深度学习与神经网络第二次课程项目

王逸群 19307110397

2022.4.9

目录

1 神经网络

1.1 初始设置

本项目使用 CIFAR-10 数据集，其中包含 60000 张 32×32 的彩色图片，被平均分为 10 类：飞机、汽车、鸟、猫、鹿、狗、青蛙、马、船、货车。

参考 VGG 网络架构，基于 pytorch 框架，设计神经网络初始架构。对于输入的图像，先进行两轮卷积、激活、池化操作，使图像边长由 32 变为 16 再变为 8，图像频道数由 3 变为 16 再变为 32；接着进行三轮线性、激活操作，使神经元数量由 32*8*8 变为 128 再变为 10。初始架构的参数数量为 285162，类存储于`Code/nn.py`，具体内容如下：

```
class NN(nn.Module):
    def __init__(self, in_channels = 3, hidden_channels = (16, 32),
                 hidden_neurons = (128, 128), num_classes = 10):
        super().__init__()
        self.hidden_channels = hidden_channels

        self.extractor = nn.Sequential(
            # stage 1
            nn.Conv2d(in_channels = in_channels,
                      out_channels = hidden_channels[0],
                      kernel_size = 3, padding = 1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 2),

            # stage 2
            nn.Conv2d(in_channels = hidden_channels[0],
                      out_channels = hidden_channels[1],
                      kernel_size = 3, padding = 1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 2))

        self.classifier = nn.Sequential(
            nn.Linear(hidden_channels[1] * 8 * 8, hidden_neurons[0]),
            nn.ReLU(),
            nn.Linear(hidden_neurons[0], hidden_neurons[1]),
            nn.ReLU(),
            nn.Linear(hidden_neurons[1], num_classes))

    def forward(self, inputs):
        hidden = self.extractor(inputs)
        outputs = \
            self.classifier(hidden.view(-1,
                                        self.hidden_channels[1] * 8 * 8))
        return outputs
```

其余参数的初始设置如下：

损失函数：交叉熵损失函数；

优化器：Adam；

学习率：0.001；

初始设置运行结果如图 1所示，训练集上的最优错误率为 0.05860，在第 19 回合出现；测试集上的最优错误率为 0.30160，在第 8 回合出现。

图 1: 原始模型在测试集和验证集上的错误率

1.2 参数调整

1.2.1 神经元数量

本节在总体架构不变的基础上，改变神经元数量，实验设置如表 1所示，结果如表 2和图 2所示。

	hidden_channels	hidden_neurons	参数数量
原模型	(16, 32)	(128, 128)	285162
更小的模型	(4, 8)	(32, 32)	18210
更大的模型	(64, 128)	(512, 512)	4538250

表 1: 神经元数量实验设置

可以看到，随着模型的规模变大，参数数量增加，训练集和测试集的最优错误率都有所上升，但是测试集最优错误率的上升幅度非常有限。

	训练集最优错误率	回合	测试集最优错误率	回合
原模型	0.05860	19	0.30160	8
更小的模型	0.37472	20	0.40570	20
更大的模型	0.00978	19	0.26310	4

表 2: 神经元数量实验结果

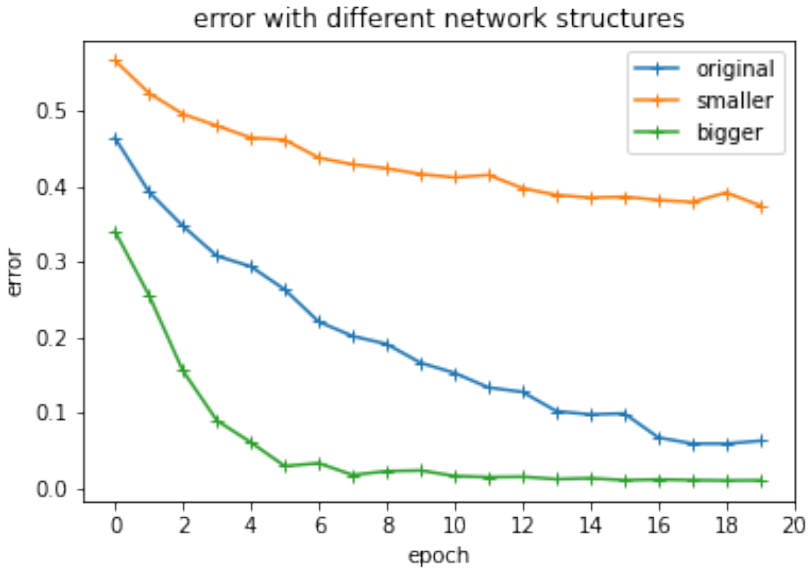

图 2: 神经元数量实验结果

1.2.2 损失函数

初始设置使用交叉熵损失函数，本节尝试使用多分类的合页损失函数。实验结果如图 3和表 3所示。

可以看到，使用多分类的合页损失函数并没有明显的提升效果。

1.2.3 正则化

初始设置未加入正则化，本节尝试使用不同的正则化参数，实验结果如图 4和表 4所示。

可以看到，正则化并不能提升结果，可能的原因是初始模型并没有出现严重的过拟合现象。

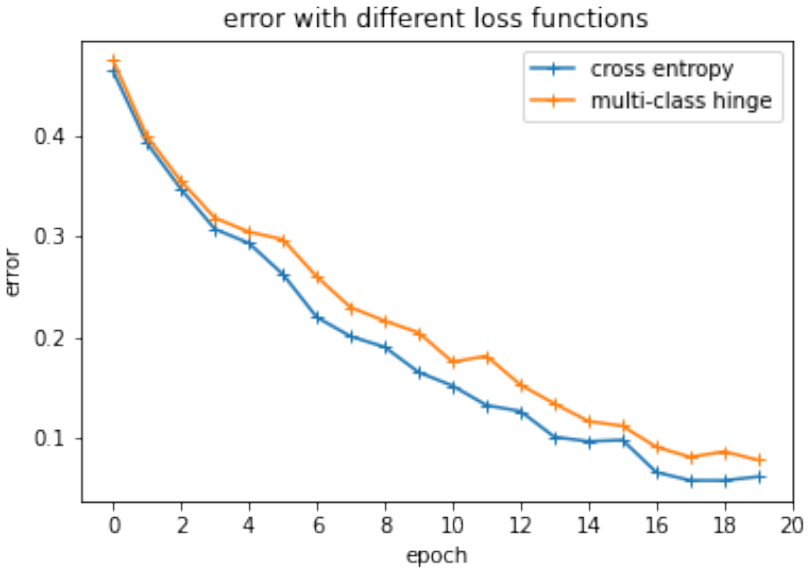

图 3: 损失函数实验结果

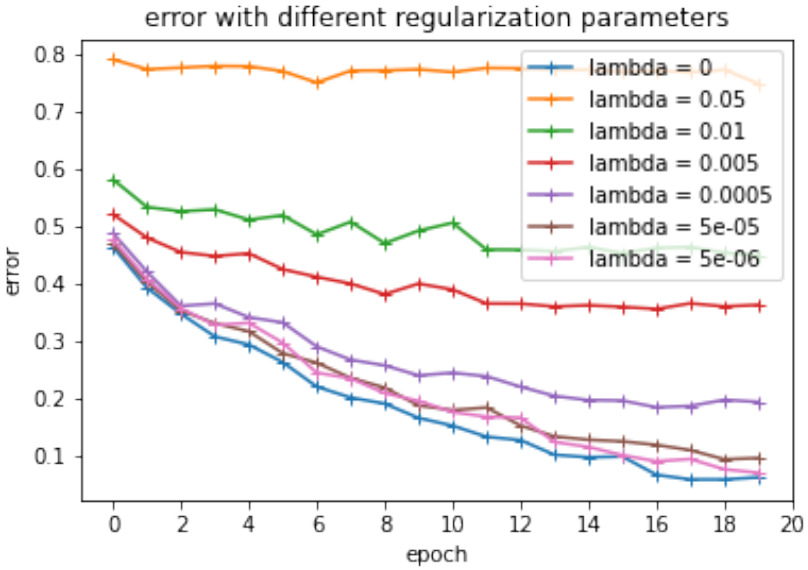

图 4: 正则化实验结果

损失函数	训练集最优错误率	回合	测试集最优错误率	回合
交叉熵	0.05860	19	0.30160	8
合页	0.07886	20	0.31200	15

表 3: 损失函数实验结果

正则化参数	训练集最优错误率	回合	测试集最优错误率	回合
0	0.05860	19	0.30160	8
0.05	0.74906	20	0.74940	20
0.01	0.45312	16	0.45140	20
0.005	0.35582	17	0.37120	17
0.0005	0.18448	17	0.30160	17
0.00005	0.09378	19	0.31440	10
0.000005	0.07034	20	0.31110	15

表 4: 正则化实验结果

1.2.4 激活函数

初始设置使用 ReLU 激活函数，本节尝试使用 tanh 和 softplus 激活函数。实验结果如图 5和表 5所示。

激活函数	训练集最优错误率	回合	测试集最优错误率	回合
ReLU	0.05860	19	0.30160	8
tanh	0.01660	19	0.31510	8
softplus	0.18318	20	0.36490	14

表 5: 激活函数实验结果

可以看到，在训练集上，tanh 激活函数的效果优于 ReLU，softplus 最次；但在测试集上，ReLU 表现最优。

1.2.5 优化器

初始设置使用 Adam 优化器，本节尝试使用随机梯度下降优化器、带有动量的随机梯度下降优化器、以及 Adagrad 优化器，实验结果如图 6和表 6所示。

可以看到，初始设置的 Adam 优化器效果最优。

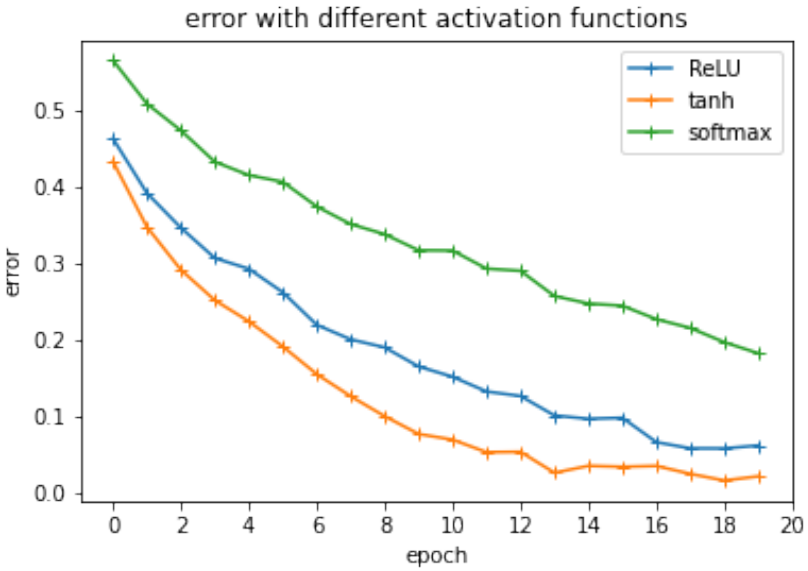

图 5: 激活函数实验结果

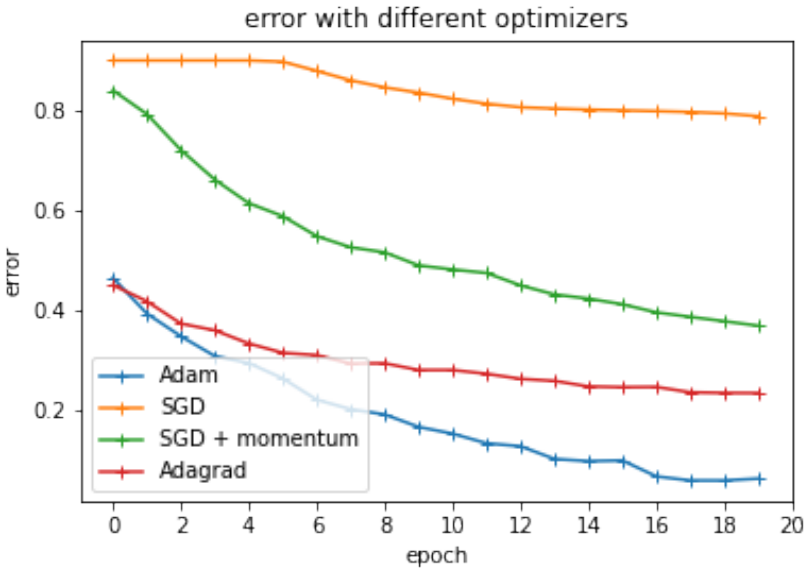

图 6: 优化器实验结果

优化器	训练集最优错误率	回合	测试集最优错误率	回合
Adam	0.05860	19	0.30160	8
SGD	0.78832	20	0.78390	20
Momentum	0.36856	20	0.39020	20
Adagrad	0.23366	20	0.31240	18

表 6: 优化器实验结果

1.2.6 批归一化

本节尝试使用批归一化，实验结果如图 7和表 7所示。

批归一化	训练集最优错误率	回合	测试集最优错误率	回合
否	0.05860	19	0.30160	8
是	0.01306	18	0.29240	5

表 7: 批归一化实验结果

可以看到，批归一化很好地提升了模型的效果，但是测试集最优错误率的上升幅度非常有限。

1.2.7 丢弃法

本节尝试使用丢弃法，实验结果如图 8和表 8所示。

丢弃概率	训练集最优错误率	回合	测试集最优错误率	回合
0	0.05860	19	0.30160	8
0.2	0.17718	20	0.30560	20
0.5	0.34874	20	0.38150	20

表 8: 丢弃法实验结果

可以看到，丢弃法使得收敛速度变慢，无法提升模型效果。

1.3 最优设置

最终选择的最优设置是带有批归一化的模型. 对于输入的图像，先进行两轮卷积、批归一化、激活、池化操作，使图像边长由 32 变为 16 再变为

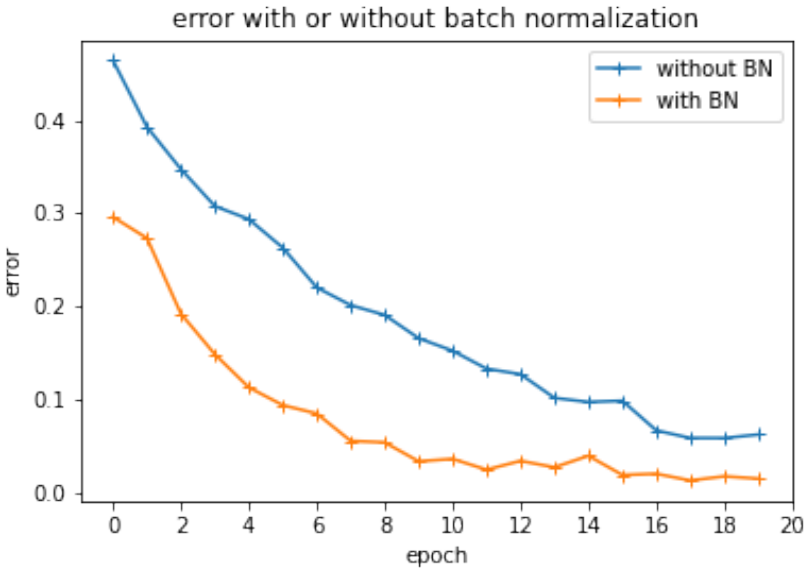

图 7: 批归一化实验结果

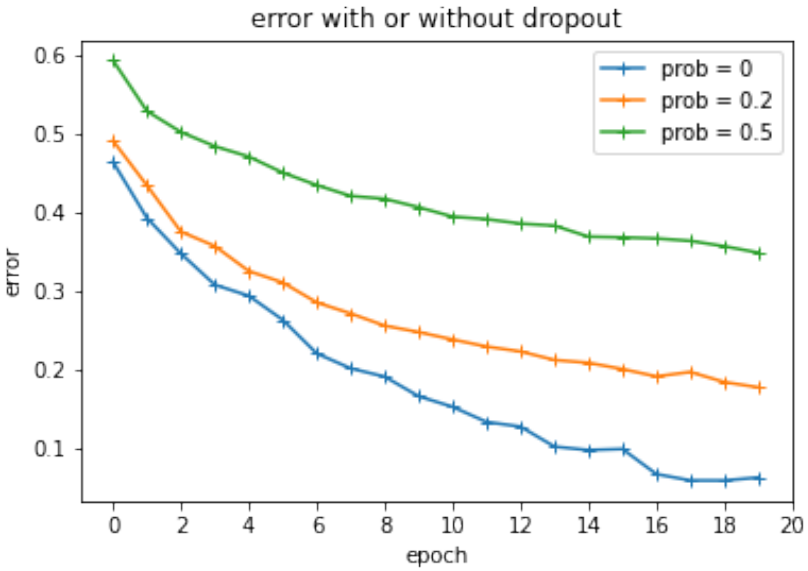

图 8: 丢弃法实验结果

8，图像频道数由 3 变为 16 再变为 32；接着进行三轮线性、批归一化、激活操作，使神经元数量由 32*8*8 变为 128 再变为 10。参数数量为 285770，类存储于Code/nn.py，具体内容如下：

```
class NN_BN(nn.Module):
    def __init__(self, in_channels = 3, hidden_channels = (16, 32),
                 hidden_neurons = (128, 128), num_classes = 10):
        super().__init__()
        self.hidden_channels = hidden_channels

        self.extractor = nn.Sequential(
            # stage 1
            nn.Conv2d(in_channels = in_channels,
                      out_channels = hidden_channels[0],
                      kernel_size = 3, padding = 1),
            nn.BatchNorm2d(hidden_channels[0]),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 2),

            # stage 2
            nn.Conv2d(in_channels = hidden_channels[0],
                      out_channels = hidden_channels[1],
                      kernel_size = 3, padding = 1),
            nn.BatchNorm2d(hidden_channels[1]),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 2))

        self.classifier = nn.Sequential(
            nn.Linear(hidden_channels[1] * 8 * 8, hidden_neurons[0]),
            nn.BatchNorm1d(hidden_neurons[0]),
            nn.ReLU(),
            nn.Linear(hidden_neurons[0], hidden_neurons[1]),
            nn.BatchNorm1d(hidden_neurons[1]),
            nn.ReLU(),
            nn.Linear(hidden_neurons[1], num_classes))

    def forward(self, inputs):
        hidden = self.extractor(inputs)
        outputs = \
            self.classifier(hidden.view(-1,
                                        self.hidden_channels[1] * 8 * 8))
        return outputs
```

实验结果如图 7和表 7所示。