神经网络·lab1

姓名： 代承谕

学号： 22307130165

一. 介绍

本项目仅使用`numpy`等基本库函数实现了简单的三层全连接神经网络（MLP）及卷积神经网络（CNN），完整代码包含网络模型，训练，测试和超参数查找四个部分，支持对网络架构进行自定义。

模型部分（`mynn`）包含网络层（`layers.py`），损失函数（`loss_fn`），优化器（`optimizers`），学习率调整器（`lr_scheduler`），评价标准（`metric.py`），模型架构（`models.py`）和运行器（`runner.py`）这些基本组件。

训练部分（`train.py`）支持在MNIST数据集上进行训练，训练时以5：1划分训练集（train set）与验证集（valid set）,训练完毕后在测试集（test set）上计算精度，并支持保存每个epoch的模型及最优模型的参数、模型架构及超参数、损失/精度-迭代次数图像及测试结果。注：若想训练卷积神经网络需要使用`train_CNN.py`。

测试部分（`test.py`）支持读取保存的模型参数，并在MNIST的测试集上进行测试。

超参数查找部分（`hyperparam_search.py`） 在一个随机选取的0.1x样本上对不同的初始学习率（init_lr），隐藏层维度（hidden_size），权重衰减系数（weight_decay_param）进行测试，分别计算不同组合在MNIST测试集上的损失和精度。

本报告主要介绍了模型架构，数据集，实验过程，模型优化等内容。其中，模型优化部分对要求1.2中的七个问题逐一作出了回应。

本项目的代码见cydai999/NeuralNetwork-PJ1

本项目的模型参数和所用数据集（MNIST）可以在https://drive.google.com/drive/folders/1GdLbs1mjBJ2-k6Kd5bLahiI7nGf_UP-v?usp=drive_link下载。

二. 模型

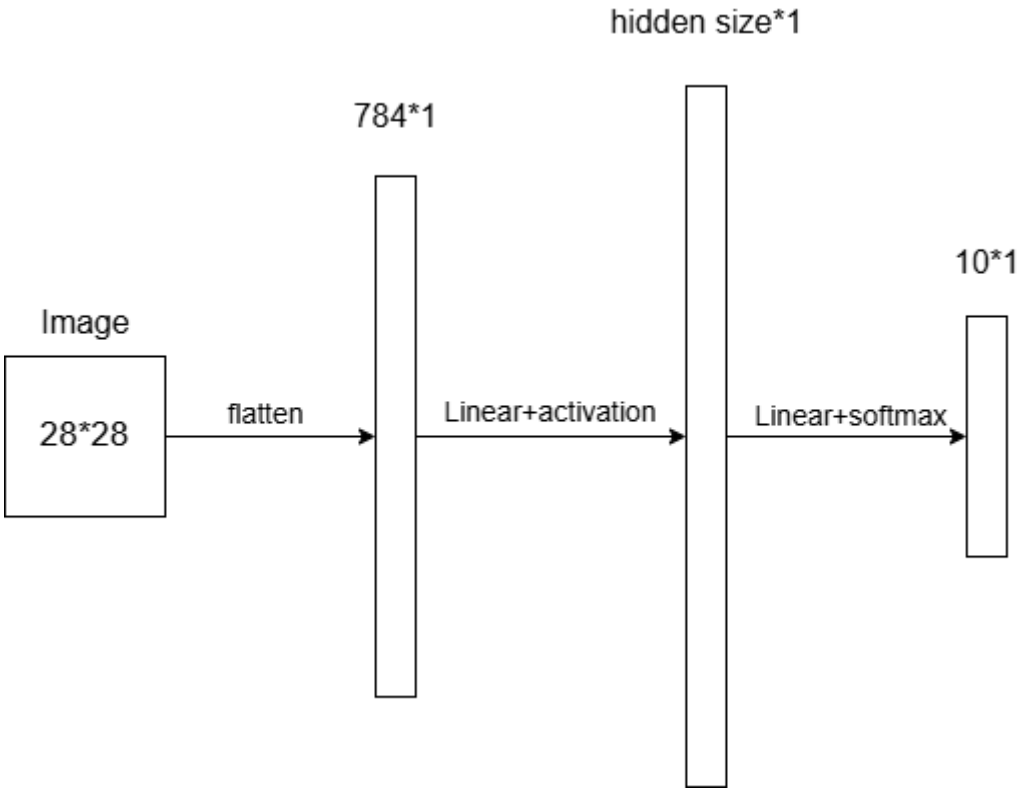
本项目中实现的网络为简单的三层全连接神经网络（MLP）及卷积神经网络（CNN），下面详细介绍模型的架构。

（一）神经网络组件

构成神经网络的所有组件均在`mynn`模块中，其中`layers.py`实现了线性层、卷积层、池化层与Sigmoid， ReLU和LeakyReLU这些激活函数的前向传播与反向传播过程；`loss_fn.py`实现了交叉熵损失，及其反向传播过程；`optimizer.py`中分别实现了随机梯度下降（SGD）及带动量的梯度下降（SGDMomentum）方法；`lr_schduler.py`中实现了不同的学习率调整策略，包含StepLR，MultiStepLR和ExponentialLR；`metric.py`中实现了精度（accuracy）计算；`models.py`实现了网络层拼接，前向/反向传播，以及保存/加载模型的方法；`runner.py`实现了模型的训练和评测功能。

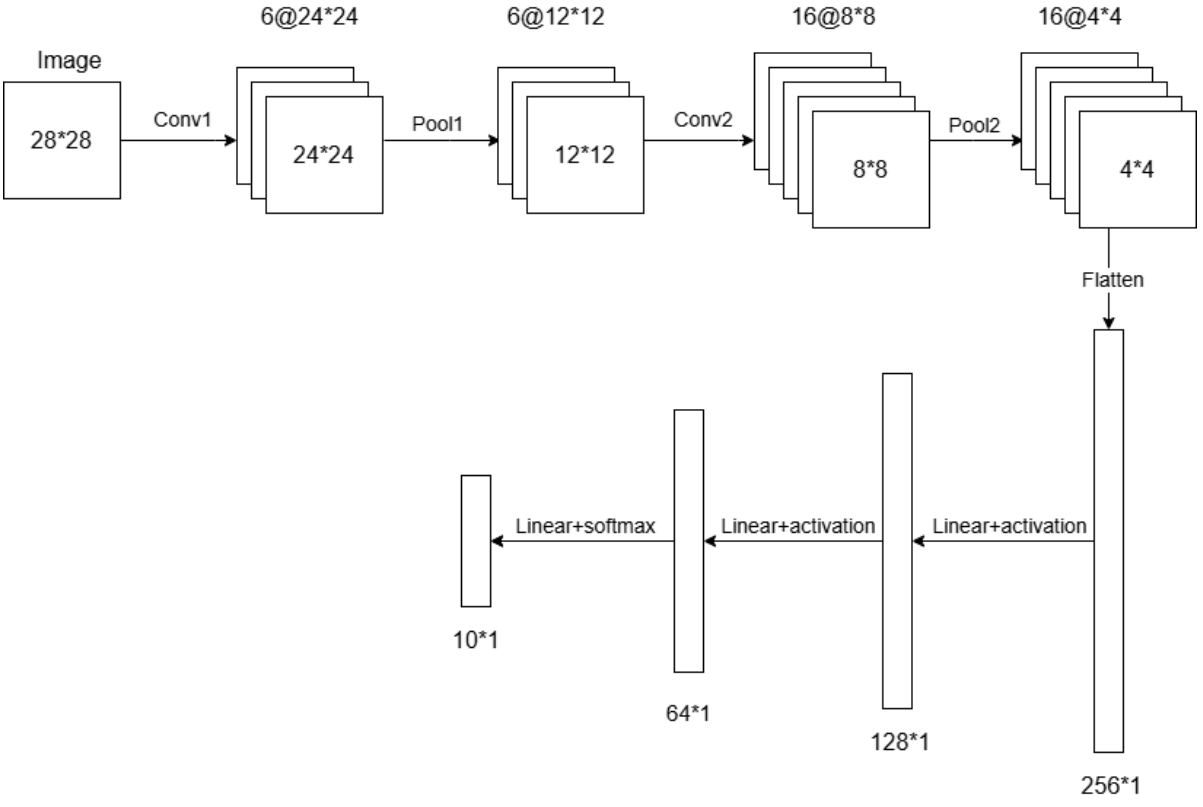
（二）全连接神经网络

三层全连接神经网络包含一个输入层，一个隐藏层和一个输出层。其中，输入层固定为784（28*28）个神经元（输入样本的尺度），输出层固定为10个神经元（分类数），隐藏层维度可调。网络包含两个线性层，在第一个线性层之后，设置了可选的激活函数以增强模型的拟合能力；在第二个线性层后，设置了softmax函数以实现归一化。网络架构如下图所示：



（三）卷积神经网络

本项目中的卷积神经网络仿照LeNet-5的构造，包含两个卷积+池化层和三个线性层。卷积层暂不支持填充与改变步长，且卷积核固定为正方形。池化层采用最大值池化。卷积层的通道数，卷积核大小，中间隐藏层大小个数可调。具体架构如下图所示：



三. 数据集

MNIST（Modified National Institute of Standards and Technology）数据集是机器学习领域最经典的图像分类基准数据集之一，主要用于手写数字识别任务。它包含70,000张28×28像素的灰度图像，涵盖了从0到9的10个阿拉伯数字类别，其中60,000张作为训练集，10,000张作为测试集。每张图像对应一个标签（0-9的整数），表示图像中的真实数字。

该数据集由美国国家标准与技术研究院（NIST）的原始样本经过重新混合和规范化处理后构建而成。图像中的手写数字经过尺寸标准化和居中处理，像素值被归一化为0到1之间的浮点数（或0-255的整数表示）。MNIST因其规模适中、结构清晰且易于加载的特点，成为机器学习算法（如逻辑回归、支持向量机）和深度学习模型（如卷积神经网络CNN）的入门级“Hello World”数据集。尽管其简单性已无法满足现代复杂模型的挑战需求，但它仍在教学、算法原型验证和基准测试中发挥重要作用。

四. 实验过程

这一部分主要介绍查找最佳超参数的过程和训练过程。

（1）参数查找

查找过程分为三个阶段：

1. 选择最优激活函数
2. 选择最优优化器
3. 查找最优超参数

以下分别介绍这几个过程。

1. 选择最优激活函数：

   为了观察不同激活函数的性能，在其他超参数相同的情况下分别对以Sigmoid, ReLU和LeakyReLU作为激活函数的网络进行训练与评测。所用超参数如下所示：

   ```
   {"model": {"size_list": [784, 1000, 10], "weight_decay_list":
   [0.0001, 0.0001]}, "optimizer": {"type": "momentum",
   "init_lr": 0.01}, "lr_scheduler": {"type": "stepLR",
   "step_size": 5, "gamma": 1}, "metric": {"type": "accuracy"},
   "loss function": {"type": "cross entropy"}, "train":
   {"batch_size": 32, "epoch": 20, "log_iter": 100}}
   ```

   结果如下：

Sigmoid

{"loss": 0.2944600544702009, "score": 0.9124}

ReLU

{"loss": 0.1968143834221269, "score": 0.9427}

LeakyReLU

{"loss": 0.19772292254917895, "score": 0.9423}

可以观察到，Sigmoid激活函数的效果最差，而ReLU与LeakyReLU的效果相近。而LeakyReLU可以解决可能存在的dead ReLU问题，因此之后选用LeakyReLU作为激活函数。

2. 选择最优优化器：

为了观察不同优化器的性能，在其他超参数相同的情况下分别使用SGD优化器和SGDMomentum优化器对神经网络进行训练和评测。所用超参数如下所示：

{"model": {"size_list": [784, 1000, 10], "act_func": "LeakyReLU", "weight_decay_list": [0.0001, 0.0001]}, "optimizer": {"init_lr": 0.01}, "lr_scheduler": {"type": "stepLR", "step_size": 5, "gamma": 1}, "metric": {"type": "accuracy"}, "loss function": {"type": "cross entropy"}, "train": {"batch_size": 32, "epoch": 20, "log_iter": 100}}

结果如下：

SGD

{"loss": 0.2547590204184765, "score": 0.9225}

SGDMomentum

{"loss": 0.19772292254917895, "score": 0.9423}

可以观察到，带动量的SGD优化器其效果要优于不带动量的SGD优化器。这可能是因为，动量机制的引入使得优化过程能更快脱离局部最优点或鞍点， 同时加快收敛速度。因此，之后选用SGDMomentum作为优化器。

3. 查找最优超参数：

选定激活函数与优化器之后，可以开始对超参数的查找。查找过程涉及到四个超参数：学习率（init_lr），隐藏层维度（hidden_size），权重衰减系数（weight_decay_param）和学习率衰减系数（gamma）。实际查找的过程可以分为两个阶段：首先查找学习率，隐藏层维度和权重衰减系数的最优组合，然后选取不同的学习率衰减系数观察模型性能。

a) 组合查找学习率，隐藏层维度和权重衰减系数：

每个超参数可能的取值如下所示：

```
init_lrs: [1e-1, 1e-2, 1e-3]
hidden_size: [2000, 1000, 500]
weight_decay_param: [1e-2, 1e-3, 1e-4]
```

具体实现过程是对这三个参数每个可能的组合对应的网络在相同数据上进行训练和测试。由于模型数量过多，没有在整个数据集上训练，而是随机选取了一个0.1x的样本数据集。其他超参数如下所示：

```
{"model": {"act_func": "LeakyReLU"}, "optimizer": {"type":
"Momentum"}, "lr_scheduler": {"type": "stepLR", "step_size":
5, "gamma": 1}, "metric": {"type": "accuracy"}, "loss
function": {"type": "cross entropy"}, "train": {"batch_size":
32, "epoch": 20, "log_iter": 100}}
```

所有参数组合在测试集上的损失和精度如下：

init_lr	hidden_size	weight_decay_param	loss	accuracy
1e-1	500	1e-2	0.267	0.9203
1e-1	500	1e-3	0.163	0.9581
1e-1	500	1e-4	0.186	0.9611
1e-1	1000	1e-2	0.167	0.9465
1e-1	**1000**	**1e-3**	**0.160**	**0.9595**
1e-1	1000	1e-4	0.186	0.9578
1e-1	2000	1e-2	0.162	0.9515
1e-1	2000	1e-3	0.151	0.9611
1e-1	2000	1e-4	0.182	0.9611
1e-2	500	1e-2	0.182	0.9441

init_lr	hidden_size	weight_decay_param	loss	accuracy
1e-2	500	1e-3	0.192	0.9436
1e-2	500	1e-4	0.190	0.9436
1e-2	1000	1e-2	0.182	0.9449
1e-2	1000	1e-3	0.193	0.9441
1e-2	1000	1e-4	0.197	0.9439
1e-2	2000	1e-2	0.178	0.9447
1e-2	2000	1e-3	0.200	0.9459
1e-2	2000	1e-4	0.215	0.9420
1e-3	500	1e-2	0.289	0.9165
1e-3	500	1e-3	0.276	0.9187
1e-3	500	1e-4	0.275	0.9192
1e-3	1000	1e-2	0.258	0.9235
1e-3	1000	1e-3	0.264	0.9200
1e-3	1000	1e-4	0.255	0.9251
1e-3	2000	1e-2	0.243	0.9269
1e-3	2000	1e-3	0.244	0.9261
1e-3	2000	1e-4	0.252	0.9263

分析：

1. **init_lr**:

   较高的学习率可以加快收敛速度，但会导致训练过程出现震荡，可能无法达到最优点；较低的学习收敛速度慢，需要更长时间才能收敛，但训练过程更加稳定。在实验中，取学习率为1e-3时模型性能明显低于1e-1和1e-2时。

2. **hidden_size**:

   在其他参数相同的情况下，隐藏层维数越高，模型参数越多，其拟合能力就越强，但训练开销也越大，同时可能导致过拟合问题。在实验中，观察到一般隐藏层维度越高，在测试集上的精度越高，但提高隐藏层维度的收益较低（以学习率取1e-1为例，隐藏层取2000维比1000维只能带来较低的精度提升，如下表所示）。

init_lr	hidden_size	weight_decay_param	loss	accuracy
1e-1	500	1e-2	0.267	0.9203
1e-1	1000	1e-2	0.167	0.9465
1e-1	2000	1e-2	0.162	0.9515
1e-1	500	1e-3	0.163	0.9581
1e-1	1000	1e-3	0.160	0.9595
1e-1	2000	1e-3	0.151	0.9611
1e-1	500	1e-4	0.186	0.9611
1e-1	1000	1e-4	0.186	0.9578
1e-1	2000	1e-4	0.182	0.9611

3. **weight_decay_param**:

   较大的权重衰减参数意味着更强的泛化能力，但可能导致模型本身的拟合能力较弱，因此选择时需要综合考虑其他因素（学习率和隐藏层维度）。

综合考虑模型性能与训练开销，最终选择的超参数组合如下：

```
init_lr: 1e-1
hidden_size: 1000
weight_decay_param: 1e-3
```

b) 查找学习率衰减系数

在固定其他参数的请况下，选用不同的学习率衰减系数，观察模型性能。所用学习率衰减系数如下所示：

```
gammas: [1, 0.5, 0.1, 0.05]
```

其他超参数如下所示：

```
{"model": {"size_list": [784, 1000, 10], "act_func": "LeakyReLU",
"weight_decay_list": [0.001, 0.001]}, "optimizer": {"type":
"Momentum", "init_lr": 0.1}, "lr_scheduler": {"type": "stepLR",
"step_size": 5}, "metric": {"type": "accuracy"}, "loss function":
{"type": "cross entropy"}, "train": {"batch_size": 32, "epoch":
20, "log_iter": 100}}
```

结果如下表：

gamma	loss	accuracy
1	0.114	0.9688
0.5	0.120	0.9674
0.1	0.129	0.9668
0.05	0.130	0.9649

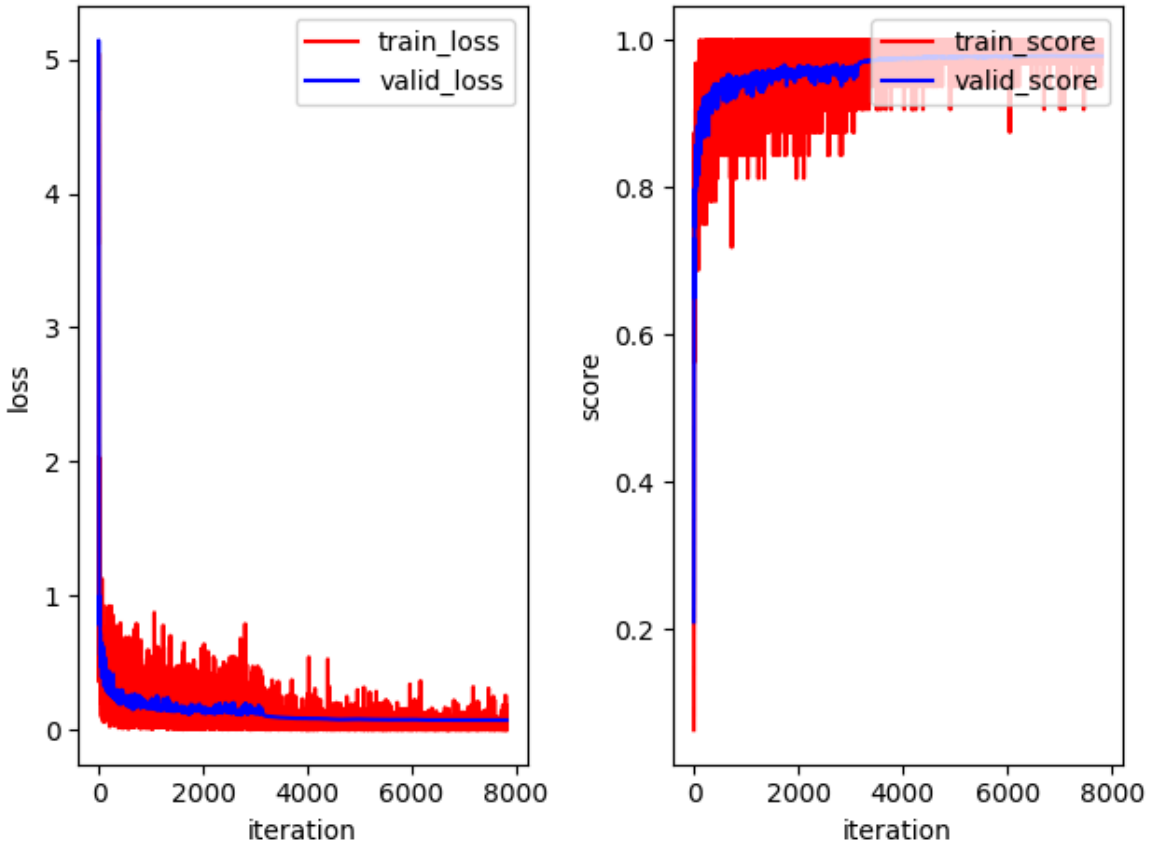
可以观察到，学习率调整机制似乎并没有起到效果。然而，在使用了学习率衰减机制的训练图像中，确实能观察到损失发生陡降现象。可能是由于StepLR设置的衰减周期较短，导致学习率衰减过快，在初期确实能够加快收敛，但随着训练时间延长，反而会起到反效果。为加速训练初期收敛速度，选用gamma=0.1作为学习率衰减系数，每隔2个epoch衰减一次。

(2) 模型训练

选用如下配置对模型进行训练：

```
{"model": {"size_list": [784, 1000, 10], "act_func": "LeakyReLU",
"weight_decay_list": [0.001, 0.001]}, "optimizer": {"type":
"Momentum", "init_lr": 0.1}, "lr_scheduler": {"type": "stepLR",
"step_size": 2, "gamma": 0.1}, "metric": {"type": "accuracy"},
"loss function": {"type": "cross entropy"}, "train":
{"batch_size": 32, "epoch": 5, "log_iter": 100}}
```

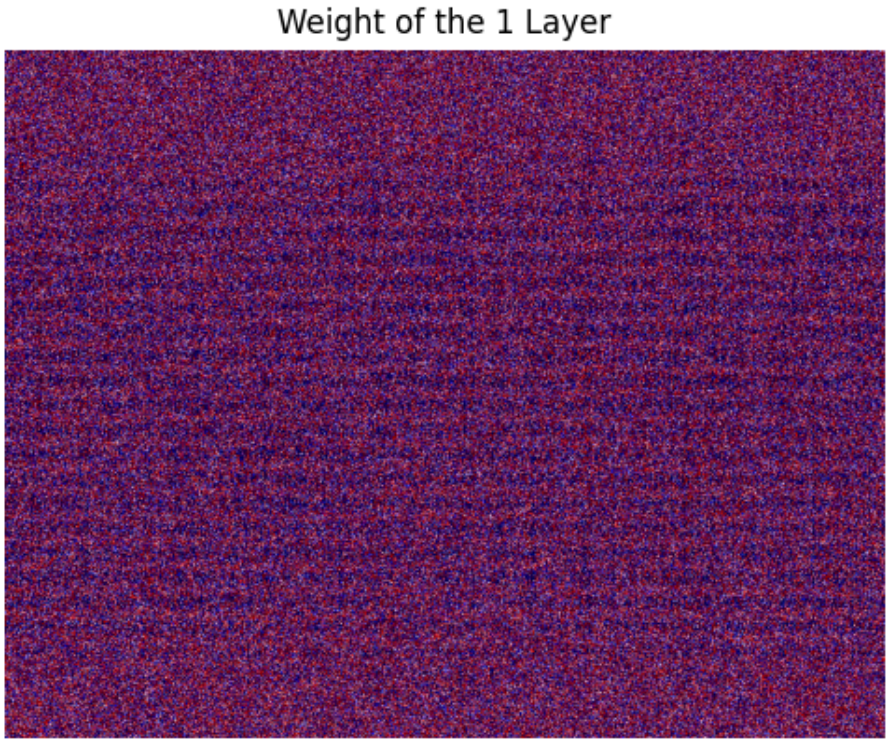
训练过程中的精度损失可视化如下：



最终结果如下所示：


```
{"loss": 0.06292302725672888, "score": 0.9806, "training_time":
2104.749430656433}
```


模型参数可视化如下图所示（仅可视化第一个线性层参数，即784*1000的矩阵）：



分析：

观察右侧的精度-迭代次数图像，可以看到模型在验证集上的精度总体保持上升趋势，刚开始上升速度较快，之后逐渐趋于平缓。值得注意的是，在第3000个iteration左右精度出现跃升现象，这是由于在该处发生了学习率衰减，变为了先前的0.1倍，导致模型能够更好地找到极值点。

上图是对第一个线性层权重参数的可视化，可以看到其内部存在一定的模式，但不够清晰明显。这可能是因为全连接层难以直接捕捉图像数据中的特征，要想得到更明显的特征需要使用卷积神经网络（CNN）。

五. 优化

这一部分将对要求1.2中的七个问题逐一作出回应。

（1）**改变隐藏层层数**

已在参数查找中“查找最优超参数”部分进行分析。

（2）**改变学习率**

1. 添加学习率调度器

   已在参数查找中“查找最优超参数”部分进行分析。

2. 使用动量优化器

已在参数查找中“选择最优优化器”部分进行分析。

（3）添加正则化方法

1. $l2$正则化

为测试$l2$正则化的效果，在前述最优模型基础上分别取`weight_decay_param=1e-3`（添加$l2$正则化）和`weight_decay_param=0`（不添加$l2$正则化）对模型进行训练，结果如下表：

weight_decay_param	loss	score
1e-3	0.063	0.9806
0	0.068	0.9803

理论上，添加$l2$正则化可以防止模型参数结构过于复杂，提高泛化能力，但是这里的效果不是很明显。原因可能是MNIST数据集比较简单，数据没有太大噪声，过拟合风险低，因此$l2$正则化效果不显著。

2. 早停法

为测试早停法有效性，设置`patience=2`（若发现两个epoch后，模型在验证集上表现未上升，则提前停止），`epoch=10`，分别测试使用和不使用早停法的表现。

（4）交叉熵损失

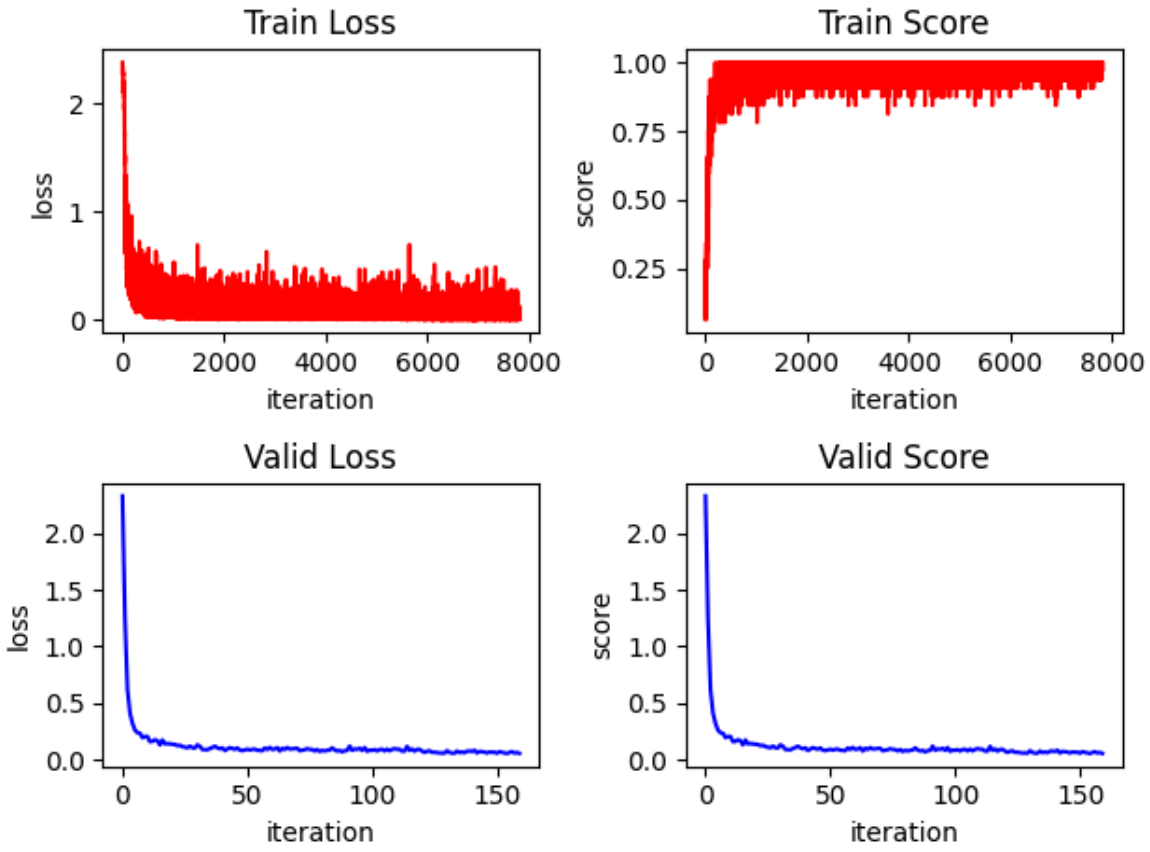
由于从一开始就是用的交叉熵损失，这部分略过。

（5）卷积神经网络

对前述卷积神经网络，采用如下参数进行训练：

```
{"model": {"type": "CNN", "channel_list": [1, 6, 16],
"kernel_list": [[5, 5], [5, 5]], "linear_size_list": [256, 128,
64, 10], "act_func": "LeakyReLU", "weight_decay_list": [0.001,
0.001], "linear_weight_decay_list": [0.001, 0.001, 0.001]},
"optimizer": {"type": "Momentum", "init_lr": 0.01},
"lr_scheduler": {"type": "stepLR", "step_size": 5, "gamma": 0.1},
"metric": {"type": "accuracy"}, "loss function": {"type": "cross
entropy"}, "train": {"batch_size": 32, "epoch": 5, "log_iter":
100}}
```

训练过程如下所示：



测试结果如下所示：

```
{"loss": 0.05227139650246726, "score": 0.9836, "training_time":
1007.4357085227966}
```

与MLP结果对比如下：

	loss	accuracy
CNN	0.052	0.9836
MLP	0.054	0.9828

分析：

卷积神经网络以比MLP更少的参数，实现了比MLP等同甚至更好的效果。这是因为卷积层可以更好地提取图像特征，从而提升分类准确性。

（6）**数据增强**

使用Albumentations库函数对训练集进行数据增强操作，具体代码为：

```python
import albumentations as A

def augment(img):
    transform = A.Compose([
        A.Rotate(limit=10, p=0.5),      # 以0.5的概率进行10度以内旋转
        A.ShiftScaleRotate(           # 以0.5概率进行图像尺寸1/10内尺度平移
            shift_limit=0.1,
            scale_limit=0.0,
            rotate_limit=0,
            p=0.5
        )
    ])
    return transform(image=img)['image']
```

将增强后的图像和标签与原训练集合并，作为新的训练集对模型进行训练。结果与不使用数据增强对比如下：

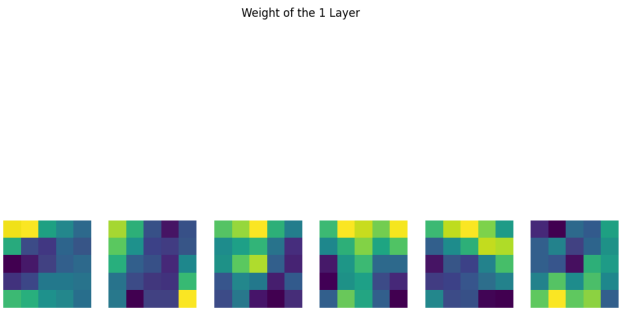
Data_augmentation	loss	accuracy
True	0.054	0.9826
False	0.063	0.9806

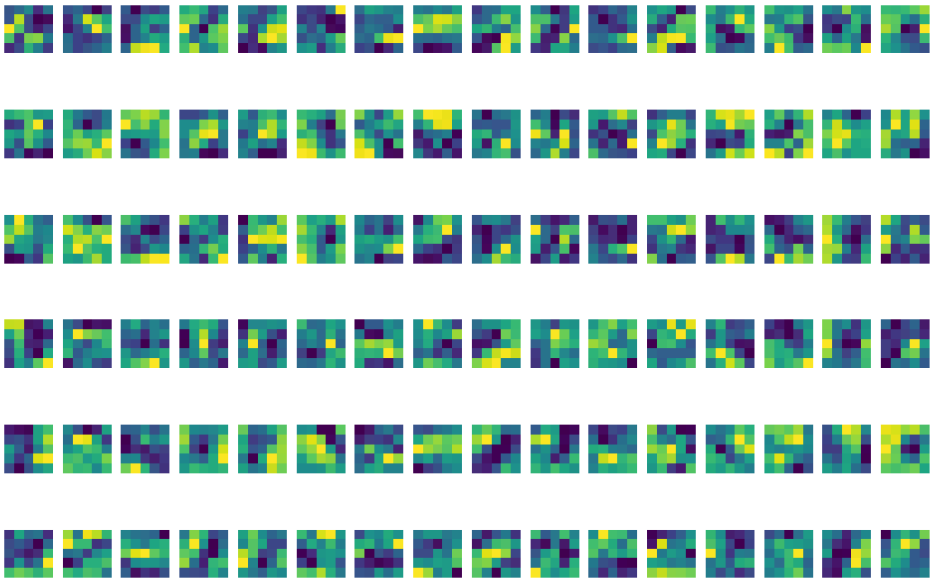
分析：

数据增强可以增强训练的稳定性，提升模型泛化性与鲁棒性，从而实现测试结果的提升。

（7）**参数可视化**

MLP的参数可视化已在实验过程中的"模型训练"部分展示。以下为CNN网络逐层参数可视化（只展示卷积层）：





分析:

卷积核确实展示出一些特定的结构，如水平横线、斜线等，但由于像素点数量太少，不具有较大可解释性，想要实现更好的可视化效果可能需要更大的卷积核。

六. 总结与思考

本项目实现了简单的三层全连接神经网络和卷积神经网络，在MNIST数据集上进行训练和测试， 同时实现了基本的超参数查找过程， 也探索了不同的优化方式。由于MNIST数据集比较简单，即使不做任何优化其测试精度也能达到0.9左右，在经过超参数查找后精度更是达到0.98。然而，在面对更加复杂的数据集（如CIFAR-10）时，该模型可能无法实现很好的效果。可能的进一步改进的方向有：

1. 增加神经网络层数。更深的神经网络可能会拥有更好的拟合性能。
2. 使用更复杂的网络架构，如添加Inception模块或使用残差连接技术。
3. 使用随机失活技术（dropout）以防止过拟合。