



UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

CENTRO UNIVERSITARIO UAEM ZUMPANGO

INGENIERÍA EN COMPUTACIÓN



## REDES NEURONALES

Compuertas lógicas con perceptrón simple

Alumno: Martín Márquez Cervantes

Profesor: Dr. Asdrúbal López Chau

Fecha de entrega: 1 de Noviembre 2019

### Introducción

Las compuertas lógicas comunes como la compuerta AND, OR y NOT representan un problema linealmente separable puesto que si graficamos los posibles estados que estos toman podemos separar como por ejemplo en la Figura 1.

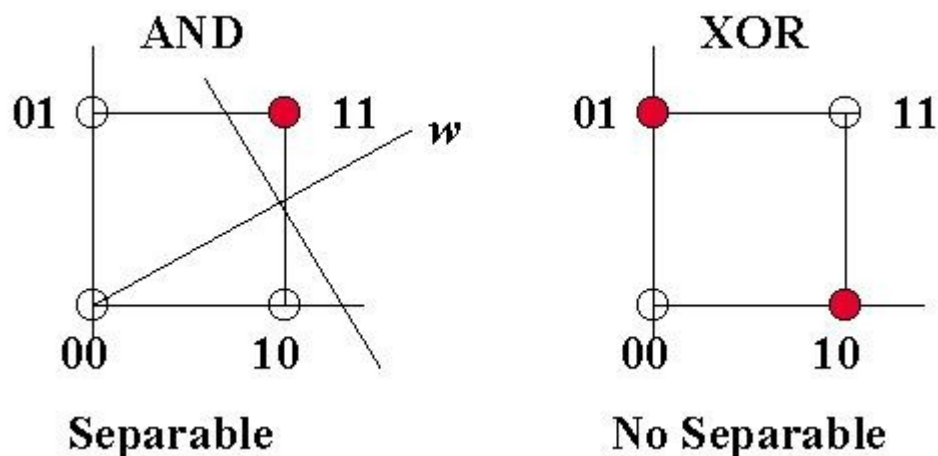


Figura 1 . Compuertas AND y XOR.

Sin embargo esto abre paso a preguntarse si en todas las compuertas se puede resolver el problema linealmente separable que como ya vimos se puede resolver con un perceptrón simple, pero al investigar más estos problemas nos damos cuenta que hay compuertas que no pueden ser resueltas, es el caso de la compuerta XOR que se muestra en la Figura 1 como un problema no linealmente separable, he aquí donde entran al rescate las redes neuronales multicapa, innovando con una solución

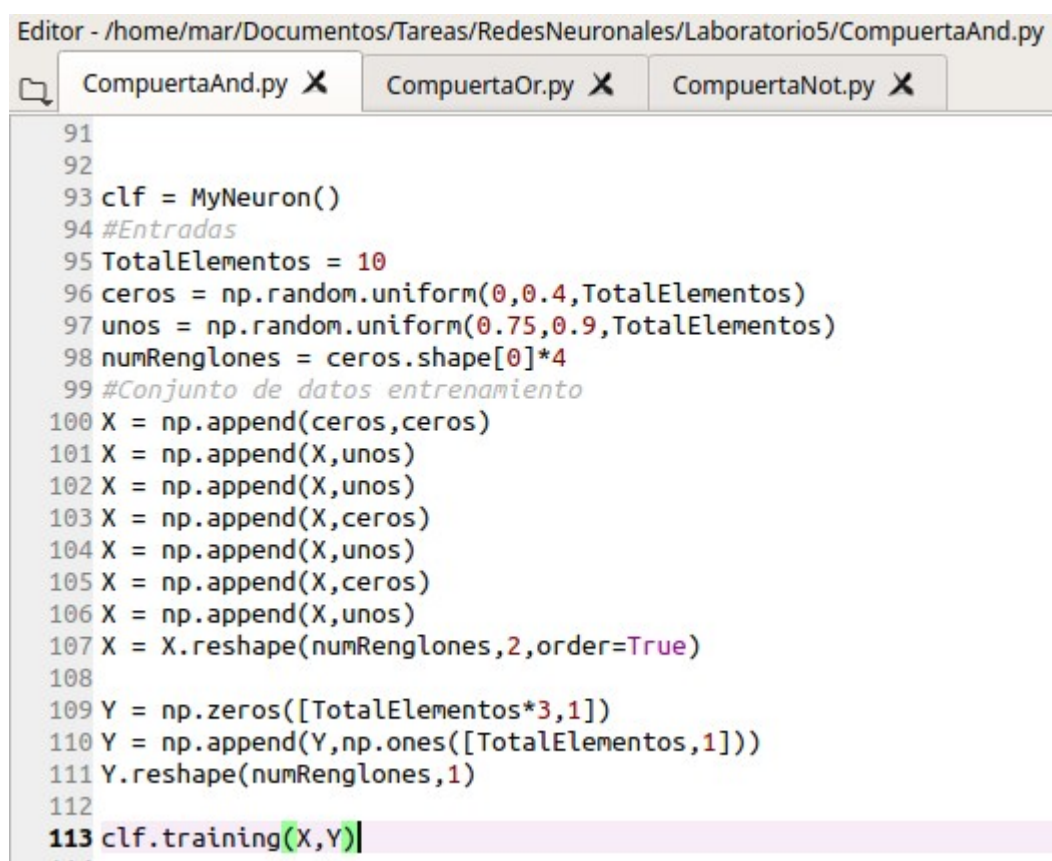
ingeniosa que después se realizara. Por el momento en esta práctica entrenaremos a un perceptrón para poder clasificar las compuertas AND, OR y NOT que después se utilizaran para poder implementar una red neuronal multicapa y resolver el problema de la compuerta XOR.

## Desarrollo.

Para esta ocasión utilizaremos el código de perceptrón simple puesto que implementaremos solamente esta vez nuestro conjunto de datos de prueba y de entrenamiento.

### Compuerta AND

Para la compuerta AND generamos nuestros datos de entrenamiento.



```
Editor - /home/mar/Documentos/Tareas/RedesNeuronales/Laboratorio5/CompuertaAnd.py
CompuertaAnd.py X CompuertaOr.py X CompuertaNot.py X

91
92
93 clf = MyNeuron()
94 #Entradas
95 TotalElementos = 10
96 ceros = np.random.uniform(0,0.4,TotalElementos)
97 unos = np.random.uniform(0.75,0.9,TotalElementos)
98 numRenglones = ceros.shape[0]*4
99 #Conjunto de datos entrenamiento
100 X = np.append(ceros,ceros)
101 X = np.append(X,unos)
102 X = np.append(X,unos)
103 X = np.append(X,ceros)
104 X = np.append(X,unos)
105 X = np.append(X,ceros)
106 X = np.append(X,unos)
107 X = X.reshape(numRenglones,2,order=True)
108
109 Y = np.zeros([TotalElementos*3,1])
110 Y = np.append(Y,np.ones([TotalElementos,1]))
111 Y.reshape(numRenglones,1)
112
113 clf.training(X,Y)
```

Figura 2 Conjunto de datos.

En las entradas generamos dos arrays numpy aleatorios, limitados por  $0 < \text{ceros} < 0.4$  y  $0.75 < \text{unos} < 0.9$  y los ingresaremos a nuestra matriz X que simulara la tabla de verdad con estos valores oscilando, representando la tensión de entrada de una tabla de verdad.

Una vez rellena y re-organizada en matriz, de igual forma generamos sus etiquetas en un vector con las mismas filas pero con una sola columna que nos representa la salida lógica de la compuerta, en el array de numpy Y insertamos treinta elementos ceros de entrenamiento y diez elementos unos para poder etiquetar los cuarenta datos de entrenamiento correctamente.

Después el método training entra en acción mandando la matriz de datos y etiquetado.

```
CompuertaAnd.py X CompuertaOr.py X CompuertaNot.py X
16 #entrenamiento perceptron
17 def training(self,X,Y):
18     #inicializar w con valores pseudo-aleatorios
19     w = np.random.rand(3)
20     #Nota: Dimensiones de w son el numero de columnas de x+1
21     #-----
22     #Paso 2 : Agregar una columna de unos a la matriz x
23     X=np.append(np.ones((X.shape[0],1)),X,axis=1)
24
25     #Antes de modificar w guardamos para impresión
26     wInicial = w
27     XN = X.shape[0] #40
28     #Algoritmo perceptron
29     for i in range(1,21):
30         for j in range(XN):
31             if np.dot(w,X[j]) >= 0:
32                 y=1
33             else:
34                 y=0
35             w=w+(Y[j]-y)*X[j]
36     #guardamos w modificada en wG para despues utilizarla en algoritmo pre
37     self.wG=w
38     #impresión de resultados
39     print("w Inicial: "+str(wInicial))
40     print("w Final: "+str(w))
41     #graficación de vectores
42     plt.plot(wInicial,'.-')
43     plt.plot(w,'.-')
44     print('Linea azul W Inicial')
45     print('Linea naranja W Final aplicando algoritmo')
46
```

Figura 3 Método Training.

En la Figura 3 Podemos ver que se inicializa el array w con 3 elementos, luego a X insertamos la columna de unos para poder hacer el producto interno entre estos dos, en la línea 29 entramos al algoritmo de entrenamiento que se repite veinte veces el ciclo entre los productos internos entre w y las entradas x1, x2 y el bias x0 = 1.

El algoritmo actualiza w y este queda al final representado por la siguiente grafica.

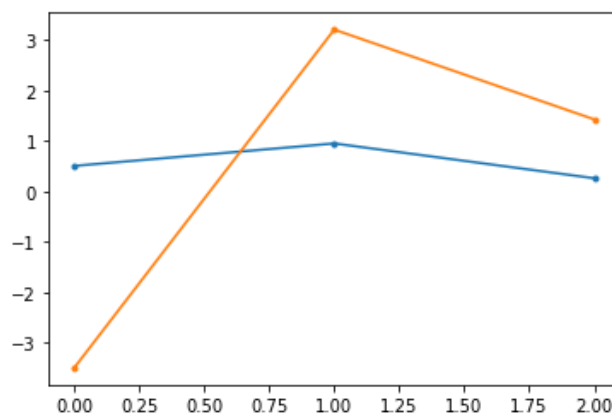


Figura 4 Grafica de los pesos sinápticos.

En la Figura 4 vemos como w inicial (Azul) se va actualizando hasta llegar a w final (naranja).

Aplicando el algoritmo y teniendo ya los pesos, podemos ahora si predecir.

```
115 #conjuntos de datos de prueba 20 elementos
116 cerosTest = np.zeros(5)
117 unosTest = np.ones(5)
118 #Conjunto de datos
119 XT = np.append(cerosTest,cerosTest)
120 XT = np.append(XT,unosTest)
121 XT = np.append(XT,unosTest)
122 XT = np.append(XT,cerosTest)
123 XT = np.append(XT,unosTest)
124 XT = np.append(XT,cerosTest)
125 XT = np.append(XT,unosTest)
126 XT = XT.reshape(20,2,order=True)
127 YT = np.zeros(15)
128 YT = np.append(YT,np.ones(5))
129 YT.reshape(YT.size,1)
130
131 XT=np.append(np.ones((XT.shape[0],1)),XT,axis=1)
132
133 Predicciones = []
134 for i in range(XT.shape[0]):
135     Predicciones.append(clf.predic(i,XT))
136 Predicciones = np.array(Predicciones)
137 #impresión
138 for i in range(XT.shape[0]):
139     print("Indice " +str(i) + " prediccion " +str(Predicciones[i]))
```

Figura 5 Código de predicción.

De la misma forma en la que se genero los datos de entrenamiento ahora se generan veinte elementos de prueba para poder predecir, está vez con clases concretas (ceros y unos lógicos).

## Resultados y pruebas Desarrolladas.

Los resultados par ala compuerta and aplicando la función de sigmoid son los siguientes.

```
Indice 1 prediccion 0.02070079457393189
Indice 2 prediccion 0.02070079457393189
Indice 3 prediccion 0.02070079457393189
Indice 4 prediccion 0.02070079457393189
Indice 5 prediccion 0.10055879890061299
Indice 6 prediccion 0.10055879890061299
Indice 7 prediccion 0.10055879890061299
Indice 8 prediccion 0.10055879890061299
Indice 9 prediccion 0.10055879890061299
Indice 10 prediccion 0.5274844304969292
Indice 11 prediccion 0.5274844304969292
Indice 12 prediccion 0.5274844304969292
Indice 13 prediccion 0.5274844304969292
Indice 14 prediccion 0.5274844304969292
Indice 15 prediccion 0.8551629506502246
Indice 16 prediccion 0.8551629506502246
Indice 17 prediccion 0.8551629506502246
Indice 18 prediccion 0.8551629506502246
Indice 19 prediccion 0.8551629506502246
```

Figura 6 Predicciones con la función de activación sigmoid.

Donde en la Figura 6 los valores menores a 0.5 representan un porcentaje bajo y se interpretan como cero y los valores mayores a 0.5 representan los unos. En esta ocasión el perceptrón se equivocó en los índices diez al catorce y se pueden visualizar en la matriz de confusión.

```
TP = 5
TN = 10
FP = 5
FN = 0

Precisión de clasificación: 75.0 %
Precisión : 50.0 %
Recall : 100.0 %
F-Score : 66.66666666666667
```

Figura 7 Resultados compuerta AND Sigmoid.

En la compuerta OR y NOT el desarrollo es similar solo que cambia al momento de asignar las etiquetas de entrenamiento y de prueba como en la Figura 8.

```
#Clases para Or
YOR = np.zeros([TotalElementos,1])
YOR = np.append(YOR,np.ones([TotalElementos*3,1]))
YOR.reshape(numRenglones,1)

#clase para compuerta Or
YT = np.zeros(5)
YT = np.append(YT,np.ones(15))
YT.reshape(YT.size,1)

XT=np.append(np.ones((XT.shape[0],1)),XT,axis=1)
```

Figura 8 Datos de entrenamiento OR.

Para la compuerta OR nuestra gráfica de pesos sinápticos, predicciones utilizando tangente hiperbólica y resultados es el siguiente.

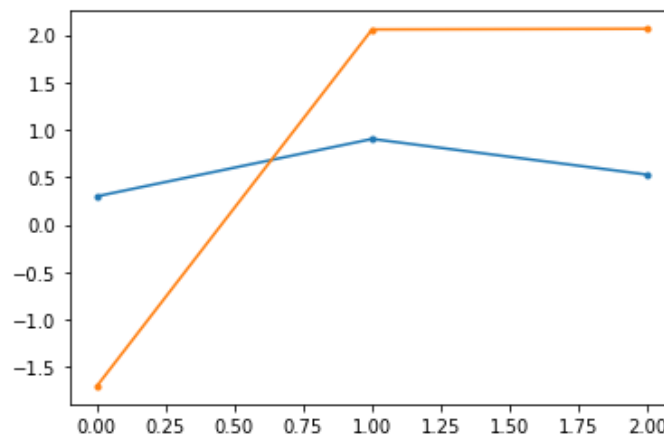


Figura 9 Gráfica de pesos w compuerta OR.



```

Indice 0 prediccion -0.9357991454496829
Indice 1 prediccion -0.9357991454496829
Indice 2 prediccion -0.9357991454496829
Indice 3 prediccion -0.9357991454496829
Indice 4 prediccion -0.9357991454496829
Indice 5 prediccion 0.3445248014297161
Indice 6 prediccion 0.3445248014297161
Indice 7 prediccion 0.3445248014297161
Indice 8 prediccion 0.3445248014297161
Indice 9 prediccion 0.3445248014297161
Indice 10 prediccion 0.33873198497388846
Indice 11 prediccion 0.33873198497388846
Indice 12 prediccion 0.33873198497388846
Indice 13 prediccion 0.33873198497388846
Indice 14 prediccion 0.33873198497388846
Indice 15 prediccion 0.9841537262189545
Indice 16 prediccion 0.9841537262189545
Indice 17 prediccion 0.9841537262189545
Indice 18 prediccion 0.9841537262189545
Indice 19 prediccion 0.9841537262189545

```

Figura 10 Resultados predicciones función tangente Hiperbólica.

```

TP = 15
TN = 5
FP = 0
FN = 0

Matrix Confussion
[[15  5]
 [ 0  0]]

Precisión de clasificación: 100.0 %
Precisión : 100.0 %
Recall : 100.0 %
F-Score : 100.0

```

Figura 11 Matriz de confusión compuerta OR.

## Conclusiones.

Con este tipo de ejemplos pienso que se puede entender muy bien el tema de los problemas que puede o no resolver las redes neuronales simples como el perceptrón y pienso que es importante ver la historia y la naturaleza del surgimiento de éstas pues podemos comprender desde un inicio como se crearon, comparten y funcionan las redes neuronales independientemente del lenguaje que uses es muy importante conocer como funciona.

## Referencias.

[decsai.ugr.es/pub/castro/Actividades/Redes-Neuronales/Apuntes](https://decsai.ugr.es/pub/castro/Actividades/Redes-Neuronales/Apuntes)  
<https://disi.unal.edu.co/~lctorress/RedNeu/LiRna004.pdf>

