Neural Network

Daniel Alvarado Bonilla, 2014089192

Resumen—This document describes the implementation of a neural network capable of identifying the letters A, B, D, E, and F in handheld photos of pieces of paper. The network was implemented using the C language, while the preprocessing of the images was done utilizing Python with the *PIL*, *cv2* and *numpy* libraries. Instead of using matrixes as sugest, structures are implemented throught the code.

Palabras Clave—Inteligencia Artificial, Red Neuronal, C, Letras

1 Introducción

Las redes neuronales resurgieron en los años ochenta tras un periodo de inactividad y actualmente, en la segunda década del siglo XXI, se han consolidado como un "próspero campo de investigación" [1, p. 5] [2]. Sin embargo, poco después de su génesis en los cuarentas gracias al trabajo de Warren McCulloch y Walter Pitts, la publicación de un famoso libro sepultó por varios años las redes neuronales y desvió los fondos de investigación hacia otros temas [2].

Ese famoso y casi fulminante libro es *Perceptrons: an introduction to computational geometry* por Marvin Minsky y Seymour Papert, publicado en 1969. En dicho libro, los autores "mostraron las deficiencias del modelo perceptrón" [2, p. 13] al probar matemáticamente que "un único perceptrón [...] era incapaz de aprender la función de 'o' exclusivo (conocida como XOR)" [3].

En su momento, Minsky y Papert criticaron la falta de rigurosidad matemática con la cual se estaban desarrollando las distintas aplicaciones de redes neuronales e insistieron en la necesidad de una base teórica sólida para poder determinar la efectividad de las mismas. Con esto, buscaban evitar que el progreso de esta área de estudio se realizara "solamente con intuición y experimentación" [1, p. 5].

Sin embargo, como se mencionó, las redes neuronales salieron de su tumba en los años ochenta. Este resurgimiento se dio gracias, principalmente, al establecimiento de hitos teóricos como "el descubrimiento de retro-propagación de errores [...] y nuevos avances en hardware que incrementaron las capacidades de procesamiento" [2, p. 13].

Hoy en día, "es innegable la efectividad de las redes neuronales profundas en aplicaciones prácticas como el reconocimiento del habla y la visión artificial" [1, p. 5]. Redes de este tipo se utilizan en asistentes personales en celulares con los cuales se puede conversar, como quien habla con otro ser humano; y en vehículos que se manejan solos.

En este documento, se describe la implementación de una red neuronal pequeña que es capaz de reconocer las letras A, B, C, D, E y F en imágenes que corresponden al solucionario de un examen de selección única. Por esta razón, las letras están encerradas en casillas numeradas según la pregunta a la correspondiente.

2 Marco Teórico

Una red neuronal artificial es un "'modelo computacional' con características particulares como la habilidad de adaptarse o aprender, de generalizar, de agrupar u organizar datos; y cuya operación está basada en el procesamiento paralelo" [2, p. 13]. Así, sus dos grandes puntos fuertes son su estructura paralelizada y su capacidad de aprender y generalizar; es decir, la habilidad de producir "salidas razonables para entradas no presentes durante el entrenamiento (aprendizaje)" [4].

En otras palabras, una "red neuronal es un procesador distribuido masivamente paralelo compuesto de unidades de procesamiento simples que tiene una propensión natural por al-

macenar conocimiento experimental y hacerlo disponible para su uso" [4, p. 2].

Las redes neuronales artificiales son abstracciones matemáticas de las redes neuronales biológicas presentes en el cerebro [4]. Se podría incluso decir que son una *simplificación excesiva* de los modelos biológicos [2], pero lo cierto es que su inspiración es completamente biológica. Como menciona Kriesel [5], "las redes neuronales tecnológicas son tan solo caricaturas de la naturaleza" [p. 13].

Algunas características importantes de las redes neuronales son su capacidad de ser *no-lineales* y de generar mapeos entre entradas y salidas, lo cual se logra gracias a un proceso de aprendizaje supervisado. Debido a lo anterior, las redes son altamente adaptables a nuevos cambios que puedan aparecer en el ambiente de operaciones. Para modificar el mapeo generado bastaría con volver a entrenar la red con un conjunto de datos nuevo [4].

Las unidades básicas de procesamiento de una red neuronal son, como es de esperarse, las neuronas. La primera abstracción de una neurona (es decir, una red neuronal de una capa) fue el *perceptrón* de Rosenblatt. "El perceptrón es la forma más simple de una red neuronal usada para la clasificación de patrones linealmente separables" [4, p. 48].

La neurona (perceptrón) se compone de tres partes principales: las sinapsis o enlaces de conexión, el sumador y la función de activación. Los enlaces de conexión conectan las neuronas de una capa con la otra y poseen un peso o valor asignado, los cuales son modificados a medida que el entrenamiento avanza. El sumador realiza la suma de todas las señales de entrada con sus respectivos pesos de los enlaces de conexión. Por último, la función de activación toma los resultados de cada neuronas obtenidos por el sumador y los transforma a valores dentro de un rango establecido [4].

Algunas funciones de activación utilizadas son la *función del umbral*, la *función sigmoide* [4], el *rectificador* (ReLU), la *función softmax*, entre otras. Cada una provee resultados diferentes y son útiles en distintos escenarios.

Como se mencionó brevemente, una red neuronal puede tener una o varias capas (arquitectura). En general, todas poseen al menos una de entradas (*input layer*) y una de salidas (*output layer*). Si la red sólo posee las dos mencionadas, se dice que es una red de una sola capa. Si, por el contrario, la red tiene más niveles, se dice que es una red neuronal *multicapa* o *profunda*. A estas capas intermedias se les llama *hidden layers*.

Una red neuronal será tan efectiva, primero, como abundantes sean los datos con los cuales es entrenada. Como mencionan Wang & Perez [7], "[e]s conocimiento general que mientras más datos tenga un algoritmo de ML, más efectivo puede ser" [p. 1]. Asimismo, es necesario que estos datos sean de alta calidad y fidedignos a la realidad que representan [8]. Sin embargo, generar u obtener muchos datos distintos puede ser una actividad demandante y difícil.

Para solucionar esto, se utilizó datos dados por distintos estudiantes del grupo de Inteligencia Artificial, obteniendo así 1518 imágenes.

3 IMPLEMENTACIÓN

3.1. Descripción del problema

Los estudiantes del profesor Esteban Arias Méndez del Tecnológico de Costa Rica resuelven los exámenes de distintos cursos utilizando una plantilla de respuestas, la cual posee una cuadrícula numerada que corresponde a cada una de las preguntas de selección única. En esta plantilla, los estudiantes escriben las respuestas utilizando las letras A, B, C, D, E, F; según corresponda.

Esta red, agilizará el proceso de detección de las letras utilizadas en el exámen. Donde podrá identificar dada una foto la letra. En caso de no contener una letra, la red podrá determinar lo mismo. El comportamiento cuando se utiliza una letra que no sea de las anteriores es desconocido.

3.2. Procesamiento de las imágenes

Para todo el pre-procesamiento de las imágenes se utilizó el lenguaje Python.

Para el entrenamiento de la red neuronal que se encarga de identificar las letras, se crearon un total de 1518 imágenes: obtenidas por los datos de diferentes estudiantes que crearon distintas

versiones de las letras mencionadas anteriormente. Cada letra se divide en una carpeta, dicha carpeta contiene un aproximado de 220 letras.

Anterior a este, se intentó utiliza un método diferente. El cual creaba *batches* de 32 imágenes, pero no fue efectivo al entrenarlo con la red.

El proceso empleado en estás imagenes se descompone de la siguiente forma:

Primero, se le modifica al tamaño a la imágen, mediante la función de la librería *PIL*, *resize*. El tamaño de la imágen enseguida es de 28x28.

Segundo, utiliza la biblioteca ***cv2*** para leer las imágenes aumentadas de 28x28 píxeles como una matriz (cada elemento de la matriz representa un píxel con un valor entre 0 y 255).

Tercero, con la biblioteca mencionada anteriormente, se transforma dicha matriz a un arreglo de 784 elemntos, se le aplica una calculo a cada píxel para obtener un valor entre 0 y 1. Este arreglo de elementos entre 0 y 1 es dado a la red para así entrenarla.

Sin embargo, para que dicha red pueda obetener los datos, las imágenes son guardadas en archivos txt. Distribuidos por letras, es decir, todas las imágenes de la letra A son guardadas en un archivo A.txt, lo anterior para facilitar la lectura de los mismos en el lenguage C y la clasificación de las imágenes en las estructuras.

3.3. Arquitectura de las redes neuronales

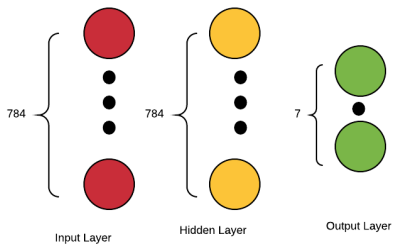

Figura 1. Modelo

La red neuronal que identifica las letras (ver Fig. 1) está compuesta por una capa de entradas (*input layer*) de 784 nodos: uno por cada píxel de las imágenes de tamaño 28x28. Además, cuenta con una *hidden layer* se utilizó una de tamaño 784, por último, una capa de salidas (*output layer*) de 7 nodos: uno por cada letra a identificar y uno para caracteres desconocidos.

Para el backtracking de este modelo de perceptrón, es decir la etapa en la cual se debe de efectuar un pequeño cambio en los pesos para así disminuir el error de la red, fue basado en lo siguiente:

- Se calcula los cambios en los pesos de la capa de salida, dicho cambio llamado delta, utilizando la derivada del error (valor esperoda − valor obtenido). Lo anterior basándose en *Min Square Error*, según [6].
- Para los pesos de la capa oculta ocurre un proceso similar, calculando el error de un nodo oculto con la suma del error de todos sus nodos de salida. En otras palabras sumando todos los delta de los nodos de salida para un nodo oculto.

Para este modelo de perceptrón la función de activación seleccionada fue *Sigmoid*. Posterior a esto mediante la derivada de la misma y el error obtenido se ajustan los pesos, el paso anterior usando *Min Square Error*.

4 Problemas Encontrados

Para construir la red se emplearon distintos métodos, los cuales por alguna razón fueron inefectivos hasta el último . En el Desarrollo se explicará con detalle la estructura utilizada para que dicha red pudiese ser entrenada. Los métodos utilizados previamente se describen a continuación. El primer problema encontrado fue al leer las imágenes desde el programa, para poder alimentar la red. Previamente se guardaban las imágenes en batches de un tamaño de 32, cada batch en un archivo txt, donde cada imágen tenía 784 píxeles. Es decir, en cada archivo se encontraban (784*32) píxeles.

Otra parte del problema fue al intentar hacer uso de recomendado en la realización de redes neuronales, matrices. Primero, una matriz de 32x784, la cual sería multiplicada con una de 784x784, por último la restante de 32x784 se multiplicaría con una de 784x7. Así dando el resultado de 32x7, clasificando cada una de las 32 imágenes.

Al mezclar estos enfoques, el resultado no fue el esperado. La red no lograba aprender.

5 DESARROLLO

Para obtener los resultados deseados, se tuvo que cambiar por completo el foco del problema. El previo, mostraba como la red no aprendía. Después de un aproximado de 1000 *Epochs*, el modelo no aprendía. Ya que de 1518 imágenes probadas, se obtenía un resultado de 2 correctas y 1516 incorrectas. Al ser probada con una letra, la salida de la red era de la siguiente forma: 1111110. Comportamiento totalmente erroneo.

Para solucionarlo, se implementó lo siguiente, divido en dos secciones:

5.1. Código

Se implementaron 4 estructuras en el lenguaje de C, claves.

- Image: para poder representar las imágenes de la mejor form, se implementó una estructura con los atributos. Pixles el cual contiene los 784 pixeles de dicha imagen, con un valor entre 0 y 1. Letter para almacenar el valor de la letra, por ejemplo "A". Y un arreglo para el output deseado. g
- Output Neuron: Esta neurona está compuesta por un array (tamaño de 784) que almacena todos los pesos conectados a ella, su valor de salida y muy importante el bias. No usado usualmente pero empleado en redes que utilizan *Sigmoid* como función de activación.
- Hidden Neuron: Neurona de la capa oculta, contiene al igual que la anterior un array (tamaño de 784) almacenando todos los pesos conectados a esta, su valor y su bias.
- Input Neuron: Una neurona simple para almacenar el valor del píxel de entrada.

Para poder guardar los pesos, se guardaron en archivos de .dat.

El método utilizado para leer las imágenes de los archivos txt fue el siguiente;

```
Images_Array* get_images(char* path, int number_of_images,
     Images_Array* images, char letter) {

    FILE *file = NULL;
    file = fopen(path, "r");

    if (file == NULL) {
        printf("Error opening file\n");
        return NULL;
    }
    int counter, max, pixel_counter, img_counter;
    pixel_counter = 0;
    counter = 0;
    max = number_of_images * 784;
    double* all_images_in_file = malloc(sizeof(double) * max);
    counter = 0;
    while (!feof(file)) {
        fscanf(file, "%lf", &all_images_in_file[counter]);
        counter++;

    }
    fclose(file);
    Image* new_img;
    counter = 0;
    img_counter = 0;
    while (img_counter < number_of_images) {
        pixel_counter = 0;
        //printf("Creating new image\n");
        new_img = malloc(sizeof(Image));
        new_img->letter = letter;
        set_img_output(letter, new_img->expected_output);
        new_img->next = NULL;
        while (pixel_counter < 784) {
            new_img->pixels[pixel_counter] = all_images_in_file[
                counter];
            counter++;
            pixel_counter++;
        }
        img_counter++;
        add_to_imgs_array(images, new_img);
    }
    free(all_images_in_file);
    return images;
}
```

Este lee de un arhivo A.txt, el cual contiene una lista de todas las imágenes utilizadas para el entrenamiento representando la letra "A". Primero almacena de forma dinámica una lista (all_images_in_file) de tamaño 784 x 1518 mediante el uso de *fscanf()* se logra. Luego, se recorren dos ciclos claves para poder efectuar la carga de las imagenes. El ciclo exterior, recorre cada una de las 1518 imágenes, contenidas en un arreglo en *Heap Memory*. En este mismo, cada vez se crea una nueva estructura de tipo *Image*. Adentro de este ciclo, existe otro donde se ejecuta el contador de píxeles, para así almacenarlos todos en la nueva imagen.

El método principal de la red, fue derivado del siguiente[9]:

```
        for (int j=0; j<numHiddenNodes; j++) {
            double activation=hiddenLayerBias[j];
             for (int k=0; k<numInputs; k++) {
                activation+= ( PIXEL_SCALE(*(training_inputs +i*
                     numInputs + k)) * hiddenWeights[k *
                     numHiddenNodes + j]);
            }
            hiddenLayer[j] = sigmoid(activation);
        }
        for (int j=0; j<num_output_nodes; j++) {
            double activation=outputLayerBias[j];
            for (int k=0; k<numHiddenNodes; k++) {
                activation += hiddenLayer[k] * outputWeights[k *
                     num_output_nodes + j];
            }
            outputLayer[j] = sigmoid(activation);
        }

        // Backpropagation

        double deltaOutput[num_output_nodes];
        for (int j=0; j<num_output_nodes; j++) {
            double errorOutput = (*(training_outputs +i*
                 num_output_nodes + j) -outputLayer[j]);
            deltaOutput[j] = errorOutput*dSigmoid(outputLayer[j]);
        }

        double deltaHidden[numHiddenNodes];
        for (int j=0; j<numHiddenNodes; j++) {
            double errorHidden = 0.0f;
            for(int k=0; k<num_output_nodes; k++) {
                errorHidden += deltaOutput[k] * outputWeights[j *
                     num_output_nodes +k];
```

```
            }
            deltaHidden[j] = errorHidden*dSigmoid(hiddenLayer[j]);
        }

        for (int j=0; j<num_output_nodes; j++) {
            outputLayerBias[j] += deltaOutput[j]*lr;
            for (int k=0; k<numHiddenNodes; k++) {
                outputWeights[k * num_output_nodes + j]+=
                      hiddenLayer[k] * deltaOutput[j] * lr;
            }
        }

        for (int j=0; j<numHiddenNodes; j++) {
            hiddenLayerBias[j] += deltaHidden[j]*lr;
            for(int k=0; k<numInputs; k++) {
                hiddenWeights[k *numHiddenNodes + j] += *(
                      training_inputs +i*numInputs + k) *
                      deltaHidden[j]*lr;
            }
        }
    }
```

En el anterior, se puede ver como primero se calcula el valor de cada neurona oculta mediante los pesos y la entrada (píxel de entrada). Luego, con esta salida y los pesos de la capa de salida, se calcula el valor de cada neurona de salida. Es decir, se obtiene 7 valores, representando cada letra.

Luego, se calcula el error, posterior el delta. Y por último haciendo uso del delta se cambian los valores de los pesos. En la red utilizada sin embargo, es un tanto distnto, puesto que no se usan matrices sino que estructuras para representar las neuronas de salida y las neuronas de la capa oculta.

6 ANÁLISIS DE RESULTADOS

Para los resultados se realizaron entrenamientos con un mismo modelo, obteniendo la mejor exactitud con 784 nodos en la capa oculta. Se utilizan un total de 1518 imágenes El valor *Learning rate* fue moderado mediante pruebas, también especificadas a continuación. Para las 1518 imágenes se efectuaron 1000 epocs para el entrenamiento de dicha red. // Resultados obtenidos con 784 nodos en la capa oculta, *Learning Rate* de 0.1: Resultados obtenidos con 784 nodos en la capa oculta además de mezclar el arreglo de imágenes en cada epoch: Un ejemplo seleccionado para probar la red, fue utilizando una imágen un tanto díficil, donde la red fue exitosa.

7 CONCLUSIONES

Luego investigar acerca de los diferentes perceptrones a modelar para crear nuestra Red Neuronal, se concluye que para nuestro problema la mejor opción fue aquella de tres diferentes capas (no se realizaron pruebas con más), conformado por una capa de entrada, una capa oculta y por último la capa de salida. Las distinas pruebas tenían una cantidad e 32, 64

```
========== RECONOCER LETRAS ==========


Entrenar red? [1 o 0]
1
===== ENTRENANDO =====
Creating new weights...
Epochs: 1000
Epoch: 0 Accuracy: 0.142951
Epoch: 100 Accuracy: 0.961792
Epoch: 200 Accuracy: 0.969038
Epoch: 300 Accuracy: 0.978261
Epoch: 400 Accuracy: 0.987484
Epoch: 500 Accuracy: 0.982872
Epoch: 600 Accuracy: 0.992754
Epoch: 700 Accuracy: 0.990777
Epoch: 800 Accuracy: 0.992754
Epoch: 900 Accuracy: 0.993412
Correct: 1508
Incorrect: 10
Total: 1518
Accuracy: 0.993412
============== DONE ==================
```

Figura 2. Resultados - 1000 epochs

```
========== RECONOCER LETRAS ==========


Entrenar red? [1 o 0]
1
===== ENTRENANDO =====
Creating new weights...
Epochs: 1000
Epoch: 0 Accuracy: 0.142951
Epoch: 100 Accuracy: 0.944005
Epoch: 200 Accuracy: 0.991436
Epoch: 300 Accuracy: 0.992095
Epoch: 400 Accuracy: 0.989460
Epoch: 500 Accuracy: 0.993412
Epoch: 600 Accuracy: 0.990119
Epoch: 700 Accuracy: 0.990119
Epoch: 800 Accuracy: 0.993412
Epoch: 900 Accuracy: 0.988142
Correct: 1507
Incorrect: 11
Total: 1518
Accuracy: 0.992754
```

Figura 3. Resultados - 1000 epochs - 2

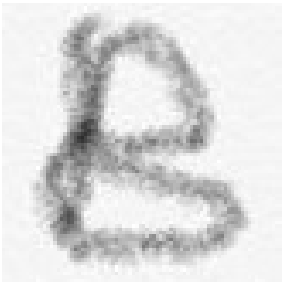

Figura 4. B

```
========== RECONOCER LETRAS ==========


Entrenar red? [1 o 0]
0

===== PROBANDO =====
Network output:
A: 0.001326
B: 1.000000
C: 0.000000
D: 0.000004
E: 0.000001
F: 0.000000
_: 0.000000
Letter is a: B
```

Figura 5. Resultado obtenido por la Red

Figura 6. E

```
========== RECONOCER LETRAS ==========


Entrenar red? [1 o 0]
0

===== PROBANDO =====
Network output:
A: 0.000000
B: 0.000000
C: 0.000038
D: 0.000000
E: 0.906714
F: 0.000000
_: 0.000000
Letter is a: E
```

Figura 7. Resultado obtenido por la Red

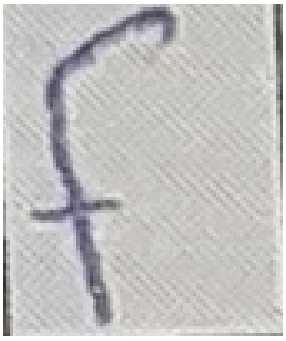

Figura 8. F

```
========== RECONOCER LETRAS ==========


Entrenar red? [1 o 0]
0

===== PROBANDO =====
Network output:
A: 0.000001
B: 0.003120
C: 0.000013
D: 0.000000
E: 0.026959
F: 0.927726
_: 0.000056
Letter is a: F
```

Figura 9. Resultado obtenido por la Red

y 784 nodos/neuronas en la capa oculta. Los resultados más rápidos obtenidos y exactos, fueron aquellos utilizando 784 neuronas en la capa oculta. Posterior realizando una prueba después del último epoch, resultados fueron excelentes. Con un total de 1518 imágenes, ya la exactitud de la red era de 0.992754, con 10 fallós en total. Esta red siendo simple, logra reconocer letras. Pero pueden utilizarse en muchísimos ámbitos, siendo realmente útiles en un futuro cercano.

REFERENCIAS

[1] Bottou, L. (2017). Perceptrons An Introduction to Computational Geometry Reissue of the 1988 Expanded Edition with a new foreword by Léon Bottou. Recuperado de: https://leon.bottou.org/publications/pdf/perceptrons-2017.pdf.

[2] Kröse, B. & van der Smagt, P. (1996). An Introduction to Neural Networks. Recuperado de: https://www.infor.uva.es/~teodoro/neuro-intro.pdf.

[3] Ballantyne, A. (2017). Minsky's "And / Or" Theorem: A Single Perceptron's Limitations. Recuperado de: https://alan.do/minskys-and-or-theorem-a-single-perceptron-s-limitations-490c63a02e9f.

[4] Haykin, S. (2009). Neural Networks and Learning Machines. Recuperado de: https://dai.fmph.uniba.sk/courses/NN/haykin.neural-networks.3ed.2009.pdf.

[5] Kriesel, D. (2007). A Brief Introduction to Neural Networks. Recuperado de: http://www.dkriesel.com/_media/science/neuronalenetze-en-zeta2-2col-dkrieselcom.pdf.

[6] Kim, S. (2019). Back-propagation Demystified in 7 minutes. Recuperado de: https://towardsdatascience.com/back-propagation-demystified-in-7-minutes-4294d71a04d7

[7] Wang, J. & Perez, L. (s. f.). The Effectiveness of Data Augmentation in Image Classification using Deep Learning. Recuperado de: http://cs231n.stanford.edu/reports/2017/pdfs/300.pdf.

[8] Tanaka, F. & Aranha, C. (2019). Data Augmentation Using GANs. *Proceedings of Machine Learning Research XXX, pp. 1-16* Recuperado de: https://arxiv.org/pdf/1904.09135.pdf.

[9] Becerra, S. (2019). Simple neural network implementation in C. Recuperado de: https://towardsdatascience.com/simple-neural-network-implementation-in-c-663f51447547.

APÉNDICE

A. Código en Python para el procesamiento de las imágenes