Batch Normalization

Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe, Christian Szegedy, Google Inc.

Hong Cheng
chenghong@kw.ac.kr
Advisor: Prof. Hyukjoon Lee

Computer Intelligence Network Laboratory, Dept. of CE, KwangWoon University

CINe Lab

Content

- ❑ Introduction
- ❑ Towards Reducing Internal Covariate Shift
- ❑ Normalization via Mini-Batch Statistics
- ❑ Experiments
- ❑ Conclusion

1. Introduction

- **Stochastic Gradient Descent(SGD)**
  - Proved to be an effective way of training deep networks
  - SGD optimizes the parameters Θ of the network, so as to minimize the loss

$$\Theta = \arg\min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} \ell(\mathrm{x}_i, \Theta)$$

    - where $x_{1...N}$ is the training data set.
  - The training proceeds in steps, at each step we consider a minibatch $x_{1...m}$ of size m.
  - The mini-batch is used to **approximate** the gradient of the loss function with respect to the parameters, by computing

$$\frac{1}{m} \frac{\partial \ell(\mathrm{x}_i, \Theta)}{\partial \Theta}$$

  - Simple and effective
  - Requires careful tuning of the model hyper-parameters
    - the learning rate used in optimization
    - the initial values for the model parameters
  - Training is complicated
    - the inputs to each layer are affected by the parameters of all preceding layers

1.Introduction

- **Using mini-batches of examples**
  - as opposed to one example at a time
  - the gradient of the loss over a mini-batch is **an estimate of the gradient over the training set**
  - computation over a batch can be **much more efficient** than m computations for individual examples
- **Covariate shift and new distribution**
  - When **the input distribution** to a learning system **changes**, it is said to experience ***covariate shift***
  - The change in the distributions of layers' inputs presents a problem
    - The notion can be extended beyond the learning system as a whole, to apply to its parts such as a sub-network or a layer
    - Typically handled via **domain adaptation**
  - Because the layers need to continuously adapt to the new distribution.

1.Introduction

- **Consider a network computing**

$$\ell = F_2(F_1(\mathrm{u}, \Theta_1), \Theta_2)$$

  - $\mathbf{F_1}$ and $\mathbf{F_2}$ are arbitrary transformations
  - $\boldsymbol{\Theta_1}$, $\boldsymbol{\Theta_2}$ are to be learned so as to minimize the loss $\boldsymbol{\ell}$
  - Learning $\boldsymbol{\Theta_2}$ can be viewed as if the inputs $\mathbf{x} = \mathbf{F_1}(\boldsymbol{u}, \boldsymbol{\Theta_1})$ are fed into the sub-network

$$\ell = F_2(\mathrm{x}, \Theta_2)$$

  - A gradient descent step

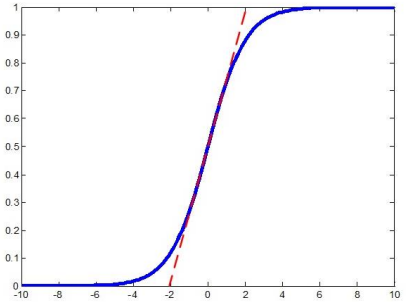
$$\Theta_2 \leftarrow \Theta_2 - \frac{\alpha}{m} \sum_{i=1}^{m} \frac{\partial F_2(\mathrm{x}_i, \Theta_2)}{\partial \Theta_2}$$

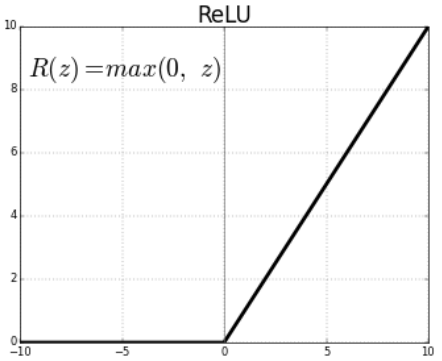
    - batch size $\boldsymbol{m}$ and learning rate $\boldsymbol{\alpha}$
  - The input distribution properties that **make training more efficient**
  - $\boldsymbol{\Theta_2}$ does not have to readjust to compensate for the change in the distribution of $\mathbf{x}$.

1.Introduction

- **Fixed distribution of inputs to a sub-network**
  - have **positive** consequences for the layers **outside** the sub-network
  - **Consider a layer**
    - with a **sigmoid activation function** $z = g(Wu + b)$
    - u is the **layer input**,
    - the **weight matrix** W and **bias vector** b are the layer parameters to be learned
  - This means that for those with small absolute values $x = Wu + b$, the gradient flowing down to u will vanish and the model will train slowly.
  - In practice, the **saturation problem** and the resulting vanishing gradients are usually addressed by using
    - Rectified Linear Units , $ReLU(x) = max(x, 0)$
    - Careful initialization
    - small learning rates

$$g(x) = \frac{1}{1+\exp(-x)}$$





2.Towards Reducing Internal Covariate Shift

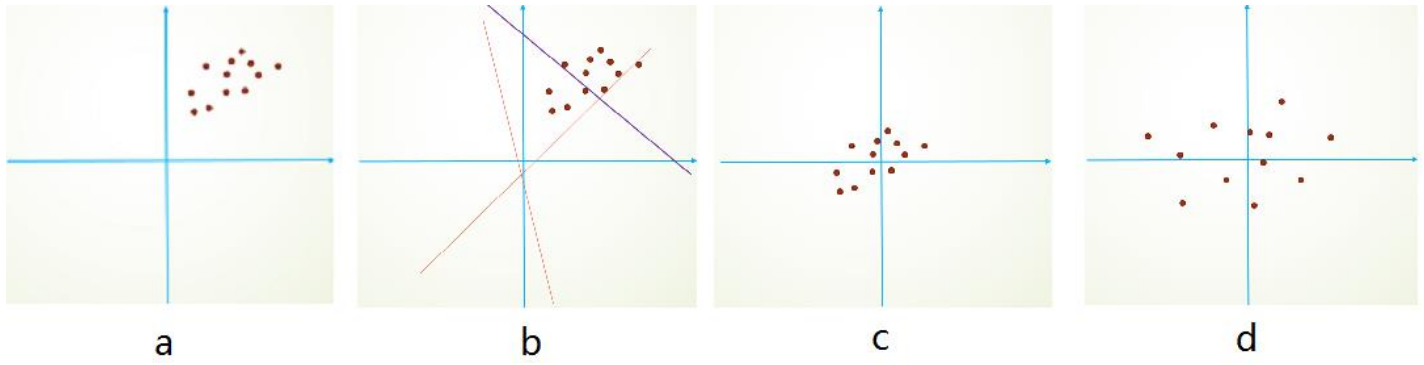
- ***Internal Covariate Shift***
  - **the change** in the distribution of network activations **due to** the ***change*** in network parameters during training.
  - Reduce this to improve the training by fixing the distribution of the layer inputs x as the training progresses
- **Whitening**
  - It has been long known (LeCun et al., 1998b; Wiesler & Ney, 2011) that the network training converges faster if its inputs are whitened
  - the network training **converges faster** if inputs are linearly transformed
    - zero means
    - unit variances
    - decorrelated

2.Towards Reducing Internal Covariate Shift

- **Whitening**

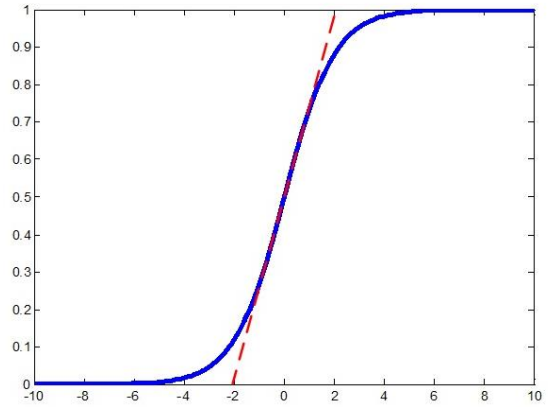
$$x_{PCAwhite,i} = \frac{x_{rot,i}}{\sqrt{\lambda_i + \varepsilon}}$$

- diagonal values λ_i



$$g(x) = \frac{1}{1+\exp(-x)}$$

- By whitening the inputs to each layer,
- we would take a step towards achieving the **fixed distributions** of inputs that would **remove the ill effects of the internal covariate shift**.



2.Towards Reducing Internal Covariate Shift

It is no accident that the diagonal values are λ_1 and λ_2. Further, the off-diagonal entries are zero; thus, $x_{\text{rot},1}$ and $x_{\text{rot},2}$ are uncorrelated, satisfying one of our desiderata for whitened data (that the features be less correlated).

To make each of our input features have unit variance, we can simply rescale each feature $x_{\text{rot},i}$ by $1/\sqrt{\lambda_i}$. Concretely, we define our whitened data $x_{\text{PCAwhite}} \in \Re^n$ as follows:

$$x_{\text{PCAwhite},i} = \frac{x_{\text{rot},i}}{\sqrt{\lambda_i}}.$$

ZCA Whitening

Finally, it turns out that this way of getting the data to have covariance identity I isn't unique. Concretely, if R is any orthogonal matrix, so that it satisfies $RR^T = R^T R = I$ (less formally, if R is a rotation/reflection matrix), then $R\,x_{\text{PCAwhite}}$ will also have identity covariance. In **ZCA whitening**, we choose $R = U$. We define

$$x_{\text{ZCAwhite}} = U x_{\text{PCAwhite}}$$

2.Towards Reducing Internal Covariate Shift

When implementing PCA whitening or ZCA whitening in practice, sometimes some of the eigenvalues λ_i will be numerically close to 0, and thus the scaling step where we divide by $\sqrt{\lambda_i}$ would involve dividing by a value close to zero; this may cause the data to blow up (take on large values) or otherwise be numerically unstable. In practice, we therefore implement this scaling step using a small amount of regularization, and add a small constant ϵ to the eigenvalues before taking their square root and inverse:

$$x_{\text{PCAwhite},i} = \frac{x_{\text{rot},i}}{\sqrt{\lambda_i + \epsilon}}.$$

When x takes values around $[-1, 1]$, a value of $\epsilon \approx 10^{-5}$ might be typical.

For the case of images, adding ϵ here also has the effect of slightly smoothing (or low-pass filtering) the input image. This also has a desirable effect of removing aliasing artifacts caused by the way pixels are laid out in an image, and can improve the features learned (details are beyond the scope of these notes).

2.Towards Reducing Internal Covariate Shift

- **Whitening with the optimization steps**
  - the gradient descent step may attempt to update the parameters in a way that requires the normalization to be updated
  - reduces the effect of the gradient step.
  - **consider a layer**
    - with the input u that adds the learned bias b
    - Normalizes the result by subtracting the mean of the activation computed over the training data:

$$\hat{x} = x - E[x] \quad x = u + b \quad \boldsymbol{X} = \{x_{1\ldots N}\} \quad E[x] = \frac{1}{N}\sum_{i=1}^{N} x_i$$

    - If a gradient descent step ignores the dependence of $E[x]$ on b
    - It update $b \leftarrow b + \Delta b \quad \Delta b \propto -\partial \ell / \partial \hat{x}$
    - Then $u + (b + \Delta b) - E[u + (b + \Delta b)] = u + b - E[u + b]$
    - Consequently, the combination of the update to b and subsequent change in normalization led to **no change in the output of the layer nor the loss**.
    - **As the training continues, b will grow indefinitely while the loss remains fixed**.

2.Towards Reducing Internal Covariate Shift

- **The issue with the above approach**
  - Gradient descent optimization does not take into account the fact that the normalization takes place
    - To address this issue, for any parameter values, network always produces activations with **desired distribution**.
    - allow the gradient of the loss with respect to the model parameters to account for the normalization, and for its dependence on the model parameters Θ
- **Then the normalization**

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

  - Let again x be a layer input, treat as a vector
  - X be the set of these input over the training data set
    - Which depends not only on the given training example x but on all example X
    - Each of X depends on Θ if x is generated by another layer.

2.Towards Reducing Internal Covariate Shift

- **For backpropagation**
  - compute the **Jacobians**

$$\frac{\partial \text{Norm}(\text{x}, \mathcal{X})}{\partial \text{x}} \quad \text{and} \quad \frac{\partial \text{Norm}(\text{x}, \mathcal{X})}{\partial \mathcal{X}}$$

  - Within this framework, whitening the layer inputs is **expensive**
    - as it requires computing the covariance matrix and its inverse square root
    - to produce the whitened activations

$$\text{Cov}[\text{x}] = \text{E}_{\text{x}\in\mathcal{X}}[\text{x}\text{x}^T] - \text{E}[\text{x}]\text{E}[\text{x}]^T \qquad \text{Cov}[\text{x}]^{-1/2}(\text{x} - \text{E}[\text{x}])$$

  - This motivates us to seek an alternative that performs input normalization in a way
    - that is differentiable
    - does not require the analysis of the entire training set after every parameter update.

3.Normalization via Mini-Batch Statistics

▪ Two Necessary Simplifications

- ❖ Since the full whitening of each layer's inputs is costly and not everywhere differentiable
- ▪ **Each mini-batch produces estimates of the mean and variance of each activation**
  - ▪ Since we use mini-batches in stochastic gradient training
- ▪ **Normalize each scalar feature independently**
  - ▪ By making it have the **mean of zero** and the **variance of 1**
  - ▪ **Instead of** ~~whitening the features in layer inputs and outputs jointly~~
- ▪ For a layer with d-dimensional input $x = (x^{(1)} \ldots x^{(d)})$, we will normalize each dimension

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

  - ▪ **Expectation and variance** are computed over the **training data set**
- ▪ As shown in (LeCun et al., 1998b), such normalization speeds up convergence, even when the features are not decorrelated.

3.Normalization via Mini-Batch Statistics

- **Normalize each scalar feature independently**
  - Simply normalizing each input of a layer may change what the layer can represent.
    - For instance, normalizing the inputs of a sigmoid would constrain them to the linear regime of the nonlinearity.
  - To address this, we make sure that
  - ***The transformation inserted in the network can represent the identity transform***
  - For each activation $x^{(k)}$, a pair of parameters $\gamma^{(k)}$, $\beta^{(k)}$,
    - **Scale and shift the normalized value:**

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

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

  - These parameters are learned along with the **original model parameters**, and restore the representation power of the network.
  - **If that were the optimal thing to do, we could recover the original activations**

3.Normalization via Mini-Batch Statistics

- **Use mini-batches in stochastic gradient training**
  - In the batch setting we would use the **whole set** to normalize activations.
  - However, this is **impractical** when using stochastic optimization
  - Therefore, we make the second simplification
  - ***Each mini-batch produces estimates of the mean and variance of each activation***
  - The statistics used for normalization can **fully participate in** the gradient backpropagation.
- The use of mini-batches
  - Enabled by computation of **per-dimension variances** rather than **joint covariances**
  - In the joint case, regularization would be required
    - Since the mini-batch size is likely to be smaller than the number of activations being whitened, resulting in singular covariance matrices.

3.Normalization via Mini-Batch Statistics

- **BN Transform Algorithm1**
  - Consider a mini-batch **B** of size **m**.
    - Since the normalization is applied to each activation independently
    - let us focus on a particular activation ***x(k)*** and omit **k** for clarity.
  - We have ***m*** values of this activation in the mini-batch, $\mathcal{B} = \{x_{1\ldots m}\}$
  - Normalized values $\hat{x}_{1\ldots m}$
  - Linear transformations $y_{1\ldots m}$

$$\mathrm{BN}_{\gamma,\beta} : x_{1\ldots m} \rightarrow y_{1\ldots m}$$

    - ϵ is a constant added to the mini-batch variance for numerical stability

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.

3.Normalization via Mini-Batch Statistics

BN Transform Algorithm1

$$y = \mathrm{BN}_{\gamma,\beta}(x)$$

- can be added to a network to **manipulate any activation**
- The parameter γ and β are **to be learned**
- BN transform does not independently process the activation in each training example
- $\mathrm{BN}_{\gamma,\beta}(x)$ depends both on the training example and the other examples in the mini-batch
- The scaled and shifted values ***y*** are passed to other network layers.

- The distributions of values of any $\hat{x}$ has the expected value of ***0*** and the variance of ***1***,

$$\sum_{i=1}^{m} \widehat{x}_i = 0 \text{ and } \frac{1}{m}\sum_{i=1}^{m} \widehat{x}_i^2 = 1$$

- Each normalized activation $\widehat{x}^{(k)}$ can be viewed as an input to a sub-network composed of the linear transform $y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$ followed by the other processing done by the original network.

3.Normalization via Mini-Batch Statistics

- **BN Transform Algorithm1**

  - With **chain rule**

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i-\mu_{\mathcal{B}})^2$$

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

$$y_i \leftarrow \gamma\widehat{x}_i+\beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$$

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

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

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

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

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

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

3.Normalization via Mini-Batch Statistics

3.1 Training and Inference(test) with Batch-Normalized Networks

- **The insert of BN transform**

$$x \rightarrow BN(x)$$

  - Then train model with SGD or any of variants
  - The normalization of activations that depends on the mini-batch allows efficient training,
  - neither necessary nor desirable during inference

- **In test**
  - normalization $\widehat{x} = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}}$
  - Unbiased variance estimate $\mathrm{Var}[x] = \frac{m}{m-1} \cdot \mathrm{E}_{\mathcal{B}}[\sigma^2_{\mathcal{B}}]$
  - Where the expectation is over training mini-batches of size ***m*** and σ^2_B are their sample variances.

- **Train Batch-Normalized Network Algorithm 2**

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

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

1: $N_{\mathrm{BN}}^{\mathrm{tr}} \leftarrow N$ // Training BN network

2: **for** $k = 1 \ldots K$ **do**

3: Add transformation $y^{(k)} = \mathrm{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\mathrm{BN}}^{\mathrm{tr}}$ (Alg. 1)

4: Modify each layer in $N_{\mathrm{BN}}^{\mathrm{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead

5: **end for**

6: Train $N_{\mathrm{BN}}^{\mathrm{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$

7: $N_{\mathrm{BN}}^{\mathrm{inf}} \leftarrow N_{\mathrm{BN}}^{\mathrm{tr}}$ // 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:

$$\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$

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

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

12: **end for**

3.Normalization via Mini-Batch Statistics

3.2 Batch-Normalized Convolutional Network

- **Affine transformation followed by element-wise nonlinearity**

$$z = g(Wu + b)$$

  - where W and b are learned parameters of the model
  - $g(\cdot)$ is nonlinearity such as sigmoid or ReLU
  - For both fully-connected and convolutional layer

  - add BN transform immediately before the nonlinearity, by normalizing $x = Wu + b$.
    - x is more likely to have a symmetric, non-sparse distribution, that is "more Gaussian" (Hyv¨arinen & Oja, 2000);
    - normalizing it is likely to produce activations with a stable distribution
  - bias b can be ignored
    - its effect will be canceled by the subsequent mean subtraction (the role of the bias is subsumed by β in **Alg. 1**).
  - BN transform is applied independently to each dimension of $x = Wu$, with a separate pair of learned parameters $\gamma(k)$, $\beta(k)$ per dimension.

$$z = g(Wu + b) \quad \Rightarrow \quad z = g(\mathrm{BN}(Wu))$$

3.Normalization via Mini-Batch Statistics

3.3 Batch-Normalized Convolutional Network

- **BN for convolutional layer**
  - Additionally, we want the normalization to obey the convolutional property
    - different elements of the same feature map, at different locations, are normalized in the same way
  - jointly normalize all the activations in a mini-batch, over all locations
  - **For Alg. 1**, let B be the set of all values in a feature map across both the elements of a mini-batch and spatial locations
  - for a mini-batch of size m and feature maps of size $p \times q$
  - use the effective mini-batch of size $m' = |B| = m \cdot p\ q$
  - We learn a pair of parameters $\gamma(k)$ and $\beta(k)$ per feature map, rather than per activation.

  - **Alg. 2** is modified similarly
  - during **inference** the BN transform applies the same linear transformation to each activation in a given feature map.

3.Normalization via Mini-Batch Statistics

- **3.3 Batch Normalization enables higher learning rates**

- **Normalizing activations throughout the network**
  - prevents small changes to the parameters from amplifying into larger and suboptimal changes in activations in gradients
    - for instance, it prevents the training from getting stuck in the saturated regimes of nonlinearities.

- **Makes training more resilient to the parameter scale**
  - Backpropagation through a layer is unaffected by the scale of its parameters.
  - for a scalar a

$$\mathrm{BN}(W\mathrm{u}) = \mathrm{BN}((aW)\mathrm{u})$$

$$\frac{\partial \mathrm{BN}((aW)\mathrm{u})}{\partial \mathrm{u}} = \frac{\partial \mathrm{BN}(W\mathrm{u})}{\partial \mathrm{u}}$$

$$\frac{\partial \mathrm{BN}((aW)\mathrm{u})}{\partial (aW)} = \frac{1}{a} \cdot \frac{\partial \mathrm{BN}(W\mathrm{u})}{\partial W}$$

  - The scale does not affect the layer Jacobian nor, consequently, the gradient propagation.
  - Moreover, larger weights lead to smaller gradients
  - stabilize the parameter growth.

3.Normalization via Mini-Batch Statistics

- **3.3 Batch Normalization enables higher learning rates**
- **Conjecture: BN may lead the layer Jacobians to have singular values close to *1***
  - Consider two consecutive layers with normalized inputs
  - the transformation between these normalized vectors: $\hat{z} = F(\hat{x})$
  - If we assume that $\hat{x}$ and $\hat{z}$ are **Gaussian and uncorrelated**
  - $F(\hat{x}) \approx J\hat{x}$ is a linear transformation for the given model parameters
  - then both $\hat{x}$ and $\hat{z}$ have unit covariances

$$I = \mathrm{Cov}[\hat{z}] = J\mathrm{Cov}[\hat{x}]J^T = JJ^T. \text{ Thus, } JJ^T = I$$

  - so all singular values of ***J*** are equal to ***1***
    - preserves the gradient magnitudes during backpropagation.

3.Normalization via Mini-Batch Statistics

3.4 Batch Normalization regularizes the model

Estimate Value - Generalization - Dropout

- When training, **the example is seen in conjunction with other examples** in the mini-batch
- the training network **no longer producing deterministic values** for a given training example

- In the experiments, this effects is found to be **advantageous to the generalization** of the network

- Whereas **dropout** is typically used to **reduced overfitting**
- In the BN network, dropout can be **either removed or reduced in strength**.

4.Experiments

4.1 Activations over time

- **Predict the digit class on the MNIST dataset**
  - a 28x28 binary image as input
  - 3 fully-connected hidden layers with 100 activations each
    - Each hidden layer computes y = g(Wu+b) with sigmoid nonlinearity
    - weights W initialized to small random Gaussian values
    - The last hidden layer is followed by a fully-connected layer with 10 activations (one per class) and cross-entropy loss.
  - trained the network for 50000 steps, with 60 examples per mini-batch
  - Added Batch Normalization to each hidden layer of the network, as in Sec. 3.1.

4.Experiments

4.1 Activations over time

- **Predict on the MNIST dataset**

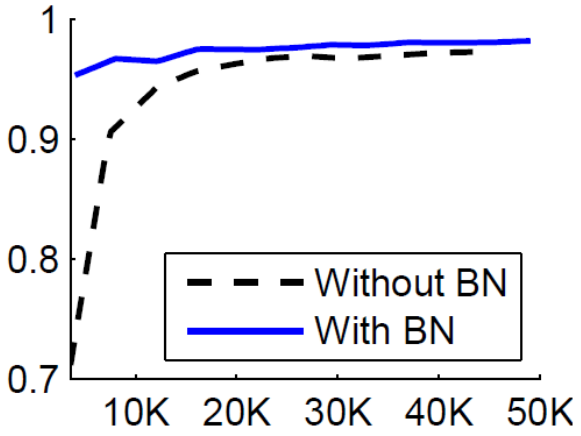


Fig 1(a)

The test accuracy of the MNIST network

Batch Normalization helps the network train faster and achieve higher accuracy

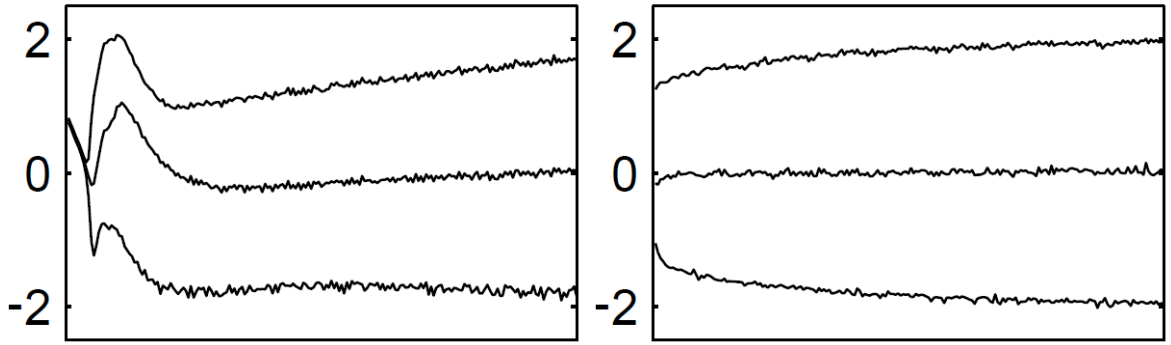


Fig 1 (b) (c)

The evolution of input distributions to a typical sigmoid, over the course of training, $\{15, 50, 85\}_{th}$ ***percentiles***

Batch Normalization makes the distribution more stable and reduces the internal covariate shift.

4.Experiments

- ## 4.2 ImageNet classification
  - ### 4.2.1 Accelerating BN Networks
    - **Simply adding Batch Normalization to a network does not take full advantage of our method.**
    - changed the network and its training parameters, as follows:
      - **Increase learning rate.**
      - **Remove Dropout.**
      - **Reduce the L2 weight regularization.**
      - **Accelerate the learning rate decay.**
      - **Remove Local Response Normalization**
      - **Shuffle training examples more thoroughly.**
      - **Reduce the photometric distortions.**

4.Experiments

- ## 4.2 ImageNet classification
  - We applied **Batch Normalization** to a new variant of the **Inception network** trained on the **ImageNet classification task**
    - a large number of **convolutional** and **pooling layers**
    - a **softmax** layer to predict the image class, out of 1000 possibilities.
    - Convolutional layers use **ReLU** as the nonlinearity
    - contains $13.6 \cdot 10^6$ parameters
    - other than the top softmax layer, has **no fully-connected layers**
    - More detail are given in the **Appendix**
  - We refer to this model as ***Inception*** in the rest of the text.
    - Model was trained using Stochastic Gradient Descent with **momentum**, using the mini-batch size of **32**
    - using a large-scale, distributed architecture

4.Experiments

- ## 4.2 ImageNet classification

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	double #3×3 reduce	double #3×3	Pool +proj
convolution*	7×7/2	112×112×64	1						
max pool	3×3/2	56×56×64	0						
convolution	3×3/1	56×56×192	1		64	192			
max pool	3×3/2	28×28×192	0						
inception (3a)		28×28×256	3	64	64	64	64	96	avg + 32
inception (3b)		28×28×320	3	64	64	96	64	96	avg + 64
inception (3c)	stride 2	28×28×576	3	0	128	160	64	96	max + pass through
inception (4a)		14×14×576	3	224	64	96	96	128	avg + 128
inception (4b)		14×14×576	3	192	96	128	96	128	avg + 128
inception (4c)		14×14×576	3	160	128	160	128	160	avg + 128
inception (4d)		14×14×576	3	96	128	192	160	192	avg + 128
inception (4e)	stride 2	14×14×1024	3	0	128	192	192	256	max + pass through
inception (5a)		7×7×1024	3	352	192	320	160	224	avg + 128
inception (5b)		7×7×1024	3	352	192	320	192	224	max + 128
avg pool	7×7/1	1×1×1024	0						

4.Experiments

4.2 ImageNet classification

4.2.1 Accelerating BN Networks

- **Simply adding Batch Normalization to a network does not take full advantage of our method.**
- changed the network and its training parameters, as follows:
  - **Increase learning rate.**
    - In a batch-normalized model, we have been able to achieve a training speedup from higher learning rates, with no ill side effects (Sec. 3.3).
  - **Remove Dropout.**
    - As described in Sec. 3.4, Batch Normalization fulfills some of the same goals as Dropout. Removing Dropout from Modified BN-Inception speeds up training, without increasing overfitting.
  - **Reduce the L2 weight regularization.**
    - While in Inception an L2 loss on the model parameters controls overfitting, in Modified BN-Inception the weight of this loss is reduced by a factor of 5. We find that this improves the accuracy on the held-out validation data.
  - **Accelerate the learning rate decay.**
    - In training Inception, learning rate was decayed exponentially. Because our network trains faster than Inception, we lower the learning rate 6 times faster.

4.Experiments

4.2 ImageNet classification

4.2.1 Accelerating BN Networks

- **Simply adding Batch Normalization to a network does not take full advantage of our method.**
- changed the network and its training parameters, as follows:
  - **Remove Local Response Normalization**
    - While Inception and other networks (Srivastava et al., 2014) benefit from it, we found that with Batch Normalization it is not necessary.
  - **Shuffle training examples more thoroughly.**
    - We enabled within-shard shuffling of the training data, which prevents the same examples from always appearing in a mini-batch together. This led to about 1% improvements in the validation accuracy, which is consistent with the view of Batch Normalization as a regularizer (Sec. 3.4): the randomization inherent in our method should be most beneficial when it affects an example differently each time it is seen.
  - **Reduce the photometric distortions.**
    - Because batch-normalized networks train faster and observe each training example fewer times, we let the trainer focus on more “real” images by distorting them less.

Why Batch-Normalization?

- **lateral inhibition**
  - In neurobiology, there is a concept called “lateral inhibition”.
  - Now what does that mean?
    - This refers to the **capacity** of an excited neuron to subdue its neighbors. We basically want a significant peak so that we have a form of local maxima.
    - This tends to create a contrast in that area, hence increasing the sensory perception. Increasing the sensory perception is a good thing!
- **LRN layer**
  - **Local Response Normalization (LRN) layer** implements the lateral inhibition
  - This layer is useful when we are dealing with **ReLU neurons**. Why is that?
    - Because ReLU neurons have unbounded activations and we need LRN to normalize that.
    - We want to detect high frequency features with a large response.
    - If we normalize around the local neighborhood of the excited neuron, it becomes even more sensitive as compared to its neighbors.
    - At the same time, it will dampen the responses that are uniformly large in any given local neighborhood. If all the values are large, then normalizing those values will diminish all of them.
    - So basically we want to encourage some kind of inhibition and boost the neurons with relatively larger activations.

4.Experiments

- ## 4.2 ImageNet classification
  - ### 4.2.2 Single-Network Classification
    - evaluated the following networks
    - all trained on the LSVRC2012 training data, and tested on the validation data:
      - **Inception**: the network described at the beginning of Section 4.2, trained with the initial learning rate of 0.0015.
      - **BN-Baseline**: Same as Inception with Batch Normalization before each nonlinearity.
      - **BN-x5**: Inception with Batch Normalization and the modifications in Sec. 4.2.1. The initial learning rate was increased by a factor of 5, to 0.0075. The same learning rate increase with original Inception caused the model parameters to reach machine infinity.
      - **BN-x30**: Like BN-x5, but with the initial learning rate 0.045 (30 times that of Inception).
      - **BN-x5-Sigmoid**: Like BN-x5, but with sigmoid nonlinearity $g(t) = \frac{1}{1+\exp(-x)}$ instead of ReLU. We also attempted to train the original Inception with sigmoid, but the model remained at the accuracy equivalent to chance.

4.Experiments

- ## 4.2 ImageNet classification
  - ### 4.2.2 Single-Network Classification

Increasing the learning rate further (BN-x30) causes the model to train somewhat slower initially, but allows it to reach a higher final accuracy.

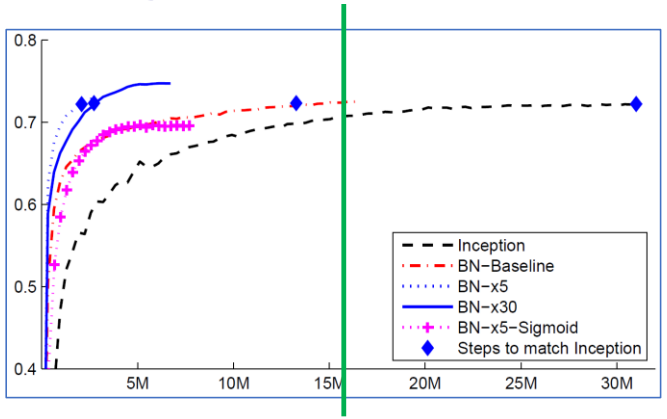


Figure 2: Single crop validation accuracy of Inception and its BN variants, vs. the number of training steps.

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%

half

1/14

Figure 3: the number of training steps required to reach the maximum accuracy of Inception and the maximum accuracy achieved by the network.

- In Figure 2, we show the validation accuracy of the networks, as a function of the number of training steps. Inception reached the accuracy of 72.2% after $31 \cdot 10^6$ training steps.
- The Figure 3 shows, for each network, the number of training steps required to reach the same 72.2% accuracy, as well as the maximum validation accuracy reached by the network and the number of steps to reach it.

4.Experiments

4.2 ImageNet classification

4.2.3 Ensemble Classification

- The current reported best results on the ImageNet Large Scale Visual Recognition Competition are reached by the Deep Image ensemble of traditional models (Wu et al., 2015) and **the ensemble model of (He et al., 2015).**
- The latter reports the **top-5 error** of **4.94%**, as evaluated by the ILSVRC server.
- Here we report a top-5 validation error of **4.9%**, and test error of **4.82%** (according to the ILSVRC server).
- This improves upon the previous best result, and exceeds the estimated accuracy of human raters according to (Russakovsky et al., 2014).

4.Experiments

4.2 ImageNet classification

4.2.3 Ensemble Classification

- For our ensemble, we used **6 networks based on BN-x30**, modified by following:
  - Increased initial weights in the convolutional layers;
  - using Dropout (with the Dropout probability of 5% or 10%, vs. 40% for the original Inception);
  - using non-convolutional, per-activation Batch Normalization with last hidden layers of the model.
- Each network achieved its maximum accuracy after about $6 \cdot 10^6$ training steps.
  - The ensemble prediction was based on the **arithmetic average** of class probabilities predicted by the constituent networks.
  - The details of ensemble and multi-crop inference are similar to (Szegedy et al., 2014).

Model	Resolution	Crops	Models	Top-1 error	Top-5 error
GoogLeNet ensemble	224	144	7	-	6.67%
Deep Image low-res	256	-	1	-	7.96%
Deep Image high-res	512	-	1	24.88	7.42%
Deep Image ensemble	variable	-	-	-	5.98%
BN-Inception single crop	224	1	1	25.2%	7.82%
BN-Inception multicrop	224	144	1	21.99%	5.82%
BN-Inception ensemble	224	144	6	20.1%	**4.9%***

Figure 4: Batch-Normalized Inception **comparison with previous** state of the art on the provided validation set comprising 50000 images.
*BN-Inception ensemble has reached 4.82% top-5 error on the 100000 images of the test set of the ImageNet as reported by the test server.

- We demonstrate in **Fig. 4** that batch normalization allows us to set new state-of-the-art by a healthy margin on the ImageNet classification challenge benchmarks.

5.Conclusion

- Our proposed method draws its power from normalizing activations. This ensures that the normalization is appropriately handled by **any optimization method** that is being used to train the network.
- To enable stochastic optimization methods commonly used in deep network training, we perform the normalization for each mini-batch, and backpropagate the gradients through the normalization parameters.
- Batch Normalization adds only two extra parameters per activation, and in doing so preserves the representation ability of the network.
- The resulting networks can be trained with saturating nonlinearities, are more tolerant to increased training rates, and often do not require Dropout for regularization.

5.Conclusion

- In this work, we have not explored the full range of possibilities that Batch Normalization potentially enables.
- Our future work includes applications of our method to **Recurrent Neural Networks** (Pascanu et al., 2013), where the internal covariate shift and the vanishing or exploding gradients may be especially severe, and which would allow us to more thoroughly **test the hypothesis** that normalization improves gradient propagation (**Sec. 3.3**).
- We plan to investigate whether Batch Normalization can help with **domain adaptation**, in its traditional sense – i.e. whether the normalization performed by the network would allow it to more easily generalize to new data distributions, perhaps with just a recomputation of the population means and variances (Alg. 2).

Thank you

Computer Intelligence Network Laboratory, Dept. of CE, KwangWoon University