1. Tensorflow2构建网络以及训练，测试流程
2. 使用 tf,keras,datasets 获取数据集并预处理
3. 使用 tf.keras,Model 和 tf.keras.layers 搭建模型
4. 训练模型，使用tf.keras.losses计算损失函数，使用tf.keras.optimizer优化模型
5. 测试模型，使用tf.keras.metrics 计算评估指标
6. 获取数据集以及预处理数据
7. 获取数据集

可以直接在代码中下载，也可以提前下载好数据集mnist.npz文件，放入用户目录下的 .keras/dataset目录下。（Windows目录为 C:\\Users\\用户名），Linux目录为 /home/用户名

mnist = tf.keras.datasets.mnist

1. 查看训练数据和测试数据

获取到数据集后，我们可以查看一下数据集的训练数据和测试数据，通过load\_data()方法返回两个元组，每个元组包括训练集合测试集的数据和标签。

(train\_data, train\_label), (test\_data, test\_label) = mnist.load\_data()

1. 数据预处理

根据项目的实际情况进行预处理。此处的mnist数据集中读出的数据缺少一个颜色通道数维度，因此需要扩大一个维度，并且将灰度值归一化到0-1之间。

train\_data = np.expand\_dims(train\_data.astype(np.float32) / 255., axis=-1)

这里我们将数据集的获取与数据预处理封装到一个类中，方便代码的复用。

***#数据获取与预处理类***class MNISTLoader():  
 def \_\_init\_\_(self):  
 mnist = tf.keras.datasets.mnist ***#获取已经下载好的数据集  
 #获取训练数据，标签以及测试数据，标签*** (self.train\_data, self.train\_label), (self.test\_data, self.test\_label) = mnist.load\_data()  
 ***#数据是三维的，少一位通道数，我们需手动添加一维  
 #图像数据为0-255，需要归一化到0-1之间*** self.train\_data = np.expand\_dims(self.train\_data.astype(np.float32) / 255., axis=-1)  
 self.test\_data = np.expand\_dims(self.test\_data.astype(np.float32) / 255., axis=-1)  
 self.train\_label = self.train\_label.astype(np.int32)  
 self.test\_label = self.test\_label.astype(np.int32)  
 ***#训练数据数量，测试数据数量*** self.num\_train\_data, self.num\_test\_data = self.train\_data.shape[0], self.test\_data.shape[0]  
  
 def get\_batch(self, batch\_size):  
 ***#随机取batch\_size数据以及标签  
 #随机取索引*** index = np.random.randint(0, self.num\_train\_data, batch\_size)  
 return self.train\_data[index, :], self.train\_label[index]

1. 搭建模型

同样是将模型的搭建封装在一个类中，继承tf.keras.Model类。重写 \_\_init\_\_()函数，该函数中定义模型的各个层；重写call()函数，该函数定义了各个层间的操作。

第一个层为Flatten层，将拉平数据的后三个维度称为一体

第二个层为全连接层，神经元为100个，激活函数为relu函数

第三个层为全连接层，神经元为10个，即分类树，没有激活函数

注意，最后的全连接层输出的结果通过softmax函数求出概率。

class MyModel(tf.keras.Model):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 ***#Flatten层将除第一维外的维度展平*** self.flatten = tf.keras.layers.Flatten()  
 self.dense1 = tf.keras.layers.Dense(units=100, activation=tf.nn.relu)  
 self.dense2 = tf.keras.layers.Dense(units=10)  
  
 def call(self, inputs): ***#[batch\_size, 28, 28, 1]*** x = self.flatten(inputs) ***#[batch\_size, 784]*** x = self.dense1(x) ***#[batch\_size, 100]*** x = self.dense2(x) ***#[batch\_size, 10]*** output = tf.nn.softmax(x) ***#归一化求概率*** return output  
  
***#循环次数***

1. 训练模型

选好batch，学习率，循环次数。将前面定义好的模型类，数据加载类实例化，同时定义一个模型的优化器tf.keras.optimizers.Adam。

训练步骤

1. 随机抽取一个batch的训练数据
2. 计算模型的预测值
3. 计算loss，计算loss对variable的导数
4. 将导数和参数传入优化器，更新模型的参数以最小化loss

***#循环次数***num\_epoch = 5  
***#一个batch的数据量***batch\_size = 50  
***#学习率***learning\_rate = 1e-3  
  
***#模型实例***model = MyModel()  
***#数据加载器实例***data\_loader = MNISTLoader()  
***#优化器***optimizer = tf.keras.optimizers.Adam(learning\_rate)  
  
***#训练接中batch的数量***num\_batch = int(data\_loader.num\_train\_data / batch\_size)  
  
***#训练过程***for i in range(num\_epoch):  
 for batch\_index in range(num\_batch):  
 ***#导入一个batch的训练数据和标签*** x, y = data\_loader.get\_batch(batch\_size)  
 with tf.GradientTape() as tape:  
 y\_pred = model(x) ***#预测值  
 #计算损失值*** loss = tf.keras.losses.sparse\_categorical\_crossentropy(y\_true=y, y\_pred=y\_pred)  
 loss = tf.reduce\_mean(loss)  
 print("batch %d: loss %f" %(batch\_index, loss.numpy()))  
 ***#计算梯度*** grads = tape.gradient(loss, model.variables)  
 optimizer.apply\_gradients(zip(grads, model.variables))

1. 测试模型

使用tf.keras.metrics评估模型，使用对应的评估器在测试集上测试性能，将真实结果与预测结果比较，输出预测正确的样本占总样本数的比例。

通过传入参数y\_pred预测值, y\_true真实值，update\_state()更新评估器的状态，迭代结束后，通过result(()方法输出准确率。

***#模型的测试  
#实例化对应损失函数的评估器***tester = tf.keras.metrics.SparseCategoricalAccuracy()  
  
***#测试集的batch数量***num\_batch\_test = int(data\_loader.num\_test\_data / batch\_size)  
for batch\_index in range(num\_batch\_test):  
 ***#测试数据的标签*** start\_index, end\_index = batch\_index \* batch\_size, (batch\_index+1) \* batch\_size  
 ***#预测值*** y\_pred = model.predict(data\_loader.test\_data[start\_index : end\_index])  
 ***#更新评估器的状态*** tester.update\_state(y\_true=data\_loader.test\_data[start\_index:end\_index], y\_pred=y\_pred)  
 print("test accuracy : %f" % tester.result())