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. [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. Deep Learning

2.3. Tipos de aprendizaje

2.3.1. 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 tratará de impedirlo, siempre hay ruido en los datos empleados, implicando que algunas etiquetas de Y=$\{y_1, y_2, ..., y_N\}$ pueden ser erróneas.

2.3.2. Aprendizaje No Supervisado

En este tipo de aprendizaje, los datos no contienen ninguna información respecto a lo que debe predecir la red. De esta forma, el conjunto de datos D se compone exclusivamente de valores X=$\{x_1, x_2, ..., x_N\}$. [2]

2.3.3. 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]

2.4. Tipos de problemas en machine learning

2.4.1. Clasificación

2.4.2. Regresión

2.5. 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]

2.6. Entrenamiento

2.7. División de datos en entrenamiento y test

Para permitir la generalización del modelo, 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. Una vez finalizado, se accede al conjunto test y se visualiza el rendimiento del modelo sobre el mismo. Como los datos de test no se emplearon en todo el proceso de entrenamiento, aportan una estimación sobre la generalización del modelo fuera de la muestra con la que se entrenó.

2.8. Redes Neuronales Totalmente Conectadas

2.8.1. Neurona

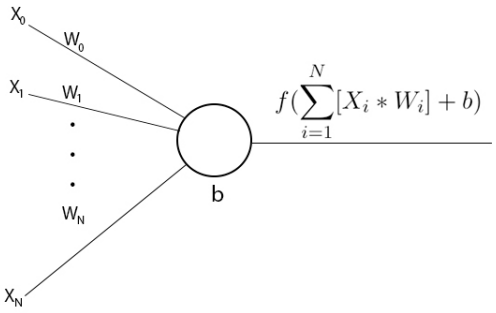

Figura 2.1: Imagen de una neurona

Una neurona se compone 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 $w_i \in W$.
La neurona los emplea para realizar una suma ponderada y posteriormente añadir un bias b, además de aplicar una función de activación f sobre el resultado obtenido.

2.8.2. Estructura por capas

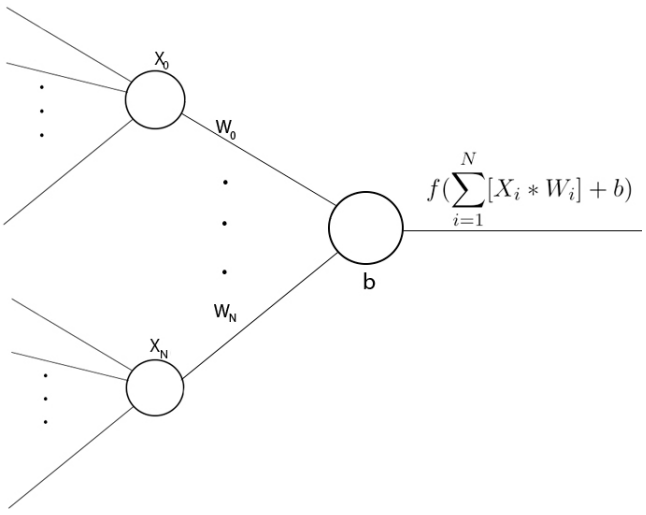

Figura 2.2: Imagen de una capa de neuronas

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.

2.8.3. Funciones de activación

ReLU

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

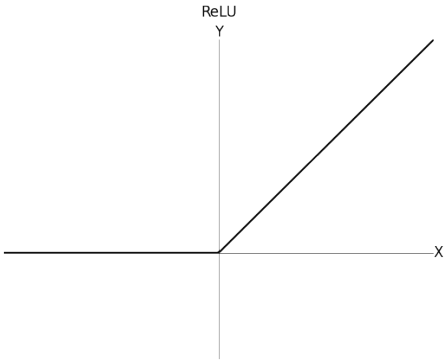

Figura 2.3: Imagen de la función de activación ReLU

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

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

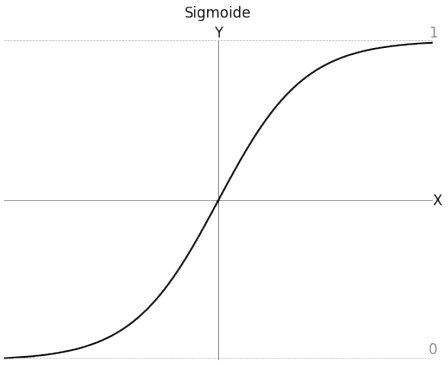

Figura 2.4: Imagen de la función de activación Sigmoide

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

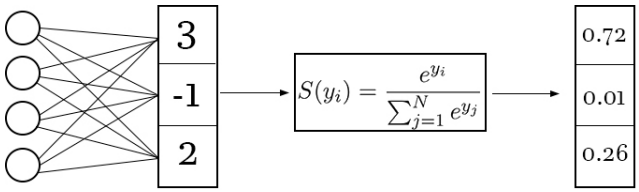

Figura 2.5: Imagen de la función de activación SoftMax

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]

2.8.4. One-hot encoding

2.8.5. Función de error o pérdida

Como solo tenemos dos clases y estamos en clasificación binaria, usaremos Sigmoid Cross Entropy Loss.

$$H(x) = -\frac{1}{N}\sum_{i=1}^{N}[y_i * log(\hat{y}_i) + (1 - y_i) * log(1 - \hat{y}_i)] \tag{2.3}$$

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

2.8.6. ForwardPropagation

2.8.7. Descenso del gradiente

Es un método de optimización que busca el mínimo local en una función diferenciable.

2.8.8. BackPropagation

Regla de la cadena

2.9. Redes Neuronales Convolucionales

2.9.1. Tipos de capas

2.9.2. Estructura por capas

2.9.3. ForwardPropagation

Capítulo 3

Aportaciones

3.1. Redes Neuronales Totalmente Conectadas

3.1.1. BackPropagation con 1 capa oculta

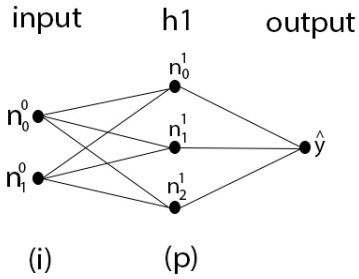

Figura 3.1: Red Neuronal totalmente conectada con 1 capa oculta

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

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 2.3 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

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

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

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

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

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

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

Pesos capas input-h1

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

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

Capa input

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

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

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].