How Does Batch Normalization Help Optimization?

Group 03 : S Aniruddha & R Rohith

Introduction

1. Batch Normalization [1] is a popular technique to speed up the training of neural networks.
2. Traditionally, the reason attributed to BatchNorm's success is that it reduces Internal Covariate Shift.
3. In this poster, we encapsulate the work done by Santurkar et al. [2], where they view the performance gain of BatchNorm through the lens of optimization instead of reduction of internal covariate shift.
4. Specifically, we explore the more fundamental effect of BatchNorm: its ability to smoothen both the loss function and the gradient of the loss function, which leads to faster convergence during training.

Stochastic Gradient Descent

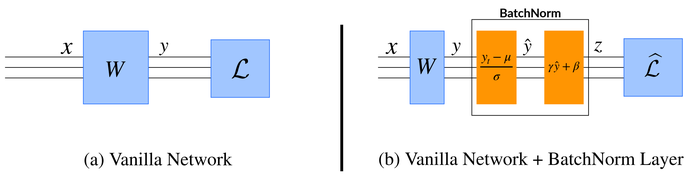
1. Consider a neural network with parameters Θ and training dataset $x_1, ..., x_N$. We need to minimize the loss function l to find the optimal parameters:

$$\Theta = \arg\min_{\Theta}[\frac{1}{N}\sum_{i=1}^{N} l(x_i, \Theta)] \tag{1}$$

2. Evaluating the gradient of this objective function is very expensive (sum of the results of many costly gradient computations).
3. Stochastic Gradient Descent is a variant of the steepest descent method used to mitigate this problem.
4. Instead of using all N points to compute the gradient, SGD statistically estimates the gradient by using a minibatch of size M. Mathematically, the estimate of the gradient is given by:

$$\sum_{i=1}^{M}[\frac{1}{M}\frac{\partial l(x_i, \Theta)}{\partial \Theta}] \tag{2}$$

BatchNorm and Internal Covariate Shift



(a) Vanilla Network (b) Vanilla Network + BatchNorm Layer

Figure 1. BatchNorm layer between 2 hidden layers

1. **Batch Norm** is a network layer that is inserted between 2 hidden layers. Its role is to normalize the outputs of the first hidden layer before passing them on to the next hidden layer.
2. During training, BatchNorm processes one minibatch at each timestep. In figure 2 below, β and γ are learnable parameters, whereas μ and σ are calculated from the minibatch activations.
3. BatchNorm ensures that its outputs for different minibatches have the same mean β and standard deviation γ.
4. During inference, we process one data point at a time. Therefore, BatchNorm uses its stored values μ_{mov} & σ_{mov} (moving average calculated during training) instead of μ & σ.
5. It is empirically well-established that Neural Networks trained with BatchNorm achieve faster convergence.

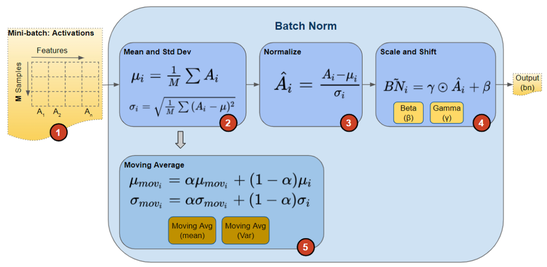


Figure 2. Parameters of BatchNorm

6. **Internal Covariate Shift (ICS)** is defined as the change in the distribution of network layer activations caused due to the change in network parameters during training. This results in a change in the distribution of layer inputs of subsequent layer.
7. The authors of [1] hypothesized that such continual distribution changes to inputs make it harder to train since the layers have to adapt continuously to the new distribution.
8. Hence, they present BatchNorm as a technique to reduce Internal Covariate Shift.

Is Internal Covariate Shift the reason for BatchNorm's success?

1. Consider a standard VGG network trained on CIFAR-10 with and without BatchNorm. We can see that there is a consistent gain in training speed when BatchNorm is used in figure 3.
2. Now, consider a network trained with *random* noise injected *after* BatchNorm layers. Specifically, each activation for each sample in the minibatch is perturbed using i.i.d. noise sampled from a *non-zero* mean and *non-unit* variance distribution. Also, the noise distribution is *changed* at each time step.
3. Even though such noise injection deliberately produces a severe covariate shift, this network performs much better than the standard network and is almost similar to the one with BatchNorm.

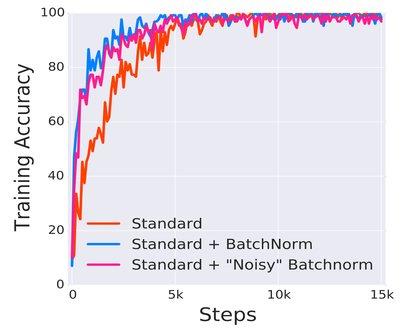


Figure 3. Training accuracy vs. number of steps

Therefore, we can conclude that reducing Internal Covariate Shift does not explain BatchNorm's efficacy.

Smoothening Effect of BatchNorm

Why does BatchNorm work then?

1. The authors of [2] hypothesize that BatchNorm re-parametrizes the underlying optimization problem to make its landscape significantly more smooth.
2. This smoothening effect is manifested in two ways: (i) the loss function becomes more Lipschitz, and (ii) the *gradients* of the loss become more Lipschitz too.
3. Definitions:
   1. f is L-Lipschitz if $|f(x_1) - f(x_2)| \leq L\|x_1 - x_2\|$, for all x_1 and x_2.
   2. f is β-smooth if its gradient is β-Lipschitz.
4. Since we cannot visualize the function directly, we look at the following three plots to get an idea of the smoothness of the loss function and its gradients:
   - **Loss Landscape**: To demonstrate the impact of BatchNorm on the Lipschitzness of the loss, the variation of the loss as we take a small step along the direction of the gradient at a point at each training step is plotted in figure 4.
   - **Gradient Predictiveness**: To observe the effect of BatchNorm on the Lipschitzness of the gradients, we consider the ℓ_2 distance between the loss gradient at a given point of the training and the gradients corresponding to different points along the original gradient direction in figure 5.
   - **Effective β-smoothness**: Finally, we look at the maximum difference (in ℓ_2-norm) in gradient over distance moved in that direction in figure 6.

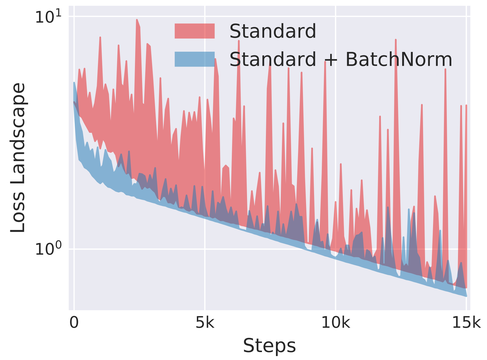


Figure 4. Loss Landscape

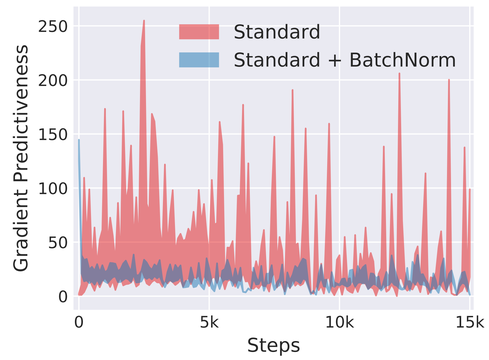


Figure 5. Gradient Predictiveness

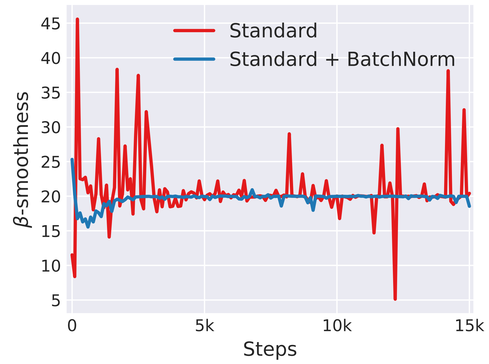


Figure 6. Effective β-smoothness

- We can see that the standard network shows higher variation in both the loss function and estimated gradients.
- This illustrates that BatchNorm leads to a smoother optimization landscape and more predictive gradients.

How does smoothening help?

1. The loss landscapes of Deep Neural Networks are usually non-convex and non-smooth, with several kinks, sharp minima and flat regions.
2. In such a case, SGD will be unstable and sensitive to learning rate and parameter initialization.
3. Improved Lipschitzness of the loss function and the gradients leads to a more accurate estimate of the gradient in SGD.
4. This allows us to take more significant steps in the estimated steepest descent direction without worrying about the abrupt landscape changes.
5. Consequently, BatchNorm leads to faster convergence rates and robustness to hyperparameter changes.

Conclusion

1. We saw how introducing BatchNorm layers helps the optimization process by improving Lipschitzness of both the loss landscape and the gradients.
2. Though we have primarily discussed the experimental results from [2], the paper also discusses supporting theoretical results where they prove improved Lipschitzness of the loss by analyzing the gradient (both wrt. activation and wrt. weight space) magnitude.

References

[1] S. Ioffe and C. Szegedy.
Batch normalization: Accelerating deep network training by reducing internal covariate shift.
In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, 2015.

[2] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry.
How does batch normalization help optimization?
In *Advances in Neural Information Processing Systems*, volume 31, 2018.