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

Sergey Ioffe Christian Szegedy

Google Inc.,sioffe@google.com Google Inc.,szegedy@google.com

This paper describes methods to address various issues related to the training of deep neural network. The aim of Batch Normalization is to keep neuron activation in Unit Gaussian Range (i.e zero mean and unit variance) and by doing so it ensures stable distribution of activation values inside the network which helps to converge the network faster during training.

Points discussed in this paper.

* Internal Covariate Shift.
* Vanishing Gradient.
* Batch Normalization as a solution.
* Batch Normalized Convolutional Networks.
* Advantages of Batch Normalization.
* Future work
* **Internal Covariate Shift (ICS)**

Well, batch normalization is a technique used to improve the training performance by reducing internal covariate shift. Given a neural network composed of several layers, the forward propagation can be abstracted as function composition: each layer can indeed be viewed as a function that takes as input a vector from Rn and outputs a vector from Rm (i.e. for a layer of ‘m’ neurons preceded by a layer of ‘n’ neurons). For each function there is a linear mapping from Rn to Rm , given a matrix of w of weights of dimension m, n followed by the addition of biases and b (b ⋲ Rm) and finally non-linear activation (e.g Sigmoid activation function).

Where, (x) = 1 / (1 + e-x) : sigmoid activation function,

= (1- (x))(x) : derivative of sigmoid activation

When a network is viewed as functions composed with one another, it is easier to notice that the output of a function (except for the first layer also known as input layer) would be affected by the small changes of all preceding functions. Let’s consider the sigmoid activation function. Let, z = (W*u* + b), for *u* the input and W, b respectively the weights and the biases. It can be verified that as |x| grows, derivative of sigmoid i.e. approaches 0, meaning that the gradient of sigmoid function vanishes. In case of our layer, for each of its dimension x = W*u* + b, except for low values of the components of x, the gradient will vanish. In such case, at each forward-propagation and back-propagation, a large portion of neurons will permanently stop learning.

This problem has ability to amplify the network as it goes deeper. For each back-propagation and for each layer, small changes in the output of the preceding layer might shift the distribution of the inputs of the layer to vanishing gradient zone. At the end of training, a lot of neuron might be in saturated regime of nonlinearities and they will be prevented from learning anything, which will probably decrease the accuracy of network. This problem is called as Internal Covariate Shift and when neural network experience this problem during training, it is said to have internal covariate shift problem. The ideal network is one whose layers remain stable during training which ultimately improves the overall performance because layers need to be continuously adapted to the new distribution and network also converges faster if inputs are whitened ( i.e. zero mean and unit variance).

We can take care of internal covariate shift by carefully initializing the weight parameters of the network. Popular weight initialization methods were Xavier’s weight initialization and He weight initialization. But in case of He initialization, we have to take into account new and complicated hyper-parameter. By the way Rectified Linear unit (ReLU) were also introduced to solve, at least partially, the problem of vanishing gradient. Partially because it has dying ReLU problem. Meaning, ReLU unit can die during training. So, Leaky ReLU was also introduced to solve dying ReLU problem, where, instead threshold to 0 for negative activation values, it introduced some negative slope which does not saturates.

ICS and vanishing gradients made it harder to go deeper with neural network. Even when not too many weights and biases were sent to saturated regime of non-linearity, back-propagation step could be more about correcting the change done by previous layer. This could also be handled by dropout. Dropout can be viewed as a way to make the computation of each neuron independent and ensuring that there are more redundant nodes in the network that can also contribute during prediction. Overall, ICS was counteracted with careful initialization, very low learning rate and the addition of dropout.

* **Vanishing Gradient :**

Saturating non-linearities such as Sigmoid and tanh activation function presents vanishing gradient problem when their activation values get saturated. So we cannot use this non linearity in training deep neural network. Alternatively, we have ReLU (Rectified Linear unit) which does not saturates but researchers and machine learning practitioners recommend Leaky ReLU (it’s varient) because ReLU also saturates in negative region of activation values. Also in order to handle this issue, we require smaller learning rate and careful parameter initialization.

* **Batch Normalization as a solution:**

The solution came with the realization that when the inputs of network are transformed so that they have a zero mean and unit variance, the network converges faster. At first Ioffe et. al. tried to subtract mean computed over all training values from training samples. By doing that, they realized that some parameters could increase indefinitely through back-propagation while overall loss would remain unchanged. To solve this issue, the authors divided the output of a layer by variance of batch values. Ioffe et. al tried to introduce this transformation inside the network with batch of activation values is then transformed to normalized activation values and the beauty of this framework is that it is end-to-end differentiable, so that both the scaling and shifting parameter (γ, β) can be learned.

Suppose we have a batch of activation values and we want to normalize it having d dimension as X = (X1, X2,....,Xd), then we can normalize the Kth dimension as follows –

X~(K)  = (X(K) - E[X(K)]) / + ) , where X~(K)  is normalized activation .

Where E[X(K)] – Mean of training batch values & – it’s variance

Where, is introduced to prevent division by zero in case the variance equaled zero.

We also need to scale and shift these values because just normalizing a layer may change its representational power and would limit the layer in terms of what it can represent. For example, if we normalize the sigmoid activation function, then the output would be bound to linear region only. Here we estimate mean and variance of activation. So the normalized activation given by following equation –

Yk = γ(K) X~(K)  + β(K)

Where *γ* and *β* are parameters to be learned used for scaling and shifting.

This YK is the final normalized activation data ensures that as the model is training, the layers can learn on the input distribution that exhibit less internal covariate shift and can hence accelerate the training.

At training time, a batch of activation is supplied and BN transform is applied to all of them.

At testing time, the normalization is done by taking running averages of mean and variance during training in order to ensure that the output deterministically depends on the input.

* **Batch Normalized Convolutional Networks :**

Batch Normalization (BN) can be applied to any set of activation values in the network. Let’s say we have an transformation function followed by an element-wise non-linearity: z = g(Wu+b). where W, b are learnable parameters of the model and g(.) is the non-linearity such as Sigmoid or ReLU. It covers both fully-connected and convolutional layers.

We add BN immedietly before the non-linearity, by normalizing x = (Wu+b). Well, there is always a question that shall we add BN after non-linearity or before? Adding BN after ReLU also make sense as in this case, it normalizes only positive activation values. We could try out both and check results. We could even normalize the layer inputs ‘u’, but it is likely the output of other nonlinearity and shape of its distributions may change during training. Instead we normalize (Wu+b) as it is likely to have symmetric, non-sparse distribution which produces activation with stable distributions during training. Since we normalize (Wu+b), the bias b can be ignored since its effect will be canceled by subsequent mean subtraction. Thus z becomes, z = g(BN(Wu)), where BN is applied to each dimension of z, with a pair of learned parameters *γk* and *βk* per dimension.

For convolution layers, normalization should follow convolution property – i.e different elements of same feature map, at different locations are normalized in the same way. So all the activations in a mini-batch are normalized over all the locations and parameters *(γk* and *βk)* are learned per feature map instead of per activation.

Ioffe et. al. succeeded in training their network with much higher learning rate, while reducing the need for dropout and also allowing the usage of saturating non-linearities for deep network. As a matter of fact, they reduced the training time by a factor of 14 and they even gained a few percentage points accuracy for image classification. Today batch normalization is commonly used for vision-related task and deep learning in general.

**Advantages of Batch Normalization :**

* It reduces internal co-variate shift.
* It gives ability to converge faster during training.
* It allows using higher learning rate.
* It allows the use of saturating non-linearity.
* It regularizes the model and reduces the need for dropout.
* It reduces the dependence of gradient on the scale of parameters or their initial parameters.
* Future work

Future work includes application of BN to Recurrent Neural Network, where the internal co-variate shift, vanishing gradient or exploding gradients may be a major issue. It remains to be unexplored whether BN can help with domain adaptation i.e whether normalized network would allow it to more easily generalize to new data distribution. So, further theoretical analysis of the algorithm is required for better performance and improvements.