APRIL 23, 2019

TEXAS
The University of Texas at Austin

BATCH NORMALIZATION

EE 381V – Large Scale Optimization

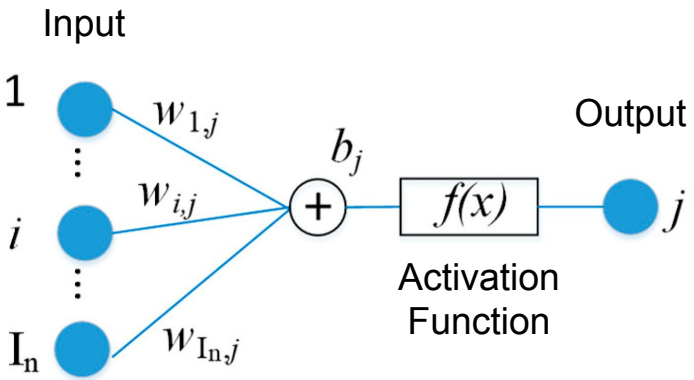
Xuewen Yao, Rebecca Adaimi, Jean Abou Rahal, Shorya Consul
The University of Texas at Austin

Outline

- Introduction to Neural Networks
- Internal Covariate Shift (ICS)
- Batch Normalization (BN) – the works
- How does BN help optimization?
- Simulations

What is a Neural Network?

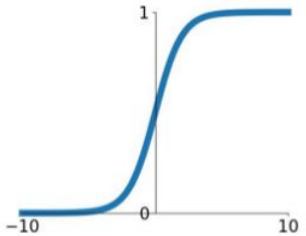
A Simple Neuron



Activation Functions

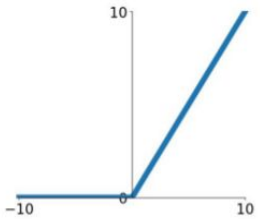
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



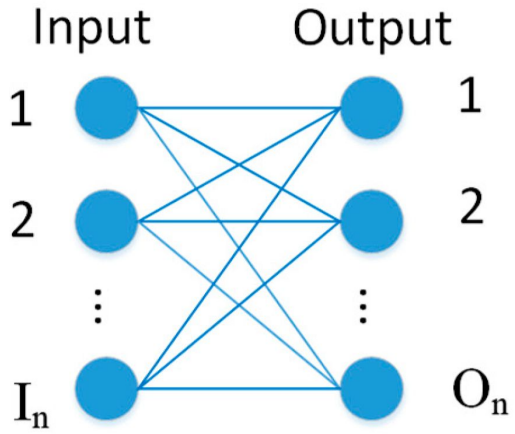
ReLU

$$\max(0, x)$$

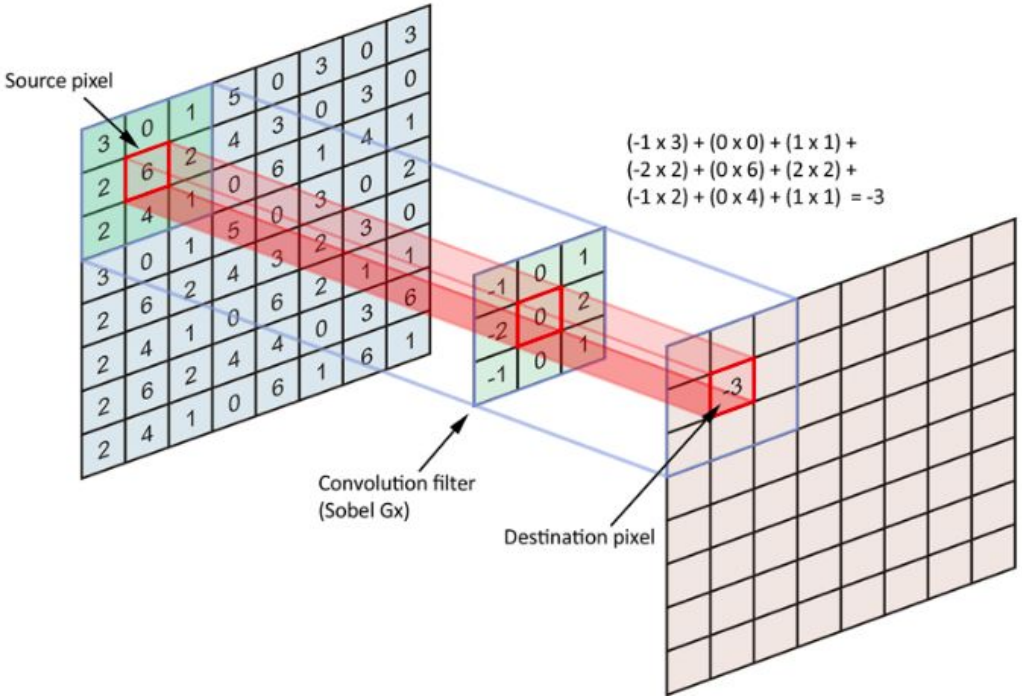


What is a Neural Network?

A **Fully Connected Layer** is a layer where every node in the first layer is connected to every node in the second layer.



Convolutional Neural Network



Internal Covariate Shift (ICS)

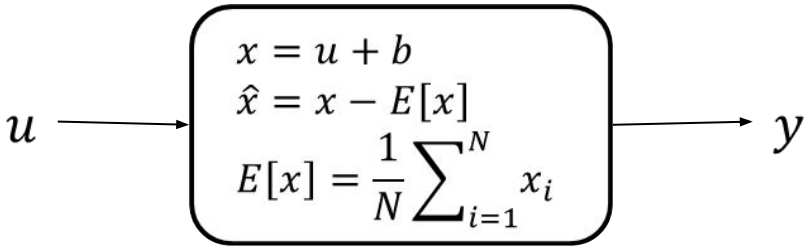
- Distribution of each layer's input changes during training as parameters or previous layers change.

$$l = F_2(F_1(u, \theta_1), \theta_2)$$
$$x = F_1(u, \theta_1)$$
$$l = F_2(x, \theta_2)$$

How can we reduce internal covariate shift?

Towards Reducing ICS

What about **whitening activations** at every training step?



GD optimization does not take into account that the normalization takes place.

If gradient step ignores the dependence of $E[x]$ on b, then:

$$b \leftarrow b + \Delta b \quad where \quad \Delta b \propto -\partial l / \partial \hat{x}$$

$$\Rightarrow u + (b + \Delta b) - E[u + (b + \Delta b)] = u + b - E[u + b]$$

Batch Normalizing Transform (over a mini-batch)

- Two simplifications
  - Normalize each scalar feature independently instead of jointly
  - Each mini-batch produces estimates of the mean and variance of each activation

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.

- Normalization alone may change what the layer can represent
- Transformation inserted in the network can represent the identity transform

Training with Batch-Normalized Networks

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 $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$

Specify a subset of activations and insert the BN transform for each of them

Inference with Batch-Normalized Networks

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**

Use the population statistics rather than mini-batch ones

Batch-Normalized Convolutional Networks

- Affine transformation followed by an element-wise nonlinearity

$$\mathrm{z} = g(W\mathrm{u} + \mathrm{b}) \longrightarrow \mathrm{z} = g(\mathrm{BN}(W\mathrm{u}))$$

nonlinearity such as ReLU and Sigmoid

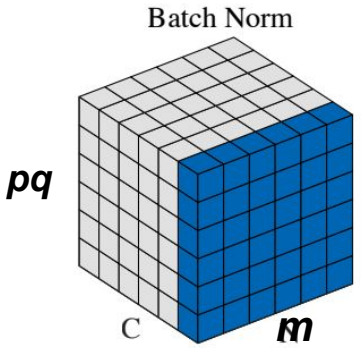
learnt parameters

- For convolutional layers, jointly normalize all the activations in a mini-batch over all locations

$$m' = |\mathcal{B}| = m \cdot pq$$

Mini-batch size

size of feature maps



Advantages of Batch Normalization

- Enable higher learning rate
  - Prevents small changes to the parameters from amplifying into larger and suboptimal changes in activations in gradients
  - Makes training more resilient to the parameter scale

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

parameter scaled parameter

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

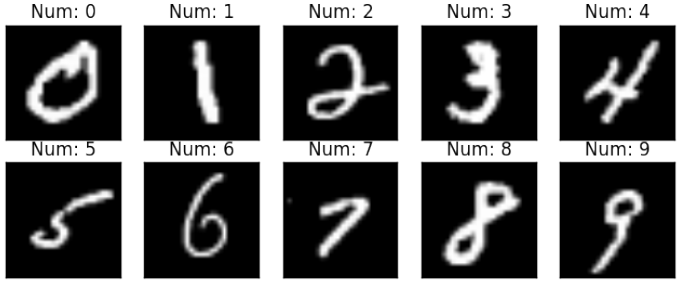
$$\frac{\partial \mathrm{BN}((aW)u)}{\partial (aW)} = \frac{1}{a} \cdot \frac{\partial \mathrm{BN}(Wu)}{\partial W}$$

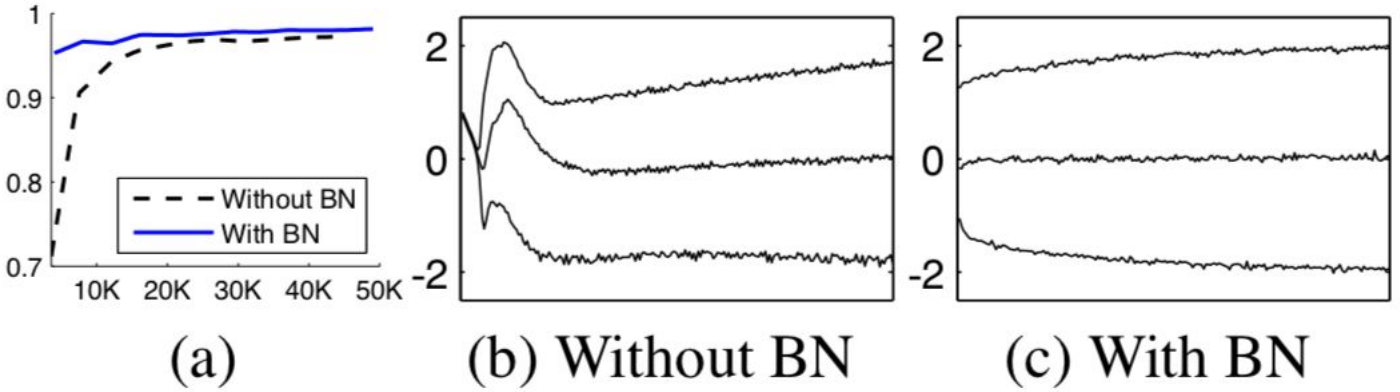
  - Makes layer Jacobians to have singular values closer to 1, which is beneficial for training.

- Regularize the model
  - a training example is seen in conjunction with other examples in the mini-batch, and the training network no longer producing deterministic values for a given training example.

Experiments - Activations over Time

- Predict the digit class on the MNIST dataset
- 28X28 binary images, 3 hidden FC layers with 100 activations each, sigmoid non-linearity, cross-entropy loss
- BN added to each hidden layer

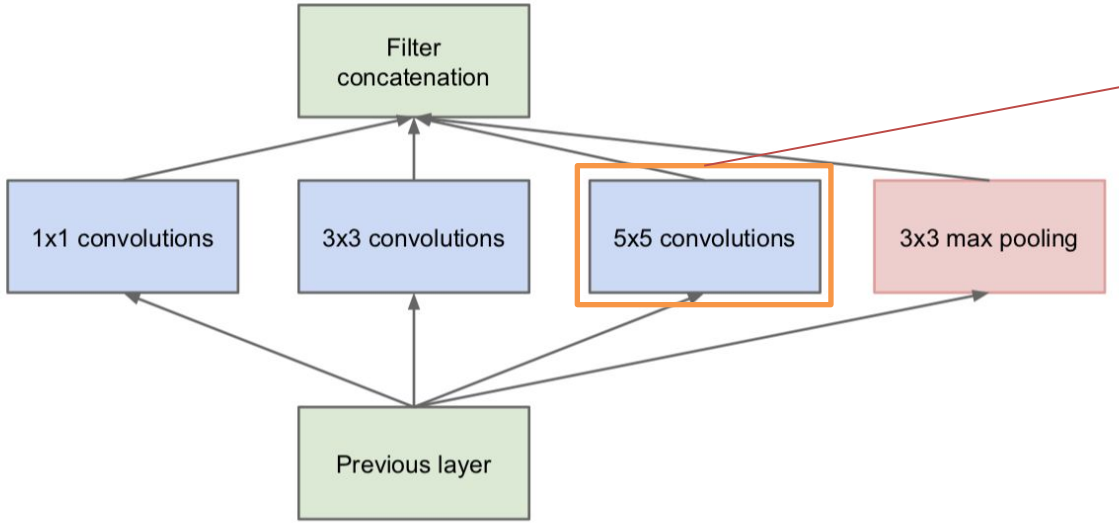




Input distribution to sigmoid, shown as {15, 50, 85}th percentiles

Experiments - ImageNet Classification

- Predicting images out of 1000 possibilities
- Inception network mini-batch size 32, ReLU as nonlinearity
- BN added to each nonlinearity

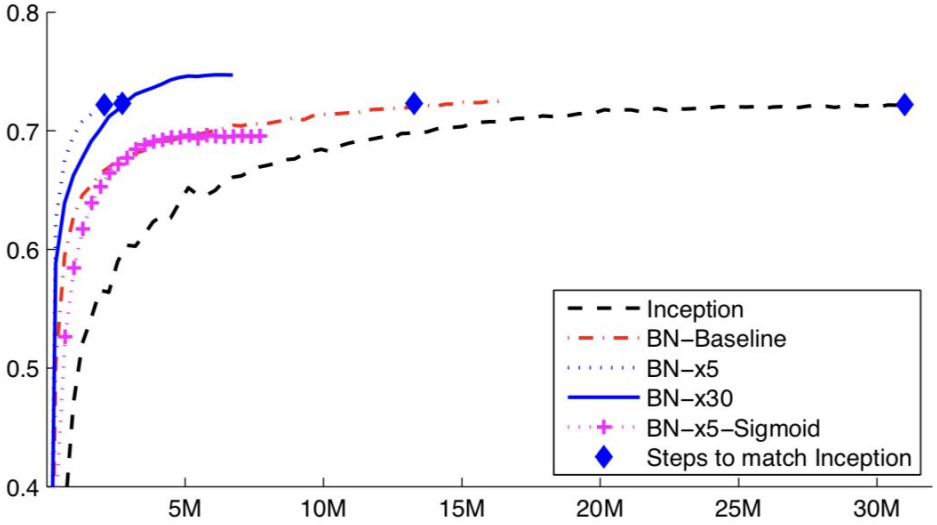


Inception module

Experiments - ImageNet Classification

- Changes made to the network and its training parameters
  - 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 distortion

Experiments - ImageNet Classification (Single-network)



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%

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.

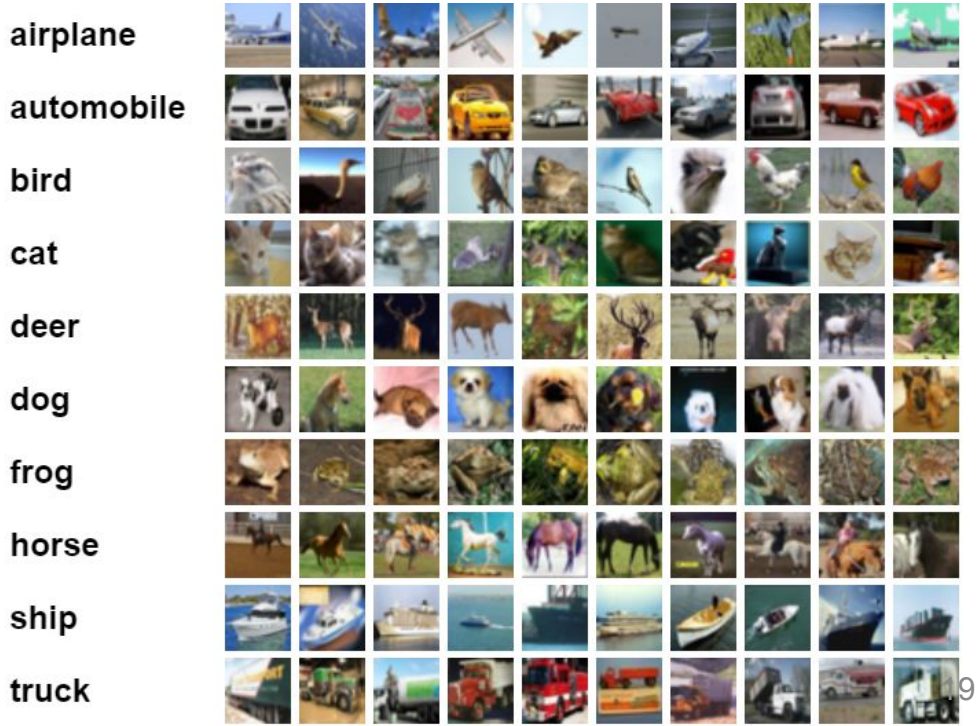
Experiments - ImageNet Classification (Ensemble)

Model	Top-1 error	Top-5 error
GoogLeNet ensemble	-	6.67%
Deep Image low-res	-	7.96%
Deep Image high-res	24.88	7.42%
Deep Image ensemble	-	5.98%
BN-Inception single crop	25.2%	7.82%
BN-Inception multicrop	21.99%	5.82%
BN-Inception ensemble	20.1%	**4.9%***

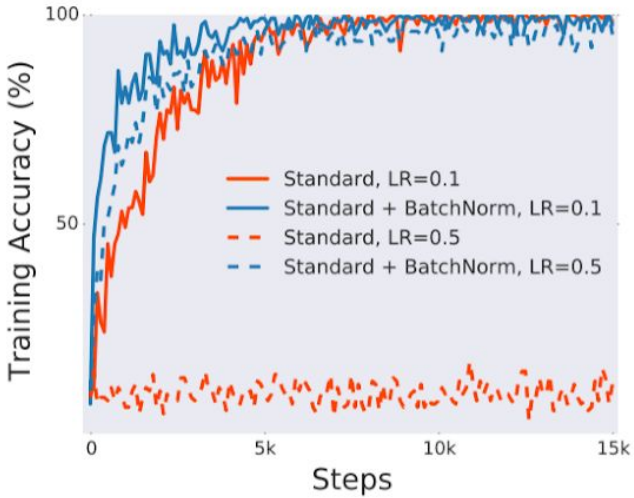
How does Batch Normalization help optimization?

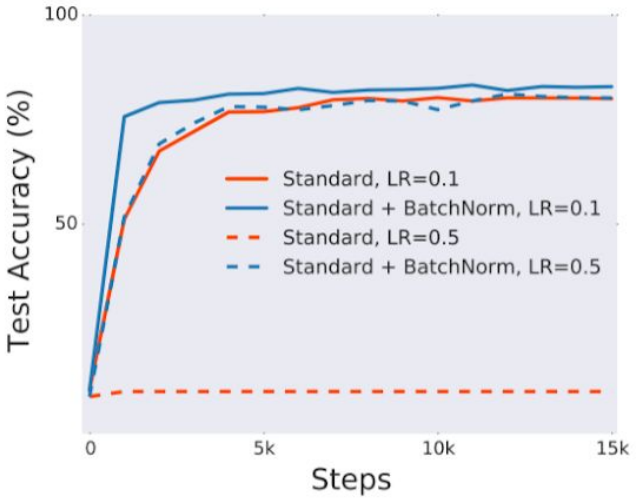
Preliminaries

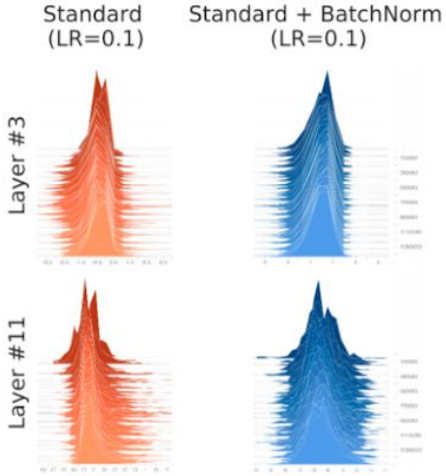
- VGG-like architecture
- Image classification on CIFAR-10
- Batch size of 128



Are BN and ICS connected?







Little concrete evidence supporting the effect of BN on ICS!

Connection between ICS and BN

(1) Is the effectiveness of BN indeed related to ICS?

(2) Is BN's stabilization of layer input distributions even effective in reducing ICS?

(1) Is the effectiveness of BN related to ICS?

Is *controlling both the mean and variance of distributions of layer inputs* directly connected to improved training performance?

Experiment: Train with random noise ***after*** BN layers.

Algorithm 1 "Noisy" BatchNorm

% **For constants** n_m, n_v, r_m, r_v.

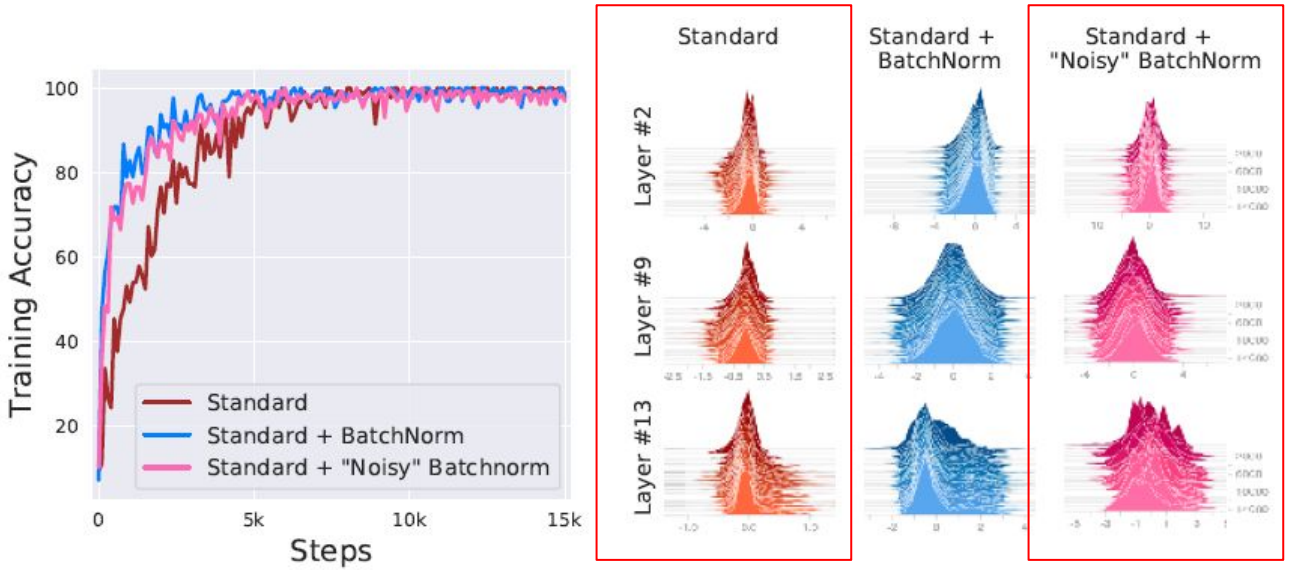
for each layer at time t **do**
  $a^t_{i,j} \leftarrow$ *Batch-normalized activation for unit j and sample i*

  for each j **do** ▷ Sample the parameters (m^t_j, v^t_j) of D^t_j from D_j
    $\mu^t \sim U(-n_\mu, n_\mu)$
    $\sigma^t \sim U(1, n_\sigma)$

  for each i **do** ▷ Sample noise from D^t_j
    for each j **do**
      $m^t_{i,j} \sim U(\mu - r_\mu, \mu + r_\mu)$
      $s^t_{i,j} \sim \mathcal{N}(\sigma, r_\sigma)$
      $a^t_{i,j} \leftarrow s^t_{i,j} \cdot a_{i,j} + m^t_{i,j}$

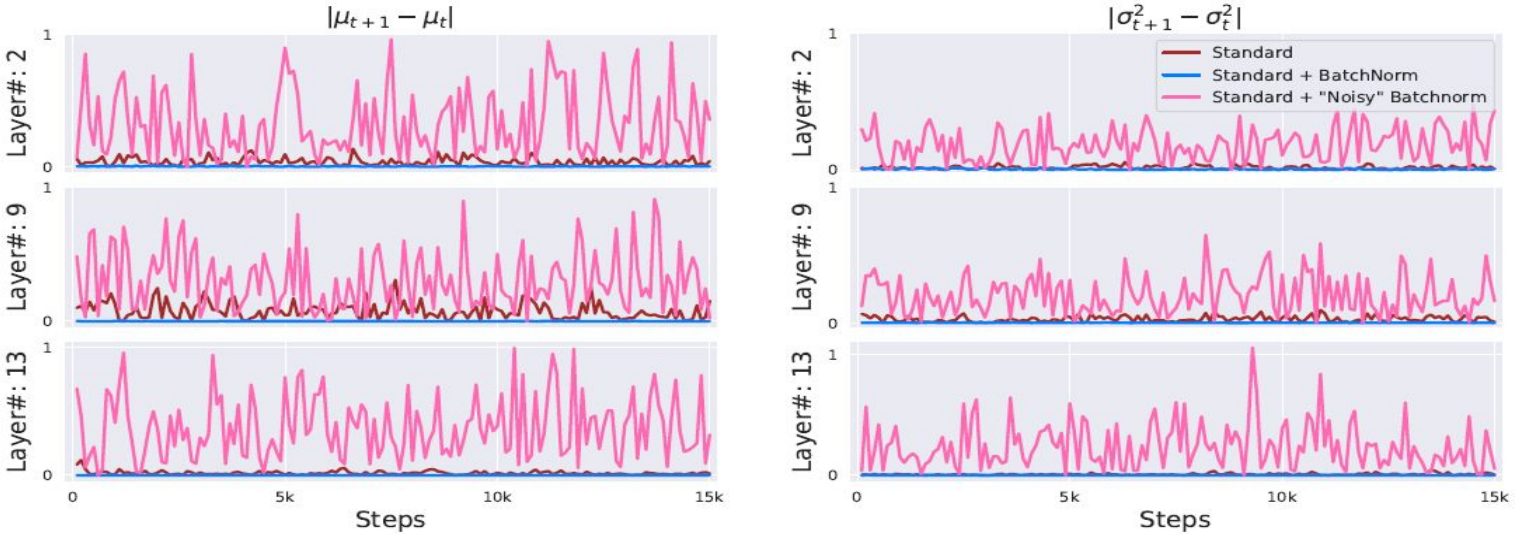
Noise drawn from different distributions

(1) Is the effectiveness of BN related to ICS? (Cont'd)



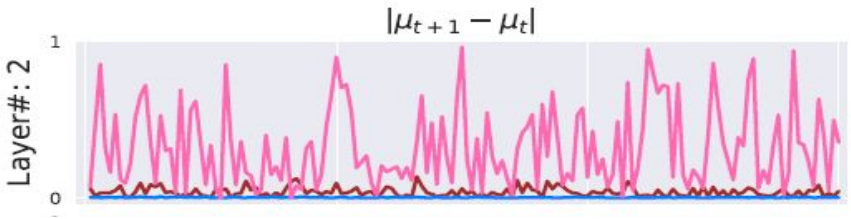
"Noisy" BN shows much more ICS than standard

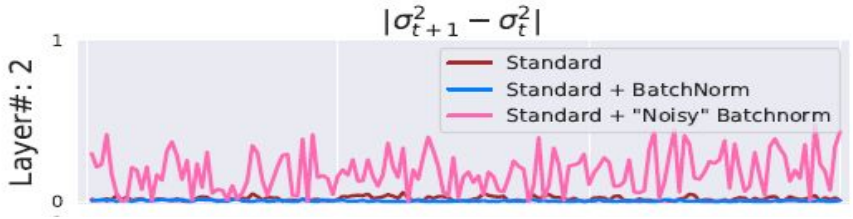
(1) Is the effectiveness of BN related to ICS? (Cont'd)



"Noisy" BN show much more ICS than standard

(1) Is the effectiveness of BN related to ICS? (Cont'd)





Large variations in mean and variance in each step for "Noisy" BN.

Connection between ICS and BN

(1) Is the effectiveness of BN indeed related to ICS?

No!

(2) Is BN's stabilization of layer input distributions even effective in reducing ICS?

(2) Is BN reducing ICS?

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*

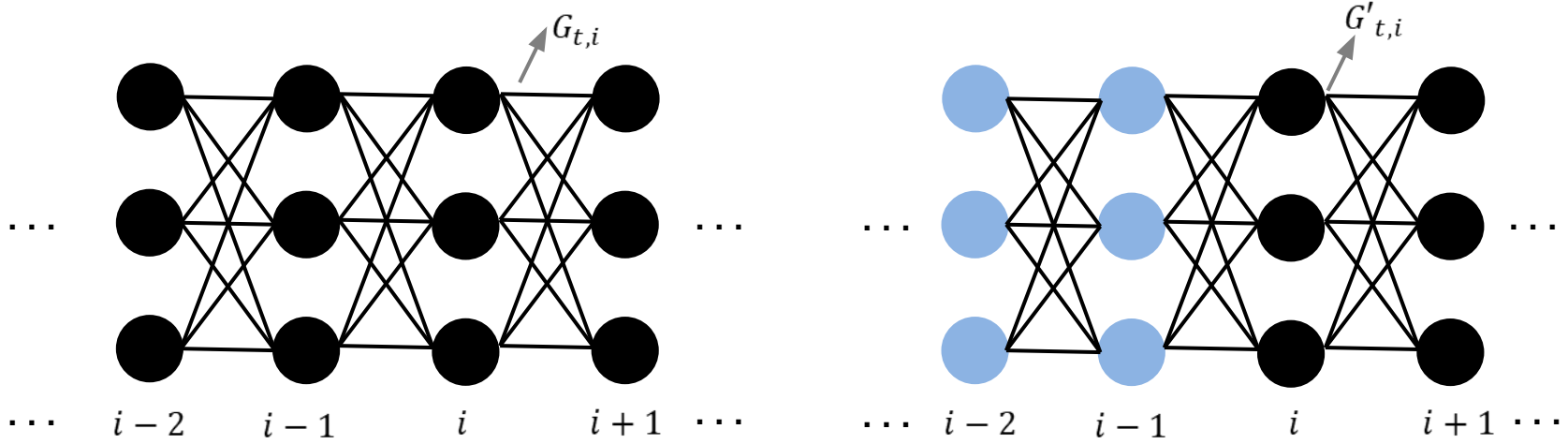
$$G_{t,i} = \nabla_{W_i^{(t)}} \mathcal{L}(W_1^{(t)}, \ldots, W_k^{(t)}; x^{(t)}, y^{(t)})$$

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

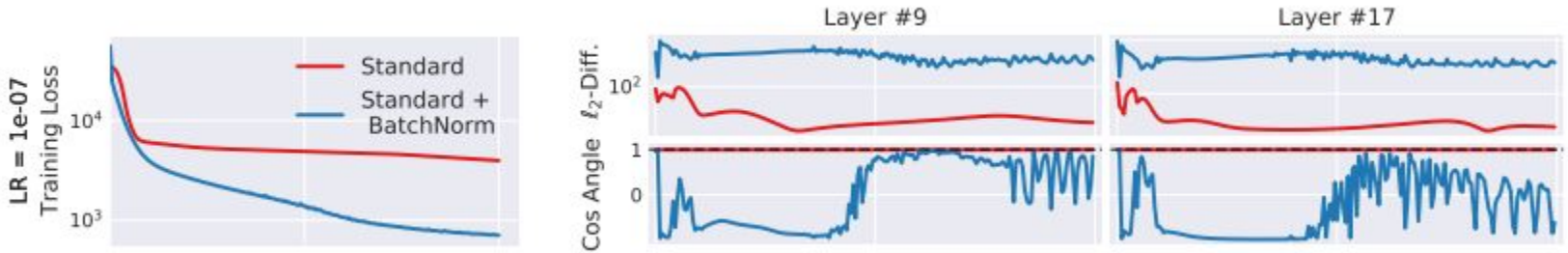
- $G_{t,i}$ = gradient of the layer parameters that would be applied during a simultaneous update of all layers.
- $G'_{t,i}$ = same gradient *after* all the previous layers have been updated with their new values.

(2) Is BN reducing ICS?

Visualizing the setup using Neural Networks



(2) Is BN reducing ICS?



DLN

- **For BN, it appears that G and G' are almost uncorrelated.**
- **Controlling the distributions layer inputs might not even reduce ICS!**

Connection between ICS and BN

(1) *Is the effectiveness of BN indeed related to ICS?*

No!

(2) *Is BN's stabilization of layer input distributions even effective in reducing ICS?*

No!

Why does BN work?

1. Smoothening effect of BN

2. Exploration of the optimization landscape

3. Is BN the best (only?) way to smoothen the landscape?

1. The smoothening effect of BN

Recall that f is L-Lipschitz if

$$|f(x_1) - f(x_2)| \leq L\|x_1 - x_2\|, \forall\, x_1, x_2$$

→ BN improves the Lipschitzness of the loss function.

↳ The loss changes at a smaller rate

Recall that f is β-smooth if

$$|\nabla f(x_1) - \nabla f(x_2)| \leq \beta\|x_1 - x_2\|, \forall\, x_1, x_2$$

→ BN's re-parametrization makes *gradients* of the loss function more Lipschitz.

↳ The loss exhibits a significantly better "effective" β-smoothness.

1. The smoothening effect of BN (Cont'd)

→ BN's re-parametrization makes gradients more reliable and predictive.

↓

Can now take a larger step in the direction of the computed gradient without the danger of having

↙ vanishing gradients or ↘ exploding gradients

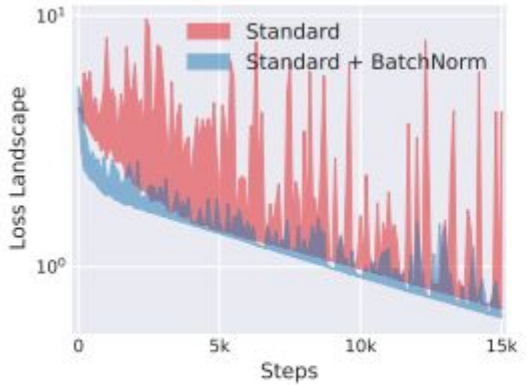
→ BN's re-parametrization makes the training significantly faster and less sensitive to hyperparameter choices.

2. Exploration of the optimization landscape

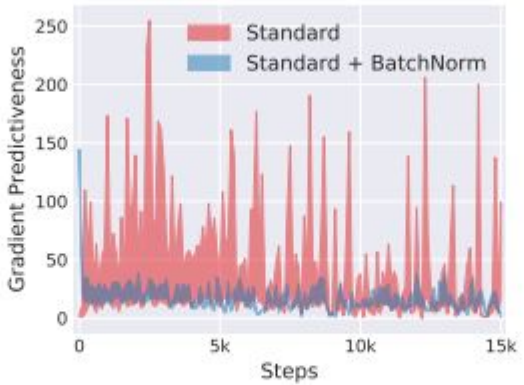
Stability of the *loss* function ↔ Lipschitzness

Goal: Demonstrate the impact of BN on the stability of the loss function.

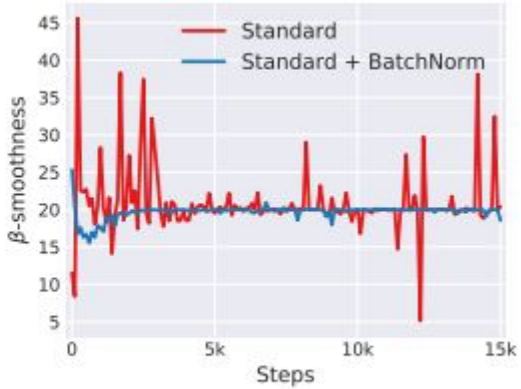
- (a) Compute the gradient of the loss at each step in the training process and measure how the loss changes as we move in that direction.
- (b) Compute the l_2 distance between the loss gradient at a given point of the training and the gradients corresponding to different points along the original gradient direction.



(a) loss landscape



(b) gradient predictiveness

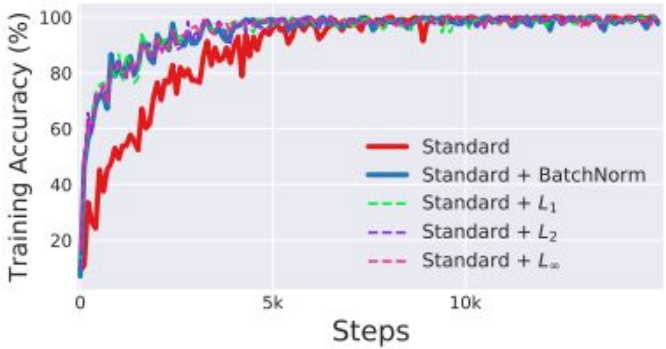


(c) "effective" β-smoothness

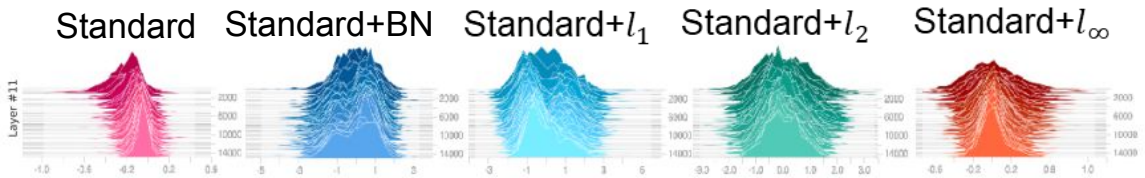
3. Is BN the best (only?) way to smoothen the landscape?

Is this smoothening effect a unique feature of BN?

- Study a few natural data statistics-based normalization strategies
  - Schemes that fix the 1st order moment, then normalize the activations by the average of their l_p-norm (for p=1,2,∞) before shifting.
    - Distributions of layer inputs are no longer Gaussian-like. ✖
    - l_p-normalization techniques lead to *larger distributional covariate shift*, yet still *yield improved optimization performance.*

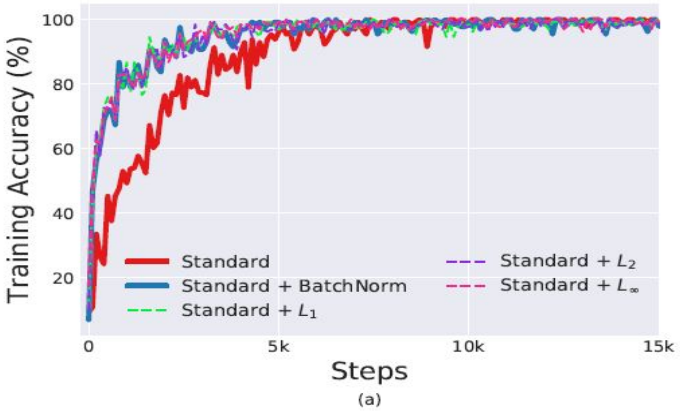


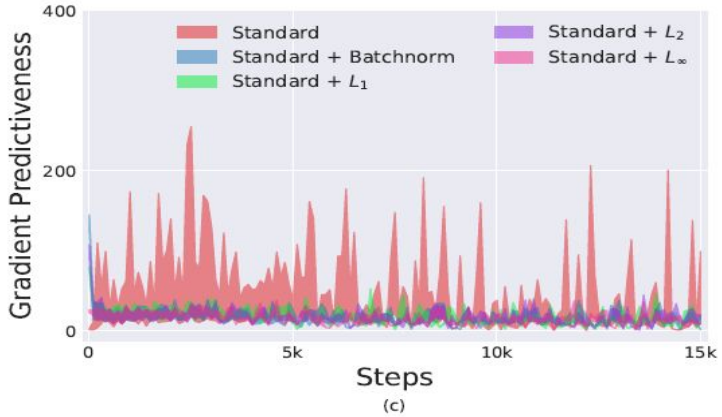
(a) VGG

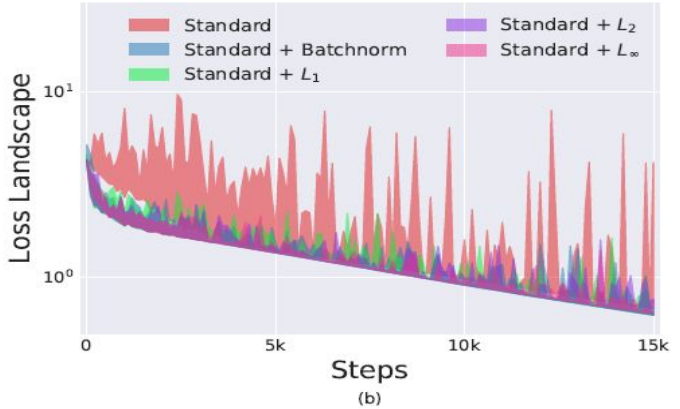


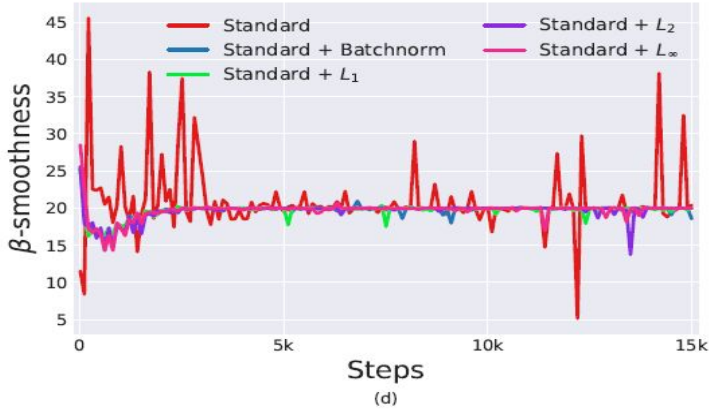
Histograms

3. Is BN the best (only?) way to smoothen the landscape?









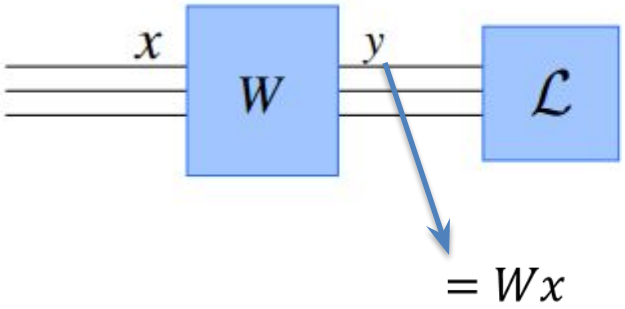
All the normalization schemes show “smoother” loss landscape.

Conclusion: BatchNorm may not be unique in its improved performance.

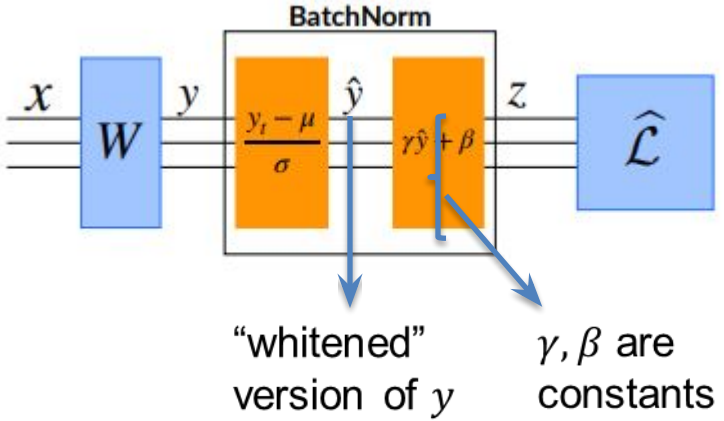
Theoretical Analysis

- Explore the effect of BN on the optimization landscape from a theoretical perspective.
- Consider an arbitrary linear layer in a Deep Neural Network (DNN).
- Compare the theoretical investigations for two network architectures: (a) the Vanilla DNN (i.e. DNN without BN); (b) same network as in (a) but with a single BN layer inserted after the fully-connected layer W.

(a) Vanilla Network



(b) Vanilla Network + Single BN layer



CAUTION: MATH AHEAD!

Theoretical Analysis

Expectation: BN causes the landscape to be more well-behaved, inducing favorable properties in **Lipschitz-continuity** and **predictability of the gradients**.

1. $\left\|\nabla_{y_j} L\right\|$ → Captures the Lipschitzness of the Loss.
2. β-smoothness → Conveys how predictive the gradient is in terms of function minimization.

1 $\left\|\nabla_{y_j} L\right\|$ → Captures the Lipschitzness of the Loss.

Theorem 4.1 (The effect of BatchNorm on the Lipschitzness of the loss). *For a BatchNorm network with loss $\widehat{\mathcal{L}}$ and an identical non-BN network with (identical) loss $\mathcal{L}$,*

$$\left\|\nabla_{y_j}\widehat{\mathcal{L}}\right\|^2 \leq \frac{\gamma^2}{\sigma_j^2}\left(\left\|\nabla_{y_j}\mathcal{L}\right\|^2 - \frac{1}{m}\left\langle \mathbf{1}, \nabla_{y_j}\mathcal{L}\right\rangle^2 - \frac{1}{m}\left\langle \nabla_{y_j}\mathcal{L}, \hat{\boldsymbol{y}}_j\right\rangle^2\right).$$

Contributes to the relative "flatness" since σ is relatively large.

Grows quadratically in the dimension

Correlation between the two terms exists → the term is positive.

Decreases significantly

2 β-smoothness → Conveys how predictive the gradient is in terms of function minimization.

Theorem 4.2 (The effect of BN to smoothness). *Let* $\hat{\boldsymbol{g}}_j = \nabla_{\boldsymbol{y}_j} \hat{\mathcal{L}}$ *and* $\boldsymbol{H}_{jj} = \frac{\partial \hat{\mathcal{L}}}{\partial \boldsymbol{y}_j \partial \boldsymbol{y}_j}$ *be the gradient and Hessian of the loss with respect to the layer outputs respectively. Then*

$$\left(\nabla_{\boldsymbol{y}_j} \hat{\mathcal{L}}\right)^\top \frac{\partial \hat{\mathcal{L}}}{\partial \boldsymbol{y}_j \partial \boldsymbol{y}_j} \left(\nabla_{\boldsymbol{y}_j} \hat{\mathcal{L}}\right) \leq \frac{\gamma^2}{\sigma^2} \left(\frac{\partial \hat{\mathcal{L}}}{\partial \boldsymbol{y}_j}\right)^\top \boldsymbol{H}_{jj} \left(\frac{\partial \hat{\mathcal{L}}}{\partial \boldsymbol{y}_j}\right) - \frac{\gamma}{m\sigma^2} \langle \hat{\boldsymbol{g}}_j, \hat{\boldsymbol{y}}_j \rangle \left\| \frac{\partial \hat{\mathcal{L}}}{\partial \boldsymbol{y}_j} \right\|^2$$

These two terms are positive

Two Conditions:

1. Hessian is PSD if the Loss is locally convex, which is true for the case of deep networks with a piecewise linear activation functions and a convex Loss at the final layer.

2. $\langle \hat{\mathbf{y}}_j , \hat{\mathbf{g}}_j \rangle > 0$ as long as the negative gradient $\hat{\mathbf{g}}_j$ is pointing towards the minimum of the Loss.

If these two conditions are satisfied, then the steps taken by the BN network are more predictive than those of the standard network.

Theorem 4.4 (Minimax bound on weight-space Lipschitzness). *For a BatchNorm network with loss* $\widehat{\mathcal{L}}$ *and an identical non-BN network (with identical loss* $\mathcal{L}$*), if*

$$g_j = \max_{||X||\leq\lambda} ||\nabla_W \mathcal{L}||^2, \qquad \hat{g}_j = \max_{||X||\leq\lambda} \left|\left|\nabla_W \widehat{\mathcal{L}}\right|\right|^2 \Longrightarrow \hat{g}_j \leq \frac{\gamma^2}{\sigma_j^2}\left(g_j - m\mu_{g_j}^2 - \lambda^2 \left\langle \nabla_{\boldsymbol{y}_j} \mathcal{L}, \hat{\boldsymbol{y}}_j \right\rangle^2\right).$$

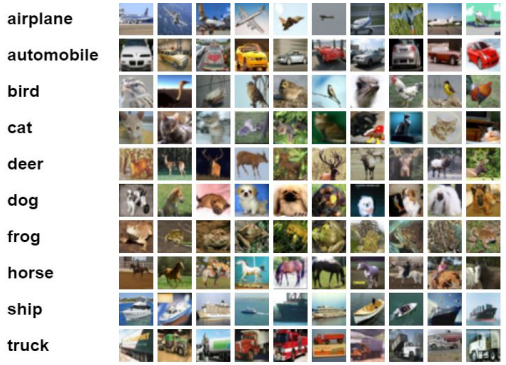
Positive terms

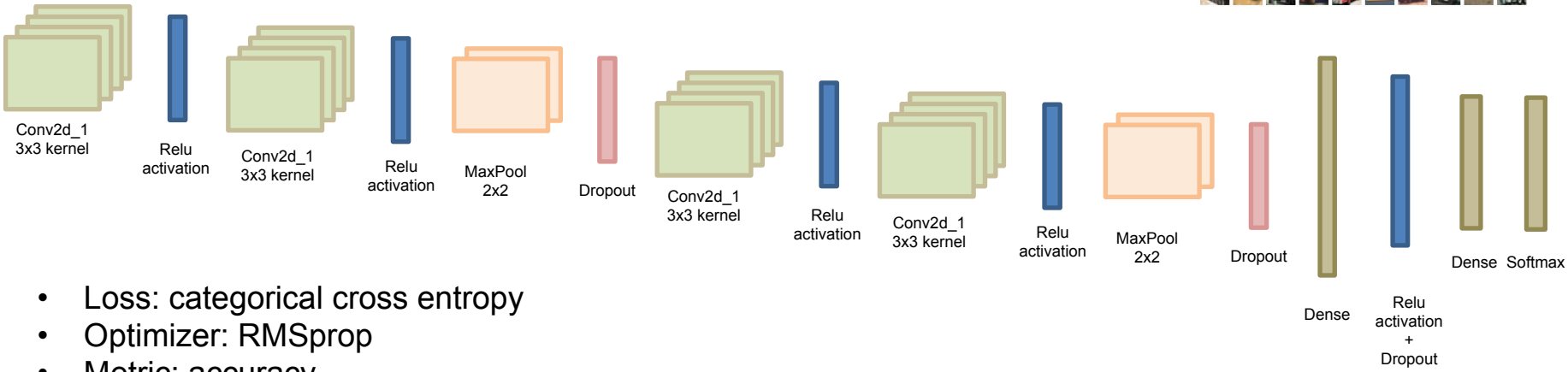
$\hat{\mathbf{g}}_j$ has a small upper bound

SIMULATIONS

Experiment

CIFAR10

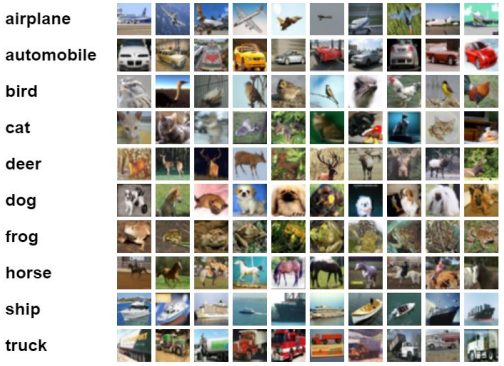


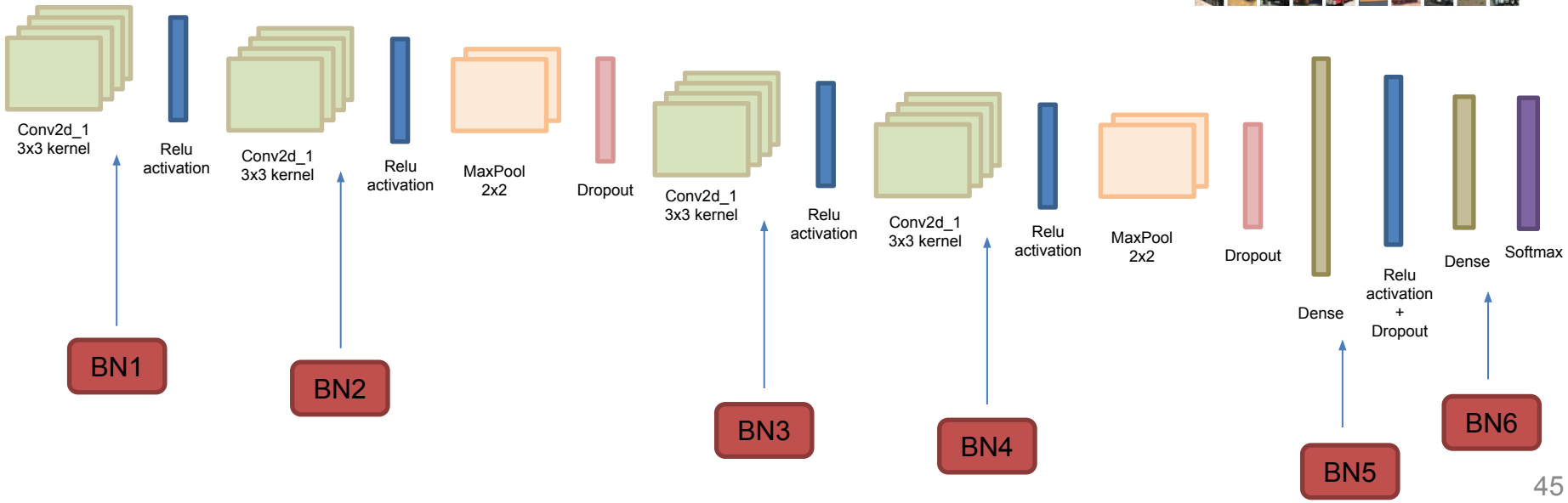


- Loss: categorical cross entropy
- Optimizer: RMSprop
- Metric: accuracy

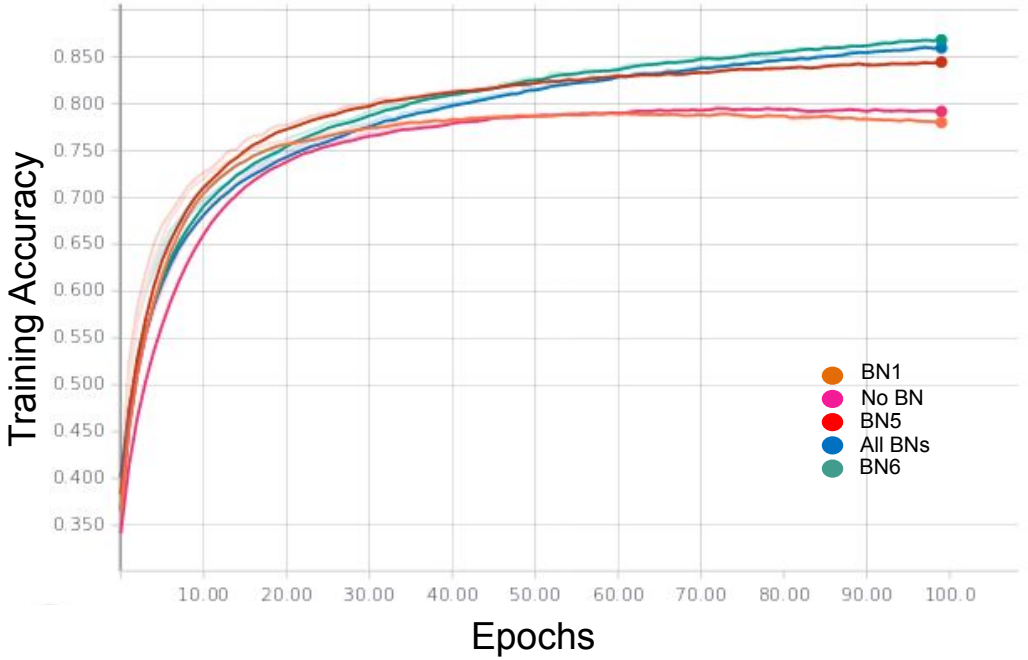
Experiment

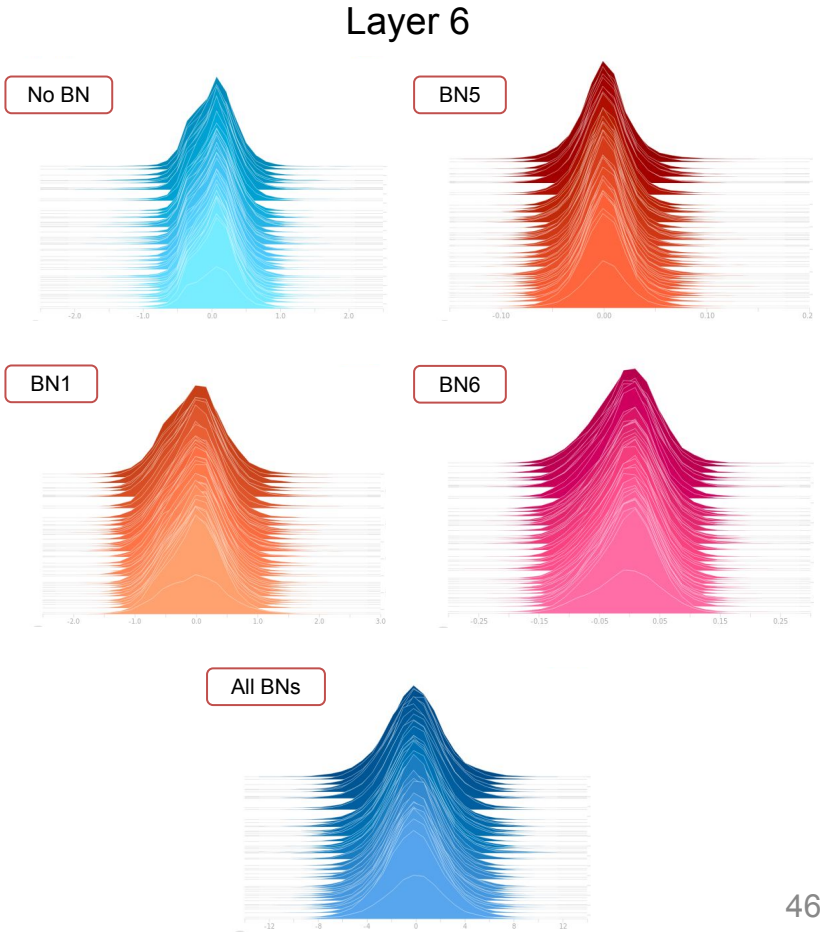
Adding Batch Normalization





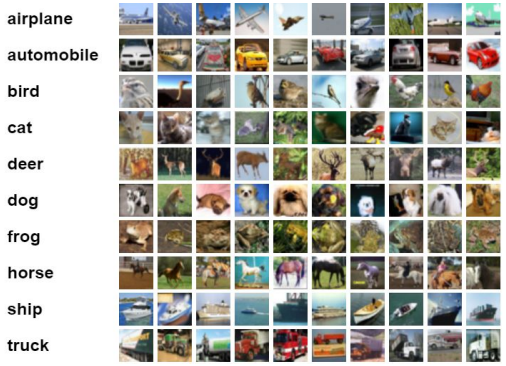
Experiment

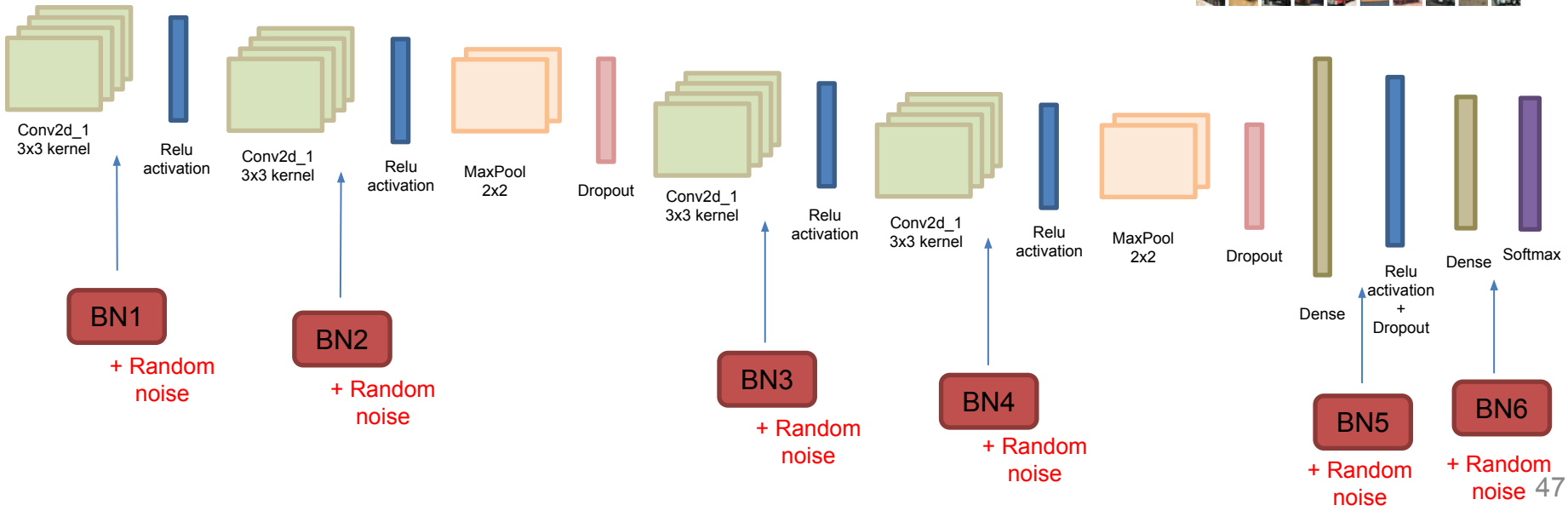

Training Accuracy
0.850
0.800
0.750
0.700
0.650
0.600
0.550
0.500
0.450
0.400
0.350
10.00
20.00
30.00
40.00
50.00
60.00
70.00
80.00
90.00
100.0
Epochs
BN1
No BN
BN5
All BNs
BN6


Layer 6
No BN
BN5
BN1
BN6
All BNs

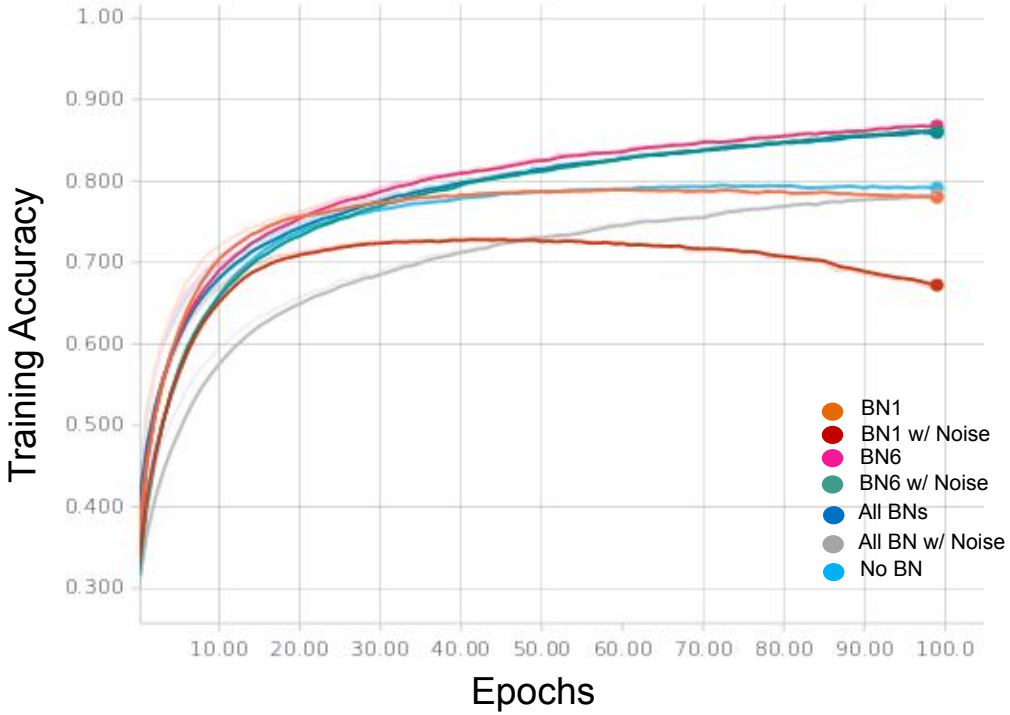
Experiment

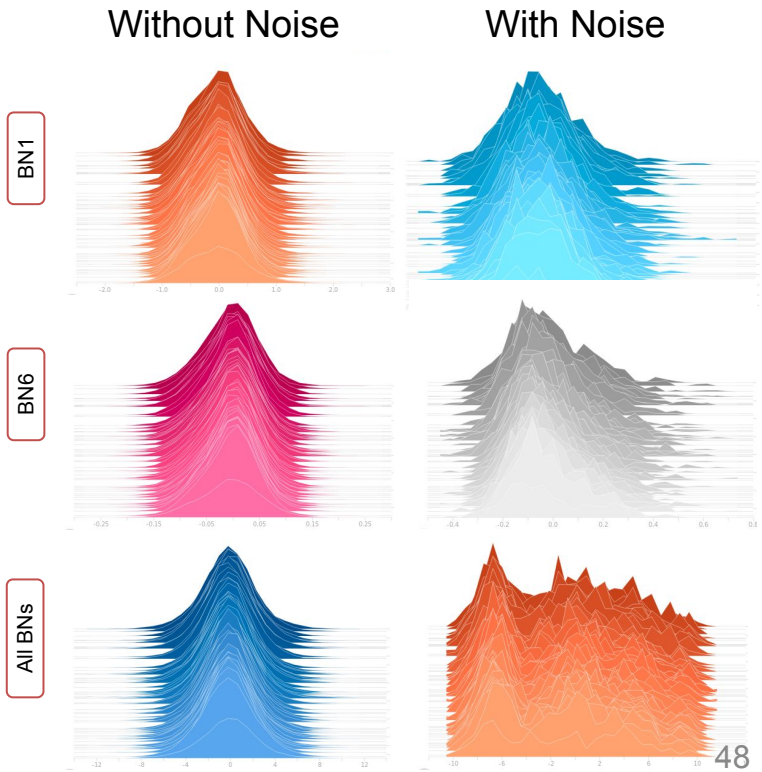
Adding Noise Layer





Experiment


Training Accuracy
1.00
0.900
0.800
0.700
0.600
0.500
0.400
0.300
10.00
20.00
30.00
40.00
50.00
60.00
70.00
80.00
90.00
100.0
Epochs
BN1
BN1 w/ Noise
BN6
BN6 w/ Noise
All BNs
All BN w/ Noise
No BN


Without Noise
With Noise
BN1
BN6
All BNs