计算机视觉第8次作业报告--基于两层神经网络的图像分类器

人工智能82班 刘志成 2183511589

相关代码已上传至我的github仓库：
https://github.com/zchliu/2020-Fall

实验题目

练习实习一个基于两层神经网络的简单图像分类器，具体目标：

1. 理解基本图像分类过程（数据驱动的训练/预测阶段）
2. 理解将数据划分为训练/验证/测试集的意义以及使用验证集来对超参数进行微调
3. 熟练使用numpy 编写高效的矢量化代码。
4. 实现并应用一个两层神经网络

实验过程

1. 在neural_net.py line 83 实现前向传播（forward pass），运行python two_layer_nnet.py

代码如下：

```
fc1 = X.dot(W1) + b1
X2 = np.maximum(0, fc1)
scores = X2.dot(W2) + b2
```

输出的scores和正确的scores以及误差如下：

```
Your scores:
[[-0.81233741 -1.27654624 -0.70335995]
 [-0.17129677 -1.18803311 -0.47310444]
 [-0.51590475 -1.01354314 -0.8504215 ]
 [-0.15419291 -0.48629638 -0.52901952]
 [-0.00618733 -0.12435261 -0.15226949]]

correct scores:
[[-0.81233741 -1.27654624 -0.70335995]
 [-0.17129677 -1.18803311 -0.47310444]
 [-0.51590475 -1.01354314 -0.8504215 ]
 [-0.15419291 -0.48629638 -0.52901952]
 [-0.00618733 -0.12435261 -0.15226949]]

Difference between your scores and correct scores:
3.6802720745909845e-08
```

2. 在neural_net.py line 101 实现损失计算（loss function），运行python two_layer_nnet.pyd

代码如下：

```
scores -= np.max(scores, axis=1, keepdims=True)
scores_exp = np.exp(scores)
softmax_matrix = scores_exp / np.sum(scores_exp, axis=1, keepdims=True)
loss = np.sum(-np.log(softmax_matrix[np.arange(N), y]))
loss /= N
loss += reg * (np.sum(W2 * W2) + np.sum(W1 * W1))
```

实验结果如下：

```
Difference between your loss and correct loss:
1.7985612998927536e-13
```

3. 在neural_net.py line 114 实现梯度计算（compute gradients），运行python two_layer_nnet.py

代码如下：

```
softmax_matrix[np.arange(N), y] -= 1
softmax_matrix /= N
dW2 = X2.T.dot(softmax_matrix)
db2 = softmax_matrix.sum(axis=0)
dW1 = softmax_matrix.dot(W2.T)
dfc1 = dW1 * (fc1 > 0)
dW1 = X.T.dot(dfc1)
db1 = dfc1.sum(axis=0)
dW1 += reg * 2 * W1
dW2 += reg * 2 * W2
grads = {'W1': dW1, 'b1': db1, 'W2': dW2, 'b2': db2}
```

实验结果如下：

```
W1 max relative error: 3.561318e-09
b1 max relative error: 2.738421e-09
W2 max relative error: 3.440708e-09
b2 max relative error: 4.447625e-11
```

4. 在neural_net.py line 159 实现产生一个batch 的数据-标签对，同时在neural_net.py line 175 实现梯度反向传播（backward pass）更新权重

代码如下：

```
loss, grads = self.loss(X_batch, y=y_batch, reg=reg)
loss_history.append(loss)
```

实验结果如下：

```
Final training loss:  0.017149607938732093
```

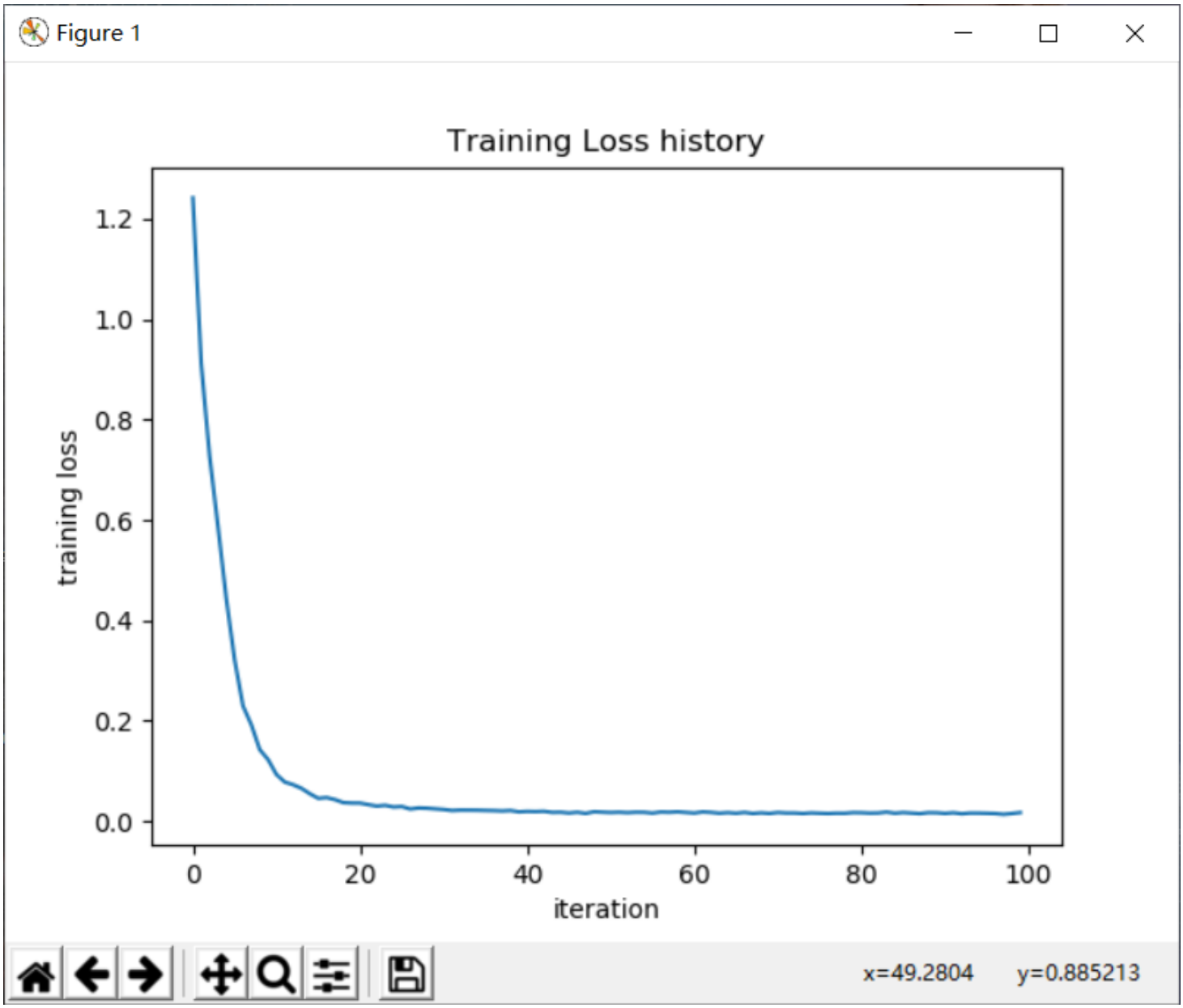
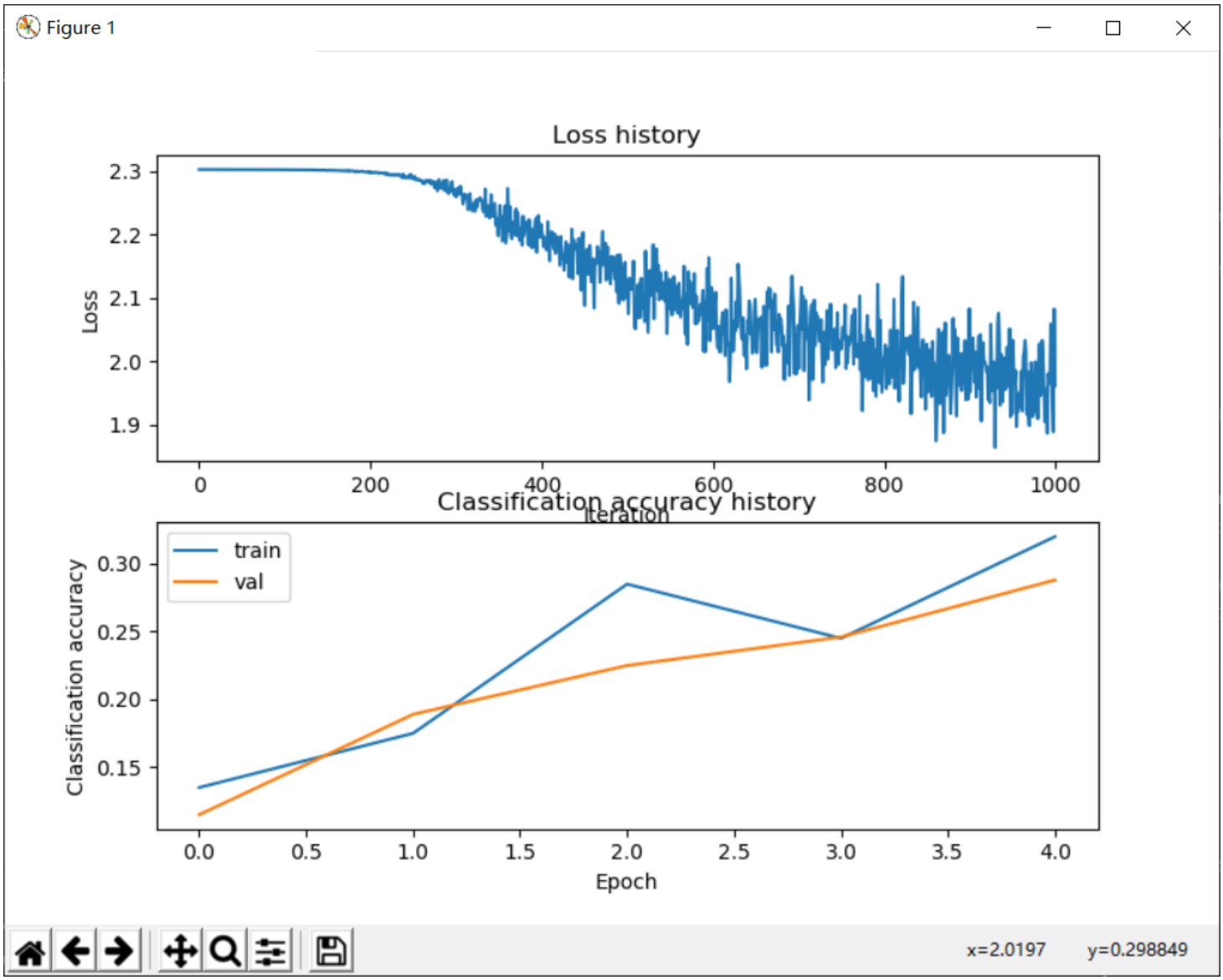


Figure 1
Training Loss history
training loss
iteration
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
x=49.2804
y=0.885213

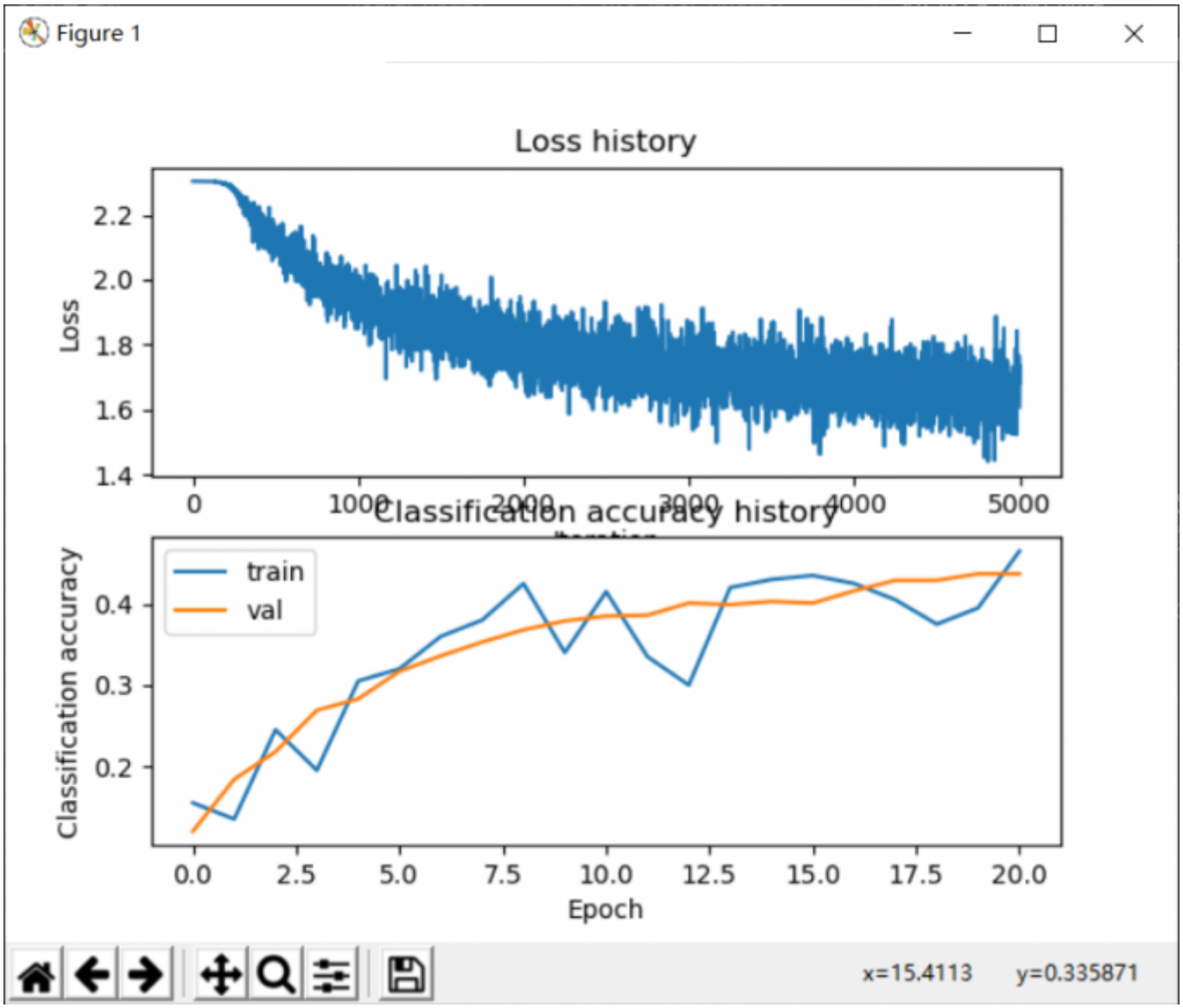


-->

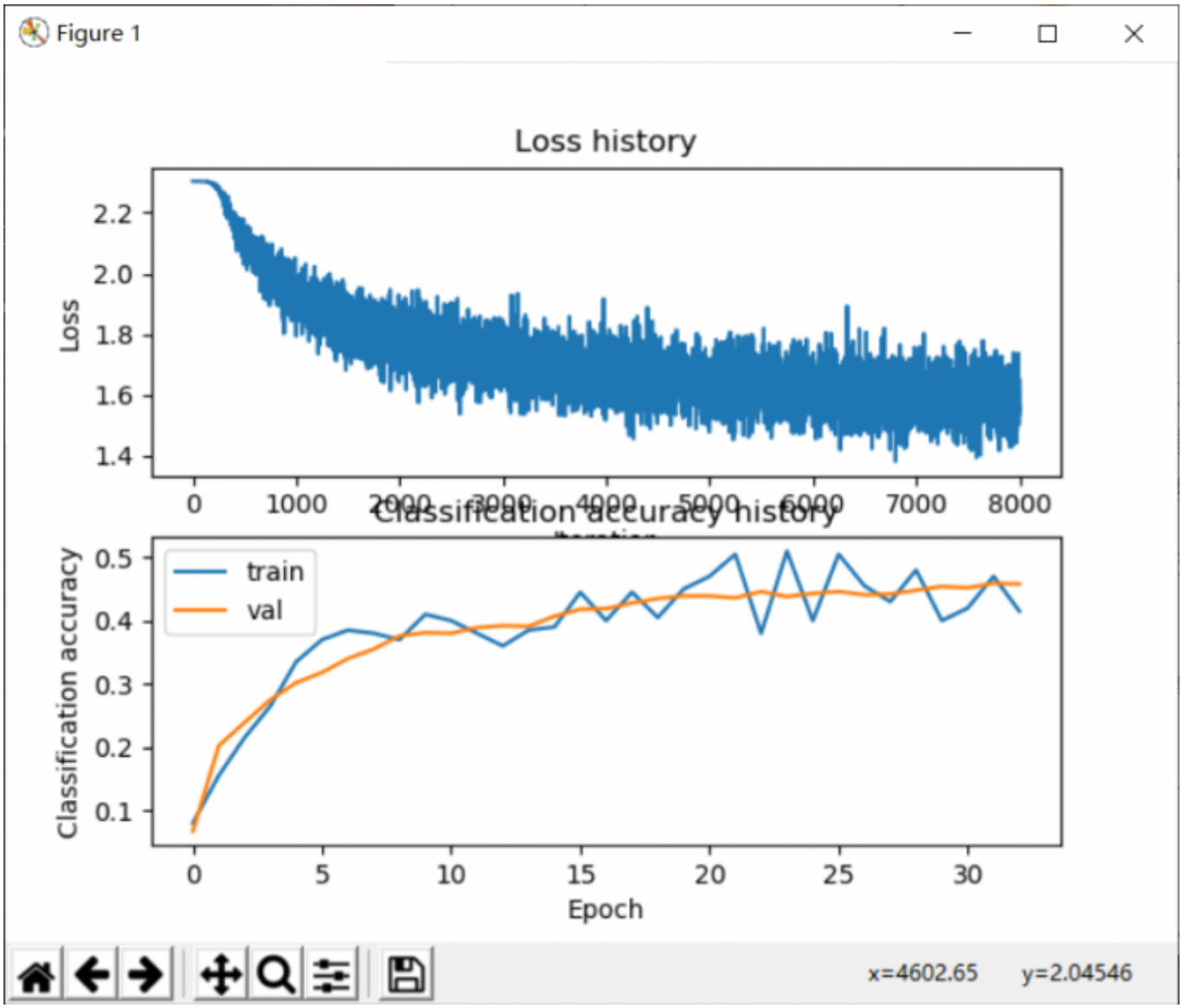
1. 观察可视化出的损失变化以及权重，分析目前在cifar10 验证集上正确率较低的原因。尝试调整超参数（神经网络隐层维度，学习率，训练时的迭代次数，正则化系数等），将模型在验证集上的正确率提升至≥48%。请在报告中解释在调整超参数过程中，模型的正确率，训练速度随之产生的变化

由图可以看出，在迭代过程中训练集和验证集的正确率都在上升，因此可以得出在cifar10上正确率较低的原因是模型欠拟合。

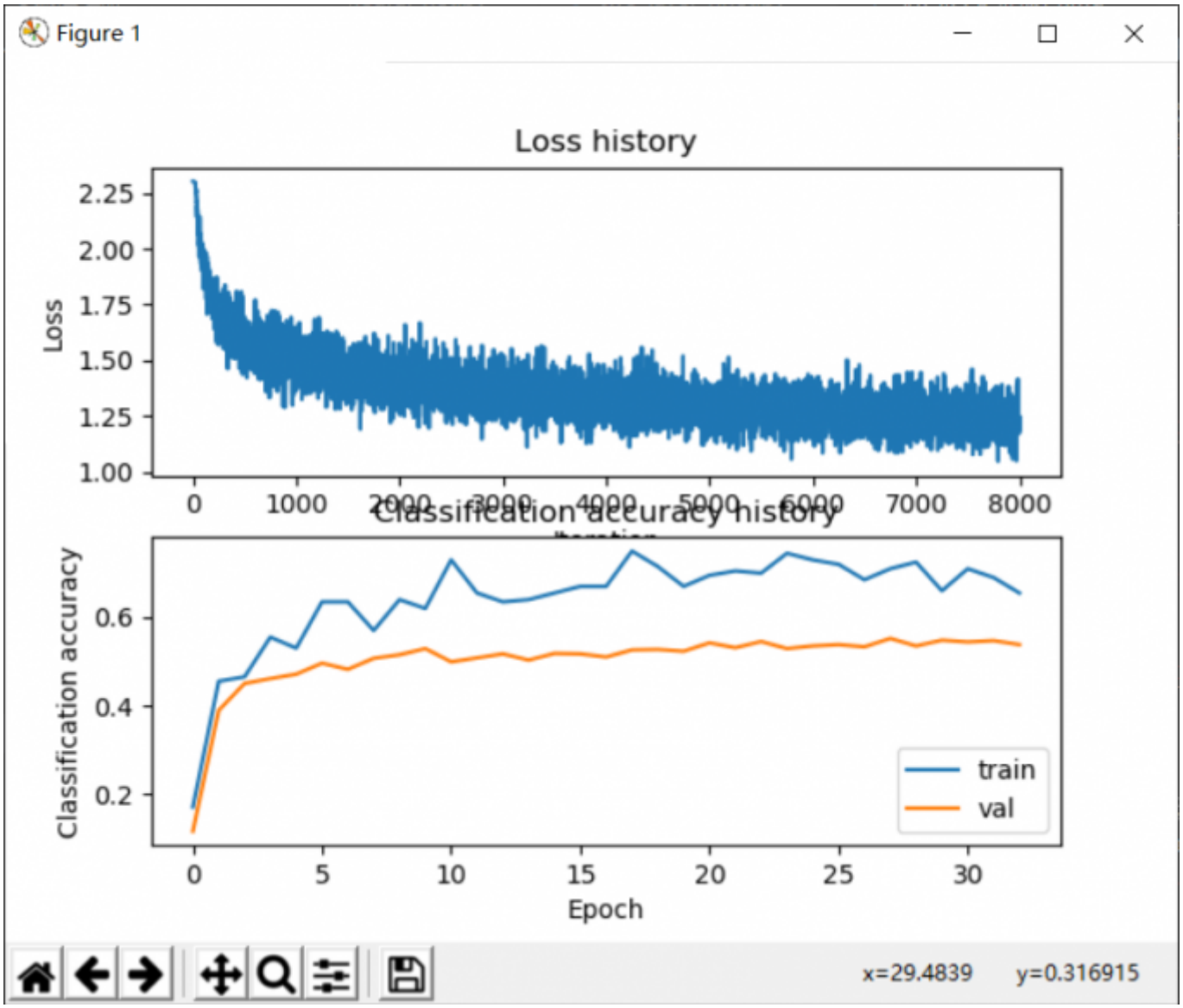
将隐层单元数增加到100，迭代数目增加到5000后可以得到正确率在0.442，如下图，可以看到模型依然处于欠拟合状态。



将隐层单元数增加到120，迭代数目增加到8000后得到的结果，正确率为0.457。



图中可以看出在后面迭代的过程中正确率增加的不是很多，因此我们尝试增大学习率在前几轮模型迭代的过程中加快速度，学习率增加为1e-3，隐层数和迭代数为120和8000保持不变，得到结果如下图，验证集正确率为0.535，达到了题目要求。



在调整参数的过程中，训练时间随着batch_size、hidden_size和num_iters的增加而增加，训练的正确率随着训练的轮数增加而增加。

总结

尽管只有一个隐藏层，但是借助这个例子对于神经网络的一些基本调参的技巧都能有所了解，在本例中给的是一个欠拟合的模型，因此可以考虑增加隐层单元的数目，增大学习率，增加训练轮数等等提高模型在验证集上的准确率。由于本例是一个欠拟合的模型，可以不用调整正则化参数，如果模型过拟合的话可以适当增加正则化的参数。