CONTROL INTELIGENTE-1

2021-2

Informe #1

Por: Cristian Alejandro Agudelo Zapata

Se solicita resolver una red neuronal multicapa para la compuerta XOR, ya que es un problema que un clasificador simple no podría resolver.

Tabla de verdad Compuerta XOR

x1	x2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Código utilizado:

```
#!/usr/bin/env python3

# -*- coding: utf-8 -*-

"""

Created on Thu Oct 21 11:23:29 2021

@author: cristian

"""

#Importancion de librerias de trabajo

import numpy as np  #numpy permite trabajar con arreglos de vectores o matrices

import matplotlib.pyplot as plt  #matplotlib ayuda con la graficacion

from keras.models import Sequential # importancion de una RNA tipo secuencial

from keras.layers.core import Dense #importancion de keras el dense que son capas a utilizar en la RNA


#Entradas XOR
```

```
training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")

#Target-Salidas XOR
target_data = np.array([[0],[1],[1],[0]], "float32")

#Se crea un modelo tipo secuencial ya que se va ejecutando en paralelo segun capas
model = Sequential()
#Dense implementa la operación: de capas, num entradas, f.activacion
model.add(Dense(32, input_dim=2, activation='tanh'))
model.add(Dense(16, activation='tanh'))
model.add(Dense(1, activation='tanh'))

model.compile(loss='mean_squared_error', #funcion de perdida error cuadradatico medio
        optimizer='adam', #Adam le permite a la red como ajustar los pesos y cesgo de manera eficiente
        metrics=['binary_accuracy']) #metrica binaria ya que se trabaja con 0-1

#Se entrena el modelo con 1000 epocas
historial=model.fit(training_data, target_data, epochs=1000,verbose=0)

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

#graficacion de las epocas y su magnitud de perdida
plt.xlabel(" Epoca")
plt.ylabel(" Magnitud de perdida")
plt.plot(historial.history["loss"])
```

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

print (model.predict(training_data))
```

```
# serializar el modelo a JSON, es decir guardamos los pesos de la red neuronal.

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!")
```

Análisis

Para empezar con el entrenamiento de la red neuronal se realizo pruebas aleatorias según función de activación, el cual luego de varios intentos se decide utilizar la tangente hiperbólica en las diferentes capas, ya que fue la que arrojaba resultados de buena predicción, luego ya con esto claro se formaba la red neuronal según capas de entrada y capa oculta.

	Capa 1	Capa oculta	Binnary_accuracy	Pesos (W)
Numero Capas	4	0	100.00%	[0.00855663], [0.8897865], [0.8916779], [0.03216176
	8	0	100.00%	[0.00651702],[0.9289663],[0.9298147],[0.00965172]
	16	0	100.00%	[0.00918972],[0.9157399],[0.9193621],[0.0127658]
	32	0	100.00%	[0.00592709],[0.93373483],[0.93715656],[0.00865844]
	4	2	100.00%	[0.09591625],[0.7554137],[0.75382024],[0.16718782]
	4	4	100.00%	[0.02060616],[0.76083744],[0.77422816],[0.12545085]
	4	8	100.00%	[0.00272967],[0.9432754],[0.9331017],[0.00697882]
	4	16	100.00%	[0.00165277],[0.963779],[0.9610701],[0.0027319]
	8	10	100.00%	[0.00097696],[0.9734679],[0.972464],[0.00203464]
	8	16	100.00%	[0.0010578],[0.97010833],[0.97163385],[0.00189584]
	16	8	100.00%	[0.00159708],[0.9625771],[0.96461564],[0.00364193]
	16	32	100.00%	[0.00141076],[0.96582305],[0.9636009],[0.00251739]
	32	16	100.00%	[0.00106297],[0.96688956],[0.96875656],[0.00196163]

Tabla1. Entrenamiento de la red neuronal

Luego de guardados los pesos en un archivo, se procede a realizar predicciones las cuales se pueda evaluar que tan acertado esta el modelo con otras entradas.

Código utilizado:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 21 12:18:44 2021

@author: cristian
"""
from keras.models import model_from_json
import numpy as np
import matplotlib.pyplot as plt
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.")
# Se compila y carga el modelo
loaded_model.compile(loss='mean_squared_error',
                     optimizer='adam',
                     metrics=['binary_accuracy'])

#Creaccion de espacio bidiminensional como parametros de entrada de la RNA
vecx=np.arange(-0.5, 1.5, 0.1)
vecy=np.arange(-0.5, 1.5, 0.1)
xtotal=[]
ytotal=[]
```

```
#Iteracion de ciclos para crear las difentes parejas de entrada
for x2 in range (len(vecy)):
   yt=[]
   for x1 in range(len(vecx)):
      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))
vecx,vecy= np.meshgrid(vecx,vecy)
ytotal=np.array(ytotal)

#Se crea el elemento de graficacion para observar las predicciones segun puntos
#En espacio tridimensional
fig= plt.figure(1)
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(vecx, vecy, ytotal,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()
```

A continuación se presentara de forma gráfica el espacio tridimensional de los principales arreglos neuronales creado para el espacio de trabajo según capas de la neurona.

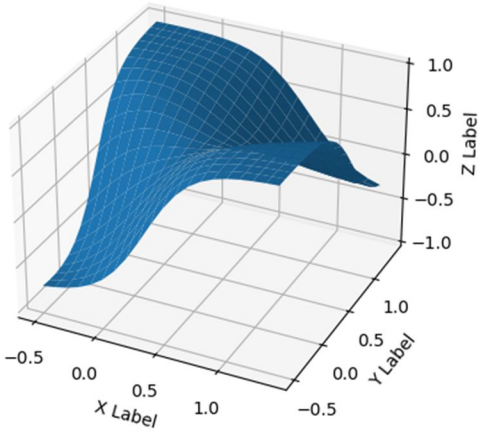


Figura1. Predicción con 4 capas

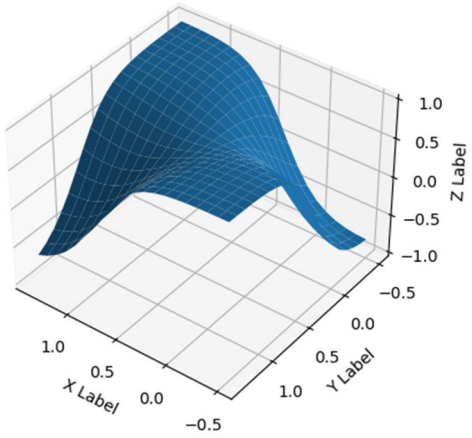


Figura 2. Predicción con 8 capas

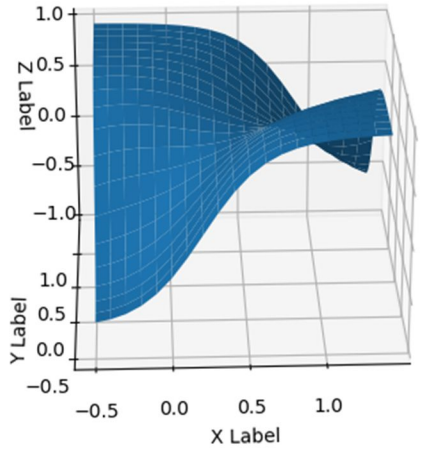


Figura 3. Predicción con 16 capas

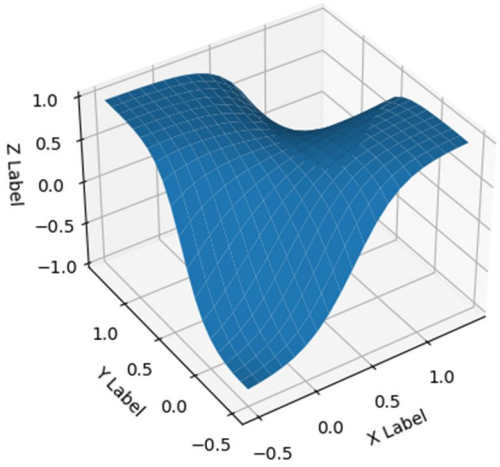


Figura 4. Predicción con 32 capas

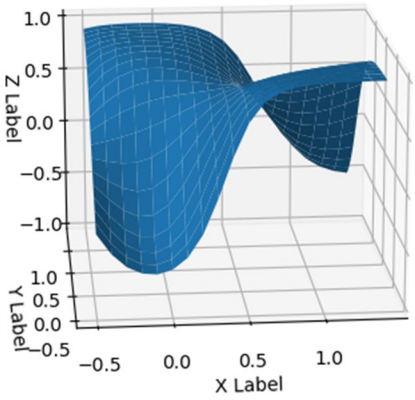


Figura 5. Predicción con 4 capas y 8 ocultas

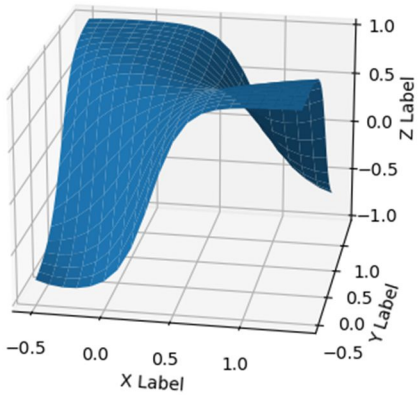
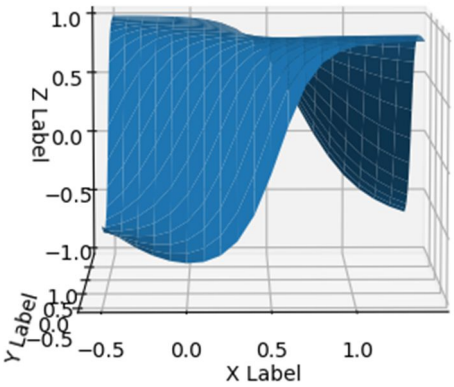


Figura 6. Predicción con 4-16 ocultas




```
binary_accuracy: 100.00%
[[0.00097696]
 [0.9734679 ]
 [0.972464  ]
 [0.00203464]]
Modelo Guardado!
```

Figura 7. 8-10 ocultas y pesos de la RNA

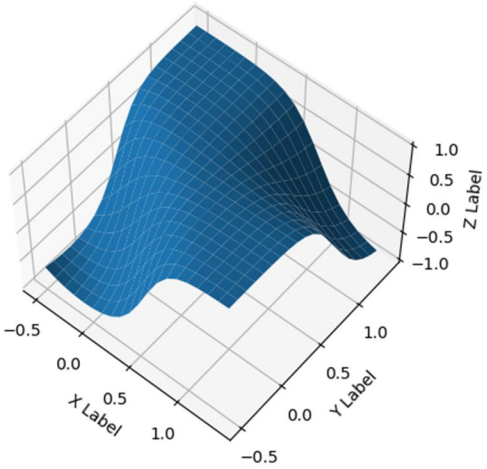


Figura 8. Predicción con 16-32

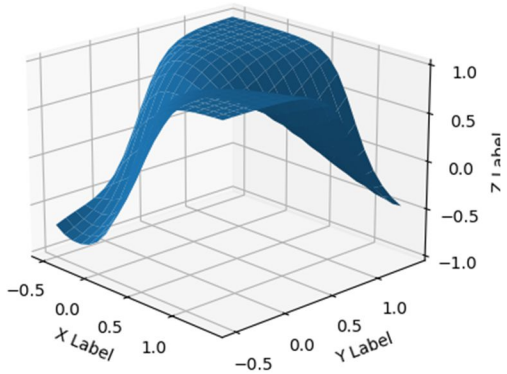


Figura 9. Predicción con 32-16 ocultas

Conclusiones

1. Se puede concluir después de varios intentos cambiando la estructura y cantidad de neuronas por cada capa de la red, que la presento mejor comportamiento en cuanto a pesos y una red sencilla fue la de 32 capas de entrada y ninguna oculta, ya en arreglo mas compuesto la mejor fue RNA fue la que tenia 8 capas de entrada y 10 ocultas.

2. Realizar el entrenamiento de las redes neuronales multicapa requiere mucho tiempo humano, ya que para un arreglo neuronal al principio puede no obtenerse los mejores resultados pero después de varias pruebas de ensayo y error pueden ir evolucionando los resultados hacia los requeridos.