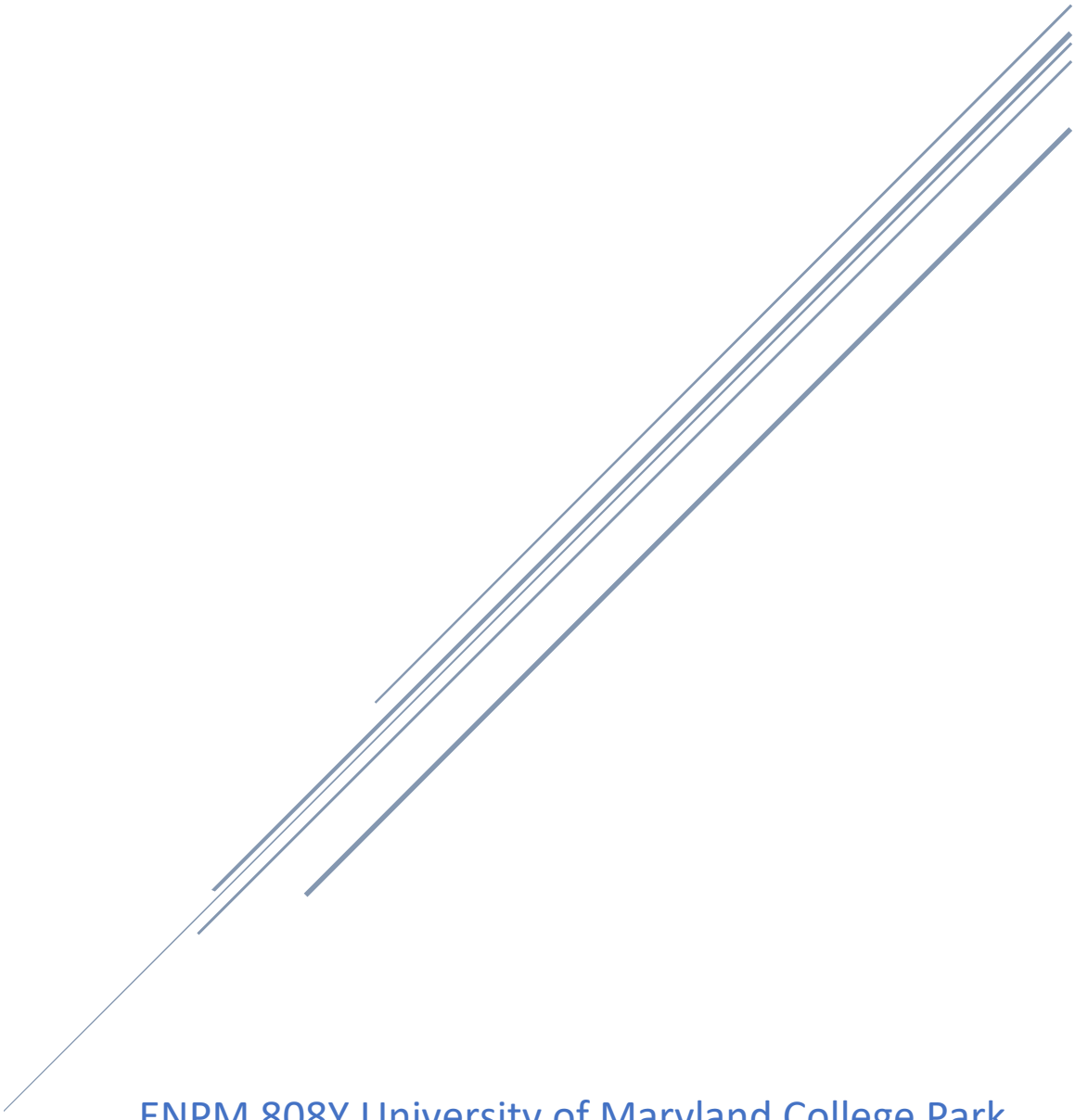
NEURAL NETWORKS HOMEWORK 2

Convolutional Neural Networks for MNIST and CIFAR data



ENPM 808Y University of Maryland College Park
Timothy Werder

Table of Contents

Problem 1: MNIST

Confusion Table

Train Label	Train Predicted	Test Label	Test Prediction	Test top 3
5	5	2	2	[2, 1, 8],
9	9	5	5	[5, 6, 8],
1	1	1	1	[1, 2, 3],
6	6	7	7	[7, 2, 3],
7	7	7	7	[7, 8, 9],
2	2	3	3	[3, 5, 8],
4	4	5	5	[5, 8, 3],
1	1	8	5	[5, 8, 9],
9	9	6	6	[6, 5, 8],
0	0	7	7	[7, 9, 3],
5	5	0	0	[0, 8, 9],
2	2	2	2	[2, 7, 1],
2	8	0	0	[0, 6, 8],
5	5	5	9	[9, 5, 8],
6	6	7	7	[7, 3, 2],
5	5	1	1	[1, 2, 3]
1	1			
5	5			
3	3			
3	3			
5	5			
6	6			
8	8			
1	1			
6	6			
5	5			
8	8			
3	3			
2	2			
0	0			
0	0			
6	0			

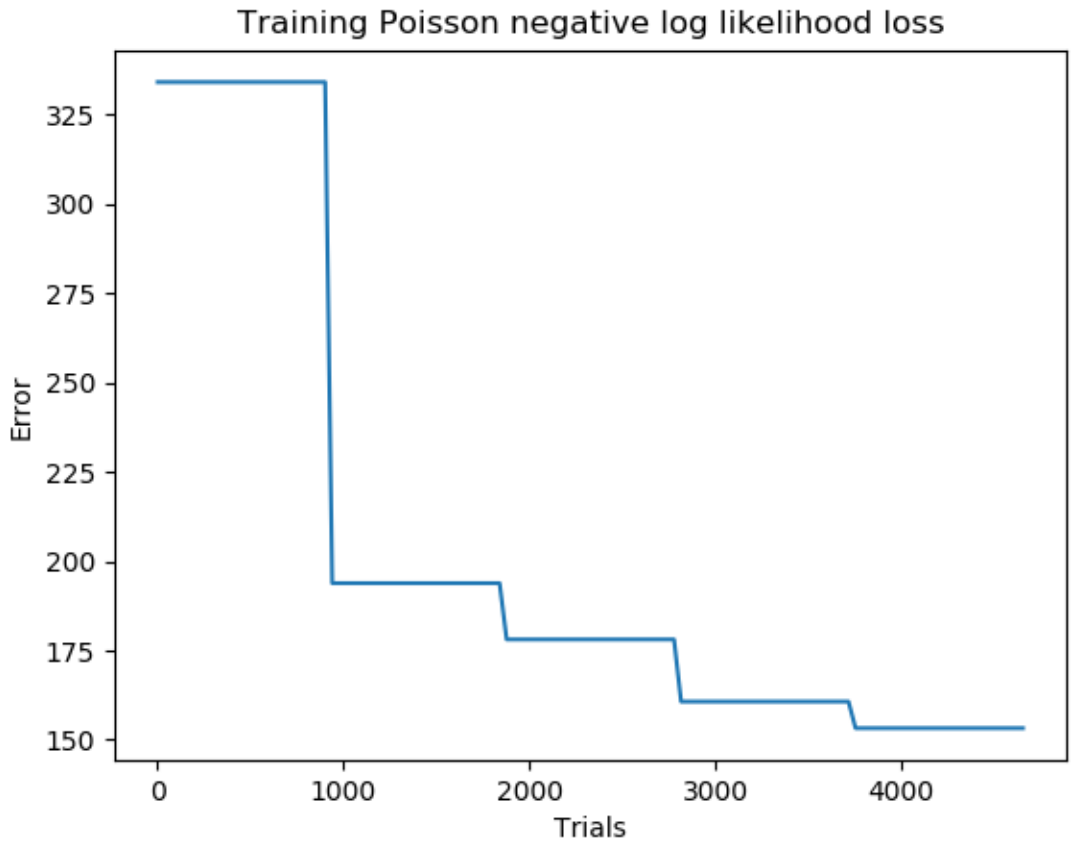
Accuracy Table

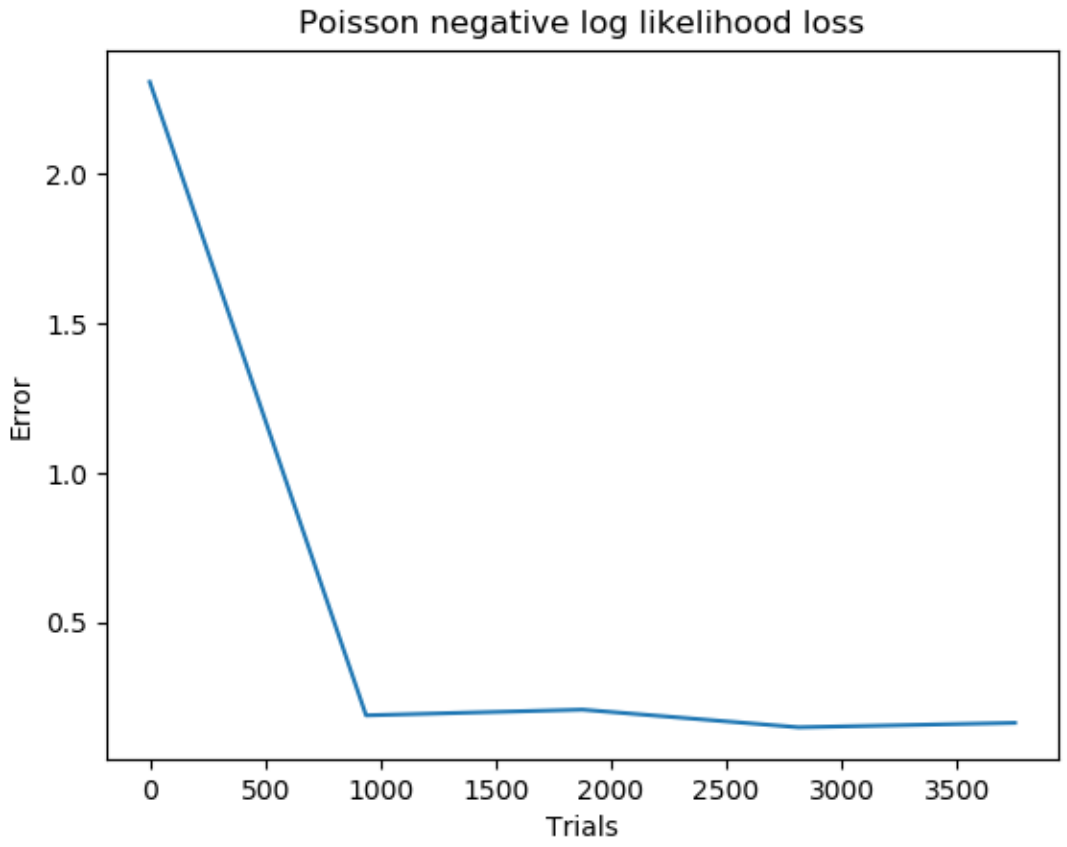
Class	0	1	2	3	4	5	6	7	8	9	Mean
Training	100	100	75	100	100	100	80	100	100	100	95.5
Test	96.32	98.67	95.15	95.34	93.38	96.86	96.34	96.49	94.76	95.83	95.91

Hyper Parameters

Learning Rate= 0.01	Seed: 1	Layering:
Time to Train: 489.388 s	Epochs: 5	Conv, relu, pool, conv, relu, pool, view, linear, relu, linear, softmax
Batch Size: 64	Optimizer: Adam	

Training and Validation loss





Discussion

I started by designing the network by creating the layers; examples called for repeated calls of convolution, relu and pooling. I then reshape the output then linearize in two steps down to the 10 classes I then use a softmax function to classify each class. I then optimize using Adam (AdaDelta with momentum) to avoid local minimums due to similar looking structures. I then perform the forward pass and check the loss using a Poisson negative log likelihood model (this takes into account that the output of the forward pass is a logarithmic softmax).

By using a more intense layering structure I was able to use less epochs, smaller batches, and overall decrease the training time while maintaining a high degree of accuracy. Finally, the training data was split-up to have a 64 batche size of shuffled data and validated against all of the training data with test data downloaded separately.

Problem 2: CIFAR

Confusion Table

Train Label	Train Predicted	Test Label	Test Prediction	Test top 3	Test Classification	Test Prediction Classification
6 9 9	6 9 9	9	1	[1, 4, 9],	Truck	Car
0 9 6	0 9 6	7	7	[7, 4, 9],	Horse	Horse
4 7 2	7 7 2	8	8	[8, 0, 9],	Ship	Ship
3 1 7	5 1 7	5	4	[4, 5, 3],	Dog	Deer
1 8 9	1 8 9	7	7	[7, 4, 5],	Horse	Horse
3 5 5	7 5 5	3	5	[5, 1, 3],	Cat	Dog
0 4 5	0 4 3	8	8	[8, 0, 9],	Ship	Ship
0 6 1	0 6 1	1	1	[1, 8, 0],	Car	Car
9 9 2	9 9 0	2	7	[7, 2, 5],	Bird	Horse
7 3 9	2 3 9	6	6	[6, 4, 3],	Frog	Frog
2 2 5	2 2 5	0	0	[0, 2, 4],	Plane	Plane
4 1 4	4 1 4	8	8	[8, 3, 0],	Ship	Ship
0 1 8	4 1 8	6	6	[6, 2, 3],	Frog	Frog
0 2 1	0 2 1	5	5	[5, 3, 2],	Dog	Dog
5 2 6	5 2 6	3	5	[5, 3, 4],	Cat	Dog
8 2 4	8 2 4	3	2	[2, 0, 8]	Cat	Bird
7 3 7	7 3 7					
1 0 5	0 0 5					
3 7 2	3 7 2					
6 0 7	6 8 7					
5 5 9	3 7 9					
3 9 5	3 1 4					
8 2 2	8 2 2					
5 5 2	7 5 2					
4 1 7	4 1 7					
5 9 5	6 9 5					
4 2	4 5					

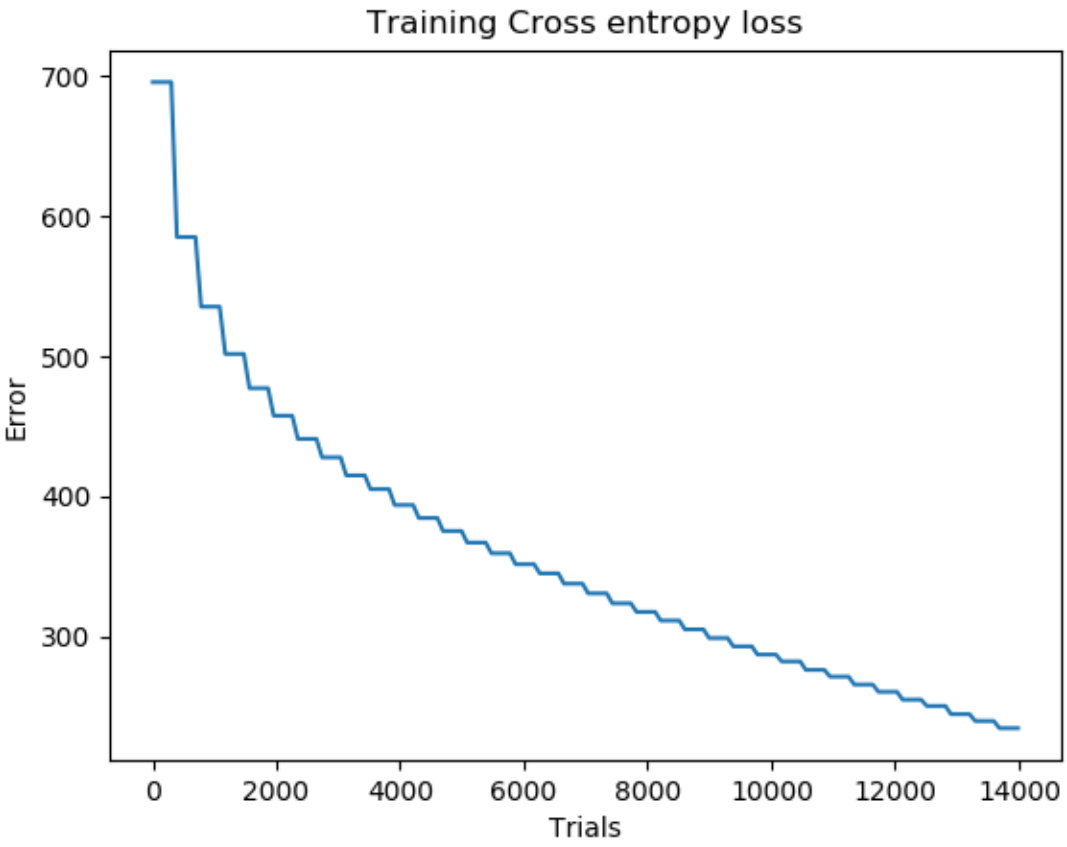
Accuracy Table

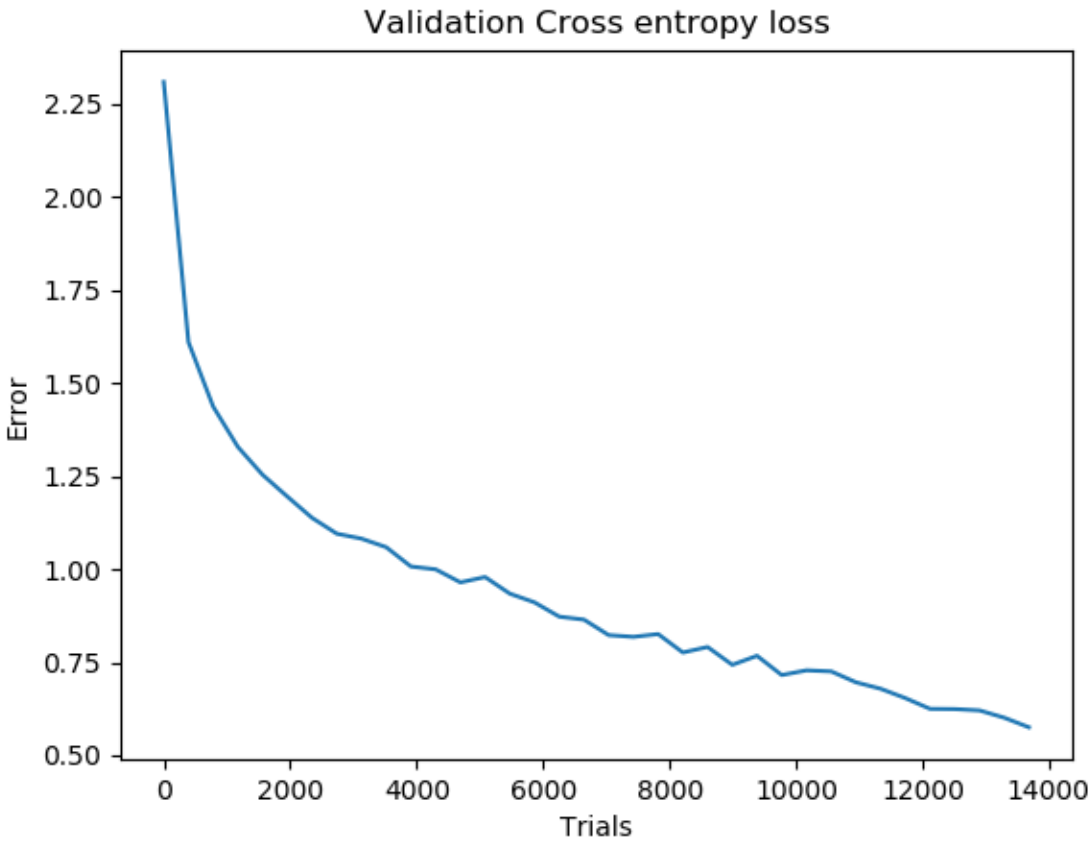
Translation	plane	car	bird	cat	deer	dog	frog	horse	ship	truck	
Class	0	1	2	3	4	5	6	7	8	9	Mean
Training	71.42	87.5	83.33	66.66	85.71	53.86	100	87.5	100	90	82.59
Test	64.3	77.3	54.4	42.0	60.8	54.6	71.1	68.4	70.2	69.2	63.23

Hyper Parameters

Learning Rate= 0.1	Seed: 1	Layering:
Time to Train: 2689.7 s	Epochs: 36	Conv, relu, pool, view, linear, relu, linear
Batch Size: 128	Optimizer: Adadelta	

Training and Validation loss





Discussion

Like the MNIST data, I started by designing the network by creating the layers; unlike the MNIST data using multiple layer increased the output (before linearization) above 12,000. This drastically increased the training time. I then reshape the output then linearize in two steps down to the 10 classes. I then optimize using AdaDelta. I then perform the forward pass and check the loss using cross entropy. These parameters were changed from the MNIST as using a SoftMax function to aid in classification caused an extreme drop off in accuracy.

In order to compensate for a smaller layering, I increased the number of epochs and the learning rate. After trial and error, 36 epochs were the goldilocks zone before loss began increasing. I also tried the lowering the learning rate to 0.01, however after 100 epochs, I was unable to achieve the same/better results without significantly more training time being needed exceeding the already lengthy training time of 45 minutes. Additionally, compared to the MNIST CNN, a batch size of 128 was used as smaller batch sizes (64) would fail to train on all input data types.

Finally, the training data was split-up to have 128 batche size of shuffled data and validated against all of the training data with test data downloaded separately.

Key takeaways

Several takeaways I took from this are how the Pytorch works and several of the various functions that can be used in CNN's.

I also came to understand how structuring different CNN's can have a significant affect on how well a CNN will perform. This is particularly targeted towards pictures with strong easily quantifiable information (black numerals on a white background) and complex imagery (dogs vs. cats).