Redes Neuronales y Aprendizaje Profundo para Visión Artificial

Practica II: Introducción a las redes neuronales

Enrique Nicanor Mariotti (mariottien@gmail.com)

Noviembre 2019

Introducción

La ejecución de los problemas y la generación de las imágenes del presente informe pueden obtenerse mediante:

```
python pr-cv-2.py -p {3, 4, 5, 6, 7, 8}
```

Para la resolución de los problemas se emplea un enfoque de *Programación Orientada a Objetos* ordenada en diferentes módulos. El mismo permite armar modelos secuenciales, no recurrentes, propagar las entradas de la red y retro-propagar los errores para resolver el problema de optimización. El método de optimización empleado es el gradiente descendente estocástico por *mini-batches*, sin inserción de momento.

Diferentes métricas, funciones de costo y de activación neuronal fueron implementadas según los requerimientos del problema.

Brevemente, el algoritmo de propagación de las entradas y retro-propagación de errores implementado puede resumirse de la siguiente manera:

1. Se inicializan los pesos ω_{ij} con valores aleatorios y pequeños, según algún método en particular.

2. Para un *mini-batch* de patrones de entrada $\overline{\xi}^{\mu}$, se propaga la red hacia adelante, según:
$$V_i = f(h_i) = f(\sum_j \omega_{ij} V_j) \tag{1}$$
Aquí, por conveniencia se ha tomado V_j como la neurona de *input* y V_i como la neurona de *output*. El valor del umbral se incluye dentro del vector de entradas por simplicidad en la notación.

3. Se calcula el gradiente de la función de costo de la capa de salida, multiplicado por el gradiente de la función de activación, es decir:
$$\delta_i = f'(h_i) L'(\varsigma_i, V_i) \tag{2}$$
Aquí, V_i es la salida de la red y ς_i es la salida real.

4. Se calculan los δ_i para la capas anteriores por retro-propagación.

$$\delta_i = f'(h_i) \sum_j \omega_{ij} \delta_j \tag{3}$$

   Aquí, el δ_j utilizado viene de la capa conexa mas próxima a la salida desde donde se retro-propaga el error.

5. Se aplica sobre los pesos una regla de aprendizaje de la forma:

$$\Delta\omega_{ij} = \mu \delta_i V_j \tag{4}$$

   Donde μ es el coeficiente de aprendizaje, en este caso, constante.

6. Se modifican los pesos de todas las conexiones, según:

$$\omega_{ij}^{n+1} = \omega_{ij}^{n} + \Delta\omega_{ij} \tag{5}$$

7. Se aplica un nuevo *mini-batch* de patrones de entrada $\overline{\xi}^{\mu}$ hasta completar el *batch*. El algoritmo se repite un numero fijo de veces.

1. Ejercicio I

Para la arquitectura de la Fig. 1, de un perceptron simple con dos entradas y un peso de *bias*, se propone calcular la salida del sistema propagando las operaciones hacia adelante y luego el calculo del gradiente de la salida respecto a los *inputs*. Los datos de entrada son $x^T = (2.8, -1.8)$, los pesos de las conexiones $w = (1.45, -0.35)$ y el valor del *bias*, $b = -4$.

El grafo computacional correspondiente a la operación $\mathbf{y} = f(\mathbf{wx} + \mathbf{b})$ se muestra en la Fig. 2.

No hay función de costo definida, ni valores objetivos a ajustar en un proceso de optimización.

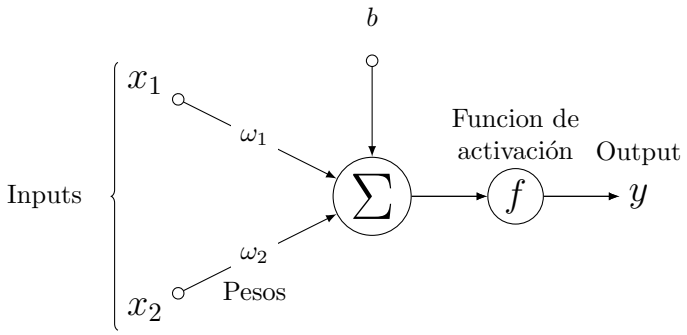

Figura 1: Modelo simple de perceptron de dos entradas.

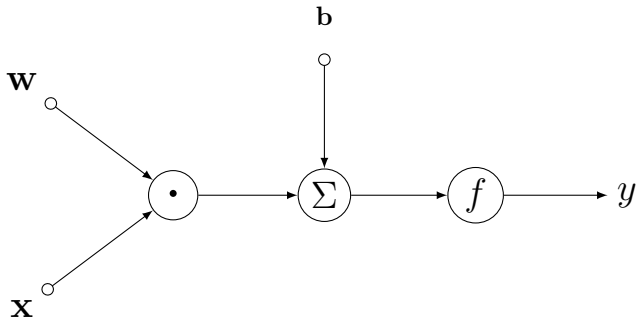

Figura 2: Grafo computacional del modelo de perceptron de la Fig. 1

- **Sigmoid:**

  La función de activación sigmoide se define como:

$$f = \frac{1}{1 + e^{-x}} \tag{6}$$

  Y su derivada respecto a la variable de entrada es:

$$f' = f(1 - f) \tag{7}$$

La propagación de la entrada a través de las operaciones de la red es:

$$f\left(\mathbf{w}\mathbf{x}+\mathbf{b}\right)=0.33 \tag{8}$$

Y el gradiente de la salida respecto a los valores de entrada:

$$\begin{aligned}\nabla_b f=\nabla_h f&=0.22\\ \nabla_x f=\mathbf{w}^T\nabla_h f&=\begin{pmatrix}0.32\\-0.07\end{pmatrix}\\ \nabla_w f=\nabla_h f\mathbf{x}^T&=\begin{pmatrix}0.61 & -0.39\end{pmatrix}\end{aligned} \tag{9}$$

- **Tangente hiperbólica:**

  La función de activación se define como:

$$f=\tanh(x) \tag{10}$$

  Y su derivada respecto a la variable de entrada es:

$$f'=(1-f^2) \tag{11}$$

  La propagación de la entrada a través de las operaciones de la red es:

$$f\left(\mathbf{w}\mathbf{x}+\mathbf{b}\right)=0.59 \tag{12}$$

  Y la retro-propagación del gradiente hacia atrás en la red es:

$$\begin{aligned}\nabla_b f=\nabla_h f&=0.65\\ \nabla_x f=\mathbf{w}^T\nabla_h f&=\begin{pmatrix}0.94\\-0.22\end{pmatrix}\\ \nabla_w f=\nabla_h f\mathbf{x}^T&=\begin{pmatrix}1.82 & -1.17\end{pmatrix}\end{aligned} \tag{13}$$

- **Exponential Linear Unit (ELU):**

  La función de activación se define como:

$$f=\begin{cases}x & x\geq 0\\ \alpha(e^x-1) & x<0\end{cases} \tag{14}$$

  Y su derivada respecto a la variable de entrada es:

$$f'=\begin{cases}1 & x\geq 0\\ f+\alpha & x<0\end{cases} \tag{15}$$

  Aquí, hemos tomado a modo de ejemplo $\alpha=0.1$. La propagación de la entrada a través de las operaciones de la red es:

$$f\left(\mathbf{w}\mathbf{x}+\mathbf{b}\right)=0.68 \tag{16}$$

Y la retro-propagación del gradiente hacia atrás en la red es:

$$\begin{aligned} \nabla_b f = \nabla_h f &= 1.00 \\ \nabla_x f = \mathbf{w}^T \nabla_h f &= \begin{pmatrix} 1.45 \\ -0.35 \end{pmatrix} \\ \nabla_w f = \nabla_h f \mathbf{x}^T &= \begin{pmatrix} 2.8 & -1.8 \end{pmatrix} \end{aligned} \tag{17}$$

- **Leaky Rectified Linear Unit (Leaky ReLU):**
  La función de activación se define como:

$$f = \begin{cases} x & x \geq 0 \\ \alpha x & x < 0 \end{cases} \tag{18}$$

Y su derivada respecto a la variable de entrada es:

$$f' = \begin{cases} 1 & x \geq 0 \\ \alpha & x < 0 \end{cases} \tag{19}$$

Aquí, hemos tomado a modo de ejemplo $\alpha = 0.1$. La propagación de la entrada a través de las operaciones de la red es:

$$f\left(\mathbf{w}\mathbf{x} + \mathbf{b}\right) = 0.68 \tag{20}$$

Y la retro-propagación del gradiente hacia atrás en la red es:

$$\begin{aligned} \nabla_b f = \nabla_h f &= 1.00 \\ \nabla_x f = \mathbf{w}^T \nabla_h f &= \begin{pmatrix} 1.45 \\ -0.35 \end{pmatrix} \\ \nabla_w f = \nabla_h f \mathbf{x}^T &= \begin{pmatrix} 2.8 & -1.8 \end{pmatrix} \end{aligned} \tag{21}$$

2. Ejercicio II

Para la arquitectura de la Fig. 3, de un perceptron con dos entradas y dos salidas, se propone calcular la salida del sistema propagando las operaciones hacia adelante, y luego el calculo del gradiente respecto a los *inputs*. Los datos de entrada son $x^T = (-3.1, 1.5)$, los pesos de las conexiones $w = \begin{pmatrix} -0.2 & 2 \\ -0.5 & -0.3 \end{pmatrix}$ y el valor del *bias*, $b^T = (-4.0, -1.0)$.

El grafo computacional correspondiente a la operación $\mathbf{y} = f(\mathbf{wx} + \mathbf{b})$ es idéntico al de la Fig. 2.

No hay función de costo definida, ni valores objetivos a ajustar en un proceso de optimización.

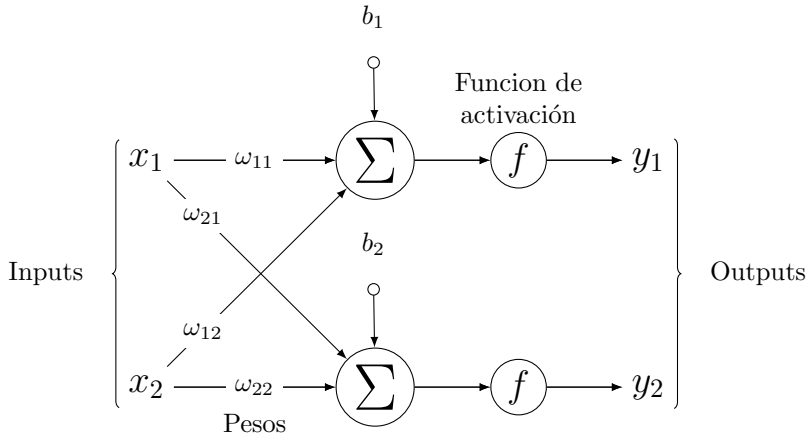

Figura 3: Modelo de un perceptron de dos entradas y dos salidas.

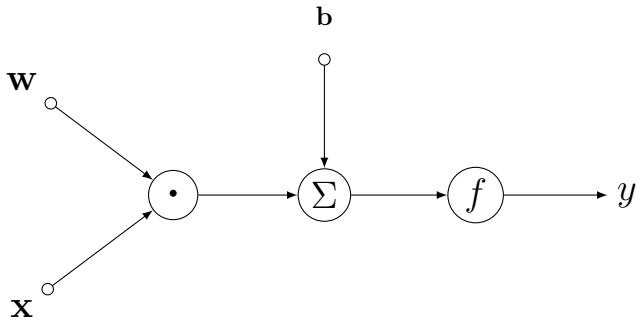

Figura 4: Grafo computacional del modelo de perceptron de la Fig. 3

- **Sigmoid:**

  La función de activación sigmoide se define como:

$$f = \frac{1}{1 + e^{-x}} \tag{22}$$

  Y su derivada respecto a la variable de entrada es:

$$f' = f(1 - f) \tag{23}$$

La propagación de la entrada a través de las operaciones de la red es:

$$f\left(\mathbf{w}\mathbf{x} + \mathbf{b}\right) = \begin{pmatrix} 0.59 \\ 0.47 \end{pmatrix} \tag{24}$$

Y la retro-propagación del gradiente hacia atrás en la red es:

$$\nabla_b f = \nabla_h f = \begin{pmatrix} 0.241 \\ 0.249 \end{pmatrix} \tag{25}$$

$$\nabla_x f = \mathbf{w}^T \nabla_h f = \begin{pmatrix} -0.172 \\ 0.407 \end{pmatrix} \tag{26}$$

$$\nabla_w f = \nabla_h f \mathbf{x}^T = \begin{pmatrix} -0.74 & 0.36 \\ -0.77 & 0.37 \end{pmatrix} \tag{27}$$

3. Ejercicio III

En la presente sección se intenta resolver el problema de clasificación de imágenes de **CIFAR10**, ejemplificadas en la Fig. 5.

Para ello se empleo el conjunto completo de entrenamiento, consistiendo de 50000 ejemplos. El conjunto de validación consiste en los 10000 ejemplos del conjunto de prueba.

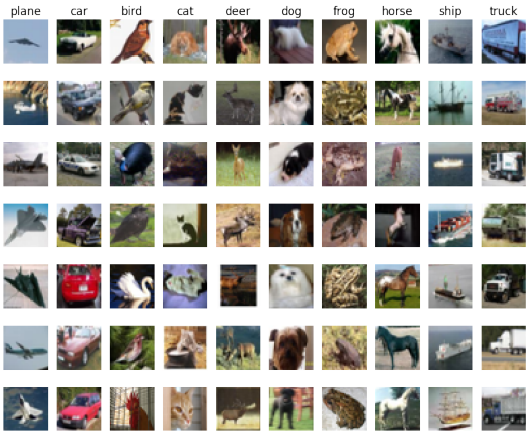

Figura 5: Segmento de los datos de entrenamiento del conjunto **CIFAR10**.

Los valores de los conjuntos de entrenamiento y prueba se encuentran normalizados en el intervalo $[0, 1]$. Además, con el objetivo de mantener los pesos de la red ω_{ij} dentro de un mismo orden de magnitud, se resta a ambos *batchs* la media de los ejemplos de entrenamiento. La Fig. 6 ilustra el valor medio de este subconjunto de los datos.

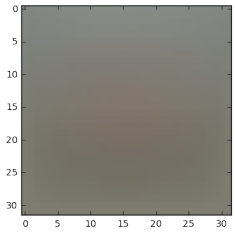

Figura 6: Media de los datos de entrenamiento del conjunto **CIFAR10**.

Para la resolución del problema se empleo una arquitectura con $N_{input} = 3072$ entradas, seguida de una capa oculta con $N_{hidden} = 100$. La salida de la red consiste en una capa de $N_{out} = 10$ neuronas, donde están codificadas las 10 clases de **CIFAR10**. El grafo computacional del modelo esta ilustrado en la Fig. 7.

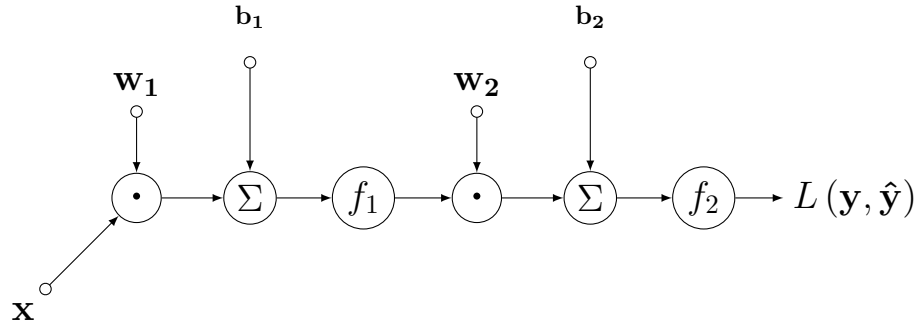

Figura 7: Grafo computacional del modelo de red neuronal multi-capa de la Sección 3.

Es decir, los gradientes de la funcion de costo respecto a los pesos de la red se pueden calcular como:

$$\begin{aligned} \nabla_{w_2} L &= \nabla_{f_2} L \nabla_{h_2} f_2 {\mathbf{h_1}}^T \\ \nabla_{w_1} L &= \left({\mathbf{w_2}}^T \cdot \nabla_{w_2} L\right) \nabla_h f_1 \mathbf{x}^T \end{aligned} \tag{28}$$

Las funciones de activación $g(h_i)$ de las neuronas de la capa oculta se eligen como $g(h_i) = \tanh(h_i)$, donde h_i es la entrada a la neurona i. La función de activación $g(h_i)$ de las neuronas de la capa de salida es lineal, es decir $g(h_i) = h_i$.

El proceso se repite usando un *mini-batch* de 32 ejemplos, un total de 50 *epochs*.

El *learning rate* se mantiene fijo en $lr = 0.001$ La regularización $L2$ de la función de costo se realiza con un coeficiente de $\lambda = 0.01$. Los pesos ω_{ij} se inicializan mediante el método de normal de *Glorot*. La función de costo a minimizar es el error medio cuadrático (*MSE*), es decir:

$$loss_{MSE} = \frac{1}{n} \sum_{i=1}^{n} \|y_i - y_{score}\|^2 \tag{29}$$

Todas las neuronas poseen entradas adicionales de *bias*.

Las curvas de *accuracy* y de la función de costo se ilustran en la Fig. 8. El comportamiento de las curvas es el esperado, aunque la complejidad del problema hace que la precision alcanzada sea modesta, siendo de alrededor de $\eta_{val} \approx 0.36$ para el conjunto de validación.

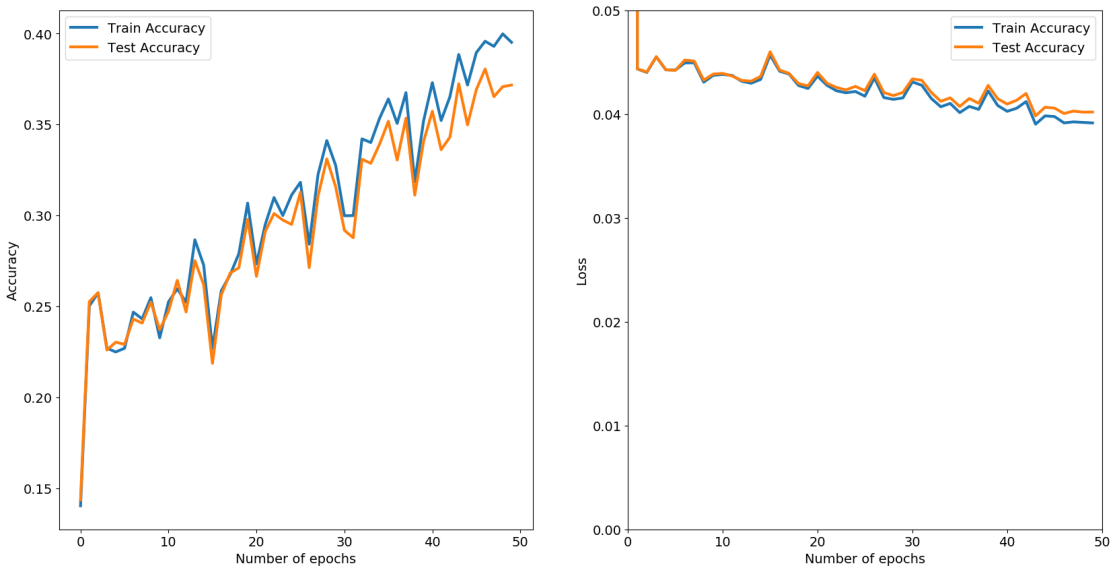

Figura 8: Evolución de la *accuracy* y de la función de costo *MSE* en función de las *epochs* para el problema de clasificación **CIFAR10**.

4. Ejercicio IV

La resolución de este problema es esencialmente la misma que la ilustrada en la Sección 3, con la única diferencia que se intenta resolver el problema de clasificación de imágenes de **CIFAR10** empleando la función de costo *Categorical Cross-Entropy*. A saber:

$$loss_{CCE} = \frac{1}{n}\sum_{i=1}^{n} -log\left(\frac{e^{y_i}}{\sum_j^c e^{y_j}}\right) \tag{30}$$

Donde y_i representa la clase ganadora. Al margen de este cambio, se respetan el pre-procesamiento de los datos,la arquitectura de la red, las funciones de activación neuronales, y las condiciones de entrenamiento y de inicialización de los pesos.

Las curvas de *accuracy* y de la función de costo se ilustran en la Fig. 9. El comportamiento de las curvas es el esperado, siendo el resultado de precision de mejor calidad que el obtenido en la Sección 3. Aquí, $\eta_{val} \approx 0.40$.

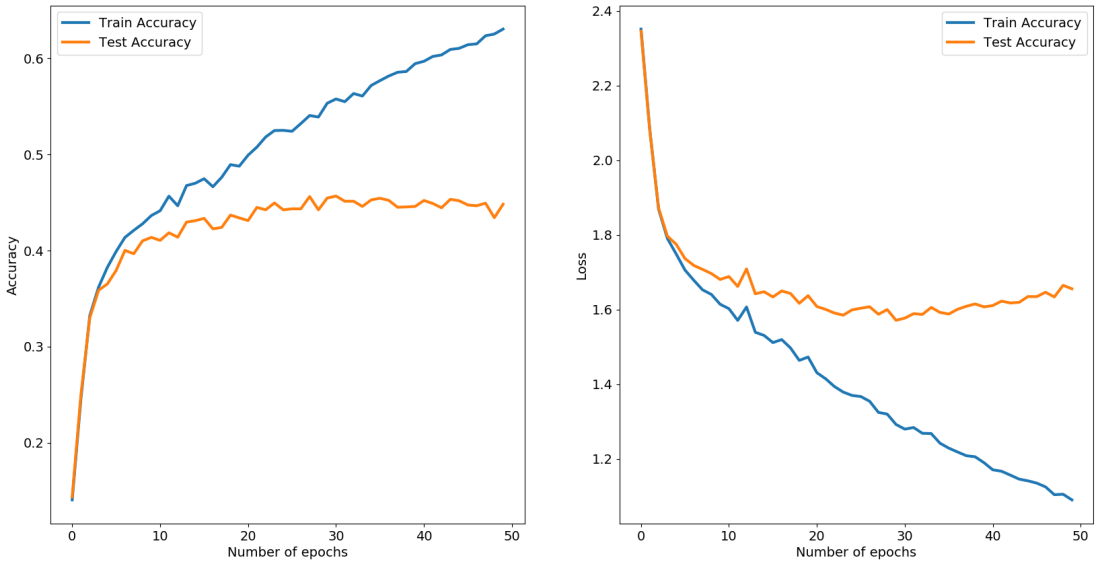

Figura 9: Evolución de la *accuracy* y de la función de costo *Categorical Cross-Entropy* en función de las *epochs* para el problema de clasificación **CIFAR10**.

5. Ejercicio V

Nuevamente, la resolución de este problema es esencialmente la misma que la ilustrada en la Sección 4, con la única diferencia que las funciones de activación $g(h_i)$ de las neuronas de la capa oculta se elige como $g(h_i) = \text{ReLU}(h_i)$, donde h_i es la entrada a la neurona i. La la función de activación $g(h_i)$ de las neuronas de la capa de salida se elige como $g(h_i) = \text{Sigmoid}(h_i)$

Al margen de este cambio, se respetan el pre-procesamiento de los datos,la arquitectura de la red, las funciones de activación neuronales, y las condiciones de entrenamiento y de inicialización de los pesos.

La función de costo a minimizar es la *Categorical Cross-Entropy* ilustrada en la Ec. 30.

Las curvas de *accuracy* y de la función de costo se ilustran en la Fig. 9. El comportamiento de las curvas es en lineas generales el esperado, y se llega a valores de precision comparables con los obtenidos en la Sección 4. Aquí, $\eta_{val} \approx 0.43$. Se observa además menor discrepancia entre la *accuracy* de ambos subconjuntos.

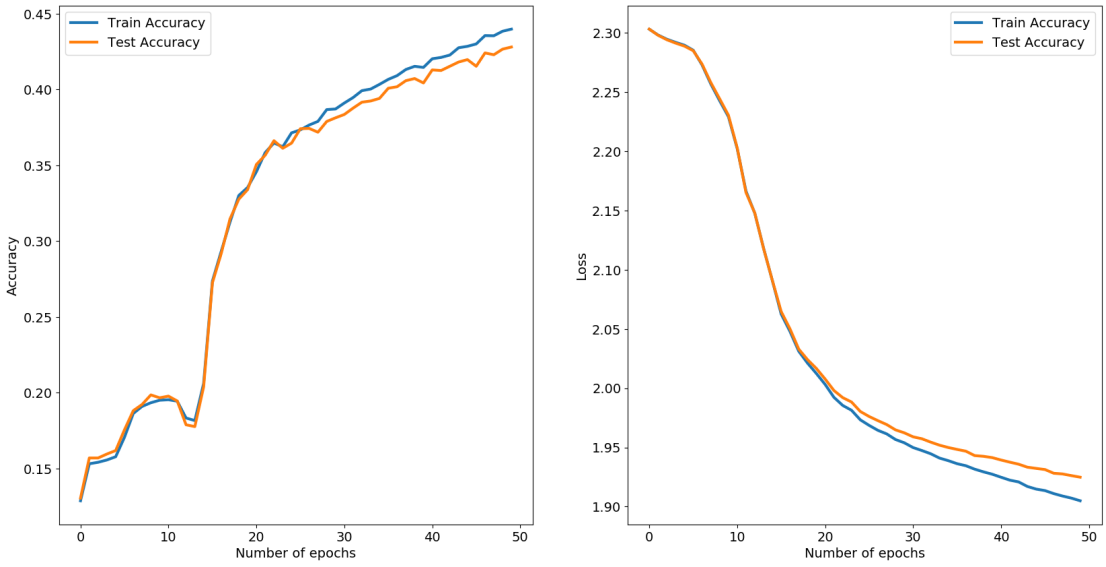

Figura 10: Evolución de la *accuracy* y de la función de costo *Categorical Cross-Entropy* en función de las *epochs* para el problema de clasificación **CIFAR10**.

6. Ejercicio VI

Con el objetivo de implementar el algoritmo de retropropagación de errores para la resolución del problema de la regla lógica **XNOR**, se estudiaron dos arquitecturas diferentes de redes multicapa, ilustradas en la Fig. 11

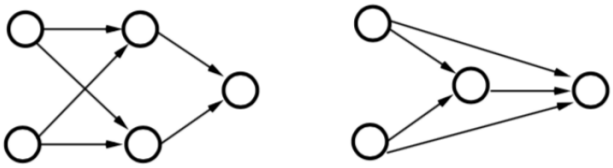

Figura 11: Arquitecturas implementadas para la resolución de la regla **XNOR**.

En la regla **XNOR** con dos entradas, tal que $\xi \in \{1, -1\}$ y una salida, también $\varsigma \in \{1, -1\}$, esta definida en el Cuadro 1:

ξ_1^μ	ξ_2^μ	ς^μ
1	1	1
-1	1	-1
1	-1	-1
-1	-1	1

Cuadro 1: Regla lógica **XNOR**.

Las funciones de activación $g(h_i)$ de todas las neuronas, se elige como $g(h_i) = \tanh(h_i)$, donde h_i es la entrada a la neurona i.

Se considera además para todas las neuronas entradas adicionales de *bias*.

El proceso se repite para el *batch* completo de los 4 ejemplos, un total de 200 *epochs*, con el objetivo de que los pesos de conexión ω_{ij} converjan.

El *learning rate* se mantiene fijo en $lr = 0.5$ La regularización $L2$ de la función de costo se realiza con un coeficiente de $\lambda = 0.01$. Los pesos ω_{ij} se inicializan mediante el método de normal de *Glorot*. La función de costo a minimizar es error medio cuadrático (*MSE*) de la Ec. 29.

Las salidas de las 2 arquitecturas para el conjunto de validación (igual al conjunto de entrenamiento, por la simplicidad del problema), se presentan a continuación:

- **Arquitectura 1:**

ξ_1^μ	ξ_2^μ	ς^μ	V^μ
1	1	1	0.9112
-1	1	-1	-0.8878
1	-1	-1	-0.8883
-1	-1	1	0.9116

Cuadro 2: Resultados de la Arquitectura 1.

- **Arquitectura 2:**

ξ_1^μ	ξ_2^μ	ς^μ	V^μ
1	1	1	0.8497
-1	1	-1	-0.8802
1	-1	-1	-0.8802
-1	-1	1	0.8802

Cuadro 3: Resultados de la Arquitectura 2.

Se concluye que ambas arquitecturas logran aprender el problema correctamente de manera asintótica, ya que presentan salidas aproximadamente iguales a las salidas esperadas.
El tiempo de resolución del problema esta en el mismo orden de magnitud para la **Arquitectura 1** y la **Arquitectura 2**.

Las curvas de *accuracy* y de la funcion de costo se ilustran en la Fig. 12 y en la Fig. 13.

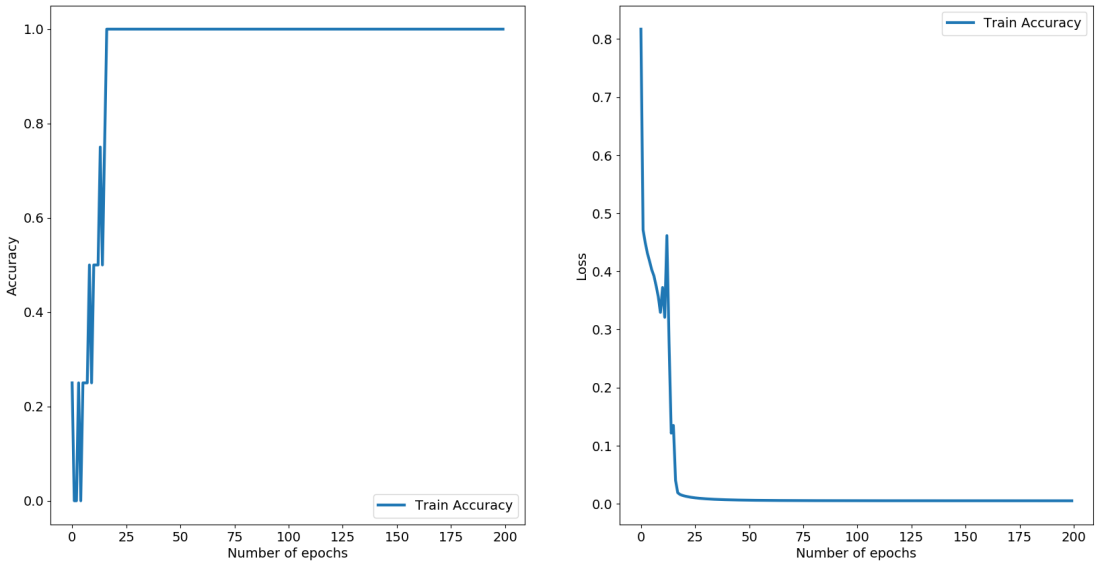

Figura 12: **Arquitectura 1:** Evolución de la *accuracy* y de la función de costo *MSE* en función de las *epochs* para el problema de la regla **XNOR**.

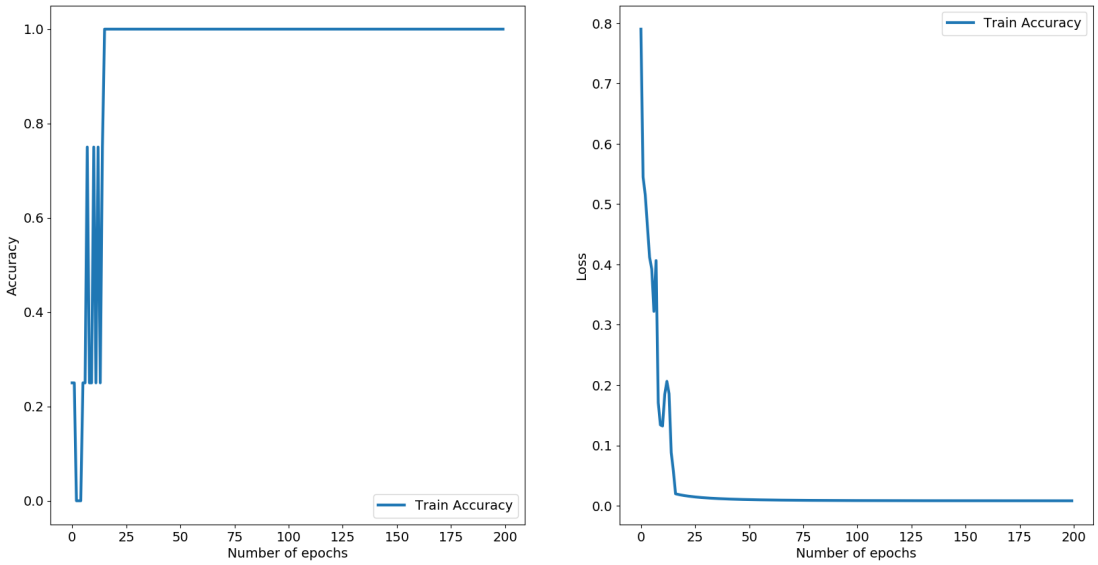

Figura 13: **Arquitectura 2:** Evolución de la *accuracy* y de la función de costo *MSE* en función de las *epochs* para el problema de la regla **XNOR**.

7. Ejercicio VII

A continuación se busca implementar una solución al problema de paridad mediante el entrenamiento de una red neuronal multicapa. El problema de paridad es una generalización de la regla **XNOR** ilustrado en la Sección 6 para **N** entradas.

Siguiendo el ejemplo anterior con entradas binarias, si $\xi_i \in \{1, -1\}$, entonces, $\varsigma = \prod \xi_i$.
La lógica del problema queda ejemplificada en el Cuadro 4.

ξ_1^μ	ξ_2^μ	ξ_3^μ	ξ_4^μ	ς^μ
1	1	1	1	1
1	-1	1	1	-1
1	-1	-1	1	1
1	-1	-1	-1	-1
...	...	...	...	...

Cuadro 4: Regla lógica de paridad para $N_{in} = 4$.

Se empleara en primera instancia una red con $N_{input} = 5$ entradas, seguida de una capa oculta con $N_{hidden} = 10$. Por otro lado, el conjunto de de entrenamiento y de validación (iguales) consta de $2^N = 32$ ejemplos generados con todas las posibles combinaciones. La arquitectura implementada esta ilustrada en las Fig. 14.

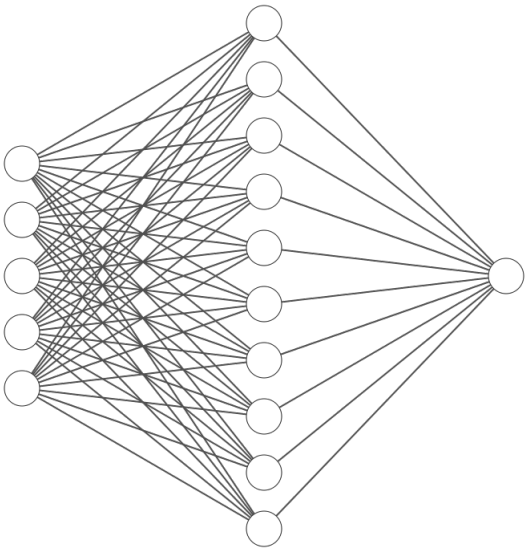

Figura 14: Arquitectura implementada para la resolución del problema de paridad.

El proceso de entrenamiento se repite usando el *batch* completo de los 32 ejemplos, un total de 1000 *epochs*, con el objetivo de que los pesos de conexión ω_{ij} converjan.

El *learning rate* se mantiene fijo en $lr = 0.35$ La regularización $L2$ de la función de costo se realiza con un coeficiente de $\lambda = 0.01$. Los pesos ω_{ij} se inicializan mediante el método de normal de *Glorot*. La función de costo a minimizar es error medio cuadrático (*MSE*) de la Ec. 29.

Todas las neuronas poseen entradas adicionales de *bias*. Las funciones de activación $g(h_i)$ de las neuronas se elige como $g(h_i) = \tanh(h_i)$, donde h_i es la entrada a la neurona i.

Las curvas de *accuracy* y de la funcion de costo se ilustran en la Fig. 15. Se concluye que la arquitectura logra aprender el problema correctamente de manera asintótica, ya que presentan salidas aproximadamente iguales a las salidas esperadas.

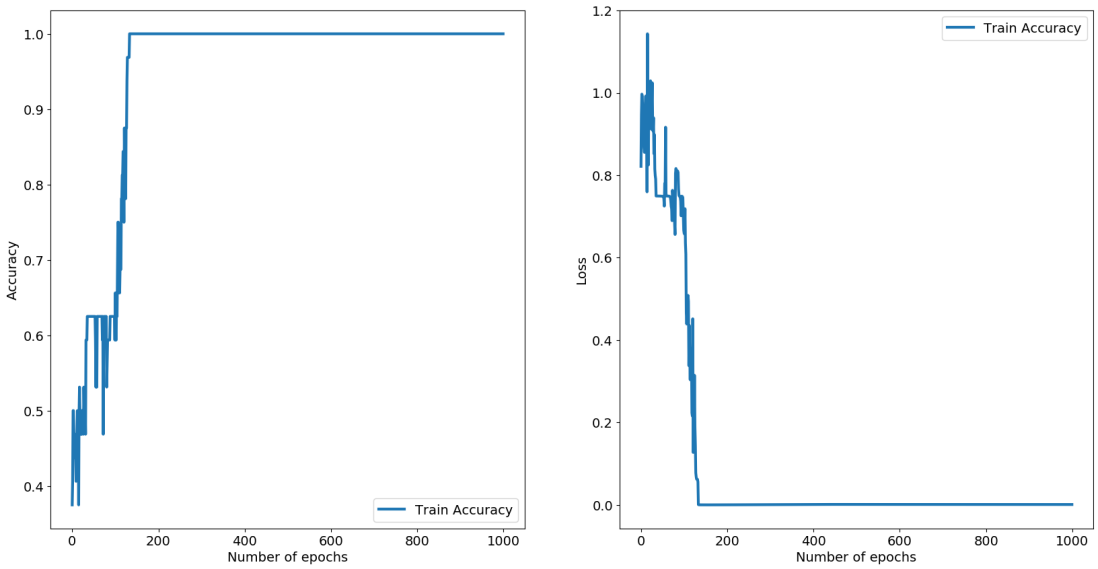

Figura 15: Evolución de la *accuracy* y de la función de costo *MSE* en función de las *epochs* para el problema de paridad.

8. Ejercicio VIII

La resolución de este problema es esencialmente la misma que la ilustrada en la Sección 4, con la única diferencia que se ha agregado una segunda capa de $N_{hidden} = 100$ neuronas antes de la capa de la salida.

Al margen de este cambio, se respetan el pre-procesamiento de los datos,la arquitectura de la red, las funciones de activación neuronales, y las condiciones de entrenamiento y de inicialización de los pesos.

La función de costo a minimizar es la *Categorical Cross-Entropy* ilustrada en la Ec. 30.

Las curvas de *accuracy* y de la función de costo se ilustran en la Fig. 9. El comportamiento de las curvas es el esperado, llegándose a ligeramente mayor precision que la obtenida previamente en la Sección 4. Aquí, $\eta_{val} \approx 0.45$.

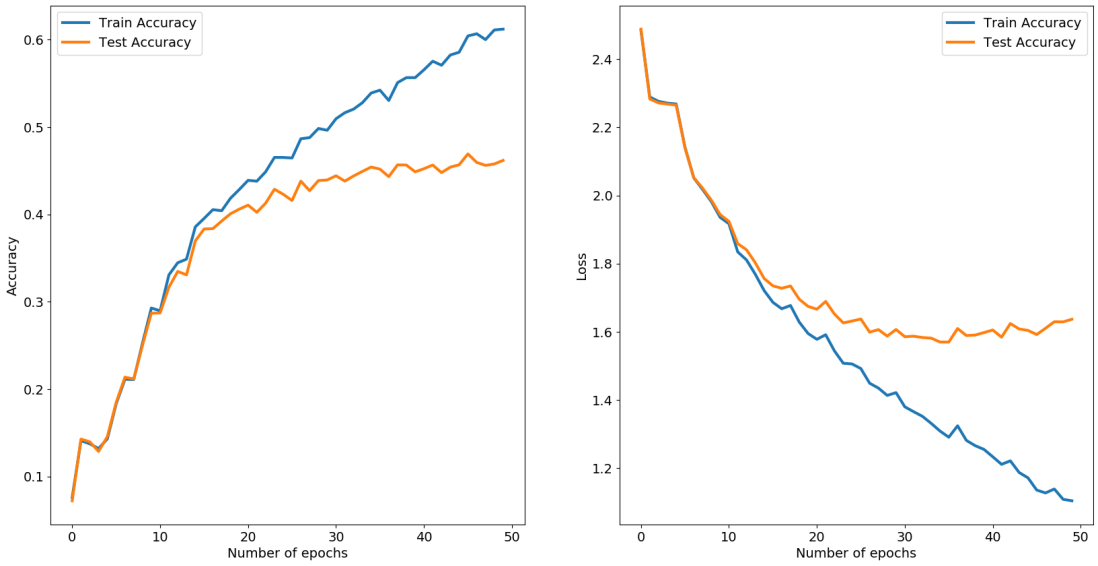

Figura 16: Evolución de la *accuracy* y de la función de costo *Categorical Cross-Entropy* en función de las *epochs* para el problema de clasificación **CIFAR10**.