实验内容

手写数字识别是计算机视觉领域的一个重要任务，广泛应用于自动邮件分拣、数字文档识别和数字化图书馆等领域。本实验中的项目旨在开发一个基于多层感知器（MLP）神经网络的手写数字识别系统，使用经典的MNIST数据集作为训练和测试数据。项目旨在通过深度学习技术实现高准确率的数字分类。具体要求：

1、观看《神经网络实践.mp4》视频课，熟知代码含义；

2、根据视频上机实操，要有个人风格的备注;

3、代码进行debug调试，查看每个参数值；

4、提出神经网络改进

实现步骤

1. 数据准备

数据文件中包含了 MNIST 数据集的图像和标签信息

2. 神经网络结构设计

使用多层感知器（MLP）神经网络来构建我们的手写数字识别模型。

3. 数据预处理和特征提取

对数据进行了预处理和归一化，对标签进行了编码。

4. 神经网络训练

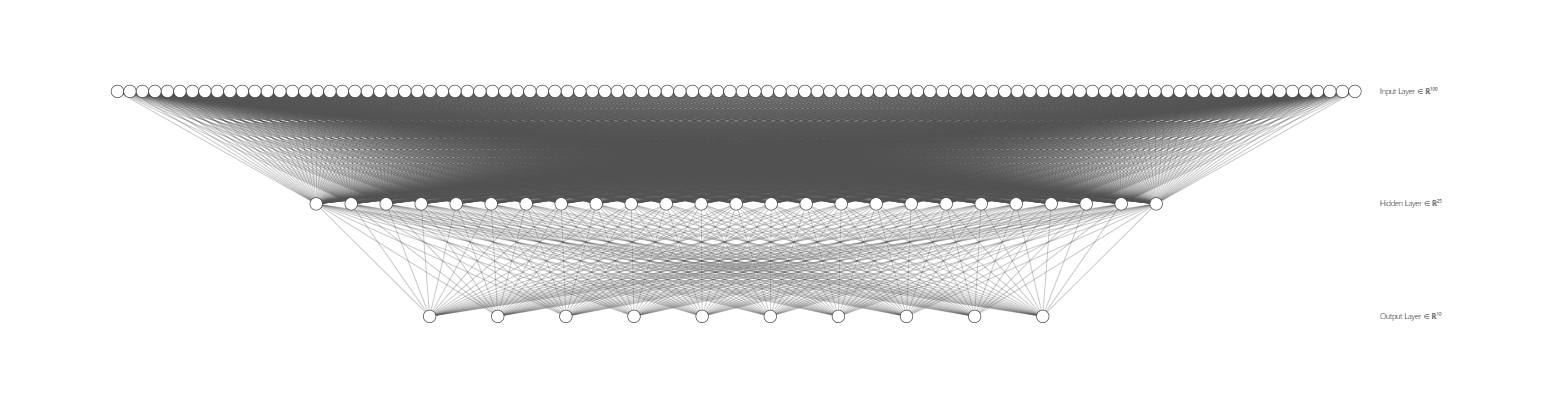
使用梯度下降算法来最小化损失函数，以逐步调整神经网络的参数（权重和偏置）。选择适当的学习率和最大迭代次数，以确保训练的有效性和效率。

5. 模型评估和性能分析

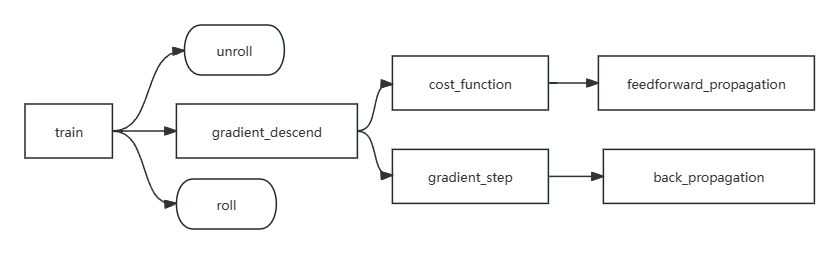
使用训练集和测试集来评估模型的性能，并计算了准确率作为性能指标。

实现方法

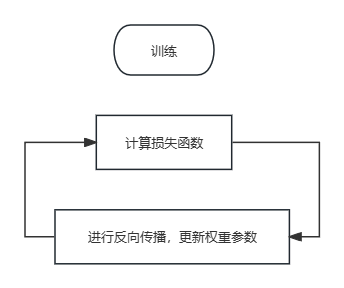
本次全连接神经网络中实现的是一个三层的网络，包括第一层输入层，第二层隐层和第三层输出层。第一层神经元个数为784（28\*28的image输入），第二层神经元个数为25，第三层神经元个数为10。在本实验中为了方便计算，偏置项和权重合并为一个参数矩阵，在第一层传播中，本为784\*25的权重矩阵和1\*25的偏置项矩阵合并为785\*25的参数矩阵，第二层传播中类似。



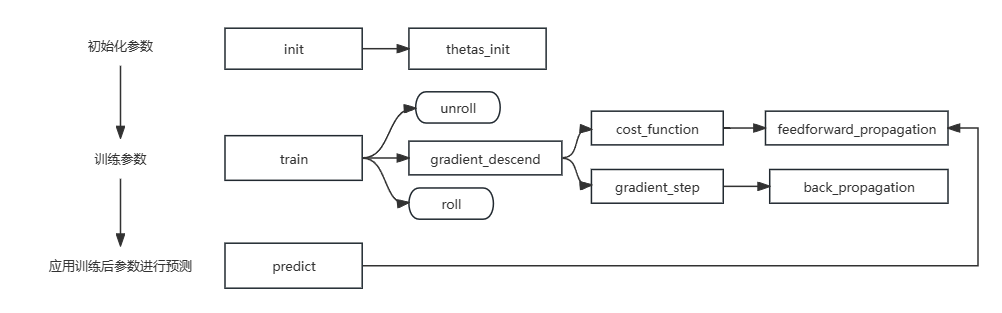
实验中MultiLayerPerceptron类中所用到的主要函数嵌套关系如图所示。



train方法接受处理好的数据，进行梯度下降法更新参数，返回迭代后更新的参数矩阵和历史损失值。其中cost\_function用来计算每次参数更新后的损失值，衡量训练程度好坏，gradient\_step用来计算参数更新的梯度值，每次参数更新后再次计算损失值，保存历史损失值方便留存查看训练效果。如果损失值随训练递减到较小值最后呈收敛态则说明参数更新成功，训练结果较佳。



MultiLayerPerceptron类完整方法示意图如下。

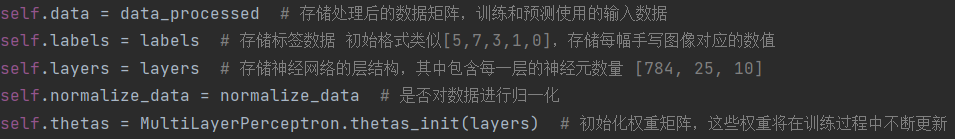


下面具体解释每个函数的作用。

注：灰色部分为python语法学习说明。

1. **初始化和数据处理：init**

初始化MLP，对输入数据进行处理并初始化权重。由于这些类的成员变量在后面频繁用到，这里给出较详细的注释。



1. **权重初始化：thetas\_init**

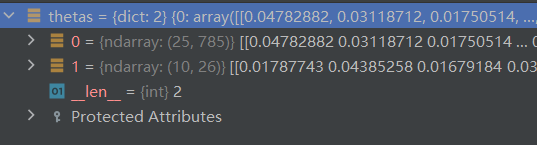
使用小随机值初始化每一层的权重矩阵。

创建一个字典 thetas，用于存储权重参数的矩阵，每个键代表一层（从输入层到输出层），对应的值是该层的权重矩阵。

使用循环迭代每一层，从输入层开始到倒数第二层，因为最后一层没有下一层的权重。

获取当前层和下一层的神经元数量，分别存储在 in\_count 和 out\_count 变量中。生成一个大小为 (out\_count, in\_count + 1) 的随机权重矩阵，使用 np.random.rand 函数生成，权重值乘以 0.05 来限制其范围在较小值。经过实验，如果将0.05调整为0.5，训练结果准确率只有30%左右。in\_count + 1是因为要加上偏置项参数，在上文中有过解释。

下面为实际调试结果，返回时显示thetas字典中有两层ndarray。



1. **训练：train**

本质上该函数中主要是为梯度下降做参数的预处理和传递。

train 函数接受两个参数：max\_iterations（最大迭代次数，默认为 1000）和 alpha（学习率，默认为 0.1）。

在梯度下降中，通常需要处理一维向量参数。函数调用thetas\_unroll 函数将权重矩阵展开成一维向量，并将结果存储在 unrolled\_theta 变量中。

然后传递训练数据、展开后的权重向量、网络层结构、以及迭代次数和学习率参数到gradient\_descent 函数，执行梯度下降算法，不断更新权重以减小损失函数的值，并返回优化后的权重参数矩阵和损失历史记录。

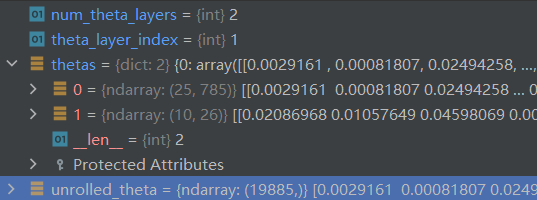
得到训练后的结果后，函数将该矩阵通过thetas\_roll 函数重新折叠，从一维向量转换成权重矩阵。它返回经过优化的权重参数self.thetas，更新了当前对象的最优参数矩阵，还返回了损失值的历史记录。

1. **权重矩阵的展开和重塑：thetas\_unroll 和 thetas\_roll**

**thetas\_unroll**

创建一个空的一维数组 unrolled\_theta，通过循环迭代每一层的权重矩阵，将每层的权重矩阵使用 flatten() 函数展平，然后将展平后的权重连接到 unrolled\_theta 中。

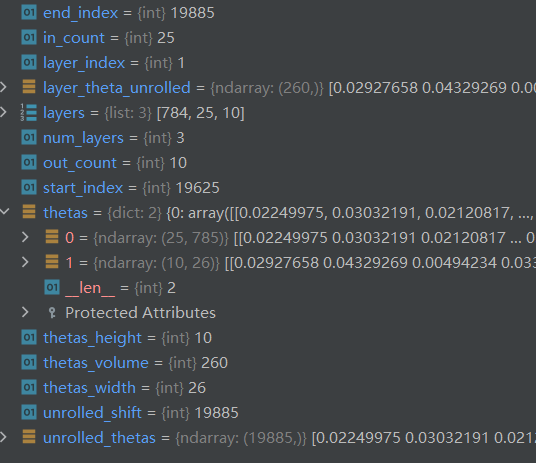
下面为实际调试结果，返回时显示unrolled\_thetas为19885\*1的一维向量。



thetas\_roll创建一个字典 thetas，用于存储各层的权重矩阵。使用 unrolled\_shift 变量来记录展开向量的当前位置（已记录数据的偏移量）。通过循环每一层获取当前层和下一层的神经元数量来计算权重矩阵的体积（即宽度乘以高度），从而确定从展开向量中的偏移量。

提取每一层展开向量中的权重数据，然后将其转换为原始的权重矩阵形状，并将其存储在 thetas 字典中，键为当前层的索引 layer\_index。遍历每层后，函数返回包含恢复后的权重矩阵的 thetas 字典。

下面为实际调试结果，返回时显示thetas恢复为原来的维度。



1. *flatten()*用于将多维数组（例如矩阵）转换为一维数组。这个函数会将多维数组中的所有元素按照顺序排列，形成一个一维数组。
2. *hstack()*在此处起到连接各展平后的一维向量为一个整体的一维向量的作用，它沿着水平方向（横向）堆叠多个数组。它将多个数组水平连接在一起，创建一个新的数组。必须确保连接的维度对应。
3. *reshape()*用于改变数组的形状，而不改变数组的数据。它接受一个新的形状（维度）作为参数，然后返回一个新的数组，具有指定的形状。此处用法是将一维向量折叠为矩阵的形状。
4. **梯度下降：gradient\_descent**

函数初始化 optimized\_theta，初始化为unrolled\_theta。

创建一个空的列表 cost\_history，用于存储每次迭代后的损失值。

使用循环进行最大迭代次数的迭代。在每次迭代中：

1. 进行前向传播，求出当前权重下的损失值，将当前损失值 cost 添加到历史损失值列表中。
2. 进行反向传播，求出梯度。使用学习率乘以梯度，然后从 optimized\_theta 中减去这个学习率调整后的梯度，以更新权重。

循环结束后，函数返回经过梯度下降训练后的优化权重 optimized\_theta 和损失历史记录 cost\_history。

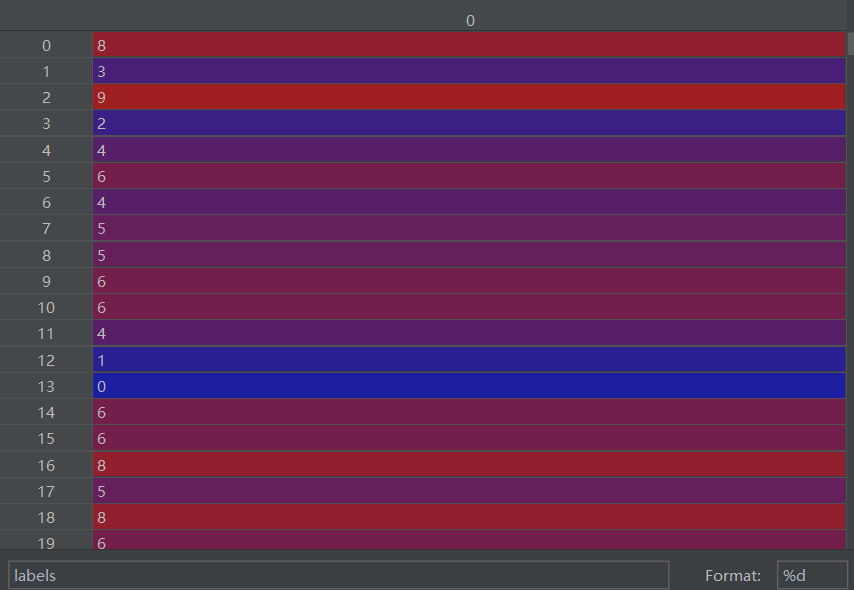
此函数中主要两个功能由cost\_function和gradient\_step实现，下文中会一一介绍。

1. **计算损失值：cost\_function**

首先获取训练数据的样本数量 num\_examples 和标签数量 num\_labels。

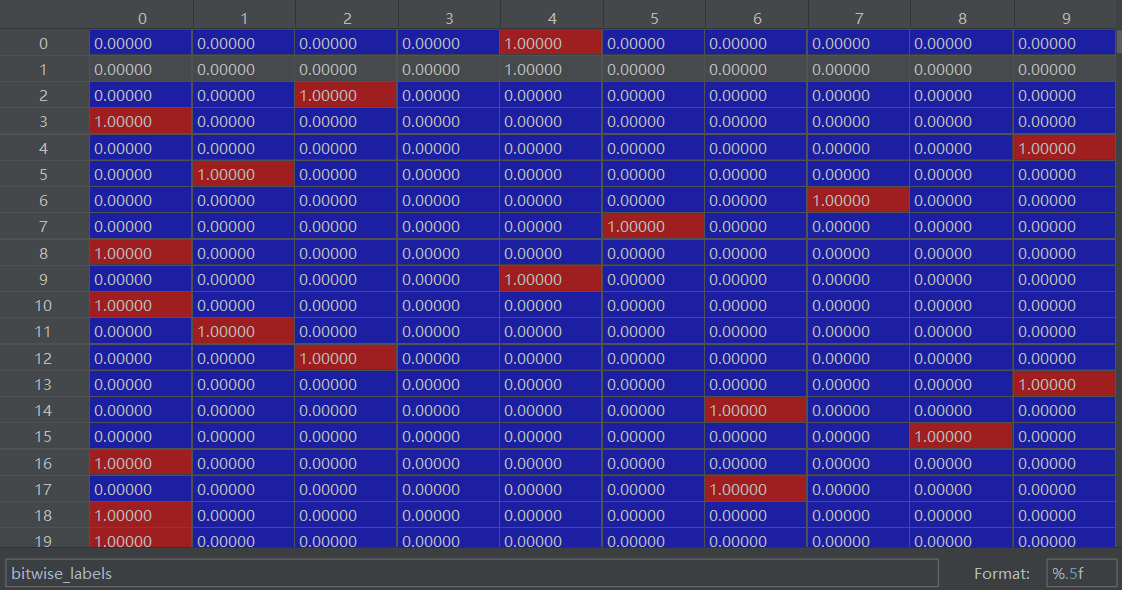
然后通过调用feedforward\_propagation 函数（下文中介绍）进行前向传播，使用当前权重 thetas 和训练数据 data 来计算网络的预测值 predictions。这些预测值表示了每个样本属于每个类别的概率。

因为上面得到的是每个类别的概率值，而labels中存储的是每幅图像对应的数值，不能直接做比对，需要进行下面的处理。





创建一个num\_examples\* num\_labels的零矩阵bitwise\_labels。这个矩阵将用于存储标签的二进制表示，其中每行对应一个样本，每列对应一个类别。使用循环迭代每个样本，将相应的标签（labels 中的整数值）在 bitwise\_labels 中设置为 1，表示哪个类别是正确的类别。



计算正确分类的损失bit\_set\_cost，它是通过对预测概率与实际标签进行逐元素乘法，然后对取对数的结果进行求和（对数概率通常用于计算分类问题的损失）。 同理计算未正确分类的损失bit\_not\_set\_cost。

最后计算总的损失 cost，并将其除以样本数量 num\_examples，以求得并返回平均损失。

1. 负索引用于从列表的末尾向前访问元素。layers[-1] 表示列表中的最后一个元素。
2. predictions[bitwise\_labels == 1] 使用布尔索引，从 predictions 中提取那些与正确标签匹配的预测概率。
3. **前向传播：feedforward\_propagation**

计算神经网络的前向传播，从输入到输出逐层计算神经元的激活值。

初始化当前层的激活值 in\_layer\_activation为训练数据 data。

对于每一层传播：

1. 获取当前层的权重矩阵 theta，通过将当前层的激活值 in\_layer\_activation 与权重矩阵 theta 的转置相乘，然后将结果传递给 S 型激活函数 sigmoid，计算当前层的输出激活值 out\_layer\_activation。

这里拿第一次传播举例（下同）：

1000\*785 dotproduct 785\*25 🡪 1000\*25



1. 在 out\_layer\_activation 的前面添加一列（偏置项），通过使用 np.hstack 函数连接一列值为1的向量和 out\_layer\_activation。

1000\*25 🡪 1000 \*26



1. 将 in\_layer\_activation 更新为 out\_layer\_activation，以便在下一次迭代中将其用作下一层的输入激活值。

在前向传播中，偏置项只用于计算下一层的输入，不包含在最终的输出中。通过 in\_layer\_activation[:, 1:] 去掉了第一列，去掉偏置项。



返回最终的 in\_layer\_activation。



in\_layer\_activation[:, 1:] 是一个针对NumPy数组的操作。

: 表示对数组的第一个维度（通常是行）进行切片操作。在这里，冒号表示选取所有行。

1: 表示对数组的第二个维度（通常是列）进行切片操作。在这里，它表示从第2列（索引1开始）到最后一列（包括最后一列）的所有列。

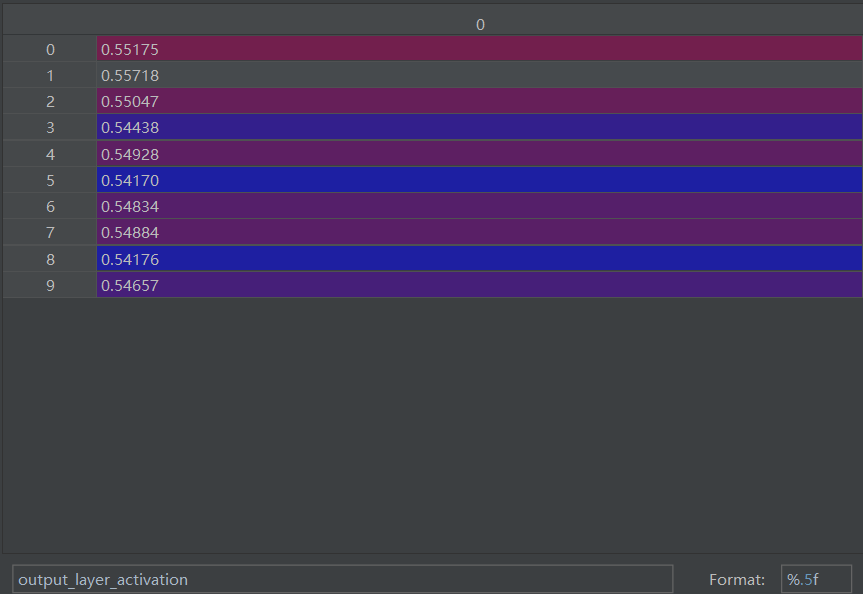
1. **反向传播：back\_propagation**

实现反向传播算法，计算权重的梯度以用于优化。这是训练的关键部分。

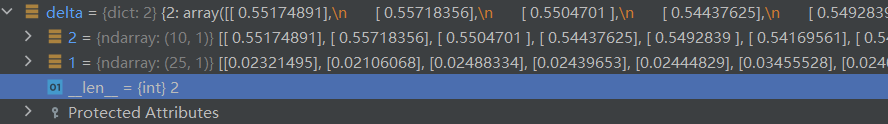
创建一个字典 deltas，用于存储每一层的梯度。初始化每一层的梯度为零。

通过循环迭代每个样本：

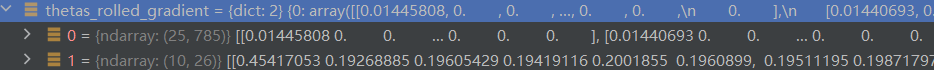
1. 初始化 layers\_inputs 和 layers\_activations 字典，用于存储每一层的输入和激活值。前向传播，求出当前层的输入和激活值（具体和上文中前向传播函数中类似），计算输出层激活值与实际标签的差异来获得输出层的误差delta。



1. 然后通过循环迭代从倒数第二层到输入层，计算每一层的 delta（通过使用权重矩阵 layer\_theta 和下一层的 delta，以及激活函数的导数 sigmoid\_gradient 来计算）。



遍历完后累加每一层的 delta 到 deltas 字典中。最后除以样本数量 num\_examples以获得平均梯度，返回 deltas 字典。



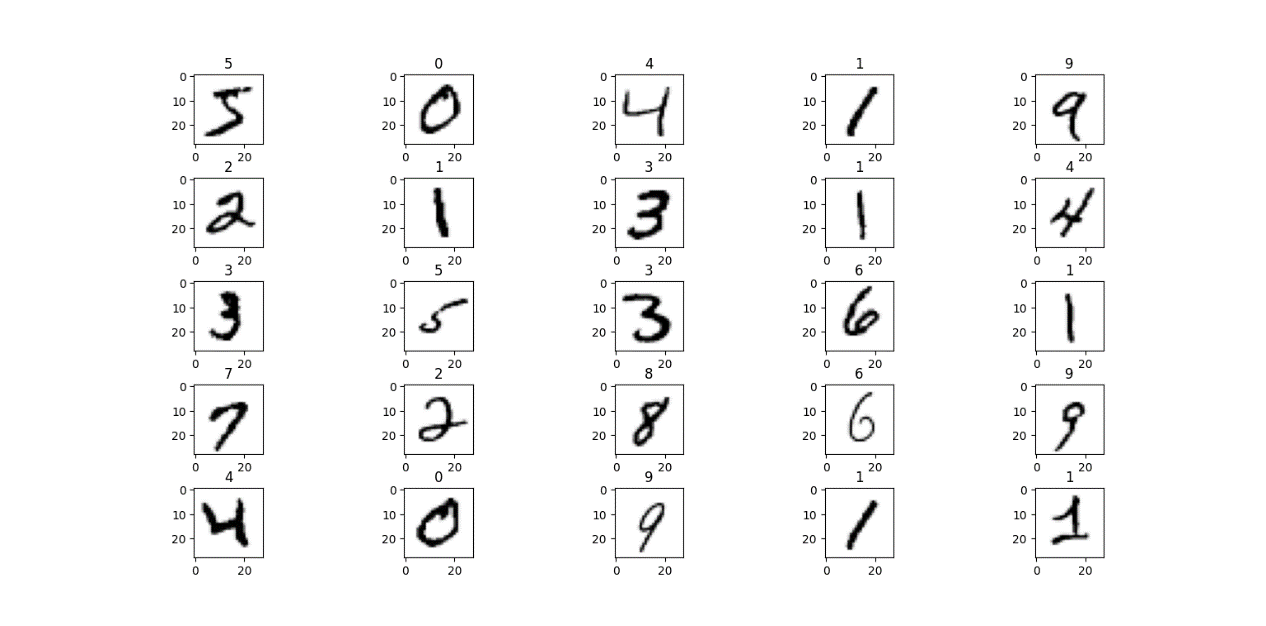
1. **预测：predict**

使用训练好的MLP对新数据进行预测。执行前向传播，计算每个样本的预测值。使用 np.argmax(predictions, axis=1)，找到每个样本的预测概率分布中概率最大的类别的索引，将预测结果的形状修改为 (num\_examples, 1)，以确保返回的是一个列向量，每行包含了相应样本的预测结果（和初始labels的形式一致）。

np.argmax(predictions, axis=1) 返回一个一维数组，其中每个元素是对应样本的预测结果中的最大值的索引。例如，如果在第一个样本的预测中，第三个类别具有最高的概率，那么结果数组的第一个元素将是3。

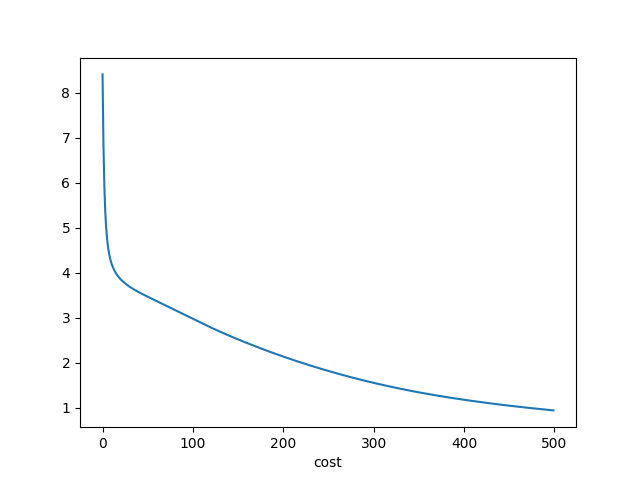
实验结果

mnist.py是对MultiLayerPerceptron类的应用，它从名为 mnist-demo.csv 的文件中读取数据。数据集包括手写数字的图像，每个图像有一个标签。下图为数据集的可视化展示。选取前 25 个样本并以 5x5 的网格方式显示它们的图像和对应的标签。

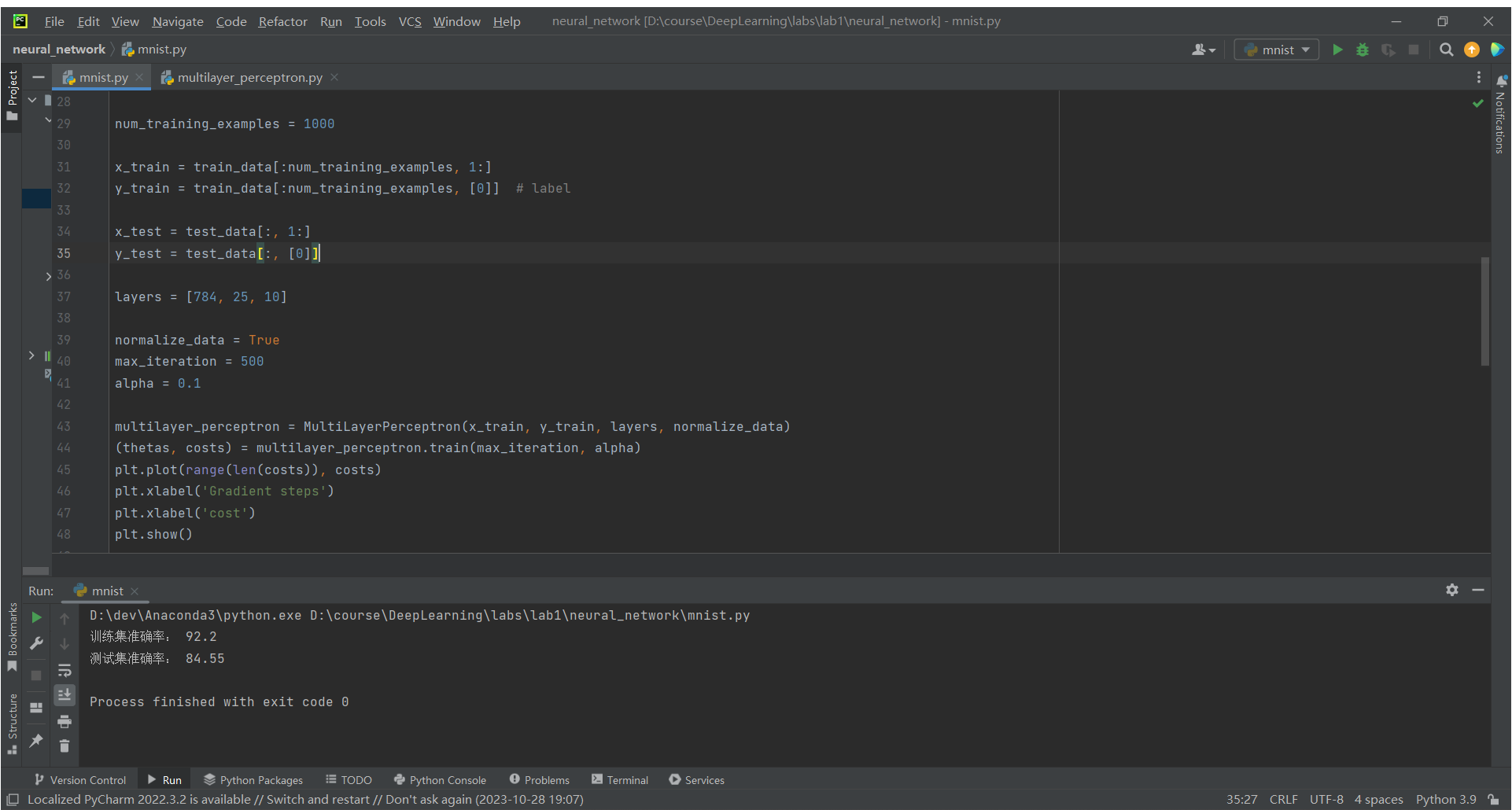


为了时效性选择用1000个样本作为总数据集，使用 data.sample(frac=0.8) 随机选择 80% 的数据作为训练数据集，剩余的 20% 数据被用作测试数据集。

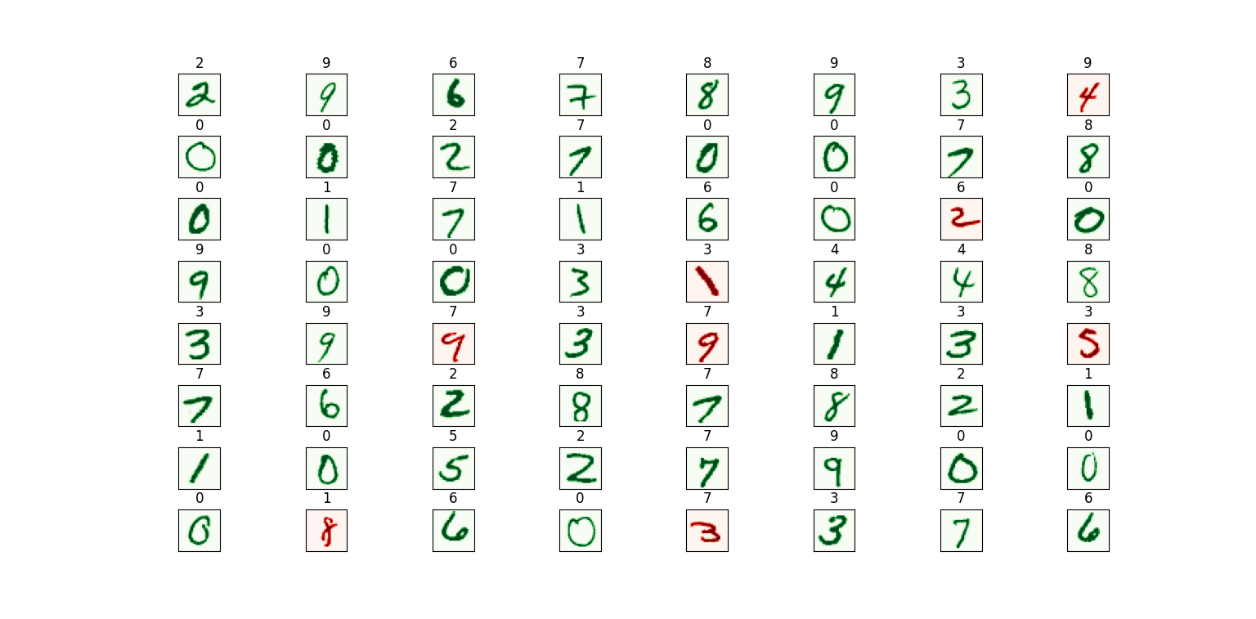
定义神经网络训练参数（学习率、最大迭代次数等）后创建 MultiLayerPerceptron 的实例，调用 train 方法进行训练，进行参数更新并记录训练过程中的损失值（costs）。使用 matplotlib 绘制损失值随迭代次数的变化图。观察到损失值呈递减状态，并逐渐收敛到较小值。



使用训练好的神经网络模型对训练集和测试集进行预测，计算训练集和测试集的准确率。可以看到训练集中准确值较测试集高，可能在训练过程中出现了过拟合。



选取前 64 个测试样本，并显示它们的图像以及模型的预测结果。预测正确的样本绿色显示，预测错误的样本红色显示。



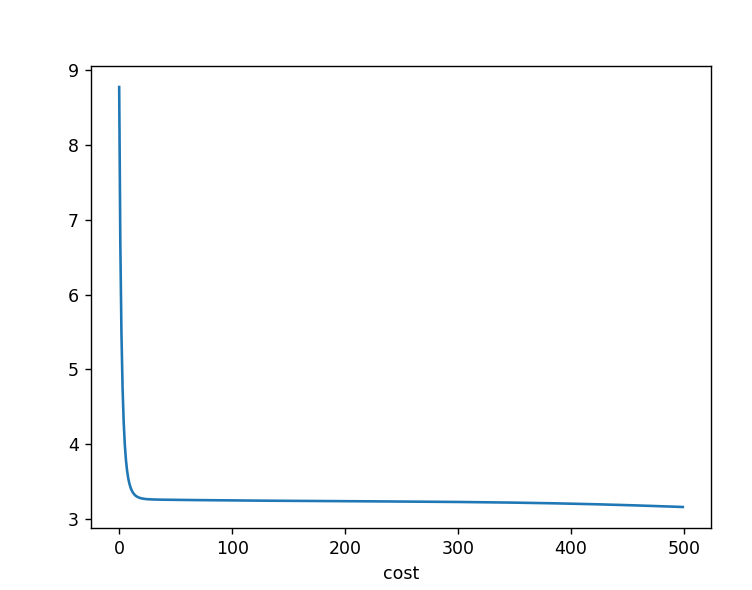
结论分析

实验中完成了一个能够识别手写数字的MLP神经网络模型。随后用数据集完成了在MNIST测试集上的准确率评估，反映了模型的性能，并实现了用于可视化损失曲线的图表和用于展示模型对测试样本分类的图像，以便直观了解模型的性能。

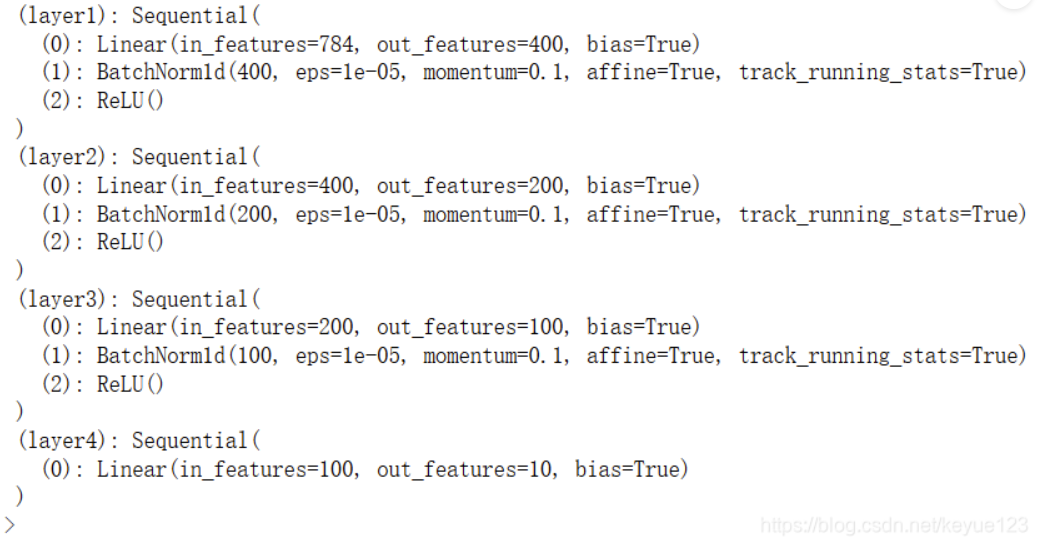
下面为一些改进的空间：

1. 网络结构优化：进一步调整神经网络的结构，尝试更多的隐藏层和神经元以提高性能。

下图为调整网络层结构为四层[784, 25, 25, 10]之后的损失曲线的图表，可以看到收敛速度加快。但是由于相关参数没有进行调整，最后的收敛值较大，不能达到较好的效果，还需要进行进一步的调整。

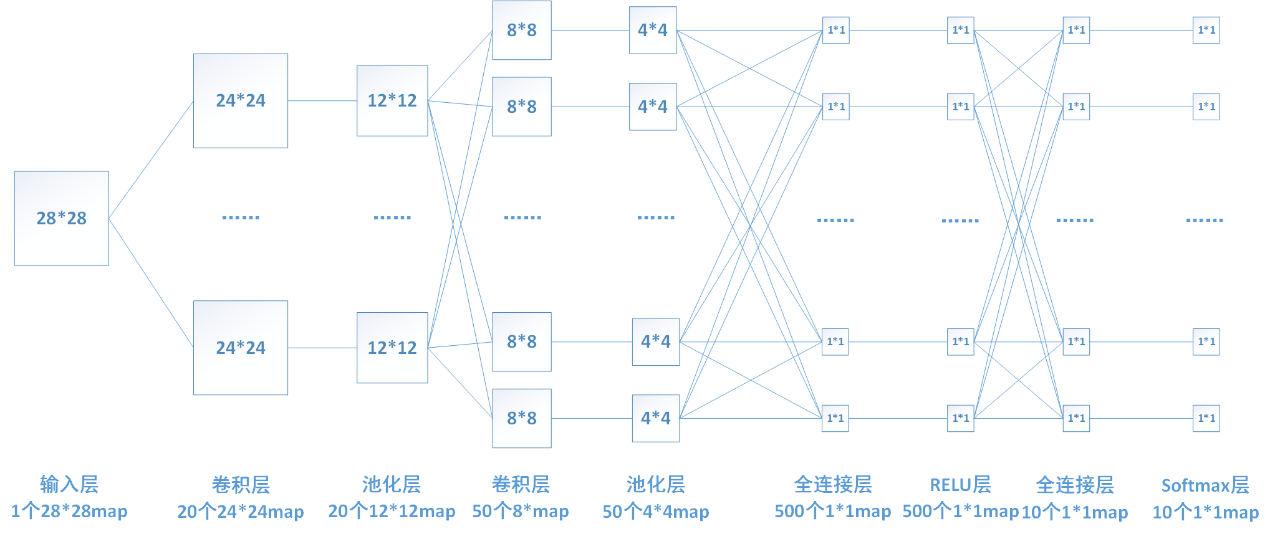


下图为一个可供参考的优化后的全连接网络模型，可以看到该模型中使用[784, 400,200,100,10]的网络结构，并使用了BatchNormld、ReLU激活函数。



1. 正则化和防止过拟合：在本实验中我们观察到的一定的过拟合现象，我们可以用正则化惩罚项等技术以降低过拟合风险。
2. 采用更复杂的网络结构：考虑使用卷积神经网络（CNN）或循环神经网络（RNN），适用于更复杂的图像或序列数据。

下图为可参考的一个针对手写数据集的优化后的卷积神经网络模型。



1. 函数结构优化：multilayer\_perceptron.py代码中有可以完善的代码片段，比如反向传播中包含了前向传播中的重复片段，或者是对thetas的roll和unroll的嵌套操作可以进行优化。提高代码复用性，完善工程结构。