Batch Normalization

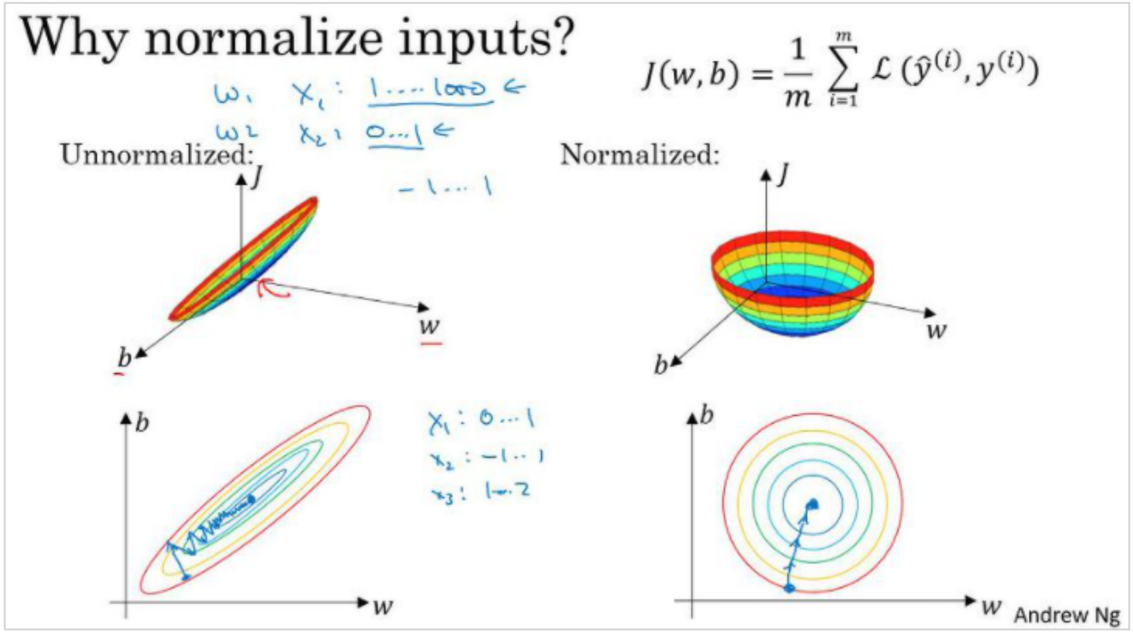
Google
(ICML 2015)

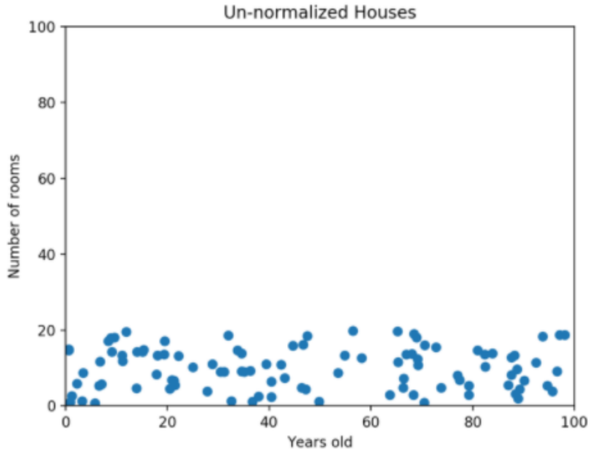
이준형

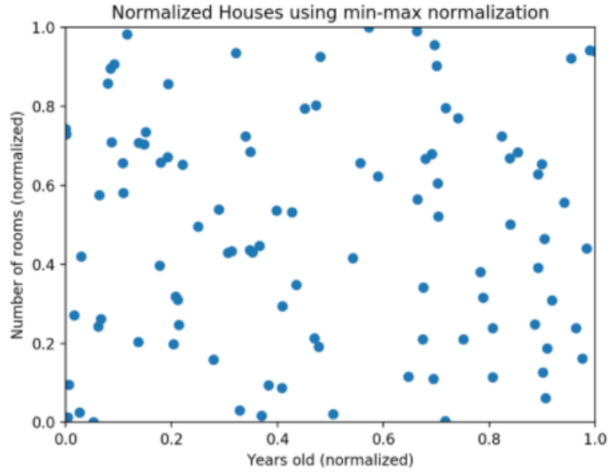
Overview

- Data Normalization
- Introduction
- Internal Covariate Shift
- Training and Inference with Batch Normalization
- Experiment
- Conclusion

Data Normalization







Optimization

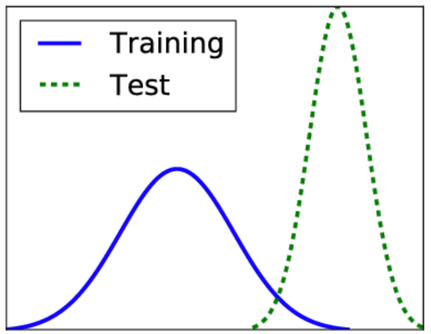
Problem

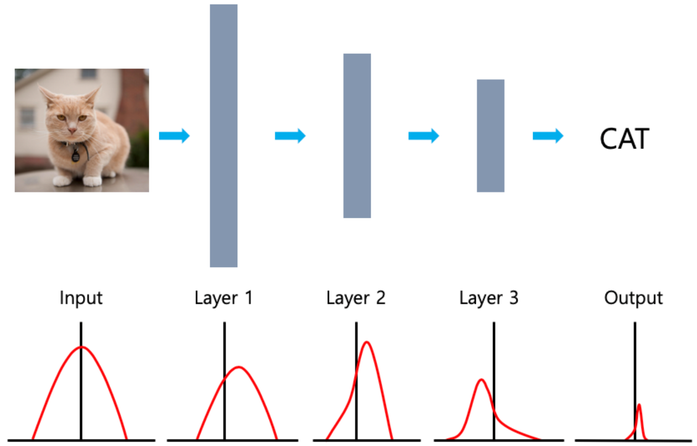
- Gradient Vanishing
- Gradient Exploding

Solution

- Change activation function
- Careful initialization
- Small learning rate

Internal Covariate Shift

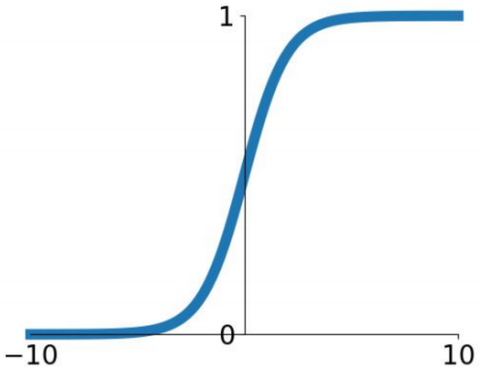




- The change in the distributions of layers' inputs presents a problem because the layer need to continuously adapt to the new distribution. When the input distribution to a learning system changes, it is said to experience **covariate shift**
- In practice, the saturation problem and the resulting vanishing gradients are usually addressed by using ReLU, careful initialization and small learning rates
- We define Internal Covariant Shift as the change in the distributions of network activations due to the change in network parameters during training.

Whitening

- Whitening of each layer’s input is costly and not everywhere differentiable. (covariance matrix and inverse computation)
- Instead of whitening the features in layer inputs and outputs jointly, we will normalize each scalar **feature independently** by making it have the **mean of zero and the variance of 1**.
- 0에서 멀어질 경우 gradient가 0에 가깝게 saturated -> Gradient Vanishing



Sigmoid

Batch Normalization - Benefits

1. 학습 속도(training speed)를 빠르게 할 수 있다
2. 가중치 초기화(weight initialization)에 대한 민감도를 감소시킨다
3. 모델의 일반화(regularization) 효과가 있다.

Batch Normalization - Intro

- Training DNN is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change.

- This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities.
  -> Internal Covariate Shift

- Our method draws its strength from **making normalization a part of the model architecture** and performing the **normalization for each training mini-batch**.

- In this paper, Batch Normalization allows us to use **much higher learning rates** and be careful about initialization. It also acts as a regularizer, **in some cases eliminating the need for Dropout**.

Batch Normalization - Mini Batch

- First, the gradient of the loss over a mini-batch is an **estimate of the gradient over the training set**, whose **quality improves as the batch size increases**

- Second, computation over a batch can be much more efficient than *m* computations for individual examples, due to the **parallelism** afforded by the modern computing platforms.

Batch Normalizing Transform

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

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

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

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

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

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

Training and Inference with Batch Norm Networks

- Using moving averages instead, we can track the accuracy of a model as it trains.
- Since the means and variances are fixed during inference, the normalization is simply a linear transform applied to each activation

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

입력값	100	110	130	120	140
이동평균	100	105	120	125	130

Experiment Results

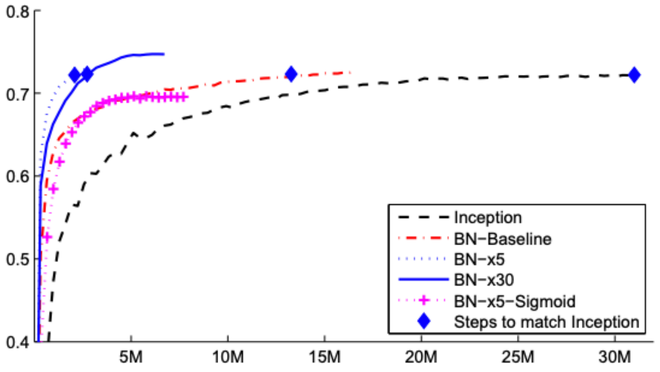


Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

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.*

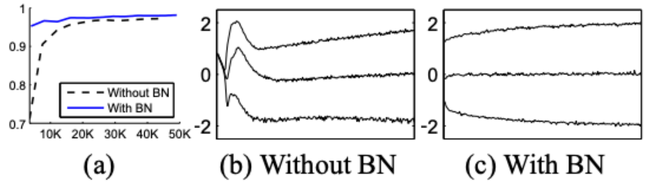


Figure 1: (a) *The test accuracy of the MNIST network trained with and without Batch Normalization, vs. the number of training steps. Batch Normalization helps the network train faster and achieve higher accuracy.* (b, c) *The evolution of input distributions to a typical sigmoid, over the course of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

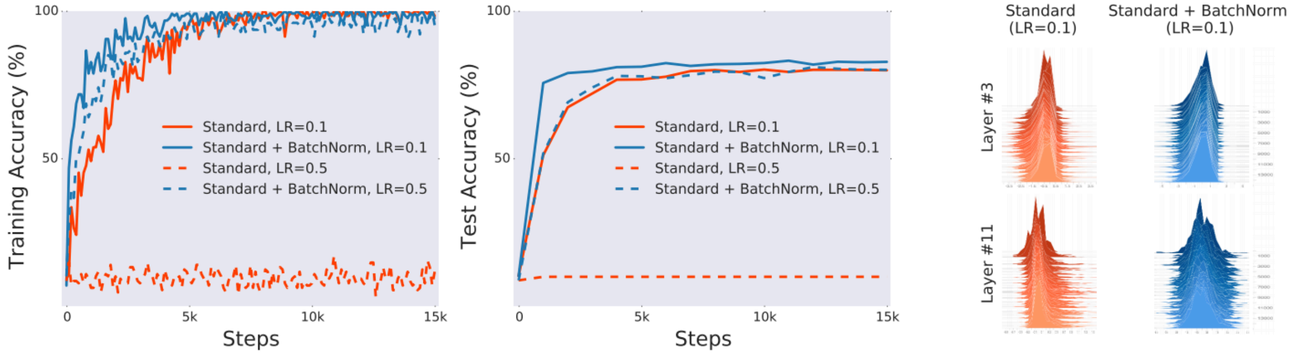
Model	Resolution	Crops	Models	Top-1 error	Top-5 error
GoogLeNet ensemble	224	144	7	-	6.67%
Deep Image low-res	256	-	1	-	7.96%
Deep Image high-res	512	-	1	24.88	7.42%
Deep Image ensemble	variable	-	-	-	5.98%
BN-Inception single crop	224	1	1	25.2%	7.82%
BN-Inception multicrop	224	144	1	21.99%	5.82%
BN-Inception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

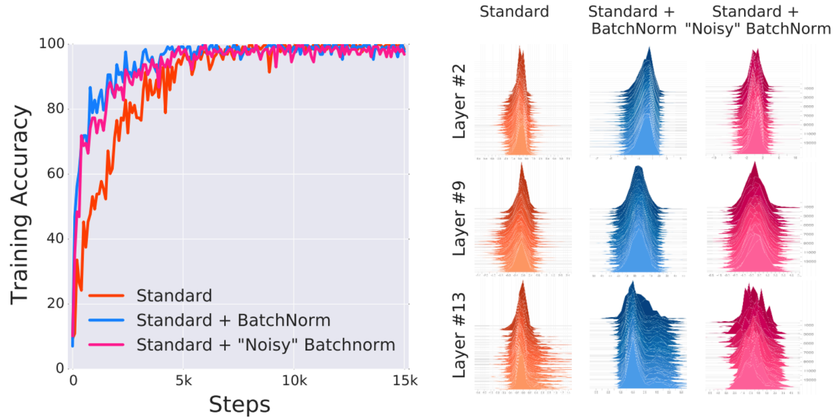
Figure 4: *Batch-Normalized Inception comparison with previous state of the art on the provided validation set comprising 50000 images. *BN-Inception ensemble has reached 4.82% top-5 error on the 100000 images of the test set of the ImageNet as reported by the test server.*

How Does Batch Normalization Help Optimization?
MIT
(2019)

- Despite its pervasiveness, the exact reasons for BatchNorm's effectiveness are still poorly understood.
- The popular belief reduce the so-called "Internal Covariate Shift"…In this work we demonstrate that such distributional stability of layer inputs has little to do with the success of BatchNorm.
- Instead, we uncover a more fundamental impact of BatchNorm on the training process: **it make the optimization landscape significantly smoother**. This smoothness induces a more predictive and stable behavior of the gradients, allowing for faster training.
- "BatchNorm might not even be reducing internal covariate shift"
- We prove that, under natural conditions, the Lipschitzness of both the loss and the gradients are improved in models with BatchNorm.
- The first manifestation of this impact is improvement in the Lipschitzness of the loss function. That is, the loss changes at a smaller rate and the magnitudes of the gradients are smaller too.

How Does Batch Normalization Help Optimization?





Optimization landscape is significantly smoother
-> gradient are more predictive