Universidad Politécnica de Cartagena

Evaluación de un nuevo método de diseño de máquinas de aprendizaje profundas

Proyecto Fin de Grado

Autor: José Antonio Toral López
Director: José Luis Sancho Gómez

Cartagena, mayo de 2021

Agradecimientos

Quisiera aprovechar este apartado para agradecer a todas esas personas que han aportado a lo largo de mi vida universitaria su granito de arena. En primer lugar, le doy gracias a mi director de TFG, José Luis Sancho Gómez, por ofrecerme realizar este trabajo con él. Creo que la sonrisa se me escapaba mientras me explicaba, en aquella primera tutoría, qué era un espacio de características latente y en qué iba a consistir este TFG demuestra la ilusión que me hizo la propuesta.

En segundo lugar, quiero dar la gracias a mis compañeros de carrera con los que tantas horas he pasado trabajando codo con codo pero también viviendo muy buenas experiencias. Me quedo con todas esas risas que, sin duda, nunca olvidaré.

Por último, pero no menos importante, dar las gracias a mis padres, a mi hermana y a todos aquellos que me han apoyado en los momentos más difíciles, que me han inspirado a dar más de mí y que me han aleccionado cuando ha sido necesario. La universidad, a parte de formarme como profesional, me ha permitido conocer a un gran número de personas maravillosas que espero seguir teniendo cerca.

¡Muchas gracias!

Introducción

En el presente trabajo, se plantean dos nuevas alternativas del algoritmo Modified Stack Denoising Autoencoders para la construcción de arquitecturas profundas con el objetivo de mejorar su desempeño. Para llegar a la conclusión final, se parte de los inicios históricos del mundo de la Inteligencia Artificial, del Aprendizaje Máquina así como de los numerosos hitos que han ido aconteciendo a lo largo de la historia. En el segundo capítulo se explica el funcionamiento de la arquitectura fundamental en la que se basa este trabajo, el autocodificador (en inglés, *autoencoder*) así como distintos tipos de éstos. Por último, en el capítulo 3, se presentan las dos nuevas propuestas, los experimentos realizados y su evaluación, concluyendo finalmente con un análisis de sus resultados.

Índice general

Capítulo 1

Aprendizaje Automático y Profundo

1.1. ¿Qué es el Aprendizaje Automático?

El **aprendizaje automático** o **aprendizaje máquina** (del inglés, *machine learning*) es la rama de la computación que estudia las técnicas que permiten que las máquinas aprendan. Se dice que una máquina ha aprendido a realizar una tarea cuando su desempeño en dicha tarea ha mejorado a través de la experiencia. Dicho de otro modo, estos algoritmos no se programan de manera explicita, si no que aprenden a resolver el problema por sí mismos minimizando el error cometido a lo largo del tiempo.

Para obtener esta experiencia, los algoritmos de *machine learning* deben interactuar con datos o con su entorno.

Según la forma en que lo hacen, el aprendizaje se puede clasificar en las siguientes categorías

- **Aprendizaje supervisado**. En esta categoría se genera un modelo predictivo que aprende a través de los datos de entrada y salida. Estos datos se encuentran etiquetados, es decir, para cada uno de los ejemplos de entrada tenemos el resultado que queremos obtener a la salida. De esta manera, el modelo genera una predicción a su salida a partir de los datos de entrada. Esta predicción se compara con el valor de salida real y aprende en función del error cometido según una métrica preestablecida. Algunos de los algoritmos que podemos implementar bajo este método de aprendizaje son las máquinas de vectores de soporte, árboles de decisión, numerosos tipos de regresiones o las conocidas redes neuronales artificiales, entre otros.

- **Aprendizaje no supervisado** Se diferencia del supervisado en que este tipo de aprendizaje sólo aprenden a partir de los datos de entrada, sin disponer de los datos etiquetados. Se emplean para, entre otras cosas, realizar tareas de clustering, ya que agrupan los datos que poseen características comunes entre sí. Destacan los algoritmos de k-medias, algoritmos de detección de anomalías y, dentro del *'Deep Learning'*, los Autocodificadores *(Autoencoders)*, de los cuales se hablará más tarde.

- **Aprendizaje por refuerzo**. Son modelos basados en la psicología conductista humana. En este tipo de aprendizaje, un agente inteligente interactuará con un entorno, donde desarrollará un conjunto de acciones para realizar una tarea. Una vez medido el desempeño del agente inteligente, sus acciones serán recompensadas o castigadas. El objetivo del agente es maximizar el valor de la función de recompensa. Destacan los algoritmos de Monte Carlo, Q-Learning o DQN entre otros.

1.2. Perspectiva histórica

Pese a que las primeras referencias académicas del Machine Learning datan de mediados del siglo XX, no ha sido hasta principios del siglo XXI, cuando la capacidad de cálculo de los ordenadores lo ha permitidio, donde hemos podido observar de nuevo su potencial. Y es que el desarrollo de esta tecnología ha sido muy irregular, ya que se han vivido ciclos de sobreoptimismo donde se esperaba que la solución a ciertos problemas fuera inminente con otros en los que los avances y la inversión eran prácticamente nulos. Se conocen a estos intervalos de bajo desarrollo tecnológico como los *inviernos de la Inteligencia Artificial*.

A continuación se mencionan brevemente algunos de los principales descubrimientos en el ámbito del Machine Learning del siglo XX.

La primera referencia a la idea de *Machine Learning*, surge en 1947 del matemático inglés Alan Turing al decir en una conferencia: "Lo que queremos es una máquina que pueda aprender de la experiencia". A lo largo de su carrera, Alan Turing trabaja en números conceptos que formarán la base de un nuevo campo, la Inteligencia Artificial.

Figura 1.1: De izquieda a derecha y de arriba a abajo. Alan Turing, Marvin Misnky, Seumour Papert, Frank Rosenblat, Arthur Samuel y Seppo Linnainmaa.

En 1951, Marvin Minsky y Dean Edmonds contruyen la primera máquina que hace uso de una red neuronal artificial, SNARC *(Stochastic Neural-Analog Reinforcement Calculator)*. Se trataba de un dispositivo que emulaba una red neuronal con 40 neuronas artificiales mediante el uso de condensadores y otros componentes electrónicos.

En 1957, Frank Rosenblatt crea uno de los pilares fundamentales de las redes neuronales, el perceptrón, un algoritmo de aprendizaje supervisado para clasificadores binarios. Aunque en un principio el perceptrón parecía prometedor también quedaron patentes rápidamente sus limitaciones y es que los preceptrones de una sola capa solo son capaces de aprender patrones linealmente separables. Este hecho quedó constatado en 1969 por Marvin Minsky y Seymour Papert en su libro llamado *Perceptrons*[4] donde demostraron que un perceptrón sería incapaz de aprender la función lógica XOR, ya que para para separar sus posibles salidas correctamente sería necesaria una frontera de decisión no lineal, cuando el perceptrón monocopa solamente es capaz de generar una frontera lineal. Aunque este inconveniente puede ser solucionado mediante la adición de más capas, éstas presentaban problemas a la hora de entrenarse. Todos estos problemas provocaron un desánimo a la hora de

financiar nuevos proyectos dando lugar a uno de los inviernos de la IA. El Perceptrón, tanto como sus fundamentos, arquitecturas y limitaciones serán explicados con más detalle en el próximo apartado.

Volviendo atrás, en 1959, Arthur Samuel publica *"Some Studies of Machine Learning Using the Game of Checkers"*, con el cual popularizó el término *Machine Learning*. En este artículo, se presenta un programa de ordenador capaz de jugar a las damas y que aprende de la experiencia. Es considerado el primer programa que aplica exitosamente el *machine learning*.

Por último, aunque la lista es mucho más larga, cabe destacar el descubrimiento de Seppo Linnainmaa que en 1970 publica un método general para la diferenciación automática. Este descubrimiento fue fundamental para el posterior desarrollo del algoritmo de retropropagación (del inglés *backpropagation*) que permitía el entrenamiento de máquinas de aprendizaje profundo por descenso del gradiente.

Ya en el siglo XXI, destaca el uso de la red profunda AlexNet [1] para resolver en el año 2012, con una tasa de error de solamente del 16 %, la tarea de clasificación ImageNet, un conjunto de datos con más de 14 millones de imágenes etiquetadas en más de 20.000 clases diferentes. Este evento demostró que era posible entrenar redes profundas para tareas de clasificación complejas, dando lugar al nuevo despertar de la Inteligencia Artificial que estamos viviendo.

1.3. El perceptrón

1.3.1. El perceptrón monocopa

El perceptrón monocapa es un algoritmo de aprendizaje supervisado para ser usado originalmente en problemas de clasificación binaria y como se ha explicado en el apartado 1.2 fue uno de los pilares fundamentales de la historia del *Machine Learning*. Se trata de un algoritmo bioinspirado en las neuronas del reino animal. Aunque las neuronas se pueden clasificar según varios criterios y hay numerosos grupos, generalmente constan de un núcleo o cuerpo celular del cual surgen unas ramificaciones. Estas ramificaciones son las dendritas (que actúan como entrada de la neurona) y el axón (que actúa como salida). Las neuronas funcionan, de manera muy simplificada, recibiendo por sus entradas un conjunto de señales electroquímicas. Estas señales son procesadas en el cuerpo celular y producirá o no, una respuesta que será transmitida a través del axón para generar una señal electroquímica de salida que será enviada a otra neurona. De esta forma, varias neuronas forman una red neuronal.

De igual manera el perceptrón monocapa constituye el tipo más sencillo posible

de red neuronal artificial y es usada para resolver la clasificación de un conjunto específico de patrones, aquellos que sean linealmente separables (es decir, patrones que se encuentran a ambos lados de un hiperplano).

Para encontrar a qué clase pertenece un objeto, es decir, ser capaz de crear una regla que permita separar un conjunto de objetos en dos conjuntos más pequeños, el perceptrón toma una decisión de clasificación basada en un valor procedente de una combinación lineal de las características de entrada del objeto.

El esquema de una neurona artificial es:

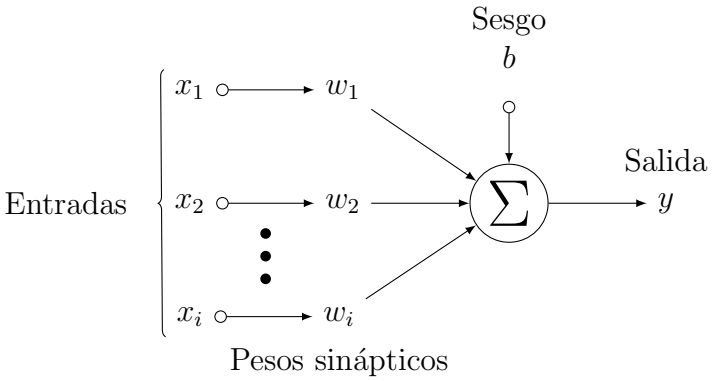


Figura 1.2: Representación gráfica de un perceptrón de una sola neurona

Donde:

- x_n son cada una de las entradas de la neurona.
- w_n es el parámetro llamado peso (*weight*) de la conexión entre la n-ésima entrada y la neurona.
- b es el parámetro de sesgo (*bias*). Este se caracteriza por ser independiente al conjunto de parámetros de entrada. Otra forma de expresarlo es como un peso sináptico más, w_0, que multiplica a una entrada extra con valor fijo 1.

  El conjunto de los pesos sinápticos W, más el sesgo b conforman el conjunto de parámetros entrenables θ.
- y es el valor de salida de la neurona.

El valor de esta salida y será:

$$f(x) = \begin{cases} 1 & \text{si } \sum_{n=1}^{i} x_n w_n > -b \\ 0 & \text{Otro caso} \end{cases}$$

Por lo tanto, la tarea de aprendizaje del perceptrón se resume en ajustar los parámetros que modelan el hiperplano para que separe correctamente los dos clases. Dicho de otro modo, el correcto desempeño de la tarea de clasificación lineal, se consigue con el ajuste del conjunto de parámetros θ de los cuales depende la forma de la frontera de decisión.

Este hecho, fue demostrado por Rosenblat en 1958, quien aseguró que si los patrones usados para entrenar un perceptrón son sacados de dos clases linealmente separables, el aprendizaje del perceptrón converge. Esto es conocido como el *teorema de convergencia del perceptrón*[2].

Teorema de convergencia del perceptrón. *Si el conjunto de patrones de entrenamiento*

$$\{x^1, z^1\}, \{x^2, z^2\}, ..., \{x^p, z^p\}$$

es linealmente separable entonces el perceptrón simple encuentra una solución en una número finito de iteraciones, es decir, consigue que la salida de la red coincida con la salida deseada para cada uno de los patrones de entrenamiento.

Este ajuste de parámetros θ se puede llevar a cabo mediante numerosos métodos numéricos, pero actualmente el más usado es el descenso por gradiente.

El descenso por gradiente (en inglés, *gradient descent*) es un algoritmo de optimización iterativo que permite encontrar el mínimo local de una función diferenciable. La idea del descenso del gradiente es evaluar el desempeño del sistema a partir de la función de coste y posteriormente calcular su gradiente (que indica la dirección de máximo crecimiento de la función de coste). El cálculo del gradiente se usa para ir en sentido contrario y llegar así al mínimo local. Existe un parámetro, llamado tasa de aprendizaje (en inglés, *learning rate*) que tiene una gran importancia en el gradient descent. La tasa de aprendizaje es un factor que multiplica al gradiente e indica la velocidad con la que se llevará a cabo el aprendizaje. Así, valores bajos en la tasa de aprendizaje provocarán que el aprendizaje demore mucho tiempo o incluso se pare y valores altos pueden evitar que se llegue al mínimo local.

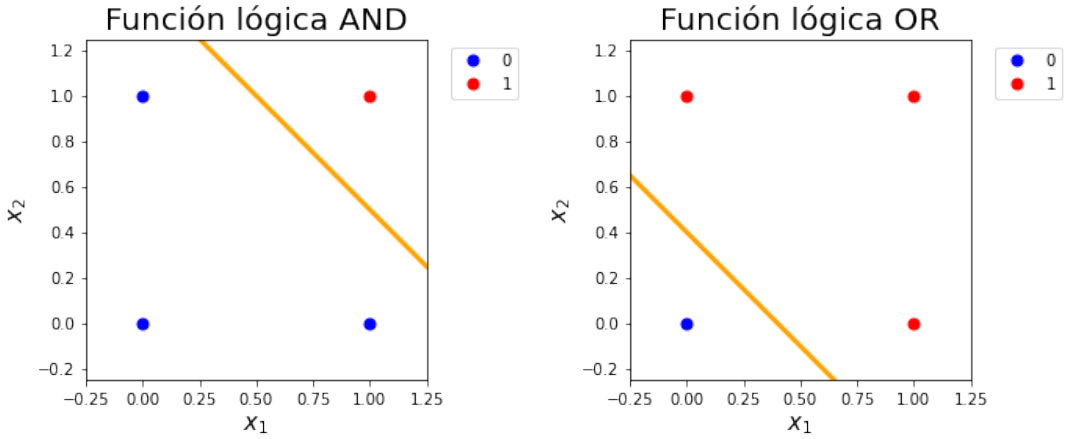


Figura 1.3: Clasificación lineal para los problemas AND y OR

La expresión general para la actualización de los parámetros entrenables θ es:

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} \mathrm{J}(\theta) \tag{1.1}$$

- θ_j es el parámetro j-ésimo perteneciente al conjunto de parámetros θ.
- α es la tasa de aprendizaje (en inglés, *learning rate*).
- $\frac{\partial}{\partial \theta_j} \mathrm{J}(\theta)$ es la derivada parcial de la función de coste respecto al elemento θ_j o en otras palabras como varía la función de costes $J(\theta)$ respecto al parámetro θ_j.

De esta forma, gracias a los clasificadores lineales podemos resolver una gran cantidad de problemas. Dos problemas clásicos resolubles con perceptrones monocapa son los problemas AND y OR cuya resolución con un clasificador lineal se puede observar en la Figura 1.3.

1.3.2. Problemas no-linealmente resolubles.

Pese al gran número de problemas resolubles con este tipo de arquitectura, la limitación de tener una frontera de decisión lineal restringe mucho el número de problemas que podemos abordar. Un ejemplo típico de la limitación de los perceptrones monocapa es el problema de la función lógica XOR o el de las dos medias lunas. Tal

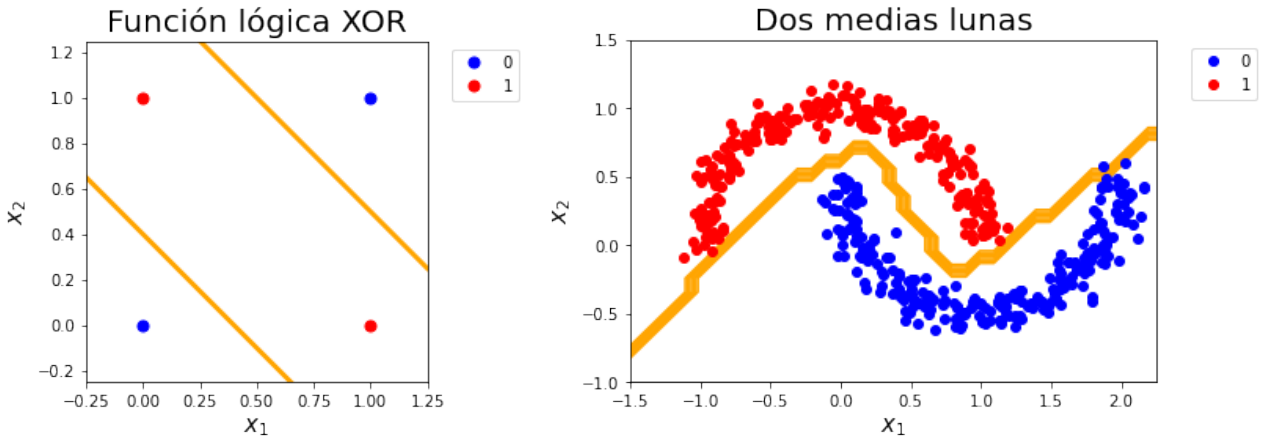


Figura 1.4: Ejemplos de problemas no resolubles para un perceptrón monocapa

y como se puede observar en la Figura 1.4 ambos problemas no pueden resolverse con un perceptrón monocopa o lo que es lo mismo, con una única frontera de decisión lineal.

Para solucionar ambos problemas (y en general para resolver problemas no linealmente separables) la solución es añadir varias neuronas para formar una red neuronal. Y es que al igual que en las neuronas biológicas, la salida de las neuronas artificiales pueden actuar como la entrada de otra permitiendo generar un aprendizaje jerarquizado. Pero introducir la salida de una neurona como entrada de otra presenta un problema y es que, como hemos visto, la salida de un perceptrón es una frontera de decisión lineal e introducir una expresión lineal dentro de otra da lugar a otra función lineal como puede observarse a continuación.

$$
\begin{aligned}
y^{(1)} &= x\omega_1 + b_1 \\
y^{(2)} &= y^{(1)}\omega_2 + b_2 \\
y^{(2)} &= (x\omega_1^1 + b^1)\omega_1^2 + b^2 \\
y^{(2)} &= x\omega_1\omega_2 + b_1\omega_2 + b_2
\end{aligned}
$$

Y por último si agrupamos los términos que dependen de la entrada x de los que no, obtenemos nuevamente una expresión lineal:

$$y^{(2)} = x\omega' + b'$$

Dado que concatenar neuronas por si solo no crea fronteras de decisión no lineales, es necesario introducir un nuevo elemento en las neuronas, la función de activación.

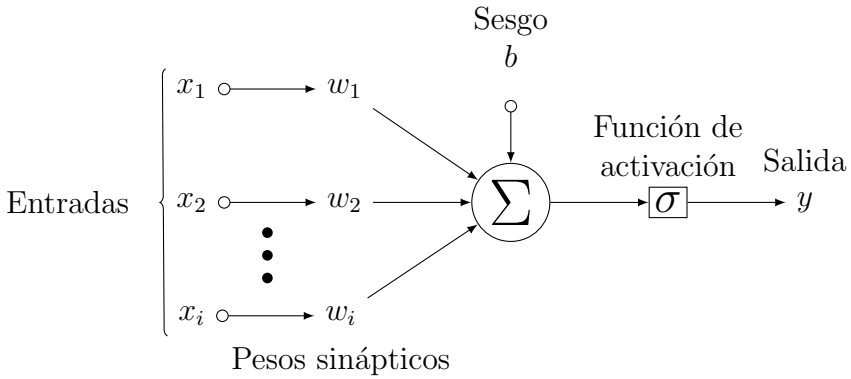


Figura 1.5: Perceptrón monocapa con función de activación

Al aplicar estas funciones a la salida de una neurona se consigue dotar a esta de un comportamiento no lineal y limitar el valor de salida, generalmente, en un rango determinado como (0,1) o (-1,1). Se buscan funciones con derivadas simples, para minimizar el coste computacional.

Con este último añadido, se puede observar la estructura del perceptrón monocapa completo en la Figura 1.5.

Algunas de las funciones de activación más usadas son.

- Sigmoide: transforma los valores de salida de la neurona a una escala (0, 1). Tiene una convergencia lenta debido a que sus asíntotas horizontales anulan el gradiente. No está centrada en 0. Suele usarse en la última capa para problemas de clasificación binaria.

$$f(x) = \frac{1}{1 + e^{-}x} \tag{1.2}$$

- Tangente hiperbólica: transforma los valores de salida a una escala (-1, 1). Presenta los mismos problemas de convergencia y gradiente que la función sigmoide. Está centrada en 0.

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{1.3}$$

- ReLU (del inglés, Rectified Lineal Unit): anula los valores negativos y deja los positivos tal y como están. No está acotada en un rango de valores. Muy

utilizada en aplicaciones de aprendizaje profundo debido a su rápido entrenamiento. Posee un inconveniente, si su entrada es menor a 0, los parámetros de esta neurona no se actualizarán durante el entrenamiento.

$$f(x) = max(0, x) = \begin{cases} 0 \text{ si } x < 0 \\ \text{x si } x >= 0 \end{cases} \tag{1.4}$$

- Leaky-ReLU: igual que ReLU, pero aplicando una correción a los valores negativos para evitar la muerte de la neurona.

- Softmax: transforma las salidas a una representación en forma de probabilidad, de forma que el sumatorio de todas las salidas da 1. Se utiliza en la última capa de los clasificadores multiclase cuando solo se desea predecir la pertenencia a una de estas clases.

1.3.3. Perceptrones Multicapa

Una vez se solucionó el problema que impedía generar fronteras de decisión no lineal fue posible agrupar numerosos perceptrones en distintas capas. Así fue como se crearon las primeras redes neuronales artificiales o, en este caso, los perceptrones multicapa o MLP (del inglés, *Multilayer Perceptron*). Y gracias a éstos fue posible resolver problemas mucho más complejos.

Un perceptrón multicapa es una red neuronal que contiene numerosos perceptrones agrupados en capas. Un MLP consta al menos de 3 capas: una capa de entrada, una capa oculta y una capa de salida. Aunque según algunos autores, la capa de entrada no debe de considerarse como tal. El esquema general de un MLP puede observarse en la Figura 1.6 donde:

- a_j^k es la salida de la unidad j de la capa k. La salida de las unidades de la última capas son denotadas como y_j.

- Θ^n es la matriz de pesos de la capa n hasta la capa $n+1$. Si la capa n tiene i unidades y la capa $n+1$ tiene j unidades, entonces Θ^n tiene dimensión $j \times (i+1)$.

A lo largo de la historia de este campo, numerosos científicos han demostrado la eficacia de los MLP actúan como aproximadores universales. Esto fue demostrado con dos teoremas, el teorema de Kolmogorov [5] y el de Cybenko [6].

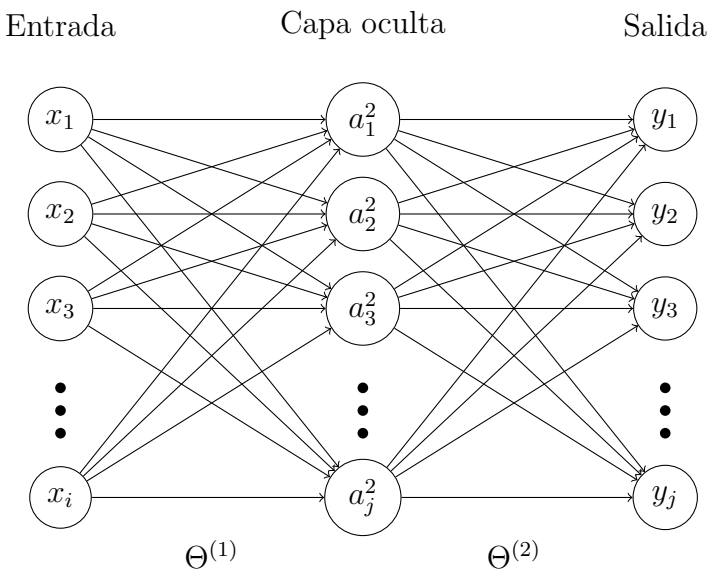


Figura 1.6: Esquema general de un MLP

Teorema de Kolmogorov. *Cualquier función continua real* $f(x_1, x_2, ..., x_n)$ *para* $n \geq 2$ *puede representarse de la forma:*

$$f(x_1, x_2, ..., x_n) = \sum_{j=1}^{2n+1} \sigma_j \left(\sum_{i=1}^{n} \phi_{ij}(x_i) \right) \tag{1.5}$$

donde σ_j *son funciones de activación apropiadamente escogidas y* $\phi_{ij}(x_i)$ *funciones continuas monotonamente crecientes definidas en el intervalo* $[0, 1]$ *e independientes de* f*.*

Teorema de Cybenko. *Basta con una capa oculta de unidades sigmoidales (en número indefinido) para expresar cualquier función continua arbitrariamente bien.*

Como puede observarse, pese a que demuestran la capacidad de los MLP para aproximarse a cualquier función continua, estos teoremas no son constructivos para el dimensionamiento de los MLPs ya que el primer teorema no indica la función de activación a usar y el segundo no indica el número de nodos necesarios.

Pese a que los MLP parecen la solución a todos los problemas, presentan numerosas desventajas. La primera y más significativa es que se han demostrado varios contraejemplos de funciones que no se pueden aprender a través de un MLP de una

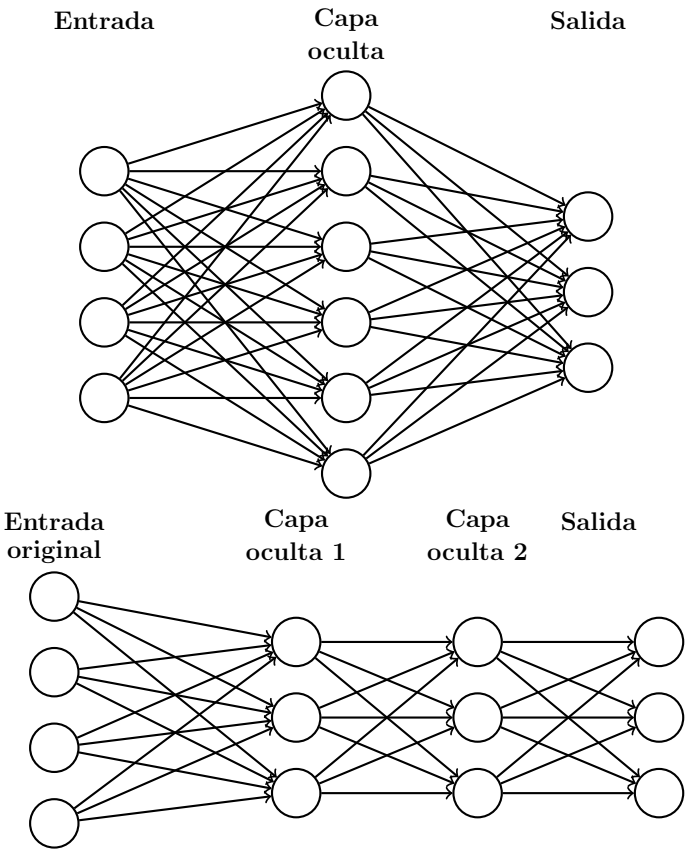


Figura 1.7: Comparación a la hora de usar el mismo número de neuronas en una única capa oculta o en varias. Arriba: un MLP con 4 entradas, 3 salidas y 6 neuronas ocultas en una única capa crean un total de 42 conexiones a computar. Abajo: mismo MLP pero repartiendo las 6 neuronas en 2 capas ocultas, hacen un total de 30 conexiones.

sola capa o que requieren un número infinito de nodos. Este número de nodos arbitrariamente alto tiene un sobrecoste computacional que hace a los MLP de una sola capa muy ineficientes para algunas tareas. La solución para disminuir este coste es repartir el número de nodos en numerosas capas. Este sobrecoste computacional puede observarse fácilmente en la Figura 1.7.

Si suponemos un problema con 4 entradas, 3 salidas y 6 nodos ocultos se podrían hacer, por ejemplo, las dos arquitecturas de la Figura 1.7. En la arquitectura superior observamos un MLP con una única capa con un total de 42 conexiones entre neuronas. Por otro lado, manteniendo el mismo número de nodos ocultos pero repartiéndolos en 2 capas de 3 nodos cada una, observamos que el número de conexiones es 30. Este

ahorro en el coste computacional se ve incrementado al usar arquitecturas de red más complejas. Así, la incertidumbre en el número de nodos necesarios para abordar un problema, la ineficiencia computacional de esta solución y la mejor capacidad de generalización al usar numerosas capas ocultas ha provocado que las arquitecturas de red profundas se posicionen como la opción preferida a la hora de abordar problemas complejos dando lugar al auge del aprendizaje profundo (del inglés, *deep learning*).

1.4. Deep Learning

El **Aprendizaje Profundo** (del inglés, *Deep Learning*) es una subrama de algoritmos de Machine Learning inspirados en las redes neuronales artificiales y éstas a su vez, lo están en las redes neuronales biológicas.

La característica más evidente del aprendizaje profundo es el uso de múltiples capas para extraer de manera progresiva características de alto nivel a partir de una entrada dada. Un ejemplo típico para visualizar este aprendizaje jerarquizado es el de las redes encargadas de tareas de visión artificial. En estas redes las primeras capas aprenden a reconocer características más elementales (bordes, degradados, líneas...) mientras que las últimas capas aprenden a identificar conceptos más complejos (letras, vehículos, caras, animales).

El aprendizaje profundo ofrece numerosas ventajas respecto al *'Shallow Learning'*. La principal de ellas es la de escalar su desempeño fácilmente a la hora de crear modelos más complejos que consuman una mayor cantidad de datos. Además de la escalabilidad, otro beneficio de los modelos de aprendizaje profundo es su capacidad para realizar la extracción automática de características a partir de los datos sin procesar, por eso también es llamado aprendizaje de características. Esta ventaja permite el desarrollo de modelos mucho más robustos, sin la necesidad de llevar a cabo una evaluación previa de características útiles para el aprendizaje. Esto permite realizar una abstracción en problemas complejos, ya que es la misma red la que se encarga de crear conceptos complejos a partir del otros conceptos más simples aprendidos en las capas más cercanas a la entrada. Por último, una consecuencia de esto, es que con la creación de un único modelo complejo pre-entrenado se pueden modificar las últimas capas para dar lugar a otros modelos aprovechando el conocimiento ya adquirido. Esta técnica se llama transferencia del aprendizaje (del inglés, *transfer learning*). Y ha sido de gran de importancia a la hora de abordar problemas complejos sin tener que empezar de cero.

Capítulo 2

Deep Autoencoders

2.1. ¿Qué es un autoencoder?

Un **autocodificador** (del inglés, *autoencoder*) es una red neuronal que es entrenada para intentar crear una copia de su entrada en la salida. Una definición más técnica, es que son redes neuronales diseñadas para codificar la entrada, generalmente, en una versión comprimida y lo suficientemente representativa para luego decodificarla de nuevo y conseguir una reconstrucción lo más semejante posible a la entrada original.

El autoencoder está formado por 2 partes:

- El **codificador** (del inglés, *encoder)* que es una función determinista f_θ que transforma un vector de entrada $\mathbf{x}$ en una representación oculta $\mathbf{y}$. Generalmente la función del *encoder* puede expresarse como:

$$\overline{\mathbf{y}} = f_\theta(\mathbf{x}) = \sigma(W^T\mathbf{x} + \mathbf{b})$$

  donde W^T es la traspuesta de la matriz de pesos de la capa oculta y $\mathbf{b}$ es el vector de sesgos.

- El **decodificador** (del inglés, *decoder)* es el encargado de reconstruir el vector $\overline{\mathbf{x}}$ a partir de la representación oculta $\mathbf{y}$. Al igual que el *encoder*, el funcionamiento del *decoder* puede expresarse como:

$$\overline{\mathbf{x}} = g_{\theta'}(\mathbf{y}) = \sigma(W'^T\mathbf{y} + \mathbf{b}')$$

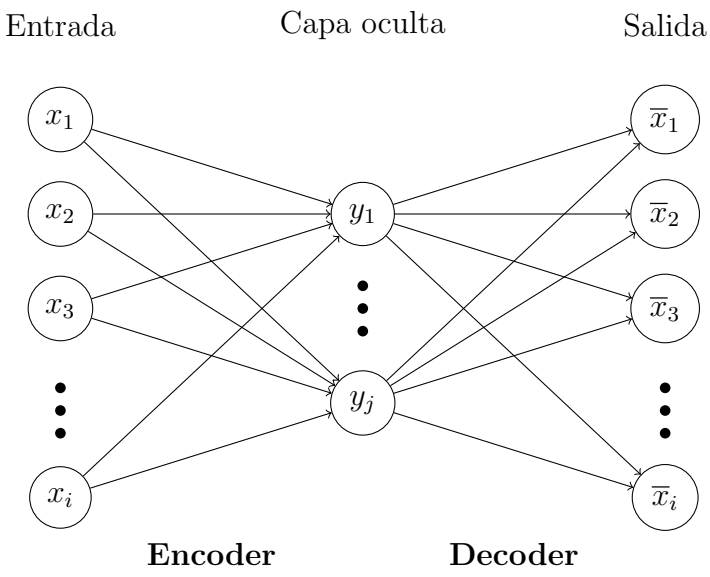


Figura 2.1: Esquema general de un autoencoder.

donde W'^T es la traspuesta de la matriz de pesos de la salida y $\overline{\mathbf{b'}}$ es el vector de sesgos.

Algunas de las principales aplicaciones de los autoencoders son:

- Reducción de la dimensionalidad: esto es posible ya que los autoencoders son obligados a aprender a reconstruir el valor de su entrada a partir de una versión comprimida de esta.
- Compresión de imagen: aunque es posible comprimir imágenes con autoenconders presentan el inconveniente de funcionar solo para los grupos específicos de imágenes con los que fueron entrenados. Incluso logrando un buen desempeño, no suelen destacar sobre los algoritmos clásicos de compresión de imagen.
- Eliminación de ruido: si se dispone de un conjunto de imágenes es sencillo añadir digitalmente cualquier tipo de ruido. Una vez se disponen de las dos versiones de las imágenes (versión original y versión corrompida por ruida) se pueden entrenar un autoencoder para que a partir de la versión ruidosa de una imagen conseguir su original sin perturbar. Este tema será tratado extensamente en la sección 2.2.2.
- Extracción de características: codificar una entrada en un versión comprimida

para luego volver a reconstruirla involucra aprender que características son más útiles para reducir el error de reconstrucción.

- Detección de anomalías. Si se entrena un autoencoder a reproducir con precisión aquellas características más frecuentes, se observará un empeoramiento de su rendimiento al reconstruir cuando se encuentre con datos de entrada que se salgan de la norma.
- Destacan otras tareas como sistemas de reconstrucción, traducción automática, generación de imágenes, etc.

2.2. Tipos de autoencoders

2.2.1. Deep autoencoders

Al igual que se explicó en el apartado 1.4 las ventajas de usar arquitecturas profundas para la construcción de redes neuronales, usar numerosas capas para la creación de autoencoders presenta varios beneficios como la disminución del coste computacional al añadir varias capas frente a usar arquitecturas superficiales, la mejor capacidad de generalización o el poder usar menos datos. Es por esto por lo que la mayoría de autoencoders usan arquitecturas profundas. Puede observarse una visualización de un autoencoder profundo en la Figura 2.2.

2.2.2. Stacked Denoising Autoencoders (SDAE)

Dado que la propuesta presentada en este trabajo se fundamenta en el algoritmo Stacked Denoising Autoencoders (SDAE) [3], se tratará en profundidad aspectos relevantes como son las representaciones útiles, el criterio de supresión de ruido y una explicación paso a paso de la construcción de esta arquitectura.

2.2.2.1. Consideraciones previas

Representaciones útiles

Una primera e intuitiva definición de *"buena representación"* podría ser aquella que permite a un sistema desarrollar un mejor desempeño en una tarea específica (por ejemplo, una clasificación multiclase) que no sería tan alto si no hubiera aprendido previamente esa representación. Usando esta definición podríamos usar como criterio de buena representación el error en la tarea de clasificación que realizará

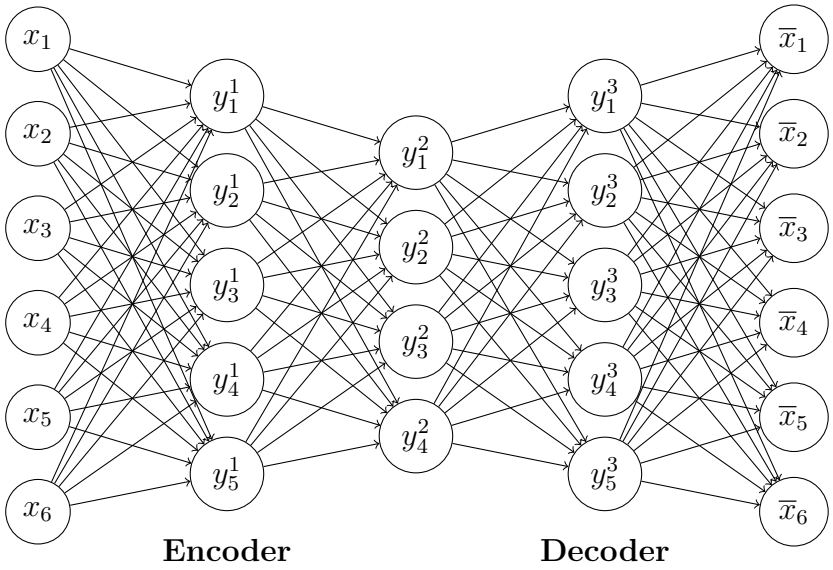


Figura 2.2: Esquema general de un autoencoder profundo

a posteriori un sistema. Sin embargo, se ha demostrado que pre-entrenar una red neuronal bajo un criterio no supervisado puede mejorar el desempeño de la posterior tarea de clasificación.

Retener información de la entrada

Otro criterio para confirmar que una representación es buena, podría ser que dicha representación contenga suficiente información de la entrada. En términos de teoría de la señal consistiría en maximizar la información mutua $\mathbb{I}(X;Y)$ entre la entrada $\mathbf{X}$ y la representación $\mathbf{Y}$.

Tras un extenso desarrollo matemático explicado en [3] se observa que minimizar el error de reconstrucción de un autoencoder equivale a maximizar la información mutua entre la entrada X y la representación aprendida Y.

Criterio de supresión de ruido

El criterio de máxima información mutua entre X e Y no es por sí mismo suficiente para asegurar una buena representación, ya que un autoencoder podría maximizar trivialmente la información mutua forzando que a que Y sea igual a X. Otra posible solución trivial se puede dar si un autoencoder tiene una dimensionalidad Y igual o mayor que X. Ésto provoca una reconstrucción perfecta a partir de la solución

identidad.

Son por todas estas posibles soluciones triviales por lo que el criterio de reconstrucción es insuficiente para garantizar una representación útil y por lo que hay que restringir el problema para producir representaciones verdaderamente útiles, aunque ello conlleve un error de reconstrucción no nulo.

Algunas de las restricciones más usadas tradicionalmente son: introducir cuellos de botella en la arquitectura del autoencoder para obtener una representación Y que sea una versión comprimida e incompleta de X o forzar a las matrices del encoder y decoder a ser simétricas.

También ha surgido gran interés en los últimos años en las representaciones dispersas (del ingles, sparse representations), que son aquellas que tienen solamente un pequeño números de parámetros diferentes a cero.

Por último, en el mismo artículo surge un nuevo criterio, el criterio de supresión de ruido. Dice así:

"Una buena representación es aquella que puede obtenerse de manera robusta de una entrada corrupta y que será útil para recuperar la correspondiente entrada limpia".

2.2.2.2. El algoritmo SDAE

A diferencia de los algoritmos clásicos, donde se define una arquitectura y se entrena en un único entrenamiento bajo un criterio supervisado, los SDAE tienen 2 fases diferentes.

- **Fase de crecimiento del AE**: durante esta fase se obtienen una serie de capas a través de sucesivos entrenamientos no supervisados. Se denotan a cada uno de estos entrenamientos como iteraciones. En cada uno de estas iteraciones se entrena un *autoencoder* para conseguir una representación interna a partir de una entrada con ruido para reconstruir a la salida una versión sin ruido de dicha entrada. En cada una de estas iteraciones se conseguirá una única capa que será usada posteriormente en la tarea de clasificación.

- **Fase de entrenamiento final**: en esta fase se apilan todas las capas obtenidas en la fase de crecimiento dando lugar a una arquitectura profunda. Es esta arquitectura la que será entrenada en el problema de clasificación deseado.

En cada una de las iteraciones de la fase de crecimiento se empieza por añadir un MLP con una capa oculta a la arquitectura de red ya existente. Se corrompe únicamente, la entrada del MLP añadido y es entrenado para reconstruir a su salida una

versión limpia de los datos de entrada. En cada una de las iteraciones son entrenadas solamente las capas del último MLP añadido, dejando los pesos del resto de capas congelados. Tras este entrenamiento se almacena la primera capa del MLP entrenado mientras que se descarta la segunda y a la arquitectura resultante se le añade un nuevo MLP. El anterior proceso de entrenamiento es repetido hasta alcanzar la profundidad deseada. Finalmente, se entrenan todas las capas resultantes para una tarea específica de clasificación.

A continuación se explican las primeras iteraciones, del algoritmo para un supuesto problema de clasificación con 6 parámetros de entrada, 5 posibles clase de salida y que será resuelto con un MLP de 3 capas ocultas. Este ejemplo volverá a ser usado posteriormente para explicar las propuestas planteadas en este trabajo.

Primera iteración de la fase de crecimiento.

- Se parte de un MLP con una única capa oculta.
- Se corrompe la entrada x mediante la adición de ruido. En concreto, en este trabajo se usará ruido térmico gaussiano con media cero y se alternará con cuatro valores de varianza diferentes (v = 0, 0.05, 0.2 y 0.4). Añadiendo este ruido se consigue la versión corrupta de la entrada, que será denotada como $\tilde{x}$. En la Figura 2.3 puede observarse como afecta este ruido a dos conjuntos de datos que se han usado para la obtención de los resultados.

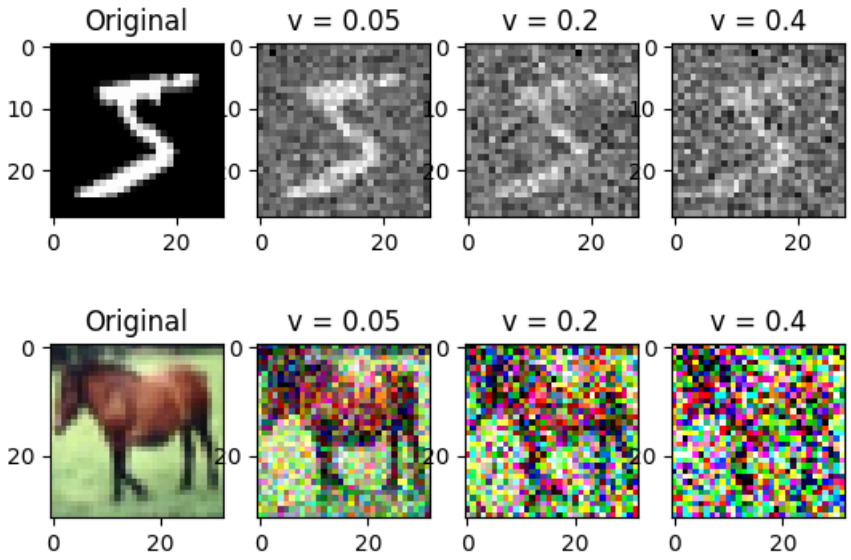


Figura 2.3: Visualización del ruido aplicado sobre ejemplos de los conjunto de datos MNIST y CIFAR-10.

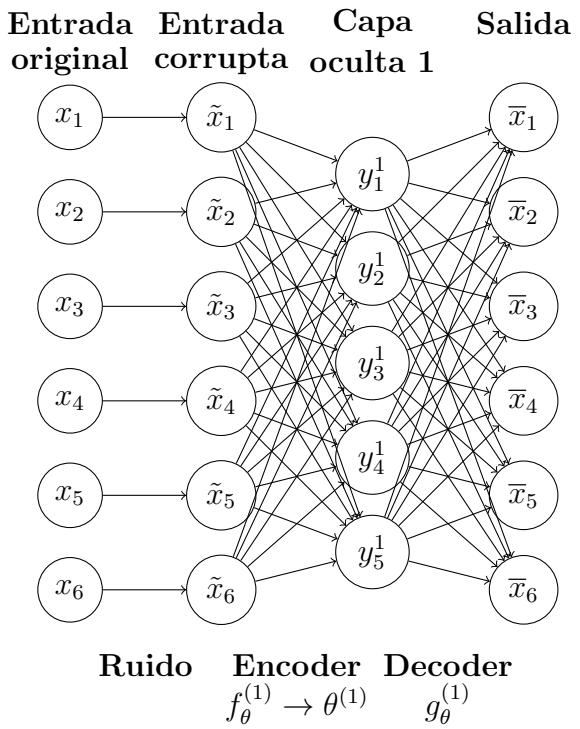


Figura 2.4: Primera iteración del algoritmo SDAE.

- Durante el entrenamiento, la primera capa del MLP, a la que denominaremos encoder, se encarga de generar una representación $y^{(1)}$ a partir de $\tilde{\mathbf{x}}$. Esto es: $y^{(1)} = f_\theta^{(1)}(\tilde{x})$. La segunda capa o decoder usa la representación $y^{(1)}$ para obtener una reconstrucción $\overline{x}$ lo más semejante posible a la entrada $\mathbf{x}$, es decir, la función del decoder es $\overline{x} = g_{\theta'}^{(1)}(y)$. El error de reconstrucción se mide como $L_H(x, \overline{x})$.
  Del entrenamiento obtendremos dos conjuntos de parámetros entrenados, correspondientes a cada una de las capas de este MLP. Estos conjuntos son $\theta^{(1)}$ y $\theta'^{(1)}$ obtenidos del encoder y decoder respectivamente. Se almacenan los parámetros de la primera capa entrenada y se descartan los de la segunda.

  Esta iteración se puede observar de manera gráfica en la Figura 2.4.

  Si se desea continuar y construir una arquitectura de red profunda, los siguientes pasos son:

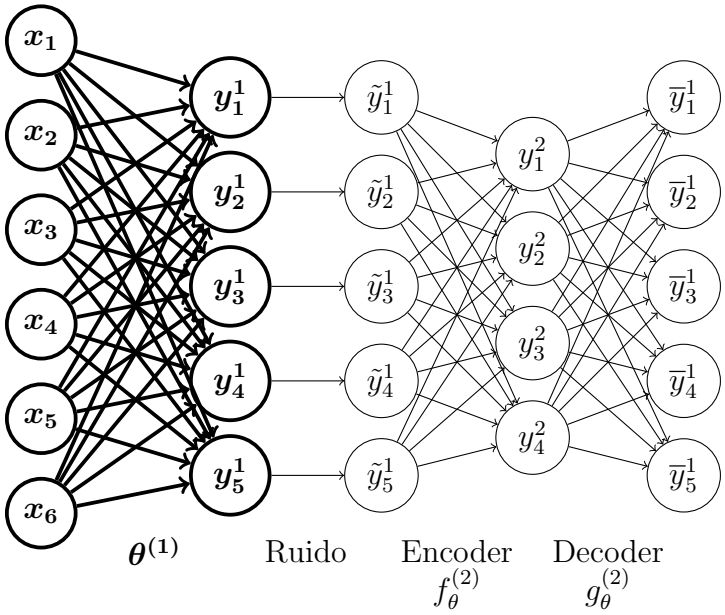


Figura 2.5: Segunda iteración del algoritmo SDAE.

- Se parte de la entrada original $\mathbf{X}$.
- Se añade la capa obtenida en la primera iteración, es decir, la capa correspondiente el encoder $f_\theta^{(1)}$ con sus respectivos parámetros $\theta^{(1)}$. El conjunto $\theta^{(1)}$ será *congelado*, es decir, no se actualizará sus valores durante las próximas iteraciones. Las capas con parámetros *congelados* serán dibujadas con un mayor grosor en las representaciones gráficas.
- Se corrompe la salida de la representación $y^{(1)}$, con el mismo tipo de ruido que corrompió la entrada en la iteración 1, generando así una representación corrupta $\tilde{y}^{(1)}$.
- Se añade un nuevo MLP con un capa oculta.
- El autoencoder intenta reconstruir la representación $y^{(1)}$ a partir de la nueva representación $y^{(2)}$. El error de reconstrucción se mide como $L_H(y^{(1)}, \overline{y}^{(1)})$.

La segunda iteración del algoritmo se puede observar en la Figura 2.5.
En esta última iteración se han obtenido los parámetros $\theta^{(2)}$ pertenecientes al encoder $f_\theta^{(2)}$.

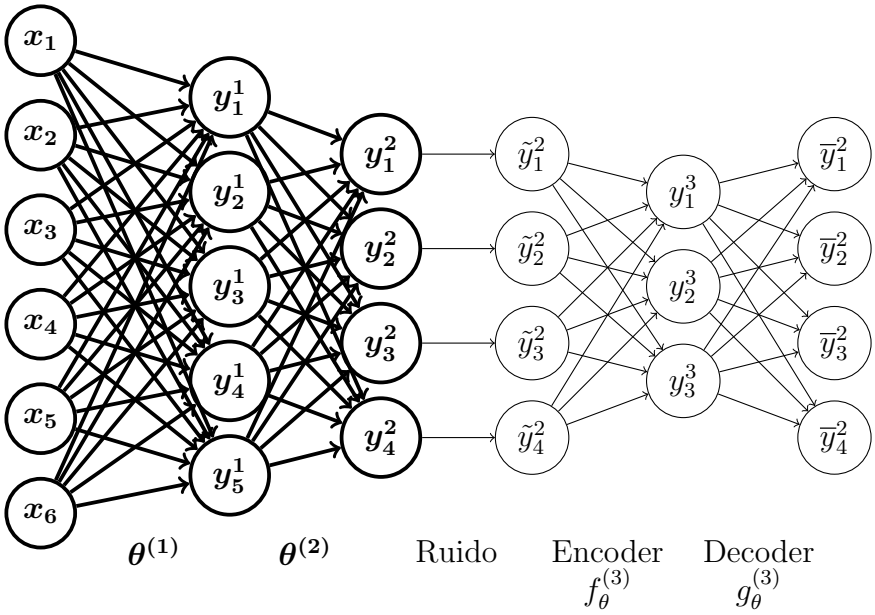


Figura 2.6: Tercera iteración del algoritmo SDAE.

Para la tercera iteración del algoritmo, se añade la nueva capa obtenida en la iteración 2, se asignan sus parámetros $\theta_{(2)}$ y se congelan. Por último se corrompe la salida de la última capa y se entrena.

La representación gráfica de esta tercera iteración puede observarse en la figura 2.6.

Una vez conseguida la profundidad deseada se puede llevar a cabo el entrenamiento de clasificación bajo un criterio supervisado. Para ello se apilan todos los *encoders* que se han ido obteniendo a lo largo de las distintas iteraciones. Se asignan a cada una de las capas sus respectivos parámetros $\theta^{(n)}$ que de cara al entrenamiento final de clasificación se permitirá que se actualicen. Por último, se añade una capa de salida acorde al problema de clasificación a resolver (clasificación binaria, multiclase, etc...) Siguiendo con el ejemplo, y suponiendo una profundidad de 3 capas ocultas y un problema de clasificación con 5 clases podemos observar la arquitectura final en la Figura 2.7.

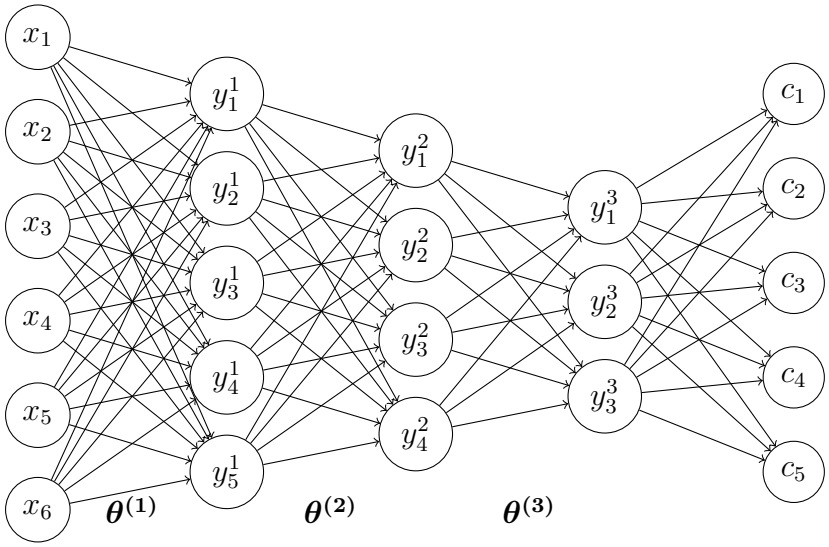


Figura 2.7: Arquitectura final del SDAE.

2.2.3. Modified SDAE (MSDAE)

Para medir el desempeño de la propuesta se comparará con una versión modificada del algoritmo SDAE explicado en el Punto 2.2.2. La modificación introducida consiste en la reconstrucción que genera el decoder. En cada iteración del entrenamiento del SDAE se reconstruía la última representación interna y a la que se le añadía ruido. Sin embargo, este MSDAE intentará reconstruir en cada una de estas iteraciones la entrada original x. Así, el error de reconstrucción será siempre $L_H(x, \overline{x})$ en vez de $L_H(y^{(n)}, \overline{y}^{(n)})$

2.2.4. Complete MSDAE (CMSDAE)

En el artículo *Exploiting label information to improve auto-encoding based classifiers* [7] se propone un método para incrementar el desempeño de los MSDAE. Éste consiste en realizar una preclasificación de la entrada y añadir esta predicción como un parámetro de entrada extra del MSDAE. El autoencoder trabaja con la información extra guiando el aprendizaje haciendo que en cada iteración (salvo en el entrenamiento final de clasificación) se consiga una reconstrucción de la entrada libre de ruido de la entrada y la clase correspondiente. De esta forma se consigue una mejorar en la tarea final de clasificación ya que la extracción de características útiles está parcialmente controlada por las etiquetas de las clases. Este tipo de ar-

quitectura, será la referencia con la que se compararán las 2 nuevas propuestas. A continuación, en la figura 2.8 se puede observar la arquitectura del CMSDAE.

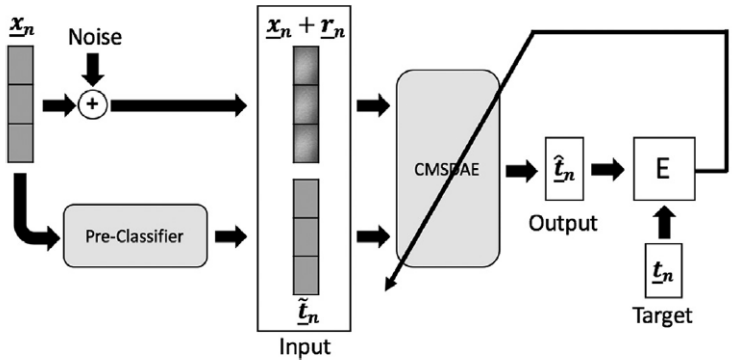


Figura 2.8: Representación de la arquitectura CMSDAE.

Capítulo 3

Propuesta

En el anterior punto se pudo observar la construcción de un SDAE paso a paso. Pese a haber demostrado un mejor desempeño en tareas de clasificación respecto a los métodos clásicos de entrenamiento (en los que todas las capas se entrenan en una única iteración) los SDAE presentan un inconveniente. Éste es que al entrenar un MLP con una única capa oculta, la frontera de decisión que se creará en el espacio latente será lineal. Para solucionar esto, se proponen dos alternativas que consisten en añadir en cada entrenamiento un MLP con dos capas ocultas. El objetivo es conseguir una frontera de decisión no lineal que se espera que dote al modelo de una mayor capacidad de generalización.

3.1. Propuesta 1

En esta primera propuesta se plantea un algoritmo de crecimiento del autoencoder semejante al de MSDAE pero entrenando en cada iteración un MLP de dos capas ocultas en vez de una. Estas dos capas ocultas se encargarán de la tarea de codificación y la salida de la tarea de decodificación y reconstrucción de la entrada. Tras haber entrenado este MLP como autoencoder se almacena la información de la primera capa oculta.

La primera propuesta consta de los siguientes pasos

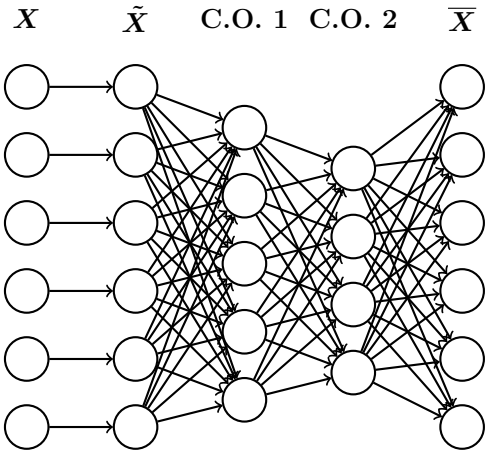
- Se parte de un MLP con dos capas ocultas.
- Se corrompe la entrada x mediante la adición de ruido.

- De este MLP con dos capas ocultas, una vez vez se haya entrenado se consiguen 3 conjuntos de parámetros en vez de los dos que obteníamos en el MSDAE. Denominaremos a estos parámetros $\theta_1^{(1)}$, $\theta_1^{(2)}$ y $\theta_1^{(3)}$ ya que pertenecen respectivamente a las capas 1, 2 y 3 y se han obtenido en la iteración número 1.
- Se almacenan únicamente los parámetros $\theta_1^{(1)}$ correspondientes a la primera capa del MLP y se desecha el resto.

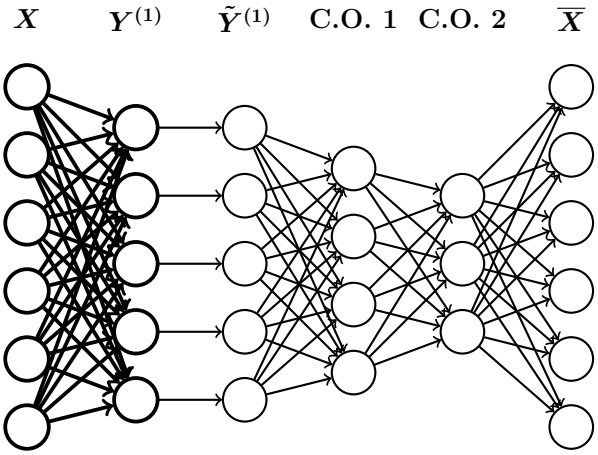
Para continuar añadiendo profundidad a la red:

- Se crea una nueva arquitectura que parte de la entrada original $\mathbf{x}$.
- Se clona la primera capa del anterior MLP. A esta capa se le asignan los parámetros $\theta_1^{(1)}$. Estos parámetros se congelan y no serán actualizados.
- Se corrompe la salida de la representación $y^{(1)}$ con el mismo tipo de ruido que corrompió la entrada en el entrenamiento 1, generando así una representación corrupta $\tilde{y}^{(1)}$.
- Se añade un nuevo MLP con dos capas ocultas.

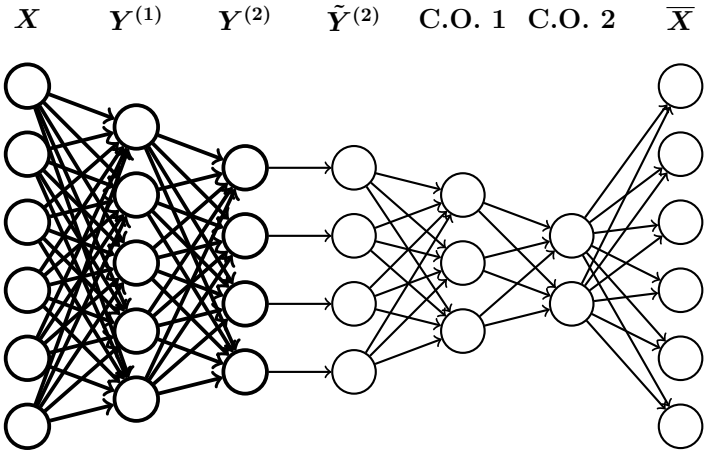
Si lo representamos de manera gráfica:



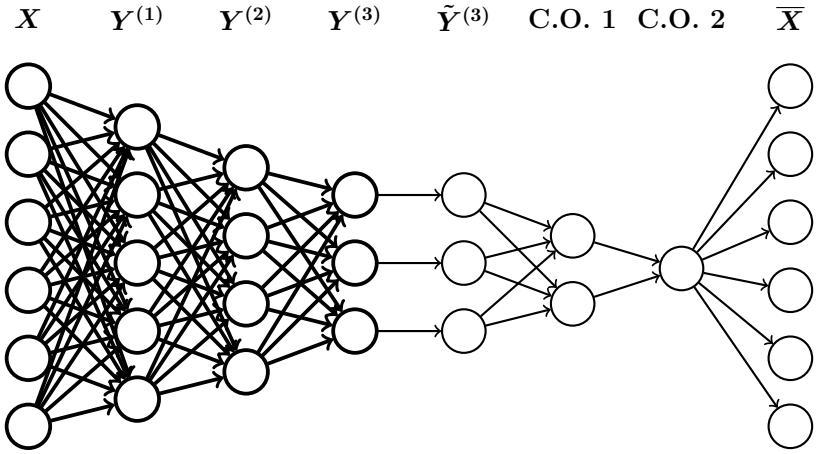
Iteración 1



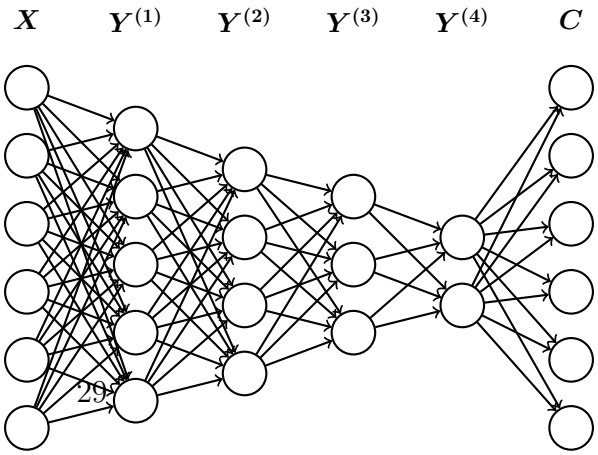
Iteración 2



Iteración 3



Iteración 4



Arquitectura final

Figura 3.1: 1ª y 2ª fila: Visualización de las 4 iteraciones de la fase de crecimiento para la construcción de una arquitectura de 4 capas ocultas a través del algoritmo de la Propuesta 1. Abajo: Arquitectura final resultante.

De esta forma, en cada iteración del entrenamiento se añade una capa de profundidad a la arquitectura final. Una vez alcanzada la profundidad deseada se procede con la tarea de clasificación bajo un criterio supervisado.

3.2. Propuesta 2

En la propuesta A, observamos que al descartar los parámetros que aprende la segunda capa oculta MLP se pierde gran parte de la información usada por la capa de salida para reconstruir la señal de entrada libre de ruido. La información de esta capa oculta eliminada volverá a ser aprendida en una futura iteración volviendo el proceso de aprendizaje poco eficiente. Para suplir este problema se propone una segunda solución semejante a la propuesta A pero en cada iteración en vez de almacenar únicamente los parámetros $\theta_n^{(1)}$ se almacenan también los parámetros $\theta_n^{(2)}$ correspondientes a la segunda capa oculta del MLP.

La representación gráfica de todo el proceso de entrenamiento de la propuesta B puede observarse en la la Figura 3.2

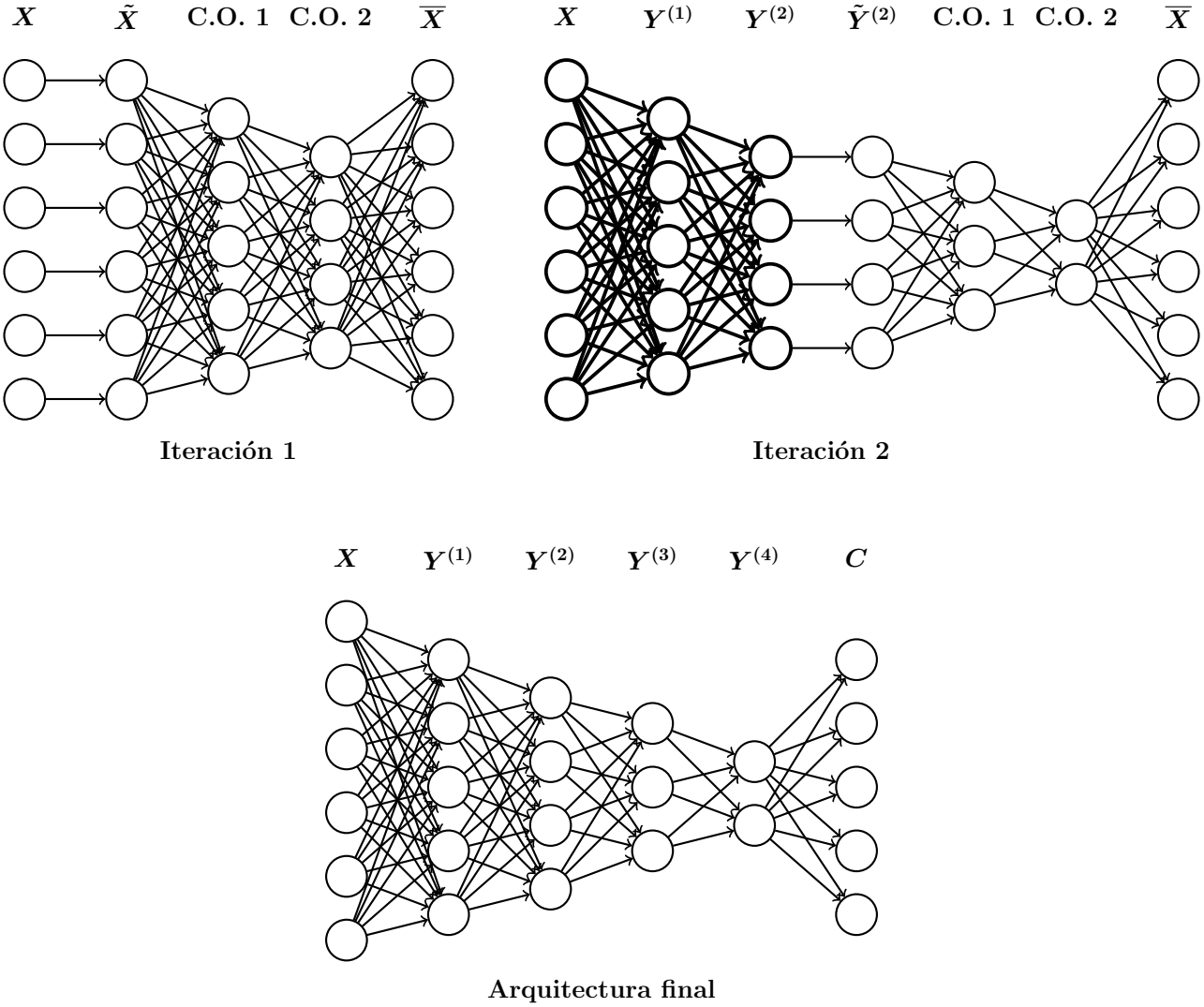


Figura 3.2: Arriba: Visualización de las 2 iteraciones de la fase de crecimiento para la construcción de una arquitectura de 4 capas ocultas a través del algoritmo de la Propuesta 2. Abajo: Arquitectura final resultante.

Con esta alternativa en cada iteración del entrenamiento se obtendrán dos capas de la arquitectura final en vez de una. De igual forma, una vez alcanzada la profundidad deseada se procede con la tarea de clasificación bajo un criterio supervisado. Una clara ventaja de esta propuesta es que el número de entrenamientos necesarios para alcanzar una profundidad específica se reduce a la mitad en comparación al algoritmo MSDAE y a la propuesta A.

3.3. Experimentos

3.3.1. Sets de datos

Para la obtención de los resultados se han usado 5 set de datos distintos con distintos niveles de complejidad, éstos son: MNIST, Fashion-MNIST, Letter-Recognition, MAGIC Gamma Telescope y CIFAR-10.

Sus características son:

3.3.1.1. MNIST

Se trata de un problema de clasificación multi-clase que consta de un total de 70.000 imágenes de números escritos a mano pertenecientes a las 10 clases de dígitos existentes. Cada imagen consta de 28x28 píxeles en blanco y negro, lo que hace un total de 784 variables de entrada que adoptan valores enteros entre 0 y 255. De estas 70.000 imágenes 50.000 van destinadas al conjunto de entrenamiento, 10.000 a validación y 10.000 a test. Los porcentajes del número de ejemplos usados en entrenamiento, validación y test serán los mismos para todos los set de datos, es decir, el 70 % de los ejemplos serán destinados para entrenar y el 30 % restante irá equitativamente repartido entre validación y test.

3.3.1.2. Fashion-MNIST

Tal y como su nombre indica se trata de una versión de MNIST. Consta de 70.000 imágenes con una resolución de 28x28 de 10 clases distintas de prendas de ropa. Estas 10 clases son: camiseta, pantalón, jersey, vestido, abrigo, sandalia, camisa, zapatillas de deporte, bolso y bota.

3.3.1.3. Letter Recognition

Es un problema de clasificación multiclase, se parte de una imagen rectangular con píxeles en blanco y negro con una de las 26 letras mayúsculas del alfabeto inglés,

cada una de estas imágenes es simplificada a 16 variables de entrada. Cada una de estas variables pueden tener un valor entero comprendido entre 0 y 15.

- **x-box:** Posición horizontal de la caja.
- **y-box:** Posición vertical de la caja.
- **width:** Ancho de la caja.
- **high:** Altura de la caja.
- **onpix:** Número total de píxeles.
- **x-bar:** Media de la componente x de los píxeles.
- **y-bar:** Media de la componente y de los píxeles.
- **x2bar:** Varianza media de la componente x.
- **y2bar:** Varianza media de la componente y.
- **xybar:** Correlación media entre x e y.
- **x2ybr:** Media de x * x * y.
- **xy2br:** Media de x * y * y.
- **x-ege:** Recuento medio de bordes de izquierda a derecha.
- **xegvy:** Correlación de x-ege con y.
- **y-ege:** Recuento medio de bordes de abajo hacia arriba.
- **yegvx:** Correlación de y-ege con x.

Consta de un total de 20.000 ejemplos, que de acuerdo a la división antes mencionada se corresponden con 14.286 para entrenamiento, 2.857 para validación y otros 2.857 para test.

3.3.1.4. MAGIC Gamma Telescope

Es un problema de clasificación binaria donde los datos han sido generados por métodos de Montecarlo para simular el registro de partículas gamma de alta energía en un telescopio Cherenkov de rayos gamma. Estos telescopios observan la radiación emitida por las tormentas electromagnéticas provenientes del espacio, los rayos que conforman estas tormentas electromagnéticas se filtran a través de la atmósfera y la información de esta radiación emitida permite discernir si la señal recibida procede de un rayo gamma(g) o de un hadrón(h), éstas serán las dos clases que habrá que clasificar. Como información extra para entender el significado de los variables debe mencionarse que tras hacer un preprocesamiento de los datos, las imágenes capturadas (llamadas en inglés *shower images*), por el sensor tienen forma de elipse y cada una de ellas está conformada por varios cientos de rayos gamma hasta miles de ellos.

Las variables que conforman el problema son:

- **fLength:** Longitud, en milímetros, del eje mayor de la elipse.
- **fWidth:** Longitud, en milímetros, del eje menor de la elipse.
- **fSize:** $10 \cdot log_{10}$ de la suma del contenido de todos los píxeles.
- **fConc:** Relación entre la suma de los dos píxeles de mayor valor y fSize.
- **fConc1:** Relación entre el pixel de mayor valor y fSize.
- **fAsym:** Distancia del pixel con mayor valor al centro, proyectada sobre el eje mayor.
- **fM3Long:** Raíz cúbica del tercer momento a lo largo del eje mayor, en milímetros.
- **fM3Trans:** Raíz cúbica del tercer momento a lo largo del eje menor, en milímetros.
- **fAlpha:** Ángulo entre el eje mayor y el vector al origen, en grados.
- **fDist:** Distancia entre la posición de la partícula al centro de la elipse, en milímetros.

En la Figura 3.3 se puede observar una representación gráfica del tipo de patrón de entrada usados.

Es el único conjunto de datos que no está compensado, hay 12.332 ejemplos de la clase 'g' y 6.688 de la clase 'h'.

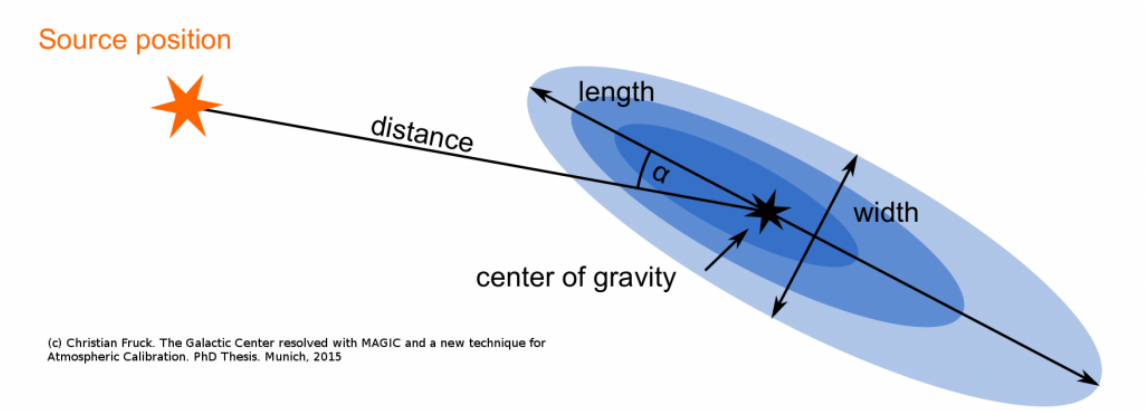


Figura 3.3: Representación de una *shower image* usada para identificar partículas a través de los datos recogidos de un telescopio Cherenkov.

3.3.1.5. CIFAR-10

Al igual que MNIST y Fashion-MNIST se trata de un problema de clasificación multiclase que consta de 60.000 imágenes RGB. Tienen una resolución de 32x32, que junto a los 3 canales de color hacen un total de 3072 variables de entrada. Las 10 clases a la que puede pertenecer un imagen son: avión, automóvil, pájaro, gato, ciervo, perro, rana, caballo, barco y camión.

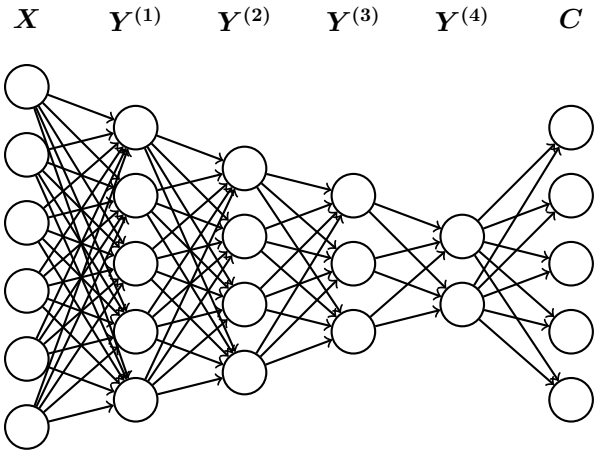
3.3.2. Arquitecturas creadas

Se han creado tanto para el MSDAE como para ambas propuestas 2 versiones de red diferentes. En una primera versión cada capa oculta contiene un menor número de neuronas que la anterior, de forma que se va realizando un cuello de botella respecto a la entrada. El número de neuronas de la primera capa oculta es igual al 90 % de los parámetros de entrada de la red, la segunda capa tendrá una dimensión igual al 80 % de los parámetros de entrada, la tercera capa un 70 % y la cuarta capa un 60 %. A partir de ahora esta arquitectura como arquitectura en cuello de botella.

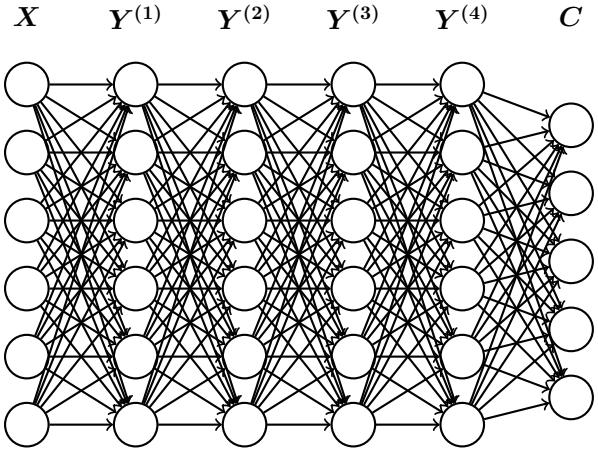
El número de neuronas por capa quedan, para cada conjunto de datos, de la siguiente forma:

Dataset	Entrada	Capa oculta 1	Capa Oculta 2	Capa oculta 3	Capa Oculta 4
MNIST	784	706	627	549	470
Fashion-MNIST	784	706	627	549	470
Letter Recognition	16	14	13	11	10
Magic04	11	10	9	8	7
CIFAR-10	3072	2765	2458	2150	1843

Análogamente se ha creado un segundo tipo de arquitectura en la que se mantienen constantes el número de neuronas ocultas en capa. En esta variante, el número de neuronas ocultas en cada capa es siempre igual al número de parámetros de entrada.



Arquitectura cuello de botella



Arquitectura de capas paralelas

Figura 3.4: Comparación entre 2 redes de 4 capas ocultas las dos arquitecturas empleadas en este trabajo.

3.3.3. Entrenamiento

Especificaciones técnicas del equipo

El entrenamiento de las redes se ha llevado a cabo en un equipo portátil con las siguientes especificaciones técnicas.

- Microprocesador Intel i7-9750H con 6 núcleos a 2.60 GHz, con una frecuencia máxima de 4.50 GHz.
- 16.0 GB de memoria RAM DDR4 a 2666 MHz.
- Tarjeta Gráfica NVIDIA GeForce GTX 1650 con 4GB GDDR5 VRAM y 896 núcleos CUDA.
- Sistema Operativo Windows 10.

Lenguaje de programación

La implementación del algoritmo se ha realizado en el lenguaje Python 3, en concreto se ha usado la librería Keras sobre el backend Tensorflow 2.0. En todo momento, se ha intentado no interferir en el entrenamiento haciendo un uso mínimo del equipo mientras entrenaba.

Hiperparámetros de la red

Para el entrenamiento se ha usado el optimizador ADAM [8]. Los parámetros de este optimizador son: *learning rate*=0.001, $\beta_1 = 0{,}9$, $\beta_2 = 0{,}999$, $\epsilon = 1E-7$. Para el entrenamiento de clasificación final, el valor del *learning rate* disminuye a 0.0001. Los parámetros β_1 y β_2 son las tasas de decaimiento exponencial para las estimaciones del primer y del segundo momento. Si se disminuyen, el entrenamiento se verá ralentizado. Por último, ϵ es un valor infinitesimal cuyo objetivo es evitar una división entre 0.

Dado que una función de activación de tipo sigmoide puede provocar un desvanecimiento del gradiente en redes profundas se ha usado una función de activación ReLU para todas las capas. La única excepción es la capa de salida del clasificador final, cuya función de activación es softmax.

Se ha escogido un tamaño de lote (en inglés, *batch size*) de 200 ejemplos. Ésto es, en vez de realizar la actualización de los pesos tras la propagación de un ejemplo de entrenamiento, se realizará tras la propagación de 200 ejemplos.

El entrenamiento se lleva a cabo con un número de épocas arbitrariamente alto, en concreto 100.000. Se ha programado un mecanismo de interrupción temprana, en inglés *early stopping* cuyos parámetros en Keras son: monitor='val_loss,

min_delta=0.0001, patience = 4. Éstos significan que si la función de pérdidas definida para el conjunto de validación no disminuye su valor más de 0.0001 durante 4 épocas consecutivas el entrenamiento se detiene.

Dado que en el proceso de entrenamiento total realmente estamos abordando dos problemas distintos (el primero no-supervisado y otro supervisado) deberán de usarse dos funciones de pérdidas diferentes. Para el aprendizaje no-supervisado, dado que se trata de un problema de regresión, en el que se intenta predecir un conjunto de valores continuos en la salida, se usará el error cuadrático medio. Una característica principal de esta función de coste es que el error cuadrático medio crece de forma exponencial cuanto mayor es la diferencia entre el valor predicho por la red y la salida deseada. Por otro lado, para el problema de clasificación se usará la función de pérdidas entropía cruzada categórica (en inglés, *categorical cross-entropye*). Esta función de pérdidas está diseñada para ser usada con la función de activación softmax de forma que si la predicción de pertenencia a la clase correcta es cercana 0 adquiere un valor alto, este valor de error disminuye rápidamente al aumentar la probabilidad de pertencia a la clase correcta y se frena conforme se acerca a la probabilidad de acierto del 100 %. Dicho de otro modo, esta función de pérdidas sanciona el no acierto de la clase verdadera pero en especial cuando la red está segura que el ejemplo no pertenece a dicha clase (con valores de probabilidad muy cercanos a cero).

Evaluación de los resultados

Para la evaluación de los resultados, dado que los conjuntos de set de datos están compensados, se medirá la precisión (tanto por ciento de los ejemplos clasificados correctamente) que es capaz de realizar la arquitectura final. Esta evaluación se lleva a cabo en las dos propuestas presentadas y una tercera arquitectura construida a través del algoritmo MSDAE. Se evaluará también el tiempo necesario para entrenar toda una arquitectura completa.

Cada una de las arquitecturas creadas se ha entrenado 25 veces para obtener su media μ y se desviación típica σ. Se considerará que una propuesta es mejor que el MSDAE correspondiente si se cumple que la media de su precisión en la tarea de clasificación final es mayor que la media del MSDAE más la media de las desviaciones típicas, es decir:

$$\mu_2 > \mu_1 + \frac{\sigma_1+\sigma_2}{2}$$

En caso de cumplirse esta condición en cualquiera de las propuestas, el resultado se resaltará en negrita.

3.4. Resultados

Para la arquitectura de cuello de botella los resultados son:

Precisión final - MNIST			
Varianza Ruido	MSDAE	Propuesta 1	Propuesta 2
0	97,49±0,49	97,71±0,16	97,69±0,29
0.05	97,65±0,48	97,85±0,16	97,83±0,26
0.2	97,99±0,24	98,17±0,17	98,14±0,08
0.4	98,28±0,11	98,34±0,07	98,24±0,08
Precisión final - FASHION MNIST			
Varianza Ruido	MSDAE	Propuesta 1	Propuesta 2
0	87,88±0,95	88,39±0,69	88,12±0,91
0.05	88,35±0,86	88,69±0,85	88,61±0,84
0.2	89,32±0,43	89,33±0,49	89,29±0,55
0.4	89,29±0,51	89,16±0,43	89,08±0,6
Precisión final - Letter Recognition			
Varianza Ruido	MSDAE	Propuesta 1	Propuesta 2
0	74,69±1,97	75,68±2,3	75,55±1,95
0.05	75,05±2,33	76,85±1,62	76,35±1,59
0.2	76,14±2,33	77,08±1,64	76,23±2,13
0.4	76,46±2,19	77,13±1,89	77,72±1,38
Precisión final - MAGIC Gamma Telescope			
Varianza Ruido	MSDAE	Propuesta 1	Propuesta 2
0	83,77±1,81	84,55±1,23	84,18±0,86
0.05	84,02±1,34	84,37±1,09	84,49±0,89
0.2	84±1,43	84,62±0,92	84,16±1,02
0.4	84,25±1,06	84,3±1,06	84,53±1,07
Precisión final - CIFAR-10			
Varianza Ruido	MSDAE	Propuesta 1	Propuesta 2
0	54,28±0,82	54,9±0,62	**55,1±0,71**
0.05	54,94±0,6	54,67±0,74	55,42±0,75
0.2	54,88±0,48	54,76±0,57	**55,64±0,59**
0.4	55,27±0,63	54,08±0,67	54,93±0,62

Figura 3.5: Evaluación de todos los modelos para la arquitectura de cuello de botella sobre sus respectivos conjuntos de test.

Para la arquitectura que mantiene la dimensión constante en todas las capas:

Precisión final - MNIST			
Varianza Ruido	MSDAE	Propuesta 1	Propuesta 2
0	97,5±0,77	97,8±0,22	97,68±0,27
0.05	97,79±0,21	97,7±0,79	97,93±0,2
0.2	98,12±0,16	**98,35±0,08**	98,2±0,13
0.4	98,34±0,06	98,35±0,08	98,31±0,09
Precisión final - FASHION MNIST			
Varianza Ruido	MSDAE	Propuesta 1	Propuesta 2
0	87,67±1,57	88,36±0,96	87,52±1,87
0.05	88,37±1,19	88,77±0,76	88,5±0,95
0.2	89,47±0,51	89,62±0,41	89,26±0,44
0.4	89,33±0,33	89,23±0,48	89,18±0,45
Precisión final - Letter Recognition			
Varianza Ruido	MSDAE	Propuesta 1	Propuesta 2
0	79,99±1,67	79,59±1,48	79,41±1,75
0.05	79,95±2,01	80,43±1	80,04±1,56
0.2	80,79±1,66	80,95±1,57	80,53±1,72
0.4	81,22±1,68	81,15±1,68	80,96±1,2
Precisión final - MAGIC Gamma Telescope			
Varianza Ruido	MSDAE	Propuesta 1	Propuesta 2
0	84,69±1,39	84,97±0,89	85,08±1,23
0.05	84,96±0,74	85,29±0,73	85,18±0,74
0.2	84,86±0,91	85,03±0,5	85,47±0,59
0.4	85,13±0,75	85,31±0,62	85,59±0,6
Precisión final - CIFAR-10*			
Varianza Ruido	MSDAE	Propuesta 1	Propuesta 2
0	55,11±0,19	54,69±0,42	**54,69±0,42**
0.05	55,92±0,42	55,47±0,48	55,47±0,48
0.2	55,41±0,27	55,2±0,39	**55,2±0,39**
0.4	55,76±0,58	55,15±0,01	55,15±0,01

Figura 3.6: Evaluación de todos los modelos para la arquitectura de capas paralelas sobre sus respectivos conjuntos de test. * Al ejecutar el conjunto de datos CIFAR-10 en este tipo de red se suele obtener un error de memoria insuficiente en la mayoría de entrenamientos. Es por esto por lo que solo se han realizado 2 entrenamientos en lugar de los 25 correspondientes

De igual forma, se ha registrado el tiempo necesario para construir la arquitectura final más su correspondiente entrenamiento. Para determinar si una propuesta es más rápida que su MSDAE correspondiente se observará si su media en el tiempo de entrenamiento total es menor a la media del MSDAE menos la media de las desviaciones típicas. Es decir:

$$\mu_2 < \mu_1 - \frac{\sigma_1+\sigma_2}{2}$$

Para la arquitectura que crea un cuello de botella:

Tiempo(seg.) - MNIST			
Varianza Ruido	MSDAE	Propuesta 1	Propuesta 2
0	90,86±6,6	93,79±7,09	**62,66±7,93**
0.05	91,48±6,78	101,03±10,46	**63,54±6,39**
0.2	87,67±8,95	95,95±9,33	**54,46±5,52**
0.4	78,26±4,64	85,65±7,54	**47,23±3,96**
Tiempo(seg.) - FASHION MNIST			
Varianza Ruido	MSDAE	Propuesta 1	Propuesta 2
0	82,66±8,28	93,37±7,92	**55,51±7,05**
0.05	71,55±6,33	77,45±6,46	**47,04±5,88**
0.2	73,6±7,19	84,12±10,82	**48,12±5,73**
0.4	86,84±11,34	101,59±6,23	**52,96±4,38**
Tiempo(seg.) - Letter Recognition			
Varianza Ruido	MSDAE	Propuesta 1	Propuesta 2
0	171,24±29,72	198,29±26,83	**143,19±23,69**
0.05	165,9±21,76	196,57±26,75	150,09±23,03
0.2	153,44±23,84	173,93±20,58	145,22±23,2
0.4	163,39±24,55	185,8±32,78	147,04±24,7
Tiempo(seg.) - MAGIC Gamma Telescope			
Varianza Ruido	MSDAE	Propuesta 1	Propuesta 2
0	122,81±25,51	142,66±20,21	**94,97±14,72**
0.05	116,68±25,1	128,19±26,38	**90,43±23,12**
0.2	117,86±19,28	113,7±23,13	**74,85±14,59**
0.4	113,54±17,53	101±18,89	**74,4±15,45**
Tiempo(seg.) - CIFAR-10			
Varianza Ruido	MSDAE	Propuesta 1	Propuesta 2
0	327,67±31,26	337,83±19,7	**224,9±16,47**
0.05	294,14±19,05	287,64±34,9	354,59±704,37
0.2	172,62±13,67	288,04±60,23	213,97±29,19
0.4	351,24±28,58	327,9±105,63	**226,12±67,62**

Figura 3.7: Tiempos de entrenamiento total para la arquitectura de cuello de botella

Para la arquitectura que mantiene la dimensión entre capas, los tiempos de entrenamiento son:

Tiempo(seg.) - MNIST			
Varianza Ruido	MSDAE	Propuesta 1	Propuesta 2
0	95,08±7,24	100,34±9,2	**63,51±7,07**
0.05	96,32±6,97	116,56±14,98	**71,52±11,02**
0.2	100,53±9,08	117,06±9,01	**60,15±7,29**
0.4	79,06±3,84	92,79±5,95	**49,66±3,76**
Tiempo(seg.) - FASHION MNIST			
Varianza Ruido	MSDAE	Propuesta 1	Propuesta 2
0	86,33±7,23	93,91±10,37	**59,05±6,84**
0.05	75,82±5,9	81,88±5,41	**53,15±5,46**
0.2	92,73±8,11	99,47±8,95	**54,33±5,92**
0.4	106,46±8,41	117,16±9,17	**62,66±6,33**
Tiempo(seg.) - Letter Recognition			
Varianza Ruido	MSDAE	Propuesta 1	Propuesta 2
0	175,22±24,6	181,38±20,63	**134,26±13,62**
0.05	146,59±21,71	177,9±23,53	150,14±27,22
0.2	138,13±14,51	185,99±34,85	140,22±25,57
0.4	142,29±17,03	197,34±33,04	137,1±15,93
Tiempo(seg.) - MAGIC Gamma Telescope			
Varianza Ruido	MSDAE	Propuesta 1	Propuesta 2
0	80,91±13,36	94,6±16,11	**66,19±12,96**
0.05	75,15±10,85	89,06±13,92	**61,31±10,89**
0.2	69,46±8,55	78,53±10,98	59,35±12,22
0.4	70,08±7,74	74,59±11,39	**58,21±7,12**
Tiempo(seg.) - CIFAR-10*			
Varianza Ruido	MSDAE	Propuesta 1	Propuesta 2
0	459,73±47,3	452,1±23,79	**321,48±40,77**
0.05	411,3±16,68	499,89±38,38	**300,71±26,18**
0.2	391,14±176,68	423,93±52,01	296,22±37,03
0.4	650,38±48,2	**500,09±105,88**	**371,15±129,08**

Figura 3.8: Tiempos de entrenamiento total para la arquitectura de capas paralelas

3.5. Análisis de los resultados

Lo primero que se observa en casi todos los experimientos es que al aumentar la varianza del ruido con el que se corrompe la señal de entrada aumenta la tasa de precisión de la mayoría de arquitecturas. El mejor desempeño aparece la mayoría de las veces para varianzas iguales a 0.2 ó 0.4. Este hecho confirma el criterio de supresión de ruido como válido para la extracción de características útiles.

Respecto al rendimiento de la Propuesta 1, de las 40 versiones de red diferentes (contando todas las combinaciones entre tipos de ruido y arquitecturas) y pese a que las mayoría de ellas tienen un desempeño promedio superior, solamente en una de ellas consigue un rendimiento significativamente mejor respecto a su MSDAE correspondiente. En cuanto al tiempo de ejecución, solamente en una versión de red se consigue un tiempo menor y es concretamente para el conjunto de datos CIFAR-10 sobre la arquitectura de capas paralelas. Tal y como se ha explicado, para esta combinación de arquitectura y conjunto de datos, el experimento solo se ha podido realizar 2 veces en vez de las 25 necesarias por problemas con el hardware, por lo que este resultado podría no ser significativo. En los 39 redes restantes, la Propuesta 1 consigue un tiempo de ejecución mayor incluso que su equivalente MSDAE.

De igual forma, el rendimiento de la Propuesta 2 no presenta una mejora significativa en la mayoría de las redes creadas. Destaca el caso de CIFAR-10, donde se consiguió para ambas arquitecturas una tasa de acierto significativamente superior respecto a su MSDAE. Al ser CIFAR-10 el problema más complejo usado en este trabajo y el único en el que se obtienen mejores resultados puede suponerse que la ventaja de esta propuesta se encontraría al ser explotada en problemas de mayor complejidad. En cuanto al tiempo necesario para esta propuesta, se observa que en 30 de las 40 redes creadas se obtiene un tiempo de ejecución significativamente menor, llegando a conseguir en algunos casos un ahorro de tiempo superior al 30 %, como es el caso de conjunto de datos FASHION-MNIST para la arquitectura cuello de botella, donde el ahorro de tiempo respecto a MSDAE es del 38.87 %. Este potencial ahorro de tiempo, unido al hecho de que no se obtienen promedios de precisión menores a los obtenidos con MSDAE presentan a la Propuesta 2 como una alternativa factible.

Por último, al comparar los dos tipos de arquitecturas existentes no se aprecian grandes diferencias salvo para el conjunto de datos Letter Recognition, donde la propuesta de capas paralelas obtiene unos resultados algo mayores. Aun así, es de esperar que al tener un número algo mayor de neuronas sea capaz de conseguir un mejor rendimiento. En cuanto al tiempo, ambas arquitecturas obtienen resultados bastantes diferentes dependiendo del problema abordado. La arquitectura de cuello de botella es más rápida que la de capas paralelas en los problemas MNIST y FASHION-MNIST pero sucede lo contrario con los problemas Letter Recognition y MAGIC Gamma Telescope. Para esta comparación se excluirá el conjunto de datos CIFAR-10 por los problemas ya mencionados.

3.6. Conclusión final y futuros trabajos

Como se ha expuesto en el anterior punto, la restricción de entrenar capa a capa presente en los algoritmos SDAE, MSDAE y CMSDAE parece ser demasiado restrictiva de forma que se puede seguir obteniendo un desempeño semejante pero en un menor tiempo en el caso de entrenar dos capas simultáneamente.

Destacar, que esta afirmación está sujeta a las limitaciones de los conjuntos de datos empleados ya que estos no destacan por su complejidad para ser resueltos o directamente son considerados conjuntos "de juguete". Es por ésto, unido a los resultados obtenidos en el conjunto de datos CIFAR-10, que sería necesario abordar problemas más complejos de cara a ver una posible mejoría en el desempeño de ambas propuestas.

Concluir, que independientemente de los resultados obtenidos, este trabajo me ha permitido ampliar profundamente los conocimientos en este campo habiendo podido alcanzar un entendimiento mucho más profundo de como funcionan las redes neuronales, me he visto motivado a leer varios artículos científicos por primera vez, a estudiar las diferentes formas de evaluación para distintos problemas y por último, a mejorar mis habilidades como programador.

Al estar centrado este trabajo en el ámbito de la Inteligencia Artificial y habiendo expuesto que son necesarios más experimentos para asegurar la viabilidad de ambas propuestas, me gustaría tomarme la libertad de terminar con una frase que considero bastante adecuada:

"Solo podemos ver un poco del futuro, pero lo suficiente para saber que hay mucho que hacer."

Alan Turing

Bibliografía

[1] Alex Krizhevsky et al. ImageNet Classification with Deep Convolutional Neural Networks. Communications of the ACM.

[2] F. Rosenblatt (1958). The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65(6).

[3] Pascal Vincent et al. (2010) Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. Journal of Machine Learning Research 11 3371-3408

[4] Marvin Minsky, Seymour Papert (1969). Perceptrons: an introduction to computational geometry. ISBN: 0 262 13043 2

[5] Kolmogorov, A. N. (1957). On the Representation of Continuous Functions of Several Variables by Superposition of Continuous Functions of one Variable and Addition. Doklady Akademii. Nauk USSR, 114, 679-681.

[6] Cybenko, G. (1989). Approximation by Superpositions of a Sigmoidal Function. Math. Control Signals Systems, 2, 303-314.

[7] Adrián Sánchez-Morales, José-Luis Sancho-Gómez, Aníbal R. Figueiras-Vidal. (2019). Exploiting label information to improve auto-encoding based classifiers. Neuroconputing.

[8] Diederik P. Kingma, Jimmy Lei Ba. (2017). ADAM: A method for stochastic optimization.