Respuestas de Trabajando con el gradiente descendente

Diego Iván Perea Montealegre (2185751) diego.perea@uao.edu.co

Facultad de Ingeniería, Universidad Autónoma de Occidente

Cali, Valle del Cauca

1. Realice manualmente (sin usar ninguna herramienta de simulación) la actualización de pesos de una red Adaline que resuelva el problema de la función lógica OR (ver tabla). Pruebe al menos una vez cada una de las combinaciones de entrada, luego de hacer esto, calcule el valor de la de la pérdida (loss) con que quedó la red entrenada. Use como valores de pesos iniciales w1=0.5 w2=-0.5 b=-1.5. Seleccione un valor del parámetro de aprendizaje entre 0.5 y 1.0

X1	X2	D
0	0	0
0	1	1
1	0	1
1	1	1

1.

$d = valor\ de\ la\ salida\ , y = salida\ , p = patrones\ , M = Neuronas\ de\ salida\ , \alpha = valor\ de\ aprendizaje\ , e = error$

$w_i(t+1) = w_i(t) + \alpha(d-y)x_i$ Ecuación de peso

$b(t+1) = b(t) + eM$ Ecuación de bias

$e = d - y$ Ecuación de error

$y = x_i w_i + x_n w_n + b$ Ecuación de salida

$L = \frac{1}{MP} \sum_{P=1}^{p} \sum_{k=1}^{M} (d_k - y, k)$ Ecuación de perdida

Primera combinación

$x_1 = 0 \quad x_2 = 0 \quad d = 0 \quad w_1 = 0.5 \quad w_2 = -0.5 \quad \alpha = 0.7 \quad b = -1.5$

Salida

$$y = x_1 w_1 + x_2 w_2 + b$$

$$y = 0 * 0.5 + 0 * -0.5 + (-1.5)$$

$$y = -1.5$$

Error

$$e_1 = d - y$$

$$e_1 = 0 - (-1.5)$$

$$e_1 = 1.5$$

Peso

$$w_1(t+1) = 0.5 + 0.7 * 1.5 * 0$$
$$w_1(t+1) = 0.5$$

$$w_2(t+1) = -0.5 + 0.7 * 1.5 * 0$$
$$w_2(t+1) = -0.5$$

Bias $$b(t+1) = -1.5 + 1.5(1)$$

$$b(t+1) = 0$$

Segunda combinación

$x_1 = 0 \quad x_2 = 1 \quad d = 1 \quad w_1 = 0.5 \quad w_2 = -0.5 \quad \alpha = 0.7 \quad b = 0$

Salida

$$y = x_1 w_1 + x_2 w_2 + b$$

$$y = 0 * 0.5 + 1 * -0.5 + (0)$$

$$y = -0.5$$

Error

$$e_2 = d - y$$

$$e_2 = 1 - (-0.5)$$

$$e_2 = 1.5$$

Peso

$$w_1(t+1) = 0.5 + 0.7 * (1 - (-0.5) * (0))$$
$$w_1(t+1) = 0.5$$

$$w_2(t+1) = -0.5 + 0.7 * (1 - (-0.5) * (1)$$
$$w_2(t+1) = 0.55$$

Bias $$b(t+1) = 0 + 1.5(1)$$

$$b(t+1) = 1.5$$

Tercera combinación

$x_1 = 1 \quad x_2 = 0 \quad d = 1 \quad w_1 = 0.5 \quad w_2 = 0.55 \quad \alpha = 0.7 \quad b = 1.5$

Salida

$$y = x_1 w_1 + x_2 w_2 + b$$

$$y = 1 * 0.5 + 0 * 0.55 + (1.5)$$

$$y = 2$$

Error

$$e_3 = d - y$$

$$e_3 = 1 - (2)$$

$$e_3 = -1$$

Peso

$$w_1(t+1) = 0.5 + 0.7 * (1 - (-1) * (1))$$
$$w_1(t+1) = 1.9$$

$$w_2(t+1) = 0.55 + 0.7 * (1 - (-0.5) * (0)$$
$$w_2(t+1) = 0.55$$

Bias $$b(t+1) = 1.5 + (-1) * 1$$

$$b(t+1) = 0.5$$

Cuarta combinación

$x_1 = 1 \quad x_2 = 1 \quad d = 1 \quad w_1 = 1.9 \quad w_2 = 0.55 \quad \alpha = 0.7 \quad b = 0.5$

Salida

$$y = x_1 w_1 + x_2 w_2 + b$$

$$y = 1 * 1.19 + 1 * 0.55 + (0.5)$$

$$y = 2.95$$

Error

$$e_4 = d - y$$

$$e_4 = 1 - (2.95)$$

$$e_4 = -1.95$$

Peso

$$w_1(t+1) = 1.9 + 0.7 * (1 - (2.95) * (1))$$
$$w_1(t+1) = 0.535$$

$$w_2(t+1) = 0.55 + 0.7 * (1 - (2.95) * (1)$$
$$w_2(t+1) = -0.815$$

Bias $$b(t+1) = 0.5 + (-1.95) * 1$$

$$b(t+1) = -1.45$$

Total de perdida

$w_1 = 0.535 \quad w_2 = -0.515 \quad \alpha = 0.7 \quad b = -1.45 \quad M = 1 \quad P = 4$

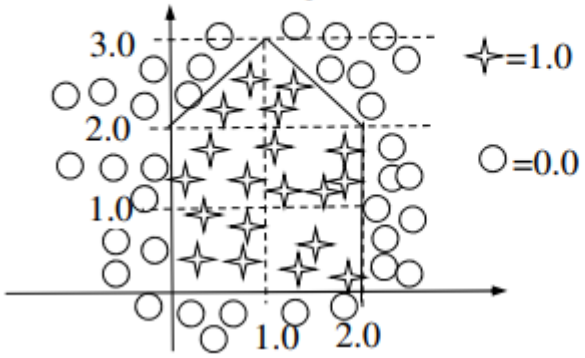
$$L = \frac{1}{MP} \sum_{P=1}^{p} \sum_{k=1}^{M} (d_k - y, k)\}$$

$$L = \frac{1}{4}(e_1)^2 + (e_2)^2 + (e_3)^2 + (e_4)^2$$

$$L = \frac{1}{4}(1.5)^2 + (1.5)^2 + (-1)^2 + (-1.95)^2$$

$$L = 2.32$$

2. Encuentre una red neuronal multicapa superficial compuesta de neuronas con función de activación tipo escalón que permita realizar la separación de los patrones mostrados en la figura.



Se realiza la selección del trazado para estimar la clasificación de las estrellas frente a los círculos dado una línea de forma de pendiente

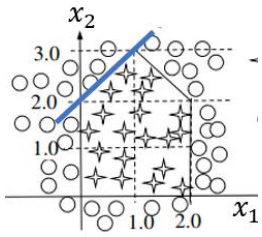


Figura 1. Trazo de primera línea

$$x_2 = -\frac{w_1}{w_2} \cdot x_1 - \frac{b}{w_2} \qquad x_2 = mx_1 + l$$

$$P_1 = (0,2) \quad P_2 = (1,3) \quad m = \frac{3-2}{1-0} = 1$$

$$x_2 = x_1 + l$$

$$2 = 0 + l$$

$$l = 2$$

$$x_2 = x_1 + 2 \qquad \rightarrow \qquad -\frac{w_1}{w_2} = 1 \qquad \rightarrow \qquad -\frac{b}{w_2} = 2$$

$$w_1 = -w_2 \quad \rightarrow \quad b = -2w_2 \qquad \text{Darle valor de } w_2 = 1$$

$$w_1 = -1 \quad \rightarrow \quad b = -2 \qquad \text{Remplazar en } x_2$$

$$x_2 = -\frac{-1}{1} \cdot x_1 - \frac{-2}{1} \qquad \rightarrow \quad x_2 = x_1 + 2$$

Pesos Resultantes $P_1 = (-1,1)$ $P_2 = (1,-1)$

Con $w_2 = 1$ se dirige a los ceros del sistema $P_1 = (-1,1)$ intuyendo $w_2 = 1$ para así en dirección $P_2 = (1,-1)$

Resultando -> $w_1 = 1$ $w_2 = -1$ $b = 2$

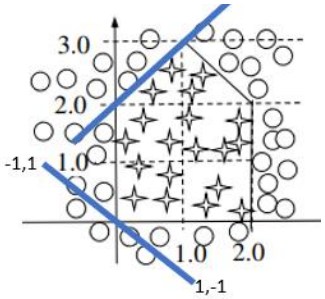


Figura 2. Trazo de pesos primera línea

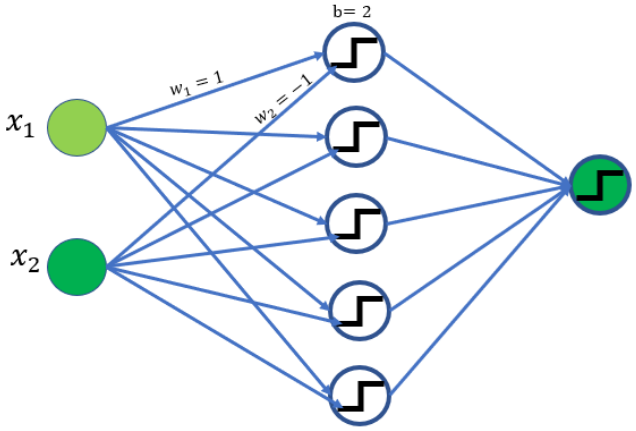


Figura 3. Red neuronal multicapa con primera línea

Dado a este trazo se procede a realizar el segundo trazo

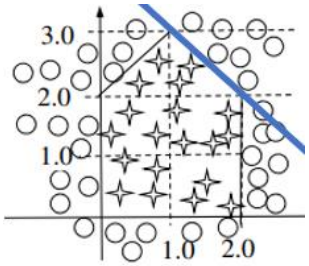


Figura 4. Trazo de segunda línea

$$x_2 = -\frac{w_1}{w_2} \cdot x_1 - \frac{b}{w_2} \qquad x_2 = mx_1 + l$$

$$P_1 = (1,3) \quad P_2 = (2,2) \quad m = \frac{2-3}{2-1} = -1$$

$$x_2 = x_1 + l$$

$$2 = -1 + l$$

$$l = 3$$

$$x_2 = x_1 + 3 \quad \rightarrow \quad -\frac{w_1}{w_2} = 1 \quad \rightarrow \quad -\frac{b}{w_2} = 3$$

$$w_1 = -w_2 \quad \rightarrow \quad b = -3w_2 \quad \rightarrow w_1 = w_2 \qquad \text{Darle valor de } \; w_2 = 1$$

$$w_1 = 1 \quad \rightarrow \quad b = -3 \qquad \text{Remplazar en } x_2$$

$$x_2 = -\frac{1}{1} \cdot x_1 - \frac{-3}{1} \qquad \rightarrow \quad x_2 = x_1 + 3$$

Pesos Resultantes $P_1 = (1,1)$ $P_2 = (-1,-1)$

Con $w_2 = 1$ se dirige a los ceros del sistema $P_1 = (1,1)$ intuyendo $w_2 = -1$ para así en dirección $P_2 = (-1,-1)$

Resultando -> $w_1 = -1$ $w_2 = -1$ $b = 3$

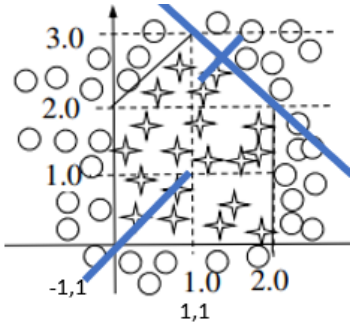


Figura 5. Trazo de pesos segunda línea

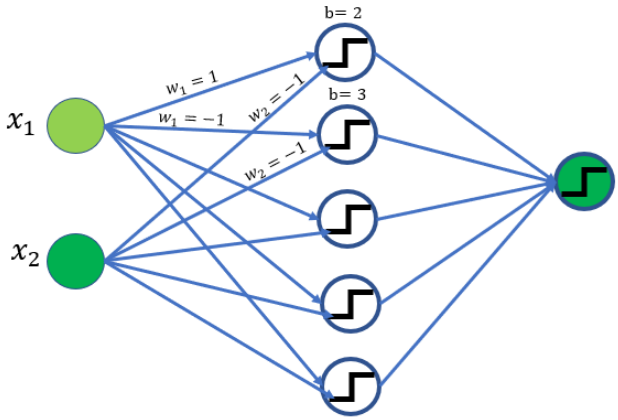


Figura 6. Red neuronal multicapa con segunda línea

Dado a este trazo se procede a realizar el tercer trazo

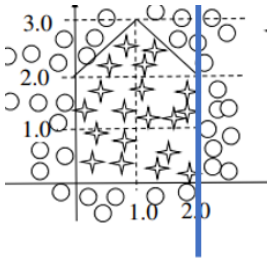


Figura 7. Trazo de tercera línea

$$x_1 = -\frac{w_2}{w_1} \cdot x_2 - \frac{b}{w_1} \qquad x_1 = mx_2 + l$$

$$P_1 = (2,3) \quad P_2 = (2,1) \quad m = \frac{1-3}{2-2} = 0$$

$$x_1 = x_2 + l$$

$$2 = 0 + l$$

$$l = 2$$

$$x_1 = 0x_2 + 2 \qquad \rightarrow \qquad -\frac{w_2}{w_1} = 0 \qquad \rightarrow \qquad -\frac{b}{w_1} = 2$$

$$w_2 = -0w_1 \quad \rightarrow \quad b = -2w_1 \qquad \text{Darle valor de } w_1 = 1$$

$$w_2 = 0 \quad \rightarrow \quad b = -2 \qquad \text{Remplazar en } x_1$$

$$x_1 = -\frac{0}{1} \cdot x_2 - \frac{-2}{1} \qquad \rightarrow \quad x_1 = 0x_2 + 2$$

Pesos Resultantes $P_1 = (1,0)$ $P_2 = (-1,0)$

Con $w_1 = 1$ se dirige a los ceros del sistema $P_1 = (1,0)$ intuyendo $w_2 = -1$ para así en dirección $P_2 = (-1,0)$

Resultando -> $w_1 = -1$ $w_2 = 0$ $b = 2$

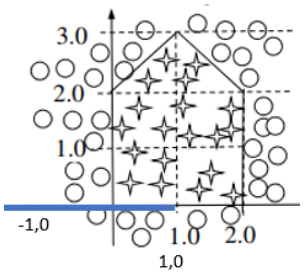


Figura 8. Trazo de pesos tercera línea

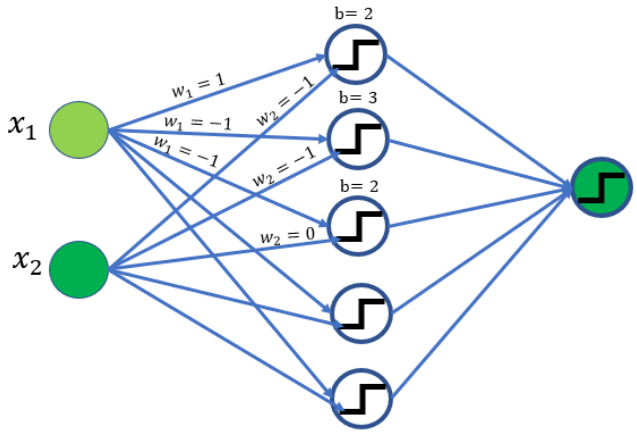


Figura 9. Red neuronal multicapa con tercera línea

Dado a este trazo se procede a realizar el cuarto trazo

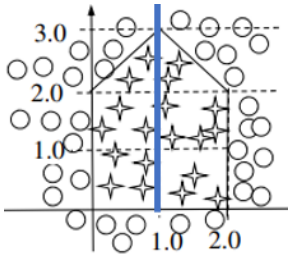


Figura 10. Trazo de cuarta línea

$$x_1 = -\frac{w_2}{w_1} \cdot x_2 - \frac{b}{w_1} \qquad x_1 = mx_2 + l$$

$$P_1 = (1,3) \quad P_2 = (1,1) \quad m = \frac{1-3}{1-1} = 0$$

$$x_1 = x_2 + l$$

$$2 = 0 + l$$

$$l = 2$$

$$x_1 = 0x_2 + 2 \qquad \rightarrow \qquad -\frac{w_2}{w_1} = 0 \qquad \rightarrow \qquad -\frac{b}{w_1} = 2$$

$w_2 = -0w_1 \quad \rightarrow \quad b = -2w_1$ Darle valor de $w_1 = 1$

$w_2 = 0 \quad \rightarrow \quad b = -2$ Remplazar en x_1

$$x_1 = -\frac{0}{1} \cdot x_2 - \frac{-2}{1} \qquad \rightarrow \quad x_1 = 0x_2 + 2$$

Pesos Resultantes $P_1 = (1,0)$

Con $w_1 = 1$ se dirige a los unos del sistema $P_1 = (1,0)$

Resultando -> $w_1 = 1 \quad w_2 = 0 \quad b = -2$

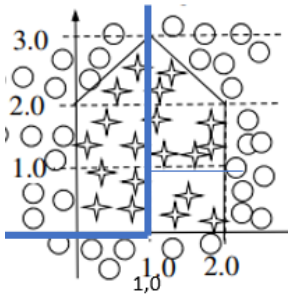


Figura 11. Trazo de pesos cuarta línea

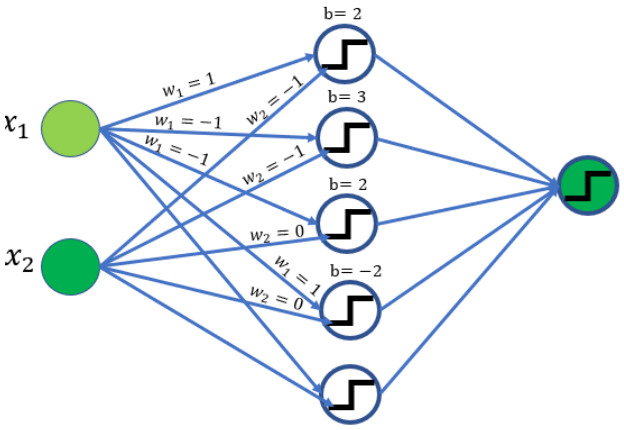


Figura 12. Red neuronal multicapa con cuarta línea

Dado a este trazo se procede a realizar el quinto trazo

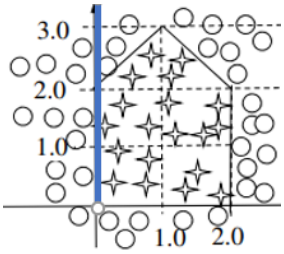


Figura 13. Trazo de quinta línea

$$x_1 = -\frac{w_2}{w_1} \cdot x_2 - \frac{b}{w_1} \qquad x_1 = mx_2 + l$$

$$P_1 = (0,1) \quad P_2 = (0,1) \quad m = \frac{1-1}{0-0} = 0$$

$$x_1 = x_2 + l$$

$$2 = 0 + l$$

$$l = 2$$

$$x_1 = 0x_2 + 2 \qquad \rightarrow \qquad -\frac{w_2}{w_1} = 0 \qquad \rightarrow \qquad -\frac{b}{w_1} = 2$$

$w_2 = -0w_1 \quad \rightarrow \quad b = -2w_1$ Darle valor de $w_1 = 1$

$w_2 = 0 \quad \rightarrow \quad b = -2$ Remplazar en x_1

$x_1 = -\frac{0}{1} \cdot x_2 - \frac{-2}{1} \quad \rightarrow \quad x_1 = 0x_2 + 2$

Pesos Resultantes $P_1 = (1,0)$

Con $w_1 = 1$ se dirige a los unos del sistema $P_1 = (1,0)$

Resultando -> $w_1 = 1 \quad w_2 = 0 \quad b = -2$

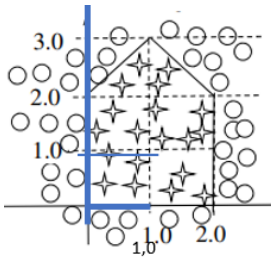


Figura 14. Trazo de pesos quinta línea

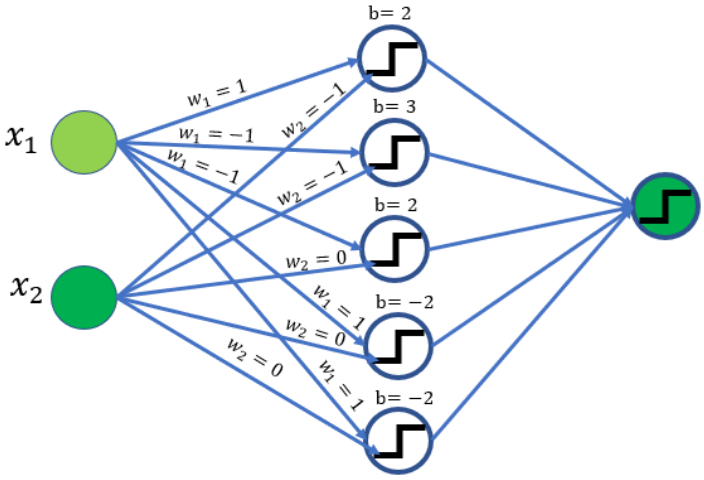


Figura 15. Red neuronal multicapa step con quinta línea

Desarrollo de las salidas según el trazo :

Primer trazo neurona

$$y = x_1 w_1 + x_2 w_2 + b$$

$$y = 0 + 2 * -1 + 2$$

$$y = 0$$

Segundo trazo neurona

$$y = x_1 w_1 + x_2 w_2 + b$$

$$y = -1 * -1 + 2 * -1 + 3$$

$$y = 2$$

Tercer trazo neurona

$$y = x_1 w_1 + x_2 w_2 + b$$

$$y = 2 * -1 + 0 + 2$$

$$y = 0$$

Cuarto trazo neurona

$$y = x_1 w_1 + x_2 w_2 + b$$

$$y = 2 * 1 + 0 + (-2)$$

$$y = 0$$

Quinto trazo neurona

$$y = x_1 w_1 + x_2 w_2 + b$$

$$y = -1 * -1 + 2 * -1 + 3$$

$$y = 0$$

Salida de la neurona

$$Con\ b = 1\ , a\ seleccionar$$

$$y = 0 + 2 + 0 + 0 + 1$$

$$y = 3$$

3. Resuelva el problema de la función lógica OR usando una red Adaline y una red multicapa superficial en tensorflow-keras. Pruebe al menos tres valores del parámetro de aprendizaje. Use tensorboard para visualizar la evolución de la pérdida (loss) y los grafos de las redes neuronales creadas. ¿Qué puede concluir al variar el valor del parámetro de aprendizaje?

Se procede a realizar la Red Adaline para la compuerta OR , en el que se toma en cuenta las siguientes librerías

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import datetime,os
from tensorboard.plugins.hparams import api as hp
```

Figura 16. importación de librería necesarias para la ejecución de la Red Adaline OR

Para crear la Red Adaline se declara las variable x_train como las entradas y de salida y_train , para que realmente sea una Red Adaline se tuvo que realizar una función en el que se denominó como

model_or , en el que esta se configuraba la arquitectura , de tipo secuencial (una después de otra) , dos neuronas de entrada y una neurona de salida.

```
RED ADALINE

[8]  x_train=np.array([[0,0],[0,1],[1,0],[1,1]]) #las dos entradas
     y_train=np.array([[0],[1],[1],[1]])#las salida
     #Para saber las dimensiones de la entrada y salidas:
     #entrada_dimension=x_train.shape[1]
     #salida_dimension=y_train.shape[1]

[9]  def model_or():
       model=Sequential()
       model.add(Dense(1,input_dim=2,activation='relu'))

       model.summary()
       model.compile(loss='mean_squared_error',optimizer=tf.keras.optimizers.SGD(learning_rate=0.001))
       return model
```

Figura 17. Configuración de arquitectura de Red Adaline OR

Se llama la función model_or en la que se guerda en la variable modelo para asi mejorar el entendimiento del código y procesos a ejecutar, posteriormente se configura el entrenamiento en el que tensoboard se visualizará las épocas y los , para observar la eficiencia y la predicción dicha de la red Adaline creado con los parámetros anteriores mencionados.

```
[10] #Ejecución de función
     modelo=model_or()

     Model: "sequential_1"
     _________________________________________________________________
      Layer (type)                 Output Shape              Param #
     =================================================================
      dense_1 (Dense)              (None, 1)                 3

     =================================================================
     Total params: 3
     Trainable params: 3
     Non-trainable params: 0
     _________________________________________________________________

[11] #Entrenamento para que tensoboard take data FORMA 1

     log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
     tensorboard_callback = tf.keras.callbacks.TensorBoard(
         log_dir=log_dir,histogram_freq=1)
     hparams_callback = hp.KerasCallback(log_dir,{
         'num_relu_units': 512,
         'dropout': 0.2
     })
```

Figura 18. Configuraciones de Tensorboard OR

Además se realiza el entrenamiento del modelo, en el que tiene épocas de 500 y un batch_size de 1

```
history1=modelo.fit(x_train,y_train,epochs=500 , batch_size=1,callbacks=[tensorboard_callback,hparams_callback])

Epoch 1/500
4/4 [==============================] - 0s 7ms/step - loss: 0.1562
Epoch 2/500
4/4 [==============================] - 0s 6ms/step - loss: 0.1552
Epoch 3/500
4/4 [==============================] - 0s 7ms/step - loss: 0.1543
Epoch 4/500
4/4 [==============================] - 0s 7ms/step - loss: 0.1534
Epoch 5/500
4/4 [==============================] - 0s 7ms/step - loss: 0.1525
Epoch 6/500
4/4 [==============================] - 0s 6ms/step - loss: 0.1516
Epoch 7/500
4/4 [==============================] - 0s 6ms/step - loss: 0.1508
Epoch 8/500
4/4 [==============================] - 0s 7ms/step - loss: 0.1500
Epoch 9/500
```

Figura 19. Entrenamiento de la red Adaline OR

Se llama a la funcion predict ara que prdiga los resultados frente a las entradas

```
[14] #Prediccion
     prediccion_original=modelo.predict(x_train)
     #prediccion_filtrada=(prediccion_original > 0.7) + 0
     prediccion_filtrada=(prediccion_original > 0.4) + 0
     print("Entradas: \n",x_train)
     print("Respuesta de red: \n",prediccion_original)
     print("Predicion hecha : \n",prediccion_filtrada)

     Entradas:
      [[0 0]
      [0 1]
      [1 0]
      [1 1]]
     Respuesta de red:
      [[0.05686129]
      [0.78343564]
      [0.65611535]
      [1.3826897 ]]
     Predicion hecha :
      [[0]
      [1]
      [1]
      [1]]
```

Figura 20. Predicción de la Red Adaline OR

Al observar las predicciones hechas se puede ver que son correctas , pero para visualizar cuanto loss frente a las épocas que se dieron , y presentar confiabilidad se muestra la comparación a traves de tensorboard .

```
!tensorboard dev upload --logdir logs \
    --name "Red adaline OR" \
    --description "(optional) Simple comparison of several hyperparameters"
```

Figura 21. Visualización en Tensorboard

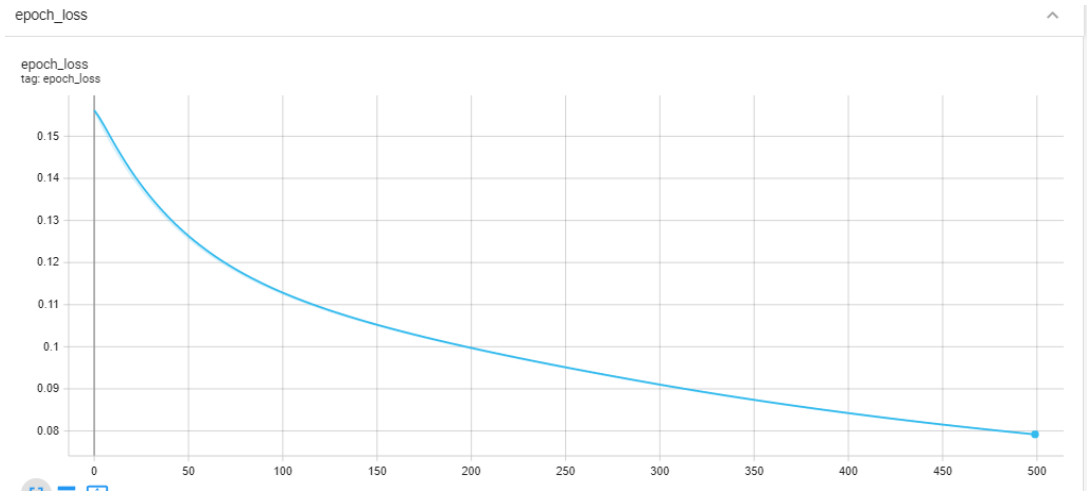


Figura 22. Loss vs epochs

En la anterior figura se observa como al no tener una capa oculta la red Adaline pudo realizar la predicción de la compuerta OR , dando resultados satisfactorios .

Red multicapa superficial , para la realización de esta red se debe poner una capa oculta en la cual se evidencia el cambio de capas, a diferencia de la adaline que no tenia capa oculta.

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import datetime,os
from tensorboard.plugins.hparams import api as hp
x_train=np.array([[0,0],[0,1],[1,0],[1,1]]) #las dos entradas
y_train=np.array([[0],[1],[1],[1]])#las salida
#Para saber las dimensiones de la entrada y salidas:
#entrada_dimension=x_train.shape[1]
#salida_dimension=y_train.shape[1]
```

Figura 23. importación de librería y configuración red

Se procede a poner dos neuronas de entrada , con 16 neuronas en la capa oculta con función de activación relu y una neurona de salida con función de activación sigmoid, con una tasa de aprendizaje (learning_rate) de 0.1

```
[18] def model_or():
       model=Sequential()
       model.add(Dense(16,input_dim=2,activation='relu'))
       model.add(Dense(1,activation='sigmoid'))
       model.summary()
       model.compile(loss='mean_squared_error',optimizer=tf.keras.optimizers.SGD(learning_rate=0.1))
       return model

[19] #Ejecución de función
     modelo=model_or()

     Model: "sequential_2"
     _________________________________________________________________
      Layer (type)                 Output Shape              Param #
     =================================================================
      dense_2 (Dense)              (None, 16)                48

      dense_3 (Dense)              (None, 1)                 17

     =================================================================
     Total params: 65
     Trainable params: 65
     Non-trainable params: 0
```

Figura 24. Configuración de arquitectura de Red multicapa superficial OR

```
#Entrenamento para que tensoboard take data FORMA 1

log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir=log_dir,histogram_freq=1)
hparams_callback = hp.KerasCallback(log_dir,{
    'num_relu_units': 512,
    'dropout': 0.2
})

history1=modelo.fit(x_train,y_train,epochs=500 , batch_size=1,callbacks=[tensorboard_callback,hparams_callback])
4/4 [==============================] - 0s 10ms/step - loss: 0.0059
Epoch 309/500
4/4 [==============================] - 0s 11ms/step - loss: 0.0059
Epoch 310/500
4/4 [==============================] - 0s 9ms/step - loss: 0.0059
```

Figura 25. Configuración de tensoboard y entrenamiendo de modelo

```
#Prediccion
prediccion_original=modelo.predict(x_train)
#prediccion_filtrada=(prediccion_original > 0.7) + 0
prediccion_filtrada=(prediccion_original > 0.7) + 0
print("Entradas: \n",x_train)
print("Respuesta de red: \n",prediccion_original)
print("Predicion hecha : \n",prediccion_filtrada)
```

```
Entradas:
 [[0 0]
 [0 1]
 [1 0]
 [1 1]]
Respuesta de red:
 [[0.10109857]
 [0.96764004]
 [0.9683007 ]
 [0.9994035 ]]
Predicion hecha :
 [[0]
 [1]
 [1]
 [1]]
```

Figura 26. Predicción de la Red multicapa superficial OR con parámetro de aprendizaje de 0.1

```
!tensorboard dev upload --logdir logs \
    --name "Red adaline OR mULTICAPA" \
    --description "(optional) Simple comparison of several hyperparameters"
```

Figura 27. Visualización en Tensorboard de Red multicapa superficial OR

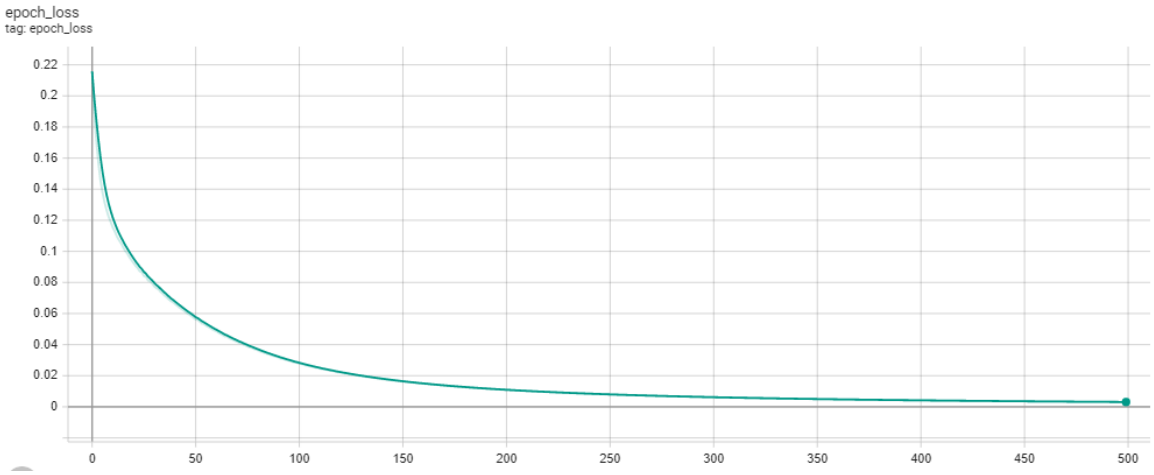


Figura 28. Loss vs epochs con parámetro de aprendizaje de 0.1

Ahora se comienza a modificar solo el learning_rate del codigo con diferentes valores, en este caso con 0.01.

```
[28] def model_or():
       model=Sequential()
       model.add(Dense(16,input_dim=2,activation='relu'))
       model.add(Dense(1,activation='sigmoid'))
       model.summary()
       model.compile(loss='mean_squared_error',optimizer=tf.keras.optimizers.SGD(learning_rate=0.01))
       return model
```

Figura 29. Modificación de parámetro de aprendizaje a 0.01

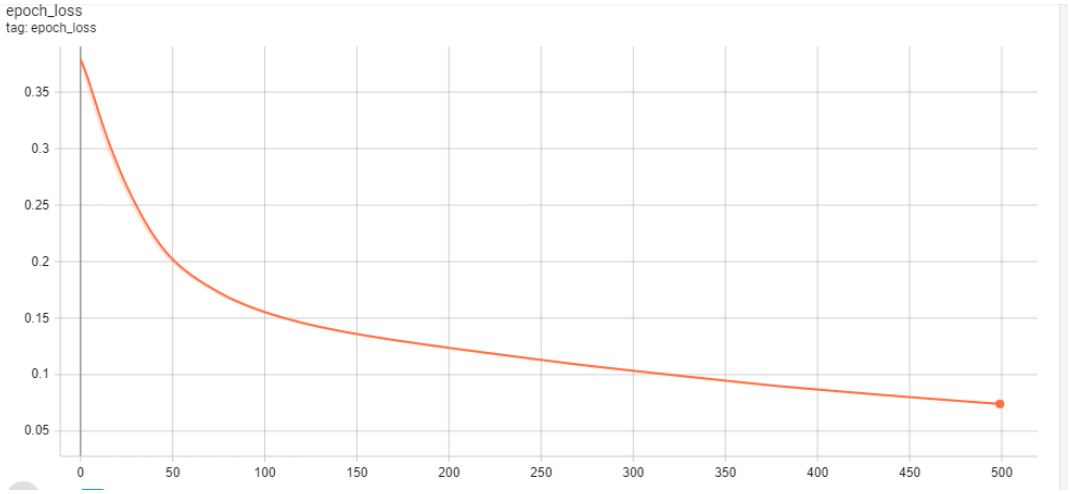


Figura 30. Loss vs epochs con parámetro de aprendizaje de 0.01

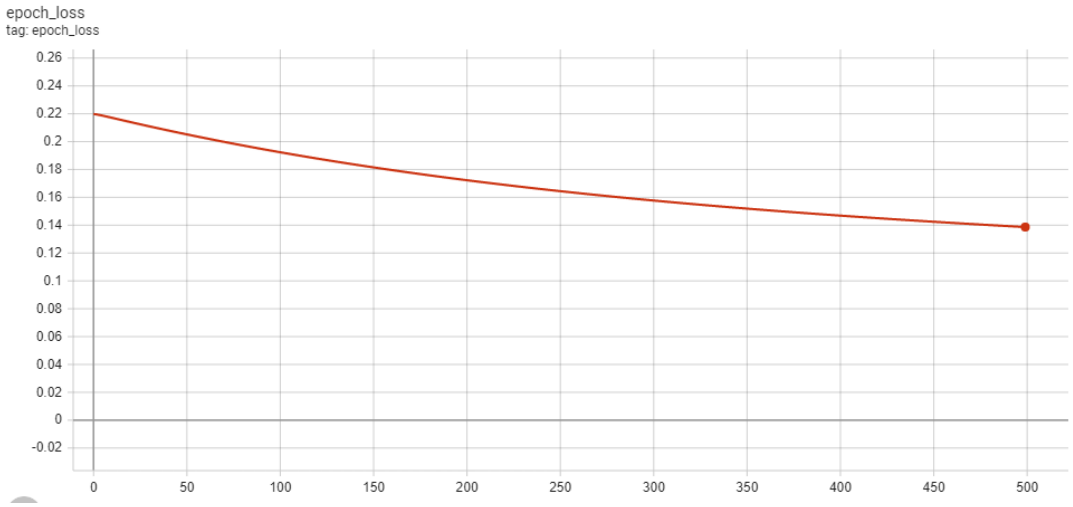


Figura 31. Loss vs epochs con parámetro de aprendizaje de 0.001

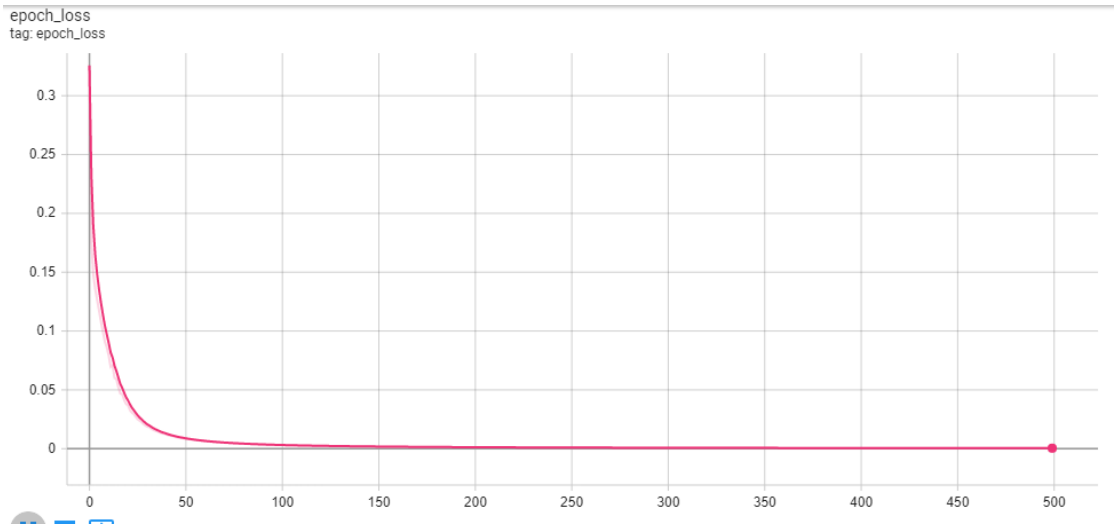


Figura 32 . Loss vs epochs con parámetro de aprendizaje de 0.5

Al observar la figuras de las modificaciones del parámetro de aprendizaje , entre mas pequeño sea el parámetro de aprendizaje genera mayor error , pero si este es demasiado grande puede ocurrir también errores grandes , por lo que se debe tener prioridad a valores cercanos como lo fueron el 0.5 , 0,1 y 0,01 , además comparando la red adaline y la red multicapa superficial se puede decir para este dicho problema de las compuertas OR no es necesario utilizar capas oculta si no la mejor opción de solución de desarrollo es la adaline ,

4. Resuelva el problema de la función lógica XOR (ver la siguiente tabla) usando una red Adaline y una red multicapa superficial en tensorflow-keras. Use tensorboard para visualizar la evolución de la pérdida (loss) y los grafos de las redes neuronales creadas.

X1	X2	D
0	0	0
0	1	1
1	0	1
1	1	0

Red Adaline en XOR , se empieza a poner la librerías necesarias para desarrollar dicha red

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import datetime,os
from tensorboard.plugins.hparams import api as hp
```

Figura 33. importación de librería necesarias para la ejecución de la Red Adaline XOR

```python
def model_xor():
  model=Sequential()
  model.add(Dense(1,input_dim=2,activation='relu'))

  model.summary()
  model.compile(loss='mean_squared_error',optimizer=tf.keras.optimizers.SGD(learning_rate=0.001))
  return model
```

```python
#Ejecución de función
modelo=model_xor()
```

```
Model: "sequential_19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_19 (Dense)            (None, 1)                 3

=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
```

Figura 34. Configuraciones de arquitectura de Red Adaline XOR

```python
#Entrenamento para que tensoboard take data FORMA 1

log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir=log_dir,histogram_freq=1)
hparams_callback = hp.KerasCallback(log_dir,{
    'num_relu_units': 512,
    'dropout': 0.2
})
```

```
history1=modelo.fit(x_train,y_train,epochs=500 , batch_size=1,callbacks=[tensorboard_callback,hparams_callback])
4/4 [==============================] - 0s 8ms/step - loss: 0.2716
Epoch 454/500
4/4 [==============================] - 0s 9ms/step - loss: 0.2716
Epoch 455/500
4/4 [==============================] - 0s 11ms/step - loss: 0.2715
Epoch 456/500
```

Figura 35. Configuración de tensoboard y entrenamiento de modelo para XOR

```python
#Prediccion
prediccion_original=modelo.predict(x_train)
#prediccion_filtrada=(prediccion_original > 0.7) + 0
prediccion_filtrada=(prediccion_original > 0.7) + 0
print("Entradas: \n",x_train)
print("Respuesta de red: \n",prediccion_original)
print("Predicion hecha : \n",prediccion_filtrada)
```

```
Entradas:
 [[0 0]
 [0 1]
 [1 0]
 [1 1]]
Respuesta de red:
 [[0.08775268]
 [0.        ]
 [0.7708285 ]
 [0.14242229]]
Predicion hecha :
 [[0]
 [0]
 [1]
 [0]]
```

Figura 36. Predicción de la Red Adaline XOR

```
!tensorboard dev upload --logdir logs \
    --name "Red adaline" \
    --description "(optional) Simple comparison of several hyperparameters"
```

Figura 37. Visualización en Tensorboard de Red Adaline XOR

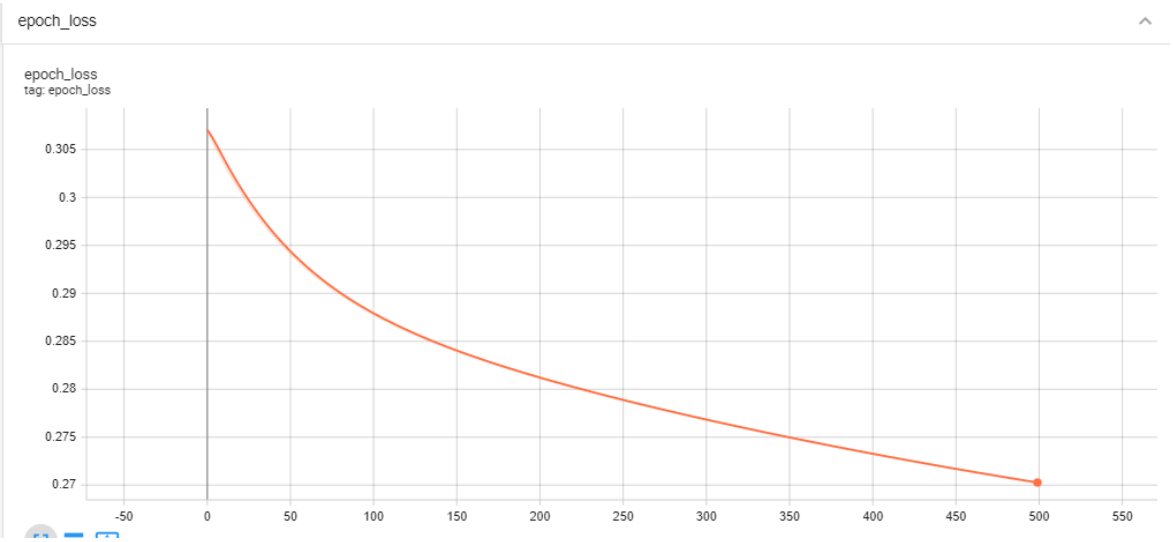


Figura 38. Loss vs epochs de Red Adaline XOR

Al observar la anterior figura se evidencia que al desarrollar una red adaline para la compuerta XOR es muy complejo dado una perdida mayor y haciendo que la predicción no sea exacta; Dejando que la aplicación de la red adaline en este tipo de problema no es debida.

Red multicapa superficial para la compuerta XOR

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import datetime,os
from tensorboard.plugins.hparams import api as hp

Red multicapa en copuerta XOR

[2] x_train=np.array([[0,0],[0,1],[1,0],[1,1]]) #las dos entradas
    y_train=np.array([[0],[1],[1],[0]])#las salida
    #Para saber las dimensiones de la entrada y salidas:
    #entrada_dimension=x_train.shape[1]
    #salida_dimension=y_train.shape[1]

[3] def model_xor():
      model=Sequential()
      model.add(Dense(16,input_dim=2,activation='relu'))
      model.add(Dense(1,activation='sigmoid'))
      model.summary()
      model.compile(loss='mean_squared_error',optimizer=tf.keras.optimizers.SGD(learning_rate=0.1))
      return model
```

Figura 39. Importación de librería necesarias y Configuraciones de arquitectura de Red multicapa XOR

```
[4] #Ejecución de función
    modelo=model_xor()

    Model: "sequential"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #
    =================================================================
     dense (Dense)               (None, 16)                48

     dense_1 (Dense)             (None, 1)                 17

    =================================================================
    Total params: 65
    Trainable params: 65
    Non-trainable params: 0
    _________________________________________________________________

[5] #Entrenamento para que tensoboard take data FORMA 1

    log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
    tensorboard_callback = tf.keras.callbacks.TensorBoard(
        log_dir=log_dir,histogram_freq=1)
    hparams_callback = hp.KerasCallback(log_dir,{
        'num_relu_units': 512,
        'dropout': 0.2
    })
```

```
[6] history1=modelo.fit(x_train,y_train,epochs=500 , batch_size=1,callbacks=[tensorboard_callback,hparams_callback])

    Epoch 472/500
    4/4 [==============================] - 0s 11ms/step - loss: 0.0050
    Epoch 473/500
    4/4 [==============================] - 0s 11ms/step - loss: 0.0049
    Epoch 474/500
    4/4 [==============================] - 0s 12ms/step - loss: 0.0049
    Epoch 475/500
    4/4 [==============================] - 0s 12ms/step - loss: 0.0049
    Epoch 476/500
    4/4 [==============================] - 0s 11ms/step - loss: 0.0049
    Epoch 477/500
    4/4 [==============================] - 0s 14ms/step - loss: 0.0049
    Epoch 478/500
    4/4 [==============================] - 0s 10ms/step - loss: 0.0049
    Epoch 479/500
    4/4 [==============================] - 0s 10ms/step - loss: 0.0049
    Epoch 480/500
    4/4 [==============================] - 0s 12ms/step - loss: 0.0048
    Epoch 481/500
    4/4 [==============================] - 0s 13ms/step - loss: 0.0048
```

Figura 40. Configuración de tensoboard y entrenamiento de modelo multicapa para XOR

```
[7] #Prediccion
    prediccion_original=modelo.predict(x_train)
    #prediccion_filtrada=(prediccion_original > 0.7) + 0
    prediccion_filtrada=(prediccion_original > 0.7) + 0
    print("Entradas: \n",x_train)
    print("Respuesta de red: \n",prediccion_original)
    print("Predicion hecha : \n",prediccion_filtrada)

    Entradas:
     [[0 0]
     [0 1]
     [1 0]
     [1 1]]
    Respuesta de red:
     [[0.10680494]
     [0.95078754]
     [0.9543013 ]
     [0.04607353]]
    Predicion hecha :
     [[0]
     [1]
     [1]
     [0]]
```

Figura 41. Predicción de la Red multicapa XOR

```
!tensorboard dev upload --logdir logs \
    --name "(Nombre_tensor) My latest experiment" \
    --description "(optional) Simple comparison of several hyperparameters"
```

Figura 42. Visualización en Tensorboard de Red multicapa XOR

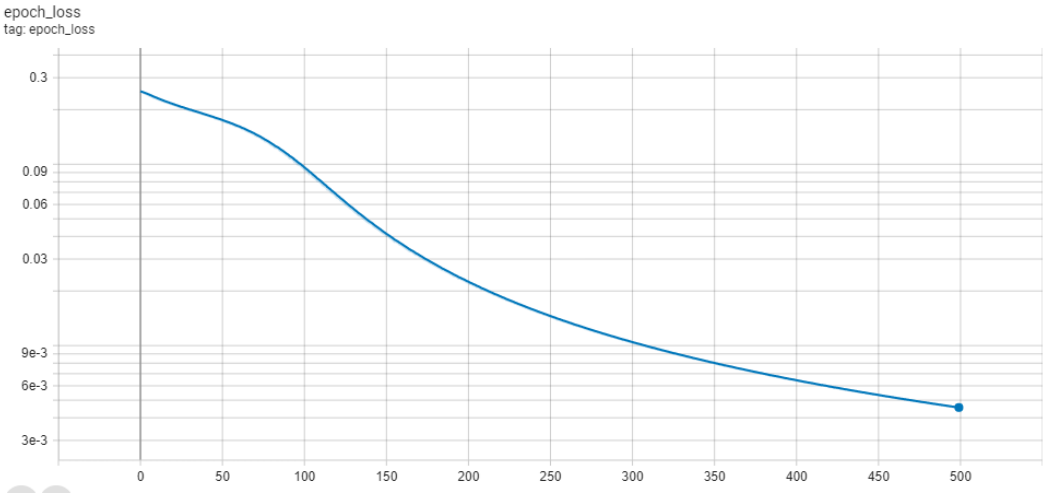


Figura 43. Loss vs epochs de Red multicapa XOR

Al observar la anterior figura demuestra la predicción y exactitud de esta red multicapa , prediciendo perfectamente los resultados y dando a tener en cuenta que este tipo de problema como lo es la compuerta XOR es de un grado de complejidad mas alto que la compuerta OR por lo que a utilizar capas ocultas disminuye inmensamente el error dando el resultado peregrinamente.

5. Use el playground de tensorflow (https://playground.tensorflow.org/) y verifique con los data set "*circle*" y "*spiral*" el efecto de ir aumentado las capas en una red neuronal. Comienze con una red de una sola capa de procesamiento y vaya agregando capas ocultas y/o neuronas hasta resolver el problema convenientemente. Documento el proceso realizado

En la figura 44 se observa que hay 2 entradas , con dos neuronas en dos capas oculta , en el cual se esta utilizando una tasa de aprendizaje del 0.03 y función de activación tangencial hiperbólica simbolizada con (tanh) , en tipos de problemas de clasificación .

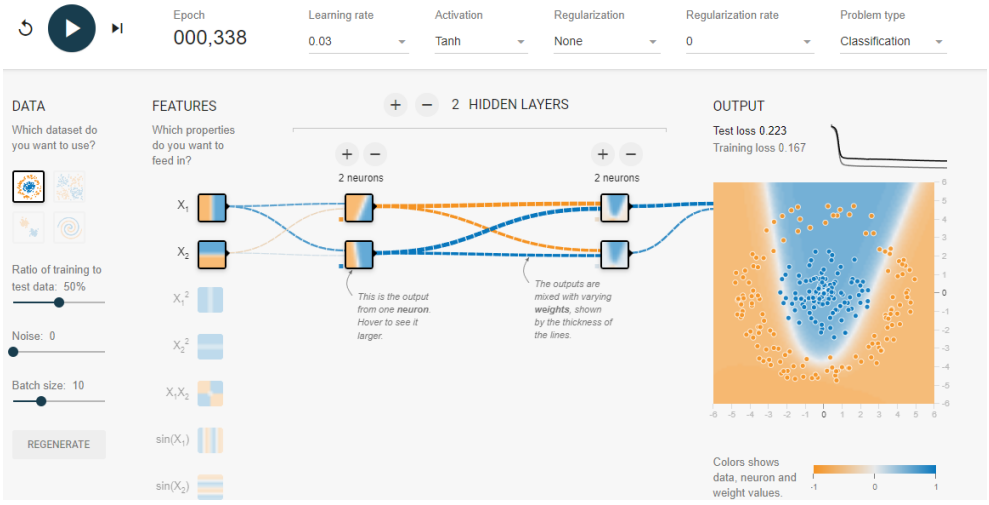


Figura 44. Data set circle con dos capas oculta.

Aumentando la capas ocultas y numero de neuronas se clasifica de manera mas acertada que con dos capas , es decir aumenta la precisión de la clasificación , tal como se observa en la figura 45.

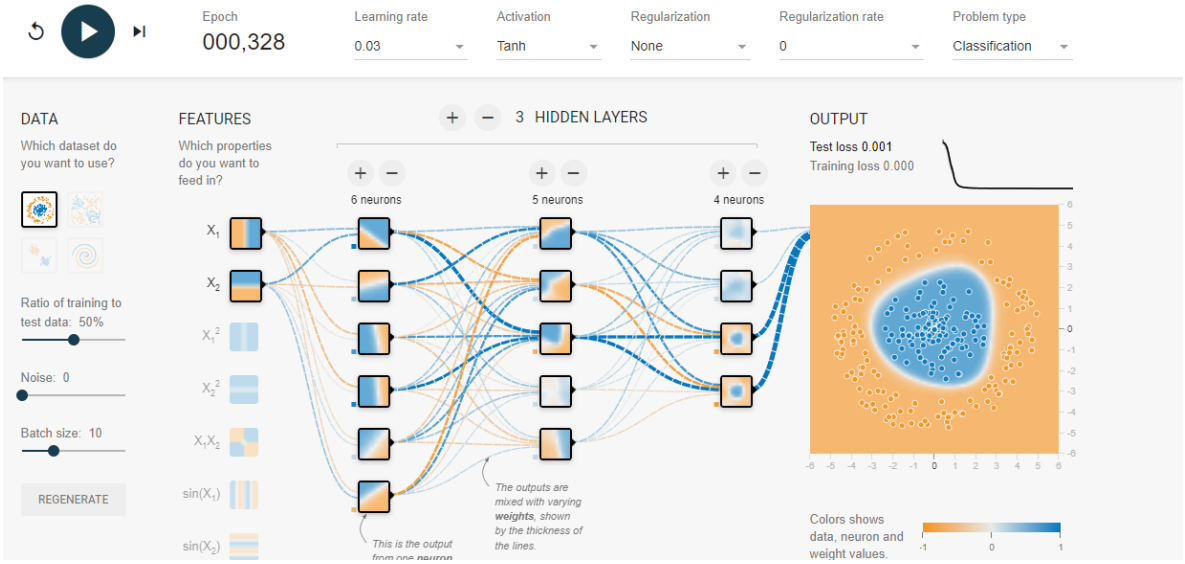


Figura 45. Data set circle con cuatro neuronas capa oculta.

Al mostrar el data set del spiral los puntos azules y amarillos esta deforma dispersa muy diferente como lo estaba en el data set de circle , por que las capas y numero de neuronas deben cambiar debido a la diferente aplicación. Se realizó el mismo tasa de aprendizaje y función de activación que con la hecha en l data set de circle.

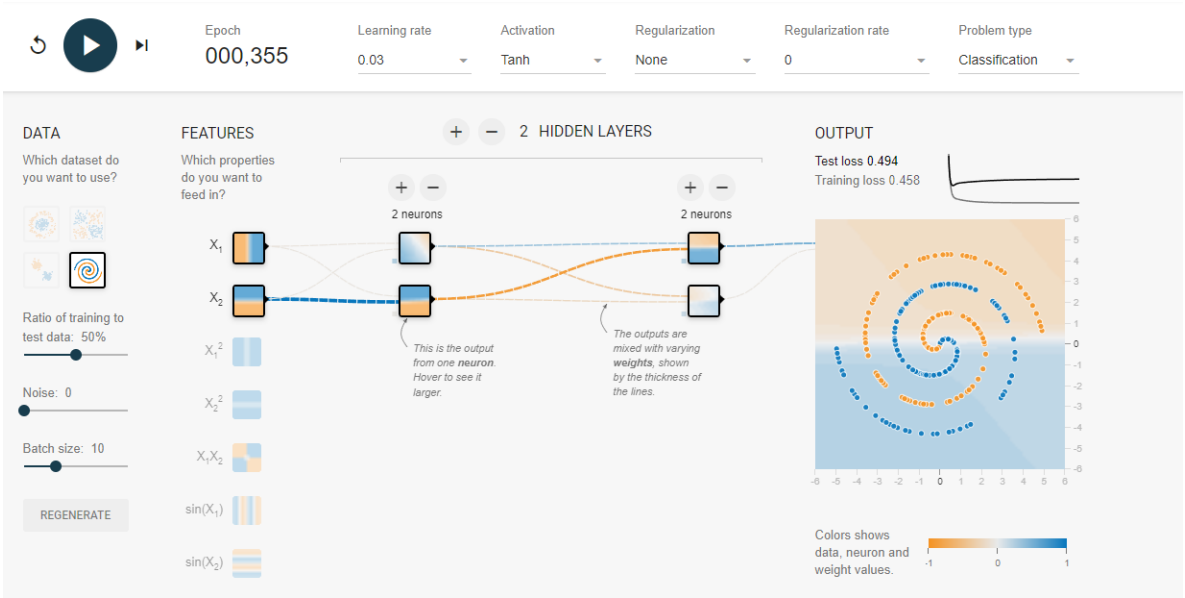


Figura 46. Data set spiral con dos neuronas en dos capas ocultas

Al aumentar la capas ocultas y el numero de neuronas en esas capas , en el mismo tiempo de acción no alcanza para clasificar los elementos, se necesita mas tiempo debido a que las entradas son lineales para conseguir dicha clasificación.

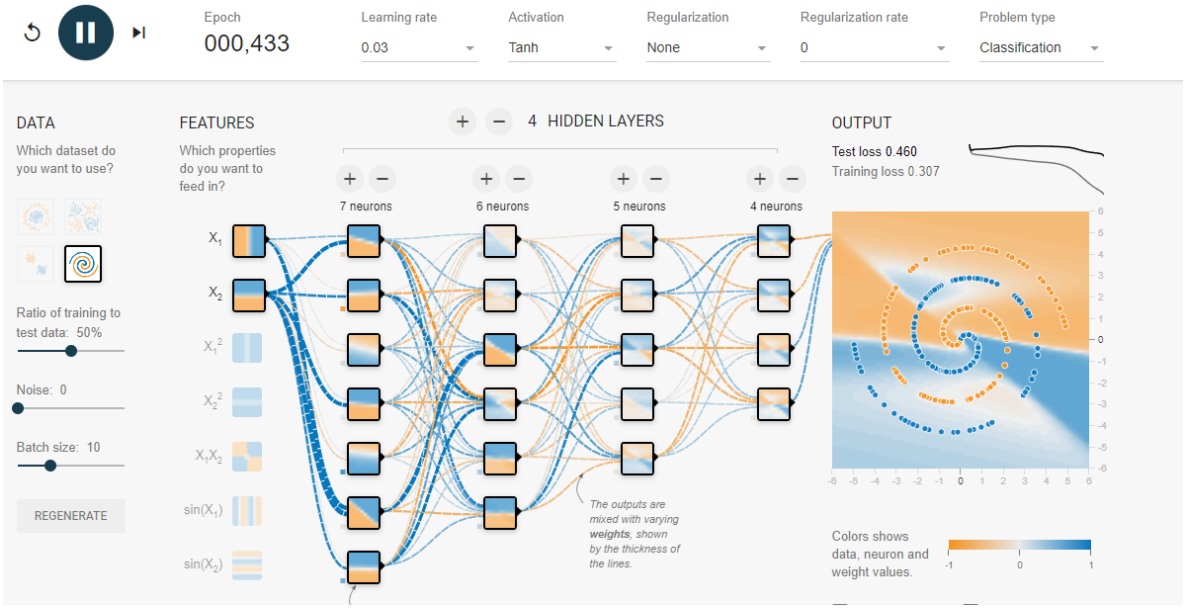


Figura 47. Data set spiral con aumento de capas ocultas y numero de neuronas.

Con usar entradas no lineales se puede mejorar la clasificación en un menor tiempo y menor numero de neuronas , también hay que tener en cuenta la función de activación de aplicar debido a que esto mejorara la clasificación del data set , como se observa en la figura 49 .

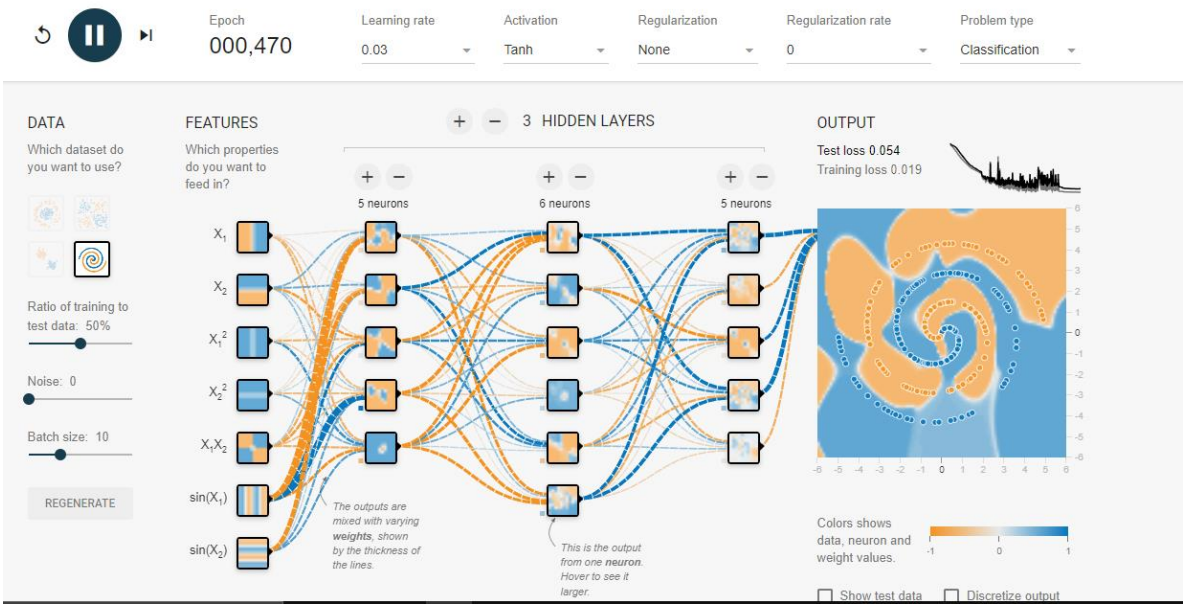


Figura 48. Data set spiral con entradas no lineales y lineales

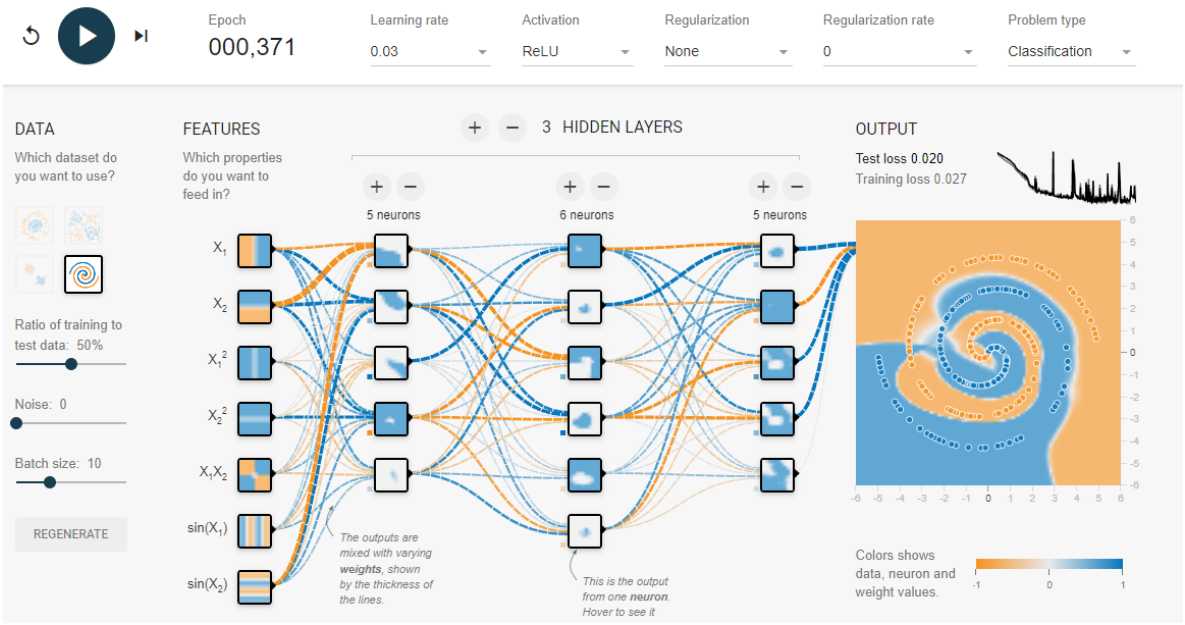


Figura 49. Data set spiral con función activación RELU

6. Resuelva en tensorflow-keras el problema de clasificación de datos circulares y de espiral trabajados en playground de tensorflow en el punto anterior. (https://playground.tensorflow.org/)
Use tensorboard para visualizar la evolución de la pérdida (loss) y los grafos de las redes neuronales creadas.

Se inició con el dataset del circle

```
from sklearn.datasets import make_circles
from keras.models import Sequential
from keras.layers import Dense
from matplotlib import pyplot
from numpy import where
```

Figura 50. Importaciones de librerías necesarias dataset circle

```
# generate 2d classification dataset
X, y = make_circles(n_samples=1000, noise=0.1, random_state=1 ,factor=0.1)
# scatter plot, dots colored by class value
for i in range(2):
  samples_ix = where(y == i)
  pyplot.scatter(X[samples_ix, 0], X[samples_ix, 1])
pyplot.show()
```

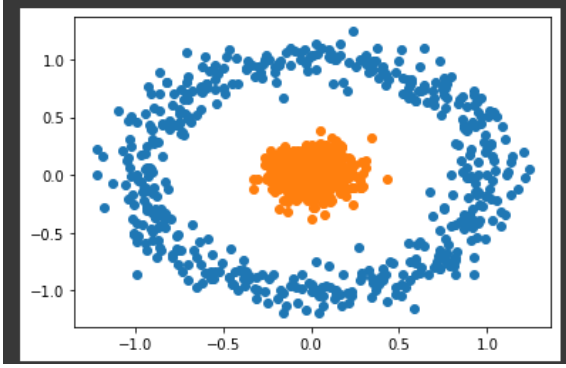


Figura 51. Configuraciones y generación de puntos de circulo 2d, dataset circle

```
# split into train and test
n_test = 500
trainX, testX = X[:n_test, :], X[n_test:, :]
trainy, testy = y[:n_test], y[n_test:]
```

Figura 52. Separación de datos para entrenamiento y testeo , dataset circle

```
[28] # define model
     model = Sequential()
     model.add(Dense(100, input_shape=(2,), activation='relu'))
     model.add(Dense(1, activation='sigmoid'))

[29] # compile model
     model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
     # fit model
     history = model.fit(trainX, trainy, validation_data=(testX, testy), epochs=300, verbose=0)
```

Figura 53. Definición del modelo y ajuste para entrenamiento, dataset circle

Se evalúa el modelo, informando la precisión de clasificación en el tren y conjuntos de prueba de alrededor en donde se observa que dio 100% en ambos.

```
[30] # evaluate the model
     _, train_acc = model.evaluate(trainX, trainy, verbose=0)
     _, test_acc = model.evaluate(testX, testy, verbose=0)
     print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))

     Train: 1.000, Test: 1.000
```

Figura 54. Evaluación del modelo, dataset circle

Después de la evaluación del modelo indica el buen porcentaje de satisfacción para el aprendizaje , para observar el los y la acuracy se grafican

```
# plot loss during training
pyplot.subplot(211)
pyplot.title('Loss')
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
# plot accuracy during training
pyplot.subplot(212)
pyplot.title('Accuracy')
pyplot.plot(history.history['accuracy'], label='train')
pyplot.plot(history.history['val_accuracy'], label='test')
pyplot.legend()
pyplot.show()
```

Figura 55. Código de graficar los y acurracy, dataset circle

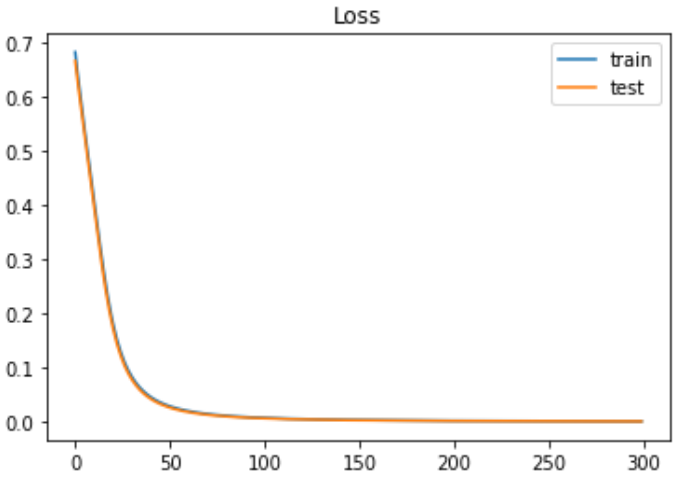


Figura 56. Grafica los vs epochs, dataset circle

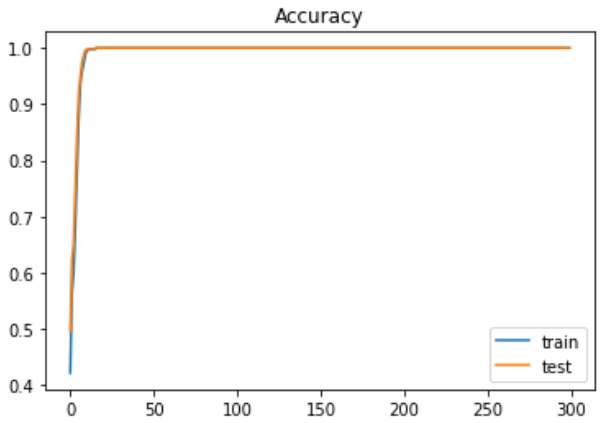


Figura 57. Grafica accuracy vs epochs, dataset circle

Al observar los anteriores gráficos se precisa que la elección de la arquitectura del modelo fue la ideal para resolver el problema dando que el valor del aprendizaje sea muy bueno en el transcurso de las épocas y no desarrollarse un overfitting que puede ser probable en dichos problemas o al momento de realizar aprendizajes.

Ahora se realiza el proceso con el spiral dataset

```
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense

def twospirals(n_points, noise=.5):
    """
     Returns the two spirals dataset.
    """
    n = np.sqrt(np.random.rand(n_points,1)) * 780 * (2*np.pi)/360
    d1x = -np.cos(n)*n + np.random.rand(n_points,1) * noise
    d1y = np.sin(n)*n + np.random.rand(n_points,1) * noise
    return (np.vstack((np.hstack((d1x,d1y)),np.hstack((-d1x,-d1y)))),
            np.hstack((np.zeros(n_points),np.ones(n_points))))

X, y = twospirals(1000)

plt.title('training set')
plt.plot(X[y==0,0], X[y==0,1], '.', label='class 1')
plt.plot(X[y==1,0], X[y==1,1], '.', label='class 2')
plt.legend()
plt.show()
```

Figura 58. Importe de librerías y creación de spiral dataset

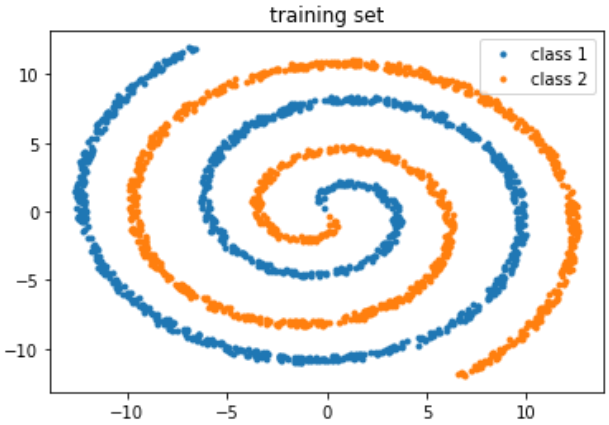


Figura 59. Visualización de spiral data set

```
mymlp = Sequential()
mymlp.add(Dense(12, input_dim=2, activation='tanh'))
mymlp.add(Dense(1, activation='sigmoid'))

mymlp.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

# trains the model
mymlp.fit(X, y, epochs=150, batch_size=10,  verbose=0)
```

Figura 60. Creación de arquitectura y entrenamiento del modelo

```
X_test, y_test = twospirals(1000)

yy = np.round(mymlp.predict(X_test).T[0])

plt.subplot(1,2,1)
plt.title('training set')
plt.plot(X[y==0,0], X[y==0,1], '.')
plt.plot(X[y==1,0], X[y==1,1], '.')
plt.subplot(1,2,2)
plt.title('Neural Network result')
plt.plot(X_test[yy==0,0], X_test[yy==0,1], '.')
plt.plot(X_test[yy==1,0], X_test[yy==1,1], '.')
plt.show()
```

Figura 61. Código de visualización de comparación del training set y el resultado de la red neuronal

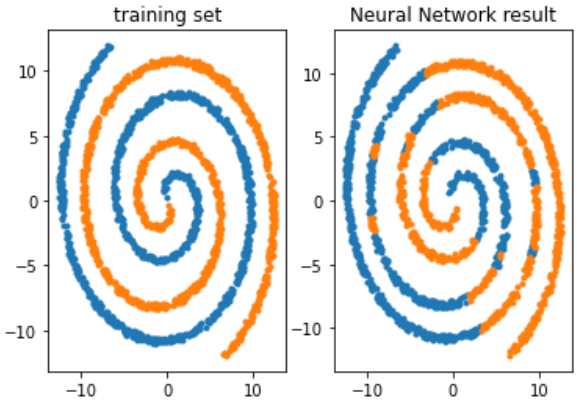


Figura 62. Resultado de la red neuronal entrenada con una capa oculta de 12 neuronas

Contemplando la anterior figura la red neuronal entrenada no es lo suficiente buena para resolver le problema de clasificar los puntos , esto es debido a que las capa ocultas y numero de neuronas no son lo suficientes para abordarlo , por eso se procede aumentar el numero de capas ocultas y numero de neuronas

```
mymlp = Sequential()
mymlp.add(Dense(12, input_dim=2, activation='tanh'))
mymlp.add(Dense(12, activation='tanh'))
mymlp.add(Dense(12, activation='tanh'))
mymlp.add(Dense(1, activation='sigmoid'))

mymlp.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

# Fit the model
history=mymlp.fit(X, y, epochs=150, batch_size=10,  verbose=0)

yy = np.round(mymlp.predict(X_test).T[0])

plt.subplot(1,2,1)
plt.title('training set')
plt.plot(X[y==0,0], X[y==0,1], '.')
plt.plot(X[y==1,0], X[y==1,1], '.')
plt.subplot(1,2,2)
plt.title('Neural Network result')
plt.plot(X_test[yy==0,0], X_test[yy==0,1], '.')
plt.plot(X_test[yy==1,0], X_test[yy==1,1], '.')
plt.show()
```

Figura 63. Aumento de numero de capas ocultas y numero de neuronas

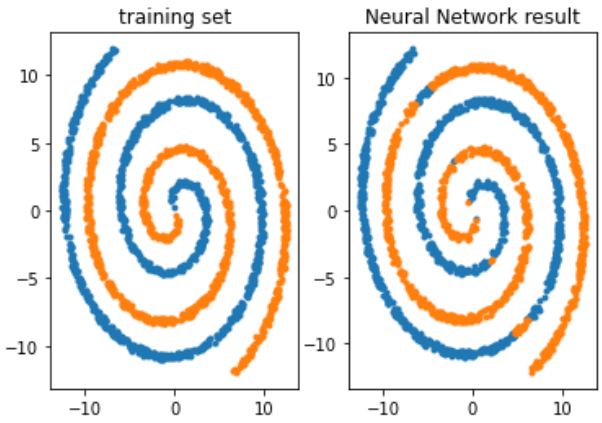


Figura 64. Resultado de la red neuronal entrenada con tres capas ocultas de 12 neuronas

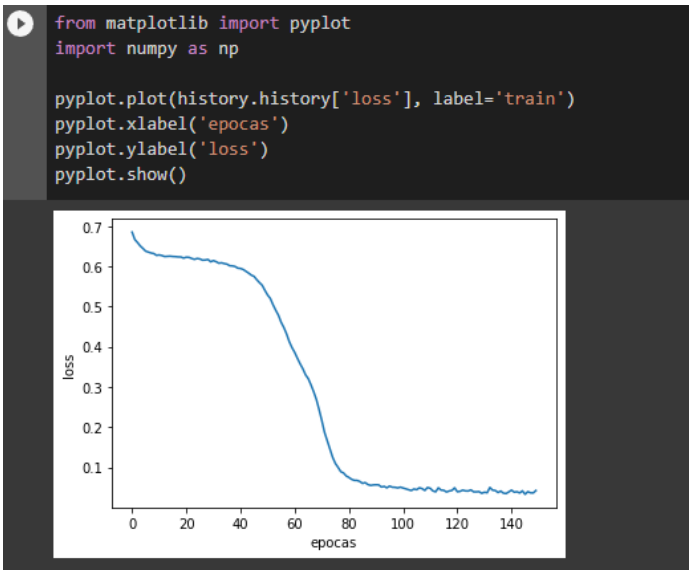


Figura 65. Grafica de los vs epochs

La anterior figura se denota la solución abarcada al data set spiral de forma satisfactoria debido a que el los frente a 150 épocas realizadas presento un loss muy bajo , dando un entrenamiento ideal para la solución del entrenamiento. Por lo que se puede fijar como es por dentro de forma de keras lo que se realizaba en el punto 5 ya realizado

7. Realice una comparación al aplicar gradiente descendente convencional y gradiente descente con momentum usando simulaciones que se pueden realizar usando el siguiente enlace (https://www.deeplearning.ai/ai-notes/optimization/)
Utilice por lo menos dos paisajes de costo (*cost landscape*) y modifique los parámetros de entrenamiento. Documente el proceso realizado

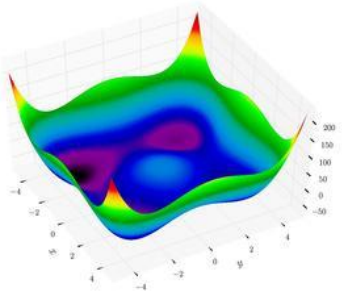


Figura 66. Paisaje seleccionado de costo 1

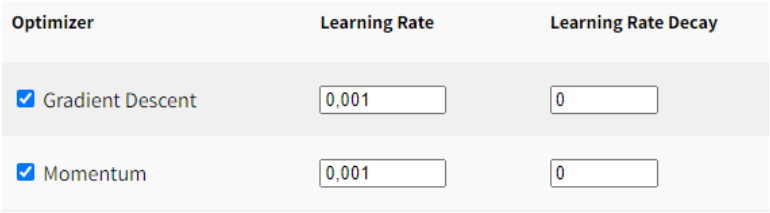


Figura 67. Tasa de aprendizaje de 0.001 en descenso de gradiente y momentum, paisaje costo 1

Para el momentum (impulso) y el descenso de gradiente se usó una tasa de aprendizaje de 0.001 , debido al momento se dieron menos épocas para que se estabilice la red y entrenarla, como se observa en la siguiente figura

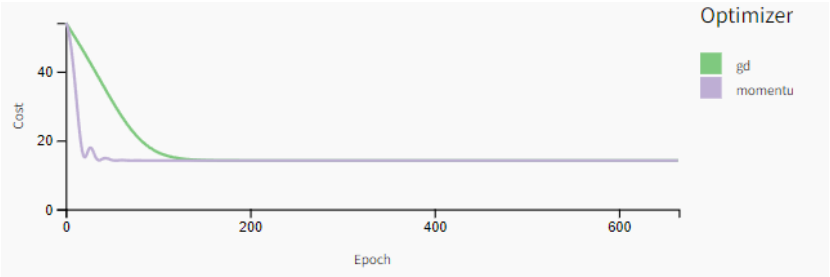


Figura 68. Cost vs epoch de descenso de gradiente y momentum, paisaje de costo 1

Para visualizar una mejor forma el comportamiento de la acción del desplazamiento frente a las diferentes parámetros de w1,w2 , está presente en la consecutiva figura

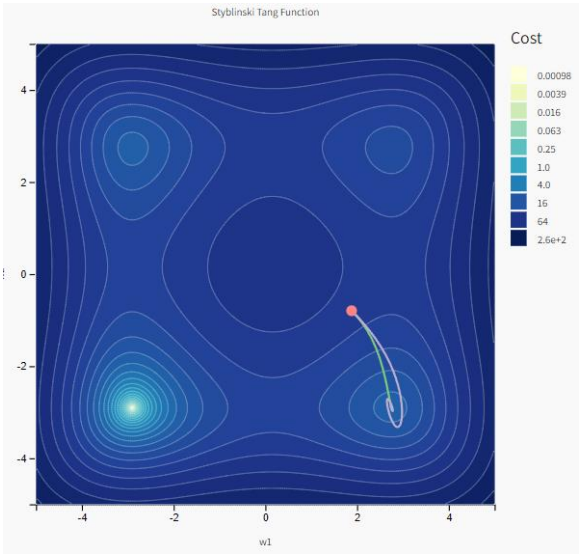


Figura 69. Simulación 2D de Costos según parámetros (w1,w2) , paisaje costo 1

Optimizer	Learning Rate	Learning Rate Decay
☑ Gradient Descent	0,004	0
☑ Momentum	0,004	0

Figura 70. Tasa de aprendizaje de 0.004 en descenso de gradiente y momentum, paisaje costo 1

Al tener las diferentes figuras, para analizar el comportamiento del paisaje de costo 1 se aumenta la tasa de aprendizaje a 0.004 en el descenso de gradiente y el momentum , en el que el cambio no es evidenciable por lo que proporcionaron las mismas épocas para que se estabilizara cada uno de los parámetros , esto se refleja en la siguiente figura.

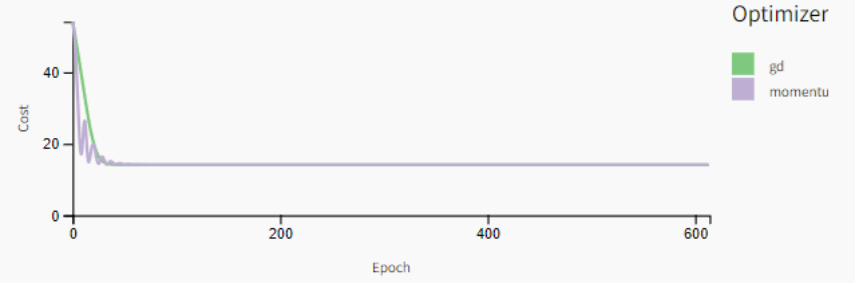


Figura 71. Cost vs epoch de descenso de gradiente y momentum, paisaje de costo 1 , tasa de aprendizaje de 0.004

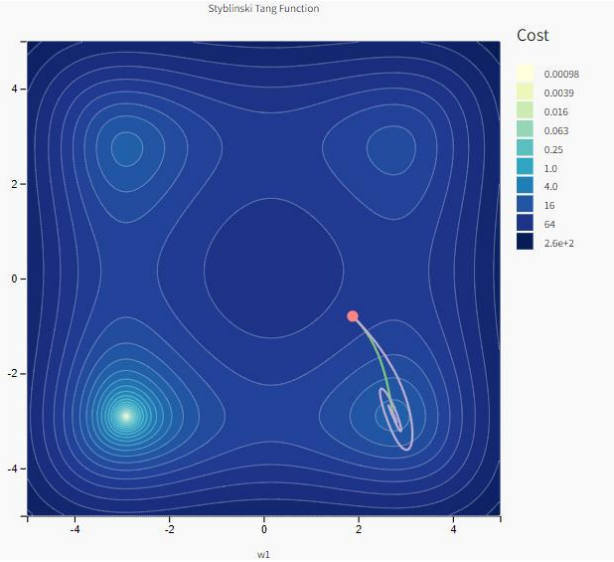


Figura 72. Simulación 2D de Costos según parámetros (w1,w2) con tasa de aprendizaje de 0.004 , paisaje costo 1

De forma de análisis se dispone dar selección de un diferente paisaje

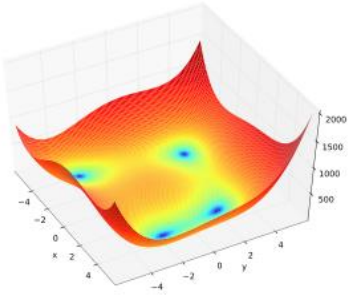


Figura 73. Paisaje seleccionado de costo 2

Figura 74. Tasa de aprendizaje de 0.004 en descenso de gradiente y momentum, paisaje costo 2

Al proporcionar una tasa de aprendizaje de 0.004 ubicada para el descenso de gradiente y el momentum(impulso) se demandaron menos épocas para que se estabilizara la red y que se entrenara, pero esto es debido a un paisaje diferente, por lo que proporcionado esa tasa de aprendizaje de 0.004 fueron parejas, y tuvo menos tiempo el momentum para que se estabilizara en comparación con el descenso de gradiente.

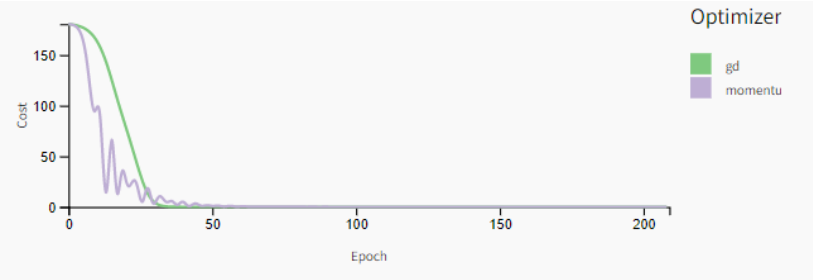


Figura 75. Cost vs epoch de descenso de gradiente y momentum, paisaje de costo 2 , tasa de aprendizaje de 0.004

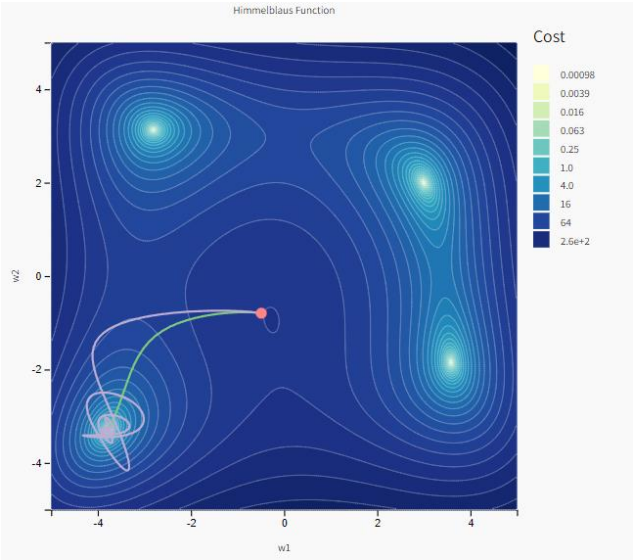


Figura 76. Simulación 2D de Costos según parámetros (w1,w2) con tasa de aprendizaje de 0.004 , paisaje costo 2

Al tener las diferentes figuras, para analizar el comportamiento del paisaje de costo 2 se aumenta la tasa de aprendizaje a 0.01 en el descenso de gradiente y el momentum.

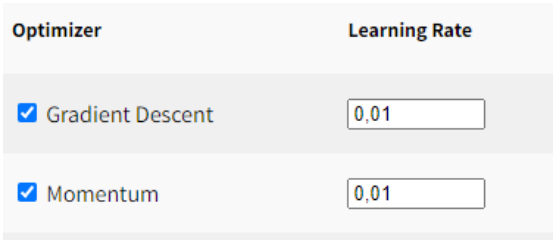


Figura 77. Tasa de aprendizaje de 0.01 en descenso de gradiente y momentum, paisaje costo 2

Prestando atención en la modificación el momentum adquirió más épocas para poder estabilizar la red y entrenarla , comparado con el descenso de gradiente; Se podría decir que en este tipo de paisaje al aumentar la tasa de aprendizaje en el descenso de gradiente se disminuye las épocas de entrenamiento.

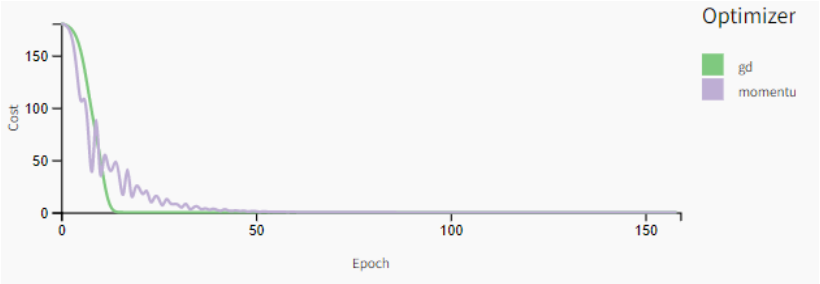


Figura 78. Cost vs epoch de descenso de gradiente y momentum, paisaje de costo 2 , tasa de aprendizaje de 0.01

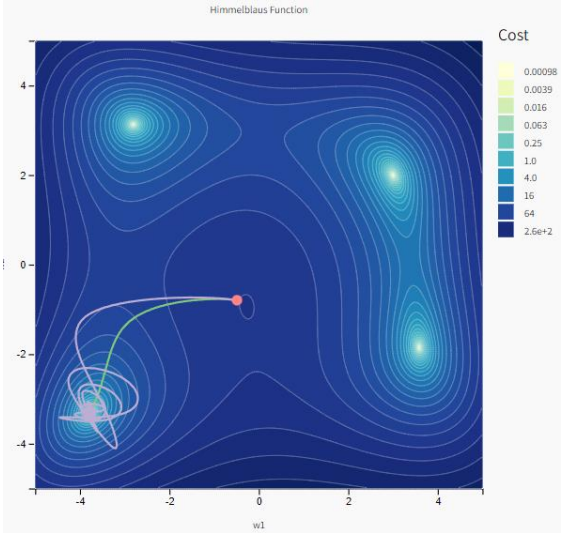


Figura 79. Simulación 2D de Costos según parámetros (w1,w2) con tasa de aprendizaje de 0.01 , paisaje costo 2

8. Realice una breve investigación sobre qué es la diferenciación automática y muestre un ejemplo de cómo se puede implementar para entrenar redes neuronales artificiales
   Nota: Se sugiere buscar el término en inglés *"automatic differentiation"*

Diferenciación automática (AD) es un término usado en general describir las diversas técnicas numéricas para calcular las derivadas de una función de una o más variables. Mientras que las reglas para la diferenciación son sencillos de entender, su implementación a mano es a menudo requiere mucho tiempo y es propenso a errores. Las técnicas de AD ayudan a superar ambos estos inconvenientes y ofrecer al usuario involucrado en el modelado no lineal y optimización la promesa de información derivada precisa y correcta con un esfuerzo aparentemente mínimo.[9]

Por lo que la diferenciación automática es un método entre la diferenciación simbólica y la diferenciación numérica; La diferenciación numérica calculada una aproximación a la derivada de una función en un punto utilizando los valores y propiedades de la misma

La diferenciación simbólica esta dado en el algebra como resolución directamente y se sustituye en el valor del problema , por lo que la diferenciación automática aplica la diferenciación simbólica a

los operadores como constantes , funciones de potencia , funciones exponenciales , funciones logarítmicas , funciones trigonométricas entre otros ; Siendo esta la diferentes foras mas básicas , después se sustituyen en valores numéricos manteniendo los resultados intermedios y finalmente se aplican a toda la función ; Por lo tanto su aplicación es bastante flexible y puede ocultar completamente a los usuarios el proceso de resolución diferencial. Puesto a que se aplica la ley de diferenciación simbólica a funciones basias o constantes , puede combinar de manera flexible la estructura condicional de l leguaje de programación , utilizando diferenciación automática y debido a que su calculo s grafico , se puede optimizar bastante siendo la razón por lo que se usa ampliamente en los sistemas modernos de aprendizaje profundo , una clara idea de saber sobre esta tema y de su funcionamiento.

Hay dos enfoques de programación diferentes para diferenciar código: sobrecarga de operadores y transformación fuente a fuente. En el enfoque de sobrecarga del operador, las operaciones aritméticas básicas y A las funciones se les asignan rutinas que calculan las derivadas de los operadores. salidas además del cálculo del valor de la función. el código fuente de la función se diferencia progresivamente llamando a estas rutinas al mismo tiempo que se realiza cada operación en la evaluación del programa. Existen dos enfoques para la diferenciación de programas: reenvío automático diferenciación automática (FAD) y diferenciación automática inversa (RAD).

La diferenciación automática inversa (RAD) está relacionada matemáticamente con el análisis de sensibilidad adjunto para ecuaciones diferenciales, cuyo uso también se remonta a la década de 1960. El método fue entonces ampliamente utilizado en ingeniería nuclear. [10] y predicción meteorológica [11], por ejemplo. El algoritmo RAD funciona en términos del gráfico computacional asociado con la evaluación de un función. Al igual que FAD, para implementar el algoritmo RAD requerimos la declaración de una nueva variable escalar conocida como la variable adjunta $\bar{t}$ para cada nodo del gráfico computacional.

Ventajas de FAD

Como FAD, RAD proporciona derivadas matemáticamente precisas. El algoritmo RAD es eficiente en términos del número de pasos computacionales. Esto hace que RAD sea especialmente adecuado para problemas con un gran número de variables independientes. El método RAD también es eficiente con respecto a los requisitos de memoria. En particular, el requisito de memoria es independiente del número de variables.

Desventajas de FAD

El algoritmo RAD requiere la modificación del código fuente por parte del usuario. De manera similar a FAD, la modificación solo implica cambiar la parte declarativa del código fuente, mientras que el resto del código permanece sustancialmente igual. Para implementar RAD, el usuario también debe declarar un nuevo tipo de variable y los operadores de sobrecarga asociados con este nuevo tipo. Las implementaciones del algoritmo RAD implican listas enlazadas, árboles binarios, punteros y una cuidadosa gestión de la memoria. Requiere un lenguaje que admita la sobrecarga de operadores, como Fortran 90.

Ejemplo de aplicación: estimación de máxima verosimilitud

Los problemas de mínimos cuadrados no lineales ponderados surgen cuando las incertidumbres S_i en los datos de entrada se han caracterizado adecuadamente. Si solo tenemos estimaciones aproximadas si para σi , eso puede ser apropiado considerar las σi como incógnitas pero para las cuales tenemos algún modelo estadístico. Por ejemplo, supongamos que el modelo es:

$$y_i \in N(\phi(x_i, \mathbf{a}), \sigma_i^2), \quad \log s_i^2 \in N(\log \sigma_i^2, \rho_i^2)$$

donde ρi es conocida. La estimación de máxima verosimilitud de σ = (σ1, . . . , σm) T y a se determina minimizando una función logarítmica de verosimilitud de la forma

$$\begin{aligned} F(\sigma, \mathbf{a}) &= \sum_{i=1}^{m} \log \sigma_i^2 + \sum_{i=1}^{m} \frac{1}{2} \left(\frac{y_i - \phi_i(x_i, \mathbf{a})}{\sigma_i} \right)^2 \\ &+ \sum_{i=1}^{m} \frac{1}{2} \left(\frac{\log s_i^2 - \log \sigma_i^2}{\rho_i} \right)^2 . \end{aligned}$$

Software de optimización como el componente E04LBF en la biblioteca NAG [12] requiere que calculemos el vector gradiente de las primeras derivadas y el Matriz hessiana de segundas derivadas de F con respecto a σ ya. Nosotros han calculado estos utilizando el algoritmo RAD implementado en Fortran 90.

Bibliografía

[1]"Una sencilla Red Neuronal en Python con Keras y Tensorflow | Aprende Machine Learning", *Aprendemachinelearning.com*, 2022. [Online]. Available: https://www.aprendemachinelearning.com/una-sencilla-red-neuronal-en-python-con-keras-y-tensorflow/. [Accessed: 28- Aug- 2022]

[2]"Keras: mi primer modelo de clasificación binaria de redes neuronales - programador clic", *Programmerclick.com*, 2022. [Online]. Available: https://programmerclick.com/article/83031913685/. [Accessed: 28- Aug- 2022]

[3]"Deep Learning với Tensorflow Module 2 Phần 1: Xây dựng mô hình phân loại trong Neural network - Mô hình 2 class (binary class)", *Mvt-blog.com*, 2022. [Online]. Available: https://mvt-blog.com/posts/deep-learning-with-tensorflow-module-3-part-1-neural-network-classification. [Accessed: 28- Aug- 2022]

[4]2022. [Online]. Available: https://machinelearningmastery.com/impact-of-dataset-size-on-deep-learning-model-skill-and-performance-estimates/. [Accessed: 28- Aug- 2022]

[5]2022. [Online]. Available: https://machinelearningmastery.com/how-to-calculate-precision-recall-f1-and-more-for-deep-learning-models/. [Accessed: 28- Aug- 2022]

[6]*Youtube.com*, 2022. [Online]. Available: https://www.youtube.com/watch?v=HMsMUg1BzzY. [Accessed: 28- Aug- 2022]

[7]"Solving the Two Spirals problem with Keras", *Glowingpython.blogspot.com*, 2022. [Online]. Available: https://glowingpython.blogspot.com/2017/04/solving-two-spirals-problem-with-keras.html. [Accessed: 28- Aug- 2022]

[8]H. shape?, S. Nanda, F. Dernoncourt and H. Du, "How to classify data which is spiral in shape?", *Cross Validated*, 2022. [Online]. Available: https://stats.stackexchange.com/questions/235600/how-to-classify-data-which-is-spiral-in-shape. [Accessed: 28- Aug- 2022]

[9]*Eprintspublications.npl.co.uk*, 2022. [Online]. Available: https://eprintspublications.npl.co.uk/2828/1/cmsc26.pdf. [Accessed: 28- Aug- 2022]

[10]D. G. Cacuci. Sensitivity theory for nonlinear systems I: nonlinear functional analysis approach. Journal of Mathematical Physics, 22(12):2794–2802, 1981.

[11]I. M. Navon and U. Muller. FESW - a finite-element Fortran IV program for solving the shallow water equation. Advances in Engineering Software, 1:77–84, 1970.

[12]The Numerical Algorithms Group Limited, Wilkinson House, Jordan Hill Road, Oxford, OX2 8DR. The NAG Fortran Library, Mark 20, Introductory Guide, 2002. http://www.nag.co.uk/