AOS2 - Deep Learning

Université de Technologie de Compiègne

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

Haojie LU - Zinan ZHOU

A19

1 Introduction

Batch Normalization(BN) proposed by Google in 2015 [1], has been considered as an important achievement of Deep learning in the past year. It has been proved to have an excellent performance in most cases.

A big problem in deep learning is that : with the deepening of network, the model training becomes more difficult and its convergence becomes slower. Many papers have been proposed to solve this problem, such as the ReLU and then Residual Network. But BN essentially explains this phenomenon and solves it from a different way.

The BN's main idea is to fix the distribution of the input to each hidden layer, which allows us to use much higher learning rates without the risk of divergence and be less careful about initialization, and in some cases eliminates the need for Dropout.

In this report, we will firstly talk about the phenomenon of internal covariate shit. Then we will introduce the idea of Batch normalization and its implementation in detail, with some our own understanding about why BN can accelerate training. Next, some result in experiments would be used to prove its good performance in practice. At last, we will make a conclusion and some disadvantage will be discussed in that part.

2 Internal Covariate Shift

As can be seen from the paper name, BN is used to solve the problem of "Internal Covariate Shift". For a deep network with many hidden layers, the distribution of each layer's inputs changes during training, as the parameters of the previous layers change, which results in the layers need to continuously adapt to the new distribution. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating non-linearities. We refer to it as Internal Covariate Shift.

3 Batch Normalization

3.1 Main idea

In order to reduce the ill effects of the internal covariate shift, we propose our basic idea about Batch Normalization: we make normalization as a part of the model architecture and perform the normalization for each training mini-batch, which can be accomplished via a normalization step that fixes the means and variances of layer inputs.

Actually, this idea comes from many previous studies having shown that if the input image is whitened in image processing, which transforms the input data distribution to standard normal distribution, the neural network will converge faster. From this point, for a deep neural network, each hidden layer can be seen as the input layer of the next layer. That is original idea inspiring BN.

Then, we would explain why BN can make it easy to train models with saturated activation function. As the network depth deepens or during the training process, the distribution of input value before entering a non-linear activation(that is, $x = WU + B$) will gradually shift or change. Generally, the overall distribution gradually approaches the saturated regime of nonlinearity, so this leads to a small gradient during backpropagation, which is the essential reason that the network training converges slower.

Batch Normalization can force the distribution of the input value of any neuron in each layer through certain normalization back to the standard normal distribution so that the activation input value falls in a region where the non-linear function is more sensitive to the input. That means that its gradient will be larger in order to reduce vanishing gradient problem.

Next, we will give a more vivid example as below, used sigmoid as activation function.

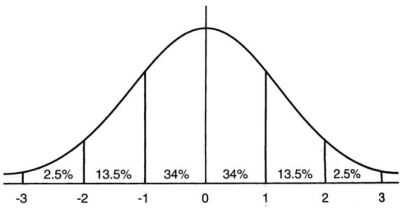

Figure 1: Standard normal distribution.

From Fig.1, we can find that within a standard deviation range, there is a 68% probability of x whose value falls within the range of [-1,1] and a 95% probability falling within [-2,2]. Consider a layer with sigmoid activation function $g(x) = \frac{1}{1+exp(-x)}$ and then look at its distribution and its derivatives in Fig.2.

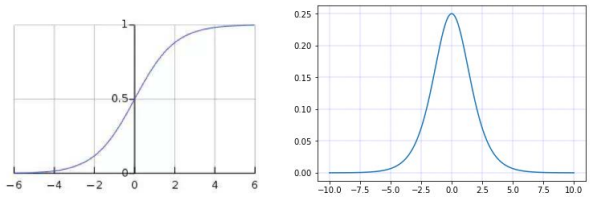

Figure 2: Sigmoid(x) and its derivatives

Assuming that the original normal distribution of x before Batch Normalization with the mean -6 and the variance 1, it means that 95% of the values fall between [-8, -4], then the value of Sigmoid (x) function is obviously close to 0. This is a typical gradient saturation region, where the corresponding derivative is close to 0. That means that the gradient change is small or even disappears which is highly possible to lead a vanishing gradient problem.

However, with BN the mean and the variance of input x can be transformed to respectively 0 and 1. It means that 95% of the x values falling within the interval [-2,2]. Obviously this region is the area where the sigmoid (x) function is close to the linear transformation, which means a much bigger value of gradient than 0. It would be less likely to get stuck in the saturated regime and the training would accelerate.

Nevertheless, someone may have a question : if we use Relu as our activation function, BN would not make sense? In fact, we use here sigmoid to help us to understand why BN can accelerate training from another side. It does not say that BN can only make sense by relying on activation functions such as Sigmoid and Tanh. The core idea of BN is to to reduce internal covariate shift. No matter what activation function is used, as the number of network layers deepens, internal covariate shift will occur, which will also cause network learning slow down.

3.2 Training with BN

Since the full whitening of each layere's inputs is costly, we make two necessary simplification.

- Instead of whitening the features in layer inputs and outputs jointly, we normalize each scalar feature independently. For a layer with d-dimensional input $x = (x^{(1)}...x^{(d)})$, the BN operation performs the following transformation, where the expectation and variance are computed over the training data set:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}\left[x^{(k)}\right]}{\sqrt{\mathrm{Var}\left[x^{(k)}\right]}}$$

  However, normalizing the inputs of a sigmoid would constrain them to the linear regime of the nonlinearity, which results in a decreasing of the network's expression ability. To address this, BN introduces, for each activation $x^{(k)}$, a pair of parameters $\gamma^{(k)}, \beta^{(k)}$, which scale and shift the normalized value:

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

- We use mini-batches in stochastic gradient training, each mini-batch produces estimates of the mean and variance of each activation.

In summary, the specific process of BN is as follows.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithom 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

3.3 Inference

During inference, it is obvious that there is only one instance of the input, and it is impossible to get the mean and variance we need in transformation. Therefore, the global statistics obtained from all training instances are directly used to replace the mean and variance. In order to get them, we try to compute the mean and variance over that of multiple training mini-batches :

$$E[x] \leftarrow E_{\mathrm{B}}\left[\mu_{\mathrm{B}}\right]$$
$$\mathrm{Var}[x] \leftarrow \tfrac{m}{m-1} E_{\mathrm{B}}\left[\sigma^2_{\mathrm{B}}\right]$$

Having added the scaling β and shift γ, the total BN transformation in inference process can written as below. Since the means and variances are fixed during inference, the normalization is simply a linear transform applied to each activation.

$$y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\varepsilon}} \cdot x + \left(\beta - \frac{\gamma \cdot E[x]}{\sqrt{\mathrm{Var}[x]+\varepsilon}}\right)$$

4 Conclusion

Currently, the most commonly used deep learning basic models include forward neural network (MLP), CNN and RNN. At present, BN has tried these basic network structures. In general, BN has been very successful in MLP and CNN except RNN. Because of the good effect, BatchNorm has basically become the standard for various networks (except RNN).

The great effects are reflected in, first, the convergence process is greatly accelerated and greatly improving the training speed. Secondly, increase the classification effect. This is a regularization expression similar to Dropout to prevent over-fitting can replace Dropout. Finally, Simplify the parameter adjustment process, the initialization requirements are not high. So we can use a large learning rate.

However, despite the many benefits of Batch Norm, there are still many problems.

BN is heavily dependent on the training examples in Mini-Batch. If the batch size is small, the task effect will be significantly reduced. The reason for this is that a small Batch Size means that there are fewer data samples and therefore no valid statistics can be obtained.
Because a large batch size is needed to reasonably estimate the mean and variance of the training data, this may cause insufficient memory and it is also difficult to apply it to RNN models with different training data lengths. In addition to BN, there are other normalizations. For example, one advantage of Layer Normalization (LN) is that it does not require batch training and can be normalized within a single piece of data.

References

[1] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.