Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Abstract

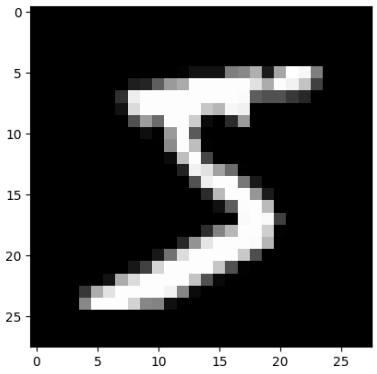
Stochastic Gradient Descent is usually a powerful method to train models. However, poor initialization and bad hyperparameters settings could lead to inefficient or even crashed models. For example, learning rate controls stride length in each update, when setting low, the model could take a lot of time to converge and probably stuck at the local minimum, which is a suboptimal result. If learning rate is high, we could significantly reduce the training time and bigger chance to find optimal solution, but we might face the risk that the model could not even converge, since the model may "jump" around the minimum and could not reach the lowest point.

To solve these problems, we gladly introduce Batch Normalization which is proposed by Sergey Ioffe and Christian Szegedy. Batch Normalization reduces the importance of initialization and enables a larger learning rate, which speeds up the training process and improves the performance at the same time. According to the paper, Batch Normalization meant to reduce Internal Covariate Shift, which is change of distribution during previous layers and cause the model must re-learn the pattern lying under the data. The concept of Batch Normalization will be discussed in later content.

1 Datasets

1.1 Small dataset

For the first experiment we used MNIST, the same dataset used in the paper. This is composed of high-quality and curated images of handwritten numbers from zero to nine. It contains 70,000 binary images (60,000 for training and 10,000 for testing), 28x28 pixels. As an example, in the figure below we can see an image classified as number 5. An advantage of the MNIST dataset is that it can be easily imported to Python using keras.datasets library.



1.2 Large dataset

To further test our model, we trained on BNG(glass) dataset (available on https://www.openml.org/search?type=data&status=active&id=265). It is composed by 137,781 datapoints, classified in 7 categories with 9 features. There was no missing data. It can be imported into Python using sklearn.datasets library. Differently from MNIST, this is tabular data, so at the beginning of our model there is no need to flatten the data.

2 Problem definition

2.1 Task Definition

The concept of Batch Normalization is simple; it basically normalizes the inputs of each layer based on batches. To implement Batch Normalization, we add a layer before each layer and normalize the inputs. Within the Batch Normalization layer, once the data entry achieves pre-defined batch size, it normalizes the inputs with in-batch mean and in-batch variance, a constant ϵ is added to variance to avoid divided by 0.

Also, the authors introduced two trainable parameters gamma and beta, which performs linear transformations for each data point. The output of the layer becomes $y = \gamma \dot{x} + \beta$ where $\dot{x}$ is the normalized data point. These parameters allow Batch Normalization to keep flexibility.

2.2 Algorithm Definition

To implement Batch Normalization, we replicated the experiment from the paper. Since our dataset (MNIST) are images, first we need to flatten the input. Later we use three hidden layers, with 'sigmoid' activation function. This was implemented with the package Tensorflow. To build the 'BatchNormLayer', we created a layer within Tensorflow via subclassing.

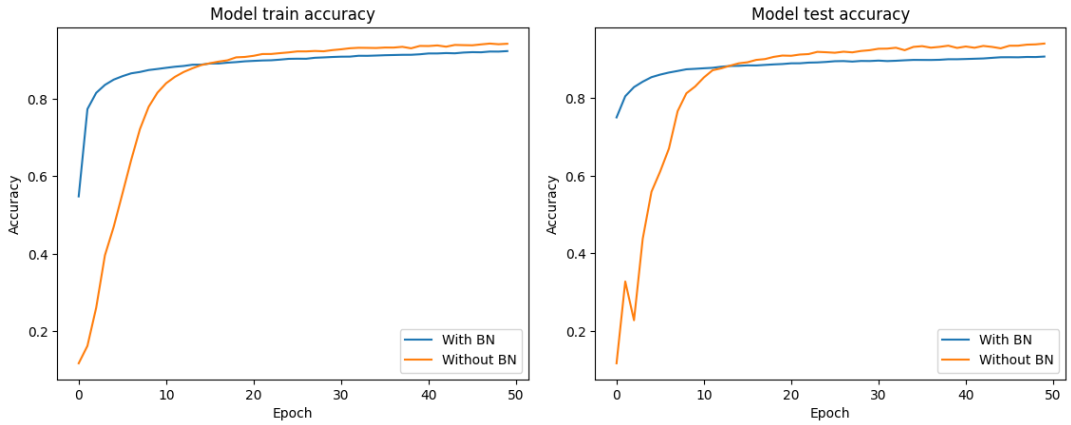
As in the paper, we used batch size of 60, loss function is cross entropy, stochastic gradient descent as our optimizer and we added a callback function 'PercentileLogger' to keep track of the dispersion during training.

3 Experimental evaluations

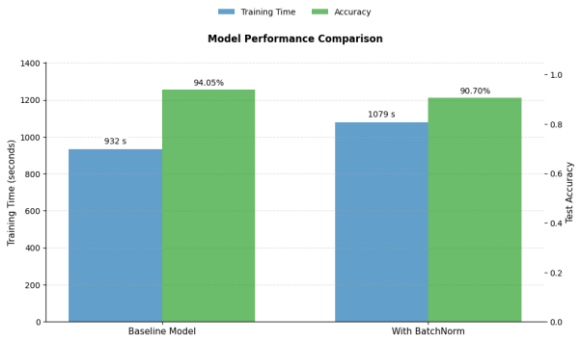
For all the experiments we created two models, one with Batch Normalization (BN) and the other without (Base).

3.1 Small dataset

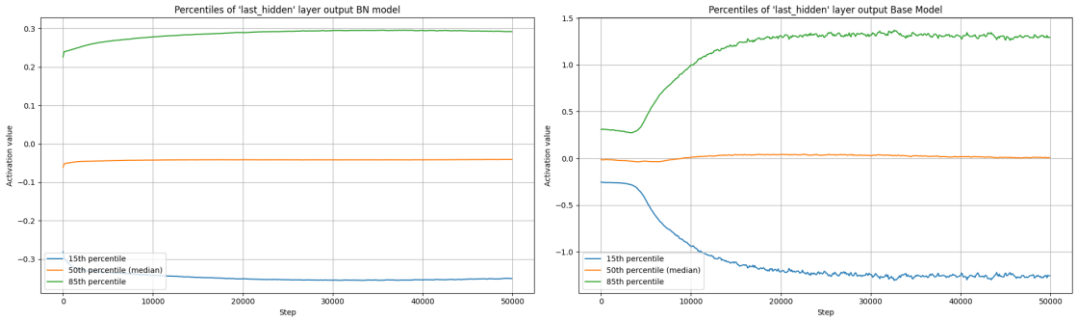
First, looking at the accuracy of the models during training, we can see how the BN model learned much faster, getting an accuracy of over 80% in the third epoch. While the slope for the model without BN is much slower.



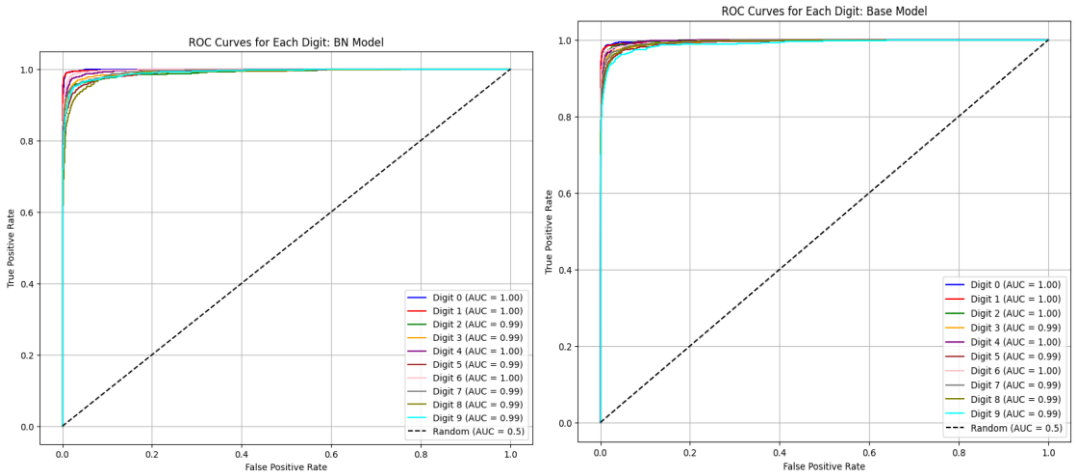
The final accuracy for the Base model was 94% and for the BN model 90.70%. We also tracked down the training time. Which was 932 seconds for the Base model and 1079 seconds for the BN model. In this case, the Base model had a better performance, but in more complex data or when we have limitations to train so many epochs BN models would be a better alternative.



In the paper it is also discussed the dispersion of the weights. As we can see in the image the dispersion for the BN model is more still, it does not have great changes along the training curve. While the base model gets dispersed.

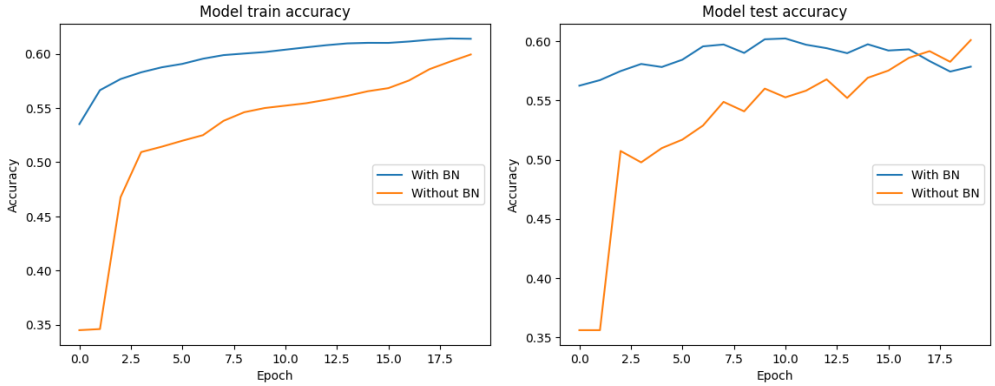


As a second measurement metric we look at the ROC curve for every classification. We found that in both models all the possible classifications are being predicted with the same quality.

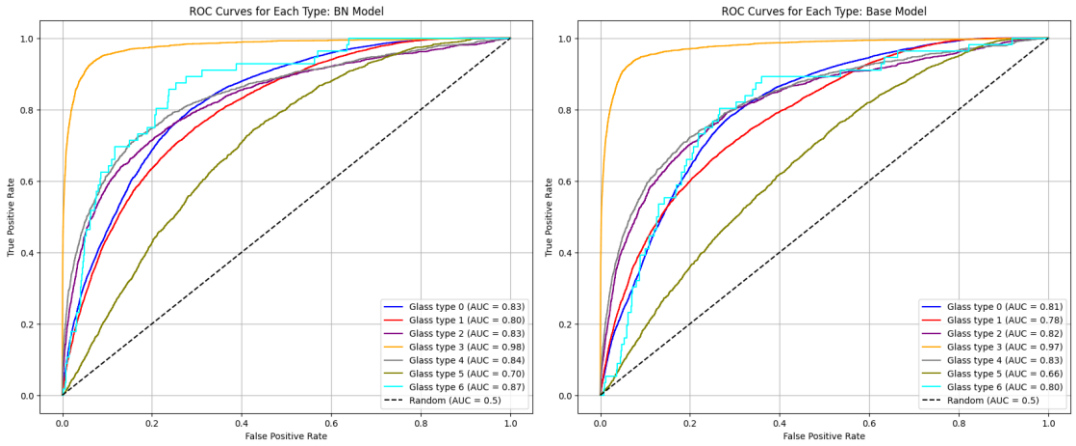


3.2 Large dataset

When applying our models to the larger dataset the only thing we changed was that we did not need to flatten the input, since the data is tabular. When looking at the accuracy, we see it behaves similarly to the small dataset. With the BN model having higher training and testing accuracy since the beginning and the base model slowly reaching it.



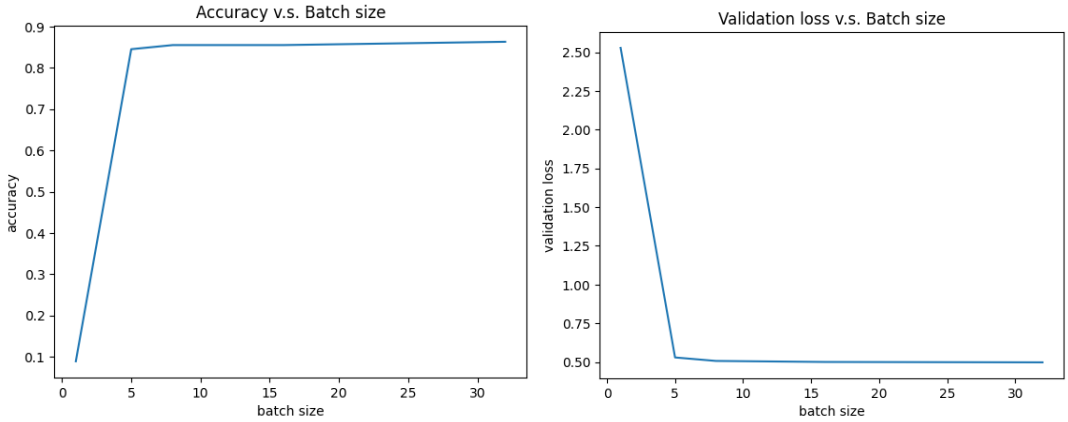
When looking at the ROC curves we can see that the BN model has better values. It is also notable that some categories are not easily predicted by any of the models, type 5 appears to be difficult to predict for both models.



4 Discussion

Although Batch Normalization did improve the performance of models, it has been challenged that the improvement is not because of reducing internal covariate shift. Some of the research shows that Batch Normalization improves the performance of models which have low or even no internal covariate shift. The reason why Batch Normalization works remains a mystery until now. Many believes that it is the mutual effect of Smoothing landscape, improving gradient flow and implementing Regularization that accomplished the success of Batch Normalization.

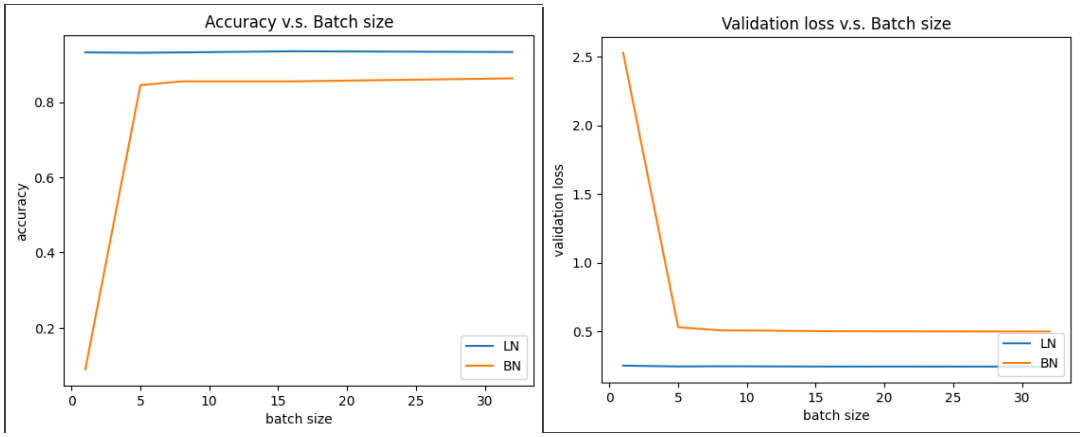
Besides the theory loophole, we have discovered that Batch Normalization is highly dependent on batch size. Theorically speaking, small batch size could lead to noisy batches and further harm the performance of Batch Normalization. We have plotted the accuracy and validation loss versus batch size using MINIST data. See graph below.



As we can see, the accuracy significantly decreases, and the validation loss dramatically increases when the batch size is small, which aligns our assumptions that small batch size could lead to getting noisy batches which could contains outliers thus harm the performance.

It is often that we do not have many data points and must choose a small batch size. Batch size dependency has restricted the application of Batch Normalization in this scenario. To address this problem, we have found two alternative ways to improve the performance with small batch size also keep the benefits of normalization.

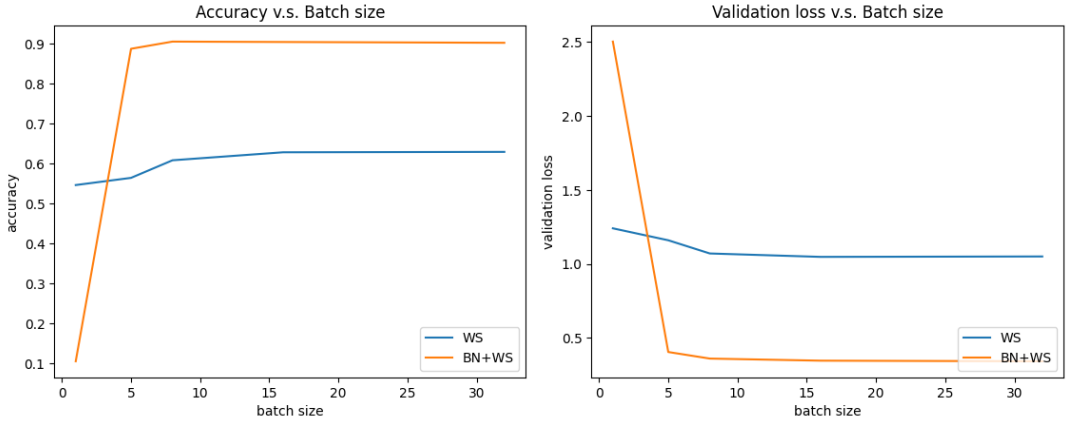
One of the methods is to change Batch Normalization to Layer Normalization. Different from Batch Normalization, Layer Normalization normalizes single data point's feature. The mean and the variance depend on all the features in one data entry.



The blue line represents Layer Normalization, and the orange line represents Batch Normalization. In this experiment, Layer Normalization has better performance compared to Batch Normalization especially when batch size is small.

The other method is combining Weight Standardization and Batch Normalization. Weight Standardization is an independent normalization technique, which means it could be used in combination with other normalization techniques. The concept of Weight Standardization is also simple which is basically normalizing the weights in each layer.

To confirm the effect of Weight Standardization, we also plotted the performance of model only with weight standardization. The graph is shown below:



The graph shows that the model with Weight Standardization performs worse than model with Batch Normalization. Both accurracy and loss curves are more stable with different batch sizes than with Batch Normalization, proving that Weight Standardization has the potential to fix batch size dependency in Batch Normalization.

However, in our experiment, combining these two did not fix the problem, we believe that it is due to the mutual effects of Batch Normalization and Weight Standardization with extremely small batch sizes causing more complicated situations for the model. One thing worth mentioning is that the model with Weight Standardization and Batch Normalization performed the best of all of our models.

5 Conclusion

Despite the original concept of Batch Normalization might not be true, there is no doubt that Batch normalization improves performance in most of the cases. Our work demonstrated that Batch Normalization improves performance compared to the original model. To improve the applicability of Batch Normalization, we proposed two methods to reduce batch size dependence which is Layer Normalization and combining Weight Standardization. Although eventually Weight Standardization did not successfully improve Batch Normalization with small batch size, the potential of Weight Standardization still cannot be ignored.

Future works:

1. Test different batch size focusing in the range from one to ten, observing the curve difference in the model with Weight Standardization and Batch Normalization
2. Try Layer Normalization with different datasets, especially with those who have correlated data points.

Reference

1. **Sergey Ioffe, Christian Szegedy (2015).** *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*

2. **Jimmy Lei Ba, Jamie Ryan Kiros, Geoffrey E. Hinton (2016).** *Layer Normalization*

3. **Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, Alan Yuille (2019).** *Micro-Batch Training with Batch-Channel Normalization and Weight Standardization*