DL Assignment 1

Meinan Gou meinang@kth.se

1 Introduction

In this assignment, I used Python to train and test a one layer network with multiple outputs to classify images from the CIFAR-10 dataset. The network was trained using mini-batch gradient descent applied to a cost function that computes the cross-entropy loss of the classifier applied to the labelled training data and an L2 regularization term on the weight matrix.

2 Methodology

The Python file contains the following functions:

- LoadBatch(): I imported from the provided file.
- initialize_paras(): Initialize W and b given mean and standard deviation values.
- EvaluateClassifier(): Returns the result after softmax($Wx + b$), where softmax() is imported from the provided file.
- one_hot(): Computes the one-hot matrix of the ground truth labels.
- ComputeCrossEntropy(): Computes the cross-entropy l between the ground truth labels and computed outputs.
- ComputeCost(): The sum of the cross-entropy and the regularization.
- ComputeAccuracy(): Computes the ratio of correctly classified samples.
- ComputeGradients(): Computes gradients analytically.
- ComputeGradientsSlow(): Computes gradients numerically using $f'(x) = \frac{f(x+h)-f(x-h)}{2h}$, where h is a very small number (1e-6).
- MiniBatchGD(): Computes and stores the gradients, the costs and the accuracy at each epoch and returns them.
- relative_error(): Computes the relative error of two arrays with the formula on the instruction.
- montage(): Visualizes W. Modified and imported from the provided file.

3 Results

3.1 Comparison of computing gradients analytically and numerically

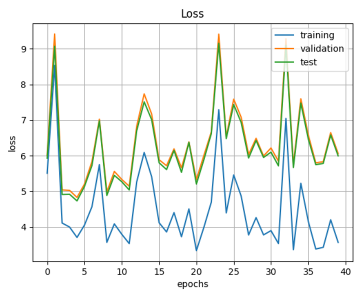
I used the formula

$$\frac{|g_a - g_n|}{max(eps, |g_a| + |g_n|)} \tag{1}$$

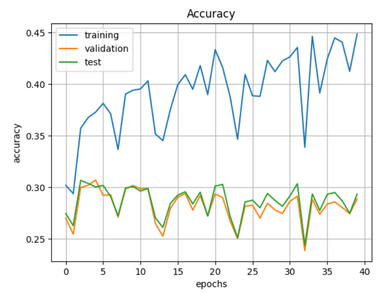
where I set *eps* to 1e-8 to compute the relative error between analytical and numerical results. I used the first 20 samples in the training data for calculation and the function relative_error() to compute the result. The maximum relative error in both W and b is at the level of e-7, which satisfies the criterion.

3.2 Training progress

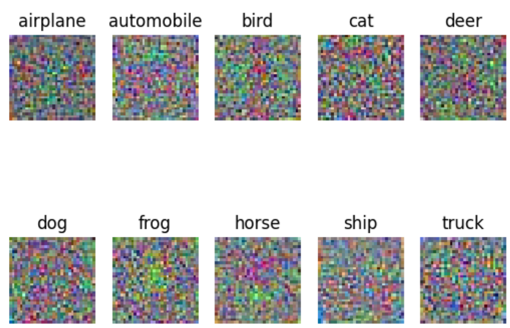
Parameters setting 1: λ=0, n_epochs=40, n_batch=100, η=0.1



(a) loss



(b) Accuracy

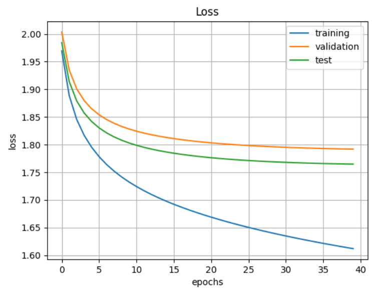


(c) W

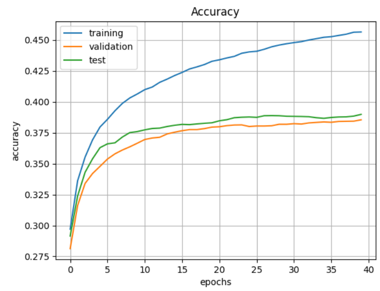
Figure 1: Plots for setting 1

Best test accuracy=0.3069.

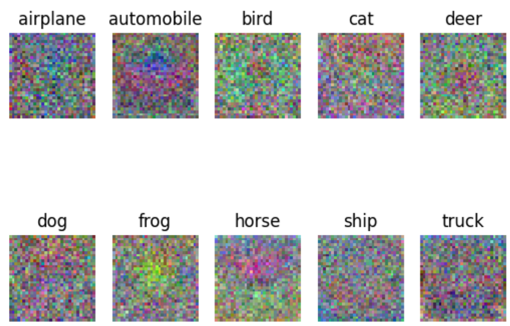
Parameters setting 2: λ=0, n_epochs=40, n_batch=100, η=0.001



(a) loss



(b) Accuracy

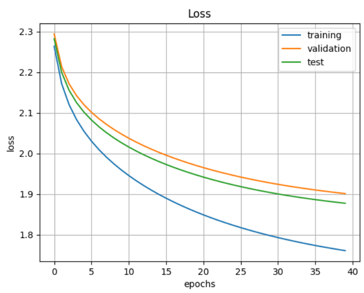


(c) W

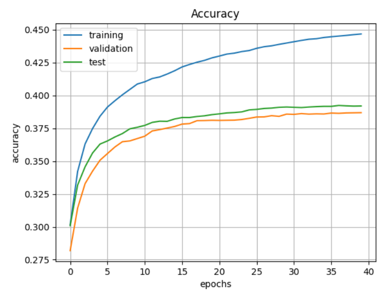
Figure 2: Plots for setting 2

Best test accuracy=0.3898.

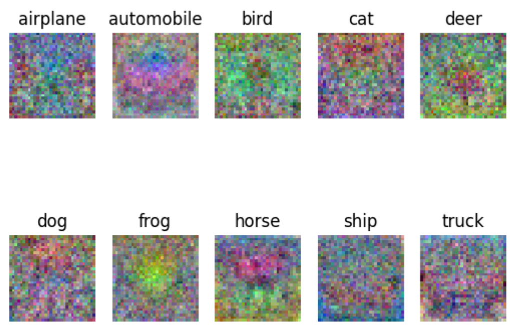
Parameters setting 3: λ=0.1, n_epochs=40, n_batch=100, η=0.001



(a) loss



(b) Accuracy

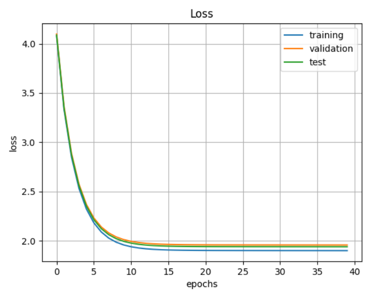


(c) W

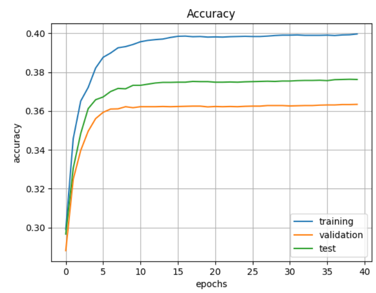
Figure 3: Plots for setting 3

Best test accuracy=0.3924.

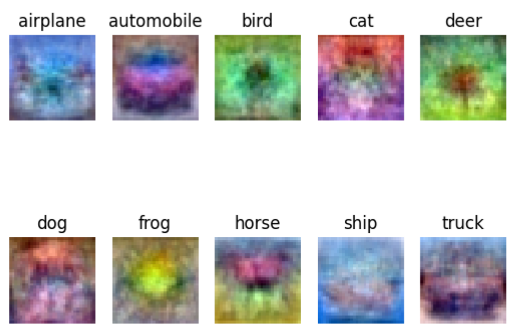
Parameters setting 4: λ=1, n_epochs=40, n_batch=100, η=0.001



(a) loss



(b) Accuracy



(c) W

Figure 4: Plots for setting 4

Best test accuracy=0.3763.

4 Conclusions

By comparing Figure 1 and Figure 2, where η is the only difference. I see that too large η would fail the training. This is because both W and b get too much update during the training process. Therefore, the parameters would oscillate around the local optimal point but never reach it. Thus, $\eta = 0.001$ has better performance than $\eta = 0.1$.
By comparing Figure 2, Figure 3 and Figure 4, where the only difference is λ, the regularization term. Increasing in λ (up to 1) would cause lower test accuracy. However, appropriate λ would avoid overfitting. $\lambda = 0.1$ seems a good choice.