3.1

In [1]:
```
#3.1
import tensorflow as tf
# tạo nút trong biểu đồ tính toán
node1 = tf.constant(3, dtype=tf.int32) #tạo các hằng số với kiểu dữ liệu int
node2 = tf.constant(5, dtype=tf.int32)
node3 = tf.add(node1, node2) #tổng 2 hằng đố

# tạo 1 phiên tf giúp cho các biểu đồ tính toán lm vc trong phiên này
sess = tf.compat.v1.Session()
# evaluating node3 and printing the result
#print("sum of node1 and node2 is :",sess.run(node3))
# đóng phiên
sess.close()
import tensorflow.compat.v1 as tf
x = tf.constant(5,tf.float32) #tạo hằng số tf với kiểu dữ liệu float
y = tf.constant([5], tf.float32) #tạo hằng số tf chứa 1 mảng với kiểu dữ liệu float
z = tf.constant([5,3,4], tf.float32) #tạo hằng số tf chứa 1 mảng với kiểu dữ liệu float
t = tf.constant([[5,3,4,6],[2,3,4,7]], tf.float32) #tạo hằng số tf chứa 1 ma trân 2x4 với kiểu dữ liệu float
u = tf.constant([[[5,3,4,6],[2,3,4,0]]], tf.float32) #tạo hằng số tf chứa 1 ma trân 1x2x4 với kiểu dữ liệu float
v = tf.constant([[[5,3,4,6],[2,3,4,0]],
[[5,3,4,6],[2,3,4,0]],
[[5,3,4,6],[2,3,4,0]]
], tf.float32) #tạo 1 hằng chứa 1 tensor 3 chiều
print(y)

#sử dụng TensorFlow để tạo và quản lý các biến và hẹn định trong một biểu đồ tính toán
```

```
tf.Tensor([5.], shape=(1,), dtype=float32)
```

3.2

```
#3.2
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution() #tắt tính toán ngay lập tức khi dùng bản 2. vì code nay sử dụng cho bản 1. tính toán truyề

#định nghĩa các biến và kiểu dữ liệu tương ứng
x1 = tf.Variable(5.3, tf.float32)
x2 = tf.Variable(4.3, tf.float32)
x = tf.multiply(x1,x2) #tích
init = tf.global_variables_initializer() #tạo 1 operation để khởi tạo tất cả các biến đc định nghĩa bên trên
with tf.Session() as sess:  #tạo phiên tính toán
    sess.run(init) #khởi tạo tất cả các biến trong phiên tính toán
    t = sess.run(x) #khởi tạo và tính giá trị của biến x
    print(t) #in ra

# sử dụng TensorFlow để tạo và tính toán một biểu thức trong một phiên tính toán.
```

```
22.79
```

```
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()#tắt tính toán ngay lập tức khi dùng bản 2. vì code nay sử dụng cho bản 1. tính toán truyề

#định nghĩa biến dưới dạng ma trận và kiểu dữ liệu của nó
x1 = tf.Variable([[5.3,4.5,6.0],
[4.3,4.3,7.0]
], tf.float32)
x2 = tf.Variable([[4.3,4.3,7.0],
[5.3,4.5,6.0]
], tf.float32)

x = tf.multiply(x1,x2) #tích
init = tf.global_variables_initializer() #tạo 1 operation để khởi tạo tất cả các biến đc định nghĩa bên trên
with tf.Session() as sess: #tạo phiên tính toán
    sess.run(init)#khởi tạo tất cả các biến trong phiên tính toán
    t = sess.run(x) #khởi tạo và tính giá trị biến x
    print(t)

# sử dụng TensorFlow để tạo và tính toán một ma trận trong một phiên tính toán.
```

```
[[22.79 19.35 42.  ]
 [22.79 19.35 42.  ]]
```

```
import tensorflow.compat.v1 as tf
# khởi tạo biến node là 1 ma trận 2x2 chứa các giá trị 0
node = tf.Variable(tf.zeros([2,2]))
# tạo phiên tính toán và khởi tạo biến
with tf.Session() as sess:
    # khởi tạo tất cả các biến toàn cục
    sess.run(tf.global_variables_initializer())
    # in giá trị node
    print("Tensor value before addition:\n",sess.run(node))
    # elementwise addition to tensor
    node = node.assign(node + tf.ones([2,2])) # phép cộng phần tử của 2 biến ma trận
    # in lại giá trị node
    print("Tensor value after addition:\n", sess.run(node))
    sess.close() # đóng phiên
```

```
Tensor value before addition:
 [[0. 0.]
 [0. 0.]]
Tensor value after addition:
 [[1. 1.]
 [1. 1.]]
```

3.3

```
#3.3
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()#tắt tính toán ngay lập tức khi dùng bản 2. vì code nay sử dụng cho bản 1. tính toán truyền

#Placeholder là một cách để bạn đặt giá trị đầu vào vào biểu đồ tính toán sau khi đã xây dựng nó
x = tf.placeholder(tf.float32,None) #placeholder x có thể chứa một số lượng phần tử bất kỳ.
y = tf.add(x,x) #cộng giá trị
with tf.Session() as sess: #tạo phiên tính toán
    x_data= 5 #khởi tạo giá trị
    result = sess.run(y,feed_dict={x:x_data}) #khởi tạo  biến y và gán giá trị biến x
    print(result) #in giá trị

#sử dụng placeholder trong TensorFlow để chứa dữ liệu đầu vào và làm cho biểu đồ tính toán
#có thể thực hiện với các giá trị đầu vào cụ thể sau khi đã xây dựng biểu đồ.
```

```
10.0
```

```
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution() #tắt tính toán ngay lập tức khi dùng bản 2. vì code nay sử dụng cho bản 1. tính toán truyề

# hình dạng (shape) [None, 3]. Điều này có nghĩa rằng x có thể chứa một số hàng bất kỳ và mỗi hàng chứa 3 phần tử.
x = tf.placeholder(tf.float32,[None,3])
y = tf.add(x,x) #định nghĩa biến y là tổng 2x
with tf.Session() as sess:
    x_data= [[1.5, 2.0, 3.3]] #khởi tạo giá trị
    result = sess.run(y,feed_dict={x:x_data}) #khởi tạo biến y là cung cấp giá trị x
    print(result)
```

```
[[3.  4.  6.6]]
```

```python
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()#tắt tính toán ngay lập tức khi dùng bản 2. vì code nay sử dụng cho bản 1. tính toán truyền

x = tf.placeholder(tf.float32,[None,None,3]) # x có thể chứa một số lượng hàng và cột bất kỳ và mỗi hàng-chỗ sẽ chứa 3 phần tử.
y = tf.add(x,x) #định nghĩa biến y
with tf.Session() as sess: #tạo phiên
    x_data= [[[1,2,3]]]
    result = sess.run(y,feed_dict={x:x_data})
    print(result)

#sử dụng placeholder trong TensorFlow để chứa dữ liệu đầu vào và làm cho biểu đồ tính toán
#có thể thực hiện với các giá trị đầu vào cụ thể sau khi đã xây dựng biểu đồ.
```

```
[[[2. 4. 6.]]]
```

```python
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()#tắt tính toán ngay lập tức khi dùng bản 2. vì code nay sử dụng cho bản 1. tính toán truyền

x = tf.placeholder(tf.float32,[None,4,3]) #x có thể chứa một số hàng bất kỳ và mỗi hàng-chỗ sẽ chứa một ma trận 4x3.
y = tf.add(x,x) #định nghĩa biến y
with tf.Session() as sess: #tạo phiên
    x_data= [[[1,2,3],
    [2,3,4],
    [2,3,5],
    [0,1,2]
    ]]
    result = sess.run(y,feed_dict={x:x_data}) #khởi tạo giá trị y và cung cấp giá trị x
    print(result)
```

```
[[[ 2.  4.  6.]
  [ 4.  6.  8.]
  [ 4.  6. 10.]
  [ 0.  2.  4.]]]
```

```python
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()#để làm cho mã hoạt động theo kiểu truyền thống dựa trên TensorFlow 1.x.

x = tf.placeholder(tf.float32,[2,4,3]) #x có hình dạng chính xác là ma trận 3 chiều với kích thước là [2, 4, 3].
y = tf.add(x,x) #định nghĩa y
with tf.Session() as sess: #tạo phiên
    x_data= [[[1,2,3],
                [2,3,4],
                [2,3,5],
                [0,1,2]
                ],
                [[1,2,3],
                [2,3,4],
                [2,3,5],
                [0,1,2]
                ]]
    result = sess.run(y,feed_dict={x:x_data}) #khởi tạo giá trị y và cung cấp giá trị cho x
    print(result)

#sử dụng placeholder trong TensorFlow để chứa dữ liệu đầu vào và làm cho biểu đồ tính toán
#có thể thực hiện với các giá trị đầu vào cụ thể sau khi đã xây dựng biểu đồ.
```

```
[[[ 2.  4.  6.]
  [ 4.  6.  8.]
  [ 4.  6. 10.]
  [ 0.  2.  4.]]

 [[ 2.  4.  6.]
  [ 4.  6.  8.]
  [ 4.  6. 10.]
  [ 0.  2.  4.]]]
```

```
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()#để làm cho mã hoạt động theo kiểu truyền thống dựa trên TensorFlow 1.x.

x = tf.placeholder(tf.float32,[2,4,3]) #x có hình dạng chính xác là ma trận 3 chiều với kích thước là [2, 4, 3].
y = tf.placeholder(tf.float32,[2,4,3]) #y có hình dạng chính xác là ma trận 3 chiều với kích thước là [2, 4, 3].
z = tf.add(x,y) #định nghĩa tổng
u = tf.multiply(x,y) #định nghĩa tích
with tf.Session() as sess: #tạo phiên
    x_data= [[[1,2,3],
              [2,3,4],
              [2,3,5],
              [0,1,2]
              ],
              [[1,2,3],
              [2,3,4],
              [2,3,5],
              [0,1,2]
              ]]
    y_data= [[[1,2,3],
              [2,3,4],
              [2,3,5],
              [0,1,2]
              ],
              [[1,2,3],
              [2,3,4],
              [2,3,5],
              [0,1,2]
              ]]
    result1 = sess.run(z,feed_dict={x:x_data, y:y_data}) #khởi tạo giá trị z và cung cấp giá trị x,y
    result2 = sess.run(u,feed_dict={x:x_data, y:y_data}) #khởi tạo giá trị u và cung cấp giá trị x,y
    print("result1 =", result1)
    print("result2 =", result2)
```

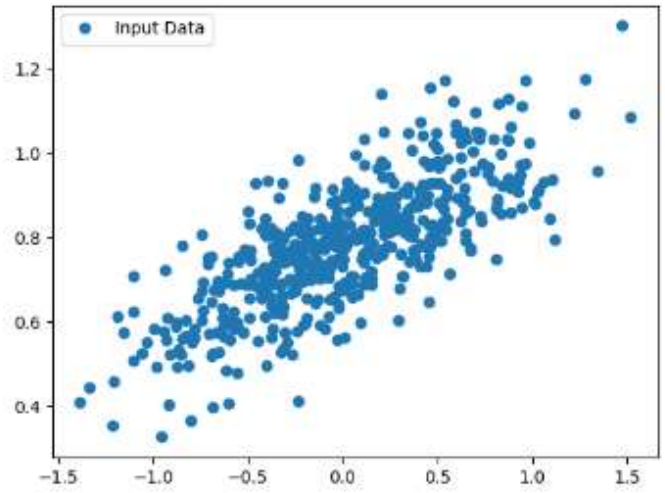
```
result1 = [[[ 2.  4.  6.]
  [ 4.  6.  8.]
  [ 4.  6. 10.]
  [ 0.  2.  4.]]

 [[ 2.  4.  6.]
  [ 4.  6.  8.]
  [ 4.  6. 10.]
  [ 0.  2.  4.]]]
result2 = [[[ 1.  4.  9.]
  [ 4.  9. 16.]
  [ 4.  9. 25.]
  [ 0.  1.  4.]]

 [[ 1.  4.  9.]
  [ 4.  9. 16.]
  [ 4.  9. 25.]
  [ 0.  1.  4.]]]
```

3.4

```python
#3.4
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()##để làm cho mã hoạt động theo kiểu truyền thống dựa trên TensorFlow 1.x.

x1 = tf.constant(5.3, tf.float32) #định nghĩa hằng số
x2 = tf.constant(1.5, tf.float32)

w1 = tf.Variable(0.7, tf.float32) #định nghĩa biến
w2 = tf.Variable(0.5, tf.float32)

u = tf.multiply(x1,w1) #định nghĩa tích
v = tf.multiply(x2,w2)
z = tf.add(u,v) #định nghĩa tổng
result = tf.sigmoid(z) #tính giá trị sigmoid của z và lưu vào biến result.
# Hàm sigmoid : σ(x) = 1 / (1 + e^(-x))

init = tf.global_variables_initializer() #khởi tạo các biến toàn cục đã được định nghĩa trước đó
with tf.Session() as sess:
    sess.run(init) #chạy khởi tạo tất cả các biến
    print(sess.run(result))

#Mục đích của đoạn code là tính giá trị của hàm sigmoid trên một biểu đồ tính toán TensorFlow sử dụng các hằng số và biến, và in
```

```
0.9885698
```

```python
import numpy as np
import matplotlib.pyplot as plt

number_of_points = 500 # để xác định số lượng điểm dữ liệu mà chúng ta sẽ tạo.
#khởi tạo danh sách rỗng để lưu trữ các điểm được tạo ra
x_point = []
y_point = []

#a và b đại diện cho các hệ số trong phương trình tuyến tính y = ax + b, và chúng được sử dụng để tạo dữ liệu ngẫu nhiên dựa trê
a = 0.22
b = 0.78

for i in range(number_of_points): #xét 500 điểm dữ liệu
    x = np.random.normal(0.0,0.5) #tạo một giá trị x ngẫu nhiên từ một phân phối chuẩn (Gaussian) có trung bình 0 và độ lệch chu
    y = a*x + b +np.random.normal(0.0,0.1) # một lượng nhiễu ngẫu nhiên thông qua np.random.normal(0.0, 0.1)
    #thêm giá trị x, y vào danh sách
    x_point.append([x])
    y_point.append([y])

#vẽ biểu đồ và các điểm dữ liệu dánh dấu là o
plt.plot(x_point,y_point, 'o', label = 'Input Data')
plt.legend() #được sử dụng để hiển thị chú thích về biểu đồ
plt.show()

#mục đích của đoạn code này là tạo dữ liệu ngẫu nhiên dựa trên một phương trình tuyến tính và vẽ biểu đồ để hiển thị dữ liệu đó.
#Điều này có thể sử dụng để minh họa hoặc kiểm tra các phương pháp phân loại hoặc hồi quy tuyến tính.
```




```
]: import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()#để làm cho mã hoạt động theo kiểu truyền thống dựa trên TensorFlow 1.x.

#định nghĩa placeholder gồm có thể 1 số hàng và mỗi hàng có 3 phần tử
x1 = tf.placeholder(tf.float32,[None,3])
x2 = tf.placeholder(tf.float32,[None,3])

#định nghĩa biến w1 và w2 đại diện cho các trọng số trong phương trình tuyến tính
w1 = tf.Variable([0.5,0.4,0.7],tf.float32)
w2 = tf.Variable([0.8,0.5,0.6], tf.float32)

#u1 và u2 lưu trữ tích của trọng số tương ứng với x1 và x2
u1 = tf.multiply(w1,x1)
u2 = tf.multiply(w2,x2)

#v lưu trữ tổng của u1 và u2.
v = tf.add(u1,u2)
z = tf.sigmoid(v) #z lưu trữ giá trị sigmoid của v.
init = tf.global_variables_initializer() #khởi tạo tất cả các biến toàn cục định nghĩa trước đó
with tf.Session() as sess: #tạo phiên
    x1_data= [[1,2,3]]
    x2_data= [[1,2,3]]
    sess.run(init) # khởi tạo
    result = sess.run(z,feed_dict={x1:x1_data, x2:x2_data}) #tính giá trị của z, tức là giá trị sigmoid của v, và in giá trị này
    print(result)

#Mục đích của đoạn code này là tính giá trị sigmoid của một biểu đồ tính toán sử dụng placeholders và biến trong TensorFlow, với
```

```
[[0.785835   0.85814893 0.9801597 ]]
```

```python
import tensorflow as tf
import numpy as np

# định nghĩa hai ma trận 3x3 matrix1 và matrix2 với các giá trị được chỉ định và kiểu dữ liệu là 'int32'.
matrix1 = np.array([(2,2,2),(2,2,2),(2,2,2)],dtype = 'int32')
matrix2 = np.array([(1,1,1),(1,1,1),(1,1,1)],dtype = 'int32')

#in
print (matrix1)
print (matrix2, "\n")

#chuyển đổi hai ma trận từ NumPy arrays thành TensorFlow constants để có thể sử dụng chúng trong tính toán TensorFlow.
matrix1 = tf.constant(matrix1)
matrix2 = tf.constant(matrix2)

#tích 2 ma trận
matrix_product = tf.matmul(matrix1, matrix2)
#tổng 2 ma trận
matrix_sum = tf.add(matrix1,matrix2)

#định nghĩa ma trận matrix_3 3x3 với các giá trị và kiểu dữ liệu là 'float32'.
matrix_3 = np.array([(2,7,2),(1,4,2),(9,0,2)],dtype = 'float32')
print (matrix_3, "\n")

#tính định thức của matrix_3.
matrix_det = tf.linalg.det(matrix_3)

with tf.compat.v1.Session() as sess: #tạo phiên tính toán
    result1 = sess.run(matrix_product) #tính toán các giá trị
    result2 = sess.run(matrix_sum)
    result3 = sess.run(matrix_det)
    print (result1)
    print (result2)
    print (result3)
#note: không cần dùng khởi tạo các biến vì k dùng kiểu biến mà đang dùng kiểu hằng số
#Mục đích của đoạn code này là thực hiện các phép tính ma trận như nhân ma trận, cộng ma trận và tính định thức bằng sử dụng Ten
```

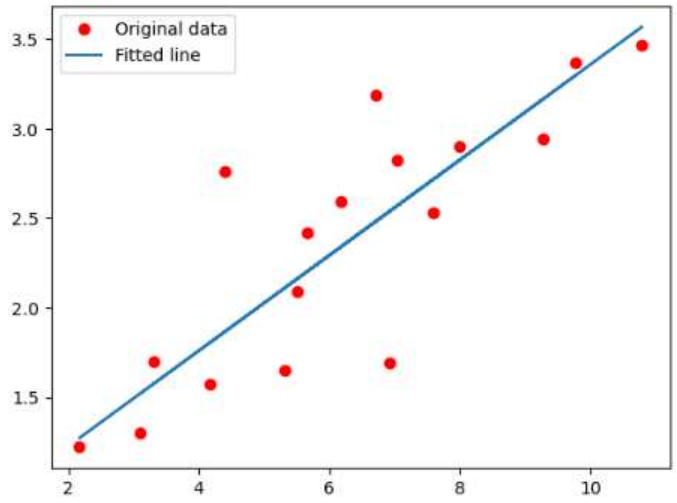
```
[[2 2 2]
 [2 2 2]
 [2 2 2]]
[[1 1 1]
 [1 1 1]
 [1 1 1]]

[[2. 7. 2.]
 [1. 4. 2.]
 [9. 0. 2.]]

[[6 6 6]
 [6 6 6]
 [6 6 6]]
[[3 3 3]
 [3 3 3]
 [3 3 3]]
55.999992
```

3.5

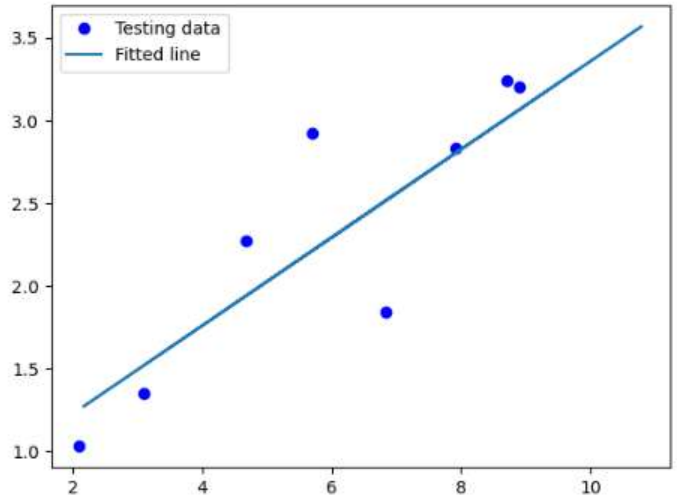
```
#3.5
# importing the dependencies
import tensorflow.compat.v1 as tf
import numpy as np
import matplotlib.pyplot as plt

# Thông số mô hình
#Các tham số này bao gồm tỷ lệ học (learning rate), số lượng epoch (vòng lặp huấn luyện),
#và bước hiển thị thông tin trong quá trình huấn luyện
learning_rate = 0.01
training_epochs = 2000
display_step = 200

# dữ liệu huấn luyện
train_X = np.asarray([3.3, 4.4, 5.5, 6.71, 6.93, 4.168, 9.779, 6.182, 7.59, 2.167, 7.042, 10.791, 5.313, 7.997, 5.654, 9.27, 3.1]
train_y = np.asarray([1.7, 2.76, 2.09, 3.19, 1.694, 1.573, 3.366, 2.596, 2.53, 1.221, 2.827, 3.465, 1.65, 2.904, 2.42, 2.94, 1.3]
n_samples = train_X.shape[0]

# tập dữ liệu kiểm tra
test_X = np.asarray([6.83, 4.668, 8.9, 7.91, 5.7, 8.7, 3.1, 2.1])

# định nghĩa placeholder sử dụng để cung cấp dữ liệu cho mô hình trong quá trình huấn luyện và kiểm tra.
X = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)

# tập dữ liệu kiểm tra
test_y = np.asarray([1.84, 2.273, 3.2, 2.831, 2.92, 3.24, 1.35, 1.03])

#định nghĩa biến cho mô hình sẽ học tối ưu hóa
W = tf.Variable(np.random.randn(), name="weight")
b = tf.Variable(np.random.randn(), name="bias")

# xây dựng mô hình tuyến tính  tính đơn giản với đầu vào X được tính bằng cách nhân trọng số W với X và cộng thêm bias b.
linear_model = W*X + b

# định nghĩa hàm mất mát đo lường sự sai lệch giữa dự đoán linear_model và đầu ra thực tế y và được tính bằng sai số bình phương
cost = tf.reduce_sum(tf.square(linear_model - y)) / (2*n_samples)

# sử dụng thuật toán gradient descent với tỷ lệ học learning_rate để cập nhật trọng số W và bias b sao cho hàm mất mát giảm dần.
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# khởi tạo các biến toàn cục trước đó định nghĩa
init = tf.global_variables_initializer()
```

```
# tạo phiên và tính toán huấn luyện mô hình
with tf.Session() as sess:
    # khởi tạo
    sess.run(init)

    #Trong mỗi vòng lặp, chúng ta thực hiện một bước gradient descent để cập nhật trọng số và bias, sau đó in thông tin về cost y
    # Fit all training data
    for epoch in range(training_epochs):
        # Thực hiện bước giảm độ dốc
        sess.run(optimizer, feed_dict={X: train_X, y: train_y})

        # hiển thị nhật kí qua các vòng
        if (epoch+1) % display_step == 0:
            c = sess.run(cost, feed_dict={X: train_X, y: train_y})
            print("Epoch:{0:6} \t Cost:{1:10.4} \t W:{2:6.4} \t b:{3:6.4}".format(epoch+1, c, sess.run(W), sess.run(b)))

    # Sau khi huấn luyện xong, chúng ta in ra cost function cuối cùng và các trọng số của mô hình.
    print("Optimization Finished!")
    training_cost = sess.run(cost, feed_dict={X: train_X, y: train_y})
    print("Final training cost:", training_cost, "W:", sess.run(W), "b:", sess.run(b), '\n')

    # hiển thị đồ thị
    #vẽ đồ thị dữ liệu gốc và đường thẳng tương ứng với mô hình sau khi huấn luyện để trực quan hóa kết quả.
    plt.plot(train_X, train_y, 'ro', label='Original data')
    plt.plot(train_X, sess.run(W) * train_X + sess.run(b), label='Fitted line')
    plt.legend()
    plt.show()

    # đánh giá mô hình trên tập kiểm tra
    #tính toán cost function trên dữ liệu kiểm tra và in ra kết quả. Điều này giúp đánh giá hiệu suất của mô hình trên dữ liệu mớ
    testing_cost = sess.run(tf.reduce_sum(tf.square(linear_model - y)) / (2 * test_X.shape[0]), feed_dict={X: test_X, y: test_y})
    print("Final testing cost:", testing_cost)
    print("Absolute mean square loss difference:", abs(training_cost - testing_cost))

    # Hiển thị dữ liệu kiểm tra
    plt.plot(test_X, test_y, 'bo', label='Testing data')
    plt.plot(train_X, sess.run(W) * train_X + sess.run(b), label='Fitted line')
    plt.legend()
    plt.show()

#Mục đích của đoạn code này là xây dựng một mô hình hồi quy tuyến tính đơn giản bằng TensorFlow và huấn luyện nó trên dữ liệu
#huấn luyện. Sau đó, đánh giá hiệu suất của mô hình trên dữ liệu kiểm tra và trực quan hóa kết quả.
```

```
Epoch:   200     Cost:    0.1295         W:0.3816        b:-0.1226
Epoch:   400     Cost:    0.1093         W:0.3536        b:0.07607
Epoch:   600     Cost:   0.09682         W:0.3316        b:0.2319
Epoch:   800     Cost:   0.08917         W:0.3143        b:0.3542
Epoch:  1000     Cost:   0.08446         W:0.3008        b:0.4501
Epoch:  1200     Cost:   0.08156         W:0.2902        b:0.5253
Epoch:  1400     Cost:   0.07978         W:0.2819        b:0.5843
Epoch:  1600     Cost:   0.07868         W:0.2754        b:0.6305
Epoch:  1800     Cost:   0.07801         W:0.2703        b:0.6668
Epoch:  2000     Cost:   0.07759         W:0.2662        b:0.6953
Optimization Finished!
Final training cost: 0.07759212 W: 0.26623613 b: 0.69528544
```




```
Final testing cost: 0.07605784
Absolute mean square loss difference: 0.0015342832
```



3.6

```
#3.6

import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()

X1 = tf.Variable(2., tf.float32)
X2 = tf.Variable(3., tf.float32)
X3 = tf.Variable(4., tf.float32)

w1 = tf.constant(0.05, tf.float32)
w2 = tf.constant(0.07, tf.float32)
w3 = tf.constant(0.09, tf.float32)
b = tf.constant(0.6, tf.float32)

model= w1*X1 + w3*X3 + w2*X2 + b

res_tanh =tf.tanh(model)
res_sig = tf.sigmoid(model)
res_relu = tf.nn.relu(model)

init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    print(sess.run(res_tanh))
    print(sess.run(res_sig))
    print(sess.run(res_relu))
```

```
0.8537978
0.78074276
1.2700001
```

3.7

```
#3.7
import tensorflow.compat.v1 as tf
import numpy as np

X = tf.placeholder(tf.float32)
W = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)

model = tf.add(tf.matmul(X,W) , b)

res_tanh =tf.tanh(model)

# lấy dữ liệu, vd
x_data = [[2,3,4]]
w_data = np.array([[0.05, 0.07, 0.09]])
b_data = 0.6
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    print(sess.run(res_tanh , feed_dict={X:x_data , W:np.transpose(w_data) , b:b_data}))
```

```
[[0.8537976]]
```

4.1. Trình bày hiểu biết của mình về tensorflow và 5 ví dụ minh họa

TensorFlow là một thư viện phần mềm mã nguồn mở. TensorFlow ban đầu được phát triển bởi các nhà nghiên cứu và kỹ sư làm việc trong Nhóm Google Brain trong tổ chức nghiên cứu Trí tuệ Máy của Google với mục đích tiến hành học máy và nghiên cứu mạng nơ-ron sâu, nhưng hệ thống này đủ chung để áp dụng trong nhiều lĩnh vực khác! Trước tiên chúng ta hãy cố gắng hiểu từ TensorFlow thực sự có nghĩa là gì! TensorFlow về cơ bản là một thư viện phần mềm để tính toán số bằng cách sử dụng biểu đồ luồng dữ liệu trong đó:

- Các nút trong biểu đồ đại diện cho các hoạt động toán học.

- Các cạnh trong biểu đồ đại diện cho các mảng dữ liệu đa chiều (được gọi là tensor) được giao tiếp giữa chúng. (Xin lưu ý rằng tensor là đơn vị dữ liệu trung tâm trong TensorFlow).

API TensorFlow

TensorFlow cung cấp nhiều API (Application Programming Interfaces). Chúng có thể được phân thành 2 loại chính:

1. API cấp thấp:
    - Kiểm soát lập trình hoàn chỉnh
    - Được đề xuất cho các nhà nghiên cứu máy học
    - cung cấp mức độ kiểm soát tốt đối với các mô hình
    - TensorFlow Core là API cấp thấp của TensorFlow.
2. API cấp cao:
    - được xây dựng trên nền tảng TensorFlow Core
    - dễ học và sử dụng hơn TensorFlow Core
    - Làm cho các tác vụ lặp đi lặp lại dễ dàng và nhất quán hơn giữa những người dùng khác nhau
    - tf.contrib.learn là một ví dụ về API cấp cao.

```
# importing tensorflow
import tensorflow as tf

# creating nodes in computation graph
node1 = tf.constant(3, dtype=tf.int32)
node2 = tf.constant(5, dtype=tf.int32)
node3 = tf.add(node1, node2)

# create tensorflow session object
sess = tf.compat.v1.Session()

# evaluating node3 and printing the result
print("sum of node1 and node2 is :",sess.run(node3))
# closing the session
sess.close()
```

Variable

TensorFlow cũng có các nút Biến có thể chứa dữ liệu biến. Chúng chủ yếu được sử dụng để giữ và cập nhật các thông số của một mô hình đào tạo. Các biến là bộ đệm trong bộ nhớ có chứa tensor. Chúng phải được khởi tạo rõ ràng và có thể được lưu vào đĩa trong và sau khi đào tạo. Sau đó, bạn có thể khôi phục các giá trị đã lưu để thực hiện hoặc phân tích mô hình. Một sự khác biệt quan trọng cần lưu ý giữa hằng số và Biến là:

Dưới đây là một ví dụ sử dụng variable:

```
# importing tensorflow
import tensorflow as tf
```

```
# creating nodes in computation graph
node = tf.Variable(tf.zeros([2,2]))

# running computation graph
with tf.Session() as sess:

    # initialize all global variables
    sess.run(tf.global_variables_initializer())

    # evaluating node
    print("Tensor value before addition:\n",sess.run(node))

    # elementwise addition to tensor
    node = node.assign(node + tf.ones([2,2]))

    # evaluate node again
    print("Tensor value after addition:\n", sess.run(node))
```

placeholder

Một biểu đồ có thể được tham số hóa để chấp nhận các đầu vào bên ngoài, được gọi là trình giữ chỗ. Trình giữ chỗ là một lời hứa sẽ cung cấp giá trị sau này. Trong khi đánh giá biểu đồ liên quan đến các nút giữ chỗ, một tham số feed_dict được chuyển đến phương thức chạy của phiên để chỉ định Tensor cung cấp các giá trị cụ thể cho các trình giữ chỗ này. Hãy xem xét ví dụ được đưa ra dưới đây:

```
# importing tensorflow
import tensorflow as tf

# creating nodes in computation graph
a = tf.placeholder(tf.int32, shape=(3,1))
b = tf.placeholder(tf.int32, shape=(1,3))
c = tf.matmul(a,b)

# running computation graph
with tf.Session() as sess:
    print(sess.run(c, feed_dict={a:[[3],[2],[1]], b:[[1,2,3]]}))
```

Mô hình hồi quy tuyến tính

Dưới đây là việc triển khai mô hình hồi quy tuyến tính sử dụng API TensorFlow Core.

```
# importing the dependencies
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

```
# Model Parameters
learning_rate = 0.01
training_epochs = 2000
display_step = 200

# Training Data
train_X = np.asarray([3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.182,7.59,2.167,
                      7.042,10.791,5.313,7.997,5.654,9.27,3.1])
train_y = np.asarray([1.7,2.76,2.09,3.19,1.694,1.573,3.366,2.596,2.53,1.221,
                      2.827,3.465,1.65,2.904,2.42,2.94,1.3])
n_samples = train_X.shape[0]

# Test Data
test_X = np.asarray([6.83, 4.668, 8.9, 7.91, 5.7, 8.7, 3.1, 2.1])
test_y = np.asarray([1.84, 2.273, 3.2, 2.831, 2.92, 3.24, 1.35, 1.03])

# Set placeholders for feature and target vectors
X = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)

# Set model weights and bias
W = tf.Variable(np.random.randn(), name="weight")
b = tf.Variable(np.random.randn(), name="bias")

# Construct a linear model
linear_model = W*X + b

# Mean squared error
cost = tf.reduce_sum(tf.square(linear_model - y)) / (2*n_samples)

# Gradient descent
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# Initializing the variables
init = tf.global_variables_initializer()

# Launch the graph
with tf.Session() as sess:
    # Load initialized variables in current session
    sess.run(init)

    # Fit all training data
    for epoch in range(training_epochs):
```

```
        # perform gradient descent step
        sess.run(optimizer, feed_dict={X: train_X, y: train_y})

        # Display logs per epoch step
        if (epoch+1) % display_step == 0:
            c = sess.run(cost, feed_dict={X: train_X, y: train_y})
            print("Epoch:{0:6} \t Cost:{1:10.4} \t W:{2:6.4} \t b:{3:6.4}".
                format(epoch+1, c, sess.run(W), sess.run(b)))

    # Print final parameter values
    print("Optimization Finished!")
    training_cost = sess.run(cost, feed_dict={X: train_X, y: train_y})
    print("Final training cost:", training_cost, "W:", sess.run(W), "b:",
        sess.run(b), '\n')

    # Graphic display
    plt.plot(train_X, train_y, 'ro', label='Original data')
    plt.plot(train_X, sess.run(W) * train_X + sess.run(b), label='Fitted line')
    plt.legend()
    plt.show()

    # Testing the model
    testing_cost = sess.run(tf.reduce_sum(tf.square(linear_model - y)) / (2 *
test_X.shape[0]),
                        feed_dict={X: test_X, y: test_y})

    print("Final testing cost:", testing_cost)
    print("Absolute mean square loss difference:", abs(training_cost - testing_cost))

    # Display fitted line on test data
    plt.plot(test_X, test_y, 'bo', label='Testing data')
    plt.plot(train_X, sess.run(W) * train_X + sess.run(b), label='Fitted line')
    plt.legend()
    plt.show()
```

tf.contrib.learn

tf.contrib.learn là một thư viện TensorFlow cấp cao giúp đơn giản hóa cơ chế học máy, bao gồm:

- Chạy vòng đào tạo
- Chạy vòng đánh giá
- Quản lý tập dữ liệu
- Quản lý cho ăn

```
# importing the dependencies
```

```
import tensorflow as tf
import numpy as np

# declaring list of features
features = [tf.contrib.layers.real_valued_column("X")]

# creating a linear regression estimator
estimator = tf.contrib.learn.LinearRegressor(feature_columns=features)

# training and test data
train_X = np.asarray([3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.182,7.59,2.167,
                      7.042,10.791,5.313,7.997,5.654,9.27,3.1])
train_y = np.asarray([1.7,2.76,2.09,3.19,1.694,1.573,3.366,2.596,2.53,1.221,
                      2.827,3.465,1.65,2.904,2.42,2.94,1.3])
test_X = np.asarray([6.83, 4.668, 8.9, 7.91, 5.7, 8.7, 3.1, 2.1])
test_y = np.asarray([1.84, 2.273, 3.2, 2.831, 2.92, 3.24, 1.35, 1.03])

# function to feed dict of numpy arrays into the model for training
input_fn = tf.contrib.learn.io.numpy_input_fn({"X":train_X}, train_y,
                                  batch_size=4, num_epochs=2000)

# function to feed dict of numpy arrays into the model for testing
test_input_fn = tf.contrib.learn.io.numpy_input_fn({"X":test_X}, test_y)

# fit training data into estimator
estimator.fit(input_fn=input_fn)

# print value of weight and bias
W = estimator.get_variable_value('linear/X/weight')[0][0]
b = estimator.get_variable_value('linear/bias_weight')[0]
print("W:", W, "\tb:", b)

# evaluating the final loss
train_loss = estimator.evaluate(input_fn=input_fn)['loss']
test_loss = estimator.evaluate(input_fn=test_input_fn)['loss']
print("Final training loss:", train_loss)
print("Final testing loss:", test_loss)
```

4.2. Giải thích các dòng dánh dấu # trong Bài tập 3.5

```
# importing the dependencies
```

khai báo thư viện.

# Model Parameters
learning_rate = 0.01 # tỉ lệ học, hệ số ảnh hưởng tới bước di chuyển

training_epochs = 2000 # số lần duyệt qua hết các tập dữ liệu trong tập huấn luyện.
display_step = 200 # lấy 200 để in ra 10 lần kết quả trong quá trình tối ưu mô hình.
# Training Data
khởi tạo dữ liệu huấn luyện
Test Data
khởi tạo dữ liệu kiểm tra
Set placeholders for feature and target vectors
khai báo biến giữ vị trí
Set model weights and bias
khởi tạo trọng số và khoảng cách
Construct a linear model
khai báo biểu thức mô hình
Mean squared error
khai báo biểu thức tính khoảng cách Mean squared error
Gradient descent
tìm giá trị tối ưu bằng thuật toán xuống dốc
Initializing the variables
gọi các giá trị toàn cục
Launch the graph: khởi chạy đồ thị
Load initialized: variables in current session chạy các biến toàn cục
Fit all training data: tìm kiếm tham số tối ưu cho mô hình
perform gradient descent step: lấy giá trị tối ưu
Display logs per epoch step: hiển thị 1 epoch
Print final parameter values: hiển thị kết quả huấn luyện cuối
Graphic display : trực quan hóa kết quả huấn luyện
Testing the model: kiểm tra mô hình với dữ liệu kiểm tra
Display fitted line on test data: trực quan hóa kết quả kiểm tra

```
# importing the dependencies
import tensorflow.compat.v1 as tf
import numpy as np
import matplotlib.pyplot as plt
tf.compat.v1.disable_eager_execution()
# Model Parameters
learning_rate = 0.01
training_epochs = 2000
display_step = 200
# Training Data
train_X = np.asarray([3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.182,7.59,2.167,7.042,10.791,5.313,7.997,5.654,9.27,3.1])
train_y = np.asarray([1.7,2.76,2.09,3.19,1.694,1.573,3.366,2.596,2.53,1.221,2.827,3.465,1.65,2.904,2.42,2.94,1.3])
n_samples = train_X.shape[0]
# Test Data
test_X = np.asarray([6.83, 4.668, 8.9, 7.91, 5.7, 8.7, 3.1, 2.1])
# Set placeholders for feature and target vectors
X = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
# Set model weights and bias)
test_y = np.asarray([1.84, 2.273, 3.2, 2.831, 2.92, 3.24, 1.35, 1.03])
W = tf.Variable(np.random.randn(), name="weight")
b = tf.Variable(np.random.randn(), name="bias")
# Construct a linear model
linear_model = W*X + b
# Mean squared error
cost = tf.reduce_sum(tf.square(linear_model - y)) / (2*n_samples)
# Gradient descent
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
# Initializing the variables
init = tf.global_variables_initializer()
# Launch the graph
with tf.Session() as sess:
 # Load initialized variables in current session
 sess.run(init)
```

```
# Fit all training data
 for epoch in range(training_epochs):
 # perform gradient descent step
     sess.run(optimizer, feed_dict={X: train_X, y: train_y})
     # Display logs per epoch step
     if (epoch+1) % display_step == 0:
         c = sess.run(cost, feed_dict={X: train_X, y: train_y})
         print("Epoch:{0:6} \t Cost:{1:10.4} \t W:{2:6.4} \t b:{3:6.4}".format(epoch+1, c, sess.run(W), sess.run(b)))

 # Print final parameter values
 print("Optimization Finished!")
 training_cost = sess.run(cost, feed_dict={X: train_X, y: train_y})
 print("Final training cost:", training_cost, "W:", sess.run(W), "b:",sess.run(b), '\n')

 # Graphic display
 plt.plot(train_X, train_y, 'ro', label='Original data')
 plt.plot(train_X, sess.run(W) * train_X + sess.run(b), label='Fitted line')
 plt.legend()
 plt.show()

 # Testing the model
 testing_cost = sess.run(tf.reduce_sum(tf.square(linear_model - y)) / (2 * test_X.shape[0]),feed_dict={X: test_X, y: test_y})

 print("Final testing cost:", testing_cost)
 print("Absolute mean square loss difference:", abs(training_cost - testing_cost))
 # Display fitted line on test data
 plt.plot(test_X, test_y, 'bo', label='Testing data')
 plt.plot(train_X, sess.run(W) * train_X + sess.run(b), label='Fitted line')
 plt.legend()
 plt.show()
```

4.3. Từ tensorflow đến keras. Trình bày hiểu biết của mình về keras và 5 ví dụ minh họa

Keras là một giao diện lập trình ứng dụng mạng nơ-ron (API) cho Python được tích hợp chặt chẽ với TensorFlow, được sử dụng để xây dựng các mô hình học máy. Các mô hình của Keras cung cấp một cách đơn giản, thân thiện với người dùng để xác định mạng nơ-ron, sau đó sẽ được xây dựng cho bạn bởi TensorFlow.

Mặt khác, Keras hoàn hảo cho những người không có nền tảng vững chắc về Deep Learning, nhưng vẫn muốn làm việc với các mạng thần kinh. Sử dụng Keras, bạn có thể xây dựng một mô hình mạng thần kinh một cách nhanh chóng và dễ dàng bằng cách sử dụng mã tối thiểu, cho phép tạo mẫu nhanh. Chẳng hạn:

```
# Import the Keras libraries required in this example:
from keras.models import Sequential
from keras.layers import Dense, Activation
# Create a Sequential model:
model = Sequential()
# Add layers with the add() method:
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

Keras ít bị lỗi hơn TensorFlow và các mô hình có nhiều khả năng chính xác hơn với Keras so với TensorFlow. Điều này là do Keras hoạt động trong giới hạn của khuôn khổ của nó, bao gồm:

- Tốc độ tính toán: Keras hy sinh tốc độ cho sự thân thiện với người dùng.
- Lỗi cấp thấp: đôi khi bạn sẽ nhận được thông báo lỗi phụ trợ TensorFlow mà Keras không được thiết kế để xử lý.
- Hỗ trợ thuật toán – Keras không phù hợp để làm việc với một số thuật toán và mô hình học máy cơ bản nhất định như phân cụm và Phân tích thành phần chính (PCM).
- Biểu đồ động - Keras không hỗ trợ tạo biểu đồ động.

Tổng quan về mô hình Keras

Mô hình là thực thể cốt lõi mà bạn sẽ làm việc khi sử dụng Keras. Các mô hình được sử dụng để xác định mạng nơ-ron TensorFlow bằng cách chỉ định các thuộc tính, chức năng và lớp bạn muốn.

Keras cung cấp một số API bạn có thể sử dụng để xác định mạng thần kinh của mình, bao gồm:

- API tuần tự, cho phép bạn tạo từng lớp mô hình cho hầu hết các vấn đề. Nó đơn giản (chỉ là một danh sách các lớp đơn giản), nhưng nó bị giới hạn trong các ngăn xếp lớp đầu vào đơn, đầu ra đơn.
- API chức năng, là một API đầy đủ tính năng hỗ trợ các kiến trúc mô hình tùy ý. Nó linh hoạt và phức tạp hơn API tuần tự.
- Model Subclassing, cho phép bạn thực hiện mọi thứ từ đầu. Thích hợp cho nghiên cứu và các trường hợp sử dụng rất phức tạp, nhưng hiếm khi được sử dụng trong thực tế.

Cách xác định mạng nơ-ron với API tuần tự của Keras

Sequential API là một framework để tạo các model dựa trên các instance của lớp sequential(). Mô hình có một biến đầu vào, một lớp ẩn với hai tế bào thần kinh và một

lớp đầu ra với một đầu ra nhị phân. Các lớp bổ sung có thể được tạo và thêm vào mô hình.

```
# Define the model:
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(2, input_dim=1, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Mô hình bao gồm các thông tin sau:

- Các lớp và thứ tự của chúng trong mô hình.
- Hình dạng đầu ra (số lượng phần tử trong mỗi chiều của dữ liệu đầu ra) của mỗi lớp.
- Số lượng tham số (trọng số) trong mỗi lớp.
- Tổng số tham số trong mô hình.

Hàm summary() được sử dụng để tạo và in tóm tắt trong bảng điều khiển Python:

```
# Print a summary of the created model:
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(2, input_dim=1, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
print(model.summary())
```

Cách xác định mạng nơ-ron với API chức năng của Keras

API chức năng Keras cho phép bạn:

- Xác định nhiều mô hình đầu vào hoặc đầu ra
- Xác định các mô hình chia sẻ các lớp
- Tạo biểu đồ mạng tuần hoàn

Các mô hình API chức năng được xác định bằng cách tạo các thể hiện của các lớp và kết nối chúng trực tiếp với nhau theo cặp. Một mô hình sau đó được định nghĩa chỉ định các lớp hoạt động như đầu vào và đầu ra cho mô hình.

Tạo một lớp đầu vào

Trong mô hình API chức năng, không giống như mô hình API tuần tự, trước tiên bạn phải tạo và xác định một lớp đầu vào độc lập chỉ định hình dạng của dữ liệu đầu vào. Lớp đầu vào có một đối số hình dạng là một bộ đại diện cho kích thước của dữ liệu đầu vào. Khi dữ liệu đầu vào là một chiều, hình dạng phải chừa chỗ rõ ràng cho hình dạng của kích thước lô nhỏ được sử dụng khi tách dữ liệu khi đào tạo mạng. Do đó, tuple hình dạng luôn được xác định với một chiều cuối cùng treo, ví dụ. (2,).

```
# Define the input layer:
from keras.layers import Input
```

visible = Input(shape=(2,))
Các lớp trong mô hình được kết nối theo cặp bằng cách chỉ định đầu vào đến từ đâu khi xác định từng lớp mới. Một ký hiệu dấu ngoặc được sử dụng, chỉ định lớp đầu vào.
# Connect the layers, then create a hidden layer as a Dense
# that receives input only from the input layer:
from keras.layers import Dense
visible = Input(shape=(2,))
hidden = Dense(2)(visible)
Mô hình API chức năng có được sự linh hoạt của nó bằng cách kết nối các lớp từng phần theo cách này.
Cách sử dụng mô hình Keras để đưa ra dự đoán
Sau khi một mô hình được xác định bằng API tuần tự hoặc chức năng, các hàm khác nhau cần được tạo để chuẩn bị đào tạo và lắp mô hình, trước khi chúng ta có thể sử dụng nó để đưa ra dự đoán:
Trong ví dụ này, một mô hình tuần tự Keras được triển khai để phù hợp và dự đoán dữ liệu hồi quy:
# PREPARE THE DATA
# Import libraries required in this example:
import random
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.metrics import mean_squared_error
# Generate a sample dataset from random data:
random.seed(123)
def CreateDataset(N):
 a,b,c,y = [],[],[],[]
 for i in range(N):
  aa = i/10+random.uniform(-4,3)
  bb = i/30+random.uniform(-4,4)
  cc = i/40+random.uniform(-3,3)-5
  yy = (aa+bb+cc/2)/3
  a.append([aa])
  b.append([bb])
  c.append([cc])
  y.append([yy])
 return np.hstack([a,b,c]), np.array(y)
N = 150
x,y = CreateDataset(N)
x_ax = range(N)
plt.plot(x_ax, x, 'o', label="original value", markersize=3)

```
plt.plot(x_ax, y, lw=1.5, color="red", label="y")
plt.legend(['original value'])
plt.show()
```

4.4. Chạy 3 ví dụ dưới đây và giải thích

```
#add thư viện
import tensorflow as tf
import numpy as np
#print("TensorFlow version:", tf.__version__)

#tải bộ dữ liệu MNIST
mnist = tf.keras.datasets.mnist

#x_ bộ dữ liệu hình ảnh(chia thành tập huấn luyện và tập kiểm tra)
#y_ là các nhãn tương ứng
#Bộ dữ liệu MNIST chứa các hình ảnh của các chữ số viết tay từ 0 đến 9.
(x_train, y_train), (x_test, y_test) = mnist.load_data()

#chuẩn hóa giá trị pixel của hình ảnh 0 tới 255.
#chia cho 255 để giá trị pixel chạy từ 0 tới 1, giúp mô hình học tốt hơn
x_train, x_test = x_train / 255.0, x_test / 255.0

#tạo mô hình nơ-ron sử dụng lớp 'Sequential'
model = tf.keras.models.Sequential([
tf.keras.layers.Flatten(input_shape=(28, 28)), #được sử dụng để làm phẳng hình ảnh 28x28 thành 1 vector 2D có 784 phần tử
tf.keras.layers.Dense(128, activation='relu'), #với 128 đơn vị và hàm kích hoạt ReLU
tf.keras.layers.Dropout(0.2), #với tỉ lệ dropout 0.2 để tránh overfitting
tf.keras.layers.Dense(10) #10 đơn vị đầu ra để dự đoán lớp của các chữ số
])

#dự đoán lớp cho 1 ví dụ từ tập huấn luyện x_train[:1] và được lưu vào biến pre
predictions = model(x_train[:1])
predictions

#xác định hàm mất mát. sử dụng cho bài toán phân loại 3 lớp, trong đó các nhãn là số nguyên
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
#tính giá trị hàm mất mát giữa dự đoán predictions và nhãn thực tế y_train[:1]
loss_fn(y_train[:1], predictions)

#Biên soạn mô hình để huấn luyện.
#Sử dụng trình tối ưu hóa 'adam', hàm mất mát đã được xác định ở trước
#và độ đo 'accuracy' để theo dõi hiệu suất mô hình.
model.compile(optimizer='adam',
loss=loss_fn,
metrics=['accuracy'])
#Huấn luyện mô hình trên tập huấn luyện (x_train và y_train) trong 5 epochs (vòng lặp qua toàn bộ tập dữ liệu huấn luyện).
#Mục tiêu là cải thiện mô hình để dự đoán chính xác hơn.
model.fit(x_train, y_train, epochs=5)

#Đánh giá hiệu suất của mô hình trên tập kiểm tra (x_test và y_test).
#Kết quả bao gồm giá trị mất mát và độ chính xác của mô hình trên tập kiểm tra.
#verbose=2 chỉ định cách hiển thị thông tin đánh giá.
model.evaluate(x_test, y_test, verbose=2)

#dùng để huấn luyện và đánh giá một dạng nơ-ron sử dụng dữ liệu từ bộ dữ liệu MNIST
```

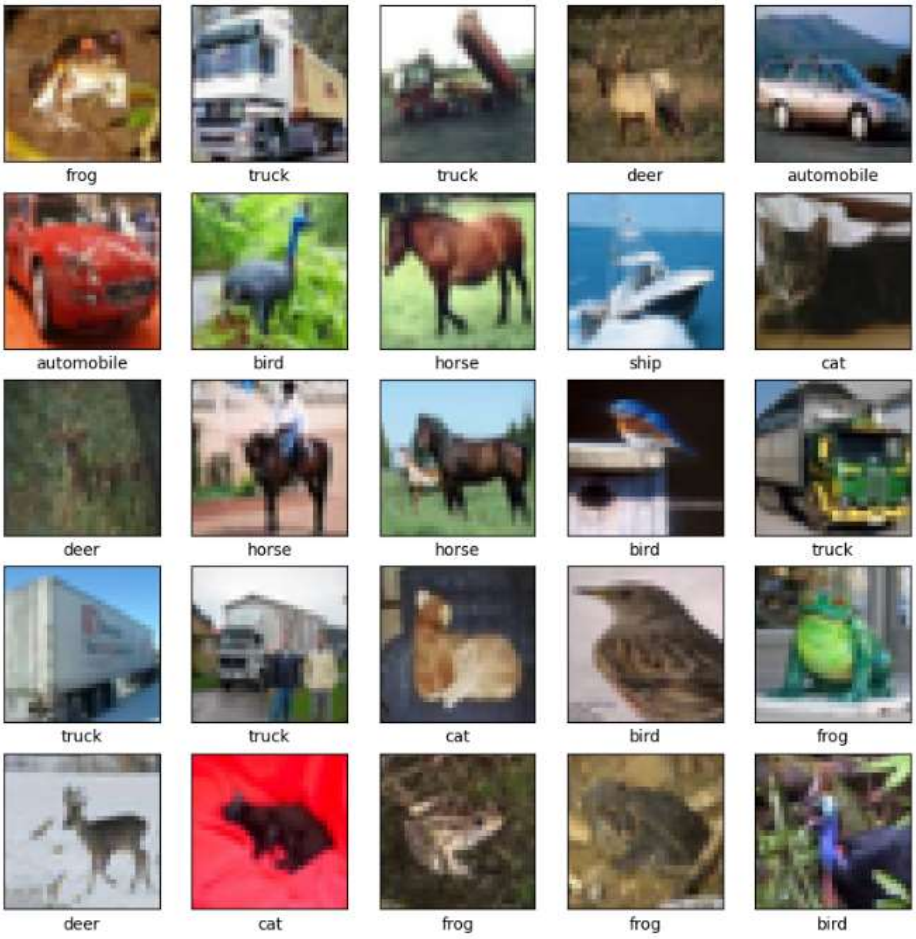
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 2s 0us/step
Epoch 1/5
1875/1875 [==============================] - 5s 2ms/step - loss: 0.2972 - accuracy: 0.9136
Epoch 2/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1411 - accuracy: 0.9580
Epoch 3/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1058 - accuracy: 0.9679
Epoch 4/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0865 - accuracy: 0.9732
Epoch 5/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0757 - accuracy: 0.9759
313/313 - 1s - loss: 0.0697 - accuracy: 0.9788 - 739ms/epoch - 2ms/step

[0.06969288736581802, 0.9787999987602234]
```

Vd2:

In [2]:

```
# xây dựng và huấn luyện một mạng nơ-ron sử dụng dữ liệu từ bộ dữ liệu CIFAR-1
#TensorFlow, TensorFlow Keras (cho việc xây dựng mô hình), và Matplotlib (để hiển thị hình ảnh).
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt

#Tải bộ dữ liệu CIFAR-10,
#chứa hình ảnh của 10 loại đối tượng khác nhau,
#mỗi loại có 6,000 hình ảnh trong tập huấn luyện và 1,000 hình ảnh trong tập kiểm tra.
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
#chuẩn hóa giá trị pixel của ảnh từ 0 tới 255 thành 0 1 để giúp mô hình hoạt động tốt hơn
train_images, test_images = train_images / 255.0, test_images / 255.0

#định nghĩa  một danh sách tên lớp tương ứng với các số nguyên trong nhãn CIFAR-10. Điều này sẽ giúp chúng ta hiển thị tên lớp th
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
'dog', 'frog', 'horse', 'ship', 'truck']

#tạo hình vẽ có kích thước 10x10 inch để hiển thị các ảnh
plt.figure(figsize=(10,10))

#vòng lặp 25 để hiển thị 25 ảnh từ tập huấn luyện trong 1 lưới 5x5
#Các hình ảnh được lấy từ train_images, và tên lớp tương ứng được hiển thị bên dưới mỗi hình ảnh.
for i in range(25):
    plt.subplot(5,5,i+1) #tạo ô con trong lưới 5x5, hiển thị ảnh có đối số i+1
    plt.xticks([])  #loại bỏ các dấu chấm trên trục x/y của ô con giúp chỉ hiện thị ảnh
    plt.yticks([])
    plt.grid(False)  #tắt các lưới trên hình ảnh
    plt.imshow(train_images[i])  #hiển thị ảnh lên ô con nhưng chưa hiển thị trên màn hình

    # sử dụng train_labels[i][0] để truy cập nhãn của hình ảnh thứ i
    #sau đó dùng nhãn này làm chỉ mục để lấy tên lớp tương ứng từ danh sách class_names.
    plt.xlabel(class_names[train_labels[i][0]])

#hiển thị ảnh lên màn hình
plt.show()

#Mục đích chính là để kiểm tra và hiểu về dữ liệu trước khi xây dựng và huấn
#Luyện một mô hình máy học trên bộ dữ liệu này.
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 161s 1us/step
```

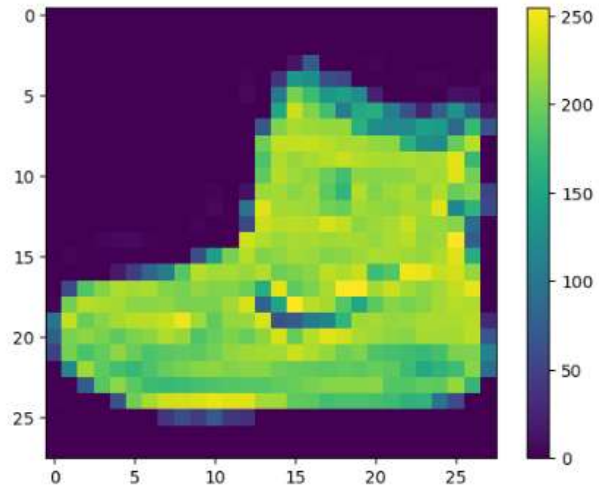


Vd3:

In [4]:

```
#xây dựng và huấn luyện mô hình
import tensorflow as tf
# thực hiện các phép toán số học và hình ảnh
import numpy as np
import matplotlib.pyplot as plt
#hiển thị phiên bản
print(tf.__version__)

#tải dữ liệu Fashion MNIST
fashion_mnist = tf.keras.datasets.fashion_mnist
#chia dữ liệu tải về thành tập huấn luyện và tập kiểm tra
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

#định nghĩa danh sách tên lớp tương ứng với các nhãn là số trong tập dữ liệu
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat','Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

#kiểm tra kích thước của tập huấn luyện
train_images.shape

#hiển thị hình ảnh đầu tiên từ tập huấn luyện
plt.figure()
plt.imshow(train_images[0])
plt.colorbar() #hiển thị thêm màu cho biết giá trị màu sác tương ứng với từng điểm ảnh
plt.grid(False) #tắt lưới hình ảnh
plt.show() #làm cho hình ảnh hiện lên màn

#chuẩn hóa giá trị pixel cho ảnh từ 0 tới 1
train_images = train_images / 255.0
test_images = test_images / 255.0

#tạo hình vẽ có kích thước 10x10 inch để hiển thị các ảnh để chứa 1 lưới 5x5 chứa 25 ảnh
plt.figure(figsize=(10,10))

for i in range(25):
    plt.subplot(5,5,i+1)  #tạo ô con trong lưới 5x5, hiển thị ảnh có đối số i+1
    plt.xticks([])  #Loại bỏ các dấu chấm trên các trục
    plt.yticks([])
    plt.grid(False) #tắt lưới trên hình ảnh
    plt.imshow(train_images[i], cmap=plt.cm.binary) #hiển hị ảnh từ tập huấn luyện và để dưới dang trắng đen
    plt.xlabel(class_names[train_labels[i]]) #đặt tên lớp tương ứng dưới mỗi ảnh
plt.show() #hiển thị toàn bộ ảnh lên màn

#Mục đích chung của đoạn mã này là tải dữ liệu Fashion MNIST,
#hiển thị một số hình ảnh từ tập huấn luyện cùng với tên lớp tương ứng để kiểm tra
#và hiểu dữ liệu trước khi xây dựng và huấn luyện một mô hình máy học trên bộ dữ liệu này.
```

```
2.13.0
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 4s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0s/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 1s 0us/step
```




Ankle boot
T-shirt/top
T-shirt/top
Dress
T-shirt/top
Pullover
Sneaker
Pullover
Sandal
Sandal
T-shirt/top
Ankle boot
Sandal
Sandal
Sneaker
Ankle boot
Trouser
T-shirt/top
Shirt
Coat
Dress
Trouser
Coat
Bag
Coat