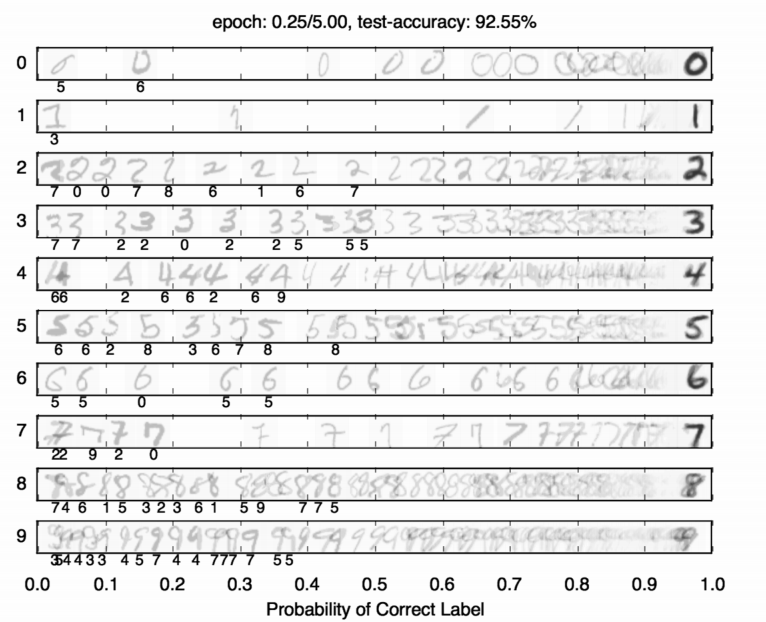
project 5 machine learning

2250575 张育晨

1. 问题概述：

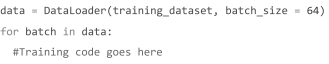
在这个项目中，我将建立神经网络，借此完成任务，如对手写体数字的分类识别。



已有代码的理解：

所需的函数都在models.py中import，主要包括一些pytorch库中的函数，类等功能，具体如下。

1. tensor()：是主要使用的数据结构，与numpy中的array相似。
2. relu(input)：激活函数，返回max(input, 0)。
3. linear：一个类，用于生成一个线性层。用途是将input和链接权重weight两个向量取点积。具体使用方式是self.layer=Linear(本层输入向量长度, 本层输出向量长度)，此时pytorch自动生成随机连接参数并在后面训练更新他们。
4. movedim(input\_vector, initial\_dimention\_position, final\_dimention\_position)：输入一个矩阵，将初始的维度位置与结果的维度位置交换。在q3中有作用。
5. cross\_entropy(prediction, target)：分类预测的损失函数，预测距离训练集的标准输出越远，损失函数返回的损失值越大，用于q3~q5。
6. mse\_loss(prediction, target)：回归预测的损失函数，用于q2。
7. dataset：pytorch的数据集类，存储项目提供的数据。可以通过以下方式转化为dataloader的形式来便于批处理。



每小题的概述

q1：实现一个二元预测。输出的标签为1或-1。数据集中的点(x, y)，取全部y作为一个torch.tensor，然后可知这个tensor中只有1或-1。因为在一个数据集（监督学习的训练集）的点中，y是标准的输出，而二分类的标准输出在此处只有1或-1。实现的方式是构建单层感知机模型。

q2：给出x预测sin(x)。

q3：手写体数字识别。

q4：语言识别。

q5：2。

1. 算法设计
2. Perception感知机
   1. 算法功能：完成init(self, dimensions)函数。具体来说是要初始化模型链接权重的参数，并且以parameter()对象的形式保存，以便pytorch识别其为连接参数。

设计思路：利用ones函数，更新self.w。self.w = Parameter(ones(1, dimensions))。随机生成1\*dimentions的链接权重。这里dimentions指的是数据集每个点的（输入向量x）维度。

* 1. 算法功能：实现run(self, x)函数的功能。实现模型输入的变量x（一个tensor）和刚刚的链接权重self.w之间的点积。实现神经网络每个节点中的加法器的功能。

设计思路：直接使用tensordot函数实现点积功能。

* 1. 算法功能：实现get\_prediction(self, x)函数的功能，当上面加法器的点积结果为非负数的时候，返回1；否则返回-1。总体功能类似于一个激活函数。

设计思路：直接使用简单的if else语句即可。

* 1. 算法功能：完成train(self)函数，实现循环训练更新参数的功能。

设计思路：

通过现在已经建立的，输入维度等于输入向量x，输出维度为1的单层神经网络，将数据集里的所有x输入。对每一个向量x而言，得到预测结果y’。如果预测结果y’与标准输出不同，则更新参数。然后判断训练集中的下一个输入x，直到判断完本训练集。如果在对整个集合的判断过程中，有一个预测错误，就再循环对训练集判断一次，直到预测成功率为100%。

1. 非线性回归预测

算法功能：构建一个带有隐藏层的神经网络，来实现非线性的回归预测。目标是训练这个模型，使能够准确预测输入x的输出sin(x)（在实际操作中不用管这一部分的数学意义，因为标准输入输出已经在dataset中给出，只需要利用训练即可）。整体来说输入为1维，输出也为1维，隐藏层维数待定。

设计思路：

首先设计隐藏层，设计只有一个隐藏层，然后隐藏层的节点为100个。利用 self.hidden\_layer = Linear(1, self.hidden\_layer\_node\_num)来实现。然后设计输出层，即从100维回到1维，利用self.output\_layer = Linear(self.hidden\_layer\_node\_num, 1)来实现。

之后设计向前传播函数，即用来从输入预测输出的函数。此处采用output\_layer( relu( hidden\_layer(x) ) )的方式来向前传播。

1. 数字分类识别

算法功能： 建立分类预测神经网络模型， 识别MNIST数据集中的手写体数字。每个数字为28x28像素，存储于784维浮点数向量之中。输出为10维向量，每个维度代表本手写体是否被分类到一个数字的类之中，组成一个独热码。

设计思路：

设计包含两个隐藏层，一个输出层的神经网络。两个隐藏层从下到上节点数为200和100。在训练的过程中使用SGD优化器。然后采用给出的dataset. get\_validation\_accuracy()函数来判断每一次通过整个训练集训练一次后当前模型的效果，按照题目的通过要求，当validation\_accuracy>0.975的时候，停止训练。

对于一个batch块的每一次训练，采取以下的步骤：①从每个batch块中拿取数据。②清空优化器梯度，对这个块训练一遍，并且计算得到损失。③反向传播计算梯度，更新参数。

另外为增加学习效率，加快模型收敛速度，采用动态调节学习率的方式，采用scheduler = optim.lr\_scheduler.StepLR(optimizer, step\_size=5, gamma=0.1)，在每次数据集学习完毕后，动态调整一下学习率。

1. 语言识别

算法功能： 给一段文字的碎片，识别这种文字属于哪种语言。因为不同的单词含有不同数量的字母，所以构建的神经网络需要可以接受不同长度的输入。

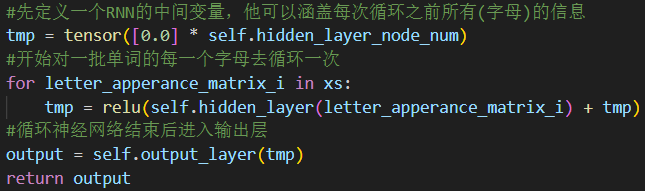
设计思路：

采用RNN的设计思路，总体来说思路是一开始先只将第一个字母输入，得到一个输出，然后将这个输出和第二个字母组合在一起，作为新的输入。这样每次的输入就可以包含本个字母以及他之前的字母的所有信息，同时单词的长度仅决定在RNN中循环的次数，而不会对其结构产生任何影响。

下面具体的说明我是如何设计的这个rnn网络。

首先在网络结构方面，我采用一层隐藏层，也就是只有一层循环的层，这层包括128个节点。然后输出层自然是一个从隐藏层节点数到语言种类数5的层。

然后是**最重要的向前传播函数的设计**。**首先明确这个函数的输入输出**。举一个例子，对一个batch\_size为8的批处理块，可以保证块中每个单词都只含3个字母。在这种情况下**输入**可以理解为包含三个矩阵，每个矩阵都是8\*47（47表示的是五种语言一共合在一起有47个互不相同的字母）。这里以第一个矩阵为例，第一个矩阵第一行是一个独热码，代表了这个批处理块的第一个单词的第一个字母是什么。对于这个函数的**输出**，是一个batch\_size\*5的矩阵，矩阵每一行代表了输入块中一个单词对应为五种语言的可能性得分。最后讨论**RNN网络向前传播的原理**。对这一批单词中的每个字母，按照在单词中出现从前往后的顺序，循环输入到前面网络结构设计的时候设计的那一个隐藏层之中去，得到的隐藏层输出记录为临时变量tmp（神经网络最开始刚刚建立，没有数据输入的时候tmp为维度等于隐藏层节点数，每个维度都为0的向量）。但在每次输入到隐藏层的时候，不能只考虑本个字母，还要考虑之前的所有字母和他们的顺序产生的影响，这就需要把上一次隐藏层的输出tmp（包含之前输入的字母和他们的顺序信息）也在本次的传播过程中考虑到，最终给出的传播公式为tmp = relu(self.hidden\_layer (letter\_apperance\_matrix\_i) + tmp)。



接下来考虑损失函数，只需要按照之前的思路，设计为cross\_entropy()函数即可。

最后考虑训练的过程，整体如之前，按照：“①从每个batch块中拿取数据。②清空优化器梯度，对这个块训练一遍，并且计算得到损失。③反向传播计算梯度，更新参数。” 的这步骤，来训练即可。

此处值得特别注意的是，从batch块中拿取一个数据的时候，需要调整拿出的vector中的顺序。模型的 run 方法期望输入张量的形状为 (length\_of\_word, batch\_size, num\_chars)，因为在 RNN 中按字符处理数据，而每个时间步都需要处理整个批次的单词。因此需要将单词长度(length\_of\_word)维度移动到第一个位置，以便 RNN 按字符序列处理输入。在实际操作中，利用movedim(batch\_xs, 0, 1) 函数将单词长度维度移动到第一个位置，以适应 RNN 的输入要求。这样处理后的张量可以正确地传递给模型的 run 方法进行处理。

最后控制其训练20次后即可得到较好的模型参数，结束训练。

1. 应用卷积层CNN

CNN：

卷积层在对多维度输入训练的时候，可以更好的考虑空间信息。其在应对二维数据的时候，不再需要将其每行首尾相接，转化为一维向量。要实现这样的功能，需要的权重参数也是一个矩阵。

算法功能：利用卷积神经网络来进行手写体图片分类工作。

设计思路：

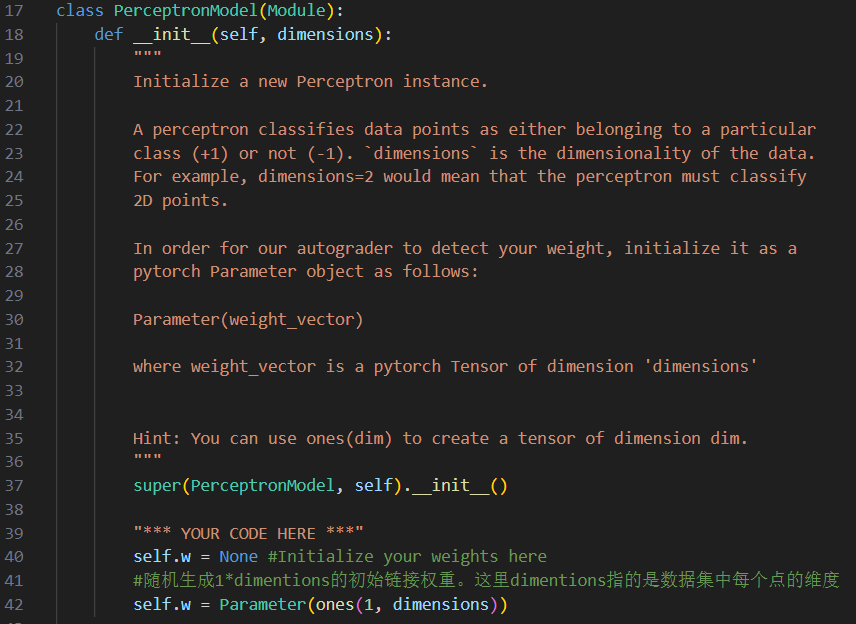
首先**设计单独的卷积层**，这里的卷积层不能像之前一样通过调用现有的Linear来实现，需要自行设计编写。卷积层的基本思路是小矩阵（需要训练的权重矩阵）在大矩阵（输入矩阵）上从左到右，从上到下滑动，在每一个滑动的位置，对两个矩阵重合部分的重合矩阵，每个对应位置相乘得到新矩阵，然后把新矩阵各个位置相加，得到的加和为本次滑动的对应输出矩阵中的元素。

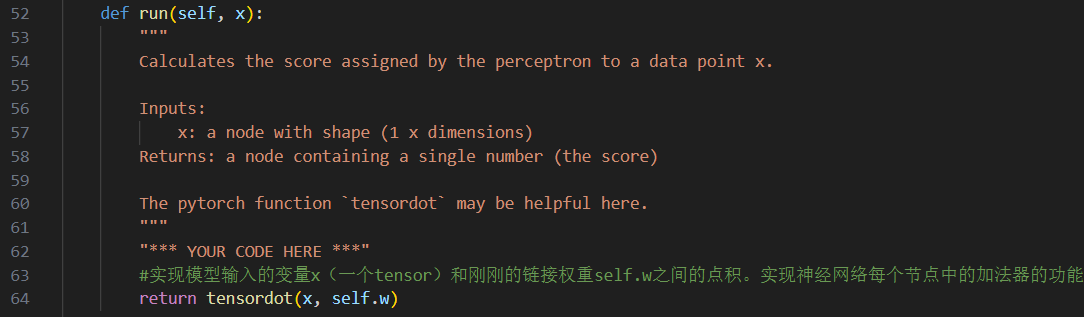
在向前传播的函数run的设计之中，由于他已经为我写好了卷积层的应用方式，输出的是一个一维的向量。故而只需要完全仿照q3的手写体识别的方式，设计两层网络，然后向前传播。

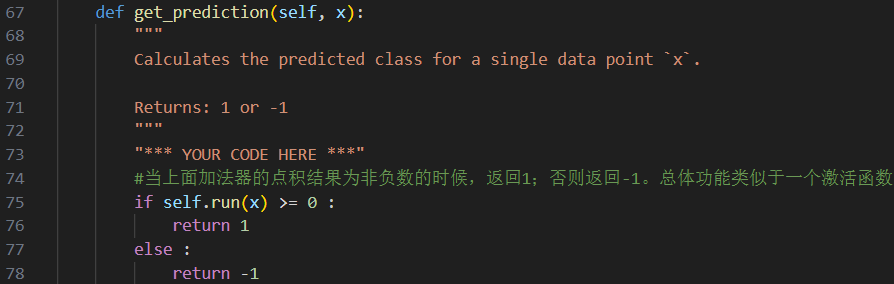
在训练的函数中，也完全仿照q3手写体识别的训练方式，批量训练，然后训练集过一遍之后检测一下此时模型的训练结果，达标就退出训练。另外也可以仿照q3设计一个学习率调整策略的函数。

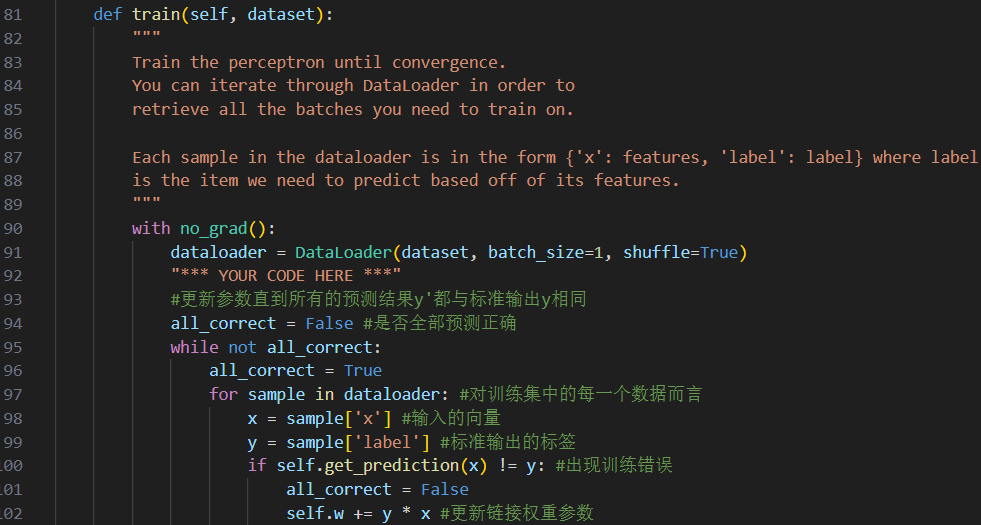
1. 算法实现

q1 Perception感知机

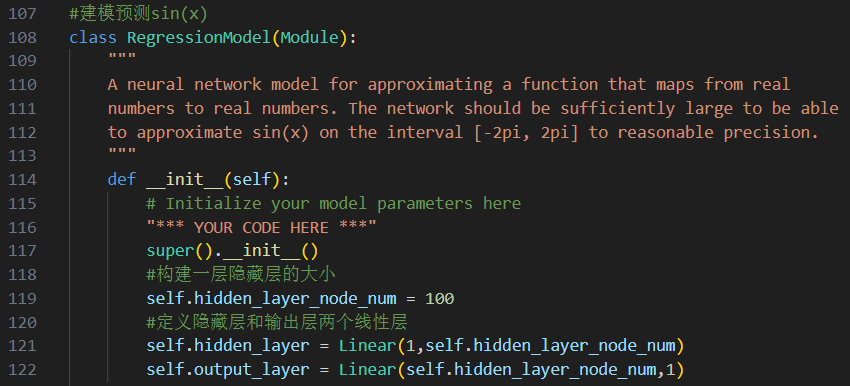


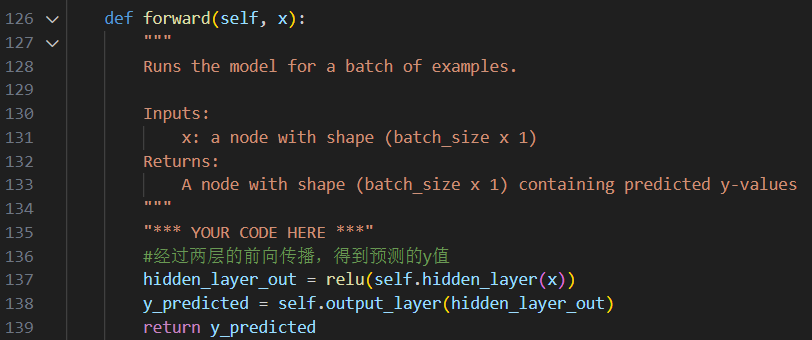


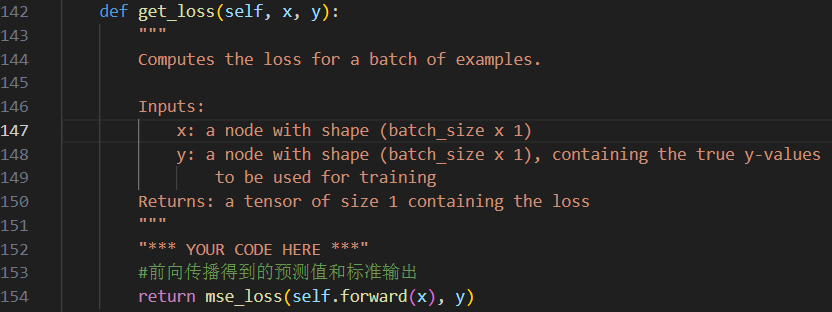


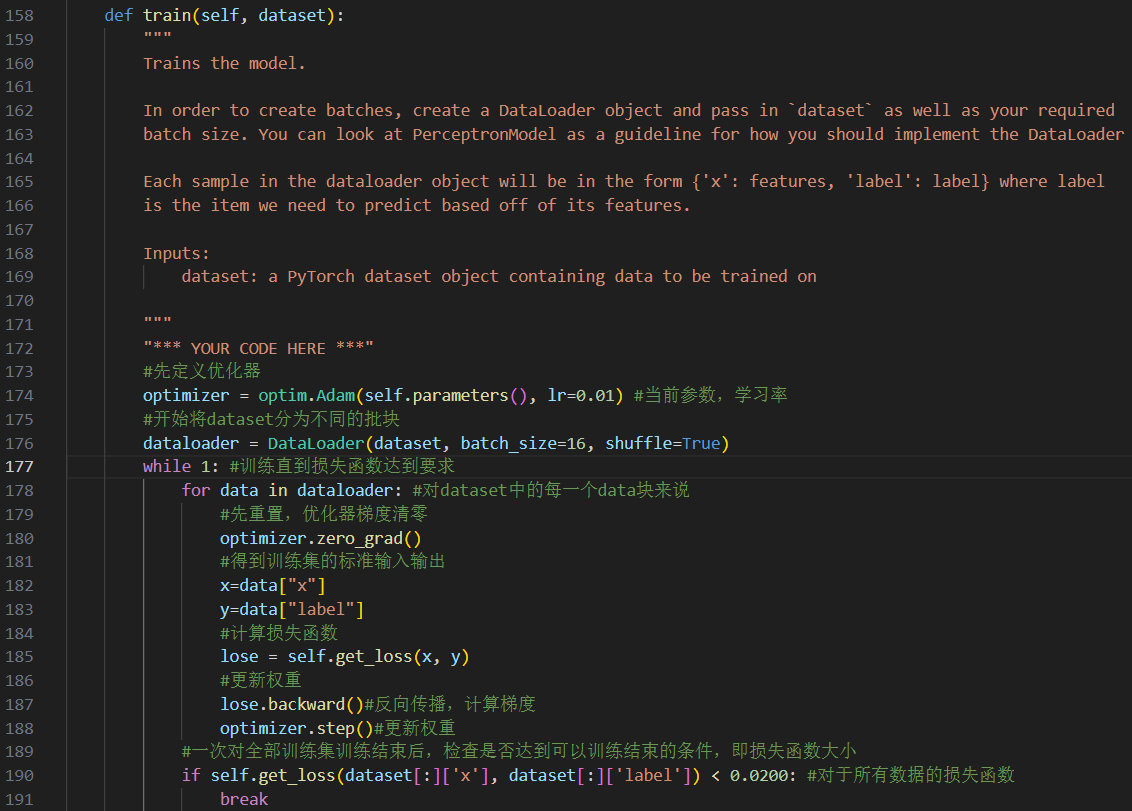


q2 非线性回归预测sin(x)

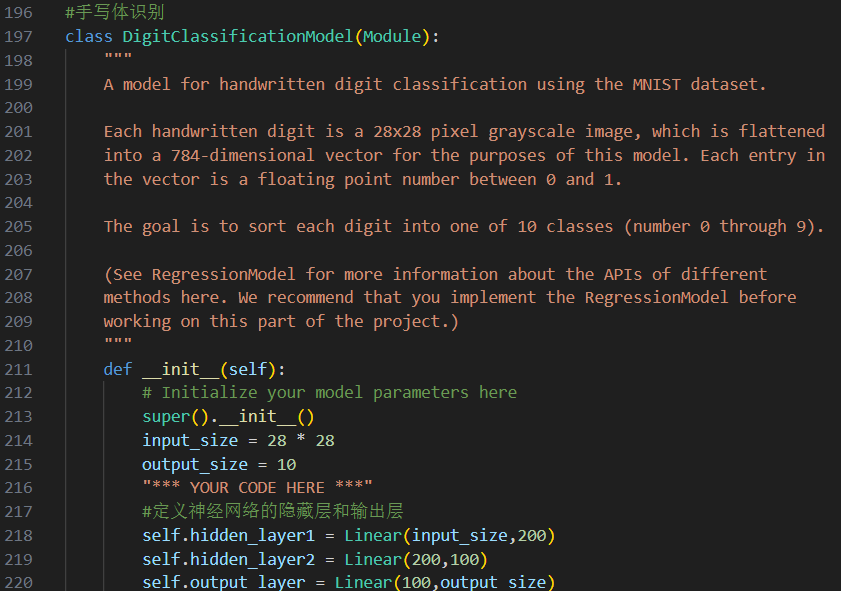


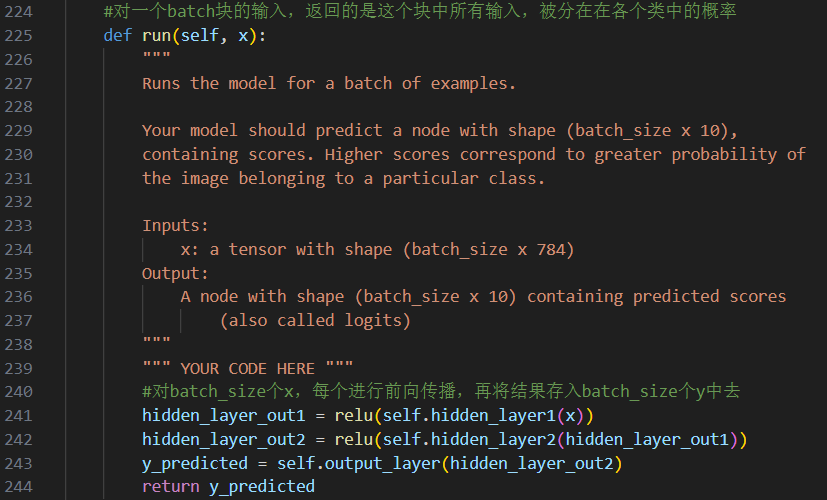


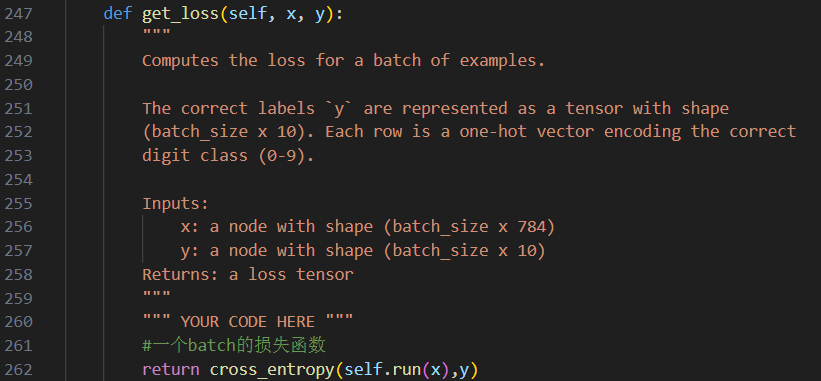


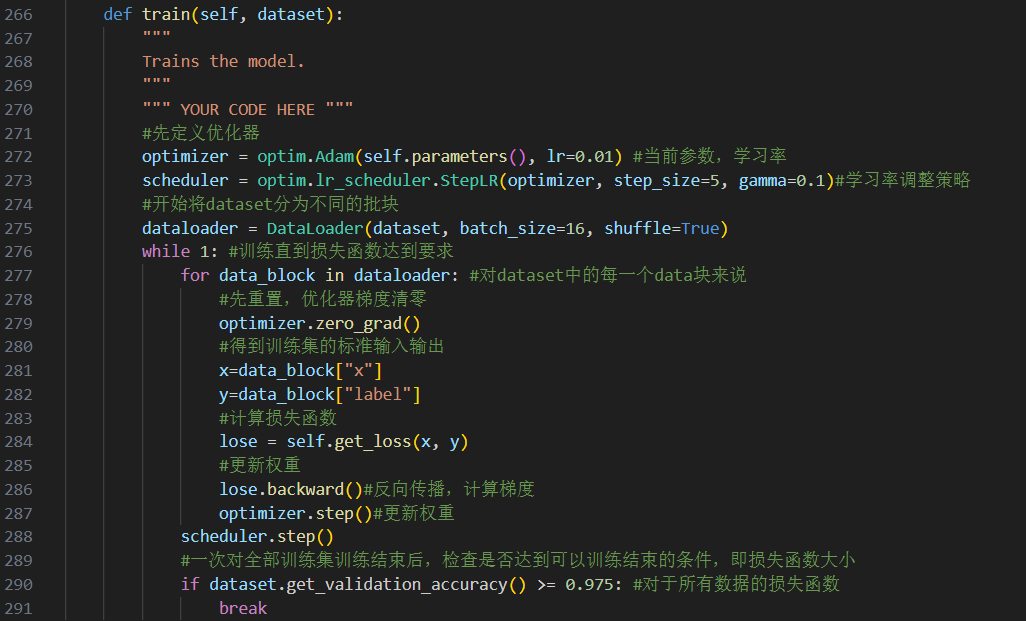


q3 手写体识别网络

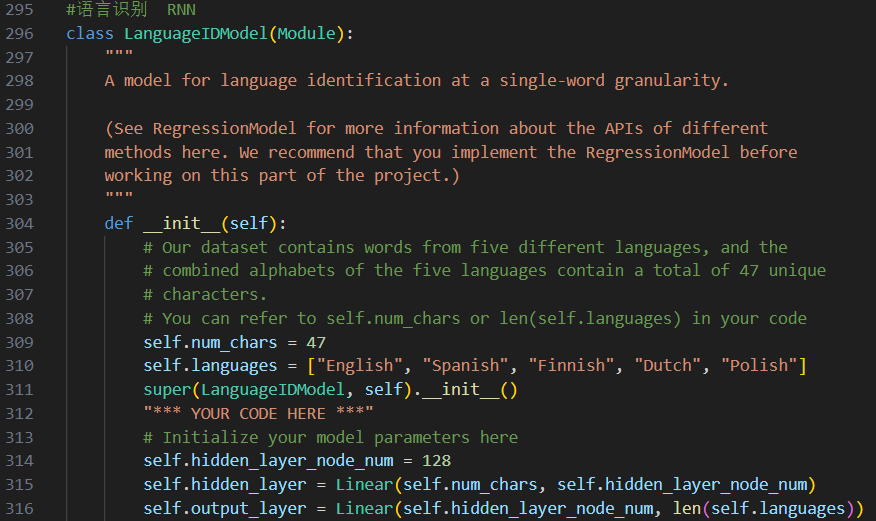


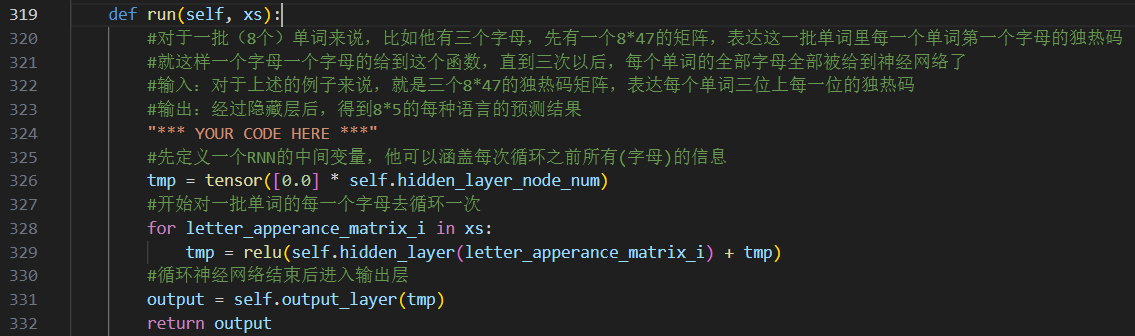


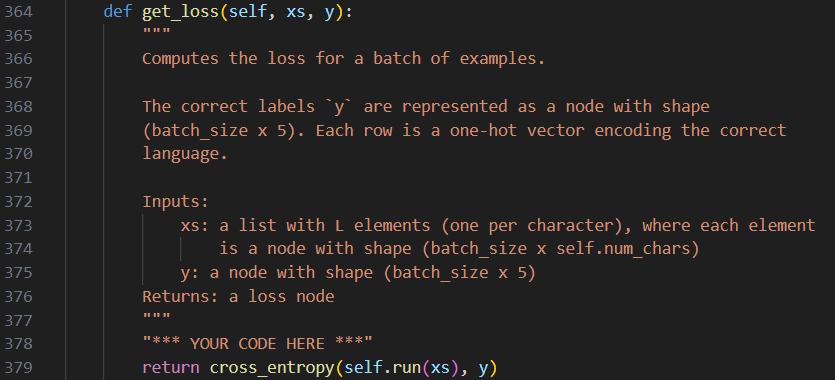


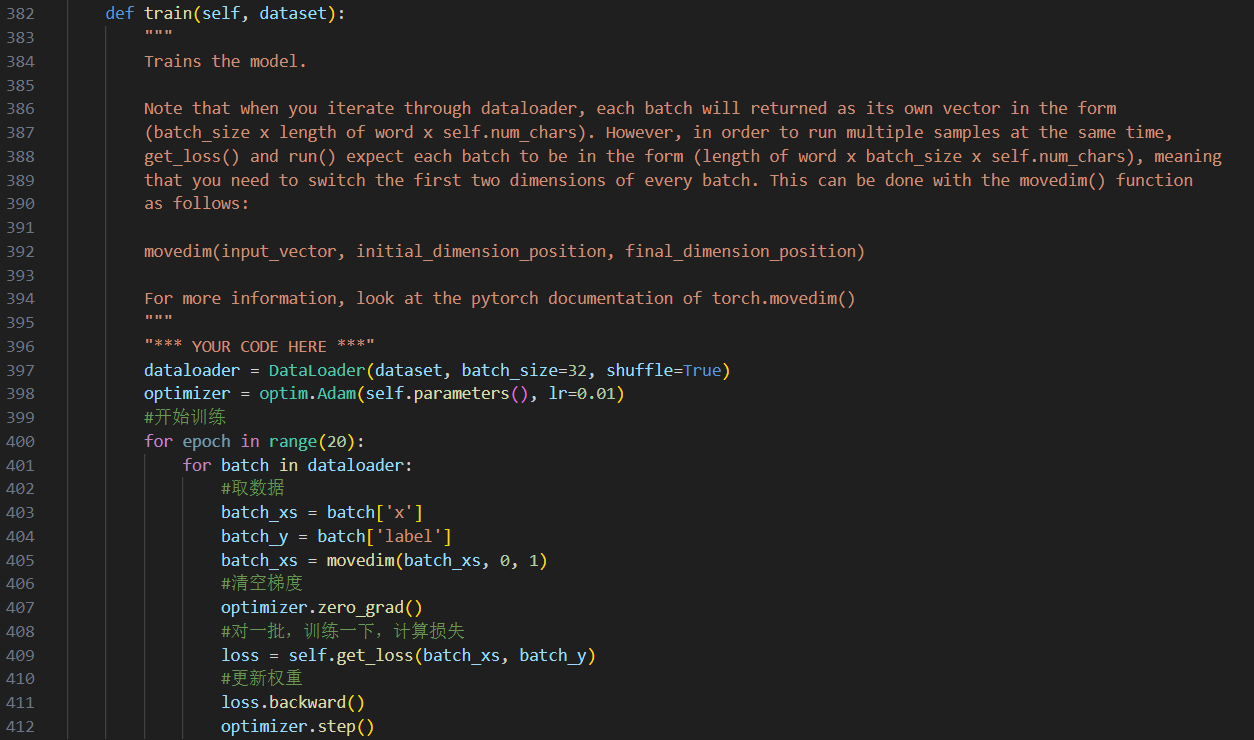


q4 语言识别

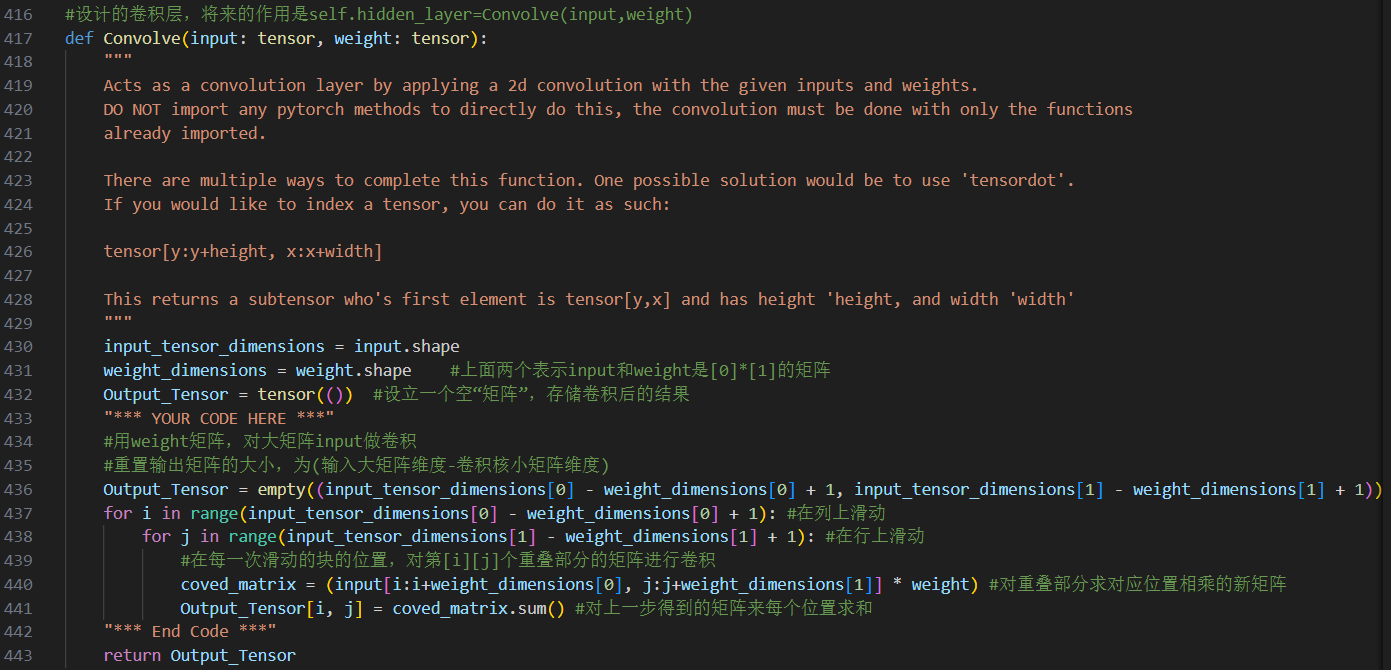


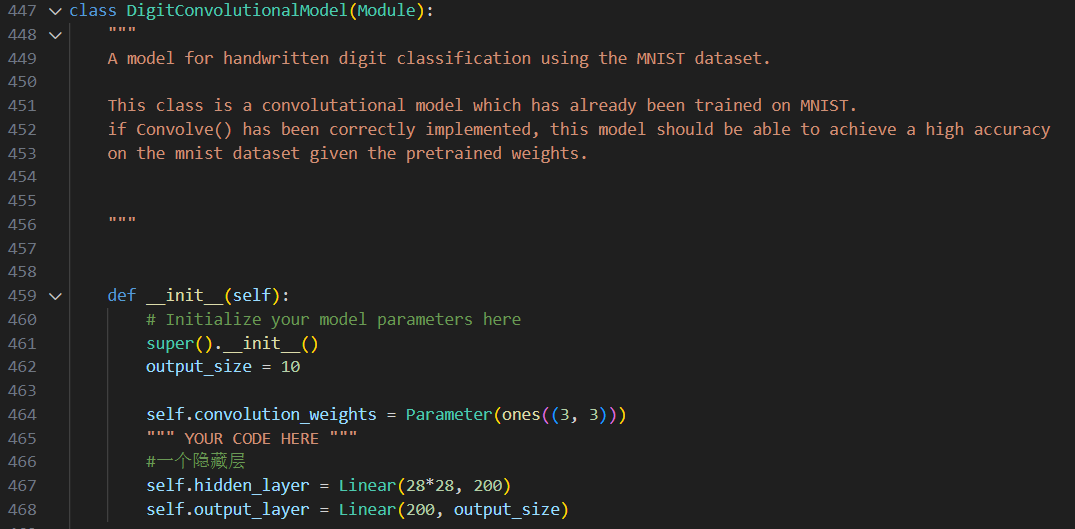


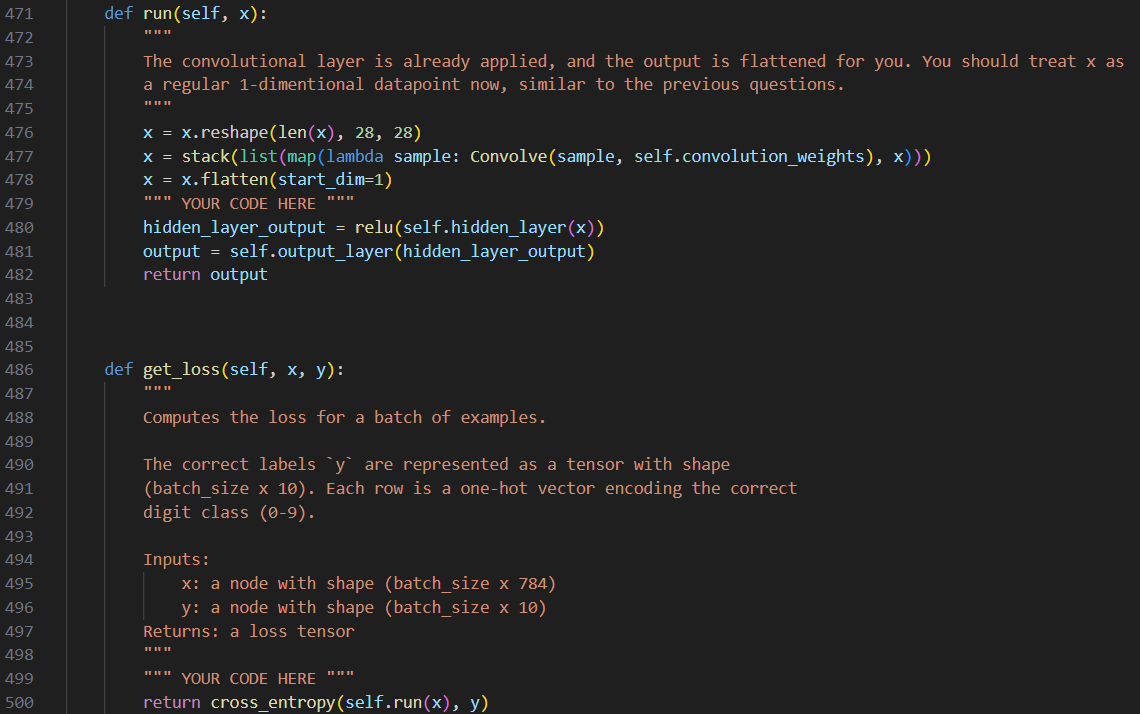


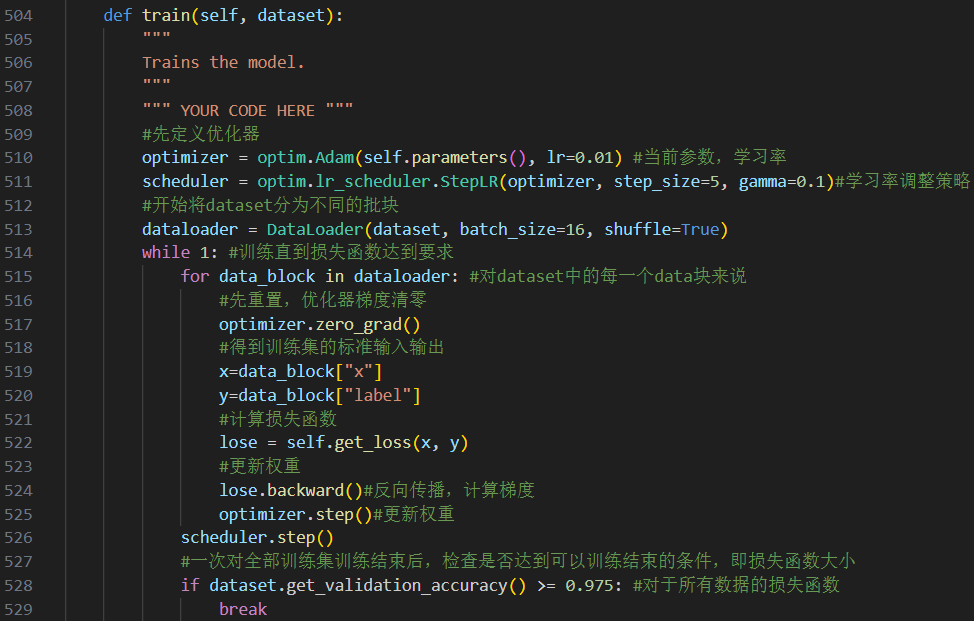


q5



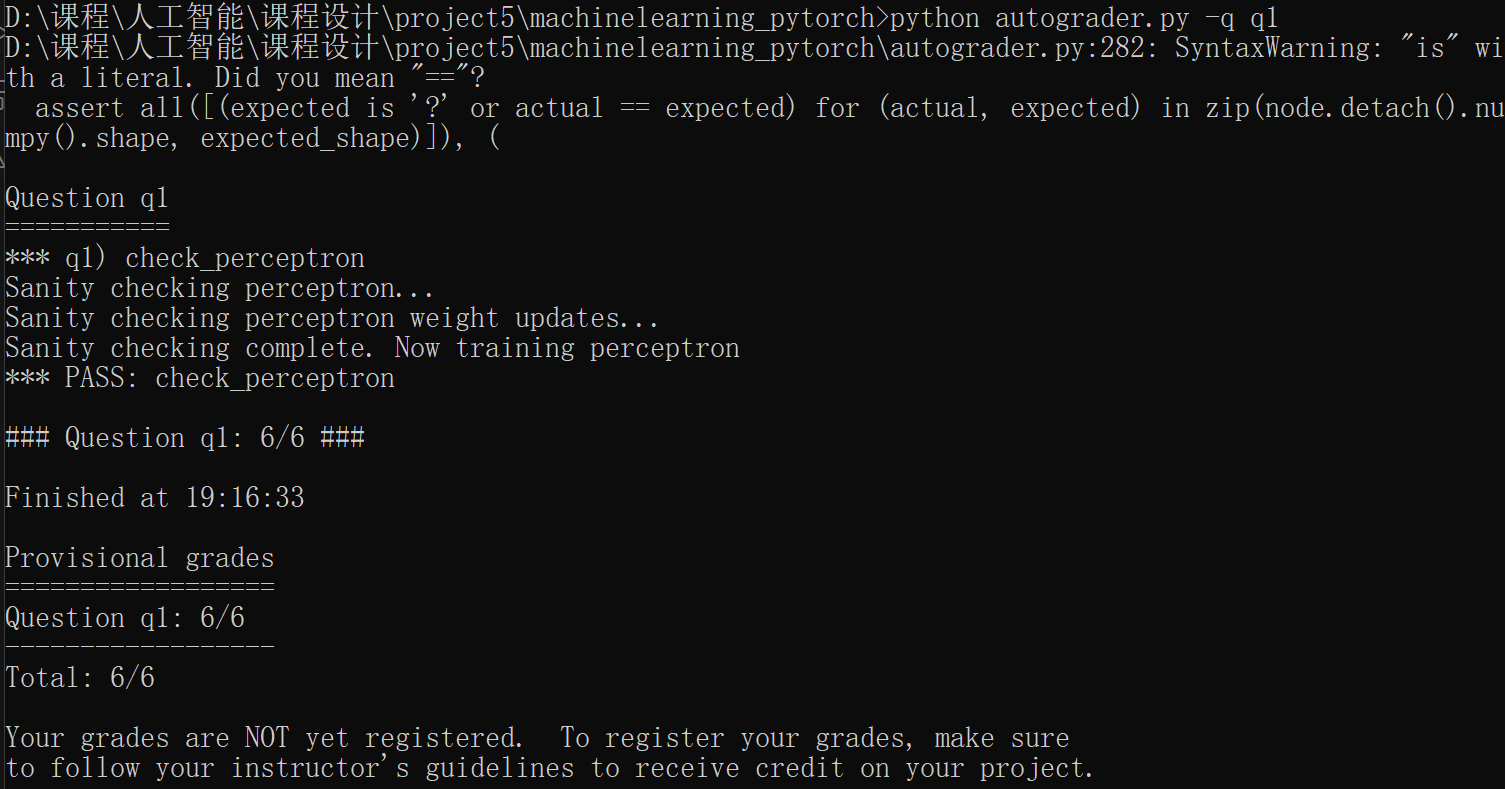




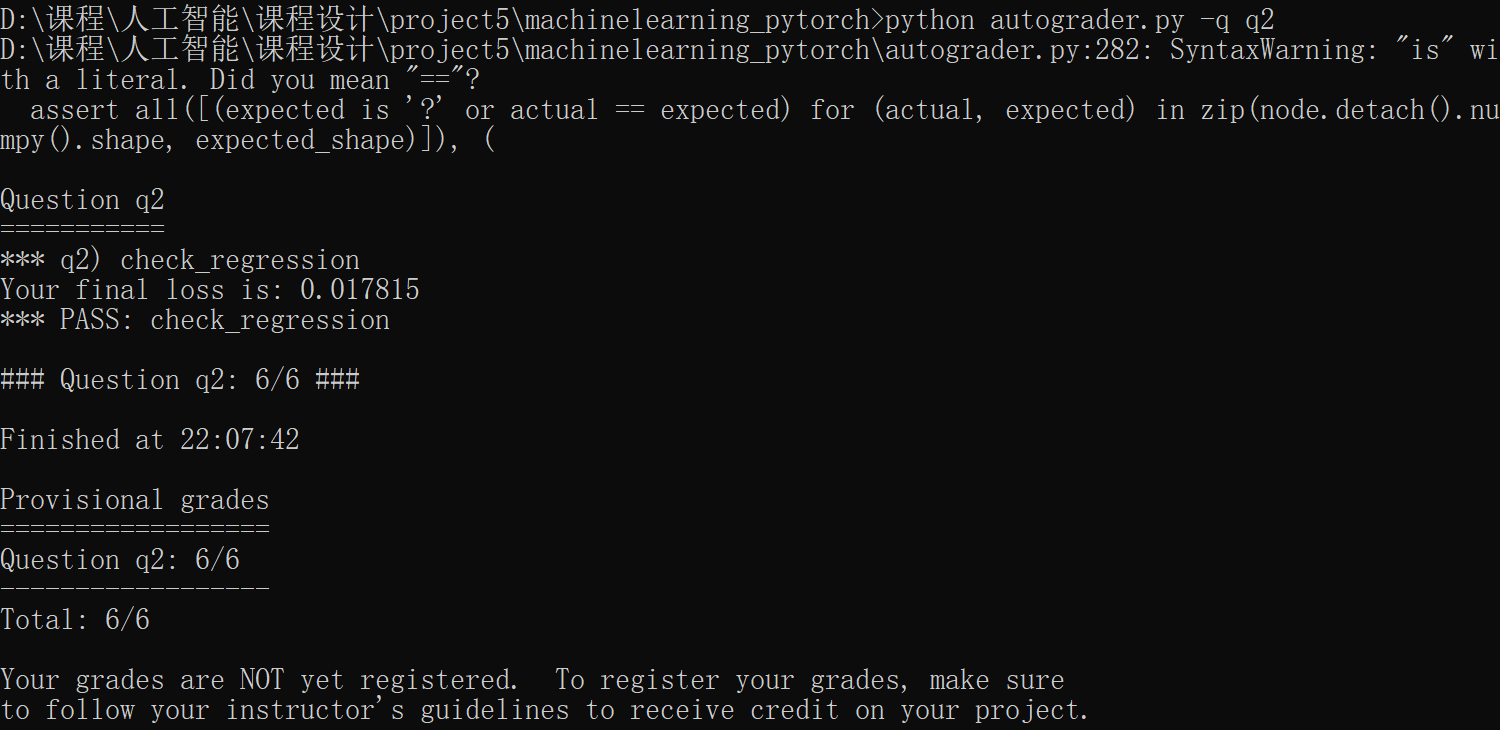


1. 实验结果

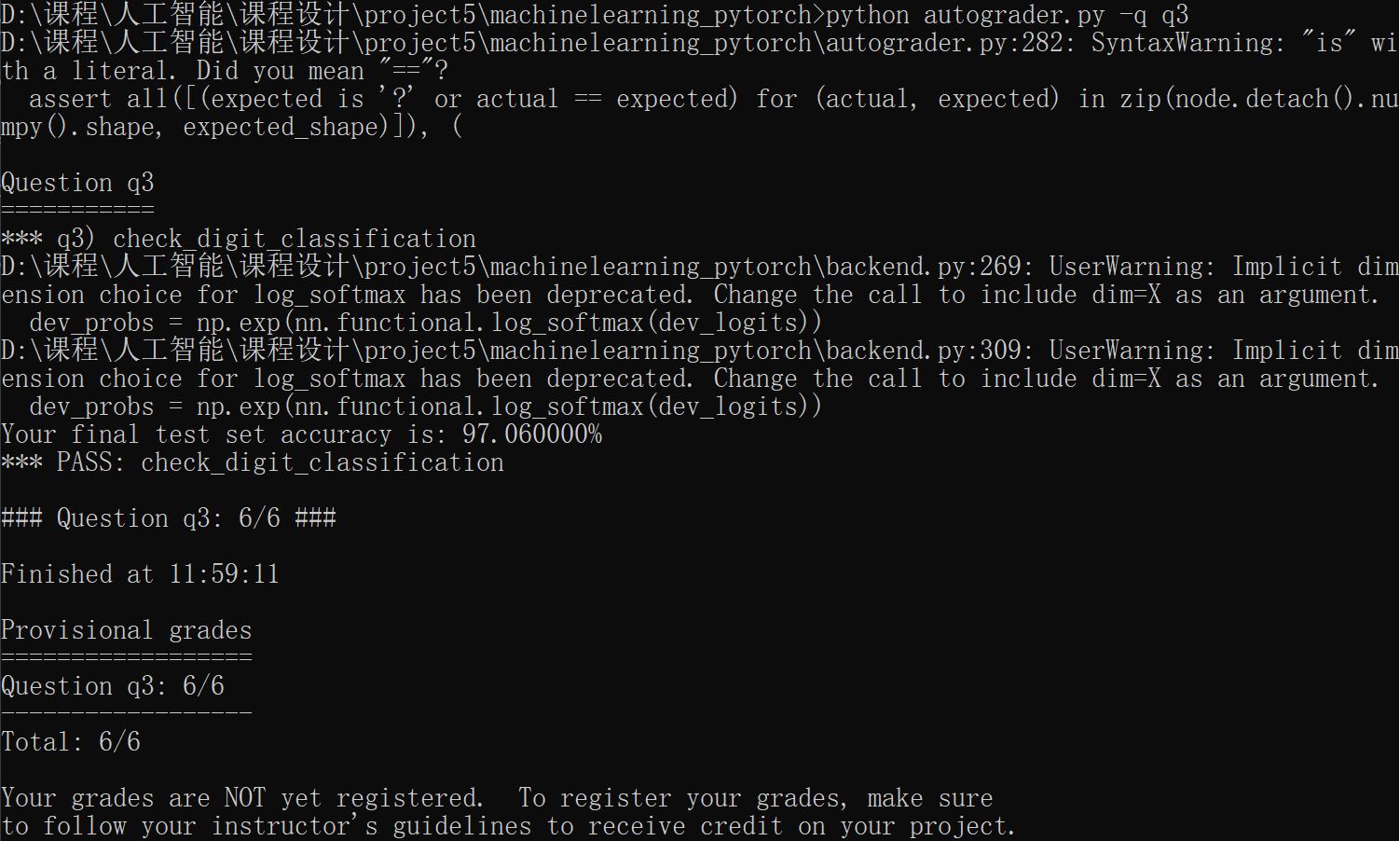
q1



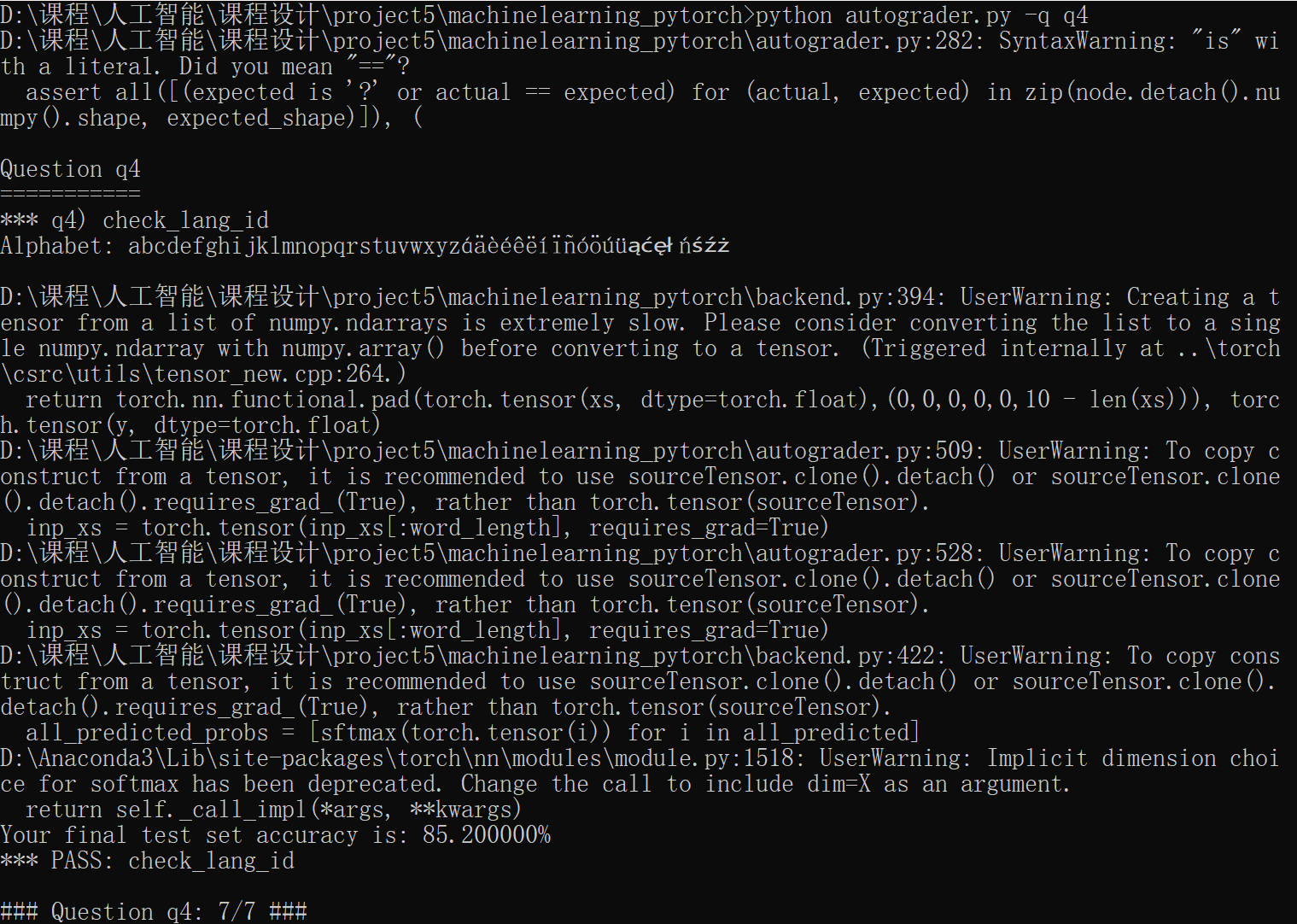
q2

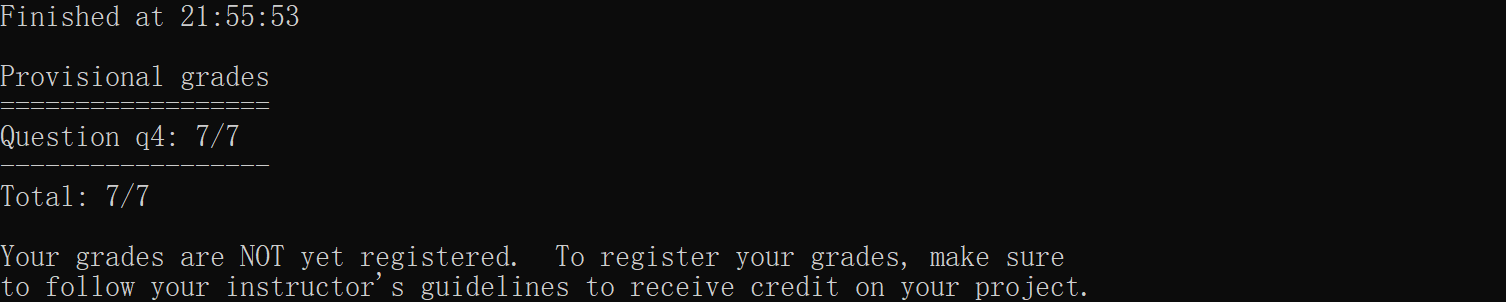


q3



q4





1. 总结与分析
2. 在编译的过程中，会出现OMP: Error #15: Initializing libiomp5md.dll, but found libiomp5md.dll already initialized.的报错。经过查询网络资料，得知需要在import os的后面加一句os.environ["KMP\_DUPLICATE\_LIB\_OK"]="TRUE"就可以解决问题。但是这样会改变不应该改变的代码。本质原因是由于在应用程序中同时加载了两个或多个版本的libiomp5md.dll库文件，需要修改环境路径后才可以在不修改代码的前提下成功通过编译。
3. q2向前传播的过程中，只对中间层的输出加relu激活函数，来实现非线性变换，而输出层不可以加。因为输出层需要有可能输出负的值，并且输出层需要是连续的而不能是离散的。
4. 在q3的设计中，我意识到不同结构的神经网络对训练的效率有很大的影响，最开始我采用和q2一样的双层结构，隐藏层100个接点，可以在很长的一段时间后逼近97%，但始终难以稳定的超过97%，后来换为3层，但仍然几乎难以稳定在97%以上。查阅资料后，发现除了神经网络的结构，优化器的学习率也对训练效率有影响，过低的学习率会导致学习速度过慢，而过高可能会导致模型始终无法收敛。但是自己设计一个固定的学习率，可能难以找到最合适的。所以采用scheduler来动态调节学习率。
5. q4中RNN的设计过程，只对中间层进行在时间上的循环，每次隐藏层中需要使用relu激活函数。在全部循环完成后，才进入输出层，输出层中不要relu激活函数。