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

Sergey Ioffe, Christian Szegedy

Presenters:

Sohan Patnaik and Yatindra Indoria

Reading Session XI

Kharagpur Data Analytics Group
IIT Kharagpur

August 22, 2021

Important Notes

- We expect you all to have certain queries regarding the presentation, certain intricate doubts maybe... **Please put them in the chat of this meeting (if you feel shy) else unmute and speak**.
- Finally, a **Google Form** [Link] will be released for feedback but most importantly, we ask you to put up name of any AI-related paper to be presented in the upcoming sessions!

Link to GitHub Repo : Click Here
Link to join Slack Workspace : Click Here
Link to KDAG YouTube Channel : Click Here
Link to doc on good AI Blogs/Resources/Topics : Click Here

Outline

- Introduction
  - Internal Covariate Shift
  - Batch Normalization
- Reducing Internal Covariate Shift
- Normalization via Mini-batch
  - Backpropagation
  - Training and Inference
  - Batch Normalized CNNs
  - Higher Learning Rates
- Experiments and Results
  - Activations over time
  - ImageNet Classification
- Conclusion

Introduction

1) Gradient descent and it's variations (momentum, Adagrad, Stochastic, etc.) are widely used in the training of deep learning models, as the computation over a mini-batch of data (of size 'm') is faster and more efficient in modern machines, than 'm' separate operations.

2) **Internal Covariate Shift:** As the training proceeds, the 'data' distribution in the hidden-layers, itself changes. We refer to the **change in the distributions of internal nodes of a deep network,** in the course of training, as 'Internal Covariate Shift'.

   As the input increases in magnitude, derivative of the 'Saturated Non Linearity' (eg. Sigmoid function) goes to zero, which makes the models very hard to train, (i.e. 'x' being in the saturated region, slows down convergence).

3) **Batch Normalization:** The mean of a distribution is made '0', and the variance '1'.

Reducing Internal Covariate-Shift

- Whitened Inputs: The inputs having mean zero and unit covariance have faster convergence towards optimality
- Inputs can be whitened either by directly modifying the network or by changing the parameters of the optimization algorithm
    - Problem: the gradient descent step may attempt to update the parameters in a way that requires the normalization to be updated, which reduces the effect of the gradient step.
- For example,

- **Issue with the previous approach:** Gradient descent optimization does not take into account the fact that the normalization takes place.
- **Solution:** If the network always produces activations of the desired distribution for any parameter values, the issue would be resolved.
- Let x be the layer input and X be the set of these inputs, the normalization can be written as

$$\hat{x} = \text{Norm}(x, \mathcal{X})$$

- If the training data is quite large, calculating the covariance matrix would be computationally expensive.

Normalization via Mini-Batch

- As full whitening of each layers input is costly, each of the input features is normalized independently.

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

- Simply normalizing poses a problem i.e., let the activation be sigmoid, simply normalizing would constrain the activation in the linear regime of sigmoid, restricting the model to capture complex information.

- To tackle this, two more parameters are introduced to scale the input and shift them out of the linear regime.

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

Backpropagation

Batch Normalization can be simply added as a layer to the network which will manipulate the activation before feeding into the next layer.

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

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

$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

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

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

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

Training and Inference with Batch-Normalized Networks

- A subset of activations and the BN transform for each of them is introduced i.e., any layer that previously received x as the input, now receives BN(x).
- The model can be trained using Batch Gradient Descent.
- After the model is trained, the normalization used is

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

- Unbiased variance estimate is used

$$\mathrm{Var}[x] = \frac{m}{m-1} \cdot \mathrm{E}_{\mathcal{B}}[\sigma^2_{\mathcal{B}}]$$

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

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

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

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

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

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

Batch-Normalized CNN

- Affine transformation: Transformation that preserves collinearity.
- Similar to usual normalization, CNNs to have the Batch Normalization layer with one constraint.
- The different elements of the same feature map, at different locations, are normalized in the same way so that they still obey the convolution property.

Higher Learning Rates

- The gradients do not vanish or explode.
- Larger values of the weights leads to smaller gradients.
- The gradient becomes more uniform, hence allowing the use of higher learning rates.

Experiments and Results

1) **Activations Over Time:**

MNIST Dataset, 28x28 images as input, 3 fully connected layers with 100 activations each, each hidden layer with 'Sigmoid' nonlinearity, with 'W' initialised with random gaussian values, last hidden layer followed by a fully connected layer with 10 activations and cross entropy loss. Network trained for 50,000 steps, with each mini-batch containing 60 examples.

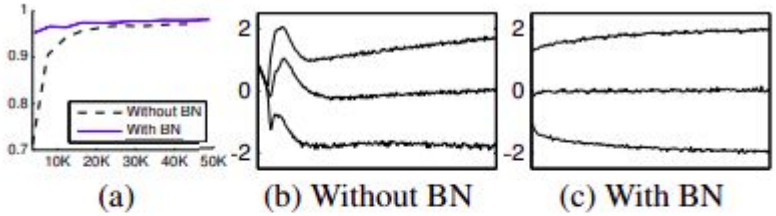

Figure 1. (a) *The test accuracy of the MNIST network trained with and without Batch Normalization, vs. the number of training steps. Batch Normalization helps the network train faster and achieve higher accuracy.* (b, c) *The evolution of input distributions to a typical sigmoid, over the course of training, shown as* $\{15, 50, 85\}$*th percentiles. Batch Normalization makes the distribution more stable and reduces the internal covariate shift.*

Experiments and Results

2) **ImageNet Classification**

ImageNet dataset, model with high number of convolutional and pooling layers, followed by a softmax classifier with 1000 classes, network has ~13.6x10^6 parameters (excluding the softmax layer). Model was trained on a distributed architecture, with 5 concurrent steps on each of 10 model replicas, using asynchronous SGD with momentum (mini-batch size of 32).

Some changes other than using batch normalisation:

- Increasing learning rate
- Remove Dropout
- Shuffle training examples more thoroughly
- Reduce the L2 weight regularization
- Accelerate the learning rate decay
- Remove local response normalisation
- Reduce the photometric distortions

Results

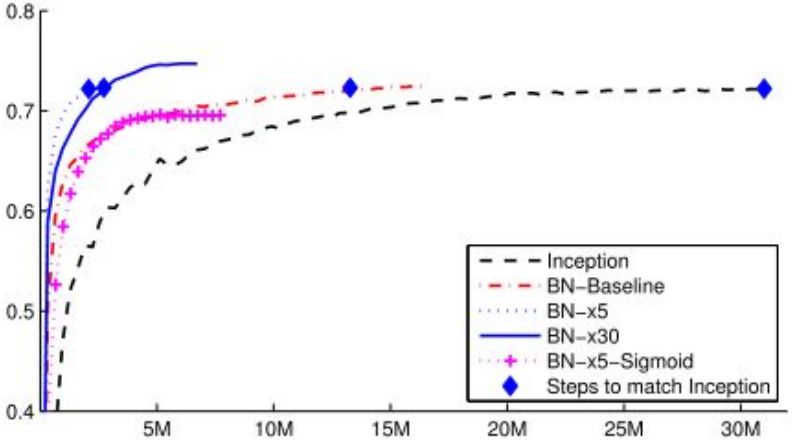

Figure 2. Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.

Model	Steps to 72.2%	Max accuracy
Inception	$31.0 \cdot 10^6$	72.2%
BN-Baseline	$13.3 \cdot 10^6$	72.7%
BN-x5	$2.1 \cdot 10^6$	73.0%
BN-x30	$2.7 \cdot 10^6$	74.8%
BN-x5-Sigmoid		69.8%

Figure 3. For Inception and the batch-normalized variants, the number of training steps required to reach the maximum accuracy of Inception (72.2%), and the maximum accuracy achieved by the network.

Conclusion

- Internal Covariate Shift complicates the training of machine learning algorithms.
- Normalization is appropriately handled by any optimization algorithm.
- Batch Normalization adds only two extra parameters.
- It also ensures a stable distribution of activations during training.

Thank You