深度学习方法与实践实验二

2020 年 3 月 5 日

姓名	董佩杰	学　　号	2016012963
年级	2020 级	指导老师	牛新

一、用 Tensorflow2.1 拟合余弦函数

1. 题目说明：

假设有函数 $y = \cos(ax + b)$，其中 a 为学号倒数第 5 和第 4 位，b 为学号最后两位（例如学号 19020139 对应 $\cos(20x + 39)$）。首先从此函数中以相同步长（点与点之间在 x 轴上距离相同），在 $0 < (ax + b) < 2pi$ 范围内，采样出 2000 个点，然后利用采样的 2000 个点作为特征点进行三次函数拟合(三次函数形式为 $y = w1 \times x + w2 \times x^2 + w3 \times x^3 + b$，其中 wi 为可训练的权值，b 为可训练的偏置值，x 和 y 为输入的训练数据 ）。

2. 具体要求：

1. 要求使用 TF2.x
2. 用 model.fit 和自定义循环两种训练方法实现三次函数拟合的全部流程
3. 分别使用回调函数和 model.save 模式保存拟合的模型
4. 针对两种模型存储方式分别编写模型恢复程序分别，并同时绘制图像
5. 记录和打印保存前和恢复后的 loss,并查看是否一致。

二、完成情况

1.模型构建

下图主要是数据构建以及模型定义：

```
x_t = np.arange(-1 * b / a, (2 * np.pi - b) / a,
                np.pi / (a * 1000),
                dtype=np.float32)

x_t = x_t[:, np.newaxis]

x_train = np.concatenate((x_t, np.power(x_t, 2), np.power(x_t, 3)), axis=1)
y_train = np.cos(a * x_t + b)

train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train)).batch(4)

# define model
inputs = tf.keras.Input(shape=(3, ), name="inputs")
outputs = tf.keras.layers.Dense(units=1, input_dim=3, activation="linear")(inputs)
model = tf.keras.Model(inputs=inputs, outputs=outputs, name="linear")
```

图 1 模型构建

2. 训练流程

下图主要是模型的自定义训练以及测试流程：

```
loss_object = tf.keras.losses.MeanSquaredError()
optimizer = tf.keras.optimizers.Adam(0.01)
train_loss = tf.keras.metrics.Mean(name='train_loss')
test_loss = tf.keras.metrics.Mean(name='test_loss')

#try not use tf.function to debug,time?
@tf.function
def train_step(data, labels):
    with tf.GradientTape() as tape:
        predictions = model(data)
        loss = loss_object(predictions, labels)
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))
    train_loss(loss)
@tf.function
def test_step(data, labels):
    predictions = model(data)
    t_loss = loss_object(predictions, labels)
    test_loss(t_loss)

EPOCHS = 200

for epoch in range(EPOCHS):
    start = time.time()
    # 在下一个epoch开始时，重置评估指标
    train_loss.reset_states()
    test_loss.reset_states()
    for data, label in train_dataset:
        train_step(data, label)
    for data, label in train_dataset:
        test_step(data, label)
    end = time.time()
    template = 'Epoch {}, Loss: {:.3f},  Test Loss: {:.3f}，Time used: {:.2f}'
    print(template.format(epoch + 1, train_loss.result(), test_loss.result(),
                          end - start))
    if epoch % 200 == 199:
        #model save here
        model.save("save_%d_model.h5" % epoch)
```

图 2 使用自定义循环

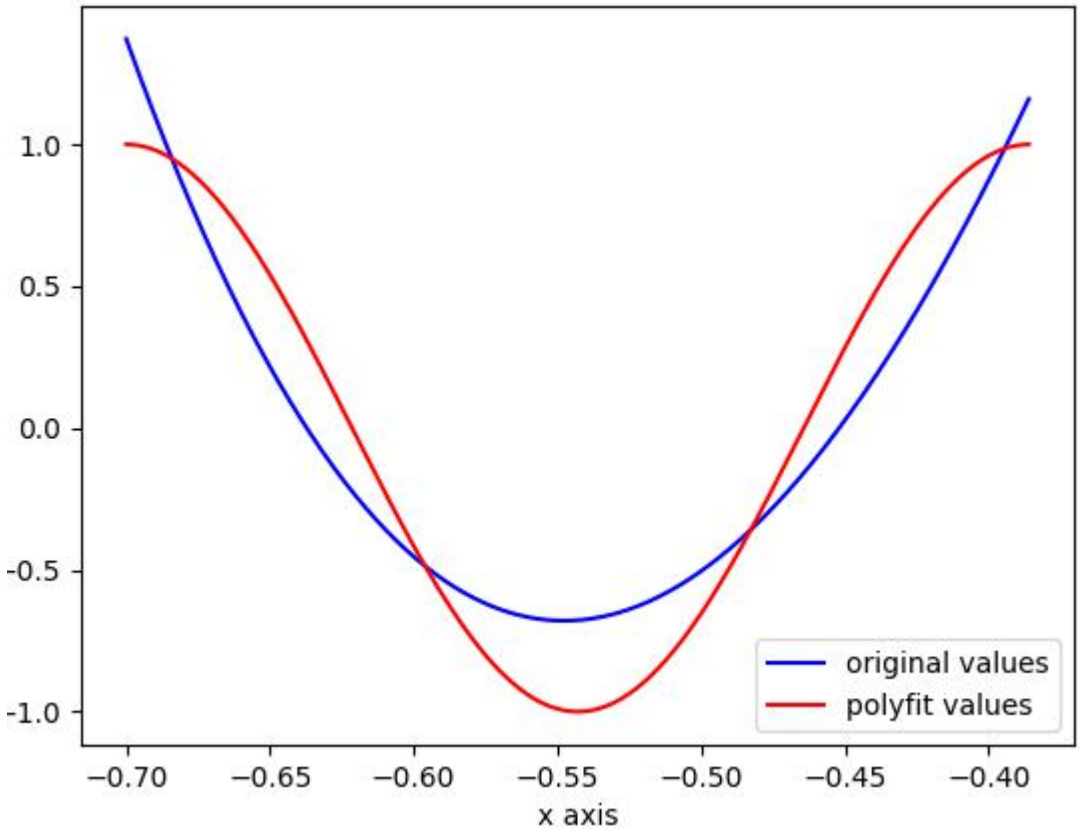
使用 fit 进行训练：

```
checkpoint_path = "./checkpoints/weights.{epoch:02d}.h5"
checkpoint_dir = os.path.dirname(checkpoint_path)
cp_callback = tf.keras.callbacks.ModelCheckpoint(checkpoint_path,
                                                 monitor="mse",
                                                 mode='min',
                                                 save_weights_only=True,
                                                 verbose=1,
                                                 period=5)

Lm1.compile(optimizer=tf.keras.optimizers.Adam(0.01),
            loss='mse',
            metrics=['mse'])

Lm1.fit(x_train, y_train, epochs=1000, batch_size=64, callbacks=[cp_callback])
loss, acc = Lm1.evaluate(x_train, y_train)
print("saved model, loss: {:5.2f}".format(loss))
```

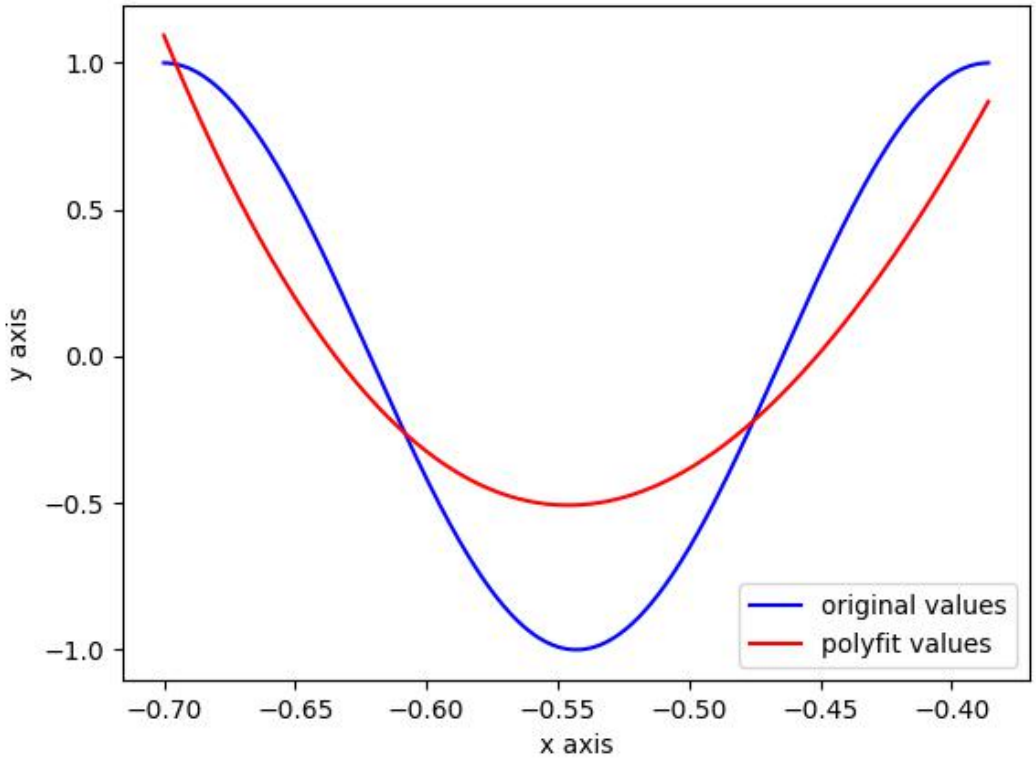
图 3 使用 fit 训练截图

3. 训练结果

以下结果是用自定义循环方式，epoch=10000 的结果，最终 loss=0.05：

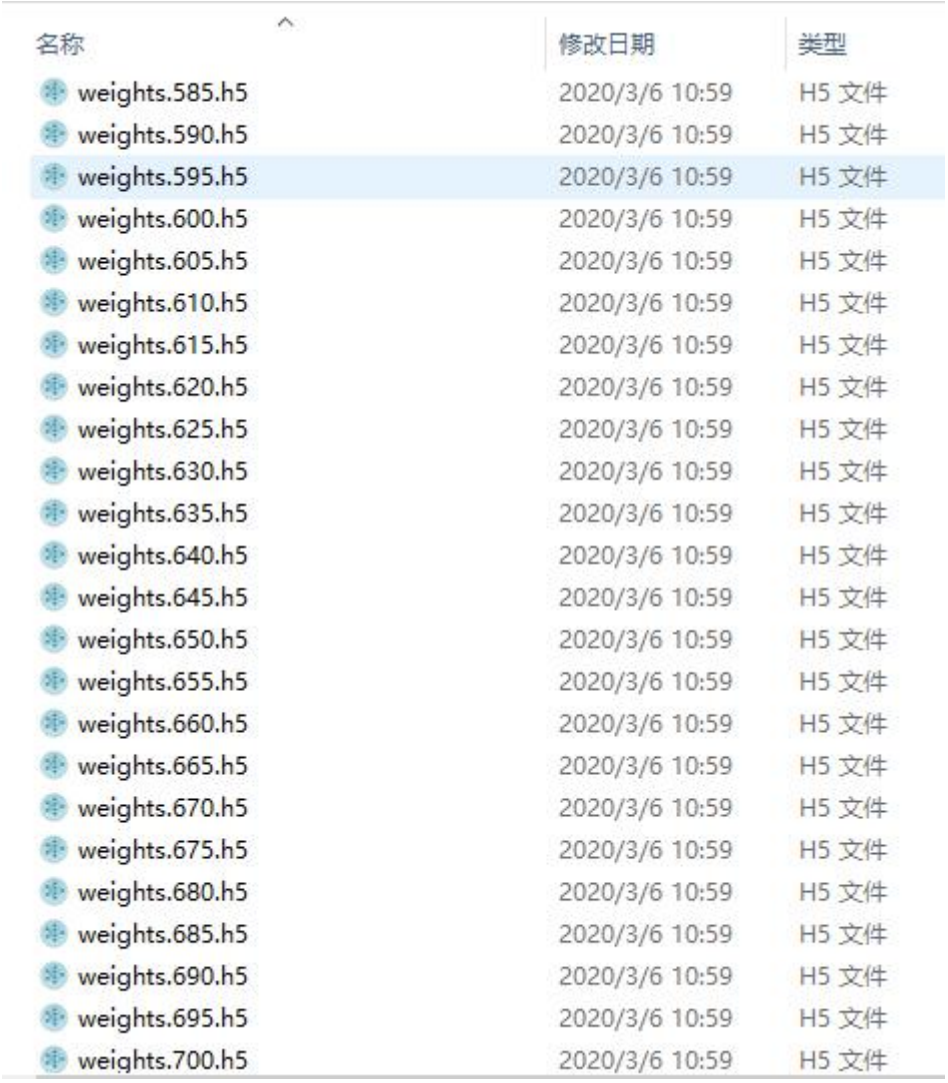


下图结果是使用 fit 方法进行拟合的结果，epoch=1000，最终 loss=0.09：



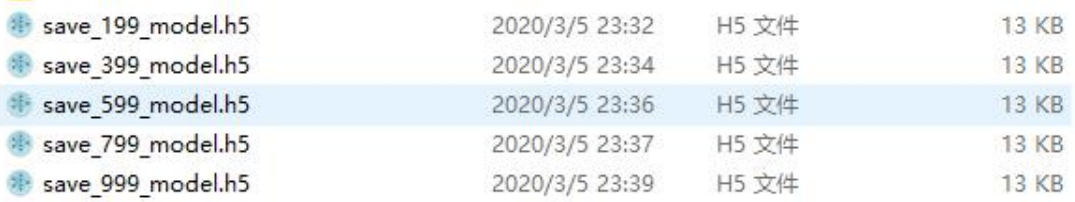
4.使用回调函数和 model.save 模式保存拟合的模型

通过回调函数保存的结果如下：



名称	修改日期	类型
weights.585.h5	2020/3/6 10:59	H5 文件
weights.590.h5	2020/3/6 10:59	H5 文件
weights.595.h5	2020/3/6 10:59	H5 文件
weights.600.h5	2020/3/6 10:59	H5 文件
weights.605.h5	2020/3/6 10:59	H5 文件
weights.610.h5	2020/3/6 10:59	H5 文件
weights.615.h5	2020/3/6 10:59	H5 文件
weights.620.h5	2020/3/6 10:59	H5 文件
weights.625.h5	2020/3/6 10:59	H5 文件
weights.630.h5	2020/3/6 10:59	H5 文件
weights.635.h5	2020/3/6 10:59	H5 文件
weights.640.h5	2020/3/6 10:59	H5 文件
weights.645.h5	2020/3/6 10:59	H5 文件
weights.650.h5	2020/3/6 10:59	H5 文件
weights.655.h5	2020/3/6 10:59	H5 文件
weights.660.h5	2020/3/6 10:59	H5 文件
weights.665.h5	2020/3/6 10:59	H5 文件
weights.670.h5	2020/3/6 10:59	H5 文件
weights.675.h5	2020/3/6 10:59	H5 文件
weights.680.h5	2020/3/6 10:59	H5 文件
weights.685.h5	2020/3/6 10:59	H5 文件
weights.690.h5	2020/3/6 10:59	H5 文件
weights.695.h5	2020/3/6 10:59	H5 文件
weights.700.h5	2020/3/6 10:59	H5 文件

通过 model.save 保存结果如下



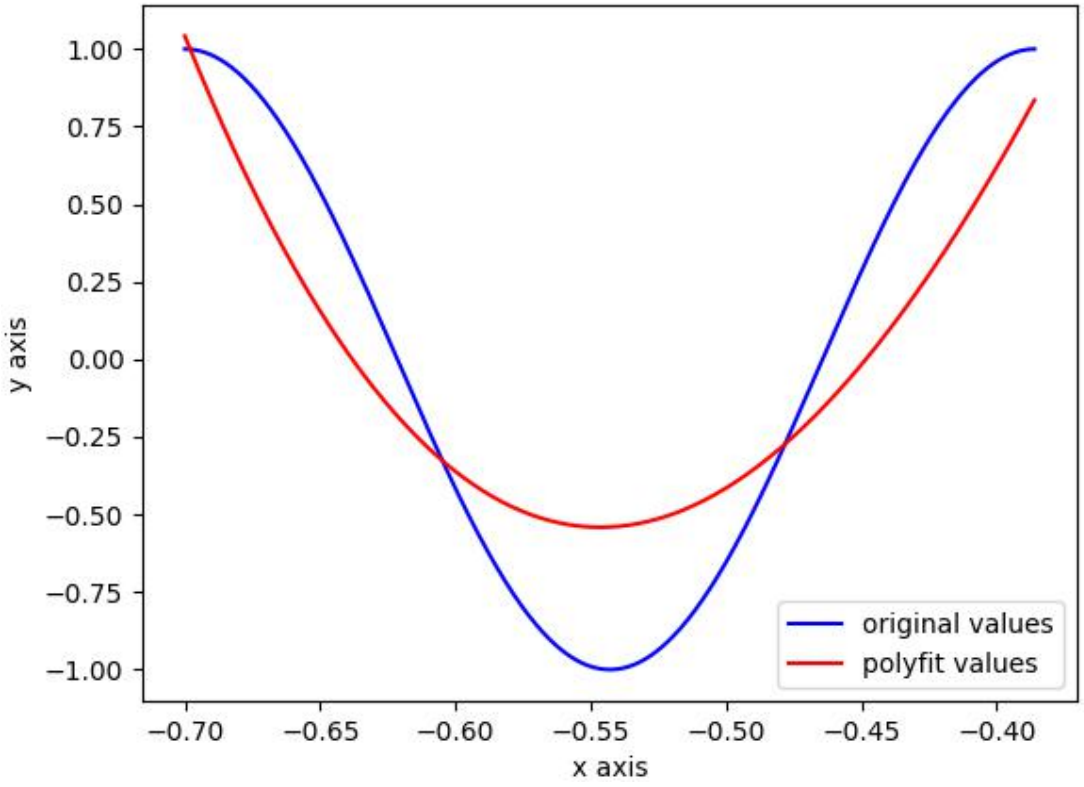
save_199_model.h5	2020/3/5 23:32	H5 文件	13 KB
save_399_model.h5	2020/3/5 23:34	H5 文件	13 KB
save_599_model.h5	2020/3/5 23:36	H5 文件	13 KB
save_799_model.h5	2020/3/5 23:37	H5 文件	13 KB
save_999_model.h5	2020/3/5 23:39	H5 文件	13 KB

5. 记录和打印保存前和恢复后的 loss, 并查看是否一致。

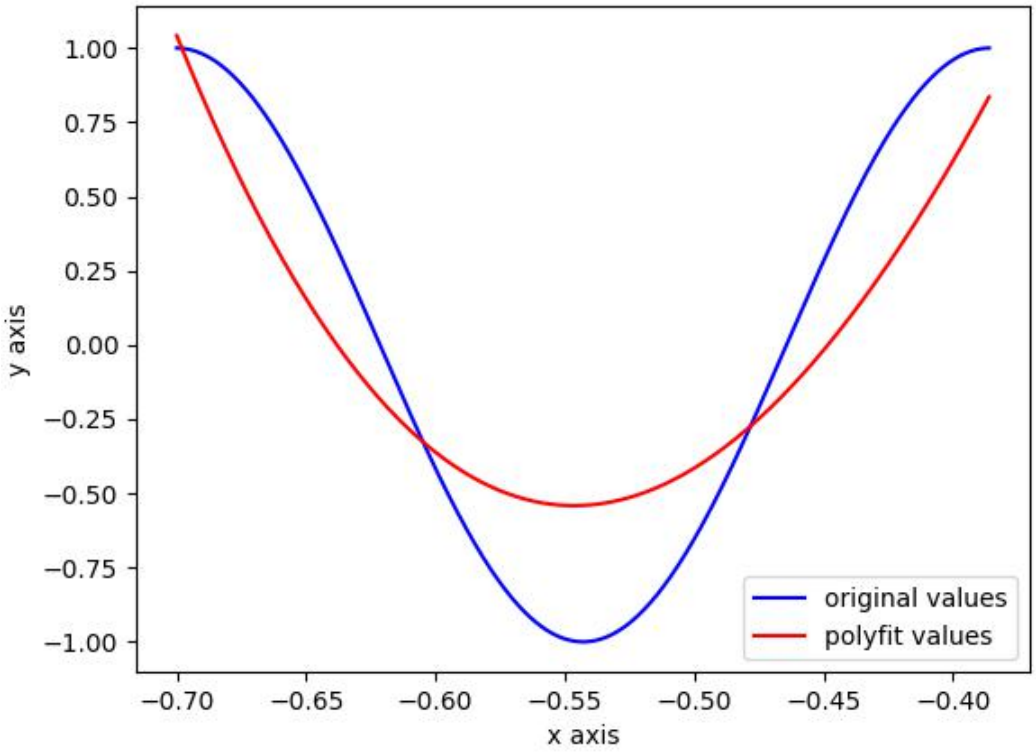
```
Epoch 999/1000
2000/2000 [==============================] - 0s 24us/sample - loss: 0.0956 - mse: 0.0956
Epoch 1000/1000
  64/2000 [..............................] - ETA: 0s - loss: 0.0892 - mse: 0.0892
Epoch 01000: saving model to ./checkpoints/weights.1000.h5  最后一个epoch保存结果
2000/2000 [==============================] - 0s 22us/sample - loss: 0.0950 - mse: 0.0950
2000/2000 [==============================] - 0s 144us/sample - loss: 0.0948 - mse: 0.0948
saved model, loss:  0.09
2000/2000 [==============================] - 0s 88us/sample - loss: 0.0948
Restored model, loss:  0.09                          重新加载以后loss依然是0.09
```

6. 绘制图像结果

最后一个 epoch 结果：



Restore 以后结果：



尝试并报告 loss_object = tf.keras.losses.MeanSquaredError()改为 tf.keras.losses.MeanSquaredError（去括号）的效果：

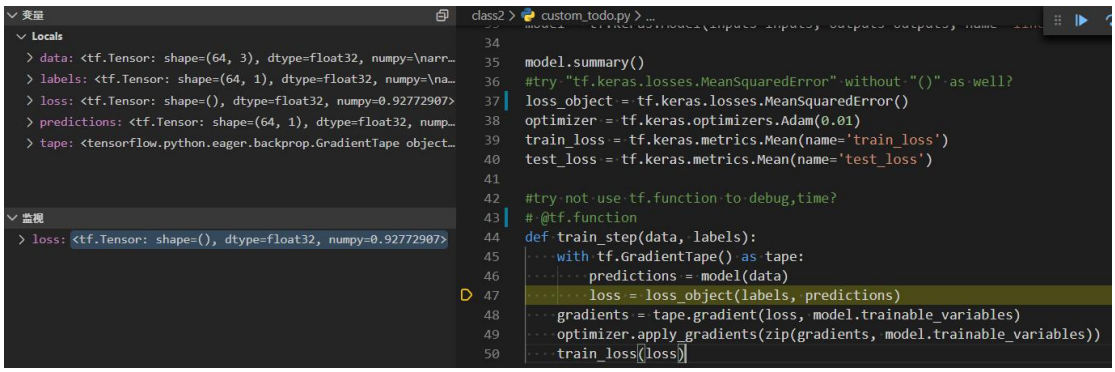
```
    if key not in cls.all():
  File "E:\ProgramData\Miniconda3\envs\tensorflow\lib\site-packages\tensorflow_core\python\ops\math_ops.py", line 1351, in tensor_equals
    return gen_math_ops.equal(self, other, incompatible_shape_error=False)
  File "E:\ProgramData\Miniconda3\envs\tensorflow\lib\site-packages\tensorflow_core\python\ops\gen_math_ops.py", line 3229, in equal
    name=name, ctx=_ctx)
  File "E:\ProgramData\Miniconda3\envs\tensorflow\lib\site-packages\tensorflow_core\python\ops\gen_math_ops.py", line 3258, in equal_eager_fallback
    _attr_T, _inputs_T = _execute.args_to_matching_eager([x, y], ctx)
  File "E:\ProgramData\Miniconda3\envs\tensorflow\lib\site-packages\tensorflow_core\python\eager\execute.py", line 267, in args_to_matching_eager
    ret = [ops.convert_to_tensor(t, dtype, ctx=ctx) for t in l]
  File "E:\ProgramData\Miniconda3\envs\tensorflow\lib\site-packages\tensorflow_core\python\eager\execute.py", line 267, in <listcomp>
    ret = [ops.convert_to_tensor(t, dtype, ctx=ctx) for t in l]
  File "E:\ProgramData\Miniconda3\envs\tensorflow\lib\site-packages\tensorflow_core\python\framework\ops.py", line 1314, in convert_to_tensor
    ret = conversion_func(value, dtype=dtype, name=name, as_ref=as_ref)
  File "E:\ProgramData\Miniconda3\envs\tensorflow\lib\site-packages\tensorflow_core\python\framework\constant_op.py", line 317, in _constant_tensor_conversion_fu
nction
    return constant(v, dtype=dtype, name=name)
  File "E:\ProgramData\Miniconda3\envs\tensorflow\lib\site-packages\tensorflow_core\python\framework\constant_op.py", line 258, in constant
    allow_broadcast=True)
  File "E:\ProgramData\Miniconda3\envs\tensorflow\lib\site-packages\tensorflow_core\python\framework\constant_op.py", line 266, in _constant_impl
    t = convert_to_eager_tensor(value, ctx, dtype)
  File "E:\ProgramData\Miniconda3\envs\tensorflow\lib\site-packages\tensorflow_core\python\framework\constant_op.py", line 96, in convert_to_eager_tensor
    return ops.EagerTensor(value, ctx.device_name, dtype)
TypeError: Cannot convert 'auto' to EagerTensor of dtype float
```

以上报错的原因，如果去掉括号，生成的 loss_object 就是一个类，而不是一个实例化的对象。

7. tf.function 功能

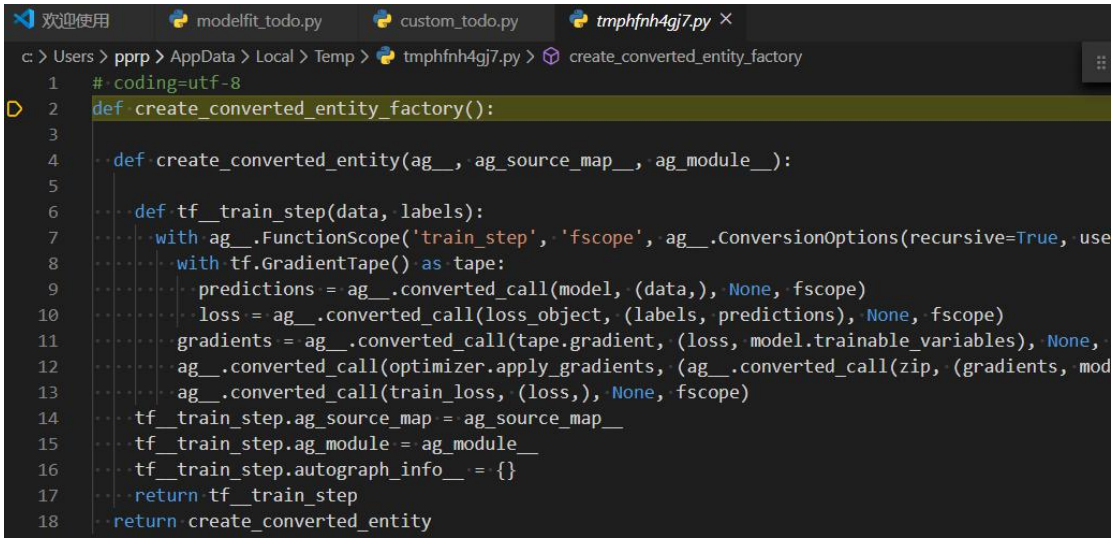
自定义循环添加@tf.function 和不添加的效果，可调试否？运行时间？

不添加效果如下：



可以进行 debug 并查看变量内容。

添加效果如下：



如果开启 tf.function，那么 debug 会进入一个新的临时文件，还发现只能进入这个临时文件一次，之后就无法进入那个临时文件。

速度上，开启 tf.function 速度要快很多，同样的 batch size=64:

```
Epoch 2237, Loss: 0.362,  Test Loss: 0.318, Time used: 0.05
Epoch 2238, Loss: 0.362,  Test Loss: 0.318, Time used: 0.06
Epoch 2239, Loss: 0.362,  Test Loss: 0.317, Time used: 0.05
Epoch 2240, Loss: 0.362,  Test Loss: 0.317, Time used: 0.07
Epoch 2241, Loss: 0.362,  Test Loss: 0.317, Time used: 0.07
Epoch 2242, Loss: 0.362,  Test Loss: 0.317, Time used: 0.06
Epoch 2243, Loss: 0.362,  Test Loss: 0.317, Time used: 0.08
Epoch 2244, Loss: 0.362,  Test Loss: 0.317, Time used: 0.10
Epoch 2245, Loss: 0.362,  Test Loss: 0.317, Time used: 0.11
Epoch 2246, Loss: 0.362,  Test Loss: 0.317, Time used: 0.10
Epoch 2247, Loss: 0.362,  Test Loss: 0.317, Time used: 0.13
Epoch 2248, Loss: 0.362,  Test Loss: 0.317, Time used: 0.10
Epoch 2249, Loss: 0.362,  Test Loss: 0.317, Time used: 0.07
Epoch 2250, Loss: 0.361,  Test Loss: 0.317, Time used: 0.06
Epoch 2251, Loss: 0.361,  Test Loss: 0.317, Time used: 0.05
Epoch 2252, Loss: 0.361,  Test Loss: 0.317, Time used: 0.11
Epoch 2253, Loss: 0.361,  Test Loss: 0.316, Time used: 0.11
```

图 4 开启 tf.function

```
Epoch 242, Loss: 0.558,  Test Loss: 0.501, Time used: 0.28
Epoch 243, Loss: 0.558,  Test Loss: 0.501, Time used: 0.30
Epoch 244, Loss: 0.558,  Test Loss: 0.501, Time used: 0.24
Epoch 245, Loss: 0.557,  Test Loss: 0.501, Time used: 0.25
Epoch 246, Loss: 0.557,  Test Loss: 0.500, Time used: 0.28
Epoch 247, Loss: 0.557,  Test Loss: 0.500, Time used: 0.28
Epoch 248, Loss: 0.557,  Test Loss: 0.500, Time used: 0.26
Epoch 249, Loss: 0.557,  Test Loss: 0.500, Time used: 0.25
Epoch 250, Loss: 0.557,  Test Loss: 0.500, Time used: 0.29
Epoch 251, Loss: 0.557,  Test Loss: 0.500, Time used: 0.26
Epoch 252, Loss: 0.557,  Test Loss: 0.500, Time used: 0.27
Epoch 253, Loss: 0.557,  Test Loss: 0.500, Time used: 0.24
Epoch 254, Loss: 0.557,  Test Loss: 0.500, Time used: 0.31
Epoch 255, Loss: 0.556,  Test Loss: 0.500, Time used: 0.27
Epoch 256, Loss: 0.556,  Test Loss: 0.499, Time used: 0.26
Epoch 257, Loss: 0.556,  Test Loss: 0.499, Time used: 0.25
Epoch 258, Loss: 0.556,  Test Loss: 0.499, Time used: 0.35
```

图 5 不开启 tf.function

一开始不理解，既然有了动态图，为何还要使用静态图。现在看来，应该是现在关闭 tf.function 的时候进行 debug，利用动态图灵活的特点进行调试。然后在稳定以后，可以开启静态图，加速训练过程。

8.发现

相同的 optimizer 和 loss 的情况下，使用两种方法相同迭代次数结果不一样。Fit 方法能够更快的收敛，使用 fit 方法只需要用 1000epoch 就能达到与自定义循环 10000epoch 差不多的结果。