

**《机器学习实验》**

**课程设计报告**

**报告题目：基于卷积神经网络的手写数字识别**

**学院名称：**

**年 级：**

**授课教师：**

**姓 名：**

**学 号：**

**二〇二三年六月**

基于卷积神经网络的手写数字识别

**摘 要**： 基于卷积神经网络的手写数字识别，对比不同学习率以及不同优化器下各种模型的测试结果，并与其他方法的测试结果进行对比。

**关键词**：CNN，SVM，决策树，手写数字识别

1. **相关工作**

手写数字识别是人工智能领域中一个经典的问题，目前已经有了很多成熟的识别方法，下面简单介绍一下手写数字识别的研究现状。

一、研究现状

在手写数字识别领域，目前已经有很多优秀的研究成果。其中LeCun等人在1989年提出的LeNet-5模型是手写数字识别的经典模型，它首次将卷积神经网络用于数字识别领域。后来又有AlexNet、VGG、GoogLeNet、ResNet等强大的深度学习模型被用于手写数字识别。

在数据集方面，最常用的是MNIST数据集，该数据集包含60000个训练样本和10000个测试样本。此外，还有EMNIST、SVHN、USPS等其他数据集，这些数据集也被广泛用于手写数字识别的研究中。

近年来，人工智能技术不断发展，同时手写数字识别也被应用于多个领域，如自动签名识别、邮政编码识别等。随着技术的不断深入，手写数字识别的性能也将得到进一步提升。

二、识别方法分类

基础分类方法：基础分类方法包括传统的统计学习方法，如KNN、SVM、朴素贝叶斯等。这些方法表现稳定，易于实现，但需要手动提取特征。

深度学习方法：近年来，深度学习方法在手写数字识别领域表现出色。常用的深度学习模型包括LeNet、AlexNet、VGG、GoogLeNet、ResNet等。这些模型能够自动学习特征并进行分类。

卷积神经网络（CNN）方法：卷积神经网络是深度学习中的一个重要分支，应用广泛。对于手写数字识别这一问题，CNN具有优秀的性能。常用的CNN模型包括LeNet、AlexNet、VGG、GoogLeNet、ResNet等。

循环神经网络（RNN）方法：循环神经网络是另一种深度学习模型，在手写数字识别领域也有广泛应用。RNN可以自动处理序列数据。常用的RNN模型包括LSTM、GRU等。

* 1. **基于卷积神经网络的方法**

基于卷积神经网络的手写数字识别是通过一种有效的深度学习算法来实现的，该算法基于多个卷积和池化层来处理手写数字图像，并最终预测图像中表示的数字。

下面是基于卷积神经网络的手写数字识别的基本步骤：

1. 数据准备：收集、清洗、缩放和标准化图像数据，使其适合于卷积神经网络进行处理。

2. 卷积层：将输入的图像分解成数字处理单元或“特征映射”，然后使用多个滤波器集合进行卷积操作，从而提取每个特征映射的特征。

3. 池化层：通过缩小卷积结果的尺寸来减少后续层中的计算负担和参数量，同时保留有用的特征信息。

4. 扁平化层：将池化输出的特征图像铺平成一维向量，然后连接到全连接层。

5. 全连接层：使用神经元集合进行分类任务，从输入层到输出层依次计算每个神经元的得分来确定图像中的数字。

6. Softmax层：将每个节点的得分转换为表示不同类别概率的概率分布。

6. 调参：将训练样本使用留出法划分出验证集，根据验证集调整超参数。

7. 训练：使用反向传播算法在网络中更新权重，以提高网络的准确性。

8. 测试：使用测试集对模型进行验证，并统计模型的预测准确率。

下面是基于卷积神经网络的手写数字识别的优缺点：

优点：

1. 卷积神经网络可以自动提取特征，不需要手动进行特征工程。

2. 卷积神经网络具有较高的识别准确率。

3. 卷积神经网络可以处理不同尺寸的输入图像。

缺点：

1. 训练卷积神经网络需要大量的数据和计算资源。

2. 卷积神经网络可能出现过拟合问题。

3. 卷积神经网络在处理小的或含有冗余信息的图像时可能会出现困难。

**1.2 基于决策树的方法**

好的，决策树由一系列节点和边组成，每个节点代表一个特征或者属性，每个边代表某个属性的值。通过这些节点和边进行决策或分类。在手写数字识别中，决策树的实现过程步骤如下：

1. 数据准备：与前面介绍的类似，准备数据集，并对数据进行预处理。

2. 特征选择：选择能够明显区分不同数字的特征。这里我采用的是Pixel Values（像素值）作为特征。

3. 调参：将训练样本使用留出法划分出验证集，根据验证集调整超参数。

4. 训练：通过对训练样本的学习，生成决策树模型，模型可以根据特征对数字进行分类。

5. 测试：使用测试集对模型进行验证，并统计模型的预测准确率。

决策树的优缺点如下：

优点：

1. 决策树易于理解和解释，对于小型数据集的分析很有用。

2. 决策树可以同时处理离散型和连续型数据。

3. 决策树不需要对数据进行缩放。

4. 可以处理包含缺失数据的数据集。

缺点：

1. 决策树容易产生过度拟合的问题。

2. 决策树对于那些各类别之间的样本数量差异比较大的数据,分类效果不够好。

3. 决策树算法是一种贪婪算法，它对于全局最优解的搜索有很大局限性，这有可能导致不是全局最优的模型结果。

4. 决策树处理连续变量时会带来一些不便。

**1.3 基于支持向量机的方法**

基于支持向量机的手写体数字识别即使用支持向量机对手写数字图像进行分类。常用的支持向量机包括线性支持向量机、非线性支持向量机和核函数支持向量机。本实验采用线性支持向量机的方法，以下是实现该方法的基本步骤：

1. 数据准备：与前面介绍的方法类似，准备数据集，并对数据进行预处理。

2. 特征提取与降维：减少因特征过多而导致的维度灾难，可以采用主成分分析（PCA）等方法降低数据特征维度。

3. 调参：将训练样本使用留出法划分出验证集，根据验证集调整超参数。

4. 训练：利用样本数据训练支持向量机模型。

5. 测试：采用测试样本验证模型并计算精度。

下面是基于支持向量机的方法的优缺点：

优点：

1. 支持向量机具有较高的识别准确率。

2. 在处理高维数据时，支持向量机的表现尤为突出。而且支持向量机在处理非线性问题时具有很好的表现。

3. 支持向量机对于数据集的大小和维数均能够适应。

4. 支持向量机能够处理带有噪声等复杂情况的数据集。

缺点：

1. 支持向量机的计算复杂度高，需要大量的计算资源。

2. 对于数据量比较大的情况，支持向量机的训练时间可能比较长，不太适合处理大数据集。

3. 支持向量机在参数设置上相对复杂，在训练模型时会消耗较多时间。

4. SVM算法的学习任务一般转化为凸二次规划（Quadratic Programming，即QP）问题，该问题的求解需要消耗大量内存。

**2. 基础内容**

**2.1 卷积神经网络**

**2.1.1 卷积层**

torch.nn.Conv2d(in\_channels=1,

out\_channels=16,

kernel\_size=3,

stride=1,

padding=1),

#The size of the picture is 14x14

torch.nn.Conv2d(in\_channels=16,

out\_channels=32,

kernel\_size=3,

stride=1,

padding=1),

**2.1.2 激活层**

torch.nn.ReLU()

torch.nn.ReLU` 是一个常用的激活函数，ReLU 是 Rectified Linear Unit 的缩写，它是一个在深度学习领域广泛使用的非线性激活函数。ReLU 函数在输入为正数时返回该值本身，输入为负数时返回0，数学表达式为 f(x) = max(0, x)。

在神经网络中，ReLU 激活函数的主要作用是增加模型的非线性表现能力，通过对输入数据的非线性变换，可以提高模型在特征提取方面的效果。同时，ReLU 激活函数的导数为常数，计算简单，可以加速神经网络的训练过程。

在 PyTorch 中，可以通过以下方式定义和使用 ReLU 激活函数：

relu = torch.nn.ReLU()

output = relu(inputs)

其中，`inputs`表示输入的数据，`output` 表示通过 ReLU 激活函数后的输出值。

通常情况下，ReLU 激活函数可以直接被添加到神经网络的层中，用于增加神经网络的非线性表现能力。需要注意的是，在某些情况下，ReLU 激活函数可能会遇到梯度消失问题，在这种情况下可以尝试使用其他类型的激活函数来解决。

**2.1.3 损失函数**

lossF = torch.nn.CrossEntropyLoss()

`torch.nn.CrossEntropyLoss()`是一个常用的损失函数，通常用于多分类任务中。它将softmax函数和负对数似然损失结合在一起，可以用于将模型的输出和实际标签之间的差异最小化。

该函数用于评价分类问题模型的输出结果和标签之间的差距，同时也会执行softmax函数计算以将输出转换为概率分布。在使用CrossEntropyLoss时，模型的输出需要是logits，而不是softmax概率。

在PyTorch中，通过以下方式来定义和使用该损失函数：

lossF = torch.nn.CrossEntropyLoss()

loss = lossF(outputs, labels)

其中，`outputs`是模型的输出，`labels`是真实标签。

CrossEntropyLoss的优点是能够很好地处理多分类问题并且能够很好地防止过拟合，应用广泛。但是该函数也有一些缺点，例如对分类不平衡的问题处理较为困难，需要调整类别权重或使用其他替代方法来解决。

**2.1.4 完整模型结构**

class CNN(torch.nn.Module):

def \_\_init\_\_(self):

super(CNN, self).\_\_init\_\_()

self.model = torch.nn.Sequential(

#The size of the picture is 28x28

torch.nn.Conv2d(in\_channels=1,

out\_channels=16,

kernel\_size=3,

stride=1,

padding=1),

torch.nn.ReLU(),

torch.nn.MaxPool2d(kernel\_size=2, stride=2),

#The size of the picture is 14x14

torch.nn.Conv2d(in\_channels=16,

out\_channels=32,

kernel\_size=3,

stride=1,

padding=1),

torch.nn.ReLU(),

torch.nn.MaxPool2d(kernel\_size=2, stride=2),

#The size of the picture is 7x7

torch.nn.Conv2d(in\_channels=32,

out\_channels=64,

kernel\_size=3,

stride=1,

padding=1),

torch.nn.ReLU(),

torch.nn.Flatten(),

torch.nn.Linear(in\_features=7 \* 7 \* 64, out\_features=128),

torch.nn.ReLU(),

torch.nn.Linear(in\_features=128, out\_features=10),

torch.nn.Softmax(dim=1))

#前向传播

def forward(self, input):

output = self.model(input)

return output

**2.2 学习率**

4种不同的学习率：0.001，0.01，0.05，0.5

**2.3 优化器**

8种不同的优化器:

optimizer = torch.optim.ASGD(net.parameters(), lr=learning\_rate)

optimizer = torch.optim.SGD(net.parameters(), lr=learning\_rate)

optimizer = torch.optim.Adagrad(net.parameters(), lr=learning\_rate)

optimizer = torch.optim.Adadelta(net.parameters(), lr=learning\_rate)

optimizer = torch.optim.RMSprop(net.parameters(), lr=learning\_rate)

optimizer = torch.optim.Adamax(net.parameters(), lr=learning\_rate)

optimizer = torch.optim.Adam(net.parameters(), lr=learning\_rate)

optimizer = torch.optim.Rprop(net.parameters(), lr=learning\_rate)

以下介绍上述8种常用的神经网络优化器。

1. ASGD（平均随机梯度下降）：ASGD与SGD类似，不同之处在于它在每一步迭代中使用了对之前梯度的平均值作为指导，而不是最新的梯度，这可能会带来更稳定的收敛性能。

2. SGD（随机梯度下降）：随机梯度下降是一种基本的优化器，它通过计算每个样本对模型参数的梯度来更新模型参数，从而逐步优化目标函数。

3. Adagrad：Adagrad算法是一种自适应学习率的方法，不同的参数通过不同的学习率来进行更新，学习率会随时间逐渐降低。它会记录之前梯度的平方和，将其作为分母来对当前步骤的梯度进行归一化，这样可以使得对于不同维度的参数，学习率的变动幅度不同。

4. Adadelta：Adadelta是基于Adagrad算法的一个改进版本，在计算梯度的平方时加入了衰减参数，用来控制过往梯度的影响。与Adagrad相比，Adadelta通常会对数据中的异常值更健壮。

5. RMSprop：RMSprop算法也是一种自适应学习率的方法，它通过对梯度平方的移动平均值来计算学习率。RMSprop相对于Adagrad的改进在于，它对梯度平方的求和进行了指数衰减，这样可以让它适应变化的梯度。

6. Adamax：Adamax 是 Adam 优化器的一种变体，在Adam中使用了衰减率的上界，而在Adamax中使用了无限范数的概念，这样可以使得学习更加稳定。

7. Adam：Adam算法是一种常用的基于自适应估计的梯度下降算法。它使用了两个动量项分别估计梯度的一阶矩和二阶矩，从而进行自适应学习率的更新，从而在优化过程中自适应调整学习率。

8. Rprop：Rprop算法是一种基于反向传播算法的优化方法，它使用了自适应的步长大小来确定下一步的梯度下降方向，可以有效地避免神经网络训练中的梯度消失问题。

这些优化器在不同情况下拥有不同的优缺点，需要根据实际问题选择合适的优化器来优化神经网络的训练过程。

**2.4 SVM**

线性SVM: svc = LinearSVC(dual=False)

**2.5 决策树**

默认参数的决策树: clf = tree.DecisionTreeClassifier()

**2.6 评估指标(本文以下评价指标均采用micro类型)**

**2.6.1** **查准率(precision)**

查准率（precision）是指分类器正确预测为正例的样本数量（TP）占所有预测为正例的样本数量（TP+FP）的比例。通俗的讲，它是针对分类器预测结果中“正例”的精确性的度量。计算公式为：

precision = TP / (TP + FP)

其中，TP表示真正例的数量，FP表示假正例的数量。

一个模型的查准率越高，说明分类器预测为正例的样本中真正的正例占比越高，模型的性能越好。但是，在不同的场景下，查准率的重要性是不同的，有些场景更关注样本中真实正例被正确预测出来的可能性，有些场景更关注错误分类的负例，因此需要结合其他指标来综合分析模型性能。

**2.6.2** **查全率(recall)**

查全率（recall），又称为召回率，是指分类器正确预测为正例的样本数量（TP）占所有正例的样本数量（TP+FN）的比例。通俗的讲，它是针对分类器预测结果中样本中“正例”的覆盖面的度量。计算公式为：

recall = TP / (TP + FN)

其中，TP表示真正例的数量，FN表示假反例的数量。

一个模型的查全率越高，说明模型能够更好地覆盖真实正例，即更多的真实正例被正确预测出来，模型的性能越好。但是，查全率与查准率具有一定的权衡关系，无法同时让查全率和查准率都达到最高水平，具体看场景需要选择不同的模型，或者采取一些结合了查准率和查全率的综合指标，比如F1 Score等。

**2.6.3 F1值**

F1值是评价分类器性能的一个常用指标，是查准率和查全率的调和平均数。F1值的取值范围为0至1之间，F1值越高，分类器性能越好。

计算F1值的公式为：

F1 = 2 \* ((precision \* recall) / (precision + recall))

其中，precision为查准率，recall为查全率。

F1值是查准率和查全率的综合指标，优势在于同时考虑了查准率和查全率的效果。当分类器在查准率和查全率两方面都取得很好的表现时，F1值也会比较高。因此，F1值经常被用来评估分类器性能，特别是在分类器需要高精确率和高覆盖率的场合。同时，F1值对于数据集分布不均匀问题也很适用。

**2.6.4 准确率（accuracy）**

准确率（accuracy）是指分类器正确分类的样本数占总样本数的比例。通俗的解释，就是分类器正确分类的程度，它是一个分类器优劣性能的重要指标之一。

计算准确率的公式为：

准确率 = (TP + TN) / (TP + TN + FP + FN)

其中，TP表示真正例的数量，TN表示真反例的数量，FP表示假正例的数量，FN表示假反例的数量。

**3. 运行步骤**

**3.1 数据清洗**

使用data目录下**数据集类型转换.py**将gz文件转换成csv文件。

**3.2 参数调整**

运行model\_adjust目录下的各个文件进行相应模型超参数的调整。

**3.3 模型训练**

运行model\_train目录下的各个文件进行相应模型训练，并保持模型训练结构。

**3.4 模型测试**

运行model\_test目录下的各个文件进行相应模型测试，并保存测试结果。

**4. 实验**

**4.1 数据集**

MINIST手写数字识别数据集

**4.2 实验环境**

硬件环境

1. 计算机：LAPTOP-K5GRC2HU
2. CPU: 11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz 2.69 GHz
3. RAM：16GB

软件环境

1. 操作系统：Windows10家庭中文版
2. 开发工具：Visual Studio Code

**4.3 实验结果**

表1 不同学习率Adam优化器对比

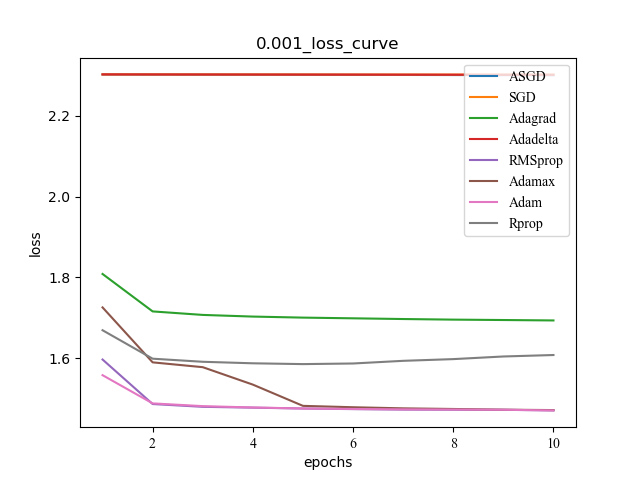
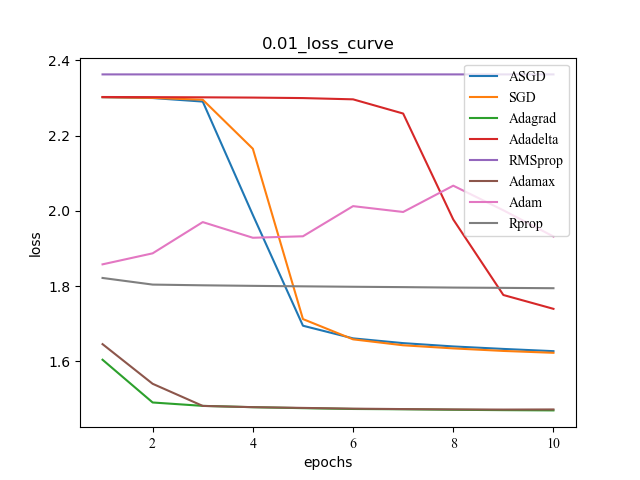
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 学习率 | Precision | Recall | F1 | Accuracy |
| 0.001 | 0.99 | 0.99 | 0.99 | 0.99 |
| 0.01 | 0.37 | 0.53 | 0.42 | 0.54 |
| 0.05 | 0.01 | 0.10 | 0.02 | 0.11 |
| 0.1 | 0.01 | 0.10 | 0.02 | 0.10 |

表2 学习率0.001时不通优化器对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 优化器 | Precision | Recall | F1 | Accuracy |
| ASGD | 0.13 | 0.21 | 0.11 | 0.23 |
| SGD | 0.06 | 0.13 | 0.06 | 0.14 |
| Adagrad | 0.63 | 0.76 | 0.68 | 0.77 |
| Adadelta | 0.05 | 0.11 | 0.03 | 0.11 |
| RMSprop | 0.99 | 0.99 | 0.99 | 0.99 |
| Adamax | 0.99 | 0.99 | 0.99 | 0.99 |
| Adam | 0.99 | 0.99 | 0.99 | 0.99 |
| Rprop | 0.87 | 0.86 | 0.85 | 0.86 |

表3 不同方法对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 方法 | Precision | Recall | F1 | Accuracy |
| SVM | 0.92 | 0.92 | 0.92 | 0.92 |
| 决策树 | 0.88 | 0.88 | 0.88 | 0.88 |
| 本文方法 | 0.99 | 0.99 | 0.99 | 0.99 |

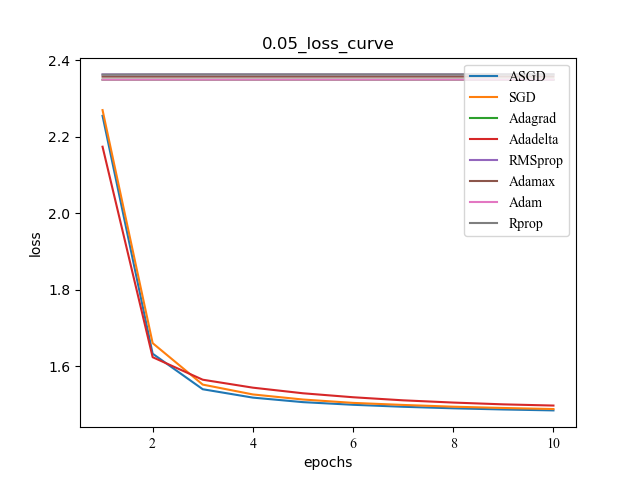
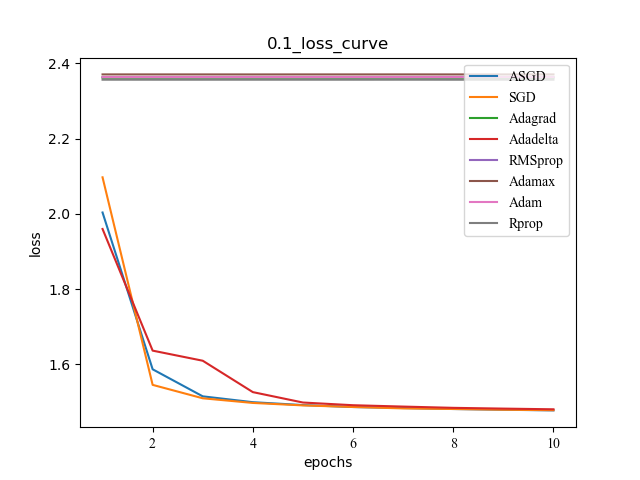
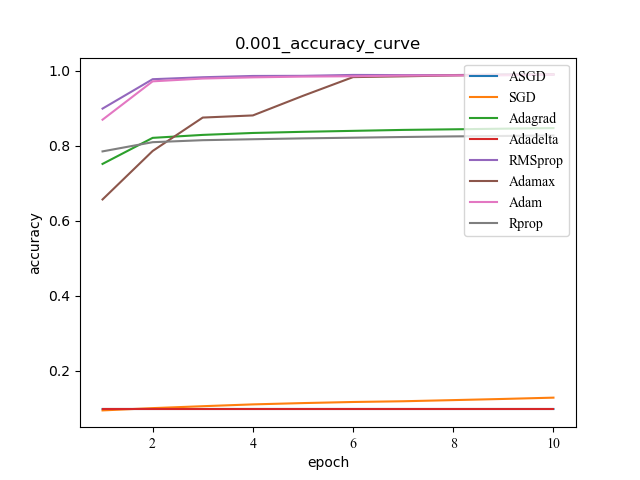
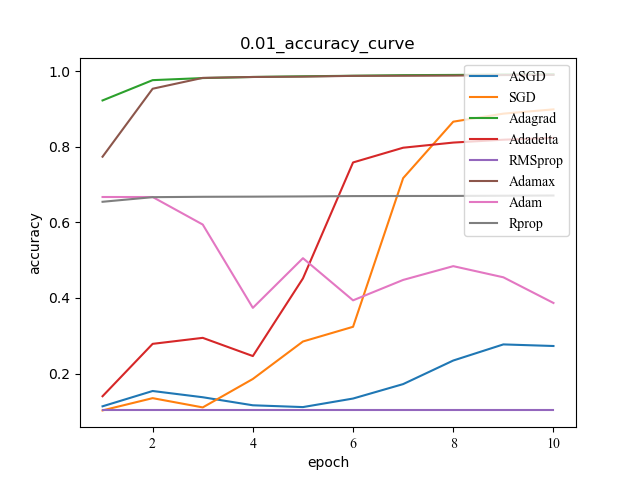
 

图1 CNN不同学习率下各种优化器的损失变化曲线对比

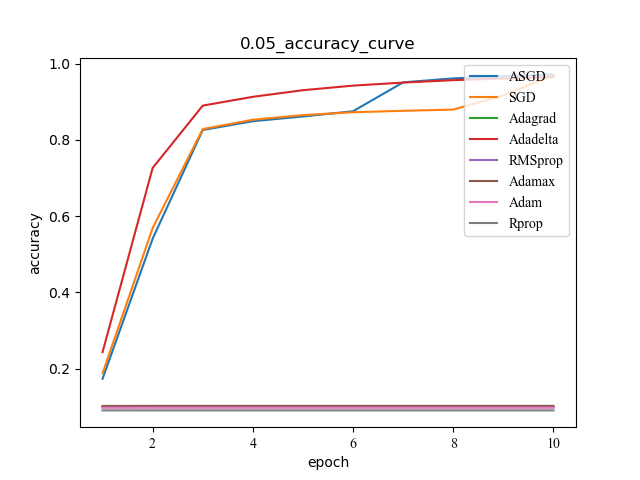
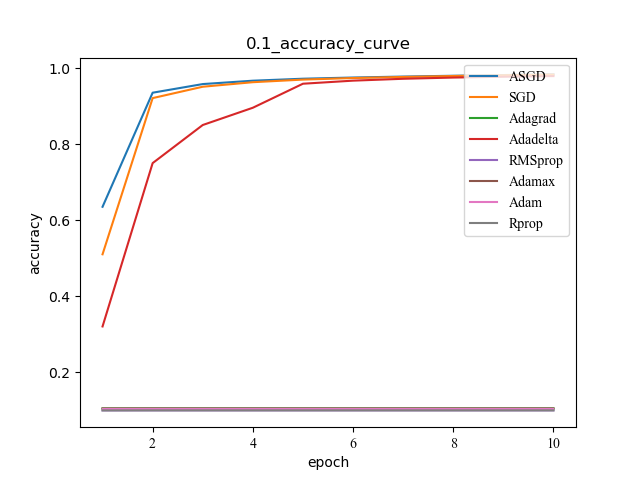
 

图2 CNN不同学习率下各种优化器的准确率变化曲线对比

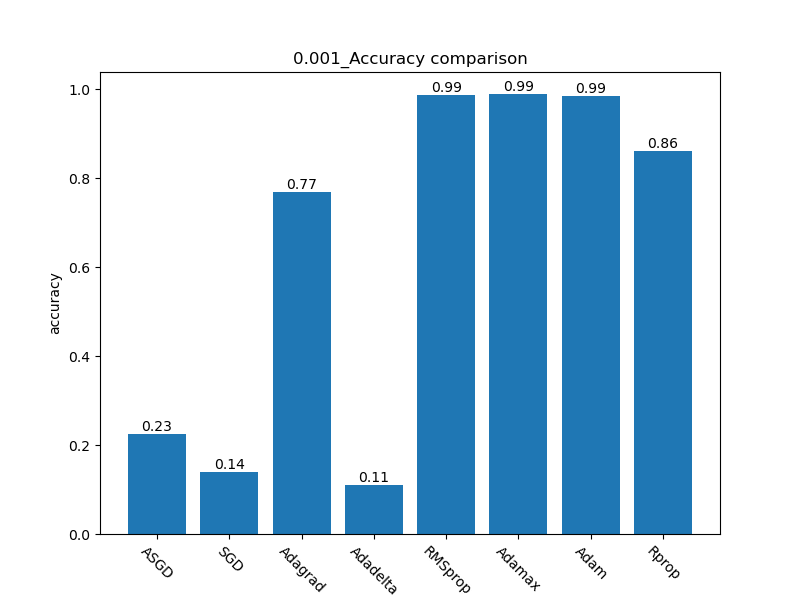


图3 学习率为0.001时CNN各个优化器测试准确率对比图

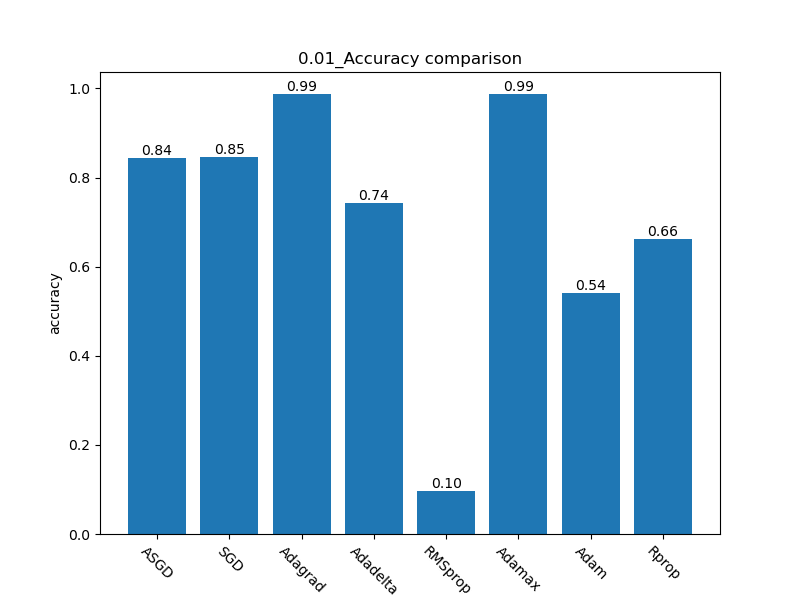


图4 学习率为0.01CNN各种优化器测试准确率对比图

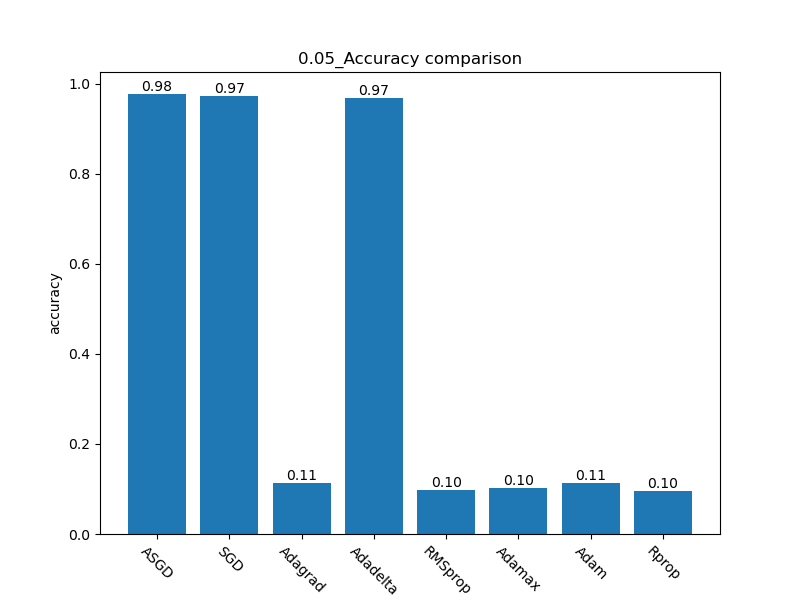


图5：学习率为0.05 CNN各种优化器测试准确率对比图

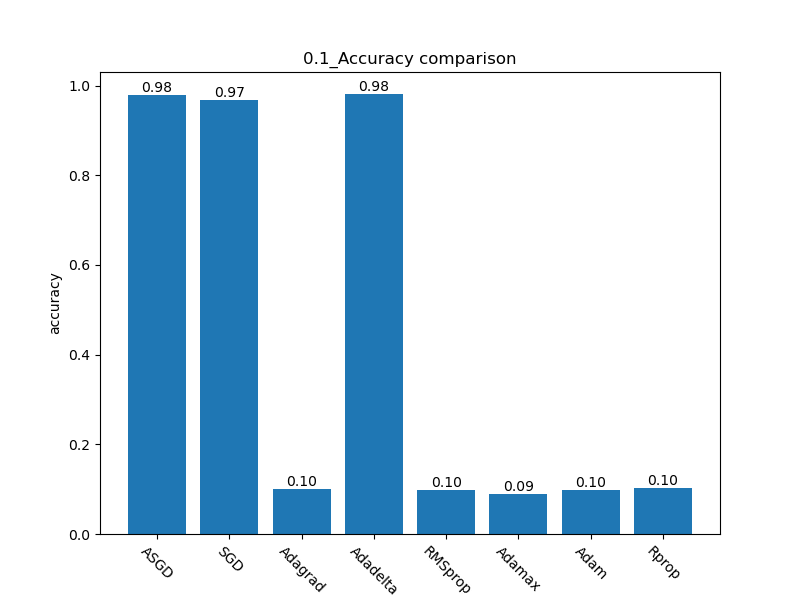


图6：学习率为0.1 CNN各种优化器测试准确率对比图

**5. 总结展望**

总结：

本文使用Pytorch实现了卷积神经网络，并对比了四种学习率下不同优化器对训练模型后测试的结果。同时将学习率为0.001优化器为Adam的CNN与调整好参数的SVM和决策树测试结果进行了对比。以下是对比得出的结论：

1. 由表1可得Adam优化器在0.001，0.01，0.05，0.1四种学习率测试结果种，0.001的表现效果是最好的。
2. 由表2 当学习率为0.001时Adam, Adamax, RMSprop三种优化器训练后的CNN模型表现效果较好。
3. 由图1~图6可以看出在不同学习率下，表现较好的模型往往是不同优化器，在不同学习率下，使用同一优化器训练模型，模型并不一定总是收敛。且在不同学习率下，均有表现较优的模型。
4. 由表3 可以得出在均调整模型超参数至最优的情况下，神经网络的识别效果在本实验达到了最优。

展望：

本文的CNN网络架构较为简单，不能较好的体现CNN的完整能力，后续将采用更为复杂的卷积网络结构完整更为复杂识别任务的测试。且对比方法是采用SVM和决策树，后续进行复杂识别任务的对比测试时，将会采用RNN，GNN，集成学习等算法进行对比测试等待。

参考文献

1. Jabde, Meenal, et al. "Comparative Study of Machine Learning and Deep Learning Classifiers on Handwritten Numeral Recognition." International Symposium on Intelligent Informatics: Proceedings of ISI 2022. Singapore: Springer Nature Singapore, 2023.

**附录**

**数据集转换代码：**

def convert(imgf, labelf, outf, n):

    '''

    这段代码的主要目的是将MNIST数据集中的二进制文件转换为CSV格式，以便在机器学习算法中使用。以下是代码的逐行分析。

    python

    f = open(imgf, "rb")

    o = open(outf, "w")

    l = open(labelf, "rb")

    从给定的文件名打开三个文件，包括一个用于训练图像数据的文件（imgf）、一个输出CSV数据到的文件（outf）和一个用于标签的文件（labelf）。

    python

    f.read(16)

    l.read(8)

    跳过前16个字节（file magic number, content count, row count, and column count）的图片信息文件和前8个字节的标签信息文件，因为它们不包含有用的数据。

    python

    images = []

    for i in range(n):

        image = [ord(l.read(1))]

        for j in range(28 \* 28):

            image.append(ord(f.read(1)))

        images.append(image)

    针对给定数量n读取标签和图像数据。在循环内部，对于每张图片，所有的28\*28像素数据都会被追加到由标签和像素值组成的一维列表中，并存入images数组。

    说明：ord() 函数返回字符的整数表示，例如，ord('a') 将返回97 （'a' 对应的ASCII码）。

    python

    for image in images:

        o.write(",".join(str(pix) for pix in image) + "\n")

    将image数组中的每个子数组都转换成一个CSV行，并使用逗号将其元素分隔开，然后将该行写入到输出文件中。

    python

    f.close()

    o.close()

    l.close()

    最后记得关闭三个已打开的文件。

    '''

    f = open(imgf, "rb")

    o = open(outf, "w")

    l = open(labelf, "rb")

    f.read(16)

    l.read(8)

    images = []

    for i in range(n):

        image = [ord(l.read(1))]

        for j in range(28 \* 28):

            image.append(ord(f.read(1)))

        images.append(image)

    # o.write("label," + ",".join("pixel{}".format(i) for i in range(28 \* 28)) + "\n")

    #该命令会在读取的csv文件第一行加上标签

    for image in images:

        o.write(",".join(str(pix) for pix in image) + "\n")

    f.close()

    o.close()

    l.close()

# convert("./data/MNIST/train-images.idx3-ubyte",

#         "./data/MNIST/train-labels.idx1-ubyte", "mnist\_train.csv", 60000)

# convert("./data/MNIST/t10k-images.idx3-ubyte",

#         "./data/MNIST/t10k-labels.idx1-ubyte", "mnist\_test.csv", 10000)

convert("data//MNIST//raw//train-images-idx3-ubyte",

        "data//MNIST//raw//train-labels-idx1-ubyte",

        "data//MNIST//raw//mnist\_train.csv", 60000)

convert("data//MNIST//raw//t10k-images-idx3-ubyte",

        "data//MNIST//raw//t10k-labels-idx1-ubyte",

        "data//MNIST//raw//mnist\_test.csv", 10000)

print("Convert Finished!")

**超参数调整代码：**

**CNN：**

import torch

import torchvision

from tqdm import tqdm

import matplotlib.pyplot as plt

import pandas as pd

import torch

from torch.utils.data import Dataset

import numpy as np

import random

class CNN(torch.nn.Module):

    def \_\_init\_\_(self):

        super(CNN, self).\_\_init\_\_()

        self.model = torch.nn.Sequential(

            #The size of the picture is 28x28

            torch.nn.Conv2d(in\_channels=1,

                            out\_channels=16,

                            kernel\_size=3,

                            stride=1,

                            padding=1),

            torch.nn.ReLU(),

            torch.nn.MaxPool2d(kernel\_size=2, stride=2),

            #The size of the picture is 14x14

            torch.nn.Conv2d(in\_channels=16,

                            out\_channels=32,

                            kernel\_size=3,

                            stride=1,

                            padding=1),

            torch.nn.ReLU(),

            torch.nn.MaxPool2d(kernel\_size=2, stride=2),

            #The size of the picture is 7x7

            torch.nn.Conv2d(in\_channels=32,

                            out\_channels=64,

                            kernel\_size=3,

                            stride=1,

                            padding=1),

            torch.nn.ReLU(),

            torch.nn.Flatten(),

            torch.nn.Linear(in\_features=7 \* 7 \* 64, out\_features=128),

            torch.nn.ReLU(),

            torch.nn.Linear(in\_features=128, out\_features=10),

            torch.nn.Softmax(dim=1))

    #前向传播

    def forward(self, input):

        output = self.model(input)

        return output

class MyDataset(Dataset):

    def \_\_init\_\_(self, data):

        self.data = np.array(data)

    def \_\_len\_\_(self):

        return len(self.data)

    def \_\_getitem\_\_(self, idx):

        label = self.data[idx, 0]  # 第一列为标签

        data = self.data[idx, 1:]  # 从第2列开始作为数据

        data = data.reshape((1, 28, 28))

        return data / 255.0, int(label)

#读取数据函数，返回属性和标签

def read\_data(filename):

    data = pd.read\_csv(filename, header=None).values.astype('float32')

    return data

def split\_dataset(dataset, ratio, random\_state=42):

    dataset\_copy = np.array(dataset).copy()

    train\_size = int(len(dataset\_copy) \* ratio)

    train\_set = []

    valid\_set = list(dataset\_copy)

    np.random.seed(random\_state)

    while len(train\_set) < train\_size:

        index = random.randrange(len(valid\_set))

        train\_set.append(valid\_set.pop(index))

    return np.array(train\_set), np.array(valid\_set)

if \_\_name\_\_ == '\_\_main\_\_':

    device = "cuda:0" if torch.cuda.is\_available() else "cpu"

    BATCH\_SIZE = 64

    EPOCHS = 10

    learning\_rate = 0.001  #0.001，0.01，0.1

    # trainData = MyDataset('data//MNIST//raw//mnist\_train.csv')

    # testData = MyDataset('data//MNIST//raw//mnist\_test.csv')

    data = read\_data('data//MNIST//raw//mnist\_train.csv')

    trainData, validData = split\_dataset(data, 0.8)

    trainData = MyDataset(trainData)

    validData = MyDataset(validData)

    trainDataLoader = torch.utils.data.DataLoader(dataset=trainData,

                                                  batch\_size=BATCH\_SIZE,

                                                  shuffle=True)

    testDataLoader = torch.utils.data.DataLoader(dataset=validData,

                                                 batch\_size=BATCH\_SIZE)

    net = CNN()

    print(net.to(device))

    lossF = torch.nn.CrossEntropyLoss()

    optimizer = torch.optim.ASGD(net.parameters(), lr=learning\_rate)

    optimizer = torch.optim.SGD(net.parameters(), lr=learning\_rate)

    optimizer = torch.optim.Adagrad(net.parameters(), lr=learning\_rate)

    optimizer = torch.optim.Adadelta(net.parameters(), lr=learning\_rate)

    optimizer = torch.optim.RMSprop(net.parameters(), lr=learning\_rate)

    optimizer = torch.optim.Adamax(net.parameters(), lr=learning\_rate)

    optimizer = torch.optim.Adam(net.parameters(), lr=learning\_rate)

    optimizer = torch.optim.Rprop(net.parameters(), lr=learning\_rate)

    history = {'Test Loss': [], 'Test Accuracy': []}

    for epoch in range(1, EPOCHS + 1):

        processBar = tqdm(trainDataLoader, unit='step')

        net.train(True)

        for step, (trainImgs, labels) in enumerate(processBar):

            print(trainImgs[trainImgs > 1])

            trainImgs = trainImgs.to(device)

            labels = labels.to(device)

            net.zero\_grad()

            outputs = net(trainImgs)

            loss = lossF(outputs, labels)

            predictions = torch.argmax(outputs, dim=1)

            accuracy = torch.sum(predictions == labels) / labels.shape[0]

            loss.backward()

            optimizer.step()

            processBar.set\_description(

                "[%d/%d] Loss: %.4f, Acc: %.4f" %

                (epoch, EPOCHS, loss.item(), accuracy.item()))

            if step == len(processBar) - 1:

                correct, totalLoss = 0, 0

                net.train(False)

                with torch.no\_grad():

                    for testImgs, labels in testDataLoader:

                        testImgs = testImgs.to(device)

                        labels = labels.to(device)

                        outputs = net(testImgs)

                        loss = lossF(outputs, labels)

                        predictions = torch.argmax(outputs, dim=1)

                        totalLoss += loss

                        correct += torch.sum(predictions == labels)

                    testAccuracy = correct / (BATCH\_SIZE \* len(testDataLoader))

                    testLoss = totalLoss / len(testDataLoader)

                    history['Test Loss'].append(testLoss.item())

                    history['Test Accuracy'].append(testAccuracy.item())

                processBar.set\_description(

                    "[%d/%d] Loss: %.4f, Acc: %.4f, Test Loss: %.4f, Test Acc: %.4f"

                    % (epoch, EPOCHS, loss.item(), accuracy.item(),

                       testLoss.item(), testAccuracy.item()))

        processBar.close()

**SVM：**

from sklearn import tree

import numpy as np

import csv

import matplotlib.pyplot as plt

import joblib

import pandas as pd

import numpy as np

import random

#读取数据函数，返回属性和标签

def read\_data(filename):

    data = pd.read\_csv(filename, header=None).values.astype(float)

    return data

def split\_dataset(dataset, ratio, random\_state=42):

    dataset\_copy = np.array(dataset).copy()

    train\_size = int(len(dataset\_copy) \* ratio)

    train\_set = []

    valid\_set = list(dataset\_copy)

    np.random.seed(random\_state)

    while len(train\_set) < train\_size:

        index = random.randrange(len(valid\_set))

        train\_set.append(valid\_set.pop(index))

    return np.array(train\_set), np.array(valid\_set)

#读取数据以及数据标准化

data = read\_data('data//MNIST//raw//mnist\_train.csv')

data\_copy = data.copy()

trainData, validData = split\_dataset(data\_copy, 0.8)

train\_data, train\_label = trainData[:, 1:] / 255, trainData[:, 0]

valid\_data, valid\_label = validData[:, 1:] / 255, validData[:, 0]

# clf = tree.DecisionTreeClassifier(criterion="entropy",

#                                   max\_depth=None,

#                                   min\_samples\_split=2,

#                                   min\_samples\_leaf=1,

#                                   max\_features=0.7,

#                                   splitter='best',

#                                   random\_state=42)

clf = tree.DecisionTreeClassifier()

clf.fit(train\_data, train\_label)

score = clf.score(valid\_data, valid\_label)

print('Accuracy of Linear Tree:', score)

#load model

# classifier = joblib.load('save\_model\\classifier.pkl')

# 以文字形式输出树

# text\_representation = tree.export\_text(classifier)

# print(text\_representation)

# # 用图片画出

# plt.figure(figsize=(30, 10), facecolor='g')  #

# a = tree.plot\_tree(classifier, rounded=True, filled=True, fontsize=14)

# plt.show()

**决策树：**

from sklearn import tree

import numpy as np

import csv

import matplotlib.pyplot as plt

import joblib

import pandas as pd

import numpy as np

import random

#读取数据函数，返回属性和标签

def read\_data(filename):

    data = pd.read\_csv(filename, header=None).values.astype(float)

    return data

def split\_dataset(dataset, ratio, random\_state=42):

    dataset\_copy = np.array(dataset).copy()

    train\_size = int(len(dataset\_copy) \* ratio)

    train\_set = []

    valid\_set = list(dataset\_copy)

    np.random.seed(random\_state)

    while len(train\_set) < train\_size:

        index = random.randrange(len(valid\_set))

        train\_set.append(valid\_set.pop(index))

    return np.array(train\_set), np.array(valid\_set)

#读取数据以及数据标准化

data = read\_data('data//MNIST//raw//mnist\_train.csv')

data\_copy = data.copy()

trainData, validData = split\_dataset(data\_copy, 0.8)

train\_data, train\_label = trainData[:, 1:] / 255, trainData[:, 0]

valid\_data, valid\_label = validData[:, 1:] / 255, validData[:, 0]

# clf = tree.DecisionTreeClassifier(criterion="entropy",

#                                   max\_depth=None,

#                                   min\_samples\_split=2,

#                                   min\_samples\_leaf=1,

#                                   max\_features=0.7,

#                                   splitter='best',

#                                   random\_state=42)

clf = tree.DecisionTreeClassifier()

clf.fit(train\_data, train\_label)

score = clf.score(valid\_data, valid\_label)

print('Accuracy of Linear Tree:', score)

#load model

# classifier = joblib.load('save\_model\\classifier.pkl')

# 以文字形式输出树

# text\_representation = tree.export\_text(classifier)

# print(text\_representation)

# # 用图片画出

# plt.figure(figsize=(30, 10), facecolor='g')  #

# a = tree.plot\_tree(classifier, rounded=True, filled=True, fontsize=14)

# plt.show()

**模型训练代码：**

**CNN：**

import torch

import torchvision

from tqdm import tqdm

import matplotlib.pyplot as plt

import pandas as pd

import torch

from torch.utils.data import Dataset

import numpy as np

import random

import sys

import os

sys.path.append('model\_adjust')

from 卷积神经网络调整 import CNN, MyDataset, read\_data

device = "cuda:0" if torch.cuda.is\_available() else "cpu"

BATCH\_SIZE = 64

EPOCHS = 10

data = read\_data('data//MNIST//raw//mnist\_train.csv')

allData = MyDataset(data)

allDataLoader = torch.utils.data.DataLoader(dataset=allData,

                                            batch\_size=BATCH\_SIZE,

                                            shuffle=True)

optimizers = [

    "ASGD", "SGD", "Adagrad", "Adadelta", "RMSprop", "Adamax", "Adam", "Rprop"

]

choices = [1]

for choice in choices:

    lr = [0.001, 0.01, 0.05, 0.1]

    for learning\_rate in lr:

        fig, ax = plt.subplots()

        ax.set\_title(f"model\_loss\_curve")

        ax.set\_xlabel("epochs")

        ax.set\_ylabel("loss per epoch")

        fig, ax = plt.subplots()

        for optimizer\_name in optimizers:

            net = CNN()

            net.to(device)

            print(

                f"optimizer\_name={optimizer\_name}, learning\_rate={learning\_rate}! "

            )

            lossF = torch.nn.CrossEntropyLoss()

            optimizers = {

                "ASGD": torch.optim.ASGD,

                "SGD": torch.optim.SGD,

                "Adagrad": torch.optim.Adagrad,

                "Adadelta": torch.optim.Adadelta,

                "RMSprop": torch.optim.RMSprop,

                "Adamax": torch.optim.Adamax,

                "Adam": torch.optim.Adam,

                "Rprop": torch.optim.Rprop,

            }

            if optimizer\_name not in optimizers:

                raise ValueError(f"Unknown optimizer: {optimizer\_name}")

            optimizer\_class = optimizers[optimizer\_name]

            optimizer = optimizer\_class(net.parameters(), lr=learning\_rate)

            total\_loss = []

            per\_loss = 0

            correct = 0

            accuracy\_list = []

            for epoch in range(1, EPOCHS + 1):

                processBar = tqdm(allDataLoader, unit='step')

                net.train(True)

                for step, (trainImgs, labels) in enumerate(processBar):

                    trainImgs = trainImgs.to(device)

                    labels = labels.to(device)

                    net.zero\_grad()

                    outputs = net(trainImgs)

                    loss = lossF(outputs, labels)

                    predictions = torch.argmax(outputs, dim=1)

                    accuracy = torch.sum(

                        predictions == labels) / labels.shape[0]

                    correct += torch.sum(predictions == labels)

                    per\_loss = per\_loss + loss.item()

                    loss.backward()

                    optimizer.step()

                    processBar.set\_description(

                        "[%d/%d] choice: %d Loss: %.4f, Acc: %.4f" %

                        (epoch, EPOCHS, choice, loss.item(), accuracy.item()))

                per\_loss = per\_loss / len(allDataLoader)

                total\_loss.append(per\_loss)

                per\_loss = 0

                accuracy = correct / (BATCH\_SIZE \* len(allDataLoader))

                accuracy\_list.append(accuracy.item())

                correct = 0

            path = f"save\_model//CNN//{learning\_rate}"

            model\_path = os.makedirs(path, exist\_ok=True)

            model\_path = path

            torch.save(net.state\_dict(),

                       f"{model\_path}//{optimizer\_name}\_cnn.pth")

            index = np.arange(1, EPOCHS + 1, 1)

            if choice:

                plt.plot(index, total\_loss, label=f"{optimizer\_name}")

                font = {

                    'family': 'Times New Roman',

                    'weight': 'normal',

                    'size': 10

                }

                plt.xticks(fontproperties='Times New Roman', fontsize=10)

                plt.xticks(fontproperties='Times New Roman', fontsize=10)

                plt.xlabel(u'epochs')

                plt.ylabel(u'loss')

                plt.legend(loc=1, prop=font)

            else:

                plt.plot(index, accuracy\_list, label=f"{optimizer\_name}")

                font = {

                    'family': 'Times New Roman',

                    'weight': 'normal',

                    'size': 10

                }

                plt.xticks(fontproperties='Times New Roman', fontsize=10)

                plt.xticks(fontproperties='Times New Roman', fontsize=10)

                plt.xlabel(u'epoch')

                plt.ylabel(u'accuracy')

                plt.legend(loc=1, prop=font)

        if choice:

            plt.title(f"{learning\_rate}\_loss\_curve")

            plt.savefig(f"loss\_curve//{learning\_rate}\_loss\_curve.png")

            plt.show(block=False)

        else:

            plt.title(f"{learning\_rate}\_accuracy\_curve")

            plt.savefig(f"accuracy\_curve//{learning\_rate}\_accuracy\_curve.png")

            plt.show(block=False)

**SVM：**

from sklearn.svm import LinearSVC

import csv

import numpy as np

import joblib

import pandas as pd

import numpy as np

import random

def split\_dataset(dataset, ratio, random\_state=42):

    dataset\_copy = np.array(dataset).copy()

    train\_size = int(len(dataset\_copy) \* ratio)

    train\_set = []

    valid\_set = list(dataset\_copy)

    np.random.seed(random\_state)

    while len(train\_set) < train\_size:

        index = random.randrange(len(valid\_set))

        train\_set.append(valid\_set.pop(index))

    return np.array(train\_set), np.array(valid\_set)

#读取数据函数，返回属性和标签

def read\_data(filename):

    data = pd.read\_csv(filename, header=None).values.astype(float)

    return data

#读取数据以及数据标准化

data = read\_data('data//MNIST//raw//mnist\_train.csv')

data, label = data.copy()[:, 1:] / 255, data.copy()[:, 0].reshape((-1, 1))

print("训练开始！")

svc = LinearSVC(dual=False)

svc.fit(data, label.T.ravel())

joblib.dump(svc, 'save\_model\\svc.pkl')

print("训练结束！")

**决策树：**

from sklearn import tree

import numpy as np

import csv

import matplotlib.pyplot as plt

import joblib

import pandas as pd

import numpy as np

import random

#读取数据函数，返回属性和标签

def read\_data(filename):

    data = pd.read\_csv(filename, header=None).values.astype(float)

    return data

def split\_dataset(dataset, ratio, random\_state=42):

    dataset\_copy = np.array(dataset).copy()

    train\_size = int(len(dataset\_copy) \* ratio)

    train\_set = []

    valid\_set = list(dataset\_copy)

    np.random.seed(random\_state)

    while len(train\_set) < train\_size:

        index = random.randrange(len(valid\_set))

        train\_set.append(valid\_set.pop(index))

    return np.array(train\_set), np.array(valid\_set)

#读取数据以及数据标准化

data = read\_data('data//MNIST//raw//mnist\_train.csv')

data\_copy = data.copy()

data, label = data\_copy[:, 1:] / 255, data[:, 0]

print("训练开始！")

clf = tree.DecisionTreeClassifier()

clf.fit(data, label)

joblib.dump(clf, 'save\_model\\tree.pkl')

#load model

# classifier = joblib.load('save\_model\\classifier.pkl')

# 以文字形式输出树

# text\_representation = tree.export\_text(classifier)

# print(text\_representation)

# # 用图片画出

# plt.figure(figsize=(30, 10), facecolor='g')  #

# a = tree.plot\_tree(classifier, rounded=True, filled=True, fontsize=14)

# plt.show()

print("训练结束！")

**模型测试代码：**

**CNN：**

**测试：**

import torch

import torchvision

from tqdm import tqdm

import matplotlib.pyplot as plt

import pandas as pd

import torch

from torch.utils.data import Dataset

import numpy as np

import random

import sys

import os

from sklearn.metrics import classification\_report

sys.path.append('model\_adjust')

from 卷积神经网络调整 import CNN, MyDataset, read\_data

data = read\_data('data//MNIST//raw//mnist\_test.csv')

allData = MyDataset(data)

device = "cuda:0" if torch.cuda.is\_available() else "cpu"

BATCH\_SIZE = len(data)

EPOCHS = 10

allDataLoader = torch.utils.data.DataLoader(dataset=allData,

                                            batch\_size=BATCH\_SIZE,

                                            shuffle=True)

optimizers = [

    "ASGD", "SGD", "Adagrad", "Adadelta", "RMSprop", "Adamax", "Adam", "Rprop"

]

learning\_rate = 0.001

optimizer\_name = "Adam"

net = CNN()

net.to(device)

print(f"optimizer\_name={optimizer\_name}, learning\_rate={learning\_rate}! ")

lossF = torch.nn.CrossEntropyLoss()

optimizers = {

    "ASGD": torch.optim.ASGD,

    "SGD": torch.optim.SGD,

    "Adagrad": torch.optim.Adagrad,

    "Adadelta": torch.optim.Adadelta,

    "RMSprop": torch.optim.RMSprop,

    "Adamax": torch.optim.Adamax,

    "Adam": torch.optim.Adam,

    "Rprop": torch.optim.Rprop,

}

if optimizer\_name not in optimizers:

    raise ValueError(f"Unknown optimizer: {optimizer\_name}")

optimizer\_class = optimizers[optimizer\_name]

optimizer = optimizer\_class(net.parameters(), lr=learning\_rate)

correct = 0

net.train(False)

path = f"save\_model//CNN//{learning\_rate}"

model\_path = os.makedirs(path, exist\_ok=True)

model\_path = path

net.load\_state\_dict(

    torch.load(f"{model\_path}//{optimizer\_name}\_cnn.pth"))  # 同上

correct, totalLoss = 0, 0

net.train(False)

with torch.no\_grad():

    for step, (testImgs, labels) in enumerate(allDataLoader):

        testImgs = testImgs.to(device)

        labels = labels.to(device)

        net.zero\_grad()

        outputs = net(testImgs)

        loss = lossF(outputs, labels)

        predictions = torch.argmax(outputs, dim=1)

        print(

            classification\_report(labels.cpu().numpy(),

                                  predictions.cpu().numpy(),

                                  digits=3))

        accuracy = torch.sum(predictions == labels) / labels.shape[0]

        print(accuracy.item())

**对比测试：**

import torch

import torchvision

from tqdm import tqdm

import matplotlib.pyplot as plt

import pandas as pd

import torch

from torch.utils.data import Dataset

import numpy as np

import random

import sys

import os

from sklearn.metrics import classification\_report

sys.path.append('model\_adjust')

from 卷积神经网络调整 import CNN, MyDataset, read\_data

data = read\_data('data//MNIST//raw//mnist\_test.csv')

device = "cuda:0" if torch.cuda.is\_available() else "cpu"

BATCH\_SIZE = len(data)

EPOCHS = 10

allData = MyDataset(data)

allDataLoader = torch.utils.data.DataLoader(dataset=allData,

                                            batch\_size=BATCH\_SIZE,

                                            shuffle=True)

optimizers = [

    "ASGD", "SGD", "Adagrad", "Adadelta", "RMSprop", "Adamax", "Adam", "Rprop"

]

lr = [0.001, 0.01, 0.05, 0.1]

for learning\_rate in lr:

    accuracy\_list = []

    with open(f"model\_test//CNN//{learning\_rate}\_Eva.txt", "w") as f:

        f.write(f"learning\_rate = {learning\_rate} model evaluate\n")

    for optimizer\_name in optimizers:

        net = CNN()

        net.to(device)

        print(

            f"optimizer\_name={optimizer\_name}, learning\_rate={learning\_rate}! "

        )

        lossF = torch.nn.CrossEntropyLoss()

        optimizers = {

            "ASGD": torch.optim.ASGD,

            "SGD": torch.optim.SGD,

            "Adagrad": torch.optim.Adagrad,

            "Adadelta": torch.optim.Adadelta,

            "RMSprop": torch.optim.RMSprop,

            "Adamax": torch.optim.Adamax,

            "Adam": torch.optim.Adam,

            "Rprop": torch.optim.Rprop,

        }

        if optimizer\_name not in optimizers:

            raise ValueError(f"Unknown optimizer: {optimizer\_name}")

        optimizer\_class = optimizers[optimizer\_name]

        optimizer = optimizer\_class(net.parameters(), lr=learning\_rate)

        correct = 0

        net.train(False)

        path = f"save\_model//CNN//{learning\_rate}"

        model\_path = os.makedirs(path, exist\_ok=True)

        model\_path = path

        net.load\_state\_dict(

            torch.load(f"{model\_path}//{optimizer\_name}\_cnn.pth"))  # 同上

        correct, totalLoss = 0, 0

        net.train(False)

        with torch.no\_grad():

            for step, (testImgs, labels) in enumerate(allDataLoader):

                testImgs = testImgs.to(device)

                labels = labels.to(device)

                net.zero\_grad()

                outputs = net(testImgs)

                loss = lossF(outputs, labels)

                predictions = torch.argmax(outputs, dim=1)

                accuracy = torch.sum(predictions == labels) / labels.shape[0]

                correct += torch.sum(predictions == labels)

                with open(f"model\_test//CNN//{learning\_rate}\_Eva.txt",

                          "a") as f:

                    f.write(

                        f"optimizer = {optimizer\_name}\n{classification\_report(labels.cpu().numpy(),predictions.cpu().numpy(),digits=2)}\n"

                    )

                    accuracy = correct / (BATCH\_SIZE \* len(allDataLoader))

                    accuracy\_list.append(accuracy.item())

                    print(accuracy.item())

                    # print((BATCH\_SIZE \* len(allDataLoader)))

    x = np.arange(1, 9)

    y = accuracy\_list

    fig = plt.figure(figsize=(8, 6))

    plt.bar(x, y)

    plt.xticks(x, optimizers, rotation=-45)

    plt.xlabel('optimizer')

    plt.ylabel('accuracy')

    plt.title(f"{learning\_rate}\_Accuracy comparison")

    for i in range(len(x)):

        plt.text(x[i], y[i], '%.2f' % round(y[i], 2), ha='center', va='bottom')

    plt.savefig(f"model\_test//CNN//{learning\_rate}\_AccCom.png")

    plt.show()

**SVM：**

from sklearn.svm import LinearSVC

import csv

import numpy as np

import joblib

import pandas as pd

import numpy as np

import random

from sklearn.metrics import classification\_report

def split\_dataset(dataset, ratio, random\_state=42):

    dataset\_copy = np.array(dataset).copy()

    train\_size = int(len(dataset\_copy) \* ratio)

    train\_set = []

    valid\_set = list(dataset\_copy)

    np.random.seed(random\_state)

    while len(train\_set) < train\_size:

        index = random.randrange(len(valid\_set))

        train\_set.append(valid\_set.pop(index))

    return np.array(train\_set), np.array(valid\_set)

#读取数据函数，返回属性和标签

def read\_data(filename):

    data = pd.read\_csv(filename, header=None).values.astype(float)

    return data

#读取数据以及数据标准化

data = read\_data('data//MNIST//raw//mnist\_test.csv')

data, label = data.copy()[:, 1:] / 255, data.copy()[:, 0].reshape((-1, 1))

print("训练开始！")

svc = LinearSVC(dual=False)

svc = joblib.load('save\_model\\svc.pkl')

results = svc.predict(data).reshape((-1, 1))

with open(f"model\_test//SVM//svm.txt", "w") as f:

    f.write(f"{classification\_report(label,results,digits=2)}\n")

print(np.sum(results == label) / label.shape[0])

**决策树：**

from sklearn import tree

import numpy as np

import csv

import matplotlib.pyplot as plt

import joblib

import pandas as pd

import numpy as np

import random

from sklearn.metrics import classification\_report

#读取数据函数，返回属性和标签

def read\_data(filename):

    data = pd.read\_csv(filename, header=None).values.astype(float)

    return data

def split\_dataset(dataset, ratio, random\_state=42):

    dataset\_copy = np.array(dataset).copy()

    train\_size = int(len(dataset\_copy) \* ratio)

    train\_set = []

    valid\_set = list(dataset\_copy)

    np.random.seed(random\_state)

    while len(train\_set) < train\_size:

        index = random.randrange(len(valid\_set))

        train\_set.append(valid\_set.pop(index))

    return np.array(train\_set), np.array(valid\_set)

#读取数据以及数据标准化

data = read\_data('data//MNIST//raw//mnist\_test.csv')

data, label = data.copy()[:, 1:] / 255, data.copy()[:, 0].reshape((-1, 1))

print(len(data))

clf = tree.DecisionTreeClassifier()

clf = joblib.load('save\_model\\tree.pkl')

results = clf.predict(data).reshape((-1, 1))

with open(f"model\_test//TREE//tree.txt", "w") as f:

    f.write(f"{classification\_report(label,results,digits=2)}\n")

print(np.sum(results == label) / label.shape[0])

#load model

# classifier = joblib.load('save\_model\\classifier.pkl')

# 以文字形式输出树

# text\_representation = tree.export\_text(classifier)

# print(text\_representation)

# # 用图片画出

# plt.figure(figsize=(30, 10), facecolor='g')  #

# a = tree.plot\_tree(classifier, rounded=True, filled=True, fontsize=14)

# plt.show()