MNIST Classification with Softmax

何东阳 2019011462

实验内容

- 实现一个简单的数字识别器
- 达到较高的识别率

实验公式

$$N = \text{batch\_size}$$
$$K = \text{dimension}$$
$$o_i = w^T x + b$$
$$y_i = \frac{exp(o_i)}{\sum_{i=1}^{N} exp(o_i)}$$
$$\text{coss} = -\frac{1}{N}\sum_{n=1}^{N}\sum_{j=1}^{K} t_j^{(n)} \ln h(o_j)^{(n)}$$
$$\text{gradient} = \frac{1}{N}\sum_{n=1}^{N}(y^{(n)} - t^{(n)})(x^{(n)})^T$$

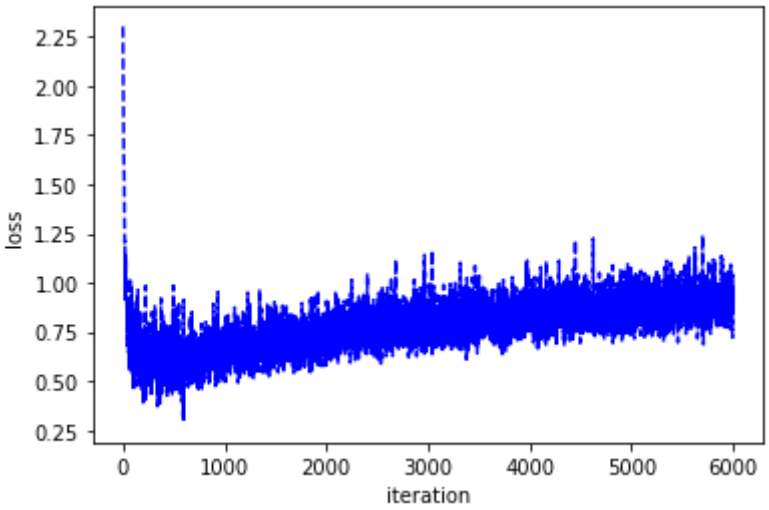
关键代码

```
def softmax_classifier(W, input, label, lamda):
    """
      Softmax Classifier
```

```
        Inputs have dimension D, there are C classes, a minibatch
have N examples.
        (In this homework, D = 784, C = 10)

        Inputs:
        - W: A numpy array of shape (D, C) containing weights.
        - input: A numpy array of shape (N, D) containing a
minibatch of data.
        - label: A numpy array of shape (N, C) containing labels,
label[i] is a
          one-hot vector, label[i][j]=1 means i-th example belong
to j-th class.
        - lamda: regularization strength, which is a hyerparameter.

        Returns:
        - loss: a single float number represents the average loss
over the minibatch.
        - gradient: shape (D, C), represents the gradient with
respect to weights W.
        - prediction: shape (N, 1), prediction[i]=c means i-th
example belong to c-th class.
    """
    # TODO: 还需要加入正则化lambda
    N = input.shape[0]
    b = np.ones((N, W.shape[1]))  # (N, C)
    o = np.dot(input, W) + b  # (N, C)
    y = np.exp(o) / np.sum(np.exp(o), 1, keepdims=True)  # (N, C)
    prediction = np.argmax(y, 1).reshape(N, 1)  # (N, 1)
    loss = -(1 / N) * np.sum(label * np.log(y)) + 1 / N * lamda /
2 * np.sum(np.dot(W, W.T))  # (int) TODO: 还可以优化
    gradient = 1 / N * np.dot(
        (y - label).T, input).T + 1 / N * lamda * W  # (N, C) *
(N, D) ->(D, C)

    return loss, gradient, prediction
```

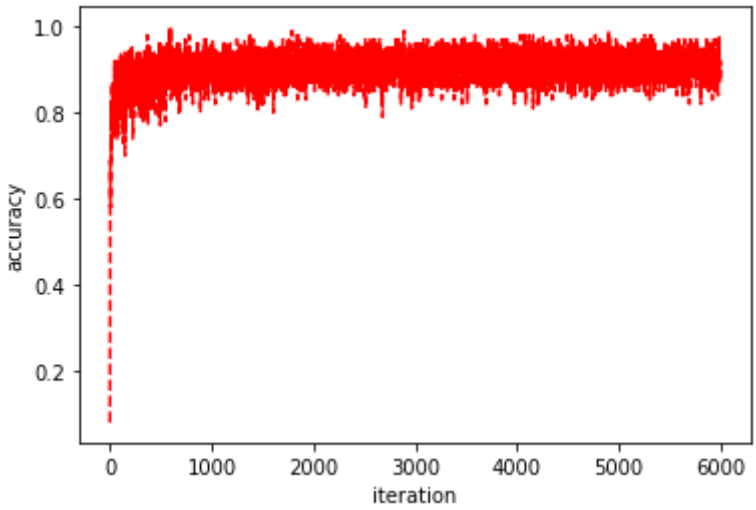
实验结果

训练损失图线



- 可以看出，训练集的损失随着迭代轮数增加逐渐减少，最终在0.7至1之间收敛稳定

识别准确率曲线



- 可以看出随着迭代轮数增加，识别准确率逐渐上升，最终收敛至0.9附近

参数设置

- 设至不同的超参数，实验结果也会有所不同，因此我尝试设置不同参数，记录了测试集准确率和训练集损失最终收敛的范围

比较learning_rate的影响

实验组数	batch_size	max_epoch	learning_rate	lamda	test accuracy	train loss
1	100	10	0.1	0.2	0.9162	0.8-0.9
2	100	10	0.5	0.2	0.9096	0.95-1.05
3	100	10	0.05	0.2	0.9156	0.7-0.9
4	100	10	0.01	0.2	0.9012	0.6-0.8

- 可以看出，学习率在0.1左右时，平均损失比较小，此时的测试集识别率也相对较高。

比较lamda的影响

实验组数	batch_size	max_epoch	learning_rate	lamda	test accuracy	train loss
1	100	10	0.01	0.2	0.9012	0.6-0.8
2	100	10	0.01	0.5	0.9005	0.7-0.8
3	100	10	0.01	0.5	0.9001	0.75-0.85
4	100	10	0.01	0.05	0.9019	0.4-0.6

- 可以看出，lamda为0.05左右时误差较小识别率也较高

比较batch_size的影响

实验组数	batch_size	max_epoch	learning_rate	lamda	test accuracy	train loss
1	100	10	0.1	0.05	0.9198	0.4-0.6
2	200	10	0.1	0.05	0.9164	0.36-0.45
3	500	10	0.1	0.05	0.9095	0.3-0.41
4	50	10	0.1	0.05	0.9209	0.8-0.9

- 可以看到，当batch_size为50时，识别准确率上升了，但训练集的损失是比较大的
- 综上我认为batch_size为100的情况是比较好的

比较max_epoch的影响

实验组数	batch_size	max_epoch	learning_rate	lamda	test accuracy	train loss
1	100	10	0.1	0.05	0.9198	0.4-0.6
2	100	20	0.1	0.05	0.9199	0.5-0.6/td>
3	100	50	0.1	0.05	0.9217	0.6-0.7
4	100	5	0.1	0.05	0.9164	0.45-0.6

- 可以看出，随着训练轮数的增加，测试集识别率逐渐上升，但是论述增加训练的时间也变长了

- 我认为最大轮数max_expoch为10是相对高效的

实验总结

- 根据实验结果可以得出超参数为下列时，实验效果是相对较好的

```
batch_size = 100
max_epoch = 10
learning_rate = 0.1

# For regularization
lamda = 0.05
```

- 本次实验相对简单，主要在矩阵运算时出了一点小问题（对各个变量的维度不太熟悉），完成本次作业后我对softmax分类算法的理解更深入了一层，代码能力也有所提升，收获很大。