Handwritten Digits with Multilayer Backpropagation Neural Networks

Shilin Zhu
Ph.D. student, Computer Science
UCSD
La Jolla, CA
shz338@eng.ucsd.edu

Yunhui Guo
Ph.D. student, Computer Science
UCSD
La Jolla, CA
yug185@eng.ucsd.edu

1 Abstract

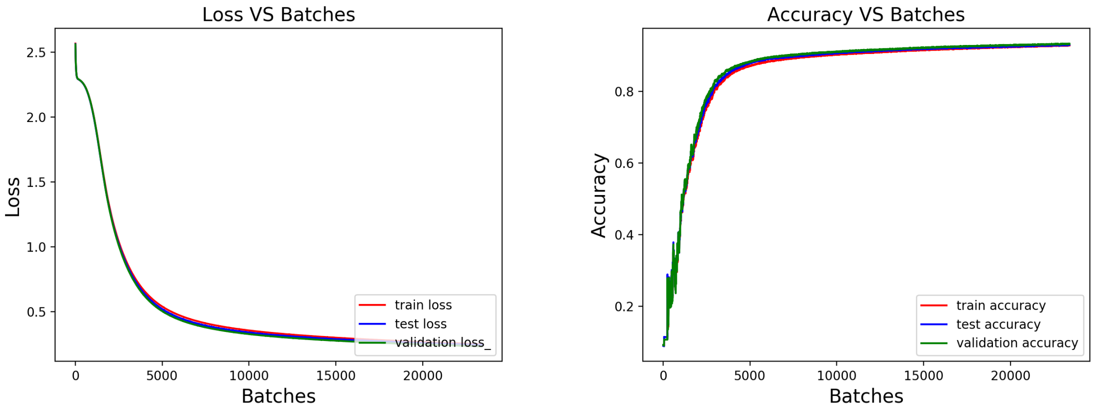
2 Classification

2.1 Mini-batch gradient descent

In this section, we use mini-batch gradient descent to classify the MNIST dataset. We split the 60000 images in the training set into two parts: the first 50000 images are used to train the model, the last 10000 images are used as validation set to do early stopping. We stop the training procedure once the loss on the validation set goes up and we save the weights that achieves the minimum loss on the validation set. And there are 10000 images in the test set.

We use one hidden layer of 64 nodes, and the mini-batch size is 128. We use a learning rate of 0.01 and sigmoid activation function. We use standard normal distribution to initialize the weights and biases. For the weights, we multiply 0.01 to prevent large initialized values. We run the network for 60 epoches.

We report the accuracy and loss on the training set, test se and validation set every batch. The following graphs show the accuracy and loss over each batch on different sets. Without any tricks, after 60 epoches, the accuracy on the test set is 0.9308%. This is no early stopping occurs, the possible reason is that the choice of learning is small



(a)

(b)

Figure 1: The loss and accuacy of different sets over the batches.

3 Adding the Tricks of the Trade”

4 Experiment with Network Topology

4.1 Experiments with differnet hidden units

We use a momentum of 0.9 and use the sigmoid in Section 4.4 of “lecun98efficient.pdf”. The initialization method of weights are as described in 4 (c) in Programming assignment 2. Learning rate is 0.01.

First, we half the hidden units. Now we have a network of 3 layers with a hidden layer with 32 nodes. We run the network for 60 epoches except early stopping occurs.

4.2 Doubling the hidden layers

4.3 More tricks

In this section, in order to improve the preformance of the network, we consider the following tricks. In our experiments, we found that the network can achieve fast convergence and higher test accuracy with the tricks.

4.3.1 ReLU

We consider using ReLU as the activation function. The ReLU function can be

$$\mathrm{ReLU}(x) = \begin{cases} x & \text{if x} > 0, \\ 0 & \text{otherwise,} \end{cases}$$

The gradient of ReLU can be caculated as,

$$\mathrm{dReLU}(x) = \begin{cases} 1 & \text{if x} > 0, \\ 0 & \text{otherwise,} \end{cases}$$

The following graphs shows the

4.3.2 Leaky ReLU

We consider using leaky ReLU as the activation function. The leaky ReLU function can be

$$\mathrm{LeakyReLU}(x) = \begin{cases} x & \text{if x} > 0, \\ 0.01x & \text{otherwise,} \end{cases}$$

The gradient of leaky ReLU can be caculated as,

$$\mathrm{dLeakyReLU}(x) = \begin{cases} 1 & \text{if x} > 0, \\ 0.01 & \text{otherwise,} \end{cases}$$

4.3.3 Nesterov momentum

We consider using Nesterov momentum.

4.3.4 Xavier initializtion

We consider using Xavier initializtion to initialize the weights.

4.3.5 Dropout