CS 7180 Special Topics in AI: Deep Learning
Northeastern University, Spring 2019

Assignment 1

Name: Tyler Brown UID: 001684955

1 Problem 1

Assignment 1 requested that we use PyTorch to build a CNN to classify handwritten digits from the MNIST dataset. We were then asked to explore regularization methodologies such as BatchNorm, Dropout, and L2. The hyperparameters in Table 1 were used on all models trained for 10 epochs.

Table 1: MNIST CNN Model Hyperparameters

	Value
Batch Size	32
Test Batch Size	1000
Epochs	10
Learning Rate	0.001
Momentum	0.9
Random Seed	42

The total number of parameters used by the CNN model I constructed is 643,755 which is less than the maximum number of 1 million parameters. My CNN model is constructed as follows:

$$\boxed{\text{Conv layer 1}} \rightarrow \boxed{\text{Conv layer 2}} \rightarrow \boxed{\text{Dropout}} \rightarrow \boxed{\text{Fully Connected 1}} \rightarrow \underset{output}{\boxed{\text{Fully Connected 2}}}$$

Categorical cross entropy is used as the loss function. Stochastic Gradient Descent (SGD) is used for optimization. Each Conv layer contained a $Conv2d$ layer, ReLU activation function, and $MaxPool2d$ layer with a kernel size and stride of 2. My approach is informed by reviewing several implementations of this problem and adopting a similar framework but with modifications for different forms of regularization, and batch normalization. Conv layer 2 includes a $BatchNorm2d$ layer with 75 features. The assignment requested Batch Normalization to be included in the model so I added it but did not experiment too much because my implementation was already above the 0.985 test accuracy threshold. The Dropout layer was excluded, $p = 0.0$, except when specifically requested by this assignment.

Table 2 contains results from using a range of probabilities between 0 and 1. The test results after 1 epoch show that varying levels of dropout didn't allow for more than 0.15 accuracy, far below 0.985. I've read that dropout usually hurts perfomance at the beginning of training but provides improvements when the model is closer to convergence. Therefore, judging dropout performance after 1 epoch should be problematic.

Table 2: Test Results after 1 Epoch with Dropout, CNN, BatchNorm

	Average Loss	Accuracy
$p = 0.0$	2.3494	6.22%
$p = 0.1$	2.3512	11.69%
$p = 0.2$	2.3517	12.20%
$p = 0.3$	2.3255	8.72%
$p = 0.4$	2.3080	12.37%
$p = 0.5$	2.3446	10.64%
$p = 0.6$	2.3429	9.00%
$p = 0.7$	2.2936	12.68%
$p = 0.8$	2.3343	10.30%
$p = 0.9$	2.3098	7.69%
$p = 1.0$	2.3137	10.13%

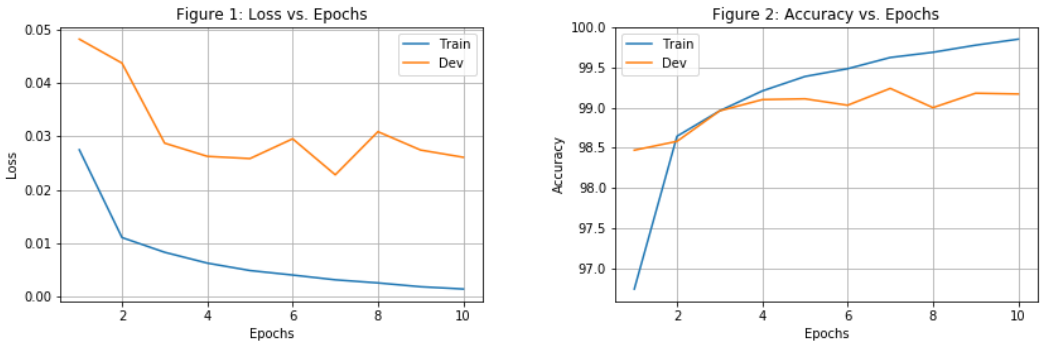
Results from Table 3 show how dropout is still associated with poor performance in this use case even after training for 10 epochs. I have seen other implementations of CNN use dropout with reasonable results so I'm open to the possibility of getting good results from dropout with alternative model specifications.

Table 3: Test Results after 10 Epochs

	Average Loss	Accuracy
CNN,BatchNorm	0.0261	99.17%
CNN,BatchNorm, Dropout=0.25	2.3230	10.06%
CNN,BatchNorm, L2=0.002	0.0340	98.95%

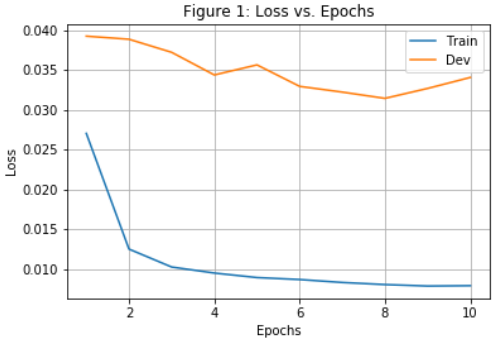
We can see that a CNN model without L2 regularization outperforms the other models. However, when we review Figure 1 containing *Loss vs. Epochs* and *Accuracy vs. Epochs* it becomes clear that the training set is overfitting relative to the development/test set.

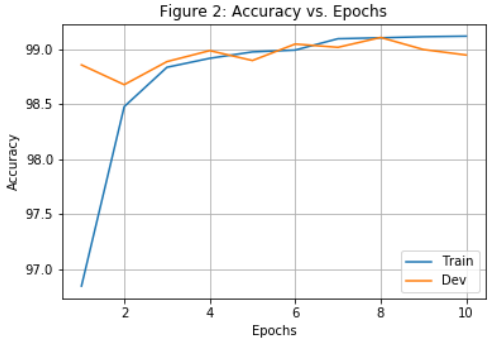
Figure 1: CNN with Batch Norm



We need to consider strategies to address overfitting in Figure 1 or the model should not generalize well in theory, even though the accuracy score is slightly higher for the development set. We cannot add data in this exercise so it's important to consider other

Figure 2: CNN with Batch Norm and L2 Regularization=0.002





regularization strategies to increase bias to reduce overfitting. Figure 2 shows what happens when L2 regularization is used. The accuracy results across epochs in Figure 2 show the development set tracking much more closely with the training set than we saw in Figure 1. It's possible that L2 regularization may benefit from early stopping after 8 epochs. This seems to be when the training set and development have accuracy scores that converge and reach their maximum.

The most effective strategies I found for increasing accuracy using a CNN, given a well accepted model specification, is increasing the number of neurons in a layer while tuning L2 regularization to prevent overfitting. In future iterations, I would explore techniques such as early stopping and randomized selection of hyperparameters such as learning rate. In past models, I've found learning rate to be a hyperparameter which is very meaningful for performance. This homework has helped me get exposure to Pytorch and AWS Sagemaker.