Tarea 13 - Implementación de Red Neuronal Multicapa usando Paradigma Orientado a Objetos

Oscar Esaú Peralta Rosales

Objetivo

- Programar Arquitectura de *RNM* genérica.
- Utilizar la RNM para clasificar los datos de insumo (*data.dat*).
- Entregar como salida:
    - La arquitectura de red empleada, además de los pesos finales.
    - Error de entrenamiento obtenido.

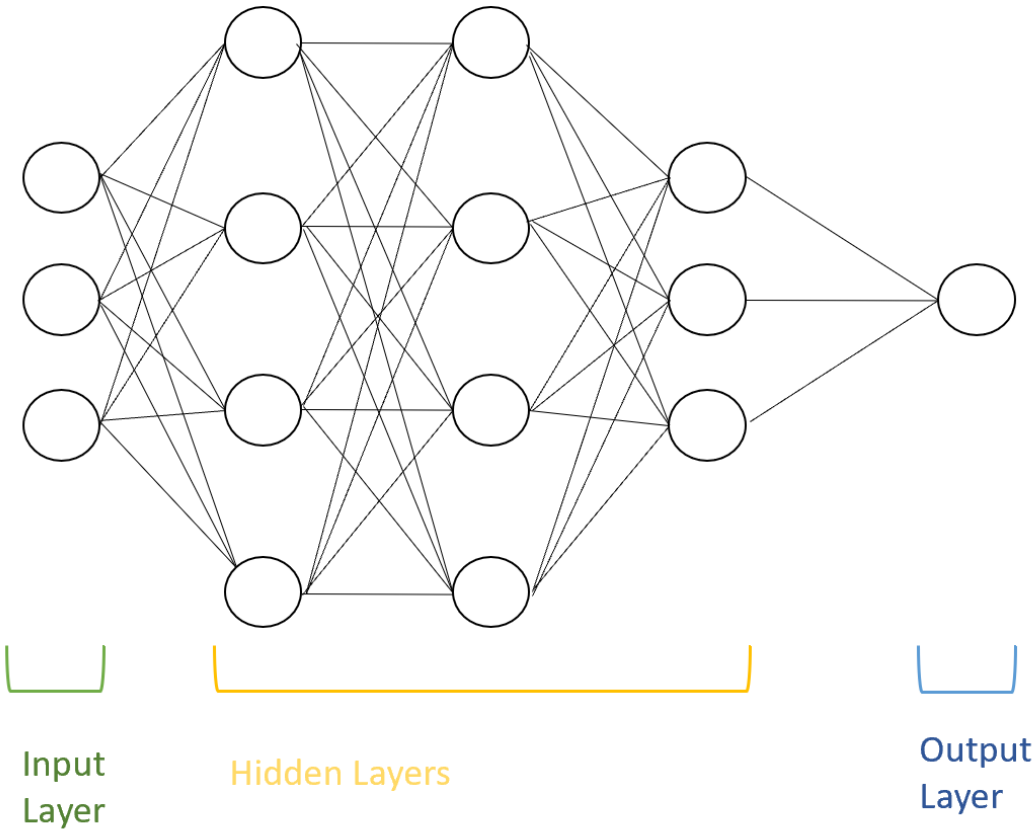
Redes Neuronales

Las Redes Neuronales son una abstracción del comportamiento y modelación de las redes neuronales biológicas. Consisten en un conjunto de *neuronas artificiales* conectadas entre si a través de enlaces para transmitir y procesar información.

Durante la transmisión de la información (Fordward propagation), cada enlace de las neuronas están conformados por una series de pesos que determinan el potencial de alguna neurona en la red, además, es combinado con una operación que limita sobre un umbral dicha importancia, está es la llamada función de activación.

El aprendizaje de las Redes Neuronales se realiza a mediante una función de costo que evalúa su potencial y la ayuda a su vez a aprender del los errores mediante el proceso de propagación hacia atrás (Back propagation).

Una red está puede estar formada por múltiples capas, donde tiene 1 o más neuronas asociadas.



Perceptrón

El Perceptrón es un tipo tipo de neurona artificial comúnmente usada para clasificar datos en dos clases. También puede ser usado para realizar una clasificación multiclase al representar cada característica esperada de la salida en una decisión binaria (contiene o no dicha característica).

El Perceptrón consta de 4 partes:

- Entrada de datos: $X = \{x_i\}$
- Pesos y bias: $W = \{w_i\}$
- Suma neta: $\sum_{i=1}^{k} w_i x_i + b$
- Función de activación: Función sigmoide, RELU, etc.

La entrada de datos a la neurona es usada para calcular el valor de la suma neta para posteriormente aplicarle la función de activación. El valor del bias nos ayuda a variar la curva de la función de activación hacia arriba o hacia abajo. La función de activación nos ayuda a mapear los datos a un nuevo rango necesitado como $(0, 1)$ o $(-1, 1)$.

Forward propagation

Sea a^L un vector con los resultados de la aplicación de la función de activación σ a la suma acumulada de cada neurona de la capa L.

Para cada capa podemos aplicar

$$z^L = W^L a^{L-1} + b^L \tag{1}$$

$$a^L = \sigma(z^L) \tag{2}$$

con W^L como la matriz de pesos de la capa L y $a^0 = X$,

Back propagation

Dada una función de Coste C podemos calcular el error obtenido contra los valores esperados $Y = \{y_i\}$ y repartir el error a las capas anteriores para realizar la corrección de pesos correspondiente usando descenso de gradiente.

Así para calcular error y el reparto de errores y corregir los pesos para al última capa está dado por:

$$\delta^L = \frac{\partial C}{\partial a^L} \frac{\partial a^L}{\partial z^L} \tag{3}$$

$$\delta^{L-1} = W^L \delta^L \frac{\partial a^{L-1}}{\partial z^{L-1}} \tag{4}$$

$$\frac{\partial C}{\partial b^{l-1}} = \delta^{l-1} \tag{5}$$

$$\frac{\partial C}{\partial w^{L-1}} = \delta^{L-1} a^{L-2} \tag{6}$$

Y el aprendizaje mediante descenso de gradiente está determinado por

$$W^L = W^L - z^{L-1} * \delta^L * \alpha \tag{7}$$

con α como el coeficiente de aprendizaje, y $z^0 = X$

Implementación y Resultados

Se implementaron 5 clases distintas:

- **Clase *Activation***: Clase abstracta que contiene las definiciones para la evaluación de las funciones de activación.
- **Clase *F_Cost***: Clase abstracta que contiene las definiciones para la evaluación de las funciones de costo.
- **Clase *Perceptron***: Abstracción para la entidad Perceptrón
- **Clase *Layer***: Abstracción para para la entidad capa, está compuesta por varios objetos tipo *Perceptron*.
- **Clase *MLP***: Abstracción para la entidad de la red de Perceptrones Multicapa, esta compuesta por varios objetos tipo *Layer*.

Cómo función de activación se implementó la función sigmoide dada por

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{8}$$

y cuya derivada es

$$sigmoid'(x) = sigmoid(x) * (1 - sigmoid(x)) \tag{9}$$

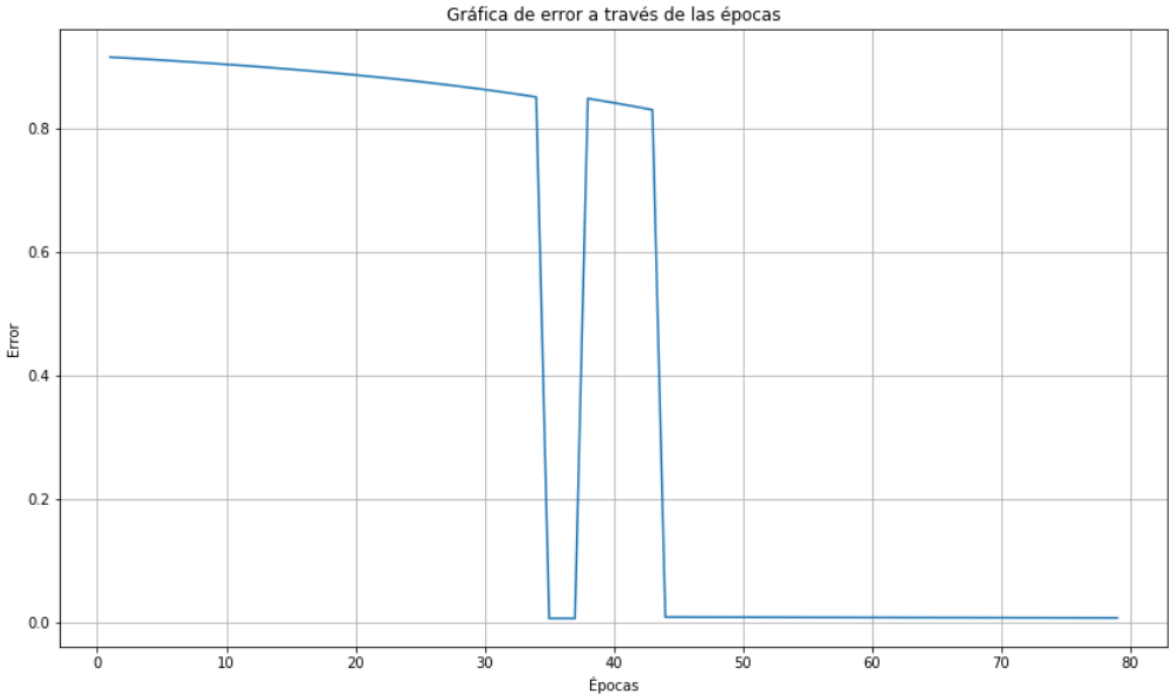
Cómo función de costo se usó la el Error Cuadrático Medio

$$ECM = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \tag{10}$$

Se realizaron pruebas con las siguientes arquitecturas de la red

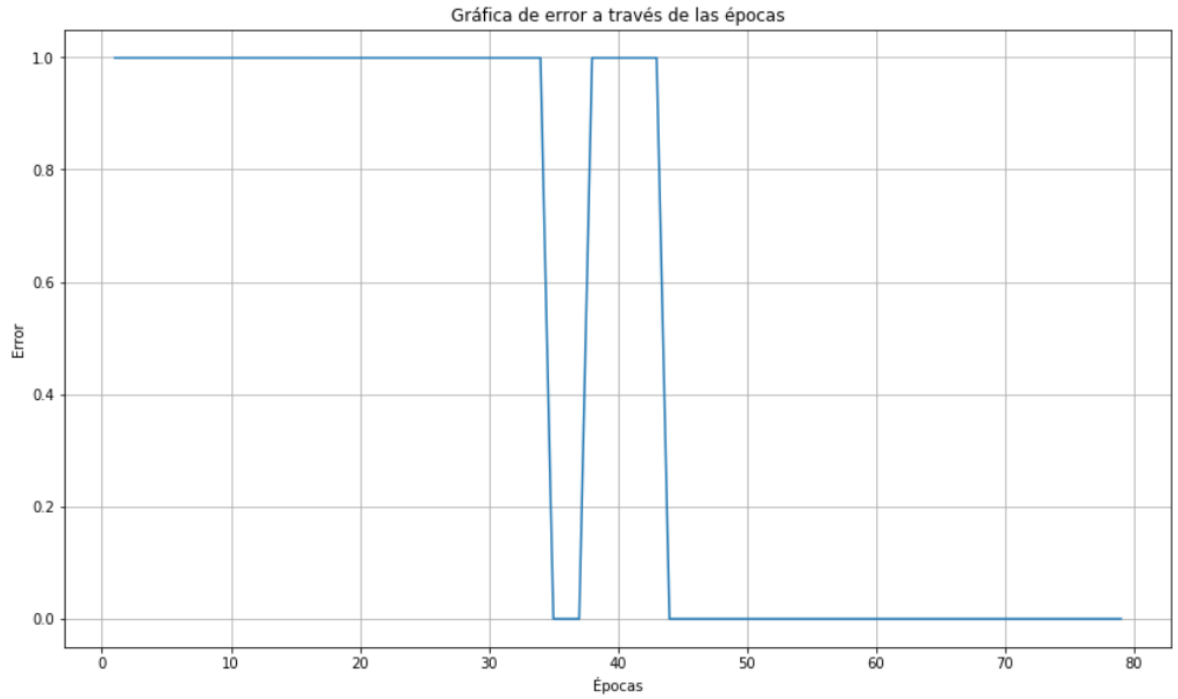
Caso 1

```
Network architecture:
Activation function: Sigmoid
Cost function: ECM
Learning rate: 0.1
Input size: 4
Layers: 5
Layer 1, No. Neurons: 8
Layer 2, No. Neurons: 16
Layer 3, No. Neurons: 8
Layer 4, No. Neurons: 3
Layer 5, No. Neurons: 1

Last Error: 0.00697571
```



Caso 2

```
Network architecture:
Activation function: Sigmoid
Cost function: ECM
Learning rate: 0.1
Input size: 4
Layers: 3
Layer 1, No. Neurons: 8
Layer 2, No. Neurons: 16
Layer 3, No. Neurons: 1

Last Error: 3.32861e-07
```



La validación se realizo con 20 datos de los 100 disponibles, los 80 datos restantes fueron ocupados para el entrenamiento,

En ambos casos se alcanzó un acertividad del 55%.

Conclusiones

La implementación de la red neuronal usando la Metodología Orientada a Objetos ayuda a estructurar y reutilizar mucho código, además de que la lectura de este se vuelve más simple y fácil de seguir.

Además se identificó que para estos datos, mientras más capas ocultas son agregadas el error de la función de coste no decrece tanto. Los mejores valores se obtuvieron jugando con una configuración que conste de una sola capa oculta.

Por otro durante las pruebas la funcionalidad del algoritmo no fue como se esperaba, puesto que estas mismas me indicaban que gran parte de los datos eran mapeados en su mayoría a 1 a pesar que se alcanzo un error durante el proceso de backprogation de 3.32861e-07. Por tanto se intentó identificar algún error sin lograr encontrar una causa a cuál atribuirle este comportamiento.