Contents

[ABSTRACT 2](#_Toc512567825)

[INTRODUCTION 4](#_Toc512567826)

[Structure and List of Hyper Parameters used for the Neural Networks 4](#_Toc512567827)

[Multi Layered Perceptron (MLP) 4](#_Toc512567828)

[Hyper Parameter List 4](#_Toc512567829)

[Convolutional Neural Network 5](#_Toc512567830)

[Hyper Parameter List 5](#_Toc512567831)

[Recurrent Neural Network 5](#_Toc512567832)

[Restricted Boltzmann Machine 6](#_Toc512567833)

[Generative Adversarial Network 6](#_Toc512567834)

[Autoencoders 7](#_Toc512567835)

[Methods 8](#_Toc512567836)

[Evaluation Criteria 8](#_Toc512567837)

[Process of Hyper Parameter Tuning 8](#_Toc512567838)

[Results 9](#_Toc512567839)

[Hyper Parameter Tuning Results for MLP Neural Network Results 9](#_Toc512567840)

[Hyper Parameter Tuning Results for Convolutional Neural Network Results 12](#_Toc512567841)

[Hyper Parameter Tuning Results for Recurrent Neural Networks Neural Network Results 16](#_Toc512567842)

[Hyper Parameter Tuning Results for Restricted Boltzmann Machine Neural Network Results 18](#_Toc512567843)

[Hyper Parameter Tuning Results for Generative Adversarial Neural Network Results 21](#_Toc512567844)

[Hyper Parameter Tuning Results for Autoencoder 24](#_Toc512567845)

[25](#_Toc512567846)

[Discussion 26](#_Toc512567847)

[Multi-Layer Perceptron Neural Network 27](#_Toc512567848)

[Convolutional Neural Network 27](#_Toc512567849)

[Recurrent Neural Network 28](#_Toc512567850)

[Restricted Boltzmann Machine 28](#_Toc512567851)

[Generative Adversarial Network 29](#_Toc512567852)

[Autoencoder 29](#_Toc512567853)

[References 32](#_Toc512567854)

**Hyper Parameter Tuning for Deep Neural Networks**

Jaini Bhansali, Student under the guidance of Asst. Professor Nik Bear Brown

The text in the document by Bhansali,Jaini is licensed under CC BY 3.0 <https://creativecommons.org/licenses/by/3.0/>

# ABSTRACT

The popularity of deep neural networks has risen in the recent past. Though deep neural networks have an increased popularity there is a demand to improve their performance with the help of Hyper Parameter Tuning. The Hyper Parameter Tuning research study involves formulating a list of hyper parameters or a combination of hyperparameters that help improve the performance of the Neural Networks and describes various prospective hyper parameters to tune for these neural networks. The Neural Networks selected are namely Multi Layer Perceptron (MLP), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Restricted Boltzmann Machine (RBM), Generative Adversarial Network (GAN) and Autoencoders. Accuracy and Loss are taken as measure of performance. Accuracy is used as a measure for MLP, CNN , RNN and Loss or Reconstruction Cost is used as a measure for generative models - RBM, GAN and Autoencoders respectively. The table below indicates the highest accuracy achieved for each neural network with details of the dataset.

**Highest Accuracy for Neural Network Types**

|  |  |  |  |
| --- | --- | --- | --- |
| **Neural Network Type** | **Dataset Name** | **Train Accuracy** | **Test Accuracy** |
| Multi Layer Perceptron | Iris Dataset | 100% | 97.72% |
| Convolutional Neural Network | CIFAR10 Dataset | 56.25% | 49.35% |
| Recurrent Neural Network | HASYV2 Dataset | 100% | 71.81% |

The below table provides a list of Generative models with details of the dataset and lowest loss or cost achieved during Hyper Parameter Tuning

**Generative Models**

|  |  |  |
| --- | --- | --- |
| **Neural Network Type** | **Dataset Name** | **Loss/Cost** |
| Restricted Boltzmann Machine | MNIST | Reconstruction Cost  -523.17 |
| General Adversarial Networks | MNIST | Generator Loss 0.2892,  Discriminator Loss 2.531 |
| Autoencoder | MNIST | Train Loss 0.0999  Validation Loss 0.1006 |

# INTRODUCTION

Hyper Parameters govern the algorithm irrespective of the data. Each algorithm has various hyper parameters that can be tuned that can improve the performance of the algorithm. There are various ways to perform hyper parameter tuning. One of the popular methods used is to tune one parameter and observe the effect on the performance of the algorithm. The second method used is to tune a combination of 2 parameters to observe its impact on the performance of the network. The third method involves selecting a list of parameters as a combination and observing its impact on the Neural Network. Method 1 and Method 2 have been used in the current research study. The various Neural Networks selected for hyper parameter tuning are as follows:

1. Multi Layered Perceptron (MLP)
2. Convolutional Neural Network (CNN)
3. Restricted Boltzmann Machine (RBM)
4. Generative Adversarial Neural Network (GAN)
5. Recurrent Neural Network (RNN)
6. Autoencoders

Below defined is the structure of the Neural Network used for hyper parameter tuning. The Hyper Parameter Tuning is done using Tensorflow, a library created by Google.

## Structure and List of Hyper Parameters used for the Neural Networks

### Multi Layered Perceptron (MLP)

The Multi Layered Perceptron (MLP) model used consists of 10 hidden layers. Gaussian Initialization was used to initialize the weights and bias of the network. It was first initialized with a learning rate of 0.01, activation function ReLU and using the Softmax Cross Entropy with Logits as the cost function. It traverses the gradient with gradient estimation Adam Optimizer.

### Hyper Parameter List

|  |  |
| --- | --- |
| Hyperparameter Name | Hyperparameter Values |
| Gradient Estimation | Adagrad, Adadelta, Stochastic Gradient Descent |
| Activation Functions | Tanh, ReLU6, Sigmoid, ReLU |
| Number of Epochs and Learning Rate | * 1. and 1000,   2. and 100 |

### Convolutional Neural Network

This Neural Network consists 2 Convolutional Layers. Each Convolutional layer has an activation function Sigmoid and an Average Pooling layer. The 2 Convolutional layers are preceded by the flattened layer and the fully connected layer. The model is a LENET5 model. It uses the Softmax Cross Entropy cost function and Optimizer as Stochastic Gradient Descent. The model is initialized for 20000 epochs.

### Hyper Parameter List

|  |  |
| --- | --- |
| Hyperparameter Name | Hyperparameter Values |
| Number of Epochs | 10000,15000,20000 (Might differ based on time to process the dataset) |
| Loss and Activation | Softmax Cross Entropy and Stochastic Gradient Descent,  Hinge Loss and Tanh,  Hing Loss and Sigmoid |
| Gradient Estimation | Stochastic Gradient Descent, Adam, Adadelta |
| Network Initialization | Xavier Glorot Initialization, Gaussian Initialization |
| Network Architecture | Average Pooling, Max Pooling |

### Recurrent Neural Network

The RNN-LSTM structure utilizes weights and bias with Gaussian initialization. The simplest form of an RNN known as the Static RNN Cell and Basic LSTM Cell is used. The RNN LSTM model uses a batch size of 128, learning rate selected is 0.001 and number of epochs is 800. The optimizer used to reduce the softmax cross entropy loss is the Adam Optimizer that is selected.

|  |  |
| --- | --- |
| Hyperparameter Name | Hyperparameter Values |
| Number of Epochs | 500,1000,2000 |
| Batch Size | 4,128,512 |
| Number of Neurons | 1,128,512 |
| Learning Rate and Number of Neurons | 0.1 and 512, 0.001 and 512, 0.001and 512 |
| Activation | Tanh, ReLU, Softsign |
| Optimizers | RMSProp, Adagrad,Adam |

### Restricted Boltzmann Machine

The number of the visible units of the RBM equals to the number of the dimensions of the training data. Number of visible units selected 784 which is the total dimension of the input data. The number of hidden units are 200. The batch size selected is 128 and number of epochs is 100 for the first trial. This uses the Contrastive Divergence with K Gibbs Samples.

List of Hyper Parameters

|  |  |
| --- | --- |
| Hyperparameter Name | Hyperparameter Values |
| Activation | ReLU, Sigmoid |
| Learning Rate | 0.01,0.00001 |
| Initialization | Xavier Initialization |
| Number of Hidden Layers | 100,200,500 |
| Regularization and Activation | L1 and Sigmoid |

### Generative Adversarial Network

This network uses the Xavier Initialization for weights and biases. The Generator and Discriminator use the ReLU activation function, with one hidden layer. It uses the solver Adam. The probability distribution is identified using the activation function sigmoid. The network is initialized with the 10000 epochs.

List of Hyper Parameters

|  |  |
| --- | --- |
| Hyperparameter Name | Hyperparameter Values |
| Activation (Generator and Discriminator) | ReLU, Leaky ReLU |
| Optimizer and Learning Rate | Adam and 0.001, SGD and 0.001, SGD and 0.1 |

### Autoencoders

This is a denoising Autoencoder. The autoencoder uses an activation of leaky ReLU. The encoder and decoder consist of a convolutional layer and uses the Sigmoid Activation with the Sigmoid Cross entropy cost function using the Adam Optimizer. It is initialized with a learning rate of 0.00001 noise factor of 0.5.

|  |  |
| --- | --- |
| Hyperparameter Name | Hyperparameter Values |
| Activation of Output Layer | Sigmoid, ReLU |
| Loss | Hinge Loss, Sigmoid Cross Entropy Loss |
| Noise | 0.5,0.3 |
| Learning Rate | 0.00001,0.01,0.1 |
| Optimizer | Adam, RMSProp, Adadelta |

It was observed that the Number of Epochs and Learning Rate was a significant parameter to tune in the MLP Neural Network. Number of Epochs, Network Initialization, Loss functions played a pivotal role in tuning the CNN Model. Number of Epochs, Batch Size and a combination of Learning Rate and Number of Neurons are significant hyper parameters to tune in the RNN LSTM model. The RBM Model was impacted most by the Network Initialization and Number of Epochs. The combination of the Gradient Estimation and Learning rate was an important parameter for the GAN and the Autoencoder.

The methods of implementation are described below

# Methods

### Evaluation Criteria

The accuracy or loss of the models that is generated when the model is first run is taken as a benchmark. If tuning the hyper parameter improves the existing bench mark or close to the bench mark then it is considered as a prospective parameter to tune for the respective model. If the model does not surpass the bench mark, then consider the parameter that is set initially.

Each algorithm is taken one at a time and the list of hyper parameters are applied to the algorithm. The MLP, CNN and RNN Neural Network performance is measured based on their accuracy, the performance of the RBM is measured based on the reconstruction cost, performance of the GAN and Autoencoder is measured on the basis of their respective losses. Each model is implemented in Python in a Jupyter Notebook using the Tensorflow Library. After initial running of the model as per the structure mentioned, each model is tuned for the selected hyper parameters.

### Process of Hyper Parameter Tuning

1. **MLP**: The MLP Structure is set up as defined above using Python. After the initial set up and ensuring the model is running, each hyper parameter as per the list was selected and tuned. The test accuracy was noted for each hyper parameter tuned. The hyper parameters tuned were Gradient Estimation, Activation and a combination of Number of Epochs and Learning Rate.
2. **CNN**: The Dataset consisted of images, hence the dataset was first pre-processed and images are one hot encoded. The training for this dataset is time consuming hence, it is trained for only 10,000 epochs. Once the structure of the model is defined the CNN model is tuned for the enlisted hyper parameters. The accuracy is noted for each hyper parameter tuned. The Accuracy is compared to the bench mark model run in the initial stage.
3. **RNN**: The dataset must be pre-processed and converted to one hot encoded values. The RNN was then tested for the various Hyper Parameters enlisted.
4. **RBM**: To use RBM in Tensorflow a library names xRBM was used for the purpose. This library is installed in Python. The initial model is set up following which the model is trained for the enlisted hyper parameters
5. **GAN**: Since the GAN is used to generate images, the images must be pre-processed. After running the GAN with the initial model it is trained for the various enlisted hyper parameters.
6. **Autoencoders**: After initializing the model, the autoencoder is then tested on various hyper parameters enlisted in the above section.

The accuracy or loss for each hyper parameter tuned is compared to the initial accuracy or loss generated by the network after fine tuning the initial model that is set up. The detailed method of implementation can be found at **[21]**

# Results

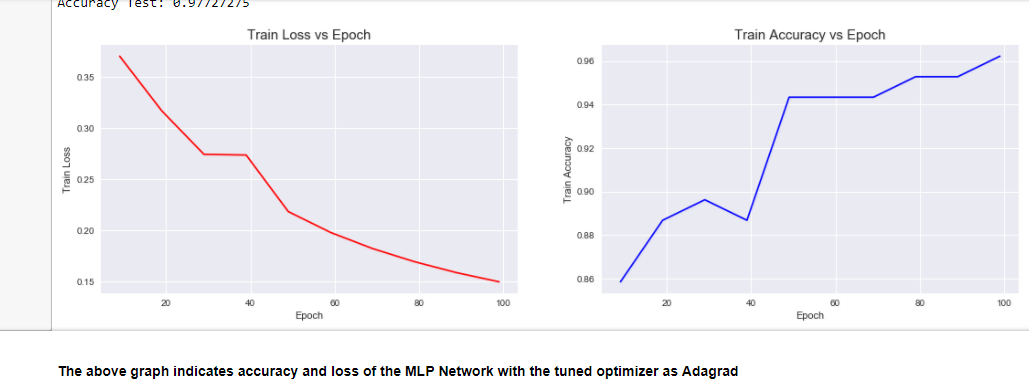
The objective of performing tuning for each Neural Network is to understand the hyper parameter values that can be used to improve the performance of the network and understand the prospective hyper parameters that can be tuned. The results of the best hyper parameters to tune for each of the neural networks is listed below.

## Hyper Parameter Tuning Results for MLP Neural Network Results

|  |  |  |  |
| --- | --- | --- | --- |
| Hyper Parameter Name | Hyper Parameter Value | Training Accuracy | Testing Accuracy |
| Gradient Estimation | Adam  Adagrad | 97.16%  96.22% | 97.72%  97.72% |
| Activation Function | Sigmoid  ReLu | 98.11%  97.16% | 100%  97.72% |
| Epoch and Learning Rate | 100 and 0.1 | 99.05% | 97.72% |

**Graphical Representation of Results for the Multi-Layer Perceptron Neural Network**

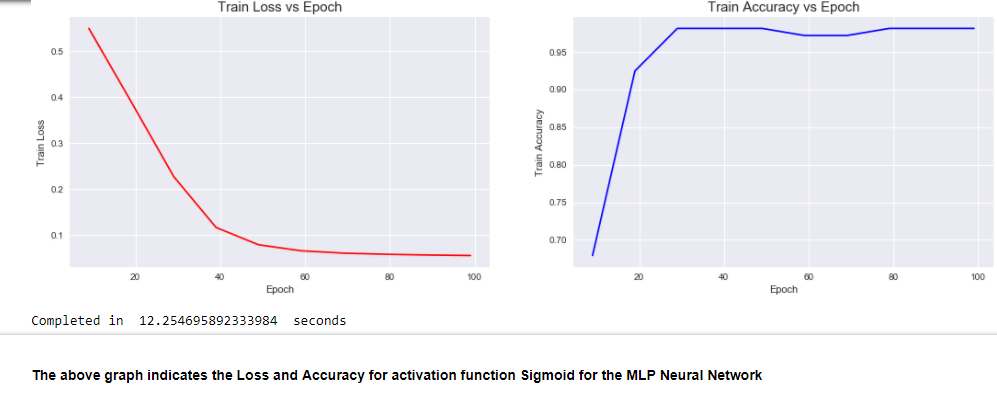
**Gradient Estimation**



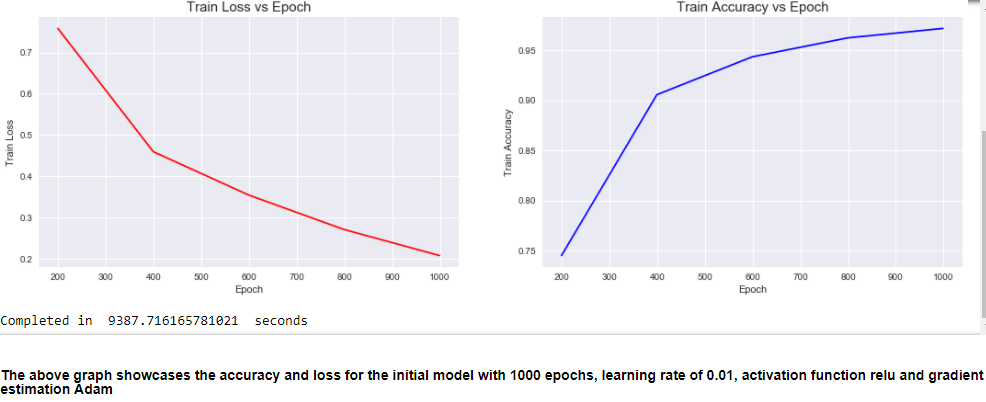
**Gradient Estimation - Adam**



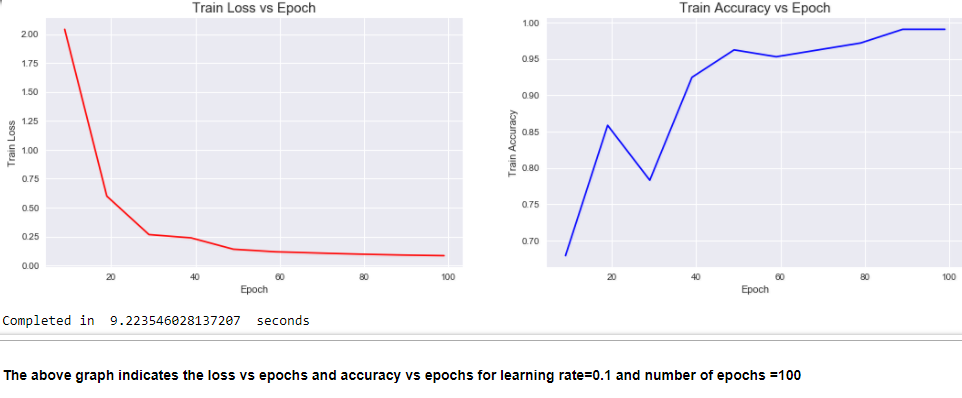
**Activation Function - Sigmoid**



**Activation Function - ReLU**



**Combination of Number of Epochs and Learning rate**



## Hyper Parameter Tuning Results for Convolutional Neural Network Results

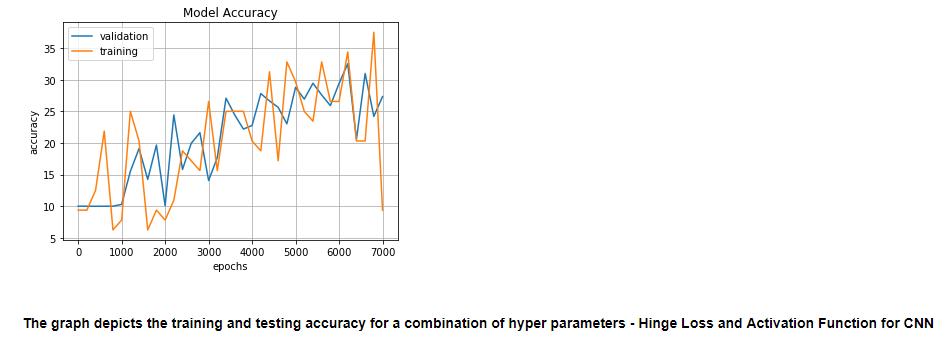
The CIFAR 10 Dataset has 10 classes of over 60000 images. Hence, training the algorithm takes extremely long. The Algorithm was first trained for 20 hours for 20000 epochs to reach a training and testing accuracy of 60.94% and 48.63% respectively. Hence, using a bench mark of 7000 epochs where the model reached an accuracy of 50% and 41.09% with stability was taken as the benchmark for hyper parameter tuning. Taking a consistent value of the 7000th epoch was 42.72% and 42.19%

|  |  |  |  |
| --- | --- | --- | --- |
| Hyper Parameter Name | Hyper Parameter Value | Training Accuracy | Testing Accuracy |
| Loss and Activation Function | Softmax Cross Entropy and SGD Optimizer  OR  Hinge Loss and Tanh | 42.19%  OR  37.50% | 42.72%  OR  24.20% |
| Activation Function | Sigmoid | 42.19% | 42.72% |
| Network Initialization | Xavier Initialization and Gaussian Initialization | 40.62% and 34.34% | 42.19% and 39.52% |
| Number of Epochs  (can be trained longer) | 20000 | 60.94% | 48.59% |
| Gradient Estimation | Stochastic Gradient Descent | 42.19% | 42.72% |

**Graphical Representation of Results for the Convolutional Neural Network Model**

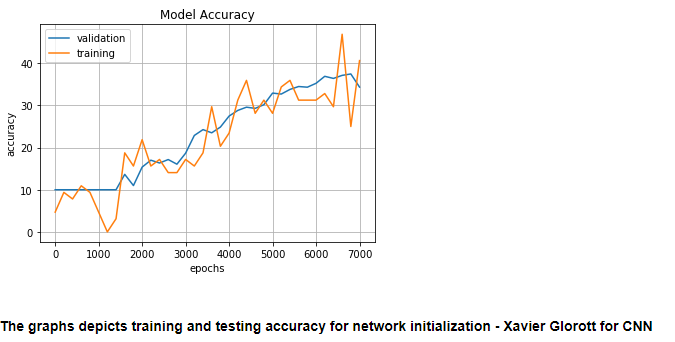
**Loss and Activation**

**Loss (Hinge Loss) and Activation (Tanh)**

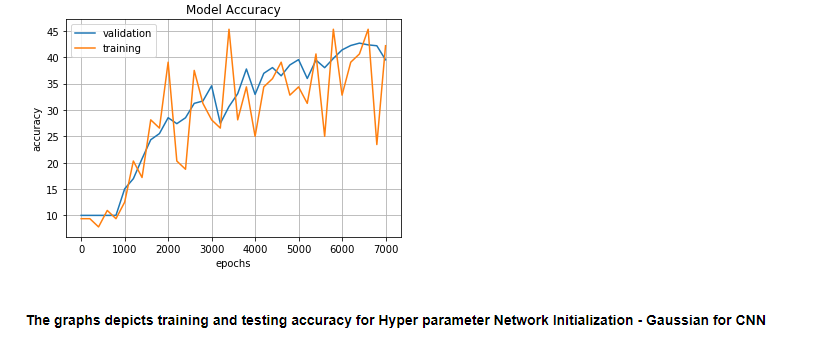


**Network Initialization**

**Network Initialization (Xavier Glorott)**

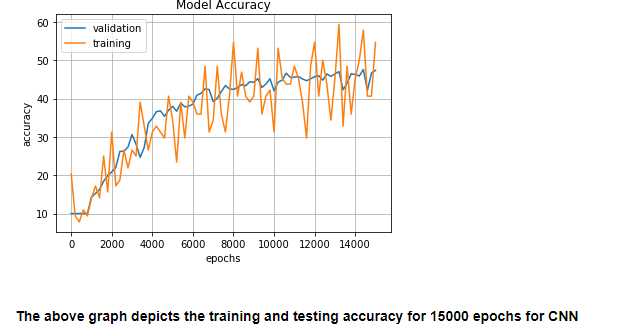


**Network Initialization (Gaussian)**



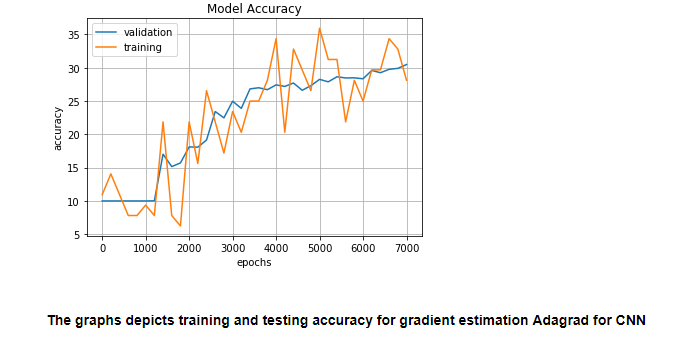
**Number of Epochs**

**Number of Epochs**



**Optimizer**

Optimizer (Adagrad)

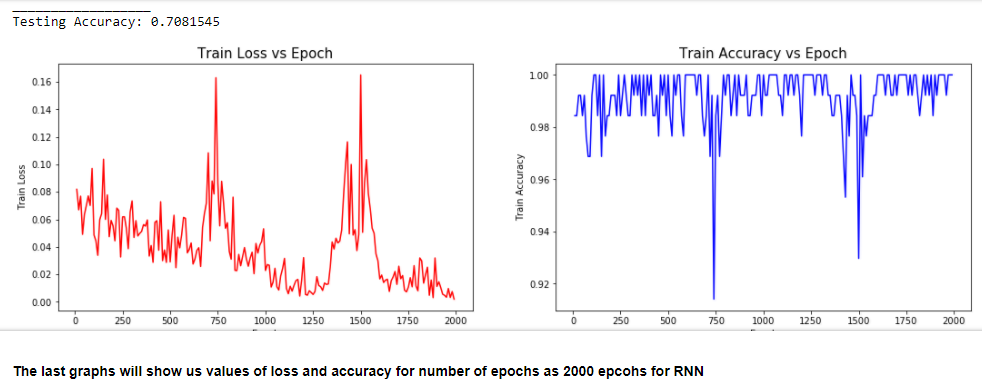


## Hyper Parameter Tuning Results for Recurrent Neural Networks Neural Network Results

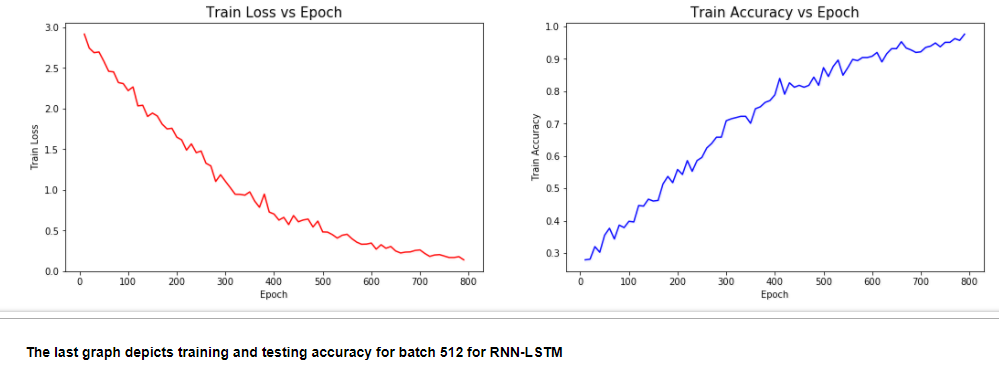
|  |  |  |  |
| --- | --- | --- | --- |
| Hyperparameter Name | Hyperparameter Values | Training Accuracy | Testing Accuracy |
| Number of Epochs | 2000 | 100% | 70.81% |
| Batch Size | 512 | 97.65% | 68.52% |
| Number of Neurons | 512 | 97.65% | 71.81% |
| Learning Rate and Number of Neurons | 0.001and 512 | 94.53% | 68.59% |
| Activation | Tanh | 92.96% | 68.81% |
| Optimizers | Adam | 94.53% | 68.59% |

**Graphical Representation of Results for the Recurrent Neural Network Model**

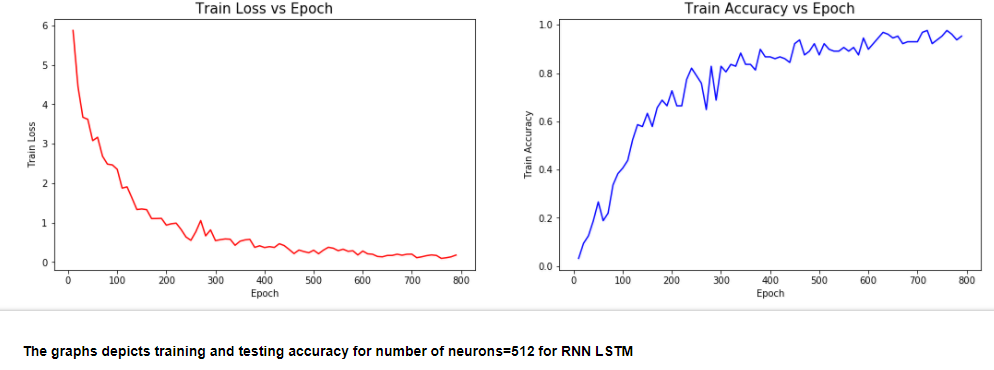
**Number of Epochs**



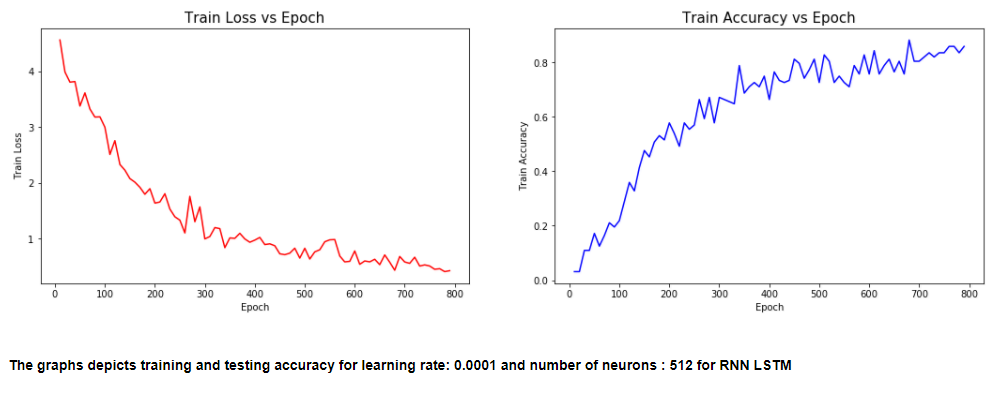
**Batch Size**



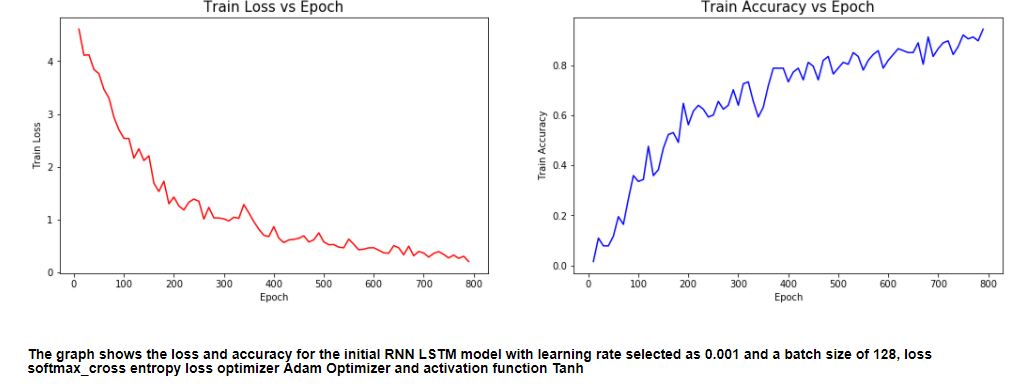
**Number of Neurons**



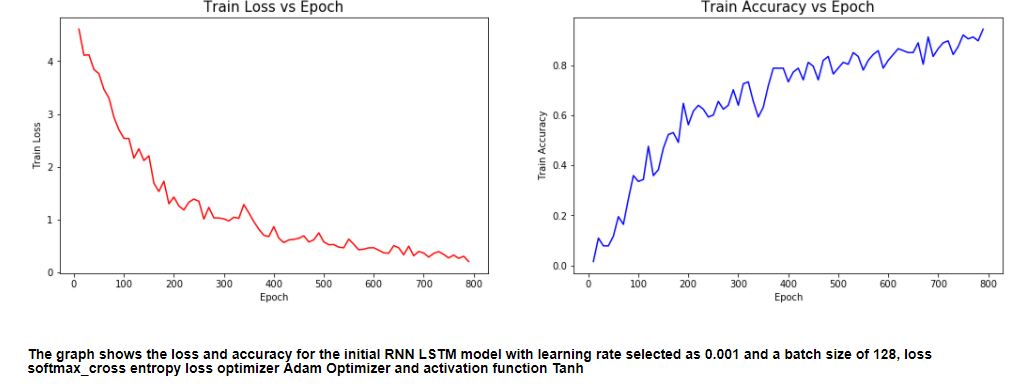
**Learning Rate and Number of Neurons**



**Gradient Estimation**



**Activation**



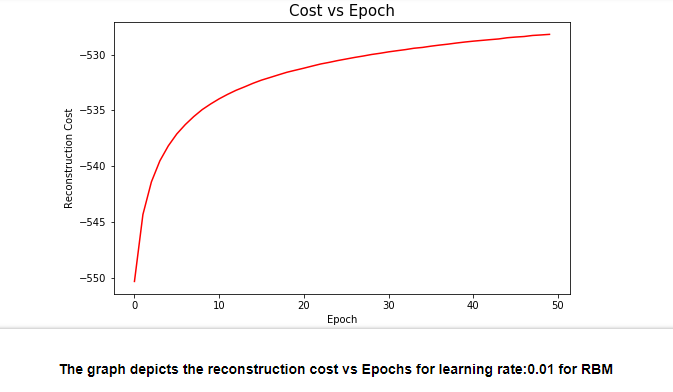
## Hyper Parameter Tuning Results for Restricted Boltzmann Machine Neural Network Results

Since, generation of images takes a long time in RBM’s I have restricted the training of the model to 50 epochs which has a loss of -525.82.

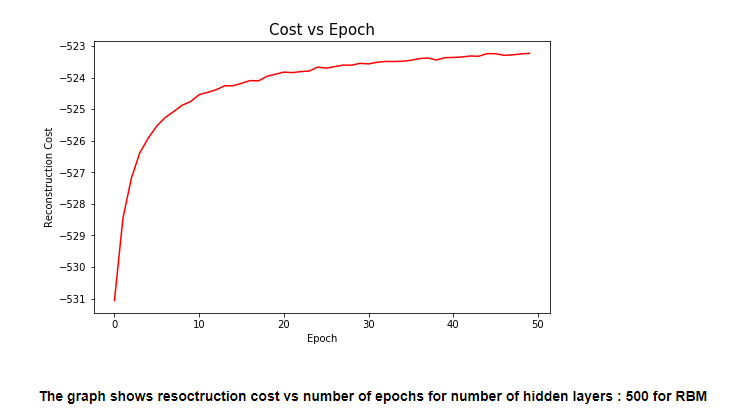
|  |  |  |
| --- | --- | --- |
| Hyperparameter Name | Hyperparameter Values | Reconstruction Cost |
| Activation | Sigmoid | -528.19 |
| Learning Rate | 0.01 | -528.19 |
| Initialization | Xavier Initialization | -527.42 |
| Number of Hidden Layers | 500 | -523.227 |
| Regularization | L1 | -528.32 |

Graphical Representation of Results for the Recurrent Neural Network Model

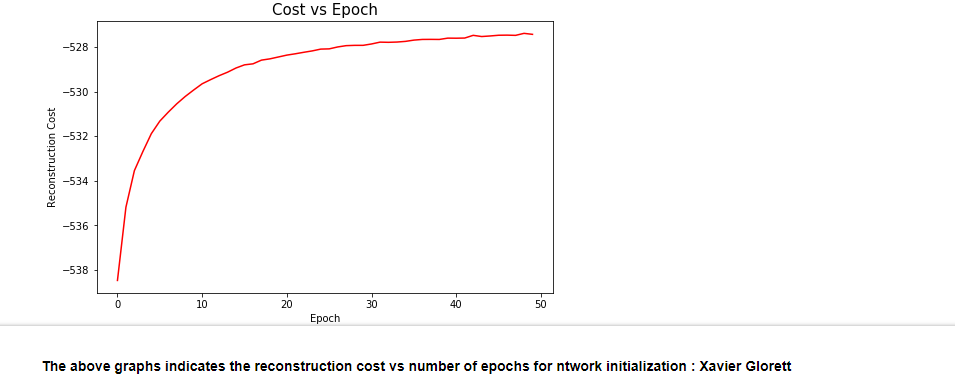
**Learning Rate**



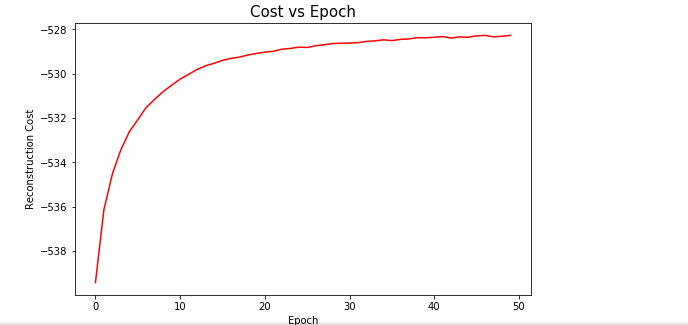
**Number of hidden layers**



**Network initialization**



**Regularization**



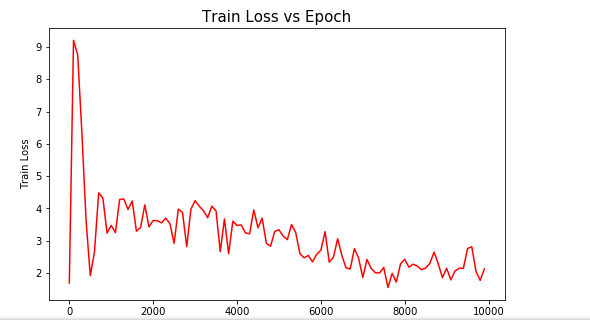
## Hyper Parameter Tuning Results for Generative Adversarial Neural Network Results

|  |  |  |  |
| --- | --- | --- | --- |
| Hyperparameter Name | Hyperparameter Values | Generator Loss | Discriminator Loss |
| Network Initialization | Xavier Initialization | 0.4891 | 4.003 |
| Activation (Generator and Discriminator) | ReLU | 2.12 | 0.9232 |
| Optimizer and Learning Rate | SGD and 0.1 | 0.3045 | 2.856 |

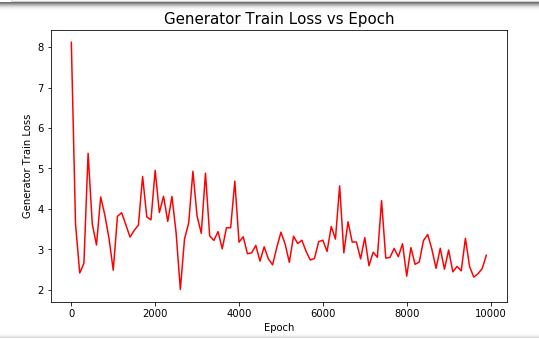
**Graphical Representation of Results for the Generative Adversarial Neural Network Model**

**Activation Function**

Generator-Loss

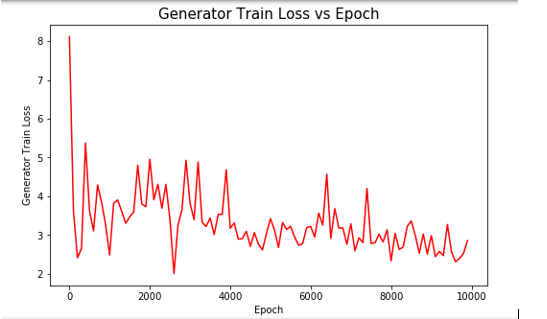


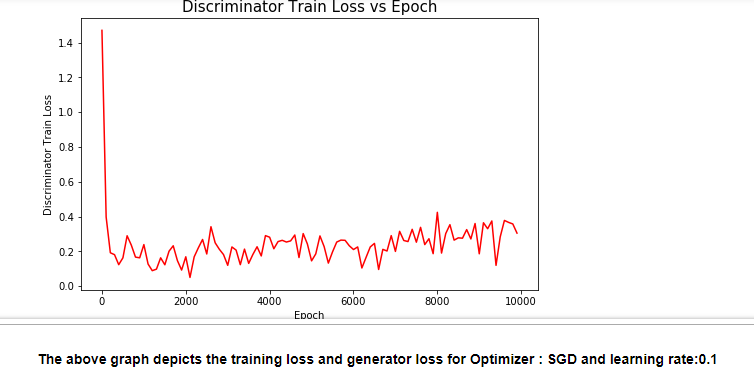
Discriminator-Loss



The above graphs depicts the Generator and Discriminator Loss for Activation Function ReLU

**Optimizer and Learning rate**



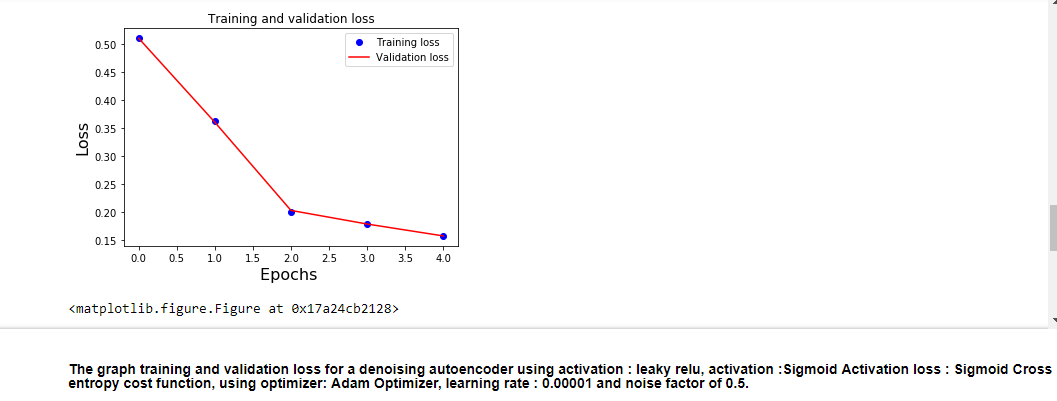


## Hyper Parameter Tuning Results for Autoencoder

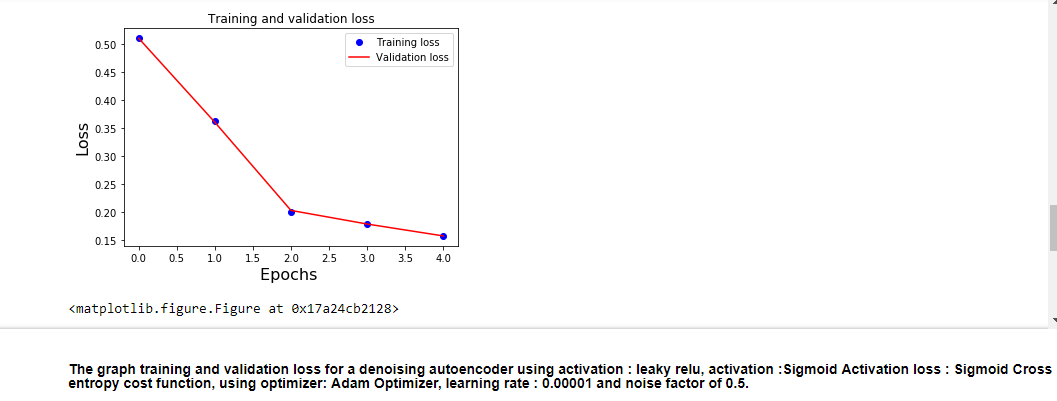
|  |  |  |  |
| --- | --- | --- | --- |
| Hyperparameter Name | Hyperparameter Values | Training Loss | Validation Loss |
| Loss | Sigmoid Cross Entropy Loss | 0.1575 | 0.1575 |
| Noise | 0.3 | 0.1256 | 0.1256 |
| Learning Rate | 0.01 | 0.0999 | 0.1006 |
| Gradient Estimation | Adam | 0.1575 | 0.1575 |

**Graphical Representation of Results for the Recurrent Neural Network Model**

**Optimizer**



**Loss**

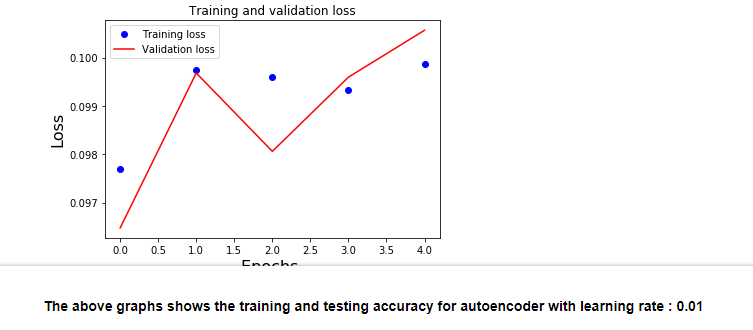


**Noise**

### 

The above graph depicts the Number of Epochs vs Loss for noise factor=0.2

**Learning rate**



# 

# Discussion

As mentioned in the methods the accuracy or loss of the models that is generated when the model is first run is taken as a benchmark. If tuning the hyper parameter improves the existing bench mark or close to the bench mark then it is considered as a prospective parameter to tune for the respective model. If the model does not surpass the bench mark, then I have considered the parameter that is present initially if present. Each of the model parameters are discussed below:

### Multi-Layer Perceptron Neural Network

From the results the model performs the best with a gradient estimation of Adagrad or Adam. The activation function ReLu and Sigmoid provide the same accuracy. Hence, using either would be useful. In the case of the combination of learning rate and number of epochs it was observed that using a combination of 0.01 and 1000 or 0.1 and 100 resulted in the same accuracy. Hence, selecting 0.1 and 100 epochs was a better option with lesser computation. Increasing the learning rate causes the network to learn faster in lesser number of epochs. However, it is possible that in some Neural Networks the learning rate must be low for the network to perform better. In the MLP Model , increasing the learning rate improved performance.

Hence, though these parameter values worked the best and must be tried first to improve accuracy it would be best practice to tune these hyper parameters (enlisted in decreasing order of importance) to improve the accuracy of the MLP Model

1. Number of epoch and Learning rate
2. Activation
3. Gradient Estimation

### Convolutional Neural Network

The CIFAR 10 Dataset has 10 classes of over 60000 images. Hence, training the algorithm takes extremely long. The Algorithm was first trained for 20 hours for 20000 epochs to reach a training and testing accuracy of 60.94% and 48.63% respectively. Hence, using a bench mark of 7000 epochs where the model reached an accuracy of 42.19% and 42.72% with stability was taken for hyper parameter tuning.

Though, it must be mentioned that the LENET5 model was not the best model as other CNN models with a different architecture performed better and took a comparatively lesser amount of time as referenced in [2]. The LENET5 Model was preferred as it was more conducive for hyper parameter tuning and a simple Tensorflow model implementation

Changing the activation for the CNN increases or decreases the accuracy. The activation function Sigmoid is the winner with the best accuracy. Training the model for greater number of epochs is preferred as the standard accuracy of CIFAR 10 can touch 80% or above if trained for a longer period. The CNN model works the best with a gradient estimation of Stochastic Gradient Descent. The combination of SGD and learning rate of 0.5 works best. But it is observed that the network would get lost in rise and fall of the Loss Scenario. This is evident as the accuracy would vary from 45 to 50 % for over 5000 epochs with losses varying drastically. There was stabilization brought in with initializing the network with Xavier or Gaussian Initialization. Hence, network initialization plays a pivotal role in CNN.

Hence, even though activation -Sigmoid, Optimization – SGD, Number of Epochs and Cost function Hinge Loss worked the best, hyper parameters to tune (enlisted in decreasing order of importance) while improving the performance of a CNN are as follows

1. Network Initialization
2. Number of Epochs
3. Gradient Estimation
4. Cost Function
5. Activation
6. Combination of Loss and Activation Function

### Recurrent Neural Network

The results for hyper parameter tuning of the RNN indicate that number of epochs as 2000, an increased batch size of 512, increasing number of neurons from 128 to 512 and the combined tuning of learning rate and number of neurons as 128 and 0.001 brought stability to the network. Though number of neurons as 512 provided greater accuracy, number of neurons as 128 and a learning rate of 0.01 brought stability to the network. This will ensure that the network plateaus well. Varying the gradient estimator Adam improved the performance as well. It was also observed during trials that a vanilla RNN provided poor performance without weight and bias initialization. Hence, presence of weights and bias plays an important role in tuning the RNN

Hence, though these values worked the best, list of prospective hyper parameters (enlisted in decreasing order of importance) to take into consideration while tuning the RNN – LSTM Model are as follows:

1. Number of Neurons
2. Number of Epochs
3. Activation Functions
4. Gradient Estimation
5. Batch Size
6. Learning Rate and Number of Neurons

### Restricted Boltzmann Machine

Tuning the RBM indicates that reconstruction cost reduced by a unit of 5 with the increase in the number of Hidden Layer to 500. On the contrary decreasing the number of hidden layers increased the reconstruction cost. The neurons are the most important processing parts of the neural network. Regularization of L1 decreased the reconstruction cost slightly. Additionally, learning rate of 0.01 also improved the reconstruction cost

Hyper Parameters to take into account (in decreasing order of priority) while tuning the RBM are as follows:

1. Number of Hidden Layers
2. Network initialization
3. Learning Rate
4. Activation Functions
5. Regularization

### Generative Adversarial Network

The performance of the GAN is said to be good when the generator loss increases and the discriminator loss decreases. This occurs as the discriminator finds the least difference between the generated image and actual image. The activation function ReLU provided a decreased discriminator loss in comparison to Leaky ReLU. Tuning the combination of optimizer and learning rate as Stochastic Gradient Descent and learning rate of 0.01 improved the performance of the GAN. Increasing the number of epochs will definitely also improve the performance further.

Hyper Parameters to take into consideration (In decreasing order of priority) while tuning a GAN are as follows

1. Learning Rate and Gradient Estimation
2. Network Initialization
3. Activation Function

### Autoencoder

Tuning the Autoencoder depicts that the hyper parameter noise, loss function as Hinge Loss and Sigmoid Cross Entropy seem to work the best. The learning rate of 0.01 provided a very low loss 0.0999 and 0.1006 for the encoder and decoder respectively. Reducing the noise helped improving the loss as presumably the auto encoder can learn the image better.

Hence, the prospective hyper parameters to tune first while tuning an Autoencoder are as follows:

1. Learning Rate
2. Noise Factor
3. Optimizer
4. Cost Function
5. Gradient Estimation

The above methods implemented are manual methods of tuning which are time consuming but provide results that help understand the impact of each parameter on the performance of the network. There are libraries in Keras that implement grid search which provide the most optimal parameters automatically, the effect of each parameter on the performance cannot be identified in this case. Hence, the manual search method helped to rank the hyper parameter based on their impact on the Neural Network.

For details on observation on each model refer to the link [21]

# 

# References

**MLP Neural Network**

The MLP code is based on the following:

[1] Tomar,Nikhil. “Iris Datset Classifivation using Tensorflow”. github.io, October 11. 2017. Web. 22 April. 2018.

(no licence specified)

Web Link:

https://github.com/nikhilroxtomar/Iris-Data-Set-Classification-using-TensorFlow-MLP/blob/master/iris.py

http://idiotdeveloper.tk/iris-data-set-classification-using-tensorflow-multilayer-perceptron/

Running the Tensorflow Session is based on the following article:

[2] Vikram K. “Deep Learning”. github.io, August 28. 2016. Web. 22 April. 2018.

(no licence specified)

Weblink:

https://github.com/Vikramank/Deep-Learning-/blob/master/Iris%20data%20classification.ipynb

.youtube.com/watch?v=a5BUunInTQU&t=1227s

[3] Understanding basics of Deep Neural Networks using the following videos by MIT

Deep Learning - https://www.youtube.com/watch?v=JN6H4rQvwgY&feature=youtu.be

**CNN**

[1] Understanding Xavier Initialization

Weblinks:

http://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initialization

[2] HVASS Laboratories - Tensorflow Tutorials 06 - CIFAR10

Weblink:

https://www.youtube.com/watch?v=3BXfw\_1\_TF4

[3] Magnus Erik Hvass Pedersen. “Tensorflow Tutorials”. github.io, Licenced by MIT ,December 16th. 2016. Web. 23 April. 2018.

Used for visualizing the CIFAR 10 Data

Weblink:

https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/06\_CIFAR-10.ipynb

[4] Ataspinar. “Building Convolutional Neural Networks ith Tensorflow”. github.io, December 16th. 2016. Web. 23 April. 2018.

LENET-5 and Like CNN code is based on the following article

https://github.com/taspinar/sidl

[5] Ataspinar. “Building Convolutional Neural Networks ith Tensorflow”., December 16th. 2016. Web. 23 April. 2018.

Web Link:

http://ataspinar.com/2017/08/15/building-convolutional-neural-networks-with-tensorflow/

**Other Links for Background research**

[6] https://en.wikipedia.org/wiki/Convolutional\_neural\_network

[7] https://en.wikipedia.org/wiki/Hinge\_loss

**RNN**

[8] The data is accessed from the following link

https://github.com/sumit-kothari/AlphaNum-HASYv2

[9] Jasdeep06. “Understanding-LSTM-in-Tensorflow-MNIST”. github.io, 10 Sept. 2017. Web. 18 April. 2018.

The netwrok Structure and exploratory data analysis functions is based from the following link

https://jasdeep06.github.io/posts/Understanding-LSTM-in-Tensorflow-MNIST/

[10] Kothari,Sumit. “Alpha- Numeric Handwritten Dataset”. , 10 Sept. 2017. Web. 18 April. 2018.

Data preprocessing is based on the below link

https://www.kaggle.com/usersumit/basic-eda-keras-ann-model/notebook

[11] **Other references**

Chentinghao,Tinghao. “Tensorflow RNN Tutorial MNIST”.,Medium, 9th January. 2018. Web. 18 April. 2018.

https://medium.com/machine-learning-algorithms/mnist-using-recurrent-neural-network-2d070a5915a2

https://github.com/chentinghao/tinghao-tensorflow-rnn-tutorial/blob/master/mnist\_rnn.ipynb

[12] Implement Tensorflow Next Batch, Stack Overflow

https://stackoverflow.com/questions/40994583/how-to-implement-tensorflows-next-batch-for-own-data?utm\_medium=organic&utm\_source=google\_rich\_qa&utm\_campaign=google\_rich\_qa

[13] Background Reserach on RNN LSTM done using the below article

https://deeplearning4j.org/lstm.html

**RBM**

[14] Omid Alemi. “Implementation of Restricted Boltzmann Machine (RBM) and its variants in Tensorflow”.,Licenced under MIT Licence,Medium, 15th August. 2017. Web. 23 April. 2018.

The code is based on code by Omid Alemi licenced under MIT

Website:

https://github.com/patricieni/RBM-Tensorflow/blob/master/Gaussian%20RBM.ipynb

https://github.com/omimo/xRBM/tree/master/examples

[15] MNist dataset Decription is based from this article

Website:

http://corochann.com/mnist-dataset-introduction-1138.html

https://en.wikipedia.org/wiki/Restricted\_Boltzmann\_machine

**GAN**

The code has been based on the below blog post with custom addition of functions and illustartions

[16] Kristiadi's,Agustinus. “Generative Adversarial Network using Tensorflow”.,Setember 17th 2016,. Web. 19 April. 2018.

Weblinks:

https://wiseodd.github.io/techblog/2016/09/17/gan-tensorflow/

https://github.com/wiseodd/generative-models/blob/master/GAN/softmax\_gan/softmax\_gan\_tensorflow.py

https://wiseodd.github.io/page3/

[17] **Back ground research**

https://en.wikipedia.org/wiki/Generative\_adversarial\_network

**Autoencoder**

[18] The code has been based on the below blog and custom changes have been made to this code to incorporate requirements of the project

Sharma,Aditya. “Understanding Autoencoders using Tensorflow”.,LearnOpenCV,November 15th 2017,. Web. 20 April. 2018.

Weblink :

Website : https://www.learnopencv.com/understanding-autoencoders-using-tensorflow-python/

[19] Mallick,Satya(spmallick),“Denoising-Autoencoder-using-Tensorflow.”,GitHub,LearnOpenCV,November 26th 2017,. Web. 20 April.

2018.

Weblink :

Website:https://github.com/spmallick/learnopencv/tree/master/DenoisingAutoencoder

[20] Tensorflow References for Background Study

https://www.tensorflow.org/api\_docs/python/tf/nn/sigmoid\_cross\_entropy\_with\_logits

<https://www.tensorflow.org/api_docs/python/tf/losses/sigmoid_cross_entropy>

[21] https://github.com/jainibhansali/BDIA/blob/master/Final\_Project\_BDIA\_Code\_Portfolio.ipynb