Learnable weight initialization for neural networks

Arkajit Bhattacharya
Delft University of Technology
Delft, Netherlands
arkajitb@gmail.com

Abstract

A new method of initializing the weights in deep neural networks is proposed. The method follows two steps. First, consider each layer as a model and perform a linear regression to keep the mean of the layer output to zero and variance after the data is passed through the activation function to one. Once each layer converges to the target mean and variance, initialize the weights of the original model with the learned weights.

Performance is evaluated on LeNet and ResNet18 architectures on FashionMNIST and Imagenette datasets. The activation functions used to analyze the performance are sigmoid, tanh and ReLU.

Findings show that the learned weights can perform similarly, and for certain scenarios, better than the different types of weight initializers used frequently in the field of deep learning. It is important to mention that this method requires the weights to be learned independently of the training of the model. Thus, there is a small time overhead. Moreover, it is required to adjust the hyperparameters(learning rate, epochs etc) to find the optimal weights.

The findings from this thesis can be used in the future to better understand how the gradient flow could be controlled through the network and finding a more generic approach towards the vanishing and exploding gradient problem.

The method requires to learn the weights followed by training the network. Learning the weights involves tweaking the hyperparameters(learning rate, number of epochs, etc). For future work, these aspects could be automated for the optimal performance of the network.

1. Introduction

1.1. Research and Problem

In recent years, deep learning has shown pronounced results in the field of image recognition, computer vision, and speech recognition tasks. These were achieved regardless of the issues faced while training deep neural networks. One of the most common issues faced while training a network is regarding the initialization of weights prior to the training of the network. Proper weight initialization prevents the layer outputs and the gradients from exploding or vanishing during the learning process. There has been a lot of research in recent years about the same. the authors of [4] introduced a proper weight initialization technique for the first time where they show that it is possible to prevent the activation output from exploding/vanishing by initializing the weights from a uniform distribution, followed by dividing(scaling) it with the number of incoming nodes for a single node. This method has proven to be very effective for certain criteria and activation functions which are symmetric around zero and outputs value within the range [-1, 1]. This observation was confirmed in [5], where the authors mention Xavier weight initialization's inefficiency when used with ReLU activation function. ReLU outputs all negative values to zero which leads to the dying ReLU problem, where nodes get neutralized, or to be precise, the weights associated with the nodes become zero. This means that the mentioned nodes cannot participate in the training process from that point onwards. Thus, the authors introduce a new weight initialization technique, popularly known as Kaiming weight initialization which is advisable to be used with ReLU activation function. That said, the issue of finding an independent weight initialization technique still prevailed.

In [8], the authors conclude that initializing weights with orthonormal values can be beneficial to the performance of the network, thus, establishing the fact that optimal weight initialization can be done regardless of the activation function used. This was followed by another technique, known as Layer Sequential Unit Variance Initialization(LSUV)[10], which learns the optimal weights by training each layer with the dataset provided to have activation output of one with a tolerance value. It achieved SOTA in few scenarios for image classification but failed to even train the model for sigmoid activation function.

LSUV introduced the idea of using the dataset provided to learn the optimal weights for a network. The proposed method uses a similar approach with a different algo-

rithm and criteria for the weights associated with the layers. Moreover, the proposed method can be seen as a normalization technique before training the network. Thus, it can be used as a replacement of batch normalization in deep neural networks.

1.2. Research Scope

The hypothesis related to the research is as follows:

1. **Is it possible to initialize weights of a network with optimal values independent of the activation function used?**
   Hypothesis: If each layer in the model is trained to have a pre-defined mean of 0 for the layer output and pre-defined variance of 1 for the activation output, the trained weights will be able to perform equally when compared with other weight initializers which are dependent on the activation function used, thus, providing a generalized approach towards the problem of optimal weight initialization.

2. **Is it possible to remove batchnorm from a deep neural network?**
   Hypothesis: The proposed method could be seen as a batch normalization process for the activation output before the training starts. Thus, it can remove batch-normalization from a network.

1.3. Contribution

The main contribution of this research is a new weight initialization technique that uses samples from the dataset provided, the activation function to be used, and the network to learn the optimal weights for initialization.

2. Related Work

To compare the performance of the new weight initialization technique, it is compared with kaiming and xavier weight initializers. The following sections describe the initializers in detail.

2.1. Xavier Weight Initializer

The goal of Xavier weight initialization[4] is to keep the variance to one for each layer during the training process. The weights are randomly initialized from a uniform distribution with mean zero and multiplied by $1/N_{Avg}$ where N_{Avg} is the average of input and output neurons. This initialization may result in dependence on the backpropagated gradient variance on the layer, and it might decrease throughout the training process. Thus, to add a normalization factor, the final weight initialization formula is:

$$W = U[\sqrt{6}/\sqrt{N_{in} + N_{out} + 1}, -\sqrt{6}/\sqrt{N_{in} + N_{out} + 1}]$$

where N_{in} is the number of incoming nodes and N_{out} is the number of outgoing nodes. This method is suitable for activation functions which are symmetric around zero and gives an output within the range of $[-1, 1]$.

2.2. Kaiming Weight Initializer

Kaiming weight initialization was introduced[5] since Xavier[4] couldn't perform well with ReLU[1] activation function. ReLU stands for Rectified Linear Unit. It fixed the problem of gradient saturation which was faced by Sigmoid and Tanh activation functions. For both tanh and sigmoid, gradient saturation occurs because it is not sensitive to data around the extremes. For instance, sigmoid scales down large values to 1 and small values to 0, thus, losing important information along the learning process. ReLU converts all values below zero and any value above zero as it is. This kills few neurons during the training process, thus, introducing sparsity in the network. The problem with combining Xavier with ReLU is related to this property. Since ReLU won't allow any value to pass if it is lower than zero, a weight initialization technique which aims at keeping the mean around zero and variance around one for a layer will lead to the elimination of most of the neurons during the training process, leading to the **dying ReLU** problem. Kaiming initialization is introduced keeping the mentioned issue in mind.

The steps for kaiming initialization are as follows:

- Create a tensor with the same size as of the input and initialize the values from a random uniform distribution.
- Scale all the values with $\sqrt{2}/N$ where N is the number of incoming nodes.

2.3. Layer Sequential Unit Variance Initialization

Layer Sequential Unit Variance Initialization(LSUV)[10] was introduced to create a weight initialization technique that doesn't depend on the activation function used for the network. For instance, Kaiming weight initializer was introduced only for ReLU activation function since it is not advisable to use Xavier weight initializer with Sigmoid or Tanh activation functions. It follows the algorithm stated below:

- Initialize the weights from a gaussian distribution with a variance of 1.
- Decompose the weight matrix with QR matrix decomposition method or Singular Value decomposition(SVD) method.
- Train the convolution and inner product layers to have an output variance of one with a small batch of samples from the dataset.

This research is inspired by LSUV as it first introduced the concept of using the dataset to train the weights.

3. Datasets and Preparation

3.1. Datasets

The datasets used for this research are explained in the following sections. The selection of the datasets were based on the complexity of the data. FashionMNIST is selected since it is easier to train and can be used for the sanity check. Once the practicality of the concept is established, the proposed method is used on imagenette which is a subset of Imagenet dataset.

3.1.1 FashionMNIST

FashionMNIST is created by zalando[11]. It consists of 60,000 training images and 10,000 test images. Each sample in the dataset is a 28x28 grayscale image with a total of ten classes. It was created as an updated version of MNIST[12]. Fig1 provides a visual representation of the dataset.

Figure 1. Collection of samples from FashionMNIST dataset[11]

3.1.2 Imagenette

Imagenette[7] is a subset of Imagenet[2] with only ten classes. It has three versions based on the size of images:Full size, 360 pixels and 120 pixels. Fig 2 provides a visual representation of samples from the imagenet[2] dataset.

3.2. Preparing data samples

To learn the weights, samples are collected randomly from the dataset provided. The labels are removed from the samples since the target of the model is to learn the optimal weights and not the classification of the data. The samples should be taken from the same dataset which is to be used for training the model once the optimal weights are learned.

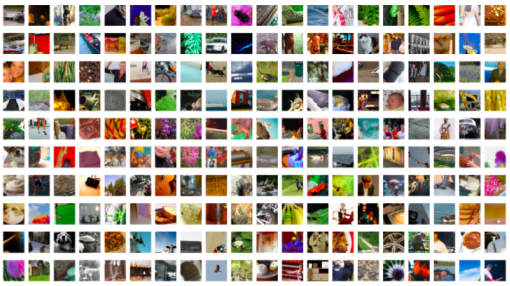

Figure 2. Collection of samples from Imagenet dataset[3]

4. Network Architectures

In this section, the deep learning models used for the research are described.

4.1. LeNet

The LeNet architecture was first mentioned by LeCun et al. in [9]. The mentioned architecture is selected because it has very few convolution layers and was adequate for testing with multiple activation functions and weight initializers. Fig 3 shows the architecture provided in the original paper.

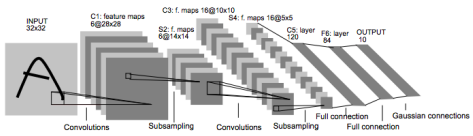

Figure 3. LeNet architecture provided in [9]. It consists of two sets of convolution layers, activation layers and pooling layers, followed by two fully connected layers and one decision layer

4.2. ResNet

ResNet architecture is introduced in [6]. It is arguably one of the groundbreaking works in the field of deep learning in the last few years. It solved the problem of vanishing gradient in deep networks with the introduction of identity shortcut connections,i.e., dividing the architecture into multiple blocks and adding the input to each block at the end of the block as well. ResNet is structured in layers, with each layer containing multiple blocks. The number of layers for different ResNets are the same. The number of blocks per layer is different for different architectures. ResNet18 has 18 layers in total.

Fig 4[6] shows a residual block with a skip connection.

There are five versions of ResNet[6] based on the number of layers in each block, namely, resnet18, resnet34, resnet50, resnet101, and resnet150. For this research, we have used resnet18 to keep it simple and easier to understand for the reader.

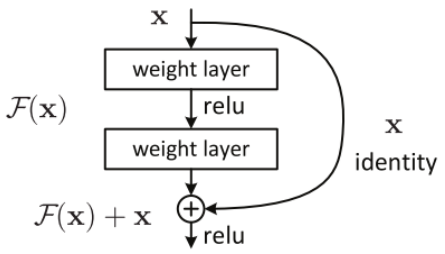

Figure 4. A residual block from ResNet architecue [6]

5. Hyperparameters

In this section, the hyperparameters selected for learning the weights are described. Each selection is justified and compared with all the possibilities.

5.1. Loss and Optimizer

Loss: The loss function selected for the proposed method is mean squared error loss. Learning the optimal weights involves keeping the mean to zero for the layer output and variance to one for the activation outputs which means that it is a regression problem.

Optimizer: Optimizers are functions that are used to update the weights or learning rate of the neural network with the aim of reducing the loss for each iteration. The optimizer used for this research is Stochastic Gradient Descent(SGD).

5.2. Learning rate

The learning rate is related to the task in hand and cannot be generalized. For instance, for LeNet, the learning rate used is 0.01. It might change for learning the weights for another architecture.

5.3. Step Learning

Step Learning is the process of decaying the learning rate based on the number of epochs provided. For this research, a learning decay of 0.01 is applied after every fifth epoch.

5.4. Constant Parameter(Alpha)

A constant parameter(alpha) is introduced to reduce the difference in magnitude, if any, between the mean loss and the variance loss. For instance, if the mean loss has a magnitude in multiples of 1000 and the variance loss is lower than one, the mean loss gets multiplied by alpha which scales down the magnitude of it which could be compared to the variance loss. The equation is provided for better understanding:

$$loss = (alpha * mean_loss) + variance_loss$$

Similarly, if the variance has a higher magnitude, it is scaled down using alpha:

$$loss = mean_loss + (alpha * variance_loss)$$

The hyperparameters are adjusted according to the dataset and model.

6. Approach

6.1. Standardization

The proposed approach is based on the concept of **standardization** of weights before the training starts for a network. In standardization, the features of the dataset are scaled to have a mean of 0 and a variance of 1. It results in all the values centered around zero with a standard deviation of one. It is an important feature when gradient descent comes into the picture. If the scales of the data features differ a lot, few weights will get updated at a faster rate when compared to the other neuron weights which is not advisable for the model to learn. Moreover, symmetric activation functions, like tanh assume that the weights are distributed around 0. Thus, keeping the mean of the layer output is important for the network to learn. To avoid the exploding/vanishing gradient problem, it is important to keep the variance within abound for each activation output. Keeping these points in mind, the approach can be described in two steps:

1. Train the model to keep the activation output variance to one. The equation for the step is $\sum_{n-1}^{0}(w*i)/n = 0$ where i is the incoming data, w is the weight associated and n is the number of nodes present in the layer.

2. Train the model to keep the mean to zero for each layer output. The equation for the step is $Var(X) = 1$, where X is the activation output.

The first step is self-explanatory. The layer is trained to keep the variance of the activation output to one using samples from the same dataset which will be used for training. This makes sure that the activation output never vanishes or explodes during a forward propagation of data and similarly, this ensures that the gradient also stays in check when the loss is propagated backward and the weights are updated accordingly. That said, if the layers are trained to have an activation output of one, the layer weights will get updated with each iteration. This means that which each iteration, the mean of the layer outputs will also change which is not the appropriate scenario since the updated weights are considered to be the optimal weights and it is advisable to initiate the weights of a layer with a mean of zero to prevent the layer outputs from vanishing or exploding throughout the training process.

6.2. Initializing weights for each layer

The weights of each layer are initialized from a normal distribution within an interval [0,0.1]. This decision could be justified by the fact that the aim of the proposed method is to learn the optimal weights, but it is important to initialize the weights in such a way that it is easier for the network to reach its target at the earliest. Thus, the weights are initialized randomly from the given interval such that all the values are near to zero and the difference is small between the values. Different intervals were tried and the mentioned interval achieved the best results.

6.3. Training each learnable layer and activation layer independently

All the layers with learnable weights and their corresponding activation layers are converted to models to be trained for the optimal weights.

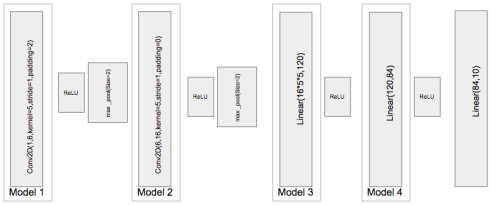

Figure 5. Each layer is considered as an independent model excluding the final layer. In LeNet, four new models are created, i.e., two convolution layers and two linear layers. Maxpool layers are not considered and added as separate entities but not involved in the learning process

Fig 5 gives a visual representation of the segmentation of all the layers. Each learnable layer and it's corresponding activation layer are considered as a model and trained separately to have a mean of 0 for the layer output and a variance of 1 for the activation output. The decision layer is excluded from the training process since the aim of the decision layer is to classify the data processed through the network.

Similar to LeNet, all the layers in ResNet18 are considered as separate neural network models and trained separately to learn the optimal weights. A visual representation of a single block with identity downsampling is provided to understand the structure of the learning process in Fig 6

The learning process for a layer is identical for both LeNet and ResNet. Thus, the visual representation of the same is provided in Fig 7

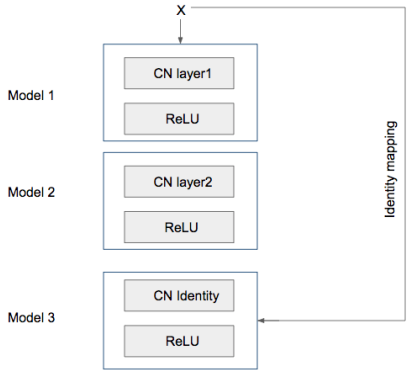

Figure 6. Each layer and it's corresponding ReLU activation layer is considered as a trainable model. The identity is recorded and transferred to the convolution layer used for downsampling and adding identity to the output

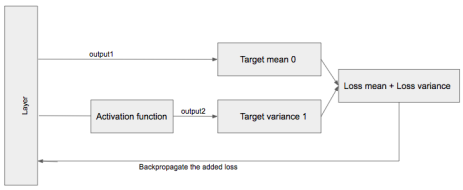

Figure 7. Data is processed through each layer.Mean loss is calculated for the layer output and variance loss is calculated for the activation output. The losses are added and back-propagated through the network. The weights are updated at each iteration

6.4. Greedy Approach

Greedy approach can be described as the method of dividing the problem into multiple parts in sequential order with a target of achieving the optimal result when the last task has been executed, without considering the whole picture.

The proposed approach follows the greedy algorithm. Each layer is converted to a model and the layers are trained sequentially, i.e., not more than one layer gets prioritized at a time. For instance, the second convolution layer in LeNet will only be trained if the first layer mean and variance loss converges.

The steps are described below in detail where two consecutive layers are considered for better understanding of the reader:

- Use the mini-batch created for learning the weights to train the first layer.
- Define the hyperparameters needed to train the layer.
- Make sure the loss converges for both mean and variance loss independently.
- Freeze the weights of the trained layer.

- Start training the second layer where the input to it is the output of the first layer.

The steps mentioned above are applied to all the layers in order. The proposed method makes sure that the previous layer has already been updated with the optimal weights.

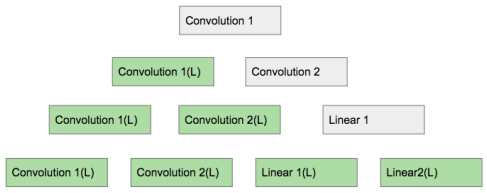

Figure 8. Each level describes the weight learning process of one layer at a time. The Green layer depicts that the layer has already been trained. The last level shows that all the layers are trained to be initialized with the optimal weight. To describe the greedy approach, LeNet5 architecture is selected

As shown in Fig 8, greedy algorithm is used to learn the optimal weights at each instance. First, weights for Convolution 1 are learned, and the weights are frozen for that particular layer. Then, the learning process starts again for the second convolution layer, where the data is first passed through the first convolution layer and that output is used as an input for the second convolution layer. This process is continued until the second last layer, i.e., excluding the decision layer. For visualizing the greedy approach, LeNet5 architecture is selected since ResNet18 architecture is complicated and difficult for the reader to understand from the visualization.

7. Ablation study

To check whether the proposed method initializes the weights with optimal values, it is important to check whether targeting only mean and variance would not give the same performance. The ablation study is divided into two parts, based on the architecture and the purpose of the research. The first set of ablation experiments is conducted to find out the practicality of the proposed method, i.e., training the layers to have a mean 0 for the layer output and variance 1 is able to learn the optimal weights for a given network and dataset. The second set of ablation experiments are conducted to understand the impact of the proposed method compared to the kaiming weight initializer after removing batch normalization from the network. The below-mentioned ablation experiments are followed to understand the impact of the alternatives for LeNet:

1. **LAE1:**Learn weights for mean 0 and variance 1 for the activation output, initialize the weights in the network and train the model. Test the model accuracy on the test dataset.
2. **LAE2:** Learn weights for only mean 0 for the activation output, initialize the weights in the network and train the model. Test the model accuracy on the test dataset.
3. **LAE3:**Learn the weights for only variance 1 for the activation output, initialize the weights in the network and train the model. Test the model accuracy on the test dataset.
4. **LAE4:**Learn the weights for mean 0 and variance 1 for the layer output, initialize the weights in the network and train the model. Test the model accuracy on the test dataset.

The below mentioned ablation experiments are followed to understand the impact of the alternatives for ResNet18:

1. **RAE1:**Remove BN and add kaiming weight initializer to all the layers. Train the model and check accuracy on test data.
2. **RAE2:** Remove BN and randomly initialize the weights for each layer. Train the model and check accuracy on test data.

7.1. Experiments and Results

In this section, the experiments for the proposed method with different combination of activation functions and weight initializers are discussed.

7.2. Ablation Experiments

In this section, results related to the ablation experiments are discussed. For LeNet, the activation function selected for the ablation study is ReLU.

Ablation Experiment	Activation Function	Test Accuracy
LAE1	ReLU	93.36%
LAE2	ReLU	92.24%
LAE3	ReLU	10%
LAE4	ReLU	93.48%

Table 1. Test Accuracy for Ablation experiments on LeNet

As shown in table 1, the model doesn't train for LAE3. For the other scenarios, the accuracy is high since the dataset selected for the same is FashionMNIST. That said, the proposed method is expected to give a higher accuracy than all the ablation experiments.

Ablation Experiment	Weight initializer	Test Accuracy
RAE1	Kaiming	10.46%
RAE2	Random	10.01%

Table 2. Test Accuracy for Ablation experiments on ResNet18

Table 2 shows the experiment results for the ablation experiments related to ResNet18. As shown, the model

doesn't train without batch-normalization. The activation function used for both the ablation experiments is ReLU. The proposed method is supposed to fix the issue of batch normalization and the model should be able to learn without the same.

8. Results and Experiments

8.1. LeNet

The purpose of using LeNet architecture is to verify the hypothesis **RQ1**,i.e., removing the dependency of weight initialization techniques on the activation function used.

8.1.1 Learning weights

There are two sets of convolution layers, activation layers and average pooling, followed by two linear layers and one decision layer in the lenet architecture. The convergence of the loss for the mean to be zero for the layer weights and variance to be one for the activation output starts with the first convolution layer. The dataset used for the first experiment is FashionMNIST. The activation function used is ReLU. Fig 9 shows the learning process for the first convolution layer for mean. It shows that the loss converges for mean for the first convolution layer. 10 shows the convergence of loss for the variance of the activation outputs for the first convolution layer. It is important to mention that while training the network, both the losses are back-propagated together and not one by one but the plots are created separately for mean and variance since the target is to reach convergence for both independently.

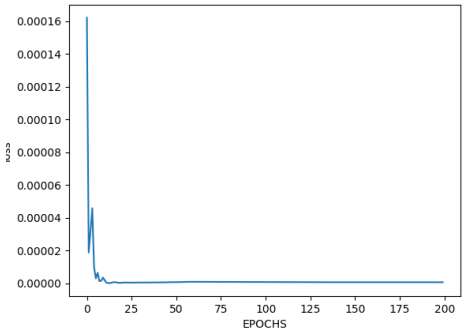

Figure 9. Loss vs Epoch for mean loss for the first convolution layer

Once the weights are learned for the first layer, the same procedure is followed for rest of the layers, excluding the decision layer. The learning graphs for the remaining layers are not provided since it is similar to the first layer.

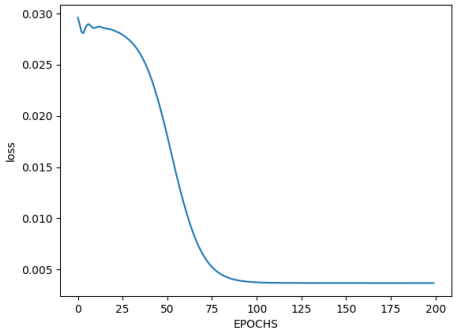

Figure 10. Loss vs Epoch for variance loss for the first convolution layer

8.1.2 FashionMNIST and analysis

FshionMNIST is used for sanity check of the proposed method. To understand the effect and to compare it's performance to the existing methods, all possible combinations of activation functions(Sigmoid,tanh,ReLU) and weight initializers(learned weights, Xavier, Kaiming) are used.

AF	LW	Kaiming	Xavier
Sigmoid	99.56%	99.02%	99.70%
Tanh	95.56%	92.24%	94.68%
ReLU	95.24%	96.36%	95.85%

Table 3. Test Accuracy for different combinations of weight initializers and activation functions

Table 3 shows that Learned weights(LW) performs slightly better than Kaiming and Xavier for Sigmoid and Tanh activation functions. For ReLU, the performance can be considered same as the difference in accuracy is low. FashionMNIST is a simple dataset to be trained. Thus, the test accuracy is high for all the combinations.

8.1.3 Imagenette and analysis

Imagenette is used to observe the performance of the method for datasets with more complexity, which can show the difference in training if the weight initializer used is not appropriate for the activation function. For instance, table 4 shows the drop in accuracy when tanh activation function is used. Theoretically, Xavier weight initializer should be able to perform well with symmetric activation functions like tanh, but for the given scenario, the proposed method performs better than all the combinations used.

As shown in table 4, the proposed method(LW) performs better than both kaiming and xavier for all the combinations of Activation functions(AF), except ReLU, which can be considered to have the same accuracy since the difference is less than one percent.

AF	LW	Kaiming	Xavier
Sigmoid	99.56%	65.43%	46.80%
Tanh	96.01%	46.02%	47.56%
ReLU	95.24%	92.40%	95.85%

Table 4. Test Accuracy for different combinations of weight initializers and activation functions

8.2. ResNet

The experiments done with ResNet18 are done with the purpose of verifying the hypothesis presented in **RQ2**, i.e., the proposed method can remove batch normalization from any network, regardless of it's depth.

The weight learning process for a layer in ResNet is similar to that of LeNet. Thus, a detailed description is not provided for the weight learning process. This section contains the experiment results and observations for Imagenette datasets. Different combinations of activation functions and weight initializers are not used since the aim for this architecture is to remove batch normalization from the network.

8.2.1 Imagenette and analysis

LW accuracy	Kaiming + BN accuracy
68.82%	72.48%

Table 5. Test Accuracy for Learned weigths(LW) and the combination of Kaiming and Batch Normalization(BN)

The table 5 shows that the proposed method is able to learn but the accuracy is lower by approximately four percent when compared to the combination of batch normalization(BN) and Kaiming weight initialization. The accuracy of the proposed method can be further improved by changing the hyperparameters and training the model in it's optimal conditions, since training a model depends on other factors and not only on the initialization of weights. That said, LW is still able to learn without introducing batch normalization in the network.

9. Conclusion

The research proposes a new technique to learn the optimal weights through training each layer of a network by following the concept of standardization. The hypothesis regarding the first research question(RQ1) is confirmed by the experiments done on LeNet architecture. The proposed method is combined with different activation functions and weight initializers to verify whether it can be used in any architecture regardless of the activation function. It achieves a better results than the existing weight initialization techniques in most of the scenarios. For the second hypothesis proposed in the second research question(RQ2), i.e., the proposed method can replace batch normalization in a network, experiments were conducted on ResNet18, and accuracy was compared by removing batch normalization and kaiming from the network and replacing it with the proposed method. The ablation study performed on ResNet18 shows that adding kaiming weight initializer or random weight initialization to the network without batch normalization cannot train the network. However, initializing the weights of the network with the method introduced can make the network learn. Moreover, there is a difference of only 5% between batchnorm and the proposed method which can be further improved by tweaking the parameters. The technique has a small time overhead since the weights need to be learned and not defined at the beginning of the training process. That said, the time overhead is small since learning the weights to have a mean of 0 and the activation output to have a variance of 1 is a linear regression problem.

10. Future Work

The method introduced can be further used to remove different types of norms, for instance, layer norm or group norm. A more generalized structure for learning the weights can be implemented, where hyperparameter optimization is already introduced for each layer for faster convergence.

References

[1] A. F. Agarap. Deep learning using rectified linear units.

[2] M. Frolovs. Imagenet.

[3] M. Frolovs. Imagenet dataset.

[4] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks, 2010.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, December 2015.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[7] J. Howard. Imagenette.

[8] W. Hu, L. Xiao, and J. Pennington. Provable benefit of orthogonal initialization in optimizing deep linear networks, Jan 2020.

[9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[10] D. Mishkin and J. Matas. All you need is a good init, Feb 2016.

[11] zalando. Fashion-mnist.

[12] zalando. Mnist.

Learnable Weight Initialization for Deep Neural Networks

Master's Thesis

Arkajit Bhattacharya