Redes Neuronales Artificiales Multicapa (MLPClassifer)

Christian Hernández,
Noviembre 2020.

Universidad Politécnica Salesiana.
Ingeniería de Sistemas.
Inteligencia Artificial.

Abstract

The architectures seen so far neural networks formed by layers of neurons, navigating the data from the input layer to the output layer, etc. correspond to what is called a multilayer perceptron. Despite the name, this architecture is not perceptron-based, the term "percentron" has simply been retained in the name for historical reasons.

Capítulo 1

Definición

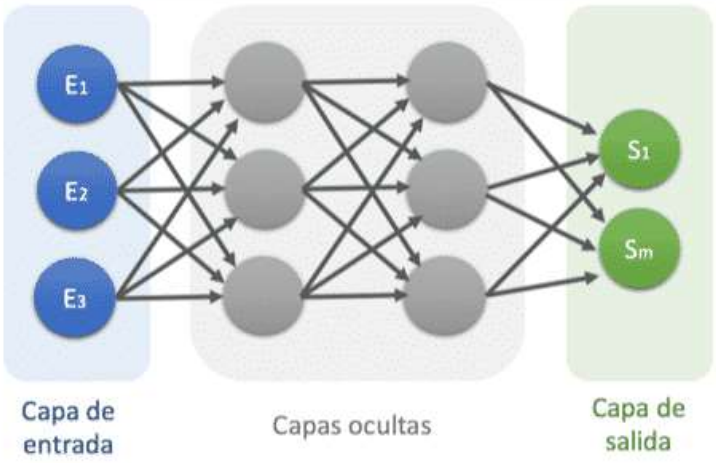
Las arquitecturas vistas hasta ahora redes neuronales formadas por capas de neuronas, navegando los datos desde la capa de entrada hasta la capa de salida, etc.
Se corresponde con lo que se denomina perceptrón multicapa. A pesar del nombre, esta arquitectura no está basada en perceptrones, simplemente se ha conservado el término "percentrón" en el nombre por motivos históricos.

En el perceptrón multicapa se pueden diferenciar unas 2 fases:

- Propagación en la que se calcula el resultado de salida de la red desde los valores de entrada hacia delante.
- Aprendizaje en la que los errores obtenidos a la salida del perceptrón se van propagando hacia atrás (backpropagation) con el objetivo de modificar los pesos de las conexiones para que el valor estimado de la red se asemeje cada vez más al real, esta aproximación se realiza mediante la función gradiente del error

  .

Arquitectura



Capa de entrada: conecta la red con el exterior, cada neurona se corresponde con cada una de las variables de entrada de la red.

Capas ocultas: son una aglomeración de capas en as que cada activación de una salida procede de la suma ponderada de las activaciones de la capa anterior conectadas, mas sus correspondientes umbrales (bias, sesgos).

Capa de salida: conecta las capas ocultas con la salida de la red que proporciona los resultados.

Ejemplo básico 1: compuerta XOR

Como se conoce, el perceptrón simple (de una sola neurona y una sola capa) no es capaz de resolver problemas que no sean separables linealmente.

Por ello, en esta sección aprenderemos cómo resolver un el sencillo problema de la compuerta XOR (que no es separable linealmente).!

Corpus

Para entrenar la red, debemos tener claro en primer lugar, cuáles son las entradas y salidas que nuestra red neuronal deberá aprender. En la siguiente tabla se puede apreciar los patrones de entrada (donde cada patrón está conformado por dos entradas x_1x1 y x_2x2), y las correspondientes etiquetas o salidas (δδ).

x_1x1	x_2x2	δδ
0	0	0
0	1	1
1	0	1
1	1	0

Repositorio

Git: https://github.com/ChristianHernand16/Inteligencia-Artificial-II.git

Práctica ANN-0:

Modifique el código anterior, a fin de usar Hot Encoding y contar con 2 salidas en lugar de 1.

Realizamos la respectiva modificación utilizando Encoding para después realizar el entrenamiento teniendo los pesos entre la entrada y la capa oculta.

Después realizamos el test para ver cómo está procesando las neuronas

```
# Importamos el Perceptron Multicapa para Clasificacion
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

lenc= LabelEncoder()
encoded = lenc.fit_transform(d)
enc = OneHotEncoder(handle_unknown='ignore', sparse=False)
encoded=encoded.reshape(len(encoded),1)
dm=enc.fit_transform(encoded)

# Creamos la red neuronal
mlp=MLPClassifier(solver = 'lbfgs', activation='logistic', verbose=True, alpha=1e-4, tol=1e-15, max_iter=10000,
                  hidden_layer_sizes=(neuronas_capa_oculta, neuronas_capa_salida))

print(mlp)
# Realizamos el proceso de entrenamiento
mlp.fit(x,dm)

# Mostramos los pesos entre la entrada y la capa oculta
print('Pesos W^(0): \n:',mlp.coefs_[0])

# Mostramos los pesos entre la capa oculta y la capa de salida
print('\nPesos W^(1): \n:',mlp.coefs_[1])

# Probamos si la red devuelve valores apropiados de acuerdo a las entradas (test):
for entrada in x:
    print('\nPrueba con {','|'.join([str(i) for i in entrada]),'} => ',mlp.predict(entrada.reshape(1,-1)))
```

Práctica ANN-1:

- Genere 1000 puntos aleatorios con coordenadas $(x1,x2)$(x1,x2). Con estos puntos, deberá realizar las siguientes tareas:
- Seleccionar de forma aleatoria 80% de los puntos para entrenar la red y el restante 20% se empleará para probar la red.
- Entrenar la red hasta lograr un error mínimo. Probar la red y presentar la matriz de confusión. Indicar el nivel de precisión (muestras correctamente clasificadas frente al total de muestras

$$precision = \frac{muestras\ \ correctamente\ \ clasificadas}{total\ \ de\ \ muestras}$$

- Realizamos la respectiva importación de las librerías correspondientes
- Generamos loas 1000 puntos
- Después realizamos las respectivas comparaciones dentro su rango y damos las coordenadas y realizamos el respectivo procesamiento

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

import numpy as np
import matplotlib.pyplot as pp
import random
n = 1000
coordenadas = []
delta = [1 if i % 2 == 0 else 0 for i in range(0, n)]
#print (delta)
for i in range(0, n):
    coordenadas.append([random.random() * 100, random.random() * 100])
datos = np.array(coordenadas)
x_train, x_test, d_train, d_test = train_test_split(datos, delta, test_size=0.80, random_state = 42 )



#mlp=MLPClassifier(solver = ' lbfgs ' , activation= ' logistic ' , verbose=True, alpha=1e-4,tol=1e-15,
#hidden_layer_sizes=(150, 2))
mlp=MLPClassifier(solver = 'lbfgs', activation='logistic', alpha=1e-4, tol=1e-15, max_iter=10000, \
                  hidden_layer_sizes=(150, 2))
print(mlp)
mlp.fit( datos, delta )
prediccion = mlp.predict(x_test)
print( ' Matriz de Confusion\n ' )
matriz = confusion_matrix(d_test, prediccion)
print(confusion_matrix(d_test, prediccion))
print( ' \n ' )
print(classification_report(d_test, prediccion))
```

Tenemos la matriz de confusiones con la presión que corresponde al procesamiento teniendo la exactitud, promedio y el valor del peso promedio teniendo valores medio ajustados

```
MLPClassifier(activation='logistic', hidden_layer_sizes=(150, 2),
              max_iter=10000, solver='lbfgs', tol=1e-15)
 Matriz de Confusion

[[169 239]
 [ 48 344]]


              precision    recall  f1-score   support

           0       0.78      0.41      0.54       408
           1       0.59      0.88      0.71       392

    accuracy                           0.64       800
   macro avg       0.68      0.65      0.62       800
weighted avg       0.69      0.64      0.62       800
```

Referencias

- Arana, E., Delicado, P., Martí-Bonmatí, L. (1999). Validation procedures in diagnostic models. Neural network and logistic regression. Investigative Radiology, 34(10), 636-642.
- Arbib, M.A. (Ed.) (1995). The handbook of brain theory and neural networks.Cambridge, Mass.: MIT Press.
- Bahbah, A.G. y Girgis, A.A. (1999). Input feature selection for real-time transient stability assessment for artificial neural network (ANN) using ANN sensitivity analysis. Proceedings of the 21 st International Conference on Power Industry Computer Applications, 295-300