NOISY BATCH NORMS AND HIGHER ORDER MOMENT NORMALIZATION

HENRY HERZOG

Contents

1. Goal of Project

The goal of this project is to improve the accuracy of batch normalization. The first method proposed to meet this goal involves requiring the batch normalization to learn noisy scale and shift parameters providing regularization in order to improve test accuracy. Then, I aim to extend this by building a method that normalizes higher order moments of the distribution so that each batch distribution is closer to being Gaussian which may enable easier training.

2. Description of Extension

In this section, we will give an in-depth description of **NoisySSNorm** and **YJNorm**, including the motivation for the ideas.

The **NoisySSNorm** was a response to the control experiment with the original batch norm from *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*(Ioffe et. al) struggling to attain test accuracy below 30 percent despite attaining training errors below 10 percent.[1] The **NoisySSNorm** differs from the batch normalization method proposed in 2 ways. First, two tensors containing Gaussian noise, denoted ϵ_1 and ϵ_2, are used to modify the learnable scale and shift parameters, γ and β. We also define a learnable parameter called the *noisescale* that will be the coefficient of ϵ_1, the noise added to gamma. This can be summarized in the following equation, where $\hat{X}$ is the batch data that has been standardized to 0 mean and unit variance channel-wise and η is the noise scale:

Equation 1: $Y = (\gamma + \eta\epsilon_1)\hat{X} + \beta\epsilon_2$

The second change from the original batch norm is that we initialize β to be non-zero. We do this by defining a scalar hyperparameter b_0 in the **NoisySSNorm** layer that initializes the parameter β so that the shift noise is initialized to be positive. The idea here is that if β is initialized to zero it could de-incentivise the model from learning any shift parameter which would eliminate the effect of the noise.
In the original batch normalization method and the method proposed above, only the first and second moments of each channel in the batch are standardized, and higher order moments are left untouched. This means that if the batch distribution is not Gaussian, it will still fail to be Gaussian after the standardization process because the higher order moments, skew and kurtosis,

Date: December 8, 2022.

can still be nonzero. In theory, having Gaussian batch distributions would allow...add research). Thus, we provide a further extension in a second method that uses the Yeo-Johnson Transform, a power transform that transforms a distribution into an approximately Gaussian distribution. The exact function applied to a distribution by the Yeo-Johnson Transform is dependent on a parameter λ that can be optimized for a given distribution. Also, the Yeo-Johnson Transform is invertible which allows data that has been transformed into a Gaussian distribution to be recovered.

We will now describe the three key functions that will be used in the **YJNorm**.[2] These functions are the Yeo Johnson Transform, the Inverse Yeo-Johnson Transform, and a function that optimizes lambda for the Yeo-Johnson transform for a given data-set.

The first function is the Yeo-Johnson Transform, which transforms a given data-set based on a parameter λ.[2] Let **yjtransform** be the function that takes in a tensor X of data and a scalar value λ, and applies the function ψ defined below to every $x \in X$ for the given λ.

$$\psi(x,\lambda) = \begin{cases} \log(1+x) & x \geq 0, \lambda = 0 \\ \frac{(1+x)^{\lambda}-1}{\lambda} & x \geq 0, \lambda \neq 0 \\ \frac{-(-x+1)^{2-\lambda}-1}{2-\lambda} & x < 0, \lambda \neq 2 \\ -\log(1-x) & x < 0, \lambda = 2 \end{cases}$$

The second function is the Inverse Yeo-Johnson Transform which gives the inverse of the function defined for the Yeo-Johnson Transform at any given λ.[2] Let **invyjtransform** be the function that takes in a tensor X of data and a scalar value λ, and applies the function ϕ defined below to every $x \in X$ for the given λ.

$$\phi(x,\lambda) = \begin{cases} e^{x} - 1 & x \geq 0, \lambda = 0 \\ (\lambda x + 1)^{\frac{1}{\lambda}} - 1 & x \geq 0, \lambda \neq 0 \\ 1 - (-(2-\lambda)x + 1)^{\frac{1}{2-\lambda}} & x < 0, \lambda \neq 2 \\ 1 - e^{-x} & x < 0, \lambda = 2 \end{cases}$$

The third function is the Yeo-Johnson optimizer. This function finds the λ that optimally maps the given data to a Gaussian distribution, using maximum likelihood estimation. In particular, we will use brent optimization.[3] Let **yjoptimize** be the function that gives the optimal λ for a given dataset x.

Now, we define the YJNorm algorithm. The algorithm differs from the original batch normalization in three key places. The first two differences occur during training and the third during test time. First, after the batch data is shifted and scaled so it has zero mean and unit variance over each channel, an optimal $\hat{\lambda}$ is calculated for each channel and that $\hat{\lambda}$ is used to apply a **yjtransform** to that channel. The second difference is there is a trainable parameter λ that dictates **invyjtransform** for each channel in the batch. This allows the layer to unlearn the original Yeo-Johnson transformation if needed. The third difference is that the optimal lambdas are stored in a moving average that is used as the λ parameter for the Yeo-Johnson Transform at test time. These changes are described in the algorithms below.

Algorithm 1: YJNorm algorithm, applied to activation x over a mini-batch.

Input: X, γ, β, λ, μ_{moving}, σ_{moving}, λ_{moving}
Output: The **YJNorm** of a batch X
During Training:

(1) Calculate the mean and variance over the channels for the batch of data
(2) Standardize the channels of X using the mean and variance calculated over the channels of the batch
(3) Calculate the optimal λ_{mle} for each channel of the batch using **yjoptimize**
(4) Apply the **yjtransform** using the corresponding λ_{mle} to each channel of X

(5) Update μ_{moving}, σ_{moving}, λ_{moving} with the calculate mean, variance, and λ_{mle}
(6) Apply the **invyjtransform** to each channel of the transformed X using λ
(7) Apply the learnable scale and shift transformations using β and γ

During Test:

(1) Standardize X over its channels using the μ_{moving} and σ_{moving} parameters.
(2) Transform each channel of X via the corresponding value in λ_{moving}

3. Experimental Results

In this section, we will discuss the empirical results for both the **NoisySSNorm** and the **YJBatchNorm**. These norms were tested by comparing their test accuracy on the CIFAR-10 data-set using CNN-based residual networks with both 22 and 56 layers. We will call the 22 layer residual network architecture **Resnet 22**, and the 56 layer residual network architecture **Resnet-56**. As a convention we will call a model by the batch norm variation used in it followed by the number of layers indicating which residual network structure was used. For instance, a 22 layer model with the **NoisySSNorm** will be called **NoisySSNorm 22**. In each experiment the models only differ in the variation of the batch normalization layer being used. The experimental set-up was strongly influenced by the vast computational restraints provided by google colab and the timeline for this project. As a result, the hyper-parameters were only tuned so that all models would be able to train, but not so that the training was optimal for each model individually.

For the experiments testing **NoisySSNorm**, the learning rate was set to 0.01, the momentum was set to 0.9, the batch size was set to 256, the SGD optimizer was used, and the models were trained for 25 epochs. The weights are initialized using He initialization. [He paper reference] For the **NoisySSNorm**, we also consider multiple variations of it that consist of only parts of the modifications made to the original batch normalization in **NoisySSNorm**. These variations are summarized in the table, along with **NoisySSNorm** and the control batch normalization layer found in the original batch normalization paper.[bnorm paper]

Fig 1: NoisySSNorm Variations	
Name of Variation	**Difference from NoisySSNorm**
Control	Original Batch Normalization Layer
NoisySSNorm	N/A
NoisyScaleNorm	Noise only added to Scale parameter γ and not to β
NoisyShiftNorm	Noise only added to shift parameter β and not to γ

(add figure label) The **NoisySSNorm** was also tested with different initializations of the β parameter and for the noise scale parameter η. Due to computational constraints, each experiment was only performed once and not averaged over multiple trials. Below is a summary of each model tested containing the test error after 25 epochs, and the lowest test error attained during training. The initialization value of every element in the β tensor is indicated in the β_0 column, and when applicable the initialization of every element in the noise scale tensor is in the η_0 column. The columns with (25) contain the test error after 25 epochs and the columns with (Min) contain the minimum error attained after any epoch up to and including the 25th epoch with the epoch in which it was attained in parentheses. This is to account for the fact that 25 epochs may over train some models. The best test error in each column is highlighted in bold. Due to computational constraints, the different initializations were only tested with the **Resnet 22** architecture

Fig 2: Experimental Results for NoisySSNorm						
Layer Name	β_0	η_0	**Resnet 22** (25)	**Resnet 56** (25)	**Resnet 22** (Min)	**Resnet 56** (Min)
Control	0	N/A	33.6	32.48	28.11 (11)	30.52 (6)
NoisySSNorm	0.2	0.2	27.39	23.24	25.78 (22)	23.24(25)
NoisySSNorm	0	0.2	25.75	N/A	24.01 (23)	N/A
NoisySSNorm	0.0	0.0	23.21	N/A	22.65(23)	N/A
NoisySSNorm	0.2	0.0	25.21	N/A	24.63 (22)	N/A
NoisyScaleNorm	N/A	0.2	35.4	36.23	35.4 (25)	36.23
NoisyShiftNorm	0.2	N/A	**22.70**	**22.83**	**22.34 (24)**	**22.83(25)**

For the experiments testing **YJNorm**, the learning rate was set to 0.01, the momentum was set to 0.9, the batch size was set to 256, the SGD optimizer was used, and the models were trained for

3 epochs. Due to computational constraints, we were only able to train a **YJNorm 6** model on 2000 training examples and 400 test examples. The weights were initialized using He initialization. The results obtained are listed below. The test error after 1 epoch and 3 epochs is listed

Fig 3: Experimental Results		
Layer Name	Test Error (1)	Test Error (3)
Control	**88.32**	**74.34**
YJNorm	90.22	89.88

4. Discussion

In this section, we will first discuss the experimental results concerning **NoisySSNorm** and then discuss the results concering **YJNorm**.

The results of our experiments suggest that using the **NoisySSNorm** instead of the original batch normalization layer in a residual network provides minor improvement in test accuracy in 22 layer residual networks and in 56 layer residual networks, for the hyperparameter settings described in the previous section. Specifically, for **NoisySSNorm 22** with both β_0 and η_0 initialized to have 0.0 in every entry, experiments found a minimum test error of 22.65 percent during 25 epochs of training, which was 5 percent lower than the minimum error obtained by the **Control 22** model. Similarly, for **NoisySSNorm 56** with both β_0 and η_0 initialized to have 0.2 in every entry, we found a minimum test error of 23.24 percent over 25 epochs, which was 9 percent lower than the minimum error obtained by **Control 56**. These results suggest that using the **NoisySSNorm** layer instead of the original batch normalization layer in residual networks may indeed have a regularizing effect that improves test performance, with this impact appearing to be stronger for deeper networks. However, this difference in accuracy could also be due to the learning rate used in the experiments potentially more compatible with the **NoisySSNorm**.

The experiments summarized in figure 2 indicate that the addition of the noise term to the shift parameter β is the most probable source of the difference in test error between models using **NoisySSNorm** versus the original batch normalization method. This is supported by the experiments in figure 2 that suggest changing the initializations of β and the noise scale η from 0.0 does not improve the test error. Specifically, this is supported by the experiment where th **NoisySSNorm22** model with β_0 and $\eta_0$initialized to be 0.0, obtains a slightly higher test accuracy then the **NoisySSNorm22** with nonzero initializations of β_0 and η_0. This provides evidence that the idea presented in section 2 that nonzero initialization of parameters would help improve the regularization effect in **NoisySSNorm** is likely false. In addition, the experiments with **NoisyScaleNorm** and **NoisyShiftNorm** models in figure 2 indicate that the improvement in accuracy for **NoisySSNorm** models in the first 25 epochs is likely due to the presence of noise in the shift term. This is because both the **NoisyScaleNorm 22** model and **NoisyScaleNorm 56** model exhibited an increase in test error when compared with the control while the **NoisyShiftNorm 22** model and the **NoisyShiftNorm 56** model exhibited a decrease in test error when compared with the control. Moreover, the **NoisyShiftNorm 22** model and the **Noisy Shift Norm** produced lower test errors then the other variations of **NoisySSNorm** outlined in figure 1. This further supports the claim that the regularization effect in **NoisySSNorm** is likely caused by the factor of Gaussian noise ϵ_2 added to the beta parameter as seen in **Equation 1**. While residual networks using **NoisySSNorm** in place of the original batch normalization layer, show promise for reducing test error the computational constraints of this project add uncertainty to the results in figure 2 because each experiment was only able to be ran once. Thus, further experiments with a larger computational budget are needed to validate the effectiveness of using **NoisySSNorm** layers in residual networks.

The experiments using the **YJNorm** layer instead of the original batch normalization layer suggest that the **YJNorm** layer may actually harm training. The experiments indicated that over 3 epochs the **YJNorm 8** model was unable to improve its test error beyond 2 percent while the **Control 8** model was able to improve its test error by around 14 percent. One potential explanation for this is that the performing the Yeo-Johnson transformation on the channels of a batch of data distort the information in that data excessively, and the inverse Yeo-Johnson transformation is unable to learn how to effectively recover that information. However, these

results are only based on a data set with 1000 training examples and 400 test examples and three epochs of training warranting further exploration with larger computational resources. The reason that running models with the **YJNorm** layer is so computationally expensive is because for every batch of data the layer has to loop over all the channels and compute the **yjtransform**, the **invyjtransform**, and the **yjoptimize** function on each channel. In particular, the **yjoptimize** function includes solving a maximum likelihood estimation problem for every channel in every batch of data which greatly increases the computational cost. An attempt was made to vectorize these transformations to lower the computational costs but this effort was unsuccessful due to the fact that the vmap operator in the functorch library is not compatible with functions that have piecewise definitions, which includes the **yjtransform**, the **invyjtransform**, and the **yjoptimize** function. Thus, we were unable to obtain results beyond a small model on a limited data set for the **YJNorm** and are left with only an initial indication that the use of a **YJNorm** layer makes it more difficult to train networks.

5. Conclusion

Overall, the experiments provide evidence that using the **NoisySSNorm** in a residual network lowers the test error on the CIFAR-10 when compared with the original batch normalization layer. These results hold for both 22 and 56 layer networks but are limited to the hyper-parameter settings described in section 3. Also, ablation experiments indicate that improvements in test error caused by the use of a **NoisySSNorm** layer are most likely due to the noise factor in the shift term rather than the different initialization or the noise factor in the scale term. Lastly, we obtained evidences that indicates the **YJNorm** layer harms the ability of a residual network to train succesfully. However, the results for the **YJNorm** were only obtained on a small subset of the dataset with a greatly reduced model size and no hyperparameter tuning due to computational constraints.

6. Acknowledgements

I would like to thank Hongyuan Mei and David Yunis for providing insight and support as I completed this project. They were incredibly helpful figuring out how to adapt my project when running into computational constraints.

References

[1] Sergey Ioffe, Christian Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015

[2] Yeo, In-Kwon and Johnson, Richard (2000). A new family of power transformations to improve normality or symmetry. Biometrika, 87, 954-959.

[3] Brent, Richard , "Chapter 4: An Algorithm with Guaranteed Convergence for Finding a Zero of a Function", Algorithms for Minimization without Derivatives, Englewood Cliffs, NJ: Prentice-Hall, 1973