CSci 5525 Homework 2

Luis Guzman

Thursday October 15, 2020

1. With zero padding, the input array becomes 12x12. Using a stride value of 3, there are 4 possible locations per row/column for the kernel to be evaluated. Thus,
   $n = 4, m = 4, k = 16$

   Since we have only a single 3x3 filter, W_{conv} has 9 parameters to learn.
   $W_{conv} = 3 \times 3$ (or 9×1 flattened)

   The fully connected layer has one parameter per input (including bias) and output node combination: $10 * (k + 1) = 170$
   $W_{fc} = 170 \times 1$

   and b is a scalar

2. In the first programming problem, I implemented a neural network using tensorflow to classify the mnist digits dataset. The network architecture I used was:

   - Input: 1-channel input, size 28x28
   - Fully connected layer 1: input with bias; output - 128 nodes
   - ReLU activation function
   - Fully connected layer 2: input - 128 nodes; output - 10 nodes
   - Softmax activation function
   - Use cross entropy as the loss function
   - Use SGD as optimizer
   - Set mini batch size as 32

   I set the maximum epochs to 500 and defined the stopping criteria as three epochs without seeing an improvement in the cross entropy loss. In this example, all 500 epochs were run without reaching the stopping criteria, and the final testing accuracy was 97.47%.

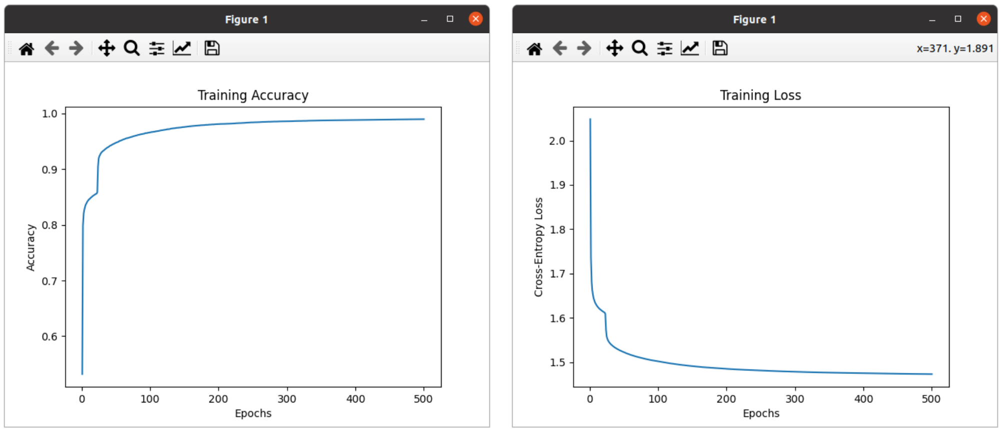

Figure 1: Neural Network accuracy and loss plotted for each training epoch

3. In the next problem, I again used tensorflow to classify the mnist dataset, this time using a convolutional neural network (CNN). The network architecture I used was:

   - Input: 1-channel input, size 28x28
   - Convolution layer: Convolution kernel size is (3, 3) with stride as 1. Input channels - 1;Output channels - 20 nodes
   - ReLU activation function
   - Max-pool: 2x2 max pool
   - Dropout layer with probabilityp= 0.50
   - Flatten input for feed to fully connected layers
   - Fully connected layer 1: flattened input with bias; output - 128 nodes
   - ReLU activation function
   - Dropout layer with probabilityp= 0.50
   - Fully connected layer 2: input - 128 nodes; output - 10 nodes
   - Softmax activation function
   - Use cross entropy as loss function

   In the first part of this question, I used an SGD optimizer with mini batch sizes of 32. At first, I used the same stopping criteria as question 2, but I found that the CNN algorithms actually performed slightly worse due to the shorter training time (80 epochs vs 500 for the neural network). I tried increasing the stopping criteria to 6 epochs of stagnant loss values, instead of 3, and then the CNN beat the neural network, despite only training for 140 iterations. The loss and accuracy plots are given in figure 2. The final testing accuracy was 97.85%.

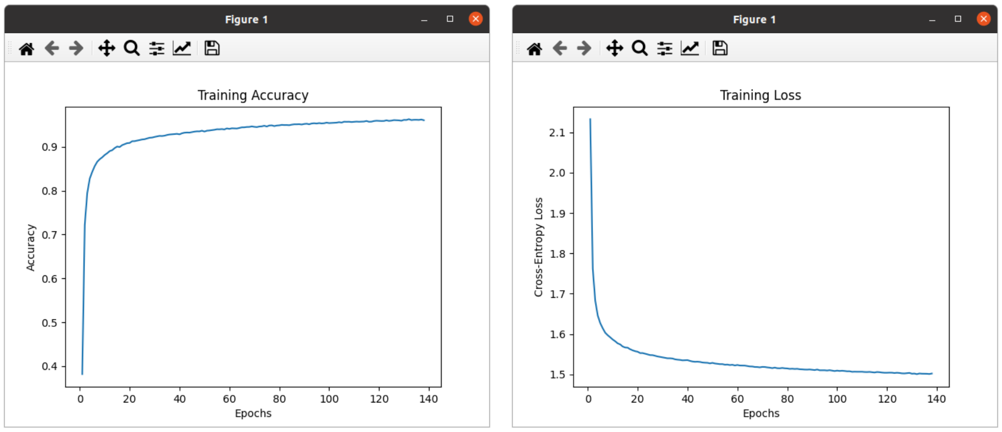

Figure 2: Convolutional Neural Network accuracy and loss plotted for each training epoch

In the next part of this question, I compared the training runtime of three different optimizers and four different batch sizes. The optimizers I compared were {SGD, Adagrad, Adam} and the batch sizes were {32, 64, 96, 128}. I used the python module timeit to time the runtimes of each training until the convergence condition had been met. I again ran this test twice: once using the stopping criteria from problem 2, and once with the 6 epoch patience value.

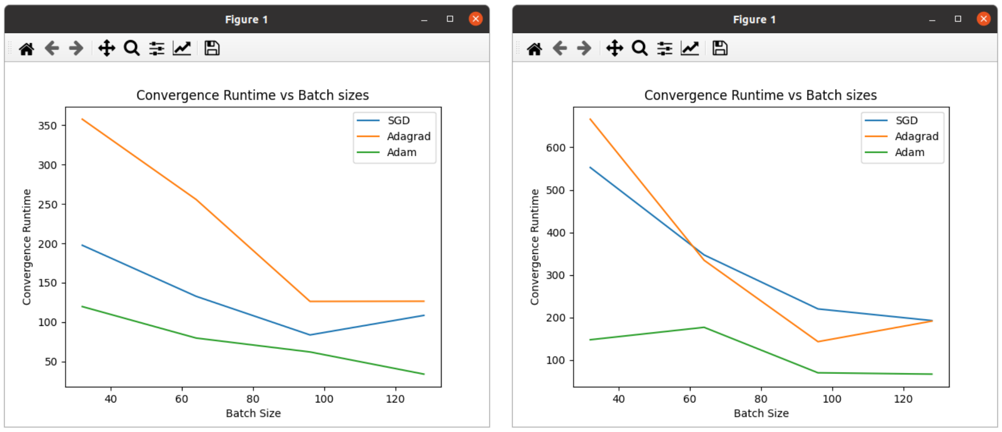

Figure 3: Convolutional Neural Network training runtime for the original (left) and the extended (right) stopping criteria

I also wanted to evaluate the accuracy of each optimizer I tested. I've included the tables that show the training accuracy and the total number of training epochs below. I found that Adam had the best classification accuracy, despite using fewer training epochs and runtime. I believe this is due to better weight initialization, because the Adam algorithm achieved an 87% accuracy at epoch 1, whereas the others started at around 20%. Adagrad tended to have the worse classification accuracy and runtimes. By nearly all performance measures, SGD sat in the middle of the other two optimizers.

	32	64	96	128		32	64	96	128
SGD	0.9347	0.9234	0.9202	0.9185		0.9650	0.9503	0.9455	0.9354
Adagrad	0.8994	0.8786	0.8779	0.8720		0.9165	0.8939	0.8809	0.8875
Adam	0.9759	0.9783	0.9819	0.9773		0.9775	0.9833	0.9817	0.9816

Table 1: Training accuracy for different batch sizes and optimizers. Original stopping criteria (left) and extended stopping criteria (right)

	32	64	96	128		32	64	96	128
SGD	57	61	71	98		192	165	193	179
Adagrad	111	116	104	112		199	159	122	175
Adam	33	36	50	30		44	88	59	61

Table 2: Training epochs for different batch sizes and optimizers. Original stopping criteria (left) and extended stopping criteria (right)