Reporte tarea 10 - Programación y algoritmos

Erick Salvador Alvarez Valencia, Centro de Investigación en Matemáticas, A.C.

***Index Terms*—Redes neuronales, aprendizaje máquina.**

I. Introducción

En el presente reporte se analizará la implementación de una red neuronal perceptrón multicapa usando programación orientada a objetos, la cual solo se implementó la funcionalidad *Fordward propagation* la cual consiste en iterar sobre las capas realizando la combinación lineal de las entradas y los pesos y posteriormente aplicando la función sigmoide. De la misma forma se mostrarán los resultados de la arquitectura de la misma usando la librería Cairo.

mds
Noviembre 6, 2017

II. Perceptrón multicapa

II-A. Descripción

Este modelo de red neuronal extiende el concepto del perceptrón simple, el cual consiste en una simple capa que procesa los datos usando la combinación lineal de los pesos aleatorios y las anteradas de la capa anterior $\sum_{i=1}^{n} x_i w_i + b_i$ donde b_i representa un valor de BIAS el cual sirve para la traslación del hiperplano generado.
Para la red multicapa se generan más de una capas ocultas las cuales realizan el mismo procedimiento que en el perceptrón simple, una combinación lineal de las salidas de la capa anterior y los pesos que conectan a las capas, se suele usar una red completamente conexa en sus capas pero se puede elegir el no hacer ciertas conexiones, para este programa se usará la red completamente conexa.
Para construir la estructura de la red se utilizó el paradigama POO en C++. A continuación se describirá la estructura generada:

- **Clase Neurona:** Esta es una clase con un método estático, el cual provee la función de activación para la salida de la neurona.
- **Clase Capa o Layer:** Esta clase contiene diversos atributos y métodos para el procesamiento de la red. Se destacan tres vectores, la entrada, la salida y el error generado por la capa, así como el vector de BIAS y sus errores. Se proveen dos métodos los cuales procesan las entradas de la capa anterior contenidas en el vector de salida de dicha capa y su procesamiento se guarda en el vector de entrada de la capa actual, y la función que procesa la capa de entrada de la capa actual aplicando la función de activación a sus elementos y generando el vector de salida.
- **Clase Red Neuronal o Neural Network:** Esta clase concentra el procesamiento principal de la red, contiene un vector de capas, en donde se incluyen las de entrada, las ocultas y las de salida. Esta clase se puede extender para generar la función de aprendizaje o *Back Propagation*.

A continuación se definirán el algoritmo de *Fordward Propagation*:

Algorithm 1 Procesamiento en una RNM.

```
1:  procedure FordWardProp(X)
2:      inputSz ← Número de neuronas en la capa 1.
3:      if X.size() = inputSz then
4:          for i ← 0 to inputSz do
5:              layers[0].setOutput(i, X[i]).
6:          noLayers ← Número de capas de la red.
7:          for i ← 1 to noLayers - 1 do
8:              computeLayerInput(i).
9:              computeLayerOutput(i).
10:         return layers[noLayers - 1].getOutput().
11:     else
12:         return X
```

En el Algoritmo 1. podemos ver que se procesa la primer capa por separado, esto debido a que la salida se iguala a la entrada de la misma ya que esta no es una capa de procesamiento, para las siguientes capas (incluida la salida) si se realiza el procesamiento habitual, en donde se utilizan dos funciones principales *computeLayerInput* y *computeLayerOutput*, las cuales se describen a continuación:

Algorithm 2 Procesamiento de las entradas de la capa oculta.

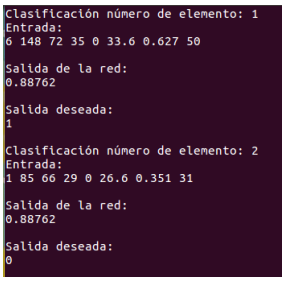
```
1:  procedure computeLayerInput(layer)
2:      w ← La matriz de pesos de la capa l - 1 a l.
3:      for i ← 0 to layers[l].size do
4:          val ← 0.
5:          for j ← 0 to layers[l - 1].size do
6:              val ← val + layers[l - 1].output[j] * w[j][i].
7:          val ← val + layers[l - 1].bias[i].
8:          layers[l].input[i] ← val.
```

Hay que destacar que se usa una matriz de pesos y un arreglo de BIAS por capa.
La función *computeLayerOutput* sólamente aplica la función sigmoide a los valores calculados anteriormente y los almacena en el vector de salida.

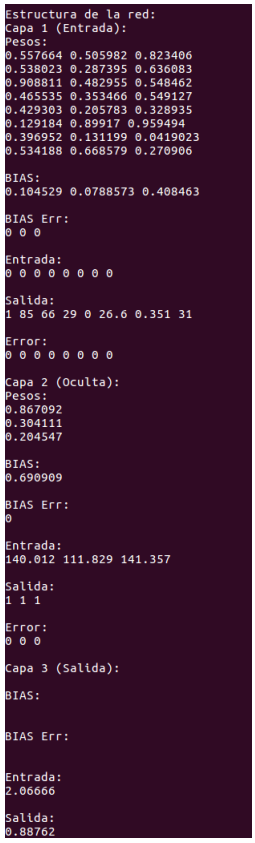
III. Ejemplo de ejecución

Para las ejecuciones del algoritmo se realizaron con el conjunto de datos de entrada brindados en la descripción de la

tarea, se hizo una función de lectura que generaba objetos de una estructura de entrenamiento la cual contiene dos vectores, el de entrada y el de salida deseada. Posteriormente se aplicó el método de Fordward Propagation en ciertos vectores del mismo conjunto. A continuación se muestra un ejemplo de salida para dichos conjuntos.



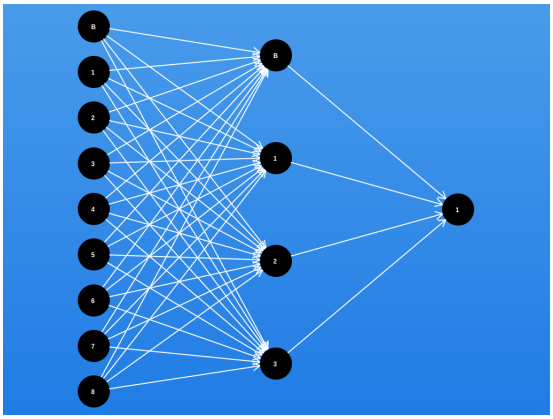
(a) Figura 1. Salida de la red para un conjunto de 2 entradas.



(b) Figura 2. Estructura de la red anterior.

Para las Figuras 1 y 2. Se utilizaron los 2 primeros vectores del conjunto de datos provisto, podemos analizar cómo quedó la red tras el procesamiento de dichos vectores (No se realizó ningún método de aprendizaje).

De la misma forma se utilizó la librería de Cairo Graphics para generar la imagen de la arquitectura de la red neural, en el ejemplo pasado se generó una red con una capa oculta. A continuación se muestra la imagen de salida:



(c) Figura 3. Arquitectura de la red anterior.

IV. Conclusiones

En la presente tarea se construyó el modelo básico de una red neuronal perceptrón multicapa, cabe destacar que como extra se intentó la programación del algoritmo de entrenamiento por descenso de gradiente estocástico, y pese no se obtuvieron los resultados deseados, se conserva el código en los archivos. También se pudo haber implementado una funcionalidad para indicar la manera de conexión de las capas ocultas, pero eso se puede implementar en trabajos futuros.