# CS7015 Deep Learning
# Programming Assignment 1
# FeedForward Neural Network

CS17S008 Nitesh Methani
CS17S013 Pritha Ganguly

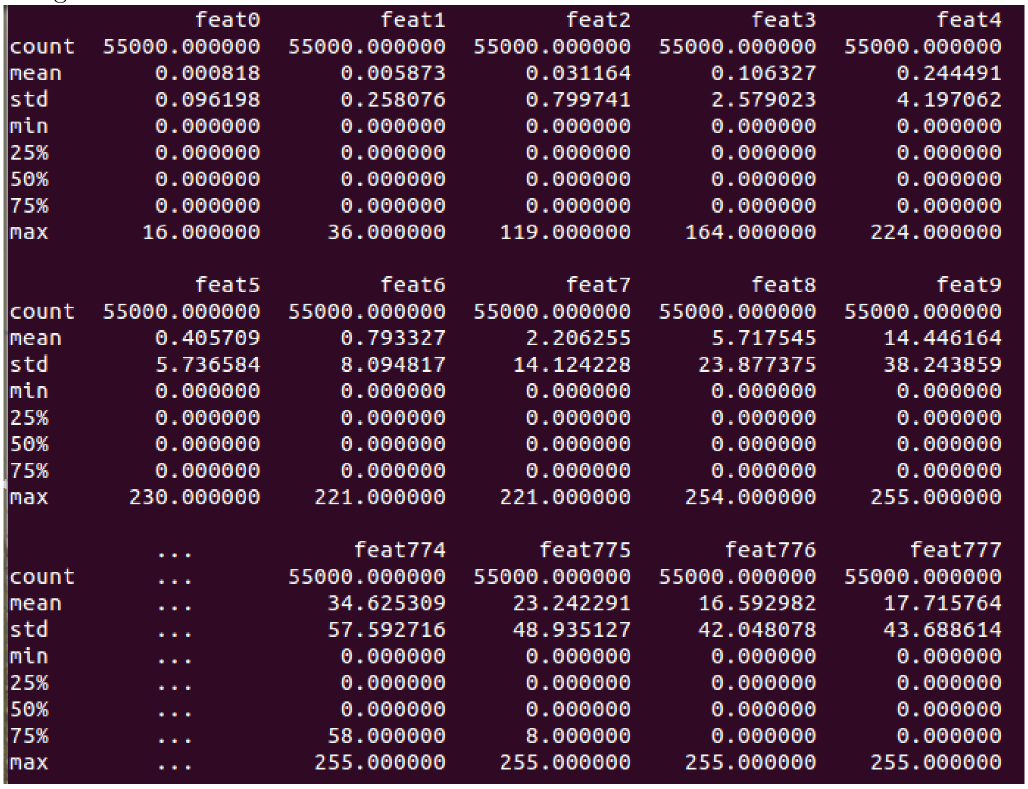
February 17, 2018

## Contents

## List of Figures

## 1 Knowing your Data

For training a classification model, it is always helpful if we know our data. We can make the following observations from the FASHION MNIST dataset :



```
              feat0          feat1          feat2          feat3          feat4
count  55000.000000   55000.000000   55000.000000   55000.000000   55000.000000
mean       0.000818       0.005873       0.031164       0.106327       0.244491
std        0.096198       0.258076       0.799741       2.579023       4.197062
min        0.000000       0.000000       0.000000       0.000000       0.000000
25%        0.000000       0.000000       0.000000       0.000000       0.000000
50%        0.000000       0.000000       0.000000       0.000000       0.000000
75%        0.000000       0.000000       0.000000       0.000000       0.000000
max       16.000000      36.000000     119.000000     164.000000     224.000000

              feat5          feat6          feat7          feat8          feat9
count  55000.000000   55000.000000   55000.000000   55000.000000   55000.000000
mean       0.405709       0.793327       2.206255       5.717545      14.446164
std        5.736584       8.094817      14.124228      23.877375      38.243859
min        0.000000       0.000000       0.000000       0.000000       0.000000
25%        0.000000       0.000000       0.000000       0.000000       0.000000
50%        0.000000       0.000000       0.000000       0.000000       0.000000
75%        0.000000       0.000000       0.000000       0.000000       0.000000
max      230.000000     221.000000     221.000000     254.000000     255.000000

                ...        feat774        feat775        feat776        feat777
count           ...   55000.000000   55000.000000   55000.000000   55000.000000
mean            ...      34.625309      23.242291      16.592982      17.715764
std             ...      57.592716      48.935127      42.048078      43.688614
min             ...       0.000000       0.000000       0.000000       0.000000
25%             ...       0.000000       0.000000       0.000000       0.000000
50%             ...       0.000000       0.000000       0.000000       0.000000
75%             ...      58.000000       8.000000       0.000000       0.000000
max             ...     255.000000     255.000000     255.000000     255.000000
```

Figure 1: Statistics of the data

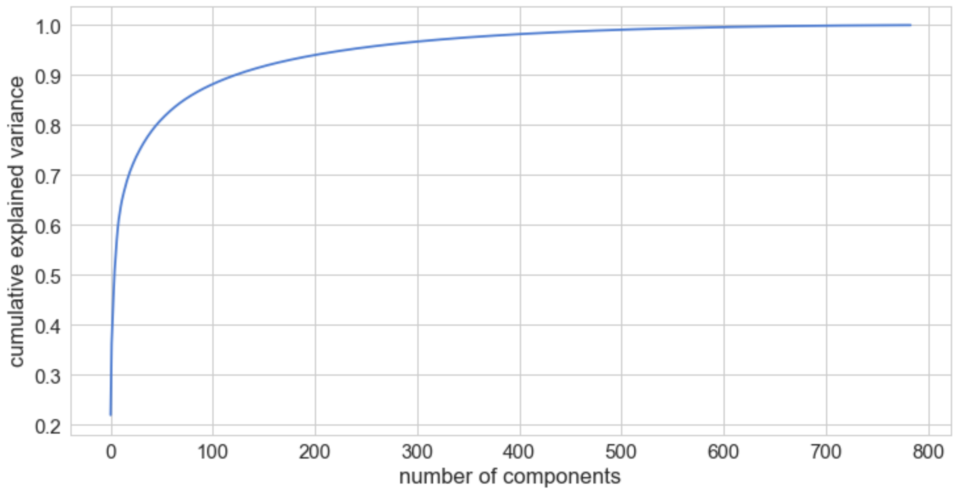


Figure 2: Choosing the no. of components from training data

- The images are 28 X 28 in size which means there are 784 features.
- The training dataset contains 55k instances while the validation set contains 5500 instances.
- While some pixels have high standard deviation with high mean (e.g. in Figure 1, feat774), some have low standard deviation with low mean (e.g. feat0). This suggests that not all the pixels are very informative and also that they are on different feature scales.
- It can be observed from Figure 2 that using only top 550 components we are describe 100% of the data. Therefore, it will be helpful if we reduce the dimensionality of the feature space from 784 to 550.

## 2 Data Normalization

In the FASHION MNIST dataset, the min and max of every feature is very different from the min and max of another feature. It is always a good practice to normalize the data when the features are on different scales. We have mean normalized our data so that training of the network can be accomplished in fewer epochs and with minimal oscillations.

## 3 Data Shuffling

The optimization algorithms considered for training are their respective mini-batch variants which means they rely on stochastic gradient descent, which means that they rely on the randomness to find a minimum. Shuffling mini-batches makes the gradients more variable, which can help convergence because it increases the likelihood of hitting a good direction, i.eit tries to ensure that we do not end up with bad batches.

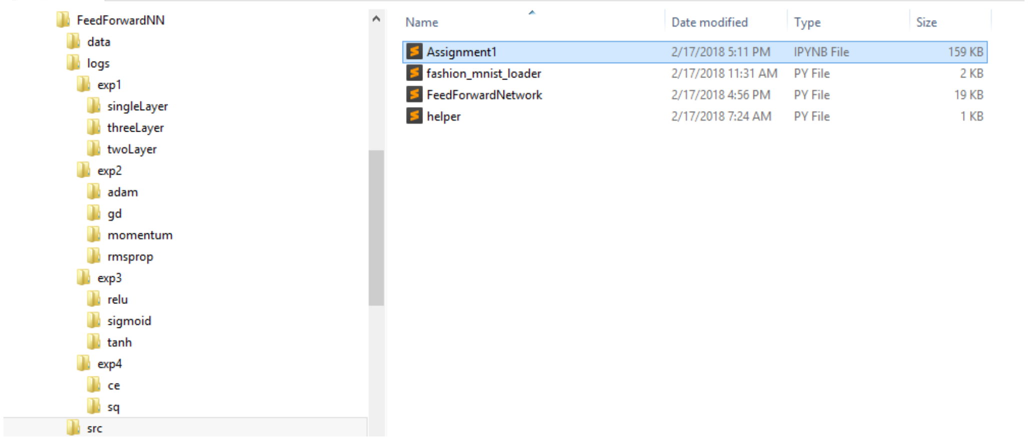
## 4 Random Initialization

Initializing the parameters of the neural network, namely the weight matrices and biases is an important step in training the network. The neurons in the hidden layer become symmetric if they the weights initialized to zero. On the other hand, if the weights are large, the algorithm might end up in the plateau which implies that the learning will become very slow. Therefore, we have used random initialization for our weights. The biases, on the other hand has been initialized with zeros. To ensure that the initial weights are small, we have multiplied them with some constant which is inversely proportional to the number of neurons in the network.

## 5 Directory Structure

The source code has been organized in the following directory structure:

(297).png



```
FeedForwardNN
  data
  logs
    exp1
      singleLayer
      threeLayer
      twoLayer
    exp2
      adam
      gd
      momentum
      rmsprop
    exp3
      relu
      sigmoid
      tanh
    exp4
      ce
      sq
  src
```

| Name | Date modified | Type | Size |
|---|---|---|---|
| Assignment1 | 2/17/2018 5:11 PM | IPYNB File | 159 KB |
| fashion_mnist_loader | 2/17/2018 11:31 AM | PY File | 2 KB |
| FeedForwardNetwork | 2/17/2018 4:56 PM | PY File | 19 KB |
| helper | 2/17/2018 7:24 AM | PY File | 1 KB |

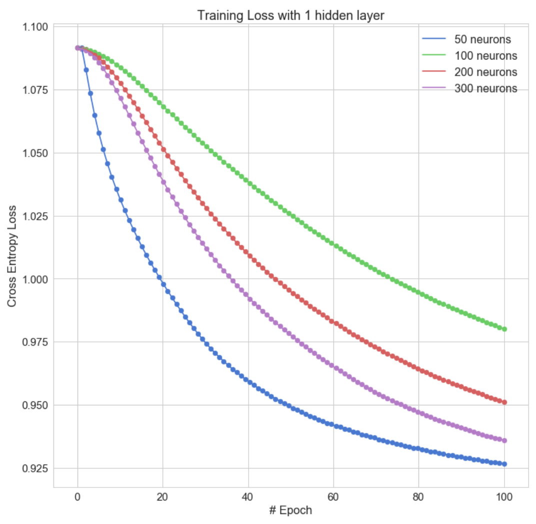
Figure 3: Directory structure

## 6 Vectorization

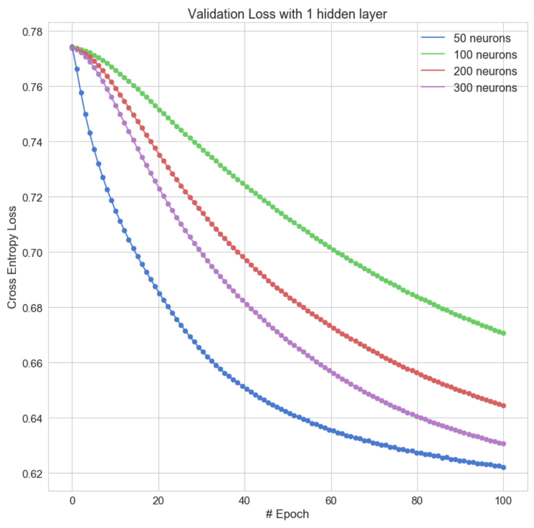
Vectorization of inputs allows us to efficiently compute the gradients on m samples at one go. Therefore, it helps us to train the network with all training samples without the use of for loops. By vectorizing our code, we can use the power of linear algebra (namely broadcasting capabilities, dot product of numpy python library).

# 7 Experiments

## 7.1 Experiment 1

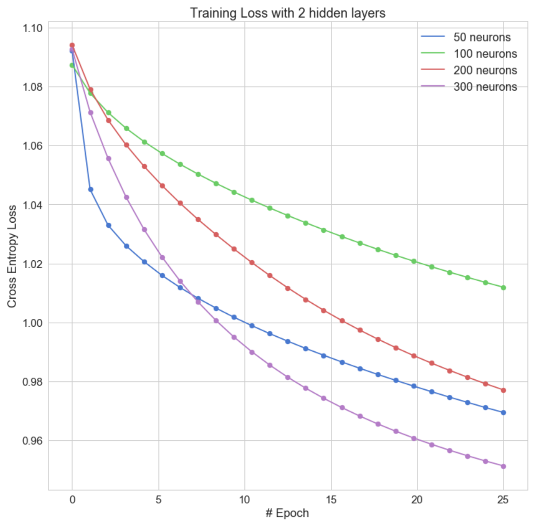


(a) Train loss for different hidden neurons

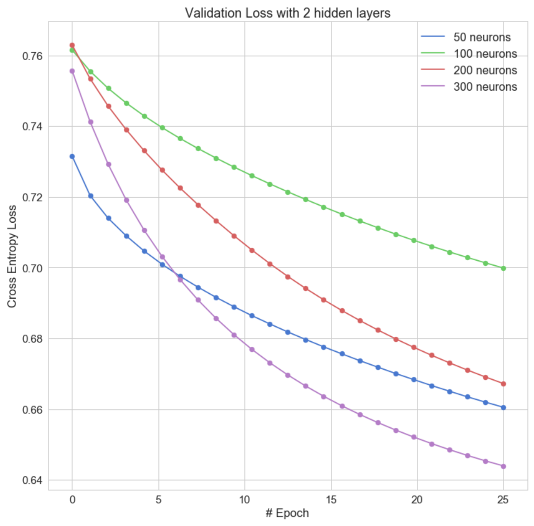


(b) Validation loss for different hidden neurons

The ADAM algorithm when run with a learning rate $\eta = 0.001$ converges faster for a network with a single hidden layer of 50 neurons than for any other number. This trend is uniform with respect to both training and validation losses.

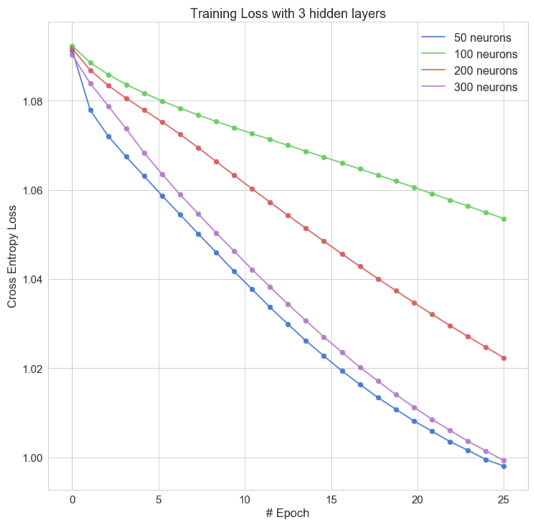


(a) Train loss for different hidden neurons

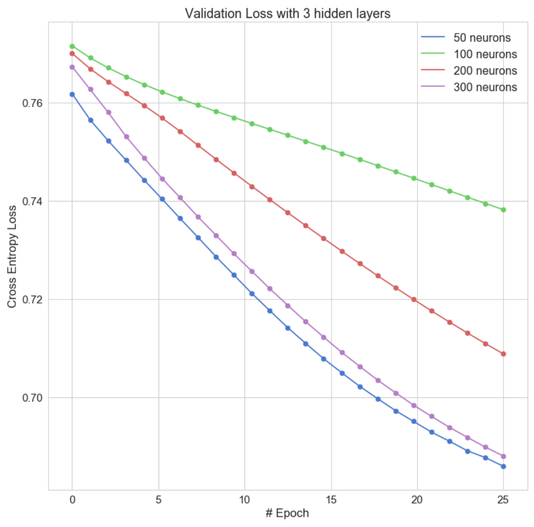


(b) Validation loss for different hidden neurons

We can observe that as we increase the number of hidden layers to 2 with 300 neurons each, i.e as our model become heavy, ADAM starts performing better as compared to other architectures. As we increase the number of hidden layers, the network is able to learn more features and hence it converges in lesser number of epochs.

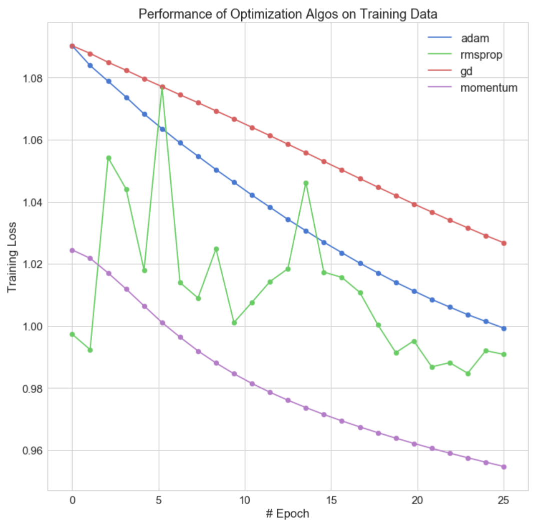


(a) Train loss for different hidden neurons

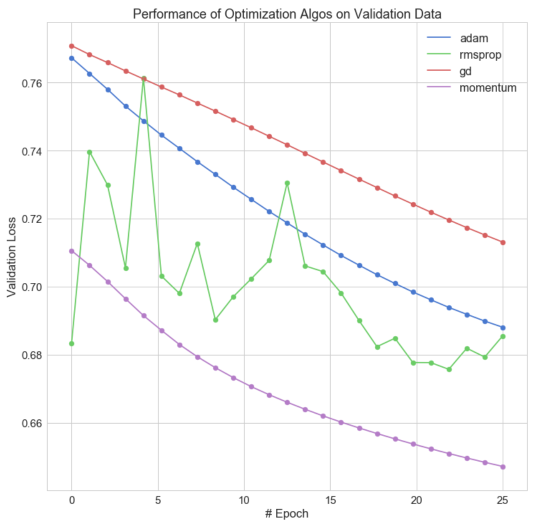


(b) Validation loss for different hidden neurons

## 7.2 Experiment 2



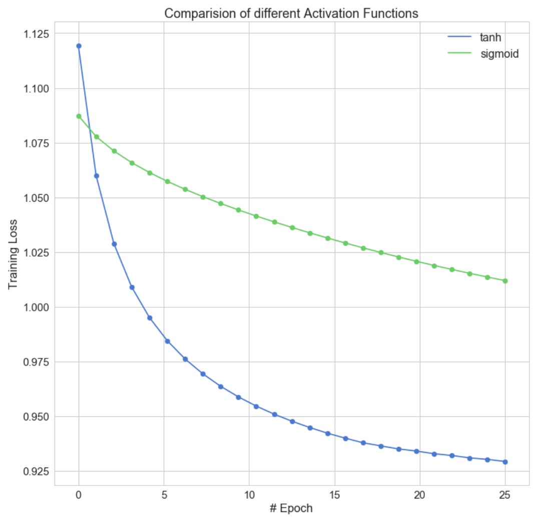
(a) Train loss for different algos



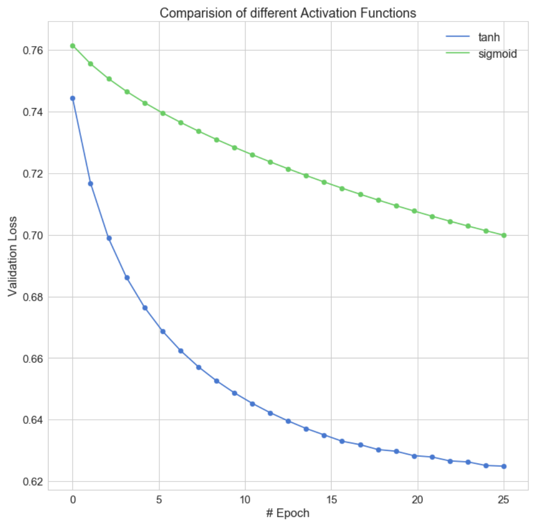
(b) Validation loss for different algos

As expected, Vanilla gradient descent converges very slowly as compared to other optimization algorithms. ADAM has a better decay rate than any other algorithm but for the given network of 3 hidden layers with 300 neurons each, the decay rate of momentum gradient descent beats all. RMSprop on the other hand performs poorly

## 7.3 Experiment 3



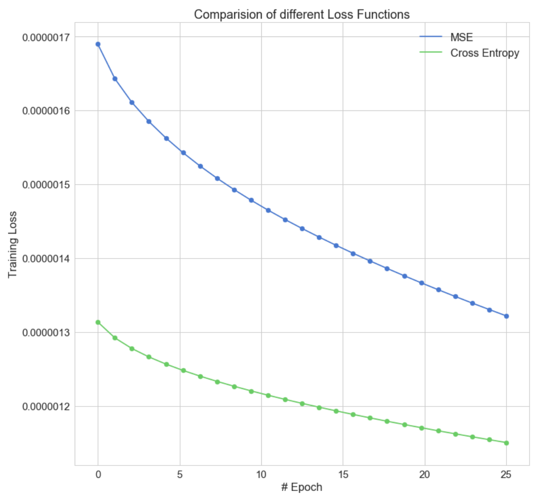
(a) Train loss for different activation fns



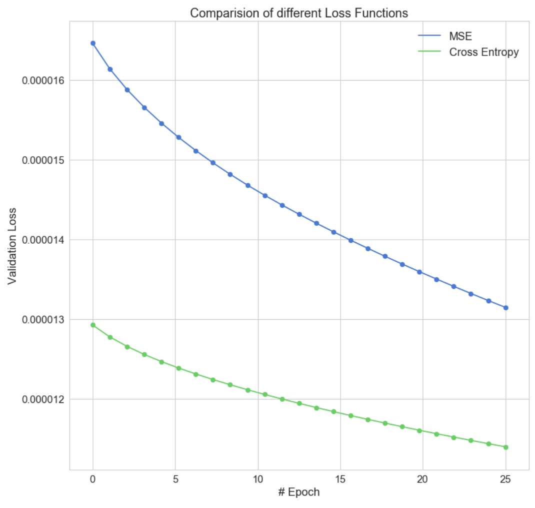
(b) Validation loss for different activation fns

We can observe here that for a network with two hidden layers, the decay rate of ADAM with tanh activation function is much much higher than that of sigmoid activation function. The sigmoid activation function has the potential problem that it saturates at zero while tanh saturates at plus and minus one. So if the activity in the network during training is close to zero then the gradient for the sigmoid activation function may go to zero, also known as the vanishing gradient problem.

## 7.4 Experiment 4



(a) Train loss for different loss fns



(b) Validation loss for different loss fns

For the given classification task, as expected the cross entropy loss function gives much better result as it's outputs give a probability distribution over the k-classes whereas mean-squared loss for the classification problem doesn't make much sense.

## 8 Difficulties faced

We faced certain difficulties while training the feedforward neural network.

- Divide by zero errors : We solved this issue by adding a very small constant to the denominator.
- Overflow issues while computing sigmoid and entropy loss : We solved this issue by using relevant functions from the scipy and sklearn library.

## 9 Kaggle Accuracy

The best accuracy we have got till now is 83.69% using momentum gradient descent with $\gamma = 0.9$ and $\eta = 0.1$. We have also use annealing and trained it on batch size of 20 for 200 epochs.

## 10 References

[Slides] Mitesh Sir's slides, *course slides*

[softmax] https://deepnotes.io/softmax-crossentropy *Stable softmax*

[DL] https://www.coursera.org/specializations/deep-learning, *Deep Learning*