TRABAJO FIN DE GRADO

INGENIERÍA INFORMÁTICA

Implementación optimizada sobre sistemas heterogéneos de algoritmos de Deep Learning para clasificación de imágenes

Autor
David Sánchez Pérez

Directores
José Miguel Mantas Ruiz

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, mes de Febrero 2024

Implementación optimizada sobre sistemas heterogéneos de algoritmos de Deep Learning para clasificación de imágenes

Autor
David Sánchez Pérez

Directores
José Miguel Mantas Ruiz

Título del Proyecto: Subtítulo del proyecto

Nombre Apellido1 Apellido2 (alumno)

Palabras clave: palabra_clave1, palabra_clave2, palabra_clave3, ...

Resumen

Poner aquí el resumen.

Project Title: Project Subtitle

First name, Family name (student)

Keywords: Keyword1, Keyword2, Keyword3, ...

Abstract

Write here the abstract in English.

Yo, **Nombre Apellido1 Apellido2**, alumno de la titulación TITULACIÓN de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI XXXXXXXXX, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Nombre Apellido1 Apellido2

Granada a X de mes de 201 .

D. **Nombre Apellido1 Apellido2 (tutor1)**, Profesor del Área de XXXX del Departamento YYYY de la Universidad de Granada.

D. **Nombre Apellido1 Apellido2 (tutor2)**, Profesor del Área de XXXX del Departamento YYYY de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Título del proyecto, Subtítulo del proyecto***, ha sido realizado bajo su supervisión por **Nombre Apellido1 Apellido2 (alumno)**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a X de mes de 201 .

Los directores:

Nombre Apellido1 Apellido2 (tutor1) **Nombre Apellido1 Apellido2 (tutor2)**

Agradecimientos

Poner aquí agradecimientos...

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Resumen

1.2. Estado del arte

1.3. Objetivos

Capítulo 2

Conceptos previos

2.1. Machine Learning

Se entiende como el campo de las ciencias de computación que en vez de enfocarse en el diseño de algoritmos explícitos, optan por el estudio de técnicas de aprendizaje. Este enfoque tiene un gran éxito en tareas computacionales donde no es factible diseñar un algoritmo de forma explícita. [5]
En vez de averiguar las distintas reglas a seguir para llegar a una solución, esta alternativa permite simplemente suministrar ejemplos de lo que debería pasar en distintas situaciones, y dejar que la máquina aprenda y extraiga ella misma sus propias conclusiones. De esta forma, el procedimiento en aprendizaje supervisado consiste en 'entrenar' con una muestra de N ejemplos, extraer información de ellos, y posteriormente poder evaluar de forma 'correcta' (bajo un margen de error controlado) otra muestra de M ejemplos, siendo M >N. [6]
Este enfoque ha contribuido en el avance de áreas como reconocimiento de voz, visión por ordenador, procesamiento de lenguaje natural, etc.

2.2. Componentes necesarios para el aprendizaje supervisado

Datos de entrada X y de salida Y que el modelo empleará para aprender y tomar decisiones. Ambos se unen para formar un dataset de entradas-salidas D=$\{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$. Para que el aprendizaje sea posible, debe existir una función F: X $\rightarrow$ Y tal que y_i = F(x_i) para i$\in\{1...N\}$. De esta forma, en función del dataset D, el modelo tratará de encontrar una función G que aproxime F para dicho conjunto. Además, se suelen aplicar técnicas que permitan una mejor generalización del modelo, expandiendo las capacidades del mismo y permitiendo que su conocimiento pueda ser útil incluso fuera de la muestra de datos inicial. [6]

2.3. División de datos en entrenamiento y test

Para visualizar la generalización del modelo, el conjunto de datos D se suele dividir en 2 subconjuntos, (entrenamiento y test) de forma que se pueda estimar si realmente 'aprende' o solo memoriza.
Una vez realizada la división, se entrena el modelo con los datos del conjunto de entrenamiento. Cuando se termina el entrenamiento, se accede al conjunto test y se visualiza el rendimiento del modelo sobre el mismo. Como los datos de test no se emplearon en ningún momento, aportan una estimación sobre la generalización del modelo fuera de la muestra con la que se entrenó.

2.4. Tipos de aprendizaje

2.4.1. Aprendizaje Supervisado

Es el que se empleará en este proyecto. Se caracteriza por la presencia de una etiqueta 'correcta' y_i asociada a cada dato de entrada x_i. Posteriormente, la red empleará ambos valores para, a partir de x_i, tratar de deducir y_i. [6]
Aunque se trate de impedir, la existencia de ruido en los datos es inevitable, implicando que algunas etiquetas de Y=$\{y_1, y_2, ..., y_N\}$ puedan ser erróneas. Este tipo de aprendizaje se divide a su vez en problemas de clasificación y regresión, centrándose en predecir etiquetas o valores numéricos, respectivamente.

2.4.2. Aprendizaje No Supervisado

En este tipo de aprendizaje los datos no contienen ninguna información respecto a Y. De esta forma, el conjunto de datos D se compone exclusivamente de valores X=$\{x_1, x_2, ..., x_N\}$. [6]

2.4.3. Aprendizaje Por Refuerzo

En este caso tampoco existe un y_i 'correcto' asociado a cada x_i. En su lugar, se asocia a cada x_i una etiqueta con un valor posible de y_i, además de una medida que indica como de bueno es el mismo. [6]

2.5. Redes Neuronales Totalmente Conectadas

2.5.1. Neurona

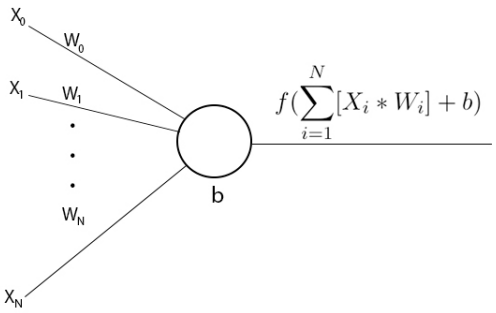


Figura 2.1: Imagen de una neurona

Una neurona parte de una serie de datos de entrada X=$\{x_1, x_2, ..., x_N\}$ tal que cada x_i∈X se encuentra asociado a un peso $w_i \in W$.
Esta los emplea para realizar una suma ponderada y posteriormente añadir un sesgo b, además de aplicar una función de activación f sobre el resultado obtenido.

2.5.2. Estructura por capas

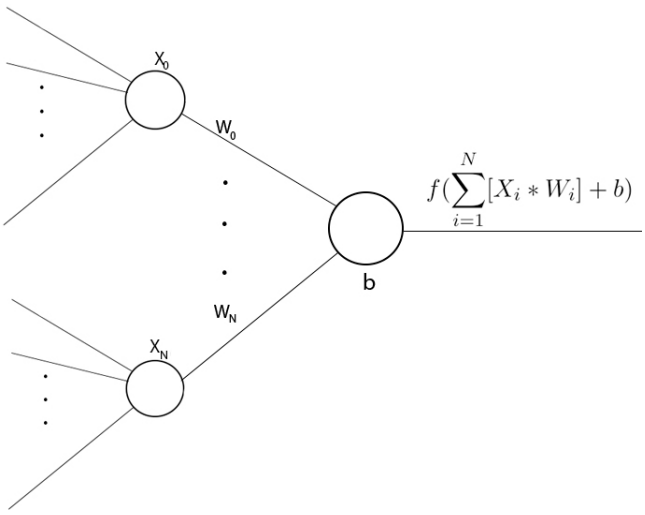


Figura 2.2: Imagen de una capa de neuronas

Las neuronas se suelen agrupas por capas, de tal forma que la salida de una compone la entrada de la siguiente, formando así modelos más sofisticados.

2.5.3. Funciones de activación

ReLU

$$ReLU(x) = max(0, x) \tag{2.1}$$

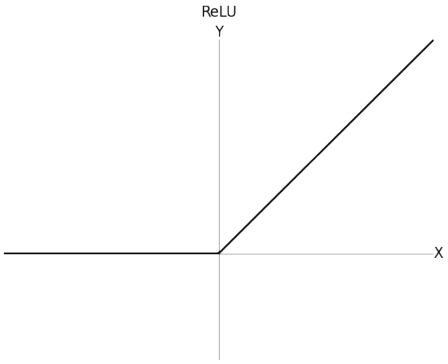


Figura 2.3: Imagen de la función de activación ReLU

A cambio de un bajo coste computacional, aporta no linealidad a la neurona, permitiendo a esta aprender funciones de mayor complejidad.
Como su gradiente es 0 o 1, evita una reducción excesiva del mismo para valores positivos, mitigando así el problema del desvanecimiento del gradiente, caracterizado por la presencia de gradientes muy pequeños en backpropagation y provocar un aprendizaje lento. [7]

Sigmoide

$$sigmoide(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

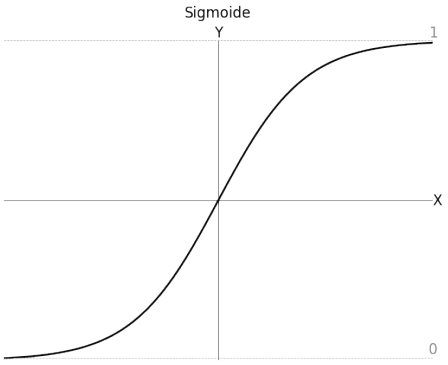


Figura 2.4: Imagen de la función de activación Sigmoide

Se trata de una función interesante en el ámbito de la clasificación binaria, pues se caracteriza por transformar un valor de entrada en una salida comprendida en el rango [0-1].
Aunque sea monótona creciente y diferenciable en todos los puntos, tiende a saturarse con valores extremos (positivos o negativos). Por tanto, su aplicación dependerá del caso concreto a tratar. [8]

SoftMax

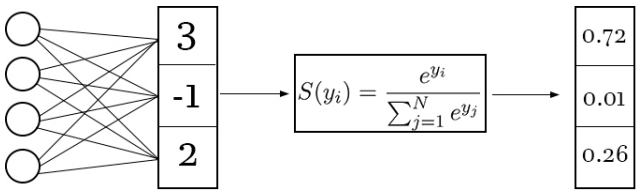


Figura 2.5: Imagen de la función de activación SoftMax

Para n entradas, produce n salidas con valores en el rango [0-1] que mantienen la proporción de entrada y cuya suma es 1. Por tanto, se pueden interpretar como la probabilidad de pertenencia a cada clase, siendo especialmente útil en clasificación multiclase. [9]

2.5.4. One-hot encoding

2.5.5. Función de error o pérdida

Entropía Cruzada

$$E(y, \hat{y}) = -\sum_{i=1}^{H} [y_i * log(\hat{y}_i)] \tag{2.3}$$

Es una métrica empleada en aprendizaje automático para medir qué tan bien se desempeña un modelo de clasificación. La pérdida o error se mide como un valor en el rango [0-1], siendo 0 un modelo perfecto y 1 otro totalmente erróneo. [10]

H es el número de clases al que puede pertenecer cada dato de entrada $x_i \in X$.

Sigmoid Cross Entropy Loss

Es un caso particular de entropía cruzada caracterizado por la presencia de un número de clases igual a 2.

$$E(x) = -\frac{1}{N} \sum_{i=1}^{N} [y_i * log(\hat{y}_i) + (1 - y_i) * log(1 - \hat{y}_i)] \tag{2.4}$$

y = etiqueta real
$\hat{y}$ = predicción

2.5.6. Descenso del gradiente

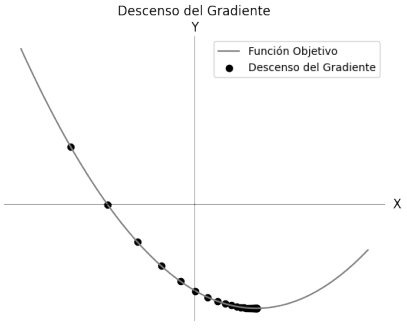


Figura 2.6: Ejemplo de funcionamiento del descenso del gradiente

Es un método de optimización iterativo que busca el mínimo local en una función diferenciable. En la figura 2.6 se muestra un ejemplo del mismo,

donde cada punto representa una iteración del algoritmo.
Su nombre viene del término 'gradiente', siendo este una generalización multivariable de la derivada y denominado por el símbolo ∇. Para una función f y un punto p, este indica la dirección del máximo incremento en la misma. El descenso del gradiente usa esta información para, una vez obtenido el gradiente, desplazarse en dirección contraria, es decir, en dirección del mínimo. Además, la distancia que se recorre en cada iteración viene dada por un hiperparámetro denominado "learning rate" o α. [11] [12] [13]

Entrenamiento

De esta forma, el procedimiento para entrenar una red neuronal consiste en, para una entrada x_i y una etiqueta asociada y_i, emplear x_i para producir una predicción $\hat{y}_i$ (**ForwardPropagation** o Propagación hacia delante) que posteriormente se podrá comparar con y_i mediante una función de error H(x) y obtener una medida de lo buena o mala que fue la misma. Una vez obtenido dicho "error", se aplica el algoritmo del descenso del gradiente para cada parámetro de la red (**BackPropagation** o retropropagacion). [10]

$$W_{t+1} = W_t - \alpha * \frac{\partial H(x)}{\partial W_t} \tag{2.5}$$

$$b_{t+1} = b_t - \alpha * \frac{\partial H(x)}{\partial b_t} \tag{2.6}$$

Así, se actualizarán los parámetros de la red neuronal según las fórmulas 2.5 y 2.6. En ellas, W_t y b_t indican los valores del peso W y bias b en el instante o iteración t, de la misma forma que W_{t+1} y b_{t+1} representan los valores de los mismos en el instante t+1. [14]

2.5.7. Inicialización de pesos y sesgos

Inicialización de pesos

Como función de activación se empleará ReLU. Por tanto, tal y como se indica en la bibliografía, se inicializan los pesos mediante la "inicialización He" o "inicialización Kaiming He". Esta consiste en, para un peso w, inicializarlo según una distribución gaussiana con una media de 0.0 y una desviación típica de $\sqrt{\frac{2}{N\_in}}$, siendo N_in el número de neuronas en la capa de entrada. [15] [16] [17].

Inicialización de sesgos

De la misma forma, se sigue la bilbiografía y los sesgos se inicializarán a 0.0 . [18] [19]

2.5.8. Tipos de codificaciones

En el campo de machine learning existen varios tipos de codificaciones. De esta forma, para codificar 3 clases distintas se podrían codificar o bien mediante {1, 2, 3} (codificación de etiquetas), o mediante {100, 010, 001} (codificación one-hot), por ejemplo. En este proyecto se empleará la codificación one-hot, pues aporta unas ventajas que se contemplarán con detalle en secciones posteriores.

2.5.9. Propagación hacia delante con softmax

Suponemos que para un x_i dado, se obtiene S($\hat{y}$) = [S($\hat{y}_1$), S($\hat{y}_2$), S($\hat{y}_3$)] = [0.04, 0.7, 0.26]
Para dicho x_i, y_i = [0, 0, 1]
En este caso, el modelo cree que x_i pertenece a la clase 2 (0.7 es el mayor número del vector S($\hat{y}$)). Sin embargo, y_i indica que x_i pertenece a la clase 3.

Se calcula el error de entropía cruzada:

$$E(y, S(\hat{y})) = -(0 * log(0.04) + 0 * log(0.7) + 1 * log(0.26)) \tag{2.7}$$

$$E(y, S(\hat{y})) = -log(0.26) = 0.585 \tag{2.8}$$

[1]

2.6. Redes Neuronales Convolucionales

2.6.1. Capa convolucional

Componentes

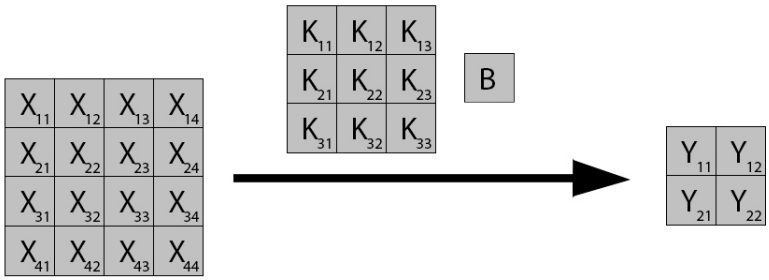
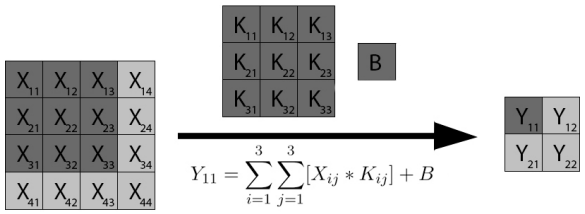


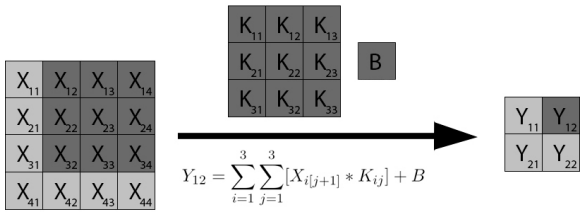
Figura 2.7: Componentes en una capa convolucional

Una capa convolucional parte de un volumen de entrada X, un kernel de filtros K, un sesgo B y una función de activación para, mediante una convolución, obtener un volumen de salida Y. [20] [21].

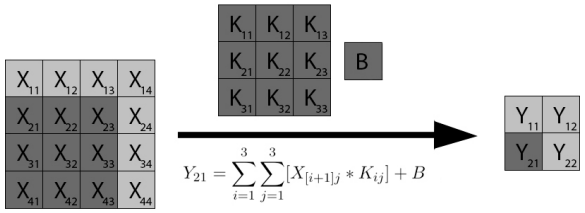
Propagación hacia delante



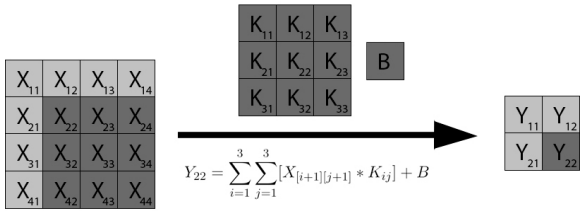
(a) Cálculo Y_{11}



(b) Cálculo Y_{12}



(c) Cálculo Y_{21}

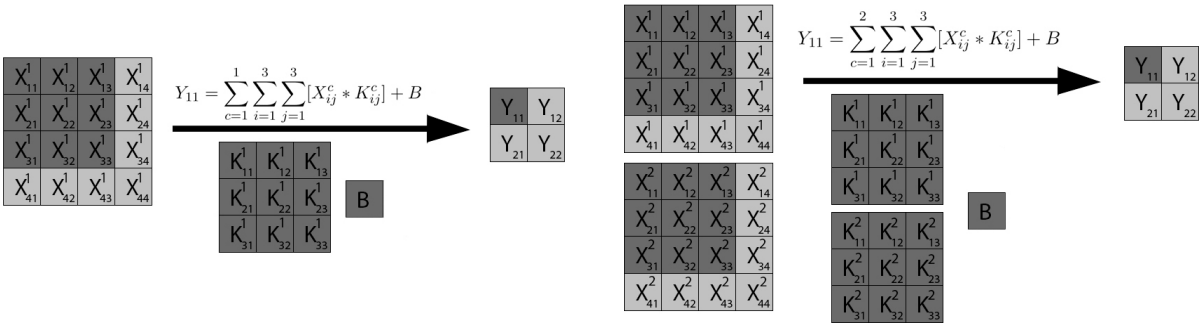


(d) Cálculo Y_{22}

Figura 2.8: Propagación hacia delante en una capa convolucional

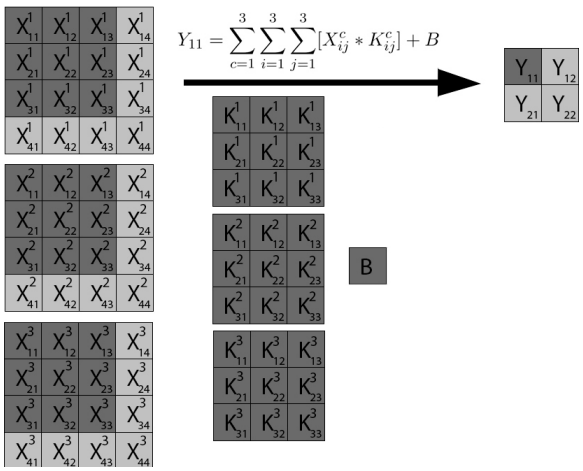
Una convolución entre 2 volúmenes de datos X y K, consiste en "deslizar K sobre X" tal y como se muestra en la Figura 2.8. De esta forma, en cada

"paso" se recorren ambos volúmenes, multiplicando los elementos de X y K que se encuentren en la misma posición. Posteriormente, se suma cada resultado obtenido, además de un sesgo B y finalmente aplicar una función de activación. [20]
En la Figura 2.8 se emplea un volumen X con un solo canal de profundidad. Sin embargo, este no es el caso común. Por tanto, se denotará como X_{ij}^{c} al elemento de X que se encuentre en la posición (i,j) del canal de profundidad c. [21].



(a) Cálculo Y_{11} con entrada X de 1 canal de profundidad

(b) Cálculo Y_{11} con entrada X de 2 canales de profundidad



(c) Cálculo Y_{11} con entrada X de 3 canales de profundidad

Figura 2.9: Propagación hacia delante en una capa convolucional

De esta forma, en la Figura 2.9 se muestra como una convolución con C canales de profundidad se descompone en la suma de C convoluciones con un canal de profundidad.
Para una entrada X con C canales de profundidad, convolución(X,K) = convolución(X^1, K^1) + convolución(X^2, K^2) + ... + convolución(X^C, K^C).

Por último, en cada "paso" del "deslizamiento" se suma un solo sesgo y se aplica una sola vez la función de activación, independientemente del número de canales de profundidad que presente la entrada X.

2.6.2. Capa de agrupación o pooling

Componentes

Propagación hacia delante

2.6.3. Capa de aplanado

Propagación hacia delante

Capítulo 3

Aportaciones

3.1. Redes Neuronales Totalmente Conectadas

3.1.1. Gradiente de la función de pérdida respecto a SoftMax, [1] [2]

Siendo Z_i la entrada i de la última capa, se denotará mediante O_i una vez se le aplique SoftMax.

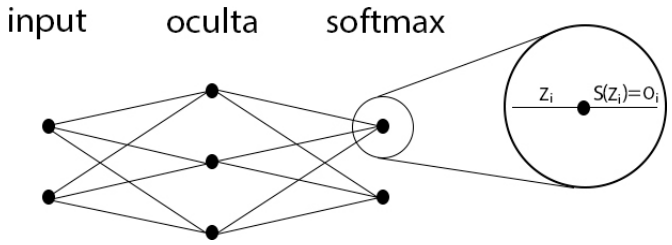


Figura 3.1: Estructura de una red con softmax

$$E = -\sum_{i=1}^{H}[y_i * log(O_i)] \tag{3.1}$$

Según esta notación, la función de error 2.3 se convierte en la fórmula 3.1.

Gradiente de la función de error

$$\frac{\partial E}{\partial Z_k} = \frac{\partial(-\sum_{i=1}^{H}[y_i * log(O_i)])}{\partial Z_k} = -\sum_{i=1}^{H}[\frac{\partial(y_i * log(O_i))}{\partial Z_k}] \tag{3.2}$$

Como y_i es independiente respecto a Z_k, se trata como una constante.

$$\frac{\partial E}{\partial Z_k} = -\sum_{i=1}^{H}[y_i * \frac{\partial(log(O_i))}{\partial Z_k}] \tag{3.3}$$

Se aplica la regla de la cadena, pues O_i no depende directamente de Z_k,.

$$\frac{\partial E}{\partial Z_k} = -\sum_{i=1}^{H}[y_i * \frac{\partial(log(O_i))}{\partial O_i} * \frac{\partial O_i}{\partial Z_k}] \tag{3.4}$$

$$\frac{\partial E}{\partial Z_k} = -\sum_{i=1}^{H}[\frac{y_i}{O_i} * \frac{\partial O_i}{\partial Z_k}] \tag{3.5}$$

Derivada de softmax respecto de su entrada, $\frac{\partial O_i}{\partial Z_k}$

Hay 2 casos posibles, $\frac{S(Z_i)}{Z_i}$ o $\frac{S(Z_i)}{Z_j}$, donde i $\neq$ j.

Caso $\frac{S(Z_i)}{Z_i}$

$$\frac{\partial f(x)}{\partial g(x)} = \frac{f'(x) * g(x) - g'(x) * f(x)}{g(x)^2} \tag{3.6}$$

$$S(z_i) = \frac{e^{Z_i}}{e^{Z_1} + ... + e^{Z_H}} \tag{3.7}$$

$$\frac{\partial S(Z_1)}{\partial Z_1} = \frac{[\frac{\partial e^{Z_1}}{\partial Z_1} * (e^{Z_1} + ... + e^{Z_H})] - [\frac{\partial(e^{Z_1}+...+e^{Z_H})}{\partial Z_1} * e^{Z_1}]}{(e^{Z_1} + ... + e^{Z_H})^2} \tag{3.8}$$

Se aplica $\frac{\partial e^{Z_1}}{Z_1} = e^{Z_1}$

$$\frac{\partial S(Z_1)}{\partial Z_1} = \frac{[e^{Z_1} * \sum_{i=1}^{H} e^{Z_i}] - [e^{Z_1} * e^{Z_1}]}{(\sum_{i=1}^{H} e^{Z_i})^2} \tag{3.9}$$

$$\tag{3.10}$$

Se saca factor común e^{Z_1}

$$\frac{\partial S(Z_1)}{\partial Z_1} = \frac{e^{Z_1}([\sum_{i=1}^{H} e^{Z_i}] - e^{Z_1})}{(\sum_{i=1}^{H} e^{Z_i})^2} \tag{3.11}$$

$$\frac{\partial S(Z_1)}{\partial Z_1} = \frac{e^{Z_1}}{\sum_{i=1}^{H} e^{Z_i}} * \frac{[\sum_{i=1}^{H} e^{Z_i}] - e^{Z_1}}{\sum_{i=1}^{H} e^{Z_i}} \tag{3.12}$$

Se recuerda que $\frac{\sum_{i=1}^{H} e^{Z_i}}{\sum_{i=1}^{H} e^{Z_i}} = 1$ y que $\mathrm{S}(Z_1) = \frac{e^{Z_1}}{\sum_{i=1}^{H} e^{Z_i}}$

$$\frac{\partial S(Z_1)}{\partial Z_1} = S(Z_1) * (1 - S(Z_1)) \tag{3.13}$$

Caso $\frac{S(Z_i)}{Z_j}$, con i $\neq$ j

$$\frac{\partial S(Z_2)}{\partial Z_1} = \frac{[\frac{\partial e^{Z_2}}{\partial Z_1} * (e^{Z_1} + ... + e^{Z_H})] - [\frac{\partial (e^{Z_1} + ... + e^{Z_H})}{\partial Z_1} * e^{Z_2}]}{(e^{Z_1} + ... + e^{Z_H})^2} \tag{3.14}$$

$$\frac{\partial S(Z_2)}{\partial Z_1} = \frac{[0 * [\sum_{i=1}^{H} e^{Z_i}]] - [e^{Z_1} * e^{Z_2}]}{(\sum_{i=1}^{H} e^{Z_i})^2} \tag{3.15}$$

$$\frac{\partial S(Z_2)}{\partial Z_1} = \frac{-e^{Z_1} * e^{Z_2}}{(\sum_{i=1}^{H} e^{Z_i})^2} \tag{3.16}$$

$$\frac{\partial S(Z_2)}{\partial Z_1} = \frac{-e^{Z_1}}{\sum_{i=1}^{H} e^{Z_i}} * \frac{e^{Z_2}}{\sum_{i=1}^{H} e^{Z_i}} \tag{3.17}$$

$$\frac{\partial S(Z_2)}{\partial Z_1} = -S(Z_1) * S(Z_2) \tag{3.18}$$

Combinación de casos

De esta forma, tendremos que dividir el proceso en 2 partes, cuando i sea igual a j, y cuando i $\neq$ j, perteneciendo a la primera todos los casos menos uno.
Parte izquierda cuando i!=k, parte derecha cuando i=k.
Retomamos la fórmula 3.5, aplicando 3.18 en la parte izquierda y 3.13 en la derecha.

$$\frac{\partial E}{\partial Z_k} = -[\sum_{i!=k}^{H} [\frac{y_i}{O_i} * -O_i * O_k] + \frac{y_k}{O_k} * O_k * (1 - O_k)] \tag{3.19}$$

Se simplifica O_i en la parte izquierda y O_k en la derecha.

$$\frac{\partial E}{\partial Z_k} = -[\sum_{i!=k}^{H} [-y_i * O_k] + [y_k * (1 - O_k)]] \tag{3.20}$$

Se extrae O_k de la suma, pues es independiente respecto al índice i

$$\frac{\partial E}{\partial Z_k} = -[-O_k \sum_{i!=k}^{H} -y_i + [y_k * (1 - O_k)]] \tag{3.21}$$

Simplificación One-Hot

Al emplear la codificación one-hot en Y, se sabe que la suma de sus elementos es igual a 1.

$$\sum_{i=1}^{H} y_i = 1 \tag{3.22}$$

$$\sum_{i!=k}^{H} y_i = \sum_{i=1}^{H} y_i - y_k = 1 - y_k \tag{3.23}$$

Se emplea 3.23 para simplificar la suma en 3.21.

$$\frac{\partial E}{\partial Z_k} = [O_k * (1 - y_k)] - [y_k * (1 - O_k)] \tag{3.24}$$

$$\frac{\partial E}{\partial Z_k} = O_k - O_k * y_k - y_k + O_k * y_k \tag{3.25}$$

Se simplifica $O_k * y_k$.

$$\frac{\partial E}{\partial Z_k} = O_k - y_k = gradiente\_Z_k \tag{3.26}$$

3.1.2. BackPropagation con 1 capa oculta [3] [4]

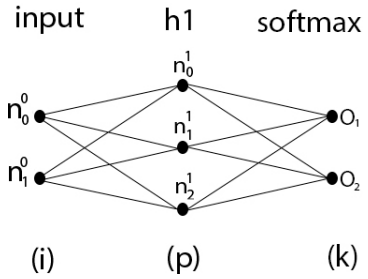


Figura 3.2: Red Neuronal totalmente conectada con 1 capa oculta

La Figura 3.2 se compone de 'puntos' interconectados mediante líneas, representando neuronas y pesos que las conectan respectivamente. Cada punto corresponde a una neurona, y cada línea a un peso.
La Figura 3.2 presenta 3 capas (input, h1, softmax) que corresponden a capa de entrada, capa oculta h_1, y capa de salida respectivamente. El superíndice indica la capa a la que pertenece una neurona o peso, mientras que el

subíndice indica el número del mismo en su respectiva capa. En el caso de los pesos, se requieren 2 subíndices para identificar a cada uno (pues un peso une 2 neuronas).
La capa de entrada se compone de 2 neuronas (n_0^0 y n_1^0).
La capa oculta h_1 tiene 3 neuronas (n_0^1, n_1^1, y n_2^1)
El peso W_{jk}^i referencia al peso que une las neuronas n_j^i y n_k^{i+1}.
De forma adicional, se recuerda que Z_i representa la entrada i de la capa SoftMax, y O_i su salida.

Capa SoftMax

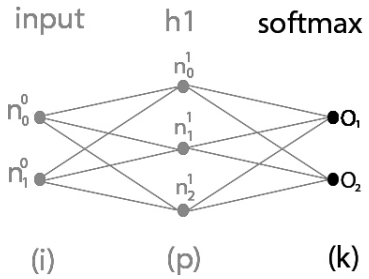


Figura 3.3: Imagen de backpropagation en la capa softmax

Sea la neurona n_j^i, se define como a_j^i el valor de dicha neurona antes de aplicar sobre ella su función de activación asociada, y z_j^i el obtenido tras aplicarla.

Tal y como se calculó previamente, el gradiente de la función de pérdida respecto a cada Z_i viene dado por la fórmula 3.26.

Pesos capas h1-SoftMax

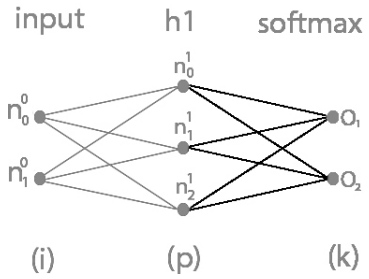


Figura 3.4: Imagen de backpropagation en los pesos entre la capa oculta y la capa SoftMax

Una vez calculado el gradiente hasta la capa softmax, se puede calcular el gradiente respecto a cada peso W_{pk}^1 que se encuentra conectado a esta

desde la capa anterior. Es decir, para cada $h_p^1 \in h_1$, se calcula $\frac{dE(x)}{dW_{pk}^1}$. Usando la regla de la cadena, equivale a realizar lo siguiente:

$$\frac{\partial Z_k}{\partial W_{pk}^1} = \frac{\partial(z_p^1 * W_{pk}^1 + b_k^2)}{\partial W_{pk}^1} = z_p^1 \tag{3.27}$$

$$\frac{\partial E(x)}{\partial W_{pk}^1} = gradiente\_Z_k * \frac{\partial Z_k}{\partial W_{pk}^1} = gradiente\_Z_k * z_p^1 \tag{3.28}$$

Sesgos capa softmax

$$\frac{\partial E}{\partial b_k^2} = \frac{\partial E}{\partial Z_k} * \frac{\partial Z_k}{b_k^2} \tag{3.29}$$

$$\frac{\partial Z_k}{\partial b_k^2} = \frac{\partial([\sum_{c=1}^{P} z_c^1 * W_{pk}^1] + b_k^2)}{\partial b_k^2} = 1 \tag{3.30}$$

$$\frac{\partial E}{\partial b_k^2} = gradiente\_Z_k \tag{3.31}$$

Capa oculta h1

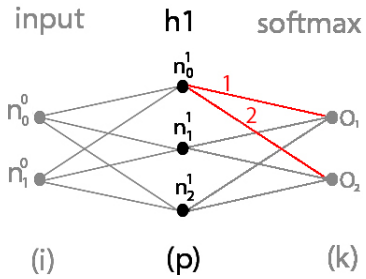


Figura 3.5: Imagen de los 'caminos' desde la capa softmax hasta n_0^1

En la figura 3.5 se muestra como hay más de un 'camino' desde la capa softmax hasta n_p^1. Por tanto, para obtener el gradiente de la pérdida respecto a n_p^1, habría que calcular la suma de todos los 'caminos' hacia este.

$$\frac{\partial E_{total}}{\partial a_p^1} = \sum_{k=1}^{K} \frac{\partial E_k}{\partial a_p^1} = \sum_{k=1}^{K} gradiente\_Z_k * \frac{\partial Z_k}{\partial z_p^1} * \frac{\partial z_p^1}{\partial a_p^1} \tag{3.32}$$

$$\frac{\partial Z_k}{\partial z_p^1} = \frac{\partial([\sum_{c=1}^{P} z_c^1 * W_{ck}^1] + b_k^2)}{\partial z_p^1} = W_{pk}^1 \tag{3.33}$$

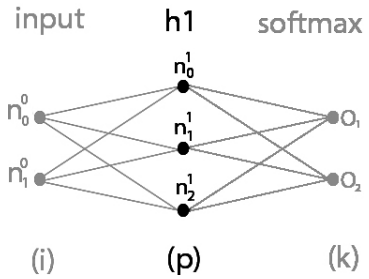


Figura 3.6: Imagen de backpropagation en la capa oculta h1

En la capa oculta h1 se emplea la función de activación sigmoide.

$$sigmoide(x) = \frac{1}{1 + e^{-x}} \tag{3.34}$$

$$sigmoide'(x) = \frac{sigmoide(x)}{1 - sigmoide(x)} \tag{3.35}$$

De esta forma,

$$\frac{\partial z_p^1}{\partial a_p^1} = \frac{\partial sigmoide(a_p^1)}{\partial a_p^1} = sigmoide(a_p^1) * (1 - sigmoide(a_p^1)) \tag{3.36}$$

Se retoma la fórmula 3.32 mediante la aplicación de 3.33 y 3.36

$$\frac{\partial E_{total}}{\partial a_p^1} = \sum_{k=1}^{K} gradiente\_Z_k * W_{pk}^1 * sigmoide(a_p^1) * (1 - sigmoide(a_p^1)) \tag{3.37}$$

$$\frac{\partial E_{total}}{\partial a_p^1} = gradiente\_h1_p \tag{3.38}$$

Pesos capas input-h1

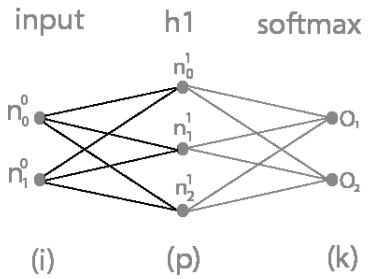


Figura 3.7: Imagen de backpropagation en los pesos entre la capa input y la capa oculta h1

$$\frac{\partial a_p^1}{\partial W_{ip}^0} = \frac{\partial [\sum_{c=1}^{I} z_c^0 * W_{cp}^0] + b_p^1)}{\partial W_{ip}^0} = z_i^0 \tag{3.39}$$

$$\frac{\partial E}{\partial W_{ip}^0} = \frac{\partial E_{total}}{\partial a_p^1} * \frac{\partial a_p^1}{W_{ip}^0} \tag{3.40}$$

$$\frac{\partial E(x)}{\partial W_{ip}^0} = gradiente\_h1_p * \frac{\partial a_p^1}{\partial W_{ip}^0} = gradiente\_h1_p * z_i^0 \tag{3.41}$$

Sesgos capa h1

$$\frac{\partial E}{\partial b_p^1} = \frac{\partial E_{total}}{\partial a_p^1} * \frac{\partial a_p^1}{b_p^1} \tag{3.42}$$

$$\frac{\partial a_p^1}{\partial b_p^1} = \frac{\partial ([\sum_{c=1}^{I} z_c^0 * W_{ip}^0] + b_p^1)}{\partial b_p^1} = 1 \tag{3.43}$$

$$\frac{\partial E}{\partial b_p^1} = gradiente\_h1_p \tag{3.44}$$

Capa input

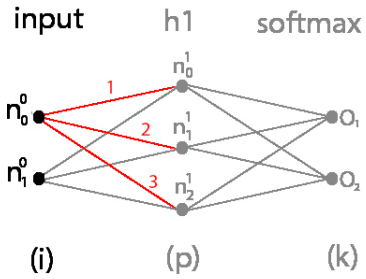


Figura 3.8: Imagen de los 'caminos' desde la capa oculta h1 hasta n_0^0

$$\frac{\partial E_{total}}{\partial a_i^0} = \sum_{p=1}^{P} \frac{\partial E_{total}}{\partial a_p^1} * \frac{\partial a_p^1}{\partial z_i^0} * \frac{\partial z_i^0}{\partial a_i^0} \tag{3.45}$$

$$\frac{\partial a_p^1}{\partial z_i^0} = \frac{\partial([\sum_{c=1}^{I} z_c^0 * W_{ip}^0] + b_p^1)}{\partial z_i^0} = W_{ip}^0 \tag{3.46}$$

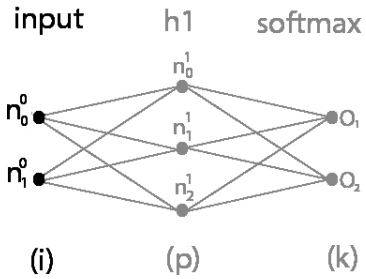


Figura 3.9: Imagen de backpropagation en la capa input

Como la capa input no presenta ninguna función de activación asociada, z_i^0 es igual a_i^0.

$$\frac{\partial z_i^0}{\partial a_i^0} = 1 \tag{3.47}$$

$$\frac{\partial E_{total}}{\partial a_i^0} = \sum_{p=1}^{P} gradiente\_h1_p \tag{3.48}$$

3.1.3. Retropropagación con 2 capas ocultas

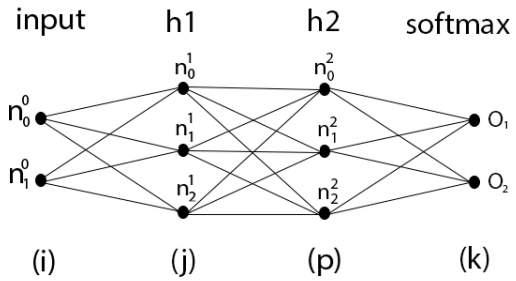


Figura 3.10: Red Neuronal totalmente conectada con 2 capas ocultas

Tal y como muestra la Figura 3.10, en este caso se emplea una red totalmente conectada con 2 capas ocultas (h1 y h2).

Capa SoftMax

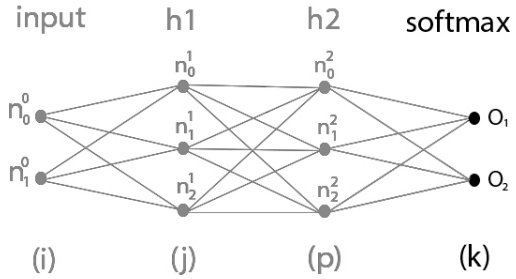


Figura 3.11: Imagen de backpropagation en la capa softmax

Al igual que en el apartado anterior (3.1.2), el gradiente de la función de pérdida respecto a cada Z_i viene dado por la fórmula 3.26.

Pesos capas h2-SoftMax

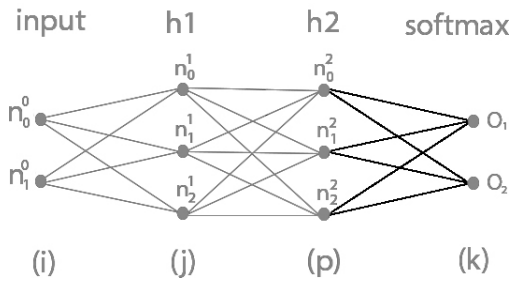


Figura 3.12: Imagen de backpropagation en los pesos entre la capa oculta h2 y la capa SoftMax

Se realiza el cálculo del gradiente de la función de pérdida respecto a cada peso W^2_{pk} que une las neuronas de la capa oculta h2 con las de la capa de salida softmax.

$$\frac{\partial Z_k}{\partial W^2_{pk}} = \frac{\partial(z^2_p * W^2_{pk} + b^3_k)}{\partial W^2_{pk}} = z^2_p \tag{3.49}$$

$$\frac{\partial E(x)}{\partial W^2_{pk}} = gradiente\_Z_k * \frac{\partial Z_k}{\partial W^2_{pk}} = gradiente\_Z_k * z^2_p \tag{3.50}$$

Como es de esperar, las fórmulas 3.49 y 3.50 son casi idénticas a 3.27 y 3.28 respectivamente, salvo por el superíndice empleado ($1 \neq 2$). Esto tiene sentido pues esta parte también es común al apartado anterior.

Sesgos capa softmax

$$\frac{\partial E}{\partial b^3_k} = \frac{\partial E}{\partial Z_k} * \frac{\partial Z_k}{b^3_k} \tag{3.51}$$

$$\frac{\partial Z_k}{\partial b^3_k} = \frac{\partial([\sum_{c=1}^{P} z^2_c * W^2_{pk}] + b^3_k)}{\partial b^3_k} = 1 \tag{3.52}$$

$$\frac{\partial E}{\partial b^3_k} = gradiente\_Z_k \tag{3.53}$$

Capa oculta h2

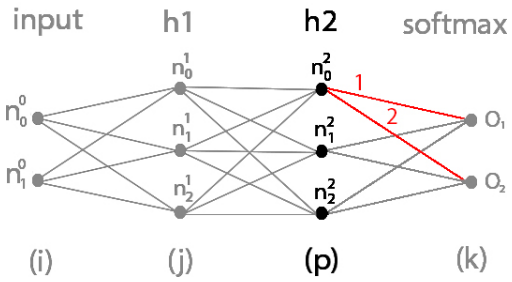


Figura 3.13: Imagen de los 'caminos' desde la capa softmax hasta n_0^2

Tal y como se comentó anteriormente, hay más de un 'camino' desde la capa softmax hasta n_p^2. Por tanto, para obtener el gradiente de la pérdida respecto a cada n_p^2, habría que calcular la suma de todos los ellos.

$$\frac{\partial E_{total}}{\partial a_p^2} = \sum_{k=1}^{K} \frac{\partial E_k}{\partial a_p^2} = \sum_{k=1}^{K} gradiente\_Z_k * \frac{\partial Z_k}{\partial z_p^2} * \frac{\partial z_p^2}{\partial a_p^2} \tag{3.54}$$

$$\frac{\partial Z_k}{\partial z_p^2} = \frac{\partial([\sum_{c=1}^{P} z_c^2 * W_{ck}^2] + b_k^3)}{\partial z_p^2} = W_{pk}^2 \tag{3.55}$$

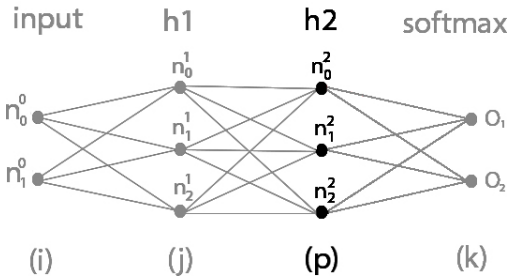


Figura 3.14: Imagen de backpropagation en la capa oculta h2

En la capa oculta h2 se emplea la función de activación sigmoide.

$$sigmoide(x) = \frac{1}{1 + e^{-x}} \tag{3.56}$$

$$sigmoide'(x) = \frac{sigmoide(x)}{1 - sigmoide(x)} \tag{3.57}$$

De esta forma,

$$\frac{\partial z_p^2}{\partial a_p^2} = \frac{\partial sigmoide(a_p^2)}{\partial a_p^2} = sigmoide(a_p^2) * (1 - sigmoide(a_p^2)) \tag{3.58}$$

Se retoma la fórmula 3.54 mediante la aplicación de 3.55 y 3.58.

$$\frac{\partial E_{total}}{\partial a_p^2} = \sum_{k=1}^{K} gradiente\_Z_k * W_{pk}^2 * sigmoide(a_p^2) * (1 - sigmoide(a_p^2)) \tag{3.59}$$

$$\frac{\partial E_{total}}{\partial a_p^2} = gradiente\_h2_p \tag{3.60}$$

Una vez más, la fórmula obtenida (3.60) coindice con la calculada previamente (3.38).

Pesos capas h1-h2

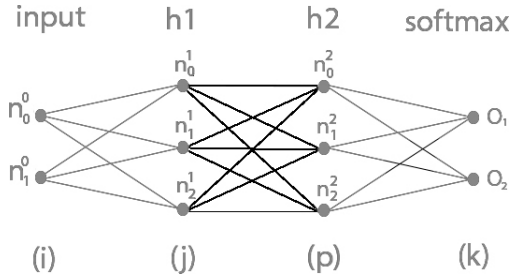


Figura 3.15: Imagen de backpropagation en los pesos entre las capas ocultas h1 y h2

$$\frac{\partial a_p^2}{\partial W_{jp}^1} = \frac{\partial [\sum_{c=1}^{J} z_c^1 * W_{cp}^1] + b_p^2)}{\partial W_{jp}^1} = z_j^1 \tag{3.61}$$

$$\frac{\partial E}{\partial W_{jp}^1} = \frac{\partial E_{total}}{\partial a_p^2} * \frac{\partial a_p^2}{W_{jp}^1} \tag{3.62}$$

$$\frac{\partial E(x)}{\partial W_{jp}^1} = gradiente\_h2_p * \frac{\partial a_p^2}{\partial W_{jp}^1} = gradiente\_h2_p * z_j^1 \tag{3.63}$$

La fórmula 3.63 vuelve a coincidir con 3.41

Sesgos capa h2

$$\frac{\partial E}{\partial b_p^2} = \frac{\partial E_{total}}{\partial a_p^2} * \frac{\partial a_p^2}{b_p^2} \tag{3.64}$$

$$\frac{\partial a_p^2}{\partial b_p^2} = \frac{\partial([\sum_{c=1}^{J} z_c^1 * W_{jp}^1] + b_p^2)}{\partial b_p^2} = 1 \tag{3.65}$$

$$\frac{\partial E}{\partial b_p^2} = gradiente\_h2_p \tag{3.66}$$

La fórmula 3.66 coincide con 3.44.

Capa oculta h1

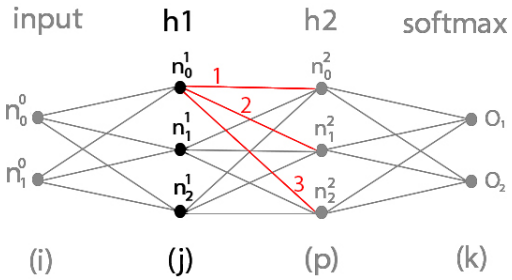


Figura 3.16: Imagen de los 'caminos' desde la capa softmax hasta n_0^1

De igual forma que se realizó en la capa h2, se calcula la suma de todos los 'caminos' hacia cada neurona n_j^1.

$$\frac{\partial E_{total}}{\partial a_j^1} = \sum_{k=1}^{K} \frac{\partial E_k}{\partial a_j^1} = \sum_{p=1}^{P} gradiente\_h2_p * \frac{\partial a_p^2}{\partial z_j^1} * \frac{\partial z_j^1}{\partial a_j^1} \tag{3.67}$$

$$\frac{\partial a_p^2}{\partial z_j^1} = \frac{\partial([\sum_{c=1}^{J} z_c^1 * W_{cp}^1] + b_p^2)}{\partial z_j^1} = W_{jp}^1 \tag{3.68}$$

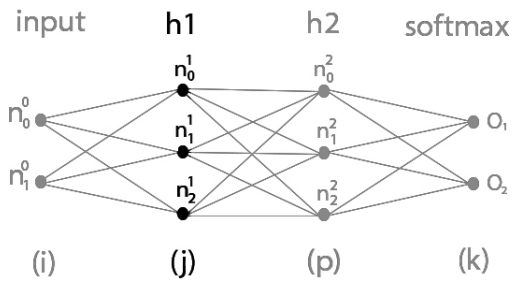


Figura 3.17: Imagen de backpropagation en la capa oculta h1

En la capa oculta h1 se emplea la función de activación ReLU.

$$ReLU(x) = max(0, x) \tag{3.69}$$

$$ReLU'(x) = 1 \; si \; x > 0, \; 0 \; en \; caso \; contrario \tag{3.70}$$

De esta forma,

$$\frac{\partial z_j^1}{\partial a_j^1} = 1 \; si \; x > 0, \; 0 \; en \; caso \; contrario \tag{3.71}$$

$$\frac{\partial E_{total}}{\partial a_j^1} = \sum_{p=1}^{P} gradiente\_h2_p * W_{jp}^1 * ReLU'(a_j^1) \tag{3.72}$$

$$\frac{\partial E_{total}}{\partial a_j^1} = gradiente\_h1_j \tag{3.73}$$

El cálculo para obtener la fórmula 3.73 es muy parecido al realizado para 3.60, pero es "nuevo" respecto a la sección anterior, pues esta capa no existe en dicho caso.

Pesos capa input-h1

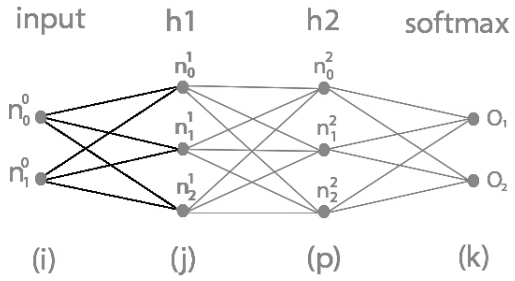


Figura 3.18: Imagen de backpropagation en los pesos entre la capa input y la capa oculta h1

$$\frac{\partial a_j^1}{\partial W_{ij}^0} = \frac{\partial([\sum_{c=1}^{I} z_c^0 * W_{cj}^0] + b_j^1)}{\partial W_{ij}^0} = z_i^0 \tag{3.74}$$

$$\frac{\partial E}{\partial W_{ij}^0} = \frac{\partial E_{total}}{\partial a_j^1} * \frac{\partial a_j^1}{W_{ij}^0} \tag{3.75}$$

$$\frac{\partial E(x)}{\partial W_{ij}^0} = gradiente\_h1_j * \frac{\partial a_j^1}{\partial W_{ij}^0} = gradiente\_h1_j * z_i^0 \tag{3.76}$$

Sesgos capa h1

$$\frac{\partial E}{\partial b_j^1} = \frac{\partial E_{total}}{\partial a_j^1} * \frac{\partial a_j^1}{b_j^1} \tag{3.77}$$

$$\frac{\partial a_j^1}{\partial b_j^1} = \frac{\partial([\sum_{c=1}^{I} z_c^0 * W_{ij}^0] + b_j^1)}{\partial b_j^1} = 1 \tag{3.78}$$

$$\frac{\partial E}{\partial b_j^1} = gradiente\_h1_j \tag{3.79}$$

La fórmula 3.79 coincide con 3.44.

Capa input

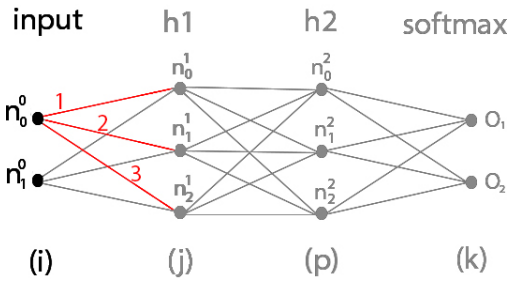


Figura 3.19: Imagen de los 'caminos' desde la capa oculta h1 hasta n_0^0

$$\frac{\partial E_{total}}{\partial a_i^0} = \sum_{j=1}^{J} \frac{\partial E_{total}}{\partial a_j^1} * \frac{\partial a_j^1}{\partial z_i^0} * \frac{\partial z_i^0}{\partial a_i^0} \tag{3.80}$$

$$\frac{\partial a_j^1}{\partial z_i^0} = \frac{\partial([\sum_{c=1}^{I} z_c^0 * W_{ij}^0] + b_j^1)}{\partial z_i^0} = W_{ij}^0 \tag{3.81}$$

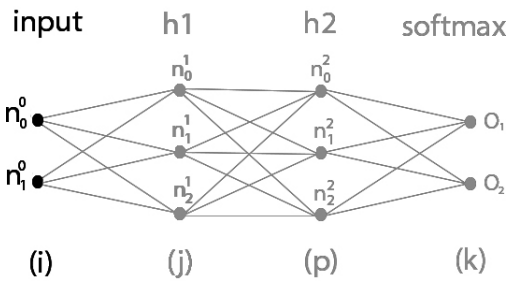


Figura 3.20: Imagen de backpropagation en la capa input

Como la capa input no presenta ninguna función de activación asociada, z_i^0 es igual a_i^0.

$$\frac{\partial z_i^0}{\partial a_i^0} = 1 \tag{3.82}$$

$$\frac{\partial E_{total}}{\partial a_i^0} = \sum_{p=1}^{P} gradiente\_h1_p \tag{3.83}$$

3.1.4. Conclusiones

Se definen como capas ocultas "intermedias" todas menos la última de ellas. Tal y como se ha mostrado anteriormente, comparten la mayoría del cálculo en cuanto a retropopagación. De esta forma, se puede dividir una red neuronal totalmente conectada en 4 grupos {capa input, capas ocultas intermedias, última capa oculta, capa de salida o capa softmax}.
A continuación se realiza el cálculo necesario para la retropropagación de una capa de neuronas l determinada. Suponemos que la capa l+1 tiene Q neuronas, la capa l-1 tiene K neuronas, y todas las capas ocultas intermedias usan ReLU como función de activación.

Gradiente respecto a la entrada

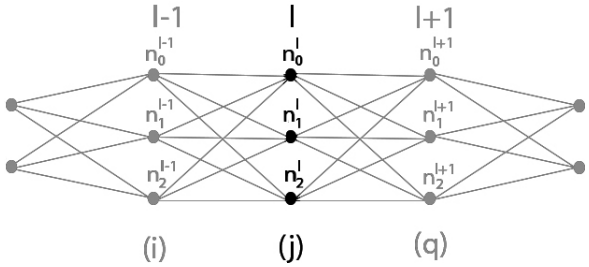


Figura 3.21: Imagen de backpropagation en la capa l

$$\frac{\partial E_{total}}{\partial a_j^l} = \sum_{q=1}^{Q} \frac{\partial E_{total}}{\partial a_q^{l+1}} * \frac{\partial a_q^{l+1}}{\partial z_j^l} * \frac{\partial z_j^l}{\partial a_j^l} \tag{3.84}$$

$$\frac{\partial a_j^{l+1}}{\partial z_j^l} = \frac{\partial([\sum_{c=1}^{K} z_c^l * W_{ij}^l] + b_j^{l+1})}{\partial z_j^l} = W_{ij}^l \tag{3.85}$$

$$\frac{\partial z_j^l}{\partial a_j^l} = ReLU'(a_j^l) \tag{3.86}$$

$$\frac{\partial E_{total}}{\partial a_j^l} = \sum_{q=1}^{Q} gradiente\_h_{l+1_q} * W_{ij}^l * ReLU'(a_j^l) \tag{3.87}$$

$$\frac{\partial E_{total}}{\partial a_j^l} = gradiente\_h_{l_j} \tag{3.88}$$

Gradiente respecto a los pesos

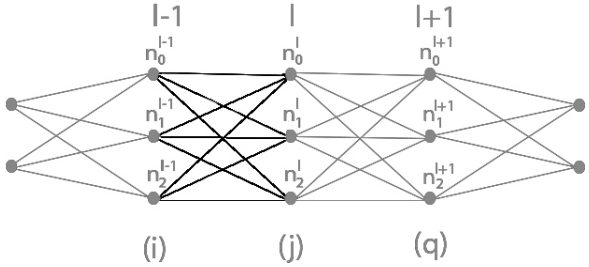


Figura 3.22: Imagen de backpropagation en los pesos entre la capa l-1 y l

$$\frac{\partial E}{\partial W_{ij}^{l-1}} = \frac{\partial E_{total}}{\partial a_j^l} * \frac{\partial a_j^l}{W_{ij}^{l-1}} \tag{3.89}$$

$$\frac{\partial a_j^l}{\partial W_{ij}^{l-1}} = \frac{\partial([\sum_{c=1}^{K} z_c^{l-1} * W_{cj}^{l-1}] + b_j^l)}{\partial W_{ij}^{l-1}} = z_i^{l-1} \tag{3.90}$$

$$\frac{\partial E(x)}{\partial W_{ij}^{l-1}} = gradiente\_h_{l_j} * \frac{\partial a_j^l}{\partial W_{ij}^{l-1}} = gradiente\_h_{l_j} * z_i^{l-1} \tag{3.91}$$

Gradiente respecto a sesgos

$$\frac{\partial E}{\partial b_j^l} = \frac{\partial E_{total}}{\partial a_j^l} * \frac{\partial a_j^l}{b_j^l} \tag{3.92}$$

$$\frac{\partial a_j^l}{\partial b_j^l} = \frac{\partial([\sum_{c=1}^{K} z_c^{l-1} * W_{ij}^{l-1}] + b_j^l)}{\partial b_j^l} = 1 \tag{3.93}$$

$$\frac{\partial E}{\partial b_j^l} = gradiente\_h_{l_j} \tag{3.94}$$

Capítulo 4

Adaptación GPU

4.1. GPU en Redes Neuronales Totalmente Conectadas

4.2. GPU en Redes Neuronales Convolucionales

Capítulo 5

Comparación con distintas plataformas

5.1. cuDNN

Bibliografía

[1] mehran@mldawn.com. Back-propagation through cross-entropy softmax, 2021. https://www.mldawn.com/back-propagation-with-cross-entropy-and-softmax/ [Accessed:29/02/2024].

[2] mehran@mldawn.com. The derivative of softmax function wrt z, 2021. https://www.mldawn.com/the-derivative-of-softmaxz-function-w-r-t-z/ [Accessed:05/03/2024].

[3] Prakash Jay. Back-propagation is very simple. who made it complicated ?, 2017. https://medium.com/@14prakash/back-propagation-is-very-simple-who-made-it-complicated-97b794c97e5c [Accessed:06/03/2024].

[4] Chamanth mvs. No more confusion on back-propagation, 2022. https://pub.aimind.so/no-more-confusion-on-backpropagation-7adfc271539f [Accessed:07/03/2024].

[5] Izzat El Hajj Wen-emi W.Hwu, David B.kirk. *Programming Massively Parallel Processors*. Morgan Kaufmann, 50 Hampshire Street, 5th Floor, Cambridge, MA 02139, United States, 4 edition, 2022.

[6] Hsuan-Tien Lin Yaser S. Abu-Mostafa, Malik Magdon-Ismail. *Learning From Data*. California Institute of Technology Pasadena, CA 91125, USA, 1 edition, 2012.

[7] Douglas Karr. Unidad lineal rectificada, 2024. https://es.martech.zone/acronym/relu/ [Accessed:25/02/2024].

[8] Javi. La función sigmoide: Una herramienta clave en redes neuronales, 2023. https://jacar.es/la-funcion-sigmoide-una-herramienta-clave-en-redes-neuronales/ [Accessed:25/02/2024].

[9] Jason Brownlee. Softmax activation function with python, 2020. https://machinelearningmastery.com/softmax-activation-function-with-python/ [Accessed:24/02/2024].

[10] Saurav Maheshkar. What is cross entropy loss? a tutorial with code, 2023. https://wandb.ai/sauravmaheshkar/cross-entropy/reports/What-Is-Cross-Entropy-Loss-A-Tutorial-With-Code--VmlldzoxMDA5NTMx#:~:text=Cross%20entropy%20loss%20is%20a,close%20to%200%20as%20possible. [Accessed:29/02/2024].

[11] Descenso del gradiente, 2024. https://es.wikipedia.org/wiki/Descenso_del_gradiente#:~:text=El%20descenso%20del%20gradiente%20o,en%20direcci%C3%B3n%20contraria%20al%20gradiente. [Accessed:26/02/2024].

[12] Gradiente, 2024. https://es.wikipedia.org/wiki/Gradiente [Accessed:26/02/2024].

[13] Robert Kwiatkowski. Gradient descent algorithm — a deep dive, 2021. https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21#:~:text=Gradient%20descent%20(GD)%20is%20an,e.g.%20in%20a%20linear%20regression). [Accessed:26/02/2024].

[14] ml4a. How neural networks are trained, 2020. https://ml4a.github.io/ml4a/how_neural_networks_are_trained/ [Accessed:27/02/2024].

[15] Jason Brownlee. Weight initialization for deep learning neural networks, 2021. https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/ [Accessed:14/03/2024].

[16] Sandeep Jain. Kaiming initialization in deep learning, 2023. https://www.geeksforgeeks.org/kaiming-initialization-in-deep-learning/ [Accessed:14/03/2024].

[17] Adrian Rosebrock. Understanding weight initialization for neural networks, 2021. https://pyimagesearch.com/2021/05/06/understanding-weight-initialization-for-neural-networks/ [Accessed:14/03/2024].

[18] Yahia Zakaria. Initial bias values for a neural network, 2017. https://stackoverflow.com/questions/44883861/

initial-bias-values-for-a-neural-network [Accessed:14/03/2024].

[19] Glen Meyerowitz. Bias initialization in a neural network, 2018. https://medium.com/@glenmeyerowitz/bias-initialization-in-a-neural-network-2e5d26fed0f0 [Accessed:14/03/2024].

[20] Afshine Amidi y Shervine Amidi. Convolutional neural networks cheatsheet, 2018. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks [Accessed:19/03/2024].

[21] Stanford. Convolutional neural networks (cnns / convnets), 2020. https://cs231n.github.io/convolutional-networks/ [Accessed:19/03/2024].