Deep Learning 2020 **(Due Date : 25.11.2020)**

Assignment # 2

Instructor: Jakub Tomczak *Name:* Shwetambari Tiwari, *VUNetid:* sti860

1 Implementation

For the neural network implemented earlier. The architecture studied was $784(input) \rightarrow Linear(784, 300) \rightarrow Sigmoid \rightarrow Linear(300, 10) \rightarrow Softmax$. The total number of parameters for this network are **238,510**.

$$Number\ of\ parameters = \underbrace{(784 * 300) + (1 * 300)}_{Hidden\ Layer1} + \underbrace{(300 * 10) + (1 * 10)}_{Hidden\ Layer2} = 238,510 \tag{1.1}$$

For the convolutional neural network, the total number of parameters is **2710**. The number of parameters in Cov2d layers is lesser than the fully connected layers. We can also say that activation size goes down as we go deeper into the neural network.

	No of parameters
Conv 1	((3 * 3 * 1) + 1) * 16 = 160
Pool 1	0
Conv 2	((3 * 3 * 16) + 1) * 4 = 580
Pool 2	0
FC	(196 + 1) * 10 = 1970

Table 1: Computation for Convolutional Neural Network

Here, we see that solving an image classfication problem for MNIST dataset with a conventional neural network is not computationally optimal. CNN tries to solve the problems of a simple neural network architecture. In a simple neural network, training the large number of parameters is computationally expensive. Another problem is that the network is prone to overfitting because having many too parameters helps to network memorize the data. Convolutional layers reduce the number of parameters and speed up the training of the model significantly. A feature detector enables parameter sharing while convolving through the input. The other advantage of convolution neural network is that the output of each layer is dependent on a small number of inputs, instead of the whole input set.

Implementation Details The convolutional neural network is implemented as a module class in PyTorch. Juptyer Notebook for the same is submitted. The inbuilt datasets are used via Pytorch (1). The MNIST dataset is normalized. The training data is further split into training and validation set. The training set had 60,000 records which were split into 50,000 and 10,000 for training and validation set respectively. A seperate test data is loaded of 10,000 records for testing the peformance of the model. A batch size of 64 is considered for faster computation.

2 Analysis

1. On the training and validation set loss curves per epoch are plotted and are shown in the Figure 1. The training is conducted for 15 epochs. Learning rate is 0.001. It can be seen here, that for chosen parameters training set reduces gradually and smoothly. And, the validation set also shows a similar trend, but has a higher loss value in comparison to the training set. Similarly, in case accuracy, the training set performs better and shows a gradual increase, while the validation set has a similar trend but lower accuracy than the training set. A point to note is that the overall accuracy is above 95% both for training and validation set. The loss curves can converge on further training for higher number of epochs. Also, the learning rate and batch size can be tweaked for a better performance over the network.

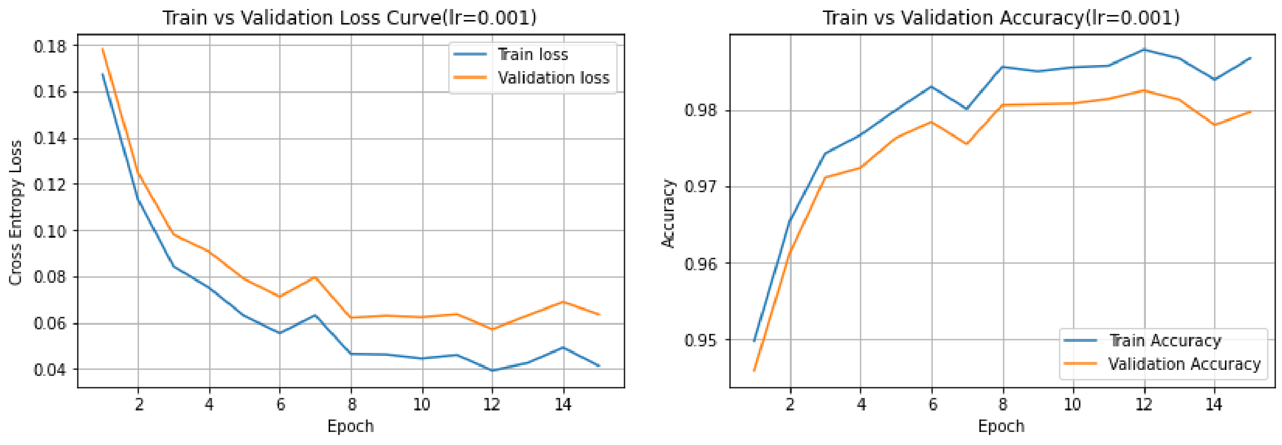

Figure 1: Comparison of Training Vs Validation-MNIST dataset

The training loss per batch (batch size 64) is also observed and is shown in the Figure 2. This shows that the loss reduces gradually and smoothly as the training progresses. The data observed is for the first 1000 records of the batched loss obtained during the training process.

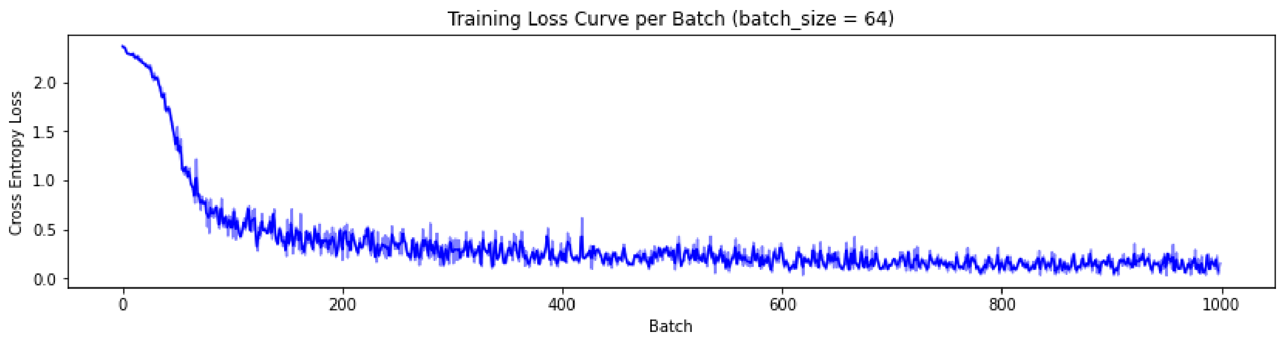

Figure 2: Loss per batch on Training data

An analysis of training and test set is also conducted at the end of the training procedure. The results are shown in Figure 3. Similar setting is used to check the network performance. A learning rate of 0.001, batch size of 64 and epochs 15 are the parameters for this test. According to the results obtained, the model looks underfit and can be trained further as it diverges after epoch 6 for the training curve.

Test Classification Error and Accuracy The accuracy obtained on the test set was 98.1%. Therefore, the test classification error is 1.9%. It is also interesting The MLP implemented earlier had the best accuracy of 84.61% for the same MNIST dataset. With the implementation of CNN, the accuracy is improved vastly to be above 98%.

2. Adam optimizer is run 3 times and the standard deviation and average of the loss per epoch is observed for the validation set. This shows that the early in the training procedure the standard deviation for loss is high and as the training progresses the deviation in the loss slowly converges. A similar trend is observed for accuracy but the deviation is lesser in accuracy in comparison to loss.

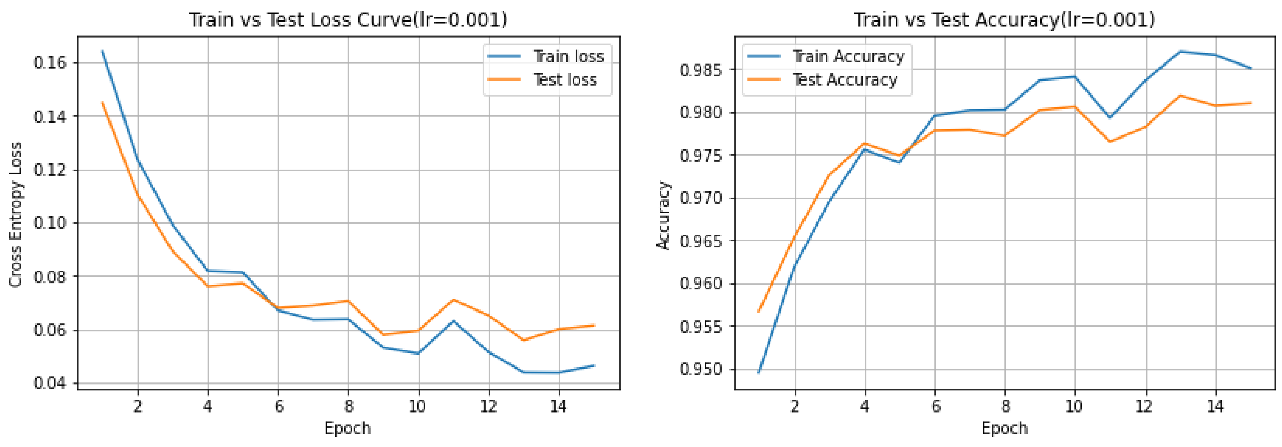

Figure 3: Comparison of Training Vs Test-MNIST dataset

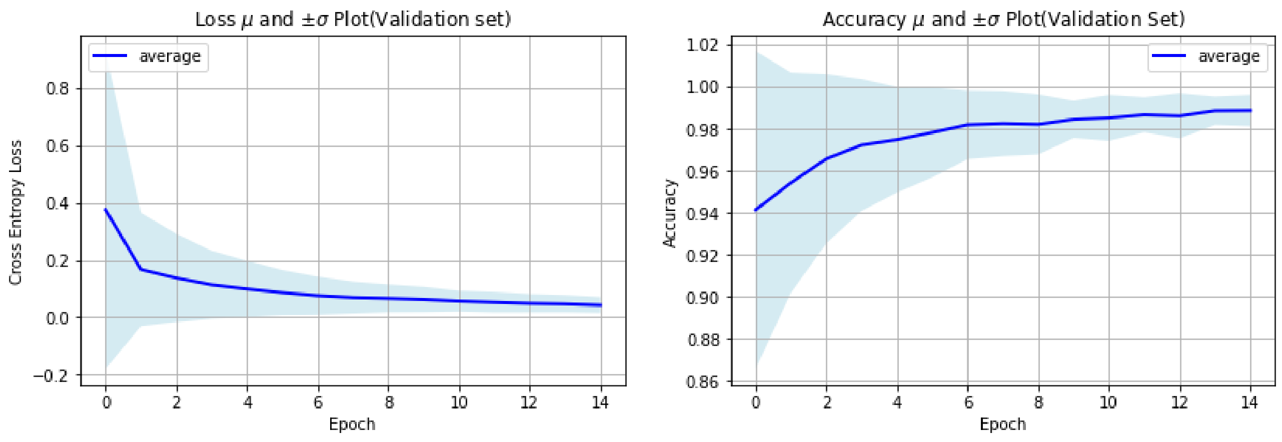

Figure 4: Comparison of Learning rates on Adam Optimizer

3. Adam optimizer is compared for different step sizes. The compared step sizes are 0.01, 0.001, 0.0001 and 0.0000001. Based on the experiments conducted in earlier assignment, the learning rate of 0.1 is sufficiently high for a good performance in a neural network, hence it was not considered to conduct this test. This experiment is conducted on the validation set. The result of the learning rates comparison is shown in Figure 5. It can be seen here that the learning rate of 0.001 and 0.0001 have similar performance. While the smallest learning rate 1e-08 performs poorly. Hence, a learning rate of 0.001 was chosen to conduct the test on test data as with the learning rate of 0.0001 the training time would be higher.

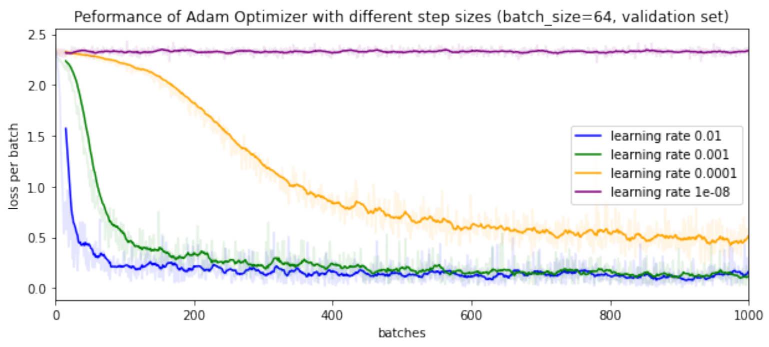

Figure 5: Comparison of Learning rates on Adam Optimizer

References

Pytorch -. https://pytorch.org/docs/stable/torchvision/datasets.html.