Computer Vision
CSCI-GA.2272-001
Assignment 2

Lakshay Sharma
ls4170@nyu.edu

Submitted: Nov. 2, 2017

1 Introduction

This assignment is an introduction to using PyTorch for training simple neural net models. Two different datasets will be used:

- MNIST digits [handwritten digits]
- CIFAR-10 [32x32 resolution color images of 10 object classes].

2 Requirements

You should perform this assignment in PyTorch; modify the ipython notebook provided.

3 Warmup [10%]

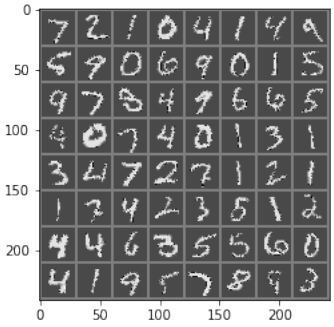
It is always good practice to visually inspect your data before trying to train a model, since it lets you check for problems and get a feel for the task at hand.

3.1

MNIST is a dataset of 70,000 grayscale hand-written digits (0 through 9). 60,000 of these are training images. 10,000 are a held out test set.

Solution:

Figure 1: MNIST dataset examples

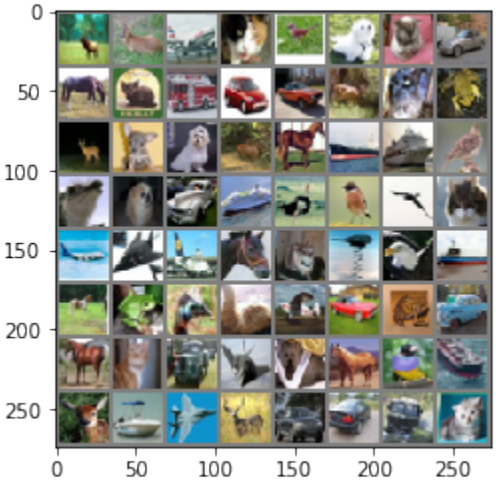


3.2

CIFAR-10 is a dataset of 60,000 color images (32 by 32 resolution) across 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck). The train/test split is 50k/10k.

Solution:

Figure 2: CIFAR-10 dataset examples



4 Training a Single Layer Network on MNIST [20%]

4.1

Start by running the training on MNIST. By default if you run the given notebook successfully, it will train on MNIST.

This will initialize a single layer model train it on the 50,000 MNIST training images for 10 epochs (passes through the training data).

The loss function (cross-entropy[1]) computes a logarithm of the softmax on the output of the neural network, and then computes the negative log-likelihood w.r.t. the given 'target'.

The default values for the learning rate, batch size and number of epochs are given in the "options" cell of the notebook. Unless otherwise specified, use the default values throughout this assignment.

Note the decrease in training loss and corresponding decrease in validation errors.

Solution:

Results for MNIST digit classification (single-layer network, epochs=10, loss=cross-entropy, learning-rate=0.01, batch-size=100)

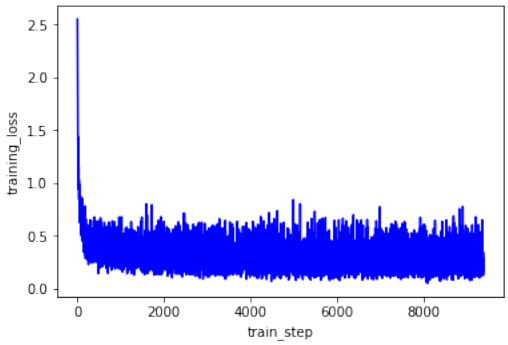


Figure 3: Training step vs. training loss

[1] http://pytorch.org/docs/master/nn.html?highlight=cross$_e$ntropytorch.nn.functional.cross$_e$ntropy

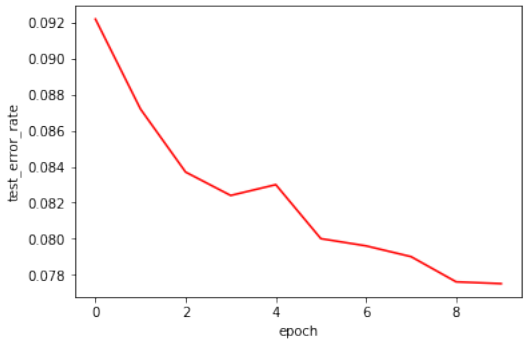


Figure 4: Training epoch vs. test error rate

4.2

Paste the following output into your report:

(a) Add code to plot out the network weights as images (one for each output, of size 28 by 28) after the last epoch. Grab a screenshot of the figure and include it in your report. Also report average test set loss.

Solution:
(after nearly 10 epochs of training)
Train Epoch: 10 [57600/60000 (96%)] Loss: 0.254032

(after 10 full epochs of training)
Test set: Average loss: 0.2747, Accuracy: 9225/10000 (92%)

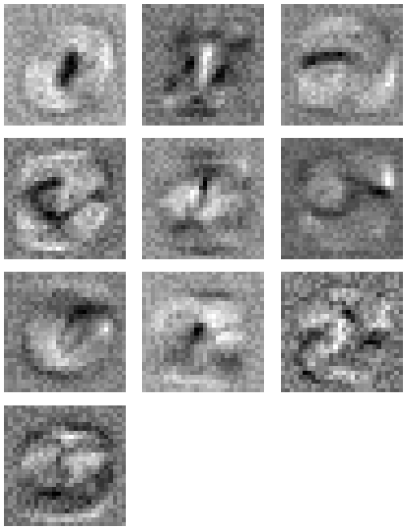


Figure 5: MNIST weights after last epoch

(b) Reduce the number of training examples to just 50. Paste the output into your report and explain what is happening to the model.

Solution:

(after nearly 10 epochs of training)
Train Epoch: 10 [57600/60000 (96%)] Loss: 0.025061

(At the end of 10 full epochs)
Test set: Average loss: 1.2312, Accuracy: 6276/10000 (63%)

As can be seen, at the end of almost 10 epochs of training, the training loss is much smaller than the case with the entire dataset. However, the test accuracy is much higher.

It is pretty evident that the model overfits to the training data, and therefore achieves low loss during training, but suffers from a high error rate during testing.

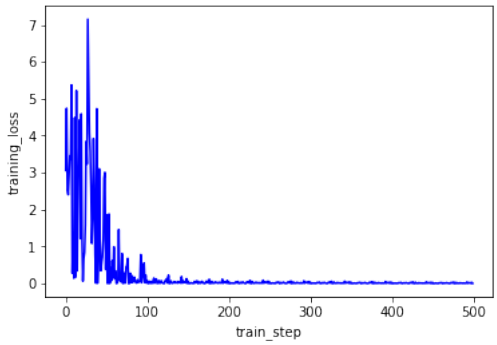


Figure 6: MNIST classification training performance (single layered network, limited to 50 batches, with a single sample per batch), learning rate = 0.01

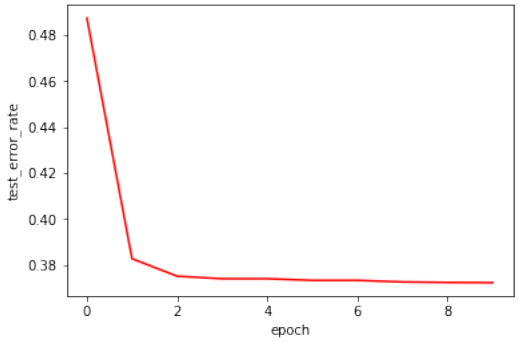


Figure 7: MNIST classification test performance (single layered network, limited to 50 batches, with a single sample per batch), learning rate = 0.01

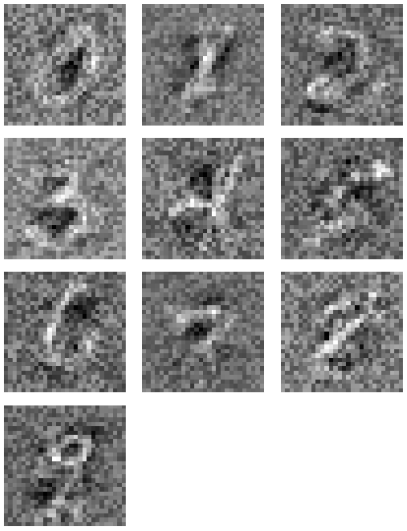


Figure 8: MNIST (limited to 50 samples per epoch) weights after last epoch

Compared to the previous case, here, the weights (in general) seem darker in shade (therefore higher in value). This further supports the finding that the model is over-fitting to the training data.

5 Training a Multi-Layer Network on MNIST [20%]

5.1

Add an extra layer to the network with 1000 hidden units and a 'tanh' non-linearity. Train the model for 10 epochs and save the output into your report.

Solution:

(after nearly 10 epochs of training)
Train Epoch: 10 [57600/60000 (96%)] Loss: 0.183416

(after 10 full epochs of training)
Test set: Average loss: 0.1608, Accuracy: 9556/10000 (96%)

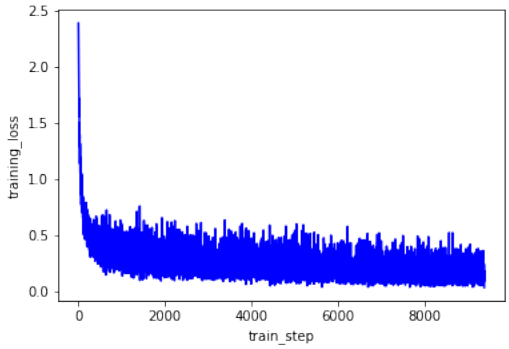


Figure 9: MNIST classification (one hidden layer (1000 units, tanh non-linearity)) training results

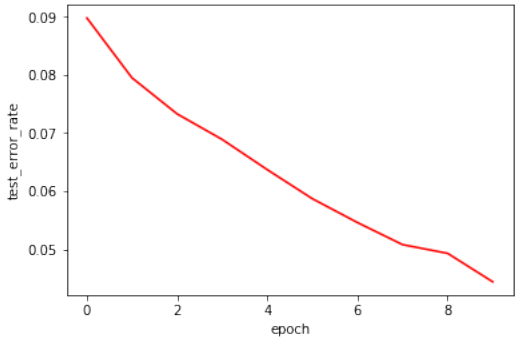


Figure 10: MNIST classification (one hidden layer (1000 units, tanh non-linearity)) test results

5.2

Now set the learning rate to 10 and observe what happens during training. Save the output in your report and give a brief explanation.

Solution:

(after nearly 10 epochs of training)
Train Epoch: 10 [57600/60000 (96%)] Loss: 156.709579

(after 10 full epochs of training)
Test set: Average loss: 143.9298, Accuracy: 7400/10000 (74%)

Both, the training loss and the test error rate, are much higher in this case.

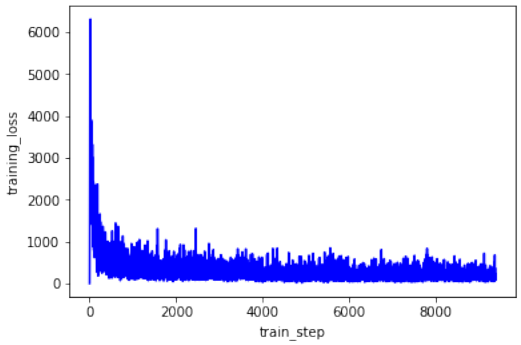


Figure 11: MNIST classification training (one hidden layer (1000 units, tanh non-linearity, learning rate = 10)) results

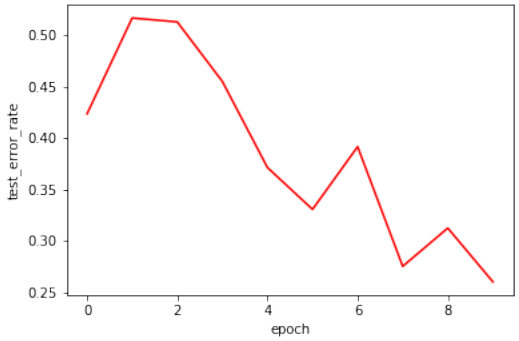


Figure 12: MNIST classification test (one hidden layer (1000 units, tanh non-linearity, learning rate = 10)) results

Due to the high learning rate, the steps being taken are too large, and the model parameters are unable to converge at a minima.This is also evident looking at the wild fluctuations in the training loss and test error rate.

6 Training a Convolutional Network on CIFAR10 [50%]

6.1

Create a convolutional network with the following architecture:

- Convolution with 5 by 5 filters, 16 feature maps + Tanh nonlinearity.
- 2 by 2 max pooling.
- Convolution with 5 by 5 filters, 128 feature maps + Tanh nonlinearity.

- 2 by 2 max pooling.
- Flatten to vector.
- Linear layer with 64 hidden units + Tanh nonlinearity.
- Linear layer to 10 output units.

Train it for 20 epochs on the CIFAR-10 training set and copy the output into your report, along with an image of the first layer filters.

Solution:
(after nearly 20 epochs of training)
Train Epoch: 20 [44800/50000 (90%)] Loss: 0.801686

(at the end of 20 full epochs)
Test set: Average loss: 0.8414, Accuracy: 35404/50000 (71%)

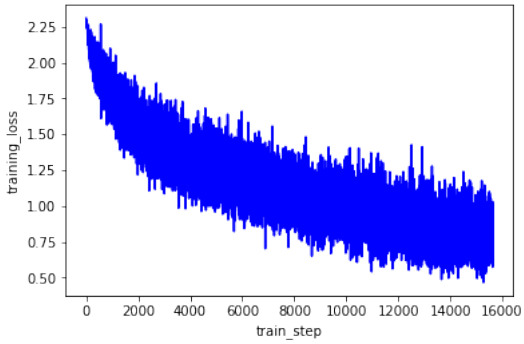


Figure 13: CIFAR10 object recognition train results (multi-layer network, epochs=20, loss=cross-entropy, learning-rate=0.01, batch-size=64)

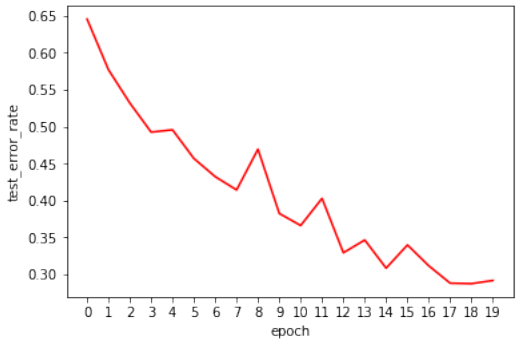


Figure 14: CIFAR10 object recognition train results (multi-layer network, epochs=20, loss=cross-entropy, learning-rate=0.01, batch-size=64)

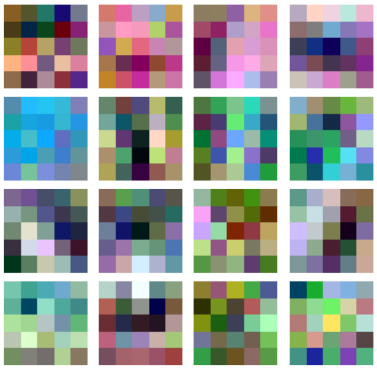


Figure 15: First layer weights (16 filters, each a 5x5 convolution with stride=1 and padding=0) for CIFAR10 object recognition model

6.2

Give a breakdown of the parameters within the above model, and the overall number.

Solution: The network consists of the following layers (in order), with associated (learnable) parameters:

- Convolution layer (`conv1`): has 16 3-channel convolution filters, each of dimension 5x5. There is a bias parameter for each of these 16 filters
- Convolution layer (`conv1`): has 128 16-channel convolution filters, each of dimension 5x5. There is a bias parameter for each of these 128 filters.

- Linear layer (`fc1`): takes as input a flattened vector of dimension 1x3200, and produces an output vector of dimension 1x64. Thus, the parameter matrix for this layer has dimensions 3200x64. Each of the 64 outputs has an associated bias.
- Linear layer (`fc2`): takes as input a flat vector of dimension 1x64, and produces an output vector of dimension 1x10. Thus, the parameter matrix for this layer has dimension 64x10. Each of the 10 outputs has an associated bias.

The following table summarizes the total learnable parameters in the model:

Layer	Param-matrix	Biases	Total
conv1	16*3*5*5	16	1,216
conv2	128*16*5*5	128	51,328
fc1	3200*64	64	204,864
fc2	64*10	10	650
Overall			258,058