#coding=utf-8

import tensorflow.\_api.v2.compat.v1 as tf#用tensorflow2.8.0的调用1.0

tf.disable\_v2\_behavior()#调整tensorflow的1.0与2.0的版本兼容问题

import numpy as np

from flask import Flask, jsonify

from flask\_cors import cross\_origin

app = Flask(\_\_name\_\_)

@app.route("/Model\_Use/<data>", methods=["GET"])

@cross\_origin(supports\_credentials=True)

def fun0(data):

X = tf.placeholder("float")

Y = tf.placeholder("float")

# 定义学习参数

W = tf.Variable(tf.constant(data[0]), name="weight") # data[0]是权重

b = tf.Variable(tf.compat.v1.random\_normal([1]), name="bias") # -1到1的随机1个零维的数字

# 定义运算

z = tf.reduce\_sum([X, W]) + b # tf.multiply用于两数相乘

# 定义损失函数(进行反向模型)

# Step3：迭代训练

# 定义Saver对象和保存路径(用Saver对象载入模型时放在定义其他参数下面)

saver = tf.train.Saver()

savedir = "log/"

# 初始化所有变量

init = tf.global\_variables\_initializer() # tf.global\_variables\_initializer用于初始化所有变量并声明

with tf.Session() as sess:

sess.run(init) # 将数据传入Session(画布)

saver.restore(sess, savedir + "linermodel.cpkt") # 用saver.restore载入模型写保存文件路径

print("x=[0.2,0.1,0.2],z=", sess.run(z, feed\_dict={X: data[1]})) # 用模型(z是预测数据,data[1]是特征数据)

return z

@app.route("/Model\_Updata/<data>", methods=["GET"])

@cross\_origin(supports\_credentials=True)

def fun1(data):

# Step1：写数据

train\_X = []

for i in range(10):

t = np.linspace(-1, 1, 3) # -1到1的随机3个数

train\_X.append(t)

train\_Y = np.linspace(-1, 1, 10)

# Step2：准备模型

# 定义占位符(进行正向模型)

X = tf.placeholder("float")

Y = tf.placeholder("float")

# 初始化学习参数

W = tf.Variable(tf.constant(data), name="weight") # 全0的1个一维的数字106个w

b = tf.Variable(tf.compat.v1.random\_normal([1]), name="bias") # -1到1的随机1个零维的数字

# 定义运算

z = tf.reduce\_sum([X, W]) + b # tf.multiply用于两数相乘

# 定义损失函数(进行反向模型)

cost = tf.reduce\_mean(tf.square(Y - z)) # tf.reduce\_mean用于一维的数字求平均值

# 定义学习率(学习率绝对的循环次数，成反比)

learning\_rate = 0.01

# 设置优化函数(用梯度下降)

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

# Step3：迭代训练

# 初始化所有变量

init = tf.global\_variables\_initializer() # tf.global\_variables\_initializer用于初始化所有变量并声明

# 定义迭代次数

training\_epochs = 20

display\_step = 2 # 设置查看节点

# 启动Session

saver = tf.train.Saver({"weight": W, "bias": b}) # 将W,b先给保存文件(也可用于在存完后修改W，b)

saverdir = "log/" # 相对路径(没有文件夹就创建)

with tf.Session() as sess:

sess.run(init) # 将数据传入Session(画布)

for epoch in range(training\_epochs):

for (x, y) in zip(train\_X, train\_Y):

sess.run(optimizer, feed\_dict={X: x, Y: y}) # 放入优化函数及用占位符放入训练数据

if epoch % display\_step == 0: # 用与在节点上看数据情况

loss = sess.run(cost, feed\_dict={X: x, Y: y})

print("epoch:", epoch + 1, "cost=", loss, "w=", sess.run(W), "b=", sess.run(b))

print("Finished")

saver.save(sess, saverdir + "linermodel.cpkt") # 用saver.save(画布,保存路径)保存模型

# Step4：用模型

# print("x=[0.2,0.1,0.2],z=", sess.run(z, feed\_dict={X: [0.2, 0.1, 0.2]}))

# print("w:", sess.run(W), "b", sess.run(b))

ww = sess.run(W)#ww是权重

bb = sess.run(b)

return ww

if \_\_name\_\_ == '\_\_main\_\_':

app.run()