Normalization Techniques

Devansh Arpit

Table of Contents

1. Goal
2. Key Insights
3. Batch Normalization
4. Weight Normalization
5. Scaled Weight Normalization
6. Layer Normalization
7. Analysis of Batch Normalized Networks
8. Conclusion

MILA

Improve convergence speed of SGD in deep networks

Table of Contents

1. Goal
2. Key Insights
3. Batch Normalization
4. Weight Normalization
5. Scaled Weight Normalization
6. Layer Normalization
7. Analysis of Batch Normalized Networks
8. Conclusion

MILA

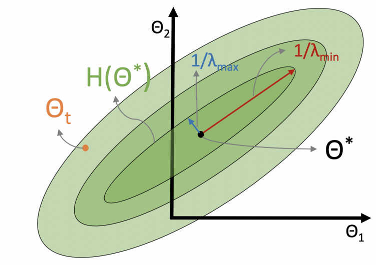
Key Insights

- Optimization perspective:
  What is the role of loss surface geometry and learning rate on the convergence speed of gradient descent?

- Covariate Shift:
  What is the role of input data distribution on convergence?

- Consider loss function $\mathcal{L}(\theta)$.
- Taylor expand around minimum θ^*,

$$\mathcal{L}(\theta) \approx \mathcal{L}(\theta^*) + \frac{1}{2}(\theta - \theta^*)^T \mathbf{H}(\theta^*)(\theta - \theta^*)$$



- Gradient w.r.t. θ

$$\frac{\partial \mathcal{L}(\theta)}{\partial \theta} \approx \mathbf{H}(\theta^*)(\theta - \theta^*)$$

- Gradient descent updates

$$\theta_{t+1} = \theta_t - \eta \mathbf{H}(\theta^*)(\theta_t - \theta^*)$$

- Change of variables

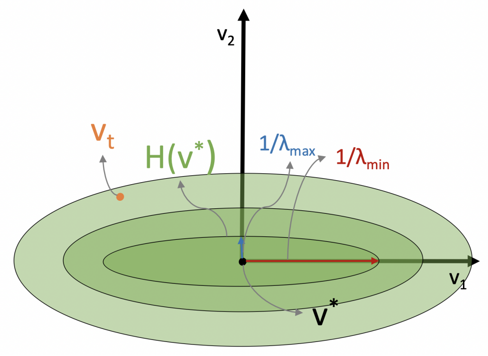
$$\mathbf{v}_t = \mathbf{U}^T(\theta_t - \theta^*)$$
$$(\mathbf{H}(\theta^*) = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T)$$

- Gradient descent updates in **v**-space

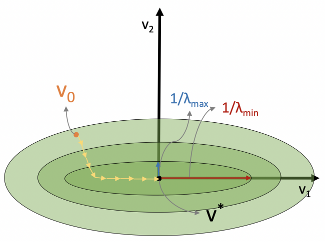
$$\mathbf{v}_{t+1} = (\mathbf{I} - \eta\boldsymbol{\Lambda})\mathbf{v}_t$$

- Update for i^{th} element

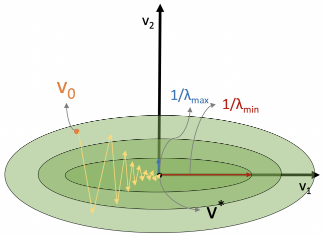
$$v_{t+1}^i = (1 - \eta\lambda_i)v_t^i$$



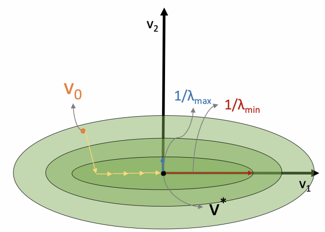
Key Insights 1/2– Optimization Perspective (Lecun, 1998)



(a) $\eta < 1/\lambda_{\text{max}}$



(b) $1/\lambda_{\text{max}} < \eta < 2/\lambda_{\text{max}}$



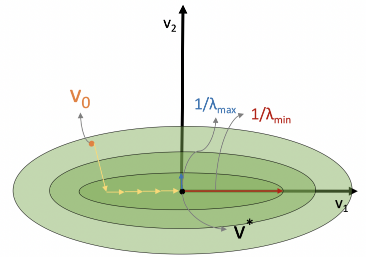
(c) $\eta = 1/\lambda_{\text{max}}$

How to do better? Common choices:

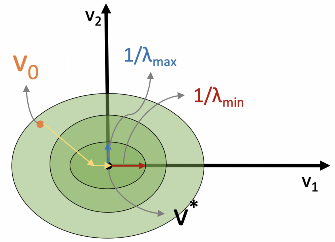
Strategy	Pros	Cons
1. Adaptive methods (Eg. ADAM)	Fast	Diagonal approximation
2. Second order methods (Eg. Newton's method)	Faster	Expensive!
3. Normalization methods	Fastest	Hard to design

Key Insights 1/2– Optimization Perspective (Lecun, 1998)

Key idea behind Normalization techniques:



(a) $\eta = 1/\lambda_{\text{max}}$ for non-isotropic loss ($\lambda_{\text{min}} << \lambda_{\text{max}}$)



(b) $\eta = 1/\lambda_{\text{max}}$ for isotropic loss ($\lambda_{\text{min}} \approx \lambda_{\text{max}}$)

Example: The perfect normalization for linear regression

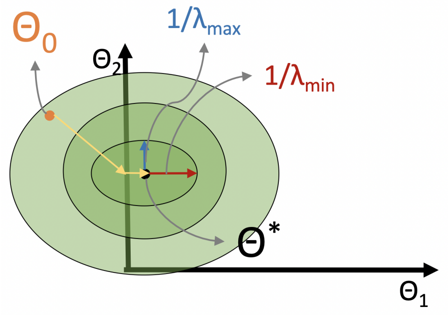
$$\mathcal{L}(\theta) := \frac{1}{2} \cdot \sum_{i=1}^{N} \|y_i - \theta^T \mathbf{x}_i\|^2$$

$$\mathbf{H}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^T$$

For $\sim$ whitened data: $\mathbf{H}(\theta) \approx c\mathbf{I}$

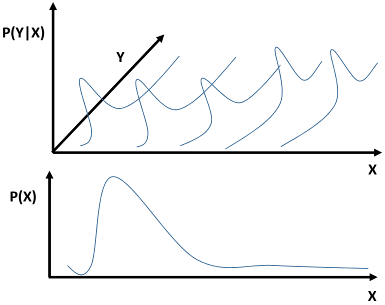
Update equation becomes:

$$\begin{aligned}\theta_{t+1} &= \theta_t - \eta \mathbf{H}(\theta^*)(\theta_t - \theta^*) \\ &\approx \theta_t - \eta c(\theta_t - \theta^*)\end{aligned}$$



- Transfer mapping from domain 1 to domain 2:

$$\mathbf{X} \xrightarrow{P(\mathbf{Y}|\mathbf{X})} \mathbf{Y}$$

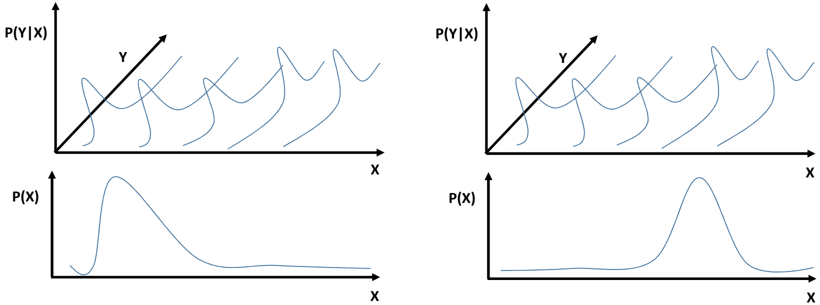


Domain 1 Domain 2

- $P(\mathbf{Y}|\mathbf{X})$ identical for both Domain
- Learn best approximation $P_\theta(\mathbf{Y}|\mathbf{X}) \approx P(\mathbf{Y}|\mathbf{X})$

Key Insights 2/2– Covariate Shift (Shimodaira, 2000)

- Transfer mapping from domain 1 to domain 2:
$\mathbf{X} \xrightarrow{P(\mathbf{Y}|\mathbf{X})} \mathbf{Y}$

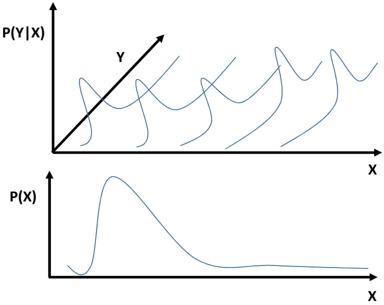


Domain 1 Domain 2

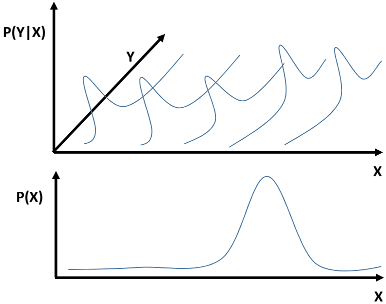
- $P(\mathbf{Y}|\mathbf{X})$ identical for both Domain
- Learn best approximation $P_\theta(\mathbf{Y}|\mathbf{X}) \approx P(\mathbf{Y}|\mathbf{X})$
- Maximum Likelihood focuses on high density regions of $\mathbf{X}$

Key Insights 2/2– Covariate Shift (Shimodaira, 2000)

- Transfer mapping from domain 1 to domain 2:
  $\mathbf{X} \xrightarrow{P(\mathbf{Y}|\mathbf{X})} \mathbf{Y}$



Domain 1



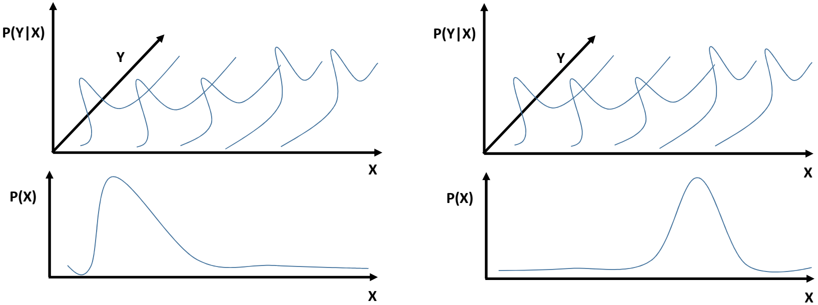
Domain 2

- $P(\mathbf{Y}|\mathbf{X})$ identical for both Domain
- Learn best approximation $P_\theta(\mathbf{Y}|\mathbf{X}) \approx P(\mathbf{Y}|\mathbf{X})$
- Maximum Likelihood focuses on high density regions of $\mathbf{X}$
- Low density regions of $\mathbf{X}$ are artifacts in $P_{\theta^*}(\mathbf{Y}|\mathbf{X})$

Key Insights 2/2– Covariate Shift (Shimodaira, 2000)

- Transfer mapping from domain 1 to domain 2:

$$\mathbf{X} \xrightarrow{P(\mathbf{Y}|\mathbf{X})} \mathbf{Y}$$



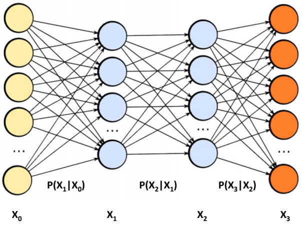
Domain 1 Domain 2

- $P(\mathbf{Y}|\mathbf{X})$ identical for both Domain
- Learn best approximation $P_{\theta}^{*}(\mathbf{Y}|\mathbf{X}) \approx P(\mathbf{Y}|\mathbf{X})$
- Maximum Likelihood focuses on high density regions of $\mathbf{X}$
- Low density regions of $\mathbf{X}$ are artifacts in $P_{\theta^*}(\mathbf{Y}|\mathbf{X})$
- $P_{\theta^*}(\mathbf{Y}|\mathbf{X})$ for Domain 1 $\neq P_{\theta^*}(\mathbf{Y}|\mathbf{X})$ for Domain 2

Key Insights 2/2– Covariate Shift

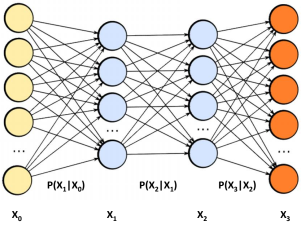
- Internal Covariate Shift (Ioffe and Szegedy, 2015)
  $\mathbf{X}_i \xrightarrow{P(\mathbf{X}_{i+1}|\mathbf{X}_i)} \mathbf{X}_{i+1}$

- Multi-layer end-to-end learning model



- Learn the best approximation $P_\theta(\mathbf{X}_{i+1}|\mathbf{X}_i) \approx P(\mathbf{X}_{i+1}|\mathbf{X}_i)$

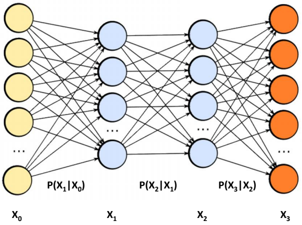
- Internal Covariate Shift (Ioffe and Szegedy, 2015)
  $\mathbf{X}_i \xrightarrow{P(\mathbf{X}_{i+1}|\mathbf{X}_i)} \mathbf{X}_{i+1}$
- Multi-layer end-to-end learning model



- Learn the best approximation $P_\theta(\mathbf{X}_{i+1}|\mathbf{X}_i) \approx P(\mathbf{X}_{i+1}|\mathbf{X}_i)$
- During SGD updates, hidden layer $P(\mathbf{X}_i)$ keeps shifting

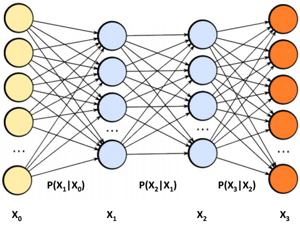
Key Insights 2/2– Covariate Shift

- Internal Covariate Shift (Ioffe and Szegedy, 2015)
  $\mathbf{X}_i \xrightarrow{P(\mathbf{X}_{i+1}|\mathbf{X}_i)} \mathbf{X}_{i+1}$
- Multi-layer end-to-end learning model



- Learn the best approximation $P_\theta(\mathbf{X}_{i+1}|\mathbf{X}_i) \approx P(\mathbf{X}_{i+1}|\mathbf{X}_i)$
- During SGD updates, hidden layer $P(\mathbf{X}_i)$ keeps shifting
- $\implies$ target $P_{\theta*}(\mathbf{X}_{i+1}|\mathbf{X}_i)$ keeps shifting

- Internal Covariate Shift (Ioffe and Szegedy, 2015)
  $\mathbf{X}_i \xrightarrow{P(\mathbf{X}_{i+1}|\mathbf{X}_i)} \mathbf{X}_{i+1}$
- Multi-layer end-to-end learning model



- Learn the best approximation $P_\theta(\mathbf{X}_{i+1}|\mathbf{X}_i) \approx P(\mathbf{X}_{i+1}|\mathbf{X}_i)$
- During SGD updates, hidden layer $P(\mathbf{X}_i)$ keeps shifting
- $\implies$ target $P_{\theta^*}(\mathbf{X}_{i+1}|\mathbf{X}_i)$ keeps shifting
- Learning $P_\theta(\mathbf{X}_{i+1}|\mathbf{X}_i)$ using SGD is slow

Key Insights

Take home message:

- Smooth loss surface
- Intuitions from linear regression and covariate shift:
  - Zero input mean
  - Isotropic input variance

Table of Contents

1. Goal
2. Key Insights
3. Batch Normalization
4. Weight Normalization
5. Scaled Weight Normalization
6. Layer Normalization
7. Analysis of Batch Normalized Networks
8. Conclusion

Batch Normalization (Ioffe and Szegedy, 2015)

- Key idea:
  - Normalize input statistics over mini-batch

Batch Normalization (Ioffe and Szegedy, 2015)

- Consider any hidden unit's pre-activation a_i
- Then normalized pre-activation under BN is given by:

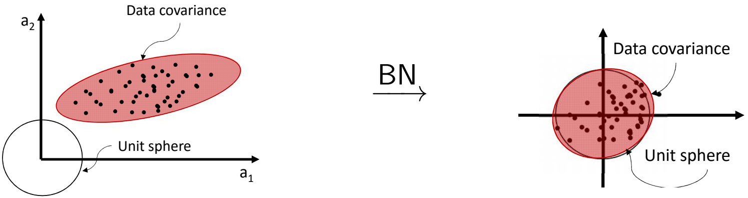
$$\mathrm{BN}(a_i) = \frac{(a_i - \mathbb{E}_{\mathcal{B}}[a_i]))}{\sqrt{\mathrm{var}_{\mathcal{B}}(a_i)}}$$

MILA

Batch Normalization (Ioffe and Szegedy, 2015)

- Consider any hidden unit's pre-activation a_i
- Then normalized pre-activation under BN is given by:

$$\text{BN}(a_i) = \frac{(a_i - \mathbb{E}_{\mathcal{B}}[a_i]))}{\sqrt{\text{var}_{\mathcal{B}}(a_i)}}$$



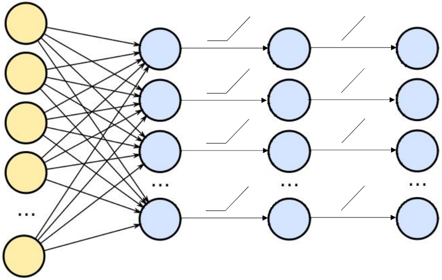
- Pretty close!

(approximation is worst when principal components are maximally away from axis)

Batch Normalization (Ioffe and Szegedy, 2015)

- A traditional vs. Batch Normalized (BN) ReLU layer:

Traditional: $\mathbf{x}$ $\quad a_i = \mathbf{W}_i^T \mathbf{x} + \beta_i$ $\quad \mathbf{h} = \mathrm{ReLU}(\mathbf{a})$ $\quad \mathbf{o} = \mathbf{h}$

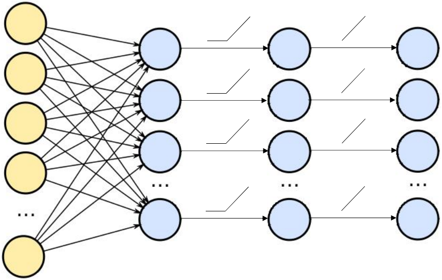


BN: $\mathbf{x}$ $\quad \hat{a}_i = \frac{\gamma_i(\mathbf{W}_i^T(\mathbf{x} - \mathbb{E}_{\mathcal{B}}[\mathbf{x}]))}{\sqrt{\mathrm{var}_{\mathcal{B}}(\mathbf{W}_i^T\mathbf{x})}} + \beta_i$ $\quad \mathbf{h} = \mathrm{ReLU}(\hat{\mathbf{a}})$ $\quad \mathbf{o} = \mathbf{h}$

Batch Normalization (Ioffe and Szegedy, 2015)

- A traditional vs. Batch Normalized (BN) ReLU layer:

Traditional: $\mathbf{x}$ $\quad a_i = \mathbf{W}_i^T \mathbf{x} + \beta_i \quad \mathbf{h} = \text{ReLU}(\mathbf{a}) \quad \mathbf{o} = \mathbf{h}$

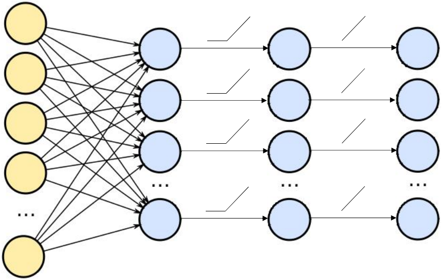


BN: $\mathbf{x}$ $\quad \hat{a}_i = \frac{\gamma_i (\mathbf{W}_i^T (\mathbf{x} - \mathbb{E}_{\mathcal{B}}[\mathbf{x}]))}{\sqrt{\text{var}_{\mathcal{B}}(\mathbf{W}_i^T \mathbf{x})}} + \beta_i \quad \mathbf{h} = \text{ReLU}(\hat{\mathbf{a}}) \quad \mathbf{o} = \mathbf{h}$

- γ_i and β_i are learnable scale and bias parameters

- A traditional vs. Batch Normalized (BN) ReLU layer:

Traditional: $\mathbf{x}$ $\quad a_i = \mathbf{W}_i^T \mathbf{x} + \beta_i$ $\quad \mathbf{h} = \mathrm{ReLU}(\mathbf{a})$ $\quad \mathbf{o} = \mathbf{h}$

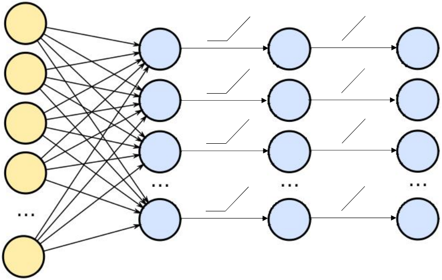


BN: $\mathbf{x}$ $\quad \hat{a}_i = \frac{\gamma_i(\mathbf{W}_i^T(\mathbf{x} - \mathbb{E}_{\mathcal{B}}[\mathbf{x}]))}{\sqrt{\mathrm{var}_{\mathcal{B}}(\mathbf{W}_i^T \mathbf{x})}} + \beta_i$ $\quad \mathbf{h} = \mathrm{ReLU}(\hat{\mathbf{a}})$ $\quad \mathbf{o} = \mathbf{h}$

- γ_i and β_i are learnable scale and bias parameters
- Notice bias β_i is outside the normalization (why?)

- A traditional vs. Batch Normalized (BN) ReLU layer:

Traditional: $\mathbf{x}$ $\quad a_i = \mathbf{W}_i^T\mathbf{x} + \beta_i \quad \mathbf{h} = \mathrm{ReLU}(\mathbf{a}) \quad \mathbf{o} = \mathbf{h}$



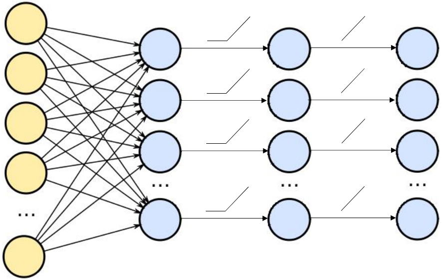
BN: $\mathbf{x}$ $\quad \hat{a}_i = \frac{\gamma_i(\mathbf{W}_i^T(\mathbf{x} - \mathbb{E}_{\mathcal{B}}[\mathbf{x}]))}{\sqrt{\mathrm{var}_{\mathcal{B}}(\mathbf{W}_i^T\mathbf{x})}} + \beta_i \quad \mathbf{h} = \mathrm{ReLU}(\hat{\mathbf{a}}) \quad \mathbf{o} = \mathbf{h}$

- γ_i and β_i are learnable scale and bias parameters
- Notice bias β_i is outside the normalization (why?)
- Why not apply BN post-activation ? (hint: Gaussianity)

Batch Normalization (Ioffe and Szegedy, 2015)

- A traditional vs. Batch Normalized (BN) ReLU layer:

Traditional: $\mathbf{x}$ $\quad a_i = \mathbf{W}_i^T\mathbf{x} + \beta_i$ $\quad \mathbf{h} = \text{ReLU}(\mathbf{a})$ $\quad \mathbf{o} = \mathbf{h}$

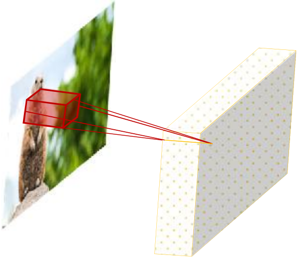


BN: $\mathbf{x}$ $\quad \hat{a}_i = \frac{\gamma_i(\mathbf{W}_i^T(\mathbf{x} - \mathbb{E}_{\mathcal{B}}[\mathbf{x}]))}{\sqrt{\text{var}_{\mathcal{B}}(\mathbf{W}_i^T\mathbf{x})}} + \beta_i$ $\quad \mathbf{h} = \text{ReLU}(\hat{\mathbf{a}})$ $\quad \mathbf{o} = \mathbf{h}$

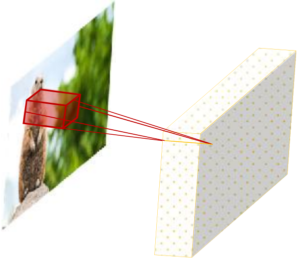
- γ_i and β_i are learnable scale and bias parameters
- Notice bias β_i is outside the normalization (why?)
- Why not apply BN post-activation ? (hint: Gaussianity)
- During test time– use running average estimates of mean and standard deviation

Batch Normalization (Ioffe and Szegedy, 2015)

- How to extend to convolutional layers?
  Simply treat each depth vector (yellow line) as a separate sample



- How to extend to convolutional layers?
  Simply treat each depth vector (yellow line) as a separate sample



- Back-propagate through the normalization

Effects of batch normalization:

- Parameter scaling:
  - Representations become scale invariant
    $\mathrm{BN}(c\mathbf{W}\mathbf{x}) = \mathrm{BN}(\mathbf{W}\mathbf{x})$
  - Gradients become inversely proportional to parameter scale
    $\frac{\partial \mathrm{BN}(c\mathbf{W}\mathbf{x})}{\partial c\mathbf{W}} = (1/c)\frac{\partial \mathrm{BN}(\mathbf{W}\mathbf{x})}{\partial \mathbf{W}}$
  - Robust to choice of learning rate

Effects of batch normalization:

- Parameter scaling:
  - Representations become scale invariant
    $\text{BN}(c\mathbf{W}\mathbf{x}) = \text{BN}(\mathbf{W}\mathbf{x})$
  - Gradients become inversely proportional to parameter scale
    $\frac{\partial \text{BN}(c\mathbf{W}\mathbf{x})}{\partial c\mathbf{W}} = (1/c)\frac{\partial \text{BN}(\mathbf{W}\mathbf{x})}{\partial \mathbf{W}}$
  - Robust to choice of learning rate
- Regularization effect:
  - Not clear why

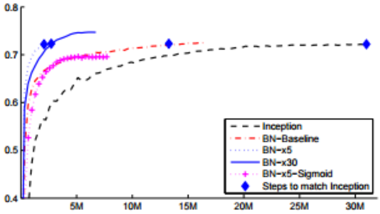


Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

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.*

Figures taken from Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." ICML (2015).

Table of Contents

1. Goal
2. Key Insights
3. Batch Normalization
4. Weight Normalization
5. Scaled Weight Normalization
6. Layer Normalization
7. Analysis of Batch Normalized Networks
8. Conclusion

Weight Normalization (Salimans and Kingma, 2016)

Key idea:

- Decouple the scale and direction of weights
- Initialize parameters to have 0 mean unit variance pre-activations
- Back-propagate through the normalization

Weight Normalization (Salimans and Kingma, 2016)

- Consider any hidden unit's pre-activation a_i
- Then weight normalized pre-activation is given by:

$$\text{WeighNorm}(a_i) = \frac{\gamma_i(\mathbf{W}_i^T \mathbf{x})}{\|\mathbf{W}_i\|_2} + \beta_i$$

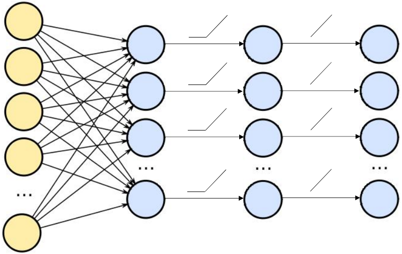
Initialize: $\gamma_i = \frac{1}{\sqrt{\text{var}_{\mathcal{B}}(\mathbf{W}_i^T \mathbf{x})/\|\mathbf{W}_i\|_2}} \quad \beta_i = -\frac{\mathbb{E}_{\mathcal{B}}[\mathbf{W}_i^T \mathbf{x}/\|\mathbf{W}_i\|_2]}{\sqrt{\text{var}_{\mathcal{B}}(\mathbf{W}_i^T \mathbf{x})/\|\mathbf{W}_i\|_2}}$

- Equivalently pre-activations get normalized using 1 mini-batch initially
- Optimization doesn't have explicit mean and variance normalization

MILA

- A traditional vs. Weight Normalized ReLU layer:

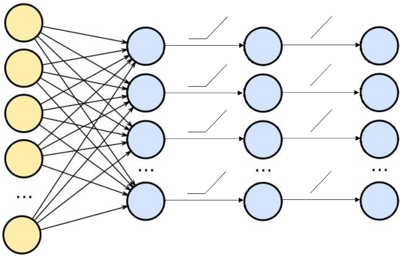
Traditional: $\mathbf{x}$ $\quad a_i = \mathbf{W}_i^T \mathbf{x} + \beta_i$ $\quad \mathbf{h} = \mathrm{ReLU}(\mathbf{a})$ $\quad \mathbf{o} = \mathbf{h}$



WeightNorm: $\mathbf{x}$ $\quad \hat{a}_i = \frac{\gamma_i (\mathbf{W}_i^T \mathbf{x})}{\|\mathbf{W}_i\|_2} + \beta_i$ $\quad \mathbf{h} = \mathrm{ReLU}(\hat{\mathbf{a}})$ $\quad \mathbf{o} = \mathbf{h}$

- A traditional vs. Weight Normalized ReLU layer:

Traditional: $\mathbf{x}$ $\quad a_i = \mathbf{w}_i^T \mathbf{x} + \beta_i$ $\quad \mathbf{h} = \mathrm{ReLU}(\mathbf{a})$ $\quad \mathbf{o} = \mathbf{h}$



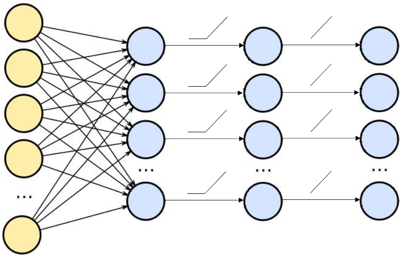
WeightNorm: $\mathbf{x}$ $\quad \hat{a}_i = \frac{\gamma_i(\mathbf{w}_i^T \mathbf{x})}{\|\mathbf{w}_i\|_2} + \beta_i$ $\quad \mathbf{h} = \mathrm{ReLU}(\hat{\mathbf{a}})$ $\quad \mathbf{o} = \mathbf{h}$

- γ_i and β_i are learnable scale and bias parameters

Weight Normalization (Salimans and Kingma, 2016)

- A traditional vs. Weight Normalized ReLU layer:

Traditional: $\mathbf{x}$ $\quad a_i = \mathbf{w}_i^T \mathbf{x} + \beta_i$ $\quad \mathbf{h} = \mathrm{ReLU}(\mathbf{a})$ $\quad \mathbf{o} = \mathbf{h}$

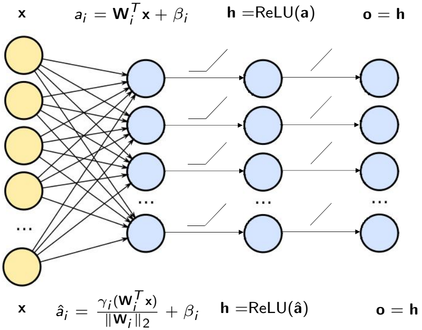


WeightNorm: $\mathbf{x}$ $\quad \hat{a}_i = \frac{\gamma_i(\mathbf{w}_i^T \mathbf{x})}{\|\mathbf{w}_i\|_2} + \beta_i$ $\quad \mathbf{h} = \mathrm{ReLU}(\hat{\mathbf{a}})$ $\quad \mathbf{o} = \mathbf{h}$

- γ_i and β_i are learnable scale and bias parameters
- train and test normalizations identical

- A traditional vs. Weight Normalized ReLU layer:

Traditional:



WeightNorm:

- γ_i and β_i are learnable scale and bias parameters
- train and test normalizations identical
- Applicable with batch-size 1

Effect of backpropagating through normalization:

- Let $\tilde{\mathbf{w}} = \frac{\gamma \mathbf{w}}{\|\mathbf{w}\|_2}$, then using SGD:

$$\Delta \mathbf{w} = -\eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}}$$
$$= -\eta \frac{\gamma}{\|\mathbf{w}^t\|_2} \left(\mathbf{I} - \frac{\mathbf{w}^t \mathbf{w}^{tT}}{\|\mathbf{w}^t\|^2} \right) \frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{w}}}$$

- Thus, $\mathbf{w} \perp \Delta \mathbf{w}$

Effect of backpropagating through normalization:

- Let $\tilde{\mathbf{w}} = \frac{\gamma \mathbf{w}}{\|\mathbf{w}\|_2}$, then using SGD:

$$\begin{aligned}\Delta \mathbf{w} &= -\eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}} \\ &= -\eta \frac{\gamma}{\|\mathbf{w}^t\|_2} \left(\mathbf{I} - \frac{\mathbf{w}^t \mathbf{w}^{tT}}{\|\mathbf{w}^t\|^2} \right) \frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{w}}}\end{aligned}$$

- Thus, $\mathbf{w} \perp \Delta \mathbf{w}$
- Let $c = \|\Delta \mathbf{w}\| / \|\mathbf{w}\|$, notice: $\|\mathbf{w}^{t+1}\| = \sqrt{(1 + c^2)} \|\mathbf{w}^t\|$

Weight Normalization (Salimans and Kingma, 2016)

Effect of backpropagating through normalization:

- Let $\tilde{\mathbf{w}} = \frac{\gamma \mathbf{w}}{\|\mathbf{w}\|_2}$, then using SGD:

$$\Delta \mathbf{w} = -\eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}}$$
$$= -\eta \frac{\gamma}{\|\mathbf{w}^t\|_2} \left(\mathbf{I} - \frac{\mathbf{w}^t \mathbf{w}^{tT}}{\|\mathbf{w}^t\|^2} \right) \frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{w}}}$$

- Thus, $\mathbf{w} \perp \Delta \mathbf{w}$
- Let $c = \|\Delta \mathbf{w}\| / \|\mathbf{w}\|$, notice: $\|\mathbf{w}^{t+1}\| = \sqrt{(1 + c^2)} \|\mathbf{w}^t\|$
- Large $c \implies$ large $\|\mathbf{w}^{t+1}\| \implies$ small gradient $\Delta \mathbf{w}$

MILA

Weight Normalization (Salimans and Kingma, 2016)

Effect of backpropagating through normalization:

- Let $\tilde{\mathbf{w}} = \frac{\gamma \mathbf{w}}{\|\mathbf{w}\|_2}$, then using SGD:

$$\begin{aligned}\Delta \mathbf{w} &= -\eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}} \\ &= -\eta \frac{\gamma}{\|\mathbf{w}^t\|_2} \left(\mathbf{I} - \frac{\mathbf{w}^t \mathbf{w}^{tT}}{\|\mathbf{w}^t\|^2} \right) \frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{w}}}\end{aligned}$$

- Thus, $\mathbf{w} \perp \Delta \mathbf{w}$
- Let $c = \|\Delta \mathbf{w}\| / \|\mathbf{w}\|$, notice: $\|\mathbf{w}^{t+1}\| = \sqrt{(1 + c^2)} \|\mathbf{w}^t\|$
- Large $c \implies$ large $\|\mathbf{w}^{t+1}\| \implies$ small gradient $\Delta \mathbf{w}$
- Small $c \implies \|\mathbf{w}^{t+1}\| \approx \|\mathbf{w}^t\| \implies$ gradient scale $\Delta \mathbf{w}$ unchanged

Effect of backpropagating through normalization:

- Let $\tilde{\mathbf{w}} = \frac{\gamma \mathbf{w}}{\|\mathbf{w}\|_2}$, then using SGD:

$$\Delta \mathbf{w} = -\eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}}$$
$$= -\eta \frac{\gamma}{\|\mathbf{w}^t\|_2} \left(\mathbf{I} - \frac{\mathbf{w}^t \mathbf{w}^{tT}}{\|\mathbf{w}^t\|^2} \right) \frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{w}}}$$

- Thus, $\mathbf{w} \perp \Delta \mathbf{w}$
- Let $c = \|\Delta \mathbf{w}\| / \|\mathbf{w}\|$, notice: $\|\mathbf{w}^{t+1}\| = \sqrt{(1 + c^2)} \|\mathbf{w}^t\|$
- Large $c \implies$ large $\|\mathbf{w}^{t+1}\| \implies$ small gradient $\Delta \mathbf{w}$
- Small $c \implies \|\mathbf{w}^{t+1}\| \approx \|\mathbf{w}^t\| \implies$ gradient scale $\Delta \mathbf{w}$ unchanged
- Learning rate decays every iteration

MILA

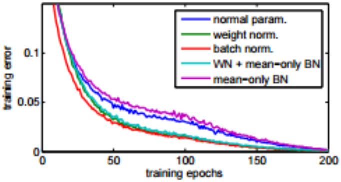


Figure 1: Training error for CIFAR-10 using different network parameterizations. For *weight normalization*, *batch normalization*, and *mean-only batch normalization* we show results using Adam with a learning rate of 0.003. For the normal parameterization we instead use 0.0003 which works best in this case. For the last 100 epochs the learning rate is linearly decayed to zero.

Model	Test Error
Maxout [6]	11.68%
Network in Network [17]	10.41%
Deeply Supervised [16]	9.6%
ConvPool-CNN-C [26]	9.31%
ALL-CNN-C [26]	9.08%
our CNN, mean-only B.N.	8.52%
our CNN, weight norm.	8.46%
our CNN, normal param.	8.43%
our CNN, batch norm.	8.05%
ours, W.N. + mean-only B.N.	**7.31%**

Figure 2: Classification results on CIFAR-10 without data augmentation.

Figures taken from Salimans, Tim, and Diederik P. Kingma. "Weight normalization: A simple reparameterization to accelerate training of deep neural networks." Advances in Neural Information Processing Systems. 2016.

Table of Contents

1. Goal
2. Key Insights
3. Batch Normalization
4. Weight Normalization
5. Scaled Weight Normalization
6. Layer Normalization
7. Analysis of Batch Normalized Networks
8. Conclusion

Key Observation:

- Vanilla weight-norm has gradient exploding/vanishing problem at initialization

Key idea:

- Appropriate scaling of pre-activation in weight-norm
- Benefits at initialization:
  - Preserves input norm throughout hidden layers
  - Prevents gradient explosion/vanishing problem
  - Applicable to deep ReLU networks

Scaled Weight Normalization (Arpit and Bengio, 2019)

- Consider any hidden unit's pre-activation a_i
- Then scaled weight normalized pre-activation is given by:

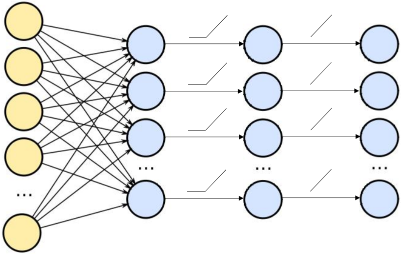
$$\text{IWeighNorm}(a_i) = \sqrt{\frac{2\text{fan-in}}{\text{fan-out}}} \cdot \frac{\gamma_i(\mathbf{W}_i^T \mathbf{x})}{\|\mathbf{W}_i\|_2} + \beta_i$$

$$\text{Initialize: } \gamma_i = 1 \quad \beta_i = 0$$

- Equivalently initialize $\gamma = \sqrt{\frac{2\text{fan-in}}{\text{fan-out}}}$

- A traditional vs. (scaled) Weight Normalized ReLU layer:

Traditional: $\mathbf{x}$ $\quad a_i = \mathbf{W}_i^T\mathbf{x} + \beta_i$ $\quad \mathbf{h} = \text{ReLU}(\mathbf{a})$ $\quad \mathbf{o} = \mathbf{h}$

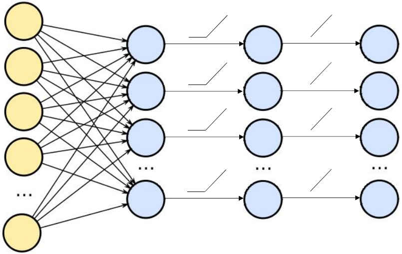


SWeightNorm: $\mathbf{x}$ $\quad \hat{a}_i = \sqrt{\frac{2\text{fan-in}}{\text{fan-out}}} \cdot \frac{\gamma_i(\mathbf{W}_i^T\mathbf{x})}{\|\mathbf{W}_i\|_2} + \beta_i$ $\quad \mathbf{h} = \text{ReLU}(\hat{\mathbf{a}})$ $\quad \mathbf{o} = \mathbf{h}$

Scaled Weight Normalization (Arpit and Bengio, 2019)

- A traditional vs. (scaled) Weight Normalized ReLU layer:

Traditional: $\mathbf{x}$ $\quad a_i = \mathbf{W}_i^T \mathbf{x} + \beta_i$ $\quad \mathbf{h} = \mathrm{ReLU}(\mathbf{a})$ $\quad \mathbf{o} = \mathbf{h}$



SWeightNorm: $\mathbf{x}$ $\quad \hat{a}_i = \sqrt{\frac{2\text{fan-in}}{\text{fan-out}}} \cdot \frac{\gamma_i (\mathbf{W}_i^T \mathbf{x})}{\|\mathbf{W}_i\|_2} + \beta_i$ $\quad \mathbf{h} = \mathrm{ReLU}(\hat{\mathbf{a}})$ $\quad \mathbf{o} = \mathbf{h}$

- γ_i and β_i are learnable scale and bias parameters
- train and test normalizations identical
- Applicable with batch-size 1

MILA

Theory:

- Consider any hidden layer:
  $\mathbf{v} = ReLU\left(\sqrt{\frac{\text{2fan-in}}{\text{fan-out}}} \cdot \hat{\mathbf{R}}\mathbf{u}\right)$
  We show: $\mathbb{E}[\|\mathbf{v}\|^2] = \|\mathbf{u}\|^2$

Theory:

- Consider any hidden layer:
  $$\mathbf{v} = ReLU\left(\sqrt{\frac{\text{2fan-in}}{\text{fan-out}}} \cdot \hat{\mathbf{R}}\mathbf{u}\right)$$
  We show: $\mathbb{E}[\|\mathbf{v}\|^2] = \|\mathbf{u}\|^2$
- For a sufficiently wide deep network $f_\theta(.)$:
  $\|f_\theta(\mathbf{x})\| \approx \|\mathbf{x}\|$

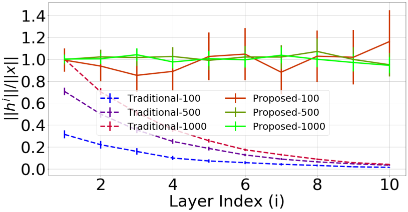
Theory:

- Consider any hidden layer:
  $\mathbf{v} = ReLU\left(\sqrt{\frac{2\text{fan-in}}{\text{fan-out}}} \cdot \hat{\mathbf{R}}\mathbf{u}\right)$
  We show: $\mathbb{E}[\|\mathbf{v}\|^2] = \|\mathbf{u}\|^2$
- For a sufficiently wide deep network $f_\theta(.)$:
  $\|f_\theta(\mathbf{x})\| \approx \|\mathbf{x}\|$
- For backward pass:

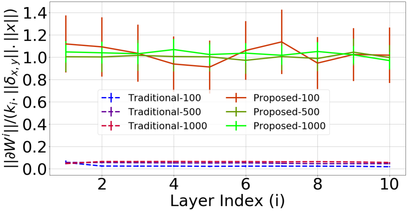
$$\left\|\frac{\partial \ell(f_\theta(\mathbf{x}), \mathbf{y})}{\partial \mathbf{W}^l}\right\|_F \approx \sqrt{\frac{2\text{fan-in}}{\text{fan-out} \cdot \|\mathbf{W}_i^l\|^2}} \cdot \|\delta(\mathbf{x}, \mathbf{y})\|_2 \cdot \|\mathbf{x}\|_2$$

$$\left(\delta(\mathbf{x}, \mathbf{y}) := \frac{\partial \ell(f_\theta(\mathbf{x}), \mathbf{y})}{\partial \mathbf{a}_L}\right)$$

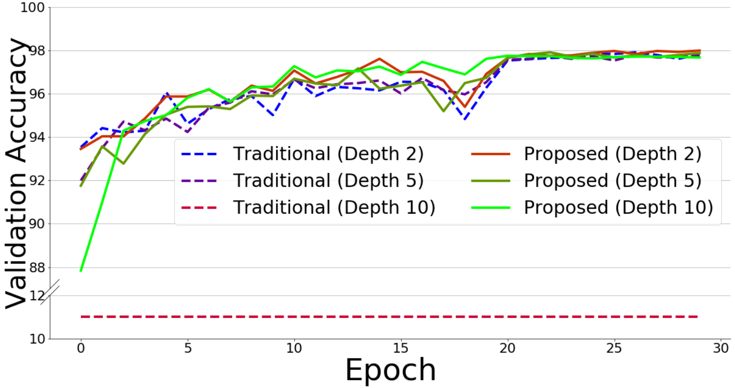
Scaled Weight Normalization (Arpit and Bengio, 2019)



Forward Pass



Backward Pass

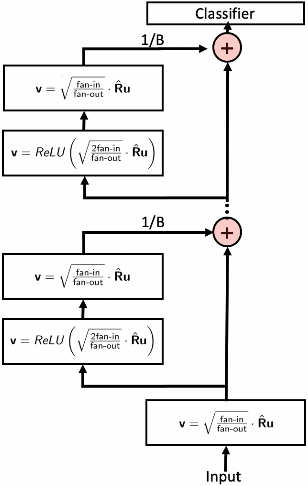


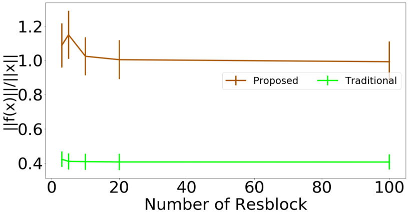
Training on MNIST

Theory for Residual Networks with B resblocks:

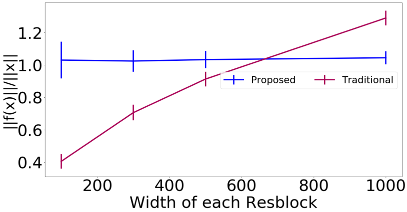
- Design ReLU layers as:
  $$\mathbf{v} = ReLU\left(\sqrt{\frac{\text{2fan-in}}{\text{fan-out}}} \cdot \hat{\mathbf{R}}\mathbf{u}\right)$$
- Design linear layers as:
  $$\mathbf{v} = \sqrt{\frac{\text{fan-in}}{\text{fan-out}}} \cdot \hat{\mathbf{R}}\mathbf{u}$$
- Divide output of each resblock by B
- For sufficiently large B and width:

$$\frac{\|\mathbf{x}\|^2}{e^2} \leq \|f_\theta(\mathbf{x})\|^2 \leq e^2 \cdot \|\mathbf{x}\|^2$$





Vary the number of residual blocks (width = 100)



Vary width of each residual block (number of resblocks = 10)

Table of Contents

1. Goal
2. Key Insights
3. Batch Normalization
4. Weight Normalization
5. Scaled Weight Normalization
6. Layer Normalization
7. Analysis of Batch Normalized Networks
8. Conclusion

MILA

Layer Normalization (Ba et al, 2016)

Key idea:

- Compute statistics for each sample separately

Layer Normalization (Ba et al, 2016)

Key idea:

- Compute statistics for each sample separately
- Trick: Compute mean and standard deviation across units instead of mini-batch

Key idea:

- Compute statistics for each sample separately
- Trick: Compute mean and standard deviation across units instead of mini-batch
- Aimed towards application to recurrent neural nets:
  - RNNs have variable number of temporal layers
  - There can be more number of layers at test time (BN is not applicable)

- Consider a temporal hidden layer's pre-activation $\mathbf{a}^t = \mathbf{W}_{hh}\mathbf{h}^{t-1} + \mathbf{W}_{xh}\mathbf{x}^t$
- Then layer normalized pre-activation $\mathbf{a}^t$ is given by:

$$\mathrm{LN}(\mathbf{a}^t) = \frac{\gamma}{\sigma^t} \odot (\mathbf{a}_t - \mu^t) + \beta$$

$$\mu^t = \frac{1}{H}\sum_{i=1}^{H} a_i^t \quad \sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^t - \mu^t)^2}$$

$$\mathrm{LN}(\mathbf{a}^t) = \frac{\gamma}{\sigma^t} \odot (\mathbf{a}_t - \mu^t) + \beta$$

$$\mu^t = \frac{1}{H}\sum_{i=1}^{H} a_i^t \quad \sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^t - \mu^t)^2}$$

Effects of layer normalization:

- Invariance to weight scaling and translation
- Invariance to data rescaling and translation
- Norm of weight controls effective learning rate

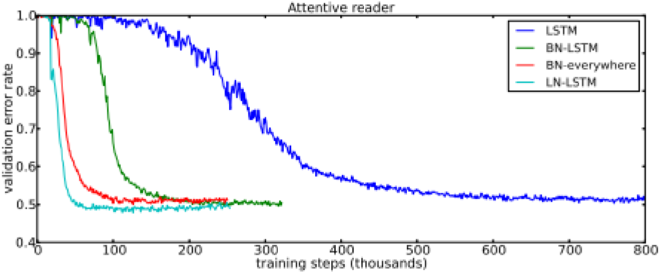


Figure 2: Validation curves for the attentive reader model. BN results are taken from [Cooijmans et al., 2016].

Figures taken from Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer normalization." arXiv preprint arXiv:1607.06450 (2016).

Other popular Normalization Techniques:

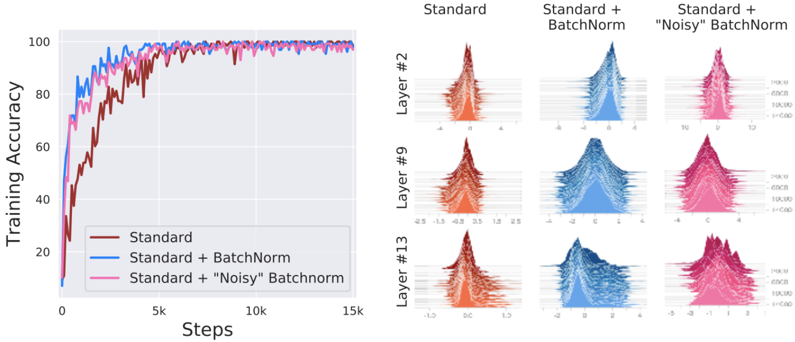
- Normalization Propagation (Arpit et al 2016)
- Instance Normalization (Ulyanov et al 2016)
- Group Normalization (Wu et al 2018)

Table of Contents

1. Goal
2. Key Insights
3. Batch Normalization
4. Weight Normalization
5. Scaled Weight Normalization
6. Layer Normalization
7. Analysis of Batch Normalized Networks
8. Conclusion

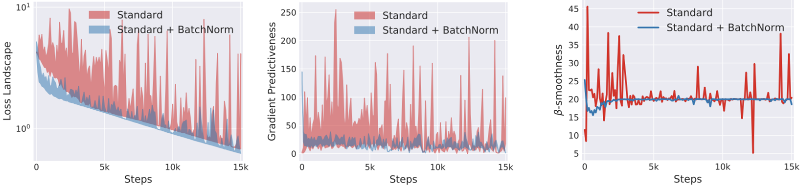
MILA

Internal Covariate Shift may not have a big impact



Figures taken from Santurkar et al, 2018. "How Does Batch Normalization Help Optimization?." NeuRIPS (2018).

BatchNorm conditions the loss landscape



(a) loss landscape (b) gradient predictiveness (c) "effective" β-smoothness

Figures taken from Santurkar et al, 2018. "How Does Batch Normalization Help Optimization?." NeuRIPS (2018).

Role of Weight Decay in BatchNorm Networks

- Traditionally– weight decay (WD) directly controls capacity:

$$\min_{\mathbf{w}} \ell(\mathcal{D}; \mathbf{w}) + \lambda \|\mathbf{w}\|^2$$

- BatchNorm network outputs are weight scale invariant:

(Also true for some other normalization method)

$$\mathrm{BN}(c\mathbf{Wx}) = \mathrm{BN}(\mathbf{Wx})$$

- Weight scale no longer capacity control
- What does WD do?

MILA

Role of Weight Decay in BatchNorm Networks

- WD has a different regularization effect
- SGD update rule in BatchNorm networks (Hoffer et al 2018):

$$\hat{\mathbf{w}}_{t+1} \approx \hat{\mathbf{w}}_t - \frac{\eta}{\|\mathbf{w}\|_2} \cdot (\mathbf{I} - \hat{\mathbf{w}}_t \hat{\mathbf{w}}_t^T) \nabla \ell(\hat{\mathbf{w}}_t)$$

- WD scales effective learning rate (LR)
- Large WD $\equiv$ Large LR
- Large LR has a regularization effect

  (Smith and Le 2017, Jastrzebski et al 2017 etc)

Table of Contents

1. Goal
2. Key Insights
3. Batch Normalization
4. Weight Normalization
5. Scaled Weight Normalization
6. Layer Normalization
7. Analysis of Batch Normalized Networks
8. Conclusion

Summary

- Convergence speed depends on:
  - Loss surface geometry
  - Internal covariate shift
- Normalization techniques are an integral part of modern deep learning
- Why BatchNorm improves convergence?
  - Smoothing loss landscape
  - Alleviating internal covariate shift?
- Regularization effect of BatchNorm
- Regularization effect of weight decay
- Advantage of other normalization techniques