## Case study 3:Phân loại ảnh thời trang

### Import thư viện

import pandas as pd

import numpy as np

import tensorflow as tf

import matplotlib

import matplotlib.pyplot as plt

from random import \*

### Thu thập dữ liệu và load dữ liệu

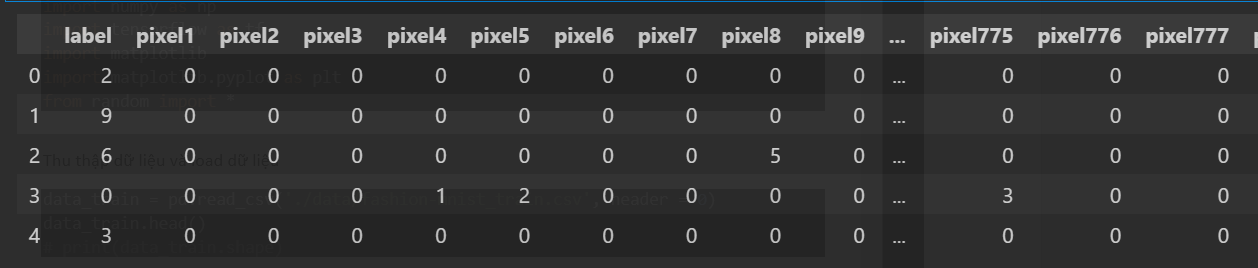
Load dữ liệu dử dụng pandas

data\_train = pd.read\_csv('./data/fashion-mnist\_train.csv', header = 0)

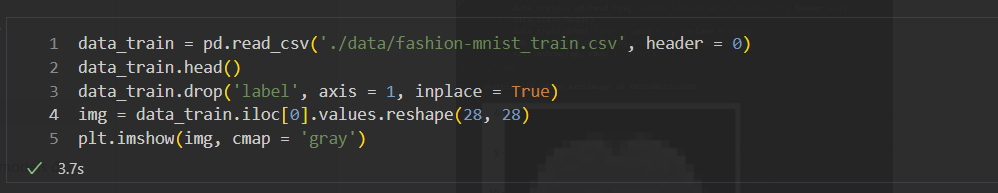
data\_train.head()

# print(data\_train.shape)

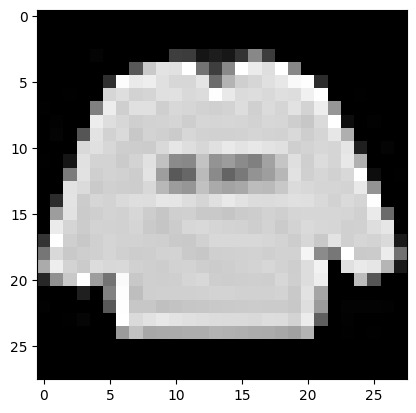
Output:



Hiển thị ảnh:



Output:



Cột labels

# tensorflow yêu cầu labels là 1 vector 2 chiều

labels = data\_train['label'].values.reshape(1, 60000)

# tập train có 60000 ảnh 28x28=784 pixels

train = data\_train.drop('label', axis=1).transpose()

print(labels.shape)

# đưa giá trị pixels về 0-1 bằng cách chia cho 255 để giảm độ lớn của giá trị

train = np.array(train / 255.0)

print(train)

reshape label thành mảng 2 chiều vì bước sau nhân ma trận nên phải là 2 chiều trong khi ban đầu là 1 chiều

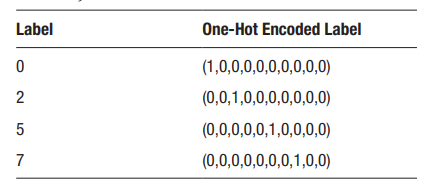
### Xây dựng mạng neural với tensorflow

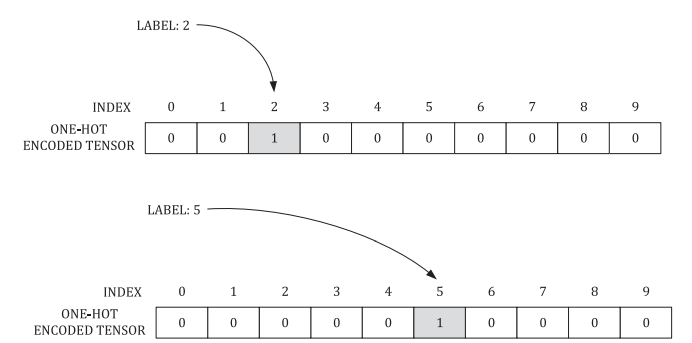
import tensorflow.compat.v1 as tf

tf.disable\_v2\_behavior()

label là kết quả mong muốn, y\_ là dự đoán của model nên 2 tensor phải có cùng kích thước. Mạng sẽ dự đoán ra 1 vector có 10 phần tử, trong khi một nhãn trong bộ dữ liệu chỉ là một phần tử(một số).

y\_ kích thước (10,1), Y (1,1) nên phải thay đổi labels ban đầu thành 1 ma trận như sau:





labels\_ = np.zeros((60000, 10))

labels\_[np.arange(60000), labels] = 1

labels\_ = labels\_.transpose()

labels\_ = np.array(labels\_)

Đầu vào là 784 pixel, ảnh kích thước 28\*28

n\_dim = 784

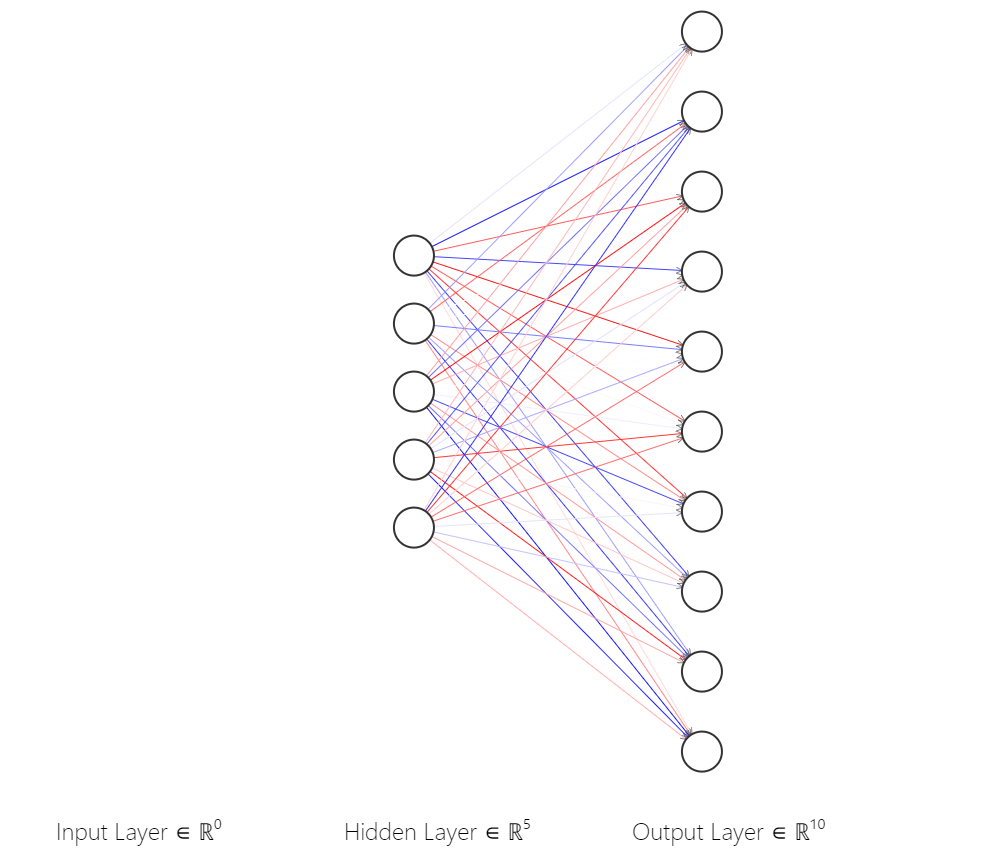
tf.reset\_default\_graph()

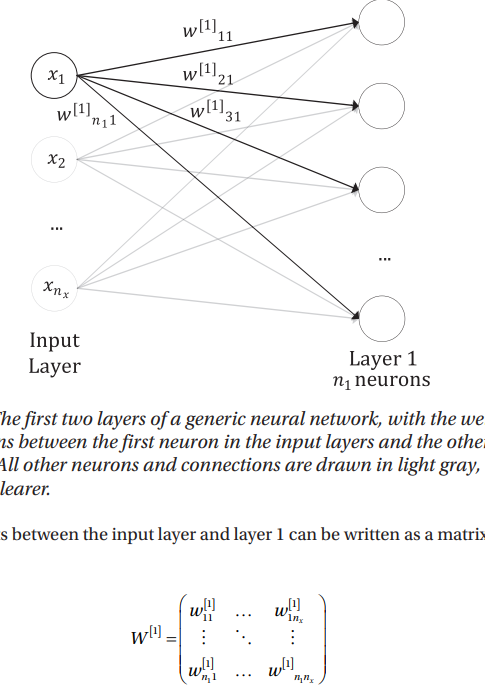
# Number of neurons in the layers

n1 = 5 # Number of neurons in layer 1

n2 = 10 # Number of neurons in output layer

mạng neural như sau:





Trọng số giữa layer đầu vào với layer 1 là W như hình trên.

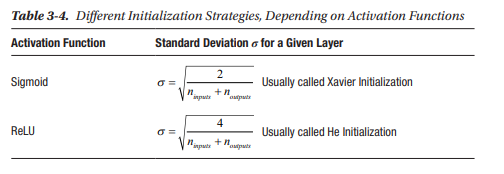
Sử dụng hàm phân phối chuẩn để khởi tạo ma trận trọng số với độ lệch chuẩn như sau

# standard deviation = stddev = 0.1

# độ lệch chuẩn = 0.1 bởi vì khi tính toán giá trị của z1 ứng với neural 1, z2, z3,… thì giá trị của nó sẽ rất lớn

# khi z1 rất lớn thì hàm relu của python sẽ có khả năng trả về trá trị NaN

**Cách chọn stddev**



Tại layer L, số lượng neural của layer L-1 là **n** inputs = n1, Layer của L+1 là **n** ouput = n2

Ví dụ tại layer 3: xem xét relu activation function, độ lệch chuẩn tính như sau:

**stddev = sqrt(4 / sqrt(n4+n2))**

cost\_history = np.empty(shape=[1], dtype = float)

learning\_rate = tf.placeholder(tf.float32, shape=())

X = tf.placeholder(tf.float32, [n\_dim, None])

Y = tf.placeholder(tf.float32, [10, None])

W1 = tf.Variable(tf. truncated\_normal ([n1, n\_dim], stddev=.1))

b1 = tf.Variable(tf.zeros([n1,1]))

W2 = tf.Variable(tf. truncated\_normal ([n2, n1], stddev=.1))

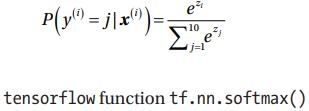
b2 = tf.Variable(tf.zeros([n2,1]))

# Let's build our network...

Z1 = tf.nn.relu(tf.matmul(W1, X) + b1) # tính tích

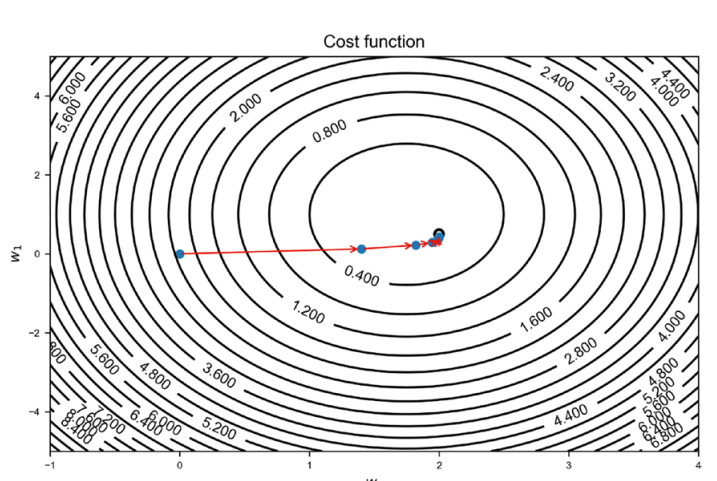
Z2 = tf.nn.relu(tf.matmul(W2, Z1) + b2)

Sử dụng hàm softmax để phân loại nhiều nhãn

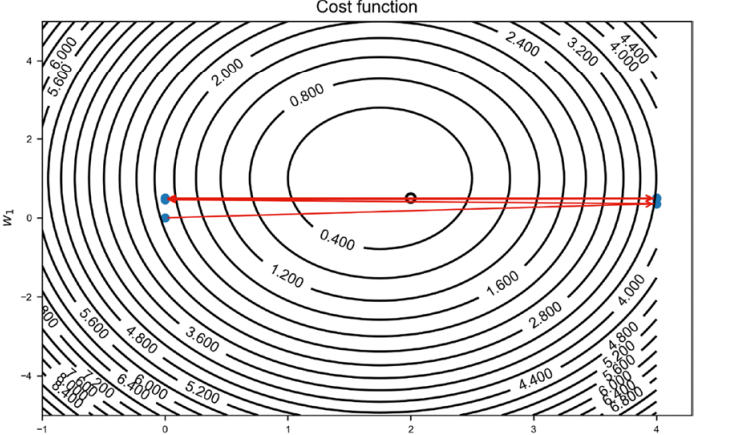


y\_ = tf.nn.softmax(Z2,0)

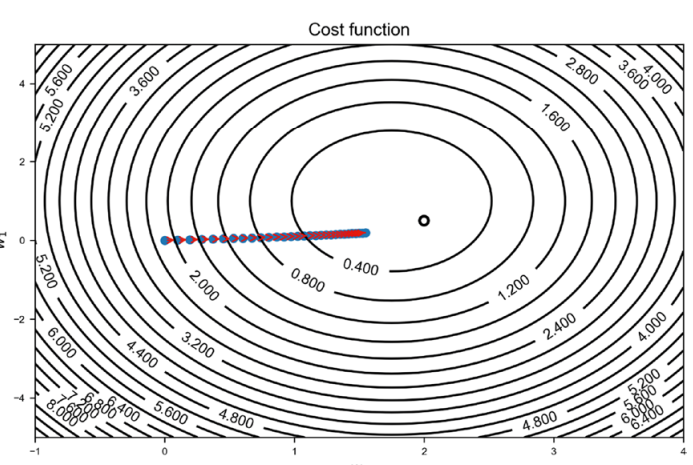
Mục tiêu của việc huấn luyện mô hình đó chính là việc cực tiểu hoá - **minimize** hàm cost function, tức là phải thay đổi các tham số trong mạng để hàm cost của chúng ta thay đổi theo chiều hướng nhỏ dần giá trị. Công việc đó được xử lý bằng tf.train.GradientDescentOptimizer. Hàm tối ưu - optimizer này sẽ có một tham số là learning\_rate = 0.001 để giúp điều chỉnh tốc độ học.



Khi sử dụng learning\_rate hợp lý thì hàm cost function hội tụ tốt hơn càng gần 0 thì model càng chính xác



Khi learning\_rate lớn thì không thể hội tụ tại điểm tối thiểu



Khi learning\_rate quá nhỏ thì hội tụ rất chậm

Không có quy tắc chọn learning\_rate, thử nhiều learning\_rate khác nhau và chọn ra tỉ lệ tốt nhất.

cost = - tf.reduce\_mean(Y \* tf.log(y\_)+(1-Y) \* tf.log(1-y\_))

optimizer = tf.train.AdamOptimizer(learning\_rate).minimize(cost)

init = tf.global\_variables\_initializer()

### sử dụng session() để tiến hành training models đã xây dựng:

sess = tf.Session()

sess.run(tf.global\_variables\_initializer())

training\_epochs = 100

cost\_history = []

for epoch in range(training\_epochs+1):

    sess.run(optimizer, feed\_dict = {X: train, Y: labels\_, learning\_rate:0.001})

    cost\_ = sess.run(cost, feed\_dict={ X:train, Y: labels\_, learning\_rate:0.001})

    cost\_history = np.append(cost\_history, cost\_)

    if (epoch % 20 == 0):

        print("Reached epoch",epoch,"cost J =", cost\_)

với epoch=100

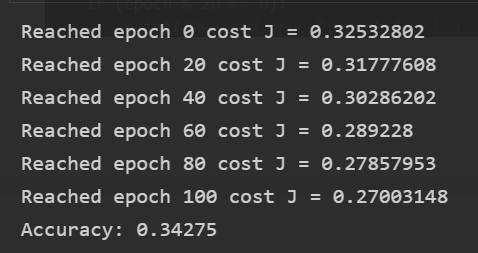
### Đánh giá mô hình

correct\_predictions = tf.equal(tf.argmax(y\_,0), tf.argmax(Y,0))

accuracy = tf.reduce\_mean(tf.cast(correct\_predictions, "float"))

print ("Accuracy:", accuracy.eval({X: train, Y: labels\_, learning\_rate:0.001}, session = sess))

output:



### Đánh giá mô hình trên tập test

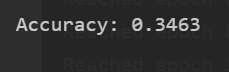
correct\_predictions = tf.equal(tf.argmax(y\_,0), tf.argmax(Y,0))

accuracy = tf.reduce\_mean(tf.cast(correct\_predictions, "float"))

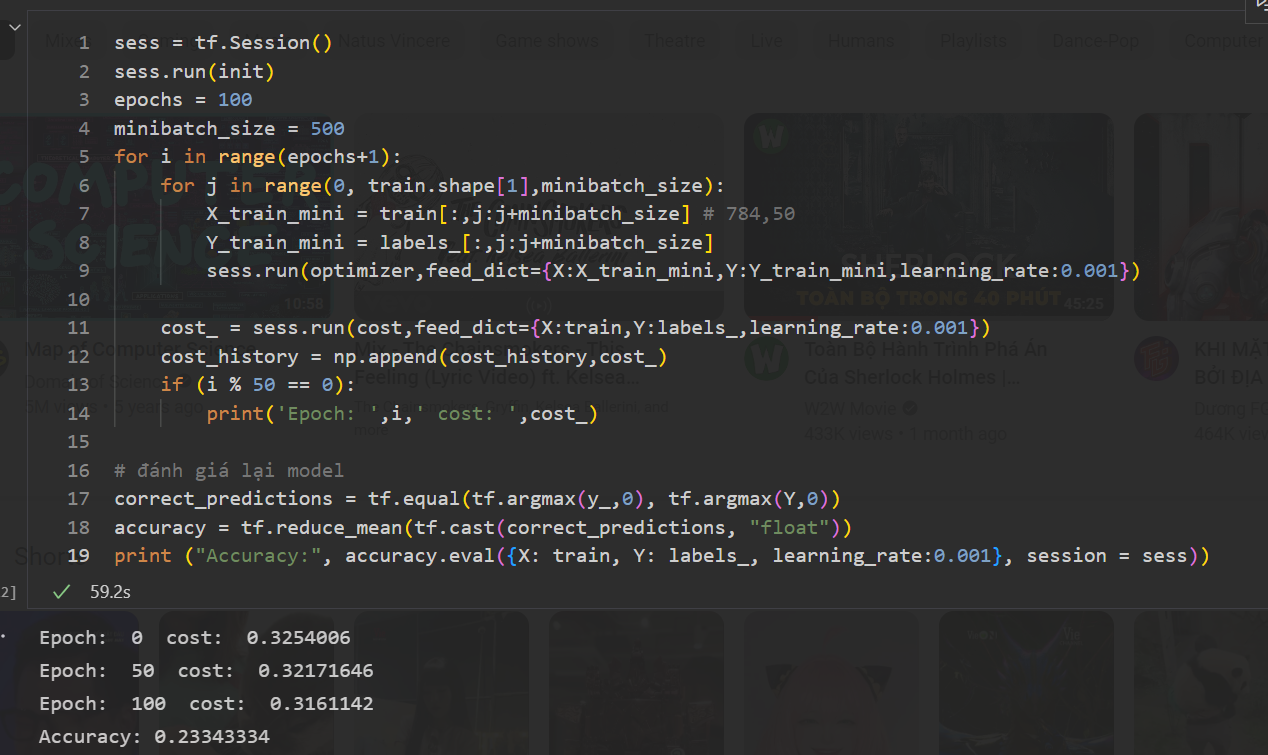
print ("Accuracy:", accuracy.eval({X: dev, Y: labels\_dev\_, learning\_rate:

0.001}, session = sess))

Output:



Huấn luyện model với learning\_rate:0.001, batch\_size:500, optimizer: Gradient descent

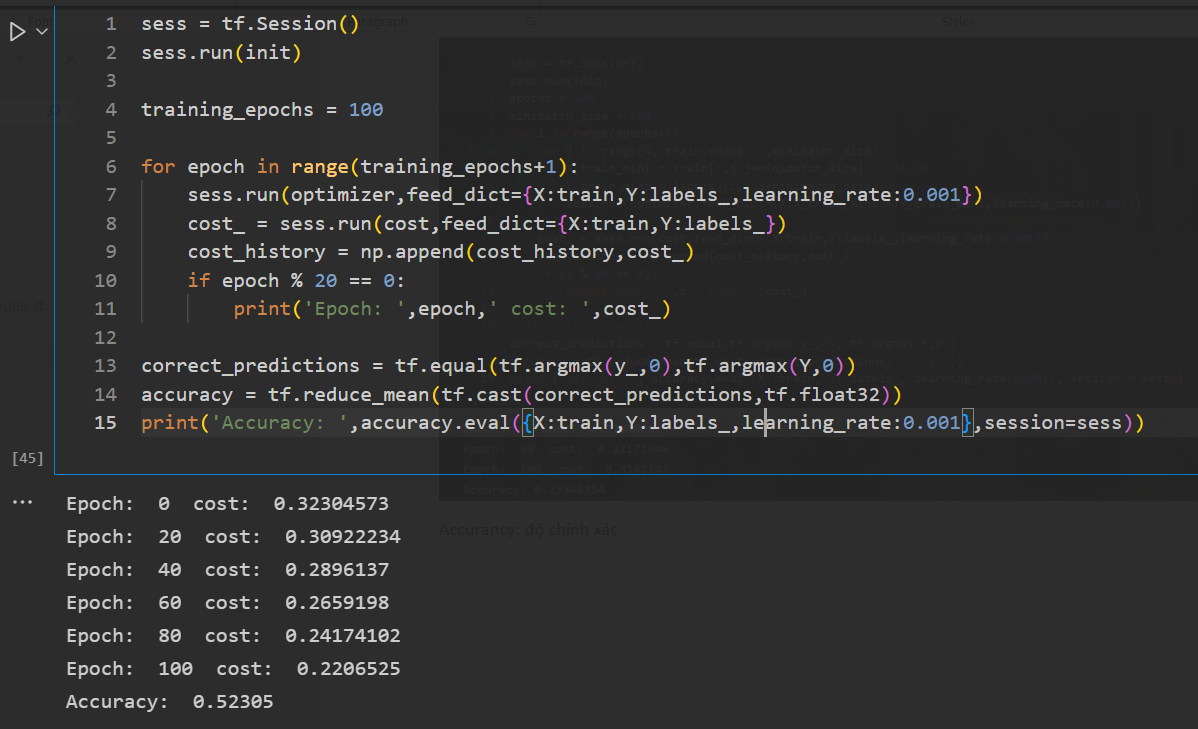


Accurancy: độ chính xác

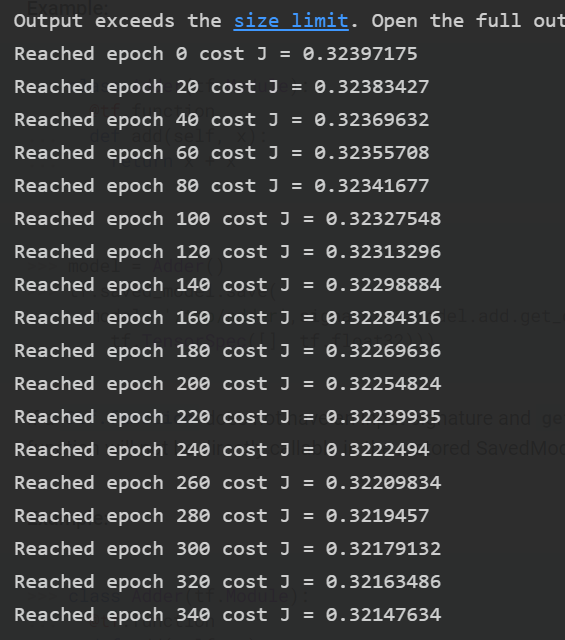
Với mạng neural 1 lớp hidden bên trên:

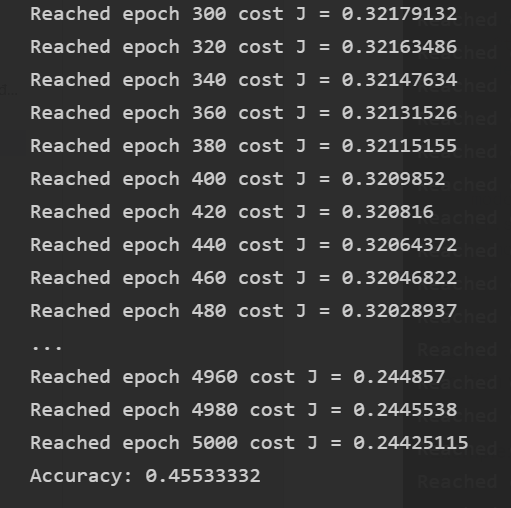
Điều chỉnh với optimizer là Adam, learning\_rate = 0.001

Độ nhớ của model đạt 52%



Train model với epochs = 5000





Vẽ biểu đồ cost:

