Report: Assignment 2.1

Name: Shubham Sarda Entry Number:2018TT10958

Part C

Tuning is done on batch size, learning rate, activation function and optimization algorithm used. Tuning is done separately for each kind of optimization algorithm and both architectures. MSE loss function had been tested and it gave very poor results therefore not considered during further tuning (8-10 % accuracy compared to 90+% accuracy by cross entropy loss).

For every algorithm, we first select best batch size from a widely spread parameter search space. Then depending on the best batch size, we search for the best batch size in a more confined search which is close to the initial found value of best batch size. We plot the training loss over number of epochs for different batch sizes to select the best batch size. While finding best batch size, generally learning rate (lr) is set at a generic value of 0.01 so that it neither too large or too small. This lr value was tested initially for all algorithms. It converged well for all therefore its set as the default value. Also, we set the activation function as relu. This is because initial sample tests showed that relu works well with almost all optimization algorithms. Also, relu has been the conventional choice for image classification problems. Thus, instead of trying all permutations and combinations of hyperparameters, we smartly find the optimal parameters one by one. This helps save ton of time without any compromise on loss value.

We then find the best activation function out of relu, sigmoid or tanh. During this search, we set optimal batch size found previously and lr as 0.01. We plot train loss vs epochs for different activation functions.

Then we find the best lr in a broadly spread search space. This is followed by another testing in a confined search space close to the initial found value. This helps us find the most optimal value of lr. During lr search, we set optimal batch size and activation function found previously. Again, we plot the train loss vs epochs for different lr to select the best lr.

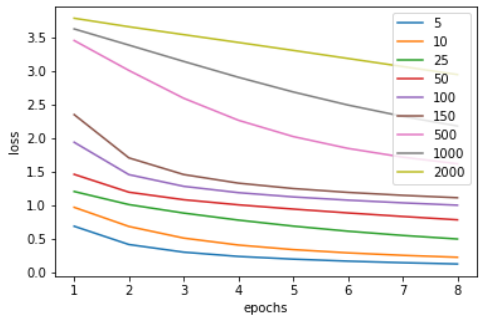
We are supposed to do this tuning for 2 architectures (given in the problem statement):

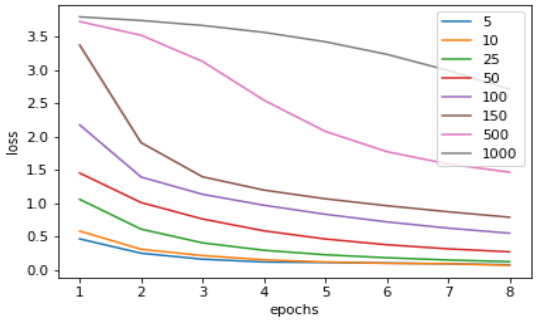
Architecture 1 : [Input,256,46] **Architecture 2** : [Input, 512,256,128,64,46]

Note: Last layer is always fixed at 46 as we classify into one of 46 Devanagri Characters

Mini Batch Gradient Descent:

Finding Batch Size:





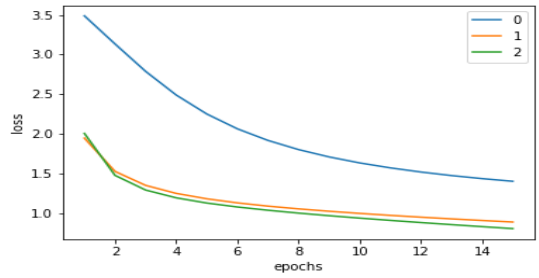
Architecture 1 Architecture 2

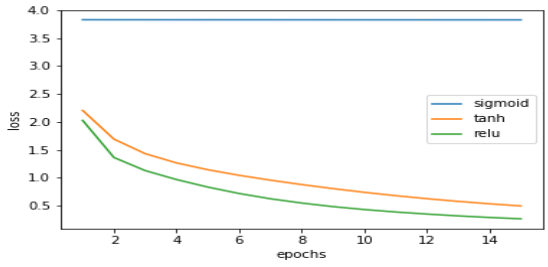
Here, it is clear that lower batch size gives lower loss. We also noted the time taken to complete 8 epochs. Time taken by the batch size from 5 to 2000 are 337, 175, 85, 58, 38, 34 ,27, 26 and 24 seconds respectively. We select batch size as 50 since it provides the least loss in small time. Since between 5 to 100, we're already considering many values, therefore, in this case another confined search isn't needed.

Similarly for architecture 2

Finding Activation Function:

0: Sigmoid, 1: tanh and 2: relu

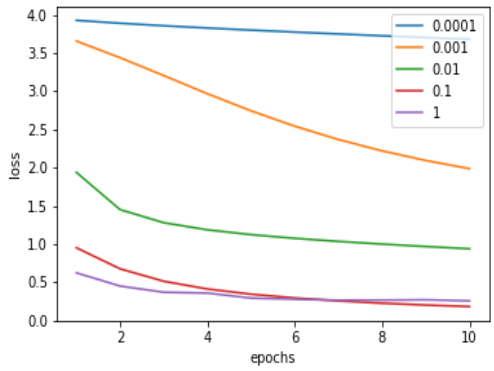


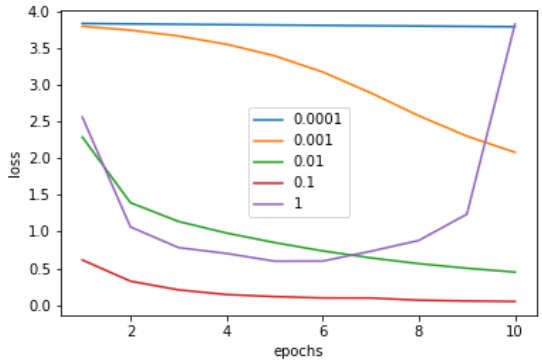


Architecture 1 Architecture 2

Relu is the clear choice of activation function for both architectures.:

Finding LR:

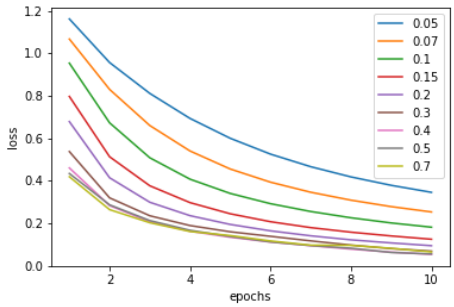


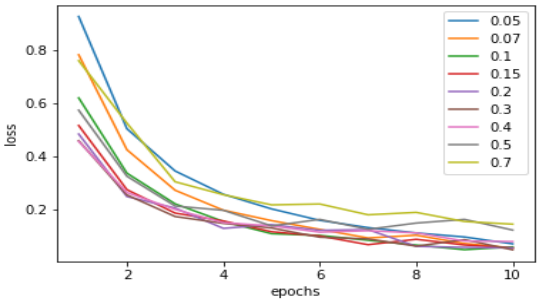


Architecture 1 Architecture 2

For architecture 1, optimal lr is close to .1 and 1. For 2nd architecture as well its between -.01 and 1. Thus, we do another confined space search to find optimal lr.

Architecture 1 Architecture 2





For 1st architecture .4 was chosen as lr, whereas for 2nd, .3 is the optimal lr. Even though the graph is a bit messy, we had also printed value of losses to figure out the best lr.

Similar analysis was done with remaining optimization algorithms. Optimal values and graphs are given below.

Adaptive mode of lr was also tried with values near the optimal lr, but it didn't give better results.

Architecture 1:

Hyperparameter	**Momentum**	**Nesterov Accelerated Gradient Decent**	**RMSPROP**	**Adam**	**Nadam**
Activation Function	Relu	Relu	Sigmoid	Relu	Sigmoid

Batch Size	100	150	750	1000	150
Learning Rate	.05	2e-2	8e-3	8e-3	8e-3

Architecture 2:

Hyperparameter	**Momentum**	**Nesterov Accelerated Gradient Decent**	**RMSPROP**	**Adam**	**Nadam**
Activation Function	Relu	Relu	Sigmoid	Relu	Sigmoid
Batch Size	150	250	450	750	150
Learning Rate	.01	4e-2	8e-3	8e-3	8e-3

Plots for different algorithms can be found at the end of the report.

Now, with the best parameters found for each optimization algorithm, neural network is trained for 300 seconds and the one which gives least train error is selected.

For architecture 1, conventional gradient descent gave the best loss: **0.000359**

Activation Function =Relu, Lr=.4 and batch size=50

For architecture 2, gradient descent with momentum gave the best loss: **0.000398**

Activation Function =Relu, Lr=.01 and batch size=150

Part D

For part C, it took nearly 24hrs of training to find optimal parameters of both architectures. Further, a lot of manual tuning was required for 2nd search in confined space etc. Thus, it's not very feasible to do the same for multiple architectures in part D. In part C, we noticed that optimal parameters of both architectures in part C were very similar/close to each other. Thus, we simply experiment by small modifications in the two public architectures (already proven architectures) and check for hyperparameters in the very close range of the optimal ones found for them in previous part. Due to the time constraints, we can't use too deep neural networks.

Architectures 1 and 2 are tweaked along with their optimal hyperparameters to experiment with new structures. A holdout set (7820 samples) of 10% train data is used as test set. Later, for some tests, holdout set was also reduced to 4600 and 3000 samples to check if validation accuracy increased or not. Public test set provided is used as the validation set. Architecture with max accuracy on validation and test is selected. Training is done for 5 minutes. Most experiments are done with momentum as it gives best results out of remaining algorithms.

Activation function used is same as the optimal one found for various algorithms in part C.

Results for each architecture are printed to find the best one.

Experimenting With Architecture 1:

Architecture (hidden layers +output layer)	Optimization Algorithm	Learning Rate	Batch Size	Samples in test set	Test Set Accuracy	Validation Set Accuracy
[256,46]	Gradient descent	0.4	50	7820	0.9427	0.942
[256,46]	Momentum	.05	100	7820	0.9441	0.941
[128,46]	Momentum	.05	100	7820	0.914	0.917
[256,46]	RMSPROP	8e-3	750	7820	0.9289	0.9282
[256,46]	Adam	8e-3	1000	7820	0.9193	0.9254
[256,46]	Nesterov	2e-2	150	7820	0.9386	0.9380
[256,46]	Nadam	8e-3	150	7820	0.8860	0.8869
[512,46]	Momentum	.05	100	7820	0.9491	.9482
[512,46]	Momentum	.06	100	7820	0.9502	0.9508
[512,46]	Momentum	.08	150	7820	.95179	0.9491
[1024,46]	Momentum	.05	100	7820	0.9519	0.9506
[1024,46]	Momentum	.09	100	7820	0.9528	0.9552
[1024,46]	Momentum	.09	100	3000	0.957	0.9571
[1024,46]	Momentum	.09	100	4600	0.9558	0.9569
[4112,46]	Momentum	.09	100	7820	0.9441	.9486
[2056,46]	Momentum	.05	100	3000	.9536	.9580

[2056,46]	Momentum	.05	100	4600	0.9541	.9552
[2056,46]	Momentum	.05	100	7820	.9517	.9523

Experimenting with architecture 2

Architecture (hidden layers +output layer)	Optimization Algorithm	Learning Rate	Batch Size	Samples in test set	Test Set Accuracy	Validation Set Accuracy
[512,256,128,64,46]	Nadam	8e-3	150	7820	0.918	.9215
[512,256,128,64,46]	Rmsprop	8e-3	450	7820	.9327	.9284
[512,256,128,64,46]	Adam	8e-3	750	7820	.9313	.9315
[512,256,128,64,46]	Gradient Descent	.3	50	7820	.9387	.928
[512,256,128,64,46]	Nesterov	2e-2	150	7820	0.9391	.9417
[512,256,128,64,46]	Momentum	.01	150	7820	.9484	.9478
[1024,256,46]	Momentum	.01	150	7820	.9434	.945
[512,256,512,256,46]	Momentum	.01	150	7820	.9517	.9510
[256,128,256,128,64,46]	Momentum	.01	150	7820	.9452	.9458
[256,128,256,128,46]	Momentum	.01	150	7820	.9496	.9454
[512,256,128,128,46]	Momentum	.01	150	7820	.9492	.9469
[512,256,128,128,46]	Momentum	.02	150	7820	.9528	.9569
[512,256,128,128,46]	Momentum	.02	150	3000	.9533	.9547
[512,256,128,128,46]	Momentum	.03	150	7820	.9476	.9489
[512,256,256,46]	Momentum	.02	150	7820	.9551	.9517
[512,256,256,46]	Momentum	.03	200	7820	.9535	.9545
[512,256,256,46]	Momentum	.03	200	4600	**.9591**	**.9584**
[512,256,256,46]	Momentum	.03	200	3000	.9546	.9565
[512,512,46]	Momentum	.02	150	7820	.9501	.9504
[512,512,46]	Momentum	.02	150	4600	.9532	.9521
[1024,1024,46]	Momentum	.03	150	7820	.9526	.9523
[512,256,256,46]	Momentum	.01	150	7820	.9480	.9482
[512,256,256,46]	Momentum	.02	150	7820	.9537	.9534
[512,256,46]	Momentum	.03	150	7820	.9542	.9543
[512,256,46]	Momentum	.03	150	3000	.9526	.953
[512,256,46]	Momentum	.04	150	3000	.9566	.9534

Hence, we select the architecture: **[512,256,256,46]**

Optimization: Momentum, Lr=.03, Lr type = Fixed. batch size=200, Activation =relu

Number of epochs=150 Time limit:5 min

PLOTS FOR PART C

Note: 1) All LHS plots are for architecture 1 and RHS plots for architecture 2

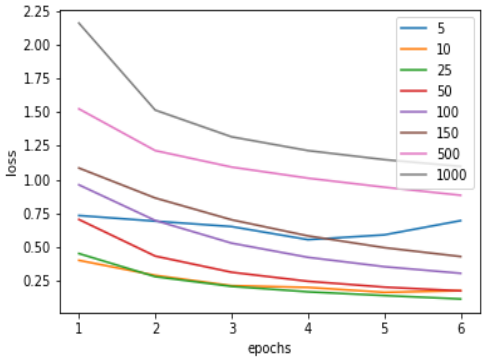
2) When plots seem very close to each other, actual value of loss is seen (printed during hyperparameter tuning)

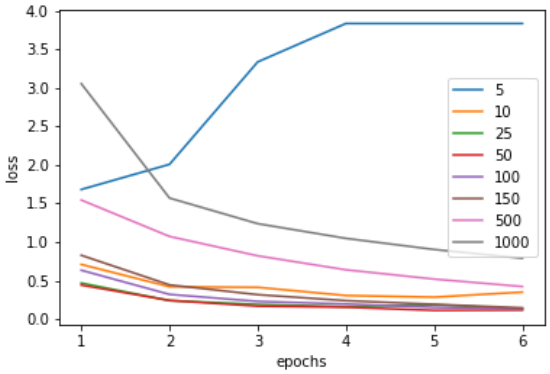
3) For a few activation functions plots: 0=sigmoid, 1=tanh and 2=relu

4) Kindly zoom in to see the plots clearly

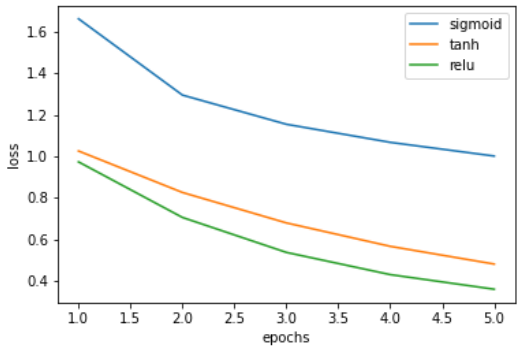
Momentum

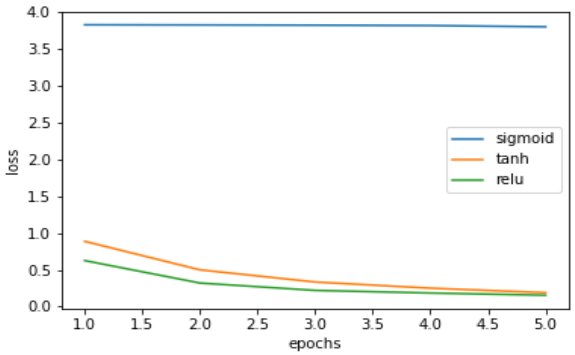
Batch Size:



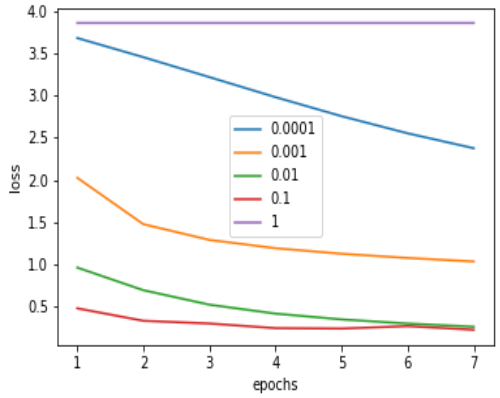


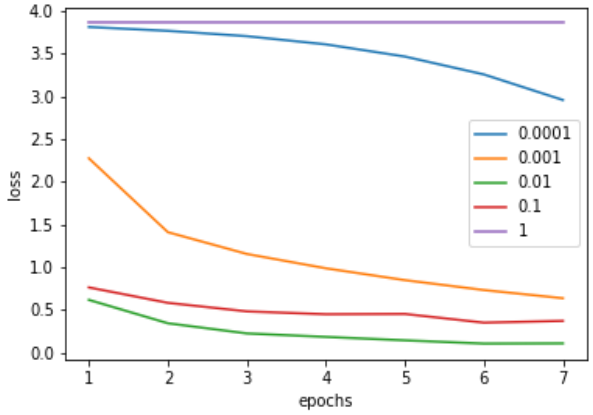
Activation Function:

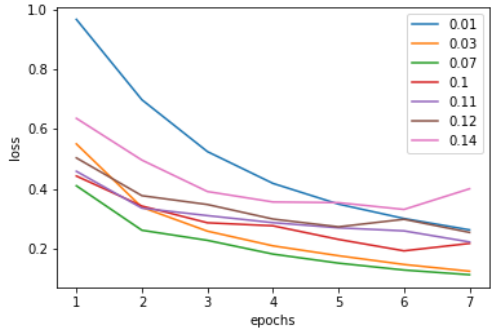


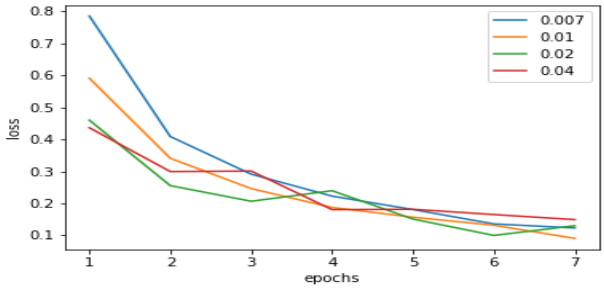


LR: Broad search followed by confined search





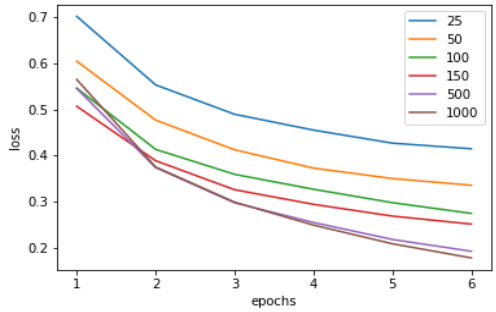


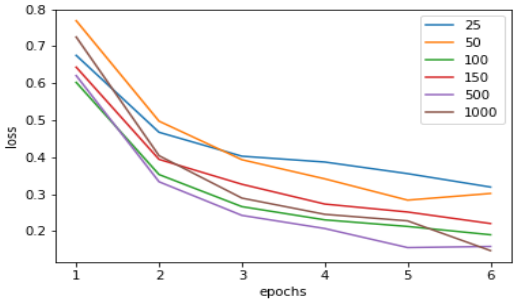


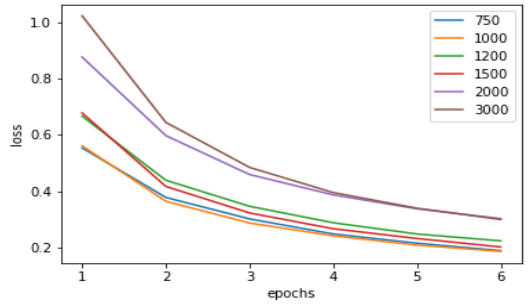
For Architecture 1: avg of .07 and .03 was taken as optimal lr since for further epochs, .03 line could go below .07 plot. Hence avg taken to get the best of both.

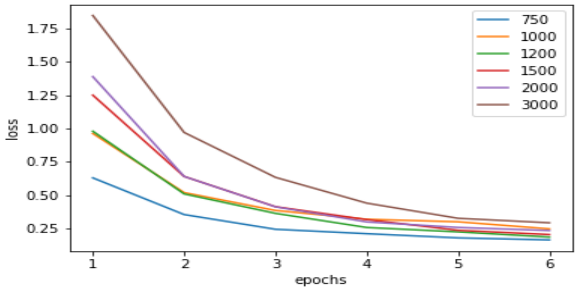
Adam

Batch Size: Broad search followed by confine search

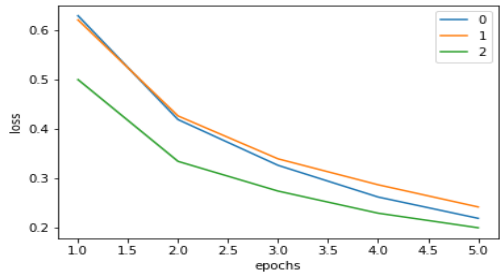


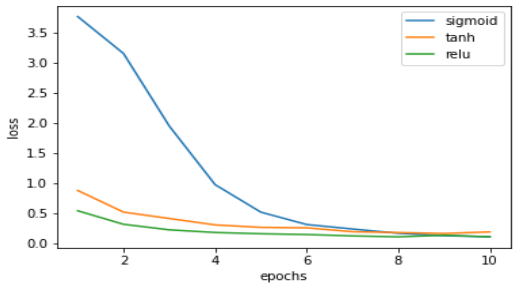




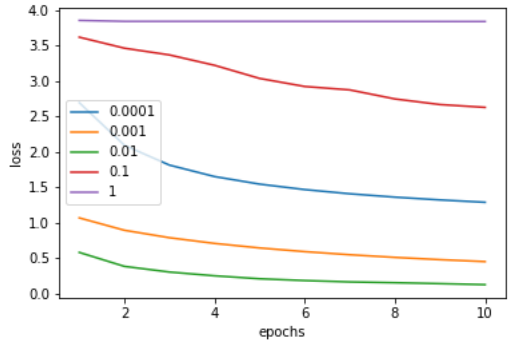


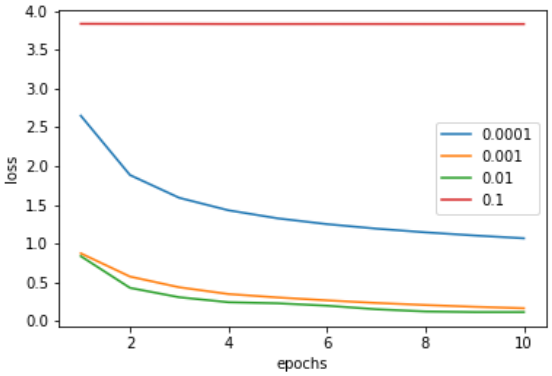
Activation Function:

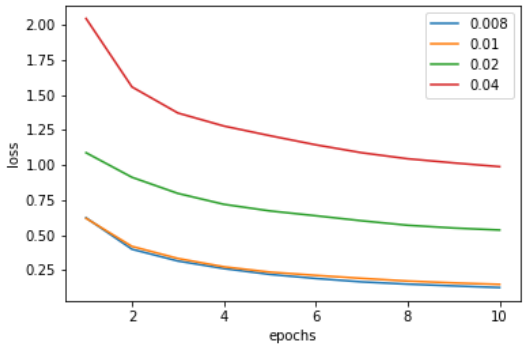


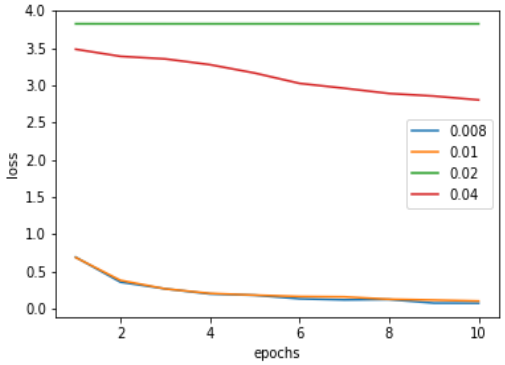


LR:



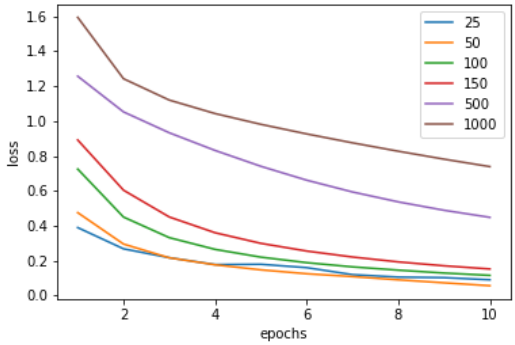


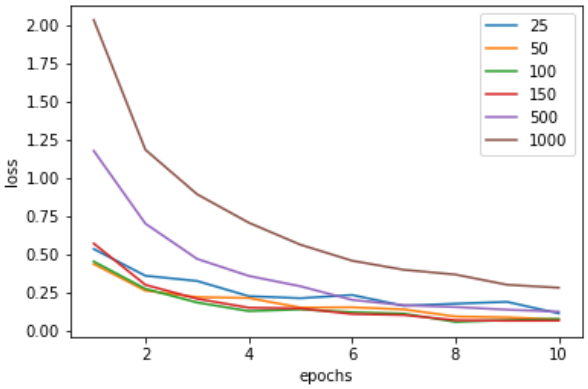


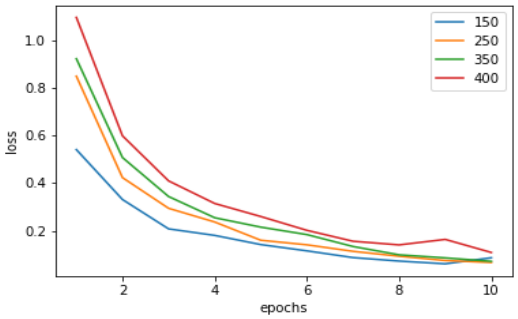


Nesterov Accelerated Gradient Descent

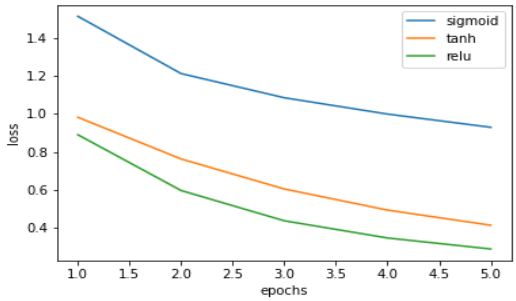
Batch Size: Confined space search only for architecture 2 as for architecture 1, 150 was chosen as it took less time compared to 25, 50 and 100 but gave almost same loss after 10 epochs.

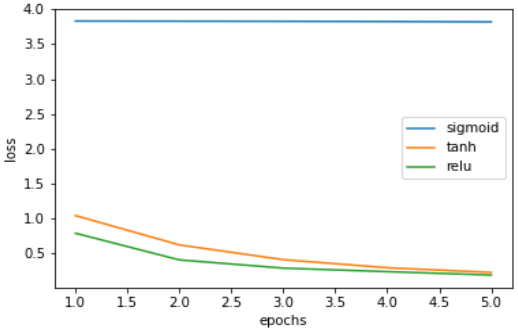




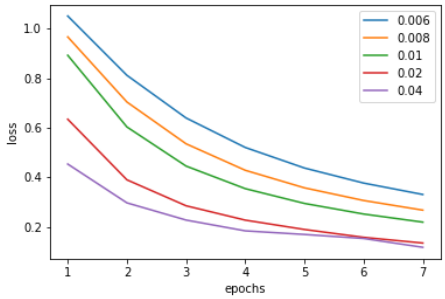


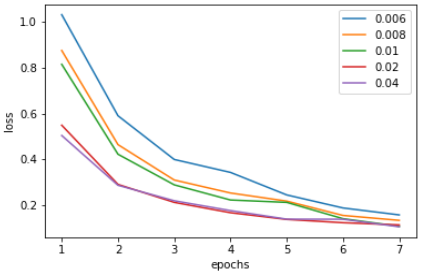
Activation Function:





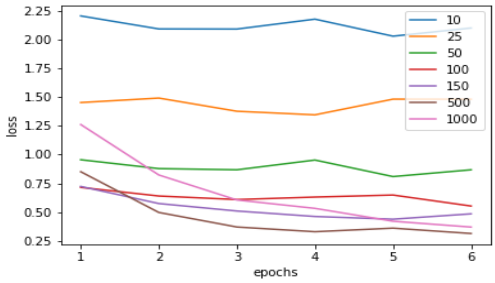
LR:

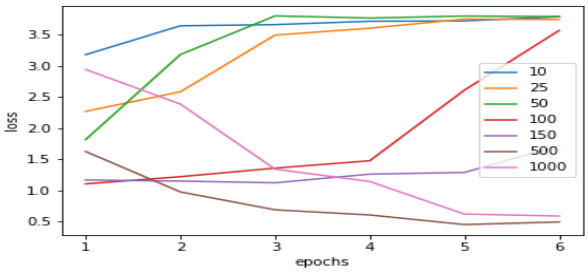


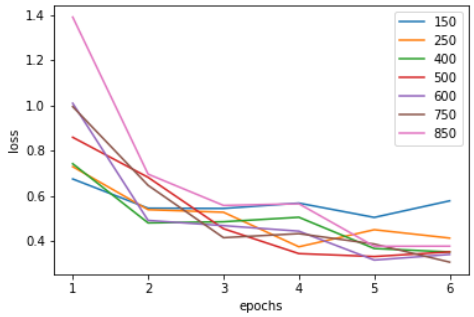


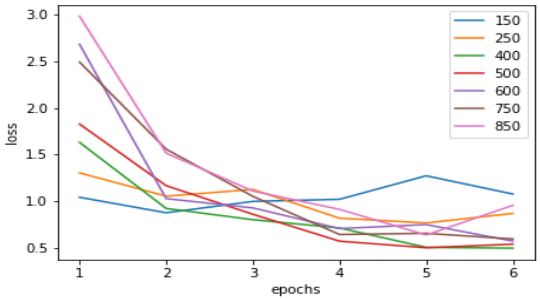
RMSPROP

Batch Size



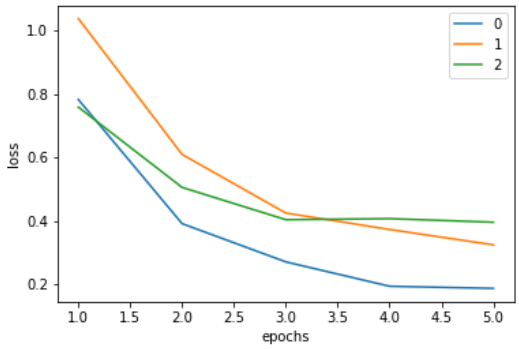


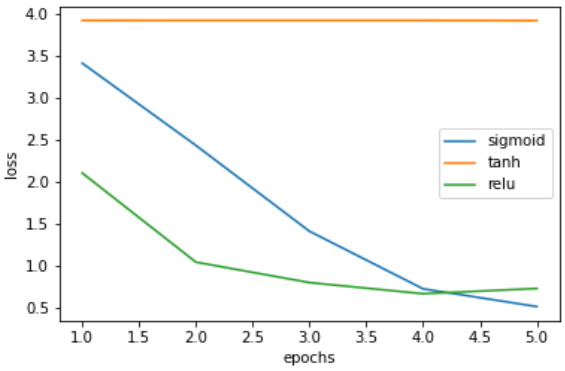




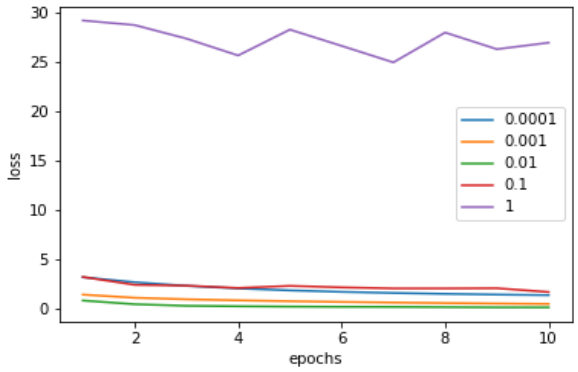
For architecture 2: avg of 400 and 500 is taken

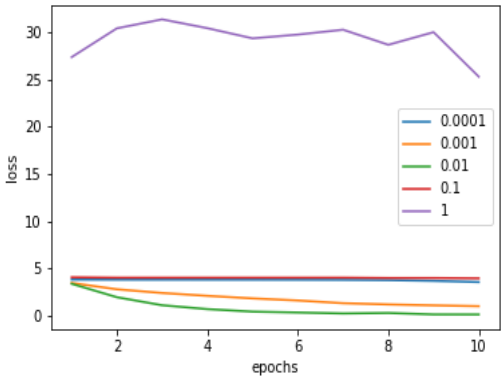
Activation Function

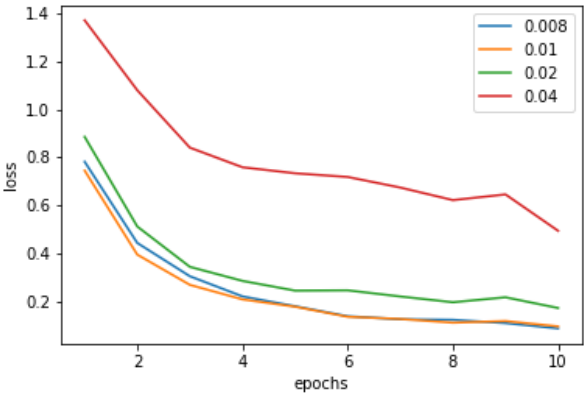


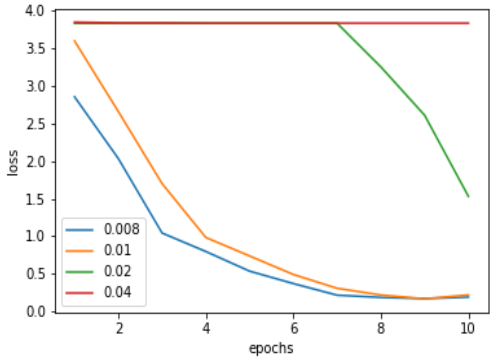


LR:



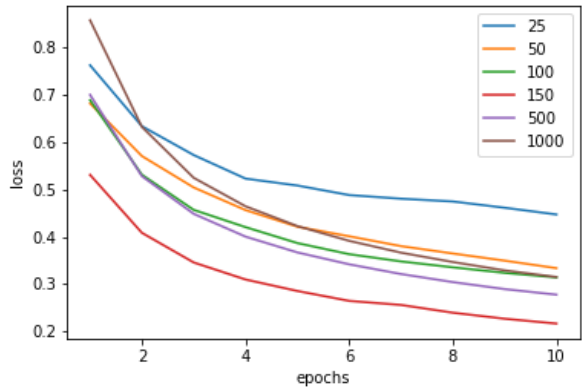


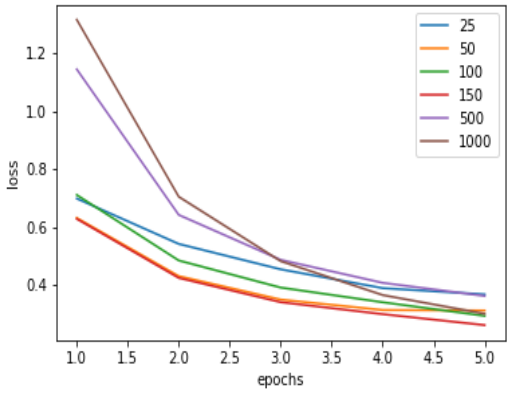


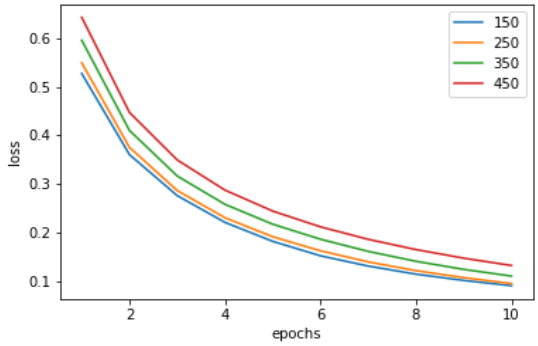


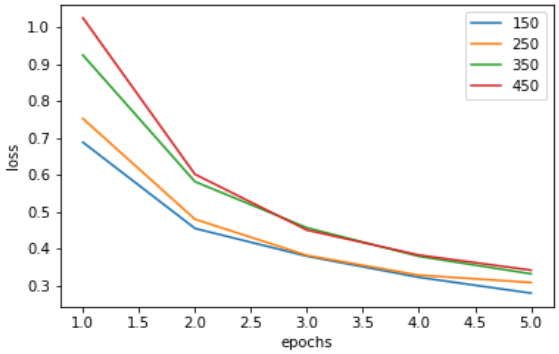
Nadam

Batch Size:

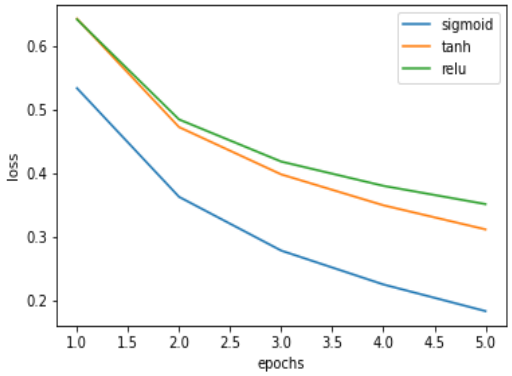


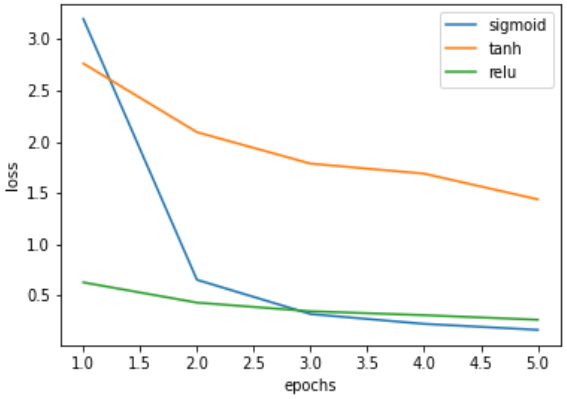




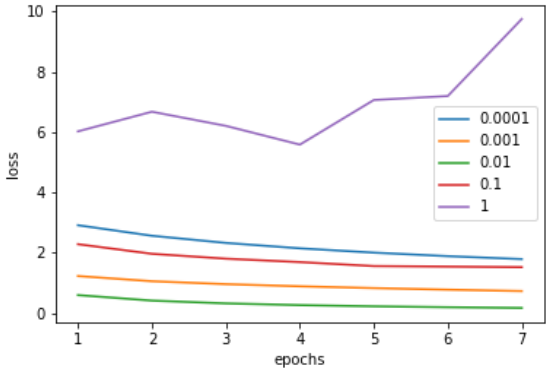


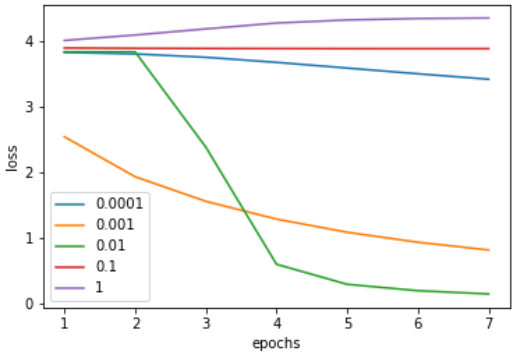
Activation Function:

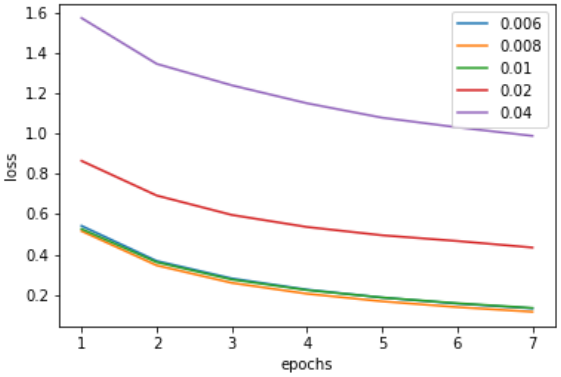


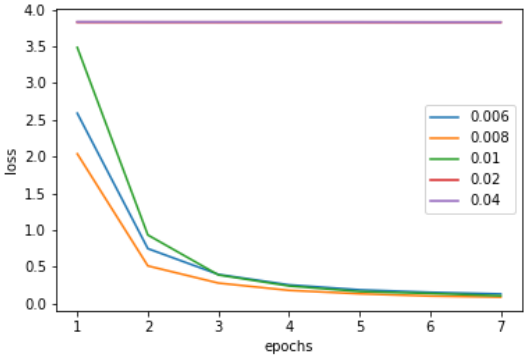


LR:


loss
epochs
0.0001
0.001
0.01
0.1
1


loss
epochs
0.0001
0.001
0.01
0.1
1


loss
epochs
0.006
0.008
0.01
0.02
0.04


loss
epochs
0.006
0.008
0.01
0.02
0.04