Perceptron Multicapa

May 14, 2019

1 Importamos las librerias necesarias

In [2]:
```
#!/usr/bin/python

import numpy as np
```

2 Definimos la función Sigmoid y su derivada

In [3]:
```
def sigmoid(x):
    """
    Metodo encargado de retorna la resultado
    de la funcion Sigmoid evaluada en x.

    Parametros:
        n: int
            Valor en la que se evaluara la función.

    Retorna:
        int: Funcion Sigmoid evaluada en x.
    """
    return 1.0/(1.0 + np.exp(-x))

def sigmoid_der(x):
    """
    Metodo encargado de retorna la resultado de la
    derivada de la funcion Sigmoid evaluada en x.

    Parametros:
        n: int
            Valor en la que se evaluara la función.

    Retorna:
        int: Derivada de la funcion
            Sigmoid evaluada en x.
    """
    return x * (1.0 - x)
```

3 Definimos la clase Perceptron

In [4]:
```
class Multicapa:
    """
    Perceptron Multicapa
    Codificación de un perceptron multicapa
    capaz de aprender la compuerta XOR.
    """
    def __init__(self, entrada):
        """
        Metodo constructor del preceptron,
        inicialza los valores por defecto.

        Parametros:
            entrada: array-1d
                Arreglo con valores iniciales.
        """
        self.entrada = entrada              # Vector de entrada.
        self.l = len(self.entrada)          # Par de datos iniciales (Capa oculta).
        self.li = len(self.entrada[0])      # Nř de cada par (Capa inicial).

        # Vector de pesos de la capa de entrada.
        self.wi = np.random.random((self.li, self.l))

        # Vector de pesos de la capa oculta.
        self.wh = np.random.random((self.l, 1))

    def respuesta(self, entrada):
        """
        Salida del perceptron multicapa, aplica el producto
        punto entre w (pesos) y entrada (data).

        Parametros:
            entrada: array-1d
                Arreglo con valores aleatorios iniciales.

        Retorna:
            array-1d: Respuesta del perceptron de los datos de entrada
                      despues de la capa de entrada y la capa oculta.
        """

        # Producto punto entre los datos de entrada y los pesos de la capa de entrada.
        s1 = sigmoid(np.dot(entrada, self.wi))

        # Producto punto entre la salida anterior y la capa oculta.
        s2 = sigmoid(np.dot(s1, self.wh))

        # Se retorna la salida de los datos despues de la capa oculta.
```

```
        return s2

    def entrenamiento(self, entrada, salidas, iteraciones = 100000, error = 0.005):
        """
        Metodo encargado de entrenar el perceptron multicapa.

        Parametros:
            entrada: Array, forma [[x1, y1], [x2, y2], ... , [xn, yn]]
                Vector con los datos de entrada, cada elemento
                es un par de datos de entrada.

            salidas: Array, forma [[s1], [s2], ... , [sn]]
                Vector con los datos de salida, cada elemento es
                la salida correspondiente del vector de entrada.

            iteraciones: int, numero de iteraciones maxima que tendra
                el perceptron multicapa para entrenar.
        """

        # Determina si el perceptron aprendio segun el criterio.
        aprendio = False

        # Numero de iteracion que le tomo al perceptron aprender.
        iter = 0

        # Valor promedio del error en la iteracion actual.
        promedio = 0

        while not aprendio:          # Mientras no aprenda.
            # Se asigna a la variable local los valores de entrada.
            l0 = entrada

            # Se realiza un producto punto entre los datos de entrada y el
            # vector peso de la capa de entrada.
            l1 = sigmoid(np.dot(l0, self.wi))

            # Se toma la salida anterior y se realiza un producto punto con
            # el vector peso de la capa oculta.
            l2 = sigmoid(np.dot(l1, self.wh))

            # Se toma el vector de error de la salida del perceptron con el real.
            l2_err = salidas - l2

            # Se haya el delta error de la capa oculta multiplicando el error
            # de la capa oculta con la derivada de la funcion sigmoid.
            l2_delta = np.multiply(l2_err, sigmoid_der(l2))

            # Se toma el vector error de la capa de entrada de un producto
```

```
                    # punto entre el vector delta error de la capa oculta y el
                    # vector peso de la capa de salida.
                    l1_err = np.dot(l2_delta, self.wh.T)

                    # Se haya el delta error de la capa de entrada multiplicando el
                    # error de la capa de la capa de entrada con la derivada de la
                    # funcion sigmoid.
                    l1_delta = np.multiply(l1_err, sigmoid_der(l1))

                    # Se actualizan los pesos de respectivas capas con los datos
                    # correspodientes y vectores delta de errores.
                    self.wh += np.dot(l1.T, l2_delta)
                    self.wi += np.dot(l0.T, l1_delta)

                    # Se contabiliza la iteracion para el criterio de aprendizaje.
                    iter += 1
                    promedio = np.mean(abs(l2_err))

                    # Criterio de salida:
                    # Si el error promedio es menor o igual 0.005, o
                    # la iteracion de aprendizaje sobrepasa el 100000
                    if error >= promedio or iter >= iteraciones:
                        # Se imprime las iteraciones necesarias para aprender.
                        print("Iteraciones {}".format(iter))

                        # Se imprime el error promedio de la ultima iteracion.
                        print("Error promedio {}".format(promedio))

                        # Salida del perceptron.
                        aprendio = True
```

4 Finalmente definimos el main para probar el Perceptron

```
In [5]: if __name__ == "__main__":
            entrada = np.array([[0,0], [0,1], [1,0], [1,1]])    # Vector de entrada.
            salidas = np.array([[0], [1], [1], [0]])            # Vector de salida.

            n = Multicapa(entrada)                  # Se instancia del perceptron.
            print(n.respuesta(entrada))             # Salida sin entrenar.
            print("")
            n.entrenamiento(entrada, salidas)       # Se entrena el perceptron multicapa.
            print(n.respuesta(entrada))             # Salida despues de entrenar.
```

```
[[0.70599688]
 [0.74714183]
 [0.73136076]
 [0.76787252]]
```

```
Iteraciones 82622
Error promedio 0.004999978678808071
[[0.00457964]
 [0.99488506]
 [0.99487496]
 [0.00518018]]
```

Se puede observar como la primera salida es el resultado del perceptron sin entrenar, una vez entrado, se imprime el numero de iteraciones que le tomo aprender con un error promedio no mayor al 0.005

Seguido se muestra la salida del perceptron con los mismos datos de entrada de la primera salida, siendo estas muy proximos a los reales dado el criterio de aprendizaje:

```
0.004 =~ 0
0.994 =~ 1
0.994 =~ 1
0.005 =~ 0
```