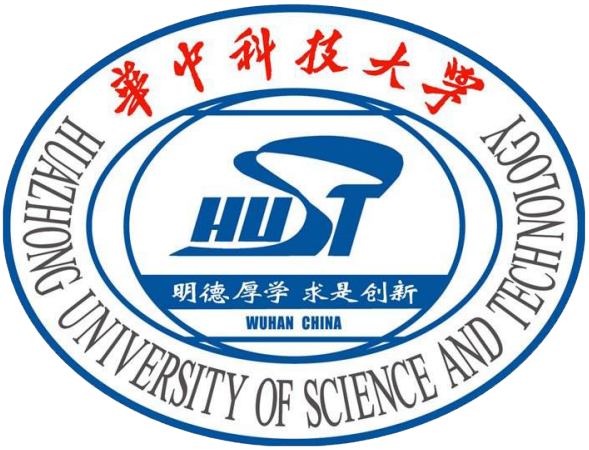
华中科技大学计算机科学与技术学院

《机器学习》结课报告



专　　业	计算机科学与技术
班　　级	CS2203
学　　号	U202215643
姓　　名	王国豪
成　　绩	
指导教师	张　腾
时　　间	2024 年 5 月 22 日

目录

1 实验要求

总体要求：

使用经典的 MNIST 手写数字数据集作为实验数据源，自己动手进行模型训练，严禁直接调用已经封装好的各类机器学习库，使用机器学习及相关知识对数据进行建模和训练，并进行相应参数调优和模型评估。

训练集说明：数据文件 train.csv 和 test.csv 包含手绘数字的灰度图像，从 0 到 9。每张图像的高度为 28 像素，宽度为 28 像素，总共 784 像素。每个像素都有一个与之关联的像素值，表示该像素的亮度或暗度，数字越大表示越暗。此像素值是介于 0 和 255 之间的整数（含 0 和 255）。训练数据集（train.csv）有 785 列。第一列称为“标签”，是用户绘制的数字。其余列包含关联图像的像素值。训练集中的每个像素列都有一个类似 pixelx 的名称，其中 x 是介于 0 和 783 之间的整数（含 0 和 783）。要在图像上找到这个像素，假设我们已将 x 分解为 x = i * 28 + j，其中 i 和 j 是介于 0 和 27 之间的整数，包括 0 和 27（含）。然后 pixelx 位于 28 x 28 矩阵的第 i 行和第 j 列（按零索引）。

2 算法设计与实现

使用卷积神经网络（CNN）模型和 K 最近邻（KNN）算法解决手写数字识别问题。对于 CNN 模型，它包括卷积层、池化层和全连接层，通过梯度下降法训练模型参数，最终在测试集上达到一定的分类准确率。而对于 KNN 算法，它是一种基于实例的学习方法，通过测量不同特征之间的距离，找到与新样本最相似的 K 个训练样本，并通过多数表决的方式进行分类。同样地，我们可以通过调整 KNN 的超参数 K 和距离度量方法，最终在测试集上获得一定的分类准确率。

2.1 数据集的分析与数据处理

数据的读取和预处理：从”train.csv” 文件中读取数据, 此时可以打印输出观察一下文件里面的数据组成, 见图2.1, 并使用 pd.read_csv() 将其转换成 pandas 的 DataFrame, 然后再转换成 numpy 数组 data。然后我们可以利用 matplotlib 包将里面的部分数据可视化，增加我们的理解。见图2.2

	label	pixel0	pixel1	...	pixel781	pixel782	pixel783
count	42000.000000	42000.0	42000.0	...	42000.0	42000.0	42000.0
mean	4.456643	0.0	0.0	...	0.0	0.0	0.0
std	2.887730	0.0	0.0	...	0.0	0.0	0.0
min	0.000000	0.0	0.0	...	0.0	0.0	0.0
25%	2.000000	0.0	0.0	...	0.0	0.0	0.0
50%	4.000000	0.0	0.0	...	0.0	0.0	0.0
75%	7.000000	0.0	0.0	...	0.0	0.0	0.0
max	9.000000	0.0	0.0	...	0.0	0.0	0.0

[8 rows x 785 columns]

图 2.1: 文件内容打印输出

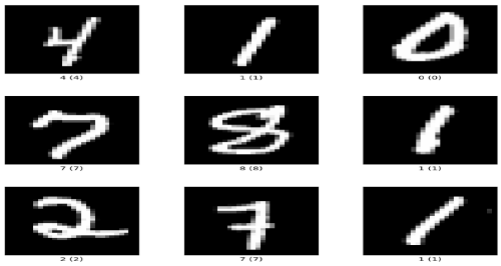


图 2.2: 部分数据可视化

将数据集划分为训练集和测试集: 计算训练集的大小 num_train 为数据集总量的 90%。使用 np.random.permutation() 对数据集的索引进行随机排序, 得到 permuted_indices。取前 num_train 个索引作为训练集索引 train_indices, 剩余的索引作为测试集索引 test_indices。使用这些索引从原始数据 data 中选取训练集 train_data 和测试集 test_data。

进行归一化处理：对训练集数据 train_data 的第 2 列到最后一列 (即特征) 进行 reshape 操作, 变成 4 维张量 train_X, 大小为 (样本数, 1, 28, 28)。对训练集数据 train_data 的第 1 列 (即标签) 赋值给 train_y。对测试集数据 test_data 的第 2 列到最后一列进行同样的 reshape 操作, 得到 test_X。对测试集数据 test_data 的第 1 列赋值给 test_y。最后对 train_X 和 test_X 进行归一化处理, 除以 256。

2.2 cnn 模型设计

卷积神经网络（CNN）属于一种多层的神经网络，它特别擅长处理大图像相关的机器学习问题。卷积神经网络的层级结构包括：数据输入层（Input layer），卷积计算层

(CONV layer)，激活函数层（ReLU layer)，池化层（Pooling layer）和全连接层（FC layer)。

采用" 卷积 - 池化 - 全连接" 的经典 CNN 架构，我设计了一个包含卷积层，偏置层，激活层，池化层，以及全连接层的一个神经网络，具体实现为以下代码。

```
model = Model(layers=[
    ConvolveLayer(input_size=28, in_channels=1,out_channels=6,
        kernel_size=5),
    BiasLayer(shape=(6, 24, 24)),
    SigmoidLayer(),
    PoolingLayer(pooling_size=2),
    ConvolveLayer(input_size=12, in_channels=6, out_channels=16,
        kernel_size=5),
    BiasLayer(shape=(16, 8, 8)),
    SigmoidLayer(),
    PoolingLayer(pooling_size=2),
    ReshapeLayer(From=(16, 4, 4), To=(256,)),
    LinearLayer(input_size=256, output_size=80),
    SigmoidLayer(),
    LinearLayer(input_size=80, output_size=10),
])
```

2.2.1 卷积层实现

卷积神经网络中的卷积操作是通过滑动窗口机制，将卷积核（也称为过滤器）应用于输入数据，以提取特征。

在前向传播中,输入数据通过卷积操作生成输出。假设输入数据 $\mathbf{X}$ 形状为 $(\text{batch\_size}, C_{\text{in}}, H, W)$，卷积核 $\mathbf{K}$ 形状为 $(C_{\text{out}}, C_{\text{in}}, K_h, K_w)$，输出数据 $\mathbf{Y}$ 形状为 $(\text{batch\_size}, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$。

前向传播公式：

输出的高度和宽度计算如下：

$$H_{\text{out}} = H - K_h + 1$$

$$W_{\text{out}} = W - K_w + 1$$

对于每个输出位置 (i, j)，卷积操作公式如下：

$$Y_{b,o,i,j} = \sum_{c=0}^{C_{\text{in}}-1} \sum_{m=0}^{K_h-1} \sum_{n=0}^{K_w-1} X_{b,c,i+m,j+n} \cdot K_{o,c,m,n}$$

前向传播代码：

```python
def forward(self, x):
    self.last_input = x
    batch_size = x.shape[0]
    y = np.zeros(shape=(batch_size, self.out_channels, self.out_size,
        self.out_size))

    for i in range(self.out_size):
        for j in range(self.out_size):
            y[:, :, i, j] += np.einsum(
                "bimn,oimn->bo",
                x[:, :, i:i + self.kernel_size, j:j + self.kernel_size], # x[batch,i_channel,K,K]
                self.kernels.value # [o_channel,i_channel,K,K]
            )

    return y
```

在反向传播中，我们需要计算损失相对于输入数据 **X** 和卷积核 **K** 的梯度。反向传播公式：

1. 损失相对于输入数据 **X** 的梯度：

$$\frac{\partial L}{\partial X_{b,c,i,j}} = \sum_{o=0}^{C_{\text{out}}-1} \sum_{m=0}^{K_h-1} \sum_{n=0}^{K_w-1} \frac{\partial L}{\partial Y_{b,o,i-m,j-n}} \cdot K_{o,c,m,n}$$

2. 损失相对于卷积核 **K** 的梯度：

$$\frac{\partial L}{\partial K_{o,c,m,n}} = \sum_{b=0}^{\text{batch\_size}-1} \sum_{i=0}^{H_{\text{out}}-1} \sum_{j=0}^{W_{\text{out}}-1} \frac{\partial L}{\partial Y_{b,o,i,j}} \cdot X_{b,c,i+m,j+n}$$

2.2.2 激活函数层实现

Sigmoid 函数的数学定义如下：

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

该函数将输入 x 映射到 $(0,1)$ 之间的值。其输出值可以解释为概率，因此常用于二分类问题的输出层。

在前向传播过程中，Sigmoid 函数将输入 x 映射到 $(0,1)$ 之间的值。具体实现如下：

```
def forward(self, x):
    """sigmoid函数体"""
    self.last_output = 1.0 / (1 + np.exp(-x))
    return self.last_output
```

这里，x 是输入数据，self.last_output 存储了 Sigmoid 函数的输出值，以便在反向传播时使用。

在反向传播过程中，需要计算损失函数 L 对输入 x 的梯度。Sigmoid 函数的导数可以表示为：

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

其中，$\sigma(x)$ 是 Sigmoid 函数的输出值。在代码中，self.last_output 就是 $\sigma(x)$。反向传播的实现如下：

```
def backward(self, grad_Loss):
    """反向传播，计算Loss关于x的梯度"""
    # s'(x) = s(x)(1-s(x))
    return self.last_output * (1 - self.last_output) * grad_Loss
```

这里，grad_Loss 是损失函数 L 对 Sigmoid 函数输出的梯度，即 $\frac{\partial L}{\partial \sigma(x)}$。通过链式法则，可以得到损失函数 L 对输入 x 的梯度：

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial \sigma(x)} \cdot \sigma'(x)$$

2.2.3 池化层实现

一个平均池化层 (PoolingLayer) 的前向传播和反向传播过程

前向传播：

获取输入张量 `x` 的批量大小 `batch_size`、通道数 `channel_num`、高度 `H` 和宽度 `W`。计算池化后的输出张量 `pooled_image` 的大小, 其高度和宽度均为输入的 `H` 和 `W` 除以池化窗口大小 `n`。使用嵌套循环遍历输入张量的每个位置 `(i, j)`, 计算该位置对应的池化窗口内元素的平均值, 并将结果赋给输出张量的对应位置 `(i, j)`。返回计算得到的池化后的输出张量 `pooled_image`。

反向传播：

初始化一个与输入张量 `self.x` 形状相同的梯度张量 `x_grad`。再次遍历输入张量的每个位置 `(i, j)`, 将下一层传回的梯度 `grad_Loss` 均匀地分配给该位置对应的池化窗口内的所有元素。这是因为在池化操作中, 每个输出元素是由对应窗口内所有元素的

平均值计算得到的, 因此在反向传播时需要将梯度均匀分散回到原始的输入元素中。最后返回计算得到的梯度张量 `x_grad`。

池化后的输出张量 `pooled_image` 的形状为 `(batch_size, channel_num, H//n, W//n)`。其中第 `(i, j)` 个元素的计算公式为:

```
pooled_image[:, :, i, j] = np.mean(x[:, :, i*n:(i+1)*n, j*n:(j+1)*n], axis=(2, 3))
```

即将输入张量 `x` 在高度和宽度维度上划分为 `n x n` 的小窗口, 并对每个小窗口内的元素取平均值, 作为池化后输出张量中对应位置的元素值。

2.2.4 全连接层实现

前向传播方法 `forward` 接受输入 x，其形状为 $[\text{batch\_size}, \text{input\_size}]$。它会将输入 x 与权重矩阵 `self.weight` 相乘，得到输出 `ans`。前向传播的公式如下:

$$\text{ans} = x \cdot (\text{self.weight.value})^\top$$

反向传播方法 `backward` 接受输出梯度 `grad_Loss`,其形状为 $[\text{batch\_size}, \text{output\_size}]$。它会根据输出梯度 `grad_Loss` 和最后一次输入 `self.last_input`,计算权重矩阵 `self.weight` 的梯度，并计算输入 x 的梯度。

权重梯度的计算公式:

$$\text{self.weight.grad} += (\text{grad\_Loss})^\top \cdot \text{self.last\_input}$$

输入梯度的计算公式:

$$x_{\text{grad}} = \text{grad\_Loss} \cdot \text{self.weight.value}$$

最后，它返回计算得到的输入梯度 x_{grad}。

2.3 knn 模型设计

KNN 算法的实现相对简单直观，核心在于计算距离和多数投票。它是一种无参数模型，没有训练过程，预测速度较慢，但对于小规模数据集和简单分类任务仍然是一种有效的选择。

KNN 算法的核心是 KNN 函数。该函数接收一个测试样本、训练集的特征数据和标签数据以及 K 值作为输入参数。具体实现步骤如下:

1. 计算测试样本与每个训练样本之间的距离。这里使用了欧氏距离来度量样本之间的距离，即对应特征差的平方和再开根号。

2. 将距离排序，并选取距离最近的 K 个样本。

3. 统计这 K 个样本中各个类别的出现次数。

4. 根据多数投票的原则，选择出现次数最多的类别作为测试样本的预测类别。

2.4 训练模型

cnn 模型中使用 Adam 优化器和学习率为 0.01 的设置，训练模型 epoch 设置为 600，批量大小为 64，并在训练过程中打印损失。

```
    model.fit(
    X_list=train_X,
    y_list=train_y,
    epoch_num=600,
    batch_size=64,
    print_Loss=True,
    optimizer=Adam(learning_rate=0.01)
)
```

3 实验环境与平台

表 3.1: **硬件信息**

项目	信息
设备名称	Legion Y7000P (定制版)
CPU	13th Gen Intel Core i9-13900H (24) @ 5.00GHz
GPU	NVIDIA GeForce RTX 4070 Mobile
RAM	32GB DDR5
存储	1TB NVMe SSD + 2TB HDD（机械硬盘）
系统类型	x86_64
显示器	16 英寸 QHD (2560x1600) IPS 显示屏，165Hz 刷新率
操作系统	Windows 11

表 3.2: **软件信息**

项目	信息
操作系统	Windows 11 Home
版本	22000.556
IDE	PyCharm Professional 2024.1.1
Python	Python 3.12.3
Jupyter	JupyterLab v3.2.1

4 结果与分析

4.1 模型训练过程的分析思考

关于训练的 echo 次数，当 echo 达到 500 的时候正确率有 92%, 当 echo 大于等于 600 以后就稳定在 93%，但是 echo 增大也意味着训练时间的增加，权衡利弊下选择了 echo=600。

为了逐步优化模型并评估其性能，在每个训练轮次结束后，打印出当前训练集和测试集上的损失值和准确率，以便监控模型的性能。通过多轮迭代，最终可以观察到模型的训练损失逐渐降低，同时测试准确率逐渐提升，从而得到一个经过优化的手写数字识别模型。

关于学习率的设置，这也是一个让我头疼的问题，一开始学习率设定为 0.001，但是收敛过慢，效果不好，后来查看了图4.1, 慢慢增加学习率，发现学习率为 0.01 的时候效果最好。

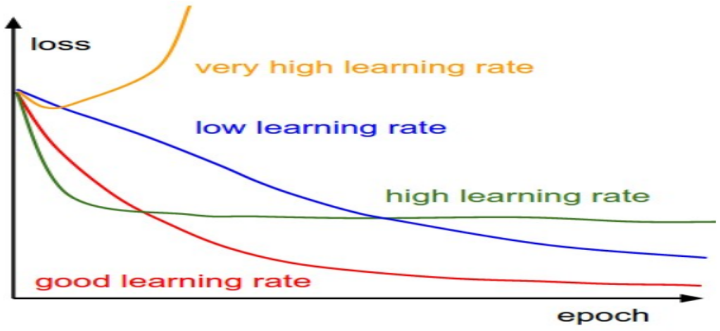


图 4.1: 不同学习率随 epoch 变化的 loss 图像

4.2 结果的分析思考

对 test.csv 文件进行处理结果放到 Kaggle 进行评测，所得分数见图4.1,cnn 算法准确率达到了 93.928%,knn 算法的准确率达到了 94.567%, 但都还有需要改进的地方。

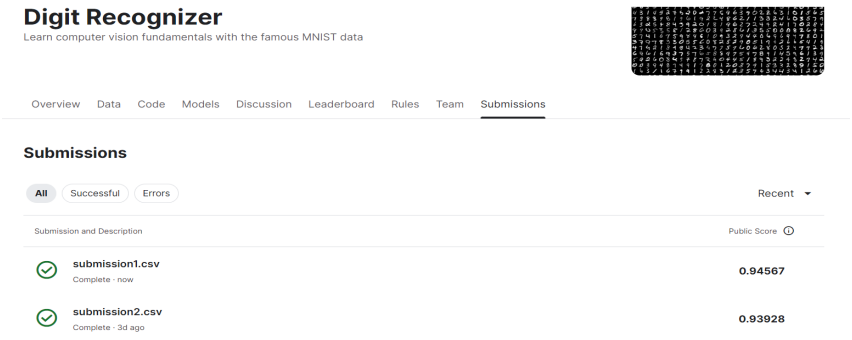


图 4.2: Kaggle 得分截图

4.3 改进算法思考

我想着还能不能改进一下正确率，要么增加卷积层数，要么换种算法实现，通过查阅资料，发现有一下几种可行且高效的算法可以帮助我改进正确率。

Deep CNN

采用深层卷积神经网络, 使用更深的卷积层和池化层，增加网络的表达能力。

SVM

通过训练，SVM 可以学习到一个分类器，该分类器可以将手写数字图像分为 0-9 十个类别。在识别过程中，SVM 通过对输入的手写数字图像进行特征提取和分类，实现手写数字的自动识别。

5 个人体会

通过完成这个实践研究，我获得了许多宝贵的学习和实践经验。本实践研究基于卷积神经网络（CNN）和 K 最近邻（KNN）算法实现手写数字识别模型，并对 MNIST 数据集进行训练和测试。以下是我在研究过程中所获得的主要体会：

从数据来源的角度来看，MNIST 手写数字数据集是一个经典的、广泛应用于机器学习和计算机视觉领域的基准数据集。其提供了大量的手写数字图像样本，为手写数字识别任务提供了丰富的资源。

在问题分析阶段，我明确了手写数字识别的任务描述，并确定了数据分析的目标。这有助于更好地理解问题的关键要素，并为后续的数据预处理和模型设计提供指导。

在数据预处理阶段，我对图像数据进行了加载、转换和批处理。通过将数据转换为合适的数据格式并进行归一化处理，我为模型的输入准备了合适的数据。同时，通过高效的数据加载和处理方法，我能够处理大量的训练和测试样本。

在模型求解阶段，我选择了卷积神经网络（CNN）和 K 最近邻（KNN）算法作为手写数字识别模型，并设计了具体的网络结构。通过训练和优化模型，我不断提升模型的性能，并在训练和测试集上进行评估。

通过实验结果和模型评估，可以得出结论：基于 CNN 和 KNN 的手写数字识别模型在 MNIST 数据集上取得了令人满意的性能。可以观察到模型的训练损失逐渐降低，同时测试准确率逐渐提升，表明模型具有较强的学习能力和泛化能力。

通过这个实践研究，我不仅学习了深度学习的基本原理，还深入了解了 CNN 和 KNN 的应用。我学会了如何进行数据预处理、模型搭建、训练和评估，并培养了对模型性能的分析和改进能力。它帮助我在深度学习领域迈出了重要的一步。通过这个项目，我还拓展了自己的编程技能和实际问题解决能力。我相信这些所学将对我的学术和职业发展产生积极影响。

参考文献

[1] 周志华. 机器学习: 第 3 章. 清华大学出版社, 2016.

[2] 蒋文斌, 彭晶, and 叶阁焰. ” 深度学习自适应学习率算法研究.” 华中科技大学学报 (自然科学版) 47.5 (2019): 79-83.

[3] 任丹, and 陈学峰. ” 手写数字识别的原理及应用.” 计算机时代 3 (2007): 17-18.