INSTITUTO TECNOLÓGICO DE LEÓN
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

"ESTRATEGIAS PARA ACELERAR EL ENTRENAMIENTO DE REDES NEURONALES PROFUNDAS"

TESIS

Que presenta:

ING. JORGE IVAN AVALOS LOPEZ

Para obtener el grado de

MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

Con la dirección de

DR. ALFONSO ROJAS DOMÍNGUEZ

LEÓN, GUANAJUATO AGOSTO 2021

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de León

León, Guanajuato, 12/junio/2021

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN
OFICIO No. DEPI-112-2021

ING. JORGE IVÁN ÁVALOS LÓPEZ
PRESENTE

De acuerdo al fallo emitido por la Comisión Revisora, integrada por los: **Dr. Alfonso Rojas Domínguez, Dr. Juan Martín Carpio Valadez, Dr. Manuel Ornelas Rodríguez, Dr. Sergio Ivvan Valdez Peña,** , considerando que llena todos los requisitos establecidos en los Lineamientos Generales para la Operación del Posgrado del Tecnológico Nacional de México, **se autoriza la impresión** del trabajo de tesis titulado: **"Estrategias para acelerar el entrenamiento de Redes Neuronales Profundas"**.. Lo que hacemos de su conocimiento para los efectos y fines correspondientes.

ATENTAMENTE

Excelencia en Educación Tecnológica®
Ciencia Tecnología y Libertad

SEP SECRETARÍA DE EDUCACIÓN PÚBLICA | TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE LEÓN
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

DR. DAVID ASAEL GUTIÉRREZ HERNÁNDEZ
JEFE DE LA DEPI

C.c.p. Expediente

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLOGICO NACIONAL DE MÉXICO

León, Guanajuato., a 9 de Julio del 2021

C. ING. LUIS ROBERTO GALLEGOS MUÑOZ
JEFE DE SERVICIOS ESCOLARES
P R E S E N T E

Por este medio hacemos de su conocimiento que la tesis titulada **"ESTRATEGIAS PARA ACELERAR EL ENTRENAMIENTO DE REDES NEURONALES PROFUNDAS"**, ha sido leída y aprobada por los miembros del Comité Tutorial para su evaluación por el jurado del acto de examen de grado al alumno **C. Jorge Iván Ávalos López,** con número de control **M19241000** como parte de los requisitos para obtener el grado de Maestro en Ciencias de la Computación (MCCOM-2011-05).

Sin otro particular por el momento, quedamos de Usted.

A T E N T A M E N T E
COMITÉ TUTORIAL

Dr. Alfonso Rojas Domínguez
DIRECTOR

Dr. Juan Martín Carpio Valadez
REVISOR

Dr. Manuel Ornelas Rodríguez
REVISOR

Dr. Sergio Ivvan Valdez Peña
REVISOR

TECNOLÓGICO NACIONAL DE MÉXICO

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

DECLARACION DE AUTENTICIDAD Y DE NO PLAGIO

Yo, Jorge Iván Ávalos López identificado con No. control M19241000, alumno (a) del programa de la **Maestría en Ciencias de la Computación**, autor (a) de la Tesis titulada:

" ESTRATEGIAS PARA ACELERAR EL ENTRENAMIENTO DE REDES NEURONALES PROFUNDAS"

DECLARO QUE:

1.- El presente trabajo de investigación, tema de la tesis presentada para la obtención del título de **MAESTRO (A) EN CIENCIAS DE LA COMPUTACIÓN** es original, siendo resultado de mi trabajo personal, el cual no he copiado de otro trabajo de investigación, ni utilizado ideas, fórmulas, ni citas completas "stricto sensu", así como ilustraciones, fotografías u otros materiales audiovisuales, obtenidas de cualquier tesis, obra, memoria, etc. en su versión digital o impresa.

2.- Declaro que el trabajo de investigación que pongo a consideración para evaluación no ha sido presentado anteriormente para obtener algún grado académico o título, ni ha sido publicado en sitio alguno.

3.- Declaro que las pruebas o experimentos derivados de esta investigación fueron realizados bajo el consentimiento de los involucrados y con fines estrictamente académicos conforme a criterios éticos de confidencialidad.

Soy consciente de que el hecho de no respetar los derechos de autor y hacer plagio, es objeto de sanciones universitarias y/o legales por lo que asumo cualquier responsabilidad que pudiera derivarse de irregularidades den la tesis, así como de los derechos sobre la obra presentada.

Asimismo, me hago responsable ante el Tecnológico Nacional de México/Instituto Tecnológico de León o terceros, de cualquier irregularidad o daño que pudiera [illegible] por el incumplimiento de lo declarado.

De identificarse falsificación, plagio, fraude, o que el trabajo de investigación haya sido publicado anteriormente; asumo las consecuencias y sanciones que de mi acción se deriven, responsabilizándome por todas las cargas pecuniarias o legales que se deriven de ello sometiéndome a las normas establecidas en los Lineamientos y Disposiciones de la Operación de Estudios de Posgrado en el Tecnológico Nacional de México.

León, Guanajuato a 16 del mes de Agosto de 2021

Nombre y firma del autor ()

Jorge Iván Avalos Lopez

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLOGICO NACIONAL DE MEXICO

ACUERDO PARA USO DE OBRA (TESIS DE GRADO)

A QUIEN CORRESPONDA

PRESENTE

Por medio del presente escrito, Jorge Iván Ávalos López (en lo sucesivo el AUTOR) hace constar que es titular intelectual de la obra denominada: " ESTRATEGIAS PARA ACELERAR EL ENTRENAMIENTO DE REDES NEURONALES PROFUNDAS ", (en lo sucesivo la OBRA) en virtud de lo cual autoriza al Tecnológico Nacional de Méxi Tecnológico de León (en lo sucesivo TECNM/IT León) para que efectúe resguardo físico y/o electrónico mediante copia digital o impresa para asegurar su disponibilidad, divulgación, comunicación pública, distribución, transmisión, reproducción, así como digitalización de la misma con fines académicos y sin fines de lucro como parte del Repositorio Institucional del TECNM/ITLeón.

De igual manera, es deseo del AUTOR establecer que esta autorización es voluntaria y gratuita, y que de acuerdo a lo señalado en la Ley Federal del Derecho de Autor y la Ley de Propiedad Industrial el TECNM/IT León cuenta con mi autorización para la utilización de la información antes señalada, estableciendo que se utilizará única y exclusivamente para los fines antes señalados. El AUTOR autoriza al TECNM /IT León a utilizar la obra en los términos y condiciones aquí expresados, sin que ello implique se le conceda licencia o autorización alguna o algún tipo de derecho distinto al mencionada respecto a la "propiedad intelectual" de la misma OBRA; incluyendo todo tipo de derechos patrimoniales sobre obras y creaciones protegidas por derechos de autor y demás formas de propiedad intelectual reconocida o que lleguen a reconocer las leyes correspondientes. Al reutilizar, reproducir, transmitir y/o distribuir la OBRA se deberá reconocer y dar créditos de autoría de la obra intelectual en los términos especificados por el propio autor, y el no hacerlo implica el término de uso de esta licencia para los fines estipulados. Nada de esta licencia mer restringe los derechos patrimoniales y morales del AUTOR.

De la misma manera, se hace manifiesto que el contenido académico, literario, la edición y en general de cualquier parte de la OBRA son responsabilidad de AUTOR, por lo que se deslinda al (TECNM/ITLeón) por cualquier violación a los derechos de autor y/o propiedad intelectual, así como cualquier responsabilidad relacionada con la misma frente a terceros. Finalmente, el AUTOR manifiesta que estará depositando la versión final de su documento de Tesis, OBRA, y cuenta con los derechos morales y patrimoniales correspondientes para otorgar la presente autorización de uso.

En la ciudad de León, del estado de Guanajuato a los 16 días del mes de Agosto de 2021.

Atentamente,

Nombre y firma autógrada de EL AUTOR

Jorge Iván Ávalos López

Dedicatoria

Dedicado a mis queridos padres, por su constante apoyo y cariño, siempre voy a estar agradecido.

Agradecimientos

A mis padres Jorge e Isabel, por su constante apoyo y cariño, y por siempre estar ahí conmigo.

A mi hermana Mayra, por quererme y apoyarme toda la vida, y creer en mí.

A mis amigos incondicionales Moisés, Alberto, Gerardo y Manuel, que han estado ahí cuando los he necesitado.

Al Dr. Alfonso Rojas por el tiempo dedicado para la realización de esta tesis.

Al Dr. Manuel Ornelas por su excelsa calidad humana, por la atención a sus estudiantes, y por ser un buen profesor.

Al Comité revisor por darme sus puntos de vista y el tiempo que dedicaron en la revisión de esta tesis.

A la División de Estudios de Posgrado del Instituto Tecnológico de León por la oportunidad de seguir con mis estudios y al Consejo Nacional de Ciencia y Tecnología por la beca otorgada a lo largo del periodo de estudios de maestría.

Resumen

El presente trabajo de tesis muestra el desarrollo de una propuesta para acelerar el entrenamiento de una red neuronal profunda, específicamente el acelerar el entrenamiento de una red neuronal convolucional para tareas de clasificación de imágenes. Además, el presente trabajo es un producto del Cuerpo Académico Metrología y Sistemas inteligentes (CAMSI), y funge como contribución al área de inteligencia artificial computacional y aprendizaje automático de la División de Estudios de Posgrado e Investigación del instituto tecnológico de León.

El documento se presenta en 5 capítulos. En el capítulo 1 se da una introducción de la tesis. El capítulo 2 se presenta una revisión del marco teórico relacionado con dos modelos de aprendizaje profundo : las redes neuronales profundas y las redes neuronales convolucionales. Seguidamente, en el capítulo 3 se presenta una revisión del estado del arte, relacionado con las propuestas para acelerar el entrenamiento de redes neuronales profundas; específicamente, este capítulo se centra en estrategias que involucran el muestreo selectivo de datos para mejorar la convergencia del descenso del gradiente estocástico. En el cuarto capítulo se expone la estrategia propuesta llamada: muestreo adaptativo para el aprendizaje profundo (AS en sus siglas en ingles). Esta propuesta es una estrategia que extrapola a la técnica de muestreo adaptativo mediante información colateral (Gopal, 2016) a modelos de aprendizaje profundo. Específicamente, la contribución de este trabajo es la formulación de una distribución de probabilidad sobre las instancias de entrenamiento que minimice la varianza de las normas de los gradientes con respecto a la función de pérdida de la red. Así mismo, la formulación del problema se presenta de forma consistente y se da un formalismo matemático adecuado. Para comprobar y observar el comportamiento de la propuesta, se experimentó con el entrenamiento de una red neuronal convolucional sobre la base de datos MNIST. AS se comparó contra dos esquemas de muestreo de datos: muestreo uniforme (el entrenamiento estándar) y selección de lotes en línea (Loshchilov & Hutter, 2016). AS logró un mejoramiento en la convergencia de la función de costo (que esto se traduce en una aceleración del entrenamiento) entre 2 y 4 órdenes de magnitud en comparativa con default random con tamaños de lote 64, 256 y 1024 usando el optimizador Adadelta; usando el optimizador Adam, AS mostró una mejora de convergencia de 1 y 2 órdenes de magnitud con los tamaños de lote 256 y 1024. En la comparativa de AS y OBS, AS mostró una mejora de convergencia de 0.78, 0.2 y 0.92 para tamaños de lote 64, 256 y 1024

usando Adadelta; usando Adam, AS logra un rendimiento similar para tamaños de lote 256 y 1024.

Abstract

The present thesis work shows the development of a proposal to accelerate the training of a deep neural network, specifically to accelerate the training of a convolutional neural network for image classification tasks. In addition, this work is a product of the Academic Body of metrology and Intelligent Systems (CAMSI) and serves as a contribution to the area of computational artificial intelligence and machine learning of the Division of Postgraduate Studies and Research of the Technological Institute of León.

The document is presented in 5 chapters. Chapter 1 introduces the thesis. Chapter 2 presents a review of the theoretical framework related to two deep learning models: deep neural networks and convolutional neural networks. Next, chapter 3 presents a review of the state of the art, related to the proposals to accelerate the training of deep neural networks; Specifically, this chapter focuses on strategies that involve selective data sampling to improve the convergence of the stochastic gradient descent. In the fourth chapter the proposed strategy called: adaptive sampling for deep learning (AS) is exposed. This proposal is a strategy that extrapolates the adaptive sampling technique through collateral information (Gopal, 2016) to deep learning models. Specifically, the contribution of this work is the formulation of a probability distribution on the training instances that minimizes the variance of the norms of the gradients with respect to the loss function of the network. Likewise, the formulation of the problem is presented consistently, and an adequate mathematical formalism is given. To verify and observe the behavior of the proposal, we experimented with the training of a convolutional neural network on the MNIST database. AS was compared against two data sampling schemes: uniform sampling (the standard training) and online batch selection (Loshchilov & Hutter, 2016). AS achieved an improvement in the convergence of the cost function (which translates into an acceleration of the training) between 2 and 4 orders of magnitude in comparison with default random with batch sizes 64, 256 and 1024 using the Adadelta optimizer; Using the Adam optimizer, AS showed a convergence improvement of 1 and 2 orders of magnitude with batch sizes 256 and 1024. In the comparison of AS and OBS, AS showed a convergence improvement of 0.78, 0.2 and 0.92 for batch sizes. batch 64, 256 and 1024 using Adadelta; using Adam, AS achieves similar performance for batch sizes 256 and 1024.

Índice General

Índice de tablas

Índice de figuras

Lista de acrónimos

Acrónimo	Descripción
DL	Deep Learning Aprendizaje Profundo
DNN	Deep Neural Network Red Neuronal Profunda
CNN	Convolutional Neural Network Red Neuronal Convolucional
IS	Importance Sampling Muestreo Enfatizado
ReLU	Rectified Linear Units Unidades Lineales Rectificadas
SGD	Stochastic Gradient Descent Descenso del Gradiente Estocástico
OBS	Online Batch Selection Selección de Lote En Línea
AS	Adaptive Sampling Muestreo Adaptativo
GPU	Graphics Processing Unit Unidad Gráfica de Procesamiento
FC	Completamente Conectada Fully connected

Capítulo 1

1 Introducción

A lo largo de los últimos años, las técnicas de aprendizaje profundo han impactado en áreas de ciencias de la computación [1] como visión por computadora, reconocimiento de patrones, procesamiento del lenguaje natural, etc.. Este impacto se debe sobre todo a la mejora en las capacidades de hardware de las unidades gráficas de procesamiento (GPUs), al cambio de paradigma de abordar tareas de reconocimiento de patrones desde el enfoque tradicional (usando descriptores) a un enfoque de aprendizaje automático mediante el uso de modelos basados en redes neuronales profundas y a la disponibilidad de grandes conjuntos de datos etiquetados que han permitido el entrenamiento de estos modelos [2].

Las redes neuronales convolucionales han sido una de las ideas más exitosas en el aprendizaje profundo [1]. Desde el 2012 mediante la creación de la red AlexNet [25], las redes neuronales convolucionales han mostrado un desempeño superior para tareas de visión por computadora [1] como clasificación de imágenes, segmentación, detección de objetos, etc.

El cuello de botella en la implementación de los modelos de aprendizaje profundo es su entrenamiento; debido a la gran cantidad de datos a procesar y los millones de parámetros entrenables, el proceso de entrenamiento suele ser muy costoso desde el punto de vista computacional [2]. El método estándar para entrenar a muchas redes neuronales profundas, en particular para tareas de clasificación, es el descenso de gradiente estocástico (SGD) y sus variaciones. Al modelo se le presenta un conjunto de ejemplos, se calcula una función de error y el gradiente de esta función de error se propaga hacia atrás a través de la red para determinar los ajustes necesarios en sus parámetros. En la práctica, en lugar de ajustar a la red una instancia de datos a la vez, el conjunto de datos de entrenamiento se divide en subconjuntos de datos llamados lotes de tamaño fijo y la red se entrena con un lote a la vez. La forma convencional de formar los lotes es a través del muestreo uniforme de las instancias de datos, pero recientemente se han realizado esfuerzos para desarrollar estrategias para acelerar el entrenamiento de redes neuronales profundas a través de esquemas de muestreo más eficientes [32,37,39].

En este trabajo de tesis se presenta una extensión del trabajo de Gopal [36], donde se extrapola un esquema de muestreo de lotes para acelerar un perceptrón multicapa a un esquema de muestreo de lotes para acelerar el entrenamiento de una red neuronal convolucional sobre la base de datos MNIST [44]. Dicha estrategia se compara contra dos enfoques : la forma convencional de entrenamiento (mediante el muestreo uniforme de datos) y la selección de lotes en línea [37].

1.1 Definición del problema

Durante la última década, el aumento de datos disponibles y el poder computacional (desarrollo de tarjetas gráficas con una mayor capacidad de cómputo), así como importantes desarrollos teóricos, impulsaron el advenimiento de técnicas de aprendizaje profundo (DL por sus siglas en inglés) y su mayor dominio en varias aplicaciones de reconocimiento de patrones como visión por computadora, clasificación de patrones, procesamiento del lenguaje natural, predicción, etc. [1].

El cuello de botella de los modelos DL es su entrenamiento; debido a la gran cantidad de datos a procesar y los millones de parámetros a optimizar, el proceso de entrenamiento suele ser muy costoso desde el punto de vista computacional [2].

Debido a esto, se precisa desarrollar estrategias para acelerar el entrenamiento de los modelos DL, con el objetivo de disminuir el tiempo de cómputo efectivo sin afectar significativamente el rendimiento de los modelos.

1.2 Hipótesis

Al Identificarse las variables que determinan el proceso de entrenamiento de los modelos de DL, es posible diseñar y aplicar estrategias para acelerar dicho entrenamiento sin afectar el rendimiento del modelo, disminuyendo así el costo computacional, tanto en número de operaciones como en tiempo requerido.

1.3 Justificación

Los modelos de DL son algoritmos que recientemente han dado un mejor rendimiento en campos como visión por computadora y reconocimiento automático de patrones sobre los algoritmos tradicionales como máquinas de soporte vectorial, redes neuronales, etc. [1].

El entrenamiento de los modelos de DL es un proceso esencial para que un sistema basado en este tipo de técnicas obtenga el mejor rendimiento posible. Sin embargo, el entrenamiento de dichos modelos es muy costoso. Crear estrategias para acelerar este proceso permitiría un uso más eficiente de dichos sistemas. [32,37,39,40,41,42] han desarrollado estrategias de distinta naturaleza para acelerar el entrenamiento de los modelos de DL, enfocándose principalmente en un muestreo de datos más inteligente, donde a los modelos se les subministre datos que sean de mayor utilidad para acelerar su entrenamiento.

Aplicar técnicas para acelerar el entrenamiento de los modelos de aprendizaje profundo permitirá un menor uso de cómputo y habría lo posibilidad de crear nuevas aplicaciones y sistemas

Este trabajo de tesis es un producto del Cuerpo Académico Metrología y Sistemas Inteligentes (CAMSI), y aporta conocimiento teórico-práctico a la División de Posgrado del Instituto Tecnológico de León, específicamente en el área de inteligencia computacional y aprendizaje automático, y servirá de referencia para futuros trabajos de tesis para nuevos estudiantes de la división.

1.4 Objetivo General

Proponer, aplicar, comparar y analizar una estrategia que permita acelerar el entrenamiento de una red neuronal profunda (Específicamente una CNN para clasificación de imágenes).

1.5 Objetivos Específicos

1. Analizar cómo se crean y se entrenan las CNNs.
2. Comparar y aplicar diferentes estrategias para acelerar el entrenamiento de las CNNs.
3. Proponer y comparar una estrategia para acelerar el entrenamiento de las CNNs en tareas de clasificación de imágenes.

Con respecto a los objetivos específicos, el objetivo 2 no se completó en su totalidad, solo se comparó y aplicó una estrategia del estado del arte (OBS [37]). Los motivos principales por los cuales no se cumplió en su totalidad este objetivo fueron los siguientes:

1.- La falta de requerimientos tecnológicos: el proyecto de tesis inicialmente se tendría que haber llevado a cabo en las instalaciones del tecnológico de León haciendo uso de su estructura digital y de los equipos de cómputo disponibles que hay en la maestría de ciencias de la computación. Sin embargo, el 24 de marzo del 2020 el gobierno federal decretó el inicio de la fase 2 de la pandemia COVID-19 en el país. Este acontecimiento produjo que las actividades escolares se realizaran en casa. Por lo tanto, para seguir con el proyecto, se optó por hacer uso del cómputo en la nube, específicamente de las herramientas de Google Colab. No obstante, Google Colab limita el tiempo en que cada usuario puede utilizar su infraestructura de manera ininterrumpida[1]. De esta forma, fue complicado realizar experimentos sobre un equipo de cómputo que no controlas y te restringen su uso. Debido a esto, no tuve el tiempo suficiente

2.- Las distintas estrategias del estado del arte para acelerar el entrenamiento de una DNN son programadas en distintas bibliotecas de Python: OBS [37] es una estrategia que esta implementada en una biblioteca que se llama Lasagne, la estrategia de Katharopoulos & Fleuret [32] esta implementada en otra biblioteca que se llamada TensorFlow y la biblioteca que se usó en esta tesis para implementar las estrategias fue PyTorch (se utilizó PyTorch porque es la biblioteca que yo domino, las demás bibliotecas de aprendizaje profundo soy deficiente ante ellas). Dado que no hay una librería estándar para implementar computacionalmente los modelos de aprendizaje profundo, fue una labor complicada y llevó un tiempo considerable (de 3 a 4 meses, en los meses de febrero a mayo) tan solo mapear la implementación de OBS a PyTorch. Mapear otra implementación, como por ejemplo la de Katharopoulos & Fleuret [32] llevaría un tiempo similar o hasta más por la complejidad del algoritmo. En el caso particular del algoritmo de Katharopoulos & Fleuret, ellos proveen su algoritmo en una librería y la hacen pública para experimentar. Sin embargo, Google Colab no reconoce dicha librería y por lo tanto, si se quisiera implementar, se tendría que hacer lo mismo que para OBS.

[1] https://research.google.com/colaboratory/faq.html

1.6 Alcances y limitaciones

Existe una rica variedad de modelos de aprendizaje profundo [8] como las maquinas Boltzmann, las redes neuronales recurrentes, redes neuronales convolucionales, redes neuronales profundas hacia adelante, etc. En este trabajo se estudian únicamente las redes neuronales convolucionales.

Al comienzo del proyecto uno de los objetivos era utilizar al menos dos bases de datos. Debido al tiempo requerido y a los recursos computacionales que supone entrenar a las CNNs se trabajó únicamente con la base de datos estándar MNIST [44], se trabajó con este conjunto de datos debido a que no presenta muchas demandas computacionales para los modelos y es una base de datos estándar para experimentación.

1.7 Organización de Tesis

El resto de este trabajo de tesis está estructurado de la siguiente forma:

Capítulo 2 En el capítulo 2 se presentan los conocimientos teóricos necesarios para el entendimiento de las DNNs y las CNNs. Este capítulo se centra en zanjar ideas básicas de diferenciación en cálculo de múltiples variables y nociones básicas de estadística. Esto con el objetivo de que se logre un mejor entendimiento de las funciones de aproximación que construyen dichos modelos y su proceso de entrenamiento. También, se tocan temas de aprendizaje máquina que es preciso estudiar para entender el cambio de paradigma del aprendizaje profundo.

Capítulo 3 En el capítulo 3 se estudian propuestas del estado del arte sobre estrategias para acelerar el entrenamiento de las DNNs. Se presentan principalmente dos enfoques, el muestreo de datos basado en la información obtenida mediante el gradiente y el muestreo basado en la información obtenida mediante la función de pérdida.

Capítulo 4 En el capítulo 4 se presenta una descripción detallada de la estrategia propuesta en este trabajo llamada muestreo adaptativo para el aprendizaje profundo. Así mismo, se presentan resultados experimentales de este método sobre la base de datos MNIST [44] y se compara con el algoritmo de entrenamiento estándar, que muestrea instancias de datos bajo una

distribución uniforme, y la propuesta de Loshchilov & Hutter [37] llamada selección de lotes en línea (OBS).

Capítulo 5 Finalmente, en el capítulo 5 se presentan las conclusiones de la propuesta con base en los resultados del capítulo 4. Además, se exhiben los alcances de esta estrategia y se expone trabajo futuro.

Capítulo 2

2 Marco Teórico

Este capítulo está organizado en dos secciones: la sección 2.1 involucra una pequeña descripción de la matemática, estadística y principios de aprendizaje máquina que se necesitan para el entendimiento de dos modelos de DL: las redes neuronales profundas (DNN por sus siglas en ingles) y las redes neuronales convolucionales (CNN por sus siglas en ingles). Posteriormente, en la sección 2.2 se presenta la descripción detallada de dichos modelos. También se describen dos métodos de regularización y cuatro variaciones del descenso del gradiente para lidiar con el sobreajuste y la aceleración de convergencia de la función de costo respectivamente.

2.1 Preliminares de Cálculo Diferencial en Varias Variables y Principios de Aprendizaje Maquina

El aprendizaje profundo (DL por sus siglas en inglés) es una serie de algoritmos que pertenecen a un subconjunto del aprendizaje automático [11]. DL involucra disciplinas como matemáticas y estadística para la descripción de sus modelos.

En la subsección 2.1.1 se da la definición matricial de la derivada de campos escalares y campos vectoriales, esto con el objetivo de invocar estas definiciones cuando se toque el tema de propagación hacia atrás en las secciones 2.2.1 y 2.2.3.

En subsecuentes secciones de este capítulo se utilizarán los términos función, transformación y aplicación como sinónimos. Para la sección 2.1, las letras mayúsculas representarán vectores o matrices y las minúsculas representarán sus componentes.
La demostración del teorema (2) se encuentra en [3].

2.1.1 Cálculo Diferencial En Campos Escalares Y Vectoriales

Funciones de R^n a R^m

Formalmente una función de R^n *a* R^m es una transformación de un espacio lineal *n*-dimensional *V*, a un espacio *m*-dimensional *W*:

$$T : V \rightarrow W.$$

Se dice entonces que el dominio de la transformación T pertenece al espacio de R^n y el recorrido pertenece al espacio de R^m.

En los modelos de DL se construyen aplicaciones donde $n > 1$ y $m \geq 1$. Cuando $m = 1$, a tales aplicaciones se les conoce como funciones reales de una variable vectorial o campo escalar. Cuando $m > 1$, a tales transformaciones se les llama funciones vectoriales de una variable, o campos vectoriales.

Gradiente de un campo escalar

Sean campos escalares de la forma $F : X \subseteq R^n \rightarrow R$**.** El gradiente de F se define como [3]:

$$\nabla F(X) = \left[\frac{\partial}{\partial x_1} F(X), \ldots, \frac{\partial}{\partial x_n} F(X)\right]^T. \qquad (1)$$

Entonces $\nabla F(X)$ es un campo vectorial para el cual existen todas las derivadas parciales $\frac{\partial}{\partial x_1} F(X), \ldots, \frac{\partial}{\partial x_n} F(X)$.

Diferenciales de campos vectoriales

Se considera ahora la derivada de campos vectoriales. Sea $F : X \subseteq R^n \rightarrow R^m$, y cada componente de R^m una función tipo $f : X \subseteq R^n \rightarrow R$. De modo que se tienen *m* funciones $F = (f_1(X), \ldots, f_m(X))$ definidas en *n* variables.

De acuerdo con el teorema de la derivada de campos vectoriales, tenemos que [3]:

$$\frac{\partial}{\partial X} F(X) = JX. \qquad (2)$$

Donde J es una matriz mxn y es llamada matriz Jacobiana para la cual existen todas sus derivadas parciales y *X* es un vector columna en R^n. La matriz Jacobiana se define como [3]:

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}. \qquad (3)$$

2.1.2 Probabilidad

Variables aleatorias

En teoría de probabilidad se le llama variable aleatoria a cualquier función que está definida en un dominio continuo o discreto y que asigna un valor al resultado de un experimento [11]. El ejemplo más sencillo de una variable aleatoria es el conjunto de posibles resultados que toma una moneda al ser lanzada una vez, codificando como 0 si la moneda cae en cruz, o 1 si la moneda cae en cara.

Distribuciones de probabilidad

Sea χ una variable aleatoria discreta. Entonces, se define a la distribución de probabilidad de χ como una función dada por $f(x) = P(\chi = \text{x})$ para cada x del conjunto discreto [4]. Por lo tanto, una distribución de probabilidad es una función que asigna un numero a todos los posibles valores de una variable aleatoria.

Con base en los postulados de probabilidad [4], se tiene el siguiente teorema (la demostración se encuentra en [4]):

Teorema 1: Cualquier función desempeña una distribución de probabilidad de una variable aleatoria discreta χ si y solo si $f(x)$ cumple los siguientes requerimientos:

a. $f(x) \geq 0$ para todo valor de x de su dominio,
b. $\sum_x f(x) = 1$ para todo valor de x de su dominio.

La distribución de probabilidad es una función que asigna un numero entre 0 y 1 a los resultados de un experimento, donde este número se puede interpretar como la frecuencia de ocurrencia de los resultados cuando el experimento se repiten infinidad de veces; por ejemplo, si tenemos que $P(\chi = \text{x}) = 0.8$, podemos esperar que, si repetimos el experimento infinitas veces, el 80 % de esas veces, la variable aleatoria χ tomará el valor de x.

Expectación y varianza

La expectación y varianza son quizá unas de las ideas más relevantes en probabilidad y estadística. Porque son medidas que representan la tendencia central y dispersión, respectivamente, de una variable aleatoria.

La media se define como [4]:

$$\mathbb{E}[\chi] = \sum_{x} x \cdot f(x). \tag{4}$$

La varianza se define como [4]:

$$\mathbb{E}[(\chi - \mathbb{E}[\chi])^2] = \sum_{x} (x - \mathbb{E}[\chi])^2 \cdot f(x). \tag{5}$$

Donde χ es una variable aleatoria discreta y $f(x)$ es el valor de su distribución de probabilidad en *x*.

Considerando la propiedad de linealidad de la expectación, de (6) se puede obtener el siguiente resultado [4]:

$$\mathbb{E}[(\chi - \mathbb{E}[\chi])^2] = \mathbb{E}[\chi^2] - (\mathbb{E}[\chi])^2. \tag{6}$$

Comúnmente a la varianza se le asigna el símbolo σ^2 o $V[\chi]$.

Se pueden construir variables aleatorias como combinaciones lineales de otras variables aleatorias. A continuación, se enuncia el siguiente teorema de combinaciones lineales de variables aleatorias [4].

Teorema 2: Sean $\chi_1, \chi_2, \ldots, \chi_n$ variables aleatorias independientes idénticamente distribuidas tal que

a) $Y = \sum_{i=1}^{n} X_i$.

Entonces,

b) $\mathbb{E}[Y] = \sum_{i=1}^{n} \mathbb{E}[X_i]$,

y,

c) $V[Y] = \sum_{i=1}^{n} V[X_i]$.

Si $X = \begin{bmatrix} \chi_1 \\ \vdots \\ \chi_n \end{bmatrix}$ y, usando (6), la ecuación (c) del Teorema 2 se puede escribir como:

$$V[Y] = \mathbb{E}[X^{\mathrm{T}}X] - \mathbb{E}[X]^{\mathrm{T}}\mathbb{E}[X]. \tag{7}$$

Muestreo Enfatizado (IS)

El muestreo enfatizado (IS por sus siglas en inglés) es una técnica Montecarlo [5] que estima propiedades de una distribución particular, donde se generan muestras a partir de una distribución diferente a la distribución de interés.

Sea χ una variable aleatoria y $P(x)$ el valor de su distribución de probabilidad en *x*. Utilizando la ecuación (5) generalizada (en [4], se da una demostración de la generalización de la esperanza matemática de las variables aleatorias tipo $f(x)$), se puede hacer la siguiente construcción:
Se tiene que

$$E_{x\sim P}[f(x)] = \sum_x f(x) \cdot P(x), \tag{8}$$

multiplicando por un 1 el valor esperado, y se obtiene

$$\sum_x f(x) \cdot P(x) = \sum_x f(x) \cdot P(x) \frac{Q(x)}{Q(x)} = \sum_x f(x) \cdot Q(x) \frac{P(x)}{Q(x)}. \tag{9}$$

Ahora, cambiando el muestreo de $x\sim P$ a $x\sim Q$, en lugar de obtener el valor de expectación de $f(x)$, se busca el valor de expectación de $\frac{P(x)}{Q(x)} f(x)$, se tiene que:

$$E_{x\sim Q}\left[f(x) \frac{P(x)}{Q(x)}\right] = \sum_x \xi(x) \cdot f(x). \tag{10}$$

Donde $\xi(x) = \frac{P(x)}{Q(x)}$ es llamado el peso de importancia y Q la distribución de importancia [5].

2.1.3 Principios de aprendizaje Máquina

¿Qué es un algoritmo de aprendizaje?

En primera instancia, un algoritmo de aprendizaje es un algoritmo computacional capaz de aprender de datos. De tal forma, los algoritmos de aprendizaje toman un subconjunto de datos (muestra de una población) y logran obtener estructuras o elementos que se repiten en los datos de manera predecible para tratar de representar a la población y después realizar una tarea.

Salim Lahmiri [6] provee de una definición de algoritmo de aprendizaje, citando:
"Un algoritmo de aprendizaje es un método utilizado para procesar datos para extraer patrones apropiados para su aplicación en una nueva situación. En particular, el objetivo es adaptar un sistema a una tarea de transformación de entrada-salida específica"

Siendo así, un algoritmo de aprendizaje realiza una tarea de transformación entrada-salida; las tareas que pueden realizar estos algoritmos son variadas [8], entre ellas están: clasificación,

regresión, traducción, transcripción, síntesis y muestreo, etc. Se dan solo las descripciones de las tareas de clasificación y regresión.

Clasificación: Dadas *m* categorías de algún conjunto de datos *n*-dimensionales, formando una partición en *m* subconjuntos; en la tarea de clasificación, el algoritmo de aprendizaje debe de producir un campo escalar o vectorial de la forma $F : X \subseteq R^n \rightarrow \{1, \dots, m\} \subset \mathrm{Z}^+ \vee F : X \subseteq R^n \rightarrow R^m, n, m \geq 1$. Entonces el algoritmo de aprendizaje asigna una entrada descrita por el vector X a una categoría del conjunto $\{1, \dots, m\}$.

Regresión: Dados un conjunto de pares de datos $D = \{(X_1, Y_1), (X_2, Y_1), \dots, (X_N, Y_N)\}$, $X_i \in R^n, Y \in R^m$; en la tarea de regresión, el algoritmo de aprendizaje debe producir un campo escalar o vectorial de la forma $F : X_i \subseteq R^n \rightarrow R^m, n, m \geq 1$. Tal que, la función de aproximación estime la relaciones entre los pares de puntos.

Aprendizaje supervisado

Los algoritmos de aprendizaje automático pueden ser categorizados en dos tipos: aprendizaje supervisado y aprendizaje no supervisado [17].

El aprendizaje supervisado, como su nombre lo dice, se origina desde el punto de vista donde un maestro o instructor (etiquetas o targets en inglés) muestra a su alumno o pupilo (máquina de aprendizaje) como aprender cierta tarea [17].

En un término abstracto, podemos definir al aprendizaje supervisado como algoritmos de aprendizaje que reciben una entrada y construyen un campo vectorial o escalar para producir una salida deseada; cada vector de entrada tiene asociado una etiqueta o también llamada categoría.

Sobreajuste y subajuste (Overfitting and underfitting)

Uno de los mayores desafíos en aprendizaje maquina es lidiar con el sobreajuste y el subajuste de los modelos [7]. Cuando el modelo de aprendizaje logra encontrar patrones de una muestra y logra describir con esa muestra a la población para realizar una tarea, se dice que el modelo ha generalizado los datos de entrenamiento.

El subajuste ocurre cuando el modelo de aprendizaje no logra capturar la tendencia subyacente de los datos de entrenamiento. En cambio, el sobreajuste sucede cuando el modelo logra capturar el ruido de los datos de entrenamiento y no es capaz de generalizar a datos nunca vistos por el algoritmo.

Capacidad de un algoritmo de aprendizaje

La capacidad de un algoritmo de aprendizaje es el control que se tiene para que un modelo este en sobreajuste o subajuste [8]. En consecuencia, la capacidad de un modelo es la habilidad de generar distintos tipos de funciones para realizar una tarea; los modelos con baja capacidad les costara trabajo producir una función que encaje en los datos, por otra parte, los modelos con alta capacidad producirán una función que encaje demasiado en los datos y memorice patrones como el ruido intrínseco de los mismos.

La capacidad es una idea indispensable para entender los modelos de aprendizaje profundo, y la razón del porque son modelos tan potentes en tareas como clasificación o regresión estriba en este concepto.

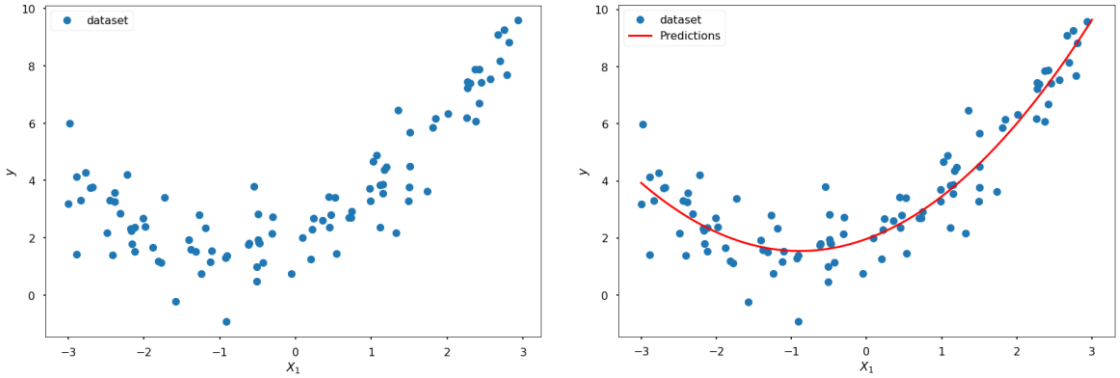
Yohsua Bengio et al. [8] describe que los algoritmos de aprendizaje tendrán un buen rendimiento cuando su capacidad sea apropiada para la complejidad de la tarea que realicen y la cantidad de datos que se provean.

A continuación, se realiza el siguiente ejercicio para poder profundizar en los conceptos de sobreajuste, subajuste y capacidad.
Se tiene los datos sintéticos de la Fig. 1-a, la tarea a realizar es un proceso de regresión, donde el objetivo es encontrar una función que permite estimar la relación de la variable x_1 y y. La estrategia a seguir es proponiendo una función polinomial con *(n+1)* parámetros a estimar. Concretamente, la función de aproximación es:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \cdots + \theta_n x_1^n. \qquad (11)$$

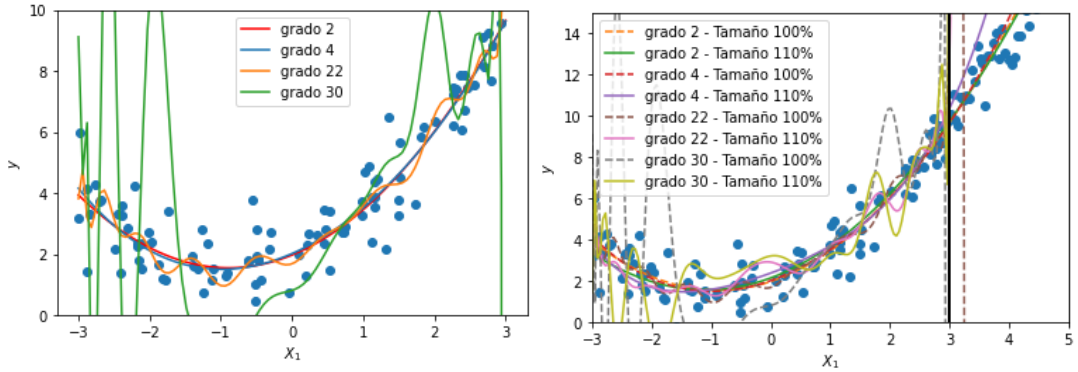
Como primera instancia, se puede considerar un polinomio con tres parámetros (grado dos), y la estimación de *y* con respecto a x_1 queda como en la Fig. 1-b. No obstante, se puede considerar

aumentar el número de parámetros del polinomio (Fig. 2-a); el resultado de aumentar los parámetros del polinomio ocasiona evidentemente que varias funciones de aproximación traten de estimar la relación entre la variable x_1 y y. De acuerdo con las definiciones descritas arriba, el polinomio de *(n+1)* parámetros (11) tendrá subajuste si no logra capturar la tendencia entre las variables x_1 y y, por otro lado, (11) tendrá sobreajuste si logra capturar el ruido de los datos y no es capaz de generalizar a datos nunca vistos por el algoritmo; adyacente a estos conceptos, la capacidad de (11) estriba en el control de la función para estar en sobreajuste o subajuste; el control de (11) son el número de parámetros que se pueden fijar y determinan el comportamiento de la función en datos de entrenamiento y prueba (Figuras 2-a, 2-b y 3).



a) **b)**

Figura 1. **a) conjunto de datos de entrenamiento; b) Conjunto de datos de entrenamiento con un ajuste polinomial de grado 2.**



a) **b)**

Figura 2. **a) conjunto de datos de entrenamiento con 4 ajustes polinomiales (con grados 2, 4, 22, 30); b) conjunto de datos de entrenamiento (a la izquierda de la linea vertical) y conjunto de datos de prueba (a la derecha de la linea vertical) con 4 ajustes polinomiales y un aumento de 10 % en los datos de entrenamiento.**

En la Fig. 2-b, se observa el comportamiento de (11) en datos nunca vistos del polinomio (datos de prueba, puntos a la derecha de la línea vertical), también se observa lo que ocurre cuando al conjunto de entrenamiento se le añaden más datos. Se puede concluir de la Figura 2-b y 3 que los polinomios de grado 22 y 30 no generalizan los datos de entrenamiento, y los polinomios de grado 2 y 4 generalizan los datos de entrenamiento; otra conclusión extraíble es que el aumento de datos de entrenamiento ocasiona un pequeño cambio en los parámetros de los polinomios con el mismo número de parámetros. Para entender dicho cambio ocasionado al aumentar el número de datos de entrenamiento se presenta la Fig. 3, como se observa, el aumento de datos de entrenamiento reduce considerablemente la suma de los errores (función de costo) entre la estimación $\hat{y}$ y la variable y para datos de prueba.

Este simple experimento permite obtener 2 conclusiones importantes:

- El aumento de la capacidad del modelo produce sobreajuste
- El aumento de datos de entrenamiento esta intrínsicamente relacionado con la capacidad del modelo, a mayor cantidad de datos de entrenamiento mayor capacidad del modelo para alcanzar generalización de los datos.

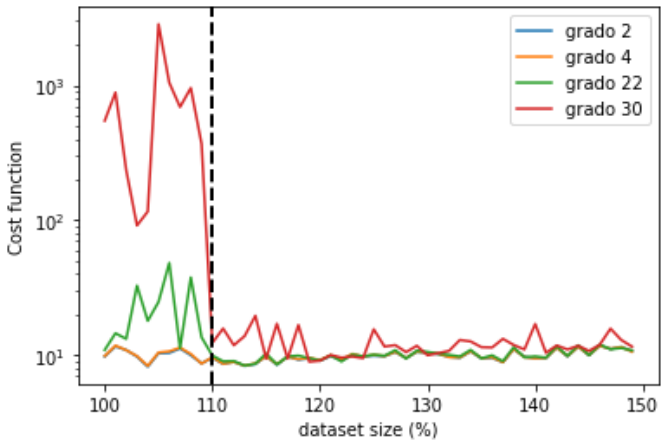


Figura 3. Evaluación de la función de perdida sobre el conjunto de prueba en función de su tamaño para 4 polinomios.

Estimación de máxima verosimilitud

Los modelos de DL se pueden considerar como modelos probabilísticos [9], esto significa que dichos modelos se pueden concebir como algoritmos de incertidumbres, y el objetivo del

entrenamiento de estos algoritmos es reducir esa incertidumbre maximizando la función de verosimilitud de los datos de entrenamiento [11].

Se describe muy brevemente la técnica de máxima verosimilitud [10].
Sea $x_1, \dots, x_n$ una muestra de variables aleatorias independientes idénticamente distribuidas tomada al azar de una población con el parámetro θ, la función de verosimilitud de la muestra esta data por

$$L(\theta) = f(x_1, \dots, x_n; \theta) \tag{12}$$

Donde $f(x_1, \dots, x_n; \theta)$ es el valor de la distribución de probabilidad conjunta de las variables aleatorias $\chi_1, \dots, \chi_n$ en el punto de la muestra $x_1, \dots, x_n$.

Entonces, el objetivo de la técnica de máxima verosimilitud es resolver el siguiente problema de optimización [4]:

$$\theta^* = \arg\max_{\theta} L(\theta) \tag{13}$$

En términos llanos, el método de máxima verosimilitud es una técnica que busca estimar el valor de un parámetro de una población, maximizando la probabilidad de ocurrencia de los datos observados.

Descenso del gradiente

Muchos modelos de aprendizaje automático y aprendizaje profundo involucran un proceso de optimización. Generalmente, estos procesos de optimización implican maximizar o minimizar una función objetivo o un criterio de evaluación [11].

El descenso del gradiente (Cauchy, 1847) es un algoritmo iterativo cuyo objetivo es estimar los puntos críticos de una función.

Si se considera un campo escalar de la forma $F : X \subseteq R^n \rightarrow R$ y se define el siguiente problema de minimización:

$$X^* = \arg\min_X F(X), \tag{14}$$

entonces el descenso del gradiente construye la siguiente regla actualización para estimar X^* [11]:

$$X_{k+1} = X_k - \alpha \nabla F(X_k). \tag{15}$$

Donde $\nabla F(X_k)$ es el gradiente de F en el punto X_k, y α es un número que suele ser llamado tasa de aprendizaje (Learning rate).

2.2 Aprendizaje Profundo (Deep Learning)

2.2.1 Redes Neuronales Profundas Hacia Adelante (DNNs)

Las redes neuronales artificiales están inspiradas en las redes neuronales biológicas. El funcionamiento muy resumido de una neurona es el siguiente: las dendritas de una neurona reciben señales de varias neuronas vecinas, estas señales pasan al núcleo o soma de la neurona para que después se sumen (Fig. 4). Si la señal supera un determinado umbral, el axón emitirá una señal a nuevas neuronas vecinas [17].

A continuación, se da un breve sumario de la historia de las redes neuronales artificiales.
Warren McCullogh y Walter Pitts publicaron en la década de 1940 un trabajo relacionado con la neurona artificial [12], este trabajo dio paso a una nueva área de investigación.

Después, en 1957 nace un nuevo tipo de neurona artificial llamada perceptrón, inventada por Frank Rosenblatt [13]. Que podía ser entrenado a través de una regla de aprendizaje. Posteriormente, en la década de 1980, Marvin Minsky y Seymour Papert mostraron las limitaciones del perceptrón [14], debido a esto las investigaciones sobre esta área cayeron rotundamente. Sin embargo, poco tiempo después, Geoffrey Hinton, Ronald William y otros revivieron a las redes neuronales artificiales con un nuevo concepto de retro-propagación para entrenar perceptrones multicapa [15].

En 2006 Geoffrey Hinton y otros introdujeron las redes de creencia profunda [16]. Este trabajo fue el inicio de un nuevo paradigma de las redes neuronales artificiales llamado aprendizaje profundo.

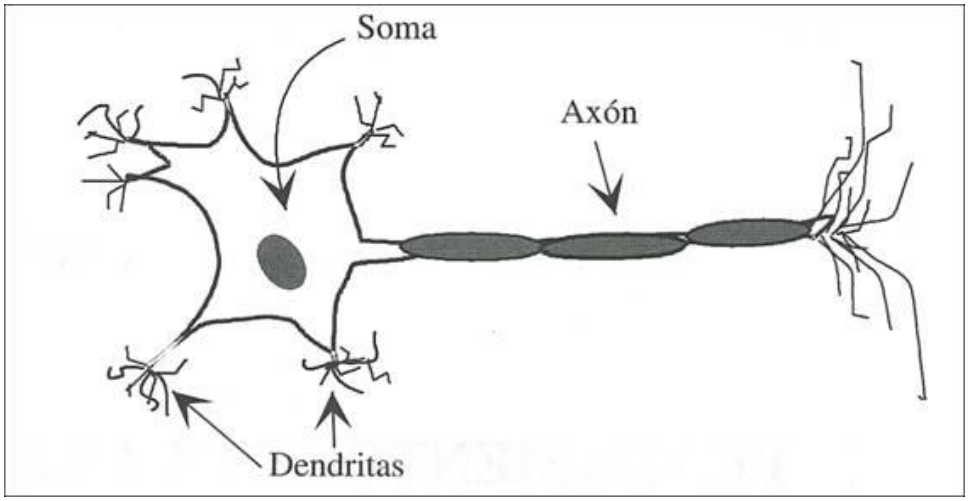


Figura 4. trazado de una neurona biológica [49]

Perceptrón

El Perceptrón consiste en una sola neurona artificial (Fig. 5) con un número ajustable de pesos. El perceptrón tiene como función ser un clasificador binario lineal que usa un hiperplano para separar clases.

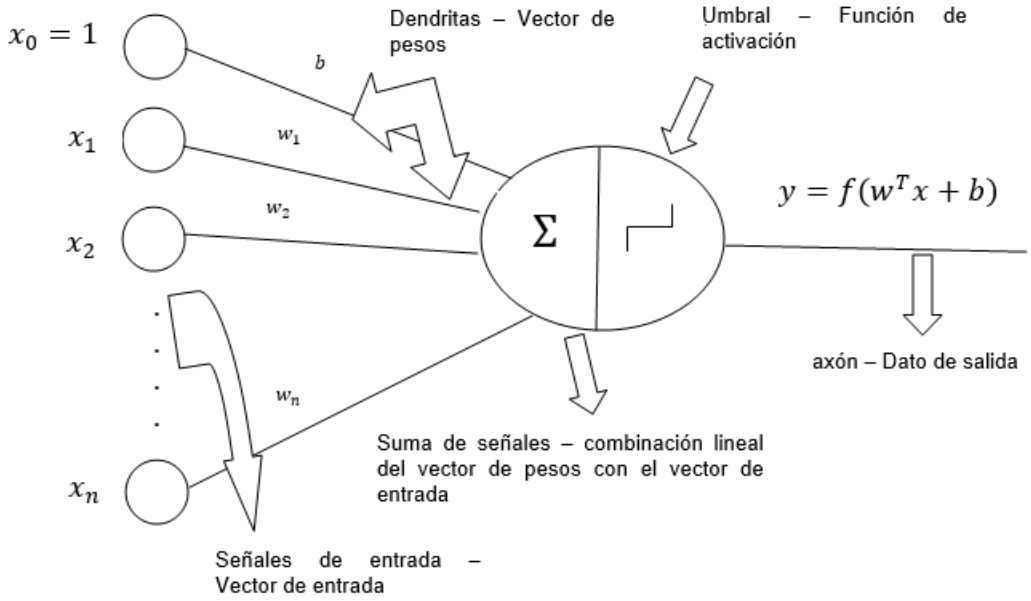


Figura 5. trazado de una neurona artificial (perceptrón)

Matemáticamente, la entrada a la neurona está representada por un vector $\mathrm{X} = (x_1, x_2, \dots, x_n)^T$ y un escalar $x_o = 1$. Los pesos de las conexiones (sinapsis en una neurona biológica) son representados por un vector $\mathrm{W} = (w_1, \dots, w_n)^T$ y sesgo w_0. La salida es calculada mediante el valor de una función evaluada en la combinación lineal del vector de entrada con el vector de pesos $f(\mathrm{W}^T\mathrm{X} + x_0 w_0)$.

Del perceptrón multicapa a una red neuronal profunda

El perceptrón multicapa suple las limitaciones del perceptrón simple en la construcción de una función no lineal entre la entrada y la salida [17]. Un perceptrón multicapa es una red neuronal con conexiones entre capas (Fig. 6).

Disintiendo un poco sobre la figura gráfica del perceptrón multicapa, es preferible pensar en el perceptrón multicapa como una función de aproximación diseñada para una tarea y su objetivo principal es alcanzar generalización sobre datos de la población.

Nada impide concatenar hacia adelante dos o varios perceptrones, cuando esto ocurre, la red neuronal o perceptrón multicapa se convierte en una DNN (Fig. 7).

Una DNN obtiene una ventaja que domina sobre el perceptrón multicapa, y es el de escoger la profundidad de la red. La profundidad de la red controla la capacidad del modelo, y eso produce la habilidad para aprender funciones más complicadas, la capacidad del modelo conlleva a que se requieran más cantidades de datos [18]. El balance entre aumentar la capacidad de una DNN añadiendo más profundidad y la cantidad de datos que se requieren, es un campo activo de investigación.

A continuación, se muestra la notación que se estará utilizando en las secciones siguientes (esta notación está totalmente influenciada por el programa especializado de Coursera: Deep Learning by Andrew Ng [19]).

Se define la siguiente notación (esta notación esta referenciada en la Fig. 7):

- $X \in R^{n\times 1}, n \geq 1$ es el vector de entrada
- $Z^{[l\rightarrow 1:L]} \in R^{n_l\times 1}, n_l \geq 1$ son las salidas de las neuronas en la capa $l-$ésima.
- $A^{[l\rightarrow 1:L]} \in R^{n_l\times 1}, n_l \geq 1$ son las evaluaciones de la función de activación a las salidas de la capa $l-$ésima.
- $W^{[l\rightarrow 1:L]} \in R^{n_l\times n_{l+1}}, n_l, n_{l+1} \geq 1$ es la matriz de pesos de la capa $l-$ésima.
- $b^{[l\rightarrow 1:L]}\ R^{n_l x 1}, n_l \geq 1$ es el vector de sesgo de la capa $l-$ésima.

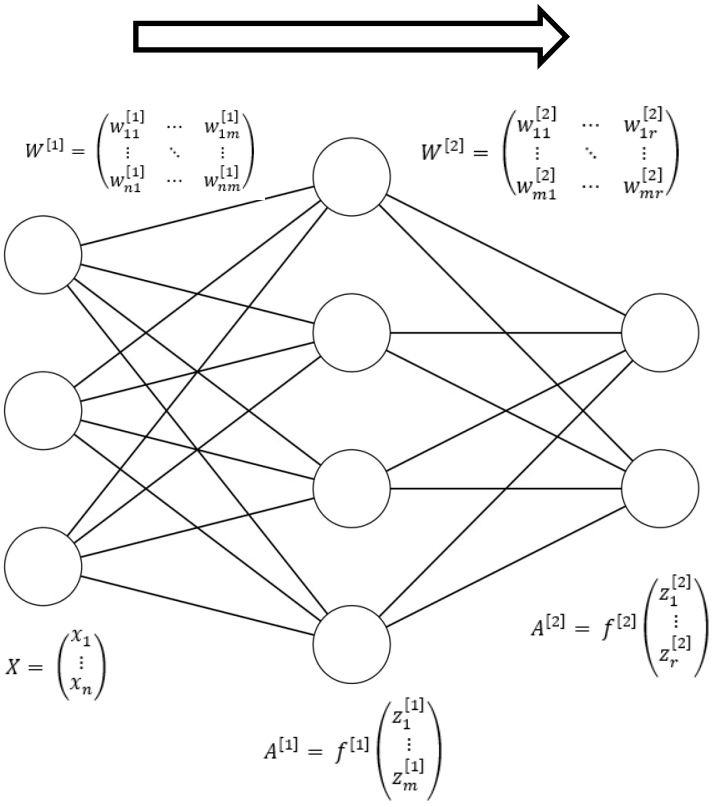


Figura 6. trazado de un perceptrón multicapa. La concatenación de perceptrones produce una matriz de pesos a diferencia de un solo vector en el perceptrón simple de la figura 2. La flecha indica la dirección de la propagación.

Propagación hacia adelante (Forward Propagation)

Una DNN es una composición de campos vectoriales de la siguiente forma

$$F(X;W): X \subseteq R^n \rightarrow A^{[1]} \subseteq R^{n_1} \rightarrow A^{[2]} \subseteq R^{n_2} \rightarrow \cdots \rightarrow A^{[L]} \subseteq R^{n_L}. \quad (16)$$

Donde se dice que F esta parametrizada por el vector de pesos W.

Como se aprecia en (16), n es la dimensión de la capa de entrada, $n_1, n_2, \ldots, n_{L-1}$ son las dimensiones de las capas ocultas y n_L es la dimensión de la capa de salida.

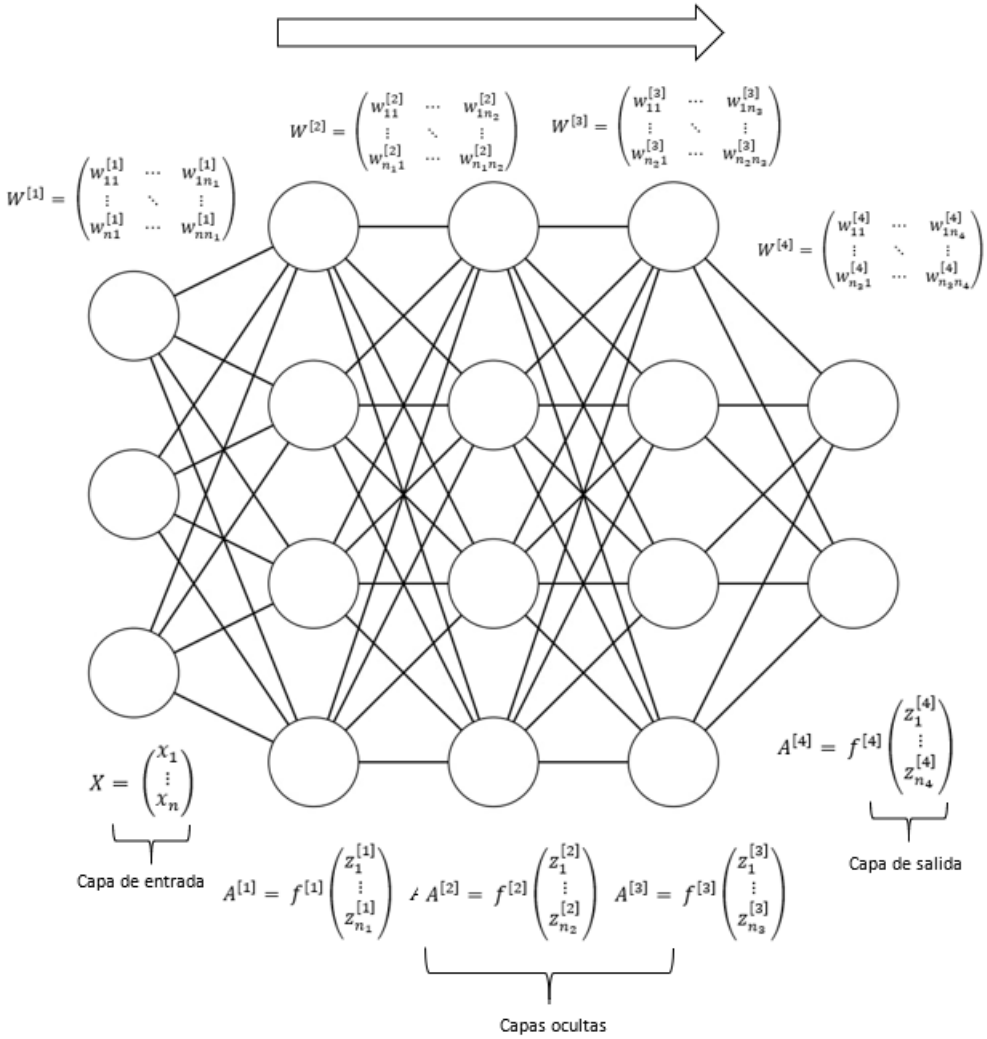


Figura 7. trazado de una DNN con una capa de entrada, tres capas ocultas y una capa de salida. La flecha indica la dirección de la propagación.

La idea de la propagación hacia adelante es propagar la información de entrada X a través de las capas ocultas y la capa salida de la DNN. La forma de propagación de la información como ya se ha explicado, es mediante una combinación lineal del vector de entrada y el vector de pesos de una capa.

Concretamente, sea $\{1, \ldots, L\}$ un conjunto de índices que indica el número de capas de la DNN. Para la $l \in \{1, \ldots, L\}$, tenemos que

$$Z^{[l]} = \left(W^{[l]}\right)^T Z^{[l-1]} + b^{[l]} \tag{17}$$

La ecuación (17) representa la propagación hacia a delante sin funciones de activación de la capa $l - ésima$.

Funciones de activación

Como se explicó en la neurona artificial (Fig. 5), la salida de una neurona es la evaluación de una función sobre la combinación lineal del vector de pesos y del vector de entrada.

Seguidamente, se muestran las funciones de activación que se utilizaron en este trabajo de tesis.

- ReLU: La función ReLU o función rectificadora se define de la siguiente manera para funciones de una sola variable real [11]:

$$f(x) = \max(0, x)\,, x \in R. \tag{18}$$

  Sin embargo, como se trabaja con campos vectoriales en las DNNs, se tiene que dar una definición de la función ReLU para campos vectoriales.

  Sea $Relu : X \subseteq R^n \rightarrow R^n$ un campo vectorial definido de la siguiente manera

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \xrightarrow{Relu(X)} \begin{pmatrix} \max(0, x_1) \\ \vdots \\ \max(0, x_n) \end{pmatrix}. \tag{19}$$

- SoftMax: La función SoftMax o también llamada función exponencial normalizada es un campo vectorial que realiza el siguiente mapeo [8]:

  Sea $Softmax : X \subseteq R^n \rightarrow R^n$ un campo vectorial definido de la siguiente manera

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \xrightarrow{softmax(X)} \begin{pmatrix} \frac{e^{x_1}}{\sum_{i=1}^{n} e^{x_i}} \\ \vdots \\ \frac{e^{x_n}}{\sum_{i=1}^{n} e^{x_i}} \end{pmatrix} \quad (20)$$

La función SoftMax puede representar una distribución de probabilidad de una variable [8], como se puede ver en (20) los valores de $softmax(X)$ están limitados en un rango de $[0,1]$ y la suma de sus elementos da como resultado 1 (cumple las condiciones del teorema 1 para distribuciones de probabilidad).

Aunque se mencionan dos funciones de activación, no hay ninguna restricción en el tipo de función de activación, entonces se puede plasmar con generalidad el siguiente campo vectorial $f^{[l]}: Z^{[l]} \subseteq R^n \rightarrow R^n$ donde l es la $l-ésima$ capa.

Entonces, la propagación hacia adelante con funciones de activación de la capa $l-ésima$ esta dada por

$$Z^{[l]} = \left(W^{[l]}\right)^T A^{[l-1]} + b^{[l]}, \quad (21)$$

$$A^{[l]} = f^{[l]}\left(Z^{[l]}\right) \quad (22)$$

Funciones de pérdida

Las DNNs son algoritmos de aprendizaje supervisado. Estos modelos paramétricos construyen una función que esta parametrizada por el parámetro W y su objetivo es mapear el vector de entrada X a un vector de salida $\hat{Y}$, donde el vector de salida tiene que acercarse (bajo cierta medida de similitud entre dos vectores) al vector etiqueta de X, denotado por Y.

Las funciones de pérdida son campos escalares que describen el error en este caso entre la salida de la DNN representada por $\hat{Y} = F(X;W)$ y la etiqueta respectiva al vector X. De modo que, una función de perdida esta descrita de la siguiente forma

$$L: \left(\hat{Y}, Y\right) \subseteq R^m \rightarrow R. \quad (23)$$

Actualmente las DNNs son concebidas como modelos paramétricos que definen una distribución de probabilidad condicional tipo $p(\hat{Y}|X;W)$ [9], esto quiere decir que el vector de salida de la DNN $\hat{Y}$, es una probabilidad condicionada por el vector de entrada X y parametrizada por la matriz W.

La función de perdida entropía cruzada (24) es usada en problemas de clasificación multi-clase y su objetivo es penalizar salidas de la DNN que se clasifiquen incorrectamente [19]; esto quiere decir que se da una penalización mayor (un valor mayor de $L(Y,\hat{Y})$) para vectores de $\hat{Y}$ que estén más lejos de Y.

$$Y = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix};\ \hat{Y} = \begin{pmatrix} \widehat{y_1} \\ \vdots \\ \widehat{y_m} \end{pmatrix};\ L(\hat{Y},Y) = -\sum_{i=1}^{m} y_i \log \widehat{y_i}. \qquad (24)$$

Propagación hacia atrás (Back Propagation)

El proceso de entrenamiento de un modelo paramétrico como una DNN consiste en ajustar sus parámetros mediante una regla [19], en función de un criterio o función de costo, y se impone un proceso de optimización a cumplir.

El método de máxima verosimilitud junto al descenso del gradiente y sus variaciones es utilizado para entrenar a las DNNs [8]. Específicamente en problemas de clasificación, una DNN producirá una salida del tipo $p(\hat{Y}|X;W)$ y usando las ecuaciones (13) y (24), se define el siguiente problema de optimización (se observa que es un proceso de minimización debido al signo negativo de la ecuación 24)

$$W^* = \arg\min_W L\big(p(\hat{Y}|X;W),Y\big). \qquad (25)$$

La regla de ajuste de parámetros para el entrenamiento de una DNN se hace aplicando el método del descenso del gradiente a la ecuación (25) [8]. Y utilizando la ecuación (15) se tiene la siguiente regla de actualización

$$W_{k+1} = W_k - \alpha\nabla L\big(p\big(\hat{Y}\big|X;W\big),Y\big). \qquad (26)$$

Hasta ahora solo se ha considerado un vector de entrenamiento X y su respectiva etiqueta Y para describir la propagación hacia adelante y la propagación hacia atrás. Sin embargo, no hay

ambigüedad en extender la propagación hacia adelante y hacia atrás en todo el conjunto de entrenamiento [19]. Lo que se debe extender es el rango de la matriz $X \in R^{n\times 1}$ a un matriz $X \in R^{n\times m}$, donde m es el número total de vectores de entrenamiento. Por lo tanto, el problema de optimización de (25) para una matriz de entrenamiento (suponiendo que los datos son muestras de variables independientes idénticamente distribuidas) se convierte en

$$W^* = \arg\min_W \frac{1}{m}\sum_{i=1}^{m} L\big(p(\hat{Y}_i|X_i;W), Y_i\big), \tag{27}$$

para m vectores de entrenamiento con sus respectivas etiquetas.

Asumiendo que los vectores de entrenamiento X tienen una distribución uniforme [8], (27) se puede considerar como un valor de expectación de la función de pérdida:

$$W^* = \arg\min_W E_{X\sim\mathcal{U}}[L\big(p(\hat{Y}|X;W), Y\big)], \tag{28}$$

donde $\mathcal{U}$ es una distribución de probabilidad uniforme definida en $(0, m]$.

Asimismo, tampoco debe haber confusión con respecto a los gradientes de (27), debido a que los gradientes se obtienen respecto a W y la suma corre en los vectores de X. Entonces el gradiente total sería la suma de los gradientes individuales de cada vector X_i.

A continuación, se muestra la deducción de los gradientes de una DNN que posteriormente se utilizará en la sección 4.2.

Se considera la DNN (16), donde, $A^{[L]}$ es el vector de salida. Haciendo $A^{[L]} = \hat{Y}$ y realizando el proceso de optimización (25), se tiene que el primer gradiente de los pesos $W^{[L]}$ y $b^{[L]}$

$$\frac{\partial L}{\partial W^L} = \left(\frac{\partial Z^L}{\partial W^L}\right)^T \big(J_{A^L}\big)^T\big(\nabla_{A^L} J\big) = \big(J_{A^L}\big)^T\big(\nabla_{A^L} J\big)(A^{L-1})^T. \tag{29}$$

Donde $J_{A^{[L]}}$ es la matriz Jacobiana (3) del vector $A^{[L]}$ con respecto al vector $Z^{[L]}$, $\nabla_{A^{[L]}} J$ es el gradiente (1) de la función de pérdida con respecto al vector $A^{[L]}$ (ver anexo A para una demostración de 29).

Obteniendo las derivadas de los pesos de las demás capas, se obtiene la siguiente regla de derivación (ver anexo B para una demostración):

Sea $J_{A^{[l]}}$ el jacobiano de la capa $A^{[l]}$ con respecto al vector $Z^{[l]}$, tal que

$$\triangle^{[l]} = \prod_{i=l}^{L-1} (J_{A^{[i]}})^T (W^{[i+1]}), \tag{30}$$

donde $\triangle^{[l]}$ es la acumulación de gradientes desde la capa $L-1$ a la capa $l-ésima$.

Entonces, se tiene que

$$\frac{\partial L}{\partial W^{[l]}} = (\triangle^{[l]}) \left(J_{A^{[L]}}\right)^T \left(\nabla_{A^{[L]}} J\right) (A^{[l-1]})^T, \tag{31}$$

$$\frac{\partial L}{\partial b^{[l]}} = (\triangle^{[l]}) \left(J_{A^{[L]}}\right)^T \left(\nabla_{A^{[L]}} J\right). \tag{32}$$

Las ecuaciones (31) y (32) son los gradientes para cualquier capa $l-ésima$.

2.2.2 Dos Métodos de Regularización Para el Aprendizaje Profundo

Los métodos de regularización son un campo activo de investigación en DL [20], los cuales involucran el desarrollo de técnicas para controlar el sobreajuste y con ello lograr una mejor generalización de los datos.

A continuación, se presentan dos de las técnicas de regularización más utilizadas en aprendizaje profundo y que se utilizaron en este trabajo de tesis: Dropout y Normalización por lotes.

Dropout

Las DNNs entrenadas en un conjunto de datos relativamente pequeño suelen sufrir de sobreajuste [8]. El Dropout es una técnica de regularización que simula una combinación aleatoria de arquitecturas de redes neuronales eliminando nodos (neuronas artificiales) aleatoriamente durante el entrenamiento [21]. Esta técnica está totalmente influenciada por el método de ensambles [8], donde un conjunto de modelos más simples reduce el sobreajuste de un modelo más complejo. Sin embargo, el entrenamiento por ensambles requiere un gasto computacional adicional de entrenamiento y mantenimiento de múltiples modelos.

Una observación importante es que, durante la etapa de prueba y predicción, el dropout se debe apagar y se debe hacer uso de la red neuronal profunda completa.

Normalización por lotes (Batch Normalization)

Loffe et al [22] observaron que en las capas ocultas de las DNNs la distribución de los valores de activación cambia constantemente durante el entrenamiento y tienden a saturarse en valores extremos. Este fenómeno ralentiza el entrenamiento y provoca sobreajuste de los datos de entrenamiento.

La técnica de normalización por lotes [22] hace frente al problema de la distribución en las capas ocultas haciendo una estandarización de la distribución de los valores de activación con respecto al tamaño del lote. De modo que, cada capa de activación es escalada por la media y la varianza calculada solo sobre un lote; este efecto de escalado reduce los valores de saturación y añade ruido a los valores de activación de las capas ocultas provocando así un efecto de regularización.

Normalización por lotes es una de las técnicas de regularización con mayor influencia en aprendizaje profundo, permitiendo entrenar eficientemente redes neuronales muy profundas [19].

2.2.3 Redes Neuronales Convolucionales (CNNs)

Las redes neuronales convolucionales en combinación con las redes neuronales profundas son de las arquitecturas o modelos del aprendizaje profundo con mayor éxito en tareas de visión por computadora como: clasificación, reconocimiento y localización de objetos, segmentación de imágenes, etc. [1].

Las CNNs son algoritmos de aprendizaje supervisado que están dotados de distintas herramientas matemáticas y computacionales para su entrenamiento eficiente. La idea de las CNNs se basa en concatenar varias capas ocultas especializadas en encontrar características relevantes de una señal (en la mayoría de los casos, la señal es una imagen en RGB) y con una jerarquía de caracterización; esto sugiere que, en las primeras capas de la CNN, se encontrarían características globales (si es una imagen, en las primeras capas se encontrarían líneas, curvas, bordes, etc.) y en las capas más profundas se encontrarían características locales de la señal [19].

En modo general, la arquitectura de una CNN se compone de capas ocultas y una o varias capas totalmente conectadas. Las capas ocultas se suelen construir vía dos tipos de capas: de convolución y de pooling. Seguidamente, después de concatenar varias capas ocultas, se suele concatenar otra red neuronal para construir la salida deseada de la tarea requerida.

Al igual que las DNNs, las CNNs se entrenan mediante el método de máxima verosimilitud junto al descenso del gradiente y sus variaciones [8].

Las redes neuronales convolucionales han sido una de las ideas más exitosas en DL, tomando su inspiración de la corteza visual de los animales (Hubel y Wiesel en 1959 realizaron investigaciones sobre el funcionamiento de la corteza visual de los animales, especialmente de las células responsables de la orientación y detección de bordes en los estímulos visuales dentro de la corteza visual primaria V1 [8]).

Yann LeCun en 1989 desarrolló la primera red neuronal convolucional, LeNet [24]. La idea de aprender kernels de convolución mediante la propagación hacia atrás para obtener características de datos estructurados como imágenes o videos, abrió una gama de posibilidades y aplicaciones en reconocimiento de patrones. Posteriormente, en 2012 se crea la red AlexNet por Hinton, Sutskever y Krizhevsky [25], esta red convolucional mostró al mundo el poder de las redes convolucionales profundas, específicamente en tareas de visión por computadora.

La operación convolución en redes neuronales convolucionales

Formalmente, la convolución es un operador que transforma dos funciones de variable real a otra función de variable real [11].

Sean f y g dos funciones en el espacio discreto, se define el operador de convolución discreto [11] como

$$(f * g)(t) = \sum_{i=-\infty}^{\infty} f(i)g(t-i). \qquad (33)$$

En terminología de DL, a la función f se le conoce como entrada (input en inglés), a la función g como kernel y a la función $(f * g)(t)$ como salida (output en inglés) o como mapa de activación.

En la práctica, la convolución en DL se suele aplicar en más de un eje. Por ejemplo, si se tiene una imagen en un espacio 2-dimensional (esto es, una matriz de dimensiones R^{lxr}), se dice que la imagen I es la entrada y se define un kernel K 2-dimensional (una matriz de dimensiones R^{nxm}), donde generalmente el kernel es más chico que la entrada. En las implementaciones de redes neuronales convolucionales, no suele usarse el operador convolución, en su lugar, se utiliza un operador relacionado llamado operador correlación cruzada [8], definido como

$$S(i,j) = (I * K)(i,j) = \sum_n \sum_m I(i+n, j+m) K(n,m). \quad (34)$$

Sin embargo, aunque en las redes convolucionales se implemente la correlación cruzada, se suele llamar convolución.
Siguiendo la terminología estándar para DL, para las siguientes subsecciones, cuando se haga referencia a la convolución, en realidad se está hablando de la correlación cruzada.

Capa de convolución

Las capas de convolución en una CNN es el proceso de aplicar la operación convolución entre entradas de una capa anterior y los kernels de la capa actual.
Se define a la $l-ésima$ capa de convolución como [19]:

$$Z^{[l]} = Z^{[l-1]} * K^{[l]}. \quad (35)$$

Donde $Z^{[l]}$ y $K^{[l]}$ es la salida y el kernel de convolución de la capa $l-ésima$ respectivamente, y $Z^{[l-1]}$ es la salida de la capa de convolución $(l-1)-ésima$ [19].

El objetivo de la salida de la capa de convolución es caracterizar la señal de entrada mediante el kernel de convolución [8]. Las primeras capas de convolución expondrán características globales de la señal, a medida que se va dando profundidad entre capa y capa se van obteniendo características locales [19].

Las CNNs obtienen la propiedad de que las características obtenidas por las capas de convolución son invariantes a la traslación [8, 19], esto es a una ventaja con respecto a varios algoritmos de aprendizaje profundo, como las DNNs.

$$I = \begin{bmatrix} I_{11} & I_{12} & I_{13} \\ I_{21} & I_{22} & I_{23} \\ I_{31} & I_{32} & I_{33} \end{bmatrix} \qquad K = \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix}$$

$$I * K = \begin{bmatrix} I_{11}K_{11} + I_{12}K_{12} + I_{21}K_{21} + I_{22}K_{22} & I_{12}K_{11} + I_{13}K_{12} + I_{22}K_{21} + I_{23}K_{22} \\ I_{21}K_{11} + I_{22}K_{12} + I_{31}K_{21} + I_{32}K_{22} & I_{22}K_{11} + I_{23}K_{12} + I_{32}K_{21} + I_{33}K_{22} \end{bmatrix}$$

Figura 8. ejemplo de la operación convolución de la señal I y el kernel K

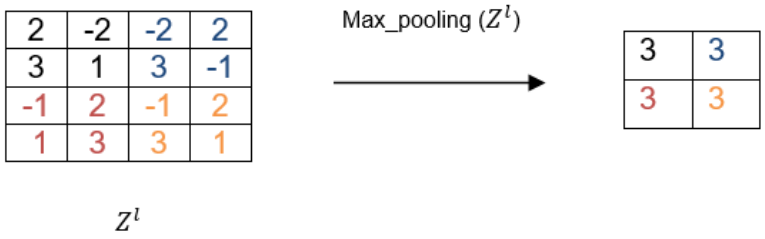


Figura 9. ejemplo de la operación Max-pooling sobre la matriz Z^l con una ventana de tamaño 2x2. Se barre en ancho y alto una ventana de tamaño 2x2 sobre la matriz Z^l dando un salto de dos. Para cada venta se obtiene el valor máximo.

Capa de pooling

La capa de pooling es una operación que se aplica regularmente a la salida de la capa de convolución. El objetivo de esta operación es la reducción del tamaño del mapa de activación, produciendo una reducción de parámetros a optimizar y por ende una rebaja en tiempo computacional.

Las capas de pooling más comunes son: max-pooling y average-pooling [19]. Las capas max-pooling y average-pooling obtienen el máximo y el promedio respectivamente sobre el tamaño de una ventana barrida en todo el mapa de activación (Fig. 9).

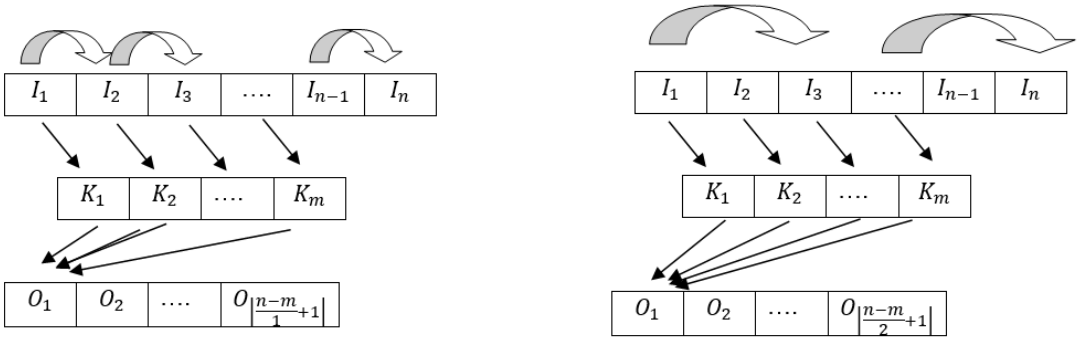


Figura 10. ejemplo de convolución con paso igual uno (figura de la izquierda) y paso igual a 2 (figura de la derecha).

Redes convolucionales con capas completamente conectadas

Las redes convolucionales completamente conectadas involucran generalmente dos pasos: el aplanamiento de la última salida de las capas ocultas y la concatenación de una o varias redes neuronales.

El aplanamiento consiste en concatenar verticalmente las filas de una matriz de dimensiones $m \times n$ para formar un vector de tamaño $m \times n$. Matemáticamente queda como

$$Sea\ Z^{[L]} = \begin{pmatrix} z_{11}^{[L]} & z_{12}^{[L]} \\ z_{21}^{[L]} & z_{22}^{[L]} \end{pmatrix} \in R^{2\times 2}\ ;\ vec\left(Z^{[L]}\right) = \begin{bmatrix} z_{11}^{[L]} \\ z_{12}^{[L]} \\ z_{21}^{[L]} \\ z_{22}^{[L]} \end{bmatrix} \in R^{4\times 1}. \qquad (36)$$

A la operación aplanamiento se le suele indicar con el símbolo $vec(\cdot)$ [11].

El vector $vec(\cdot)$ resultante del aplanamiento (36) es el vector de entrada de una DNN.

Propagación hacia adelante

Se ha descrito solamente el proceso de convolución y pooling en términos de una señal de entrada, un kernel y un mapa de activación. Sin embargo, no hay restricción en el número de kernels que se pueden imponer en las capas de convolución y tampoco hay restricción en las dimensiones de la señal de entrada.

En DL el termino tensor es usado para referirse a todas las matrices multidimensionales (un tensor es un objeto algebraico bien definido que cumple ciertas operaciones, sin embargo, no

necesariamente todas las matrices multidimensionales son tensores [23]). No obstante, se hace uso del nombre tensor para referirse a todas las matrices multidimensionales.

Se define la siguiente notación:

- $n_h^l \times n_w^l$: dimensión de la salida de la capa $l-$ ésima,
- n_c^l : profundidad de la capa $l-$ ésima o número de kernels de la capa $l-$ ésima,
- $k_h^l \times k_w^l$: dimensión del kernel $l-$ ésimo,
- s_h^l: tamaño del paso en altura del kernel $l-$ ésimo,
- s_w^l: tamaño del paso en anchura del kernel $l-$ ésimo.

Se puede representar a la señal de entrada $Z^{[l-1]}$, a n_c^l número de kernels $K^{[l]}$, y al mapa de activación $Z^{[l]}$ como tensores de la siguiente manera

- Sea $X \in R^{n_h^0 \times n_w^0 \times n_c^0}$ la señal de entrada compuesta de n_c^0 matrices de dimensiones $n_h^0 \times n_w^0$,
- Sea $Z^{[l\rightarrow 1:L]} \in R^{n_h^l \times n_w^l \times n_c^l}$ la salida $l-$ ésima de la capa de convolución, compuesta de n_c^l matrices de dimensiones $n_h^l x n_w^l$,
- Sea $b^{[l\rightarrow 1:L]} \in R$ el sesgo $l-$ ésimo,
- Sea $1^{[l\rightarrow 1:L]} \in R^{n_h^l \times n_w^l \times n_c^l}$ un tensor de unos de la capa $l-$ ésima,
- Sea $K^{[l\rightarrow 1:L]} \in R^{k_h^l \times k_w^l \times n_c^{l-1} \times n_c^l}$ el kernel $l-$ ésimo compuesto por n_c^l kernels de dimensiones $k_h^l \times k_w^l \times n_c^{l-1}$,
- Sea $Z^{[l-1]} \in R^{n_h^{l-1} \times n_w^{l-1} \times n_c^{l-1}}$ la $(l-1)-$ ésima salida de la capa de convolución, compuesta de n_c^{l-1} matrices de dimensiones $n_h^{l-1} \times n_w^{l-1}$.

Donde h, w y c hacen referencia a high, width y channels respectivamente.

En Fig. 10 se describe el termino paso, que es el tamaño del paso que el kernel hace para la operación convolución. El paso se hace tanto en anchura como en altura.

Se tiene la siguiente regla para calcular las dos primeras dimensiones de salida de la $l-esima$ capa de convolución [19]:

$$n_h^l \times n_w^l = \left(\left\lfloor \frac{n_h^{l-1} - k_h^l}{s_h^l} + 1 \right\rfloor\right) \times \left(\left\lfloor \frac{n_w^{l-1} - k_w^l}{s_w^l} + 1 \right\rfloor\right). \quad (37)$$

La propagación hacia a delante de la CNN es la misma idea que en la DNN, y es propagar la señal de entrada X por las capas de convolución a la capa de salida $Z^{[L]}$. Por lo tanto, no debe extrañar que también en las capas convolucionales se aplique una función de activación. En este caso, la función de activación está definida en todos los elementos del tensor del mapa de activación. También se agrega a la capa de salida un sesgo multiplicado por un tensor constante de unos.

Entonces, la propagación hacia a delante con funciones de activación de la $l-ésima$ capa esta dado por:

$$Z^{[l]} = A^{[l-1]} * K^{[l]} + b^{[l]}1^{[l]}, \tag{38}$$

$$A^{[l]} = f^{[l]}\left(Z^{[l]}\right). \tag{39}$$

De la misma manera, las redes convolucionales deben ser pensadas como funciones de aproximación. En el caso de las DNNs eran del tipo de la ecuación (16), para las CNNs se construyen funciones de la siguiente manera

$$F(X;K) : X \subseteq R^{n_h^0 \times n_w^0 \times n_c^0} \rightarrow \cdots \rightarrow A^{[L]} \subseteq R^{n_h^L \times n_w^L \times n_c^L}. \tag{40}$$

Como se observa en las ecuaciones (16) y (40), el proceso de construcción de la función de aproximación de la CNN es mucho más complejo que el de la función de la DNN. Este fenómeno se debe a dos factores principales: el primero es el aumento de dimensión en los datos de entrenamiento y el segundo factor es la operación que se realiza entre la entrada y los pesos (o kernels en terminología de redes convolucionales) de una capa; la convolución permite el uso compartido de pesos y la conectividad local entre valores del mapa de activación (Fig. 8). Estas dos últimas ideas permiten dilucidar la potencia de las redes convolucionales trabajando con datos estructurados como imágenes, audios o todo tipo de señales. En contraste, las DNNs carecen de conectividad local y uso compartido de pesos entre capas, debido a esto, las funciones que mapea las DNNs son más difíciles de aproximar en virtud del carecimiento de estas dos propiedades para tareas donde involucran datos estructurados [8, 11, 19].

Propagación hacia atrás

De la misma forma que en las DNNs, el proceso de entrenamiento de la CNNs consiste en ajustar mediante el descenso del gradiente (o sus variaciones) sus parámetros, en este caso, los parámetros a ajustar son los kernels. Y el proceso de optimización a cumplir es el de la ecuación (27).

En Fig. 11 se ilustra un ejemplo de arquitectura de una CNN completamente conectada, como se observa en dicha figura, la salida de la última capa de convolución va conectada a una DNN con una capa oculta, y la salida de la DNN va a una función de pérdida que penaliza los vectores de salida de acuerdo con la ecuación (24). Conforme a las ecuaciones (31) y (32), se obtiene el gradiente de la capa de entrada de la DNN (se debe de entender que la capa de entrada de la DNN es el último mapa de activación de las capas de la CNN) como $\frac{\partial L}{\partial Z^{[l]}}$. En tal caso, el objetivo de la propagación hacia atrás en las CNNs es la obtención de los siguientes gradientes: $\frac{\partial L}{\partial K^{[l]}}$ y $\frac{\partial L}{\partial b^{[l]}}$ para cada la $l-ésima$ capa.

La obtención de los gradientes $\frac{\partial L}{\partial K^{[l]}}$ y $\frac{\partial L}{\partial b^{[l]}}$ es una tarea más complicada que la obtención de los gradientes de los pesos de las DNNs, debido a esto, en el anexo C se da una demostración de las siguientes ecuaciones:

$$\frac{\partial L}{\partial K^{[l]}} = A^{[l-1]} * \frac{\partial L}{\partial Z^{[l]}}, \tag{41}$$

$$\frac{\partial L}{\partial b^{[l]}} = \sum_i \frac{\partial L}{\partial Z^{[l]}{}_i}. \tag{42}$$

Se observa en (41) que los gradientes de la capa $l-ésima$ de los kernels es el resultado de la convolución del gradiente del mapa de activación con la señal de entrada y el gradiente del sesgo es la sumatoria de los componentes del gradiente del mapa de activación.

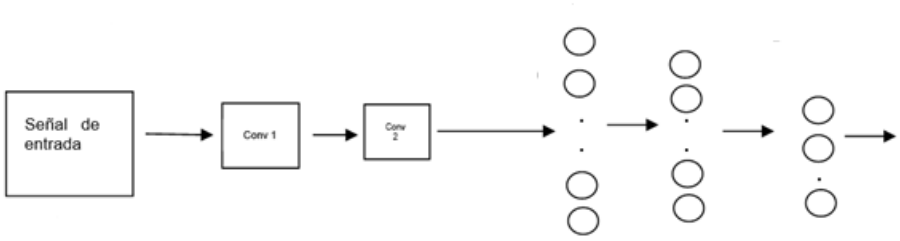


Figura 11. ejemplo de una CNN con dos capas y una red convolucional completamente conectada

2.2.4 Algoritmos de optimización para el entrenamiento de los modelos de aprendizaje profundo

Para entrenar un algoritmo de DL se debe de definir una función de costo o criterio que penalice la diferencia entre la salida del modelo y la salida deseada. El proceso de entrenamiento busca encontrar un valor de pesos W^* de forma que la función de costo sea lo más pequeña posible; este proceso de optimización sufre de dos fenómenos en funciones no-convexas como las funciones de costo en DL: los puntos silla y los mínimos locales [19].

Los puntos silla y mínimos locales ocasionan que el proceso de entrenamiento se estanque o incluso diverja. El descenso del gradiente no es un algoritmo que cumple las necesidades para entrenar modelos de aprendizaje profundo (como salirse de mínimos locales o de puntos sillas); debido a esto, se han desarrollado técnicas-trucos matemáticos que permiten al descenso del gradiente batir estas dificultades. En esta subsección se presentan el SGD, SGD por lotes, Adam y Adadelta.

Descenso del gradiente estocástico

Las aproximaciones estocásticas son una serie de técnicas desarrolladas por Herbert Robbins y Suton Monro [26] para resolver ecuaciones intratables analíticamente mediante evaluaciones aleatorias.

El descenso del gradiente estocástico (SGD por sus siglas en inglés) [27] es un algoritmo de aproximación que es usado para minimizar funciones de la siguiente forma [5]:

$$J(w) = \arg\min_{W} \; E_{X\sim P}[f(x;w)] \tag{43}$$

Donde P es una distribución de probabilidad fija y $f(x;w)$ es una función del dominio de x parametrizada por w.

En esencia, SGD es un método que consiste en la aproximación del gradiente $\nabla_w J(w)$ mediante el promedio de aproximaciones aleatorias del mismo gradiente [5].

Las ecuaciones (43) y (28) son muy similares. De tal forma, los algoritmos de DL pueden ser entrenados con el descenso del gradiente estocástico, con la diferencia respecto de su homologo (el descenso del gradiente) que en lugar de la obtención del gradiente sobre todo el conjunto de datos se obtiene solo sobre un dato aleatorio. Entonces se tiene la siguiente regla de actualización

$$W_{k+1} = W_k - \alpha \nabla L\left(p\left(\hat{Y}\middle|X = X_i; W\right), Y\right), \quad (44)$$

donde $X_i \sim U$.

Descenso del gradiente estocástico por lotes

SGD es una técnica inviable para entrenar algoritmos de DL, debido al alto costo computacional que requiere estar actualizando los parámetros solo en la evaluación del gradiente en un punto [19]. Sin embargo, en lugar de calcular el gradiente en un punto, se puede calcular en un subconjunto del conjunto de datos de entrenamiento; donde se precisa que los subconjuntos deben ser aleatorios.

Siendo así, el descenso del gradiente estocástico por lotes es una técnica que extiende a SGD con la diferencia de que, en vez de obtener un gradiente en un dato aleatorio, se obtiene el gradiente en un subconjunto de datos aleatorios. De modo que se tiene la siguiente regla de actualización

$$W_{k+1} = W_k - \alpha \frac{1}{bs}\sum_{i=1}^{bs} \nabla L\left(p\left(\hat{Y}\middle|X = X_i; W\right), Y\right), \quad (45)$$

$$X_i \sim U, bs \leq m.$$

Donde bs es el tamaño del lote y m es la cantidad de datos de entrenamiento.

El descenso del gradiente estocástico por lotes es el algoritmo estándar para entrenar modelos de DL asumiendo que los datos siguen una distribución uniforme.

Adam

En el descenso del gradiente, el hiperparámetro que determina la velocidad de convergencia del algoritmo es la tasa de aprendizaje α [11]. A causa de esto, se han desarrollado estrategias para adaptar la tasa de aprendizaje buscando una aceleración de convergencia.

La Estimación del Momento Adaptativo (Adam) [28] es un método que calcula tasas de aprendizaje adaptativo. Adam aplica un promedio ponderado exponencial de gradientes cuadrados pesados con v_k como RMSprop [29] y también mantiene un promedio ponderado exponencial de los gradientes pesados con m_k, similar a momentum [29].

Se tienen las siguientes reglas de actualización [28]:

$$g_{k+1} = \frac{1}{bs}\sum_{i=1}^{bs} \nabla L\left(p\left(\hat{Y}\middle|X = X_i; W_k\right), Y\right), \quad (46)$$

$$m_{k+1} = \beta_1 m_k + (1-\beta_1) g_{k+1}, \tag{47}$$

$$v_{k+1} = \beta_2 v_k + (1-\beta_2) {g^2}_{k+1}, \tag{48}$$

$$\hat{m}_{k+1} = \frac{m_{k+1}}{1 - \beta_1^{k+1}}, \tag{49}$$

$$\hat{v}_{k+1} = \frac{v_{k+1}}{1 - \beta_2^{k+1}}, \tag{50}$$

$$W_{k+1} = W_k - \alpha \frac{\hat{m}_{k+1}}{\sqrt{\hat{v}_{k+1} + \epsilon}}. \tag{51}$$

Donde $\alpha, \beta_1, \beta_2, \beta_1^{k+1}, \beta_2^{k+1}, \epsilon$ son hiperparámetros.

Adadelta

El método Adadelta [30] es una extensión de Adagrad [31] que busca reducir la tasa de aprendizaje agresiva y monótonamente decreciente. En lugar de acumular todos los gradientes cuadrados anteriores, Adadelta restringe la ventana de gradientes pasados acumulados a algún tamaño fijo.

Primero, se calcula el gradiente, luego se hace un promedio ponderado exponencial del cuadrado del gradiente. Después se define un nuevo gradiente sustituto y se hace un promedio ponderado exponencial al cuadrado.

La regla de actualización queda de la siguiente manera [30]:

$$g_{k+1} = \frac{1}{bs} \sum_{i=1}^{bs} \nabla L\left(p\left(\hat{Y} \middle| X = X_i; W_k\right), Y\right), \tag{52}$$

$$v_{k+1} = \rho v_k + (1-\rho) {g^2}_{k+1}, \tag{53}$$

$$\nabla W_{k+1} = -\frac{\sqrt{s_k + \epsilon}}{\sqrt{v_{k+1} + \epsilon}} g_{k+1}, \tag{54}$$

$$s_{k+1} = \rho s_k + (1-\rho) {\nabla W^2}_{k+1}, \tag{55}$$

$$W_{k+1} = W_k + \nabla W_{k+1}, \tag{56}$$

Donde ρ, ϵ son hiperparámetros.

Capítulo 3

3 Estado del arte

En la literatura se han descrito varias estrategias para acelerar la convergencia de SGD. La mayoría de ellas se centran en mejorar la forma en que se seleccionan las instancias de datos para formar los lotes de entrenamiento. En este capítulo se hace referencia a dichas estrategias como estrategias de muestreo selectivo. La idea clave detrás de estas estrategias es diseñar una distribución no uniforme para reemplazar la distribución uniforme que generalmente se emplea para muestrear instancias de entrenamiento.

3.1 Estrategias de Muestreo Selectivo

En la literatura, se han identificado tres principales enfoques al problema del muestreo selectivo [32,36] (Figura 12).

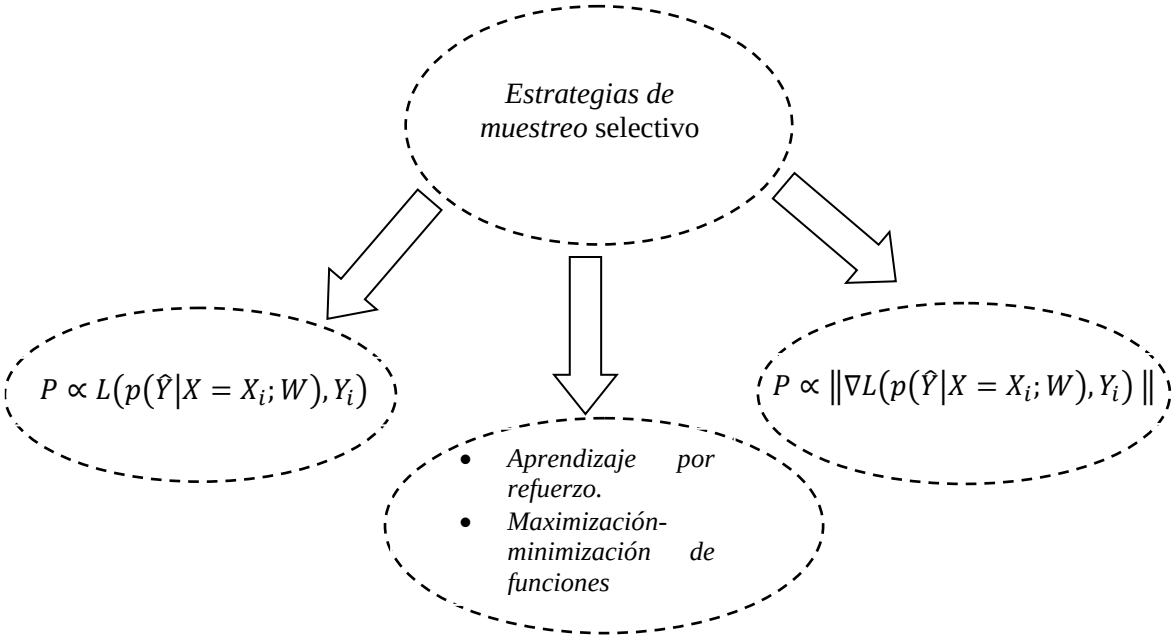


Figura 12. Tres enfoques del muestreo selectivo.

El primer enfoque construye una distribución de probabilidad de acuerdo con la norma de gradiente de cada instancia de entrenamiento; esto se hace con el objetivo de reducir la varianza entre los gradientes de las instancias muestreadas. Bajo este enfoque, Katharopoulos & Fleuret [32] propusieron acelerar el entrenamiento de una DNN usando una distribución de probabilidad basada en la norma de gradiente de cada instancia; ellos observaron que es posible acelerar la convergencia del SGD muestreando de una distribución que minimice la siguiente expresión [32]:

$$S = -E_{p_t}[\|W_{t+1} - W^*\|_2^2 - \|W_t - W^*\|_2^2], \tag{57}$$

donde p_t es una distribución de probabilidad definida sobre los datos de entrenamiento que depende de la iteración *t.* Katharopoulos & Fleuret realizaron una derivación de (57) para llegar a la conclusión de que se puede ganar una aceleración en la convergencia del SGD muestreando datos de una distribución que minimice $Tr\left(V_{p_t}\left[w_i \nabla L\left(p\left(\hat{Y}\middle|X = X_i; W\right), Y_i\right)\right]\right)$. También derivaron un límite superior de la norma del gradiente definiendo el problema de optimización (58) y lo usaron para estimar los momentos más convenientes para cambiar entre muestreo uniforme e IS a lo largo del proceso de entrenamiento.

$$\min_P E_{p_t}\left[\left\|G_{I_t}\right\|_2^2\right] \leq \min_P E_{p_t}\left[w_{I_t}^2 \widehat{G_{I_t}^2}^2\right]. \tag{58}$$

Donde $w_{I_t}^2$ es el peso de importancia al cuadrado de la iteración $t - ésima$ y $\widehat{G_{I_t}^2}$ es el límite superior de la norma del gradiente en la iteración $t - ésima$.

El método de Katharopoulos & Fleuret [32] mostró una reducción de la función de costo hasta un orden de magnitud y una mejora en los errores de prueba entre el 5 % y el 17 % para una tarea de clasificación usando una CNN.

Del mismo modo, Alain et al. [33] emplearon la norma del gradiente para definir la importancia de cada instancia. Zhao y Zhang [34] y Needell et al. [35] mostraron que la distribución de muestreo óptima es proporcional a la norma de gradiente por instancia y establecieron una conexión clara con la varianza de las estimaciones del gradiente de SGD. Gopal [36] definió una distribución de clases que es directamente proporcional a la norma de los gradientes, para tomar

muestras de aquellas clases con instancias que conducen a la máxima reducción en la varianza de los gradientes.

El segundo enfoque del muestreo selectivo consiste en construir una distribución de probabilidad basada en el valor de pérdida de cada instancia individual de entrenamiento. La función de pérdida es una medida de qué tan bien un modelo puede resolver instancias particulares. Si la pérdida de una instancia es alta, entonces debe muestrearse con más frecuencia para que el modelo pueda tener más oportunidades de aprender a resolverla. La idea es suministrar preferentemente a la DNN instancias más difíciles de clasificar o pronosticar, ya que esto debería hacer más eficiente el proceso de aprendizaje. Siguiendo este enfoque, Loshchilov & Hutter [37] emplearon la función de pérdida para definir la importancia de cada instancia, clasificaron las instancias con respecto su último valor de pérdida conocido y construyeron una distribución de probabilidad que decae exponencialmente en función de esa clasificación. Sus resultados experimentales los realizaron sobre el conjunto de bases MNIST utilizando una CNN para su proceso de clasificación, su estrategia acelera por un factor de 5 tanto para los optimizadores Adadelta [30] y Adam[28].

Asimismo, Schaul [38] y Katharopoulos & Fleuret [39] utilizan la pérdida para crear sus distribuciones muestrales. Mantienen un historial de pérdidas para instancias vistas anteriormente y para nuevas iteraciones de entrenamiento, muestrean instancias proporcionalmente a esa pérdida. Katharopoulos & Fleuret demuestran el siguiente teorema [39]:

Teorema 3: Sean $G_i = \left\|\nabla L\left(p\left(\hat{Y}\middle|X = X_i; W\right), Y_i\right)\right\|$ y $M = \max G_i$. Tal que $\exists\, K > 0 \ \wedge C < M$, entonces

$$\frac{1}{K} L\left(p\left(\hat{Y}\middle|X = X_i; W\right), Y_i\right) + C \geq G_i, \qquad \forall i. \tag{59}$$

La ecuación (59) exhibe una relación entre la norma del gradiente y la función de perdida de una DNN. En conjunto con (59), Katharopoulos & Fleuret demuestran experimentalmente que el muestreo con respecto a la función de perdida exhibe propiedades reductoras de la varianza

similares al muestreo con respecto a la norma del gradiente. La estrategia de Katharopoulos & Fleuret [39] da como resultado un entrenamiento 30% más rápido de una CNN para CIFAR10.

Además de las estrategias de Muestreo Selectivo, también se han presentado otras propuestas para acelerar el proceso de entrenamiento no solo de las DNN si no de cualquier algoritmo de aprendizaje máquina que sea entrenado con SGD. Fan y col. [40] utilizaron el aprendizaje por refuerzo para entrenar una red neuronal que le llaman red neuronal de filtrado, esta red selecciona instancias con el fin de optimizar la convergencia de una segunda red neuronal automatizando la selección de datos y adaptando el proceso de entrenamiento. El uso de la red neuronal de filtrado [40] para tareas de clasificación usando una CNN, logra una precisión comparable con el método estándar de SGD utilizando menos datos y menos iteraciones.

Joseph et al. [41] diseñaron una estrategia de selección de mini lotes basada en la maximización de una función submodular que captura información relevante de los datos utilizados para acelerar el entrenamiento. La idea consiste en maximizar una función submodular resultado de la combinación lineal de puntuaciones arrojadas por mediciones sobre los datos o sobre la DNN; una de las puntuaciones propuestas la llaman: puntuación de incertidumbre $[U(x_i)]$. La puntuación de incertidumbre es una medida sobre la incertidumbre de cada dato de entrenamiento, esta incertidumbre es la entropía del modelo en la $t-ésima$ iteración [41]:

$$U(x_i) = -\sum_{y \in C} P(y|x_i, W^t) \log P(y|x_i, W^t)\,. \qquad (60)$$

La ecuación (60) tiene la misma estructura que la entropía cruzada (24), con la diferencia que $P(y|x_i, W^t)$ en (60) va en lugar del vector de pesos en (24).
Joseph et al. [41] demuestran que su estrategia de selección de lotes proporciona una mejor generalización entrenando una CNN sobre la base de datos MNIST.

Zhao & Zhang [42] dividieron el conjunto de datos en grupos con baja varianza, y luego muestrearon estratégicamente lotes de esos grupos para mejorar la convergencia de SGD, también muestran un desarrollo teórico extenso de las tasas de convergencia de SGD.

Finalmente, Wu et al. [43] diseñaron una distribución que maximiza la diversidad de las pérdidas en un lote de entrenamiento.

3.2 Selección De Lotes en Línea Para Un Entrenamiento Mas Rápido De Redes Neuronales (OBS por sus siglas en ingles)

OBS es una estrategia de muestreo selectivo del estado del arte para acelerar el entrenamiento de las DNNs; desarrollada por Loshchilov & Hutter [37], proponen una estrategia para la selección de datos de entrenamiento con el objetivo de acelerar la convergencia de la función de costo y para los optimizadores Adam y Adadelta.

La descripción de la estrategia es la siguiente: primero todos los datos de entrenamiento son ordenados de manera descendiente con respecto a su último valor de la función de perdida; cada dato de entrenamiento esta referenciado o indexado a una distribución de probabilidad con respecto al valor de la función de perdida en cada actualización. Después, los datos son seleccionados para el proceso de entrenamiento creando un arreglo de la forma $a_j \leftarrow \sum_{i=1}^{j} pi \quad \forall\, j = 1, \dots, N$, donde N es el número de datos de entrenamiento. A continuación, se genera un número aleatorio $r \in (0,1)$ y se selecciona el menor índice i_{sel} tal que se cumpla $r < a_{isel}$.

La distribución de probabilidad que proponen es una función decreciente que depende de hiperparámetros como el número de datos de entrenamiento, y lo que ellos llaman la presión de selección.

OBS muestra buenos resultados con respecto a una aceleración de convergencia para los 2 optimizadores antes mencionados, logrando por un factor de 5 una aceleración en la función de costo. OBS resuelve la tarea de clasificación para la base de datos MNIST por medio de una red neuronal convolucional.
Sin embargo, los autores muestran sus resultados preliminares sobre la base de datos CIFAR-10, y comentan que OBS es un poco mejor que random sampling, pero no mejor que shuffling.

También argumentan que OBS funciona bien siempre y cuando los rangos de datos con respecto a los valores de pérdida sean relativamente estables con el tiempo.

3.3 Muestreo adaptativo para SGD mediante la explotación de información colateral (AS por sus siglas en ingles)

AS es una estrategia de muestreo selectivo para mejorar la convergencia de la función de costo mediante el optimizador SGD; desarrollada por Gopal [36], propone un nuevo mecanismo de muestreo de datos de entrenamiento mediante la explotación de información colateral que está asociada directa o indirectamente a los datos.

En este trabajo, Gopal propone una distribución de probabilidad dinámica sobre subconjuntos de datos en lugar de datos individuales. Específicamente, el mantiene una distribución de probabilidad sobre las clases de los datos, y durante el entrenamiento, se va adaptando para alcanzar la máxima reducción de varianza del gradiente. La idea intuitiva es que se realice un muestreo en aquellas regiones de los datos que tengan una mayor contribución de gradiente.
Gopal explica que la naturaleza de los subconjuntos de los datos debe ser lo suficientemente informativa para que exista una separación natural de los mismo y así evitar una homogeneidad en la partición del conjunto de datos de entrenamiento. Es por esta razón, que la idea más natural de partición de los datos es por medio de su etiqueta o clase.

AS realiza experimentos en 3 conjuntos de datos: ALOI, CIFAR y IPC. La regresión logística binaria y la regresión logística uno contra el resto múltiple es usada para el proceso de clasificación de estos 3 conjuntos datos. AS mostró una aceleración de la convergencia de la función de costo con respecto a la estrategia de muestreo uniforme de datos y con respecto a la estrategia de Needell et al. [35].

Capítulo 4

4 Propuesta, Metodología y Resultados

Este capítulo esta divido en cuatro secciones: la sección 4.1 describe la propuesta de este trabajo de tesis (AS). En la sección 4.2 se hace una descripción del funcionamiento del algoritmo computacional. En la sección 4.3 se reportan los resultados experimentales sobre la base de datos MNIST [44] y se hace la comparación contra un algoritmo del estado del arte: OBS [37]. Finalmente, en la sección 4.4 se reserva para discusiones y observaciones de la sección 4.3.
Para la sección 4.1 se hace el cambio a la notación de Katharopoulos & Fleuret [32].

4.1 Muestreo Adaptativo Para el Aprendizaje Profundo (Propuesta)

El muestreo adaptivo para el aprendizaje profundo es la propuesta desarrollada en este trabajo de tesis. Esta propuesta es una estrategia de muestreo selectivo que extrapola el trabajo de Gopal [36] de acelerar el entrenamiento de un perceptrón multicapa a acelerar el entrenamiento de modelos de aprendizaje profundo. Siguiendo este enfoque, a continuación, se expone el desarrollo del muestreo adaptativo para el aprendizaje profundo.

Sea $D = \{(x_1, y_1), (x_2, y_1), \dots, (x_N, y_N)\}$, $x_i \in R^n, y_i \in R^m, \forall i = 1, \dots, N$ un conjunto de entrenamiento, donde X representa el conjunto de instancias de datos y Y sus respectivas etiquetas, sea $\varphi(x_i; \theta) : R^n \to R^m$ un modelo de aprendizaje profundo (un campo vectorial como se definió en 16 y 40) para clasificación de m clases, parametrizado por el vector θ, y sea $L(\varphi(x_i; \theta), y_i) : R^m \to R^+$ la función de costo (un campo escalar como se definió en 23) para ser minimizada durante el entrenamiento. El entrenamiento con SGD significa encontrar el vector óptimo θ^*, usualmente resolviendo:

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, E_{x_i \sim \mathcal{U}}[L(\varphi(x_i; \theta), y_i)] = \underset{\theta}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} L(\varphi(x_i; \theta), y_i). \qquad (61)$$

Donde $\mathcal{U}$ representa la distribución uniforme. Guillaume et al. [5] mostraron que SGD es una herramienta poderosa para optimizar la distribución muestral de los estimadores Monte Carlo (MC), motivando la idea de emplear una distribución muestral que no es constante en el tiempo.

IS puede verse como un método para estimar (61), sustituyendo una distribución uniforme por una no uniforme. Por tanto, tenemos un estimador de MC de la forma [5]:

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, E_{x_i \sim \mathcal{V}}[\zeta(x_i) L(\varphi(x_i;\theta), y_i)] = \underset{\theta}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \frac{1}{\mathcal{V}(x_i)} L(\varphi(x_i;\theta), y_i). \quad (62)$$

Donde $\zeta(x_i) = \frac{\mathcal{U}(x_i)}{\mathcal{V}(x_i)}$ es llamado el peso de importancia y $\mathcal{V}$ la distribución de importancia [5] (en la sección 2.1.2 se proveen los pasos y definiciones para entender mejor IS).

Siguiendo el trabajo de Gopal [36], asociamos información colateral a cada x_i. Esta información colateral puede representar una etiqueta de clase, una medida que es inherente a cada instancia, o puede ser cualquier etiqueta asociada con x_i que nos permita crear una partición C del conjunto de datos $C = \{c_1, c_2, \dots, c_k\}$, $c_j = \left\{x_1^j, x_2^j, \dots, x_{m_j}^j\right\}$ $\forall j = 1, 2, \dots, k; m_j = \left|c_j\right|$. Esto es, las N instancias se dividen en k contenedores mutuamente excluyentes, que no están necesariamente equilibrados.

Se define la distribución de probabilidad sobre los contenedores como: $P = \{p_1, p_2, \dots, p_k\}$, $p_j = \mathcal{P}(C = c_j)$; y la distribución de probabilidad sobre los elementos de cada contenedor como: $Q^j = \left\{q_1^j, q_2^j, \dots, q_{|I_j|}^j\right\}, q_i^j = \mathcal{P}\left(c_j = x_{i \in I_j}^j\right)$, donde $I = \{I_1, I_2, \dots, I_k\}$ son un conjunto de índices de la partición C. En esta notación, un superíndice es usado para indicar al $j-$ésimo contenedor y un subíndice para indicar al $i-$ésimo elemento del $j-$ésimo contenedor.

Asumiendo que P y Q^j son independientes, la probabilidad de elegir la $i-$ésima instancia de entrenamiento viene dada por:

$$\mathcal{P}(X = x_i) = p_j q_i^j. \quad (63)$$

Usando este resultado, y definiendo $\mathcal{V}(x_i)$ como $p_j q_i^j$, se reescribe (6) como:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{N} \sum_{c_j \in C} \sum_{i \in c_j} \frac{1}{p_j q_i^j} L\left(\varphi\left(x_i^j;\theta\right), y_i^j\right). \quad (64)$$

Este problema se resuelve mediante el descenso de gradiente, por lo que requerimos el gradiente d^t en la iteración de entrenamiento t:

$$d^t = \frac{1}{N}\frac{1}{p_j q_i^j}\nabla_\theta L\left(\varphi\left(x_i^j;\theta^{t-1}\right), y_i^j\right). \quad (65)$$

La contribución principal de este trabajo de investigación es la propuesta de emplear distribuciones no uniformes para P y Q^j, definidas para reducir la varianza total de los gradientes y mejorar la estimación de la dirección de descenso.

En el trabajo de Gopal (del que se deriva esta propuesta) la distribución de probabilidad P se encuentra en resolver el problema de minimización (66), considerando a las distribuciones de probabilidad Q^j como estáticas (distribución uniforme). Gopal solo formula en su propuesta una distribución dinámica para P y no para Q^j.

Formalmente, se definen dos problemas de minimización, (66) y (71) respectivamente: el primero es la minimización de la varianza del gradiente con respecto a P, asumiendo un Q^j fijo; el segundo problema es la contraparte del primero, es decir, la minimización de la varianza del gradiente con respecto a cada Q^j asumiendo una P fija.

Se considera a cada elemento del vector d^t como una variable aleatoria independiente idénticamente distribuida, siguiendo el teorema 2 y la ecuación (7), se define el primer problema como:

$$\min_P V[d^t] = \min_{p_1,\ldots,p_k}\left(\mathbb{E}\left[d^{t\mathrm{T}}d^t\right] - \mathbb{E}[d^t]^{\mathrm{T}}\mathbb{E}[d^t]\right). \quad (66)$$

Específicamente, (66) es la suma de las varianzas individuales de los componentes de todos los gradientes del modelo de aprendizaje profundo, considerando a cada varianza como una variable aleatoria independiente. Evidentemente, con una inspección rápida en las ecuaciones de propagación hacia atrás, los gradientes que no pertenecen a una misma capa no son independientes de su capa anterior. Sin embargo, en este trabajo de tesis, se considera esta suposición por razones de una simplificación matemática.

La solución a (66) se debe a Gopal [36] y da la distribución óptima P en la $t-$ é$sima$ iteración:

$$p_j \propto \frac{1}{N}\sqrt{\sum_{i\in c_j}\frac{1}{q_i^j}\left\|\nabla_\theta L(\varphi\left(x_i^j;\theta^{t-1}\right), y_i^j)\right\|^2}. \quad (67)$$

Como se observa, (67) representa una acumulación ponderada de la norma al cuadrado de los gradientes del contenedor $j-ésimo$.

Usando la idea de Gopal de reformular la distribución uniforme como el producto de una distribución para los contenedores por una segunda distribución para las instancias, por ejemplo $1/N = (|c_j|/N)(1/|c_j|)$, de (64) se tiene:

$$\frac{1}{N}\sum_{c_j\in C}\sum_{i\in c_j}\frac{1}{p_j q_i^j}L\left(\varphi(x_i^j;\theta),y_i^j\right) = \sum_{c_j\in C}\frac{1}{p_j}\frac{|c_j|}{N}\frac{1}{|c_j|}\sum_{i\in c_j}\frac{1}{q_i^j}L\left(\varphi(x_i^j;\theta),y_i^j\right). \tag{68}$$

Como la distribución de probabilidad de los contenedores es constante en (68), se observa que la expresión $\frac{1}{p_j}\frac{|c_j|}{N}$ es constante cuando se corren los índices sobre el sumatorio externo. Entonces, se define el siguiente problema de optimización definido solo para las instancias de un contenedor en particular:

$$\theta_j^* = \underset{\theta}{\operatorname{argmin}}\frac{1}{|c_j|}\sum_{i\in c_j}\frac{1}{q_i^j}L\left(\varphi(x_i^j;\theta),y_i^j\right), \forall j = 1,\dots,k. \tag{69}$$

Para el cual el gradiente d_j^t en la iteración de entrenamiento t es:

$$d_j^t = \frac{1}{q_i^j}\frac{1}{|c_j|}\nabla_\theta L(\varphi(x_i^j;\theta^{t-1}),x_i^j). \tag{70}$$

Luego, de manera similar a (66) se plantea la minimización de la varianza del gradiente, pero esta vez con respecto a cada Q^j:

$$\min_{Q^j} V[d_j^t] = \min_{q_1^j,\dots,q_{|I_j|}^j}\left(\mathbb{E}\left[d_j^{t\mathrm{T}}d_j^t\right] - \mathbb{E}[d_j^t]^{\mathrm{T}}\mathbb{E}[d_j^t]\right), \forall j = 1,\dots,k \tag{71}$$

donde se encuentran las siguientes expectaciones:

$$\mathbb{E}\left[d_j^{t\mathrm{T}}d_j^t\right] = \sum_{i\in c_j} q_i^j \frac{1}{\left(q_i^j\right)^2}\frac{1}{\left(|c_j|\right)^2}\left\|\nabla_\theta L(\varphi(x_i^j;\theta^{t-1}),x_i^j)\right\|^2 \tag{72}$$

$$\mathbb{E}[d_j^t] = \sum_{i\in c_j} q_i^j \frac{1}{|c_j|}\frac{1}{q_i^j}\nabla_\theta L\left(\varphi(x_i^j;\theta^{t-1}),x_i^j\right) = \frac{1}{|c_j|}\sum_{i\in c_j}\nabla_\theta L\left(\varphi(x_i^j;\theta^{t-1}),x_i^j\right) \tag{73}$$

Se observa que $\mathbb{E}[d_j^t]$ es independiente de q_i^j y se puede ignorar. Solo queda la expresión $\mathbb{E}\left[d_j^{t\mathrm{T}}d_j^t\right]$ y se procede a minimizar la varianza siguiendo la solución de Gopal.

Haciendo $\alpha_i = \frac{1}{(|c_j|)^2} \left\| \nabla_\theta L(\varphi(x_i^j; \theta^{t-1}), x_i^j) \right\|^2$ y reescribiendo el proceso de optimización (71), se tiene que:

$$\min_{q_1^j,\ldots,q_{|I_j|}^j} \sum_{i \in c_j} \frac{\alpha_i}{q_i^j} \;\; ; \;\; s.t \sum_{i \in c_j} q_i^j = 1 \, , p_i^j > 0, \;\; \forall_i, \; i = 1, \ldots, |c_j|, \tag{74}$$

Introduciendo multiplicadores de Lagrange

$$H\left(q_1^j, \ldots, q_{|I_j|}^j, \lambda\right) = \sum_{i \in c_j} \frac{\alpha_i}{q_i^j} \; + \lambda \left(\sum_{i \in c_j} q_i^j \; - 1 \right). \tag{75}$$

Derivando (75) e igualando a cero:

$$\frac{\partial H\left(q_1^j, \ldots, q_{|I_j|}^j, \lambda\right)}{\partial q_i^j} = \frac{-\alpha_i}{\left(q_i^j\right)^2} + \lambda = 0 \tag{76}$$

De modo que:

$$q_i^j = \frac{\sqrt{\alpha_i}}{\sqrt{\lambda}}, \text{ y se debe cumplir que } \sqrt{\lambda} = \sum_{i \in c_j} \sqrt{\alpha_i} \tag{77}$$

Finalmente, la distribución óptima Q^j en la $t - ésima$ iteración viene dada por:

$$q_i^j \propto \frac{1}{|c_j|} \left\| \nabla_\theta L(\varphi(x_i^j; \theta^{t-1}), y_i^j) \right\| \tag{78}$$

Intuitivamente, la propuesta de este trabajo se puede entender como dos procedimientos de selección de datos de entrenamiento: primero, de acuerdo con (67) se eligen contenedores que tienen una mayor contribución de gradiente en relación con otros contenedores; luego, de acuerdo con (78), se muestrean las instancias con la contribución de gradiente más grande dentro de ese contenedor. La motivación detrás de esta propuesta es que estos dos procedimientos de selección deberían de dar como resultado un muestreo de instancias de datos de entrenamiento más eficiente que el muestreo uniforme, lo que debería traducirse en una reducción del tiempo requerido para el entrenamiento de una DNN.

4.2 Algoritmo computacional

El muestreo adaptativo de Gopal fue desarrollado como un mecanismo para muestrear instancias de entrenamiento con el objetivo de mejorar la convergencia de SGD para el entrenamiento de un perceptrón multicapa. Las DNNs son modelos que proceden del perceptrón multicapa, sin

embargo, el entrenamiento de estos modelos usando el enfoque de muestreo adaptativo presenta dificultades técnicas que en el trabajo de Gopal no se presentaron. A continuación, se describen estas dificultades.

Se encuentran tres grandes problemas en calcular las distribuciones de probabilidad (67) y (78).

1. Para (67), no es práctico restablecer la distribución de muestreo óptima después de cada iteración. La solución propuesta por Gopal [36] es actualizar p_j cada β iteraciones, lo que significa volver a muestrear algunas instancias de entrenamiento de cada contenedor (la cantidad apropiada de instancias depende del tamaño del mini-lote). La desventaja de este enfoque es que hay que ajustar un nuevo hiperparámetro β.
2. Para (78), calcular la norma de gradiente para una instancia individual requiere calcular una raíz cuadrada para cada instancia (se observa que esto no se requiere en (67)). Katharopoulos y Fleuret [32] muestran que el muestreo que utiliza la función de pérdida presenta propiedades de reducción de la varianza similares al muestreo de acuerdo con la norma de gradiente (teorema 3). Usando ese resultado, reemplazamos (78) con:

$$q_i^j \propto L\left(\varphi\left(x_i^j;\theta\right), y_i^j\right) \tag{79}$$

3. La última dificultad se encuentra en el cálculo de la norma de gradiente. Los modelos de DL se caracterizan por tener cientos de miles o millones de parámetros (por ejemplo, RestNet-50 tiene 23 millones de parámetros entrenables [45]), por lo que calcular el gradiente completo se vuelve prohibitivo computacionalmente. Para superar este obstáculo, Katharopoulos y Fleuret [39] derivaron un límite superior de la norma de gradiente (este límite superior hace cumplir la ecuación 58):

$$\hat{G}_i \equiv \left\| \nabla_{\theta_t^L} L\left(\varphi\left(x_i^j;\theta^{t-1}\right), y_i^j\right) \right\|^2 \geq \left\| \nabla_{\theta} L\left(\varphi\left(x_i^j;\theta^{t-1}\right), y_i^j\right) \right\|^2 \tag{80}$$

   Donde $\nabla_{\theta_t^L} L\left(\varphi\left(x_i^j;\theta^{t-1}\right), y_i^j\right)$ es el gradiente con respecto a las salidas de preactivación de la última capa en una DNN. Este límite superior se emplea en lugar del gradiente completo, lo que permite la implementación de la propuesta.

El pseudocódigo de esta propuesta para mejorar el muestreo adaptativo para el entrenamiento de modelos DL se da en **Algoritmo 1**.

El objetivo del **Algoritmo 1** es la formación de lotes mediante un muestreo no uniforme para mejorar la convergencia de SGD. A continuación, se describen los principales pasos de este algoritmo.

Como primer paso se obtiene una instancia de entrenamiento para formar un lote, esto se hace muestreando un contenedor y después muestreando una instancia del contenedor muestreado (líneas 6 y 7). Este paso se hace hasta formar un lote de tamaño fijo.
Como segundo paso se obtienen los valores de perdida de cada instancia muestreada y se guarda en un arreglo (línea 8), después se calcula el límite superior del gradiente y se acumula ese valor en un vector que representa la acumulación de gradientes de cada contenedor (líneas 10 y 11).
Como tercer paso, se recalcula la distribución de probabilidad de los contenedores cada β iteraciones (línea 13), y posteriormente se recalcula la distribución de probabilidad de los elementos de cada contenedor una vez formado el lote (línea 21).
Finalmente, una vez obtenido el lote, se realiza la actualización de parámetros para el entrenamiento del modelo (línea 20)

Durante el entrenamiento, las distribuciones de probabilidad P y Q^j pueden tender a ser cero para algunos contenedores o algunos datos individuales, respectivamente. Este comportamiento hace que esos contenedores y datos individuales no se vuelvan a seleccionar nunca. Para evitar estos problemas, P y Q^j se suavizan para mitigar la aparición de valores altos y bajos pronunciados. Esto introduce dos hiperparámetros más δ_p y δ_q, que se utilizan para modular el suavizado de P y Q^j respectivamente (líneas 14 y 22).

Una última dificultad ocurre en el cálculo del límite superior del gradiente (80), debido a que se debe de obtener analíticamente para después ser implementado con respecto a la CNN que se presenta en la sección 4.3 . A continuación, se muestra la derivación de este respectivo gradiente.

Las DNNs actualmente son entrenadas computacionalmente mediante la combinación del lenguaje de programación Python y bibliotecas especializadas en aprendizaje maquina como PyTorch o TensorFlow; el algoritmo más frecuentemente usado para entrenar a las DNNs es la propagación hacia atrás [48]. Tanto PyTorch como TensorFlow tienen incorporado en su código un motor de diferenciación (en el caso de PyTorch se llama torch.autograd [47]) que permite calcular automáticamente gradientes de cualquier grafo computacional.

Algoritmo 1: **Muestreo adaptativo para el aprendizaje profundo**

Input: $c_j = \{\{x_i, y_i\}_{i=1}^{N/k}\}_{j=i}^{k}$; epochs; $(\delta_{pbin}, \delta_q) \in (0,1)$; $\beta \in \mathbb{Z}^+$, batch size BS, No. of instances N

Initialize: $gc[1, \dots, k] \leftarrow \emptyset$; $P[p_1, p_2, \dots, p_k] \sim \mathcal{U}$; $Q^j \left[q_1^j, q_2^j, \dots, q_{|I_j|}^j\right] \sim \mathcal{U}$

$t \leftarrow 0$; $l_{(1:N/k)}^{(1:k)} \leftarrow \infty$: l es un arreglo k–dimensional para guardar los valores de perdida.

For $epoch = 1$ **to** $epochs$ **do:**

  For $b = 1$ **to** N/BS **do:**

    For $s = 1$ **to** BS **do:**

      $j \sim P$: Muestreo del contenedor j de acuerdo con (67).

      $i \sim Q^j$: Muestreo del contenedor i de acuerdo con (79).

      $l_i^j \leftarrow L(\varphi(x_i^j; \theta), y_i^j)$: Propagación hacia adelante: cálculo de la pérdida y se almacena en el arreglo l.

      d^t : Cálculo de los gradientes.

      $\hat{G}_i$: Cálculo del límite superior de los gradientes (80) y (87).

      $gc[j] \leftarrow sg[j] + \hat{G}_i$: Acumulación de gradientes del contenedor j.

      If $(t \bmod \beta) = 0$ **then:**

        $p_j \propto \sqrt{gc[j]}/N$: Recálculo de p_j de acuerdo con (67) $\forall\, j$.

        $(p_j)^t \leftarrow \delta_p (p_j)^{t-1} + (1 - \delta_p)\,(p_j)^t$: Suavizado actual de p_j usando valores previos.

        $gc[1, \dots, k] \leftarrow 0$

      End if

      $t \leftarrow t + 1$

    End for

    $d^b \leftarrow \sum_{r=1}^{BS} d^r/BS$: Cálculo del gradiente.

    Update θ^t using d^b : actualización de pesos.

    $q_i^j \propto l_i^j$: Recálculo de q_i^j de acuerdo con (79)

    $(q_i^j)^t = \delta_q (q_i^j)^{t-1} + (1 - \delta_q)\,(q_i^j)^t$: Suavizado actual de Q^j usando valores previos

  End for

End for

Return: optimized network parameters θ^*

El módulo torch.autograd consiste en construir un grafo de computación que permita rastrear y calcular desde una entrada a una salida todos los gradientes de los parámetros que constituyen a una DNN. Concretamente, torch.autograd calcula automáticamente $\frac{\partial L}{\partial W^{[l]}}$ y $\frac{\partial L}{\partial b^{[l]}}$ para cualquier $l - ésima$ capa (los parámetros $W^{[l]}$ y $b^{[l]}$ constituyen nodos hoja en el grafo de computación [48]).

Sin embargo, los diferenciales de otros nodos del grafo de computación (como podrían ser $Z^{[L]}$ o $A^{[L]}$) no se pueden obtener. Citando a la documentación [48]:
"Solo podemos obtener los gradientes para los nodos hojas del grafo de computación. Para otros nodos en el grafo, los gradientes no están disponibles."

En consecuencia, la expresión (80) se debe calcular analíticamente para que después sea implementada en el **Algoritmo 1**.
A continuación, se muestra la derivación de (80) para la CNN que se presenta en la sección 4.3

Sea la $L - esima$ capa de una CNN, donde $Z^{[L]}, A^{[L]} \in R^m$ y $f^{[L]}$ es la función SoftMax (20)

$$A^{[L]} = f^{[L]}\left(Z^{[L]}\right). \tag{81}$$

Sea $L : \left(A^{[L]}, Y\right) \subseteq R^m \rightarrow R$ la función de perdida entropía cruzada (24).
La expresión (80) es la norma del gradiente con respecto a las salidas de preactivación de la última capa en una DNN, de modo que

$$\nabla_{\theta^{[L]}} \mathrm{L}\left(\mathrm{A}^{[L]}, \mathrm{Y}\right) = \left(\mathrm{J}_{\mathrm{A}^{[L]}}\right)^{\mathrm{T}} \left(\nabla_{\mathrm{A}^{[L]}} L\right), \tag{82}$$

es el gradiente de la última capa de preactivación de una DNN.

De (24) se tiene que

$$\nabla_{\mathrm{A}^{[L]}} \mathrm{L} = \begin{bmatrix} \dfrac{\partial \mathrm{L}}{\partial \mathrm{a}_1^{[L]}} \\ \vdots \\ \dfrac{\partial \mathrm{L}}{\partial \mathrm{a}_{\mathrm{m}}^{[L]}} \end{bmatrix} = \begin{bmatrix} -\dfrac{y_1}{\mathrm{a}_1^{[L]}} \\ \vdots \\ -\dfrac{y_m}{\mathrm{a}_m^{[L]}} \end{bmatrix}. \tag{83}$$

De (20) y (3) se tiene que

$$\mathrm{J}_{\mathrm{A}^{[L]}} = \begin{pmatrix} \dfrac{\partial \mathrm{a}_1^{[L]}}{\partial z_1^{[L]}} & \cdots & \dfrac{\partial \mathrm{a}_1^{[L]}}{\partial z_m^{[L]}} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial a_m^{[L]}}{\partial z_1^{[L]}} & \cdots & \dfrac{\partial a_m^{[L]}}{\partial z_m^{[L]}} \end{pmatrix}, \tag{84}$$

donde

$$\frac{\partial \mathrm{a}_i^{[L]}}{\partial z_i^{[L]}} = \frac{\left(\sum_{k=1}^{m} e^{z_k^{[L]}}\right) e^{z_i^{[L]}}}{\left(\sum_{k=1}^{m} e^{z_k^{[L]}}\right)^2} - \frac{e^{z_i^{[L]}} e^{z_i^{[L]}}}{\left(\sum_{k=1}^{m} e^{z_k^{[L]}}\right)^2} = \mathrm{a}_i^{[L]}\left(1 - \mathrm{a}_i^{[L]}\right),$$

$$\frac{\partial \mathrm{a}_i^{[L]}}{\partial z_j^{[L]}} = -\frac{e^{z_i^{[L]}} e^{z_j^{[L]}}}{\left(\sum_{k=1}^{m} e^{z_k^{[L]}}\right)^2} = -\mathrm{a}_i^{[L]} \mathrm{a}_j^{[L]}, \tag{85}$$

$$\forall i, j = 1, \ldots, m \wedge i \neq j.$$

Entonces

$$\nabla_{\theta^{[L]}} \mathrm{L}\left(\mathrm{A}^{[L]}, \mathrm{Y}\right) = \begin{pmatrix} \mathrm{a}_1^{[L]}\left(1 - \mathrm{a}_1^{[L]}\right) & \cdots & -\mathrm{a}_1^{[L]} \mathrm{a}_m^{[L]} \\ \vdots & \ddots & \vdots \\ -\mathrm{a}_1^{[L]} \mathrm{a}_m^{[L]} & \cdots & \mathrm{a}_m^{[L]}\left(1 - \mathrm{a}_m^{[L]}\right) \end{pmatrix}^T \begin{bmatrix} -\frac{y_1}{\mathrm{a}_1^{\mathrm{L}}} \\ \vdots \\ -\frac{y_m}{\mathrm{a}_m^{\mathrm{L}}} \end{bmatrix}$$

$$= \begin{pmatrix} \left(1 - \mathrm{a}_1^{[L]}\right)(-y_1) & \ldots + \ldots & y_m \mathrm{a}_1^{[L]} \\ \vdots & \vdots & \vdots \\ y_1 \mathrm{a}_m^{[L]} & \ldots + \ldots & (-y_m)\left(1 - \mathrm{a}_m^{[L]}\right) \end{pmatrix}. \tag{86}$$

Los vectores-etiqueta Y corresponden a vectores “one-hot”. Esto quiere decir que son vectores de la misma dimensión de la salida $L - esima$, donde la clase se codifica con un uno en la posición que le corresponde al índice del vector (por ejemplo, la clase 2 le correspondería el vector $[0 \quad 0 \quad 1 \quad \ldots \quad 0]^T$, si se empieza a contar desde 0 al índice vector). Entonces, la ecuación (87) queda como

$$\nabla_{\theta^{[L]}} \mathrm{L}\left(\mathrm{A}^{[L]}, \mathrm{Y}\right) = \mathrm{A}^{[L]} - \mathrm{Y}\,. \tag{87}$$

El gradiente de (82) es simplemente una diferencia elemento a elemento del vector $\mathrm{A}^{[L]}$ y su vector-etiqueta.

4.3 Metodología y Resultados experimentales

Se investigó el muestreo adaptativo propuesto (**Algoritmo 1**) en el conjunto de datos estándar MNIST de dígitos escritos a mano (LeCun et al. [44]) con 60.000 puntos de datos de

entrenamiento. Para el proceso de clasificación se implementó una CNN (tabla 1) con dos capas de convolución y max-pooling, dos capas completamente conectadas (FC) y activación ReLU para todas las capas excepto la última (activación SoftMax, como parte de la función de pérdida). Se empleó la normalización por lotes en las capas convolucionales, se usó un dropout de 0.5 en la primera capa FC y se hizo una normalización a los datos. La función de pérdida utilizada fue la entropía cruzada (24) y como función de activación de la última capa, la función SoftMax (20).

Todos los experimentos se implementaron utilizando la biblioteca de Python PyTorch [47] y se ejecutaron en GPU a través de los notebooks de Google Colab [50]. Se evaluó la propuesta en combinación con dos optimizadores diferentes: Adadelta [18] Adam [20] (los hiperparámetros de dichos optimizadores se configuraron de la misma forma en que lo hicieron Loshchilov & Hutter [37]).

Una de las ideas principales en el trabajo de Gopal [36] es que se precisa de una previa partición de los datos de entrenamiento, esta partición se debe basar en explotar información colateral de los datos. Gopal [36] instruye que la información colateral de los datos puede ser información directa o indirecta de los mismo, siempre y cuando se pueda garantizar una separación entre cada subconjunto de la partición. La información directa de los datos que es más instintiva y fácil de conseguir es la etiqueta-clase (particionar los datos con respecto a su etiqueta garantiza una partición directa de los datos [36]). De modo que, para todos los experimentos, se estableció la etiqueta-clase como información colateral de los datos para realizar la partición.
En teoría, la selección de la información secundaria de las instancias podría ser cualquier información siempre que sea lo suficientemente informativa como para proporcionar separabilidad entre los contenedores (que normalmente correspondería a las clases de un problema de clasificación). Por ejemplo, Joseph et al. [41] han explorado algunas medidas de cuán informativas son las instancias; una de ellas es la entropía del modelo actual en la iteración de entrenamiento (60).

Tabla 1 : Arquitectura de la red convolucional

Cantidad	Capa	Canales de entrada	Canales de salida	Función de activación
1	Imagen	-	-	-
1	Conv 5× 5	3	32	-
1	Batch Norm	32	32	ReLU
1	Max Pool 2x2	32	32	-
1	Conv 5× 5	32	32	-
1	Batch Norm	32	32	ReLU
1	Max Pool	32	32	-
1	Aplanamiento	32	512	-
1	FC	512	256	ReLU
1	FC	256	10	SoftMax

4.3.1 Selección de hiperparámetros

Como se describió en la sección 4.2, el **Algoritmo 1** tiene 3 hiperparámetros para configurar : el hiperparámetro que controla la actualización de la distribución de los contendores β, y los hiperparámetros de suavizados δ_p y δ_q. El hiperparámetro β se estableció a 10, debido a que Gopal [36] comenta que es un valor adecuado para asegurar una acumulación de gradientes suficiente sin saturar ningún contenedor; los hiperparámetros de suavizado se ajustaron mediante búsqueda en cuadrícula. Previamente, antes de hacer esta búsqueda en cuadrícula, se tuvo la intuición de que los valores óptimos de suavizado podrían ser 0.9 y 0.1 para δ_p y δ_q respectivamente. Debido a que la cardinalidad de P es mucho menor que la de Q^j, por lo tanto, se podría requerir un mayor suavizado en δ_p que en δ_q. De modo que, se realizó una primera búsqueda en cuadrícula usando el optimizador Adadelta [18], con un tamaño de lote 64, y con los siguientes valores: $\delta_p \in [0.3,0.6,0.9]$ y $\delta_q \in [0.1,0.4,0.7]$. En la Fig. 13 se muestran los resultados. Después se realizó otra búsqueda en cuadricula con respecto a la Fig. 13, pero esta vez con los siguientes valores $\delta_p \in [0.902, 0.91,0.93]$ y $\delta_q \in [0.090, 0.089,0.059]$. En la Fig. 14 se muestran los resultados.

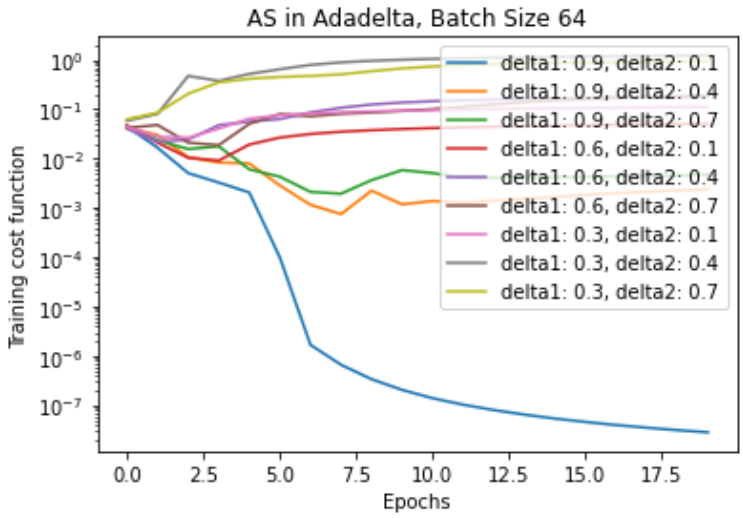


Figura 13. Busqueda en cuadrícula de los hiperparámetros δ_p y δ_q para Adadelta con tamaño de lote 64. En el rango $(0.1, 0.9)$ y $(0.1, 0.9)$ respectivamente

Como se observa en la Fig. 13, los hiperparámetros óptimos para esa búsqueda son $\delta_p = 0.9$ y $\delta_q = 0.1$; en la Fig. 14 se observa que no hay una gran variación con respecto a la función de costo para los distintos valores de δ_p y δ_q. De modo que, con base en la evidencia, se considera solo la búsqueda en cuadricula para los valores respectivos de suavizado de la Fig. 13. En las Figs. 15 y 16 se muestran las búsquedas en cuadricula para tamaños de lote de 256 y 1024.

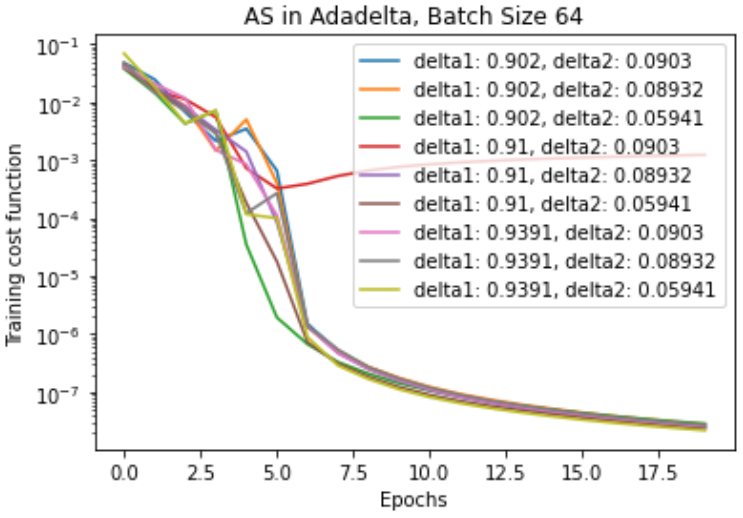


Figura 14. Busqueda en cuadrícula de los hiperparámetros δ_p y δ_q para Adadelta con tamaño de lote 64. En el rango $(0.9, 1)$ y $(0, 0.1)$ respectivamente.

En las Figs. 15 y 16 se observa que hay más valores adecuados en comparación con la Fig. 13, sin embargo, los valores óptimos de $\delta_p = 0.9$ y $\delta_q = 0.1$ son comunes en la búsqueda en cuadrícula en los tres lotes. De modo que, dichos valores se toman como óptimos para la experimentación.

Después, se procedió a hacer el mismo análisis de las Figs. 13, 15 y 16 pero para el optimizador Adam [20]. En las Figs. 17, 18 y 19 se muestran los resultados

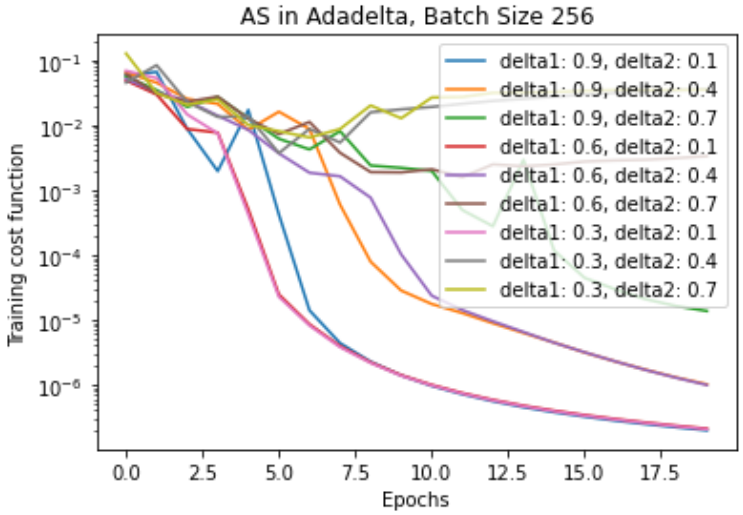


Figura 15. Busqueda en cuadrícula de los hiperparámetros δ_p y δ_q para Adadelta con tamaño de lote 256. En el rango $(0.1, 0.9)$ y $(0.1, 0.9)$ respectivamente.

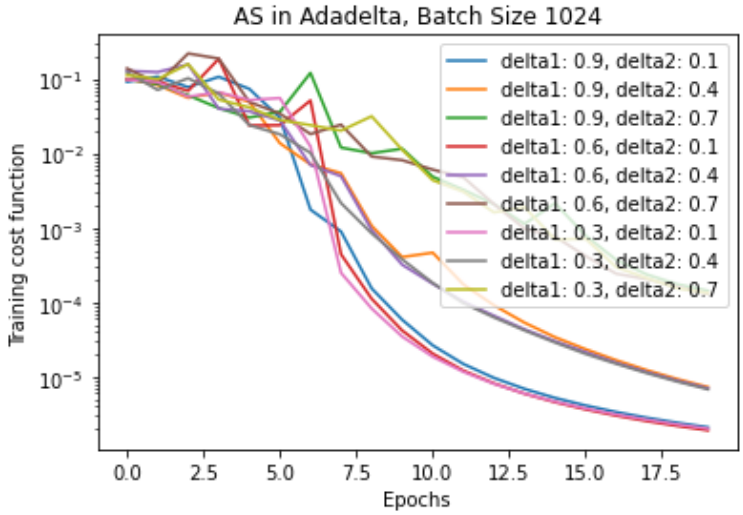


Figura 16. Busqueda en cuadrícula de los hiperparámetros δ_p y δ_q para Adadelta con tamaño de lote 1024. En el rango $(0.1, 0.9)$ y $(0.1, 0.9)$ respectivamente.

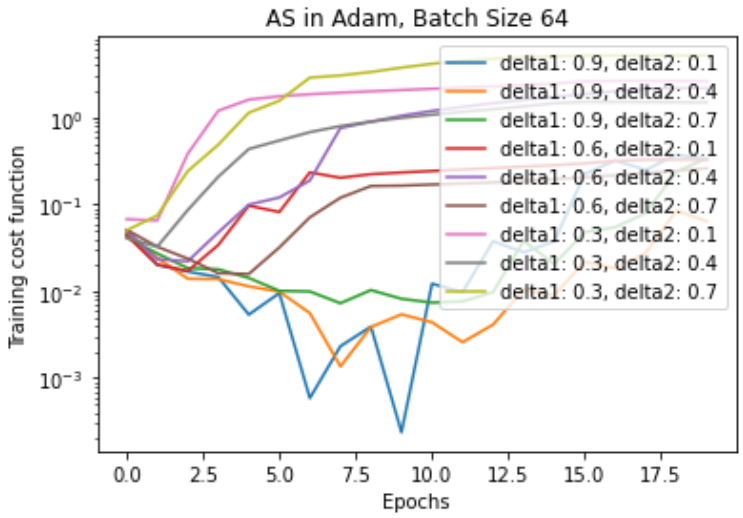


Figura 17. Busqueda en cuadrícula de los hiperparámetros δ_p y δ_q para Adam con tamaño de lote 64. En el rango $(0.1, 0.9)$ y $(0.1, 0.9)$ respectivamente.

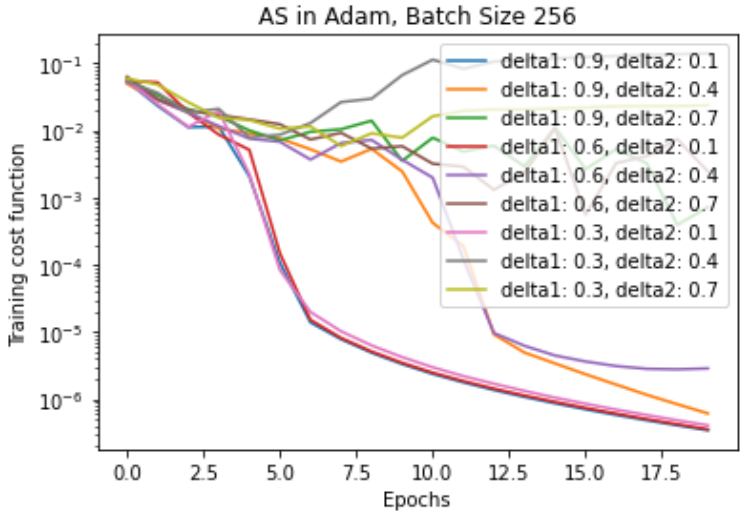


Figura 18. Busqueda en cuadrícula de los hiperparámetros δ_p y δ_q para Adam con tamaño de lote 256. En el rango $(0.1, 0.9)$ y $(0.1, 0.9)$ respectivamente.

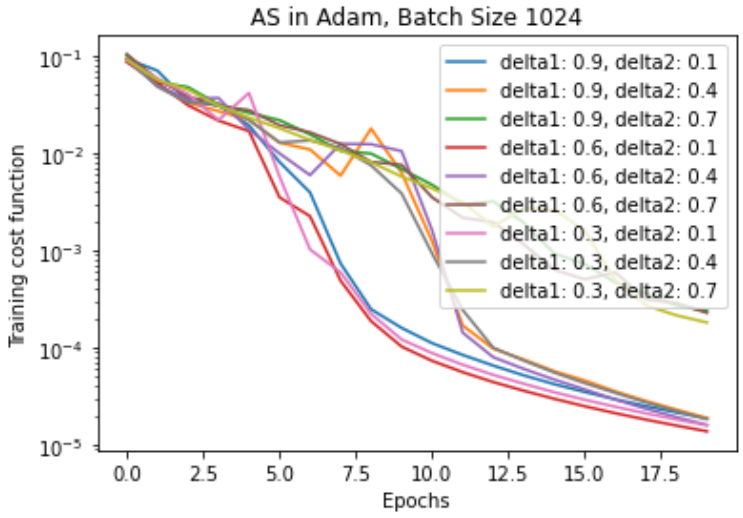


Figura 19. Busqueda en cuadrícula de los hiperparámetros δ_p y δ_q para Adam con tamaño de lote 1024. En el rango $(0.1, 0.9)$ y $(0.1, 0.9)$ respectivamente.

El comportamiento que se observa en la Fig. 17 es completamente distinto del que se observa en la Fig. 13. Sin embargo, las Figs. 15, 16, 18 y 19 tienen un comportamiento similar en el sentido que hay varios valores adecuados de δ_p y δ_q, y donde coincide con los valores adecuados $\delta_p = 0.9$ y $\delta_q = 0.1$.

En la Fig. 17, para la mayoría de los valores de suavizado, la función de costo diverge, y para los valores restantes la función de costo se mantiene constante y oscilante en un valor. En la sección 4.4 se discute la razón de este comportamiento.

No se encontraron valores adecuados de suavizado para el tamaño de lote 64 usando el optimizador Adam. Sin embargo, para comparar el rendimiento de Adam y Adadelta, se tomaron de igual forma los valores $\delta_p = 0.9$ y $\delta_q = 0.1$ para la etapa de experimentación. Seguidamente, se presenta los resultados experimentales.

4.3.2 Comparación experimental contra el Estado del Arte

El método presentado en este trabajo se conoce como Muestreo Adaptativo (AS) y se compara con dos alternativas: el algoritmo de entrenamiento estándar, que muestrea instancias de datos bajo una distribución uniforme conocida como Default Random, y la propuesta de Loshchilov & Hutter [37] llamada selección de lotes en línea (OBS). Este último algoritmo requiere dos hiperparámetros adicionales y se establecen en los valores óptimos informados en [37]: $r = 0.5, s = 10^8$ desde el principio hasta el final del entrenamiento

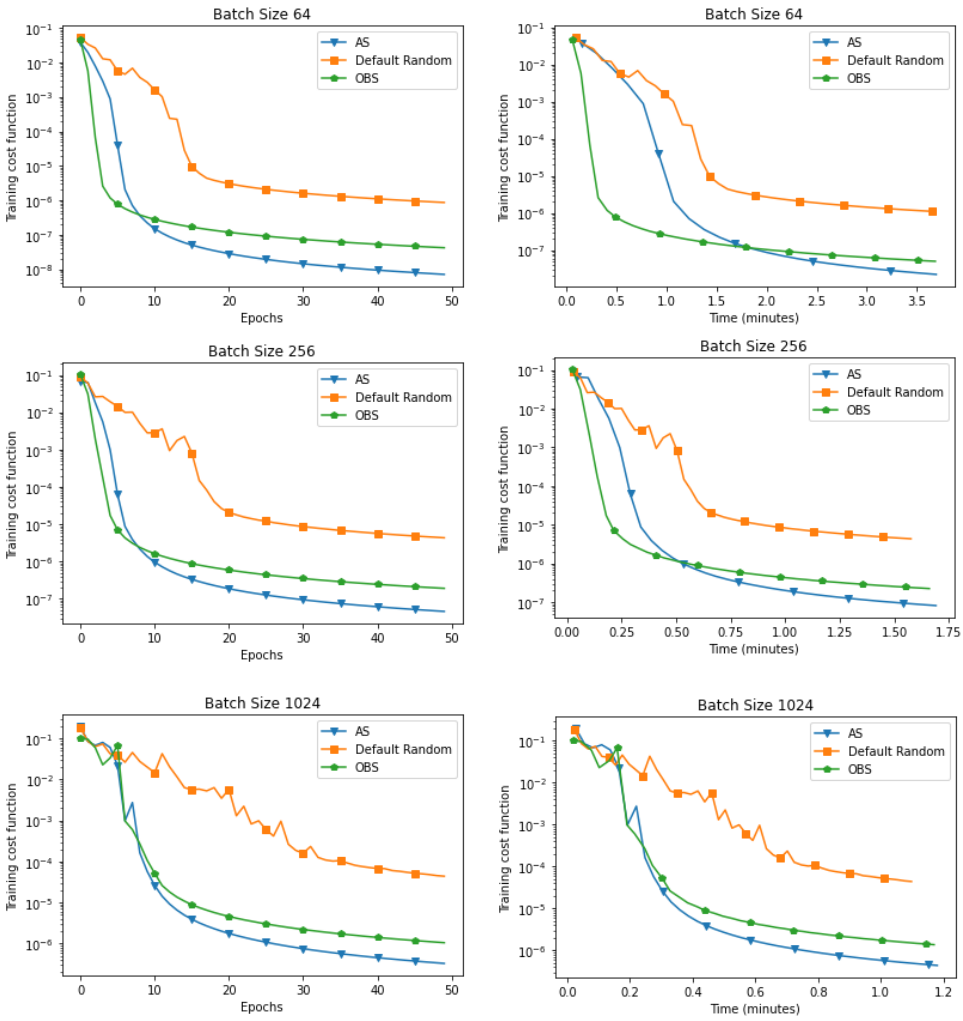


Figura 20. Comparación de la propuesta de muestreo adaptativo (AS) para DL frente a SGD con muestreo uniforme (aleatorio predeterminado) y selección de lotes en línea (OBS). Curvas de convergencia de Adadelta (tasa de aprendizaje = 1.0, rho = 0.9), mientras se entrena una CNN para la clasificación del conjunto de datos MNIST. Las curvas que se muestran son el promedio de los valores promedio de la función de costo en quince corridas.

A. Resultados de convergencia de la función de costo

Los experimentos consistieron en entrenar la CNN de la tabla 1 durante 50 épocas y reportar la función de costo promedio en 15 corridas; se repitieron los experimentos con los siguientes tamaños de lote : 64, 256 y 1024 instancias de entrenamiento. Los resultados se muestran en las Figs. 20 (Adadelta) y 21 (Adam).

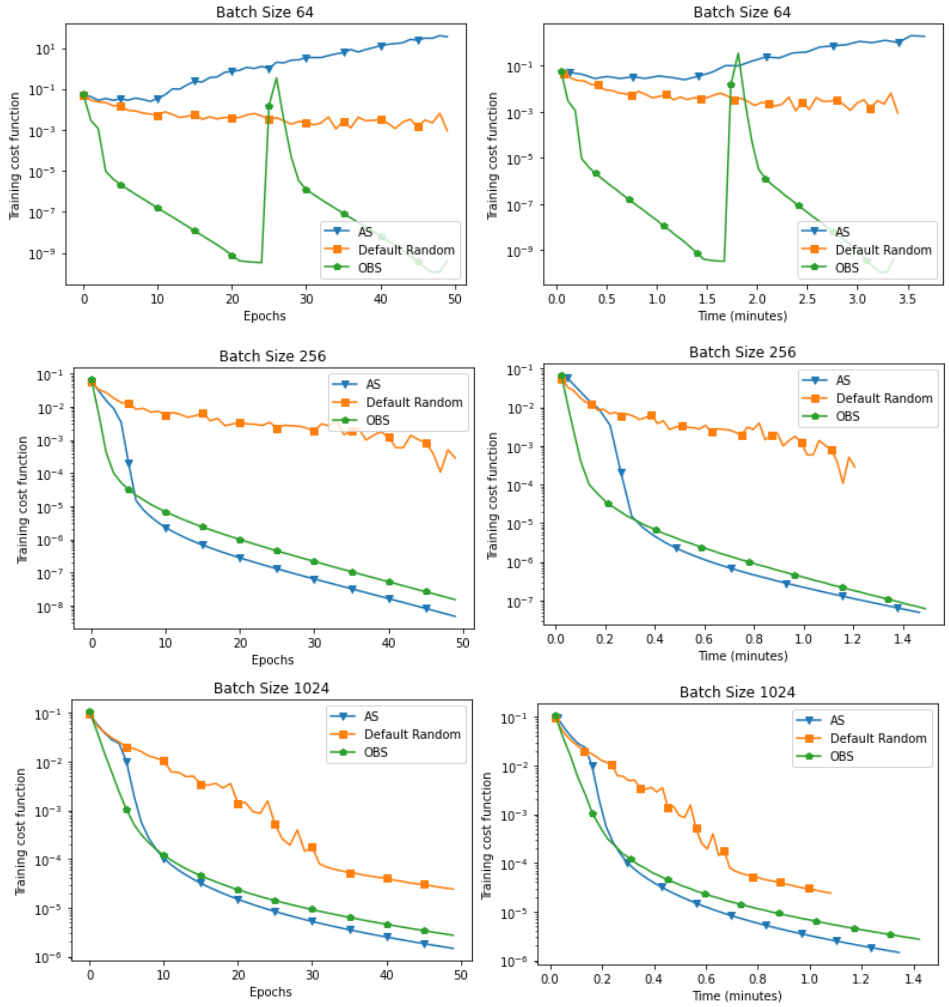


Figura 21. Comparación de la propuesta de muestreo adaptativo (AS) para DL frente a SGD con muestreo uniforme (aleatorio predeterminado) y selección de lotes en línea (OBS). Curvas de convergencia de Adam (tasa de aprendizaje = 0.001, $beta_1 = 0.9$, $beta_2 = 0.999$), mientras se entrena una CNN para la clasificación del conjunto de datos MNIST. Las curvas que se muestran son el promedio de los valores promedio de la función de costo en quince corridas.

B. Resultados en la tarea de clasificación de imágenes

El objetivo de AS es acelerar el entrenamiento de un modelo de DL sin afectar su rendimiento, esto se traduce en mejorar la convergencia de la función de costo en datos de entrenamiento. Sin embargo, para observar si AS alcanza a generalizar datos de prueba en comparación con OBS y Default Random , se tomó el mejor resultado de los experimentos de las Figs. 20 y 21 y se obtuvieron las gráficas de precisión en datos de prueba (Figs. 22 y 23 respectivamente). La precisión (Accuracy en inglés) se define como la razón entre las predicciones correctas y las predicciones totales :

$$Precisión = \frac{\#\ Predicciones\ Correctas}{\#\ Predicciones\ Totales}. \tag{88}$$

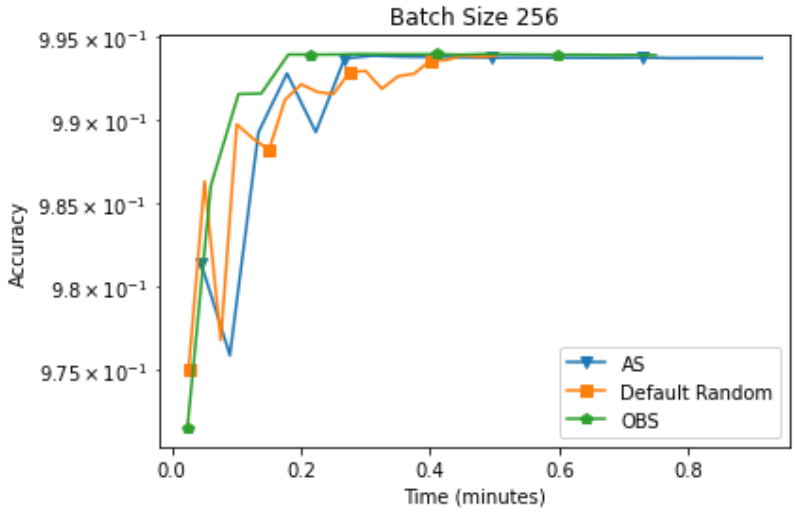


Figura 22. Comparación de nuestra propuesta de muestreo adaptativo (AS) frente al muestreo uniforme (Default Random) y OBS. Curvas de precisión de Adadelta (tasa de aprendizaje = 1.0, rho = 0.9), mientras se entrena un CNN para la clasificación del conjunto de datos MNIST. Las curvas que se muestran son el promedio del Accuracy en quince corridas y 10 000 de datos.

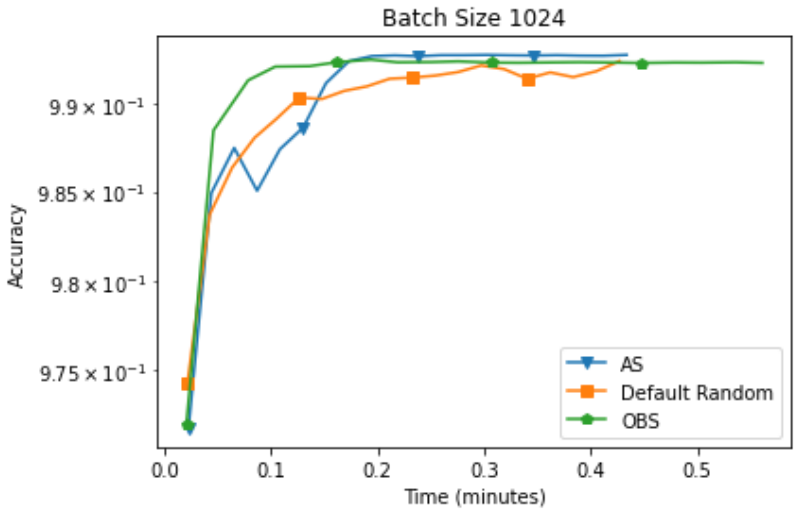


Figura 23. Comparación de nuestra propuesta de muestreo adaptativo (AS) frente al muestreo uniforme (Default Random) y OBS. Curvas de precisión de Adam (tasa de aprendizaje = 0.001, $beta_1 = 0.9$, $beta_2 = 0.999$), mientras se entrena un CNN para la clasificación del conjunto de datos MNIST. Las curvas que se muestran son el promedio del Accuracy en quince corridas y 10 000 datos.

4.4 Observaciones y Discusiones

A continuación, se discuten los resultados de la sección 4.3. También, se dan algunas conjeturas sobre el comportamiento de AS.

4.4.1 Comportamiento de AS con los optimizadores Adam y Adadelta

En la Fig. 20 se puede observar que la función de pérdida inicialmente disminuye más rápidamente con AS y OBS que con el muestreo aleatorio por defecto, lo que implica una convergencia más rápida de los parámetros de la CNN hacia sus valores óptimos. La convergencia más significativa ocurre dentro de las primeras 15 épocas (20 por defecto aleatorio); después de eso, las tasas de convergencia de los tres métodos se ralentizan, pero en ese momento tanto AS como OBS han alcanzado al menos un orden de magnitud por debajo del valor de costo alcanzado por defecto aleatorio.

En la fig. 21 se observa que el valor de la función de costo con AS diverge para un tamaño de lote de 64 e informa una convergencia más rápida para los tamaños de lote de 256 y 1024, mostrando el mismo comportamiento que en la fig. 20 con el optimizador Adadelta.

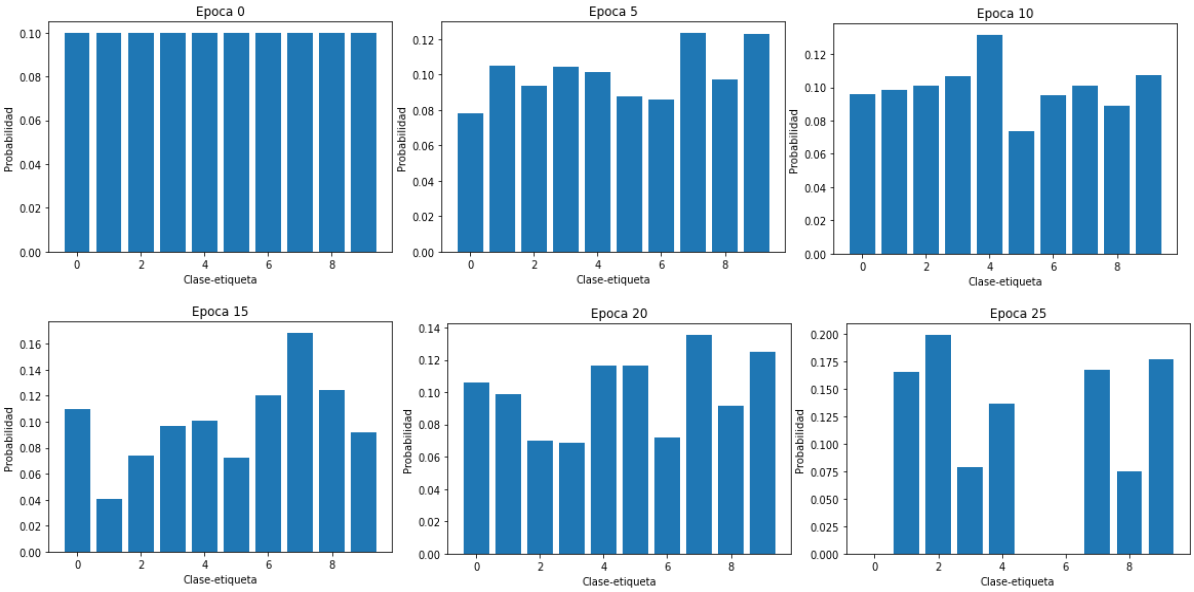


Figura 24. Seguimiento de la distribucion de probabilidad *P* (al inicio de una época) en 6 epocas sobre el entrenamiento de la CNN con un tamaño de lote 64 usando el optimizador Adam.

Para entender la causa de la divergencia y convergencia de AS usando el optimizador Adam con los distintos tamaños de lotes se presenta la siguiente discusión:

En la Fig. 17 se observa que la función de costo se estanca e incluso diverge. Este fenómeno ocurre principalmente porque algunos valores de la distribución de probabilidad *P* son cero; esto significa que en alguna parte del entrenamiento hay contenedores que no se vuelven a muestrear y eso ocasiona que a la CNN no se le vuelvan a exponer instancias de contenedores de distintas clases. En la Fig. 24 se muestra un seguimiento de la distribución de *P* en 6 épocas de entrenamiento para un tamaño de lote 64. En dicha figura se observa que en la época cero la distribución de *P* es una distribución uniforme (como se indica en el **Algoritmo 1**); en las épocas 5, 10,15 y 20 la distribución *P* se muestra dinámica sin tener ningún valor a cero. Sin embargo, en la época 25 se tienen tres contenedores en cero (el contenedor 0, 5 y 6). Esto indica que la divergencia de la función de costo de AS para el optimizador Adam con un tamaño de lote 64 se

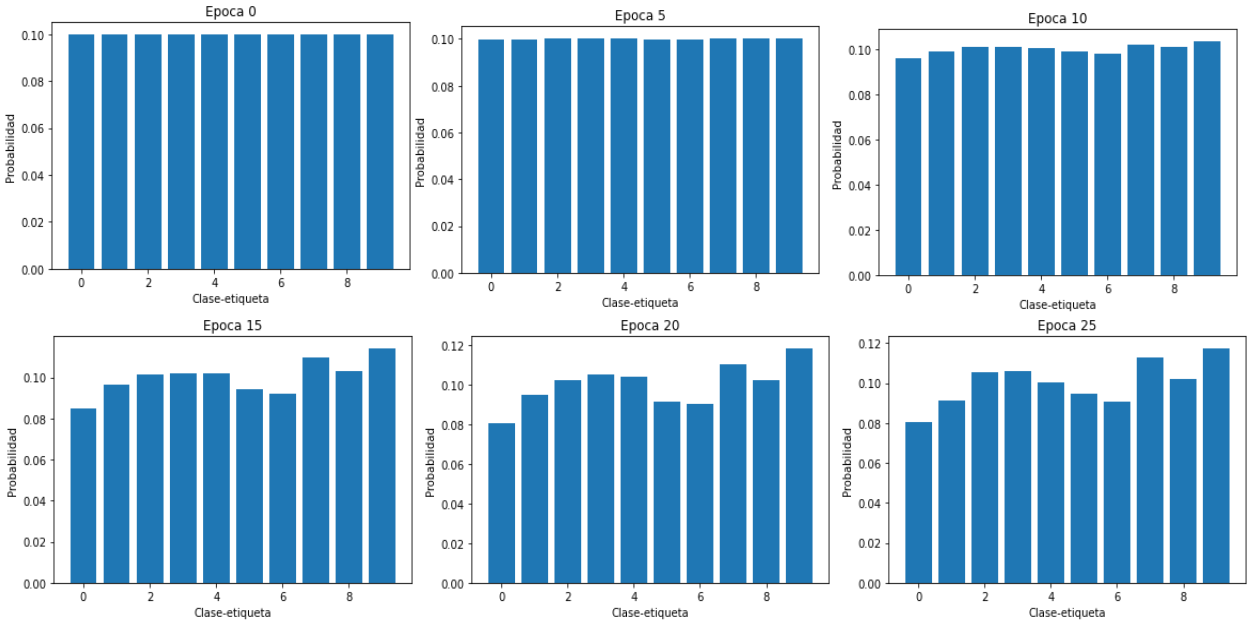


Figura 25. Seguimiento de la distribucion de probabilidad *P* (al inicio de una época) en 6 epocas sobre el entrenamiento de la CNN con un tamaño de lote de 1024 usando el optimizador Adam

debe a que hay una saturación de las normas de los gradientes; en consecuencia, conforme el entrenamiento avanza, la distribución *P* tiende a valores muy altos en algunos contenedores y valores muy bajos en otros (incluso cero). Sin embargo, el entrenamiento de la CNN con un tamaño de lote 1024 usando el optimizador Adam ocurre lo opuesto, la función de costo es suave y convergente. En la Fig. 25 se hace lo análogo con respecto a la Fig. 24, pero en este caso con un tamaño de lote 1024. En la Fig. 25 se observa que la distribución *P* no tiende a valores tan altos ni tan bajos como en la Fig.24.

La Fig. 25 índica que la convergencia de la función de costo de AS para el optimizador Adam con un tamaño de lote de 1024 se debe principalmente al muestreo de las instancias de cada contenedor y no al muestreo de los contenedores.

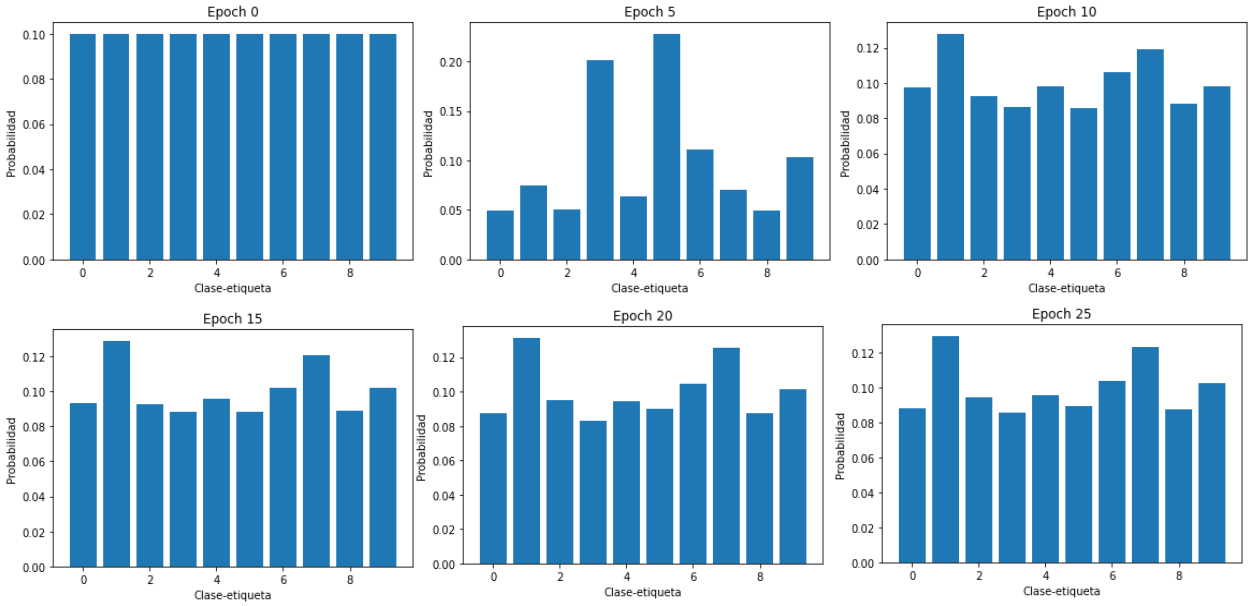


Figura 26. Seguimiento de la distribución de probabilidad P (al inicio de una epoca) en 6 epocas sobre el entrenamiento de la CNN con un tamaño de lote 64 usando el optimizador Adadelta.

Del mismo modo que en Adam, en la Fig. 26 se muestra un seguimiento de la distribución de probabilidad P cuando se entrena la CNN usando Adadelta con un tamaño de lote 64 y los hiperparámetros $\delta_p = 0.9$ y $\delta_q = 0.1$. En esta distribución de probabilidad, se observa un comportamiento distinto que en su distribución de probabilidad homologa (Fig. 24), la distribución P no se satura en unos pocos valores de clases.

Es evidente que la convergencia de AS con Adam depende del tamaño de lote, sin embargo, con Adadelta esta variable parece no afectar su comportamiento de una manera tan significativa. Se conjetura que la explicación de este comportamiento se encuentra en las ecuaciones de actualización tanto de Adam como de Adadelta. Adam es un algoritmo que aplica el promedio ponderado exponencial tanto al gradiente como a su cuadrado [19]. Por otro lado, Adadelta es un algoritmo que adapta la tasa de aprendizaje, y en lugar de acumular gradientes cuadrados pasados, restringe la ventana de gradientes acumulados a un tamaño fijo [30]. En esta diferencia puede residir el distinto comportamiento de la distribución de probabilidad P sobre los dos optimizadores.

La distribución P se construye con una acumulación de la norma al cuadro de los gradientes de la última capa de una DNN, sin embargo, los gradientes de la expresión (87) no son los mismos que se utilizan para actualizar los pesos usando Adam o Adadelta. Cuando se usan estos dos optimizadores, los gradientes pasan por una serie de ecuaciones de actualización para después ser utilizados como regla de actualización de pesos (ecuaciones 46 a 51 y 52 a 56 respectivamente). Debido a esto, cuando se construye la distribución de probabilidad P (67), los gradientes que componen dicha distribución no son los mismo con los que se actualizan los pesos.

Razonando de esta manera, se puede argumentar que cuando AS usa Adam como optimizador, los gradientes tienden a saturarse para algunas clases, y hay divergencia debido a que los gradientes de Adam traen como regla de actualización momentum y reducción de la variabilidad. Por lo tanto, esto ocasiona que los gradientes tiendan a estar en unos cuantos contenedores debido a la carga de los gradientes pasados, y este fenómeno se va más afectado si el tamaño de lote es pequeño, debido a que hay más posibilidades de que los gradientes se saturen en unos cuantos contenedores por la poca variabilidad de las instancias de las cuales se obtiene el gradiente.

Por otro lado, Adadelta no acumula gradientes pasados en su gradiente de actualización de pesos (ecuaciones 54 y 56), más bien adapta su tasa de aprendizaje (la cual depende de acumulación de gradientes pasados también). De tal forma, la distribución de probabilidad P es más estable con este optimizador debido a que los mismos gradientes que componen P son aquellos que actualizan los pesos.

Finalmente, se encontró que AS es muy sensible a los valores de los parámetros de suavizado para tamaños de lote pequeño (Fig. 13), para tamaño de lote más grande hay una mayor flexibilidad (Fig. 15 y Fig. 16). Para la distribución de probabilidad P de los contenedores, se encontró que el parámetro de suavizado óptimo δ_p es alto en comparación con el parámetro de suavizado δ_q de las distribuciones de probabilidad Q^j de las instancias (Fig. 13 y Fig.14). Esto se puede explicar debido a que la cardinalidad de P es mucho menor que la cardinalidad de cada Q^j. Por ejemplo, en el conjunto de datos MNIST $|P| = 10$ porque hay 10 clases en el conjunto de datos, mientras que $\left|Q^j\right| = N/10$ ($N = 60{,}000$) para cada una de las j-ésimas distribuciones. En consecuencia, la distribución P tiende a volverse inestable debido a la acumulación de muy pocos

valores para cada p_j , y se requiere un valor alto del parámetro δ_p para compensar esta inestabilidad. Por otro lado, para cada Q^j ocurre lo contrario y el parámetro de suavizado δ_q requerido es menor.

4.4.2 Comparativa con base en número de iteraciones

Comparando AS vs OBS en términos de épocas de entrenamiento (columna izquierda en la Fig. 20) se puede ver que OBS es más rápido durante las primeras cinco a diez iteraciones, pero luego se ralentiza de forma suave. Por el contrario, AS no se desacelera tan significativamente como OBS, por lo que en la décima época AS ha logrado superar a OBS en los dos escenarios de prueba y luego mantiene esta ventaja durante el resto del entrenamiento.

Una explicación de este comportamiento se encuentra en la relación entre los valores de pérdida y las normas de gradiente. Katharopoulos & Fleuret demostraron que los valores de pérdida pequeños implican gradientes pequeños [39] y también demostraron que los valores de pérdida relativamente grandes no están bien correlacionados con las normas de gradiente [32] (en la ecuación 59 se observa la relación entre la norma del gradiente y la función de costo). Por lo tanto, las estrategias de muestreo como AS y OBS son más ventajosas al comienzo del entrenamiento, cuando las pérdidas exhiben una gran variación y los métodos pueden ser altamente selectivos para minimizar la variación del gradiente. Cuando los valores de pérdida se hacen más pequeños a medida que avanza el entrenamiento, la correlación entre estos y las normas de gradiente aumenta, pero la ventaja de los métodos se pierde, ya que la varianza disminuye naturalmente.

4.4.3 Comparativa con base en tiempo de cómputo

En términos del tiempo de cálculo (columna derecha en la fig. 20), se puede observar una diferencia entre los diferentes tamaños de lote probados: con un tamaño de lote de 64 instancias, OBS y default random son claramente más rápidos que AS, pero a medida que el tamaño del lote aumenta a 256 y 1024 instancias esta ventaja se reduce drásticamente de uno a dos órdenes de magnitud, logrando AS batir por un orden de magnitud a OBS y por 2 órdenes de magnitud a Default Random. Se conjetura que AS funciona bien con lotes de mayor tamaño porque a medida

que se dispone de más información para cada uno de los contenedores, la distribución correspondiente se vuelve más estable (como se contrasta en las Figs. 24 y 25).

Cabe señalar que, independientemente de las diferencias relativas entre los métodos de muestreo, el entrenamiento requiere más tiempo de cálculo con lotes de menor tamaño. Sin embargo, en los experimentos el aumento no es muy significativo; para un tamaño de lote 64 cada experimento corrió en 2.5 minutos, y para los experimentos con tamaños de lote 256 y 1024 corrieron en 1.5 minutos. En la literatura se ha sugerido que aumentar el tamaño del lote puede reducir el número de actualizaciones de parámetros requeridas durante el entrenamiento, reduciendo consecuentemente el tiempo de entrenamiento [46], por lo que este punto se considera como un tema abierto que debería explorarse más a fondo.

4.4.4 Análisis de convergencia en función de la varianza de los gradientes

Como se expuso en la sección 4.1, las distribuciones de probabilidad P y Q^j son distribuciones de muestreo cuyo objetivo es muestrear instancias para minimizar la varianza del gradiente de la función de pérdida y así acelerar la función de costo definida sobre una DNN. Dicho de otra manera, la minimización de la varianza de los gradientes debería producir una aceleración de la función de costo.

A continuación, se muestra la relación entre la varianza de los gradientes y la convergencia de la función de costo. En las Fig. 27 y 28 se observa que el comportamiento de la varianza es congruente con la función de costo de las Figs. 20 y 21 respectivamente. La reducción de la varianza es máxima las primeras 10 épocas y después decrece asintóticamente. También se observa de la Fig. 28 que para el tamaño de lote 64 no se logra una disminución de varianza y se estanca oscilando alrededor de un valor. Igualmente, de Fig. 27 y Fig. 20 se observa que a mayor disminución de varianza la convergencia de la función de costo se acelera.

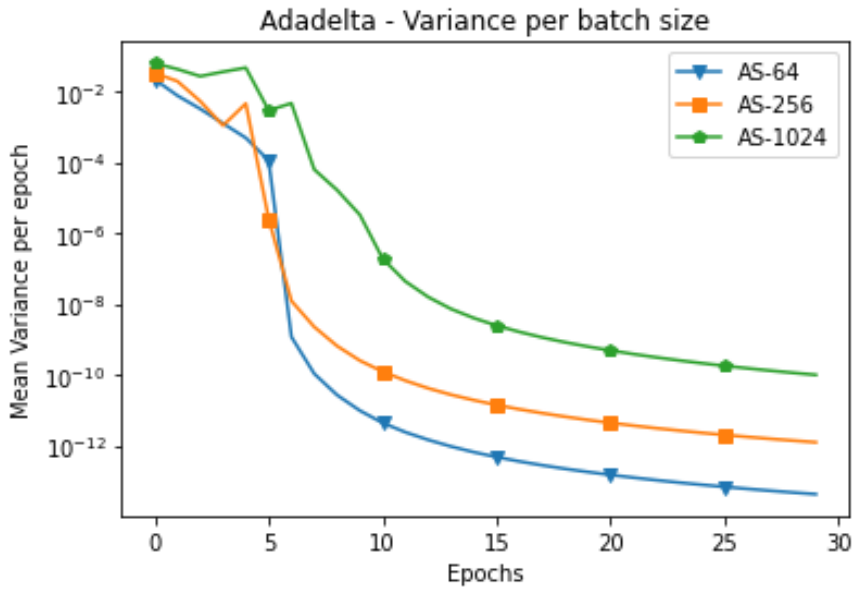


Figura 27. Promedio de la estimación de la varianza de los gradientes de (87) por época. Promedio de 15 corridas.

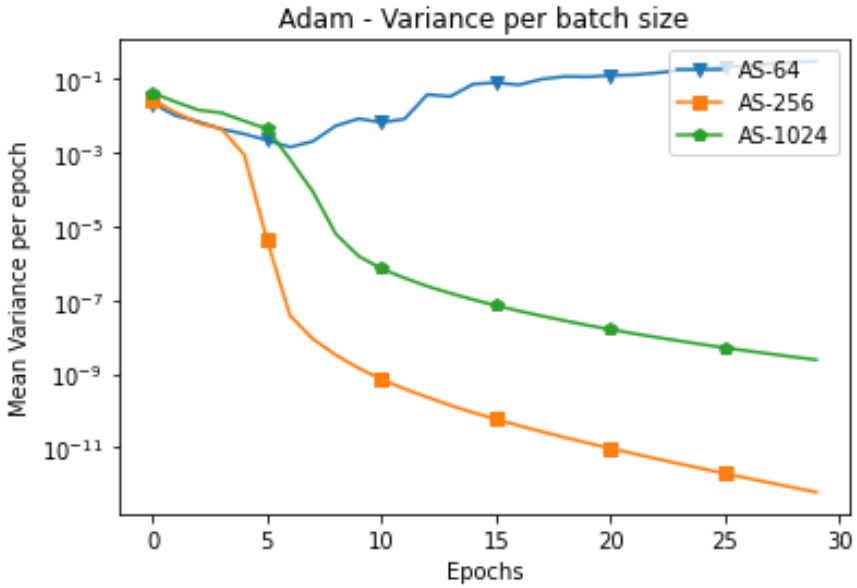


Figura 28. Promedio de la estimacion de la varianza de los gradientes de (87) por epoca. Pormedio de 15 corridas.

4.4.5 Comportamiento de AS con los optimizadores Adam y Adadelta en datos de prueba

En las curvas de precisión, entrenando la CNN con un tamaño de lote 256 y usando el optimizador Adadelta (Figura 22), las tres estrategias de muestreo demuestran que en 0.3 minutos de entrenamiento alcanzan una precisión de entre 0.90 a 0.95. Por otro lado, en las curvas de precisión, entrenando la CNN con un tamaño de lote 1024 y usando el optimizador Adam (Figura 23), en el minuto 0.2 de entrenamiento, AS, OBS y default random alcanzan una precisión aproximada de 0.999, 0.999 y 0.992 respectivamente.

Se realizó una prueba de hipótesis t-student para la diferencia de medias con varianzas iguales, definiendo las siguientes hipótesis

$$H_{01}: \mu_{AS} - \mu_{OBS} = 0, \qquad (89)$$
$$H_{a1}: \mu_{AS} - \mu_{OBS} \neq 0,$$

$$H_{02}: \mu_{AS} - \mu_{Default\,random} = 0, \qquad (90)$$
$$H_{a2}: \mu_{AS} - \mu_{Default\,random} \neq 0.$$

Para los optimizadores Adadelta y Adam con respecto a los 15 últimos valores registrados de las Figuras 22 y 23, y con un nivel de confianza de 0.95.

Tabla 2: Resultados de prueba de hipótesis de medias de los últimos 15 valores registrados de las Figuras 22 y 23

Prueba de Hipótesis – ecuación	Optimizador	P – valor	Nivel de confianza
(90)	Adadelta	0.6803	0.95
(89)	Adadelta	0.05557	0.95
(90)	Adam	0.07226	0.95
(89)	Adam	0.03716	0.95

Se observa en la tabla 2 que para los datos de prueba y el optimizador Adadelta, se acepta la hipótesis nula, y la media de AS es significativamente igual que la media de OBS y default

random. Del mismo modo, para los datos de prueba y el optimizador Adam, se acepta la hipótesis nula, y la media de AS es significativamente igual que la media de OBS y default random.

Con base en el p valor de la tabla 2, se argumenta que usando el optimizador Adadelta, con un tamaño de lote 256 y diez mil datos de prueba, existe un comportamiento similar en la capacidad de generalización tanto para AS como para OBS y default random. Del mismo modo, se argumenta que usando el optimizador Adam, con un tamaño de lote 1024 y diez mil datos de prueba, AS tiene un comportamiento similar de generalización que OBS y default random.

4.4.6 Análisis/Resumen cuantitativo de los resultados de convergencia de entrenamiento

Como se describió en la sección 4.4.1, la aceleración de la convergencia de la función de costo más significativa ocurre dentro de las primeras 15 épocas usando tanto el optimizador Adadelta y Adam y con los tres tamaños de lotes. De la tal forma, en las tablas 3 y 4 se muestra la razón de la función costo (con respecto a las Figuras 21 y 22) de AS y OBS y de AS y default random en la época 15 para el optimizador Adadelta y Adam respectivamente.

Tabla 3 : Resumen cuantitativo de convergencia de AS con Adadelta

AS con Adadelta					
	Comparativa en convergencia		Comparativa en tiempo de convergencia en la 15.ª época (min.)		
Tamaño de lote	AS / Default	AS / OBS	AS 1.4	OBS	Default
64	3.8E-5	0.78	1.47	1.01	0.98
256	9.02E-5	0.2	0.66	0.51	0.29
1024	0.83E-3	0.92	0.41	0.42	0.33

Tabla 4 : Resumen cuantitativo de convergencia de AS con Adam

AS con Adam					
	Comparativa en convergencia		Comparativa en tiempo de convergencia en la 15.ª época (min.)		
Tamaño de lote	AS / Default	AS / OBS	AS	OBS	Default
64	1.12	17E2	1.44	0.92	0.98
256	2.95E-3	0.334	0.69	0.52	0.44
1024	0.065	0.48	0.47	0.49	0.42

Como se observa en las tablas 3 y 4, en las columnas de comparativa de convergencia, AS bate en rendimiento (en términos de aceleración de la función de costo) tanto a default random como a OBS cuando la razón de las funciones de costo son valores menores a 1. Usando el optimizador Adadelta, AS es superior a default random con varios ordenes de magnitud para los tamaños de lote 64, 256 y 1024. Por otra parte, usando el optimizador Adam, AS es superior a default random con varios ordenes de magnitud para los tamaños de lote 256 y 1024. Comparando AS y OBS, se observa en la comparativa de convergencia (usando el optimizador Adam), que OBS es superior a AS con el tamaño de lote 64.

Finalmente, se observa en las tablas 2 y 3, que usando un tamaño de lote 64, AS requiere un tiempo de cómputo mayor a 0.5 minutos en comparación con OBS y Default. Para un tamaño de lote 256, AS requiere un tiempo de cómputo mayor que 0.15 y 0.20 minutos en comparación con OBS y Default, respectivamente. Finalmente, para un tamaño de lote 1024, la diferencia entre AS y OBS es de 0.01 minutos, y entre AS y Default es 0.09 minutos.

Capítulo 5

5 Conclusiones

En este trabajo de tesis se ha descrito un método novedoso para muestrear datos con el fin de formar lotes para el entrenamiento de una DNN. Este método es una estrategia de muestreo adaptativo mejorado diseñada para acelerar el proceso de entrenamiento y se basa en la definición de distribuciones óptimas sobre los contenedores de datos y las instancias para lograr una gran reducción en la varianza de las normas de gradiente, lo que conduce a una convergencia más rápida de los modelos.

El muestreo adaptativo para el aprendizaje profundo (AS) es una estrategia que extrapola el trabajo de Gopal para acelerar el entrenamiento de una CNN. Las contribuciones principales de esta estrategia son dos: la utilización y derivación de una distribución óptima para las instancias de cada contenedor; mientras que Gopal considera a las distribuciones de instancias como estáticas (distribución uniforme), AS las considera dinámicas (distribuciones no uniformes). La segunda contribución fue utilizar la aseveración de Katharopoulos & Fleuret para muestrear las instancias de los contenedores con base en los valores de perdida, y con esto se ahorró un importante costo computacional.

Esta propuesta tuvo éxito (para tamaños de lote 64, 256 y 1024) cuando se comparó con el muestreo uniforme que se usa normalmente y también con OBS, que es un método similar en el estado del arte; AS mostró una mejora de convergencia entre 2 y 4 órdenes de magnitud en comparativa con default random (Tablas 3 y 4) con tamaños de lote 64, 256 y 1024 usando Adadelta; usando Adam, AS mostró una mejora de convergencia de 1 y 2 órdenes de magnitud con los tamaños de lote 256 y 1024. En la comparativa de AS y OBS, AS mostró una mejora de convergencia de 0.78, 0.2 y 0.92 para tamaños de lote 64, 256 y 1024 usando Adadelta; usando Adam, AS logra un rendimiento similar para tamaños de lote 256 y 1024.

AS mostró experimentalmente ser robusto para el tamaño de lote 256 y 1024 tanto para el optimizador Adadelta y Adam; para tamaño de lotes pequeños, (tamaño de lote 64) hay una gran inestabilidad del algoritmo para el optimizador Adam. Asimismo, para tamaños de lote de 256 y 1024 (usando Adadelta y Adam respectivamente), AS mostró una capacidad de generalización

para datos de prueba. También, con base en la discusión de la sección 4.4, se puede conjeturar que AS es más estable para optimizadores que adapten su tasa de aprendizaje como Adadelta. Para los optimizadores que usen un promedio ponderado exponencial de los gradientes como regla de actualización de pesos (como lo hace ADAM), AS es inestable para tamaños de lote pequeños (como 64 o 256).

Este trabajo representa sólo un estudio inicial del esquema AS para el entrenamiento de una CNN. Se podrían explorar enfoques más avanzados para abordar problemas de clasificación más difíciles, como los representados por los conjuntos de datos CIFAR-10 [51] o CIFAR-100 [52]. Además, algunas variaciones de la estrategia de muestreo propuesta se pueden implementar fácilmente, como cambiar entre el muestreo con respecto a la pérdida y la norma de gradiente en diferentes puntos durante el entrenamiento o cambiar la forma de obtener los contenedores, realizando una partición de los datos que no fuera con base en la clase . Estas ideas se están considerando para trabajos futuros.

Referencias

[1] Alom, MZ et al. (2018) The history began from alexnet: A comprehensive survey on deep learning approaches. arXiv:1803.01164

[2] Wang, L et al. (2018) Superneurons: Dynamic GPU memory management for training deep neural networks. 23rd ACM SIGPLAN symposium on principles and practice of parallel programming, pp. 41-53.

[3] Apostol, T. M. (1990). Calculus, Volume 2.

[4] Freund, J. E., Miller, I., & Miller, M. (2000). Estadística Matemáticas con Aplicaciones. Pearson educación.

[5] Bouchard, G, Trouillon, T, Perez, J, and Gaidon, A (2015) Online learning to sample. arXiv:1506.09016

[6] Lahmiri, S. (2016). On simulation performance of feedforward and NARX networks under different numerical training algorithms. In Handbook of research on computational simulation and modeling in engineering (pp. 171-183). IGI Global.

[7] Ying, X. (2019, February). An overview of overfitting and its solutions. In Journal of Physics: Conference Series (Vol. 1168, No. 2, p. 022022). IOP Publishing.

[8] Ian goodfellow, yoshua bengio, and aaron courville: Deep learning.

[9] Masegosa, A. R., Cabañas, R., Langseth, H., Nielsen, T. D., & Salmerón, A. (2021). Probabilistic models with deep neural networks. Entropy, 23(1), 117.

[10] Aldrich, J. (1997). RA Fisher and the making of maximum likelihood 1912-1922. Statistical science, 12(3), 162-176.

[11] Strang, G. (2019). Linear algebra and learning from data. Cambridge: Wellesley-Cambridge Press.

[12] W. S. Mcculloch and W. Pitts, "A logical calculus nervous activity," Bull. Math. Biol., vol. 52, no. l, pp. 99–115, 1990.

[13] F. Rosenblatt, "The Perceptron - A Perceiving and Recognizing Automaton," 1957.

[14] M. L. Minsky and S. Papert, "Perceptrons: an introduction to computational geometry," *MIT Press. Cambridge, Expand. Ed.*, vol. 19, no. 88, p. 2, 1988

[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986

[16] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," Neural Comput., vol. 18, no. 7, pp. 1527–1554, 2006

[17] Martinsanz, G. P., & de la Cruz García, J. M. (2011). Aprendizaje automatico. Ediciones de la U.

[18] Dargan, S., Kumar, M., Ayyagari, M. R., & Kumar, G. (2019). A survey of deep learning and its applications: A new paradigm to machine learning. Archives of Computational Methods in Engineering, 1-22.

[19] Andrew Ng, Kian Katanforoosh, Younes Mourri. (11 de junio del 2020). Deep Learning Specialization, *https://www.coursera.org/specializations/deep-learning*

[20] Kukačka, J., Golkov, V., & Cremers, D. (2017). Regularization for deep learning: A taxonomy. arXiv preprint arXiv:1710.10686.

[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014

[22] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by

reducing internal covariate shift," in 32nd International Conference on Machine Learning, ICML 2015, 2015, vol. 1, pp. 448–456
[23] Synge, J. L., & Schild, A. (1978). Tensor calculus (Vol. 5). Courier Corporation.
[24] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. Neural computation, 1(4), 541-551.
[25] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25, 1097-1105.
[26] H. Robbins and S. Monro. A stochastic approximation method. Annals of Mathematical Statistics, 22(3), 1951
[27] L. Bottou. Online algorithms and stochastic approximations. In David Saad, editor, Online Learning and Neural Networks. Cambridge University Press, Cambridge, UK, 1998.
[28] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
[29] Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.
[30] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701.
[31] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research, 12(7).
[32] Katharopoulos, A, and Fleuret, F (2018) Not all samples are created equal: Deep learning with importance sampling. In International conference on machine learning, PMLR, pp. 2525-2534.
[33] Alain, G, Lamb, A, Sankar, C, Courville, A, and Bengio, Y (2015) Variance reduction in SGD by distributed importance sampling. arXiv:1511.06481.
[34] Zhao, P, and Zhang, T (2015) Stochastic optimization with importance sampling for regularized loss minimization. In international conference on machine learning, PMLR, pp. 1-9.
[35] Needell, D, Srebro, N, and Ward, R (2013) Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm. arXiv:1310.5715.
[36] Gopal, S (2016) Adaptive sampling for SGD by exploiting side information. In International Conference on Machine Learning, PMLR, pp. 364-372.
[37] Loshchilov, I and Hutter, F (2015) Online batch selection for faster training of neural networks. arXiv:1511.06343.
[38] Schaul, T, Quan, J, Antonoglou, I, and Silver, D (2015) Prioritized experience replay. arXiv:1511.05952
[39] Katharopoulos, A, and Fleuret, F (2017) Biased importance sampling for deep neural network training. arXiv:1706.00043.
[40] Fan, Y, Tian, F, Qin, T, Bian, J, & Liu, TY (2017) Learning what data to learn. arXiv:1702.08635.
[41] Joseph, KJ, Singh, K, and Balasubramanian, VN (2019) Submodular batch selection for training deep neural networks. arXiv:1906.08771.
[42] Zhao, P, and Zhang, T (2014) Accelerating minibatch stochastic gradient descent using stratified sampling. arXiv:1405.3080.
[43] Wu, C Y, Manmatha, R, Smola, A J, and Krahenbuhl, P (2017) Sampling matters in deep embedding learning. In Proceedings of the IEEE International Conference on Computer

Vision, pp. 2840-2848.
[44] LeCun, Y, Bottou, L, Bengio, Y, and Haffner, P (1998) Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), pp. 2278-2324.
[45] He K, Zhang X, Ren S and Sun J (2016) Deep Residual Learning for Image Recognition. Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778.
[46] Smith, SL, Kindermans, PJ, Ying, C, and Le, QV (2017) Don't decay the learning rate, increase the batch size. arXiv:1711.00489.
[47] Torch Contributors. (11 de junio de 2020). AUTOGRAD MECHANICS, *https://pytorch.org/docs/stable/notes/autograd.html*
[48] PyTorch. (11 de Junio del 2020). AUTOMATIC DIFFERENTIATION WITH TORCH.AUTOGRAD, *https://pytorch.org/tutorials/beginner/basics/autogradqs_tutorial.html*
[49] Sin nombre. (11 de junio del 2020). CONCEPTOS BÁSICOS SOBRE REDES NEURONALES, *http://grupo.us.es/gtocoma/pid/pid10/RedesNeuronales.htm*
[50] Google Colab Team. (11 de junio del 2020). Welcome To Colaboratory, *https://colab.research.google.com/notebooks/intro.ipynb*
[51] A. Krizhevsky, V. Nair, and G. Hinton, The CIFAR-10 dataset, *online http//www. cs. toronto. edu/kriz/cifar. html*, vol. 55, 2014.
[52] A. Krizhevsky, V. Nair, and G. Hinton, The CIFAR-100 dataset, *online http//www. cs. toronto. edu/kriz/cifar. html*, vol. 55, 2014.

Anexos

Anexo A

Sea un campo vectorial diferenciable $F: X \in R^n \rightarrow R^n, n > 1$ definido de la siguiente manera:

$$X = \begin{bmatrix} x_1 \\ . \\ x_n \end{bmatrix} \overset{F}{\rightarrow} \begin{bmatrix} f_1(x_1) \\ . \\ f_n(x_n) \end{bmatrix}. \tag{A.1}$$

De la ecuación (3), se tiene que la matriz jacobiana de F es una matriz diagonal

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & 0 & 0 \\ . & . & . \\ 0 & 0 & \frac{\partial f_n}{\partial x_n} \end{bmatrix}. \tag{A.2}$$

Para simplificar los cálculos, se hacen las siguientes suposiciones.

Sea la $L - esima$ capa de una DNN, donde $Z^L, A^L \in R^2$

$$Z^L = (W^L)^T A^{L-1} + b^L, \tag{A.3}$$

$$A^L = f^L(Z^L),$$

y además f^L es un campo vectorial definido como (A.1).

Expandiendo la primera ecuación de (26) y considerando las suposiciones de (A.3), se tiene que

$$\left(\frac{\partial Z^L}{\partial W^L}\right)^T \left(J_{A^L}\right)^T \left(\nabla_{A^L} J\right) = \begin{bmatrix} \frac{\partial z_1^L}{\partial W^L} & \frac{\partial z_2^L}{\partial W^L} \end{bmatrix} \begin{bmatrix} \frac{\partial a_1^L}{\partial z_1^L} & 0 \\ 0 & \frac{\partial a_2^L}{\partial a_2^L} \end{bmatrix} \begin{bmatrix} \frac{\partial J}{\partial a_1^L} \\ \frac{\partial J}{\partial a_2^L} \end{bmatrix}. \tag{A.4}$$

Desarrollando la ecuación de la derecha de (A.4), se tiene que

$$\begin{bmatrix} \frac{\partial z_1^L}{\partial W^L} & \frac{\partial z_2^L}{\partial W^L} \end{bmatrix} \begin{bmatrix} \frac{\partial a_1^L}{\partial z_1^L} & 0 \\ 0 & \frac{\partial a_2^L}{\partial a_2^L} \end{bmatrix} \begin{bmatrix} \frac{\partial J}{\partial a_1^L} \\ \frac{\partial J}{\partial a_2^L} \end{bmatrix} = \frac{\partial z_1^L}{\partial W^L} \left[\left(\frac{\partial J}{\partial a_1^L}\right) \left(\frac{\partial a_1^L}{\partial z_1^L}\right) \right] + \frac{\partial z_2^L}{\partial W^L} \left[\left(\frac{\partial J}{\partial a_2^L}\right) \left(\frac{\partial a_2^L}{\partial z_2^L}\right) \right]. \tag{A.5}$$

Se observa de la ecuación (A.5) que $\frac{\partial z_1^L}{\partial W^L}$ y $\frac{\partial z_2^L}{\partial W^L}$ son derivadas parciales de un numero con respecto a una matriz. En consecuencia, dichas derivadas se representan de la siguiente manera

$$\frac{\partial z_1^L}{\partial W^L} = \begin{bmatrix} \frac{\partial z_1^L}{\partial w_{11}^L} & \frac{\partial z_1^L}{\partial w_{12}^L} \\ \frac{\partial z_1^L}{\partial w_{21}^L} & \frac{\partial z_1^L}{\partial w_{22}^L} \end{bmatrix} = \begin{bmatrix} a_1^{L-1} & a_2^{L-1} \\ 0 & 0 \end{bmatrix}, \qquad \frac{\partial z_2^L}{\partial W^L} = \begin{bmatrix} \frac{\partial z_2^L}{\partial w_{11}^L} & \frac{\partial z_2^L}{\partial w_{12}^L} \\ \frac{\partial z_2^L}{\partial w_{21}^L} & \frac{\partial z_2^L}{\partial w_{22}^L} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ a_1^{L-1} & a_2^{L-1} \end{bmatrix}. \tag{A.6}$$

Sustituyendo (A.6) en (A.5),

$$\left(\frac{\partial Z^L}{\partial W^L}\right)^T \left(J_{A^L}\right)^T \left(\nabla_{A^L} J\right) = \begin{bmatrix} a_1^{L-1}\left(\frac{\partial J}{\partial a_1^L}\right)\left(\frac{\partial a_1^L}{\partial z_1^L}\right) & a_2^{L-1}\left(\frac{\partial J}{\partial a_1^L}\right)\left(\frac{\partial a_1^L}{\partial z_1^L}\right) \\ a_1^{L-1}\left(\frac{\partial J}{\partial a_2^L}\right)\left(\frac{\partial a_2^L}{\partial z_2^L}\right) & a_2^{L-1}\left(\frac{\partial J}{\partial a_2^L}\right)\left(\frac{\partial a_2^L}{\partial z_2^L}\right) \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial a_1^L}{\partial z_1^L} & 0 \\ 0 & \frac{\partial a_2^L}{\partial a_2^L} \end{bmatrix} \begin{bmatrix} \frac{\partial J}{\partial a_1^L} \\ \frac{\partial J}{\partial a_2^L} \end{bmatrix} \begin{bmatrix} a_1^{L-1} & a_2^{L-1} \end{bmatrix} \tag{A.7}$$

$$= \left(J_{A^L}\right)^T \left(\nabla_{A^L} J\right) \left(A^{L-1}\right)^T$$

Se puede observar que la ecuación (A.7) es extrapolable para jacobianos tipo ecuación (3).

Anexo B

Sea la siguiente DNN y función de pérdida:

$$F\left(X; b^1, W^1, .., b^l, W^l .., b^{L-1}, W^{L-1}, b^L, W^L\right) : X \subseteq R^n \rightarrow A^1 \subseteq R^{n_1} \rightarrow A^2 \subseteq R^{n_2} \rightarrow \cdots \rightarrow A^L \subseteq R^{n_L}, \tag{B.1}$$

$$L : (A^L, Y) \subseteq R^{n_L} \rightarrow R.$$

Los primeros gradientes de F con respecto a W^L y b^L se demostraron en el Anexo A y se exponen en su forma general en (26) y (27). Los gradientes de F con respecto a W^{L-1} y b^{L-1} están dados por

$$\frac{\partial L}{\partial W^{L-1}} = \left(\frac{\partial Z^{L-1}}{\partial W^{L-1}}\right)^T \left(J_{A^{L-1}}\right)^T \left(\frac{\partial Z^L}{\partial A^{L-1}}\right)^T \left(J_{A^L}\right)^T \left(\nabla_{A^L} J\right) \tag{B.2}$$

Donde

$$\frac{\partial Z^L}{\partial A^{L-1}} = \frac{\partial}{\partial A^{L-1}}[(W^L)^T A^{L-1} + b^L] = \begin{bmatrix} \frac{\partial z_1^L}{\partial a_1^{L-1}} & \cdot & \cdot & \frac{\partial z_1^L}{\partial a_{n_{L-1}}^{L-1}} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \frac{\partial z_{n_L}^L}{\partial a_1^{L-1}} & \cdot & \cdot & \frac{\partial z_{n_L}^L}{\partial a_{n_{L-1}}^{L-1}} \end{bmatrix} \tag{B.3}$$

$$= \begin{bmatrix} w_{11}^L & \cdot & \cdot & w_{1n_{L-1}}^L \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ w_{n_L 1}^L & \cdot & \cdot & w_{n_L n_{L-1}}^L \end{bmatrix} = (W^L)^T,$$

entonces

$$\frac{\partial L}{\partial W^{L-1}} = \left(\frac{\partial Z^{L-1}}{\partial W^{L-1}}\right)^T (J_{A^{L-1}})^T (W^L)(J_{A^L})^T (\nabla_{A^L} J). \tag{B.4}$$

Se observa de la ecuación (B.4) que $(J_{A^{L-1}})^T (W^L)(J_{A^L})^T$ es una matriz de dimensiones (n_{L-1}, n_L) multiplicado por el gradiente $(\nabla_{A^L} J)$. Entonces, a la ecuación (B.4) se le puede aplicar le mismo procedimiento del Anexo A, y se obtiene

$$\frac{\partial L}{\partial W^{L-1}} = (J_{A^{L-1}})^T (W^L)(J_{A^L})^T (\nabla_{A^L} J)(A^{L-2})^T. \tag{B.5}$$

Siguiendo con el mismo procedimiento, los gradientes de F con respecto a W^l y b^l están dados por

$$\frac{\partial L}{\partial W^l} = \prod_{i=l}^{L-1} (J_{A^i})^T \left(\frac{\partial Z^{i+1}}{\partial A^i}\right)^T (J_{A^L})^T (\nabla_{A^L} J)(A^{l-1})^T. \tag{B.6}$$

Donde

$$\triangle^l = \prod_{i=l}^{L-1} (J_{A^i})^T \left(\frac{\partial Z^{i+1}}{\partial A^i}\right)^T = \prod_{i=l}^{L-1} (J_{A^i})^T (W^{i+1}), \tag{B.7}$$

entonces

$$\frac{\partial L}{\partial W^l} = (\triangle^l)(J_{A^L})^T (\nabla_{A^L} J)(A^{l-1})^T. \tag{B.8}$$

Se tiene que $\frac{\partial Z^l}{\partial b^l}$ es un vector de unos de la dimensión de Z^l, entonces

$$\frac{\partial L}{\partial b^l} = (\triangle^l)(J_{A^L})^T (\nabla_{A^L} J). \tag{B.9}$$

Anexo C

Sea la siguiente CNN y función de pérdida:

$$F\left(X; b^1, K^1, .., b^l, K^l .., b^{L-1}, K^{L-1}, b^L, K^L\right): X \subseteq R^{n_h^0 x n_w^0} \to \cdots \to A^L \subseteq R^{n_h^L x n_w^L}, \qquad (C.1)$$
$$J: (A^L, Y) \subseteq R^{n_L} \to R.$$

Como se observa en (C.1), la CNN así definida no tiene profundidad, eso quiere decir que solo se considera un kernel para esta derivación.

De acuerdo con (33) y (37), sea la $l - esima$ capa

$$Z^l = A^{l-1} * K^l + b^l 1^l,$$
$$Z^l_{i,j} = \sum_n \sum_m A^{l-1}_{i+n,j+m} K^l_{n,m} + b^l 1^l_{i,j}. \qquad (C.2)$$

Donde (i, j) y (n, m) son los índices de la salida de la capa y kernel respectivamente (los índices empiezan en cero).

Se hace la suposición que se conoce el siguiente gradiente:

$$\frac{\partial J}{\partial Z^l}. \qquad (C.3)$$

Entonces, el gradiente del kernel está dado por

$$\frac{\partial J}{\partial K^l} = \frac{\partial J}{\partial Z^l}\frac{\partial Z^l}{\partial K^l}$$
$$\frac{\partial J}{\partial k^l_{n,m}} = \sum_i \sum_j \frac{\partial J}{\partial Z^l_{i,j}}\frac{\partial Z^l_{i,j}}{\partial K^l_{n,m}} \qquad (C.4)$$

De (C.4) se observa que $\frac{\partial Z^l_{i,j}}{\partial K^l_{n,m}}$ es una derivada parcial de un numero con respecto a otro número; de la figura 5, se puede observar también que los pesos $k^l_{n,m}$ están compartidos en la matriz Z^l. Se considera un strike de 1 y cero padding.

Haciendo un poco de trabajo, se consideran cuatro derivadas parciales de (C.4) para que se dilucide la idea,

$$\frac{\partial J}{\partial k^l_{0,0}} = \sum_i \sum_j \frac{\partial J}{\partial Z^l_{i,j}}\frac{\partial Z^l_{i,j}}{\partial K^l_{0,0}} = \sum_i \sum_j \frac{\partial J}{\partial Z^l_{i,j}}\frac{\partial}{\partial K^l_{0,0}}\left(A^{l-1}_{i,j} K^l_{0,0}\right) = \sum_i \sum_j \frac{\partial J}{\partial Z^l_{i,j}} A^{l-1}_{i,j}, \qquad (C.5)$$

$$\frac{\partial J}{\partial k^l_{0,1}} = \sum_i \sum_j \frac{\partial J}{\partial Z^l_{i,j}} A^{l-1}_{i,j+1},$$

$$\frac{\partial J}{\partial k_{1,0}^{l}} = \sum_{i}\sum_{j}\frac{\partial J}{\partial Z_{i,j}^{l}} A_{i+1,j}^{l-1},$$

$$\frac{\partial J}{\partial k_{1,1}^{l}} = \sum_{i}\sum_{j}\frac{\partial J}{\partial Z_{i,j}^{l}} A_{i+1,j+1}^{l-1}.$$

Entonces

$$\frac{\partial J}{\partial k_{n,m}^{l}} = \sum_{i}\sum_{j}\frac{\partial J}{\partial Z_{i,j}^{l}} A_{i+n,j+m}^{l-1} \qquad \textit{(C.6)}$$

Observando (C.2) y (C.6) es claro que (C.6) es una convolución de la matriz A^{l-1} con la matriz $\frac{\partial J}{\partial Z^{l}}$. En este caso, A^{l-1} funge como la matriz de entrada y $\frac{\partial J}{\partial Z^{l}}$ como el kernel de convolución, entonces

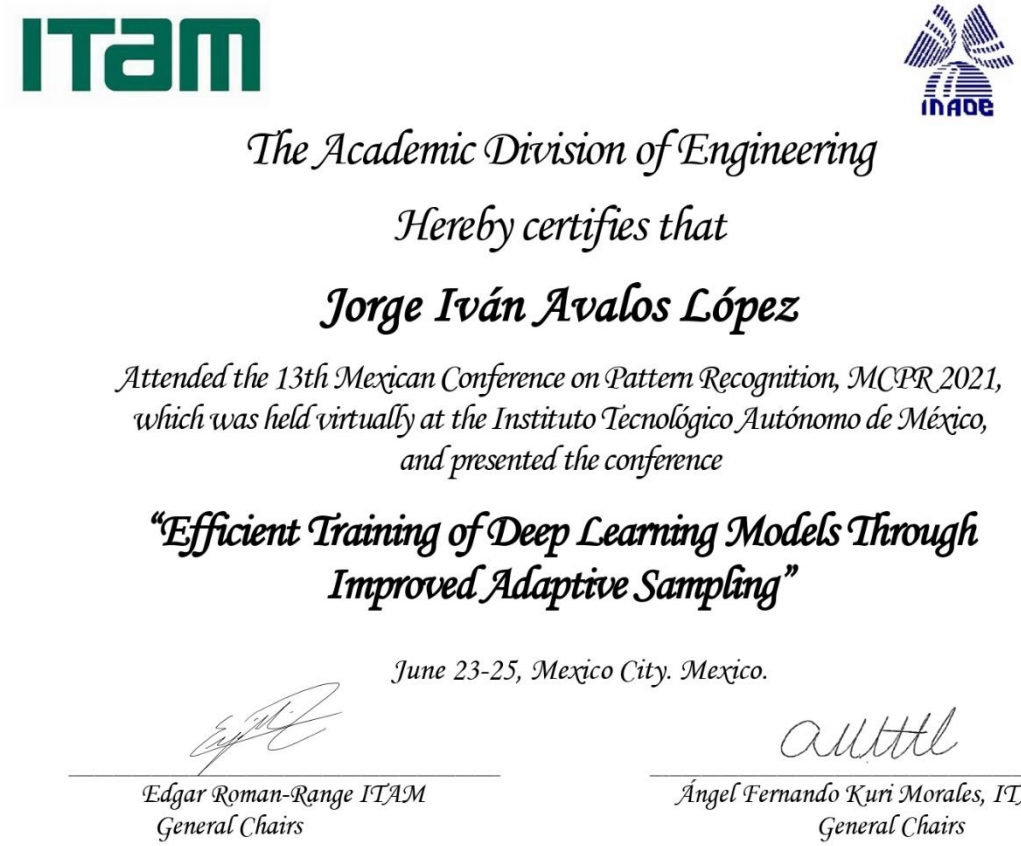
$$\frac{\partial J}{\partial K^{l}} = A^{l-1} * \frac{\partial J}{\partial Z^{l}}, \qquad \textit{(C.7)}$$

El gradiente del sesgo no presenta dificultad debido a que $\frac{\partial Z^{l}}{\partial b^{l}} = 1^{l}$, y es una matriz de unos. Entonces

$$\frac{\partial J}{\partial b^{l}} = \sum_{i}\frac{\partial J}{\partial Z^{l}{}_{i}}. \qquad \textit{(C.8)}$$

Para obtener gradientes sucesores del kernel $l - esimo$ es preciso calcular $\frac{\partial Z^{l}}{\partial A^{l-1}}$. También para expandir el procedimiento de la obtención de gradientes a más de un kernel y a una entrada de más dimensiones, es necesario añadir una sumatoria extra en las expresiones para recorrer los términos en profundidad. Así mismo, es también interesante la derivación A^{l} con respecto a Z^{l} ($\frac{\partial A^{l}}{\partial Z^{l}}$). Por motivos de exceso de contenido no se presentan esos cálculos.

Anexo D



The Academic Division of Engineering

Hereby certifies that

Jorge Iván Avalos López

Attended the 13th Mexican Conference on Pattern Recognition, MCPR 2021, which was held virtually at the Instituto Tecnológico Autónomo de México, and presented the conference

"Efficient Training of Deep Learning Models Through Improved Adaptive Sampling"

June 23-25, Mexico City. Mexico.

Edgar Roman-Range ITAM
General Chairs

Ángel Fernando Kuri Morales, ITAM
General Chairs

Efficient Training of Deep Learning Models through Improved Adaptive Sampling

Jorge Ivan Avalos-López [a], **Alfonso Rojas-Domínguez** [0000-0003-1818-4162]a*
Manuel Ornelas-Rodríguez [0000-0002-6381-2861]a, **Martín Carpio** [0000-0002-1191-2676]a
and **S. Ivvan Valdez** [0000-0002-5996-992X]b*

[a] Tecnológico Nacional de México/Instituto Tecnológico de León, 37290 León, México.
[b] Centro de Investigación en Ciencias de Información Geoespacial A.C., 76703, Querétaro, México.
✉ avaloslj2014@licifug.ugto.mx, Alfonso.Rojas@gmail.com
*CONACYT Research Fellow.

Abstract. Training of Deep Neural Networks (DNNs) is very computationally demanding and resources are typically spent on training-instances that do not provide the most benefit to a network's learning; instead, the most relevant instances should be prioritized during training. Herein we present an improved version of the Adaptive Sampling (AS) method (Gopal, 2016) extended for the training of DNNs. As our main contribution we formulate a probability distribution for data instances that minimizes the variance of the gradient-norms w.r.t. the network's loss function. Said distribution is combined with the optimal distribution for the data classes previously derived by Gopal and the Improved AS is used to replace uniform sampling with the objective of accelerating the training of DNNs. Our proposal is comparatively evaluated against uniform sampling and against Online Batch Selection (Loshchilov & Hutter, 2015). Results from training a Convolutional Neural Network on the MNIST dataset with the Adadelta and Adam optimizers over different training batch-sizes show the effectiveness and superiority of our proposal.

Keywords. Deep learning · Convolutional Neural Networks · Gradient Descent · Importance Sampling

1. Introduction

During the past decade, the increase in available data and computational power (development of graphics cards with higher computing capacity) as well as important theoretical developments, propelled the advent of Deep Learning (DL) techniques and their further dominance in several pattern recognition applications such as computer vision, pattern classification, natural language processing, forecasting and so on [1].

The bottleneck of DL models is their training; due to the very large amounts of data to be processed and the millions of trainable parameters, the training process is usually very computationally expensive [2], although relatively simple. The standard method for training many deep neural networks (DNNs), in particular for classification tasks, is stochastic gradient descent (SGD) and its variations. A network is presented with a set of examples, an error function is computed and the gradient of this error function is back-propagated through the network in order to determine the necessary adjustments to its parameters. In practice, instead of adjusting the network one data instance at a time, the training dataset is divided into so-called mini-batches of fixed size and the network is trained on one mini-batch (i.e. on a few data instances) at a time.

The conventional way of forming the mini-batches is trough uniform sampling of the data instances, but recently there have been efforts to develop strategies to accelerate the training of DNNs through more efficient sampling schemes. In 2016, Loshchilov & Hutter [4] and Alain et al. [7] explored adaptive and intelligent selection mechanisms that focus computing resources on the *most informative* examples; this is called *Importance Sampling* (IS). In 2017 Katharopoulos & Fleuret [3] proposed a *biased importance sampling*, based on a biased gradient estimate designed to reduce the variance over the stochastic gradients.

Such novel sampling strategies are built around the idea that not all samples are equally important in the training process and the most relevant instances should be prioritized to accelerate learning. Said strategies employ either the loss-values or the gradient-norms as measures of the importance, but none has used both measures jointly. Sampling based on the gradient-norms tends to show better performance, but the advantage over using loss-values is not too large [3]. Meanwhile, using the gradient-norm is more computationally expensive than using the loss-value. Developing a method that combines the two measures is an interesting idea, which we explore in this work.

Following the same guiding principle as previous works we propose a new IS scheme that accelerates the training of Convolutional Neuronal Networks (CNNs) by focusing the computation on the samples that minimize the variance of the gradients w.r.t. the loss function. As such, this work is an extension of Gopal's work [5] for the training of DNNs; specifically we propose a novel probability distribution to perform IS. Our proposal is validated using a CNN trained on the MNIST classification task and we demonstrate that our strategy speeds up the training procedure compared to the standard sampling scheme (uniform sampling) as well as compared to the proposal of Loshchilov & Hutter [4], a method considered in the state of the art.

2. Related Work

Several strategies to improve the convergence of SGD have been described in the literature. Most of these are focused on improving the way in which data instances are selected to form the training mini-batches. We refer to said strategies as *Selective Sampling* strategies. The key idea behind these methods is to design a non-uniform distribution to replace the uniform distribution that is usually employed to sample training instances.

In the literature, we have identified two approaches to the problem of Selective Sampling. The first approach constructs a probability distribution according to the gradient norm of each training instance; this is done with the objective of reducing the variance between the gradients of the sampled instances. Under this approach, Katharopoulos & Fleuret [3] proposed to accelerate the training of a DNN using a probability distribution based on the gradient-norm of each instance; they ranked the instances w.r.t. their loss-value and then sampled these based on that ranking. They also derived an upper bound on the gradient-norm, and used it to estimate the most convenient moments to switch between uniform sampling and IS along the training process. Similarly, Alain et al. [7] employed the gradient-norm to define each instance's importance. Zhao & Zhang [8] and Needell et al. [9] showed that the optimal sampling distribution is proportional to the per-instance gradient-norm and established a clear connection with the variance of the gradient estimates of SGD. Gopal [5] defined a distribution of classes that is directly proportional to the norm of the gradients, in order to sample from those classes with instances that lead to the maximum reduction in the variance of the gradients.

The second approach to Selective Sampling consists of constructing a probability distribution based on the loss value of each individual training instance. The loss function is a measure of how well a model can solve particular instances. If the loss of an instance is high then it should be sampled more frequently so that the model can have more opportunities to learn to solve it. The idea is to preferably supply the DNN with instances that are harder to classify or forecast, since this should make the learning process more efficient. Following this approach, Loshchilov & Hutter [4] employed the loss to define each instance's importance, ranked the instances w.r.t. their latest known loss value and built a probability distribution that decays exponentially as a function of that ranking. Likewise, Schaul [10] and Katharopoulos & Fleuret [11] use the loss to create their sampling distributions. They keep a history of losses for previously seen instances and for new training iterations they sample instances proportionally to that loss.

Besides Selective Sampling strategies there have also been other proposals to optimize the training of DNNs. Fan et al. [6] used reinforcement learning to train a neural network that selects instances in order to optimize the convergence of a second neural network. Joseph et al. [13] designed a mini-batch selection strategy based on the maximization of a submodular function that captures relevant information from the data used to speed up the training. Zhao & Zhang [14] divided the data set into clusters with low variance, and then strategically sampled mini-batches from those clusters. Finally, Wu et al. [15] designed a distribution that maximizes the diversity of the losses in a training batch.

Our proposed IS is an extrapolation and modification of the Adaptive Sampling (AS) algorithm of Gopal [5] to DL models, with the addition of a redefined probability distribution that combines the gradient and the loss of individual instances and of the mini-batches. This addition is possible due to important contributions by Katharopoulos & Fleuret [3],[11]. The details of our proposal are provided in the following section.

3. Adaptive Sampling for Deep Learning

Let $D = \{(x_1, y_1), (x_2, y_1), \dots, (x_N, y_N)\}$, $x_i \in \mathbb{R}^n, y_i \in \mathbb{R}^m, \forall i = 1, \dots, N$ be the training set, where X represents a set of data instances and Y their set of labels, $\varphi(x_i; \theta) : \mathbb{R}^n \to \mathbb{R}^m$ be a Deep Learning model for classification of m classes, parameterized by vector θ, and $L(\varphi(x_i; \theta), y_i) : \mathbb{R}^m \to \mathbb{R}^+$ be the loss function to be minimized during training. Training with SGD means finding the optimum vector θ^*, usually by solving:

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, E_{x_i \sim \mathcal{U}}[L(\varphi(x_i;\theta), y_i)] = \underset{\theta}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N} L(\varphi(x_i;\theta), y_i) \quad (1)$$

where $\mathcal{U}$ represents the uniform distribution. Guillaume et al. [16] showed that SGD is a powerful tool to optimize the sampling distribution of Monte Carlo (MC) estimators, motivating the idea of employing a sampling distribution which is not constant through time. IS can be seen as a method to estimate (1), surrogating a uniform distribution with a non-uniform one. Thus we have a MC estimator of the form [16]:

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, E_{x_i \sim \mathcal{V}}[\zeta(x_i) L(\varphi(x_i;\theta), y_i)] = \underset{\theta}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N} \frac{1}{\mathcal{V}(x_i)} L(\varphi(x_i;\theta), y_i) \quad (2)$$

where $\zeta(x_i) = \frac{\mathcal{U}(x_i)}{\mathcal{V}(x_i)}$ is called the importance weight and $\mathcal{V}$ the *importance distribution* [16].

Following the work of Gopal [5], we associate *side information* to each x_i. This side information can represent a class label, a measure that is inherent to each instance, or it can be any tag associated with x_i that enables us to create a partition C of dataset D: $C = \{c_1, c_2, \dots, c_k\}$, $c_j = \left\{x_1^j, x_2^j, \dots, x_{m_j}^j\right\}\ \forall j = 1, 2, \dots, k$. That is, the N instances are split into k mutually exclusive bins, which are not necessarily balanced.

Let us now define a probability distribution over the bins: $P = \{p_1, p_2, \dots, p_k\}$, $p_j = \mathcal{P}(C = c_j)$; and a probability distribution over the elements of each bin: $Q^j = \left\{q_1^j, q_2^j, \dots, q_{|I_j|}^j\right\}$, $q_i^j = \mathcal{P}\left(c_j = x_{i \in I_j}^j\right)$, where $I = \{I_1, I_2, \dots, I_k\}$ are index sets of the partition C. In our notation a superindex is used to indicate the j-th bin and a subindex indicates the i-th element of the corresponding j-th bin.

Assuming that P and Q^j are independent, the probability of picking the i-th training instance is given by:

$$\mathcal{P}(X = x_i) = p_j q_i^j \quad (3)$$

Using this result, and defining $\mathcal{V}(x_i)$ as $p_j q_i^j$, we can rewrite (2) as:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{N}\sum_{c_j \in C}\sum_{i \in c_j} \frac{1}{p_j q_i^j} L\left(\varphi\left(x_i^j;\theta\right), y_i^j\right) \quad (4)$$

This problem is solved by gradient descent, so we require the gradient d^t at training iteration t:

$$d^t = \frac{1}{N}\frac{1}{p_j q_i^j} \nabla_\theta L(\varphi\left(x_i^j;\theta^{t-1}\right), y_i^j) \quad (5)$$

As our main contribution, we propose to employ non-uniform distributions for both P and Q^j, defined to reduce the total variance of the gradients and improve the estimate of the descent direction.

Formally, we define two minimization problems, (6) and (11) respectively: the first one is the minimization of the gradient variance w.r.t P, assuming a fixed Q^j; the second problem is the counterpart of the first one, i.e. the minimization of the gradient variance w.r.t Q^j assuming a fixed P. The first problem is:

$$\min_P V[d^t] = \min_{p_1,\dots,p_k} \left(\mathbb{E}\left[{d^t}^{\mathrm{T}} d^t\right] - \mathbb{E}[d^t]^{\mathrm{T}}\mathbb{E}[d^t]\right) \quad (6)$$

The solution to (6) is due to Gopal [5] and gives the optimal distribution P at the t-th iteration:

$$p_j \propto \frac{1}{N}\sqrt{\sum_{i \in c_j} \frac{1}{q_i^j} \left\| \nabla_\theta L(\varphi\left(x_i^j;\theta^{t-1}\right), y_i^j) \right\|^2} \quad (7)$$

Using Gopal's idea of reformulating the uniform distribution as the product of one distribution for the bins times a second distribution for the instances, i.e. $1/N = \left(|c_j|/N\right)\left(1/|c_j|\right)$, from (4) we have:

$$\frac{1}{N}\sum_{c_j \in C}\sum_{i \in c_j} \frac{1}{p_j q_i^j} L\left(\varphi\left(x_i^j;\theta\right), y_i^j\right) = \sum_{c_j \in C} \frac{1}{p_j}\frac{|c_j|}{N}\frac{1}{|c_j|}\sum_{i \in c_j} \frac{1}{q_i^j} L\left(\varphi\left(x_i^j;\theta\right), y_i^j\right) \quad (8)$$

So, we have the following optimization problem defined only for the instances of a particular bin:

$$\theta_j^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{|c_j|} \sum_{i \in c_j} \frac{1}{q_i^j} L\left(\varphi\left(x_i^j; \theta\right), y_i^j\right), \forall j = 1, \dots, k \quad (9)$$

For which the gradient d_j^t at training iteration t is:

$$d_j^t = \frac{1}{q_i^j} \frac{1}{|c_j|} \nabla_\theta L(\varphi\left(x_i^j; \theta^{t-1}\right), x_i^j) \quad (10)$$

Then, akin to (6) we pose the minimization of the gradient variance, but this time w.r.t. Q^j:

$$\min_{Q^j} V\left[d_j^t\right] = \min_{q_1^j, \dots, q_{|I_j|}^j} \left(\mathbb{E}\left[{d_j^t}^{\mathrm{T}} d_j^t\right] - \mathbb{E}\left[d_j^t\right]^{\mathrm{T}} \mathbb{E}\left[d_j^t\right]\right), \forall j = 1, \dots, k \quad (11)$$

where we can find the following expectations:

$$\mathbb{E}\left[{d_j^t}^{\mathrm{T}} d_j^t\right] = \sum_{i \in c_j} q_i^j \frac{1}{\left(q_i^j\right)^2} \frac{1}{\left(|c_j|\right)^2} \left\| \nabla_\theta L(\varphi\left(x_i^j; \theta^{t-1}\right), x_i^j) \right\|^2 \quad (12)$$

$$\mathbb{E}\left[d_j^t\right] = \sum_{i \in c_j} q_i^j \frac{1}{|c_j|} \frac{1}{q_i^j} \nabla_\theta L\left(\varphi\left(x_i^j; \theta^{t-1}\right), x_i^j\right) = \frac{1}{|c_j|} \sum_{i \in c_j} \nabla_\theta L\left(\varphi\left(x_i^j; \theta^{t-1}\right), x_i^j\right) \quad (13)$$

Notice that $\mathbb{E}\left[d_j^t\right]$ is independent of q_i^j and we can ignore it. We are left with $\mathbb{E}\left[{d_j^t}^{\mathrm{T}} d_j^t\right]$ and proceed to minimize the variance following Gopal's solution (details are found in [5]). Finally, the optimal distribution Q^j at the t-th iteration is given by:

$$q_i^j \propto \frac{1}{|c_j|} \left\| \nabla_\theta L(\varphi\left(x_i^j; \theta^{t-1}\right), y_i^j) \right\| \quad (14)$$

Intuitively, (3) can be understood as two consecutive selection procedures: first, according to (7) we pick bins which have a larger gradient contribution relative to other bins; then according to (14) we sample the instances with the larger gradient contribution within that bin. The motivation behind this proposal is that these two selection procedures result in more efficient sampling of training data instances than uniform sampling, which should be translated into a reduction of the time required to train a DNN.

3.1 Computational Cost

There are three major problems in computing the probability distributions (7) and (14).

1. For (7), it is impractical to reset the optimal sampling distribution after each iteration. The solution proposed by Gopal [5] is to update p_j every β iterations, which means to re-sample a few training instances from each bin (the appropriate amount of instances depends on the mini-batch size). The downside of this approach is that there is a new hyper-parameter β to be tuned.

2. For (14), computing the gradient norm for an individual instance requires computing a square root for each instance (notice that this is not required in (7)). Katharopoulos & Fleuret [11] show that sampling using the loss exhibits similar variance reducing properties to sampling according to the gradient norm. Using that result, we replace (14) with:

$$q_i^j \propto L(\varphi\left(x_i^j; \theta\right), y_i^j) \quad (15)$$

3. The last difficulty is found in computing the gradient norm. Deep Learning models are characterized by having hundreds of thousands or millions of parameters (e.g. *RestNet-50* has 23 million trainable parameters [19]), so that computing the full gradient becomes prohibitive. In order to overcome this obstacle, Katharopoulos & Fleuret [3] derived an upper-bound of the gradient norm:

$$\hat{G}_i \equiv \left\| \nabla_{\theta_t^l} L(\varphi\left(x_i^j; \theta^{t-1}\right), y_i^j) \right\|^2 \geq \left\| \nabla_\theta L(\varphi\left(x_i^j; \theta^{t-1}\right), y_i^j) \right\|^2 \quad (16)$$

where $\nabla_{\theta_t^l} L(\varphi(x_i^j; \theta^{t-1}), y_i^j)$ is the gradient w.r.t. the pre-activation outputs of the last layer in a DNN. This upper bound is employed in lieu of the full gradient, enabling the implementation of our proposal.

The pseudocode of our proposal for Improved Adaptive Sampling for training of DL models is given in **Algorithm 1**. During training, the probability distributions P and Q^j may tend to become zero for some bins or some instances, respectively. This behavior causes those bins and instances to never be selected again. To avoid such issues, P and Q^j are smoothed to mitigate the appearance of pronounced high and low values. This introduces two more hyper-parameters δ_p and δ_q, used to modulate the smoothing of P and Q^j respectively (see lines 15 and 23 of Algorithm 1).

Algorithm 1: **Improved Adaptive Sampling for Deep Learning**

1. **Input:** $c_j = \{\{x_i, y_i\}_{i=1}^{N/k}\}_{j=i}^{k}$; epochs; $(\delta_{pbin}, \delta_q) \in (0,1)$; $\beta \in \mathbb{Z}^+$, batch size BS, No. of instances N
2. **Initialize:** $gc[1, \dots, k] \leftarrow \emptyset$; $P[p_1, p_2, \dots, p_k] \sim \mathcal{U}$; $Q^j \left[q_1^j, q_2^j, \dots, q_{|I_j|}^j\right] \sim \mathcal{U}$
   $t \leftarrow 0$; $l_{(1:N/k)}^{(1:k)} \leftarrow \infty$: l is a k–dimensional array to store loss values.
3. **For** $epoch = 1$ **to** $epochs$ **do:**
4. **For** $b = 1$ **to** N/BS **do:**
5. **For** $s = 1$ **to** BS **do:**
6. $j \sim P$: Sample bin j according to (7).
7. $i \sim Q^j$: Sample instance i according to (14).
8. $l_i^j \leftarrow L(\varphi(x_i^j; \theta), y_i^j)$: Forward pass: compute the loss, store it in array l.
9. d^t : Compute the gradient using (5).
10. $\hat{G}_i$: Compute the gradient upper bound using (16).
11. $gc[j] \leftarrow sg[j] + \hat{G}_i$: Accumulate gradients of bin j
12. $gc[j] \leftarrow gc[j] / Q^j[i]$: Divide the accumulated gradients by q_i^j
13. **If** $(t \bmod \beta) = 0$ **then:**
14. $p_j \propto \sqrt{gc[j]}/N$: Re-compute p_j according to (7) $\forall\, j$.
15. $(p_j)^t \leftarrow \delta_p (p_j)^{t-1} + (1 - \delta_p)\,(p_j)^t$: Smooth current p_j using previous values.
16. $gc[1, \dots, k] \leftarrow 0$
17. **End if**
18. $t \leftarrow t + 1$
19. **End for**
20. $d^b \leftarrow \sum_{r=1}^{BS} d^r / BS$: Compute the batch gradient.
21. Update θ^t using d^b : Backward pass.
22. $q_i^j \propto l_i^j$: Re-compute q_i^j according to (14)
23. $(q_i^j)^t = \delta_q (q_i^j)^{t-1} + (1 - \delta_q)\,(q_i^j)^t$: Smooth current Q^j using previous values
24. **End for**
25. **End for**
26. **Return:** optimized network parameters θ^*

4. Experimental Results

We investigate our proposed adaptive sampling on the standard MNIST dataset of handwritten digits (LeCun et al. [17]) with 60,000 training data points. We only conducted our experiments on training data (i.e., no validation or test data was used to evaluate the performance of the network). We used a CNN with two convolution and max-pooling layers, two fully connected (FC) layers, and ReLu activation for all the layers except for the last one (softmax activation, as part of the cost function). We employed Batch Normalization on the convolutional layers, dropout of 0.5 on the first FC layer and data normalization. The cost function used was SoftMax cross-entropy loss. All experiments were implemented using the Python library PyTorch and

executed on GPUs through Google Colab[1] notebooks. We evaluated our proposal in combination with two different optimizers: Adadelta [18] and Adam [20].

Experiments consisted in training our CNN for 50 epochs and reporting the average loss over three runs. For all the experiments C was set to be the set of class-labels. The hyper-parameters were set as follows: $\beta = 10$ to-ensure enough gradients accumulation without saturating any bin; the smoothing hyper-parameters were adjusted using grid search, with $\delta_p \in (0, 0.9)$ and $\delta_q \in (0.1, 1)$; the optimum values obtained were $\delta_p = 0.9$ and $\delta_q = 0.1$. The final hyper-parameter to be set is the size of the mini-batches; we repeated the experiments with three commonly employed batch sizes of 64, 256 and 1024 training instances.

Results are shown in Fig. 1 (Adadelta) and Fig. 2 (Adam). Our own method is referred to as *Adaptive Sampling* (AS) and is compared against two alternatives: the standard training algorithm, that samples data instances under a uniform distribution referred to as *Default Random*, and the proposal of Loshchilov & Hutter [4] called *Online Batch Selection* (OBS). This latter algorithm requires one extra hyper-parameter and it is set to the optimal value reported in [4]: $r, s = 10^8$ from the beginning to the end of training.

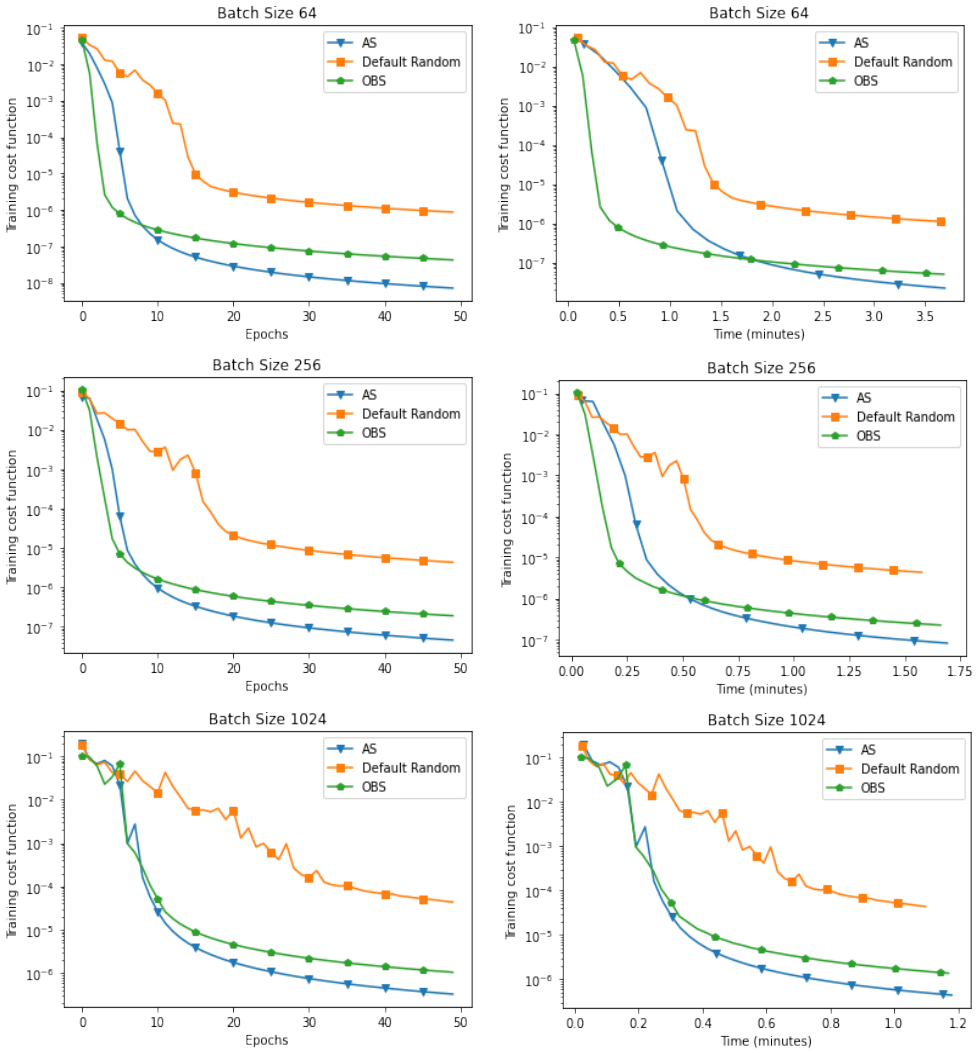


Fig. 1. *Comparison of our proposed Adaptive Sampling (AS) vs SGD with uniform sampling (Default Random) and Online Batch Selection (OBS). Convergence curves of Adadelta (learning rate=1.0, rho= 0.9), while training a CNN for classification of MNIST dataset. Curves shown are the average of the median over three runs.*

[1] https://colab.research.google.com/

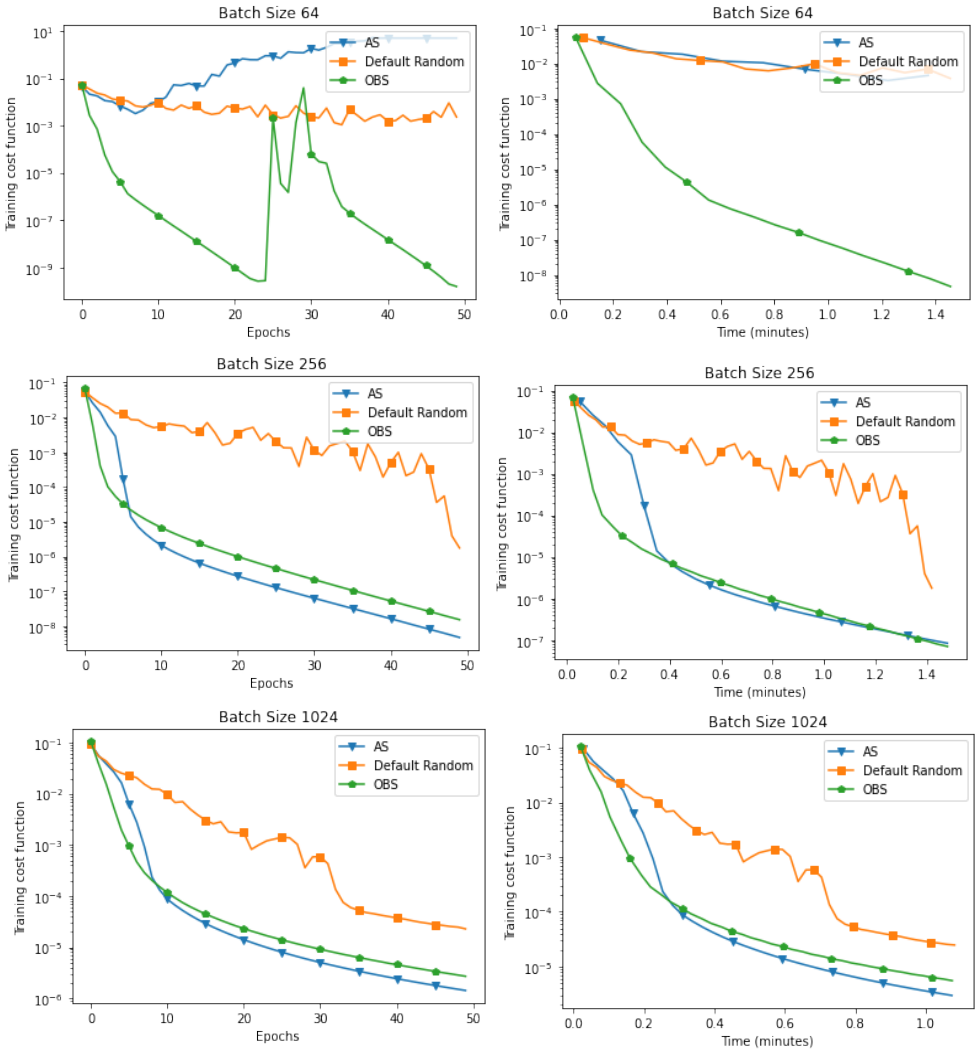


Fig. 2. *Comparison of our proposed Adaptive Sampling (AS) vs SGD with uniform sampling (Default Random) and OBS. Convergence curves of Adam (learning rate =0.001,* $beta_1$ *=0.9,* $beta_2 = 0.999$*), while training a CNN for classification of MNIST dataset. Curves shown are the average of the median over three runs.*

5. Discussion

From Fig. 1 it can be observed that the cost function initially decreases faster with AS or OBS than with the default random sampling, which implies a faster convergence of the network parameters towards their optimum values. The most significant convergence occurs within the first 15 epochs (20 for default random); after that, the convergence rates of all three methods slow down, but by that time both AS and OBS have reached at least one order of magnitude below the cost value achieved by default random. This difference is maintained during the rest of the training process. From Fig. 2 it is observed that the cost-function value with AS diverges for a mini-batch size of 64, and reports a faster convergence for mini-batch sizes of 256 and 1024, showing the same behavior as in Fig. 1 with the Adadelta optimizer.

Comparing AS vs OBS in terms of training epochs (left column in Fig. 1) it can be seen that OBS is faster during the first five iterations (except for batch-size of 1024 in which case the two methods are quite similar) but then it slows down smoothly but dramatically. In contrast, AS does not slow down as significantly as OBS, so that by the tenth epoch AS has managed to overcome OBS in all three test scenarios and then maintains this advantage for the rest of the training. Thus, AS appears to be robust to the mini-batch size in combination with Adadelta and it tends to improve the asymptotic convergence over long periods of training.

One explanation for this behavior is found in the relationship between loss-values and gradient-norms. Katharopoulos & Fleuret have shown that small loss-values imply small gradients [11] and they also demonstrated that relatively large loss-values are not well correlated with gradient-norms [3]. Therefore, sampling strategies such as AS and OBS are more advantageous in the beginning of the training, when the losses exhibit large variance and the methods can be highly selective in order to minimize the gradient variance. When the loss-values get smaller as training progresses, the correlation between these and gradient-norms becomes higher, but the advantage of the methods is lost, since the variance decreases naturally.

Nevertheless AS with Adam optimizer presents a higher sensitivity to the mini-batch size. We conjecture that this is due to the probability distribution of the bins being concentrated in a few values for a mini-batch of a relatively small size. In future work we plan to investigate the precise causes of said observed behavior.

In terms of the computing time (right column in Fig. 1) a difference can be observed between the different batch-sizes tested: with a batch size of 64 instances, OBS is clearly faster than AS, but as the batch size is increased to 256 instances this advantage is reduced, and for a batch size of 1024 instances, the speed of both methods are practically the same. We conjecture that AS works well with larger batch-sizes because, as more information is available for each of the bins, the corresponding distribution becomes more stable.

It should be noted that independently of the relative differences between sampling methods, training requires more computing time with larger batch-sizes. However, in our experiments the increase is not very significant, and in the literature it has been suggested that increasing the batch-size can reduce the number of required parameter updates during training, consequently reducing training time [12], so at this point we consider this as an open issue to be further explored.

In theory, the selection of the side information of the instances could be any information just as long as it is informative enough to provide separability between the bins (which would typically correspond to the classes of a classification problem). For instance, Joseph et al. [13] have explored some measures of how informative are the instances; one of them is the entropy of the current model at training iteration.

Finally, we found that AS is highly sensitive to the values of the smoothing parameters. For the probability distribution P of the bins, we found that the optimum smoothing parameter δ_{pbin} is high in comparison with the smoothing parameter δ_q of the probability distributions Q^j of the instances. This can be explained because the cardinality of P is much smaller than the cardinality of each Q^j. For instance, on the MNIST dataset $|P| = 10$ because there are 10 classes in the dataset, while $\left|Q^j\right| = N/10$ ($N = 60{,}000$) for each of the j-th distributions. Consequently, the P distribution tends to become unstable due to the accumulation of very few values for each p_j, and a high value of the parameter δ_{pbin} is required to compensate this instability. On the other hand, for each Q^j the opposite happens, and the smoothing parameter δ_q required is smaller.

6. Conclusion and future work

A novel method to sample data in order to form mini batches for the training of DNNs has been described. This method is an improved Adaptive Sampling strategy designed to accelerate the training process, and is based on defining optimal distributions over data bins and instances to achieve a large reduction in the variance of the gradient-norms, which leads to faster convergence of the models. Our proposal was successful when compared against the typically used uniform sampling and also against OBS, which is a similar method in the state of the art. However, this paper represents only an initial study of AS schema for DL models. More advanced approaches might be explored in order to tackle harder classification problems such as those represented by the CIFAR-10 or CIFAR-100 datasets. In addition, some variations of the proposed sampling strategy can be easily implemented, such as switching between sampling w.r.t. the loss and the gradient-norm at different points during the training. These ideas are being considered for future work.

Acknowledgment.- This work was partially supported by the National Council of Science and Technology (CONACYT) of Mexico, through Postgraduate Scholarship: 747189 (J. Ávalos) and Research Grants: CÁTEDRAS -2598 (A. Rojas) and CÁTEDRAS-7795 (S.I. Valdez).

References

1. Alom, MZ et al. (2018) *The history began from Alexnet: A comprehensive survey on deep learning approaches*. arXiv:1803.01164.

2. Wang, L et al. (2018) *Superneurons: Dynamic GPU memory management for training deep neural networks*. 23rd ACM SIGPLAN symposium on principles and practice of parallel programming, pp. 41-53.

3. Katharopoulos, A, and Fleuret, F (2018) *Not all samples are created equal: Deep learning with importance sampling*. In International conference on machine learning, PMLR, pp. 2525-2534.

4. Loshchilov, I and Hutter, F (2015) *Online batch selection for faster training of neural networks*. arXiv:1511.06343.

5. Gopal, S (2016) *Adaptive sampling for SGD by exploiting side information*. In International Conference on Machine Learning, PMLR, pp. 364-372.

6. Fan, Y, Tian, F, Qin, T, Bian, J, & Liu, TY (2017) *Learning what data to learn*. arXiv:1702.08635.

7. Alain, G, Lamb, A, Sankar, C, Courville, A, and Bengio, Y (2015) *Variance reduction in SGD by distributed importance sampling*. arXiv:1511.06481.

8. Zhao, P, and Zhang, T (2015) *Stochastic optimization with importance sampling for regularized loss minimization*. In international conference on machine learning, PMLR, pp. 1-9.

9. Needell, D, Srebro, N, and Ward, R (2013) *Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm*. arXiv:1310.5715.

10. Schaul, T, Quan, J, Antonoglou, I, and Silver, D (2015) *Prioritized experience replay*. arXiv:1511.05952

11. Katharopoulos, A, and Fleuret, F (2017) *Biased importance sampling for deep neural network training*. arXiv:1706.00043.

12. Smith, SL, Kindermans, PJ, Ying, C, and Le, QV (2017) *Don't decay the learning rate, increase the batch size*. arXiv:1711.00489.

13. Joseph, KJ, Singh, K, and Balasubramanian, VN (2019) *Submodular batch selection for training deep neural networks*. arXiv:1906.08771.

14. Zhao, P, and Zhang, T (2014) *Accelerating minibatch stochastic gradient descent using stratified sampling*. arXiv:1405.3080.

15. Wu, C Y, Manmatha, R, Smola, A J, and Krahenbuhl, P (2017) *Sampling matters in deep embedding learning*. In Proceedings of the IEEE International Conference on Computer Vision, pp. 2840-2848.

16. Bouchard, G, Trouillon, T, Perez, J, and Gaidon, A (2015) *Online learning to sample*. arXiv:1506.09016.

17. LeCun, Y, Bottou, L, Bengio, Y, and Haffner, P (1998) *Gradient-based learning applied to document recognition*. Proceedings of the IEEE, 86(11), pp. 2278-2324.

18. Zeiler, MD (2012). *Adadelta: an adaptive learning rate method*. arXiv:1212.5701.19.

19. He K, Zhang X, Ren S and Sun J (2016) *Deep Residual Learning for Image Recognition*. Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778.

20. Kingma, D P and Ba, J (2014). Adam: A method for stochastic optimization. arXiv:1412.6980.