

# 深度学习与神经网络第二次课程项目

王逸群 19307110397

2022.4.9

## 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  $32 \times 32$  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  $32 \times 8 \times 8$  变为 128 再变为 10。初始架构的参数数量为 285162，类存储于 `Code/nm.py`，具体内容如下：

```
1 class NN(nn.Module):
2     def __init__(self, in_channels = 3, hidden_channels = (16, 32),
3                 hidden_neurons = (128, 128), num_classes = 10):
4         super().__init__()
5         self.hidden_channels = hidden_channels
6
7         self.extractor = nn.Sequential(
8             # stage 1
9             nn.Conv2d(in_channels = in_channels,
10                      out_channels = hidden_channels[0],
11                      kernel_size = 3, padding = 1),
12             nn.ReLU(),
13             nn.MaxPool2d(kernel_size = 2, stride = 2),
14
15             # stage 2
16             nn.Conv2d(in_channels = hidden_channels[0],
17                      out_channels = hidden_channels[1],
18                      kernel_size = 3, padding = 1),
19             nn.ReLU(),
```

```

20         nn.MaxPool2d(kernel_size = 2, stride = 2))
21
22     self.classifier = nn.Sequential(
23         nn.Linear(hidden_channels[1] * 8 * 8, hidden_neurons[0]),
24         nn.ReLU(),
25         nn.Linear(hidden_neurons[0], hidden_neurons[1]),
26         nn.ReLU(),
27         nn.Linear(hidden_neurons[1], num_classes))
28
29     def forward(self, inputs):
30         hidden = self.extractor(inputs)
31         outputs = \
32             self.classifier(hidden.view(-1,
33                                     self.hidden_channels[1] * 8 * 8))
34     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。

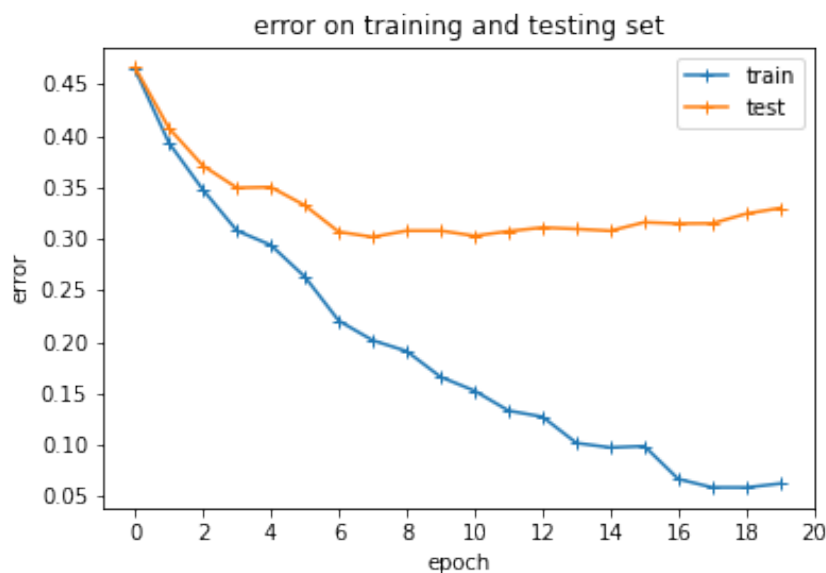


图 1: 原始模型在测试集和验证集上的错误率

## 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

	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

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。

### 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。

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

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。

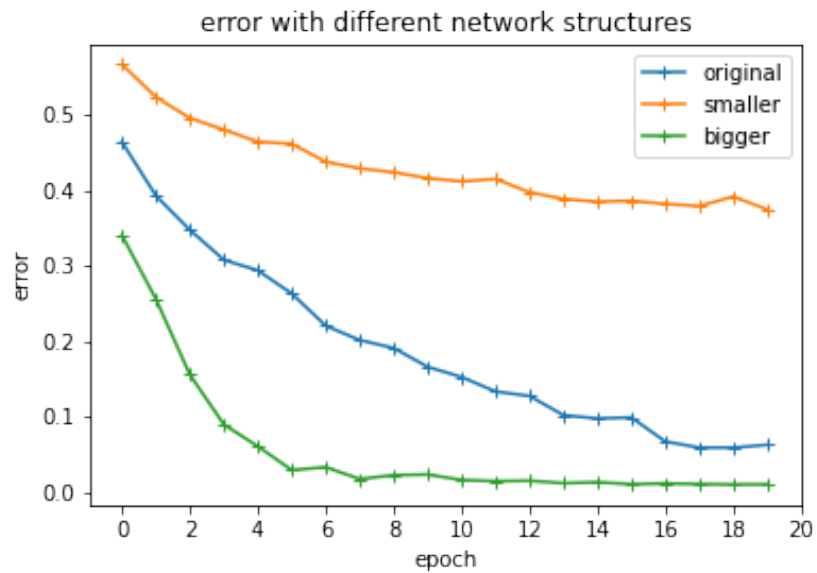


图 2: 神经元数量实验结果

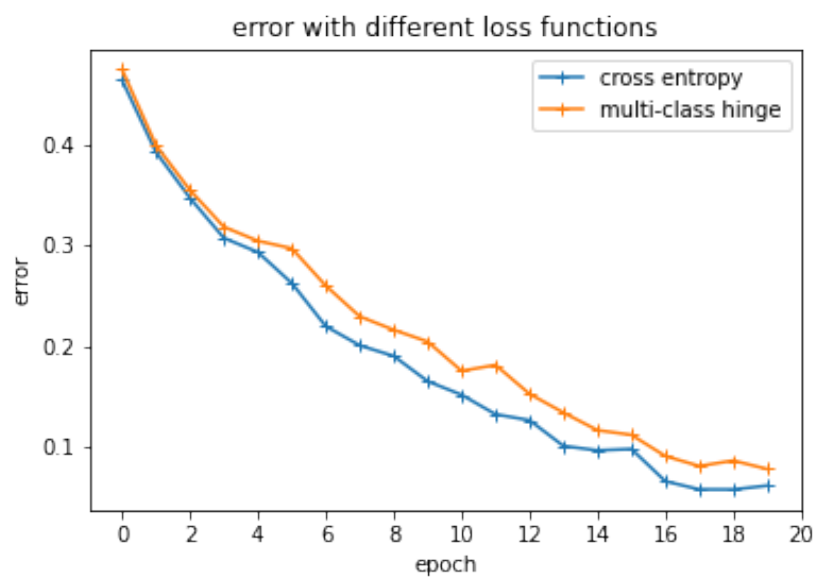


图 3: 损失函数实验结果

### 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。

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

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过拟合。

### 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				
softplus				

表 5: 激活函数实验结果

### 1.2.5 优化器

初始设置使用 Adam 优化器，本节尝试使用 SGD 优化器、带有 Momentum 的 SGD 优化器、以及 Adagrad 优化器，实验结果如图和表 6所示。

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				
Momentum				
Adagrad				

表 6: 优化器实验结果

### 1.2.6 批归一化

### 1.2.7 丢弃法

### 1.2.8 残差连接

## 1.3 可视化

### 1.3.1 卷积核可视化

### 1.3.2 损失函数曲线

## 2 批归一化

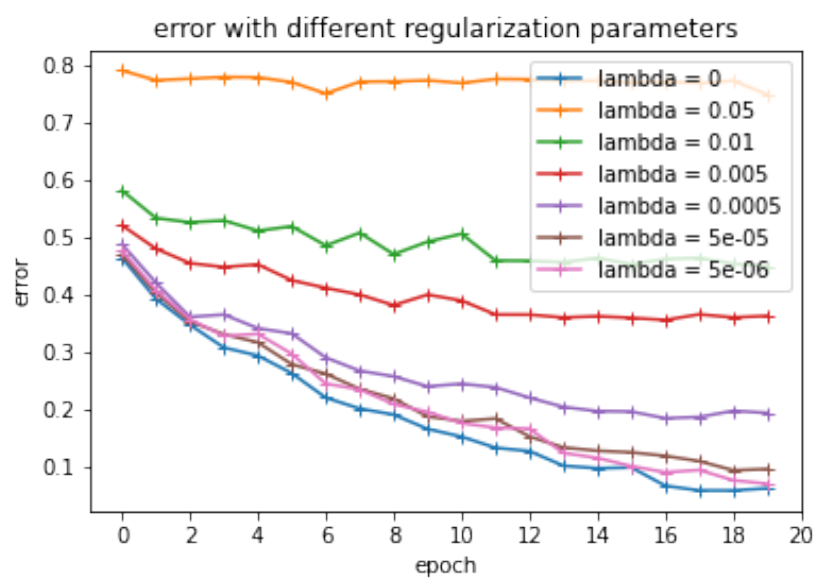


图 4: 正则化实验结果