Practica 2:
Redes Neuronales

Ana Paula Morresi
Aprendizaje profundo con aplicación a visión artificial

31 de octubre de 2025

1. Problemas

Ejercicio 1.

En base a la teoría de Grafos Computacionales, se calcularon las funciones de activación a continuación (cálculos en color celeste) y todos sus gradientes (cálculos en color rosa) para una red feedforward con dos entradas + umbral. Los cálculos se hicieron usando tres decimales.

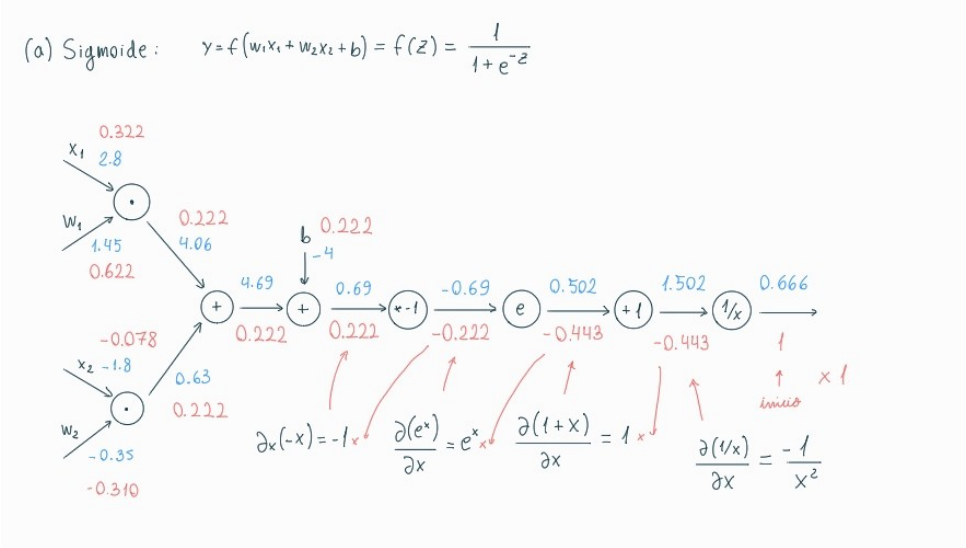

Figura 1: Grafo con función de activación sigmoide.

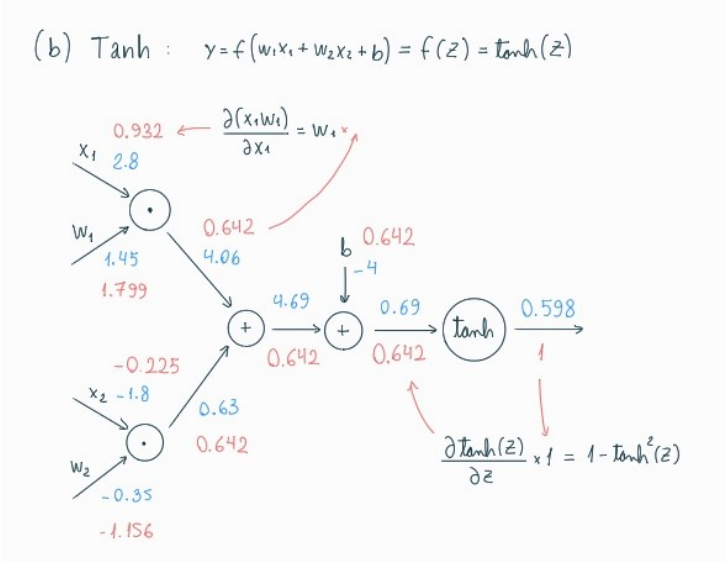

Figura 2: Grafo con función de activación tanh.

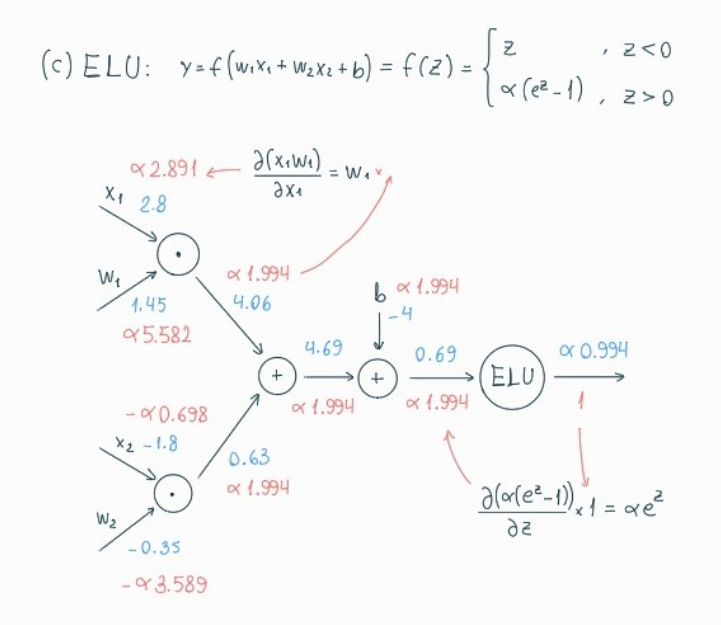

Figura 3: Grafo con función de activación ELU.

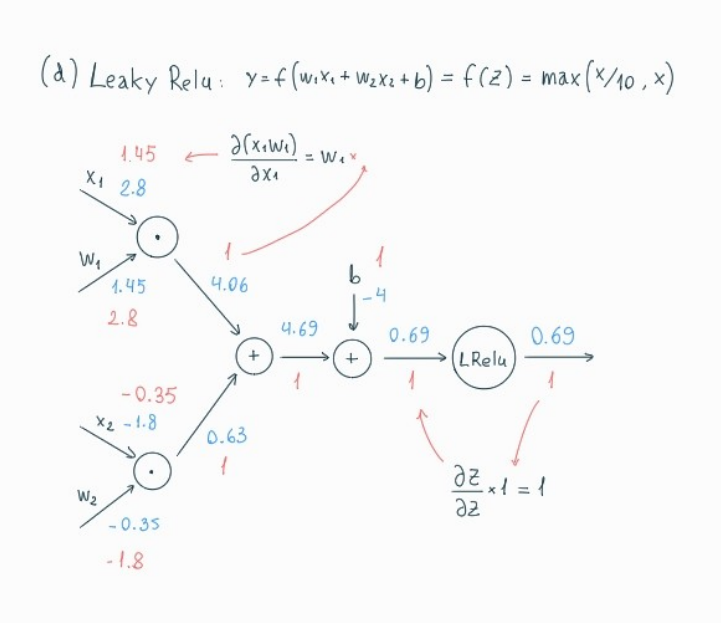

Figura 4: Grafo con función de activación Leaky Relu.

Ejercicio 2.

Para una red con dos neuronas con función de activación sigmoide y función costo MSE, se calculo el forward (en color celeste) y todos los gradientes involucrados (en color rosa). Para la función MSE no se contaba con los valores (y_1,y_2) de predicción por lo que se los dejo expresados al calcular los gradientes.

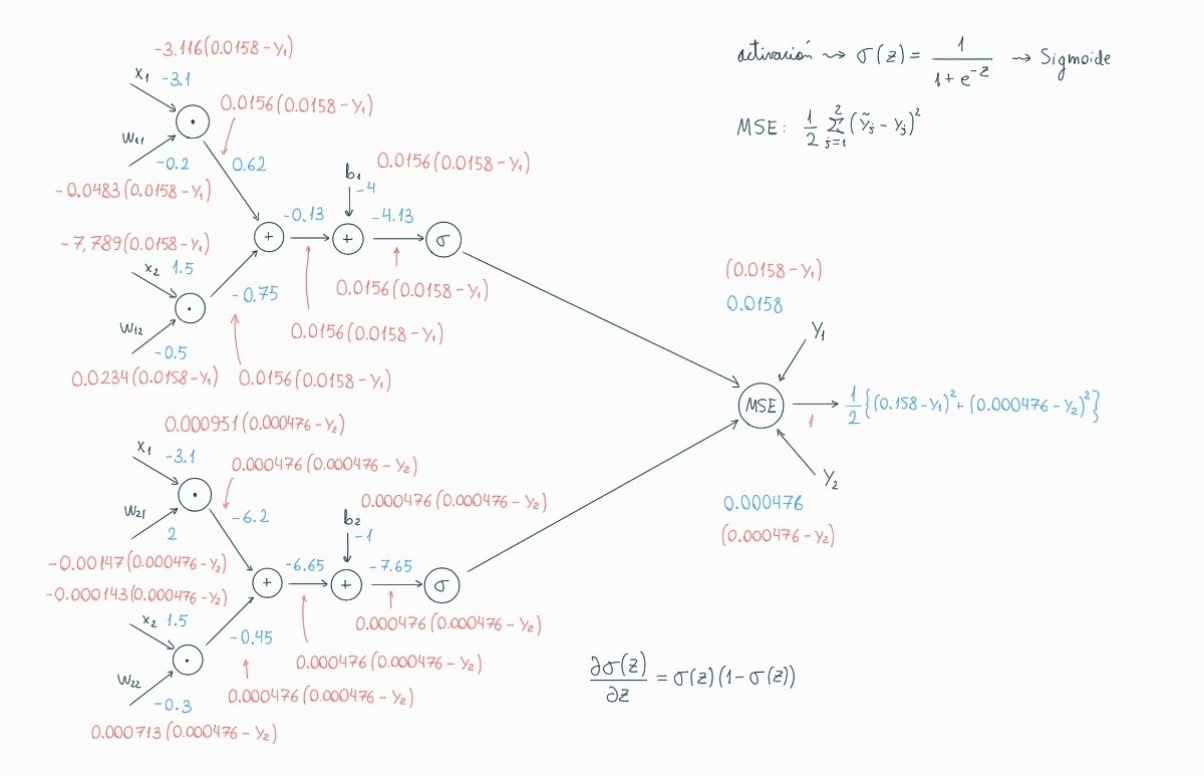

Figura 5: Grafo computacional con forward y backward calculados.

Ejercicio 3.

Se implemento una red neuronal sin POO de dos capas totalmente conectadas para resolver el problema CIFAR-10. La primer capa cuenta con 100 neuronas con función de activación sigmoidal. La segunda capa es de salida y cuenta con 10 neuronas con activación lineal. La función de costo es el MSE con termino de regularización L2. El set de datos de train cuenta con 40000 muestras, mientras que los de validación y test con 10000 muestras cada uno.

Con esto se llevo a cabo el entrenamiento con un *learning rate=5e-2*, *batch size* = 500, *lambda regularización=1e-3* y *epochs* = 500. En la Fig. 6 se pueden ver las curvas de loss y accuracy en función de las épocas. Los valores de accuracy finales obtenidos fueron:

Accuracy en train: 0.4978 || Accuracy en validation: 0.4567 || Accuracy en test: 0.4700

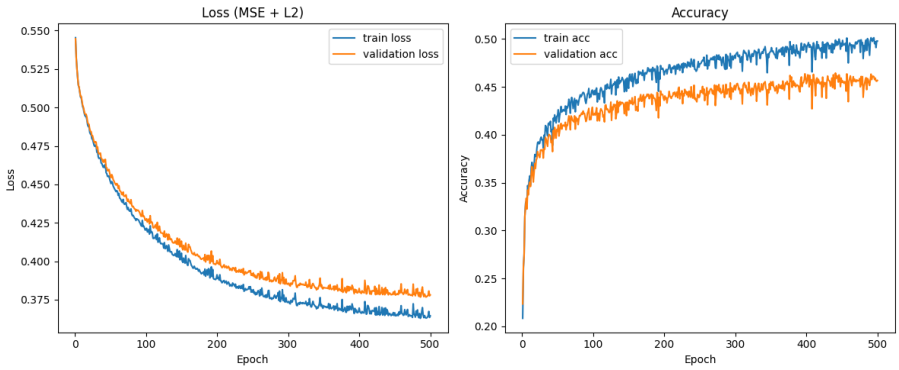

Figura 6: Curvas de loss y accuracy para los datos de train y validación de CIFAR-10.

En la Fig. 7 se puede ver la matriz de confusión. En esta, se puede observar que algunas clases se lograron aprender mejor que otras. Teniendo en cuenta que todas las clases tienen un total de 1000 ejemplos en el set de test, se puede ver que solo 6 clases lograron clasificarse correctamente por encima del 50 % de los ejemplos.

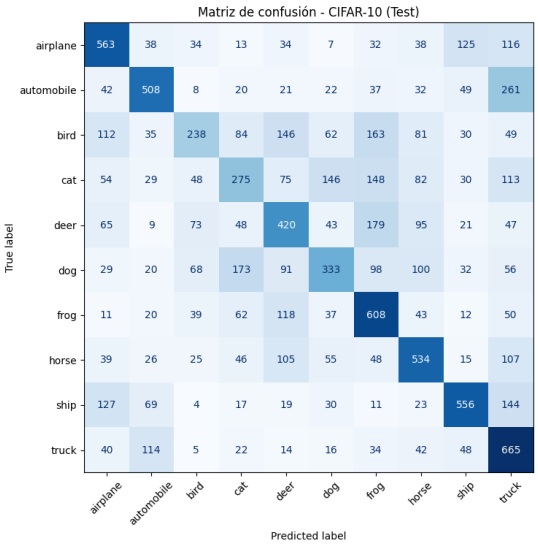

Figura 7: Matriz de confusión de los datos de test de CIFAR-10.

Ejercicio 4.

Se modifico el código anterior para que la función costo sea *Categorical Cross Entropy* (CCE). Para este caso se llevo a cabo el entrenamiento con un *learning rate=5e-2*, *batch size* = 500, *lambda regularización=1e-3* y *epochs* = 600. En la Fig. 8 se pueden ver las curvas de loss y accuracy. En la Fig. 9 se puede ver la matriz de confusión. Los valores de accuracy obtenidos fueron:

Accuracy en train: 0.5683 || Accuracy en validation: 0.5048 || Accuracy en test: 0.5070

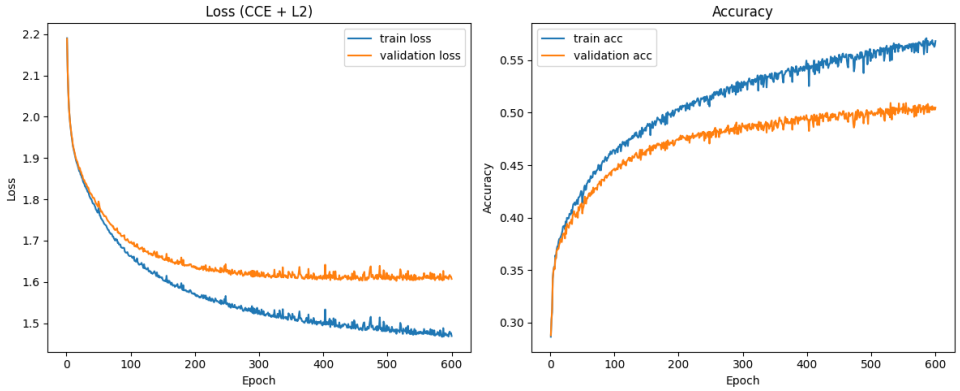

Figura 8: Curvas de loss y accuracy para los datos de train y validación de CIFAR-10 con función loss *Categorical Cross Entropy*.

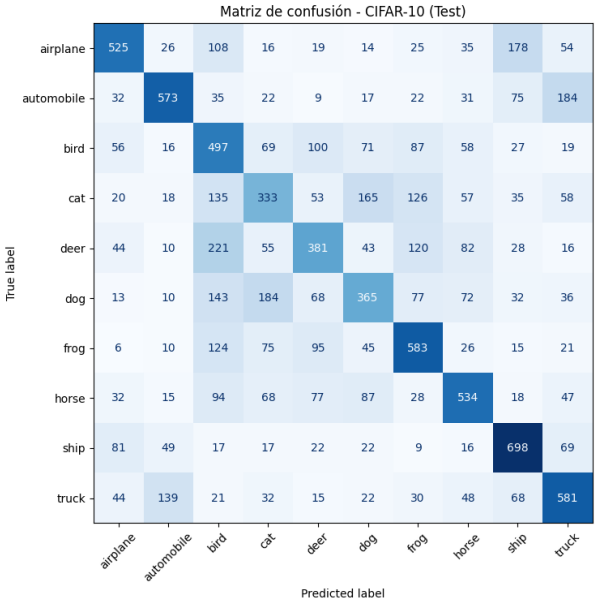

Figura 9: Matriz de confusión de los datos de test de CIFAR-10 con función loss *Categorical Cross Entropy*.

En este caso se puede ver que se logro obtener un mejor valor de accuracy, sin embargo esta mejora es leve.

Ejercicio 5.

Siguiendo con el problema anterior se modifico la función de activación de la capa oculta para que sea una ReLU y la de la capa de salida una sigmoidal.

Usando el MSE como costo se llevo a cabo el entrenamiento con un *learning rate=1e-3*, *batch size* = 200, *lambda regularización=1e-1* y *epochs* = 45. El numero de épocas se eligió tal que no estemos teniendo overfitting u otro problema. Se obtuvo lo que se observa en la Fig. 10 y Fig. 11. Los valores de accuracy obtenidos fueron:

Accuracy en train: 0.23025 || Accuracy en validation: 0.2263 || Accuracy en test: 0.2263

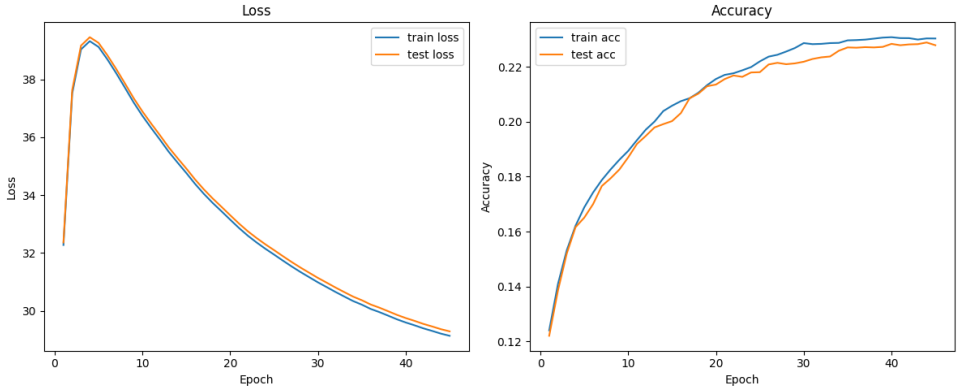

Figura 10: Curvas de loss y accuracy para los datos de train y validación de CIFAR-10 con función loss MSE.

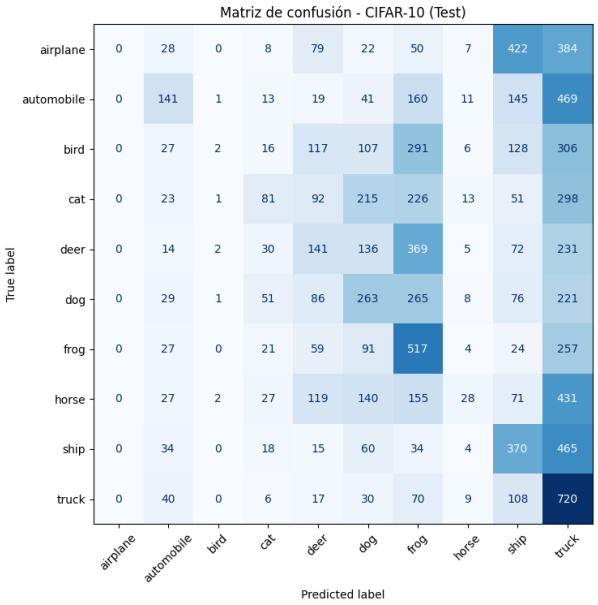

Figura 11: Matriz de confusión de los datos de test de CIFAR-10 con función loss MSE.

Se puede ver que la red no logra aprender las clasificaciones correctas.

De la misma forma se utilizo como costo *Categorical Cross Entropy*, para esto se remplazo la función de activación de la salida por softmax, ya que la sigmoidal a pesar de tener valores entre 0 y 1, no esta normalizada a todas las clases. En todo caso se

podría utilizar la función sigmoid con *Binary Cross-Entropy*, pero esto no es lo que se pide. Para el entrenamiento se usaron *learning rate=1e-2*, *batch size* = 200, *lambda regularización=1e-3* y *epochs* = 100. Se obtuvo lo que se observa en la Fig. 12 y Fig. 13. Los valores de accuracy obtenidos fueron:

Accuracy en train: 0.5509 || Accuracy en validation: 0.5029 || Accuracy en test: 0.4942

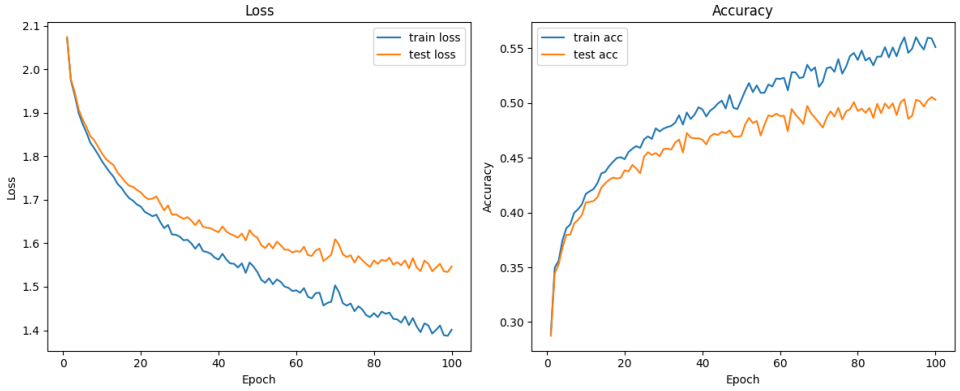

Figura 12: Curvas de loss y accuracy para los datos de train y validación de CIFAR-10 con función loss *Categorical Cross Entropy*.

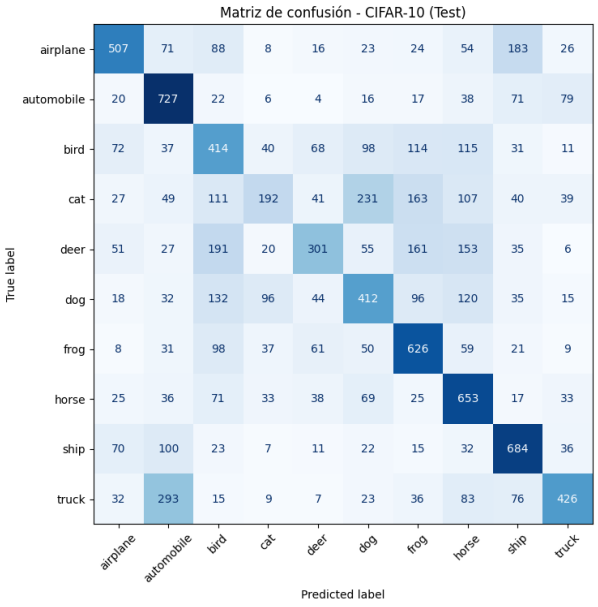

Figura 13: Matriz de confusión de los datos de test de CIFAR-10 con función loss *Categorical Cross Entropy*.

Para este caso la red logra un mejor desempeño en la clasificación que usando MSE como función costo.

Si usáramos la función ReLU + activación lineal en la ultima capa con función costo MSE, lo que ocurre es lo que se observa en las Fig. 14 y Fig. 27, esto usando *learning rate=1e-2*, *batch size* = 200, *lambda regularización=1e-3* y *epochs* = 200. Los valores de accuracy obtenidos fueron:

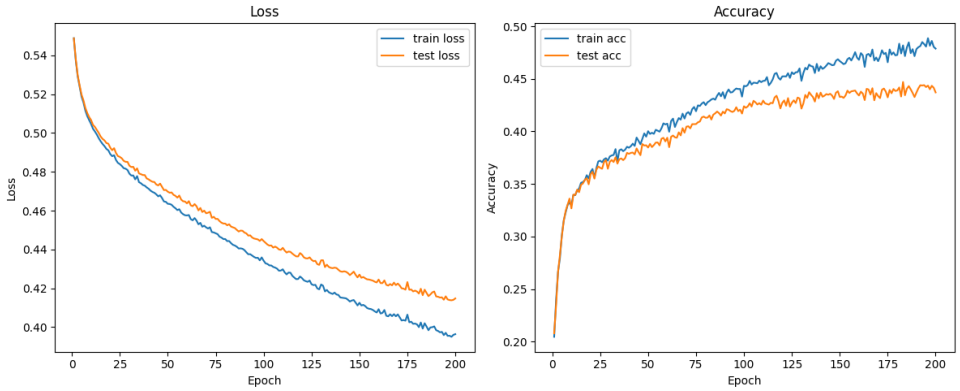

Figura 14: Curvas de loss y accuracy para los datos de train y validación de CIFAR-10 con función loss MSE.

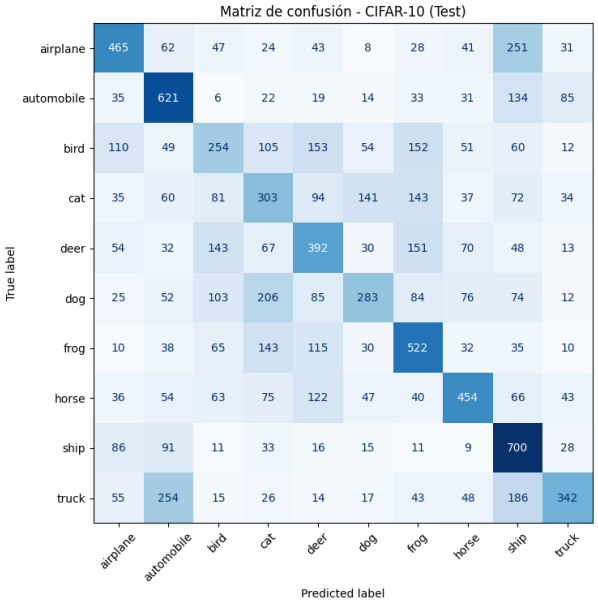

Figura 15: Matriz de confusión de los datos de test de CIFAR-10 con función loss MSE.

Accuracy en train: 0.4788 || Accuracy en validation: 0.4371 || Accuracy en test: 0.4336

Para este caso se ve que la clasificación mejora respecto a cuando se usaba la función sigmoidal en la salida, aunque sigue siendo mejor la clasificación con *Categorical Cross Entropy*.

El problema esta si queremos usar como función costo *Categorical Cross Entropy* con la salida con activación lineal. La función de costo se define como:

$$L = -\sum_{k=1}^{K} y_k \log(p_k) \tag{1}$$

donde K es el nro. de clases, y_k el valor verdadero (0 o 1) de la clase k y p_k la probabilidad predicha por el modelo para la clase k. Esta probabilidad se suele obtener de la función softmax:

$$p_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} \tag{2}$$

Luego para un conjunto de N muestras, el valor promedio es:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} y_{i,k} \log(p_{i,k}) \tag{3}$$

El problema con esto es que si $p_k = y_{predicho,k}$, debido a que aplicamos una activación lineal en la salida, ya no tenemos una probabilidad con valores entre 0 y 1, y normalizada. Ahora podemos tener valores positivos o negativos sin normalizar, lo cual va a llevar a que la perdida sea mal calculada y por ende no sea una buena métrica para el entrenamiento y los gradientes.

Ejercicio 6.

La regla XOR tiene dos entradas (1 o -1) y la salida es -1 si ambas son diferentes y 1 si ambas son iguales. Para resolver este problema se usaron las arquitecturas que se encuentran en la Fig. 16.

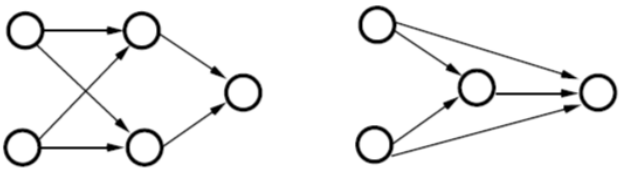

Figura 16: Arquitecturas utilizadas para el problema XOR.

Como función de activación de las neuronas se utilizo la función *tanh*, ya que esta toma valores entre -1 y 1, y como función de costo el MSE. El optimizador usado fue SGD.

Evaluando la arquitectura 1 se obtuvo lo que se observa en la Fig. 17, donde se pude ver la evolución de la precisión y de la función de costo a lo largo de las épocas.

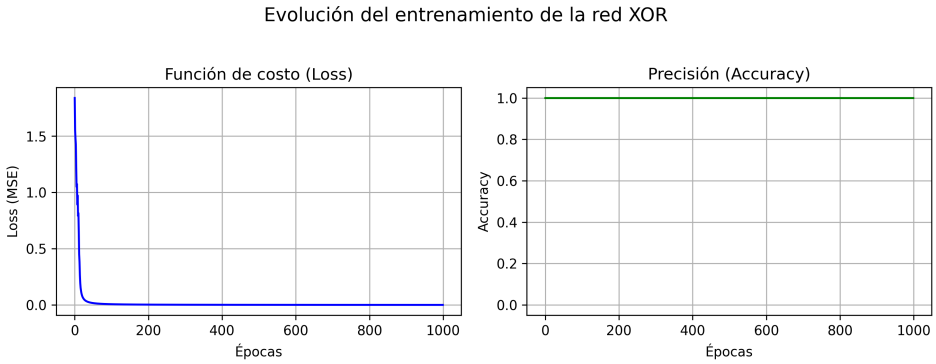

Figura 17: Precisión y la función de costo a lo largo de las épocas para la arquitectura 1.

De la misma forma, evaluando la arquitectura 2, en la Fig. 18 se puede observar lo obtenido.

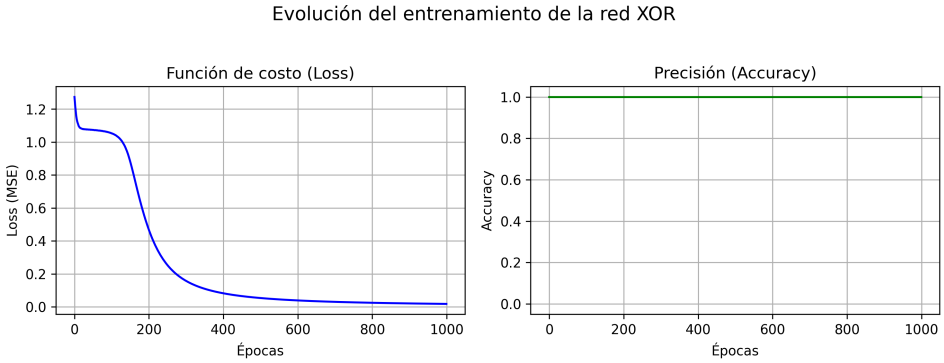

Figura 18: Precisión y la función de costo a lo largo de las épocas para la arquitectura 2.

Una cosa que se observó es que la semilla de inicialización de los pesos iniciales tiene influencia en los resultado obtenidos. Para esto se realizo un histograma con las predicciones obtenidas con cada modelo para 500 entrenamientos iguales. Esto se puede ver en la Fig. 19 para la arquitectura 1 y Fig.20 para la arquitectura 2.

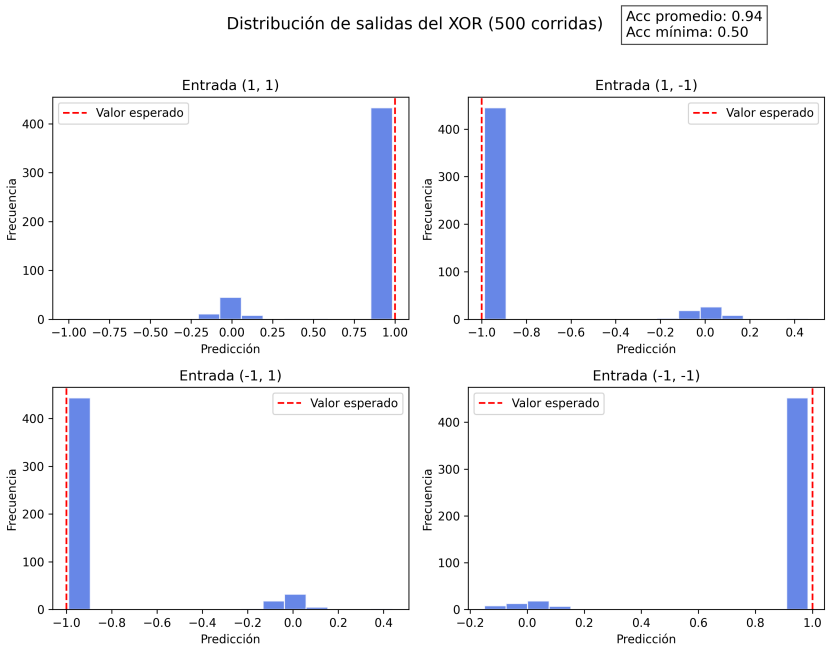

Figura 19: Histogramas de resultados para la arquitectura 1.

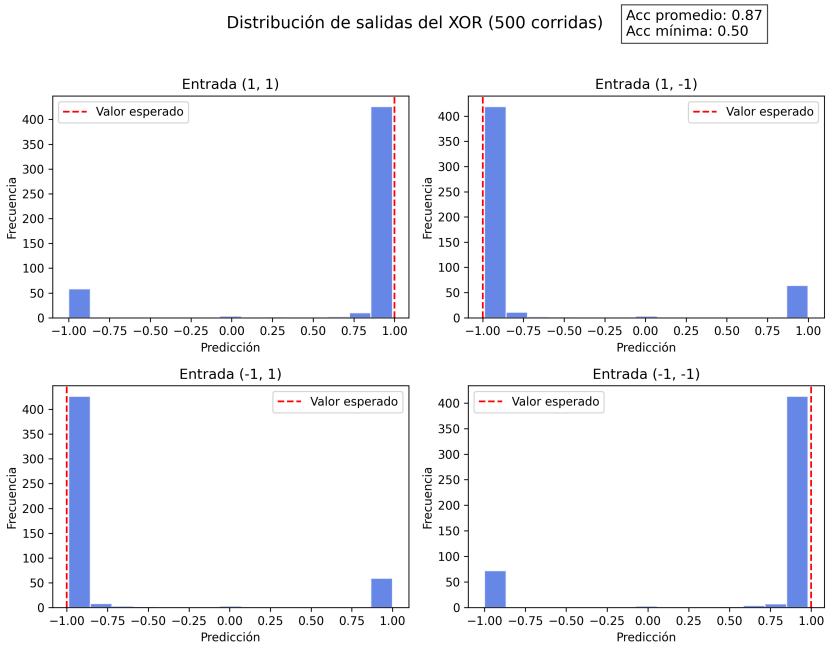

Figura 20: Histogramas de resultados para la arquitectura 2.

Se puede ver que para la arquitectura 1, el accuracy promedio es de 0.94, mientras que con la arquitectura 2 es de 0.87. Por lo que, llevando a cabo múltiples pruebas de

entrenamiento, la primera arquitectura presenta un mejor desempeño. En ambos casos también se ve que el mínimo accuracy es de 0.5, lo cual significa que la red solo aprendió dos de los 4 ejemplos de XOR. Finalmente, también se puede ver que la arquitectura 1, al clasificar mal, da valores de predicción alrededor de 0; mientras que la arquitectura 2, al clasificar mal, lo hace prediciendo alrededor de la clase opuesta a la correcta ($y_{pred} = \pm 1$).

Ejercicio 7.

El problema de paridad es una generalización del XOR para N entradas. La salida es 1 si el producto de las N entradas es 1 y es -1 si el producto de las N entradas es -1. Para resolver este problema se implemento la red de la Fig. 21.

En esta red se uso como función costo el MSE (sin regularización), como optimizador el SGD y como función de activación tanto en la capa oculta como en la salida la función *tanh*, debido a que devuelve valores entre -1 y 1. Para el entrenamiento se uso un *learning rate* de 0.01, 1000 épocas y cada corrida del modelo se repitió 100 veces para tener un promedio del comportamiento, ya que presenta variabilidad dependiendo de la inicialización con semillas arbitrarias.

En las Figs. 22, 23, 24 y 25 se puede ver los valores promedios de las curvas de loss y accuracy en función de las épocas con bandas de desvío. Esto se hizo para $N = \{2, 3, 4, 5, 6\}$ y $N' = \{2, 4, 8, 16\}$. En primer lugar para $N = 2$, se ve que a medida que se aumenta N', el accuracy mejora y converge mas rápido y cerca a 1. Para $N = 3$, tenemos mayor complejidad, por lo que se ve que valores bajos de N', como 2, no logran converger a un accuracy de 1. Para $N = 4$, vemos como el accuracy ya no logra converger a 1, pero se logra tener accuracy mayor a 0.8 con $N' = \{4, 8, 16\}$. Finalmente con $N = 6$, vemos que el accuracy no logra superar el 0.6, esto se debe a que el problema se vuelve muy complejo para resolverlo con los valores de N' usados.

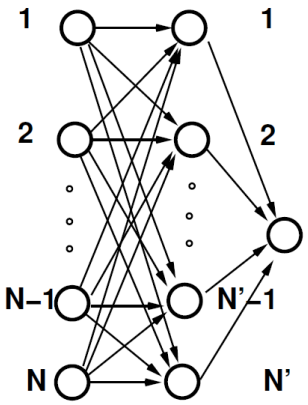

Figura 21: Arquitectura usada para el problema paridad.

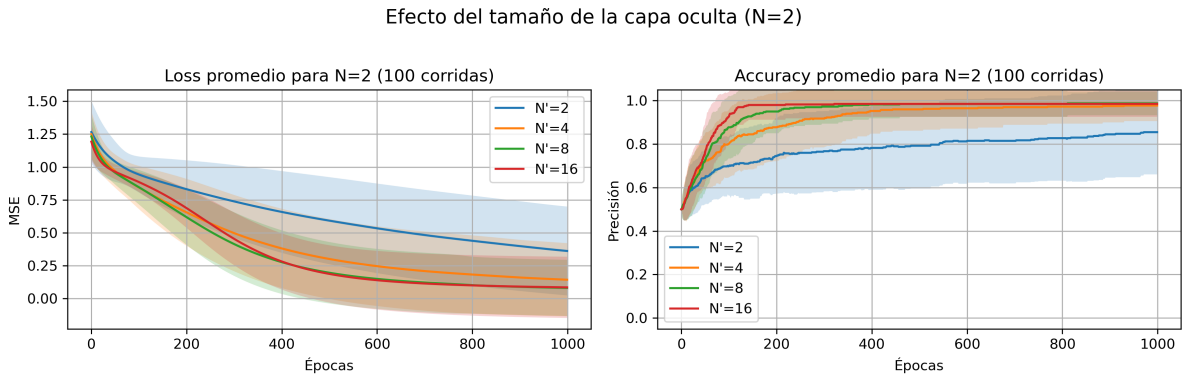

Figura 22: Curvas de loss y accuracy con $N = 2$ y $N' = \{2, 4, 8, 16\}$.

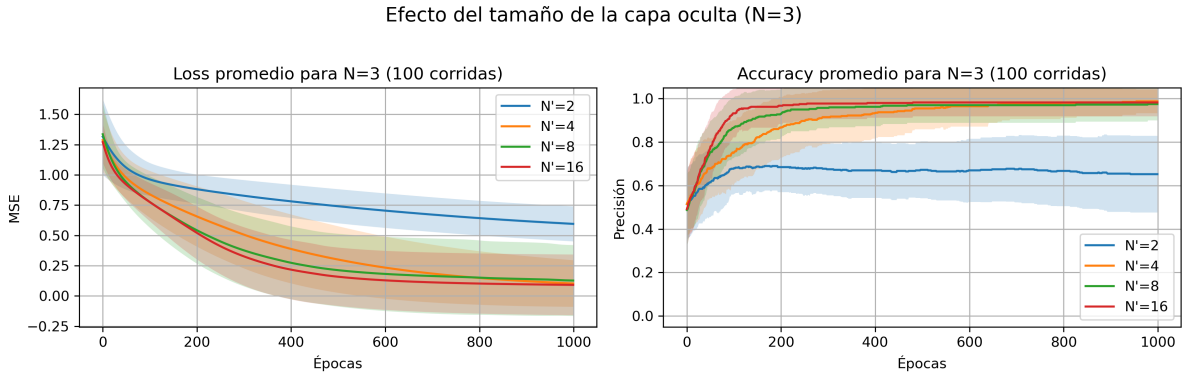

Figura 23: Curvas de loss y accuracy con $N = 3$ y $N' = \{2, 4, 8, 16\}$.

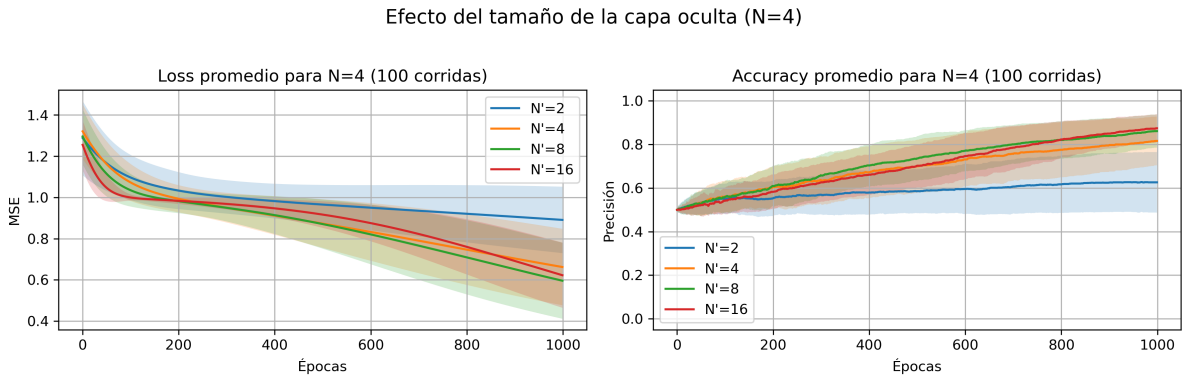

Figura 24: Curvas de loss y accuracy con $N = 4$ y $N' = \{2, 4, 8, 16\}$.

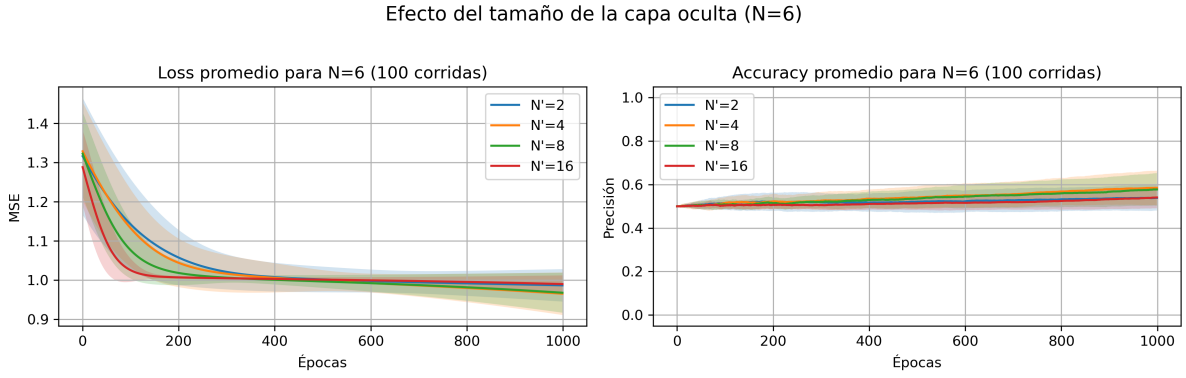

Figura 25: Curvas de loss y accuracy con $N = 6$ y $N' = \{2, 4, 8, 16\}$.

Ejercicio 8.

Usando la implementación del problema 6 se resolvió el problema 3 agregando una capa de 100 neuronas con la misma función de activación de la capa oculta existente (sigmoidal).

El entrenamiento se llevo a cabo con *learning rate=0.01*, *batch size* = 500, *lambda regularización=1e-3* y *epochs* = 2000. En la Fig. 26 se pueden observar las curvas de precisión y función de costo. Los valores de accuracy finales obtenidos fueron:

Accuracy en train: 0.4982 || Accuracy en validation: 0.4746 || Accuracy en test: 0.472

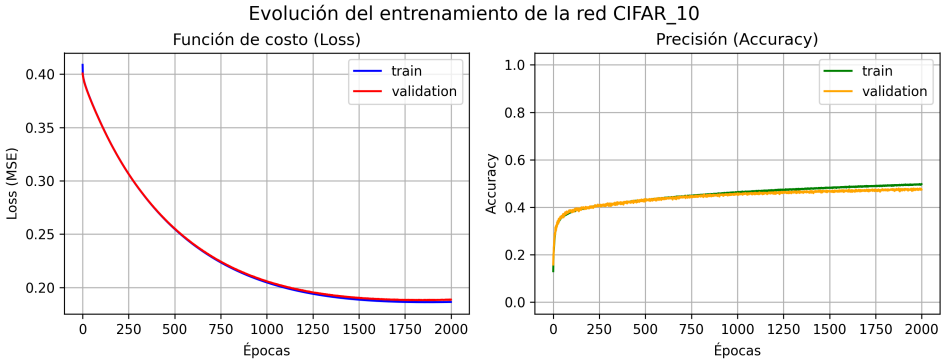

Figura 26: Curvas de loss y accuracy para el problema CIFAR-10.

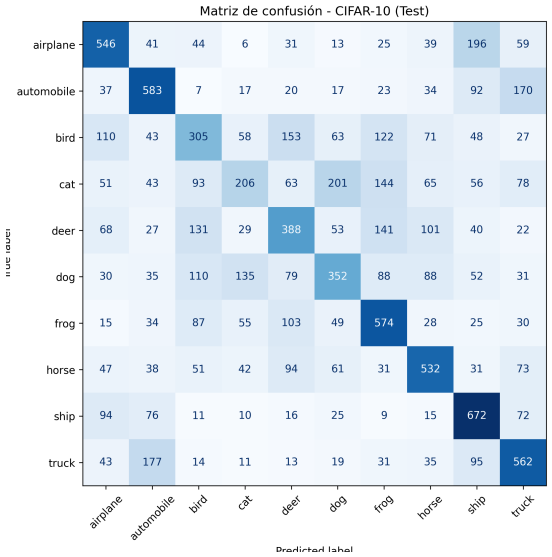

Figura 27: Matriz de confusión de los datos de test de CIFAR-10.

No se observa una mejora en el accuracy obtenido en los datos de test 47.2 % vs. 47.0 % en la red del problema 3. Teniendo en cuenta que agregamos una capa oculta de 100 neuronas, no vemos que la mejora sea significativa.

2. Apéndice

Los códigos con los ejercicios resueltos se encuentran en el repositorio: `https://github.com/AnaMorresi/DeepLearning`.