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

1. Introducción **1**
- 1.1. Resumen . . . 1
- 1.2. Estado del arte . . . 1
- 1.3. Objetivos . . . 1

2. Conceptos previos **3**
- 2.1. Machine Learning . . . 3
- 2.2. Componentes necesarios para el aprendizaje supervisado . . . 3
- 2.3. División de datos en entrenamiento y test . . . 4
- 2.4. Tipos de aprendizaje . . . 4
  - 2.4.1. Aprendizaje Supervisado . . . 4
  - 2.4.2. Aprendizaje No Supervisado . . . 4
  - 2.4.3. Aprendizaje Por Refuerzo . . . 4
- 2.5. Redes Neuronales Totalmente Conectadas . . . 5
  - 2.5.1. Neurona . . . 5
  - 2.5.2. Estructura por capas . . . 5
  - 2.5.3. Funciones de activación . . . 6
  - 2.5.4. One-hot encoding . . . 8
  - 2.5.5. Función de error o pérdida . . . 8
  - 2.5.6. Descenso del gradiente . . . 8
  - 2.5.7. Tipos de codificaciones . . . 9
  - 2.5.8. Propagación hacia delante con softmax . . . 9
- 2.6. Redes Neuronales Convolucionales . . . 10
  - 2.6.1. Tipos de capas . . . 10
  - 2.6.2. Estructura por capas . . . 10
  - 2.6.3. ForwardPropagation . . . 10

3. Aportaciones **11**
- 3.1. Redes Neuronales Totalmente Conectadas . . . 11
  - 3.1.1. BackPropagation con 1 capa oculta . . . 11

4. Adaptación GPU **17**
4.1. GPU en Redes Neuronales Totalmente Conectadas 17
4.2. GPU en Redes Neuronales Convolucionales 17

5. Comparación con distintas plataformas **19**
5.1. cuDNN 19

Índice de figuras

2.1. Imagen de una neurona 5
2.2. Imagen de una capa de neuronas 5
2.3. Imagen de la función de activación ReLU 6
2.4. Imagen de la función de activación Sigmoide 7
2.5. Imagen de la función de activación SoftMax 7
2.6. Ejemplo de funcionamiento del descenso del gradiente 8

3.1. Red Neuronal totalmente conectada con 1 capa oculta 11
3.2. Imagen de backpropagation en la capa output 12
3.3. Imagen de backpropagation en los pesos entre la capa oculta y la capa output 14
3.4. Imagen de backpropagation en la capa oculta h1 14
3.5. Imagen de backpropagation en los pesos entre la capa input y la capa oculta 15
3.6. Imagen de backpropagation en la capa input 16

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. [1]
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. [2]
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. [2]

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. [2]
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=$\{x_1, x_2, ..., x_N\}$. [2]

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. [2]

2.5. Redes Neuronales Totalmente Conectadas

2.5.1. Neurona

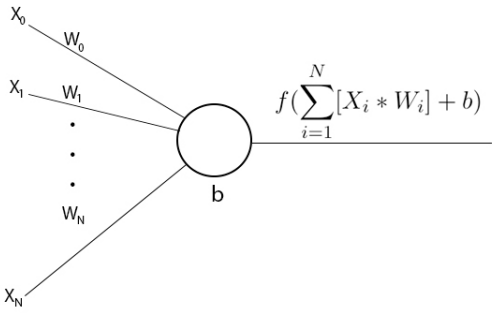

Figura 2.1: Imagen de una neurona

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

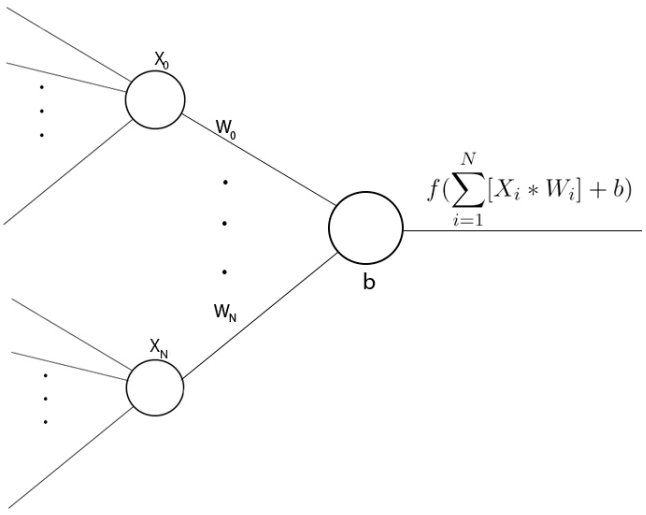

Figura 2.2: Imagen de una capa de neuronas

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

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

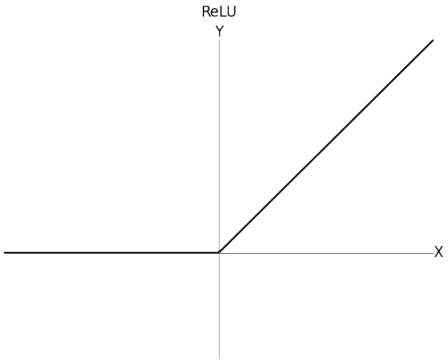

Figura 2.3: Imagen de la función de activación ReLU

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. [3]

Sigmoide

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

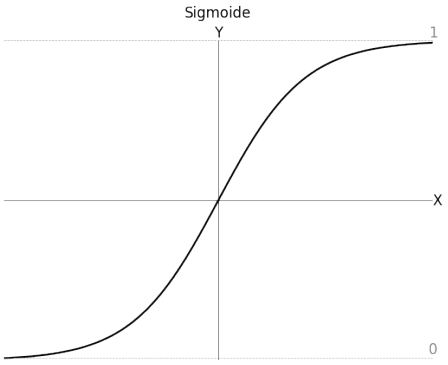

Figura 2.4: Imagen de la función de activación Sigmoide

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. [4]

SoftMax

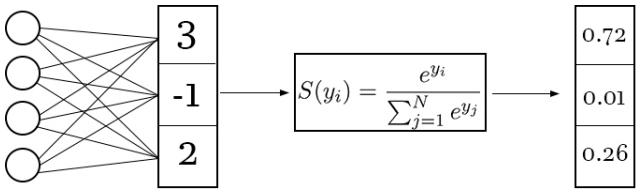

Figura 2.5: Imagen de la función de activación SoftMax

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. [5]

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. [6]

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

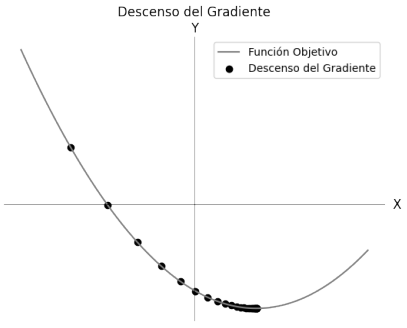

Figura 2.6: Ejemplo de funcionamiento del descenso del gradiente

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. [7] [8] [9]

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). [6]

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

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

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. [10]

2.5.7. 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.

2.5.8. 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:

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

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

[11]

2.6. Redes Neuronales Convolucionales

2.6.1. Tipos de capas

2.6.2. Estructura por capas

2.6.3. ForwardPropagation

Capítulo 3

Aportaciones

3.1. Redes Neuronales Totalmente Conectadas

3.1.1. BackPropagation con 1 capa oculta

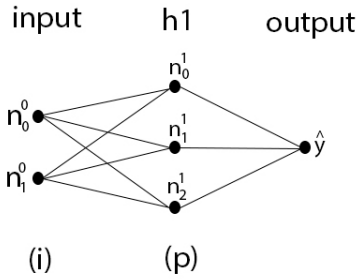

Figura 3.1: Red Neuronal totalmente conectada con 1 capa oculta

La Figura 3.1 se compone de puntos y líneas, representando neuronas y pesos que las conectan respectivamente. Cada punto es una neurona, y cada línea un peso.
La Figura 3.1 presenta 3 capas (input, h1, output) 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}.
Además, se denotará como $\hat{y}$ a la última neurona de la red, pues contendrá la predicción de la misma.

Capa output

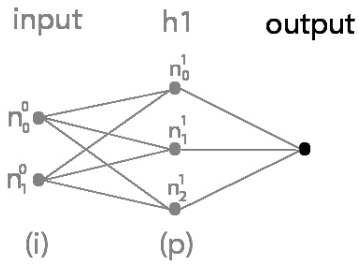

Figura 3.2: Imagen de backpropagation en la capa output

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. De forma adicional, se usará $\hat{a}$ y $\hat{z}$ para $\hat{y}$.

Así, la función de pérdida 2.4 se convierte en:

$$H(x) = -\frac{1}{N}\sum_{i=1}^{N}[y_i * log(\hat{z}_i) + (1 - y_i) * log(1 - \hat{z}_i)] \tag{3.1}$$

Para realizar el descenso del gradiente, se debe empezar calculando la derivada de la función de pérdida respecto a la predicción obtenida. Es decir, la derivada de la fórmula (3.1) respecto de las neuronas en la última capa de la red tras aplicar sus respectivas funciones de activación, que en este caso corresponde a $\hat{z}$.
Por simplicidad, podemos dividir esta derivada en 2 partes.
Parte izquierda:

$$f(x) = A * B \tag{3.2}$$
$$f'(x) = AB' + A'B \tag{3.3}$$
$$\frac{\partial y_i}{\partial \hat{z}_i} = 0 \tag{3.4}$$
$$\frac{\partial log(\hat{z}_i)}{\partial \hat{z}_i} = \frac{1}{\hat{z}_i} \tag{3.5}$$
$$\frac{\partial y_i * log(\hat{z}_i)}{\partial \hat{z}_i} = y_i * \frac{1}{\hat{z}_i} + 0 * log(\hat{z}_i) = \frac{y_i}{\hat{z}_i} \tag{3.6}$$

Parte derecha:

$$\frac{\partial(1-y_i)}{\partial\hat{z}_i} = 0 \tag{3.7}$$

$$\frac{\partial log(1-\hat{z}_i)}{\partial\hat{z}_i} = \frac{1}{1-\hat{z}_i} * (-1) \tag{3.8}$$

$$\frac{\partial(1-y_i) * log(1-\hat{z}_i)}{\partial\hat{z}_i} = (1-y_i) * \frac{1}{1-\hat{z}_i} * (-1) + 0 * log(1-\hat{z}_i) \tag{3.9}$$

$$\frac{\partial(1-y_i) * log(1-\hat{z}_i)}{\partial\hat{z}_i} = -\frac{1-y_i}{1-\hat{z}_i} \tag{3.10}$$

Finalmente, se obtiene:

$$\frac{\partial H(x)}{\partial\hat{z}_i} = -\frac{1}{N}\sum_{i=1}^{N}[\frac{y_i}{\hat{z}_i} - \frac{1-y_i}{1-\hat{z}_i}] \tag{3.11}$$

Función activación de la capa output

En la capa output se emplea la función de activación sigmoide.

$$sigmoide(x) = \frac{1}{1+e^{-x}} \tag{3.12}$$

$$sigmoide'(x) = \frac{sigmoide(x)}{1-sigmoide(x)} \tag{3.13}$$

De esta forma,

$$\frac{\partial\hat{z}}{\partial\hat{a}} = \frac{\partial sigmoide(\hat{a})}{\partial\hat{a}} = sigmoide(\hat{a}) * (1-sigmoide(\hat{a})) \tag{3.14}$$

Ahora, podemos calcular el gradiente completo hasta la capa output antes de aplicar su función de activación.

$$grad\_output = \frac{\partial H(x)}{\partial\hat{z}} * \frac{\partial\hat{z}}{\partial\hat{a}} = -\frac{1}{N}\sum_{i=1}^{N}[\frac{y_i}{\hat{z}_i} - \frac{1-y_i}{1-\hat{z}_i}] * \frac{\partial\hat{z}}{\partial\hat{a}} \tag{3.15}$$

$$grad\_output = -\frac{1}{N}\sum_{i=1}^{N}[\frac{y_i}{\hat{z}_i} - \frac{1-y_i}{1-\hat{z}_i}] * sigmoide(\hat{a}) * (1-sigmoide(\hat{a})) \tag{3.16}$$

Pesos capas h1-output

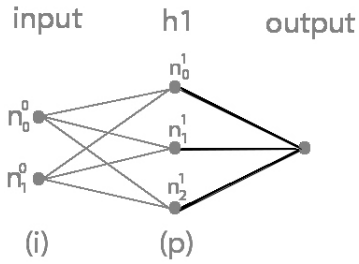

Figura 3.3: Imagen de backpropagation en los pesos entre la capa oculta y la capa output

Una vez calculado el gradiente hasta la capa output, se puede calcular el gradiente respecto a cada peso que se encuentra conectado a esta desde la capa anterior. Es decir, para cada $h_p^1 \in h_1$, se calcula $\frac{dH(x)}{dW_{p\hat{y}}^1}$. Usando la regla de la cadena, equivale a realizar lo siguiente:

$$\frac{\partial \hat{y}}{\partial W_{p\hat{y}}^1} = \frac{\partial n_p^1 * W_{p\hat{y}}^1}{\partial W_{p\hat{y}}^1} = n_p^1 \tag{3.17}$$

$$\frac{\partial H(x)}{\partial W_{p\hat{y}}^1} = \frac{\partial H(x)}{\partial \hat{y}} * \frac{\partial \hat{z}}{\partial \hat{a}} * \frac{\partial \hat{y}}{\partial W_{p\hat{y}}^1} = grad\_output * \frac{\partial \hat{y}}{\partial W_{p\hat{y}}} = grad\_output * n_p^1 \tag{3.18}$$

Capa oculta h1

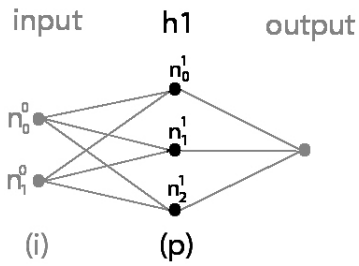

Figura 3.4: Imagen de backpropagation en la capa oculta h1

$$\frac{\partial \hat{y}}{\partial n_p^1} = \frac{\partial n_p^1 * \partial W_{p\hat{y}}^1}{\partial n_p^1} = \partial W_{p\hat{y}}^1 \tag{3.19}$$

$$deriv\_relu(x) = 0 \; si \; x <= 0, \; 1 \; en \; caso \; contrario \tag{3.20}$$

$$\frac{\partial z_p^1}{\partial a_p^1} = \frac{\partial relu(a_p^1)}{\partial a_p^1} = deriv\_relu(a_p^1) \tag{3.21}$$

$$\frac{\partial H(x)}{\partial n_p^1} = grad\_output * \frac{\partial \hat{y}}{\partial n_p^1} * \frac{\partial z_p^1}{\partial a_p^1} = grad\_output * W_{p\hat{y}}^1 * deriv\_relu(a_p^1) \tag{3.22}$$

$$\frac{\partial H(x)}{\partial n_p^1} = gradient\_n_p^1 \tag{3.23}$$

Pesos capas input-h1

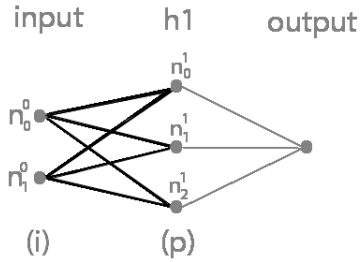

Figura 3.5: Imagen de backpropagation en los pesos entre la capa input y la capa oculta

$$\frac{\partial n_p^1}{\partial W_{ip}^0} = \frac{\partial n_i^0 * W_{ip}^0}{\partial W_{ip}^0} = n_i^0 \tag{3.24}$$

$$\frac{\partial H(x)}{\partial W_{ip}^0} = gradient\_n_p^1 * \frac{\partial n_p^1}{\partial W_{ip}^0} = gradient\_n_p^1 * n_i^0 \tag{3.25}$$

Capa input

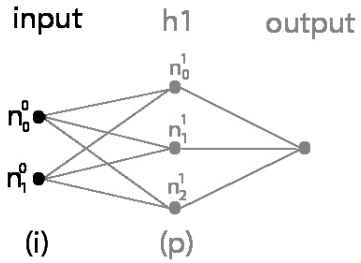

Figura 3.6: Imagen de backpropagation en la capa input

$$\frac{\partial n_p^1}{\partial n_i^0} = \frac{\partial n_i^0 * W_{ip}^0}{\partial n_i^0} = W_{ip}^0 \tag{3.26}$$

$$\frac{\partial z_i^0}{\partial a_i^0} = 1 \tag{3.27}$$

Ahora se realiza una sumatoria con todos los 'caminos posibles' hacia la misma neurona

$$\frac{\partial H(x)}{\partial n_i^0} = \sum_{p=0}^{h_1.size()} gradient\_n_p^1 * \frac{\partial n_p^1}{\partial n_i^0} * \frac{\partial z_i^0}{\partial a_i^0} \tag{3.28}$$

$$\frac{\partial H(x)}{\partial n_i^0} = \sum_{p=0}^{h_1.size()} gradient\_n_p^1 * W_{ip}^0 \tag{3.29}$$

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] 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.

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

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

[4] 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].

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

[6] 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].

[7] 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].

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

[9] 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].

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

[11] 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].