Contents

[ABSTRACT 2](#_Toc512118154)

[INTRODUCTION 3](#_Toc512118155)

[Structure and List of Hyper Parameters used for the Neural Networks 3](#_Toc512118156)

[Multi Layered Perceptron (MLP) 3](#_Toc512118157)

[Hyper Parameter List 4](#_Toc512118158)

[Convolutional Neural Network 4](#_Toc512118159)

[Hyper Parameter List 4](#_Toc512118160)

[Recurrent Neural Network 5](#_Toc512118161)

[Restricted Boltzmann Machine 5](#_Toc512118162)

[Generative Adversarial Network 5](#_Toc512118163)

[Autoencoders 6](#_Toc512118164)

[Methods 6](#_Toc512118165)

[Evaluation Criteria 6](#_Toc512118166)

[Process of Hyper Parameter Tuning 7](#_Toc512118167)

[Results 7](#_Toc512118168)

[Hyper Parameter Tuning Results for MLP Neural Network Results 7](#_Toc512118169)

[Hyper Parameter Tuning Results for Convolutional Neural Network Results 8](#_Toc512118170)

[Hyper Parameter Tuning Results for Recurrent Neural Networks Neural Network Results 9](#_Toc512118171)

[Hyper Parameter Tuning Results for Restricted Boltzmann Machine Neural Network Results 9](#_Toc512118172)

[Hyper Parameter Tuning Results for Generative Adversarial Neural Network Results 10](#_Toc512118173)

[Hyper Parameter Tuning Results for Autoencoder 10](#_Toc512118174)

[Discussion 11](#_Toc512118175)

[Multi-Layer Perceptron Neural Network 11](#_Toc512118176)

[Convolutional Neural Network 11](#_Toc512118177)

[Recurrent Neural Network 12](#_Toc512118178)

[Restricted Boltzmann Machine 12](#_Toc512118179)

[Generative Adversarial Network 12](#_Toc512118180)

[Autoencoder 13](#_Toc512118181)

[References 14](#_Toc512118182)

**Hyper Parameter Tuning for Deep Neural Networks**

Jaini Bhansali, Student under the guidance of Asst. Proff 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 also 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 he 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 the 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 LENET 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 was used. It uses a batch size of 128, learning rate selected is 0.001 and number of epochs as 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.

|  |  |
| --- | --- |
| 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 was 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 having 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 played significant hyper parameters 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 Loss was the least for reduced Noise Factor and Learning Rate.

The methods of implementation are described below

# Methods

### Evaluation Criteria

The accuracy or loss of the models that was 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 at the 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 on the basis of their accuracy, the performance of the RBM is measured on the basis of 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 the 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 10000 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 has to pre-processed and converted to one hot encoded values. The RNN was then tested for the various Hyper Parameters enlisted
4. RBM: In order 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 have to be pre-processed. After running the GAN with the initial model it was trained for the various enlisted hyper parameters.
6. Autoencoders: After initializing the model, the autoencoder was 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 set up. The details method of implementation can be found at **[1]**

# 

# Results

The objective of performing tuning for each Neural Network is to understand the hyper parameter that can be used to improve the performance of the network and also 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 or Adagrad | 98% | 97.72% |
| Activation Function | Relu or Sigmoid | 99% | 97.72% |
| Epoch and Learning Rate | 100 and 0.1 | 99% | 97% |

## 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 | 42.72% and 39.50% | 42.19% and 24.20% |
| Activation Function | Sigmoid | 50% | 41.09% |
| 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 | 42.72% | 42.19% |
| Gradient Estimation | Stochastic Gradient Descent | 42.72% | 42.19% |

## 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 | Softsign. Tanh | 92.96% | 68.81% |
| Optimizers | Adam | 94.53% | 68.59% |

## 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 and Activation | L1 and Sigmoid | -528.27 |

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

|  |  |  |  |
| --- | --- | --- | --- |
| Hyperparameter Name | Hyperparameter Values | Generator Loss | Discriminator Loss |
| Network Initialization | Xavier Initialization | 3.047 | 0.6848 |
| Activation (Generator and Discriminator) | ReLU | 2.12 | 0.9232 |
| Optimizer and Learning Rate | SGD and 0.1 | 0.3045 | 2.856 |

## 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 |

# Discussion

As mentioned in the methods the accuracy or loss of the models that was 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 at the initially. Each of the model parameters are discussed below:

### Multi-Layer Perceptron Neural Network

From the results it is clear that 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.

Hence, though these parameter values worked the best and must be tried first to improve accuracy it would be best practice to tune the following Hyper Parameters to improve the accuracy of the MLP Model

1. Optimizer
2. Activation
3. Number of epoch and Learning rate

### 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 50% and 41.09% with stability was taken for hyper parameter tuning. Taking a consistent value of the 7000th epoch was 42.72% and 42.19% for training and testing accuracy.

Changing the activation for the CNN will increase or decrease the accuracy. The activation function Sigmoid was 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 while improving the performance of a CNN are as follows

1. Network Initialization
2. Stochastic Gradient Descent
3. Cost 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 neuron 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 a number of Neuron 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 Tanh and Adam brought the same accuracy. 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 to take into consideration while tuning the RNN – LSTM Model are as follows:

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

### 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. Introducing Xavier Initialization did not help the network much but the combination of Network Initialization, Regularization of L1 and Sigmoid activation did decrease the reconstruction cost slightly. Additionally, learning rate of 0.01 also improved the reconstruction cost

Hence, though these values depicted in the results worked the best, Hyper Parameters to take into account while tuning the RBM are as follows:

1. Number of Hidden Layers
2. Regularization
3. Activation Functions
4. Regularization and Activation Functions
5. Learning rate

### Generative Adversarial Network

The performance of a Denoising GAN is said to be good when the generator loss increases and the discriminator loss decreases. This 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 Leak 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 while tuning a GAN are as follows

1. Activation Function
2. Learning Rate
3. Gradient Estimation

### Autoencoder

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

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

1. Activation Function
2. Noise Factor
3. Learning Rate
4. Optimizer

# References

[1] <https://github.com/jainibhansali/BDIA/blob/master/Final_Final_Project_BDIA.ipynb>

[2] Convolutional Neural Network with Tensorflow -

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

[3] Understanding LSTM in Tensorflow-

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

[4] Basic EDA Keras ANN Model-

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

[5] Tensorflow RNN Tutorial MNIST -

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

[6] Generative Adversarial Network using Tensorflow-

<https://github.com/wiseodd/generative-models/blob/master/GAN/softmax_gan/softmax_gan_tensorflow.py>

[7] Understanding Autoencoders using Tensorflow-

<https://github.com/spmallick/learnopencv/tree/master/DenoisingAutoencoder>

[8] Iris Data set Classfication

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

[9] Deep Learning

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