**Introduction**

Despite very recent reports studying effects of different batchsizes in neural network training, batch size selection is still a relatively less explored subject. Large batch sizes help scaling training problem into multiple nodes in distributed fashion, avoiding underutilization of hardware, and it reduces the noise in gradient updates since larger number of samples are used to estimate the gradient in each iteration. On the other hand, using larger minibatch size reduces number of updates for the same amount of data, which might increase training time overall, and recent reports suggest using large minibatch size reduces the generalization properties of the resulting model, degrading model quality [1]. In this report, we look at the effect of minibatch size by using 128, 2048 and 8192 as batch sizes to train a convolutional neural network with MNIST dataset, and compare the two cases. We confirm that gradient updates with smaller minibatch sizes are much more noisy, and requires tuning the learning rate accordingly (reducing learning rate for smaller minibatches). However, even though both models have the same training error, model that is trained with smaller minibatch has lower test error (0.70%, 0.77%, 0.97% for minibatch sizes 128, 2048, 8192, respectively).

**Requirements**

We used Python 3.6 with Tensorflow 1.4.0, and numpy package, random package, pyplot package and also the MNIST dataset. We trained neural networks on Mac OS CPU.

**Description of the program**

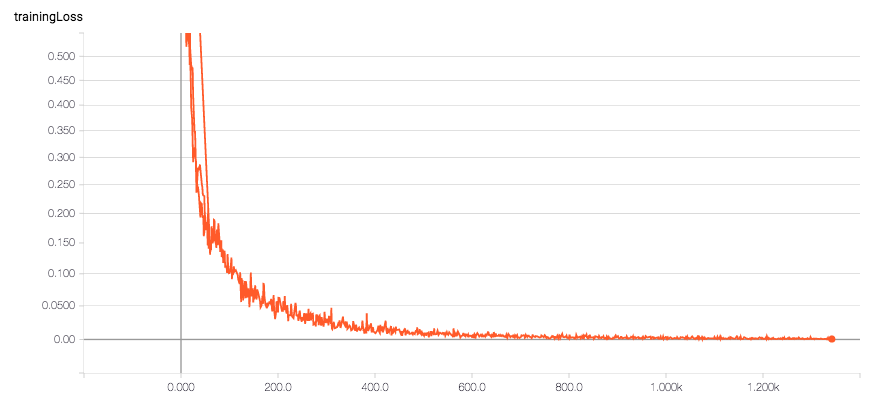
In our program, we import MNIST data and reshape it to feed into the convolutional network. We did not apply flipping or rotation to the images, since that may confuse the network for some of the digits. We used 55000 images for training, and 10000 remaining images for testing.

**Description of the network architecture, optimization and hyperparameters**

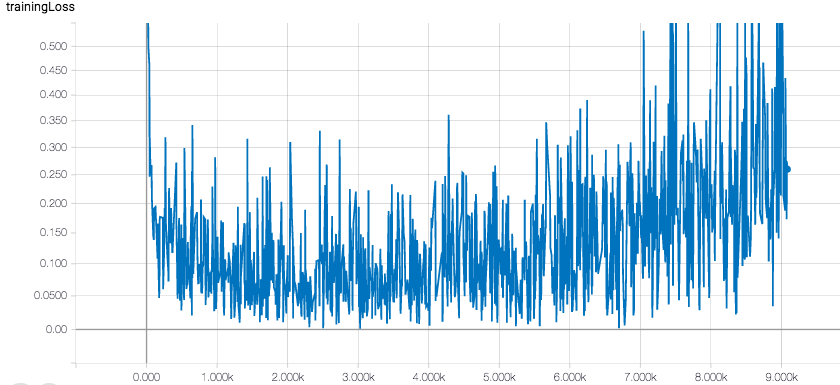
We used a convolutional neural network that has a reasonable size to be able to train in a reasonable amount of time. The complete network has 2 convolutional layers of sizes H,W,C,K = 5,5,3,32 and 5,5,32,64 (H and W are filter height and width, C and K are input and output channel sizes, respectively) both followed by relu (rectified linear unit) and max pooling layers for downsampling. We used relu since it makes optimization easier. We then flatten the resulting tensor and feed it into a fully connected layer, which is followed by a relu layer, and a dropout layer with P=0.5 as a regularizer. This is followed by another fully connected layer that is then fed into softmax. We used cross entropy as the loss function.

We initialize all the weights such that their variance is sqrt(2/fan\_in) and they are normally distributed to keep the unit variance [2]. We initialize biases as 0. We use SGD with momentum as optimizer, since it is relatively simpler and it performs well. We train all networks for 50 epochs. To achieve this, we implement (training set size=55000)/batchsize \* 50 iterations for each minibatch size and learning rate we try.

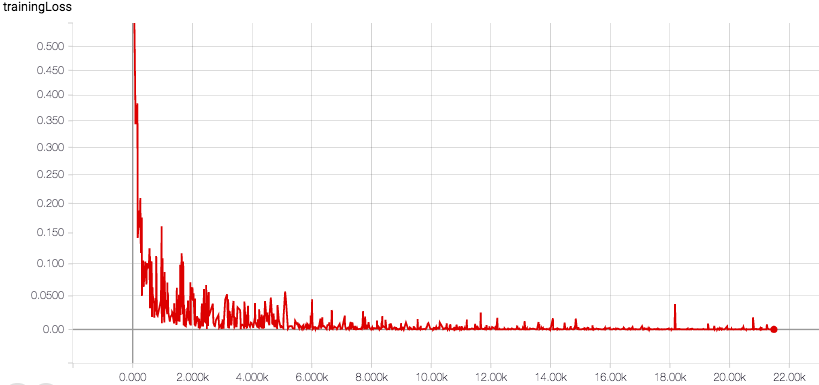
We tried minibatch sizes of 128, 2048 and 8192. For batch size of 2048 and 8192, we used 0.02 as learning rate since it performs well. However, the same learning rate gave us convergence issues for minibatch size 128, hence we used learning rate 0.002 for minibatch size of 128. Optimization progresses for different cases are shown in the figures below. Notice the noisy updates for minibatch size 128, and convergence issues when minibatch size is 32 and learning rate is 0.02. Notice that X-axis is number of iterations in all figures below.

****

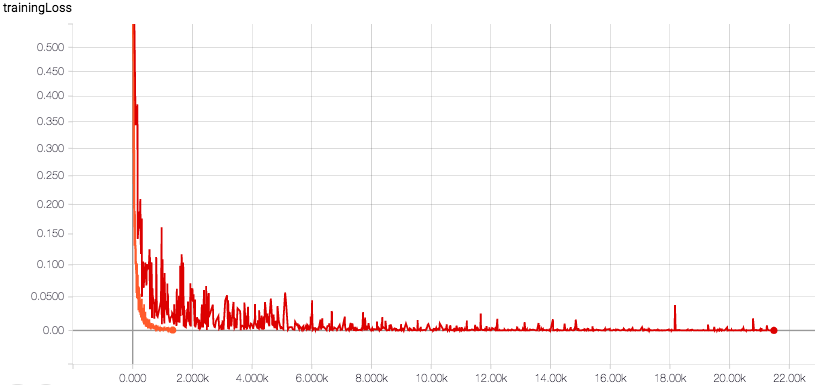
**Figure 1.** Training loss progress for the case minibatch size=2048, learning rate=0.02. X axis is number of iterations. Number of iterations until the optimization stop correspond to 50 epochs

****

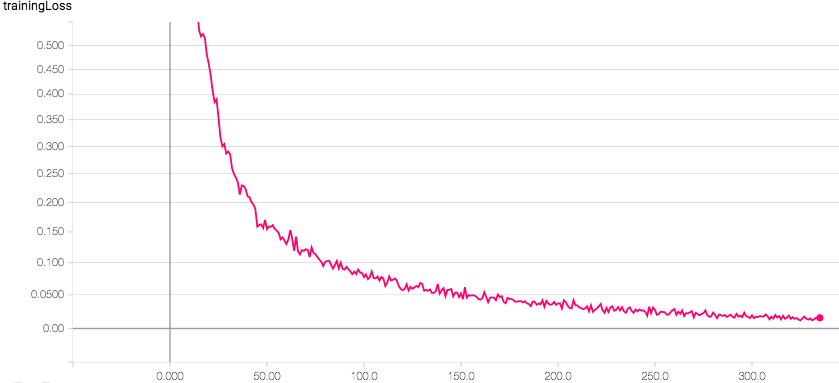
**Figure 2.** Training loss progress for the case minibatch size=128, learning rate=0.02. X axis is number of iterations. This training session is stopped early since optimization starts to diverge.

****

**Figure 3.** Training loss progress for the case minibatch size=128, learning rate=0.002. X axis is number of iterations. Number of iterations until the optimization stop correspond to 50 epochs

****

**Figure 4.** Figure 1 and 3 superimposed, where orange curve is the case minibatch size=2048, learning rate=0.02, and red curve represents minibatch size 128, learning rate=0.002. Notice the noisy updates for smaller minibatch size vs smoother curve for the larger minibatch. X axis is number of iterations. Number of iterations until the optimization stop correspond to 50 epochs for both cases, and since this requires more iterations for smaller minibatch size, red curve shows more iterations than the orange curve.

****

**Figure 5.** Training loss progress for the case minibatch size=8192, learning rate=0.02. X axis is number of iterations. Number of iterations until the optimization stop correspond to 50 epochs

**Results**

Results of optimizations are summarized as follows:

|  |  |  |
| --- | --- | --- |
| Minibatch size | Training error | Test error |
| 128 | 0.00% | 0.70% |
| 2048 | 0.00% | 0.77% |
| 8192 | 0.42% | 0.97% |

Note that even though training errors are the same for both cases, test error is lower for the case with lower minibatch size. This is despite the fact that gradient updates are noisier with smaller minibatch size, as reported in the previous section. Test error is the lowest for minibatch size 8192, but that might also be because that case didn’t fully converge with 50 epochs (notice it also has higher training error than other cases), and more epochs might be needed to reach a conclusion.

**Conclusion**

We tried to see the difference in training with different batch sizes on the same model. Even though MNIST is a data set that most machine learning models reach very high accuracies, we can still see the difference in test accuracies for different batch sizes (training with a more complex data set might have resulted in a bigger difference). We used a 4 layer Convolutional Neural Network to train. As expected, smaller minibatch has lower test error than larger minibatch sizes (0.70%, 0.77%, 0.97% for minibatch sizes 128, 2048, 8192, respectively).

**Python program**

Python programs including the results of different trainings are attached as Jupyter notebooks in a zip file.

**References**

[1] Keskar et al., “On large-batch training for deep learning: Generalization gap and sharp minima,” ICLR 2017.

[2] He et al., “Delving deep into rectifiers,” ICCV 2015.