Proyecto 2 – (Un)Confused Terminator

María Laura Pizarro Moreno

***Abstract*— Este artículo trata sobre una red neuronal que clasifica imágenes de dígitos numéricos escritos a mano utilizando el set de datos MNIST para el entrenamiento. Es importante adentrarse en el concepto de red neuronal ya que es lo que se usa en la actualidad en Inteligencia Artificial. Se hicieron varios experimentos alternando los hiper parámetros de cantidad y tamaño de capas ocultas y de porcentaje de difuminado para comparar el rendimiento en diferentes circunstancias.**

***Index Terms*— Red Neuronal, ReLU, Softmax, Cross Entropy, MNIST.**

I. INTRODUCCIÓN

Las redes neuronales son muy utilizadas en IA para resolver problemas de clasificación, regresión entre otros. Este tipo de herramienta se puede utilizar en videojuegos, investigaciones y en aplicaciones de mercado. Por ejemplo en un juego de ajedrez cuando se juega con un enemigo artificial se le puede entrenar con una red neuronal para simular que toma decisiones inteligentes y no solo aleatorias.

En la actualidad lo mas importante es que las maquinas actúen de forma inteligente para satisfacer necesidades humanas, por lo tanto se podría decir que las redes neuronales es un tema que va a tener un impacto en el futuro.

Se espera que se pueda encontrar un porcentaje de exactitud del 90% en el primer y segundo experimento, y de entre 90% y 50% en el tercer experimento. Como el primer experimento tiene mas capas ocultas se espera que tenga mejores resultados que el segundo experimento que solo tiene 1.

II. METODOLOGÍA

A. Herramientas

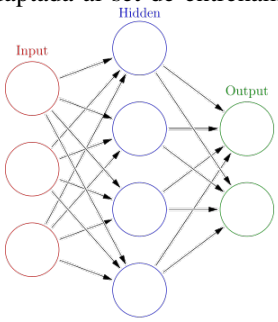
Para la ejecución de esta investigación fue necesaria la descarga de Python 2.7, Numpy 1.14.0, Sklearn 0.19.1, Pillow 5.0.0, Image, Matplot, Openpyxl 2.5.3 y el set de datos MNIST.

B. Set de datos NMIST

MNIST consta de 70000 imágenes de dígitos escritos a mano en blanco y negro de 28x28 en 10 clases. Hay 60000 imágenes de entrenamiento y 10000 imágenes de prueba. Es un subconjunto de un conjunto más grande disponible de NIST[4].

C. Red neuronal

Es un sistema computacional que es conformada por nodos de entrada, nodos ocultos y nodos de salida. Lar redes pueden tener tantos nodos como se deseen mientras que tenga como mínimo 1 cada de entrada, oculta y de salida. Su poder se encuentra en los pesos que son los que van a aparentar que el sistema es inteligente. Los pesos son números que se inician con valores aleatorios y dependiendo de los datos que va recibiendo la red son modificados para tener una red mejor adaptada al set de entrenamiento [2].



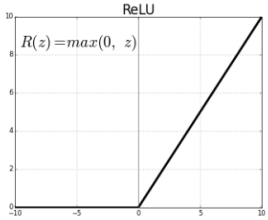
D. Función de activación

Las funciones de activación son algoritmos que ayudan a darle más importancia a las neuronas con la información más relevante, y a reducirle la importancia a la información que no le aporta mucho a la red. Existen varios algoritmos que se pueden utilizar como funciones de activación y dependiendo del problema va a ser el algoritmo que se debe utilizar [5].

E. Algoritmo ReLU

La función de activación ReLU (rectified linear unit) es la más utilizada en el mundo en este momento. Desde entonces,

se usa en casi todas las redes neuronales convolucionales o en aprendizaje profundo [3].



Como se puede observar en el gráfico, ReLU desactiva los valores negativos al cambiarlos por 0 y los positivos los mantiene igual. Su fórmula es la siguiente, en donde x es el valor al que se le debe realizar la activación:

$$f(x) = \max(0, x)$$

F. Softmax

La función Softmax calcula la probabilidad de distribución de un evento sobre 'n' eventos diferentes. En otras palabras calcula la probabilidad que tiene, en este caso, una imagen de pertenecer a cada una de las diez clases. Su formula es la siguiente:

$$p_j = \frac{e^{a_i}}{\sum_{k=1}^{N} e_k^a}$$

Se tiene Pj que es la probabilidad de las N clases y 'a' el dato a evaluar.

[1]

G. Cross entropy

Indica la distancia entre lo que el modelo cree que debe ser la distribución de salida, en este caso el resultado es 'Softmax', y la distribución correcta. Esta definido como:

$$H(y, p) = -\sum_i y_i log(p_i)$$

Se tiene una 'y' que indica la clase correcta y un p que es el resultado (el porcentaje de probabilidad) que retorno la función de probabilidad (en este caso Softmax) que es iterado por los 'i' datos.[2]

III. EXPERIMENTOS

Se realizan 3 experimentos utilizando el set de datos MNIST y se utilizan los hiper parámetros de, tamaño de capas ocultas (128, 256, 512, 1024), cantidad de capas ocultas y cantidad de difuminado. Se midie la perdida con cross entropy y Softmax. Los primeros pesos (W1, W2, W3) se generan a partir de números aleatorios. El rango de números aleatorios es aproximadamente de]-5,5[. Los pesos se normalizan con la siguiente formula

- W1: random(784 x tamaño capa oculta)/sqrt(784)
- W2: random(tamaño capa oculta x tamaño capa oculta)/sqrt(tamaño capa oculta)
- W3: random(tamaño capa oculta x 10)/sqrt(tamaño capa oculta)

random - instrucción que genera números aleatorios de N x M.

sqrt - instrucción que calcula la raíz cuadrada

Se separan los datos de MNIST por grupos (batches) de 25 imágenes que son escogidos aleatoriamente y no se repiten. De las 60000 imágenes del set de datos se apartan 10000 para hacer validaciones cada 50 batches que serán representadas con gráficos. Al final de la ejecución se va a evaluar el set de prueba con la red.

A. Experimento 1

- Hiper parámetro: Tamaño de las 2 capas ocultas (128, 256, 512, 1024)
- Porcentaje de exactitud mínima esperada: 90%

B. Experimento 2

- Hiper parámetro: Cantidad de capas ocultas (1) y tamaño de las capas ocultas (128, 256, 512, 1024)
- Porcentaje de exactitud mínima esperada: 90%

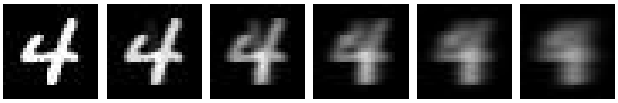
Se repite el experimento 1 pero con 1 capa.

C. Experimento 3

- Hiper parámetro: Cantidad de difuminado(0, 0.5, 1, 1.5, 2, 2.5)
- Porcentaje de exactitud mínima esperada: 50%

Se va a ir modificando el hiper parámetro hasta que se obtenga un un porcentaje de exactitud menor al esperado al evaluar con el set de prueba. Se utilizaron 2 capas ocultas de 128 neuronas.

A continuación se muestra el difuminado aplicado en las imágenes a la hora de realizar el experimento:



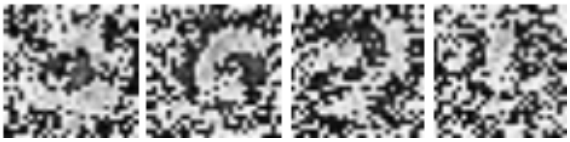
IV. RESULTADOS

A. Experimento 1

- 128

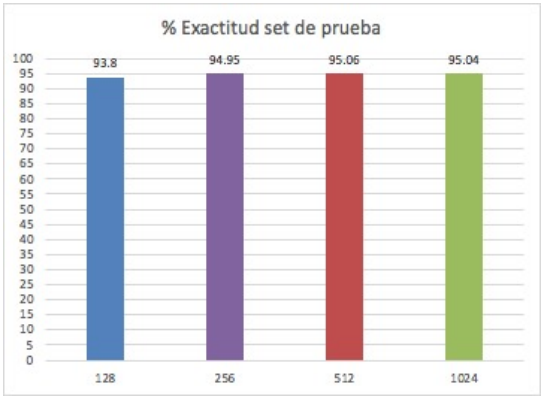
- 256

- 512

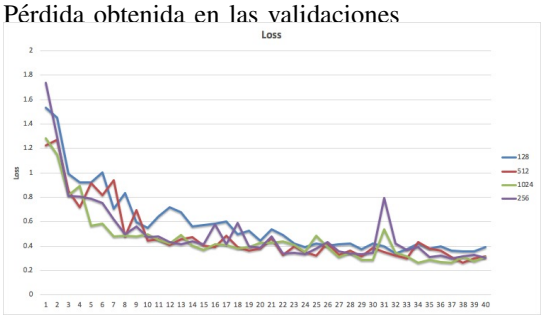


- 1024

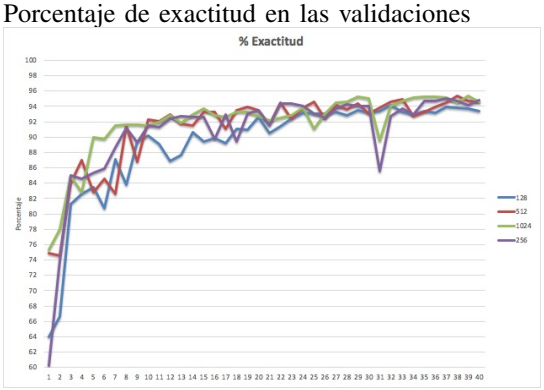
- Porcentaje de exactitud del set de prueba



- Pérdida obtenida en las validaciones



- Porcentaje de exactitud en las validaciones

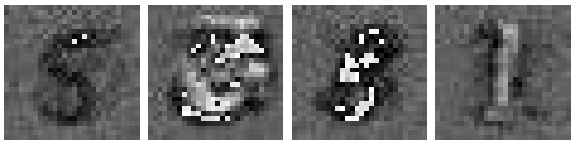


B. Experimento 2

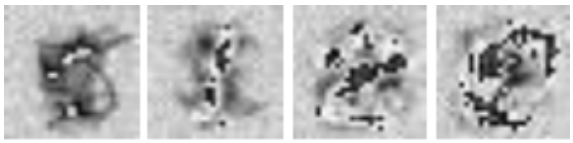
- 128

- 256

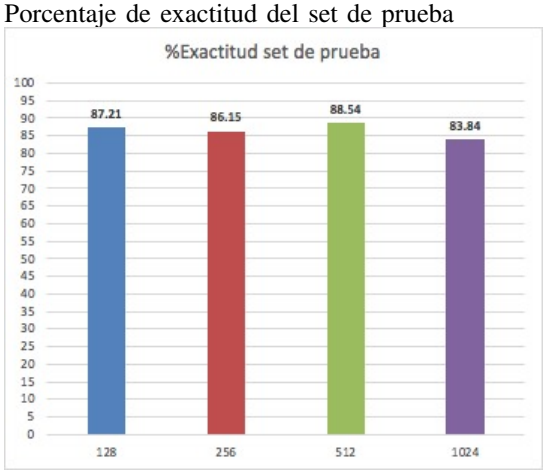
- 512



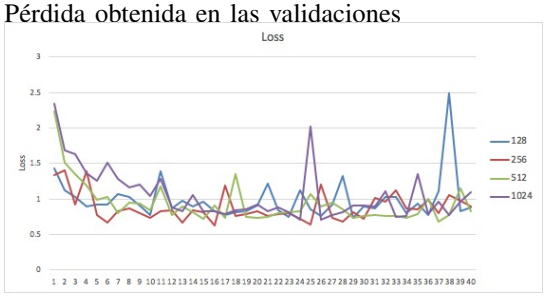
- 1024



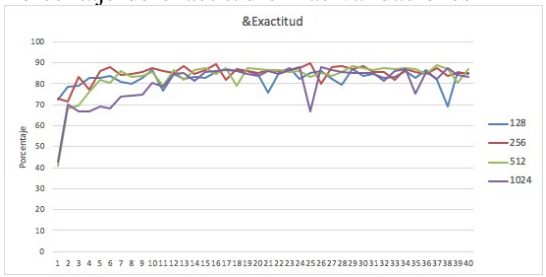
- Porcentaje de exactitud del set de prueba



- Pérdida obtenida en las validaciones



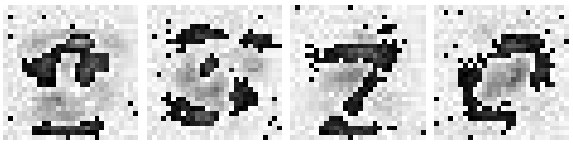
- Porcentaje de exactitud en las validaciones



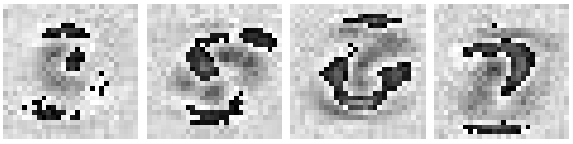
C. Experimento 3

- 0.5

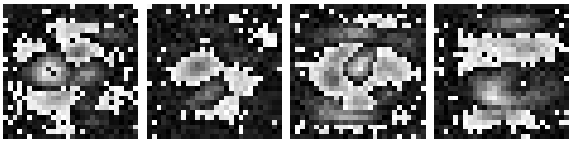
- 1



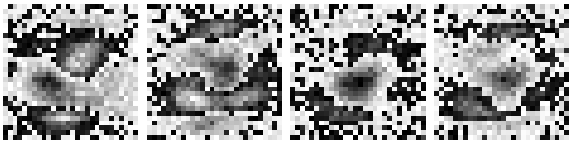
- 1.5



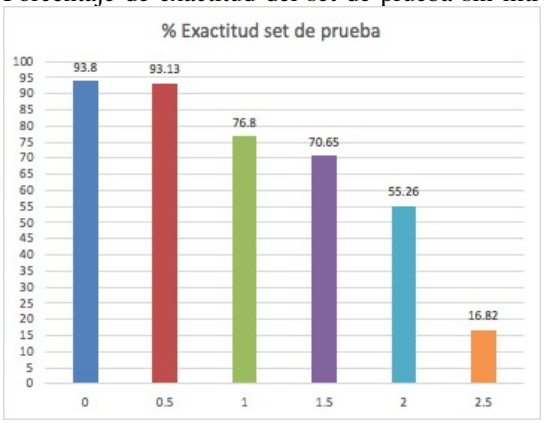
- 2



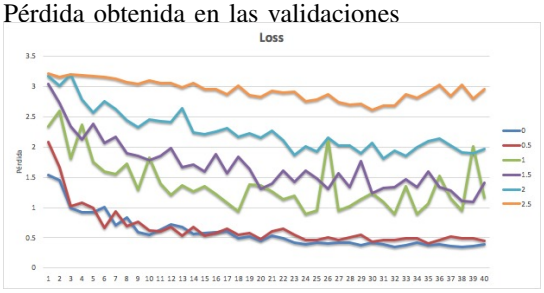
- 2.5



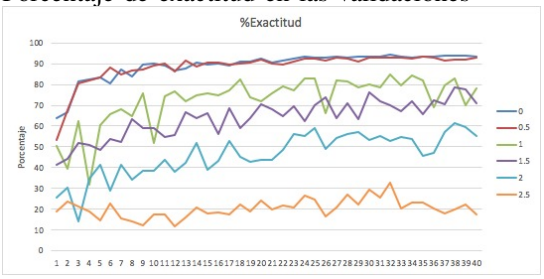
- Porcentaje de exactitud del set de prueba sin filtro



- Pérdida obtenida en las validaciones



- Porcentaje de exactitud en las validaciones



V. CONCLUSIONES

Los tres experimentos tuvieron resultados satisfactorios ya que lograron tener porcentajes altos de exactitud. La hipótesis de que el primer experimento iba a ser más exitoso que el segundo fue comprobada, los resultados del primer experimento mostraron mejores porcentajes de exactitud. El segundo experimento no cumplió con el porcentaje mínimo esperado que eral del 90% ya que obtuvo porcentajes de entre 80% y 90%. El tercer experimento no tenia un comportamiento esperado ya que cuando se incumpliera el porcentaje estipulado se iba a parar de realizar más experimentos. Aun así, el tercer experimento mostró, cómo el modelo a pesar de aprender imágenes modificadas, pudo seguir clasificando en porcentajes altos de difuminación. A futuro se podrían continuar estos experimentos con otros tipos de hiper parámetros y con mayor variedad de clases ya sea letras o signos.

REFERENCES

[1] Saimadhu Polamuri. *DIFFERENCE BETWEEN SOFTMAX FUNCTION AND SIGMOID FUNCTION*. Mar. 2017. URL: http://dataaspirant.com/2017/03/07/difference-between-softmax-function-and-sigmoid-function/.

[2] Samay Shamdasani. *Build a flexible Neural Network with Backpropagation in Python*. Oct. 2017. URL: https://dev.to/shamdasani/build-a-flexible-neural-network-with-backpropagation-in-python.

[3] Sagar Sharma. *Activation Functions: Neural Networks*. Sept. 2017. URL: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[4] *THE MNIST DATABASE of handwritten digits*. Nov. 1998. URL: http://yann.lecun.com/exdb/mnist/.

[5] Avinash Sharma V. *Understanding Activation Functions in Neural Networks*. Mar. 2017. URL: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0.