Initialization of Weights in Neural Networks

[1]Priyesh Patel, [2]Meet Nandu, [3]Purva Raut

[1,2]Student, [3]Assistant Professor
Department of Information Technology,
Dwarkadas J. Sanghvi College of Engineering, Mumbai, India

***Abstract*: When training and building a neural network, a number of subtle but important decisions needs to be taken. Zeroing down on the loss function to be used, the number of layers, kernel size, and the stride for each convolution layer, best-suited optimization algorithm for the network, etc. Compared to all these things, the choice of initialization of weights may seem trivial pre-training detail. But weight initialization contributes as a significant factor on the final quality of a network as well as its convergence rate. This paper discusses different approaches to weight initialization and compares their results on few datasets to find out the best technique that can be employed to achieve higher accuracy in relatively lower duration.**

***Index Terms*: Neural Networks, Variable Initialization**

I. Introduction

The ability of neural nets to synthesize complex nonlinearities which can give the best accuracy almost every time has made neural nets the most popular solution for almost all machine learning related problems.

There are 3 layers present in a neural network as shown by fig.1:

1. Input Layers: This layer takes large volumes of input data in the form of audio files, texts, numbers, image pixels, etc.
2. Hidden Layers: There can multiple hidden layers in a neural network. Hidden layers are responsible to perform mathematical operations, pattern analysis, feature extraction, etc.
3. Output Layer: This layer is responsible to generate the desired output.

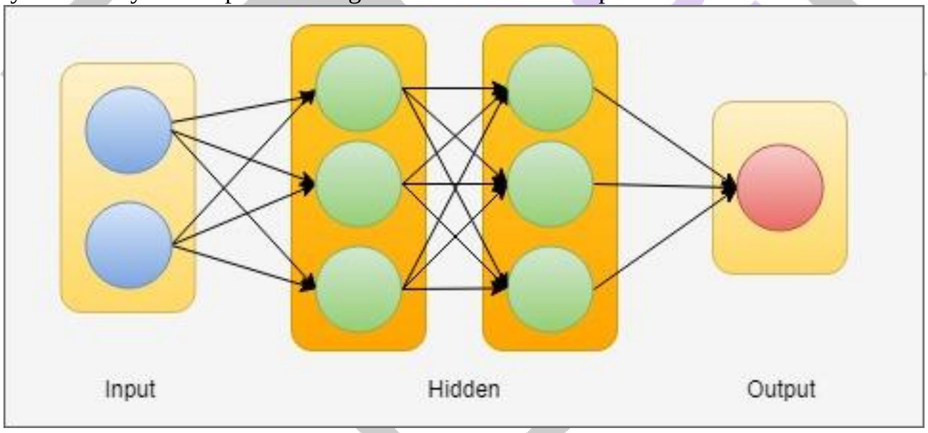

Figure 1 Three layers of a Neural Network

Each node in the network has some weights assigned to it. A transfer function is used to calculate the weighted sum of inputs and a bias is added as shown in fig.2.

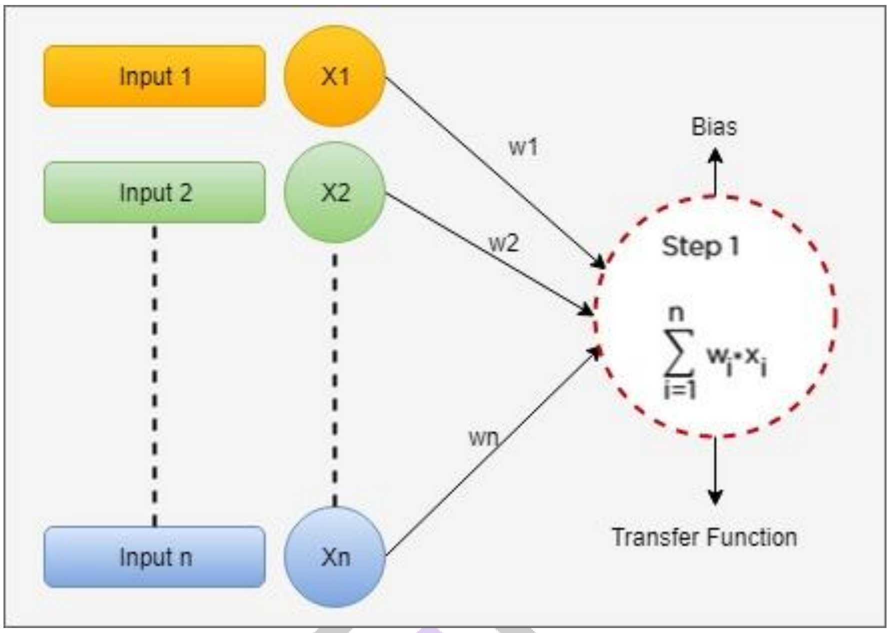

Figure 2 Weights and transfer function

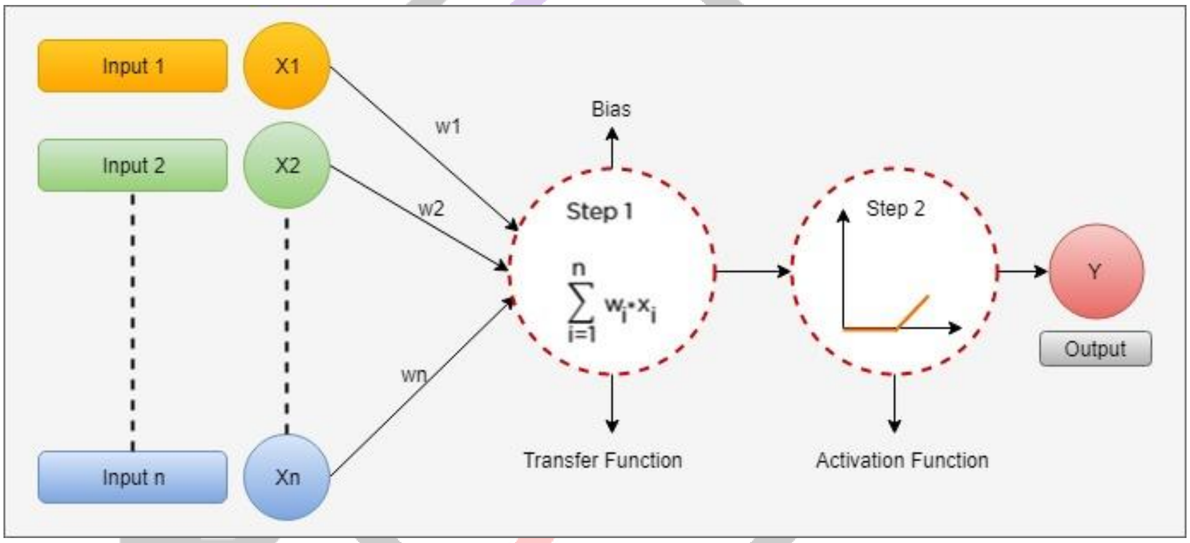

Figure 3 Activation function gives the output

The result of the transfer function is fed as input to activation functions as shown in fig.3. The decision of selecting the nodes to fire is taken by the activation functions. Based on the type of output, various activation functions that are used for specific purposes. Some of the activation functions are ReLU (Rectifier), Sigmoid, Step (Threshold), Softmax, Hyperbolic Tangent function, etc. All these activation functions have specific usages. Some of the functions are used only in the hidden layers, some in the output layers and there are a few which are used both in the hidden and output layers. The output layer produces the final predicted output based on the fired nodes.

In neural networks, a weight is present between every two adjacent layers. The linear transformation of these weights and the values in the previous layers are passed through a nonlinear activation function to produce the values of the next layer. During forward propagation, this process happens layer by layer and by back propagation, the optimum values of these weights can be found out to produce accurate outputs given an input.

Neural network research stalled in the late 80's and 90's mainly due to lack of good performance. There were a number of reasons for this – these reasons included poor computing speeds, lack of data, using the wrong type of non-linear activation functions and poor initialization of the weights in neural networks.

To help learn good mappings from input and output in neural networks, random initialization of weights is critical. Back-propagation may be trapped within multiple local minimums which are caused because the search space involving many weights during training is huge. To improve the chances of finding a good minimum during back-propagation, effective randomization of weights is necessary. It ensures that during training, the search space is adequately explored. However, carefully choosing and specifying the weight initialization randomization function reduces the risk of the training progress being slowed to the point of impracticality.

In this paper, we compare Neural Network with different initialization of weights on various datasets and compare them. Our aim is to predict the best way to initialize weights in a neural network for faster and better accuracy. This paper is organized in the following format. Section I introduces and provides a review of the Neural Network, Section II provides the information on the datasets used, Section III explains different initialization techniques compared in the paper, Section IV provides the comparison results and Section V concludes.

II. Data research

Three different datasets were selected for this study. The datasets selected were:

- MNIST database of handwritten digits: This dataset consists of 60,000 images. Each image is a 28x28 grayscale image of the 10 digits. Also, the dataset consists of 10,000 test images [1].
- Fashion-MNIST database of fashion articles: This dataset consists of 60,000 images along with 10,000 test images. Each image is a 28x28 grayscale image, associated with a label from 10 classes [1]. The class labels are:

Label	Description
0	T-shirt/Top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle Boot

- CIFAR10 small image classification: This dataset consists of 50,000 images along with 10,000 test images. Each image is a 32x32 color image, labeled over 10 categories [1]. The categories are:

Label	Description
0	Airplane
1	Automobile
2	Bird
3	Cat
4	Deer
5	Dog
6	Frog
7	Horse
8	Ship
9	Truck

III. Initialization of Weights

Zero Initialization

In zero initialization, weights of all the neurons are set to be zero. Initializing the weights with zero serves no purpose. All the neurons in each layer perform identical calculations and hence provide the same output. The entire deep net is thus equivalent to a single neuron and the prediction will be anything random which is of no use. Equation 1 gives the python code for the same.

w=np.zeros((layer_size[l],layer_size[l-1])) □□□

Random Initialization

In this, the weights are randomly distributed with a mean of zero and a standard deviation of one. This helps in symmetry-breaking and gives better accuracy. If a node in the hidden layer has inputs with these randomly distributed weights, the node will also be normally distributed around zero. But its variance and derived standard deviation will be larger than one. So the normal distribution of the hidden node will be broader than a normal distribution with a standard deviation of one. With this larger standard deviation, the value of the hidden node will take a number with is significantly larger or smaller than one. The activation function output will hence produce very minimal change. Equation 2 gives the python code for the same.

w=np.random.randn(layer_size[l],layer_size[l-1])*0.01□□□□□□□

Xavier Initialization

The variance of a given node depends on the variance of the weights from the previous layer, if the variance is reduced to 1/n where n is the number of nodes, it may produce better results [2]. The variance of the weights shifts from one to 1/n. Equation 3 shows the python code for the same.

w=np.random.randn(layer_size[l],layer_size[l-1])*np.sqrt(1/layer_size[l-1]) (3)

He-at-al Initialization

This method of initialization is similar to Xavier initialization with the only difference being that the factor of Xavier initialization is multiplied by two [3]. Initialization of weights in this method is done keeping in mind the size of the previous layer. This helps to reach a global minimum of the cost function with more efficiency and less time. Equation 4 shows the python code for the same.

w=np.random.randn(layer_size[l],layer_size[l-1])*np.sqrt(2/layer_size[l-1]) (4)

IV. Results

As seen in Table 1, the He-et-al initialization method has the highest average accuracy of 91.41%. He-et-al initialization had performed better than the rest of the initialization methods in all the test runs. Xavier initialization was the next best method giving an average accuracy of 90.03%. The poorest results were observed using the Zero initialization method, which gave an average accuracy of 10.28%.

Table 1. Model Accuracies on data.

Dataset	**MNIST**	**CIFAR-10**	**Fashion-MNIST**	**Average Accuracy**
Zero initialization	11.24	9.75	9.85	10.28
Random initialization	98.34	74.58	92.11	88.34
Xavier initialization	98.60	77.76	93.74	90.03
He-et-al initialization	98.92	80.17	95.15	91.41

Figure 4, 5 and 6 gives the visual representation of the weight initialization techniques presented in this paper with epoch to accuracy graphs.

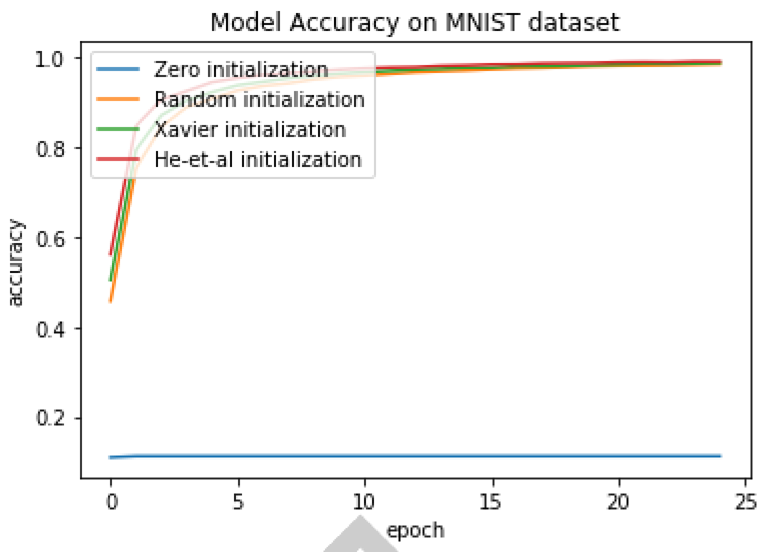

Figure 4 MNIST dataset graph

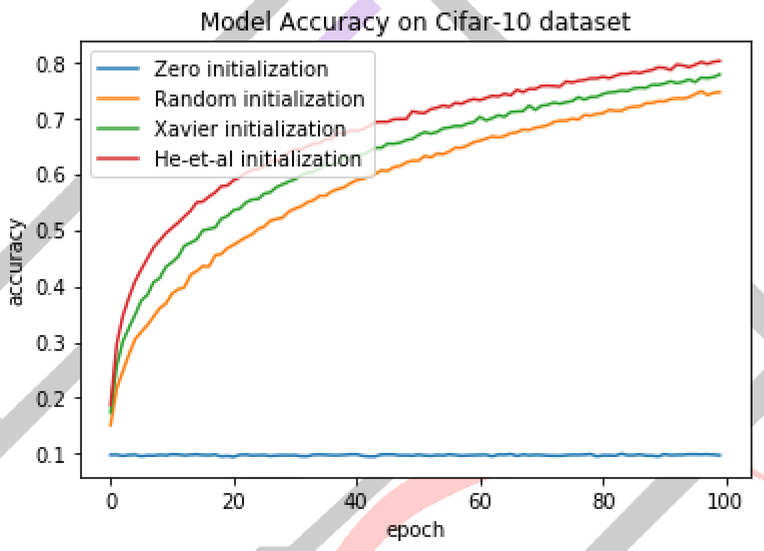

Figure 5 Cifar-10 dataset graph

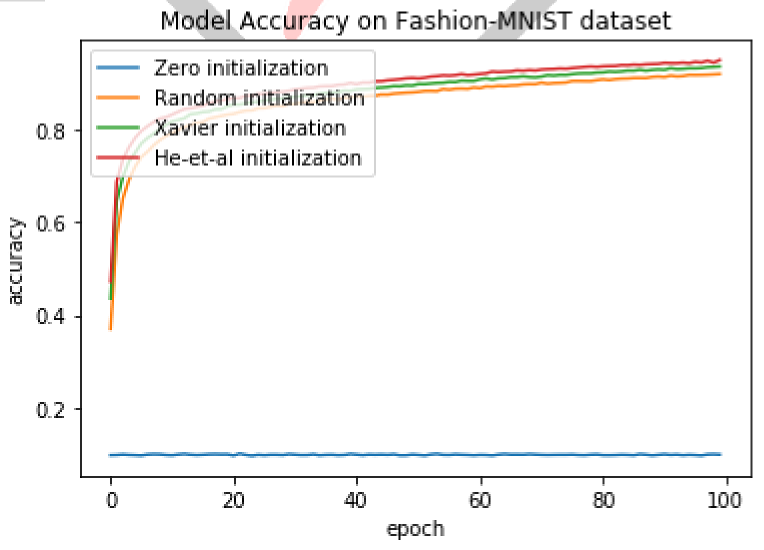

Figure 6 Fashion MNIST dataset graph

Also as seen in Table 2, the He-et-al initialization method has the least average loss of 0.2425. He-et-al initialization has performed better even in terms of loss than the rest of the initialization methods in all the test runs. Xavier initialization had the next best loss giving an average loss of 0.2838. The poorest loss was observed using the Zero initialization method, which gave an average loss of 2.3021.

Table 2. Model Losses on data.

Dataset	MNIST	CIFAR-10	Fashion-MNIST	Average Loss
Zero initialization	2.3012	2.3026	2.3026	2.3021
Random initialization	0.0549	0.7224	0.2095	0.3289
Xavier initialization	0.0464	0.6386	0.1666	0.2838
He-et-al initialization	0.0334	0.5647	0.1295	0.2425

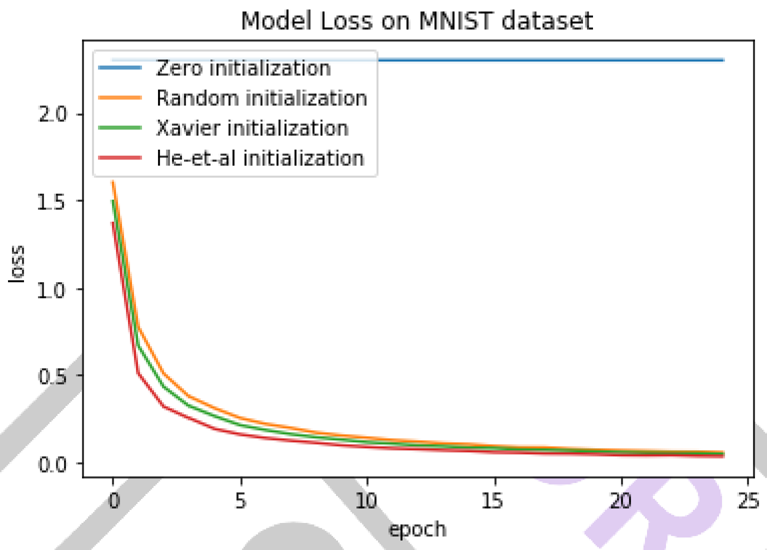

Figure 7 Model loss on MNIST dataset

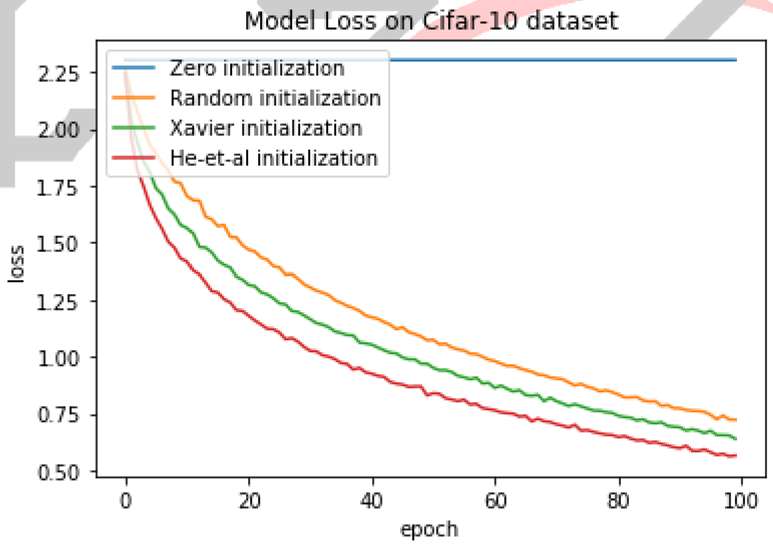

Figure 8 Model Loss on Cifar-10 dataset

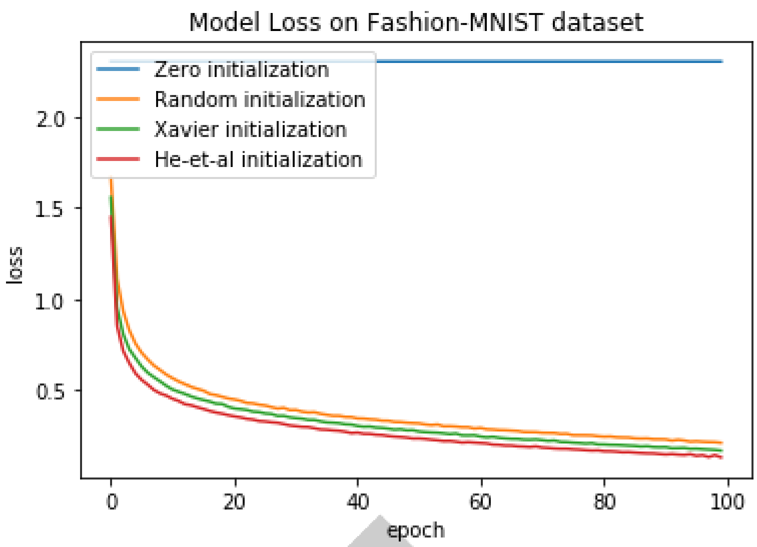

Figure 9 Model Loss on Fashion-MNIST dataset

V. Conclusion

In this paper, a study of four weight initialization methods namely Zero initialization, Random initialization, Xavier initialization, and He-et-al initialization are used to predict the optimal way to initialize the weights in a Neural Network. The results of our experiments indicate that the He-et-al initialization method of initializing weights performed better than the other available methods.

In the He-et-al initialization method, the weights are still random but differ in range depending upon the size of the previous layer of neurons. This provides a controlled initialization which in turn results in faster and efficient solving of vanishing and exploding gradient problem in Neural Networks [4].

It is very much evident that the use of the neural network is going to increase in terms of both the number and type of their applications in the future. The future of neural networks is bright and the current research seems to be heading in the right direction towards the ultimate goal of developing a human-like artificial brain or all artificial intelligence.

References

[1] Keras.io, 'Keras Documentation', 2018. Available: keras.io

[2] Glorot, X., & Bengio, Y. (2010, March). Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249-256).

[3] Ananthram, Aditya. 'Random Initialization For Neural Networks: A Thing Of The Past. Towards Data Science'. 25 Feb 2018. Available: towardsdatascience.com/random-initialization-for-neural-networks-a-thing-of-the-past-bfcdd806bf9e

[4] Pascanu, R., Mikolov, T., & Bengio, Y. (2012). Understanding the exploding gradient problem. CoRR, abs/1211.5063.