《人工智能基础》

实验报告

项目名称：手写数字识别

学生姓名：唐麒

学　　号：17301138

1. 实验数据集

1.1. 数据预处理

MNIST 数据集包含 70000 张手写数字的灰度图片（黑底白字，如下图所示），其中每张图片包含 28×28 个像素点。每张图片都有对应的标签，即图片对应的数字。数据集被分成两部分：包含 60000 张图片的训练数据集和包含 10000 张的测试数据集。

MNIST 数据集部分图片

为了使模型能对黑底白字和白底黑字的手写数字的识别均有较高的准确率，就需要将黑底白字的图片进行转换。

PaddlePaddle 接口对提供的数据进行了归一化、居中等处理，即数据集中的灰度图片以 28×28 的矩阵数据类型存储，矩阵元素的取值范围在 $[-1,1]$ 之间。

根据上述描述，可以通过对原始数据集内每一个元素取相反数的操作，获得对应的白底黑字图片（如下图所示）。具体流程可描述为分别定义训练数据集和测试数据集的数据处理函数（或接口），将数据集（包括标签）一次读入，并对数据集中的项逐一取反，将生成的矩阵与原始矩阵进行拼接，获得了黑底白字和白底黑字的混合数据集。

MNIST 数据集部分图片（处理后）

1.2. 数据加载

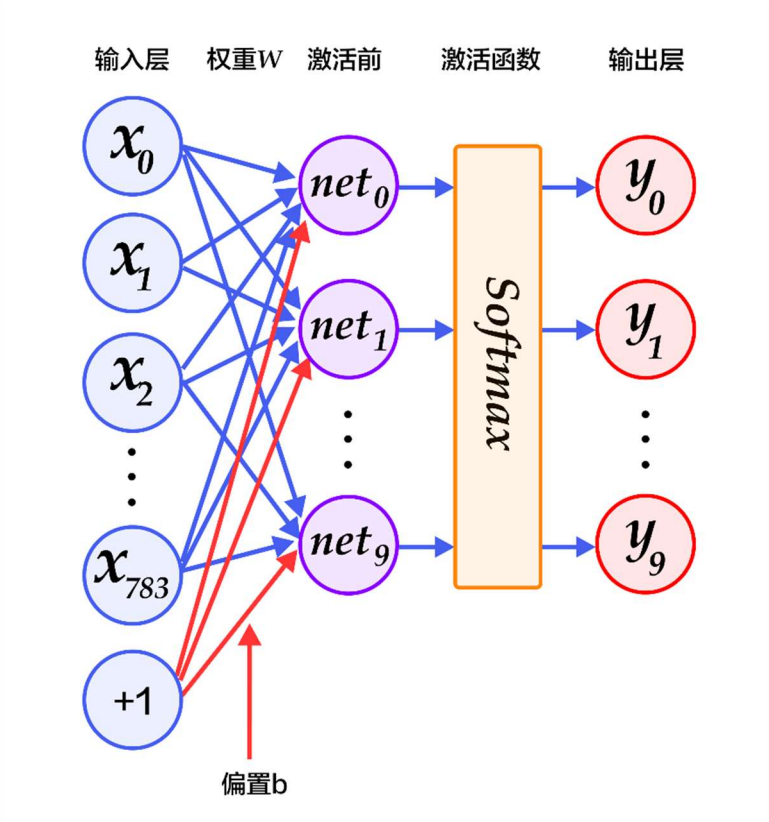
在对数据进行预处理之前，需要将数据进行加载。为了对数据进行上节描述的预处理，在将数据加载进 PaddlePaddle 框架时，通过其提供的接口 *minst.train()* 和 *mnist.test()* ，并将二者的 *buf_size* 和 *batch_size* 分别设置为 60000 和 10000，从而将数据一次性加载。在获得新的数据集之后，需要自定对应的 *reader* 进行加载（如下所示，见下页），通过 PaddlePaddle 提供的 *batch*、*shuffle* 等函数，调用自定义的 *reader* 函数，并将数据进行打乱，其中 *buf_size* 和 *batch_size* 分别设置为 512 和 128。

```
BUFF_SIZE = 512
BATCH_SIZE = 128

def train_r():
    def reader():
        train_reader = paddle.batch(paddle.reader.shuffle(padd
        le.dataset.mnist.train(),buf_size=60000),batch_size=60000)
        train_data = next(train_reader())
        train_data=np.array(train_data)
        train_image=train_data[:,0]
        train_label=train_data[:,1]

        inverse_train_image=- train_image
        train_image=np.hstack((train_image, inverse_train_image))
        train_label=np.hstack((train_label, train_label))

        for i in range(120000):
            yield train_image[i], train_label[i]
    return reader

def test_r():
    def reader():
        test_reader = paddle.batch(paddle.reader.shuffle(paddl
        e.dataset.mnist.test(),buf_size=10000),batch_size=10000)
        test_data = next(test_reader())
        test_data=np.array(test_data)
        test_image=test_data[:,0]
        test_label=test_data[:,1]

        inverse_test_image=- test_image
        test_image=np.hstack((test_image, inverse_test_image))
        test_label=np.hstack((test_label, test_label))

        for i in range(20000):
            yield test_image[i], test_label[i]
    return reader


train_reader = paddle.batch(paddle.reader.shuffle(reader=train_r(), bu
f_size=BUFF_SIZE),batch_size=BATCH_SIZE)
test_reader = paddle.batch(paddle.reader.shuffle(reader=test_r(), buf_
size=BUFF_SIZE), batch_size=BATCH_SIZE)
```

2. 网络结构

1） 多层感知机(Multilayer Perceptron, MLP)

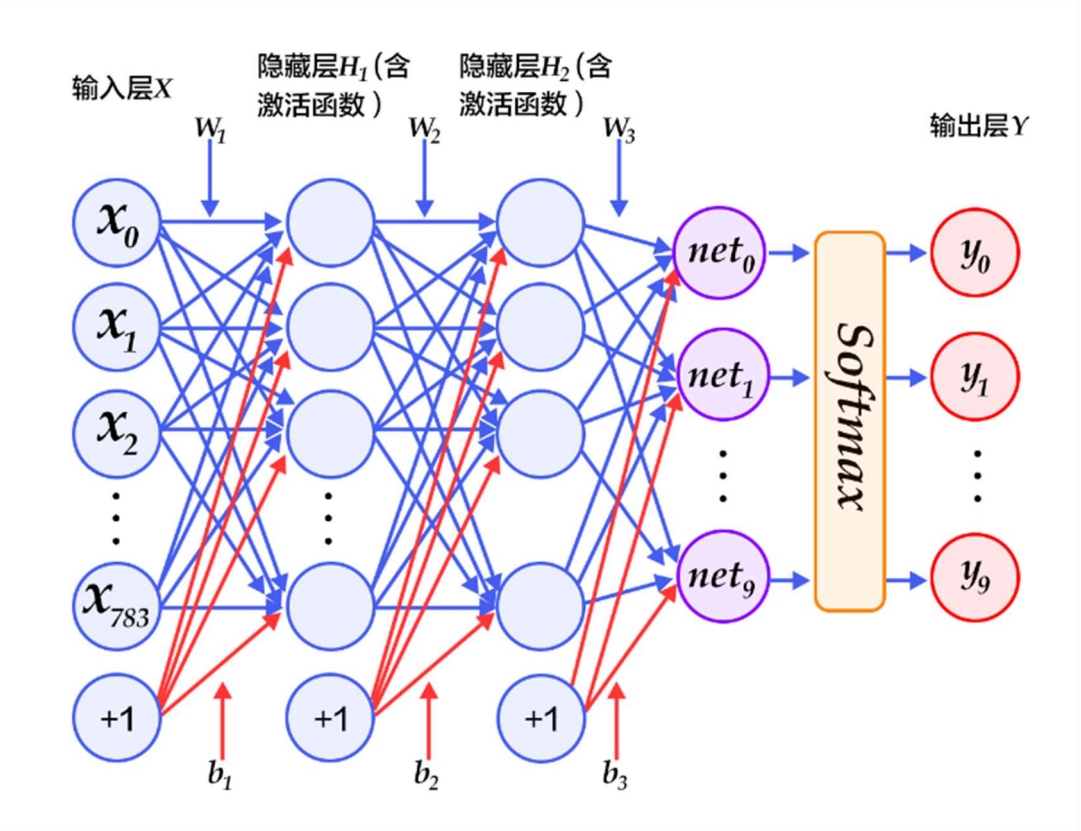
Softmax 回归模型是先将输入层经过一个全连接层得到特征，然后直接通过 Softmax 函数计算多个类别的概率并输出。



Softmax 回归模型

Softmax 回归模型采用了最简单的两层神经网络，即只有输入层和输出层，因此其拟合能力有限。为了达到更好的识别效果，在输入层和输出层中间加上若干个隐藏层，即多层感知机。

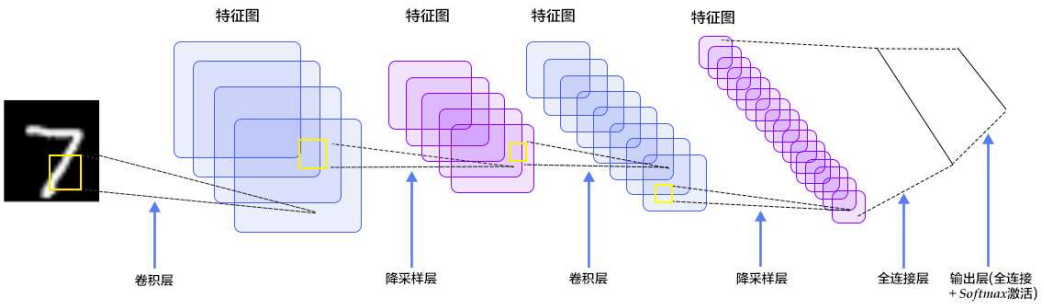
本项目的网络结构如下图所示，共有四层，一个输入层、两个大小为 100 的隐藏层（激活函数为 Relu）和一个大小为 10 的输出层（由于最后的分类结果为数字 0~9 共 10 个类别，所以输出层的大小为 10），输出层的激活函数是 Softmax。



多层感知机模型

2） Lenet-5

在多层感知器模型中，将图像展开成一维向量输入到网络中，忽略了图像的位置和结构信息，而卷积神经网络能够更好的利用图像的结构信息。LeNet-5 是一个较简单的卷积神经网络。下图显示了其结构：输入的二维图像，先经过两次卷积层到池化层，再经过全连接层，最后使用 Softmax 分类作为输出层。



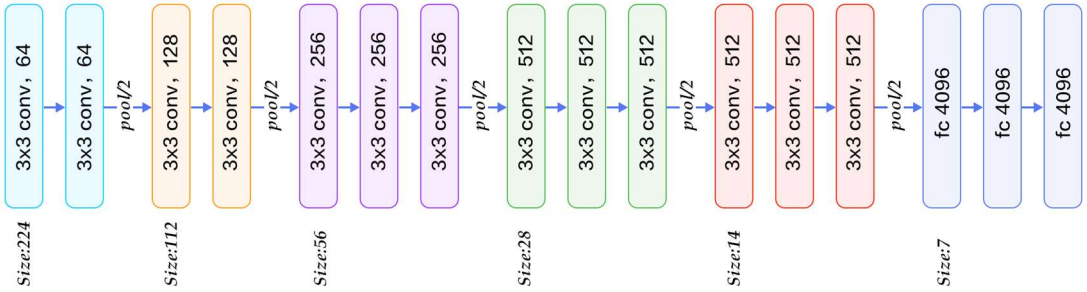
LeNet-5 模型

本项目中使用的 LeNet-5 模型包含一个输入层、两个卷积层、两个池化（降采样）层、和一个输出层（全连接层），其中：

- 卷积层 1:卷积核大小为 5×5，一共有 20 个卷积核；
- 池化层 1:池化大小为 2×2，步长为 2；
- 卷积层 2:卷积核大小为 5×5，一共有 50 个卷积核；
- 池化层 2:池化大小为 2×2，步长为 2；
- 输出层：激活函数为 Softmax 的输出层，大小为 10。

3） VGG

该模型相比以往模型进一步加宽和加深了网络结构，它的核心是五组卷积操作，每两组之间做 Max-Pooling 空间降维。同一组内采用多次连续的 3×3 卷积，卷积核的数目由较浅组的 64 增多到最深组的 512，同一组内的卷积核数目是一样的。卷积之后接两层全连接层，之后是分类层。由于每组内卷积层的不同，有 11、13、16、19 层这几种模型，下图展示一个 16 层的网络结构。



VGG 模型

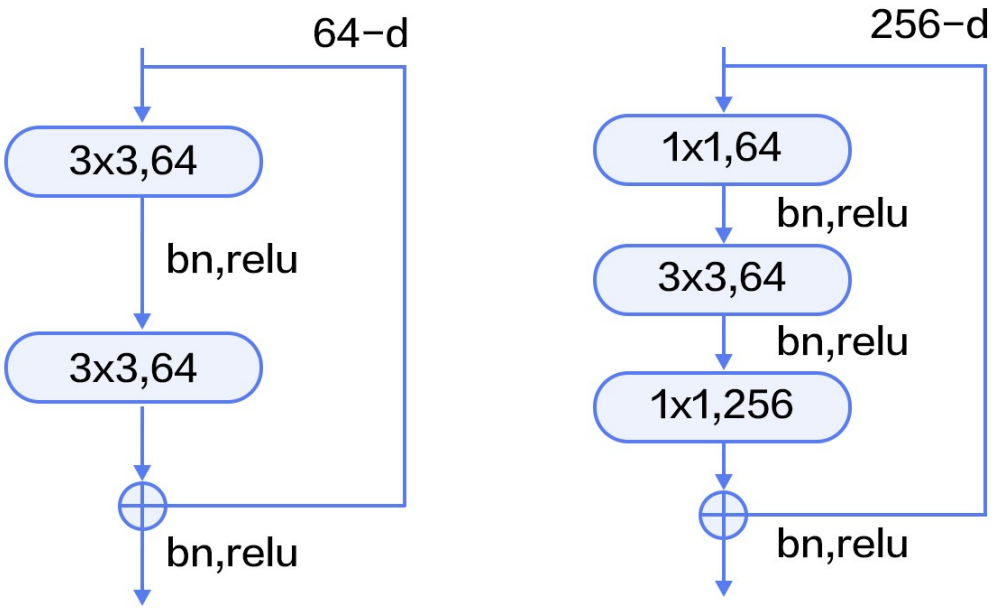
本项目定义一组卷积网络，卷积核大小为 3x3，池化窗口大小为 2x2，窗口滑动大小为 2。 共进行五组卷积操作，第一、二组采用两次连续的卷积操作，第三、四、五组采用三次连续的卷积操作。最后接两层 512 维的全连接层和一个激活函数为 Softmax 的输出层，大小为 10。

4） ResNet

VGG 网络试着探寻了一下深度学习中网络的深度与提高分类准确率的持续关系。一般印象当中，深度学习愈是深（复杂，参数多）愈是有着更强的表达能力。凭着这一基本准则 CNN 分类网络自 Alexnet 的 7 层发展到了 VGG 的 16 乃至 19 层，后来更有了 Googlenet 的 22 层。可当 CNN 网络达到一定深度后再一味地增加层数并不能带来进一步地分类性能提高，反而会招致网络收敛变得更慢，准确率也变得更差。

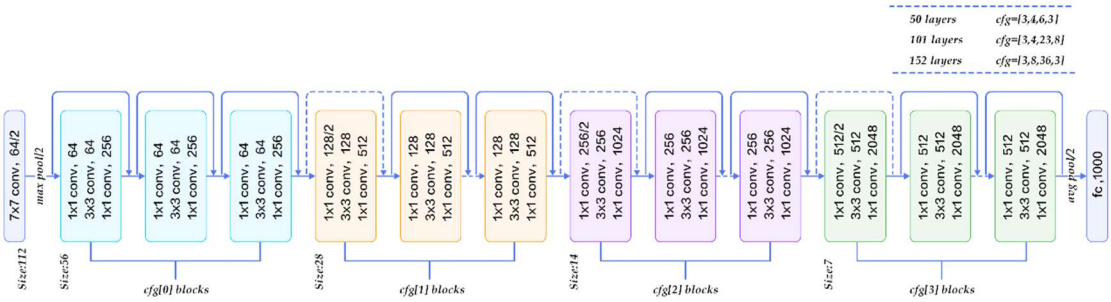
ResNet 通过使用多个有参层来学习输入输出之间的残差表示，而非像一般 CNN 网络（如 Alexnet/VGG 等）那样使用有参层来直接尝试学习输入、输出之间的映射。ResNet 有关实验表明使用一般意义上的有参层来直接学习残差比直接学习输入、输出间映射要容易得多（收敛速度更快），也有效得多（可通过使用更多的层来达到更高的分类精度）。

残差模块如下图所示，左边是基本模块连接方式，由两个输出通道数相同的 3×3 卷积组成。右边是瓶颈模块(Bottleneck)连接方式，之所以称为瓶颈，是因为上面的 1×1 卷积用来降维(图示例即 256->64)，下面的 1x1 卷积用来升维(图示例即 64->256)，这样中间 3×3 卷积的输入和输出通道数都较小(图示例即 64->64)。



残差模块

下图展示了 50、101、152 层网络连接示意图，使用的是瓶颈模块。这三个模型的区别在于每组中残差模块的重复次数不同(见图右上角)。ResNet 训练收敛较快，成功的训练了上百乃至近千层的卷积神经网络。



ResNet 模型

本项目的网络模型的连接为 1 个卷积层，卷积核大小为 3×3，一共有 16 个卷积核；1 个池化层，一个池化层，池化大小为 3×3，步长为 2；定义一组残差模块，由若干个基础残差模块堆积而成，每个基础残差模块，由三组大小分别为 1×1 ，3×3，1×1 的卷积组成的路径和一条"直连"路径组成，每一组卷积和的个数与传入本组残差模块的参数有关，分别为参数的 1 倍，1 倍和 4 倍。本项目共设置 3 组残差模块，传入残差模块卷积核的个数参数分别为 16,32,32。然后对网络做均值池化，最后是一个激活函数为 Softmax 的输出层，大小为 10。

3. 损失函数

交叉熵（cross entropy） 是分类问题中使用最为广泛的损失函数，其公式为

$$L = -[y log \hat{y} + (1 - y) \log(1 - \hat{y})]$$

PaddlePaddle 提供了可用于计算硬标签或软标签的交叉熵的函数：

1）硬标签交叉熵算法： $label[i_1, i_2, \dots, i_k]$表示每个样本的硬标签值:

$$output[i_1, i_2, \dots, i_k] = -log(input[i_1, i_2, \dots, i_k, j]),$$

$$label[i_1, i_2, \dots, i_k] = j, j! = ignore_index$$

2）软标签交叉熵算法：$label[i_1, i_2, \dots, i_k, j]$表明每个样本对应类别$j$的软标签值:

$$output[i_1, i_2, \dots, i_k] = -\sum_j label[i_1, i_2, \dots, i_k, j] \times log(input[i_1, i_2, \dots, i_k, j])$$

损失函数定义了拟合结果和真实结果之间的差异，作为优化的目标直接关系模型训练的好坏，输出层收到反馈，通过反向传播将反馈传向隐含层，并更新连接的权值。

4. 优化器

神经网络最终是一个最优化问题，在经过前向计算和反向传播后，Optimizer 使用反向传播梯度，优化神经网络中的参数。

1） SGD/SGDOptimizer

SGD 是实现随机梯度下降的一个 Optimizer 子类，是梯度下降大类中的一种方法。当需要训练大量样本的时候，往往选择 SGD 来使损失函数更快的收敛。

$$param\_out = param - learning\_rate \times grad$$

2） Momentum/MomentumOptimizer

Momentum 优化器在 SGD 基础上引入动量，减少了随机梯度下降过程中存在的噪声问题。

该优化器含有牛顿动量标志，公式更新如下：

$$velocity = \mu \times velocity + gradient$$
$$if(use\_nesterov):$$
$$\quad param = param - (gradient + \mu \times velocity) \times learning\_rate$$
$$else:$$
$$\quad param = param - learning\_rate \times velocity$$

3） Adam/AdamOptimizer

Adam 的优化器是一种自适应调整学习率的方法， 适用于大多非凸优化、大数据集和高维空间的场景。在实际应用中，Adam 是最为常用的一种优化方法。

Adam 更新如下：

$$t = t + 1$$
$$moment\_out = \beta_1 \times moment + (1 - \beta_1) \times grad$$
$$inf\_norm\_out = max(\beta_2 \times inf\_norm + \epsilon, |grad|)$$
$$learning\_rate = \frac{learning\_rate}{1 - \beta_1^t}$$
$$param\_out = param - learning\_rate \times \frac{moment\_out}{inf\_norm\_out}$$

4） Adamax/AdamaxOptimizer

Adamax 是 Adam 算法的一个变体，对学习率的上限提供了一个更简单的范围，使学习率的边界范围更简单。

Adamax 是基于无穷大范数的 Adam 算法的一个变种。Adamax 更新规则:

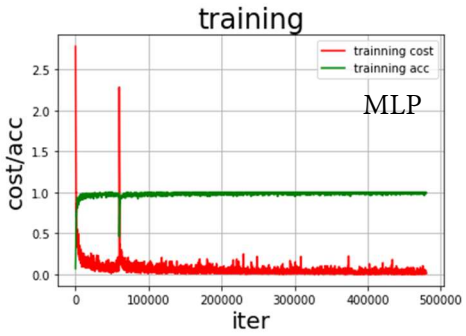
$$t = t + 1$$
$$moment\_out = \beta_1 \times moment + (1 - \beta_1) \times grad$$
$$inf\_norm\_out = max(\beta_2 \times inf\_norm + \epsilon, |grad|)$$
$$learning\_rate = \frac{learning\_rate}{1 - \beta_1^t}$$

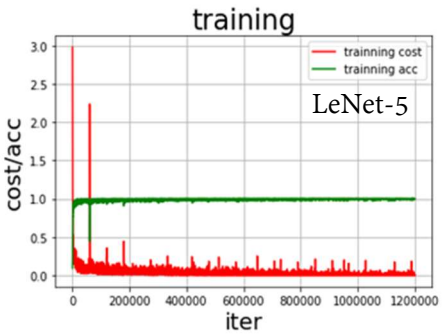
$$param\_out = param - learning\_rate \times \frac{moment\_out}{inf\_norm\_out}$$

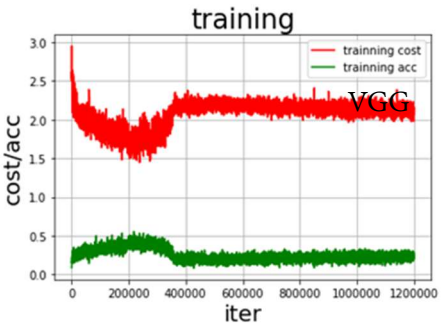
5. 实验结果

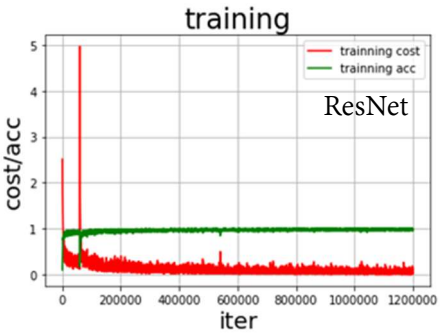
- 模型对比

	模型	损失函数	优化器	学习率	训练回合	准确率
1	MLP	交叉熵损失函数	Adam 算法	0.001	10	0.98517
2	LeNet-5	交叉熵损失函数	Adam 算法	0.001	10	**0.99826**
3	VGG	交叉熵损失函数	Adam 算法	0.001	10	0.22666
4	ResNet	交叉熵损失函数	Adam 算法	0.001	10	0.99721





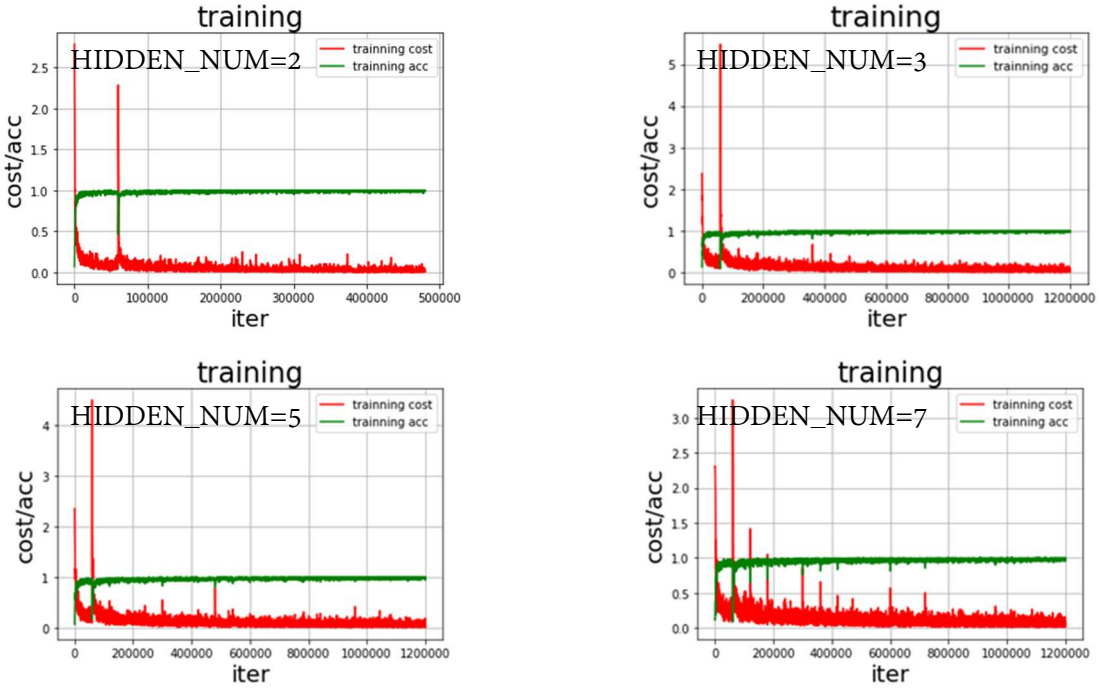




分析：在模型的选择上，依次对 MLP、LeNet-5、VGG 和 ResNet 进行了训练，随着模型的严谨，网络的层数在不断增加，即模型的复杂度在不断提高。从训练结果来看，在处理后的 MNIST 数据集上进行训练， MLP 和 CNN 均能取得比较好的训练效果，而 VGG 由于层数更深，网络收敛变得更慢，准确率也变得更差，此外，与数据集的体量也有一定的关系。而 ResNet 通过使用多个有参层来学习输入输出之间的残差表示，解决了 VGG 层数过高带来的问题，使得收敛速度更快，分类精度更高。

- 感知机隐藏层层数

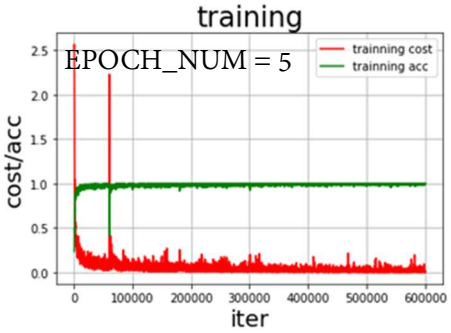
	隐藏层层数	损失函数	优化器	学习率	训练回合	准确率
1	2	交叉熵损失函数	Adam 算法	0.001	10	0.98517
2	3	交叉熵损失函数	Adam 算法	0.001	10	0.98507
3	5	交叉熵损失函数	Adam 算法	0.001	10	0.98219
4	7	交叉熵损失函数	Adam 算法	0.001	10	0.97990

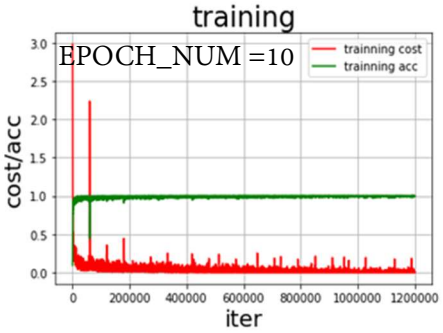


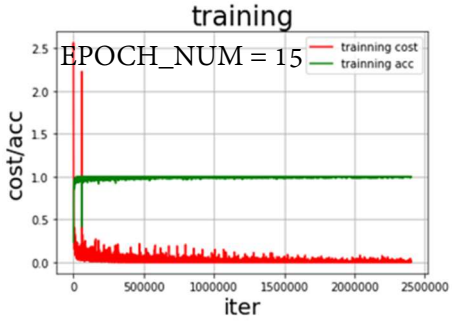
分析：可以看到，对于处理后的 MNIST 数据以及 MLP 模型，随着网络隐藏层层数的增多，模型训练的准确度在逐渐下降，与之前的对比实验中， VGG 的准确率在四个模型中最低的结果相似，初步得出结论，在深度学习中，简单地增加网络的深度，并不一定能提高模型的准确率，模型深度与准确率持续提高的关系需要建立在数据体量的持续增大和模型复杂度的进一步提高。

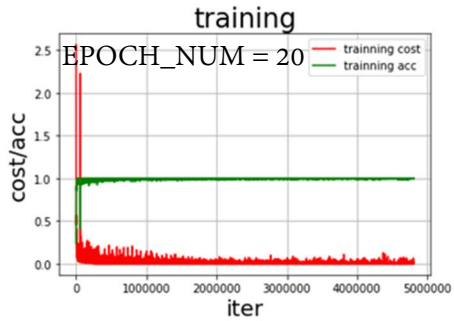
- Lenet-5 训练回合

	模型	损失函数	优化器	学习率	训练回合	准确率
1	LeNet-5	交叉熵损失函数	Adam 算法	0.001	5	0.99497
2	LeNet-5	交叉熵损失函数	Adam 算法	0.001	10	0.99826
3	LeNet-5	交叉熵损失函数	Adam 算法	0.001	15	0.99930
4	LeNet-5	交叉熵损失函数	Adam 算法	0.001	20	0.99960





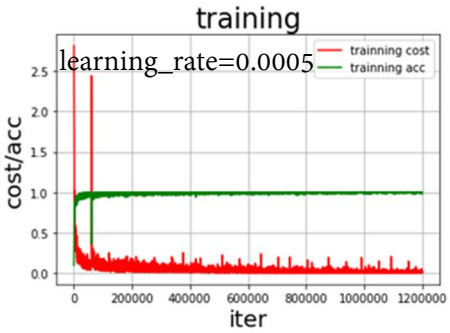


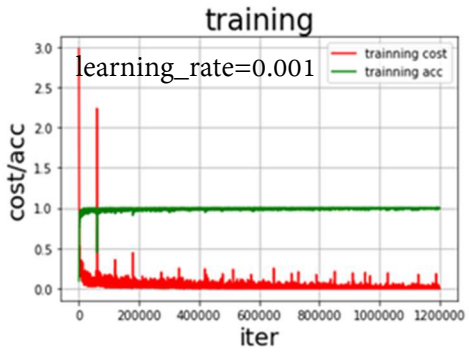


分析：在简单地调整训练回合后，可以看到 LENET-5 模型在 5~20 的范围内，随着训练回合的增大，准确率有所提高，但在之前的调试过程中也发现，当训练回合取到更大时，准确率会随着训练回合的增大而产生震荡，难以收敛，因此，并不意味着训练回合越大越好。

- Lenet-5 学习率

	模型	损失函数	优化器	学习率	训练回合	准确率
1	LeNet-5	交叉熵损失函数	Adam 算法	0.0005	10	0.99866
2	LeNet-5	交叉熵损失函数	Adam 算法	0.001	10	0.99826
3	LeNet-5	交叉熵损失函数	Adam 算法	0.0015	10	0.99761
4	LeNet-5	交叉熵损失函数	Adam 算法	0.002	10	0.99771

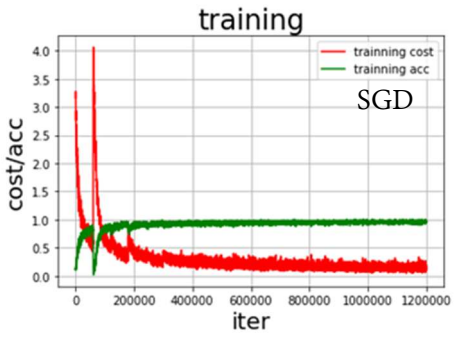


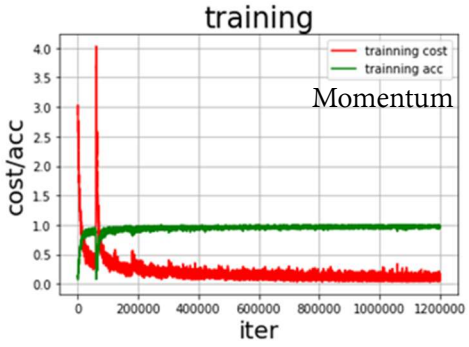


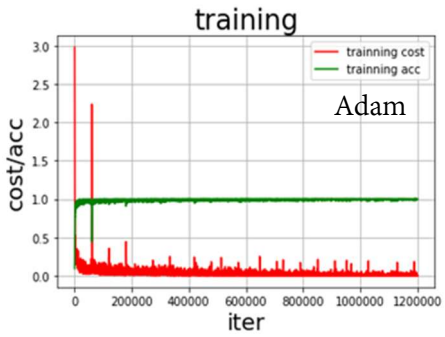
分析：在不改变优化器的前提下，对学习进行细微调整，可以看到 LENET-5 模型在 0.0005~0.002 的范围内，随着学习率的增大，准确率逐渐降低，学习率直接影响模型的收敛状态，学习率选取较小时，学习速度慢，一般在一定轮数过后使用，易使模型过拟合且收敛速度慢，而学习率选取较大时，学习速度快，一般在刚开始训练时使用，但以发生损失值爆炸和振荡的现象。

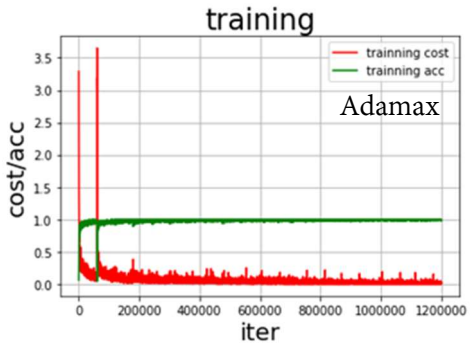
- Lenet-5 与优化器

	模型	损失函数	优化器	学习率	训练回合	准确率
1	LeNet-5	交叉熵损失函数	SGD 算法	0.001	10	0.96233
2	LeNet-5	交叉熵损失函数	Momentum 算法	0.001	10	0.97442
3	LeNet-5	交叉熵损失函数	Adam 算法	0.001	10	**0.99826**
4	LeNet-5	交叉熵损失函数	Adamax 算法	0.001	10	0.99801









分析：随机梯度下降(SGD)是最经典的方法，其他的优化器，都是在这个基础上修改完善得来的，Momentum 优化器在 SGD 基础上引入动量，减少了随机梯度下降过程中存在的噪声问题，而 Adam 综合了 Momentum 的更新方向策略和 RMProp 的计算衰减系数策略，是实际应用中最为常用的一种优化方法。通过对比结果，可以发现，上述方法在其他条件相同的情况下，模型的准确度逐渐有所提高。

6. 总结与收获

本次实验以 MNIST 手写数字作为数据集，在对数据集进行白底黑字的变化后与原数据集一起组成新的数据集用于模型的训练和测试。模型方面，以模型的复杂度为基线，从简单的多层感知机开始，逐步采用 Lenet-5 和层数更深的 VGG，发现保留输入数据二维的特征信息，模型的效果更好，但更深的网络并不一定能持续提高模型的准确度，因而需要更加复杂的模型，例如 ResNet，提高模型的收敛速度和准确率。最后，以 Lenet-5 为基础模型，探究了学习率、优化器、网络层数等参数对模型训练效果的影响，并给出简单分析。

PaddlePaddle 提供的深度学习框架和各种接口，对初学者来说很容易上手，可以根据文档给出的案例在自己的项目中进行编码、调参。本次项目给我带来的收获一方面是通过编码实现神经网络的配置，对神经网络的模型有了更深的印象和理解，另一方面是对于几种经典的卷积神经网络的原理和其优劣有了更进一步的理解。

在进行实验的过程中，主要对经典神经网络的发展和其提出的背景进行了了解，并通过对 Lenet-5、VGG、ResNet 等网络的实现来帮助自己掌握网络的结构，在对模型进行训练的过程中，也感受到了不同模型的优劣。除此之外，通过对网络层数、学习率和优化器的调整对比，既在查阅资料的过程中认识到不同参数对模型训练效果的影响，也在实际的操作中明显感受到了这些问题，更加激发更进一步学习的兴趣。