COLOMBIA POTENCIA DE LA VIDA

TIC

Actividad práctica

Programación de redes neuronales

TALENTO TECH

AZ | PROYECTOS EDUCATIVOS

Universidad Tecnológica de Pereira

Programación de redes neuronales

En este proyecto veremos tres formas de programar redes neuronales, usando:

TensorFlow **Keras** **Sklearn**

Paso 1: Configurando el Proyecto

Estos códigos son ejecutables en GoogleColab, pero si desea ejecutarlos en su computador debe configurar el Proyecto. Antes de que pueda desarrollar el programa de reconocimiento, deberá instalar algunas dependencias y crear un espacio de trabajo para guardar sus archivos.

Requirements.txt

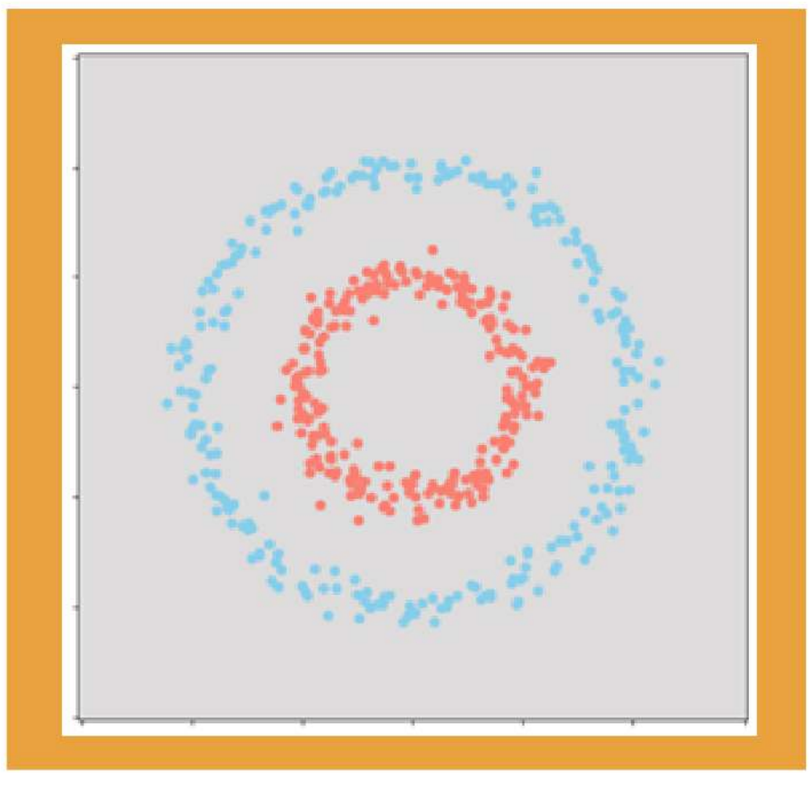
- image==1.5.20.
- numpy==1.14.3.
- tensorflow==1.4.0.

pip install -r requirements.txt

Paso 2: Importando el conjunto de datos

```
import numpy as np
import scipy as sc
import matplotlib.pyplot as plt

from sklearn.datasets import make_circles

# Creamos nuestros datos artificiales, donde buscaremos clasificar
# dos anillos concéntricos de datos.
X, Y = make_circles(n_samples=500, factor=0.5, noise=0.05)

# Resolución del mapa de predicción.
res = 100

# Coordendadas del mapa de predicción.
_x0 = np.linspace(-1.5, 1.5, res)
_x1 = np.linspace(-1.5, 1.5, res)

# Input con cada combo de coordenadas del mapa de predicción.
_pX = np.array(np.meshgrid(_x0, _x1)).T.reshape(-1, 2)

# Objeto vacio a 0.5 del mapa de predicción.
_pY = np.zeros((res, res)) + 0.5

# Visualización del mapa de predicción.
plt.figure(figsize=(8, 8))
plt.pcolormesh(_x0, _x1, _pY, cmap="coolwarm", vmin=0, vmax=1)

# Visualización de la nube de datos.
plt.scatter(X[Y == 0,0], X[Y == 0,1], c="skyblue")
plt.scatter(X[Y == 1,0], X[Y == 1,1], c="salmon")

plt.tick_params(labelbottom=False, labelleft=False)
```

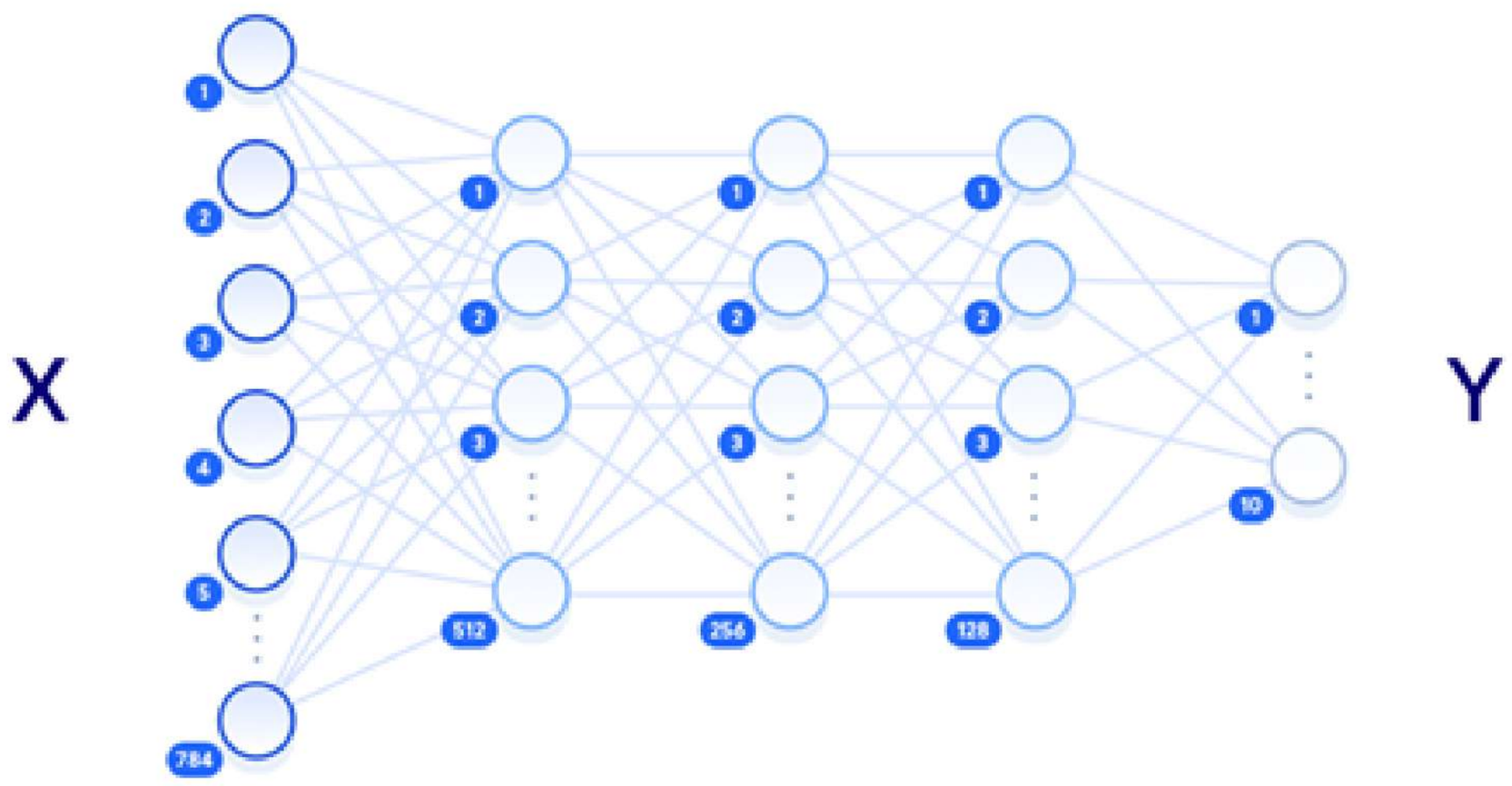
Al ejecutar el código obtiene:



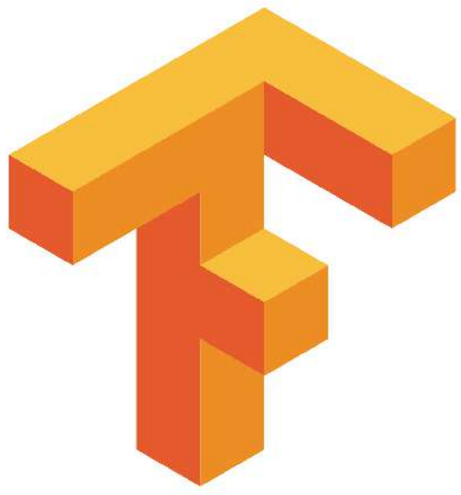
Paso 3: Definición de la arquitectura de la red neuronal.

La arquitectura de la red neuronal se refiere a elementos como el número de capas en la red, el número de unidades en cada capa y la forma en que las unidades están conectadas entre capas. Como las redes neuronales están ligeramente inspiradas en el funcionamiento del cerebro humano, aquí el término unidad se usa para representar lo que biológicamente consideraríamos una neurona.

Al igual que las neuronas que pasan señales alrededor del cerebro, las unidades toman algunos valores de las unidades anteriores como entrada, realizan un cálculo y luego pasan el nuevo valor como salida a otras unidades; estas unidades se colocan en capas para formar la red, comenzando por un mínimo con una capa para ingresar valores y una capa para generar valores. El término capa oculta se usa para todas las capas entre las capas de entrada y salida, es decir, aquellas "ocultas" del mundo real.



¿Cómo se ve en cada herramienta?

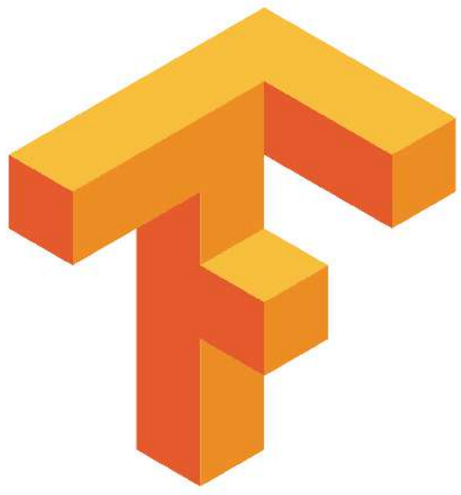


Tensorflow

```python
#import tensorflow as tf
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

from matplotlib import animation
from IPython.core.display import display, HTML

# Definimos los puntos de entrada de la red, para la matriz X e Y.
iX = tf.placeholder('float', shape=[None, X.shape[1]])
iY = tf.placeholder('float', shape=[None])

lr = 0.01           # learning rate
nn = [2, 16, 8, 1]  # número de neuronas por capa.

# Capa 1
W1 = tf.Variable(tf.random_normal([nn[0], nn[1]]), name='Weights_1')
b1 = tf.Variable(tf.random_normal([nn[1]]), name='bias_1')
l1 = tf.nn.relu(tf.add(tf.matmul(iX, W1), b1))
# Capa 2
W2 = tf.Variable(tf.random_normal([nn[1], nn[2]]), name='Weights_2')
b2 = tf.Variable(tf.random_normal([nn[2]]), name='bias_2')
l2 = tf.nn.relu(tf.add(tf.matmul(l1, W2), b2))
# Capa 3
W3 = tf.Variable(tf.random_normal([nn[2], nn[3]]), name='Weights_3')
b3 = tf.Variable(tf.random_normal([nn[3]]), name='bias_3')
# Vector de predicciones de Y.
pY = tf.nn.sigmoid(tf.add(tf.matmul(l2, W3), b3))[:, 0]

# Evaluación de las predicciones.
loss = tf.losses.mean_squared_error(pY, iY)

# Definimos al optimizador de la red, para que minimice el error.
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.05).minimize(loss)
n_steps = 1000 # Número de ciclos de entrenamiento.
iPY = [] # Aquí guardaremos la evolución de las predicción, para la animación.
```

```python
with tf.Session() as sess:
  # Inicializamos todos los parámetros de la red, las matrices W y b.
  sess.run(tf.global_variables_initializer())
  # Iteramos n pases de entrenamiento.
  for step in range(n_steps):
    # Evaluamos al optimizador, a la función de coste y al tensor de salida pY.
    # La evaluación del optimizer producirá el entrenamiento de la red.
    _, _loss, _pY = sess.run([optimizer, loss, pY], feed_dict={ iX : X, iY : Y })
    # Cada 25 iteraciones, imprimimos métricas.
    if step % 25 == 0:
      # Cálculo del accuracy.
      acc = np.mean(np.round(_pY) == Y)
      # Impresión de métricas.
      print('Step', step, '/', n_steps, '- Loss = ', _loss, '- Acc =', acc)
      # Obtenemos predicciones para cada punto de nuestro mapa de predicción _pX.
      _pY = sess.run(pY, feed_dict={ iX : _pX }).reshape((res, res))
      # Y lo guardamos para visualizar la animación.
      iPY.append(_pY)

# ----- CÓDIGO ANIMACIÓN ----- #
ims = []
fig = plt.figure(figsize=(10, 10))
print("--- Generando animación ---")

for fr in range(len(iPY)):
  im = plt.pcolormesh(_x0, _x1, iPY[fr], cmap="coolwarm", animated=True)
  # Visualización de la nube de datos.
  plt.scatter(X[Y == 0,0], X[Y == 0,1], c="skyblue")
  plt.scatter(X[Y == 1,0], X[Y == 1,1], c="salmon")
  # plt.title("Resultado Clasificación")
  plt.tick_params(labelbottom=False, labelleft=False)
  ims.append([im])
ani = animation.ArtistAnimation(fig, ims, interval=50, blit=True, repeat_delay=1000)
HTML(ani.to_html5_video())
```

¿Cómo se ve en cada herramienta?



Tensorflow

Al ejecutar el código, se obtiene:

```
Step 0 / 1000 - Loss =  0.32537392 - Acc = 0.466
Step 25 / 1000 - Loss =  0.22262195 - Acc = 0.63
Step 50 / 1000 - Loss =  0.18647127 - Acc = 0.72
Step 75 / 1000 - Loss =  0.14585125 - Acc = 0.894
Step 100 / 1000 - Loss =  0.1189033 - Acc = 0.912
Step 125 / 1000 - Loss =  0.10308567 - Acc = 0.92
Step 150 / 1000 - Loss =  0.09005698 - Acc = 0.94
Step 175 / 1000 - Loss =  0.07841048 - Acc = 0.954
Step 200 / 1000 - Loss =  0.06785429 - Acc = 0.966
Step 225 / 1000 - Loss =  0.058646675 - Acc = 0.976
Step 250 / 1000 - Loss =  0.050697666 - Acc = 0.986
Step 275 / 1000 - Loss =  0.044074446 - Acc = 0.992
Step 300 / 1000 - Loss =  0.038502567 - Acc = 0.996
Step 325 / 1000 - Loss =  0.034026798 - Acc = 0.998
Step 350 / 1000 - Loss =  0.030249048 - Acc = 1.0
Step 375 / 1000 - Loss =  0.027086396 - Acc = 1.0
Step 400 / 1000 - Loss =  0.024427509 - Acc = 1.0
Step 425 / 1000 - Loss =  0.022110028 - Acc = 1.0
Step 450 / 1000 - Loss =  0.02012782 - Acc = 1.0
Step 475 / 1000 - Loss =  0.018423121 - Acc = 1.0
Step 500 / 1000 - Loss =  0.01695205 - Acc = 1.0
Step 525 / 1000 - Loss =  0.015670493 - Acc = 1.0
Step 550 / 1000 - Loss =  0.014527266 - Acc = 1.0
Step 575 / 1000 - Loss =  0.013528361 - Acc = 1.0
Step 600 / 1000 - Loss =  0.012646711 - Acc = 1.0
Step 625 / 1000 - Loss =  0.011865986 - Acc = 1.0
Step 650 / 1000 - Loss =  0.011165066 - Acc = 1.0
Step 675 / 1000 - Loss =  0.010532951 - Acc = 1.0
Step 700 / 1000 - Loss =  0.009959498 - Acc = 1.0
Step 725 / 1000 - Loss =  0.009432784 - Acc = 1.0
Step 750 / 1000 - Loss =  0.0089501655 - Acc = 1.0
Step 775 / 1000 - Loss =  0.0085106455 - Acc = 1.0
Step 800 / 1000 - Loss =  0.008108985 - Acc = 1.0
Step 825 / 1000 - Loss =  0.0077409167 - Acc = 1.0
Step 850 / 1000 - Loss =  0.007403727 - Acc = 1.0
Step 875 / 1000 - Loss =  0.00709329 - Acc = 1.0
Step 900 / 1000 - Loss =  0.0068058507 - Acc = 1.0
Step 925 / 1000 - Loss =  0.0065393914 - Acc = 1.0
Step 950 / 1000 - Loss =  0.0062871217 - Acc = 1.0
Step 975 / 1000 - Loss =  0.006051607 - Acc = 1.0
--- Generando animación ---
```

¿Cómo se ve en cada herramienta?

Keras

```
import tensorflow as tf
import tensorflow.keras as kr

from IPython.core.display import display, HTML


lr = 0.01           # learning rate
nn = [2, 16, 8, 1]  # número de neuronas por capa.


# Creamos el objeto que contendrá a nuestra red neuronal, como
# secuencia de capas.
model = kr.Sequential()

# Añadimos la capa 1
l1 = model.add(kr.layers.Dense(nn[1], activation='relu'))

# Añadimos la capa 2
l2 = model.add(kr.layers.Dense(nn[2], activation='relu'))

# Añadimos la capa 3
l3 = model.add(kr.layers.Dense(nn[3], activation='sigmoid'))

# Compilamos el modelo, definiendo la función de coste y el optimizador.
model.compile(loss='mse', optimizer=kr.optimizers.SGD(lr=0.05), metrics=['acc'])

# Y entrenamos al modelo. Los callbacks
model.fit(X, Y, epochs=100)
```

Al ejecutar el código, se obtiene:

```
Epoch 90/100
500/500 [==============================] - 0s 43us/sample - loss: 0.0345 - acc: 1.0000
Epoch 91/100
500/500 [==============================] - 0s 41us/sample - loss: 0.0331 - acc: 1.0000
Epoch 92/100
500/500 [==============================] - 0s 47us/sample - loss: 0.0318 - acc: 1.0000
Epoch 93/100
500/500 [==============================] - 0s 40us/sample - loss: 0.0306 - acc: 1.0000
Epoch 94/100
500/500 [==============================] - 0s 46us/sample - loss: 0.0294 - acc: 1.0000
Epoch 95/100
500/500 [==============================] - 0s 42us/sample - loss: 0.0282 - acc: 1.0000
Epoch 96/100
500/500 [==============================] - 0s 47us/sample - loss: 0.0272 - acc: 1.0000
Epoch 97/100
500/500 [==============================] - 0s 45us/sample - loss: 0.0261 - acc: 1.0000
Epoch 98/100
500/500 [==============================] - 0s 47us/sample - loss: 0.0252 - acc: 1.0000
Epoch 99/100
500/500 [==============================] - 0s 45us/sample - loss: 0.0243 - acc: 1.0000
Epoch 100/100
500/500 [==============================] - 0s 44us/sample - loss: 0.0234 - acc: 1.0000
```

¿Cómo se ve en cada herramienta?


```
import sklearn as sk
import sklearn.neural_network

from IPython.core.display import display, HTML


lr = 0.01           # learning rate
nn = [2, 16, 8, 1]  # número de neuronas por capa.

# Creamos el objeto del modelo de red neuronal multicapa.
clf = sk.neural_network.MLPRegressor(solver='sgd',
                                     learning_rate_init=lr,
                                     hidden_layer_sizes=tuple(nn[1:]),
                                     verbose=True,
                                     n_iter_no_change=1000,
                                     batch_size = 64)


# Y lo entrenamos con nuestro datos.
clf.fit(X, Y)
```

Al ejecutar el código, se obtiene:

```
Iteration 190, loss = 0.12502886
Iteration 191, loss = 0.12509349
Iteration 192, loss = 0.12544402
Iteration 193, loss = 0.12504299
Iteration 194, loss = 0.12502855
Iteration 195, loss = 0.12505882
Iteration 196, loss = 0.12504925
Iteration 197, loss = 0.12503564
Iteration 198, loss = 0.12506720
Iteration 199, loss = 0.12510646
Iteration 200, loss = 0.12510412
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:
  % self.max_iter, ConvergenceWarning)
MLPRegressor(activation='relu', alpha=0.0001, batch_size=64, beta_1=0.9,
             beta_2=0.999, early_stopping=False, epsilon=1e-08,
             hidden_layer_sizes=(16, 8, 1), learning_rate='constant',
             learning_rate_init=0.01, max_iter=200, momentum=0.9,
             n_iter_no_change=1000, nesterovs_momentum=True, power_t=0.5,
             random_state=None, shuffle=True, solver='sgd', tol=0.0001,
             validation_fraction=0.1, verbose=True, warm_start=False)
```