Lecture5-Linear Regression&Classifiers

21307035邓栩瀛

> 问题:
>
> 使用 Softmax 完成 CIFAR-10 的分类工作，在给定数据集 CIFAR-10 的训练集上训练模型， 并在测试集上验证其性能
>
> 要求:
> 1） 在给定的训练数据集上， 训练一个线性分类器（Softmax 分类器， 可以参考https://cs231n.github.io/assignments2023/assignment1/）
> 2） 手动实现 softmax_loss_naive 和 softmax_loss_vectorized 的线性分类模型

1、softmax分类器介绍

Softmax分类器是一种常用的多类别分类算法，是逻辑回归模型的扩展，可以将输入样本分为两个以上的类别，通过学习从输入特征到各个类别的概率分布，将输入样本分类到最可能的类别中，并通过对输入样本进行线性变换和指数函数的归一化来实现这一目标。

Softmax分类器的数学表达式如下：

$$P(y=j|x)=\frac{e^{x^Tw_j}}{\sum_{k=1}^{K}e^{x^Tw_k}} \tag{1}$$

其中，$P(y=j|x)$ 表示给定输入 x 来自类别 j 的概率，w_j 是与类别 j 相关的权重向量，K 是类别的总数。

Softmax分类器的训练过程通常使用最大似然估计或交叉熵损失函数来优化模型参数，同时最大化正确类别的概率，并最小化错误类别的概率。模型参数训练完成后，对于新的输入样本，Softmax分类器可以预测其属于每个类别的概率分布，并将其分类到具有最高概率的类别中。

Softmax分类器具有以下特点：

- 适用于多类别分类任务
- 输出类别的概率分布，提供了对不同类别的相对置信度
- 模型参数易于解释和理解

2、训练过程

（1）初始化方法

初始化分类器的权重、损失值历史值和准确率历史值。

```
def __init__(self):
    self.W = None
    self.loss_history = []
    self.train_acc_history = []
    self.val_acc_history = []
```

（2）超参数的选择

超参数的初始化为

```
learning_rate=1e-3 # 学习率，控制权重更新的步长
reg=1e-5 # 正则化系数，用于控制权重的正则化惩罚项
num_iterations=100 # 迭代次数
batch_size=200 # 批量大小，每次迭代时从训练集中随机选择的样本数量
```

（3）训练技巧

使用了两种不同的训练方法

```
# 向量化的训练方法，使用矩阵运算来提高计算效率
train_vec(self, X, y, learning_rate, reg, num_iterations, batch_size, verbose)
# 朴素的训练方法，使用循环逐个样本计算损失和梯度
train_naive(self, X, y, learning_rate, reg, num_iterations, batch_size, verbose)
```

在训练过程中，使用了以下技巧

- 随机小批量梯度下降：在每次迭代中，从训练集中随机选择一个小批量数据进行训练，以减小计算开销和降低梯度估计的方差
- 数值稳定性处理：在计算Softmax函数的指数部分时，减去最大值，以避免指数溢出和提高数值稳定性
- 权重更新：根据计算得到的损失函数的梯度，使用梯度下降法更新权重
- 保存训练过程数据：周期性地保存损失和准确率的历史记录，用于可视化和分析训练过程

（4）损失函数`softmax_loss_naive`和`softmax_loss_vectorized`的实现

`softmax_loss_naive`

```
    def softmax_loss_naive(self, X, y, reg):
        num_train, num_features = X.shape # 获取训练样本的数量
        num_classes = np.max(y) + 1
        loss = 0.0
        dW = np.zeros_like(self.W)

        for i in range(num_train): # 迭代每个训练样本，计算scores
            scores = X[i].dot(self.W) # 样本X[i]与权重矩阵self.W的乘积
            scores -= np.max(scores)  # 用于数值稳定性
            exp_scores = np.exp(scores)
            probs = exp_scores / np.sum(exp_scores) # 归一化后的Softmax概率向量
            correct_prob = probs[y[i]] # 正确类别的Softmax概率值
            loss += -np.log(correct_prob) # 负对数似然损失项加到总损失loss中

            for j in range(num_classes): # 对每个类别进行循环，计算参数梯度
                if j == y[i]: # 如果当前类别j与样本的真实标签y[i]相等，将误差×输入样本X[i]加到梯度中
                    dW[:, j] += (probs[j] - 1) * X[i]
                else: # 否则，将类别j的Softmax概率×输入样本X[i]加到梯度中
                    dW[:, j] += probs[j] * X[i]

        loss /= num_train # 将损失值除以训练样本数量num_train，得到平均损失
        loss += 0.5 * reg * np.sum(self.W * self.W) # 添加L2正则化项到损失中，减小权重的大小
        dW /= num_train # 将参数梯度除以训练样本数量，得到平均梯度
        dW += reg * self.W # 添加L2正则化项的梯度到参数梯度中

        return loss, dW
```

`softmax_loss_vectorized`

```
def softmax_loss_vectorized(self, X, y, reg):
        num_train = X.shape[0] # 获取训练样本的数量
        scores = X.dot(self.W) # 计算scores矩阵
        scores -= np.max(scores, axis=1, keepdims=True)  # 用于数值稳定性
        exp_scores = np.exp(scores) #  对得分矩阵应用指数函数，得到指数得分矩阵
        # 计算归一化的Softmax概率矩阵
        probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
        # 获取每个样本对应正确类别的Softmax概率
        correct_probs = probs[np.arange(num_train), y]
        # 计算每个样本的负对数似然损失
        loss = -np.log(correct_probs)
        # 计算平均损失，加上L2正则化项
        loss = np.sum(loss) / num_train + 0.5 * reg * np.sum(self.W * self.W)

        dW = probs # 将概率矩阵作为参数梯度的初始值
        dW[np.arange(num_train), y] -= 1 # 对正确类别的位置减去1
        dW = X.T.dot(dW) # 计算参数梯度
        dW /= num_train # 平均化参数梯度
        dW += reg * self.W # 加上L2正则化项的梯度

        return loss, dW
```

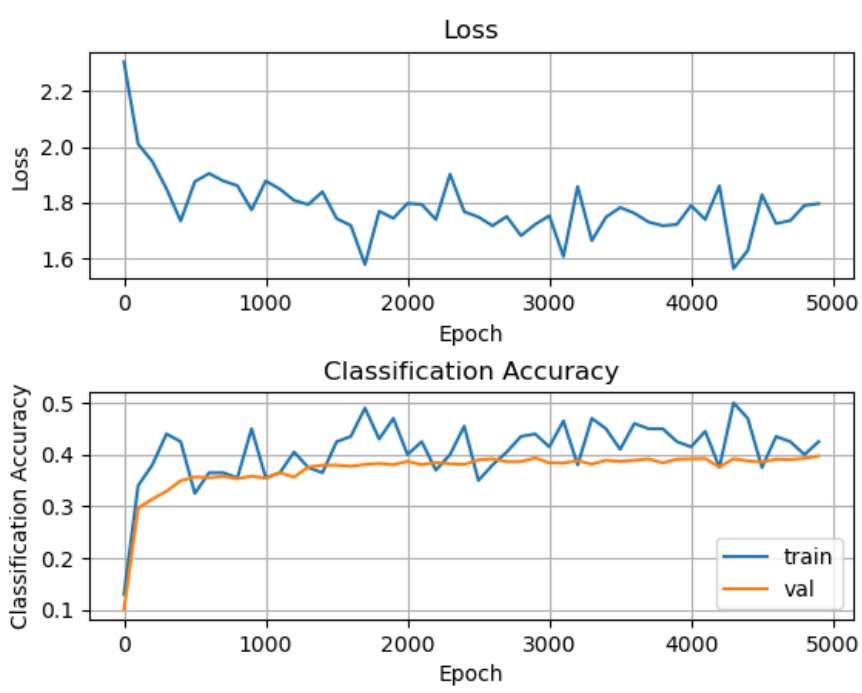
3、 实验结果及分析

实验结果

1、`learning_rate=1e-2`，`reg=1e-3`，`num_iteration=5000`，`batch_size=200`

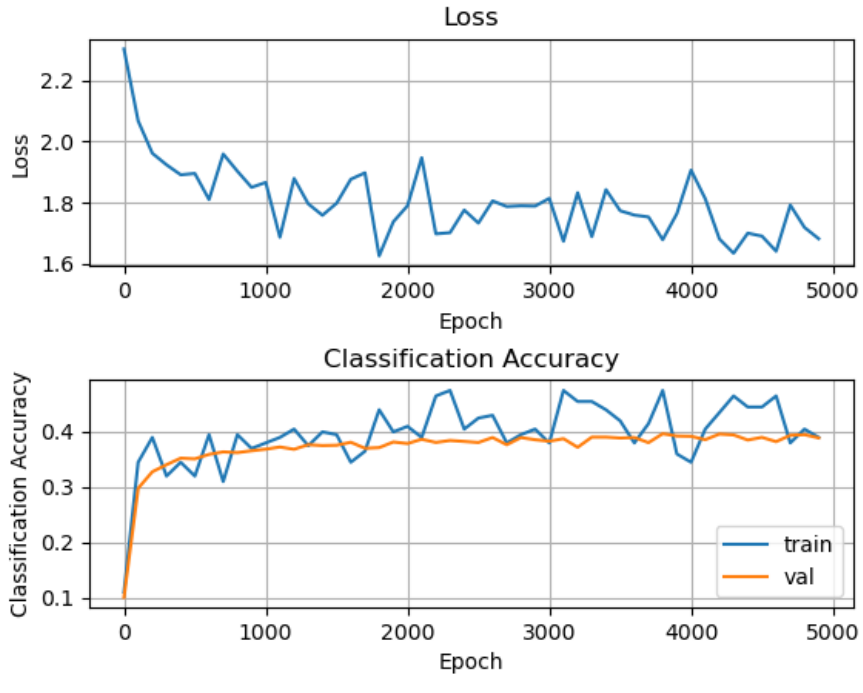
使用`softmax_loss_naive`损失函数

```
Train accuracy for naive: 0.40642
Test accuracy for naive: 0.3932
Training time for naive: 179.73367404937744 seconds
```



使用`softmax_loss_vectorized`损失函数

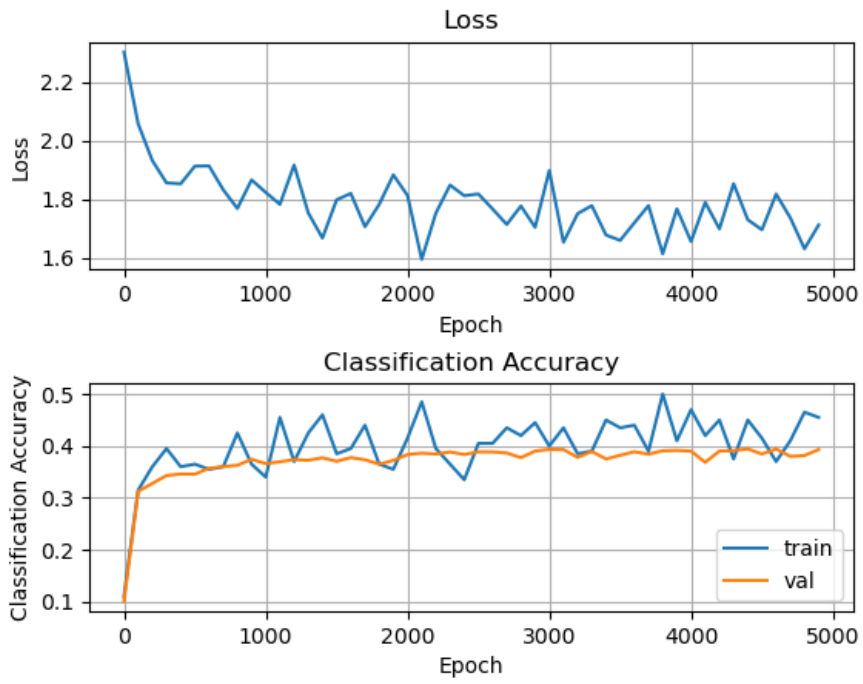
```
Train accuracy for vectorized: 0.39756
Test accuracy for vectorized: 0.3854
Training time for vectorized: 14.356752157211304 seconds
```



2、`learning_rate=1e-2`，`reg=1e-5`，`num_iteration=5000`，`batch_size=200`

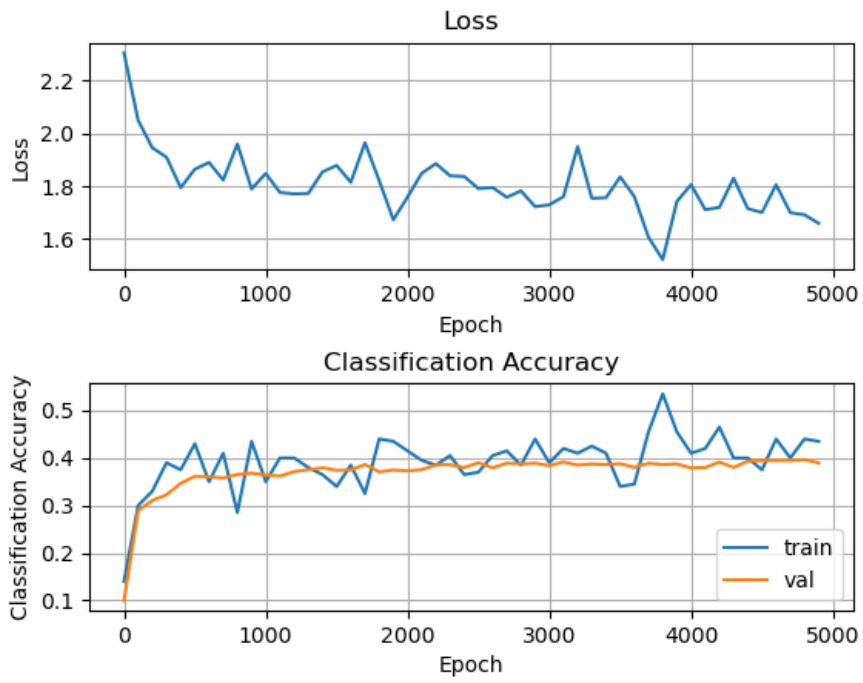
使用`softmax_loss_naive`损失函数

```
Train accuracy for naive: 0.41214
Test accuracy for naive: 0.3967
Training time for naive: 175.33598494529724 seconds
```



使用`softmax loss vectorized`损失函数

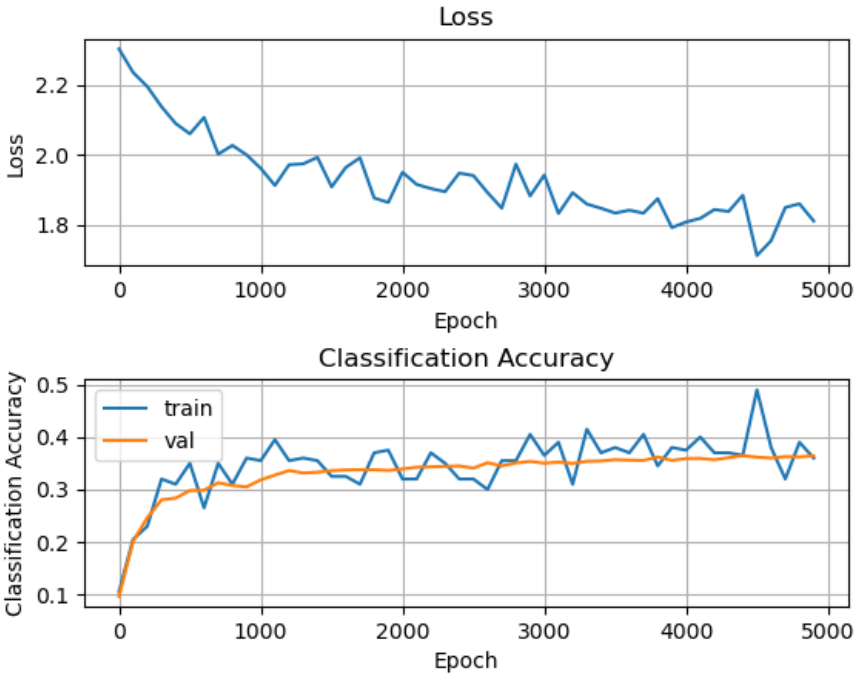
```
Train accuracy for vectorized: 0.4067
Test accuracy for vectorized: 0.3898
Training time for vectorized: 12.085041999816895 seconds
```



3、`learning_rate=1e-3`，`reg=1e-5`，`num_iteration=5000`，`batch_size=200`

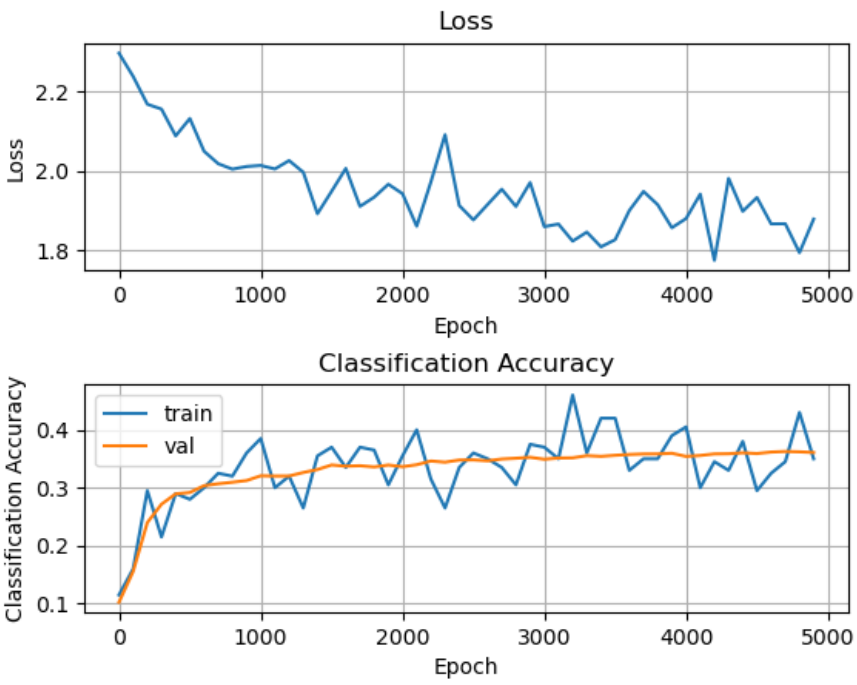
使用`softmax_loss_naive`损失函数

```
Train accuracy for naive: 0.36592
Test accuracy for naive: 0.3648
Training time for naive: 173.78056693077087 seconds
```



使用`softmax_loss_vectorized`损失函数

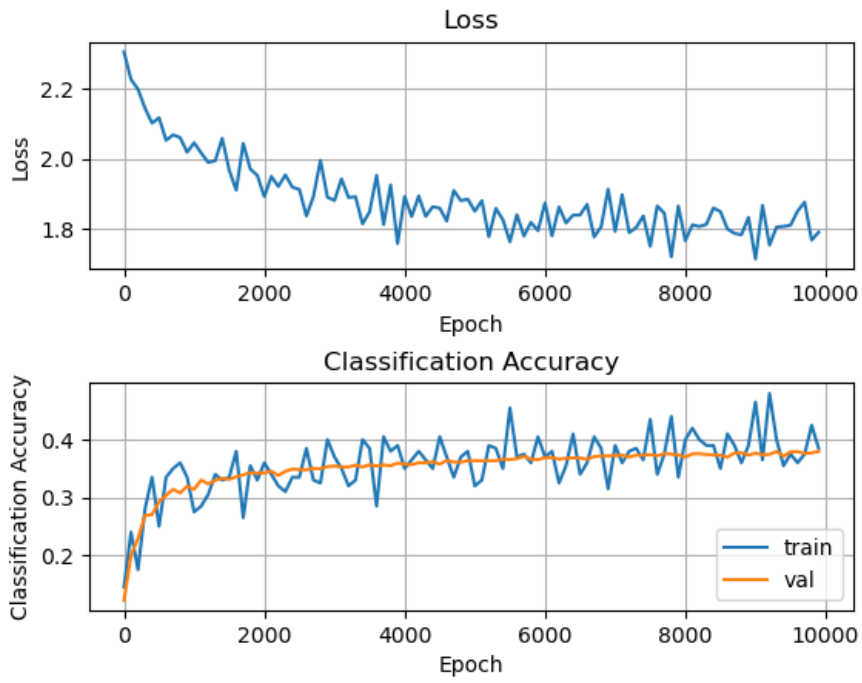
```
Train accuracy for vectorized: 0.36782
Test accuracy for vectorized: 0.3631
Training time for vectorized: 12.053623914718628 seconds
```



4、`learning_rate=1e-3`，`reg=1e-5`，`num_iteration=10000`，`batch_size=200`

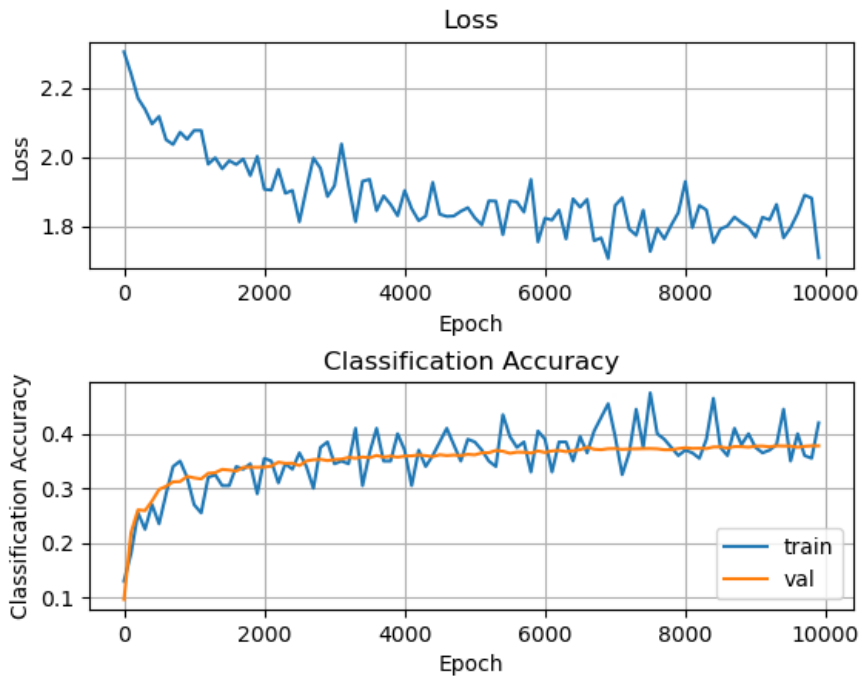
使用`softmax_loss_naive`损失函数

```
Train accuracy for naive: 0.3826
Test accuracy for naive: 0.3764
Training time for naive: 355.75747776031494 seconds
```



使用`softmax_loss_vectorized`损失函数

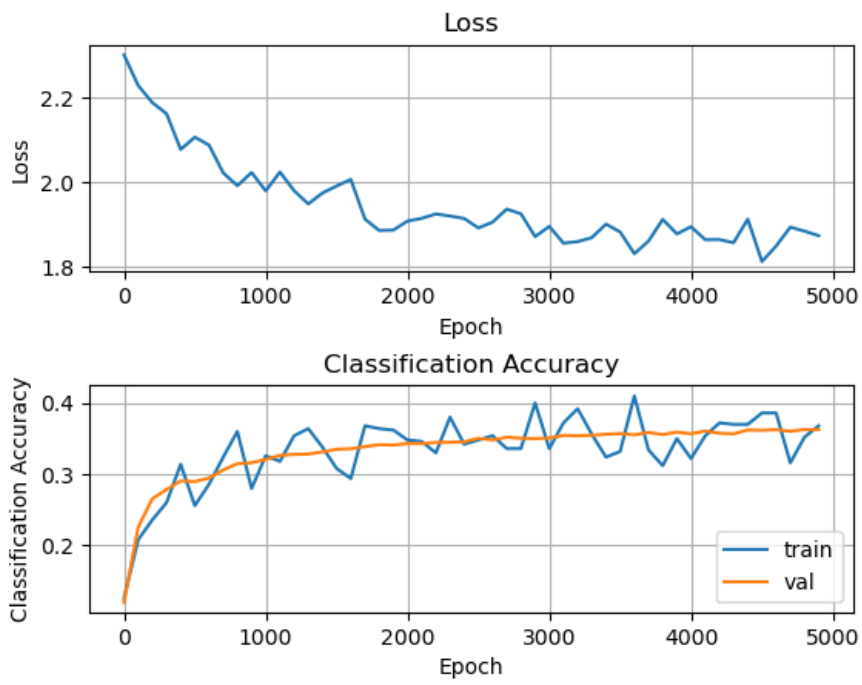
```
Train accuracy for vectorized: 0.38196
Test accuracy for vectorized: 0.3782
Training time for vectorized: 23.514239072799683 seconds
```



5、`learning_rate=1e-3`，`reg=1e-5`，`num_iteration=5000`，`batch_size=500`

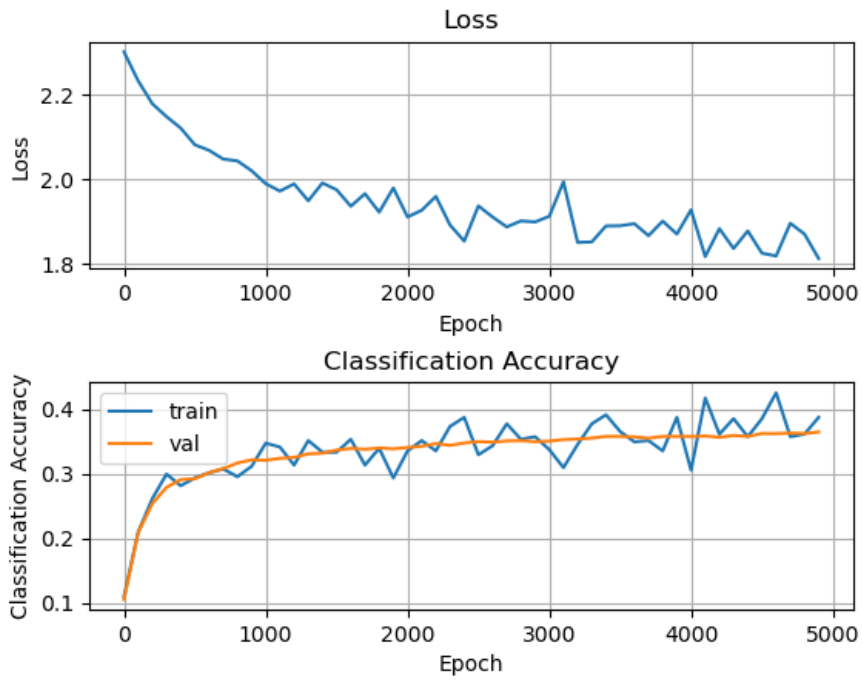
使用`softmax_loss_naive`损失函数

```
Train accuracy for naive: 0.37008
Test accuracy for naive: 0.366
Training time for naive: 449.4057836532593 seconds
```



使用`softmax_loss_vectorized`损失函数

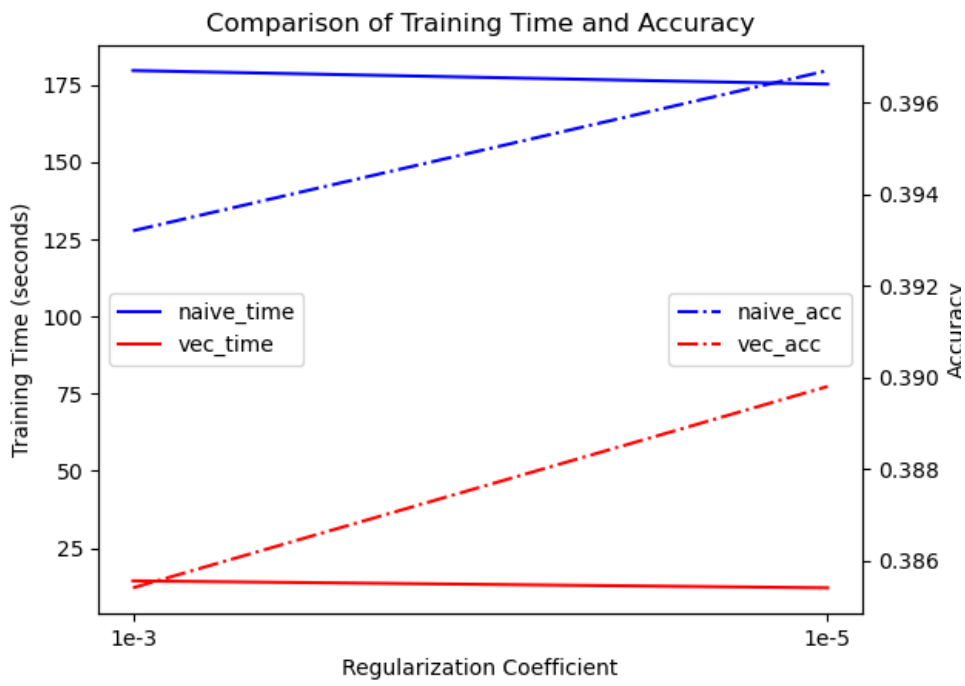
```
Train accuracy for vectorized: 0.37076
Test accuracy for vectorized: 0.3668
Training time for vectorized: 23.33294677734375 seconds
```



结果分析

1、正则化系数

`reg=1e-3`和`reg=1e-5`

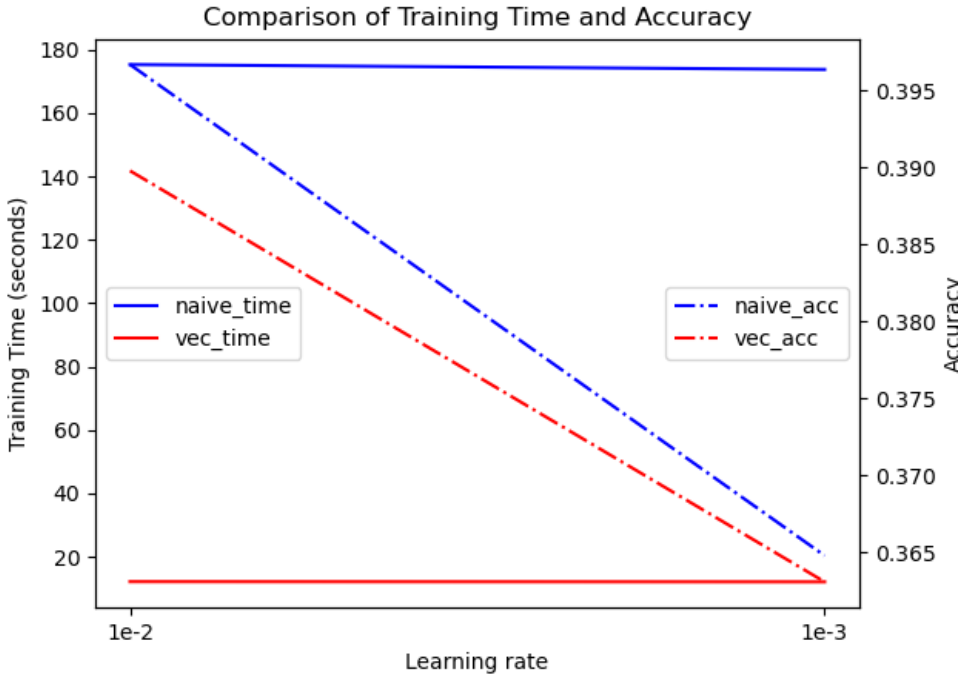


训练时间：`reg=1e-5`要略优于`reg=1e-3`的情况

验证集正确率：`reg=1e-5`的正确率略高于`reg=1e-3`的正确率

2、学习率

`learning_rate=1e-2`和`learning_rate=1e-3`



训练时间：在相同的损失函数下，时间几乎相同

损失函数值：`learning_rate=1e-3`的损失函数值更大

验证集正确率：`learning_rate=1e-3`的正确率要低于`learning_rate=1e-2`

3、迭代次数

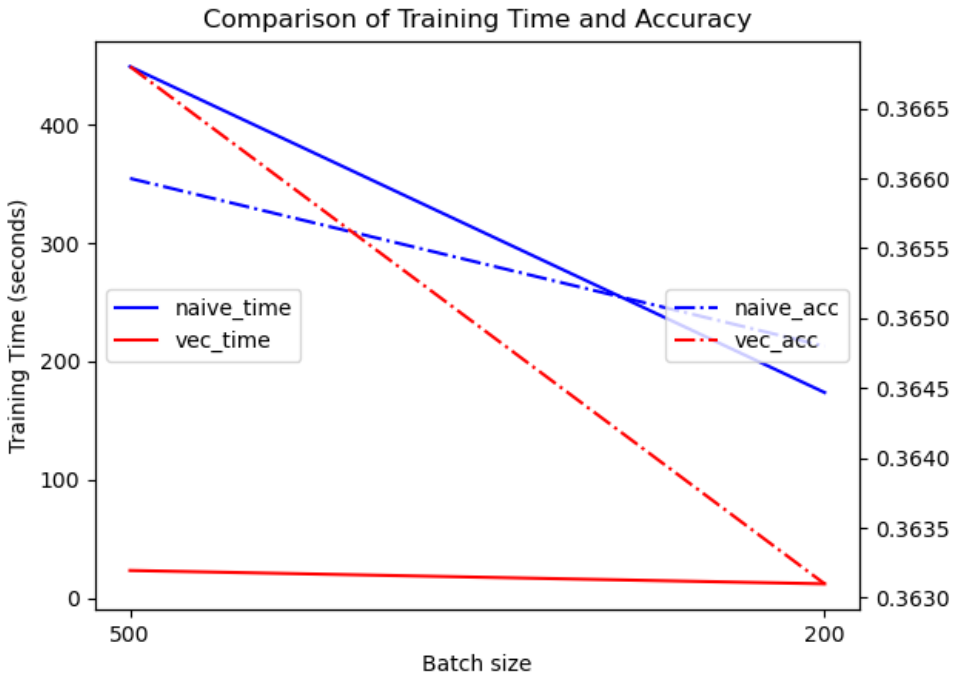
`num_iteration=10000`和`num_iteration=5000`

训练时间：时间成倍增加

验证集正确率：随着迭代次数的增加，正确率略有提升，但迭代次数到达一定程度后，增加的效果不明显

4、批量大小

`batch_size=200`和`batch_size=500`



训练时间：训练时间成倍增加

验证集正确率：随着每次训练的批量大小增加，验证集的正确率略有提高，但是提升幅度很小

4、实验总结

（1）`softmax_loss_naive`与`softmax_loss_vectorized`的比较

`softmax_loss_naive`：通过循环遍历样本和类别来计算softmax损失，对于每个样本，都需要进行一次循环来计算softmax激活和损失，以及对应的梯度。实现相对简单，易于理解，但由于使用了循环，它的计算效率较低，在处理大规模数据集时，会导致较长的计算时间。

`softmax_loss_vectorized`：通过矩阵运算来计算softmax损失，利用矩阵运算的高效性，可以同时处理多个样本和类别，从而提高计算效率。相比于naive方法，它的计算速度更快。通过适当的矩阵操作和广播，可以避免显式的循环，并在能够单次计算中处理多个样本和类别。

从实验结果也可以看出来，`softmax_loss_naive`与`softmax_loss_vectorized`在损失函数值计算以及正确率方面几乎没有什么区别，而在训练时间上，使用`softmax_loss_naive`的训练时间几乎为使用`softmax_loss_vectorized`的时间的14~15倍，当数据规模以及迭代次数更多的时候，`softmax_loss_vectorized`的优势将更加明显，是一种更加高效的方法。

（2）超参数

①在本实验中，学习率（步长）对正确率的影响相对更大，同时，学习率的选择也要结合迭代次数进行分析

②当迭代次数较少时（<1000），正确率会随着迭代次数的增加而增加，然而，当迭代次数增加到一定程度以后，正确率的值基本上稳定在一个范围，增幅非常有限

③适当的正则化可以提高模型的泛化能力，通过控制模型的复杂度，正则化可以使模型能够更好地捕捉数据中的一般规律，而不是过度拟合训练数据的特殊性。

④较大的`batch_size`可以提高训练速度和稳定性，但可能需要更多的内存，而较小的`batch_size`可以提高模型泛化能力，但可能需要更多的训练迭代才能收敛。在实际应用中，需要根据训练过程中的性能指标选择最佳的批大小取值。