Demystifying Batch Normalization: Analysis of Normalizing Layer Inputs in Neural Networks

Dinko D. Franceschi *df2603*
Jun Hyek Jang *jj2883*
Rahul Vallivel Subbiah *rs3871*

Abstract

Batch normalization was introduced as a novel solution to help with training fully-connected feed-forward deep neural networks. It proposes normalizing each training-batch in order to alleviate the problem caused by internal covariance shift. The original method claimed that Batch Normalization (BN) must be performed before the ReLu activation in the training process for optimal results. Since the inception of Batch Normalization, it has become an indispensable mechanism in training deep neural networks. However, a second method has since gained ground which stresses the importance of performing Batch Normalization after the ReLu activation in order to maximize performance. Our work is the first to demystify the aforementioned debate and offer a comprehensive answer as to the proper order for Batch Normalization in the neural network training process. We demonstrate that for convolutional neural networks it is optimal to do ReLu activation before Batch Normalization, and that in Residual Networks with skip connections the order does not affect performance.

1 Introduction

The introduction of Batch Normalization has tremendously helped with the training of fully-connected feed-forward deep neural networks. Its promising results led to the mechanism becoming widely adopted and to it becoming the accepted norm in practice. It was initially proposed that Batch Normalization should be done prior to ReLu activation [1]. However, since then BN after ReLu became the norm when importing pre-built CNN models from PyTorch and Tensorflow [2][3]. In fact, the authors of these packages claim that BN must be performed after ReLu. This pivotal shift in the ordering has raised questions by many and has stirred a debate in the field as to which order is optimal. In addition to these 2 aforementioned views, interestingly, Ian Goodfellow remarked that the order actually does not matter at all [4]. These three viewpoints are all in stark contrast to one another. Notably, until our work, there has not been any thorough analysis of this problem. Our work is the first to offer a study into whether order matters, and if so, what the proper order should be for optimal performance.

To address this problem, we performed experiments on CNN and Resnet[7] architectures. We subsequently ran simulations for both the order Batch Normalization->ReLu (original) and ReLU->Batch Normalization and analyzed the results. Furthermore, we performed a thorough mathematical examination of the gradients while training for each layer to gain further insight. Our experiments clearly show that in Convolutional Neural Networks there is an improvement in performance when we switch the order from BN -> ReLu to ReLu->BN. This effect was not observed in Residual Nets which have skip connections. Examination of gradients shows that in CNN there is an improvement in gradient flow when the ordering is ReLu->BN. There is no improvement in gradients for Residual Nets.

2 Problem Decription

During the training process, the distributions of inputs of each layer shifts. This shift occurs for the inner nodes of the network. The change in the distributions of the nodes has profound negative effects on the training process. In fact, this shift occurs layer by layer and can significantly slow down the training process. This problem is also known as the Internal Covariate Shift [1]. The solution to this problem centers around the idea of ensuring that the distribution of each layer's inputs remains fixed during training. The premise is that this fixing is best achieved through taking the inputs x and enacting linear transformations so that they have means of 0 and variances of 1 [5].

However, it is quite costly to perform such transformations to the inputs of each layer. Thus, the initial proposal for Batch Normalization states that every scalar feature is normalized independently [1]. Moreover, because mini-batches are utilized in the SG training, each mini-batch will have its own mean and variance. This is critical so that the "statistics used for normalization can fully participate in the gradient backpropagation [1]."

Another benefit of Batch Normalization is that it removes the necessity for performing Dropout. Large deep neural networks can frequently overfit due to extremely large numbers of parameters. The models can be made very complex and they subsequently become prone to overfitting. This makes it even more difficult to amalgamate and combine multiple large deep neural networks in order to be used simultaneously. Dropout is a mechanism for alleviating this issue as it temporarily removes randomly certain units from the deep neural network. [6] The downside of Dropout is that it can increase the training time significantly and so the ability of Batch Normalization to supplant Dropout is another added benefit.

The initial result from Batch Normalization showed a significant improvement over existing methods. In fact, through combining multiple models trained using Batch Normalization, the results surpassed that of the best known system on ImageNet with 4.8% test error. Interestingly, this was even higher than the accuracy of human raters [1].

These results were very promising and subsequently this paper led to Batch Normalization becoming a norm in training fully-connected feed-forward deep neural networks. As mentioned, the initially proposed order of Batch Normalization and then ReLu has since been disputed by many in the field. Specifically, the reversed order of ReLu->BN has gained ground as a result of widespread use of pre-built CNN models from PyTorch and Tensorflow. However, there is no thorough examination of why one order of Batchnorm and ReLu is superior to another. The problem has not yet been properly addressed and we consider it imperative for this to be done. Below we offer results that indicate the optimal order of Batch Normalization in the training process as well as a mathematical explanation of the mechanism.

3 Mathematics of Batch Normalization

In order to gain further insight into the aforementioned problem of Batchnorm and ReLu order, we mathematically expand the backpropagation on a simple Residual Network, and see the effects that the order of Batch Normalization and ReLU has on the gradient of weights. Below we analyze backpropagation in the perspective of the Batch Normalization layer. To begin, let us look at the mathematical expression of the output of batch normalization in the equation 1 where $\hat{x}$ is batch normalized input, γ is the scaling factor, and β is the shift factor. The partial derivative of batch normalization with respect to γ and β are described in the equation 2 and 3, where Out is the value in the layers after the Batch Normalization layer.

$$BN = \gamma\hat{x} + \beta \tag{1}$$

$$\frac{\partial Out}{\partial \gamma} = \frac{\partial Out}{\partial BN}\frac{\partial BN}{\partial \gamma} = \frac{\partial Out}{\partial BN}\hat{x} \tag{2}$$

$$\frac{\partial Out}{\partial \beta} = \frac{\partial Out}{\partial BN}\frac{\partial BN}{\partial \beta} = \frac{\partial Out}{\partial BN} \tag{3}$$

3.1 Convolution Neural Network - ReLu BN

To demonstrate the backpropagation, we consider a CNN model with convolution blocks, and we analyze the backpropagation of the batch normalization layer at the end of the first convolution block. In the case of model with order of ReLU and BN in the forward pass, the Out will be weights from the convolution block following the BN layer. Then we can express the equation 2 and 3 in terms of partial derivative of the weights with respect to BN and arrive at equations 4 and 5. In the equations, $\frac{\partial Out}{\partial BN}$ will be an array of values ranging from negative infinity to positive infinity. $\hat{x}$ will be Batch Normalized ReLU activated values that will also range from negative infinity to positive infinity. Therefore, the resulting gradients in equation 4 and 5 will be ranging from negative infinity to positive infinity.

$$\frac{\partial Out}{\partial \beta} = \frac{\partial Out}{\partial BN} \sim \frac{\partial w_i}{\partial BN} = \begin{bmatrix} 0 & -0.2 & \dots \\ -1 & 0.6 & \dots \\ \vdots & \vdots & \vdots \end{bmatrix} \tag{4}$$

$$\frac{\partial Out}{\partial \gamma} = \frac{\partial Out}{\partial BN}\hat{x} \sim \frac{\partial w_i}{\partial BN} \cdot \hat{x} = \frac{\partial w_i}{\partial BN} \cdot BN(ReLu(Input))$$

$$= \begin{bmatrix} -0.8 & 0.2 & \dots \\ -1 & -0.1 & \dots \\ \vdots & \vdots & \vdots \end{bmatrix} \cdot \frac{\partial w_i}{\partial BN} = \begin{bmatrix} -0.1 & 0.7 & \dots \\ -1.3 & 0.8 & \dots \\ \vdots & \vdots & \vdots \end{bmatrix} \tag{5}$$

3.2 Convolution Neural Network - BN ReLu

In the case of the model with order of BN and ReLu in the forward pass, the Out will be ReLu activated values. Then we can express the equation 2 and 3 in terms of partial derivative of ReLu activated values with respect to BN, and arrive at the equation 6 and 7. In this case, by the mathematics of ReLu, all the negative input to ReLu will become zero. Therefore, the ReLu activated values will range from zero to infinity, and will significantly have more values corresponding to zero compared to the counterpart model with opposite order of ReLu and BN. Similarly, in the partial derivative of ReLus, the values will be semi-definite positive, and will include much more zero values compared to the equation 4 and 5. In the end, the order of BN followed by ReLu results in 'more' sparse gradient, and during training, we do not want zeros in our gradients because zero gradient cannot reduce the loss function to update the weights. The model will train slower and less effectively when there is more sparsity in the gradients.

$$\frac{\partial Out}{\partial \beta} = \frac{\partial Out}{\partial BN} \sim \frac{\partial ReLu_i}{\partial BN} = \begin{bmatrix} 0 & 0.2 & \dots \\ 1 & 0.6 & \dots \\ \vdots & \vdots & \vdots \end{bmatrix} \tag{6}$$

$$\frac{\partial Out}{\partial \gamma} = \frac{\partial Out}{\partial BN}\hat{x} \sim \frac{\partial ReLu_i}{\partial BN} \cdot \hat{x} = \begin{bmatrix} 0 & 0.2 & \dots \\ 0 & 0.6 & \dots \\ \vdots & \vdots & \vdots \end{bmatrix} \cdot \hat{x} = \begin{bmatrix} 0 & 0.2 & \dots \\ 0 & -0.1 & \dots \\ \vdots & \vdots & \vdots \end{bmatrix} \tag{7}$$

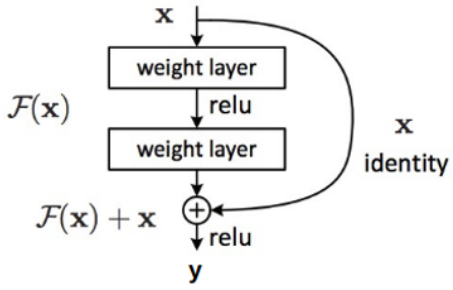


Figure 1: Residual Network with Skip Connection

Therefore, in the case of Convolution Neural Network (CNN), we expect the model to perform better if the ReLu layer comes before the Batch Normalization layer in the forward pass as there will be more non-zero values in the gradient.

3.3 Residual Neural Network

In the case of Residual Neural Network with skip connection(ResNet, DenseNet), we expect the order of ReLu and Batch Normalization layers to not affect the result because of the skip connection layers. Let us consider a model with a skip connection like in Figure 1. In this particular model, if we perform the backpropagation, we can brake it down into two parts: forward pass term and residual term. In equation 8, the forward pass term is $\frac{\partial Out}{\partial x}$ and the residual term is $\frac{\partial Out}{\partial x} \cdot F^*(x)$.

$$
\begin{aligned}
& y = x + F(x) \frac{\partial Out}{\partial x} = \frac{\partial E}{\partial out} \cdot \\
& \frac{\partial y}{\partial x} = \frac{\partial Out}{\partial y} \cdot (1 + F^*(x)) \\
& = \frac{\partial Out}{\partial y} + \frac{\partial Out}{\partial y} \cdot F^*(x)
\end{aligned}
\tag{8}
$$

In the Residual Network with skip connection, if the gradient of residual term is much larger than the forward pass term, the residual term dominates the forward pass term during the backpropagation, and the forward pass term does not have much impact on the overall gradient value (equation 9). Similarly, if the gradient of forward pass term is much larger than the residual term, the forward pass term would dominate the residual term, and the residual term would not have much impact on the overall gradient value (equation 10). Therefore, if there is a skip connection, it can be seen as the model having '2' paths to perform the backpropagation, and the order of BN and ReLu layers would not matter as the gradient can flow through the skip connection path to reach earlier layer and compensate the sparse gradients through the convolutional layers.

$$
\begin{aligned}
& When \quad F^*(x) \gg \frac{\partial Out}{\partial y}, \\
& \frac{\partial Out}{\partial x} \sim \frac{\partial Out}{\partial y} \cdot F^*(x)
\end{aligned}
\tag{9}
$$

$$
\begin{aligned}
& When \quad \frac{\partial Out}{\partial y} \gg F^*(x), \\
& \frac{\partial Out}{\partial x} \sim \frac{\partial Out}{\partial y}
\end{aligned}
\tag{10}
$$

4 Main Results

We selected a simple CNN and a Residual Net as ideal candidates for our experiments. We performed our simulations by training these models with Batchnorm followed by ReLu configuration and ReLu followed by Batchnorm configuration. Afterwards, we examine the training accuracy, test accuracy, and the training loss. With the exception of Batchnorm and ReLu, all other hyperparameters were the same. The results in Table 1 and Table 2 indicate that ReLu -> Batchnorm order performs better for

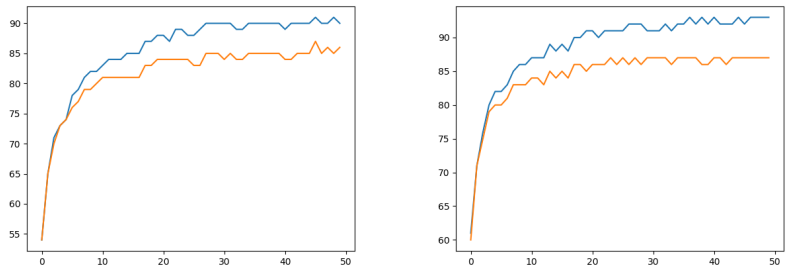
	Training accuracy	**Test Accuracy**	**Training Loss**
Batchnorm -> ReLu	91%	86%	0.271911
ReLu -> Batchnorm	**93%**	**87%**	**0.2120195**

Table 1: CNN experiment results

	Training accuracy	**Test Accuracy**	**Training Loss**
Batchnorm -> ReLu	98%	90%	0.1632
ReLu -> Batchnorm	98%	90%	0.1627

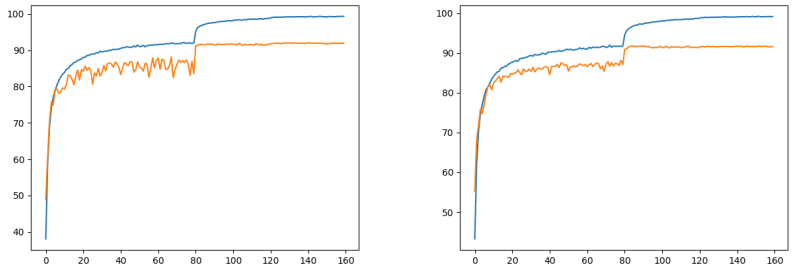
Table 2: Residual Net experiment results

CNN and that there is no difference is performance when order of Batchnorm and ReLu is changed for Residual Nets.



(a) Batchnorm -> ReLu (b) ReLu -> Batchnorm

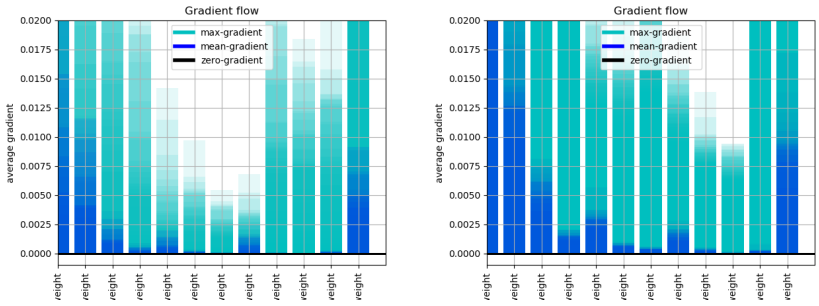
Figure 2: Training vs Test accuracy plot for CNN



(a) Batchnorm -> Relu (b) Relu -> Batchnorm

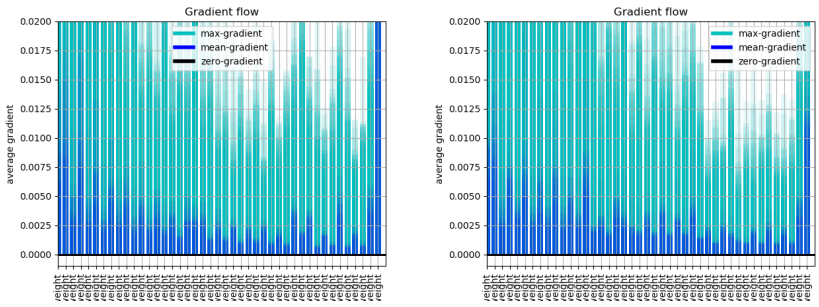
Figure 3: Training vs Test accuracy plot for Residual Net

The results clearly indicate that ReLu before Batchnorm is optimal. In order to further investigate why this behaviour is noticed in CNNs but not in Residual Nets, we recorded the gradients flowing through each layer for both of the models and for both configurations. Results are shown below.



(a) Batchnorm -> ReLu (b) ReLu -> Batchnorm

Figure 4: Gradient flow history plot for CNN



(a) Batchnorm -> ReLu (b) ReLu -> Batchnorm

Figure 5: Gradient flow history plot for Residual Net

The gradient plots clearly indicate that the ReLu -> Batchnorm configuration has more gradient flow than the Battchnorm -> ReLu configuration.

4.1 Inference from the experiments

From figures 4 and 5, we can see that the gradient flow for ReLu -> Batchnorm is better than Batchnorm -> ReLu for CNN and there is no difference in gradient flow for Residual Networks for either of the configurations.

Our reasoning is that this behaviour is not noticed in Residual Networks because of the skip connections. The skip connections provide an unimpeded path for the gradients to flow to the earlier layers, thereby compensating the poor gradient flow through the convolutional and Batchnorm layers.

5 Discussion

Our work has shown that for convolutional neural networks it is optimal to do ReLu activation before Batch Normalization, and that in Residual Networks with skip connections the order does not affect performance. In CNNs, the models trained and performed better when the ReLu layer is placed before the Batch Normalization layer as a result of better gradient flow. In Residual Neural network models with skip connections, there was no noticeable difference in training, performance, and gradient flow when we placed ReLu and BN layers in different order. This is explained by the skip connections which provide a path for the gradients to reach the previous layers.

In the future, we will investigate effects of Batch Normalization on different loss functions and optimizers for both CNNs and Residual Networks with the aim of discovering the optimal loss function and optimizer.

References

[1] Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

[2] Chollet, F. (2015). Keras.

[3] Paszke, A., Gross, S., Chintala, S., Chanan, G. (2017). Pytorch. Computer software. Vers. 0.3, 1.

[4] Goodfellow, I. (2016). Chapter 8: Optimization for Training Deep Models [Deep Learning Book]. Retrieved from Deep Learning Book.

[5] LeCun, Y. A., Bottou, L., Orr, G. B., Müller, K. R. (2012). Efficient backprop. In Neural networks: Tricks of the trade (pp. 9-48). Springer, Berlin, Heidelberg.

[6] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929-1958.

[7] He, Kaiming and Zhang, Xiangyu and Ren, Shaoqing and Sun, Jian (2016). Deep Residual Learning for Image Recognition. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).