MNIST Digits Classification with CNN

Introduction

In this homework, I build a CNN network and compare it with a comparable MLP network.

Implementation Notes

I have relied on the *im2col* functionality to perform the convolution. Its advantage over nested loops is **speed**; and the disadvantage is **memory inefficiency**.
The forward and backward passes of the average pooling layers are implemented as convolutions.
The associated functions are in *functions.py*.

Hyperparameter Selection

In theory, to select the right hyperparameters, a full exhaustive search that iterates through every combination should be done with a **validation dataset**. However, for simplicity's sake, this is done mainly on an ad hoc basis, where the parameters work well enough in combination.
Hyperparameters of interests are reproduced in *Table1*.

Results Summary

Table1: Network details

Network	CNN	MLP
Architecture	conv -> reLu -> avgpool -> conv -> reLu -> avgpool -> linear	linear -> reLu -> linear -> reLu -> linear
Loss	SoftmaxCrossEntropy	SoftmaxCrossEntropy
Learning rate	0.01	0.01
Batch size	100	100
Weight decay	0	0
Momentum	0.9	0.9
Max epoch	20	30
>97% test acc	After ~8 epochs	After ~17 epochs
Time per epoch	101s	7s
Time to reach >97% test acc	~800s	~120s
Top test acc	98.30%	98.34%
Lowest test loss	0.0586	0.0527

Comments and Discussion

Training time & Convergence:
CNN takes much longer time per epoch. This is a somewhat unfair comparison as CNN has more layers. That said, there could be a faster implementation of CNN out there.
CNN converges (defined as attaining >any% acc) after fewer epochs than MLP.

No. of Parameters:
MLP has more parameters, CNN does parameter fixing.

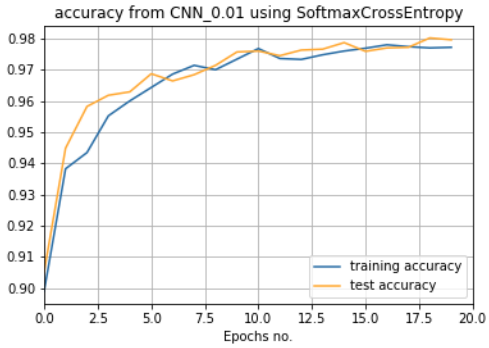
Accuracy:
MLP could attain better accuracy if left to train longer. This is unsurprising, as MLP is more general and can learn the weights of CNN if it is indeed the best model. The tradeoff is more parameters to learn, and possibly longer training.

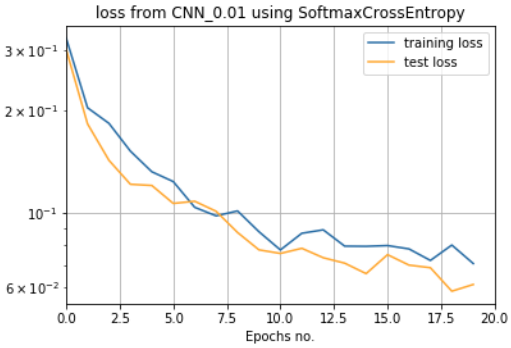
Initialization:
When *init_std = 0.01* is used, the network is often stuck at random guessing for repeated epochs. *init_std = 0.1* solves this problem. It is unclear to me how more random initial weights benefits training, as is the case here.

Overfitting:
Overfitting plagues MLP but not so much for CNN.

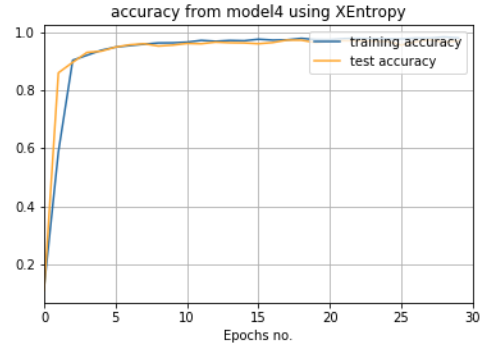
Plots
The loss (training & test) and accuracy (training & test) of each setup are plotted.

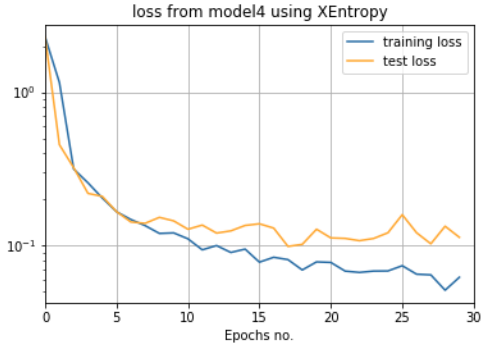
CNN





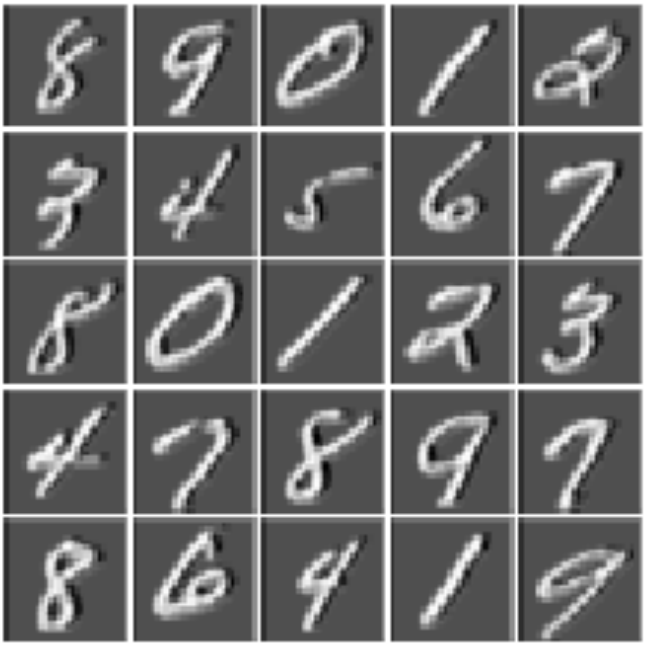
MLP





Output from CNN after reLu1 layer

We plot the output of *relu1* layer, where the 4 output channels have been collapsed into 1. Each pixel is basically a weighted average of its surrounding pixels.



Reference:

* [Deep Learning](http://www.deeplearningbook.org/)
* [CS231n](http://cs231n.github.io/convolutional-networks/)
* [Agustinus Kristiadi's Blog]
(https://wiseodd.github.io/techblog/2016/07/16/convnet-conv-layer/)
* [Caffe tutorial]
(https://nbviewer.jupyter.org/github/BVLC/caffe/blob/master/examples/00-classification.ipynb)