ILLINOIS INSTITUTE OF TECHNOLOGY

Weight standardization analysis

Thomas Ehling A20432671

& Clement Rouault A20432709

CS 577: Deep Learning, IIT Chicago
April 2019

1 Introduction

Deep neural networks have significantly outperform other solution in multiples domains, such as speech, language, vision, games, … One of the reason why deep learning has made such outstanding progress in the recent years is Batch Normalization (BN). BN enables the use of higher learning rates, greatly accelerating the learning process and allowing the use of sigmoid activation functions, previously impossible to train due to the vanishing gradient.

Though batch normalization is now a standard feature of any deep learning framework and can be used off the shelf, using it naively can lead to difficulties in practice. The current recent alternatives to BN are Layer Normalisation (LN), Weight Normalization (WN), Group Normalization (GN), and the latest one : Weight Standardization (WS).

In this paper we briefly explain the bases necessary to understand the project, especially the BN and GN methods. Then, we explain the theory and process behind WS, to finally implement and test it on our own machine learning model.

2 Previous Work

The previous work on the subject are on the regularization techniques related to BN :

- **BN :** Sergey Ioffe, Christian Szegedy (2015) “Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shift” : https://arxiv.org/pdf/1502.03167.pdf

- **WN :** Tim Salimans, Diederik P. Kingma (2016) “Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks” : https://arxiv.org/pdf/1602.07868.pdf

- **LN:** Jimmy Lei Ba, Jamie Ryan Kiros, Jamie Ryan Kiros (2016) “Layer Normalization” : https://arxiv.org/pdf/1607.06450.pdf

- **GN:** Yuxin Wu, Kaiming He (2018) “Group Normalization” : https://arxiv.org/pdf/1803.08494.pdf

3 A closer look at the Problem

We aim to understand the WS method, and analyze the performance on real-life applications. The main problem of BN is the low performance of the method on micro-batches, while other solutions does not perform as good for large batches. The searchers claims that when WS is used with GN, it outperform BN everywhere.

The theory part will rely on the research papers for BN, GN and WS.

For the experiments, we tested GN+WS vs BN for two different machine learning model :

1. Object detection model on the COCO dataset
2. Classifier on the MNIST Fashion dataset.

4 Theory

4.1 Reminder : BN equations

From the original paper :

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

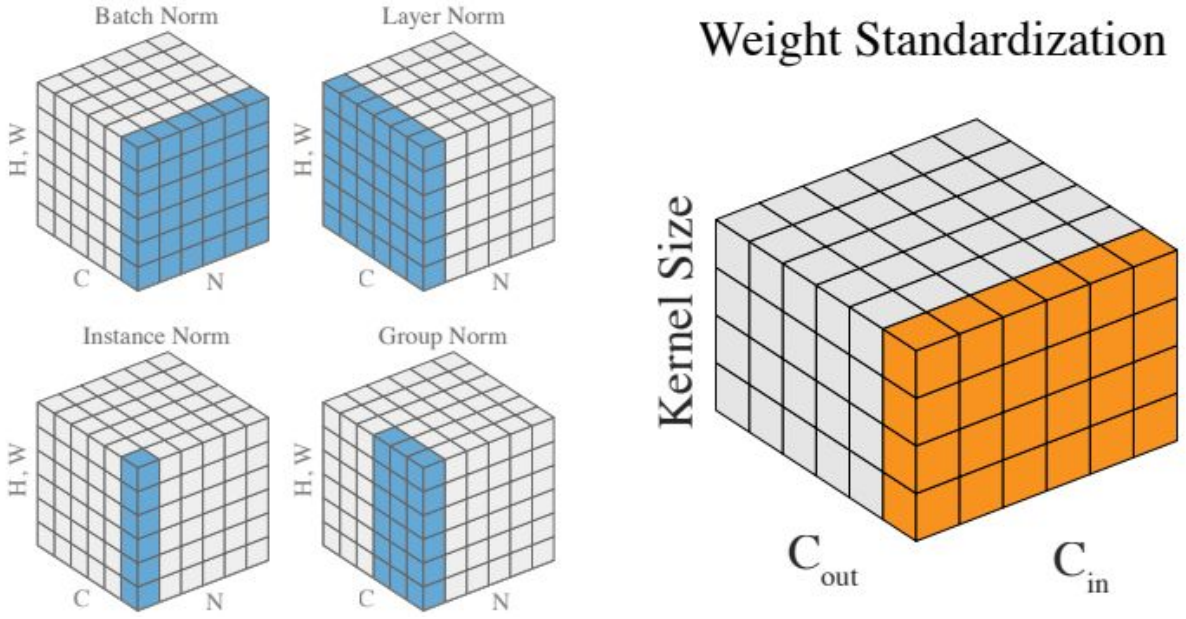
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

4.2 Weight Standardization



The figure above illustrate the dimensions impacted by the different regularizations techniques. We can see that the WS technic concern the same dimension as BN, only it affects the weights where BN normalize the batches.

This WS method changes the weights used for the forward path : the new weights used are standardized : they are reduced by the mean, then divided by the standard deviation of the initial weights. It also changes the gradient during backpropagation.

Similar to BN, WS controls the first and second moments of the weights of each output channel individually in convolutional layer.

Important : The searchers assume that the WS is used alongside normalization layers such as GN or BN.

Consider a standard convolutional layer with its bias term set to 0 :

$$\boldsymbol{y} = \hat{\boldsymbol{W}} * \boldsymbol{x}$$

In Weight Standardization, instead of directly optimizing the loss L on the original weights ŵ, the weights ŵ are parameterized as a function of W, i.e., ŵ = WS(W), and optimize the loss L on W by SGD:

$$\hat{\boldsymbol{W}} = \left[\hat{\boldsymbol{W}}_{i,j} \mid \hat{\boldsymbol{W}}_{i,j} = \frac{\boldsymbol{W}_{i,j} - \mu_{\boldsymbol{W}_{i,\cdot}}}{\sigma_{\boldsymbol{W}_{i,\cdot}+\epsilon}}\right] \tag{2}$$

$$\boldsymbol{y} = \hat{\boldsymbol{W}} * \boldsymbol{x} \tag{3}$$

where

$$\mu_{\boldsymbol{W}_{i,\cdot}} = \frac{1}{I}\sum_{j=1}^{I}\boldsymbol{W}_{i,j},\ \ \sigma_{\boldsymbol{W}_{i,\cdot}} = \sqrt{\frac{1}{I}\sum_{i=1}^{I}(\boldsymbol{W}_{i,j} - \mu_{\boldsymbol{W}_{i,\cdot}})^2} \tag{4}$$

WS has two main effects :

- **WS normalizes gradients**
- **WS smooth the landscape** by affecting the Lipschitz constant of the Loss.

We are not explaining the details of these processes, these are not a necessity for our project. Please refer to the Research paper in the “Reference” section for more informations.

5 Organization and responsibilities

What was our responsibilities ?

- We **both** work on the **theory** behind Weight Standardization.
- **Thomas** experiment with a **classifier** on the MNIST Fashion dataset
- **Clément** experiment with a **Object detection model** on the COCO dataset

If you have any questions about the implementation part (or the rest of the report), please feel free to reach out to any of us to the following email :

- crouault@hawk.iit.edu
- tehling@hawk.iit.edu

Or ask for an appointment, so we can show you a live demonstration.

6 Implementation

This part is different for both of us, as we worked on different model.

All along the process I used a Colab Notebook with my drive mounted locally. This is the .ipynb file.
I converted the Colab notebook into a python file, who will run for a batch size of 128 and 10 epochs.

Dataset included in keras so there is no need to add a path for dataset

In the Data/ folder, I included all the weights of al mytrainings, alongside the histories, in case you wanted to test any of the following results.

7 Dataset

This part is different for both of us, as we worked on different model.

I wanted to build a classifier on a well-known data-set to easily compare my results with other research and sample codes. At the same time, I wanted a multi-class classifier to increase the difficulty and increase the need to use a regularization technique. The most used data set for multi-class classification is the MNIST data-set but it is too easy to reach a very high accuracy, and it has become overused. So I finally opted for the MNIST-Fashion dataset.

The dataset is composed of cloth images, split in 10 categories :

1. T-shirt/top
2. Trouser
3. Pullover
4. Dress
5. Coat
6. Sandal
7. Shirt
8. Sneaker
9. Bag
10. Ankle boot

I will use : 50_000 / 10_000 / 10_000 for the training / validation / testing datasets.

8 Model Architecture

This part is different for both of us, as we worked on different model.

8.1 Designing the Group Normalisation layer

There is no official implementation of keras for the Group Normalisation technique yet.
I found a project where someone designed a custom layer in Keras to handle GN :

https://github.com/titu1994/Keras-Group-Normalization/blob/master/group_norm.py

I took the Class corresponding to the keras layer and implemented in my classifier. The layer can then be added to the main architecture structure by calling :

```
x = GroupNormalization(groups=2, axis=-1, epsilon=0.1)(input)
```

8.2 Designing the Weight Standardization Technique

Unfortunately, all the sample codes available on the Github of the WS research paper are using PyTorch. However they do provide some lines for the implementation on Tensorflow :

```
kernel_mean = tf.math.reduce_mean(kernel, axis=[0, 1, 2], keepdims=True,
name='kernel_mean')
kernel = kernel - kernel_mean
kernel_std = tf.keras.backend.std(kernel, axis=[0, 1, 2], keepdims=True)
kernel = kernel / (kernel_std + 1e-5)
```

WS has just been released, so there are absolutely no sample codes using it available online. So I looked into what are the actual methods to implement Weight Normalisation in order to adapt it. But here again, WN is a fairly new method are there are only few examples. I decided to implement it in a custom regularizer function that I call inside my Convolutional layers.

Example custom regularizers function from the Keras documentation :

```
def l1_reg(weight_matrix):
    return 0.01 * K.sum(K.abs(weight_matrix))
```

My custom WS function on the same pattern :

```
def ws_reg(kernel):
  kernel_mean = tf.math.reduce_mean(kernel, axis=[0, 1, 2],
keepdims=True, name='kernel_mean')
  kernel = kernel - kernel_mean
  kernel_std = tf.keras.backend.std(kernel, axis=[0, 1, 2],
keepdims=True)
  kernel = kernel / (kernel_std + 1e-5)
```

And I am available to call it this way :

```
model.add(Conv2D(32, (3, 3), kernel_regularizer=ws_reg))
```

8.2 Final model Architecture :

There is a sample code for MNIST-Fashion code classification available by Google online. I took the same model architecture, change the BN to GN and added the WS to each Convolutional layers.

The final model architecture is :

```
model_gn = Sequential()
model_gn.add(Conv2D(32, (3, 3), padding="same",
                    input_shape=(28, 28, 1), kernel_regularizer=ws_reg))
model_gn.add(Activation("relu"))
model_gn.add(GroupNormalization(axis=chanDim))
model_gn.add(Conv2D(32, (3, 3), padding="same", kernel_regularizer=ws_reg))
model_gn.add(Activation("relu"))
model_gn.add(GroupNormalization(axis=chanDim))
model_gn.add(MaxPooling2D(pool_size=(2, 2)))
model_gn.add(Dropout(0.25))

# second CONV => RELU => CONV => RELU => POOL layer set
model_gn.add(Conv2D(64, (3, 3), padding="same", kernel_regularizer=ws_reg))
model_gn.add(Activation("relu"))
model_gn.add(GroupNormalization(axis=chanDim))
model_gn.add(Conv2D(64, (3, 3), padding="same", kernel_regularizer=ws_reg))
model_gn.add(Activation("relu"))
model_gn.add(GroupNormalization(axis=chanDim))
model_gn.add(MaxPooling2D(pool_size=(2, 2)))
model_gn.add(Dropout(0.25))

# first (and only) set of FC => RELU layers
model_gn.add(Flatten())
model_gn.add(Dense(512))
model_gn.add(Activation("relu"))
model_gn.add(GroupNormalization())
model_gn.add(Dropout(0.5))

# softmax classifier
model_gn.add(Dense(10))
model_gn.add(Activation("softmax"))
```

9 Evaluation

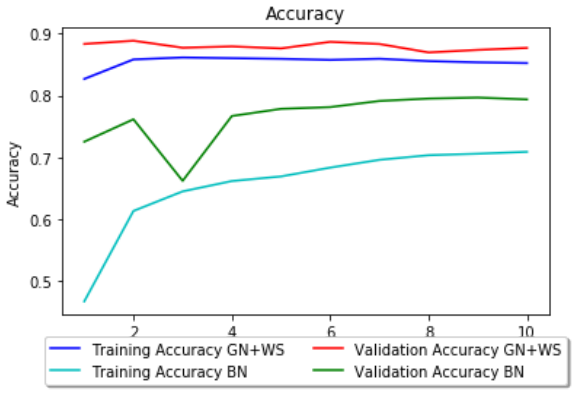
This part is different for both of us, as we worked on different model.

All the experiments have been run for 10 epochs. (This is assumed to be enough as we can see the curves overfit).

The computation time has been added to the table because I noticed a net difference between both models. The exact time is of course relative to the hardware used, and it is here only to give an idea of the ratio.

9.1 For micro-batches (batch size = 2)

Normalization	Batch size	Computation time	Accuracy
BN	2	~ `255s / epoch`	`18.76%`
GN+WS	2	~ `275s / epoch`	`91.86`%



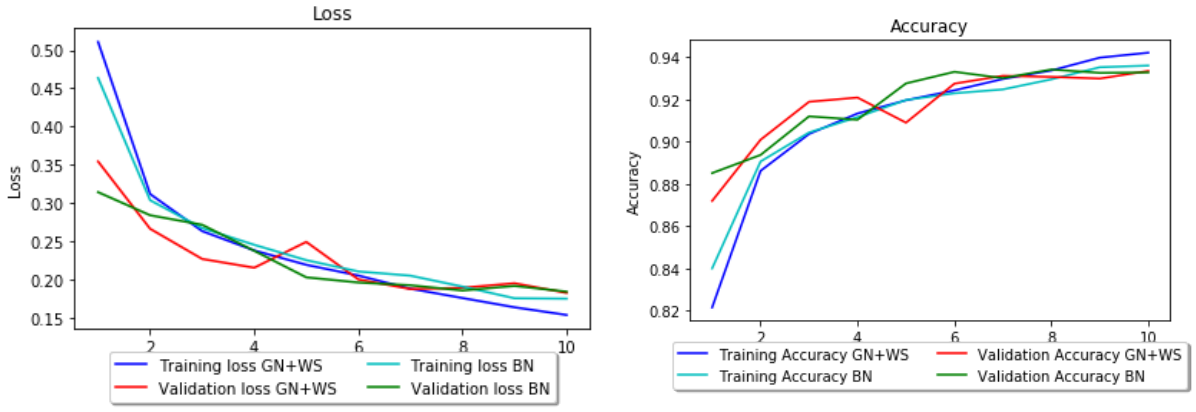
The result found correlate with the paper perfectly !

We can see that the model with GN+WS clearly outperform the one with BN.

This is really good, as it offers the possibility to use regularization techniques with micro-batches.

9.2 For a standard mini-batch size (batch size = 64)

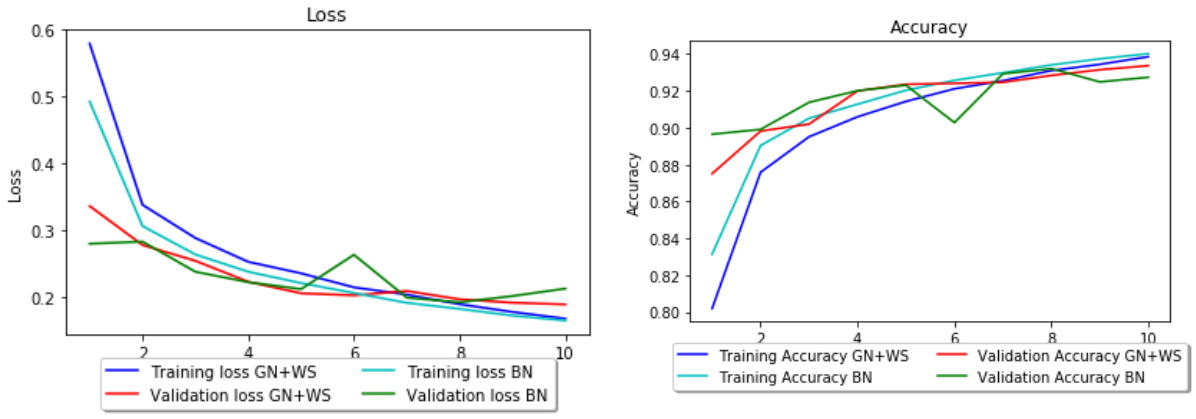
Normalization	Batch size	Computation time	Accuracy
BN	32	~ `10s / epoch`	`92.49`%
GN+WS	32	~ `20s / epoch`	`92.82`%



These results correlate with the paper. The two methods have approximately the same accuracy for regular batch size.

9.3 For large batches (batch size = 128)

Normalization	Batch size	Computation time	Accuracy
BN	128	~ `7s / epoch`	92.93%
GN+WS	128	~ `12s / epoch`	92.9%



These results correlate with the paper. The two methods have approximately the same accuracy for large batch size.

This is really impressive ! Until now, no regularization technique managed to perform well on micro-batches and perform as well as BN for large batches.

10 Conclusion

At first we thought the papers results were too impressive, and we wanted to test if it would perform as well on our own deep learning models. It turns out that our results matches perfectly the ones from the paper. The Weight Standardization coupled with the Group Normalization technique does outperform Batch Normalization.

This is great progress because BN was a breakthrough but it leads searcher to only use large batches, which are memory intensive. This method outperforms BN for micro, standard and large batches, and leads to new opportunities.

We can also notice that all the latest regularization techniques have been developed in the last few years, so we need to watch out for future updates.

References

The original research paper :

- Siyuan Qiao Huiyu Wang Chenxi Liu Wei Shen Alan Yuille : *"Weight Standardization"* (2019)
- https://arxiv.org/pdf/1903.10520v1.pdf

Other references :

- https://github.com/joe-siyuan-qiao/WeightStandardization
- http://mlexplained.com/2018/01/10/an-intuitive-explanation-of-why-batch-normalization-really-works-normalization-in-deep-learning-part-1/
- http://mlexplained.com/2018/01/13/weight-normalization-and-layer-normalization-explained-normalization-in-deep-learning-part-2/
- https://en.wikipedia.org/wiki/Batch_normalization
- https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c
- https://www.analyticsvidhya.com/blog/2018/03/comprehensive-collection-deep-learning-datasets
- https://github.com/zalandoresearch/fashion-mnist
- https://github.com/titu1994/Keras-Group-Normalization