Proyecto 4 - Inteligencia Artificial

1st Alejandro Otero
dept. of Computer Science
University of Engineering and Technology
Lima, Perú
alejandro.otero@utec.edu.pe

2nd Angélica Sánchez
dept. of Computer Science
University of Engineering and Technology
Lima, Perú
angelica.sanchez@utec.edu.pe

I. Introducción

Dentro de los campos de Inteligencia Artificial, *Machine Learning* y *Deep Learning* existen las llamadas Redes Neuronales, las cuales reflejan el comportamiento del cerebro humano permitiendo a los programas reconocer patrones y resolver distintos problemas en los campos mencionados [1]. Estas redes están compuestas principalmente por [2] [3]:

- 1 *Input layer*: contiene nodos con las características de las muestras dentro del *dataset* respectivo (pueden ser imágenes, videos, audios, entre otros).
- 1 o más *Hidden layers*: conformado por nodos que almacenan información correspondiente calculos necesarios para generar un resultado que pasará a través de estas.
- 1 *Output layer*: contiene nodos con los valores obtenidos luego de los distintos cálculos realizados a través de los *hidden layers*.

El *dataset* utilizado en este proyecto representa fotos de las letras del abecedario correspondientes al lenguaje de señas, representadas con símbolos. Esta cuenta con 7173 muestras y con 785 atributos cada una que representan información numérica de los distintos pixeles de la imágen, los cuales serán utilizadas como las variables independientes (x) y 1 atributo adicional correspondiente a la etiqueta de la imagen que representa la letra de la foto, el cual representa la variable dependiente (y). Adicionalmente, no se incluye dentro de las clases la letra J ni Z, ya que estas para ser representadas se necesita realizar un movimiento, por lo que al final son 24 clases. Por otro lado, hay un segundo *dataset* que se usa solo para poder hacer los *tests* y verificar si es que efectivamente la red neuronal funciona o no.

II. Explicación

Para trabajar con la red neuronal (perceptrón multicapa), planteamos la siguiente estructura:

1) Preparar el *dataset* a utilizar.
2) Definir las funciones de matrices (multiplicación, resta, etc.).
3) Declarar las diversas funciones de activación:
   - Sigmoide
   - Tanh
   - RELU
   - Softmax
4) Las funciones principales que realizan el procesamiento de la red neuronal son
   - *Feedforward*, el cual es el algoritmo que procesa la muestra, *input*, a través de sus n *layers* con distintos nodos y el resultado indica a qué clase pertenece la muestra a través de valores del 0 al 1, donde el valor más alto sería la clase a la que pertenece el dato.
   - *Backpropagation*, el cual hace lo opuesto al *Feedforward*, es decir, en vez de ir desde el inicio al final, va del final al inicio. En este algoritmo se "entrena" a la red neuronal, debido a que se modifican los valores de los pesos y del bias que se encuentran entre cada par de *layers* para que a medida que se llame a *Feedforward* el resultado que se obtenga cada vez sea más certero. Esto se realiza a partir del resultado, pues al saber a qué clase pertenece la muestra, entonces se resta el "y", valor original, con el "y prima", resultado obtenido. De esta manera se van "corrigiendo los pesos y el bias" en cada par de *layers* a medida que se retrocede.
5) Además, utilizamos algunas funciones auxiliares para facilitar el cálculo de los resultados.

III. Implementación

Nuestra implementación cuenta con 3 clases:

A. NeuralNetwork

Contiene distintas funciones:

- `read_dataset` que se encarga de leer los píxeles de cada imagen de muestra e insertarlo en una matriz.
- `feed_forward` que se encarga de definir los primeros valores de los pesos entre cada par de *layers*, realizar los cálculos de los *outputs* al multiplicar estos valores de los pesos previamente calculados y los *inputs*. Esta información es almacenada en un objeto de tipo `Layer` y se guarda en un contenedor para reutilizarla en las siguientes funciones.
- `back_propagation` que se encarga de actualizar los pesos y el *bias* para que el resultado de la red neuronal sea más certero. A esta función se le pasa por parámetro el resultado del algoritmo `feed_forward` y el índice de la muestra que se está evaluando para generar la matriz de "y prima". Existen dos casos en esta función: el primero

que se da cuando queremos obtener el error entre el *hidden* y el *output*.
- `variation_forward` es muy similar a `feed_forward`, con la única diferencia que los valores de pesos, estos son obtenidos del contenedor de *layers*. Los únicos valores que se modifican son los *inputs* y *outputs* entre cada par de layers.

B. ActivFunc

Contiene las distintas funciones de activación utilizadas para realizar los experimentos y comparar el rendimiento de cada una. Además, contiene las derivadas de las mismas.

C. Layer

Representa los *hidden layers* y el *output layers*, los cuales almacenan distinta información:

- Matriz de pesos que entran al *layer*
- *Inputs* que entran al *layer*
- *Outputs* resultantes de multiplicar los pesos y los *inputs* y aplicar la función de activación. Corresponderían también a los *inputs* del siguiente *layer* en caso se trate de un *hidden layer*.

IV. Experimentos

Para analizar el comportamiento de la red neuronal tuvimos que exponerla a distintos casos, en los cuales se cambiaban el número de nodos, la función de activación, entre otros casos. De esta forma podríamos ver si es que sus resultados son más certeros para un escenario u otro e identificar a qué se podría deber esto.

A. Distintas funciones de activación

Para este experimento utilizamos las 4 funciones de activación ya mencionadas. Hay que tener en cuenta que la función de *softmax* funciona justamente para el perceptrón, porque convierte el resultado en una distribución de probabilidad normalizada. A la hora de entrenar el algoritmo la función de activación que se utilizaba para cada *layer* era la misma, por lo que solo se utilizaba una sola función de activación por experimento.

B. Distintos números de layers y de nodos

En este caso se utilizaron distintas cantidades tanto de *layers*, como de nodos para analizar el comportamiento del perceptrón. Los *layers* que se creaban tenían la misma cantidad de nodos, menos la del *input* y la del *output* que debía ser 24 en todos los casos para poder determinar a la clase a la que correspondía.

C. Distintos valores del parámetro de aprendizaje

Para estos experimentos se alternó el valor del *learning rate*, pero también lo hicimos en dos escenarios: cambiando el número de *layers* y de nodos, así como la función de activación utilizada. Hay que tener en cuenta que el *learning rate* lo cambiábamos entre 0.0001 a 0.01.

V. Resultados

A. Distintas funciones de activación

Basándonos en las curvas de error durante el entrenamiento, se observa que la función de activación que mejor rendimiento tiene es **ReLU**. Se puede apreciar en las figuras Fig.1, Fig.3 y Fig.2 que para una cantidad de 70 *hidden layers* y 80 nodos por *hidden layer*, **ReLU** es la única función en donde su error convergía en 0. A diferencia de las funciones *tanh* y *sigmoid* que a pesar de realizar una menor cantidad de iteraciones, convergían en un error mayor a 3.

B. Distintos números de layers y de nodos

Basándonos en las curvas de error durante el entrenamiento, se observa que para una cantidad mayor de *hidden layers* y de nodos por *hidden layer* el error tarda menos en converger en 0, pero luego de haber llegado a 0, este se disparaba y empezaba a incrementar. Esto se puede apreciar en las figuras Fig.4, Fig.5 y Fig.**??**, donde se ha utilizado la función de activación *ReLU*.

Además, podemos confirmar lo mencionado en la sección anterior, donde comparábamos las funciones de activación y concluíamos que *ReLU* es la mejor, observando la figura Fig.7, donde se utiliza la función de activación *tanh* y vemos que la curva de error no llega a converger en 0.

C. Distintos valores del parámetro de aprendizaje

asándonos en las curvas de error durante el entrenamiento, se observa que para un valor el error tarda menos en converger en 0, pero luego de haber llegado a 0, este se disparaba y empezaba a incrementar. Esto se puede apreciar en las figuras Fig.8, Fig.9 y Fig.10, donde se ha utilizado la función de activación *ReLU* para una cantidad de 70 *hidden layers* y 80 nodos por *hidden layer*.

VI. Conclusiones

Finalmente, luego de realizar diversos experimentos en la implementación en C++ y toparnos con distintos problemas, tuvimos que alternar el desarrollo del proyecto y utilizar *python*. Nos dimos cuenta de que los de los parámetros que se pueden usar para el perceptrón el que más influye es la función de activación, en este caso *ReLU* fue la que nos dio mejores resultados. Esto se debe a que esta función no tiene el problema de desvanecimiento de la gradiente, la cual genera que la gradiente sea bastante pequeña por lo que el peso va a cambiar mínimamente y la red neuronal no va a mejorar sus resultados. Este caso se daba con *tanh* y *sigmoid* [4]. Asimismo, la cantidad de nodos y *layers* también influyen fuertemente en el error de la red neuronal, ya que si es una cantidad baja en ambos casos el perceptrón no aprende y se tienen resultados no certeros. En tercer lugar, el *learning rate* influye de manera moderada en el error del perceptrón, ya que mientras más grande este valor el error disminuía menos, pero la diferencia entre estos casos no fue muy distinta, especialmente cuando el *learning rate* era 0.05 y 0.005. Por otro lado, este tipo de algoritmos no son muy eficientes, puesto que se demoran mucho en ejecutarse y esto complicaba el proceso de experimentos y análisis lo que nos lleva a pensar

que se podrían utilizar threads, por ejemplo, para mejorar este proceso.

REFERENCES

[1] [1] "What are Neural Networks?," Ibm.com, Aug. 17, 2020. https://www.ibm.com/in-en/cloud/learn/neural-networks (accessed Nov. 21, 2021).

[2] [2] R. Stureborg, "Artificial Neural Networks for Total Beginners - Towards Data Science," Medium, Sep. 04, 2019. https://towardsdatascience.com/artificial-neural-networks-for-total-beginners-d8cd07abaae4 (accessed Nov. 21, 2021).

[3] [3] "Neural Network: Architecture, Components Top Algorithms — upGrad blog," upGrad blog, May 06, 2020. https://www.upgrad.com/blog/neural-network-architecture-components-algorithms/ (accessed Nov. 21, 2021).

[4] [4] P. Ho, "Activation functions and when to use them," diveindatascience, Jun. 13, 2019. https://patrickhoo.wixsite.com/diveindatascience/single-post/2019/06/13/activation-functions-and-when-to-use-them

VII. ANEXOS

- Link al repositorio de GitHub: https://github.com/Oteranga/Proyecto4_IA.git

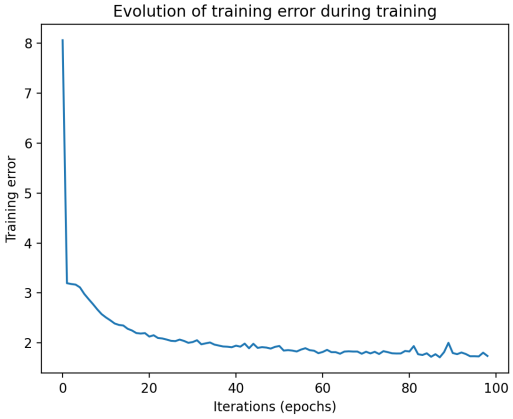

Fig. 1. Curva del error de la red neuronal utilizando la función de activación *ReLU*.

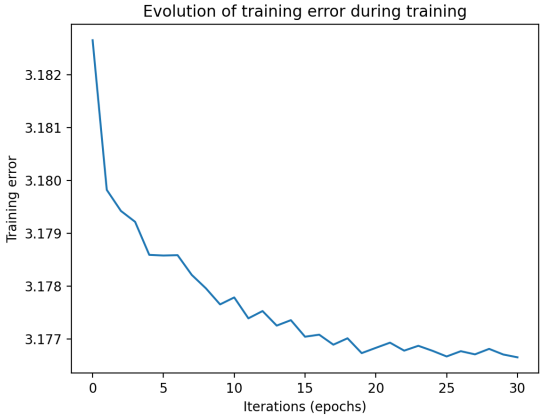

Fig. 2. Curva del error de la red neuronal utilizando la función de activación *sigmoid*.

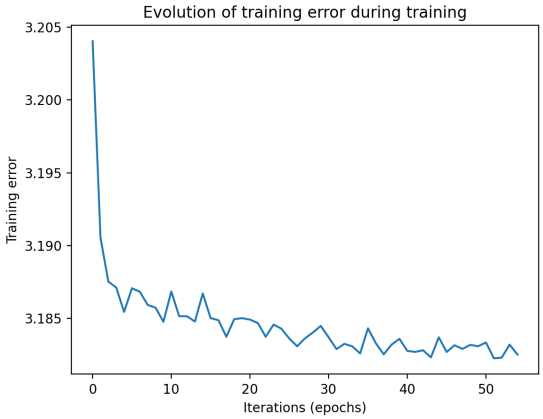

Fig. 3. Curva del error de la red neuronal utilizando la función de activación *tanh*.

Fig. 6. Curva del error de la red neuronal utilizando la función de activación *ReLU* con 150 nodos y 110 *hidden layers*.

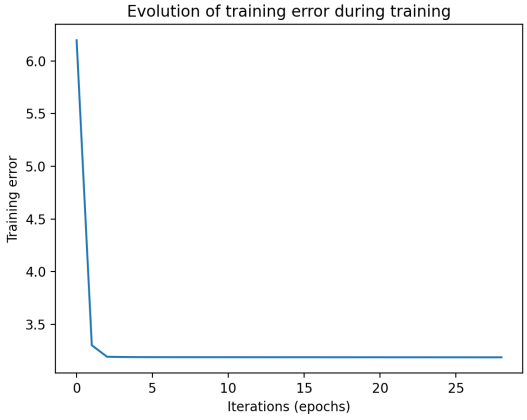

Fig. 4. Curva del error de la red neuronal utilizando la función de activación *ReLU* con 30 nodos y 40 *hidden layers*.

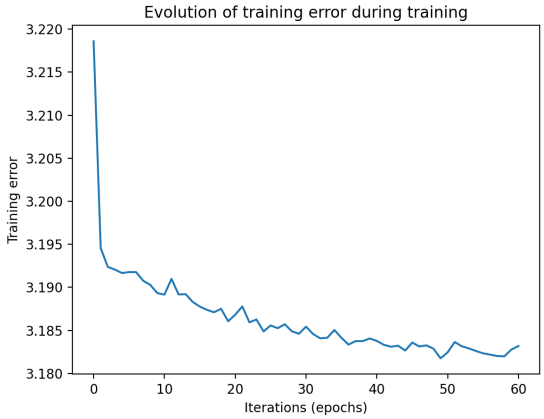

Fig. 7. Curva del error de la red neuronal utilizando la función de activación *tanh* con 30 nodos y 40 *hidden layers*.

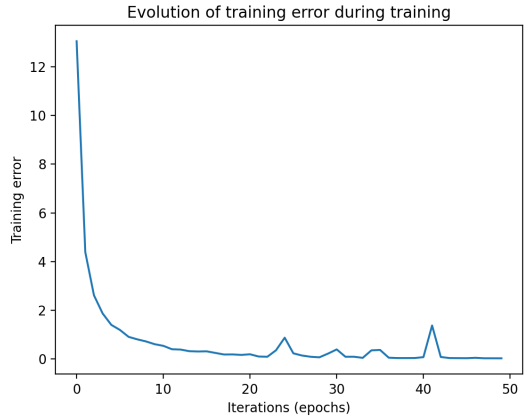

Fig. 5. Curva del error de la red neuronal utilizando la función de activación *ReLU* con 100 nodos y 90 *hidden layers*.

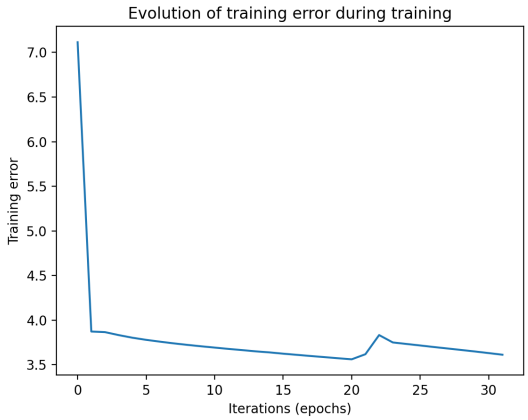

Fig. 8. Curva del error de la red neuronal utilizando la función de activación *ReLU* con un *learning rate* = 0.05

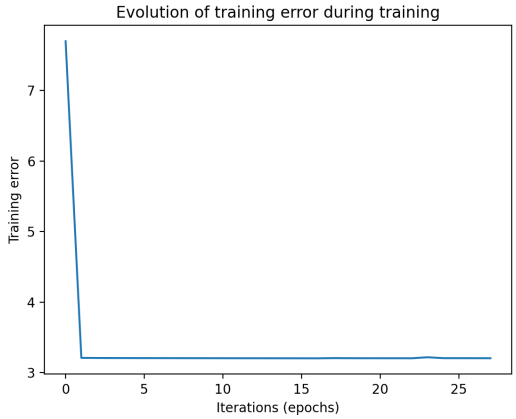

Fig. 9. Curva del error de la red neuronal utilizando la función de activación *ReLU* con un *learning rate* = 0.005

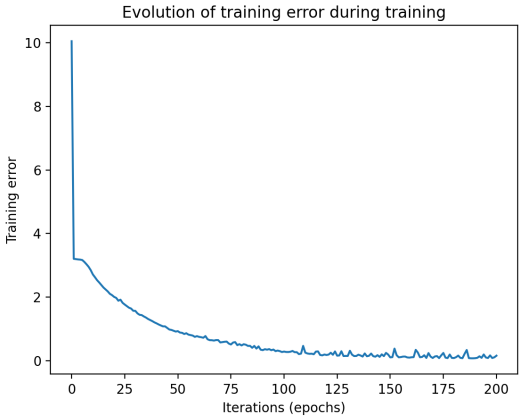

Fig. 10. Curva del error de la red neuronal utilizando la función de activación *ReLU* con un *learning rate* = 0.0005