Red Neuronal Básica, Dos Perceptrones

Mæstría en Ciencias e Ingeniería de Datos

1st Flores Rodriguez, Julio Cesar
Universidad Autónoma de Tamaulipas
Facultad de Ingeniería y Ciencias
Cd. Victoria, México
a2183018003@alumnos.uat.edu.mx

***Resumen*—El siguiente trabajo trata sobre la construcción de una red neuronal muy básica que consta de dos perceptrones, de clasificación binaria. Se exponen los elementos que componen el perceptrón, los parámetros de entrada, los parámetros de salida y comportamiento esperado.**

***Palabras Clave*—Red neuronal, perceptrones, clasificación binaria, componentes, parámetros de entrada y salida**

I. Introducción

I-A. El perceptrón

El perceptrón es la unidad básica que compone a una red neuronal, ésta neurona groso modo está compuesta como un modelo matemático, el cual para calcular su valor es necesario contar con un conjunto de parámetros y funciones, la figura 1 representa la estructura gráfica de un perceptrón.
Más sobre el perceptrón en [1].

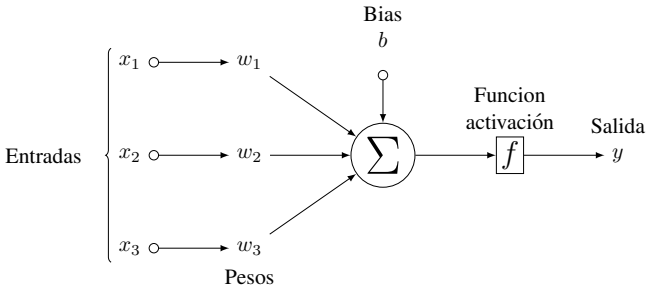

Figura 1. Modelo de un perceptrón

Las funciones de un perceptrón simple son únicamente una función de sumatoria 1 y una función de activación 2. La primera función se mantiene en todos los nodos, la segunda es variable dependiendo del tipo de problema a atacar, pero para un perceptrón básico por lo general se utiliza la función de escalón o escalonada.
En la sección II, se habla más sobre las funciones utilizadas y otros cálculos necesarios para entrenar una red neuronal muy simple.

Dentro de sus parámetros tenemos un vector X de tamaño n, otro vector de tamaño n de pesos W y un valor $Bias$, dependiendo del tipo de problema a estudiar, los parámetros cambian de valor y son ajustados a dicho estudio.

En la sección III se habla más sobre los parámetros utilizados en éste trabajo.

I-B. Red neuronal

Las redes neuronales artificiales (RNA) son utilizadas en el desarrollo de inteligencia artificial, una red neuronal es un modelo computacional diseñado para modelar la forma en que el cerebro actúa sobre una tarea en particular o una función de interés.
Las RNA son redes interconectadas masivamente en paralelo de elementos simples, (nodos o perceptrones usualmente auto-adaptativos) y con organización jerárquica en capas, las cuales interactúan con los objetos del mundo real, del mismo modo que lo hace un sistema nervioso biológico.
La información bruta se ingresa en la primera capa de neuronas, y cada capa sucesiva recibe la salida de la capa anterior.

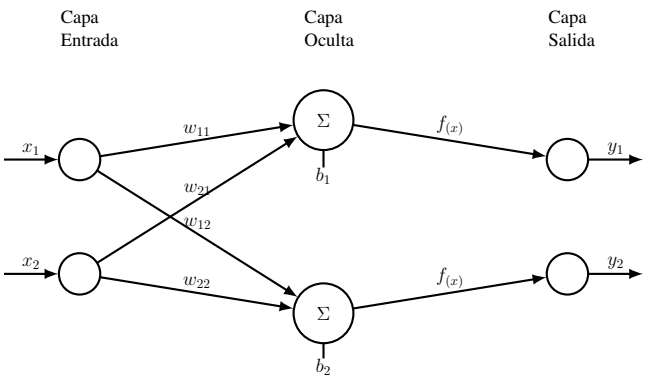

Figura 2. Modelo de red neuronal simple

En este trabajo se utiliza el modelo de red neuronal expuesto en la figura 2.

II. Planteamiento matemático

II-A. Funciones

II-A1. Producto escalar: La función de combinación o producto escalar toma dos vectores de entrada x y w para producir un valor de combinación, o entrada neta, pe. En el perceptrón, la combinación se calcula considerando el sesgo ($bias = \theta$) más la combinación lineal de los pesos sinápticos

y las entradas, es decir la sumatoria de los pesos sinápticos w por las entradas x, 1.

$$pe = \sum_{i=1}^{n} w_i \cdot x_i + \theta \tag{1}$$

Para el presente trabajo se consideran los parámetros siguientes: $i = 1, \ldots, n$, $n = 2$ y $bias = 0.1$

II-A2. Función de activación: Como función de activación se utilizo la escalonada 2, dado que solo se clasifican dos salidas no es necesario tomar otra que pueda representar valores continuos infinitos.

La función escalón o de paso binario esta basada en un umbral μ, si el valor de entrada está por encima o debajo de éste, la neurona se activa o no, y envía la señal activo o inactivo, a la siguiente capa.

$$f(x) = \begin{cases} 1 & \text{si } x > \mu \\ 0 & \text{si } x \leq \mu \end{cases} \tag{2}$$

Para este trabajo se utiliza $\mu = 0.5$ entonces se dice que la función utiliza el rango $(0, 1)$, por lo que si el resultado de 1 esta por encima del umbral μ, la señal será 1 en caso contrario 0.

Una desventaja es que no permite salidas de valores múltiples, no soporta clasificar las entradas, en una, de varias categorías, como si podría hacerlo la función sigmoide.

II-B. Cálculos y ajustes

II-B1. Pesos: El para el calculo del nuevo peso es necesario tomar en cuenta cuatro parámetros, una entrada x_{ij}, el peso w_{ij}, el error E que se puede ver en la ecuación 7 y un valor fijo alfa, α que típicamente toma el valor de: 0.2, $\alpha = 0.2$ mismo valor utilizado en este trabajo. Otra forma de llamar a alfa es: "factor de aprendizaje".

$$w'_{ij} = w_{ij} + (\alpha \cdot E \cdot x_{ij}) \tag{3}$$

$$\Delta w_{ij} = \alpha \cdot E \cdot x_{ij} \tag{4}$$

$$w'_{ij} = w_{ij} + \Delta w_{ij} \tag{5}$$

Para calcular el nuevo peso se considera la formula 3, de la cual los parámetros dentro del paracentesis se pueden ver como: Δw_{ij} de la formula 4 teniendo consecuente la formula 5.

II-B2. Bias y error: Dada cada "EPOCA" permutada, es necesario hacer un ajuste de parámetros para entrenar los pesos de la red neuronal y esta pueda clasificar cada vez mejor en menor numero de "EPOCAS"

El ajuste del bias se calcula como muestra la formula 6.

$$\theta' = \theta + S_{ij} \tag{6}$$

Donde θ' es el nuevo bias y θ bias actual y S es la salida espera, que se busca obtener.

El error E, es calculado utilizando la formula 7.

$$E = S_{ij} - y_{ij} \tag{7}$$

Donde S representa nuevamente la salida esperada, y y_i representa la salida obtenida de esa "EPOCA".

III. Parámetros y ejemplo de ejecución

Como parámetros de entrada de la red neuronal tenemos los vectores X, W, el $bias$ y las salidas esperadas S. Estos parámetros son fijos, el código accesible desde [2], los mantiene comentados para funcionar con valores aleatorios.

Cuadro I
VECTOR DE PESOS

Peso	Valor
w_{11}	3.80302985
w_{12}	-2.89200548
w_{21}	-6.59526659
w_{22}	0.74931817

Cuadro II
VALORES DE BIAS

Bias	Valor
θ_1	0.1
θ_2	0.1

Cuadro III
CONJUNTO DE DATOS, VECTORES DE ENTRADA Y SALIDAS ESPERADAS

X_1	X_2	S_1	S_2
0.95662936	0.34734398	0	1
0.87494675	0.73313805	1	0
0.83615427	0.21178763	1	0
0.83896432	0.77615795	0	1
0.55903885	0.14243021	0	1
0.90487343	0.25602497	1	0
0.37507353	0.9620205	0	1
0.16713584	0.0997963	0	1
0.06229402	0.14997918	0	1
0.19442781	0.7599588	0	1
0.24181593	0.95950415	0	1
0.08634414	0.17435251	1	0
0.44564855	0.16966406	1	0
0.0043262	0.64745794	1	0
0.68332039	0.46816593	0	1
0.53396922	0.32233097	0	1
0.253226	0.32531514	0	1
0.11206421	0.85853743	0	1
0.3899378	0.41040715	0	1
0.41176232	0.34208895	0	1

La gráfica 3, corresponde al conjunto de datos III, con un número fijo de $INSTANCIAS = 20$

Ejecutando el código fijo, podemos obtener los siguientes resultados, para las primeras instancias el entrenamiento es más tardado 4, que para las ultimas 5, y con 20 ejecuciones y un mínimo de 15 épocas (si se trabaja con pesos entre $-10, 10$), se podrá converger más rápido. Para las ultimas instancias el ajuste es mas rápido y depende más del bias.

A partir de aproximadamente las 20 instancias los errores convergen a cero más rápidamente con una época, solo ajustando el bias 6.

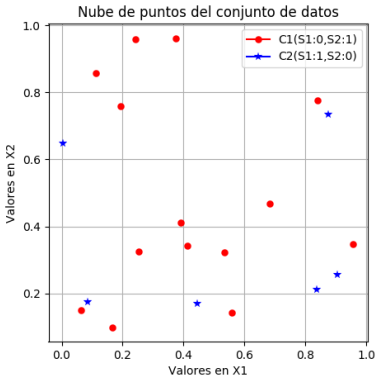

Figura 3. Gráfica correspondiente al cuadro III.

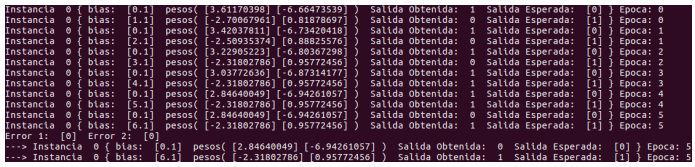

Figura 4. Primera instancia fija de entrenamiento lento.

IV. Implementación

IV-A. Código

El código fuente esta escrito en Python 3.8, al final de este documento se puede consultar un ejemplo simplificado y funcional, del código escrito para este proyecto.
Se puede acceder al código completo desde el enlace [2], incluye comentarios y el conjunto de datos de prueba (comentado) utilizado en este trabajo.

IV-B. Explicación

Las ultimas dos lineas inicializan el programa, con la ejecución de la función main.
Dentro de la función main se declaran: el número de instancias que se van a generar $INSTANCIAS = 20$, las épocas que han de pasar para que una instancia pueda ajustar sus pesos y encontrar un error mínimo, $EPOCAS = 100$, y la función de activación 'E' indicando que se utilizará la función de activación escalonada, el contador nos ayuda a determinar cuentos épocas han pasado.
Generar datos, trae conjuntos de entrada aleatorios, que después son evaluados, el número de ejemplos que se evaluarán es igual al número de instancias declarado.
Generar datos crea los pesos (4), en un rango de $(-10, 10)$, entre más grande sea el rango, es más visible el entrenamiento, después crea dos vectores de entradas X_1 y X_2, con $INSTANCIAS$ valores aleatorios entre $(0, 1)$, después para cada par de entradas se crea un par de salidas deseadas S_1 y S_2, en vectores de tamaño $INSTANCIAS$, donde para todo valor de S_1, S_2 en la misma posición debe de ser distinto y solo de valores $(1, 0)$. Finalmente se muestran y retornan.
Con los datos obtenidos se empieza a entrenar a la red neuronal

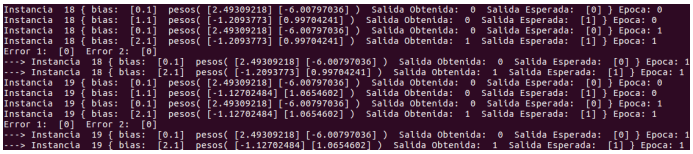

Figura 5. Ultimas 20 instancias fijas con una convergencia más rápida.

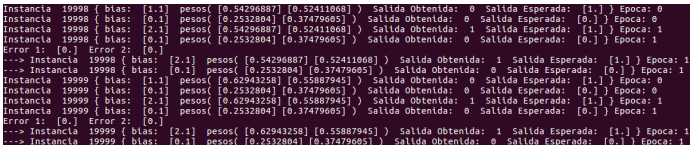

Figura 6. 20,000 Instancias aleatorias ejecutadas.

de dos perceptrones, cada perceptrón recibe las entradas X, pesos W, un bias, la función de activación que debe de ejecutar y la salidas esperada.
El perceptrón internamente ejecuta el producto escalar 1, la función de activación 2, y re ajusta los pesos y otros parámetros, como el bias θ y el error E II-B2.
Los valores actualizados son retornados, impresos y re enviados al perceptrón hasta que el error sea 0 o el número de épocas sea alcanzado por el contador, cuando sucede una de estas dos condiciones se despliegan, los errores finales que pueden o no ser iguales a cero y el mejor entrenamiento encontrado con o sin el valor esperado. Solo si $E = 0$ y $contador < EPOCAS - 1$, entonces se podrá decir que se obtuvo el valor esperado
Finalmente se gráfica los datos que se generaron en un inicio.

V. Conclusiones

Con un rango de valores aleatorios mayor los pesos podrán tomar mas tiempo en ser ajustados para converger.
Si se trabaja con muchas instancias, pero pocas épocas serán mas instancias las recorridas sin que se haya encontrado el mejor valor para los pesos.
Si se trabaja con pocas instancias pero suficientes épocas, con una pequeño puñado de estas se podrá hacer un buen entrenamiento de los pesos.
Cuando los pesos son ajustados lo suficiente el bias hace la división final de las clases.
Aproximadamente a las 20 instancias, el ajuste con pesos aleatorios entre $(-10, 10)$, puede converger de forma aceptable a aproximadamente 2 épocas.
Las primeras instancias tardan más en ser entrenadas, las ultimas demoran menos.

Referencias

[1] Marvin L. Minsky y Seymour A. Papert, "Perceptrons, An Introducction to Computacional Geometry," Expanded Edition, ISBN: 0-262-63111-3, 1988.
[2] https://github.com/Sikaerus/Red-Neuronal-Basica-Dos-Perceptrones/blob/master/Perceptron.py

```
import numpy as np
import matplotlib.pyplot as plt
import random

def perceptron(x1,x2,w1,w2,b,F,s):
        producto_escalar = (x1*w1) + (x2*w2) + b
        if F == 'E':
                y = funcion_activacion_escalon(producto_escalar)
        X = [x1, x2]
        W = [w1, w2]
        n_B, n_w1, n_w2, e = reajustes_y_calculos_error_bias_pesos(X,W,y,b,s)
        return n_B, n_w1, n_w2, y, e

def funcion_activacion_escalon(x):
        if x > 0.5:
                return 1
        elif x <= 0.5:
                return 0

def reajustes_y_calculos_error_bias_pesos(X,W,salida_obtenida,
        bias_actual, salida_esperada):
        a = 0.2
        error = salida_esperada - salida_obtenida
        nuevo_bias = bias_actual + salida_esperada
        Dw1 = a*error*X[0]
        nuevo_pesoW1 = W[0] + Dw1
        Dw2 = a*error*X[1]
        nuevo_pesoW2 = W[1] + Dw2
        return nuevo_bias, nuevo_pesoW1, nuevo_pesoW2, error

def main():
        INSTANCIAS = 20
        EPOCAS = 100
        FUNCION = 'E'
        contador = 0
        X1, X2, S1, S2, W = generar_datos(INSTANCIAS)
        for i in range(0, INSTANCIAS):
                bias1 = 0.1
                bias2 = 0.1
                while contador < EPOCAS:
                        bias1, W[0], W[2], y1, e1 = perceptron(X1[i],X2[i],W[0],W[2],
                        bias1,FUNCION,S1[i])
                        print('Instancia ', i, '{ bias: ',bias1, ' pesos('
                        ,W[0],W[2],')','Salida Obtenida: ',y1,'Salida Esperada: ',S1[i],
                        '} Epoca: ', contador)
                        bias2, W[1], W[3], y2, e2 = perceptron(X1[i],X2[i],W[1],W[3],
                        bias2,FUNCION,S2[i])
                        print('Instancia ', i, '{ bias: ',bias2, ' pesos('
                        ,W[1],W[3],')','Salida Obtenida: ',y2,'Salida Esperada: ',S2[i],
                        '} Epoca: ', contador)
                        if e1 == 0 and e2 == 0:
                                print('Error 1: ',e1, ' Error 2: ',e2)
                                break
                        contador += 1
                print('---> Instancia ', i, '{ bias: ',bias1, ' pesos(',W[0],W[2],')',
                ' Salida Obtenida: ',y1,'Salida Esperada: ',S1[i],'} Epoca: ', contador)
                print('---> Instancia ', i, '{ bias: ',bias2, ' pesos(',W[1],W[3],')',
                ' Salida Obtenida: ',y2,'Salida Esperada: ',S2[i],'} Epoca: ', contador)
                contador = 0
        graficar(X1, X2, S1, S2)

def generar_datos(INSTANCIAS):
        W = np.zeros((4, 1))
        for i in range(0, len(W)):
                W[i] = random.uniform(-10, 10)
        print(W, 'Pesos random -10,10')
        X1 = np.random.rand(INSTANCIAS, 1)
        X2 = np.random.rand(INSTANCIAS, 1)
        S1 = np.zeros(shape=(INSTANCIAS,1)); S2 = np.zeros(shape=(INSTANCIAS,1))
        for i in range(0, INSTANCIAS):
                S1[i] = random.choice([0, 1])
                if S1[i] == 0:
                        S2[i] = 1;
                elif S1[i] != 0:
                        S2[i] = 0;
        conjunto_datos = np.concatenate((X1, X2, S1, S2), axis=1)
        print('     X_{1}     X_{2}     S_{1}     S_{2}')
        print(conjunto_datos, 'Conjunto de datos de entrenamiento')
        return X1,X2,S1,S2,W

def graficar(X1, X2, S1, S2):
        plt.figure(figsize=(5,5))
        for s in range(0, len(S1)):
                if S1[s] == 0:
                        punto_rojo, = plt.plot(X1[s], X2[s], color='red', marker='o',
                         markeredgewidth=0.1, markersize=6)
                else:
                        estrella_azul, = plt.plot(X1[s], X2[s], color='blue', marker='*',
                         markeredgewidth=0.1, markersize=7)
        plt.xlabel('Valores en X1')
        plt.ylabel('Valores en X2')
        plt.grid(True)
        plt.title('Nube de puntos del conjunto de datos')
        plt.legend([punto_rojo, estrella_azul], ['C1(S1:0,S2:1)', 'C2(S1:1,S2:0)'])
        plt.show()
        plt.ion()

if __name__ == '__main__':
        main()
```