1502.03167v3

06/30/25 12:07 PM

1502.03167
v3

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe
Google Inc., sioffe@google.com

Christian Szegedy
Google Inc., szegedy@google.com

arXiv:1502.03167v3 [cs.LG] 2 Mar 2015

Abstract

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it hard to train models with saturating nonlinearities. We refer to this phenomenon as internal covariate shift, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization for each training mini-batch. Batch Normalization allows us to use much higher learning rates and be less careful about initialization, and also acts as a regularizer. In the batch-normalized inception network, it achieves the same accuracy with 14 times fewer training steps, and beat the original model by a significant margin. Using an ensemble of such networks, we improve on the state-of-the-art top-5 error on the validation set by 4.9%.

1 Introduction

Deep learning has dramatically advanced in the last year, with state-of-the-art results achieved on many tasks. Stochastic Gradient Descent (SGD) has proven to be an effective way of training deep networks, and SGD variants such as momentum and Adagrad were used to achieve the state-of-the-art. SGD optimizes the parameters Θ of the network, so that it minimizes the loss

$$\Theta = \arg\min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} \ell(x_i, \Theta)$$

where $x_1 \ldots N$ is the training dataset. With SGD, the training proceeds in steps, and at each step we consider a mini-batch $x_1 \ldots m$ of size m. The mini-batch is used to approximate the gradient of the loss function with respect to the parameters, by computing

$$\frac{1}{m} \frac{\partial \ell(x_i, \Theta)}{\partial \Theta}.$$

Using mini-batches of examples, as opposed to one example at a time, is helpful in several respects. First, the gradient of the loss over a mini-batch is an estimate of the gradient of the training set, which as the mini-batch size grows, gives a better estimate. Second, computation over a batch can be much more efficient than m computations for individual examples, due to the parallelism afforded by the modern computing platforms.

While stochastic gradient is simple and effective, 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. The training is complicated by the fact that the inputs to each layer are affected by the parameters of all preceding layers – so that small changes to the network parameters amplify as the network becomes deeper.

The change in the distributions of layers' inputs presents a problem because the layers need to continuously adapt to the new distribution. When the input distribution to a learning system changes, it is said to exhibit covariate shift. This is typically handled via domain adaptation (Jiang, 2008). However, the notion of covariate shift can be extended beyond a learning system as a whole, to apply to its parts, such as a sub-network or a layer. Consider a network computing

$$\ell = F_2(F_1(u, \Theta_1), \Theta_2)$$

where F_1 and F_2 are arbitrary transformations, and the parameters Θ_1 and Θ_2 are to be learned so as to minimize ℓ. Learning Θ_2 can be viewed as if the inputs $x = F_1(u, \Theta_1)$ are fed into the sub-network

$$\ell = F_2(x, \Theta_2).$$

For example, gradient descent step

$$\Theta_2 \leftarrow \Theta_2 - \frac{\alpha}{m} \sum_{i=1}^{m} \frac{\partial F_2(x_i, \Theta_2)}{\partial \Theta_2}$$

for batch size m and learning rate α is exactly equivalent to that for a stand-alone network F_2 with input x. Therefore the training of F_2 is not affected by the fact that x is the output of a learning system F_1.

Such layer inputs, which are the outputs of other layers, are subject to the effects of all the parameters in the preceding layers. Thus, any change in the distribution of F_1's output will be, in general, a change in the distribution of F_2's input, and will require the parameters Θ_2 to be readjusted. Consider a network with l layers, with the input x and the output ℓ. The network computes $\ell = F_{l}(\ldots F_2(F_1(x, \Theta_1), \Theta_2) \ldots, \Theta_l)$, and the parameters $\Theta_1, \ldots, \Theta_l$ are trained so as to minimize the loss ℓ. The training proceeds by feeding the inputs x to the first layer, and then the output of the first layer to the second layer, and so on. Let x_i be the output of the i-th layer, and F_i be the transformation that the layer applies to its input and parameters Θ_i. We can write the loss as

$$\ell = F_{l}(\ldots F_2(F_1(x, \Theta_1), \Theta_2) \ldots, \Theta_l)$$

and we can write the gradient descent step as

$$\Theta_i \leftarrow \Theta_i - \frac{\alpha}{m} \sum_{i=1}^{m} \frac{\partial F_i(x_i, \Theta_i)}{\partial \Theta_i}$$

for $i = 1 \ldots l$, where m is the batch size and α is the learning rate. Since x_i depends on x and all the parameters $\Theta_1 \ldots \Theta_{i-1}$ in the layers below, the loss ℓ depends on all the parameters $\Theta_1 \ldots \Theta_l$. Thus, any change in the parameters Θ_1 will affect the loss ℓ, and will affect the parameters Θ_2 of the layer F_2, as we have just seen above. Thus, the gradient $\frac{\partial F_i(x_i, \Theta_i)}{\partial \Theta_i}$ of each layer F_i depends on all the preceding layers, and changes to the distribution of any of the layers will affect the gradient of F_i and require it to adapt to the new input distribution.

Internal covariate shift can slow down the training, by requiring lower learning rates and careful parameter initialization, and it can make it hard to train models with saturating nonlinearities. We propose a novel method, which we call Batch Normalization, that takes steps towards reducing internal covariate shift, and in doing so, dramatically accelerates the training of deep neural nets. It achieves this by normalizing layer inputs, as we show in Section 2.1. Our method also has the effect of regularizing the model, and reduces the need for the Dropout regularizer (Srivastava et al., 2014). Finally, it makes the state-of-the-art image classification network (Szegedy et al., 2014) trainable to the same accuracy with significantly fewer training steps.

In Section 3 we review related work, and in Section 4 we describe the algorithm for Batch Normalization. In Section 5 we apply it to the ILSVRC 2012 image classification task, and show that it allows to greatly improve the training speed of the state-of-the-art deep network, and that the ensemble of such networks achieves the best published result on this task. We conclude with the discussion of related work and future work in Section 6.

2 Towards Reducing Internal Covariate Shift

Let us define the notion of internal covariate shift. Let $\ell = F(x, \Theta)$ be the loss to be minimized, where x is the training example and Θ are the parameters of the network. Let $x = (x_1 \ldots d)$ be the layer inputs, over the training set. Then, the training step based on the mini-batch $x_1 \ldots m$ is equivalent to minimizing $\frac{1}{m} \sum_{i=1}^{m} \ell(x_i, \Theta)$. With SGD, the training step based on the mini-batch $x_1 \ldots m$ is equivalent to minimizing $\frac{1}{m} \sum_{i=1}^{m} \ell(x_i, \Theta)$. With SGD, the training step based on the mini-batch $x_1 \ldots m$ is equivalent to minimizing $\frac{1}{m} \sum_{i=1}^{m} \ell(x_i, \Theta)$ for some learning rate α. Let us also define the normalized activation $\hat{x} = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]}}$. Then, we can write the gradient descent step as

$$\Theta \leftarrow \Theta - \frac{\alpha}{m} \sum_{i=1}^{m} \frac{\partial \ell(x_i, \Theta)}{\partial \Theta}$$