Investigating the Model Complexity

*Submitted by Abdulrahman Alabrash*

# OVERVIEW & PURPOSE

We saw in the lecture that multilayer perceptrons are universal function approximators. A two-layer MLP is sufficient to approximate any well-behaved function given enough hidden units. However, the required number of units may not be reasonable. Adding extra non-linear layers, using sparse connections and repeated weights may bring us extra representational power and may provide us with a more efficient model in terms of the number of parameters.

[MNIST](http://yann.lecun.com/exdb/mnist/) is a dataset for handwritten digits used since the late 1990’s and we know the performance of many different machine learning models on this task. Here is a question nobody knows the answer to: “What is the model with the fewest parameters that can classify all 60,000 training examples correctly?”

In this assignment, you will try to find the architecture with the minimum number of parameters that is sufficient to classify the MNIST training data without any errors. [Nielsen, (2016)](http://neuralnetworksanddeeplearning.com/chap6.html) has some MNIST experiments with very clear explanations implemented in Theano.

# 

# 

# Investigating the Model Complexity

The main purpose of this assignment is finding a model with the minimum number of parameters that is able to learn to classify MNIST training data %100 accurately. Before this, you will follow the steps given below to understand how changing model complexity might affect the learning capability. For each step express the total number of parameters of the model.

In part 5, you will implement a convolutional neural network and explore the learning curve of the model. Plot[[1]](#footnote-0) two figures such as epoch vs train and test losses (negative log likelihood), and epoch vs train and test misclassification errors. You can create a table for the experiments instead of plotting figures for the lab session. Use standard sgd with learning rate= 0.15 for updating model parameters. Train the model on mini-batches and use 100 images in each minibatch. Pass over all data 100 times (i.e. epochs=100) to train the model. You can start with the code that you implemented in the previous lab session.

**1) A softmax classifier.**

*Total Number of Parameters:784\*10+10=7850*

**2) A MLP with a hidden layer that has 30 units.**

*Total Number of Parameters:(30x784)+30+(10x30)+10*

**3) A MLP with a hidden layer that has 60 units.**

*Total Number of Parameters:(60x784)+60+(10x60)+10*

**4) A MLP with two hidden layers each has 30 units.**

*Total Number of Parameters:(30x784)+30+(30\*30)+30+(10x30)+10*

**5) Implementing a Convolutional Neural Network** that has a conv4 and a pool layer where there are three 5x5 filters. Use 2x2 pooling and use sigmoid as activation function. Use stride=1(default) for the conv4 and stride=2(default) for the pool operation. Use padding=0 for the conv4 operation. Feed the output of the pool layer to a softmax classifier. General order of the operations (cbfp) in a convolutional neural network as following**:**

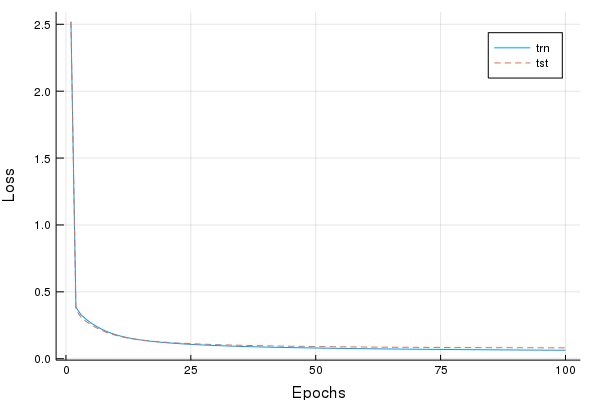
* Apply convolution (c)
* Add bias (b)
* Apply the activation function (f)
* Apply the pooling (p)

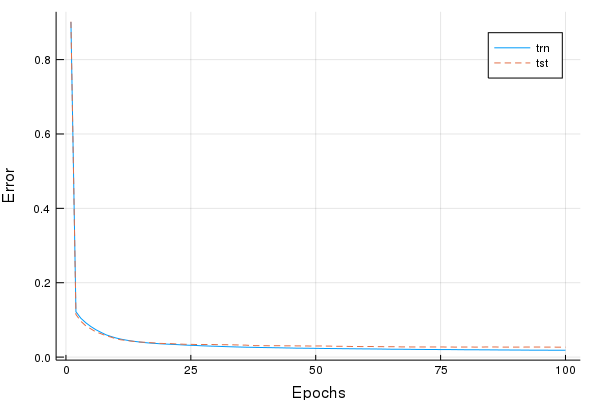
You can use the following code for the minibatching:

**include(Knet.dir("data","mnist.jl"))#include mnist related functions**

**dtrn, dtst = mnistdata(;batchsize=100, xtype=atype)#data ready to use in convolutional neural networks**

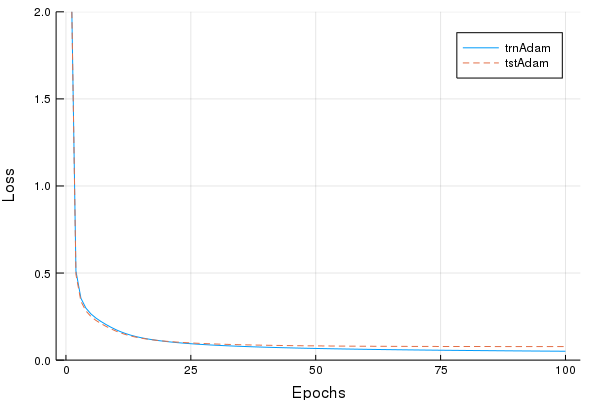
***Total Number of Parameters:12\*12\*3\*1***

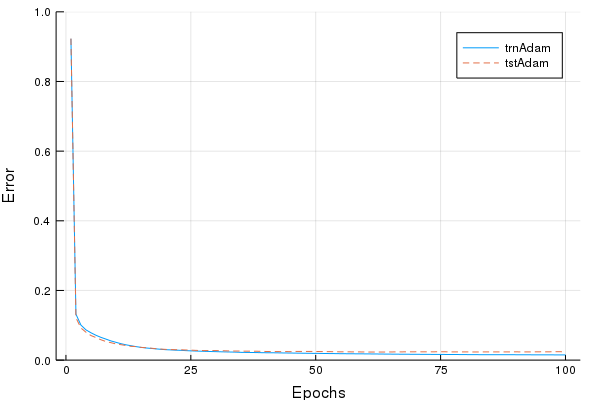
******

******

**6) Train the model with** [**Adam**](http://denizyuret.github.io/Knet.jl/latest/reference/#Knet.Adam) **optimizer using default values. Plot same figures as you did in the part 5.**

If you store your parameters as a weight array, you can create optimization parameters and update your parameters as following:

******

******

**7) Main task: Finding the model with minimum number of parameters, that gets 0 error on all 60,000 examples of the MNIST training data (not the test data).**

Hint: Implement flexible Knet models using keyword arguments to be able to change the model definition without changing any code. So, you can test different models just specifying different parameters.

Some guidance:

* Try different activation functions that are available in Knet like sigm, tanh, relu.
* Test different number of hidden layers 1,2, …
* Test different number of units in each hidden layer, e.g. first hidden layer=50, second hidden layer=100, …
* Try to stack cbfp layers.
* You may need to play with learning rate to train different models.
* You may want to use other optimization methods such as adam, rmsprop, adagrad, etc. (e.g. See the doc for [optimization methods](http://denizyuret.github.io/Knet.jl/latest/reference/#Model-optimization-1))

Please fill the following table according to your experiments that you are able to get 0 error on the training data.

|  |  |  |
| --- | --- | --- |
| Model Description  (MLP, Convolutional Neural Network, number of hidden layers, hidden sizes, number of filters, filter sizes, etc.) | Total Number of Parameters | Experiment Settings (number of epochs, learning rate, optimization method, minibatch size, etc.) |
| Mlp with two hidden layers, hidden sizes 50, 50,  activation functions relu, relu | 1000[[2]](#footnote-1) | Standard sgd, lr=0.1, batchsize=100, epochs=100 |
| Conv(3,3,1,20),  Conv(3,3,20,40),  Dense(1000,10,identity) | (3\*3\*1\*20)+20  +(3\*3\*20\*40)+40  +(1000\*10)+10 | Standard sgd,lr=0.15, batchsize=100,epochs=100 |
| Conv(3,3,1,20),  Conv(3,3,20,40,sigm),  Dense(1000,10,identity) | (3\*3\*1\*20)+20  +(3\*3\*20\*40)+40  +(1000\*10)+10 | Standard sgd,lr=0.15, batchsize=100,epochs=100 , sogmoid |
| Conv(3,3,1,20),  Conv(3,3,20,80),  Dense(2000,1000,identity),  Dense(1000,10) | (3\*3\*1\*20)+20  +(3\*3\*20\*80)+80  +(2000\*1000)+1000+(1000\*10)+10 | Standard sgd,lr=0.15, batchsize=100,epochs=100 |
|  |  |  |

# 

1. You can use whatever you want for plotting. Some suggestions: [Plots](https://github.com/JuliaPlots/Plots.jl) or [UnicodePlots](https://github.com/Evizero/UnicodePlots.jl) package or google spreadsheet. [↑](#footnote-ref-0)
2. This is an arbitrary number for the example, you should calculate the exact number for your model [↑](#footnote-ref-1)