Lecture 5

Dropout, Batch normalization

What is dropout?

- Dropout is one of the techniques for preventing overfitting in deep neural network which contains a large number of parameters.

Original Paper

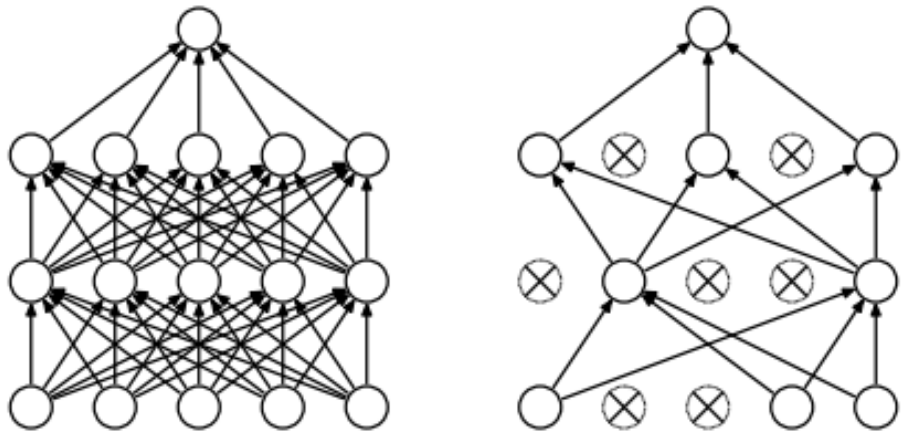
- Title:
  - Dropout: A Simple Way to Prevent Neural Networks from Overfitting (2014).
- Authors:
  - Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov
- Organization:
  - Department of Computer Science, University of Toronto
- URL:
  - https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

Overview

- The key idea is to randomly drop units from the neural network during training.

- During training, dropout samples from number of different “thinned” network.

- At test time, we approximate the effect of averaging the predictions of all these thinned networks.



(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Model

- Consider a neural network with L hidden layer.
- Let $\mathbf{z}^{(l)}$ denote the vector inputs into layer l
- $\mathbf{y}^{(l)}$ denote the vector of outputs from layer l.
- $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weights and biases at layer l. With dropout, the feed-forward operation becomes:

Model

$r_j^{(l)} \sim Bernoulli(p)$

$\tilde{\mathbf{y}}^{(l)} = \mathbf{r}^{(l)} \odot \mathbf{y}^{(l)}$, here $\odot$ denotes an element-wise product.

$z_i^{(l+1)} = \mathbf{w}_i^{(l+1)} \tilde{\mathbf{y}}^l + b_i^{(l+1)}$

$y_i^{(l+1)} = f(z_i^{(l+1)})$, where f is the activation function.

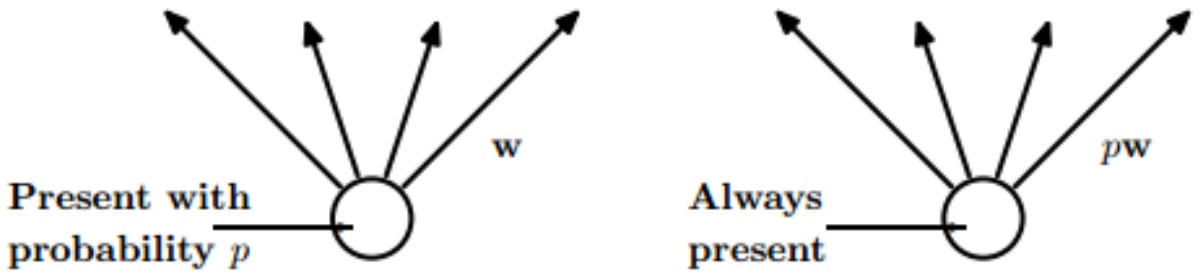
For any layer l, $\mathbf{r}^{(l)}$ is a vector of independent Bernoulli random variables each of which has probability of p of being 1. $\tilde{\mathbf{y}}$ is the input after we drop some hidden units. The rest of the model remains the same as the regular feed-forward neural network.

Training

- Dropout neural network can be trained using stochastic gradient descent.
- The only difference here is that we only back propagate on each thinned network.
- The gradient for each parameter are averaged over the training cases in each mini-batch.

Test Time

- use a single neural net without dropout.
- If a unit is retained with probability p during training, the outgoing weights of that unit are multiplied by p at test time.



(a) At training time (b) At test time

Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights **w**. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

Applying dropout to linear regression

Let $X \in \mathbb{R}^{n \times d}$ be a data matrix of n data points. $\mathbf{y} \in \mathbb{R}^n$ be a vector of targets.Linear regression tries to find a $\mathbf{w} \in \mathbb{R}^d$ that maximizes:

$$\| \mathbf{y} - X\mathbf{w} \|^2$$

When the input X is dropped out such that any input dimension is retained with probability p, the input can be expressed as $R * X$ where $R \in \{0, 1\}^{n \times d}$ is a random matrix with $R_{ij} \sim Bernoulli(p)$ and $*$ denotes element-wise product. Marginalizing the noise, the objective function becomes:

$$\min_{\mathbf{w}} \mathbb{E}_{R \sim Bernoulli(p)}[\| \mathbf{y} - (R * X)\mathbf{w} \|^2]$$

Applying dropout to linear regression

This reduce to:

$$\min_{\mathbf{w}} \| \mathbf{y} - pX\mathbf{w} \|^2 + p(1-p) \| \Gamma\mathbf{w} \|^2$$

where $\Gamma = (diag(X^T X))^{\frac{1}{2}}$. Therefore, dropout with linear regression is equivalent to ridge regression with a particular form for Γ. This form of Γ essentially scales the weight cost for weight w_i by the standard deviation of the i^{th} dimension of the data. If a particular data dimension varies a lot, the regularizer tries to squeeze its weight more.

Applying dropout to linear regression

The linear regression model can be represented as:

$$\mathbf{y} = \mathbf{w}^T \mathbf{X}$$

The objective function $J(\mathbf{w})$ in ridge regression is:

$$J(\mathbf{w}) = ||\mathbf{y} - \mathbf{w}^T \mathbf{X}||^2 + \lambda ||\mathbf{w}||^2$$

Applying dropout to linear regression

When dropout is applied to the input features, each feature is set to zero with probability p during each training iteration. Let's denote the dropout mask as $\mathbf{d}$, where each d_i is either 0 or 1.
The "dropped-out" feature matrix $\mathbf{X}_{\text{d.o.}}$ can be represented as:

$$\mathbf{X}_{\text{d.o.}} = \mathbf{d} \odot \mathbf{X}$$

The objective function $J_{\text{d.o.}}(\mathbf{w})$ when dropout is applied becomes:

$$J_{\text{dropout}}(\mathbf{w}) = \mathbb{E}_{\mathbf{d}}\left[||\mathbf{y} - \mathbf{w}^T\mathbf{X}_{\text{d.o.}}||^2\right]$$

Applying dropout to linear regression

Objective Function with Dropout:

$$J_{\text{d.o}}(\mathbf{w}) = \mathbb{E}_{\mathbf{d}}\left[||\mathbf{y} - \mathbf{w}^T\mathbf{X}_{\text{d.o}}||^2\right]$$

Expanding the Expectation

$$
\begin{aligned}
J_{\text{d.o}}(\mathbf{w}) &= \mathbb{E}_{\mathbf{d}}\left[(\mathbf{y} - \mathbf{w}^T\mathbf{X}_{\text{d.o}})(\mathbf{y} - \mathbf{w}^T\mathbf{X}_{\text{d.o}})^T\right] \\
&= \mathbb{E}_{\mathbf{d}}\left[\mathbf{y}\mathbf{y}^T - \mathbf{y}\mathbf{w}^T\mathbf{X}_{\text{d.o}} - \mathbf{X}_{\text{d.o}}\mathbf{w}\mathbf{y}^T + \mathbf{w}^T\mathbf{X}_{\text{d.o}}\mathbf{X}_{\text{d.o}}^T\mathbf{w}\right] \\
&= \mathbf{y}\mathbf{y}^T - 2\mathbf{y}\mathbf{w}^T(1-p)\mathbf{X} + \mathbf{w}^T(1-p)\mathbf{X}\mathbf{X}^T\mathbf{w} \\
&= ||\mathbf{y} - \mathbf{w}^T\mathbf{X}||^2 + (1-p)||\mathbf{w}||^2
\end{aligned}
$$

Applying dropout to linear regression

- The expanded form of the objective function with dropout includes a regularization term $(1-p)||\mathbf{w}||^2$.
- This term is similar to the regularization term in ridge regression.

Batch Normalization

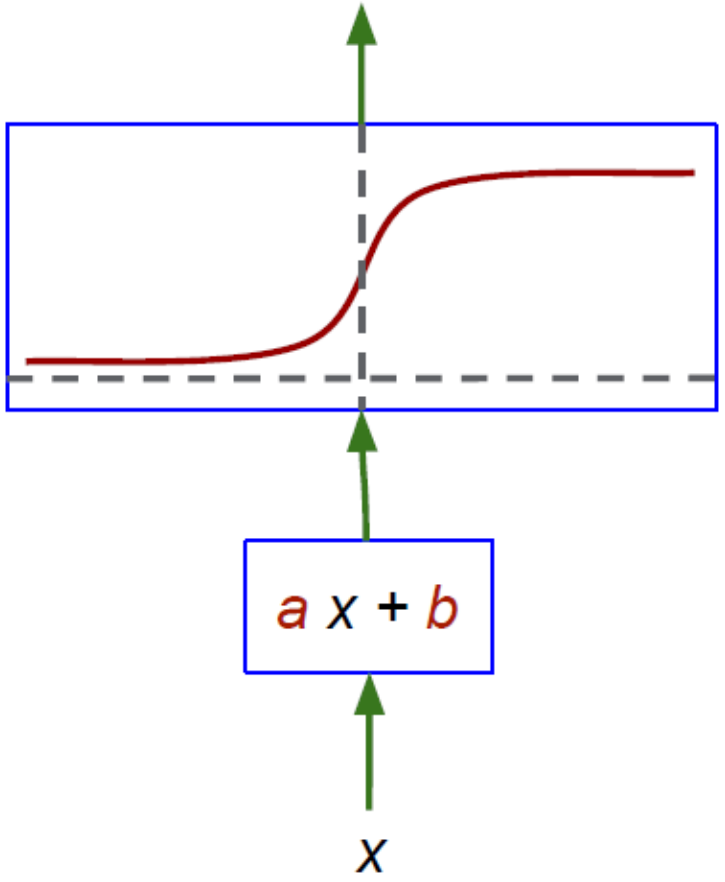
Slide modified from Sergey Ioffe *, with permission*

Slides based on

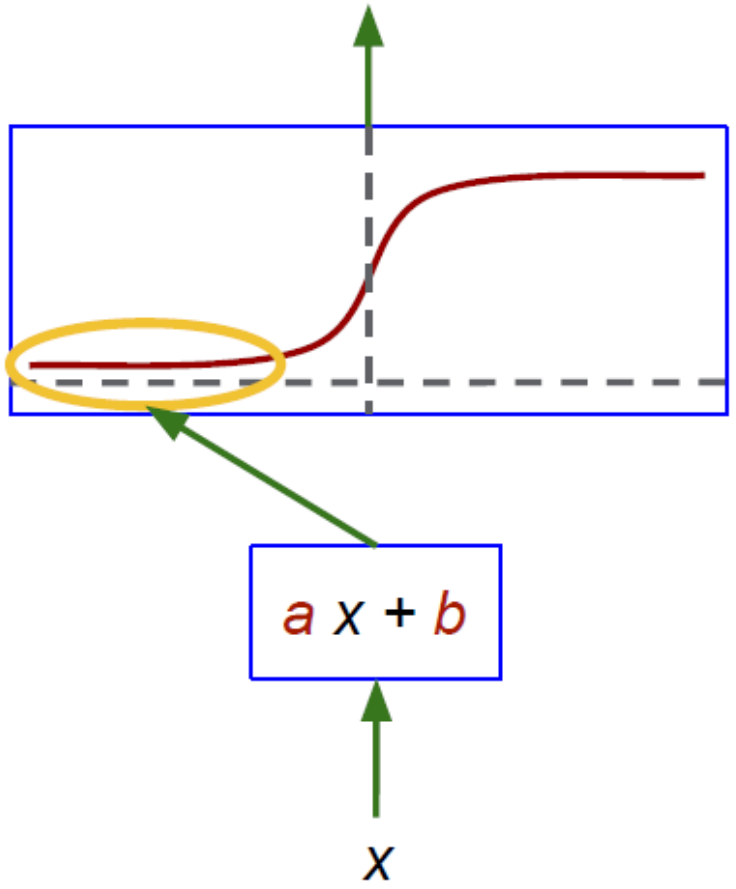
Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

By Sergey Ioffe and Christian Szegedy

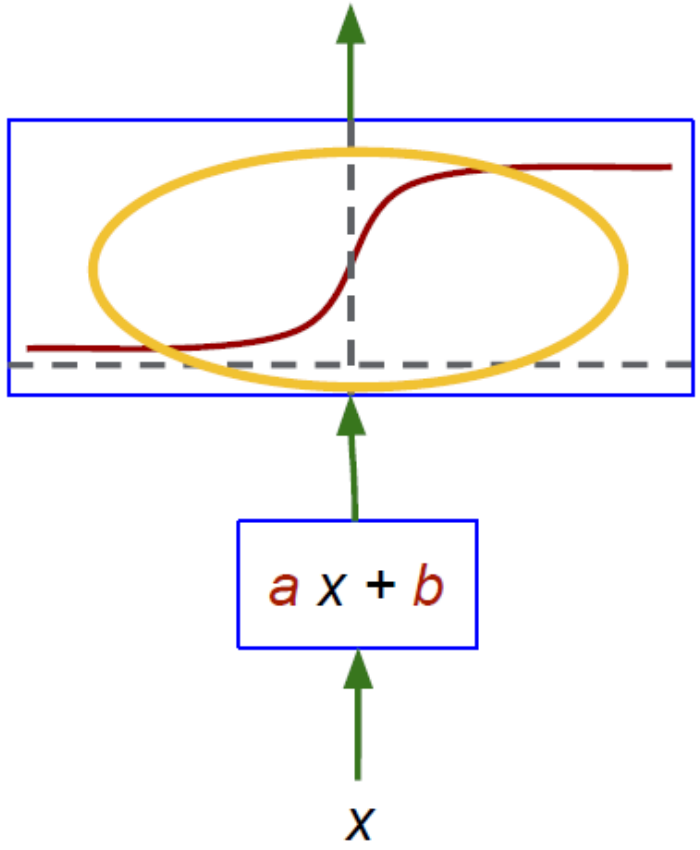
Batch Normalization



Batch Normalization

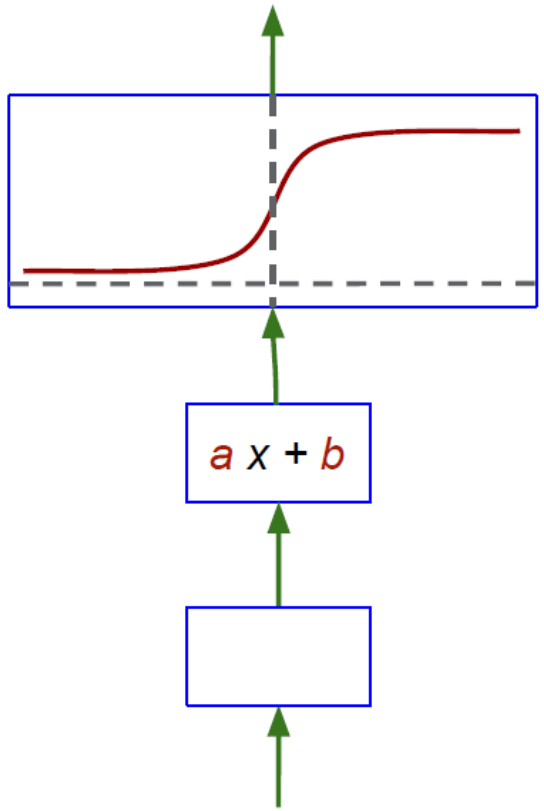


Batch Normalization



Effect of changing input distribution

- Careful initialization
- Small learning rates
- Rectifiers



Internal covariate shift

- Layer input distributions change during training

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

- Change in internal activation distribution requires domain adaptation

- Normalize each activation:

$$x \mapsto \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]}}$$

Mini-batch mean: $$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

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

Normalize: $$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$

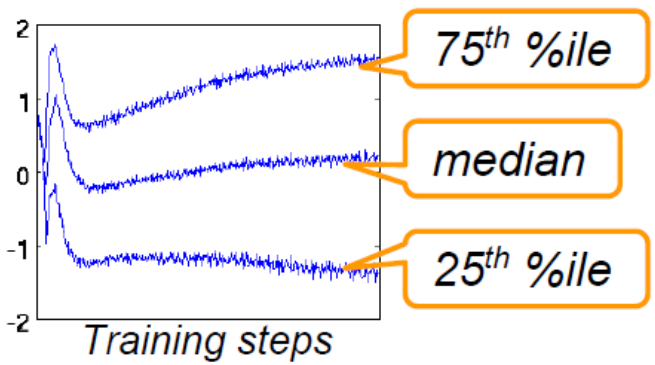
Scale and shift: $$y_i \leftarrow \gamma\widehat{x}_i + \beta$$

- Replace batch statistics with population statistics

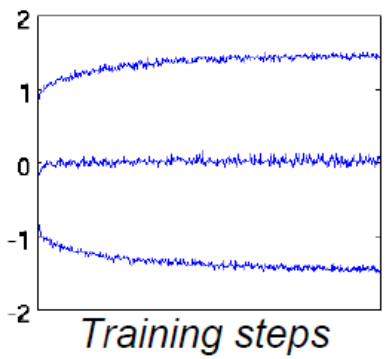
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad \Rightarrow \quad \widehat{x} \leftarrow \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}}$$

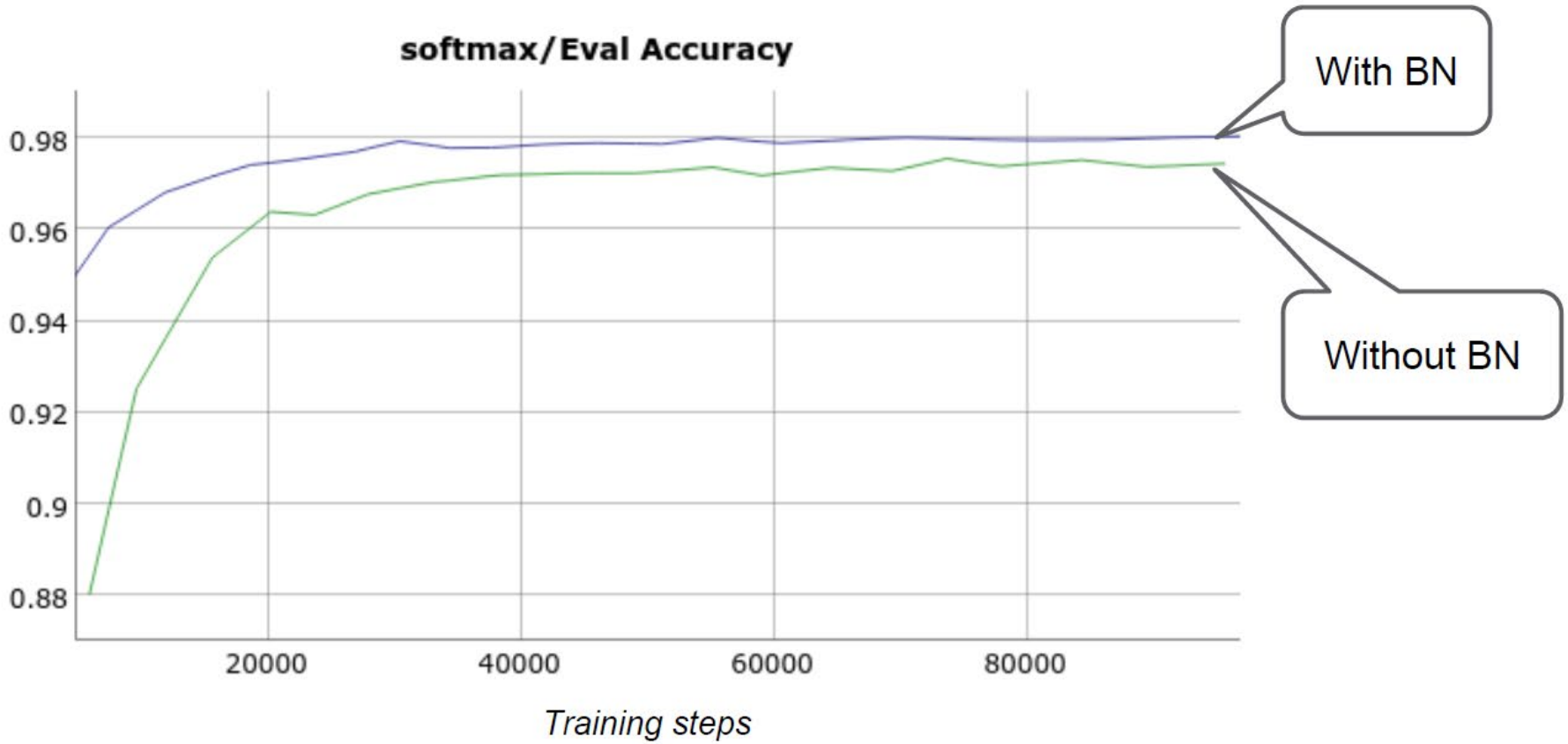
- MNIST: 3 FC layers + softmax, 100 logistic units per hidden layer
- Distribution of inputs to a typical sigmoid, evolving over 100k steps:

Without BN:

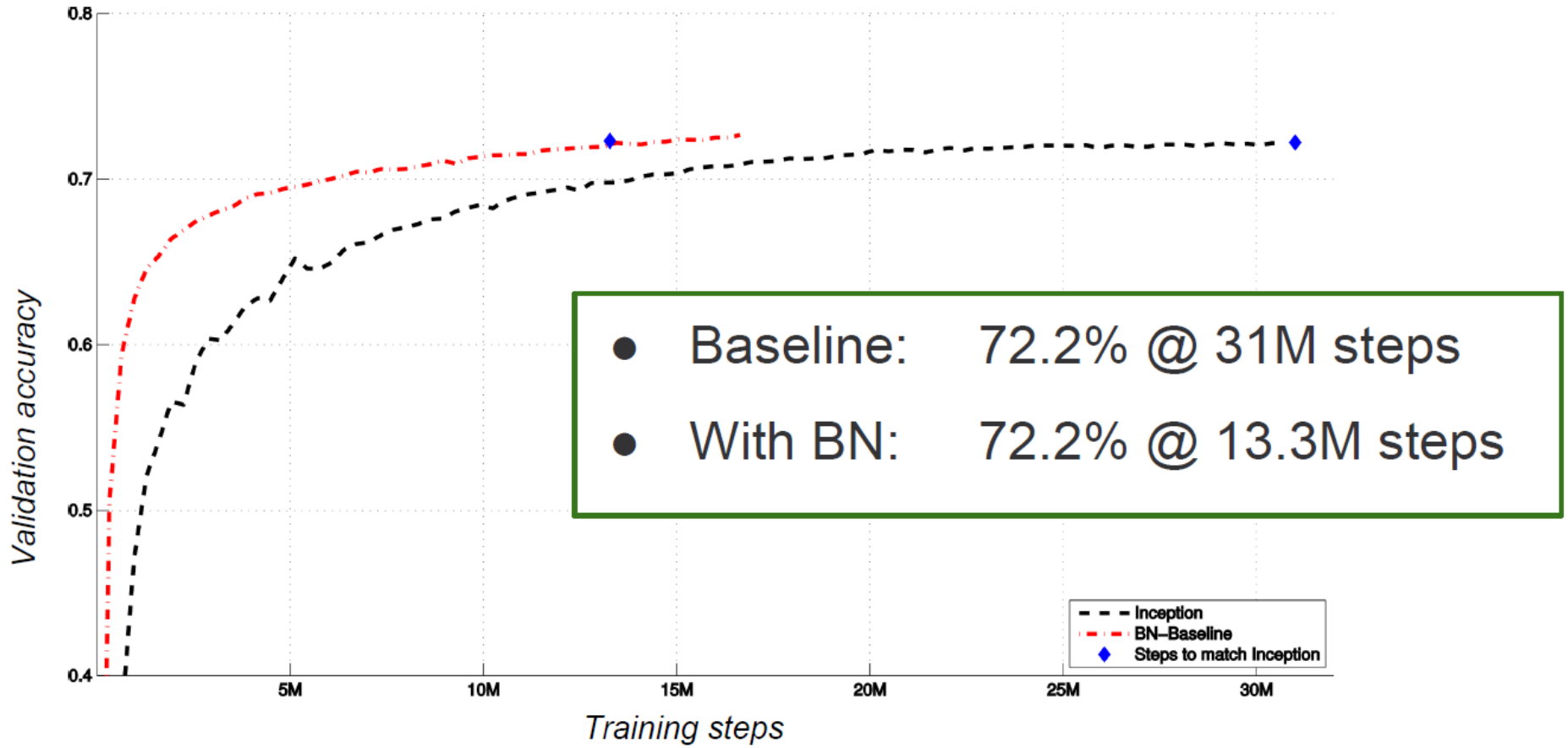


With BN:

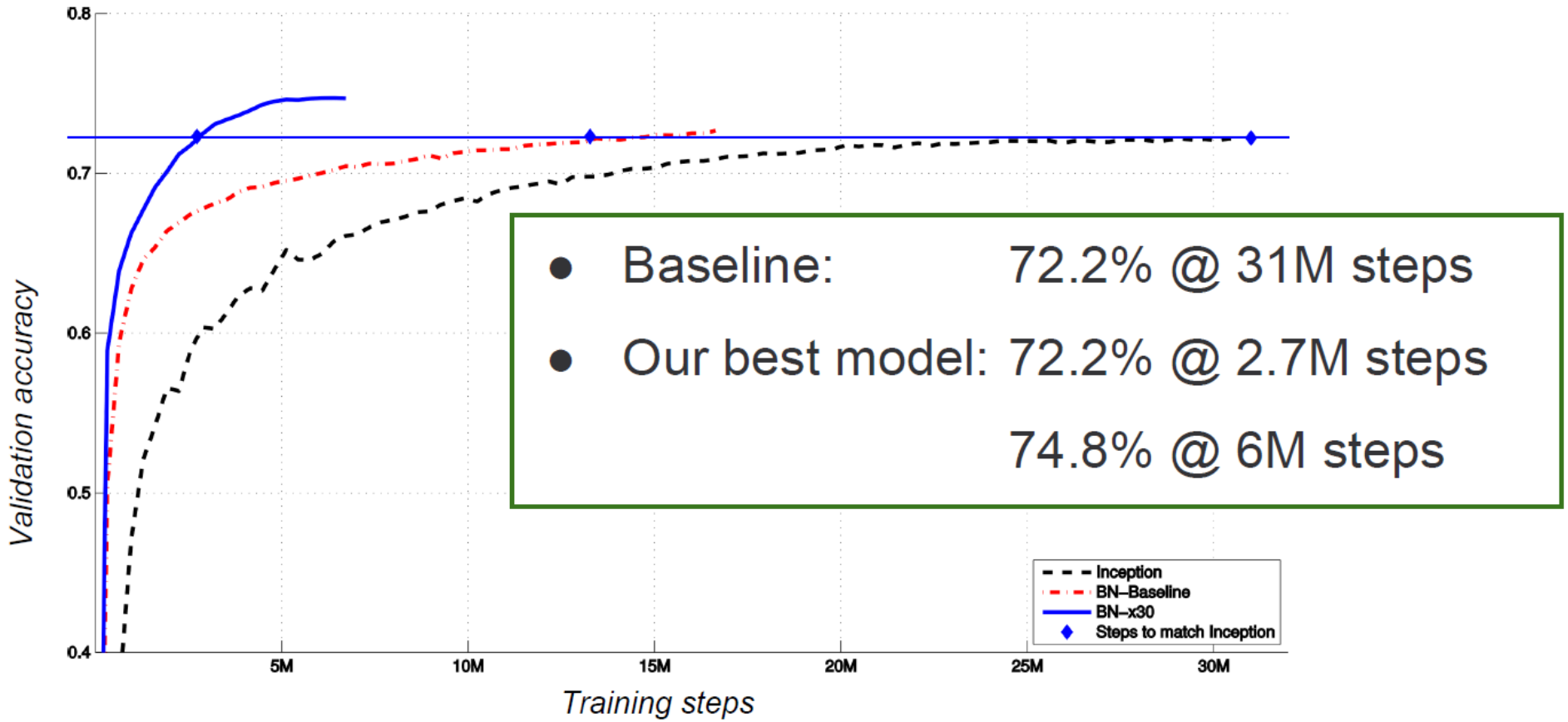



softmax/Eval Accuracy
0.98
0.96
0.94
0.92
0.9
0.88
20000
40000
60000
80000
Training steps
With BN
Without BN

- Inception: deep convolutional ReLU model
- Distributed SGD with momentum
- Batch Normalization applied at every convolutional layer
  - Extra cost (~30%) per training step


0.8
0.7
0.6
0.5
0.4
Validation accuracy
5M
10M
15M
20M
25M
30M
Training steps
Inception
BN–Baseline
Steps to match Inception
Baseline: 72.2% @ 31M steps
With BN: 72.2% @ 13.3M steps

- Batch Normalization enables higher learning rate
  - Increased 30x
- Removing dropout improves validation accuracy
  - Batch Normalization as a regularizer?



- Baseline: 72.2% @ 31M steps
- Our best model: 72.2% @ 2.7M steps
  74.8% @ 6M steps

Lecture 5

Batch Normalization revised

BatchNorm

- Batch normalization motivated by internal covariate shift (2015 lofee & Szegedy)

BatchNorm

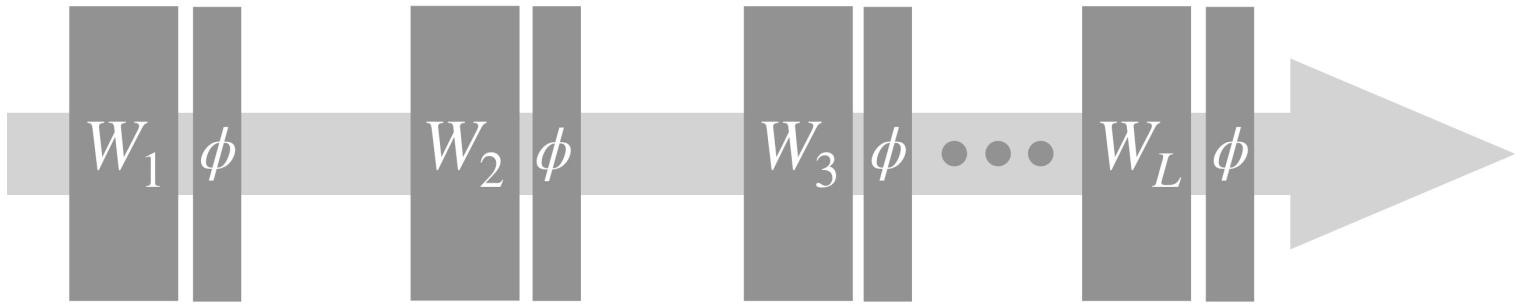
- Batch normalization motivated by internal covariate shift (2015 lofee & Szegedy)
- We have moving inputs.
- Distribution changes quite a lot.
- Solution: normalize the input of each leayer.

BatchNorm

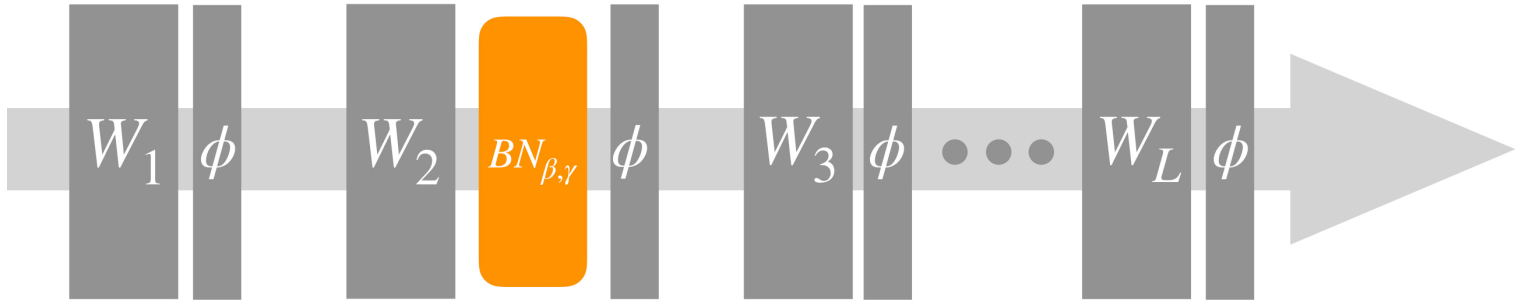
$$BN(y_j)^{(b)} = \gamma\left(\frac{y_j^{(b)} - \mu(y_j)}{\sigma(y_j)}\right) + \beta$$

- where $y_j^{(b)}$ denotes the value of the output y_j on the b-th input of a batch.
- and β and γ are learned parameters controlling the mean and variance of the output.

Standard Network

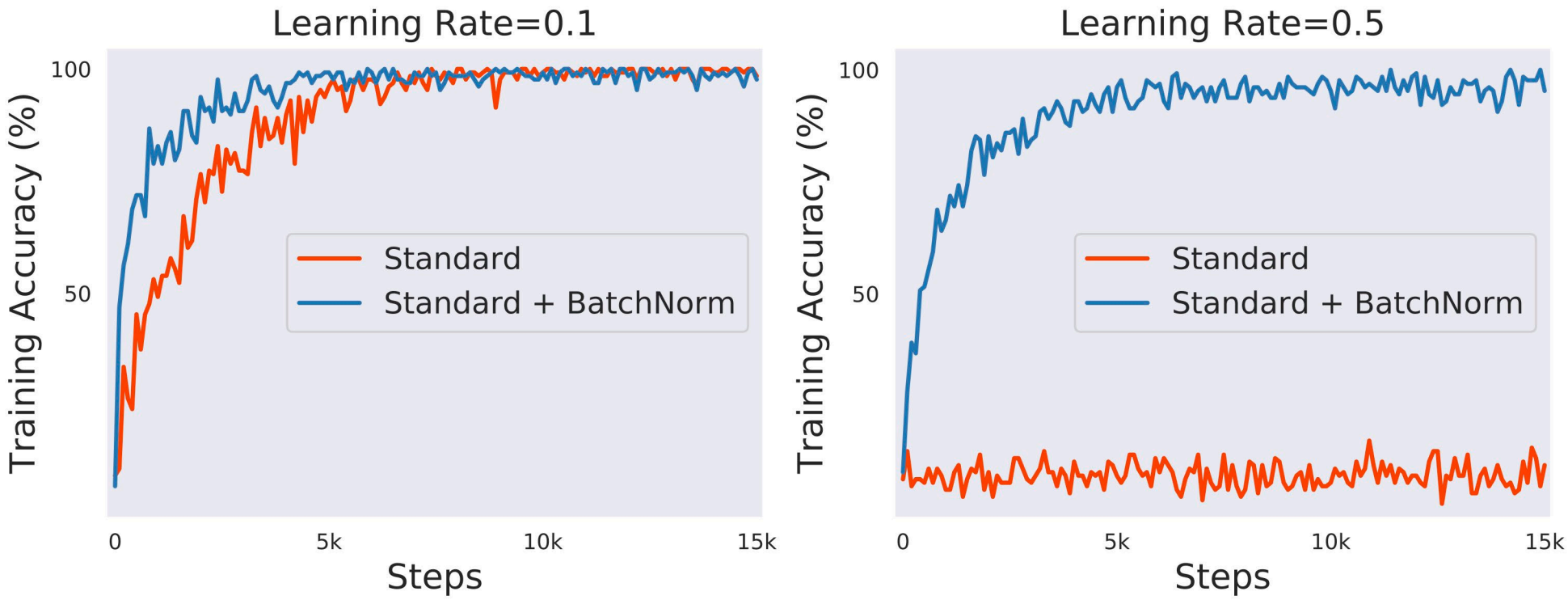


Adding a BatchNorm layer (between weights and activation function)



(Santurkar, et al., 2019)

Training with and without BatchNorm



(Santurkar, et al., 2019)

How Does Batch Normalization Help Optimization?

By Santurkar, S., Tsipras, D., Ilyas, A., & Mądry, A. (NeurIPS 2019).

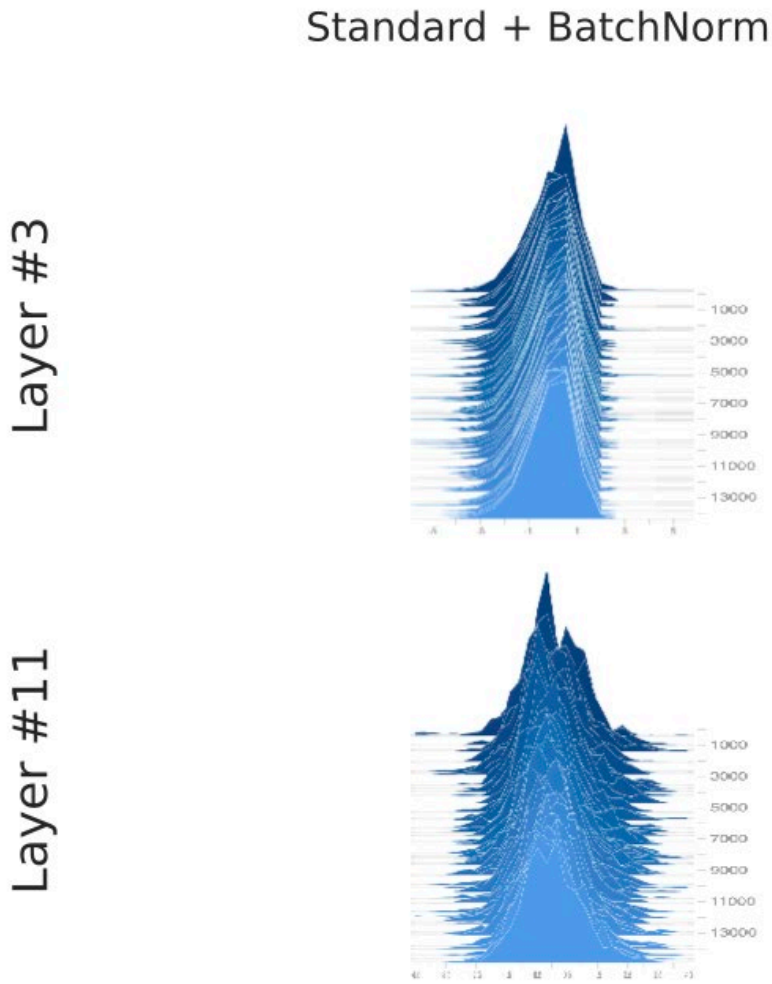
1. BatchNorm doesn't fix covariate shift.

2. If we fix covariate shift, it doesn't help.

3. If we intentionally increase ICS, it doesn’t harm.

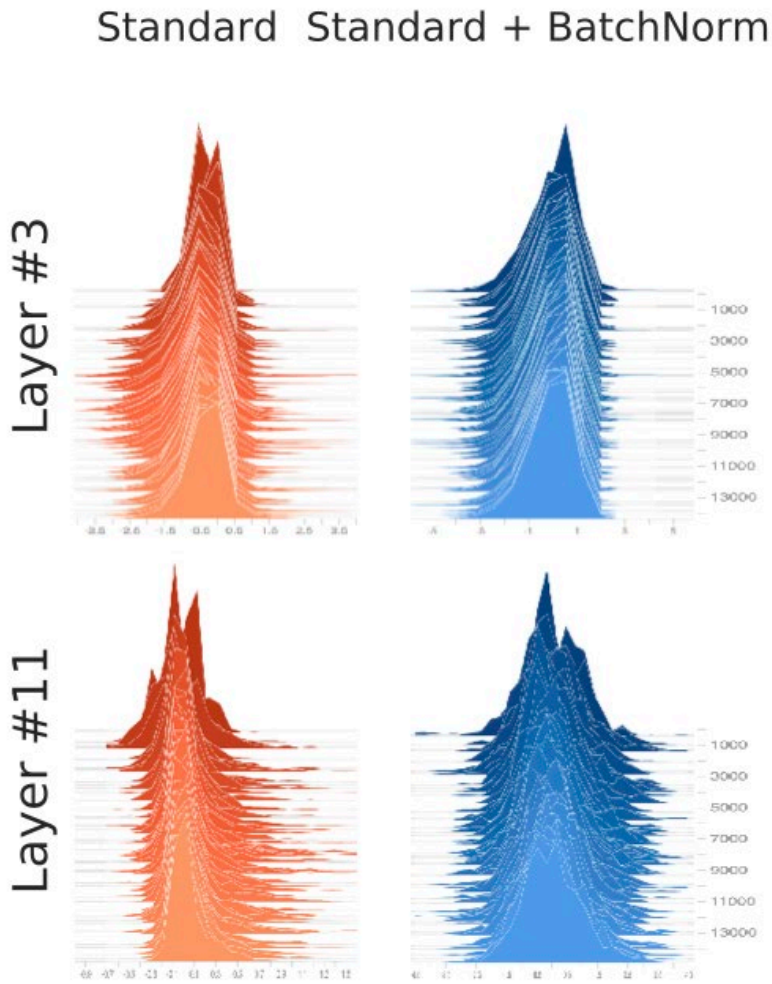
4. BatchNorm is not the only possible normalization. There are alternatives.

Training with and without BatchNorm



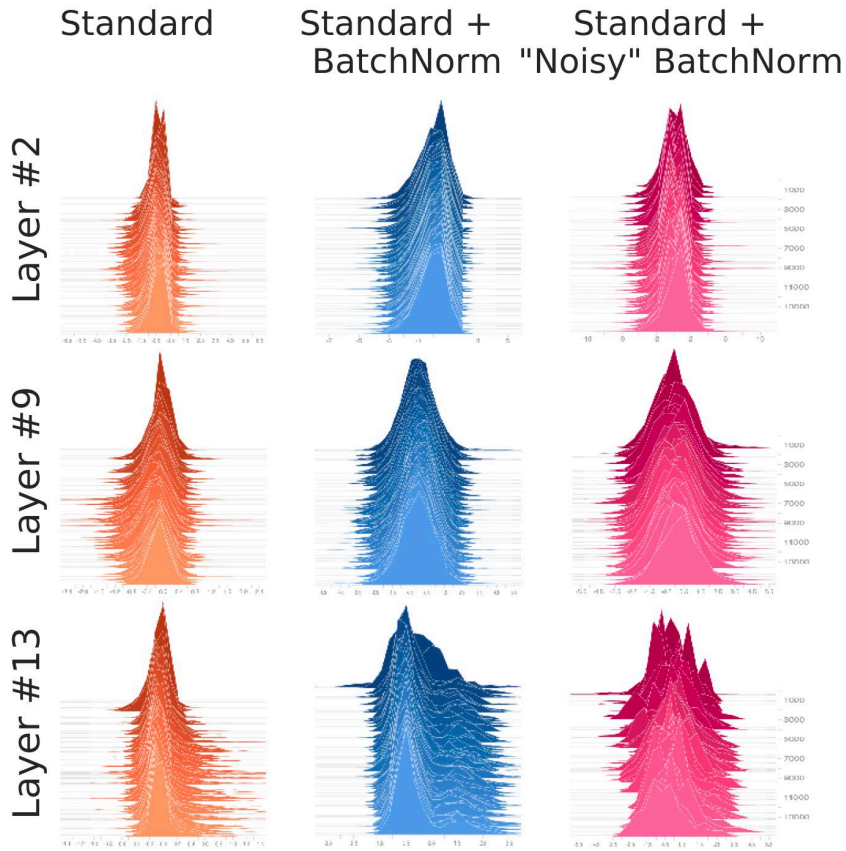
(Santurkar, et al., 2019)

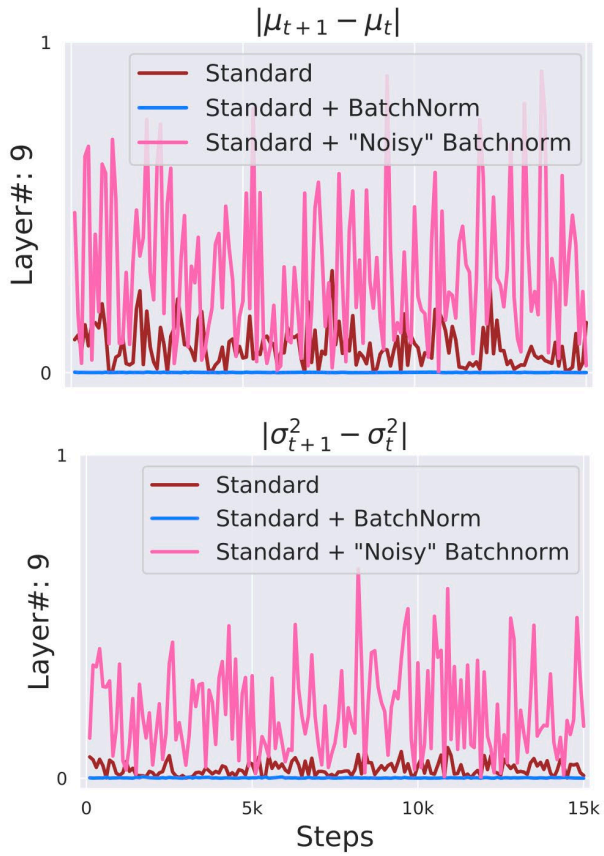
Training with and without BatchNorm



(Santurkar, et al., 2019)

Increase Internal Covariate Shift

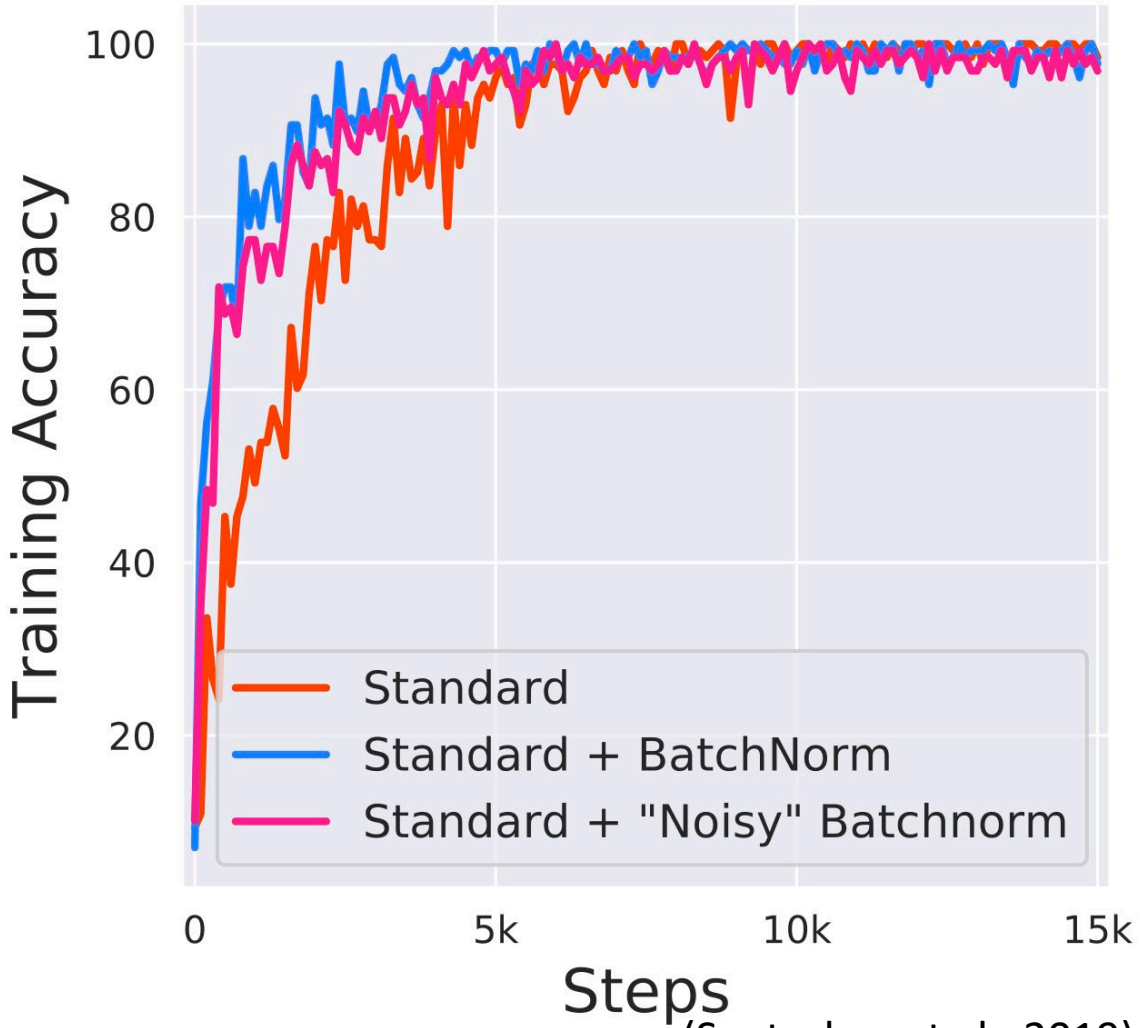




(Santurkar, et al., 2019)

Add non-stationary Gaussian noise (with a randomly sampled mean and variance at each iteration)

Increase Internal Covariate Shift



(Santurkar, et al., 2019)

- The optimization performance is unaffected.
- A network with noisy BatchNorm converges faster
- Almost the same as BN

L –Lipschitz and β –smooth

f is L-Lipschitz if $| f(x_1) - f(x_2) | \leq L \; || x_1 - x_2 ||$, for all x_1 and x_2.

f *is* β-smooth if its gradient is β-Lipschitz

BatchNorm's effect

- BatchNorm's reparameterization:

- Improves the Lipschitzness of the loss function.

  the loss changes at a smaller rate and the magnitudes of the gradients are smaller.

BatchNorm's effect

- BatchNorm's reparameterization:

- Improves the Lipschitzness of the loss function.
  the loss changes at a smaller rate and the magnitudes of the gradients are smaller.

- In gradient descent, we use the local linear approximation of the loss around the current solution to identify the best update step.

- How predictive of the nearby loss landscape this local approximation is

BatchNorm's effect

- BatchNorm's reparameterization:

- Improves the Lipschitzness of the loss function.

  the loss changes at a smaller rate and the magnitudes of the gradients are smaller.

BatchNorm's effect

- BatchNorm's reparameterization:

- Improves the Lipschitzness of the loss function.

  the loss changes at a smaller rate and the magnitudes of the gradients are smaller.

- Makes gradients of the loss more Lipschitz.

  the loss exhibits a significantly better β-smoothness

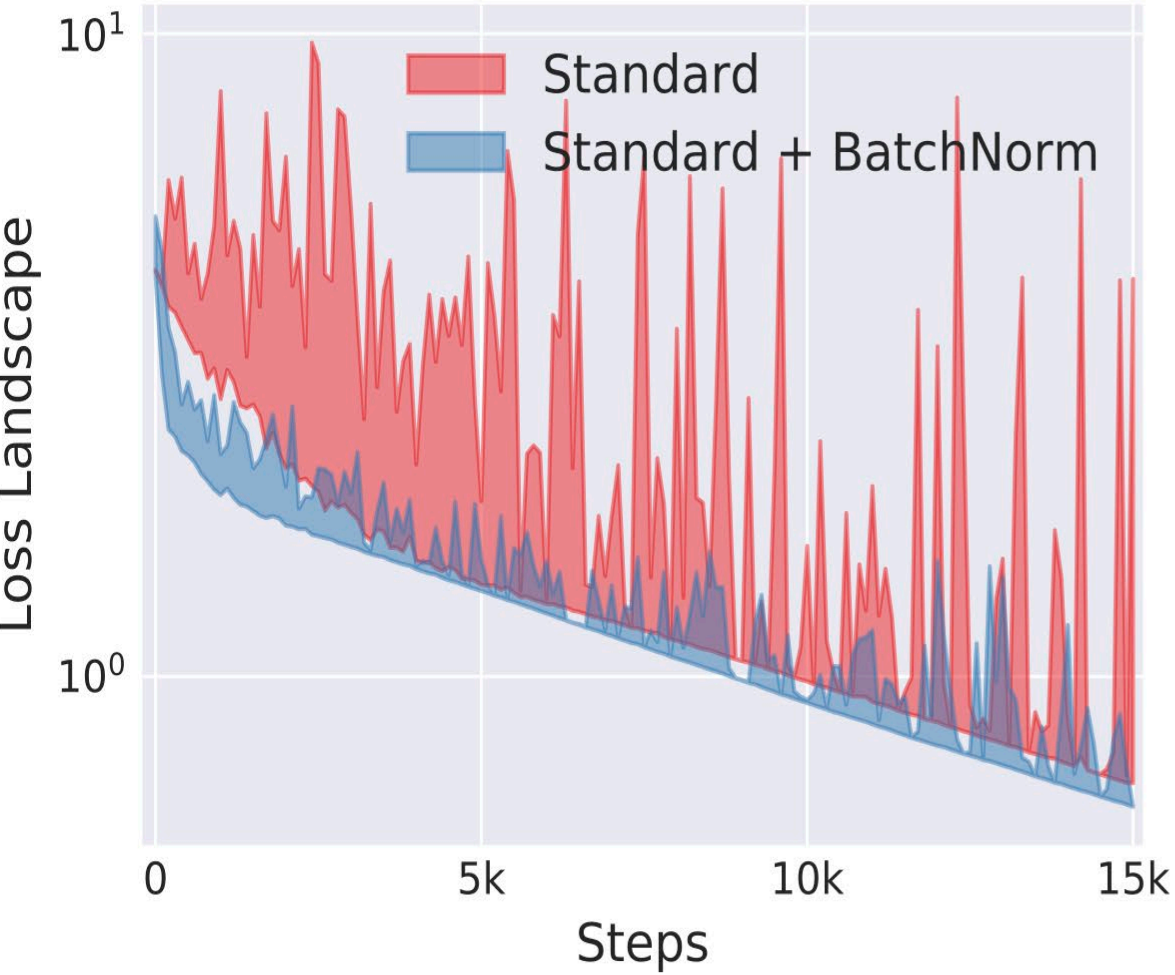
BatchNorm's effect

1. Variation of the value of the loss:

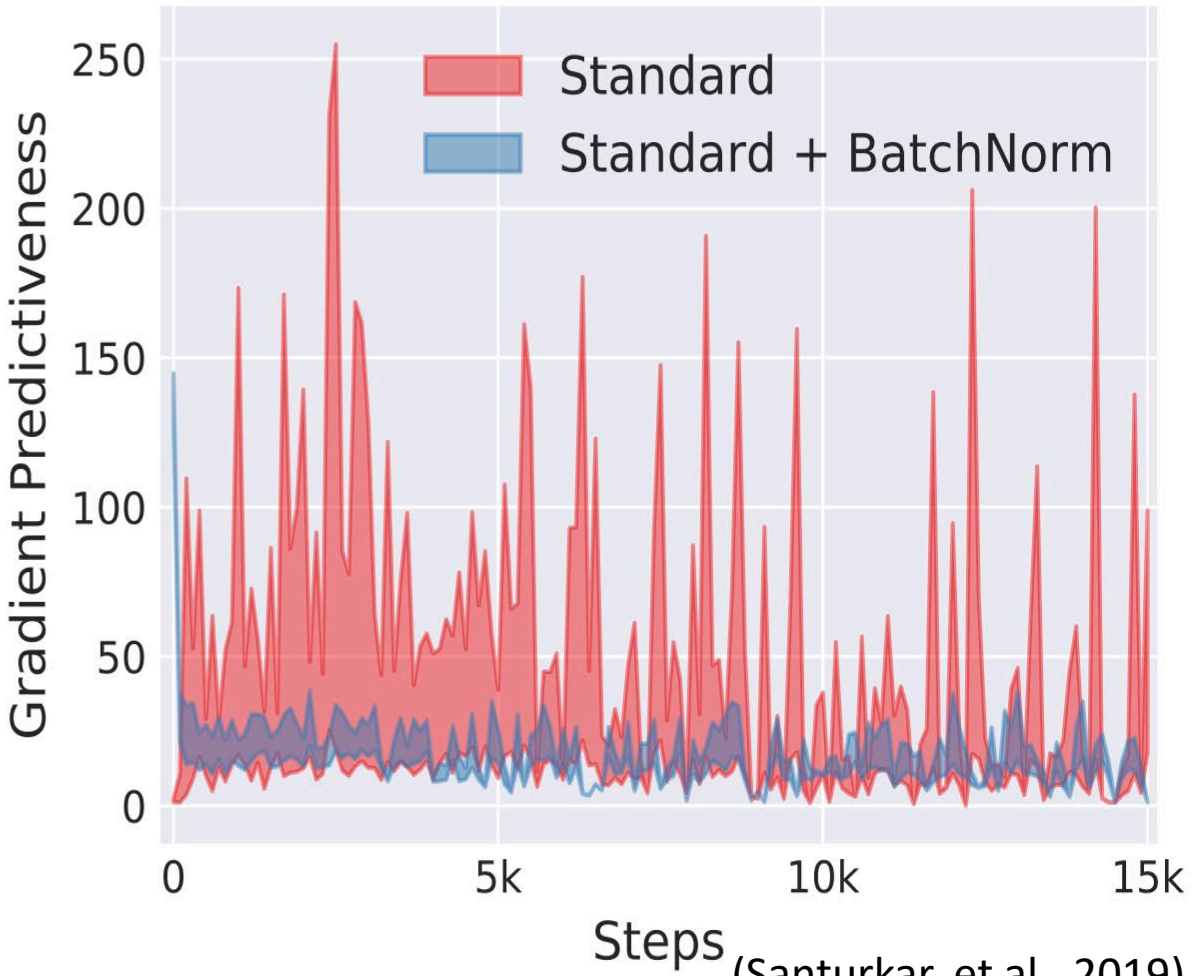
$$\mathcal{L}\big(x + \eta\nabla\mathcal{L}(x)\big), \quad \eta \in [0.05, 0.4]$$

2. Gradient predictiveness, i.e., the changes of the loss gradient:

$$||\nabla\mathcal{L}\big(x) - \nabla\mathcal{L}(x + \eta\nabla\mathcal{L}(x)\big)||, \quad \eta \in [0.05, 0.4]$$



$\mathcal{L}(x + \eta\nabla\mathcal{L}(x))$



(Santurkar, et al., 2019)

$||\nabla\mathcal{L}(x) - \nabla\mathcal{L}(x + \eta\nabla\mathcal{L}(x))||$

BatchNorm's effect

- A small variability of the loss indicates that the steps taken during training are unlikely to drive the loss uncontrollably high.

- A good gradient predictiveness implies that the gradient evaluated at a given point stays relevant over longer distances, hence allowing for larger step sizes.

How Does Batch Normalization Help Optimization?

By Santurkar, S., Tsipras, D., Ilyas, A., & Mądry, A. (NeurIPS 2019).

1. BatchNorm doesn't fix covariate shift.

2. If we fix covariate shift, it doesn't help.

3. If we intentionally increase ICS, it doesn’t harm.

4. BatchNorm is not the only possible normalization. There are alternatives.

These effects are not unique to BatchNorm

- In this normalization

$$BN(y_j)^{(b)} = \gamma\left(\frac{y_j^{(b)} - \mu(y_j)}{\sigma(y_j)}\right) + \beta$$

replace denominator by l_p norm. i.e. $\left|\left|y\right|\right|_p$

Alternatives to BatchNorm

- l_p BatchNorm
- Normalization over layers

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. In arXiv preprint arXiv:1607.06450, 2016

- Subsets of the batch

Yuxin Wu and Kaiming He. Group normalization. In European Conference on Computer Vision (ECCV), 2018.

Alternatives to BatchNorm

- Weight Normalization (normalizing the weights instead of the activations)

Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In Advances in Neural Information Processing Systems (NIPS), 2016.

- ELU and SELU are two proposed non-linearities that have a decaying slope instead of a sharp saturation and can be used as an alternative for BatchNorm

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In International Conference on Learning Representations (ICLR), 2016.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In Advances in Neural Information Processing Systems (NIPS), 2017.