计算机科学与技术学院神经网络与深度学习课程实验报告

实验题目：Softmax and three-layer-network		学号：201600181058
日期：2019-3-20	班级：16 人工智能班	姓名： 张多
Email：976539567@qq.com		

实验目的：
Complete the code and build a softmax classifier and a three-layer-network to classify the ten classes of object in CIFAR-10 dataset

实验软件和硬件环境：
Python 3.6
ThinkPad X1Carbon 8G+256G

Experiment Principles and methods:

1. To implement a neuron network, the very first thing is to initialize all kinds of parameters, especially the weights and bias on every layer. The common initialization method is to set the weights randomly(sometimes set on a gaussian distribution) and set the bias to zero.
2. Now we need to implement the forward pass. The main process is to calculate the output values after several layers and activations.
3. Soft_max(just focus on ***one*** sample):
   When doing the classification mission on the output layer, the softmax function converts the values of the outputs to the exponential probabilities, where the maximum of the probability stands for the predicted class C.
   The soft_max function is:
   $$\mathrm{p_i} = \frac{e^{scores_i}}{\sum_k e^{scores_k}}$$
4. This step needs to do the back propagation inversely according to the loss function and the structure or architecture of the network.
5. Soft_max Loss Function(just focus on ***one*** sample):
   $$\mathrm{loss} = -\frac{1}{\mathrm{N}}\sum_{\mathrm{i}} y_i \log p_i + \frac{1}{2N} reg \sum_L \|W_L\|_2^2$$
   The first part of the right side of the equation is called cross-entropy loss function. It is widely utilized in classification missions. y_{i} is not the true output value of a integer between 1 to *C*. It's a onehot vector or the indicator function to demonstrate which of the classes supposed to be right, and the right place of the vector is 1 while others are 0. The indicator function can be detailed like this:
   $$\mathrm{y\{i = right\ place\} = 1, y\{others\} = 0}$$
   Thus we have the loss function, we could calculate the loss function's derivative on weights and bias.
   The main principle of the derivation process is the chain rule. I'll demonstrate it below:

$$\frac{\partial \mathrm{L}}{\partial \mathrm{w_i}} = \frac{\partial \mathrm{L}}{\partial \mathrm{p_i}} \frac{\partial \mathrm{p_i}}{\partial \mathrm{score_i}} \frac{\partial \mathrm{score_i}}{\partial \mathrm{w_i}} \tag{1}$$

$$\frac{\partial \mathrm{L}}{\partial b_{\mathrm{i}}} = \frac{\partial \mathrm{L}}{\partial \mathrm{p_i}} \frac{\partial \mathrm{p_i}}{\partial \mathrm{score_i}} \frac{\partial score_i}{\partial b_{\mathrm{i}}} \tag{2}$$

As we can see, both in Eq(1) and Eq(2) have a similar part $\frac{\partial \mathrm{L}}{\partial \mathrm{p_i}} \frac{\partial \mathrm{p_i}}{\partial \mathrm{score_i}}$, which can be considered as the product between the partial derivative of loss or error at present layer with respect to the output after the activation and the derivative of the output after the activation with respect to the result before the activation.

Now we give this part a notation of $\boldsymbol{\delta}$, and the $\boldsymbol{\delta}$ on every layer can be calculated iteratively with the equation derived by the chain rule:

$$\delta_{\mathrm{l}} = (w_{l+1}^T \delta_{l+1}) \odot \sigma'(z_l) \tag{3}$$

where $\mathrm{z_l}$ is the result before activation and $\boldsymbol{l}$ is the index of the layer. (Eq(3) aims at vectorized formulation of a batch of samples instead of just ***one*** sample.

Next the rest part of Eq(1) and Eq(2) is very easy to calculate.

$$\frac{\partial \mathrm{score_i}}{\partial \mathrm{w_i}} = x_i \tag{4}$$

$$\frac{\partial score_i}{\partial b_{\mathrm{i}}} = 1 \tag{5}$$

Also, Eq(4) and Eq(5) can be generalized to the vectorized formulation and the final result is:

$$\frac{\partial \mathrm{L}}{\partial \mathrm{w}_l} = output_{afterActivation}\, \delta_{l+1}$$

$$\frac{\partial \mathrm{L}}{\partial b_{\mathrm{i}}} = \delta_{l+1}$$

Finally, the back propagation is completely done. Now we are heading for the weights and bias updating process.

6. The Gradient Descent

   The weights and bias updating process is very simple:

$$\mathrm{weight_{new}} = weight_{old} - LearingRate \times \frac{\partial L}{\partial weight_{old}}$$

$$\mathrm{bias_{new}} = bias_{olg} - LearingRate \times \frac{\partial L}{\partial bias_{old}}$$

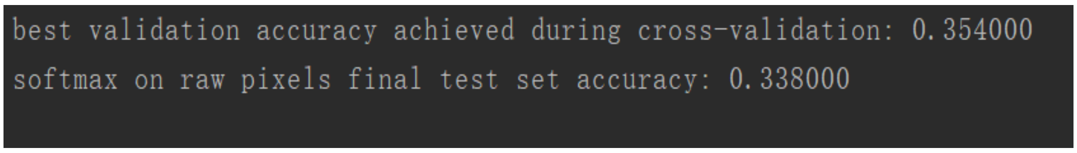
   Attention: When we implementing gradient descent, we usually use the mini-batch stochastic gradient descent (samples of batch size), not SGD (one sample in one iteration), not batch gradient descent (all samples included).

实验步骤：（不要求罗列完整源代码）

1. Complete the code snippets.
2. Load the CIFAR-10 dataset and preprocess the data into proper format.
3. Train the soft max classifier and the three-layer-network and test the result.
4. Tune the hyperparameters and optimize the network.

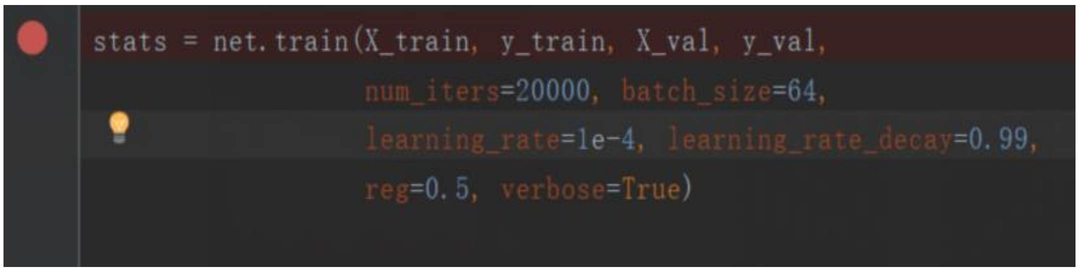
结论分析与体会：

1. At first, the soft max classifier and the three-layer-network was lead to a very bad performance as a consequence of the parameter setting.
2. After the parameters adjusting, the test accuracy gets refined and finally the soft max gets an accuracy of



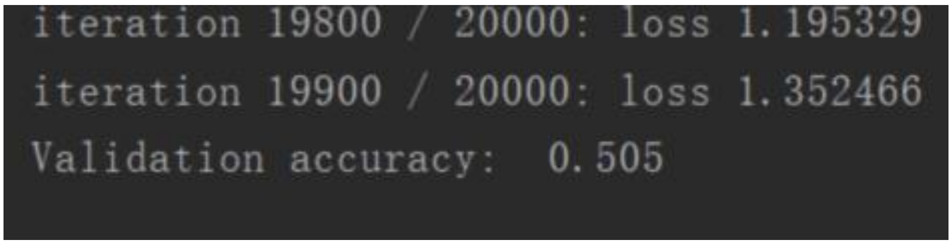
```
best validation accuracy achieved during cross-validation: 0.354000
softmax on raw pixels final test set accuracy: 0.338000
```

3. The three-layer get a best net with parameters:



```
stats = net.train(X_train, y_train, X_val, y_val,
                  num_iters=20000, batch_size=64,
                  learning_rate=1e-4, learning_rate_decay=0.99,
                  reg=0.5, verbose=True)
```

And the final test accuracy is:



```
iteration 19800 / 20000: loss 1.195329
iteration 19900 / 20000: loss 1.352466
Validation accuracy:  0.505
```

就实验过程中遇到和出现的问题，你是如何解决和处理的，自拟 1—3 道问答题：

The biggest challenge I've met is about how to set the training parameters. At fist the loss function value never decreased and sometimes even increased. After reading so many references I got a clue of the method to adjust the parameters and after tuning the hyperparameters I got a perfect result.