WRN-GN: Wide ResNets With Group Normalization

Akhil Chilumuru
Department of Computer Science
Texas A&M University
College Station, TX 77843
akhilchilumuru@tamu.edu

Abstract

Batch Normalization (*BatchNorm*) is the standard normalization technique used in today's state-of-the-art residual network variants. Although *BatchNorm* works fairly well to stabilize training, it is most effective when the batch size is large. This limits the use of *BatchNorm* to high V-Ram GPUs. In this work, we explore an alternative to *BatchNorm* called Group Normalization (*GroupNorm*) coupled with Weight Standardization (*WeightStand*). We show that using *GroupNorm* with *WeightStand* on Wide ResNets with Pre-activation Residual blocks improves it's performance significantly.

1 Introduction

Residual Networks (*Resnets*) had a large success in image classification. The early *Resnets* introduced in [1] showed that optimizing networks as deep as 1000 layers is possible with residual connections. This formed a standing ground for other variants of *Resnets* . At this point, *BatchNorm* and subsequent activation were always applied after convolution operations in *Resnets* . The order of activations in residual blocks were first explored in [2], showing that pre-activation provides an improvement in accuracy over prevalent post-activation residual blocks. Later, [3] argues that widening the residual blocks provides a more effective way of improving the performance of residual networks rather than increasing the depth. It is also noted by [4,5] that *BatchNorm* error increases rapidly when the batch size becomes smaller due to inaccurate batch statistics estimation. Motivated by these observations, this work is based on [3,4,5] and tries to achieve good performance on the CIFAR-10 dataset.

2 Related Work

Wide residual networks There are three ways to increase representational power of a residual block: (1) Adding more convolutional layers per block (2) Widening the convolutional layers by adding more feature maps (3) Increasing the kernel sizes in convolutional layers. [3] finds that using a basic residual block with 3 X 3 kernel size gives the best performance. They also note that the computational complexity increases linearly with the network depth and quadratically with the widening factor. They find that the optimal performance is achieved by using 4 convolutions per block with a widening factor of 10. For this reason, we are going to use this network architecture as our backbone.

Group Normalization Normalization at the hidden activation layers helps the network to train faster by preventing the exploding gradients and helps the gradient descent algorithm to converge quicker. Batch Normalization does it along the batch dimension. An important thing to note is that at inference time *BatchNorm* uses the pre-computed mean and variance from the training set [6]. This means that there is no normalization performed during test time. Also, the pre-computed statistics may not represent the test data if the distribution greatly differs. In addition, the batch size has a huge impact on the accuracy of the estimated statistics.

Although there are several other normalization methods like Layer Normalization, and Instance Normalization, they have not matched *BatchNorm* accuracy. Instead of using batch statistics information, [4] claims to rather avoid such computations. *GroupNorm* is a middle ground between Instance Normalization and Layer Normalization. As shown in Figure 1, it organizes the channels into different groups and computes mean and variance along the spatial dimensions and a group of channels.

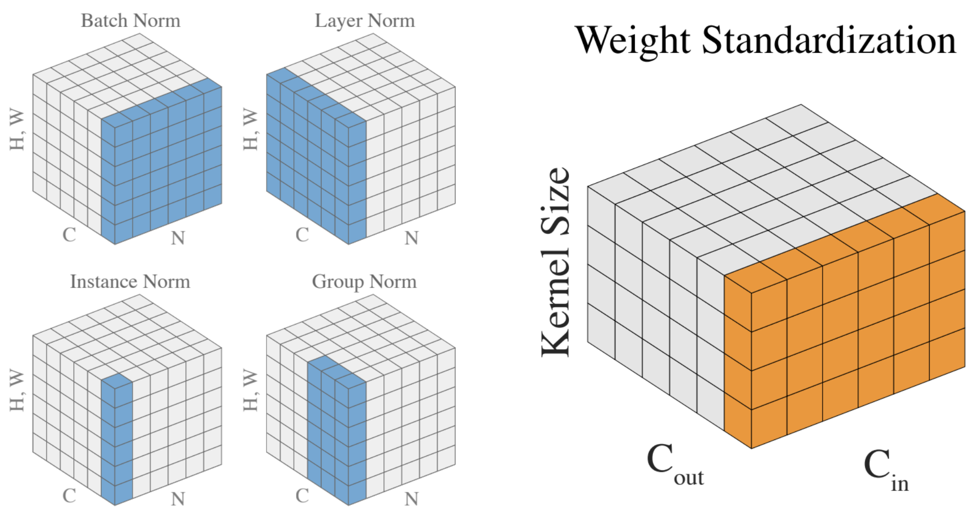

Figure 1: Left: Comparing normalization methods on activations (blue) Right: Weight Standardization in convolutional layer (orange)

Weight Standardization The reason why normalization helps in optimizing neural networks is due to smoothing of loss landscape and thus accelerating the convergence during training [7]. Although *BatchNorm* satisfies this property of normalization, it has other drawbacks aforementioned such as batch size. While *GroupNorm* could overcome this problem, in practice it alone could not outperform *BatchNorm* at large batch sizes. Thus a new technique for normalizing weights called Weight Standardization was introduced in [5]. Unlike *BatchNorm* , *WeightStand* is applied on convolutional weights. It essentially normalizes the gradients during back-propagation. [8] claims that *GroupNorm* combined with *WeightStand* outperforms *BatchNorm* at almost any batch size. [5] also shows that *WeightStand* inherently smoothens the loss landscape and thus helps in faster convergence as well.

3 Implementation

3.1 Standardized Convolutions

Weight Standardized convolutions can be implemented in **PyTorch 1.13.0** as shown below:

```
class StdConv2d(nn.Conv2d):
  def forward(self, x):
    w = self.weight
    v, m = torch.var_mean(w, dim=[1, 2, 3], keepdim=True, unbiased=False)
    w = (w - m) / torch.sqrt(v + 1e-10)
    return Conv2d(x, w, self.bias, self.stride, self.padding)
```

We use pre-activation residual networks with basic residual block as our backbone architecture. Every convolution in residual block i.e., both 3X3 convolutions and 1X1 shortcut projections are replaced with their weight standardized versions. We perform down-sampling by using convolutions with a stride of 2.

3.2 Group Norm Layer

The *BatchNorm* layer in every residual block is replaced with *GroupNorm* layer with 32 groups in a single line of code using **PyTorch 1.13.0** as follows:

```
self.gn = nn.GroupNorm(num_groups,in_filters)
```

3.2.1 Wide Residual Network

We use the best architecture that [3] claims to work on CIFAR-10 dataset i.e, a 28 layer deep residual network with a widening factor of 10. The following table shows the network architecture in brief:

Table 1: Network architecture

Size	Layer
$32\text{X}32$	$[3\text{X}3, 16]$
$32\text{X}32$	$\begin{bmatrix}3\text{X}3, 160\\3\text{X}3, 160\end{bmatrix}\text{X}4$
$16\text{X}16$	$\begin{bmatrix}3\text{X}3, 320\\3\text{X}3, 320\end{bmatrix}\text{X}4$
$8\text{X}8$	$\begin{bmatrix}3\text{X}3, 640\\3\text{X}3, 640\end{bmatrix}\text{X}4$

The network ends with an average pooling and 640-way fully connected layer.

3.3 Data augmentation

We perform the same augmentation that [1] uses: during training, 4 pixels are padded on each side and a random $32\text{X}32$ crop is sampled from an image or its horizontal flip with the per-pixel mean subtracted. For testing, we only evaluate the single view of the original $32\text{X}32$ image.

3.4 Hyper-parameter details

Our model uses a vanilla ResNet architecture with pre-activation residual blocks widened by a factor of 10, except that we replace all Batch Normalization layers with Group normalization and use weight standardized convolutional layers. We train our models using SGD with momentum of 0.9 and we use a dropout rate of 0.3. The learning rate starts from 0.1 and is divided by 5 at 60, 90, 120 and 200 epochs. We use a 45k/5k train/validation split. The model is trained with a batch size of 128 on an RTX 3070 GPU with 8GB Vram for a total of 250 epochs and it took 9 hrs to train.

4 Experiments

We experiment on the CIFAR-10 dataset with various ResNet models: ResNet-164 with bottleneck blocks and Batch normalization, Resnet-164 with bottleneck blocks, batch normalization and pre-activation, Wide ResNet with Batch normalization, and Wide ResNet with Group Normalization and Weight Standardization. Figure 2 shows the training loss plotted against the epochs and Table 4 shows the results in terms of test accuracy on the public test set.

Table 2: Results using various models

Model	top-1 test accuracy (%)
ResNet-164 Bottleneck	91.05
PreAct Reset-164 Bottleneck	92.24
PreAct WRN-28-10 BatchNorm	91.13
PreAct WRN-28-10 GroupNorm + WeightStand	**94.75**

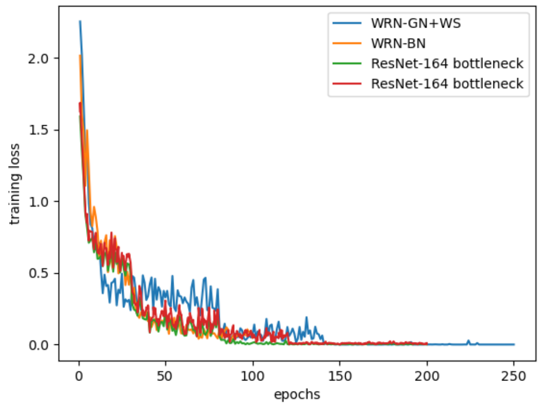

Figure 2: Training loss vs epochs

5 Conclusion

We present experiments performed with different variants of residual networks on the CIFAR-10 dataset. Based on the results, we propose that Wide residual networks with Group Normalization and Weight Standardization achieves a decent accuracy of 94.75%. There is still room for improvement because the current benchmark on the CIFAR-10 dataset is 99% top-1 accuracy. Also, this work touches just one aspect of exploring deep neural networks i.e, architecture. We think that perhaps other techniques such as data augmentation, training methods could help reach the benchmark score.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep residual learning for image recognition.* CoRR, abs/1512.03385, 2015.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Identity mappings in deep residual networks.* CoRR, abs/1603.05027, 2016.

[3] Sergey Zagoruyko and Nikos Komodakis. *Wide residual networks.* In BMVC, 2016.

[4] Y. Wu and K. He. Group normalization. *In European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.

[5] Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. *Micro-batch training with batch-channel normalization and weight standardization.* arXiv preprint arXiv:1903.10520, 2019.

[6] S. Ioffe and C. Szegedy. *Batch normalization: Accelerating deep network training by reducing internal covariate shift.* In ICML, 2015.

[7] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, *"How does batch normalization help optimization?"* in Advances in Neural Information Processing Systems (NeurIPS), 2018, pp. 2488–2498.

[8] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby. *Big Transfer (BiT): General Visual Representation Learning* (2019), arXiv preprint.