ENTRENAMIENTO DE UNA RED NEURONALES ARTIFICIALES

Sigilfredo Ibáñez Arias

Fecha de entrega: 24/04/2022
2022-1

Se requiere entrenar una red neuronal multicapa para dar solución al problema de la compuerta lógica XOR y que posteriormente pueda generalizar la salida con valores de entrada que no fueron parte del entrenamiento.

Tabla de verdad compuerta XOR

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Código de entrenamiento en Python

```
#Se importan dependencias
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense

# cargamos las 4 combinaciones de las compuertas XOR
training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")

# y estos son los resultados que se obtienen, en el mismo orden
target_data = np.array([[0],[1],[1],[0]], "float32")

model = Sequential()
model.add(Dense(25, input_dim=2, activation='relu'))
model.add(Dense(15, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
```

```
            metrics=['accuracy'])

model.fit(training_data, target_data, epochs=10000,batch_size=10)

# evaluamos el modelo
scores = model.evaluate(training_data, target_data)

print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))

salida = model.predict(training_data)

# serializar el modelo a JSON
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
# serializar los pesos a HDF5
model.save_weights("model.h5")
print("Modelo Guardado!")

#Cálculo del error cuadrático medio
mse = np.sqrt(np.sum(np.square(target_data-salida)))/4
print(salida)
print('Error: ',mse)
```

RESULTADOS

Se hizo el entrenamiento para una red neuronal artificial que fuese capaz de discriminar los valores de salida de una compuerta XOR y que posteriormente a esto tuviera la capacidad de generalizar a cualquier otro valor de entrada diferente a los valores que se le utilizaron para el entrenamiento, se probaron diferentes configuraciones de la red como se muestra en la siguiente tabla y para cada uno se calculó el error cuadrático medio para tener una estimación de cual daba un mejor resultado. Los modelos fueron guardados en un archivo JSON para posteriormente ser evaluados en la red.

Estructura	Funciones	Error
2 2 1	relu relu	0,25001258
2 2 1	sigmo sigmo	0,25012094
2 2 1	tanh tanh	0,18497697
2 2 1	relu tanh	0,22337054
2 2 1	sigmo tanh	0,24810915
2 2 1	tanh sigmo	0,19653219
2 8 1	relu relu	0,16764821
2 8 1	sigmo sigmo	0,24746561
2 8 1	tanh tanh	0,24929634
2 8 1	relu tanh	0,07475954
2 8 1	sigmo tanh	0,25382033
2 8 1	tanh sigmo	0,168853
2 8 1	relu sigmo	0,2143463
2 16 1	relu relu	0,05942339
2 16 1	sigmo sigmo	0,24887218
2 16 1	tanh tanh	0,1346817
2 16 1	relu tanh	0,09877399
2 16 1	sigmo tanh	0,25122219
2 16 1	tanh sigmo	0,13363101
2 16 1	relu sigmo	0,14114821
2 16 8 1	relu relu relu	2,18E-08
2 16 8 1	sigmo sigmo	0,24797961
2 16 8 1	tanh tanh tanh	0,05122184
2 16 8 1	tanh tanh sigmo	0,04005857
2 16 8 1	relu tanh sigmo	0,00686002
2 25 15 1	relu tanh sigmoid	0,00024947

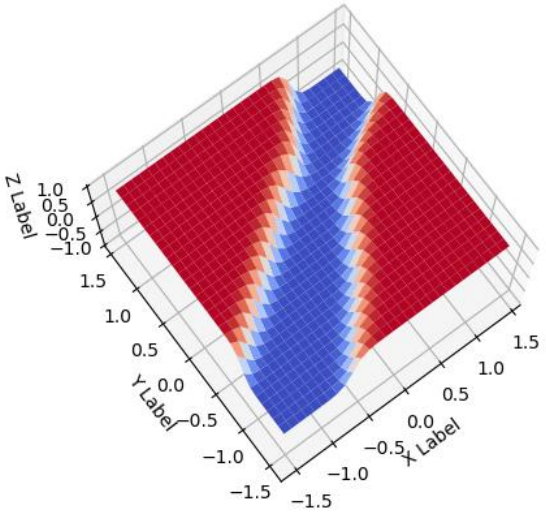
Código evaluación de la red

```
#Se importan dependencias
import numpy as np
from keras.models import model_from_json
import matplotlib.pyplot as plt
from matplotlib import cm
# cargar json y crear el modelo
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
```

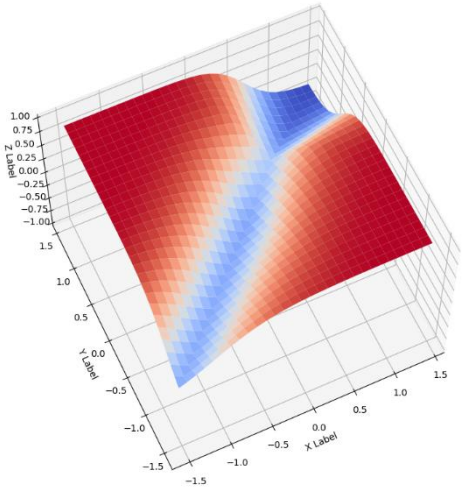
```
loaded_model = model_from_json(loaded_model_json)
# cargar pesos al nuevo modelo
loaded_model.load_weights("model.h5")
print("Cargado modelo desde disco.")

# Compilar modelo cargado y listo para usar.
loaded_model.compile(loss='binary_crossentropy',
            optimizer='adam',
            metrics=['accuracy'])
xtotal=[]
ytotal=[]

#Pares de vectores para las combinaciones de entradas
vecx=np.arange(-1.5, 1.5, 0.1)
vecy=np.arange(-1.5, 1.5, 0.1)

for x2 in range (30):
    yt=[]
    #Se crean las parejas en entradas y se evalúa el modelo
    for x1 in range(30):
        vec=vecx[x1],vecy[x2]
        vec=np.array(vec)
        vec= vec[np.newaxis]
        xtotal.append(vec)
        yf=loaded_model.predict(vec)
        yt.append(float(np.array(yf)))
    ytotal.append(np.array(yt))
    print(x2+1)

#Se genera grafica 3D
vecx,vecy= np.meshgrid(vecx,vecy)
ytotal=np.array(ytotal)

fig= plt.figure(1)
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(vecx, vecy, ytotal,cmap=cm.coolwarm,rstride=1, cstride=1)
ax.set_zlim(-1.01,1.01)
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')
plt.show()
```

Se evaluaron diferentes modelos y a continuación se muestran graficas tridimensional de los modelos que mejor resultado dinero

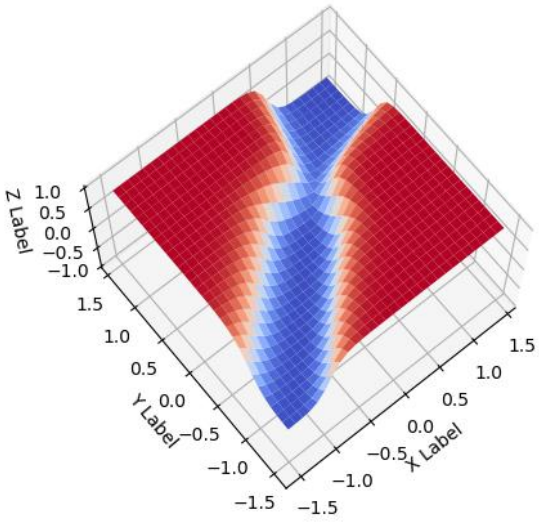
Estructura 2-16-1



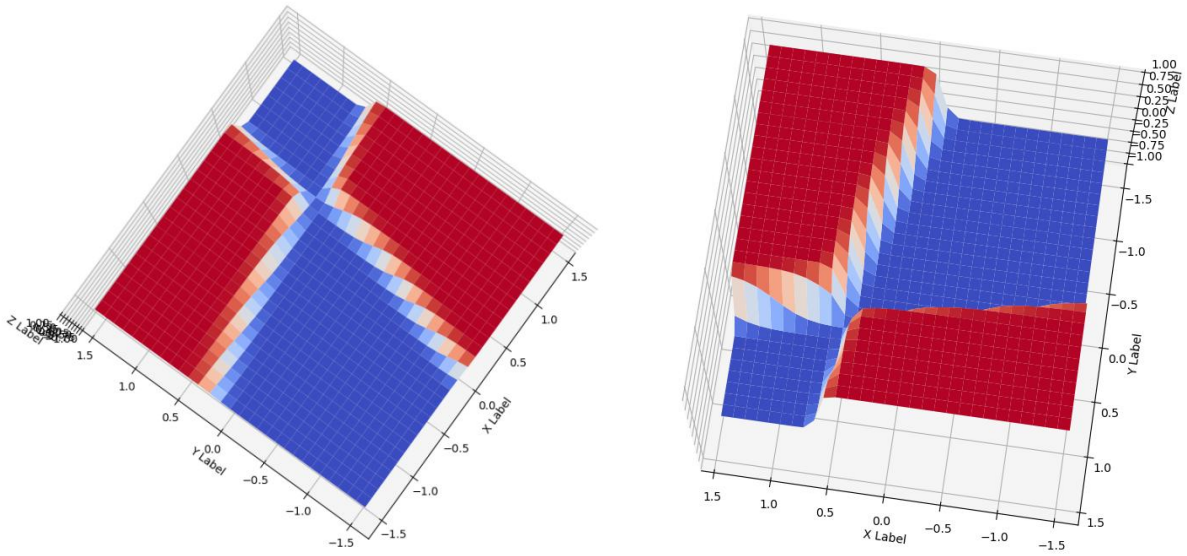
Estructura 2-16-4-1



Estructura 2-19-9-1



Estructura 2-25-15-1



CONCLUSIONES

Como se puede ver en representación grafica de los diferentes modelos evaluados el que resultado se obtuvo fue el que tenia la estructura 2-25-15-1 ya que este modelo hace una mejor discriminación de la salida para los diferentes valores de entrada diferente a los valores que se utilizaron para entrenar la red.

Se pudo observa el bueno desempeño de la función de activación sigmoidal en la última capa de la red ya que esta transforma los valores de entrada en un rango de 0,1 donde los valores altos tienden de manera asintótica a 1 y los valores bajos tienden de manera asintótica a 0.

Ninguno de los modelos entrenados alcanzo completamente a generalizar la los valores de salida para otras valores de entrada diferentes de cero y uno, esto debido a que los valores de entrenamiento de la res solo fueron cuatro combinaciones y las salidas esperadas solo estaban en el rango de cero y uno. Como se puede ver en las graficas la red neuronal alcanza a identificar los valores en ese rango