MLoN Computer Assignment 7

Group 1

Problem

Consider the "MNIST" dataset, and a DNN with J layers and $\{Nj\}$ neurons on layer j.

(a) Train DNN using SGD and your choices of hyper-parameters, L, and $\{N_j\}_{j\in[J]}$. Re- port the convergence rate on the training as well as the generalization performance. Feel free to change SGD to any other solver of your choice (give explanation for the choice).

(b) Repeat part a) with mini-batch GD of your choice of the mini-batch size, retrain DNN, and show the performance measures. Compare the training performance (speed, accuracy) using various adaptive learning rates (constant, diminishing, Ada- Grad, RMSProp).

(c) Consider design of part a) and fix $\sum_j N_j$. Investigate shallower networks (smaller J) each having potentially more neurons versus deeper network each having fewer neurons per layer, and discuss pros and cons of these two DNN architectures.

(d) Split the dataset to 6 random disjoint subsets, each for one worker, and repeat part a) on master-worker computational graph.

(e) To promote sparse solutions, you may use l_1 regularization or a so-called dropout technique. Explain how you incorporate each of these approaches in the training? Compare their training performance and the size of the final trained models.

(f) Improving the smoothness of an optimization landscape may substantially improve the convergence properties of first-order iterative algorithms. Batch-normalization is a relatively simple technique to smoothen the landscape. Using the materials of the course, propose an alternative approach to improve the smoothness. Provide numerical justification for the proposed approach.

Solution

(a)

We build a 2-hidden-layer (784-10-10-10) NN. We use RELU after each hidden fc layer. In the output layer, we apply softmax layer to output the predicted probability vector. To solve this multiclass classification problem, cross entropy loss is chosen. The performance is shown in Fig.1. The loss converges at around 25 epochs (For convenience, we observe the convergence simply from the loss figure, rather than check the gradient difference between epochs). The final test accuracy is 0.9361.

item	value
lr rate	0.001
decay rate	0.0001
momentum	0.9
epoch	100

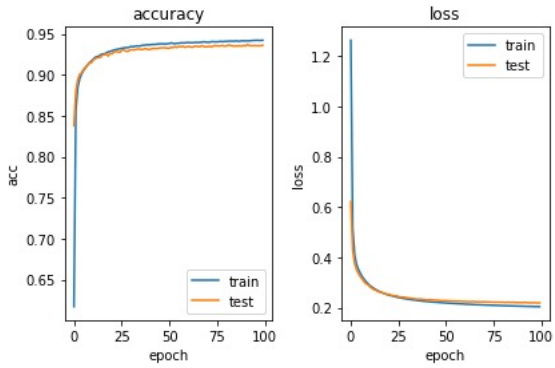

Figure 1: Accuracy and loss during the training (p1).

We show 9 correctly and 9 incorrectly predicted digits for this model in Fig.2 Fig.3.

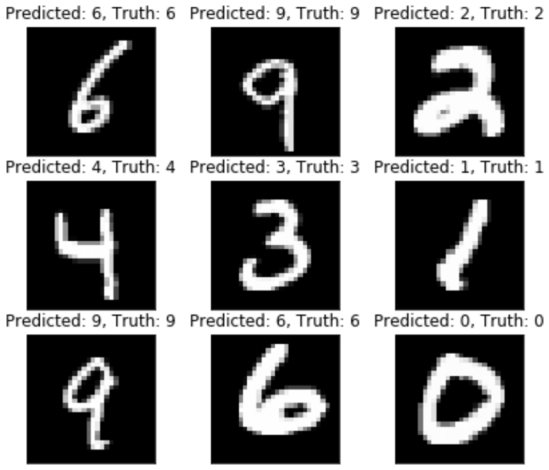

Figure 2: 10 correctly predicted images (p1).

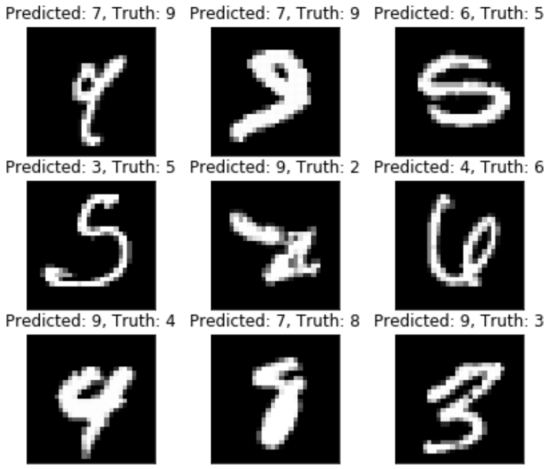

Figure 3: 10 incorrectly predicted images (p1).

(b)

We repeat the experiment with a large minibatch size(256) while remaining the same for the rest of hyper-parameters. The performance is shown in Fig.4. The loss converges at around 40 epochs. However, the final testing accuracy is only 0.9175 which is lower than the orginal sgd (batchsize=1). We think this is because we have fewer update iterations comparing with the one with small batchsize, so that the model is still in underfitting state. The correctly and incorrectly predicted digits are shown in Fig.5 Fig.7.

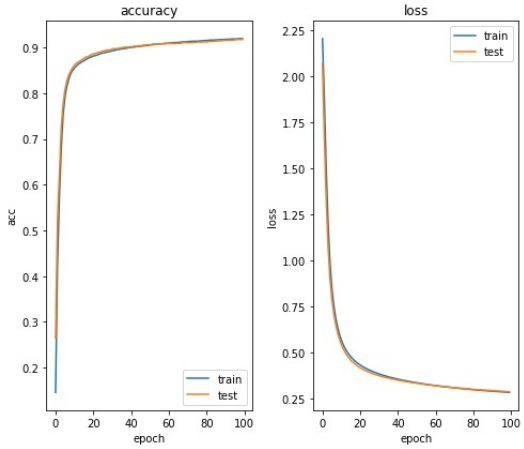

Figure 4: Accuracy and loss during the training (p2).

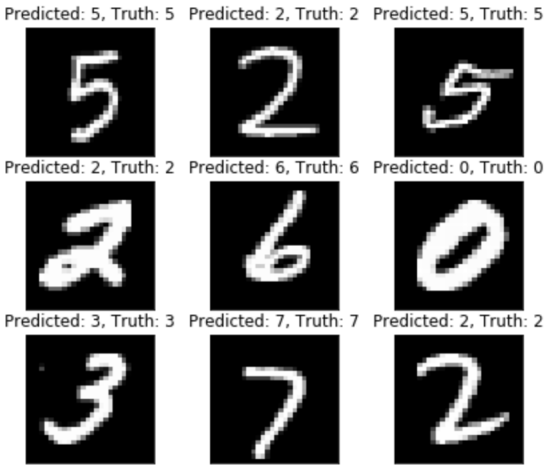

Figure 5: 10 correctly predicted images (p2).

We compare the training performance using various adaptive learning rates (constant, diminishing, adagrad and rmsprop). The result is shown in Fig.**??**. RMSProp outperforms the other methods, reaching the best accuracy and converges fastest.

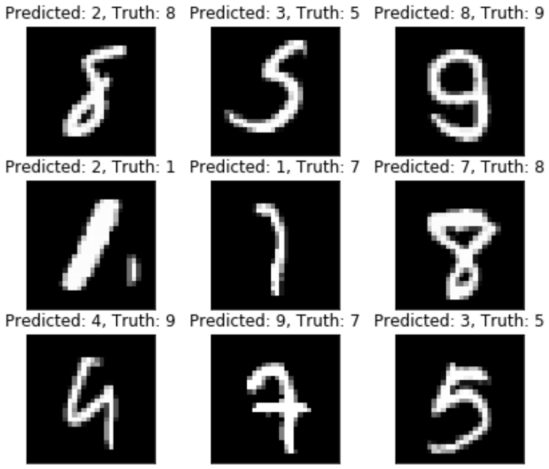

Figure 6: 10 incorrectly predicted images (p2).

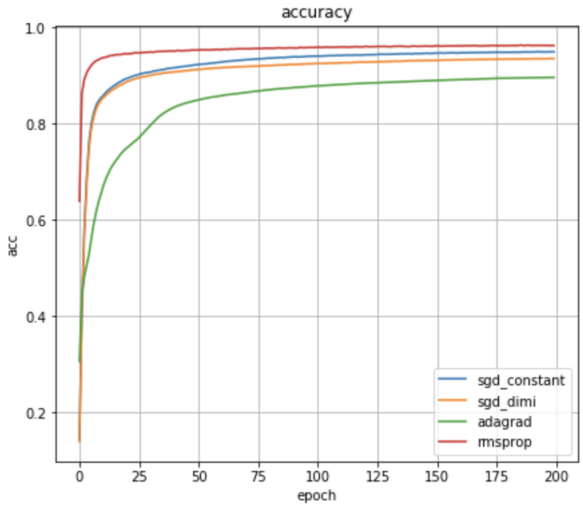

Figure 7: Performance using different optimizers (p2).

(c)

In order to investigate shallower networks(smallerJ) each having potentially more neurons versus deeper networkeach having fewer neurons per layer, we construct two new models. One is with 4 hidden layers (784-5-5-5-5-10) and one with 1 hidden layers (784-20-10). The result is shown in Fig.11. As we can see in the figure, for this task, the deepest model is hard to train and performs not very well .

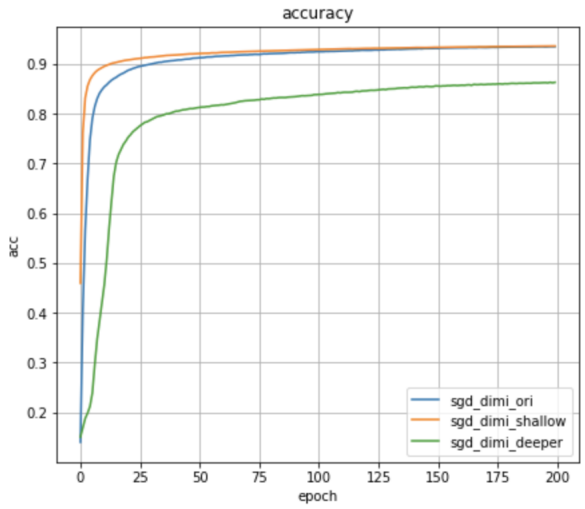

Figure 8: Performance from shallow to deep network(p3).

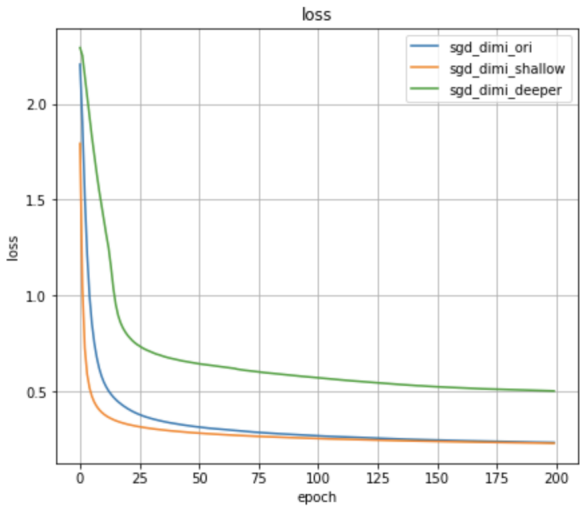

Figure 9: Training loss from shallow to deep network(p3).

(d)

We built a master-worker computational graph, a master node and 6 workers while splitting the dataset to 6 random disjoint subsets, each for one worker. The master node is responsible for collecting the gradients from the workers and then broadcast the average to all the workers. We continue to use the same NN model as problem (a), i.e. 2 hidden layer (784-10-10-10) NN, with RELU

activation function. As for the hyperparameters, we use SGD optimizer with learning rate 0.01, decay rate 0.0001, and momentum 0.9. By looking at the losses and accuracy in Fig. 10 among each worker, the converge rate is lower and the best accuracy we can get is 0.8765.

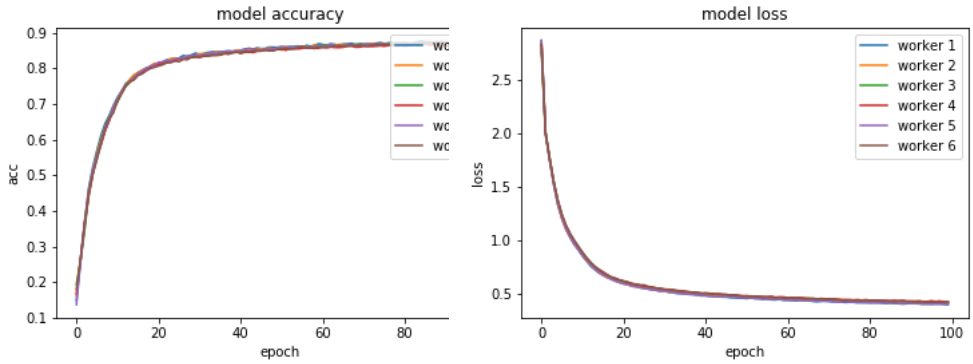

Figure 10: Accuracy and loss during the training (p4).

(e)

We can use l_1 regularization or dropout to encourage sparse solution. To incorporate l_1 regularization, we can simply add it into the loss function, by summing up the absolute value of the network weights. To incorporate the dropout technique, we add the dropout layer before the fully-connected layer. Here, we use dropout(0.1) to promote a sparse model. The result is shown in Fig.**??**. However, the model with dropout doesn't meet our expectation. We think the base model is still too shallow and no need to utilize dropout.

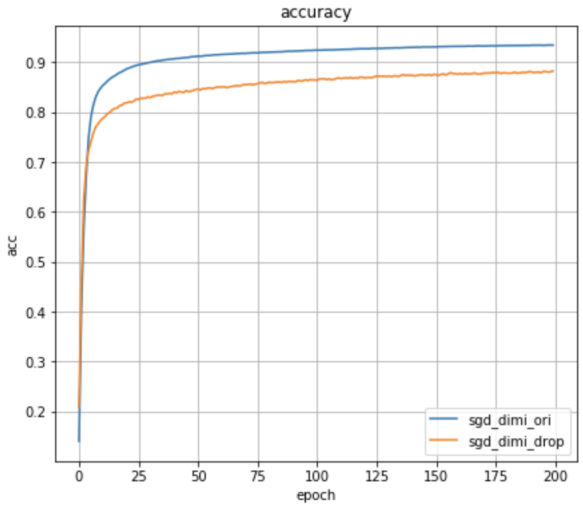

Figure 11: Performance of the model with/without dropout.

(f)

Here we use the same hyperparameters to train the same model with and without Batch Normalization, as Fig. 12, blue line indicating BN applied while orange line indicating BN not applied, where shows that applying BN could smoothen the optimization. In theory, this effect will appear more obvious when the network is getting deeper.

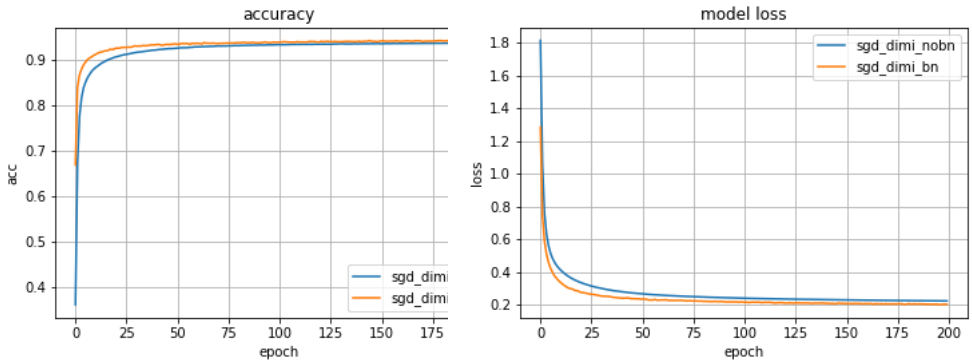

Figure 12: Accuracy and loss during the training (p6).