1. Tf2相关函数介绍
2. tf2中的自动求导机制

tf.GradientTape()自动进行求导，在tf.GradientTape()的上下文内，所有的变量和计算步骤都会被记录用于求导。

import tensorflow as tf

2.

3. x = tf.Variable(initial\_value=3.)

4. with tf.GradientTape() as tape: # 在 tf.GradientTape() 的上下文内，所有计算步骤

都会被记录以用于求导

5. y = tf.square(x)

6. y\_grad = tape.gradient(y, x) # 计算y关于x的导数

7. print([y, y\_grad])

求导用tape.gradient(y, x)进行，求张量y 对变量x的导数。

输出是：  
[array([9.], dtype=float32), array([6.], dtype=float32)]

1. 设置优化器

线性回归用SGD随机梯度下降进行优化：

optimizer = tf.keras.optimizers.SGD(learning\_rate=1e-3)

1. 自动根据梯度更新参数

当求解出损失函数关于变量的梯度时，我们可以根据这些梯度进行更新参数，依靠的是优化器。注意：括号内的参数是损失函数对变量求导的梯度和变量的列表。

optimizer.apply\_gradients(grads\_and\_vars=zip(grads, variables))

1. zip函数

zip（a, b）将a和b对应的数都合成一个个元组，返回由这些元组组成的列表。

如：

a = [1, 3, 5] ， b = [2, 4, 6] ，那么 zip(a, b) = [(1, 2), (3, 4), …, (5, 6)]

1. 变量

线性回归中要不断地更新参数，参数需要用变量来设置。

a = tf.Variable(initial\_value=0.)

5. b = tf.Variable(initial\_value=0.)

1. tf2实现线性回归

import tensorflow as tf  
import numpy as np  
import matplotlib.pyplot as plt  
  
***#随机1000个点，y = 0.1 x + 0.3***num\_points = 1000  
vector\_set = []  
for i in range(num\_points):  
 x1 = np.random.normal(0.0, 0.55)  
 y1 = x1 \* 0.1 + 0.3 + np.random.normal(0., 0.03)  
 vector\_set.append([x1, y1])  
  
***#生成样本***x\_data = [v[0] for v in vector\_set]  
y\_data = [v[1] for v in vector\_set]  
  
***#画图显示***plt.scatter(x\_data, y\_data, c='r')  
plt.show()  
  
***#参数***w = tf.Variable(tf.random.uniform([1], -1.0, 1.0), name='w')  
b = tf.Variable(tf.zeros([1], name='b'))  
***#需要更新的变量***variables = [w, b]  
  
***#采用梯度下降法优化参数  
#优化器***opt = tf.keras.optimizers.SGD(1e-3)  
***#迭代次数***num\_epoch = 10000  
for e in range(num\_epoch):  
 ***#使用tf.GradientTape()记录损失函数的梯度信息*** with tf.GradientTape() as tape:  
 ***#预测值*** y = w \* x\_data + b  
 ***#损失值*** loss = tf.reduce\_mean(tf.square(y - y\_data), name='loss')  
 ***#自动计算loss 关于更新变量的梯度*** grads = tape.gradient(loss, variables)  
 ***#自动根据梯度更新参数*** opt.apply\_gradients(zip(grads, variables))  
 if(e % 100 == 0):  
 print(variables)

1. 结果校验

我们设置的1000个随机点分布在直线y =0.1x + 0.3 周围。得出的结果不断迭代，如下所示：

[<tf.Variable 'w:0' shape=(1,) dtype=float32, numpy=array([0.10239088], dtype=float32)>, <tf.Variable 'Variable:0' shape=(1,) dtype=float32, numpy=array([0.29980734], dtype=float32)>]

可以看到，10000次迭代后，w参数变为了0.10239088， b参数变为了0.29980734，已经非常地接近正确的参数值了。