Batch Normalization Notes

Kelvin Lee

April 20, 2021

The following notes are taken from *Batch Normalization: Accelerating Deep Network Training by Reducing Indernal Covariate Shift* by Ioffe et al. [1].

Contents

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

1.1 Motivation

Training Deep Neural Networks can be difficult due to the following problem:

1.1.1 Internal Covariate Shift

Covariate shift refers to the change in the input distribution to a learning system. For neural networks, each layer's inputs are affected by parameters in all input layers. Hence, the effects caused by small changes in the input get amplified as the network becomes deeper, which results in the change in the input distribution of the internal layers. Hence, the term *internal covariate shift* refers to the change in the distribution of network activations due to the change in parameters during training. The training process is then complicated due to the fact that layers need to continuously adapt to the new distribution.

1.2 Batch Normalization

The paper [1] proposed a new mechanism known as *Batch Normalization* that aims to reduce internal covariate shift in order to accelerate the training process of deep neural nets via normalization of the layer inputs.

1.2.1 Normalization via Mini-Batch Statistics

Full whitening of each layer's inputs is costly and not everywhere differentiable, we make two necessary simplifications. The first is to normalize each scalar feature independently. For a layer with d-dimensional input $\mathbf{x} = (x^{(1)}, \cdots, x^{(d)})$, normalize each dimension

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}},$$

where the expectation and variance are computed over the training data set. It is known that such normalization speeds up convergence.

However, simply normalizing the inputs of a layer may change what the layer can represent (e.g. normalizing inputs of a sigmoid would contrain them to the linear regime of the nonlinearity.) To address this, the paper ensures that *the transformation inserted in the network can represent the identity transform.* In order to achieve this, parameters $\gamma^{(k)}, \beta^{(k)}$ are introduced for scaling and shifting the normalized value:

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

They are learned along with the model parameters so that the representation power of the network can be restored. Note that by letting $\gamma^{(k)} = \sqrt{\text{Var}[x^{(k)}]}$ and $\beta^{(k)} = \mathbb{E}[x^{(k)}]$, the original activations could then be recovered (if optimal).

In the batch setting, training steps are based on the entire training set and the whole set is used to normalize activations. This is impractical for stochastic optimization. Hence comes the

second simplification: just like we use mini-batches in SGD, use mini-batches with normalization as estimates of the mean and variance in each activation.
Consider a mini-batch $\mathcal{B}$ of size m and a particular activation $x^{(k)}$ (k will be omitted for simplicity).

$$\mathcal{B} = \{x_i\}_{i=1}^m.$$

Let $\{\widehat{x}_i\}_{i=1}^m$ be the normalized values and $\{y_i\}_{i=1}^m$ be the linear transformations. The transform

$$\mathrm{BN}_{\gamma,\beta} : \{x_i\}_{i=1}^m \to \{y_i\}_{i=1}^m$$

is called the *Batch Normalizing Transform*. The algorithm is as follows:

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

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

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

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

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

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

Figure 1: Adapted from the paper [1].

During training, the gradient of loss $\mathcal{L}$ through this transformation needs to be backpropagated and the gradients with respect to the parameters of the BN transform (γ, β). By chain rule,

$$\frac{\partial \mathcal{L}}{\partial \widehat{x}_i} = \frac{\partial \mathcal{L}}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \mathcal{L}}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^m \frac{\partial \mathcal{L}}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}\left(\sigma_{\mathcal{B}}^2 + \epsilon\right)^{-3/2}$$

$$\frac{\partial \mathcal{L}}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^m \frac{\partial \mathcal{L}}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}\right) + \frac{\partial \mathcal{L}}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{\sum_{i=1}^m -2\,(x_i - \mu_{\mathcal{B}})}{m}$$

$$\frac{\partial \mathcal{L}}{\partial x_i} = \frac{\partial \mathcal{L}}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \mathcal{L}}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2\,(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \mathcal{L}}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

$$\frac{\partial \mathcal{L}}{\partial \gamma} = \sum_{i=1}^m \frac{\partial \mathcal{L}}{\partial y_i} \cdot \widehat{x}_i$$

$$\frac{\partial \mathcal{L}}{\partial \beta} = \sum_{i=1}^m \frac{\partial \mathcal{L}}{\partial y_i}$$

This shows that the BN transform is differentiable and it ensures that the layers can learn on the input distributions that exhibit less internal covariate shift as the model is training and hence

accelerates the training. In addition, the learned affine transform applied to the normalized activations allows the BN transform to r*epresent the identity transformation and preserves the network capacity.*

1.3 Training and Inference With Batch-Normalized Networks

Training: To batch-normalize a network, simply apply the BN transform to a specified subset of activations according to Alg. 1. The applied input x now becomes $\text{BN}(x)$ and models with batch norm can be trained using BGD or SGD.

Inference: During inference, normalization dependent on mini-batch is not necessary or desirable since we want the output to depend only on the input. Hence, we use the normalization

$$\widehat{x} = \frac{x - \mathbb{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}}$$

using the population statistics. The unbiased variance estimate $\text{Var}[x] = \frac{m}{m-1}$ is used. Since the means and variances are fixed during inference, the normalization is simply a linear transform applied to each activation instead of $\text{BN}(x)$. The procedure for training training batch-normalized networks is as follows:

Input: Network N with trainable parameters Θ;
subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\text{BN}}^{\text{inf}}$

1: $N_{\text{BN}}^{\text{tr}} \leftarrow N$ // Training BN network
2: **for** $k = 1 \ldots K$ **do**
3: Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}}$ (Alg. 1)
4: Modify each layer in $N_{\text{BN}}^{\text{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
5: **end for**
6: Train $N_{\text{BN}}^{\text{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
7: $N_{\text{BN}}^{\text{inf}} \leftarrow N_{\text{BN}}^{\text{tr}}$ // Inference BN network with frozen // parameters
8: **for** $k = 1 \ldots K$ **do**
9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
10: Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\text{E}[x] \leftarrow \text{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\text{Var}[x] \leftarrow \frac{m}{m-1}\text{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

11: In $N_{\text{BN}}^{\text{inf}}$, replace the transform $y = \text{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\text{E}[x]}{\sqrt{\text{Var}[x]+\epsilon}}\right)$
12: **end for**

Algorithm 2: Training a Batch-Normalized Network

Figure 2: Adapted from the paper.

1.3.1 Advantages of Batch Normalization

- Reduces internal covariate shift.
- Reduces the dependence of gradients on the scale of the parameters or their initial values.
- Regularizes the model and reduces the need for dropout, and other regularization techniques.

- Allows use of saturating nonlinearities and higher learning rates.
- Adds two extra parameters per activation, which preserves the representation ability of the network.

References

[1] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].