Proyecto 4

1st Barrios, Alonso
Universidad de Tecnología e Ingeniería
Ciencia de la Computación
Lima, Perú
alonso.barrios@utec.edu.pe

2nd Rodriguez, Mauricio
Universidad de Tecnología e Ingeniería
Ciencia de la Computación
Lima, Perú
mauricio.rodriguez@utec.edu.pe

3rd Tenazoa, Renzo
Universidad de Tecnología e Ingeniería
Ciencia de la Computación
Lima, Perú
renzo.tenazoa@utec.edu.pe

***Abstract*—Este proyecto está desarrollado para poner a prueba lo aprendido acerca de redes neuronales. El objetivo del proyecto es crear una red neuronal capaz de clasificar un *set* de datos de *testing*de manera eficiente. La red neural será entrenada con un *set* de datos de *training* pertenecientes al mismo *dataset*.**

Index Terms*—redes neuronales, error, *sign language mnist

I. Introducción

Con el objetivo de poner a prueba la red neuronal se utilizó el *dataset* Lenguaje de señas, el cual consta de 27 455 intancias con atributos que representan los pixeles de imágenes en blanco y negro. Se buscó clasificar según sus características (pixeles) a qué letra pertenece, cabe resaltar que se excluyeron como *outputs* las letras **J** y **Z** ya que requieren movimiento en la mano para poder ser interpretados. Además se graficaron los errores obtenidos durante el entrenamiento.

II. Explicación

Para construir la red neuronal se hizo una implementación en C++ y se usó la librería *Sklearn* en Python.

A. C++

Para esta implementación se hizo uso de la librería *eigen* que nos facilitó las operaciones matriciales y vectoriales del problema. Además se hizo uso de:

- **Layer:** Esta es una clase que se usó como una estructura de datos, donde se atributos tales como los pesos (w), el bias (b), la respectiva función de activación de la capa, la cantidad de conexiones y neuronas.
- **Layer::create():** esta función se encargaba de crear la red neuronal dada una topología.
- **NeuralNetwork:** Esta clase era la encargada de entrenar el modelo, así como predecirlo. Tiene como atributos la topología de la red neuronal, el parámetro de aprendizaje (α) que se usa para la regresión, la cantidad de épocas que se usará en el entrenamiento, la función de costo así como su derivada y un *vector* de errores que se usará para almacenar los errores del entrenamiento.
- **NeuralNetwork.train():** Este método se encarga de realizar tanto el *Forward Propagation* como el *Backpropagation* a una sola fila del *dataset*.
- **NeuralNetwork.fit():** Este método se encarga de entrenar a la red neuronal fila por fila, este proceso se repetirá dependiendo de la cantidad de épocas dadas.
- **NeuralNetwork.predict():** Este método se encargará de predecir las salidas de un *dataset* de testing.

B. Sklearn

En cuanto al uso de *Sklearn* se utilizó **MPLClassifier** que es una red *Multilayer Perceptron*. Se hizo uso de las siguientes funciones:

- **MLPClassifier():** Esta función de encarga de crear la red neuronal dada una topología (*hidden_layer_sizes*), optimizador (*solver*), épocas (*max_iter*) y la función de activación (*activation*).
- **fit():** Esta función se encarga de entrenar la red neuronal dado un *x_train* y *y_train*.
- **predict():** Esta función se encarga de predecir las salidas de un *dataset* de entrenamiento. Este proceso se debe hacer después de entrenar el modelo.
- **accuracy_score():** Esta función se encargará de retornarnos el *accuracy* de la red neuronal.

C. Preprocesamiento

Antes de entrenar las redes neuronales se realizó un preprocesamiento de los datos usando las siguientes funciones:

- **normalize():** Esta función se encarga de normalizar la data con el uso del algoritmo *Min-Max*.
- **preprocessing():** Esta función se encarga de leer el *dataset* original, eliminar las columnas que tengan *na*, normalizar los datos, chocolatear los datos de entrenamiento y exportar los datos a un nuevo archivo.

III. Experimentos

A. C++

En la implementación de C++ no se obtuvieron resultados positivos con el *dataset* dado. Algunos resultados son:

- Si se usa la función de activación sigmoidea el valor de los errores incrementa, lo que nos indica que ocurre un *underfitting*.
- Si usamos las funciones de activación *tanh*, *relu* y *softmax* el valor de los errores es *nan*.

Se hizo una segunda prueba con otro *dataset* en forma circular [1]. La idea es clasificar si el punto se encuentra en el círculo interno o externo.

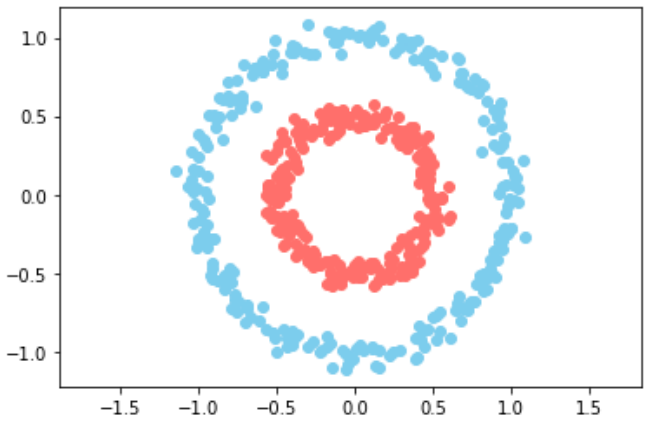

Fig. 1. *Dataset* circular

Se obtuvo el siguiente gráfico de error:

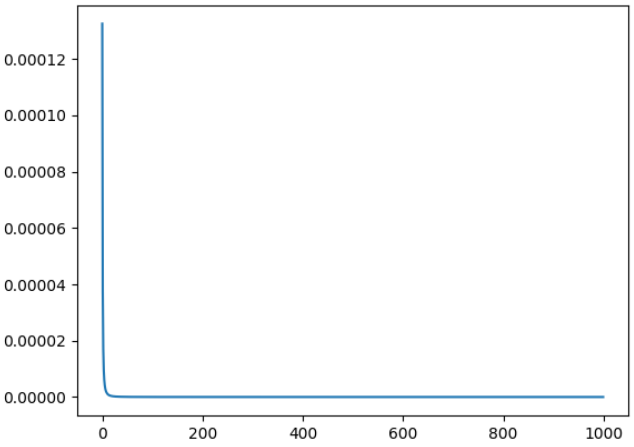

Fig. 2. Gráfica de error - *dataset* circular

Lo que nos dice que el modelo a sido entrenado correctamente, ya que se aproxima a 0. Sin embargo el *accuracy* nos sale un 50%, dado estos dos resultados podemos deducir que nos encontramos en un mínimo local.

B. Sklearn

Usando esta librería se probaron diferentes topologías así como diferentes optimizadores y funciones de activación. Se obtuvieron las siguientes gráficas de error. Se probaron 3 funciones de activación y por cada una de ellas 4 variaciones distintas.

- Variante 1: Utiliza 3 *inner layers* de 50 neuronas cada una, y una optimización de la estocástica de la gradiente descendiente propuesta por Kingma, Diederik, y Jimmy Ba
- Variante 2: Utiliza 3 *inner layers* de 50 neuronas cada una, y la estocástica de la gradiente descendiente clásica.
- Variante 3: Utiliza 3 *inner layers* de 20 neuronas cada una, y una optimización de la estocástica de la gradiente descendiente propuesta por Kingma, Diederik, y Jimmy Ba
- Variante 4: Utiliza 3 *inner layers* de 50 neuronas cada una, y la estocástica de la gradiente descendiente clásica.

1) Función de activación Relu

```
Accuracy función relu variante 1: 69.0%

Accuracy función relu variante 2: 69.0%

Accuracy función relu variante 3: 51.0%

Accuracy función relu variante 4: 57.99999999999999%
```

Fig. 3. Accuracy obtenido por variante en Relu

Tanto la variante 1 como la variante 2 obtuvieron el mismo porecentaje de *accuracy* con un 69% Por lo que en este caso la diferencia entre estocástica no fue significativa. Sin embargo, al disminuir la cantidad de neuronas por *layer*, se observa que la variante 3 tiene un mayor *accuracy* que la variante 4 por lo que la estocástica de gradiente descendiente clásica tuvo un mejor resultado.

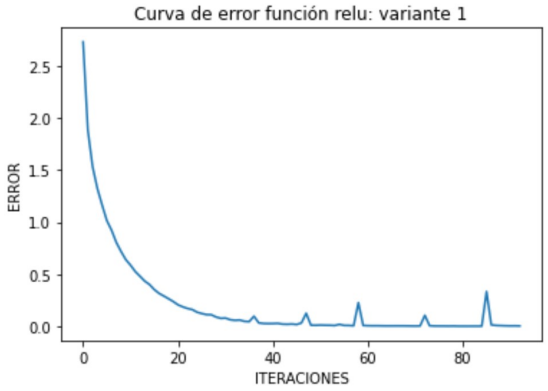

Fig. 4. Curva de error Relu: Variante 1

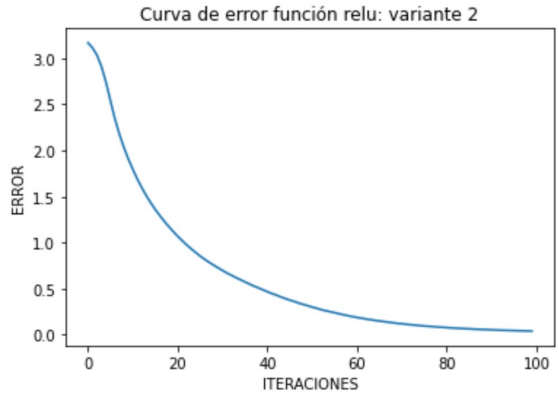

Fig. 5. Curva de error Relu: Variante 2

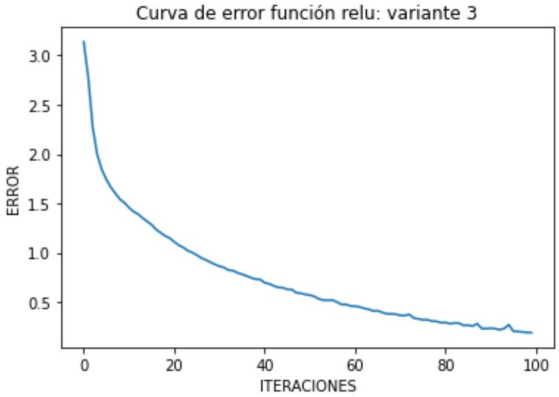

Fig. 6. Curva de error Relu: Variante 3

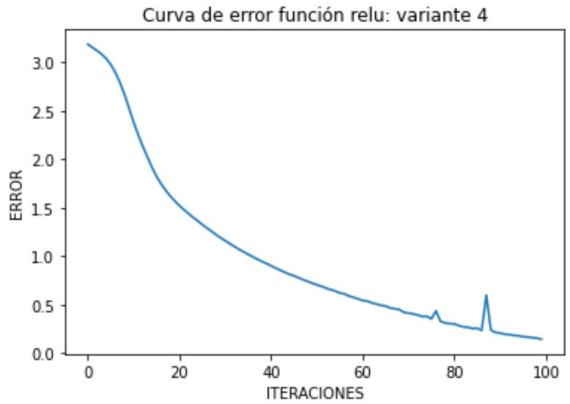

Fig. 7. Curva de error Relu: Variante 4

Podemos observar que si bien la variante 1 y la variante 2 tuvieron el mismo porcentaje de *accuracy* la variante 1 necesitó de menos iteraciones para disminuir su error. Aunque, la variante 2 lo hizo de manera más uniforme. Asimismo, la reducción de neuronas hizo que el error disminuya de forma similar para la variante 3 y 4.

2) Función de activación Tangencial Hiperbólica

```
Accuracy función tanh variante 1: 72.0%

Accuracy función tanh variante 2: 69.0%

Accuracy función tanh variante 3: 65.0%

Accuracy función tanh variante 4: 63.0%
```

Fig. 8. Accuracy obtenido por variante en Tanh

Se observa que a diferencia de la función Relu, todas las variantes tienen una alto grado de *accuracy*. Asimismo, parece que la variación de estocástica o de cantidad de neuronas por *layer* afecta en menor grado que al utilizar la función Relu.

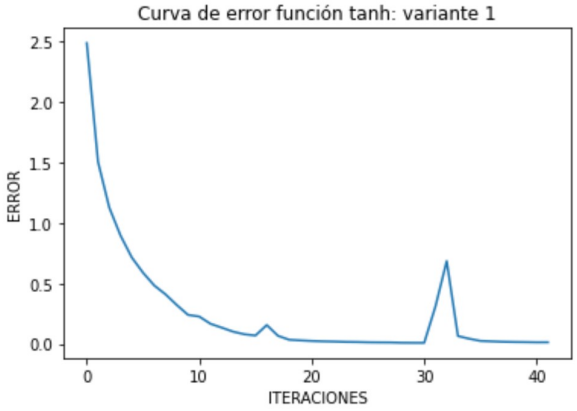

Fig. 9. Curva de error Tanh: Variante 1

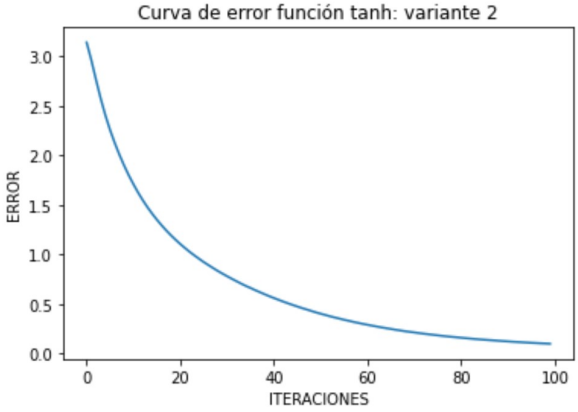

Fig. 10. Curva de error Tanh: Variante 2

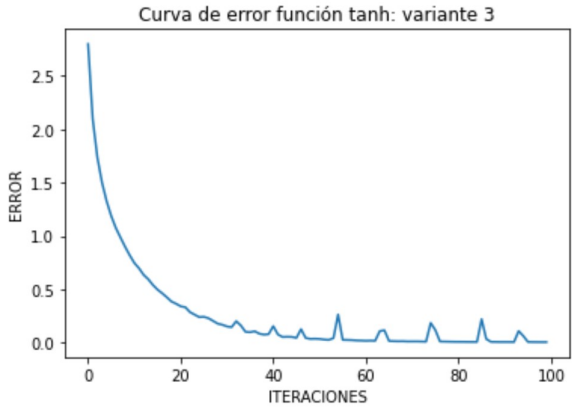

Fig. 11. Curva de error Tanh: Variante 3

Se observa que la variación que disminuye más rápido su error es la 1. Esta solo necesita de 40 iteraciones para llegar a converger, aunque se muestra irregular al inicio de las 30 iteraciones. El resto de variaciones necesita una cantidad similar de iteraciones para converger. En general, se puede observar que el error disminuye de forma más uniforme y estable en las variaciones 2 y 4 que utilizan la

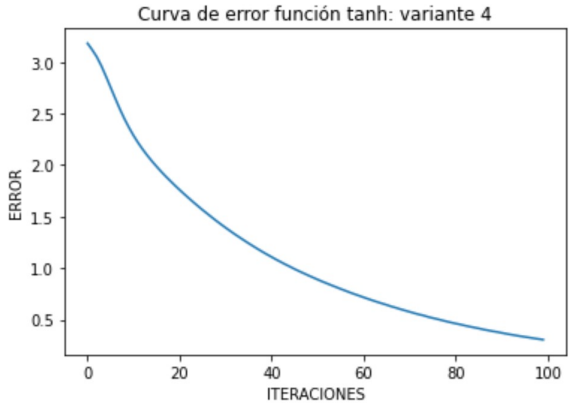

Fig. 12. Curva de error Tanh: Variante 4

estocástica clásica de gradiente descendiente. Mientras que las variaciones 1 y 3, que utilizan la optimizada, disminuyen de forma más inestable disminuyendo rápidamente al inicio y posteriormente comportándose de forma un poco volátil. Subiendo y bajando valores.

3) Función de activación Sigmoidea

Este es el caso más inusual de las 3 funciones de activación de las que se ha hecho *testing*. El *accuracy* es de tan solo 2% en las variaciones que utilizan la estocástica clásica, sin importar el número de neuronas que lleven por *layer*. Sin embargo, también se observa que en las otras variaciones el *accuracy* no es tan alto, pero es estable y no disminuye tanto al pasar de 50 a 20 neuronas por *layer*.

```
Accuracy función sigmoidea variante 1: 62.0%

Accuracy función sigmoidea variante 2: 2.0%

Accuracy función sigmoidea variante 3: 55.00000000000001%

Accuracy función sigmoidea variante 4: 2.0%
```

Fig. 13. Accuracy obtenido por variante

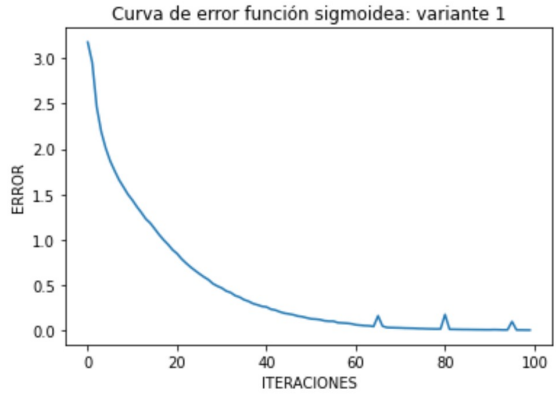

Fig. 14. Curva de error Sigmoidea: Variante 1

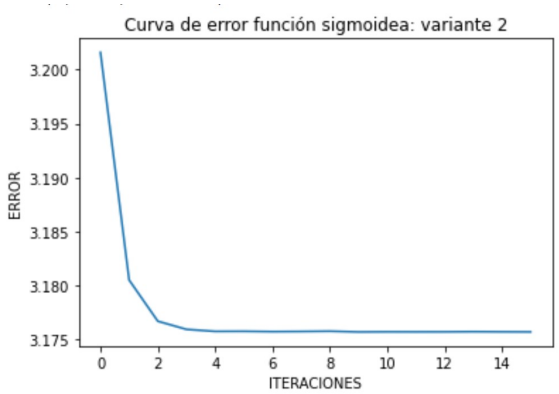

Fig. 15. Curva de error Sigmoidea: Variante 2

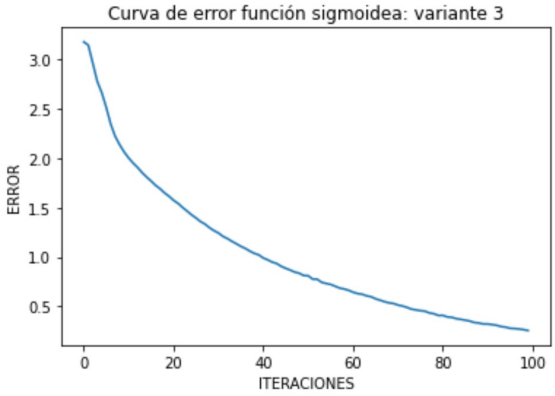

Fig. 16. Curva de error Sigmoidea: Variante 3

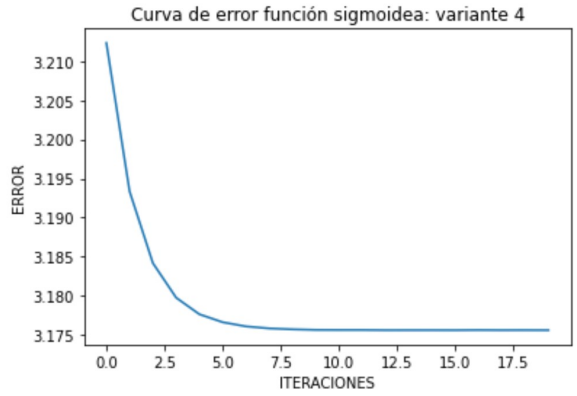

Fig. 17. Curva de error Sigmoidea: Variante 4

CONCLUSIONES

- En la implementación hecha en C++ hay posibilidad de que el error sea en el cálculo al momento de hacer *backpropagation*.
- En la implementación en C++ con el segundo *dataset* puede que se esté cayendo en un mínimo local debido a que la gráfica de error no concuerda con el *accuracy*.
- Sklearn nos facilitó de una mejor manera el entre-

namiento de los datos frente a otras librerías dadas por el mismo lenguaje.

Repositorio

Enlace al repositorio de Github:
https://github.com/alonso804/ai-project-4

References

[1] Neural networks: ¿de qué son capaces las redes neuronales artificiales? (2020, 20 octubre). IONOS Digitalguide. https://www.ionos.es/digitalguide/online-marketing/marketing-para-motores-de-busqueda/que-es-una-neural-network/.

[2] Santana, D. [Dot CSV]. (2018, 3 octubre). ¿Qué es una Red Neuronal? Parte 3 : Backpropagation — DotCSV [Vídeo]. YouTube. https://www.youtube.com/watch?v=eNIqz_noix8feature=youtu.be.