
Handwritten Digits Recognition 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

In this report, we improve the handwritten digits recognition with multi-layer back-propagation neural networks. We correctly derived and implemented back-propagation algorithm and many tricks of the trade to improve the performance of learning. Furthermore, we try out different network topologies to select the best model based on the accuracy on validation set. The results are quite satisfying: using all the tricks of the trade and model selection strategy, we successfully achieve 97.51% on the final test set.

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 set 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

2.2 Gradient checking

To verify the correctness of implementation of back-propagation, we compute the slope with respect to one weight using the numerical approximation: $\frac{\partial E^n}{\partial w_{ij}} \approx \frac{E^n(w_{ij}+\epsilon) - E^n(w_{ij}-\epsilon)}{2\epsilon}$ where we compute the numerical gradient for every weight and bias and for every example. Here we choose $\epsilon = 10^{-2}$ and according to the numerical theory, the difference between the gradients should be within $O(\epsilon^2)$ so that we expect the gradients to agree within 10^{-4} .

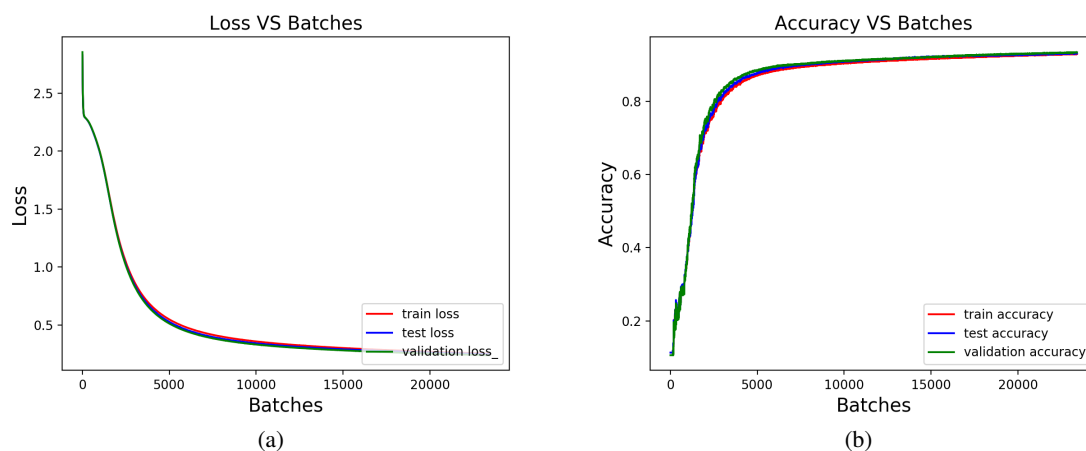


Figure 1: The loss and accuracy of different sets over the batches without any tricks.

3 Adding the Tricks of the Trade”

4 Experiment with Network Topology

4.1 Experiments with different 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 epochs except early stopping occurs. After 60 epochs, the accuracy on the test set is 0.9635%. This is no early stopping occurs, the possible reason is that the choice of learning is small enough.

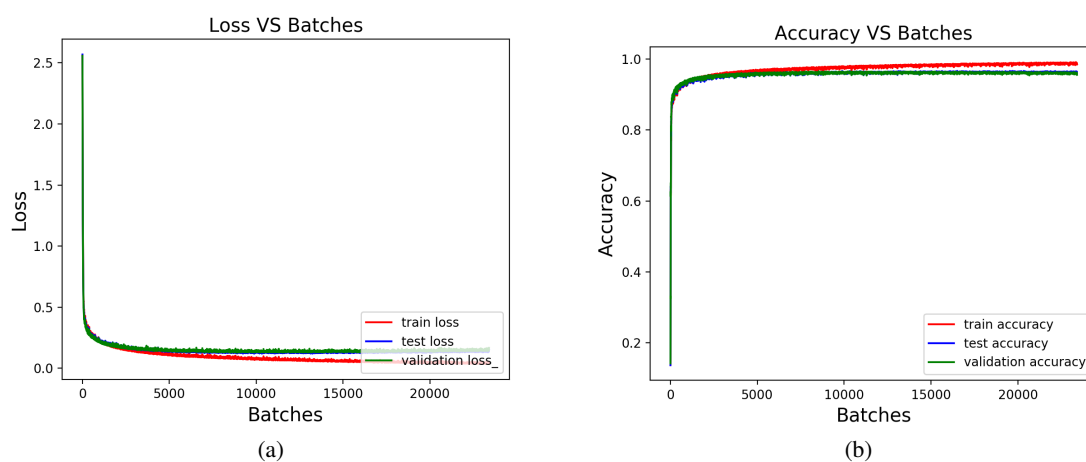


Figure 2: The loss and accuracy of different sets over the batches with a hidden layer of 32 hidden nodes.

Then, we double the hidden units. Now we have a network of 3 layers with a hidden layer with 128 nodes. We run the network for 60 epochs except early stopping occurs. After 60 epochs, the

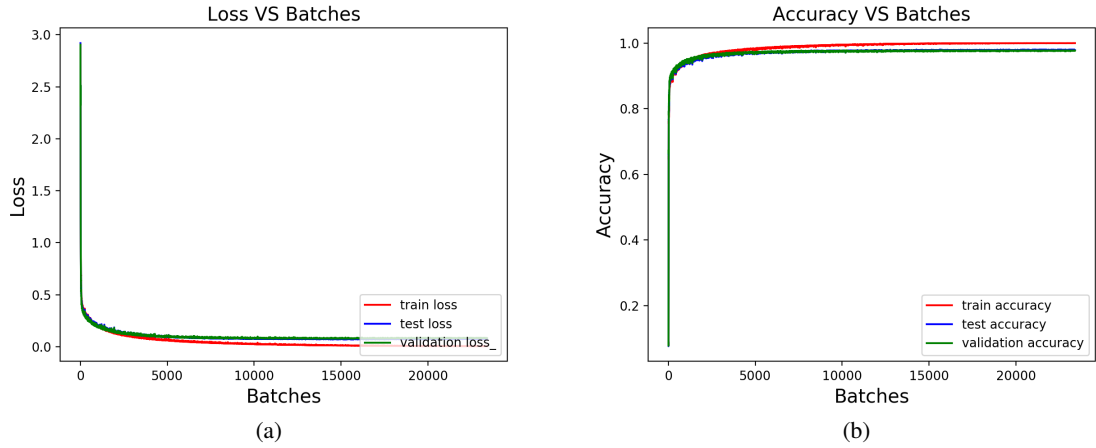


Figure 3: The loss and accuracy of different sets over the batches with a hidden layer of 128 hidden nodes.

accuracy on the test set is 0.9787%. This is no early stopping occurs, the possible reason is that the choice of learning is small enough.

4.2 Doubling the hidden layers

For a network with one hidden of 64 nodes, there are approximately about 50890 parameters. If we increase the hidden layers while keep the same number of parameters, there will be 58 hidden nodes in each hidden layer. We use a momentum of 0.9 and use the sigmoid in Section 4.4 of “le-cun98efficient.pdf”. The initialization method of weights are as described in 4 (c) in Programming assignment 2. Learning rate is 0.01. After 60 epoches, the accuracy on the test set is 0.9766%. This is no early stopping occurs, the possible reason is that the choice of learning is small enough.

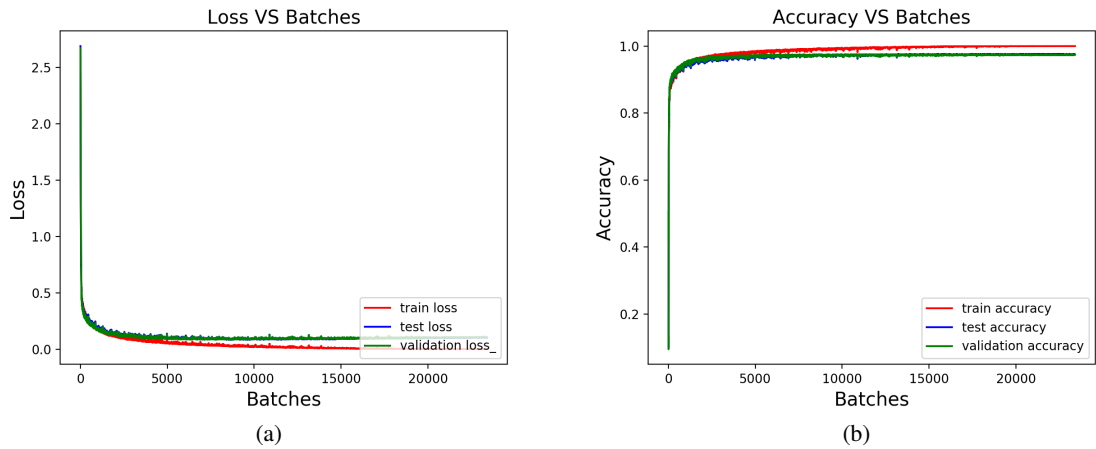


Figure 4: The loss and accuracy of different sets over the batches with two hidden layers.

4.3 More tricks

In this section, in order to improve the performance 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

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

The gradient of ReLU can be calculated as,

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

We use a three layer network with a hidden layer with 64 nodes. We use a momentum of 0.9 and use ReLU as activation function. The initialization method of weights are as described in 4 (c) in Programming assignment 2. Learning rate is 0.01.

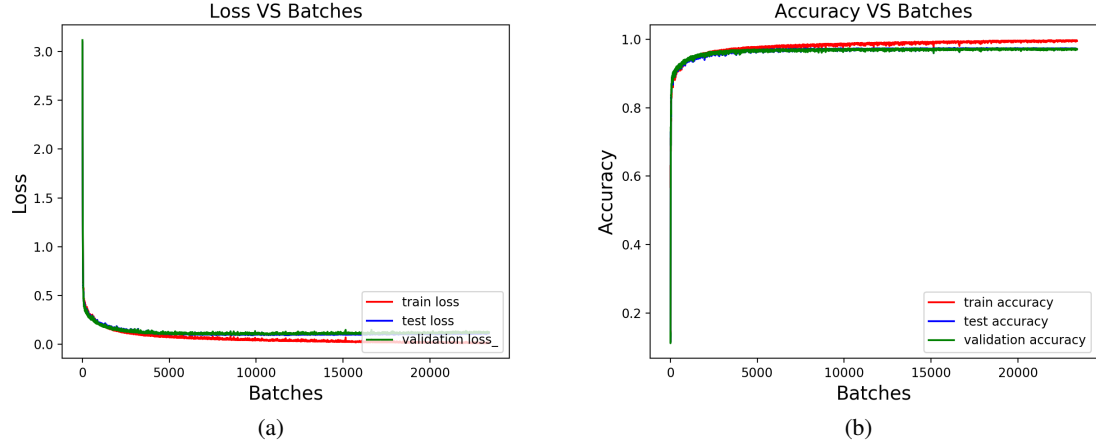


Figure 5: The loss and accuracy of different sets over the batches with ReLU.

4.3.2 Leaky ReLU

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

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

The gradient of leaky ReLU can be calculated as,

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

We use a three layer network with a hidden layer with 64 nodes. We use a momentum of 0.9 and use leaky ReLU as activation function. The initialization method of weights are as described in 4 (c) in Programming assignment 2. Learning rate is 0.01.

After 60 epoches, the accuracy on the test set is 0.9723%.

4.3.3 Nesterov momentum

We consider using Nesterov momentum. We use a three layer network with a hidden layer with 64 nodes. We use a momentum of 0.9 and use leaky ReLU as activation function. The initialization method of weights are as described in 4 (c) in Programming assignment 2. Learning rate is 0.01. After 60 epoches, the accuracy on the test set is 0.9734%.

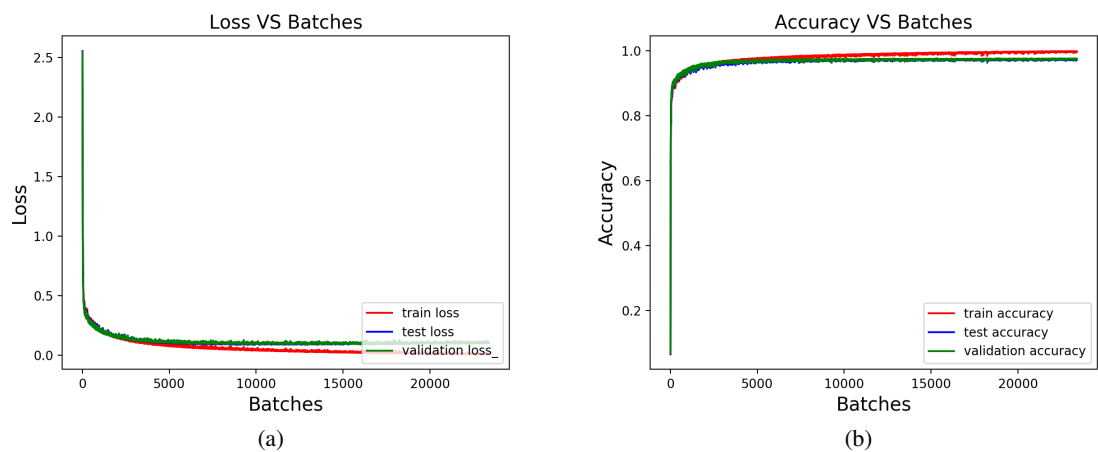


Figure 6: The loss and accuracy of different sets over the batches with leaky ReLU.

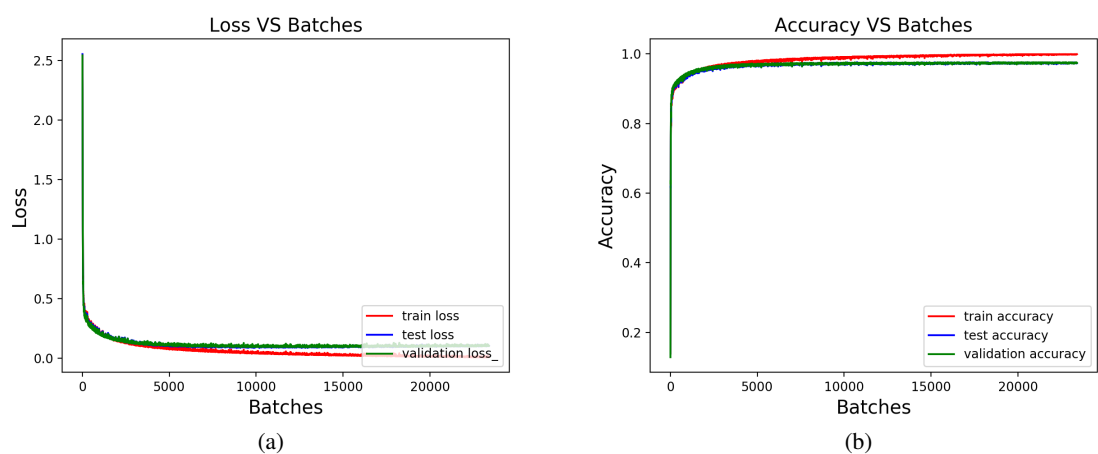


Figure 7: The loss and accuracy of different sets over the batches with Nesterov momentum.

4.3.4 Xavier initialization

We consider using Xavier initialization. We use a three layer network with a hidden layer with 64 nodes. We use a nesterov momentum of 0.9 and use leaky ReLU as activation function. The initialization method of weights are as described in 4 (c) in Programming assignment 2. Learning rate is 0.01.

0.9751%.

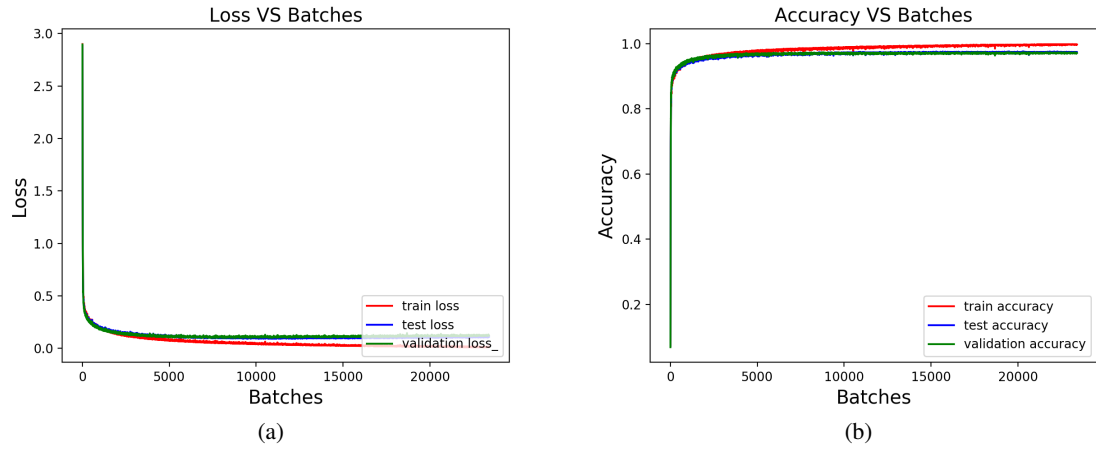


Figure 8: The loss and accuracy of different sets over the batches with Xavier initialization.