Batch Normalization

João Gabriel de Araújo Vasconcelos
Cin - UFPE
Universidade Federal de Pernambuco
Recife, Brazil
gvasconsbr@gmail.com

***Abstract*— Due to Deep Neural Networks having a change of distribution on each layer's input in it's architecture, this results on slowing down the training by requiring careful parameter initialization and lower learning rates (therefore, more training steps). In order to remediate this problem, the method of *BATCH NORMALIZATION consists of normalizing activation vectors from hidden layers using the first and second statistical moments (mean and variance) of the current batch, which turns the process of training faster and more stable. This document presents a relative/superficial review of the article *"Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift"*, written by Sergey Ioffe and Christian Szegedy - they are responsible for any acknowledgments presented in this article. For further studies on the matter, please resort to the original article.**

***Index Terms*—batch normalization, deep learning, neural networks**

I. Introduction

In the context of Deep Learning, the optimization method of Stochastic Gradient Descent (SGD) has proved to be an efficient way of training. It optimizes the parameters θ of the network, so as to minimize the loss

$$\theta = argmin \frac{1}{N} \sum_{i=1}^{N} l(xi, \theta) \tag{1}$$

where x1,...N is the training data set. In the SGD method, each training step is relative to a *mini-batch* x1,...m of size m. These mini-batches are used to approximate the gradient loss function with respect to parameters, by computing

$$\frac{1}{M} \frac{\delta l(xi, \theta)}{\delta \theta} \tag{2}$$

Using the mini-batch approach has proven to be better than a "one sample at a time" approach. The gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Also, the computations performed over a batch are more efficient than *m* computations for individual samples (due to parallelism implementation).

The "problem" with stochastic gradient descent is that it requires careful tuning of the model hyper-parameters, specifically the learning rate used in optimization, as well as the initial values for the model parameters. Also, the training is seen as troublesome, since the inputs to each layer are affected by the parameters of preceding layers - which implies that that small changes to the network parameters amplify as the network becomes deeper.

By changing the distributions of layers' inputs, a problem arises because the layers need to continuously adapt to the new distribution. Once the input distribution to a learning system changes, it is said to experience covariate shift. Covariate shift occurs when the distribution of variables in the training data is different to real-world or testing data. This means that the model may make the wrong predictions once it is deployed, and its accuracy will be significantly lower. For example, facial recognition models may have just been trained on the faces of people aged 20 to 30. When deployed, the model will naturally be less accurate when trying to map the faces of older people with different facial structures.

An important point is that the notion of covariate shift can be extended beyond the learning system as a whole, to apply to its parts, such as a sub-network or a layer. Consider, for example, a network computing

$$l = F2(F1(u, \theta 1), \theta 2) \tag{3}$$

where *F1* and *F2* are arbitrary transformations, and $\theta 1$ and $\theta 2$ are parameters to be learned so as to minimize the loss *l*. It is implied that learning $\theta 2$ can be viewed as if the inputs $x = F1(u, \theta 1)$ are fed into the sub-network

$$l = F2(x, \theta 2) \tag{4}$$

A gradient step, as follows

$$\theta 2 \leftarrow \theta 2 - \frac{a}{m} \sum_{i=1}^{m} \frac{\delta F2(xi, \theta 2)}{\delta \theta 2} \tag{5}$$

- for batch size *m* and learning rate *a* - is exactly equivalent to that for a stand-alone network $F2$ with input x. In this manner, the input distribution properties that make training more efficient – such as having the same distribution between the training and test data – apply to training the sub-network as well.

Therefore, it's implied that fixing the distribution of x over time proves to be an advantage, since $\theta 2$ does not have to readjust to compensate for the change in the distribution of x.

The change in the distributions of internal nodes of a deep network, in the course of training, is referred as Internal Covariate Shift. It's elimination offers a promise of faster training. It's proposed, therefore, a new mechanism, called Batch

Normalization, that takes a step towards reducing internal covariate shift in order to dramatically accelerates the training of deep neural nets. It accomplishes this via a normalization step that fixes the means and variances of layer inputs. Batch Normalization also has a beneficial effect on the gradient flow through the network, by reducing the dependence of gradients on the scale of the parameters or of their initial values. This allows us to use much higher learning rates without the risk of divergence. Furthermore, batch normalization regularizes the model and reduces the need for Dropout - as another regularization technique. Finally, Batch Normalization makes it possible to use saturating nonlinearities by preventing the network from getting stuck in the saturated modes.

II. Approach

A. Whitening Insertion

Internal Covariate Shift is defined as the change in the distribution of network activations due to the change in network parameters during training. To improve the training, it's of our best interest to reduce the internal covariate shift. By fixing the distribution of the layer inputs x as the training progresses, we expect to improve the training speed. Assuming (by previous researches) that the network training converges faster if its inputs are whitened – i.e., linearly transformed to have zero means and unit variances, and decorrelated. As each layer observes the inputs produced by the layers below, it would be advantageous to achieve the same whitening of the inputs of each layer. By whitening the inputs to each layer, we would take a step towards achieving the fixed distributions of inputs that would remove the ill effects of the internal covariate shift.

B. Normalization Accounted

Gradient descent optimization does not take into account the fact that the normalization takes place. To address this issue, we would like to ensure that, for any parameter values, the network always produces activations with the desired distribution. Doing so would allow the gradient of the loss with respect to the model parameters to account for the normalization, and for its dependence on the model parameters θ. We want to a preserve the information in the network, by normalizing the activations in a training example relative to the statistics of the entire training data.

III. Normalization via Mini-Batch Statistics

Since the full whitening of each layer's inputs is costly and not everywhere differentiable, we make two necessary simplifications. The first is that instead of whitening the features in layer inputs and outputs jointly, we will normalize each scalar feature independently, by making it have the mean of zero and the variance of 1.

Note that simply normalizing each input of a layer may change what the layer can represent. For instance, normalizing the inputs of a sigmoid would constrain them to the linear regime of the nonlinearity. To address this, we make sure that the transformation inserted in the network can represent the identity transform. To accomplish this we introduce, for each activation x^k, a pair of parameters γ^k,β^k, which scale and shift the normalized value:

$$y^k = \gamma^k \hat{x}^k + \beta^k \tag{6}$$

These parameters are learned along with the original model parameters, and restore the representation power of the network.

In the batch setting where each training step is based on the entire training set, we would use the whole set to normalize activations. However, this is impractical when using stochastic optimization. Therefore, we make the second simplification: since we use mini-batches in stochastic gradient training, each mini-batch produces estimates of the mean and variance of each activation. This way, the statistics used for normalization can fully participate in the gradient backpropagation.

Thus, BN transform is a differentiable transformation that introduces normalized activations into the network. This ensures that as the model is training, layers can continue learning on input distributions that exhibit less internal covariate shift, thus accelerating the training. Furthermore, the learned affine transform applied to these normalized activations allows the BN transform to represent the identity transformation and preserves the network capacity

A. Training and Inference with BatchNormalized Networks

To Batch-Normalize a network, we specify a subset of activations and insert the BN transform for each of them. Any layer that previously received x as the input, now receives BN(x). A model employing Batch Normalization can be trained using batch gradient descent, or Stochastic Gradient Descent with a mini-batch size m ¿ 1, or with any of its variants. The normalization of activations that depends on the mini-batch allows efficient training, but is neither necessary nor desirable during inference; we want the output to depend only on the input, deterministically. For this, once the network has been trained, we use the normalization

$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} \tag{7}$$

using the population, rather than mini-batch, statistics. Neglecting ϵ, these normalized activations have the same mean 0 and variance 1 as during training.

B. Batch-Normalized Convolutional Network

Batch Normalization can be applied to any set of activations in the network. Here, we focus on transforms that consist of an affine transformation followed by an element-wise nonlinearity:

$$z = g(Wu + b) \tag{8}$$

where W and b are learned parameters of the model, and g(·) is the nonlinearity such as sigmoid or ReLU. This formulation covers both fully-connected and convolutional layers. We add the BN transform immediately before the nonlinearity, by normalizing x = Wu+ b. We could have also normalized the layer inputs u, but since u is likely the output of another nonlinearity, the shape of its distribution is likely to change during training, and constraining its first and second moments would not eliminate the covariate shift. In contrast, Wu + b is more likely to have a symmetric, non-sparse distribution, that is "more Gaussian", normalizing it is likely to produce activations with a stable distribution.

For convolutional layers, we additionally want the normalization to obey the convolutional property – so that different elements of the same feature map, at different locations, are normalized in the same way. To achieve this, we jointly normalize all the activations in a minibatch, over all locations.

C. *Batch Normalization enables higher learning rates*

In traditional deep networks, too-high learning rate may result in the gradients that explode or vanish, as well as getting stuck in poor local minima. Batch Normalization helps address these issues. By normalizing activations throughout the network, it prevents small changes to the parameters from amplifying into larger and suboptimal changes in activations in gradients; for instance, it prevents the training from getting stuck in the saturated regimes of nonlinearities.

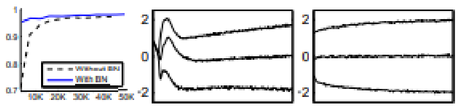
Batch Normalization also makes training more resilient to the parameter scale. Normally, large learning rates may increase the scale of layer parameters, which then amplify the gradient during backpropagation and lead to the model explosion.

D. *Batch Normalization regularizes the model*

When training with Batch Normalization, a training example is seen in conjunction with other examples in the mini-batch, and the training network no longer producing deterministic values for a given training example. In our experiments, we found this effect to be advantageous to the generalization of the network. Whereas Dropout is typically used to reduce overfitting, in a batch-normalized network we found that it can be either removed or reduced in strength.

IV. Experiments

A. *Activations over time*



(a) (b) Without BN (c) With BN

Relative to the picture: (a) The test accuracy of the MNIST network trained with and without Batch Normalization, vs. the number of training steps. Batch Normalization helps the network train faster and achieve higher accuracy. (b, c) The evolution of input distributions to a typical sigmoid, over the course of training, shown as 15, 50, 85th percentiles. Batch Normalization makes the distribution more stable and reduces the internal covariate shift.

B. *ImageNet classification*

The model was trained using a version of Stochastic Gradient Descent with momentum (Sutskever et al., 2013), using the mini-batch size of 32. The training was performed using a large-scale, distributed architecture. All networks are evaluated as training progresses by computing the validation accuracy @1, i.e. the probability of predicting the correct label out of 1000 possibilities, on a held-out set, using a single crop per image

In our experiments, we evaluated several modifications of Inception with Batch Normalization. In all cases, Batch Normalization was applied to the input of each nonlinearity, in a convolutional way, while keeping the rest of the architecture constant.

Further discussion and presentation on the experiments can be found in the original article cited on *Abstract*.

V. Conclusion

Merely adding Batch Normalization to a state-of-theart image classification model yields a substantial speedup in training. By further increasing the learning rates, removing Dropout, and applying other modifications afforded by Batch Normalization, we reach the previous state of the art with only a small fraction of training steps – and then beat the state of the art in single-network image classification. Furthermore, by combining multiple models trained with Batch Normalization, we perform better than the best known system on ImageNet, by a significant margin.

References

[1] Sergey Ioffe and Christian Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" [Submitted on 11 Feb 2015 (v1), last revised 2 Mar 2015 (this version, v3)]
[2] Johann Huber, "Batch normalization in 3 levels of understanding", Published in Towards Data Science