CIFAR-10 图像分类三层神经网络实验报告

1. 项目概述

本项目实现了一个三层神经网络分类器，用于CIFAR-10数据集的图像分类任务。整个实现过程完全基于numpy，手动实现了前向传播和反向传播算法，不依赖任何深度学习框架。代码和权重在github地址

1.1 CIFAR-10数据集

CIFAR-10是一个包含10个类别的彩色图像数据集，每个类别包含6000张32x32的彩色图像。10个类别分别是：飞机、汽车、鸟、猫、鹿、狗、青蛙、马、船和卡车。数据集分为50000张训练图像和10000张测试图像。

1.2 模型架构

本项目实现的三层神经网络架构如下：

- 输入层：3072个神经元（32x32x3的展平图像）
- 第一隐藏层：可配置大小，默认200个神经元
- 输出层：10个神经元，对应10个类别

激活函数支持ReLU、Sigmoid和Tanh，损失函数使用交叉熵，同时支持L2正则化。

2. 实验结果与分析

2.1 超参数搜索

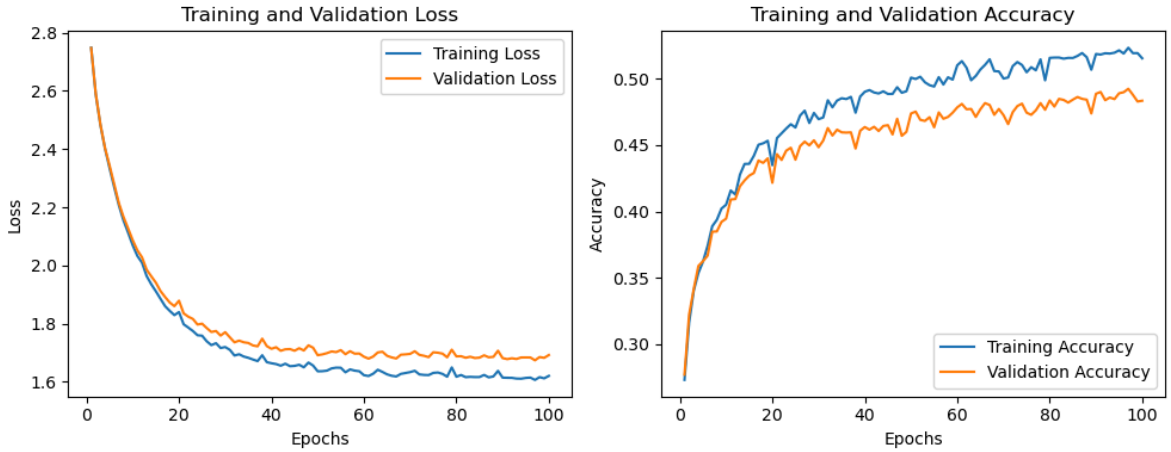
超参数搜索探索了以下几个关键参数的影响：

1. **学习率**：尝试了[0.001, 0.01, 0.05, 0.1, 0.5]
2. **隐藏层大小**：尝试了[50, 100, 200, 500]
3. **正则化系数**：尝试了[0.0, 0.0001, 0.001, 0.01, 0.1]
4. **激活函数**：尝试了[relu, sigmoid, tanh]

确认了在epoch=100的情况下 0.01 500 relu 0.01为最优解

2.2 训练曲线

下面是使用最佳超参数训练模型的训练曲线：

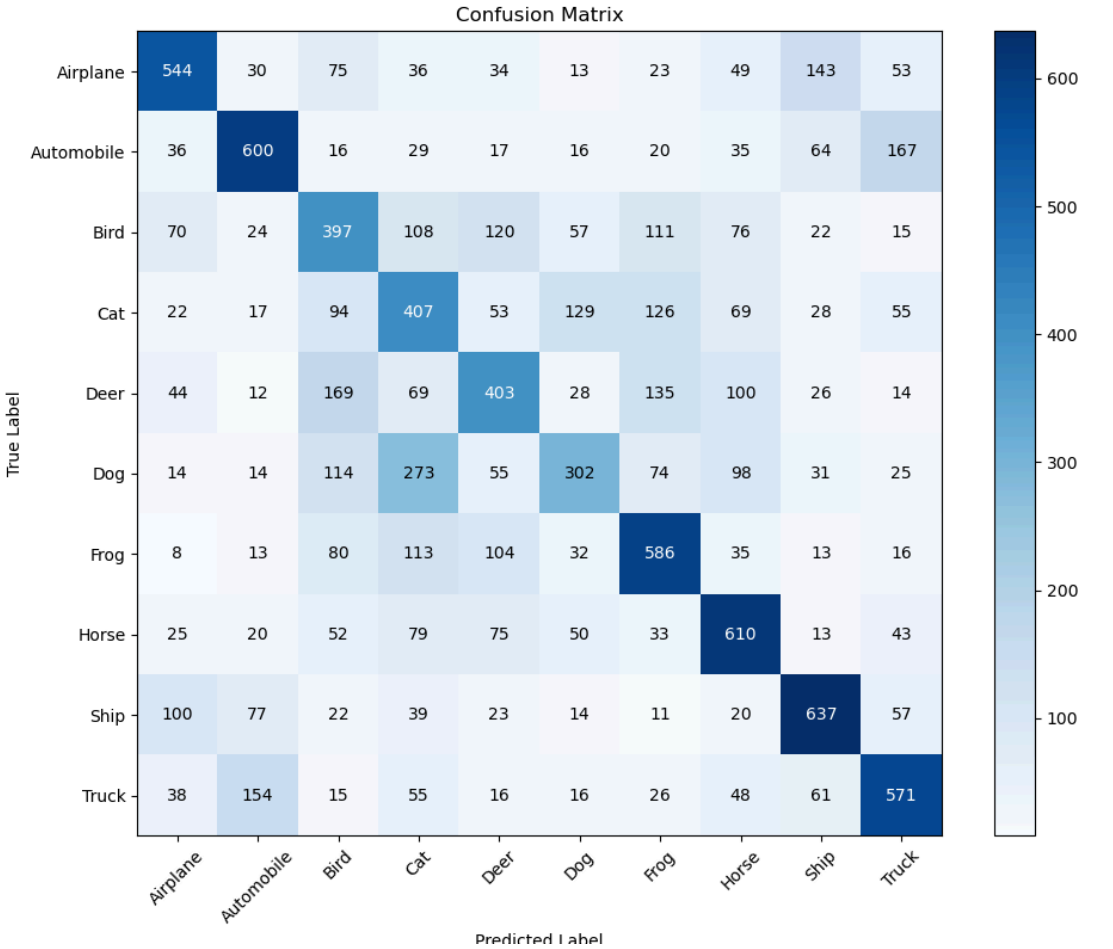


2.3 权重可视化

第一层权重可视化（重新塑形为图像）：

2.4 测试结果

在测试集上的分类准确率：



最优结果在

`--model_path ./cifar10/results/best_model.pkl`

3. 结论与改进方向

3.1 结论

本项目成功实现了一个基于numpy的三层神经网络，能够有效地对CIFAR-10数据集进行图像分类。通过手动实现反向传播算法，加深了对神经网络工作原理的理解。

3.2 改进方向

1. **网络深度**：增加网络层数可能提高模型表达能力
2. **优化算法**：实现更高级的优化算法，如Adam、RMSprop等
3. **卷积层**：针对图像任务，使用卷积层可能取得更好的效果
4. **数据增强**：使用数据增强技术扩充训练集
5. **批归一化**：添加批归一化层可能提高训练稳定性和收敛速度

How to use

train

```
python cifar.py --mode train --lr 0.01 --hidden_size1 100 --hidden_size2 100 --
activation relu --batch_size 100 --epochs 200 --reg_lambda 0.01 --lr_decay --
decay_rate 0.95
```

hyperparameter_search

```
python cifar.py --mode search --search_type all --epochs 100
```

test

```
python cifar.py --mode test --model_path ./cifar10/results/best_model.pkl
```

核心组件

SGD

```
class SGD:
    def __init__(self, lr=0.01):
        self.lr = lr

    def update(self, params, grads):
        for key in params.keys():
            params[key] -= self.lr * grads[key]
```

learning rate

```
class LearningRateScheduler:
    @staticmethod
    def exponential_decay(initial_lr, epoch, decay_rate=0.95):
        return initial_lr * (decay_rate ** epoch)

    @staticmethod
    def step_decay(initial_lr, epoch, drop_rate=0.5, epochs_drop=10):
        factor = np.power(drop_rate, np.floor((1 + epoch) / epochs_drop))
        return initial_lr * factor
```

cross entropy

```
class SoftmaxWithCrossEntropy:
    def forward(self, x, t):
        self.t = t
        self.x_shape = x.shape

        # 为数值稳定性减去最大值
        x_shifted = x - np.max(x, axis=1, keepdims=True)
        exp_x = np.exp(x_shifted)
        self.y = exp_x / np.sum(exp_x, axis=1, keepdims=True)  # softmax

        if self.t.ndim == 1:
            self.t_one_hot = np.zeros_like(self.y)
            self.t_one_hot[np.arange(self.t.shape[0]), self.t] = 1
        else:
            self.t_one_hot = self.t

        # 计算交叉熵损失
        loss = -np.sum(self.t_one_hot * np.log(self.y + 1e-10)) / self.x_shape[0]
        return loss
```

L2

```
# 在loss方法中
if reg_lambda > 0:
    W1, W2, W3 = self.params['W1'], self.params['W2'], self.params['W3']
    reg_loss = 0.5 * reg_lambda * (np.sum(W1**2) + np.sum(W2**2) + np.sum(W3**2))
    loss += reg_loss

# 在backward方法中
if reg_lambda > 0:
    dW1 += reg_lambda * self.params['W1']
    dW2 += reg_lambda * self.params['W2']
    dW3 += reg_lambda * self.params['W3']
```