Normalization Techniques in Deep Neural Networks

Aakash Bindal Follow
Feb 10, 2019 · 7 min read

> *We are going to study Batch Norm, Weight Norm, Layer Norm, Instance Norm, Group Norm, Batch-Instance Norm, Switchable Norm*

Let's start with the question,

Why do we need Normalization ?

Normalization has always been an active area of research in deep learning. Normalization techniques can decrease your model's training time by a huge factor. Let me state some of the benefits of using Normalization.

1. It normalizes each feature so that they maintains the contribution of every feature, as some feature has higher numerical value than others. This way our network can be unbiased(to higher value features).
2. It reduces **Internal Covariate Shift**. It is the change in the distribution of network activations due to the change in network parameters during training. To improve the training, we seek to reduce the internal covariate shift.
3. In this paper, authors claims that Batch Norm makes loss surface smoother(i.e. it bounds the magnitude of the gradients much more tightly).
4. It makes the Optimization faster because normalization doesn't allow weights to explode all over the place and restricts them to a certain range.
5. An unintended benefit of Normalization is that it helps network in Regularization(only slightly, not significantly).

From above, we can conclude that getting Normalization right can be a crucial factor in getting your model to train effectively, but this isn't as easy as it sounds. Let me support this by certain questions.

1. How Normalization layers behave in Distributed training ?
2. Which Normalization technique should you use for your task like CNN, RNN, style transfer etc ?
3. What happens when you change the batch size of dataset in your training ?
4. Which norm technique would be the best trade-off for computation and accuracy for your network ?

To answer these questions, Let's dive into details of each normalization technique one by one.

Batch Normalization

Batch normalization is a method that normalizes activations in a network across the mini-batch of definite size. For each feature, batch normalization computes the mean and variance of that feature in the mini-batch. It then subtracts the mean and divides the feature by its mini-batch standard deviation.

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

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

this image is taken from https://arxiv.org/pdf/1502.03167.pdf%27

But wait, what if increasing the magnitude of the weights made the network perform better?

To solve this issue, we can add γ and β as scale and shift learn-able parameters respectively. This all can be summarized as:

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

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

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

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

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

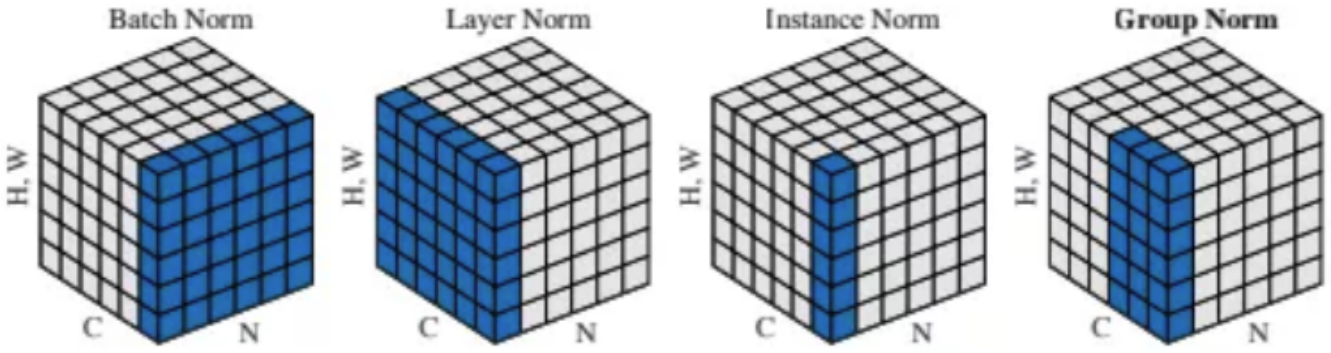
ϵ is the stability constant in the equation.

Problems associated with Batch Normalization :

1. **Variable Batch Size** → If batch size is of 1, then variance would be 0 which doesn't allow batch norm to work. Furthermore, if we have small mini-batch size then it becomes too noisy and training might affect. There would also be a problem in **distributed training**. As, if you are computing in different machines then you have to take same batch size because otherwise γ and β will be different for different systems.

2. **Recurrent Neural Network** → In an RNN, the recurrent activations of each time-step will have a different story to tell(i.e. statistics). This means that we have to fit a separate batch norm layer for each time-step. This makes the model more complicated and space consuming because it forces us to store the statistics for each time-step during training.

> *Batch norm alternatives(or better **norm**s) are discussed below in details but if you only interested in very short description(or revision just by look at an image) look at this :*



A visual comparison of various normalization methods

This image is taken from https://arxiv.org/pdf/1803.08494.pdf

Weight Normalization

Wait, why don't we **normalize weights of a layer** instead of normalizing the activations directly. Well, Weight Normalization does exactly that.

Weight normalization reparameterizes the weights (**ω**) as :

$$\boldsymbol{w} = \frac{g}{\|\boldsymbol{v}\|}\boldsymbol{v}$$

It separates the weight vector from its direction, this has a similar effect as in batch normalization with variance. The only difference is in variation instead of direction.

As for the mean, authors of this paper cleverly combine **mean-only batch normalization** and **weight normalization** to get the desired output even in small mini-batches. It means that they subtract out the mean of the minibatch but do not divide by the variance. Finally, they use weight normalization instead of dividing by variance.

> *Note: Mean is less noisy as compared to variance(which above makes mean a good choice over variance) due to the* ***law of large numbers*****.**

The paper shows that weight normalization combined with mean-only batch normalization achieves the best results on CIFAR-10.

Layer Normalization

Layer normalization normalizes input across the features instead of normalizing input features across the batch dimension in batch normalization.

A mini-batch consists of multiple examples with the same number of features. Mini-batches are matrices(or tensors) where one axis corresponds to the batch and the other axis(or axes) correspond to the feature dimensions.

$$\mu_i = \frac{1}{m}\sum_{j=1}^{m} x_{ij}$$
$$\sigma_i^2 = \frac{1}{m}\sum_{j=1}^{m}(x_{ij} - \mu_i)^2$$
$$\hat{x_{ij}} = \frac{x_{ij} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

i represents batch and j represents features. $x_{i,\ j}$ is the i,j-th element of the input data.

The authors of the paper claims that layer normalization performs better than batch norm in case of **RNN**s.

Instance(or Contrast) Normalization

Layer normalization and instance normalization is very similar to each other but the difference between them is that instance normalization normalizes across each channel

in each training example instead of normalizing across input features in an training example. Unlike batch normalization, the instance normalization layer is applied at test time as well(due to non-dependency of mini-batch).

$$y_{tijk} = \frac{x_{tijk} - \mu_{ti}}{\sqrt{\sigma_{ti}^2 + \epsilon}}, \quad \mu_{ti} = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H} x_{tilm}, \quad \sigma_{ti}^2 = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H}(x_{tilm} - mu_{ti})^2.$$

This image is taken from https://arxiv.org/pdf/1607.08022.pdf

Here, $\mathbf{x} \in \mathbb{R}^{T \times C \times W \times H}$ be an input tensor containing a batch of T images. Let $\mathbf{x}_i$ denote its tijk-th element, where k and j span spatial dimensions(**H**eight and **W**idth of the image), i is the feature channel (color channel if the input is an RGB image), and t is the index of the image in the batch.

This technique is originally devised for **style transfer**, the problem instance normalization tries to address is that the network should be agnostic to the **contrast** of the original image.

Group Normalization

As the name suggests, Group Normalization normalizes over group of channels for each training examples. We can say that, Group Norm is in between Instance Norm and Layer Norm.

∵ When we put all the channels into a single group, group normalization becomes Layer normalization. And, when we put each channel into different groups it becomes Instance normalization.

$$\mu_i = \frac{1}{m}\sum_{k \in \mathcal{S}_i} x_k, \quad \sigma_i = \sqrt{\frac{1}{m}\sum_{k \in \mathcal{S}_i}(x_k - \mu_i)^2 + \epsilon},$$

S_i is defined below.

$$\mathcal{S}_i = \{k \mid k_N = i_N, \lfloor \frac{k_C}{C/G} \rfloor = \lfloor \frac{i_C}{C/G} \rfloor\}.$$

$$\hat{x}_i = \frac{1}{\sigma_i}(x_i - \mu_i).$$

$$y_i = \gamma \hat{x}_i + \beta,$$

Here, **x** is the feature computed by a layer, and i is an index. In the case of 2D images, **i** = (**i**N , **i**C , **i**H, **i**W) is a 4D vector indexing the features in (N, C, H, W) order, where N is the batch axis, **C** is the channel axis, and **H** and **W** are the spatial height and width axes. **G** is the number of groups, which is a pre-defined hyper-parameter. C/G is the number of channels per group. ⌊.⌋ is the floor operation, and "**⌊kC/(C/G)⌋ = ⌊iC/(C/G)⌋**" means that the indexes **i** and **k** are in the same group of channels, assuming each group of channels are stored in a sequential order along the C axis. GN computes μ and σ along the (H, W) axes and along a group of C/G channels.

Batch-Instance Normalization

The problem with Instance normalization is that it completely erases style information. Though, this has its own merits(such as in style transfer) it can be problematic in those conditions where contrast matters(like in weather classification, brightness of the sky matters). Batch-instance normalization attempts to deal with this by learning how much style information should be used for each channel(**C**).

Batch-Instance Normalization is just an interpolation between batch norm and instance norm.

$$\mathbf{y} = \left(\rho \cdot \hat{\mathbf{x}}^{(B)} + (1-\rho) \cdot \hat{\mathbf{x}}^{(I)}\right) \cdot \gamma + \beta,$$

the value of ρ is in between 0 and 1.

The interesting aspect of batch-instance normalization is that the balancing parameter ρ is learned through gradient descent.

From batch-instance normalization, we can conclude that models could learn to adaptively use different normalization methods using gradient descent.

> *Understanding from above, a question may arise.*

Can we switch the normalization technique whenever needed ?

The answer would be **Yes.** Following technique does exactly that.

Switchable Normalization

This paper proposed switchable normalization, a method that uses a weighted average of different mean and variance statistics from batch normalization, instance normalization, and layer normalization.

The authors showed that switch normalization could potentially outperform batch normalization on tasks such as image classification and object detection.

The paper showed that the instance normalization were used more often in earlier layers, batch normalization was preferred in the middle and layer normalization being used in the last more often. Smaller batch sizes lead to a preference towards layer normalization and instance normalization.

References

1. https://arxiv.org/pdf/1502.03167.pdf%27
2. https://arxiv.org/pdf/1607.06450.pdf
3. https://arxiv.org/pdf/1602.07868.pdf
4. https://arxiv.org/pdf/1607.08022.pdf
5. https://arxiv.org/pdf/1803.08494.pdf
6. https://arxiv.org/pdf/1805.07925.pdf
7. https://arxiv.org/pdf/1811.07727v1.pdf

Machine Learning Neural Networks Batch Normalization Artificial Intelligence Turingai