Batch Normalization

How Does Batch Normalization Help Optimization?
(Santurkar, Tsipras, Iiyas, Madry, 2018)

Presenters: Grant Gasser, Blaine Rothrock

Link to Paper

Northwestern | McCORMICK SCHOOL OF ENGINEERING

Outline

- Input normalization
- Why do we use batches (mini-batches)?
- What is batch normalization?
- Why does batch normalization help optimization?
- Code, Experiments, Examples

Glossary

Input Normalization - normalizing the inputs to a neural network to speed up convergence

Batch Normalization (BatchNorm, BN) - normalize inputs to layers; speeds up convergence and improves performance

Internal Covariate Shift (ICS) - distribution of a layer's input changes as the parameters of previous layers change

Standardization/Whitening - a normalization method; subtracting by mean and dividing by variance; yields a result with mean=0 and variance=1 (unit variance)

Mini-batch - a subset of the training data (32, 64, 128…); what they're referring to when they say "Batch" norm

Saturation - an area where the gradients tend to 0 (leading to vanishing gradient problem)

Gamma and Beta - learnable parameters of Batch Norm; scale and shift, respectively

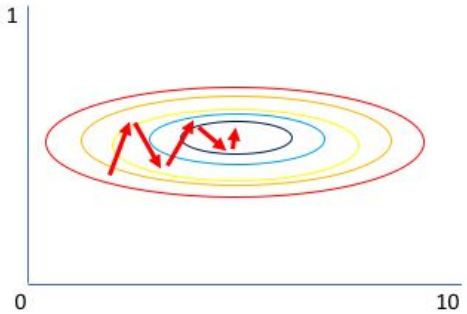
Running Mean, Running Variance - exponentially weighted (emphasize recent values) mean and variance for a batch; these are the mean and variance used to perform BN at test time; store these during training

Lipschitzness - a mathematical measure of smoothness of a function

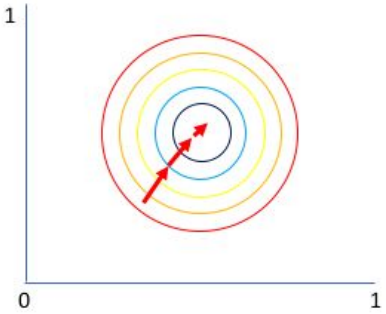
***β*-smoothness** - refers to the *derivative* of a function being Lipschitz

You May be Familiar With Input Normalization

Why normalize?



Gradient of larger parameter dominates the update



Both parameters can be updated in equal proportions

Input Normalization (Standardization/Whitening)

- Old idea in statistics
- Suppose *x* is a feature (we do this for all continuous features)
- μ is the mean of that feature
- σ is the standard deviation
- The normalized feature is *z,* which now has $\mu = 0$, $\sigma = 1$
- Why? Faster convergence.

$$z = \frac{x - \mu}{\sigma}$$

Input Normalization Example

- Problem: predict price of a house square footage, number of bedrooms, number of bathrooms, etc.
- Example: 3000 sq. feet, 3 bedrooms, 2 bathrooms
- Intuition: square footage feature would dominate the other two features because it's expressed in 1000s

```
>>> X
array([[3000,    3,    3],
       [2800,    2,    2],
       [3500,    4,    3],
       [2100,    2,    1]])
>>> for i in range(X.shape[1]):
...     Z[:, i] = (X[:,i] - X[:, i].mean()) / X[:, i].std()
...
>>> Z
array([[ 0.29851116,  0.30151134,  0.90453403],
       [-0.09950372, -0.90453403, -0.30151134],
       [ 1.29354835,  1.50755672,  0.90453403],
       [-1.49255579, -0.90453403, -1.50755672]])
```

Can we normalize the hidden layers, not just the input?

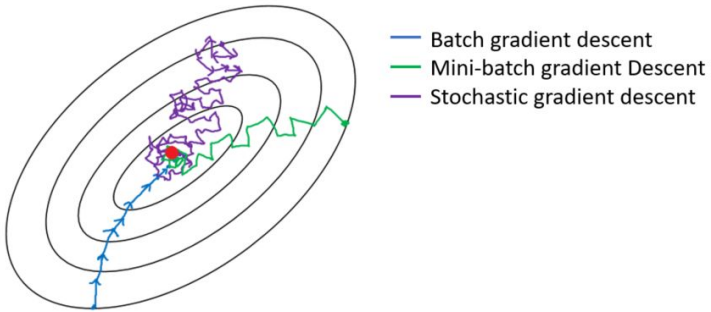
- ***Batch Normalization**: Accelerating Deep Network Training by Reducing Internal Covariate Shift (Sergey, Szegedy, 2015. Google)*
  - *~18K citations*
  - *"Training is complicated because the distribution of each layer's inputs changes while training, as the parameters of the previous layers change."*

- Supposedly works by addressing problem of internal covariate shift (ICS) by applying normalization to hidden layer inputs

- Has been shown to:
  - Speed up convergence (can use higher learning rates)
  - Allow lenient initialization (no need to be so careful)
  - Acts as regularizer, reducing the need for Dropout
  - Yield better performance (ensemble of batch norm models achieved 4.9% test error on ImageNet, superhuman performance)

First, why do we use batches?

- Computational limits, expensive to compute gradient over all training data
- Still want a decent estimate of the true gradient
  - So SGD (with just 1 example) may not be good estimate => inefficient training
- So we use **mini-batch** gradient descent
  - E.g. `batch_size = 64`
  - Cheaper while also providing good estimate of gradient
  - Typically:
    - "Batch": all data
    - "Mini-batch": subset of data
    - "SGD": 1 example



The Saturation Problem

- Consider a neural network:

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

where F_1 and F_2 are arbitrary transformations, and the parameters Θ_1, Θ_2 are to be learned so as to minimize the loss ℓ. Learning Θ_2 can be viewed as if the inputs $\mathrm{x} = F_1(\mathrm{u}, \Theta_1)$ are fed into the sub-network

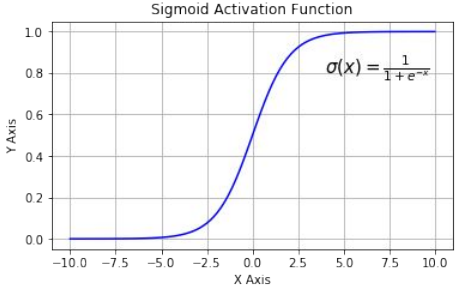
$$\ell = F_2(\mathrm{x}, \Theta_2).$$

For example, a gradient descent step

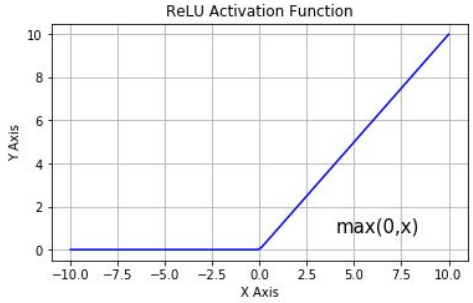
$$\Theta_2 \leftarrow \Theta_2 - \frac{\alpha}{m} \sum_{i=1}^{m} \frac{\partial F_2(\mathrm{x}_i, \Theta_2)}{\partial \Theta_2}$$

- Advantageous to have ***x*** fixed over time so $\boldsymbol{\Theta}_2$ does not have to constantly re-adjust

The Saturation Problem



- Consider a layer: $z = \sigma(Wu + b)$ where $x = Wu + b$
- As $|x|$ increases, $\sigma'(x)$ tends to 0 (vanishing gradient)
- Since x is affected by W and b, and parameters in previous layers, changes to these parameters can easily move dimensions of x to a **saturated** area and slow down convergence
- Solution to Saturation Problem before Batch Norm
  - ReLU(x) = max(0, x) as an activation function
  - Careful initialization
  - Small learning rates



Why batch normalization?

Covariate Shift

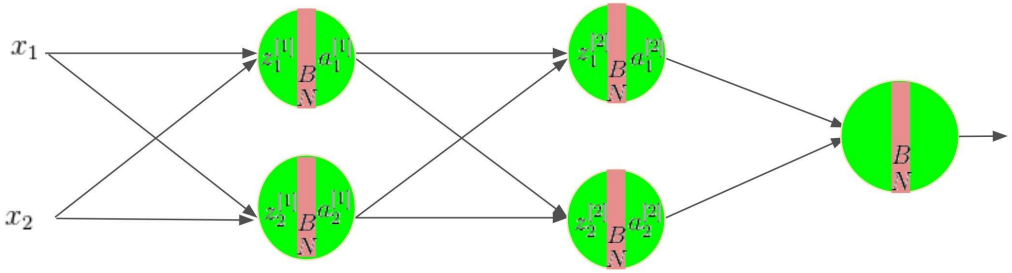
- A change in distribution of input variables between training and testing data
- Can be handled *externally* with **input normalization**

Internal Covariate Shift

- The distribution of inputs to a (hidden) layer in the network changes due to an update of parameters in the previous layers
- Handle by introducing a *internal* normalization (**batch norm)** step to the network

What is batch normalization?

- [ℓ]: the layer
- *W:* weight matrix
- *m:* the batch size
- *(i):* the hidden unit in the layer
- *g*: the nonlinear activation
- *z:* values before activation
- *ẑ:* values after BN
- *a:* values after activation
- **Key: apply BN before activation**



$$z^{[l]} = W^{[l]} a^{[l-1]} \longrightarrow \mu^{[l]} = \frac{1}{m}\sum_i z^{[l](i)}$$

$$\sigma^{[l]2} = \frac{1}{m}\sum_i (z^{[l](i)} - \mu^{[l]})^2$$

$$z^{[l](i)}_{norm} = \frac{z^{[l](i)} - \mu^{[l]}}{\sqrt{\sigma^{[l]2} + \epsilon}}$$

$$\tilde{z}^{[l](i)} = \gamma^{[l]} z^{[l](i)}_{norm} + \beta^{[l]} \longrightarrow a^{[l]} = g^{[l]}(\tilde{z}^{[l]})$$

BN Step

Batch Normalization Formulation

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

Why do we need gamma and beta?

-

Gamma and Beta

- Learnable parameters *gamma* and *beta*
- $y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$ // scale and shift
- `BatchNorm1d.weight` and `BatchNorm1d.bias`
- Initialization: gamma = 1, beta = 0
- Identity
  - Can set gamma = $\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}$
  - beta = $\mu_{\mathcal{B}}$
  - => No BN

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

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

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

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

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

Batch Normalization at Test Time

- When doing **input normalization**, need to scale the current test example with the *same* μ and σ^2 used to normalize the training set
- For **batch normalization**, scale the inputs to an activation layer with exponentially weighted average of μ and σ^2 recorded across all mini-batches during training
  - So need to store $\mu^{\{i\}[\ell]}$ and $\sigma^{2\{i\}[\ell]}$ for all i batches and ℓ layers
  - Named `running_mean` and `running_var` in Pytorch
  - Then compute batch norm using those values and the learned γ and β

How does batch normalization work?

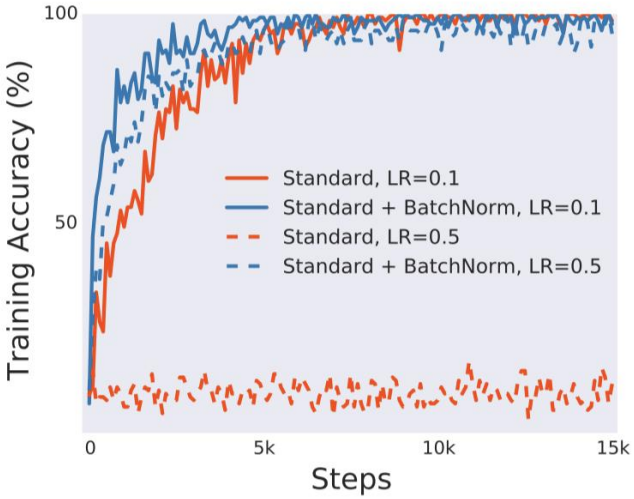
Santurkar et al. 2018

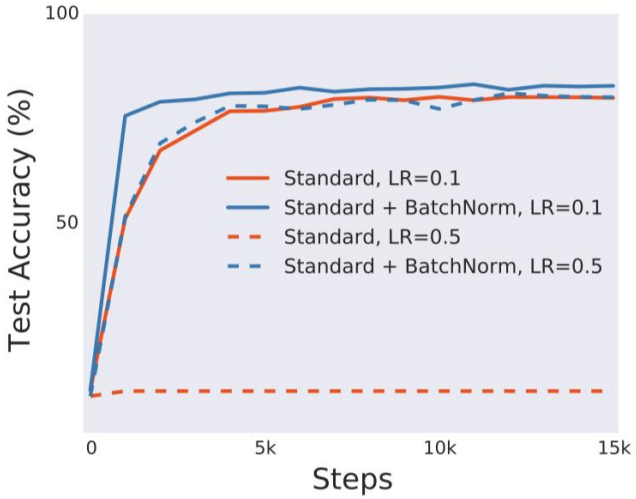
Batch Normalization Does Work, but ...

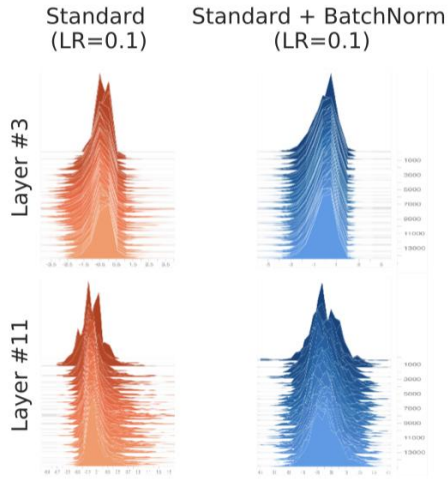
- Has been proven many times over to lead to:
  - Faster convergence
  - Less dependence on hyper parameters

- Why it works is what is in question
  - “so-called internal covariate shift”

BatchNorm might not even be reducing internal covariate shift







BatchNorm v. Non-BatchNorm on a VGG11 network on CIFAR-10

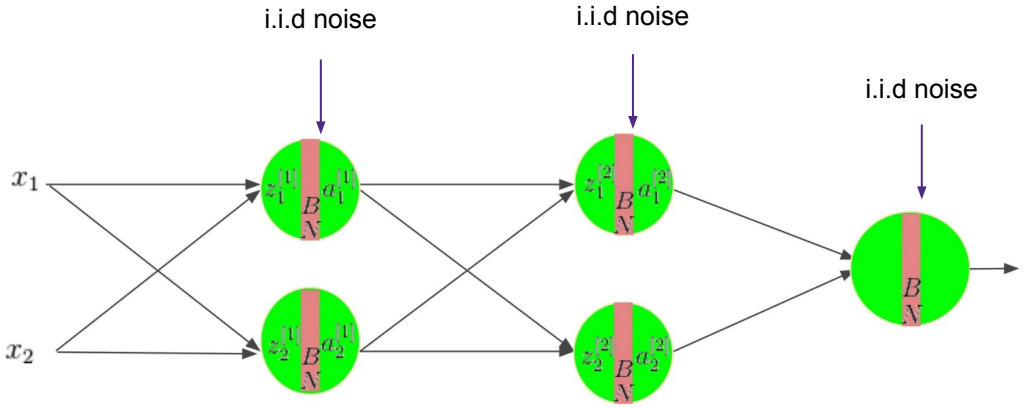
- Training gap is easily identified (left)
- Histograms on the right don't seem to show ICS

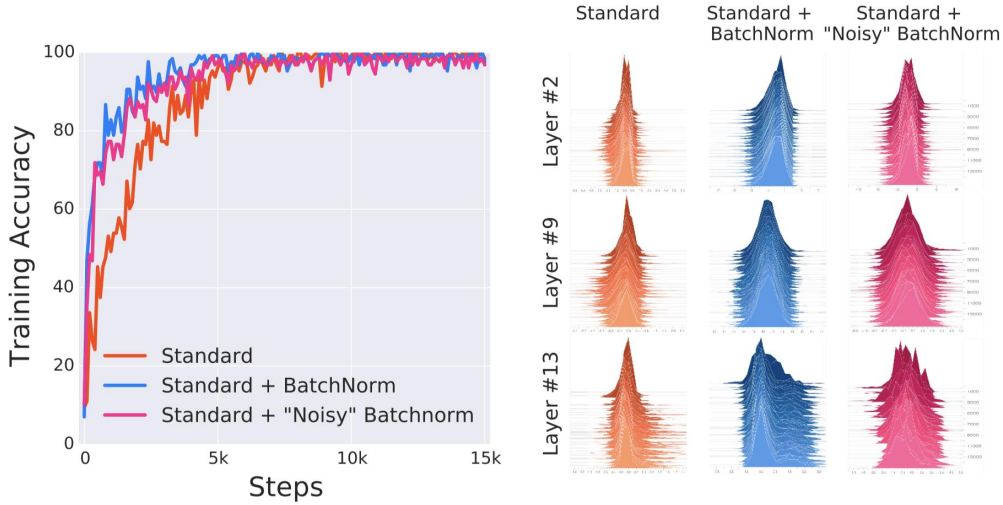
Raises Questions

1. Is the effectiveness of BN even related to ICS?
2. Is BN's stabilization of layer input distributions even reducing ICS?

Does BatchNorm's performance stem from controlling ICS? (Ioffe, Szegedy 2015)

- What happens when noise is injected **after** batch normalization
  - Force internal covariate shift

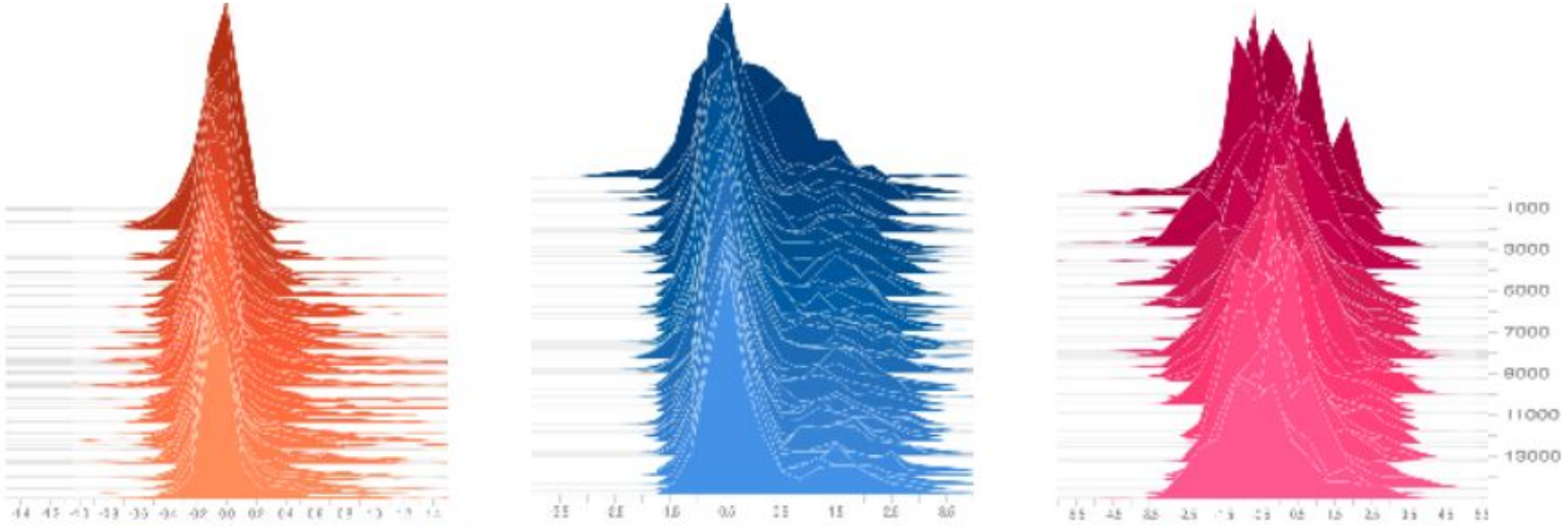




Non-BN v. BN v. BN-Noise on a VGG11 network on CIFAR-10

- BatchNorm & BatchNorm + noise accuracy are almost identical (left)
- Covariate shift is visibly present in BatchNorm + noise (right, pink) yet it still performs better than the Standard model (right, orange)

Layer #13



SCALE!

What is internal covariate shift?

The change in the distribution of network activations due to the change in network parameters during training. (Ioffe, Szegedy 2015)

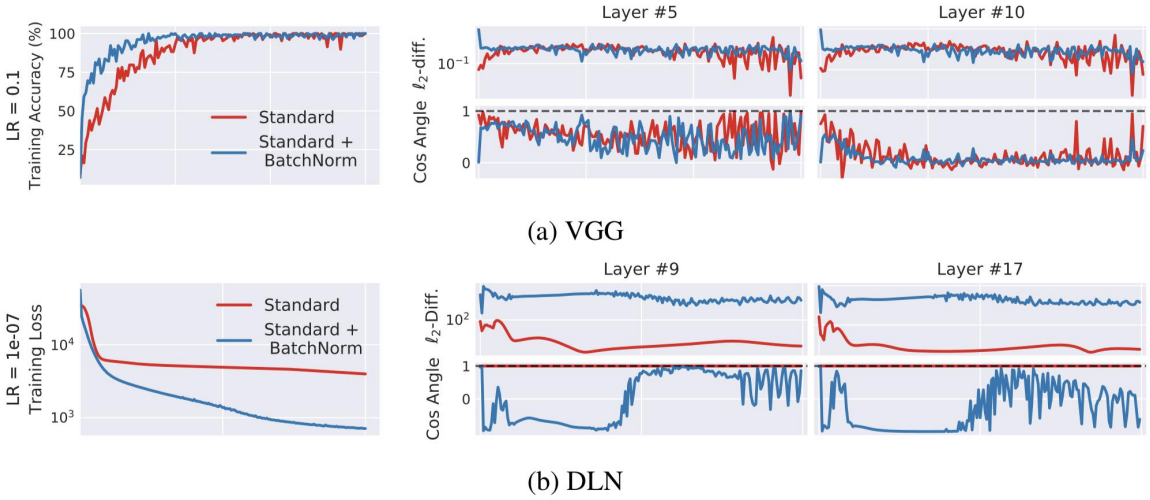
- Previous experiment assumes 👆; stability of the mean and variance of input distributions
- **Does not** have a effect on training performance

Is there a *better* way to define internal covariate shift that does link to training performance?

Definition 2.1. *Let $\mathcal{L}$ be the loss, $W_1^{(t)}, \ldots, W_k^{(t)}$ be the parameters of each of the k layers and $(x^{(t)}, y^{(t)})$ be the batch of input-label pairs used to train the network at time t. We define* internal covariate shift (ICS) *of activation i at time t to be the difference $||G_{t,i} - G'_{t,i}||_2$, where*

$$G_{t,i} = \nabla_{W_i^{(t)}} \mathcal{L}(W_1^{(t)}, \ldots, W_k^{(t)}; x^{(t)}, y^{(t)})$$

$$G'_{t,i} = \nabla_{W_i^{(t)}} \mathcal{L}(W_1^{(t+1)}, \ldots, W_{i-1}^{(t+1)}, W_i^{(t)}, W_{i+1}^{(t)}, \ldots, |W_k^{(t)}; x^{(t)}, y^{(t)}).$$



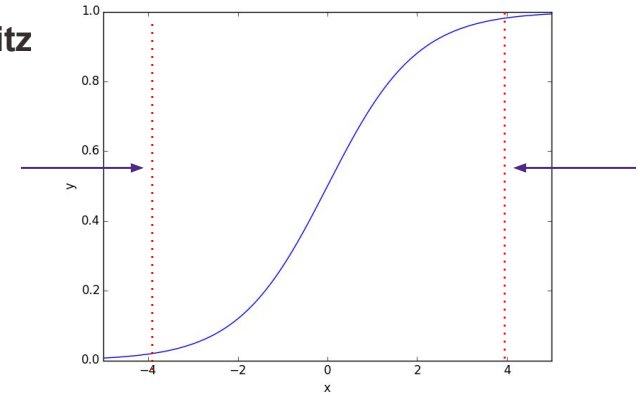
$G_{t,i}$ v. $G'_{t,i}$

- L_2- difference → ideally 0
- Cosine angle → ideally 1

Same or increase in internal covariate shift with BatchNorm

If not internal covariate shift, then what?

- Benefits of Batch Normalization:
  - Prevention of vanishing or exploding gradients
  - Robustness to hyperparameters (learning rate specifically)
  - Moving activations away from from saturation region on non-linearities
  - **More fundamental: makes optimization landscape significantly more smooth**
    - **Loss landscape is more Lipschitz**
    - **More important: gradients of loss are more Lipschitz**
      - **Better "effective" β-smoothness**



Lipschitzness: Lipschitz Continuity

A function where there exists some constant L that the absolute value of the difference of any two points cannot exceed L.

- The function is limited to how much it can change
- A f is said to be L-Lipschitz if:

$$|f(u) - f(w)| \leq L \|u - w\|$$

Da more Lipschitz da better!

To Wolfram Alpha!

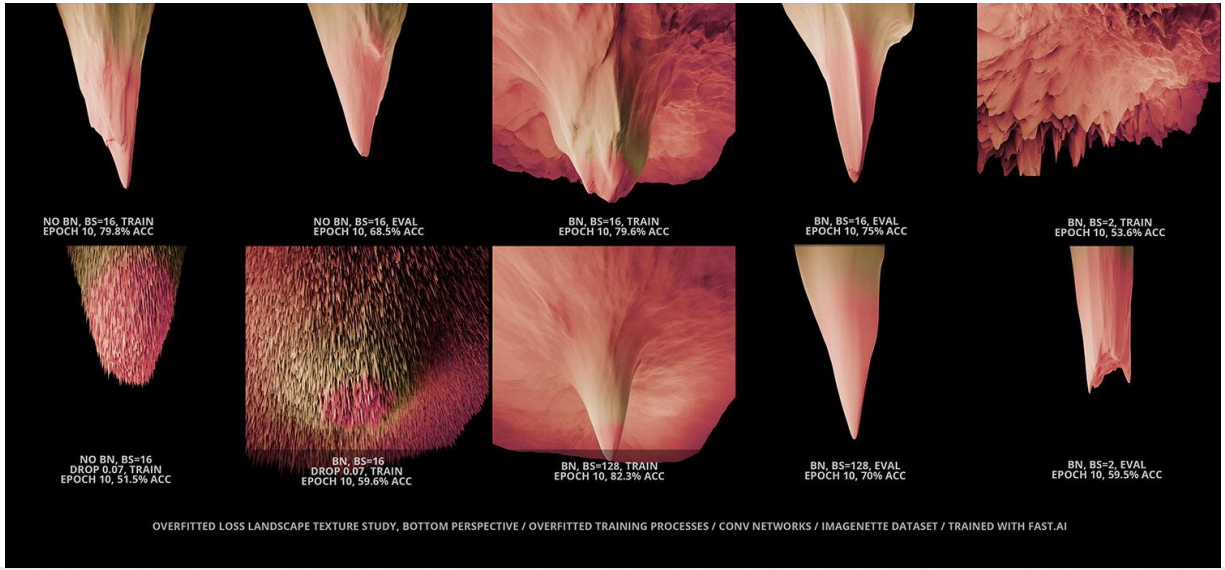
“Effective” β-smoothness

- *Gradients* of the loss are more Lipschitz
- β-smoothness:

$$||\nabla f(x) - \nabla f(y)||_2 \leq \beta ||x - y||_2.$$

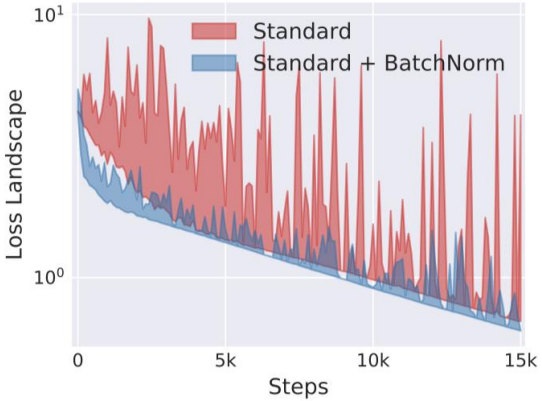
Lipschitzness and training

- Smooths the loss landscope → makes gradients more predictive and reliable
- Allows for larger learning rates → faster convergence

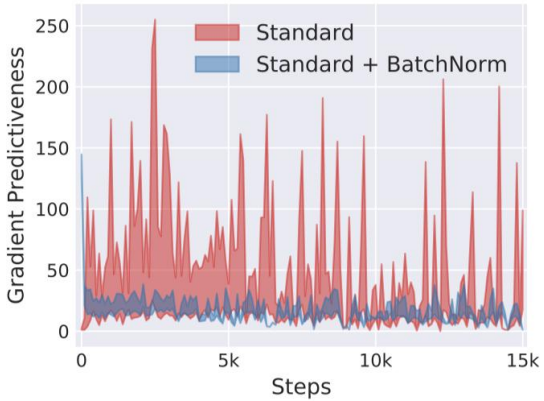


Measuring Lipschitzness (smoothness) in training

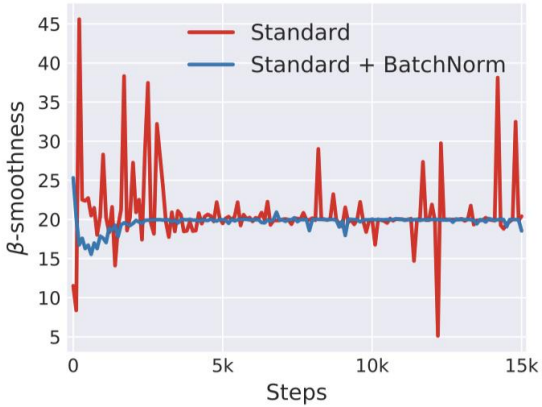
- Compute the gradient of the loss at each step in training
- Measure the change in the loss from step-to-step



(a) loss landscape



(b) gradient predictiveness



(c) "effective" β-smoothness

More on why smoothing helps optimization

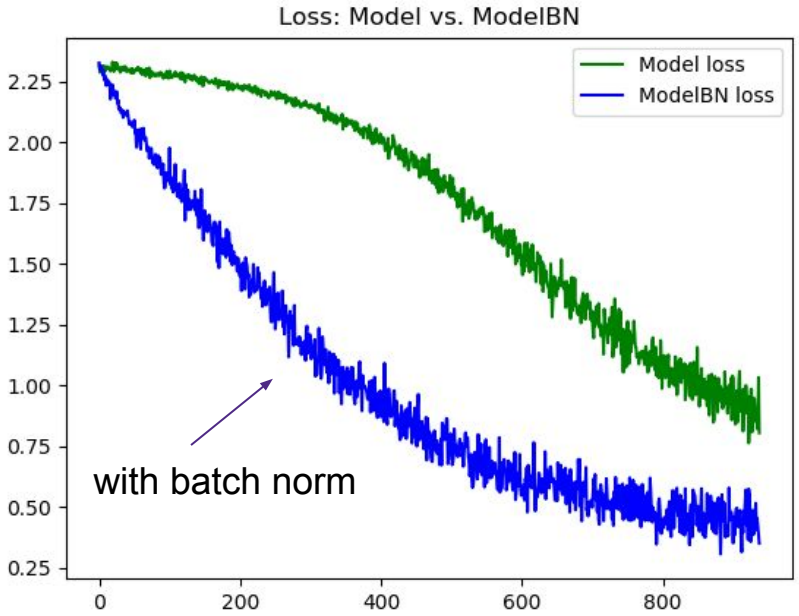
- Vanilla (non-BN) network loss functions are non-convex, often having many flat areas (=> vanishing gradient) and sharp minima/maxima (=> exploding gradients)
- Makes networks highly sensitive to choice of learning rate and weight initialization
- **Smoothening the landscape reduces these problems and allows for larger learning rates and less sensitivity to initialization!**
  - **=> Faster Convergence**

Theoretical Analysis Shmereotical Analysis

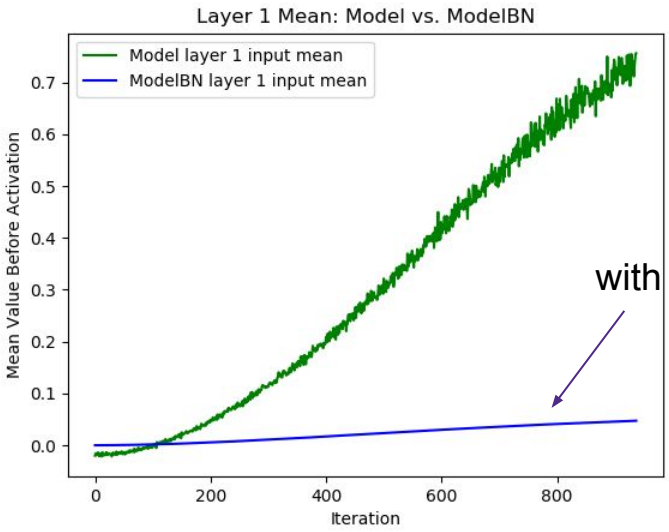
We leave it up to you to review

code time

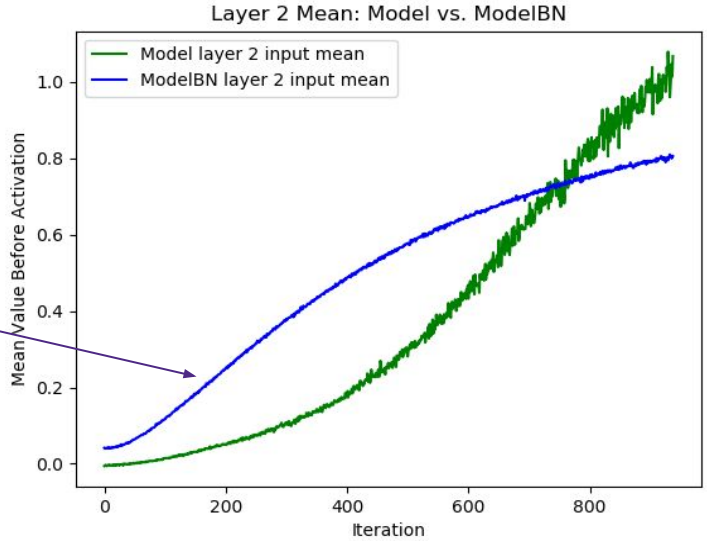
Our Experiments - Showing Batch Normalization



Our Experiments - Batch Norm Stabilization of Layer Inputs

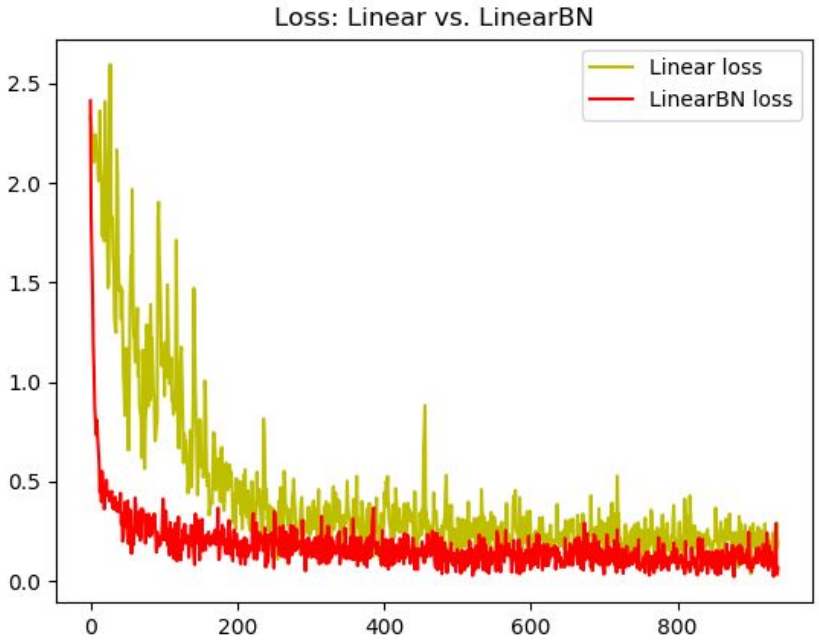


with batch norm



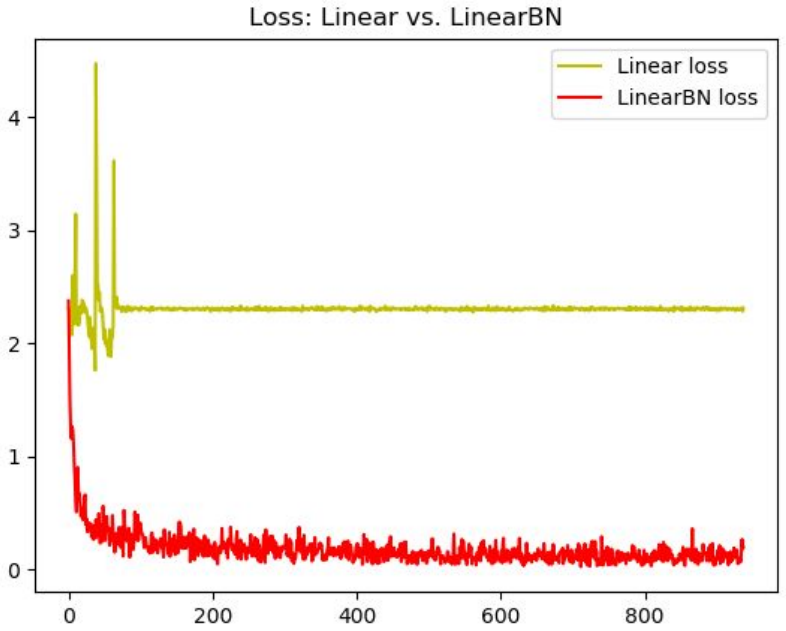
Our Experiments - BN Allows for Larger Learning Rates

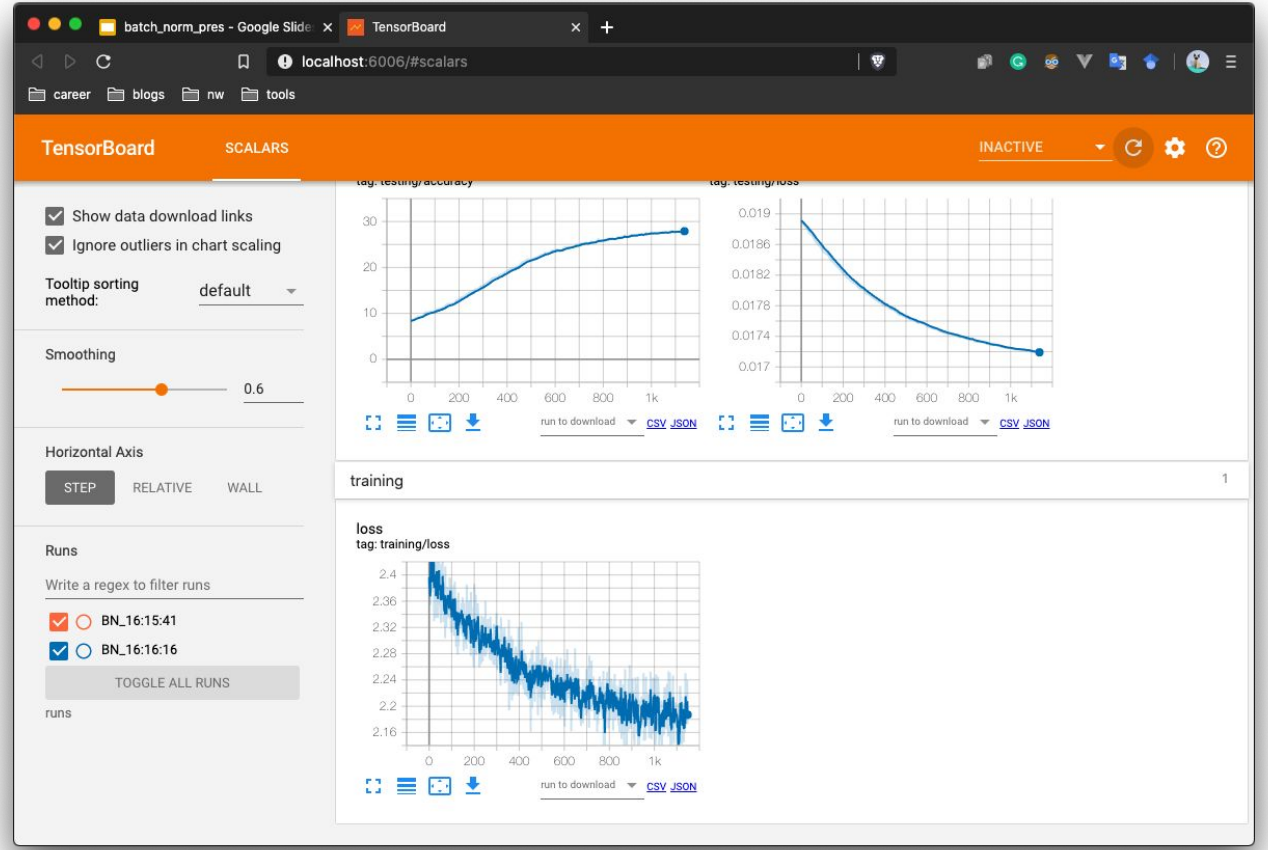
- Learning rate = **1.0**



Our Experiments - Even More Drastic

- Learning Rate = **2.0**




batch_norm_pres - Google Slide
TensorBoard
localhost:6006/#scalars
career
blogs
nw
tools
TensorBoard
SCALARS
INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method:
default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
BN_16:15:41
BN_16:16:16
TOGGLE ALL RUNS
runs
30
20
10
0
0.019
0.0186
0.0182
0.0178
0.0174
0.017
0
200
400
600
800
1k
run to download
CSV
JSON
training
1
loss
tag: training/loss
2.4
2.36
2.32
2.28
2.24
2.2
2.16

Where is BatchNorm now?

Batch Norm In Practice

- Taught in Coursera's Deep Learning Specialization (2017)
- Implemented in PyTorch and TensorFlow

Interesting Note on BN improving generalization

"Finally, our focus here was on the impact of BatchNorm on training but our findings might also shed some light on the BatchNorm's tendency to improve generalization. Specifically, it could be the case that the smoothening effect of BatchNorm's reparametrization encourages the training process to converge to more flat minima. Such minima are believed to facilitate better generalization [8, 11]. We hope that future work will investigate this intriguing possibility."

Further (more recent) Reading

- Training BatchNorm and Only BatchNorm: On the Expressive Power of Random Features in CNNs (Feb 2020)
  - Studying gamma and beta parameters of BN
- Network Deconvolution (March 2020)
  - “Faster Convergence than BatchNorm”
- Why Gradient Clipping Accelerates Training: A Theoretical Justification for Adaptivity (ICLR 2020)
  - Propose more relaxed smoothness constraint than Lipschitz
  - Use gradient clipping

Sources

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

Other: Deeplearning.ai, Pytorch, A billion different youtube videos and blogs