-triplet-loss-b4ca82c1aec8.

[16] Kingma, D. P. y Ba, J., "Adam: A method for stochastic optimization," 2014, doi:10.48550/ARXIV.1412.6980.

[17] Loshchilov, I. y Hutter, F., "Online batch selection for faster training of neural networks," 2015, doi:10.48550/ARXIV.1511.06343.

[18] Joseph, K. J., Teja R, V., Singh, K., y Balasubramanian, V. N., "Submodular batch selection for training deep neural networks," en Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19, pp. 2677–2683, International Joint Conferences on Artificial Intelligence Organization, 2019, doi:10.24963/ijcai.2019/372.

[19] Biscio, C. A. N. y Lavancier, F., "Quantifying repulsiveness of determinantal point processes," Bernoulli, vol. 22, 2016, doi:10.3150/15-bej718.

[20] Vert, J., Tsuda, K., y Schölkopf, B., "A primer on kernel methods," Kernel Methods in Computational Biology, 35-70 (2004), 2004.

[21] Zimek, A., Schubert, E., y Kriegel, H.-P., "A survey on unsupervised outlier detection in high-dimensional numerical data," Statistical Analysis and Data Mining: The ASA Data Science Journal, vol. 5, no. 5, pp. 363–387, 2012, doi:https://doi.org/10.1002/sam.11161.

[22] Aggarwal, C. C., Hinneburg, A., y Keim, D. A., "On the surprising behavior of distance metrics in high dimensional space," en Database Theory — ICDT 2001 (Van den Bussche, J. y Vianu, V., eds.), (Berlin, Heidelberg), pp. 420–434, Springer Berlin Heidelberg, 2001.

[23] Han Xiao, K. R. y Vollgraf, R., "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," arXiv preprint arXiv:1708.07747, 2017, https://arxiv.org/abs/1708.07747.

[24] McInnes, L., Healy, J., y Melville, J., "Umap: Uniform manifold approximation and projection for dimension reduction," 2018, doi:10.48550/ARXIV.1802.03426.

[25] Anari, N., Gharan, S. O., y Rezaei, A., "Monte carlo markov chain algorithms for sampling strongly rayleigh distributions and determinantal point processes," 2016, doi:10.48550/ARXIV.1602.05242.

Anexos

Anexo A. Acrónimos

Lista de acrónimos:

- ADAM: Adaptative Moment Optimization
- B: Matriz de Descomposición de Cholesky
- CNN: Convolutional Neural Network
- DPP: Determinantal Point Process
- FC: Fully Connected
- FFNN: Feedforward Neural Network
- FN: False Negatives
- FP: False Positives
- K: Kernel Marginal
- L: Matriz de L-ensamblaje
- MCMC: Markov Chain Monte Carlo
- MLP: Multilayer Perceptron
- PCA: Principal Component Analysis
- PDS: Poisson Disk Sampling
- RBF: Radial Basis Function
- SGD: Stochastic Gradient Descent
- TN: True Negatives
- TP: True Positives
- TSNE: t-Distributed Stochastic Neighbor Embedding
- UMAP: Uniform Manifold Approximation and Projection

Anexo B. Teoremas y Demostraciones

Teorema B.1 *Para cualquier* $A \subseteq \mathcal{Y}$

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

donde $I_{\bar{A}}$ *con 1's en la diagonal correspondiente a elementos de* $\bar{A} = \mathcal{Y} - A$. *Con esto*

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

Demostración Teorema B.1: Por inducción, si $A = \mathcal{Y}$ el resultado es trivial, supongamos que el resultado es cierto siempre que $\bar{A}$ tenga cardinalidad menor a k. Sea A tal que $|\bar{A}| = k > 0$ y sea i un elemento de $\mathcal{Y}$ tal que $i \in \bar{A}$, notar que

$$L + I_{\bar{A}} = \begin{pmatrix} L_{ii} + 1 & L_{i\bar{i}} \\ L_{\bar{i}i} & L_{\mathcal{Y}-\{i\}} + I_{\mathcal{Y}-\{i\}-A} \end{pmatrix},$$

por la multilinealidad del determinante, se tiene que

$$\begin{aligned} \det(L + I_{\bar{A}}) &= \begin{vmatrix} L_{ii} & L_{i\bar{i}} \\ L_{\bar{i}i} & L_{\mathcal{Y}-\{i\}} + I_{\mathcal{Y}-\{i\}-A} \end{vmatrix} + \begin{vmatrix} 1 & 0 \\ L_{\bar{i}i} & L_{\mathcal{Y}-\{i\}} + I_{\mathcal{Y}-\{i\}-A} \end{vmatrix} \\ &= \det\Big(L + I_{\overline{A \cup \{i\}}}\Big) + \det\Big(L_{\mathcal{Y}-\{i\}} + I_{\mathcal{Y}-\{i\}-A}\Big). \end{aligned}$$

Utilizando la hipótesis inductiva, en cada término, se tiene que

$$\det(L + I_{\bar{A}}) = \sum_{A \cup \{i\} \subseteq Y \subseteq \mathcal{Y}} \det(L_Y) + \sum_{A \subseteq Y \subseteq \mathcal{Y} - \{i\}} \det(L_Y) = \sum_{A \subseteq Y \subseteq \mathcal{Y}} \det(L_Y).$$

Lo que prueba el resultado. Finalmente, si k es la constante de proporcionalidad y $A = \emptyset$

$$1 = \sum_{A \subseteq Y \subseteq \mathcal{Y}} \mathcal{P}(\mathbf{Y} = Y) = \sum_{A \subseteq Y \subseteq \mathcal{Y}} k \cdot \det(L_Y) = k \det(L + I),$$

con lo que $k = \frac{1}{\det(L+I)}$ y se concluye la demostración.

Teorema B.2 *Un L-ensamblaje es un DPP cuyo kernel marginal es*

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

Demostración Teorema B.2: Utilizando el Teorema B.1, la probabilidad marginal de un conjunto A viene dada por

$$\begin{aligned}\mathcal{P}_L(A \subseteq \mathbf{Y}) &= \frac{\sum_{A\subseteq Y\subseteq \mathcal{Y}} \det(L_Y)}{\sum_{Y\subseteq \mathcal{Y}} \det(L_Y)} \\ &= \frac{\det(L + I_{\bar{A}})}{\det(L+I)} \\ &= \det\Big((L + I_{\bar{A}})(L+I)^{-1}\Big).\end{aligned}$$

Utilizando la igualdad $L(L+I)^{-1} = I - (L+I)^{-1}$ y definiendo $K = I - (L+I)^{-1}$ se obtiene que

$$\begin{aligned}\mathcal{P}_L(A \subseteq \mathbf{Y}) &= \det\Big(I_{\bar{A}}(L+I)^{-1} + I - (L+I)^{-1}\Big) \\ &= \det\Big(I - I_A(L+I)^{-1}\Big) \\ &= \det(I_{\bar{A}} + I_A K).\end{aligned}$$

Notar que la multiplicación $I_A K$ hace 0 todas las filas no correspondientes con A, de esta forma, podemos considerar la partición $\mathcal{Y} = \bar{A} \cup A$ y obtener

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

lo que concluye el resultado.

Teorema B.3 *Los valores propios no nulos de C y L son idénticos y los vectores propios correspondientes se relacionan mediante la matriz B. Esto es,*

$$C = \sum_{n=1}^{D} \lambda_n \hat{v}_n \hat{v}_n^\top,$$

es una descomposición en valores y vectores propios de C si y solamente si

$$L = \sum_{i=1}^{D} \lambda_n \left(\frac{1}{\sqrt{\lambda_n}} B^\top \hat{v}_n\right)\left(\frac{1}{\sqrt{\lambda_n}} B^\top \hat{v}_n\right)^\top,$$

es una descomposición en valores y vectores propios de L

Demostración Teorema B.3: ($\Rightarrow$) Sea $\{(\lambda_n, \hat{v}_n)\}_{n=1}^D$ una descomposición en valores y vectores propios de C. Entonces

$$\begin{aligned}\sum_{i=1}^{D} \lambda_n \left(\frac{1}{\sqrt{\lambda_n}} B^\top \hat{v}_n\right)\left(\frac{1}{\sqrt{\lambda_n}} B^\top \hat{v}_n\right)^\top &= B^\top \left(\sum_{n=1}^{D} \hat{v}_n \hat{v}_n^\top\right) B \\ &= B^\top B = L,\end{aligned}$$

donde se puede asumir que $\hat{v}_n$ son ortonormales, para cada n se tiene que

$$\begin{aligned}\left\|\frac{1}{\sqrt{\lambda_n}}B^\top\hat{v}_n\right\|^2 &= \frac{1}{\lambda_n}(B^\top\hat{v}_n)^\top(B^\top\hat{v}_n)\\ &= \frac{1}{\lambda_n}\hat{v}_n^\top C\hat{v}_n\\ &= \frac{1}{\lambda_n}\lambda_n||\hat{v}_n||\\ &= 1,\end{aligned}$$

lo anterior, recordando que $C\hat{v}_n = \lambda_n\hat{v}_n$ pues $\hat{v}_n$ es un valor propio de C. Finalmente para cada $1 \leq a, b \leq D$ distintos, se tiene que

$$\begin{aligned}\left(\frac{1}{\sqrt{\lambda_a}}B^\top\hat{v}_a\right)^\top\left(\frac{1}{\sqrt{\lambda_b}}B^\top\hat{v}_b\right) &= \frac{1}{\sqrt{\lambda_a\lambda_b}}\hat{v}_a^\top C\hat{v}_b\\ &= \frac{\sqrt{\lambda_b}}{\sqrt{\lambda_a}}\hat{v}_a^\top\hat{v}_b\\ &= 0,\end{aligned}$$

Se concluye entonces que $\left\{\left(\lambda_n, \frac{1}{\sqrt{\lambda_n}}B^\top\hat{v}_n\right)\right\}_{n=1}^D$ es una descomposición en valores y vectores propios de L. La dirección contraria ($\Leftarrow$) es análoga observando que $L = B^\top B$ y que L tiene rango a lo más D y por tanto tiene a lo más D valores propios no nulos.

Teorema B.4 *Sea $\hat{G}$ el estimador del gradiente objetivo definido como*

$$\hat{G}(\theta) = \frac{1}{|B|}\sum_{i\in B}\boldsymbol{\nabla} l(f(x_i,\theta), y_i),$$

con B sampleado a través de un proceso repulsivo $\mathcal{P}$. Entonces

$$\mathbb{V}ar(\hat{G}) = \frac{1}{k^2}\int_{\nu\times\nu}\lambda(x)\lambda(y)g(x,\theta)^\top g(y,\theta)\left[\frac{\rho(x,y)}{\lambda(x)\lambda(y)} - 1\right]dxdy + \frac{1}{k^2}\int_\nu ||g(x,\theta)||^2\lambda(x)dx,$$

donde $g(x,\theta) = \boldsymbol{\nabla} l(f(x,\theta))$, $\lambda(x)$ la cantidad de puntos esperada alrededor de x , $\rho(x,y)$ la cantidad de puntos esperada alrededor de x e y , $\nu\times\nu$ la grilla considerada y $k = |B|$ el tamaño del mini-batch.

Demostración Teorema B.4: Para esta demostración, es necesario utilizar el teorema de *Campbell* que permite calcular la esperanza de sumas de funciones medibles $f : \mathbb{R}^d \to \mathbb{R}$ mediante sus densidades producto bajo un proceso puntual $\mathcal{P}$.

$$\mathbb{E}_{\mathbb{P}}\left[\sum_{x_i\in\mathcal{P}} f(x_i)\right] = \int_{\mathbb{R}^d} f(x)\lambda(x)dx,$$

por otro lado, si $f : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

$$\mathbb{E}_{\mathbb{P}}\left[\sum_{i\neq j} f(x_i, x_j)\right] = \int_{\mathbb{R}^d\times\mathbb{R}^d} f(x,y)\rho(x,y)dxdy,$$

con lo anterior se tiene que

$$\mathbb{E}_{\mathcal{P}}\left[\hat{G}(\theta)\right] = \mathbb{E}_{\mathcal{P}}\left[\frac{1}{k}\sum_i g(x_i,\theta)\right] = \int_\nu \frac{1}{k} g(x,\theta)\lambda(x)dx.$$

La varianza de una variable aleatoria multi-dimensional puede ser escrita como $\mathbb{V}ar_{\mathcal{P}}(\hat{G}) = Tr(Cov_{\mathcal{P}}(\hat{G})) = \sum_m \mathbb{V}ar_{\mathcal{P}}(\hat{G}_m) = \sum_m \left[\mathbb{E}_{\mathcal{P}}(\hat{G}_m^2) - (\mathbb{E}_{\mathcal{P}}(\hat{G}_m))^2\right]$. Por otro lado,

$$\begin{aligned}
\mathbb{E}_{\mathcal{P}}[\hat{G}_m^2] &= \mathbb{E}_{\mathcal{P}}\left[\frac{1}{k^2}\sum_{ij} g_m(x_i,\theta)g_m(x_j,\theta)\right] \\
&= \mathbb{E}_{\mathcal{P}}\left[\frac{1}{k^2}\sum_{i\neq j} g_m(x_i,\theta)g_m(x_j,\theta)\right] + \mathbb{E}_{\mathcal{P}}\left[\frac{1}{k^2}\sum_i g_m^2(x_i,\theta)\right] \\
&= \frac{1}{k^2}\int_{\nu\times\nu} g_m(x,\theta)g_m(y,\theta)\rho(x,y)dxdy + \frac{1}{k^2}\int_\nu g_m^2(x,\theta)\lambda(x)dx,
\end{aligned}$$

además,

$$\begin{aligned}
(\mathbb{E}_{\mathcal{P}}[\hat{G}_m])^2 &= \left(\frac{1}{k}\int_\nu g_m(x,\theta)\lambda(x)dx\right)^2 \\
&= \frac{1}{k^2}\int_{\nu\times\nu} g_m(x,\theta)g_m(y,\theta)\lambda(x)\lambda(y)dxdy.
\end{aligned}$$

Juntando todo y sumando sobre m se obtiene el resultado deseado

$$\mathbb{V}ar(\hat{G}) = \frac{1}{k^2}\int_{\nu\times\nu} \lambda(x)\lambda(y)g(x,\theta)^\top g(y,\theta)\left[\frac{\rho(x,y)}{\lambda(x)\lambda(y)} - 1\right]dxdy + \frac{1}{k^2}\int_\nu ||g(x,\theta)||^2\lambda(x)dx.$$

Corolario B.1 *Una estrategia de sampleo de mini-batch mediante un proceso repulsivo disminuye la varianza del estimador del gradiente objetivo $\hat{G}(\theta)$ con respecto a una estrategia de sampleo uniforme.*

Para intuir el corolario, es necesario tener en mente que el término

$$\frac{1}{k^2}\int_{\nu\times\nu} \lambda(x)\lambda(y)g(x,\theta)^\top g(y,\theta)\left[\frac{\rho(x,y)}{\lambda(x)\lambda(y)} - 1\right]dxdy,$$

es nulo cuando el sampleo es uniforme pues $\rho(x,y) = \lambda(x)\lambda(y)$ y es negativo cuando el sampleo es mediante algún proceso repulsivo pues:

- Si x, y se encuentran lejos, $\rho(x,y) \approx \lambda(x)\lambda(y)$.
- Si x, y se encuentran cerca $\rho(x,y) < \lambda(x)\lambda(y)$ (proceso repulsivo) y asumiendo que la función de *loss* es suave en sus argumentos, entonces los gradientes se encuentran

alineados tal que $g(x,\theta)^\top g(y,\theta) > 0$.

Dicho esto, la expresión

$$\frac{1}{k^2}\int_{\nu\times\nu} \lambda(x)\lambda(y)g(x,\theta)^\top g(y,\theta)\left[\frac{\rho(x,y)}{\lambda(x)\lambda(y)} - 1\right] dxdy,$$

es negativa y por tanto disminuye la varianza del estimador $\hat{G}(\theta)$ con respecto a una estrategia de sampleo uniforme.

Anexo C. Parámetros de Arquitecturas

Tabla C.1: Parámetros Arquitectura Baseline Binario por Experimento.

Experiment	Train	Batch size	Optimizer	Epochs	Act Func
4.5.2	4.1	64	Adam	10	Tanh
4.5.3	4.1	64	Adam	50	Tanh

Tabla C.2: Parámetros Arquitectura Baseline Multiclase por Experimento.

Experiment	Train	Batch size	Optimizer	Epochs	Act Func
4.5.2	4.2	64	Adam	10	Relu
4.5.3	4.2	64	Adam	15	Relu

Tabla C.3: Parámetros Arquitectura Autoencoder por Experimento.

Experiment	Train	Batch size	Optimizer	Epochs	Latent Vector	Act Func
4.5.1	4.2	128	Adam	20	16	Sigmoid
4.5.2	4.2	128	Adam	20	32	Sigmoid
4.5.3	4.2	128	Adam	20	32	Sigmoid

Tabla C.4: Parámetros Oneshot por Experimento.

Experiment	Train	Batch size	Optimizer	Epochs	Latent Vector	Act Func
4.5.1	4.2	128	Rmsprop	10	16	Relu
4.5.2	4.2	128	Rmsprop	10	32	Relu
4.5.3	4.2	128	Rmsprop	10	32	Relu

Tabla C.5: Parámetros DPP NET por experimento.

Experiment	Train	Batch size	Optimizer	Epochs	Latent Vector	α
4.5.4	4.2	64	Adam	15	32	1/3

Anexo D. Especificaciones de Hardware

Todos los experimentos fueron realizados mediante la utilización de Google Colab y Python 3. Las especificaciones de *Hardware* son (podrían cambiar según la sesión de *Google Colab*)

- CPU Model: Intel(R) Xeon(R).
- CPU Freq: 2.30 HGz.
- CPU Cores: 4.
- RAM: 12.68 GB.
- Disk: 107.72 GB.

El entrenamiento de las redes neuronales fue realizado mediante la utilización de Tensorflow en su versión 2.9.2.