Proyecto 4

1st Ian Augusto Cortez Gorbalan
ian.cortez@utec.edu.pe
970895554

2nd Plinio Matías Avendaño Vargas
plinio.avendano@utec.edu.pe
927144823

I. Introducción

Este proyecto busca emplear el modelo supervisado denominado *Multilayer Perceptron* para realizar predicciones. En el presente trabajo, se va a utilizar un *dataset* de imágenes de lenguaje de señas. Este *dataset* contiene 27455 elementos registrados que corresponden a las imágenes almacenadas. Cada una de los elementos del *dataset* tiene como primer columna un *label* que indica que tipo de letra representa la imagen, mientras que el resto de las columnas representan los *pixel* de la imagen de 28×28. El objetivo principal es realizar predicciones sobre este *dataset* en base a los *pixel* de cada una de las imágenes representadas en el *dataset*. Asimismo, se busca graficar el error de aprendizaje producido por el modelo. Como objetivo secundario, se busca determinar la cantidad de épocas, un parámetro de aprendizaje y una función de activación apropiada para este *dataset* que permita reducir el error de aprendizaje del modelo sin generar un *overfitting* o un *underfitting*.

II. Explicación

A. Neurona

Una *Perceptron* o neurona, inspirados en su contraparte biológica, permite representar el paso de información. En las redes neuronales, esta tiene una cantidad de valores de entrada más un *bias* asociado a la neurona y solo tiene una salida. Cada neurona se encarga de obtener un valor neto con respecto a los valores de entrada y los pesos correspondientes para cada uno.

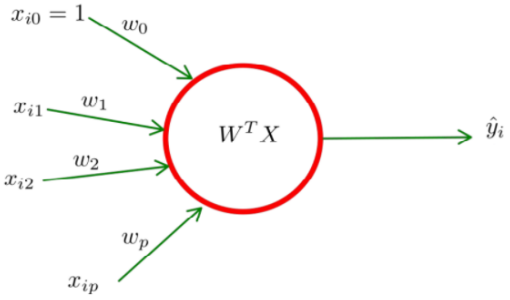

Fig. 1: Neurona Artificial

Los pesos para los valores de entrada son los valores a modificar, esto permitirá modificar o asignar un nivel de relevancia a cada input, entonces dado este modelo, el valor neto de una neurona sera el siguiente.

$$\text{Net} = W^T X + B$$

Donde W^T representa la matriz transpuesta de pesos, X representa los valores de entrada y B el valor del *bias* para esta. Después de obtener el valor neto de una neurona, este pasa por una función de activación. La cual afectará el valor que devolverá y este será el resultado que será transmitido como salida, esto es equivalente al potencial de acción en su símil biológico, las funciones de activación que se usaremos, son:

1) Sigmoide: Esta función de activación devuelve un valor entre 0 y 1 que representa una probabilidad para el valor neto obtenido por una neurona. Esta función es la siguiente:

$$h = \frac{1}{1 + e^{-\text{Net}}}$$

2) RELU: Esta función de activación devuelve un valor igual a 0 si el valor neto producido por la neurona es menor o igual que 0. Esta función es la siguiente:

$$h = max(0, \text{Net})$$

3) Tanh: Esta función de activación devuelve un valor entre -1 y 1 en base al valor neto producido por la neurona. Esta función es la siguiente:

$$h = tanh(\text{Net}) = \frac{e^{\text{Net}} - e^{\text{-Net}}}{e^{\text{Net}} + e^{\text{-Net}}}$$

B. Multilayer Perceptron

Multilayer Perceptron es un modelo supervisado de redes neuronales. Este modelo consta de múltiples capas de neuronas que se encargan de procesar un conjunto de datos de ingreso. La conexión entre estas se representara matricialmente.

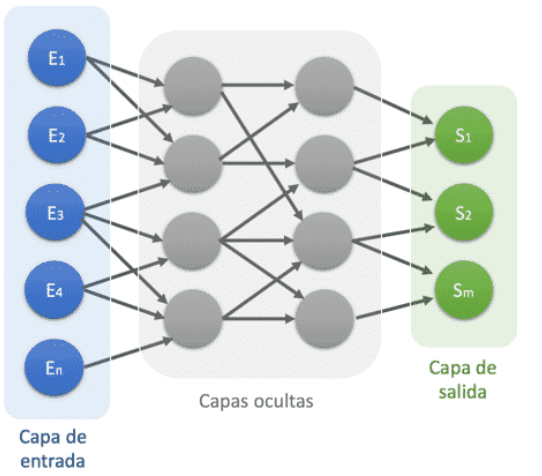

Fig. 2: *Multilayer Perceptron*

El modelo se basa en dos algoritmos importantes que le permiten producir una salida, *forward propagation* y *backward propagation*, el primero consiste en propagar los resultados de cada neurona de atrás hacia adelante, el segundo en calcular el error desde la ultima capa hasta las iniciales. Estos pasos son repetidos una cantidad de épocas específicas siempre realizando el *forward* y después el *backward*.

C. Forward propagation

En este primer paso, se toma el *dataset* y a cada neurona de la primera capa se le asigna una característica de un sample iésimo. Las neuronas luego producen un valor neto que es introducido a la función de activación (relu, tanh o sigmoidea) y estas producen las salidas que son transmitidas a la siguiente capa oculta. Este proceso se repite hasta llegar a la capa de salida en la cuál, se empleara una función *Softmax* para transformar a una probabilidad, el output de cada neurona.

1) Softmax: La función *Softmax* convierte un vector de valores reales a valores de 0 a 1 que al sumarse todos los elementos resultan en 1. El vector de entrada puede tener valores de diferentes magnitudes pero al pasarse por esta función siempre terminarán dentro del intervalo de 0 a 1.

$$\sigma(z) = \frac{e^{z_1}}{\sum_{j=1}^{K} e^{z_j}}$$

Donde z representa un vector de elementos, z_i y z_j representan elementos del vector z y K representa la cantidad de elementos del vector.

D. Backward propagation

En este paso se busca actualizar los pesos para las transiciones entra de cada una de las capas del *Multilayer Perceptron* para minimizar la función de pérdida.

$$\mathrm{L} = \frac{1}{2N}\sum_{i=0}^{N}(y_i - h_{ji})^2$$

Donde N representa el *sample size*, y_i el elemento i del vector de resultados y, $h_j i$ es un elemento i del elemento de salida de la neurona j.

En general, se busca obtener la derivada de la función de pérdida respecto a la matriz de pesos para la capa j del *MLP*. Por lo que, por la regla de la cadena obtenemos lo siguiente:

$$\frac{\partial L}{\partial W_j} = \frac{\partial L}{\partial h_j} \times \frac{\partial h_j}{\partial z_j} \times \frac{\partial z_j}{\partial W_j}$$

Las derivadas anteriores resultan en:

$$\frac{\partial L}{\partial h_j} = -(y - h_j)$$

$$\frac{\partial h_j}{\partial z_j} = h_j * (1 - h_j)$$

$$\frac{\partial z_j}{\partial W_j} = h_{j-1}$$

Al multiplicar los resultados con un factor de aprendizaje llamado α, obtenemos una matriz que nos permitirá modificar la matriz de pesos para la capa j de la siguiente forma:

$$W_j = W_j - \alpha * \frac{\partial L}{\partial W_j}$$

III. Experimentos

Cabe destacar que en cada experimento solo detallaremos el numero de capas ocultas junto a su cantidad de neuronas ya que en la primer y ultima capa estos números son constantes (784 y 24) respectivamente, ademas en todos se utilizo la funcion sigmoide en las ocultas y en la final softmax.

1) Primera prueba: Se realizo una prueba para 5 capas y 8 neuronas por capa. Asimismo, se utilizó *learning rate* de 0.01 con 100 iteraciones para realizar el entrenamiento del modelo. Esto se hizo porque se observaba una ligera variación para cada uno de los valores de error al tener una mayor cantidad de iteraciones. A continuación se muestran 2 gráficas, la primera muestra el error del *dataset* de *training* y la segunda el error del *dataset* de *testing*.

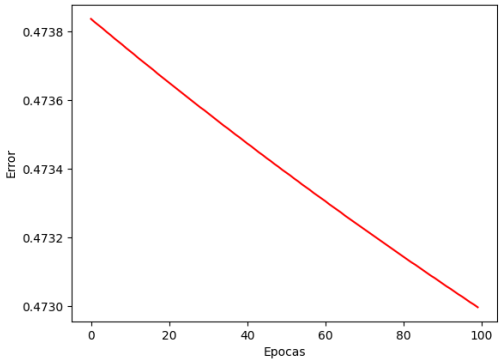

Fig. 3: Error para los datos de *training* en primera prueba

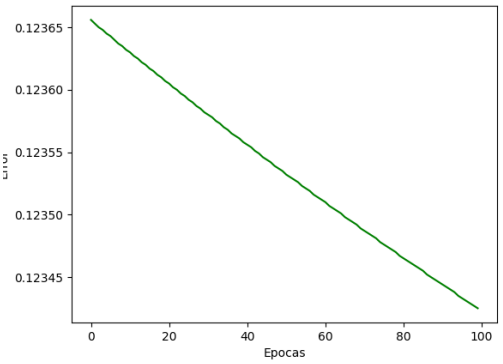

Fig. 4: Error para los datos de *testing* en primera prueba

Aciertos	Fallos
502	6670

Se realizaron 2 gráficas porque al colocarse las dos curvas obtenidas en un mismo gráfico, en las rectas mostradas no

se apreciaba un cambio significativo. Al evaluar estas dos gráficas, se aprecia que no existe *overfitting* para estos valores ya que ambos errores disminuyen y los datos de entrenamiento no hacen que el modelo tenga más error para los datos de *testing*. Por otro lado, cabe destacar que existe un *underfitting* en el modelo ya que la cantidad de capas empleadas no es la apropiada. Lo más destacable para esta prueba fue que el error para los datos de *testing* es mucho menor que el error para los datos de *training*.

2) Segunda prueba: La segunda prueba realizada empleaba 3 capas y 6 neuronas por capa. En esta prueba, se mantuvieron los parámetros de *learning rate* y la cantidad de épocas en el mismo valor.

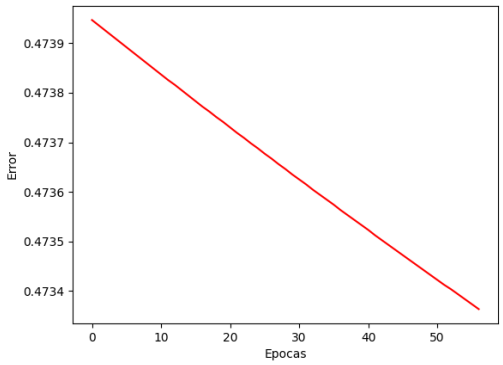

Fig. 5: Error para los datos de *training* en segunda prueba

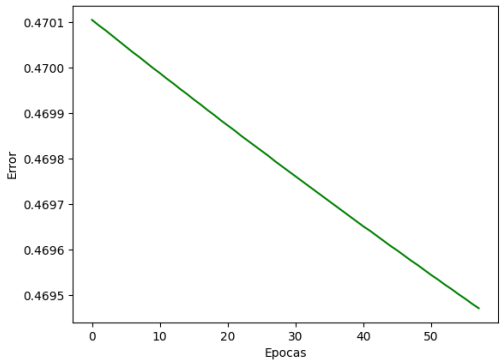

Fig. 6: Error para los datos de *testing* en segunda prueba

Aciertos	Fallos
310	6862

En estas dos gráficas se puede apreciar que los valores de error se van reduciendo sin ningún problema. En esta ocasión, a diferencia de la prueba anterior, el error de *training* y de *testing* están más cerca. Al igual que con la prueba anterior, existe un *underfitting* que evita que se adapte apropiadamente al conjunto de datos causando que el error disminuya poco hasta un punto en el que es casi insignificante.

3) Tercera prueba: La tercera prueba realizada empleaba 8 capas y 6 neuronas por capa. En esta prueba, al igual que con la segunda prueba, se mantuvieron los parámetros de *learning rate* y la cantidad de épocas en el mismo valor.

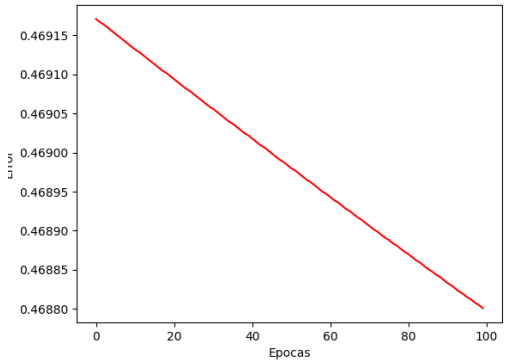

Fig. 7: Error para los datos de *training* en tercera prueba

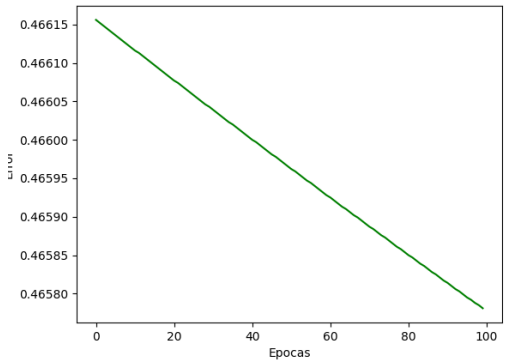

Fig. 8: Error para los datos de *testing* en tercera prueba

Aciertos	Fallos
625	6547

Estas dos gráficas son bastante similares a las mostradas en la segunda prueba. En ellas se puede apreciar que los valores de error se van reduciendo sin ningún problema y que el error de *training* y de *testing* están más cerca. Del mismo modo, existe un *underfitting* que evita que se adapte apropiadamente al conjunto de datos causando que el error disminuya poco hasta un punto en el que no afecta mucho al modelo.

IV. Conclusiones

En conclusión, se ha desarrollado un modelo de *MLP* que tiene dificultades para adaptarse apropiadamente al conjunto de datos que se la da para entrenarlo. Esto se debe principalmente a la cantidad de capas y neuronas utilizadas ya que al utilizar más de la cantidad mostradas en las pruebas llevaba a errores como *overflow* de memoria. Asimismo, es posible que aumentar la cantidad de épocas no afecte de forma significativa el aprendizaje del modelo.

References

[1] Deisenroth, M. P., Faisal, A. A., & Ong, C. S. (2021). Mathematics forMachine Learning (English Edition). Cambridge University Press.

[2] ¿Qué es una Red Neuronal? Parte 3.5 : Las Matemáticas de Backpropagation — DotCSV
[3] One LEGO at a time: Explaining the Math of How Neural Networks Learn, Omar Florez