Normalization Techniques in Training of Deep Neural Networks

Lei Huang (黄雷)
State Key Laboratory of Software Development Environment,
Beihang University

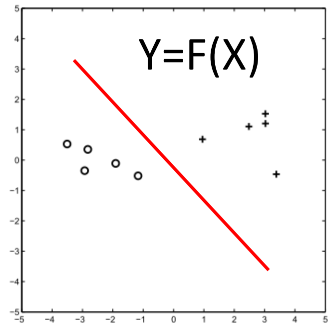
Mail:huanglei@nlsde.buaa.edu.cn

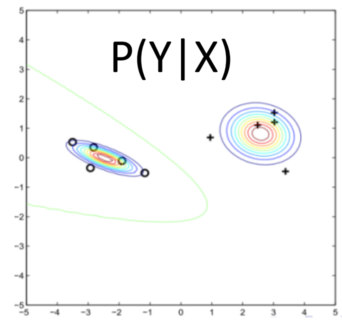
August 17th, 2017

Outline

- Introduction to Deep Neural Networks (DNNs)
- Training DNNs: Optimization
- Batch Normalization
- Other Normalization Techniques
- Centered Weight Normalization

Machine learning

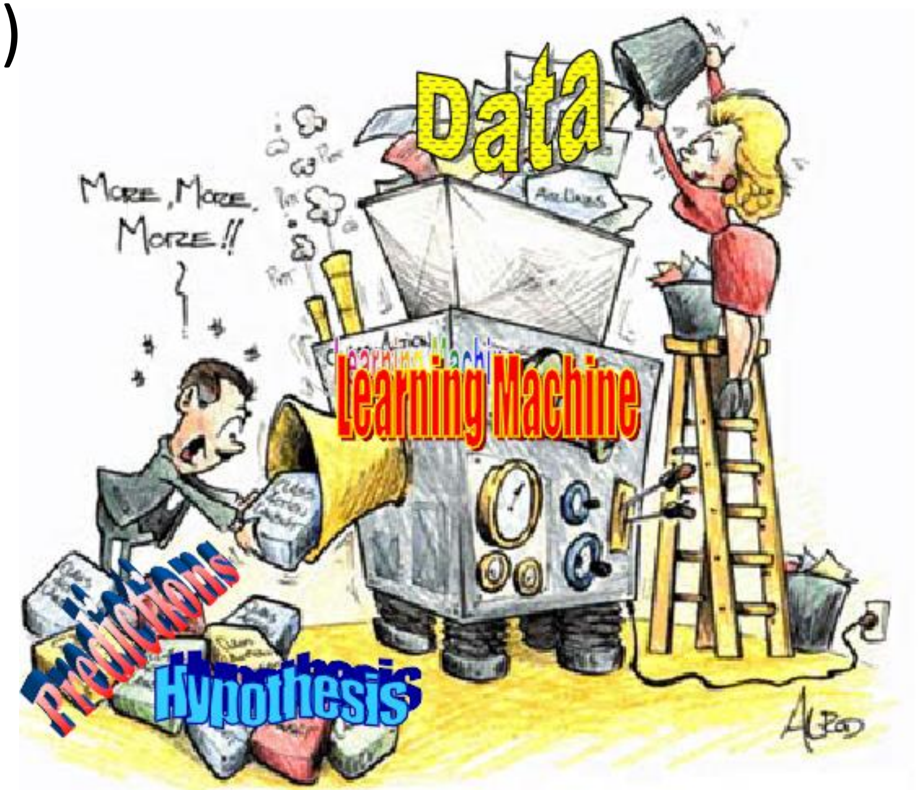




- dataset D={X, Y}
  - Input: X
  - Output: Y
  - Learning: Y=F(X) or P(Y|X)

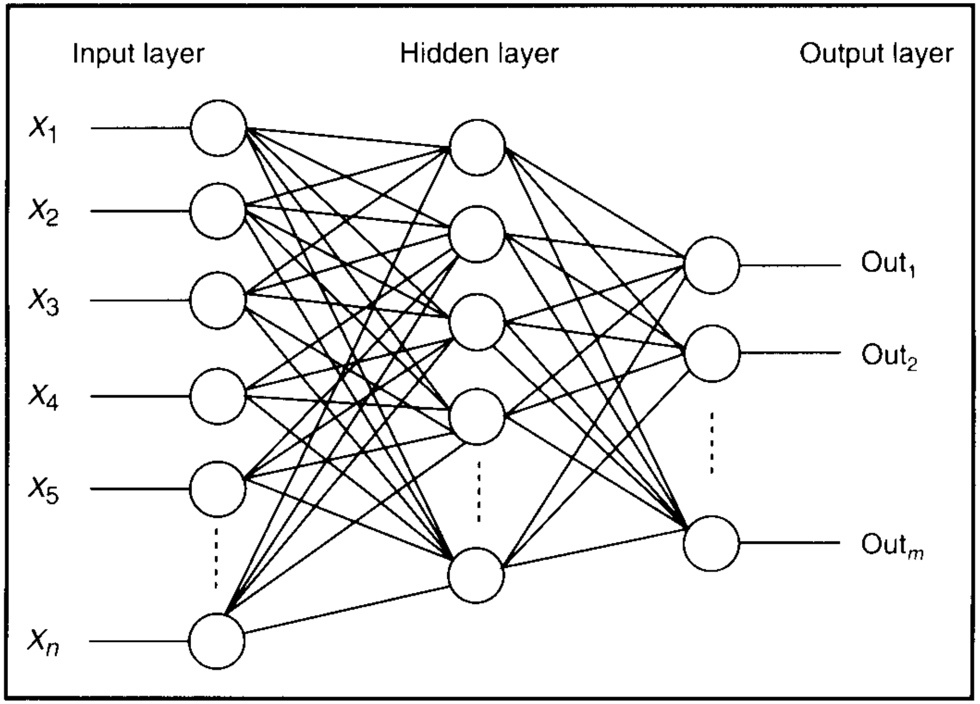
- Fitting and Generalization

- Types: view of models
  - Non-parametric model
    - Y=F(X; $x_1, x_2 \ldots x_n$)
  - Parametric model
    - Y=F(X; θ)

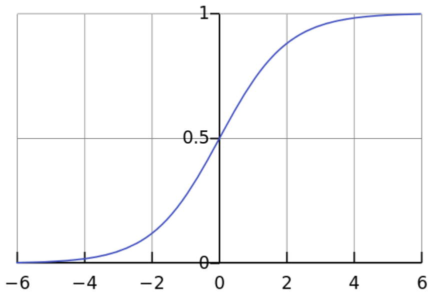


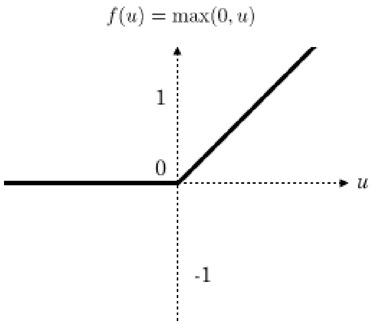
Neural network

- Neural network
  - Y=F(X)=$f_T(f_{T-1}(\ldots f_1(X)))$
  - $f_i(x) = g(Wx + b)$



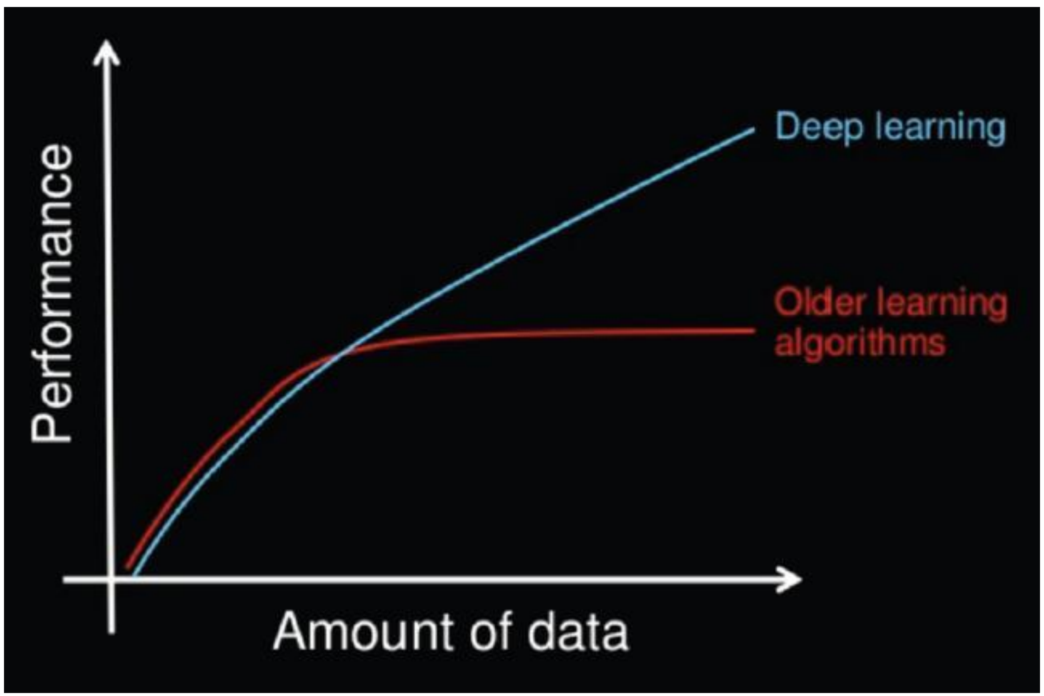
- Nonlinear activation
  - sigmod
  - Relu





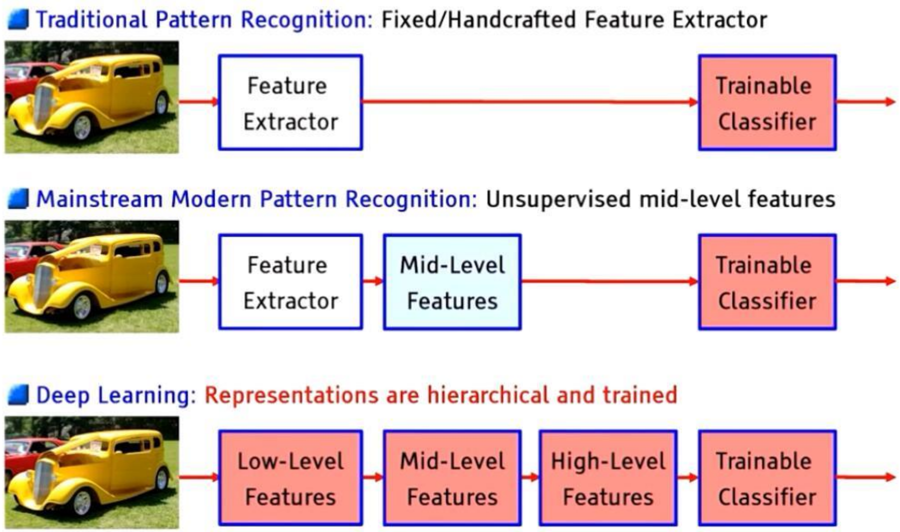
Deep neural network

- Why deep?
  - Powerful representation capacity



Key properties of Deep learning

- End to End learning
  - no distinction between feature extractor and classifier



- “Deep” architectures:
  - Hierarchy of simpler non-linear modules

Applications and techniques of DNNs

- Successful applications in a range of domains
  - Speech
  - Computer Vision
  - Natural Language processing

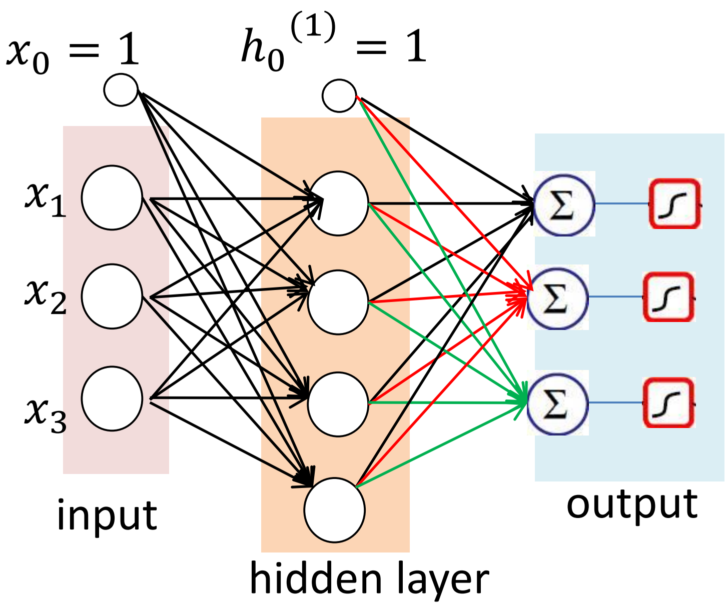
- Main techniques in using deep neural networks networks
  - Design the architecture
    - Module selection and Module connection
    - Loss function
  - Train the model based on optimization
    - Initialize the parameters
    - Search direction in parameters space
    - Learning rate schedule
    - Regularization techniques
    - …

Outline

- Introduction to Deep Neural Networks (DNNs)
- Training DNNs: Optimization
- Batch Normalization
- Other Normalization Techniques
- Centered Weight Normalization

Training of Neural Networks

- Multi-layer perceptron (example)



$(1, 0, 0)^T$

➢ 2 Backward，Calculate $\frac{dL}{d\boldsymbol{x}}$：

$$\frac{\mathrm{d}\,\mathrm{L}}{\mathbf{y}} = 2(\mathbf{y} - \hat{\mathbf{y}})$$

$$\frac{\mathrm{d}\,\mathrm{L}}{d\boldsymbol{a}^{(2)}} = \frac{\mathrm{d}\,\mathrm{L}}{\mathbf{dy}} \cdot \sigma\left(\boldsymbol{a}^{(2)}\right) \cdot \left(1 - \sigma\left(\boldsymbol{a}^{(2)}\right)\right)$$

$$\frac{dL}{d\,\boldsymbol{h}^{(1)}} = \frac{\mathrm{d}\,\mathrm{L}}{d\boldsymbol{a}^{(2)}} \boldsymbol{W}^{(2)}$$

$$\frac{\mathrm{d}\,\mathrm{L}}{\boldsymbol{a}^{(1)}} = \frac{\mathrm{d}\,\mathrm{L}}{d\,\boldsymbol{h}^{(1)}} \cdot \sigma\left(\boldsymbol{a}^{(1)}\right) \cdot \left(1 - \sigma\left(\boldsymbol{a}^{(1)}\right)\right)$$

$$\frac{dL}{d\boldsymbol{x}} = \frac{\mathrm{d}\,\mathrm{L}}{d\boldsymbol{a}^{(1)}} \boldsymbol{W}^{(1)}$$

➢ 1 Forward calculate y：

$$\boldsymbol{a}^{(1)} = \boldsymbol{W}^{(1)} \cdot \boldsymbol{x}$$
$$\boldsymbol{h}^{(1)} = \sigma\left(\boldsymbol{a}^{(1)}\right)$$
$$\boldsymbol{a}^{(2)} = \boldsymbol{W}^{(2)} \cdot \boldsymbol{h}^{(1)}$$
$$\boldsymbol{y} = \sigma\left(\boldsymbol{a}^{(2)}\right)$$

➢ 3 calculate gradient $\frac{dL}{d\boldsymbol{W}}$：

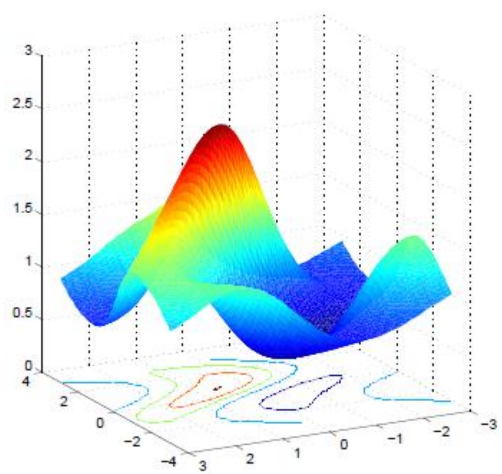
$$\frac{dL}{d\,\boldsymbol{W}^{(2)}} = \frac{\mathrm{d}\,\mathrm{L}}{d\boldsymbol{a}^{(2)}} \boldsymbol{h}^{(1)}$$

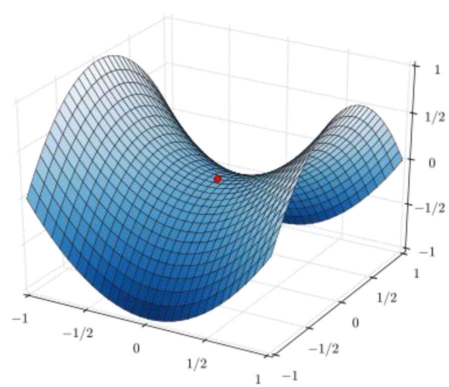
$$\frac{dL}{d\,\boldsymbol{W}^{(1)}} = \frac{\mathrm{d}\,\mathrm{L}}{d\boldsymbol{a}^{(1)}} \boldsymbol{x}$$

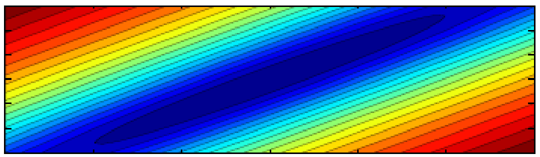
➢ MSE Loss: L=$(\mathbf{y} - \hat{\mathbf{y}})^2$

Optimization in Deep Model

- Goal: $\theta^* \in \operatorname{argmin}_\theta \mathbb{E}_{(x,y)\sim\pi}[-\log p(y \mid x,\theta)]$
- Update Iteratively: $\theta^{(t+1)} \leftarrow \theta^{(t)} - \alpha^{(t)}\nabla^{(t)}$
- Challenge:
  - Non-convex and local optimal points
  - Saddle point
  - Severe correlation between dimensions and highly non-isotropic parameter space (ill-shaped)







First order optimization

- First order stochastic gradient descent (SGD):
  - The direction of the gradient

$$\nabla = \mathbb{E}_\pi \left[d\ell / d\theta \right]$$

  - Gradient is averaged by the sampled examples
  - Disadvantage
    - Over-aggressive steps on ridges
    - Too small steps on plateaus
    - Slow convergence
    - non-robust performance.

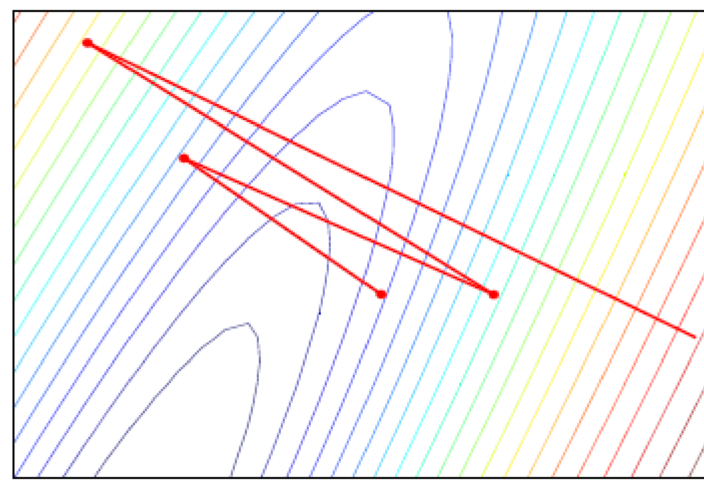
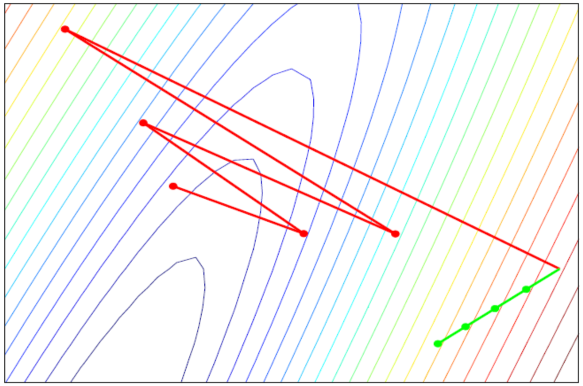


Figure 2: zig-zag iteration path for SGD

Advanced Optimization

- Estimate curvature or scale
  - Quadratic optimization
    - Newton or quasi-Newton
      - Inverse of Hessian
    - Natural Gradient
      - Inverse of FIM
  - Estimate the scale
    - AdaGrad
    - Rmsprop
    - Adam

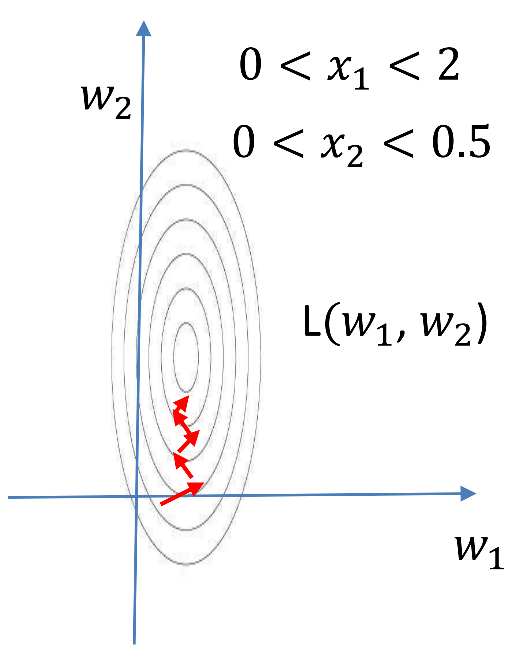


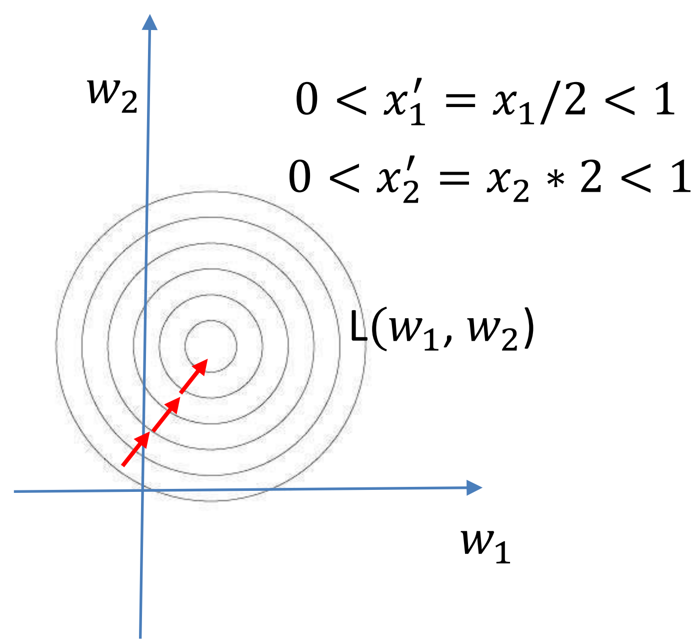
Iteration path of SGD (red) and NGD (green)

- Normalize input/activation
  - Intuition： the landscape of cost w.r.t parameters is controlled by Input/activation L=(f(x,θ),y)
  - Method: Stabilize the distribution of input/activation
    - Normalize the input explicitly
    - Normalize the input implicitly (constrain weights)

Some intuitions of normalization for optimization

- How Normalizing activation affect the optimization?
  - $y = w_1 x_1 + w_2 x_2$+b
  - L=$(y - \hat{y})$^2





Outline

- Introduction to Deep Neural Networks (DNNs)
- Training DNNs: Optimization
- Batch Normalization
- Other Normalization Techniques
- Centered Weight Normalization

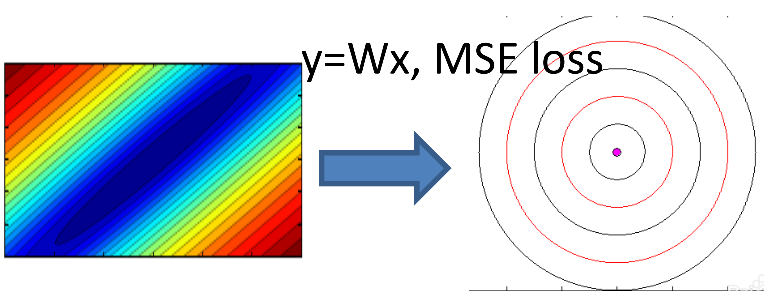
Batch Normalization--motivation

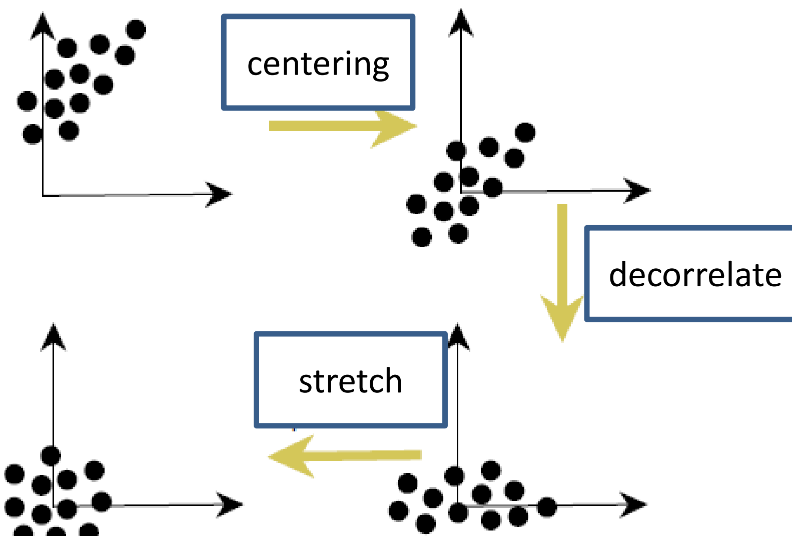
- Solving Internal Covariate Shift

$$h_2 = wh_1$$

$$h_1 = wx$$

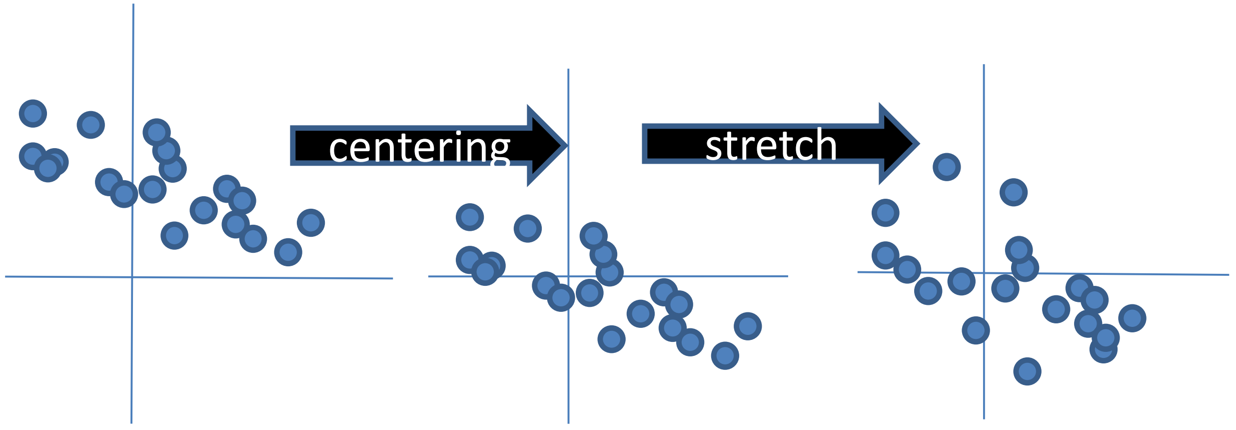
- Whitening Input benefits optimization (1998,Lecun, Efficient back-propagation)
  - Centering
  - Decorrelate
  - stretch





Batch Normalization--method

- Only standardize input: decorrelating is expensive
  - Centering
  - Stretch



- How to do it?
  - $\hat{x} = \frac{x - E(x)}{std(x)}$

Batch Normalization--training

- Forward

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--training

- Backward

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{\sum_{i=1}^{m} -2(x_i - \mu_{\mathcal{B}})}{m}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

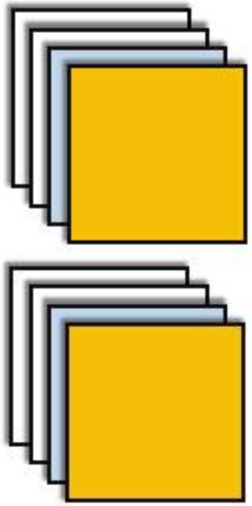
Batch Normalization--Inference

- Inference (in paper)

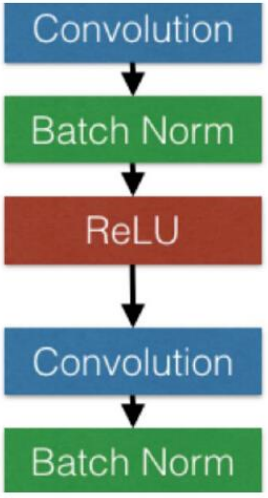
$$\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\mathrm{Var}[x] \leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

- Inference (in practice)
  - Running average
    - $E(x) = \alpha(\mu_B) + (1-\alpha)E(x)$
    - $Var(x) = \alpha(\sigma_B^2) + (1-\alpha)var(x)$

Batch Normalization—how to use

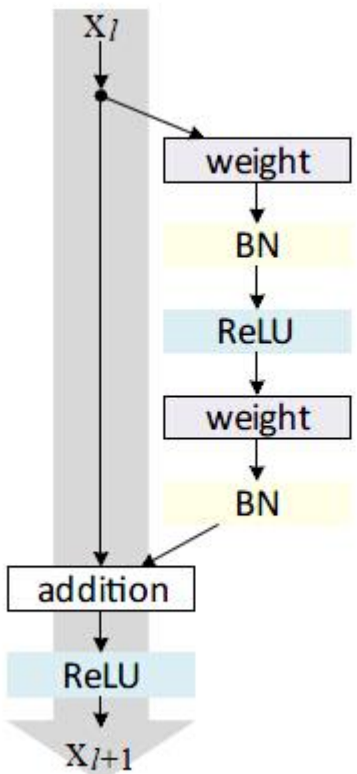


- Convolution layer
- Wrapped as a module
  - Before or after nonlinear?
    - For shallow module, after nonlinear (Layer <11)
    - For deep model, before nonlinear
  - Advantage of before nonlinear
    - For Relu, half activated
    - For sigmod, avoiding saturated region.
  - Advantage of after nonlinear
    - The intuition of whitening

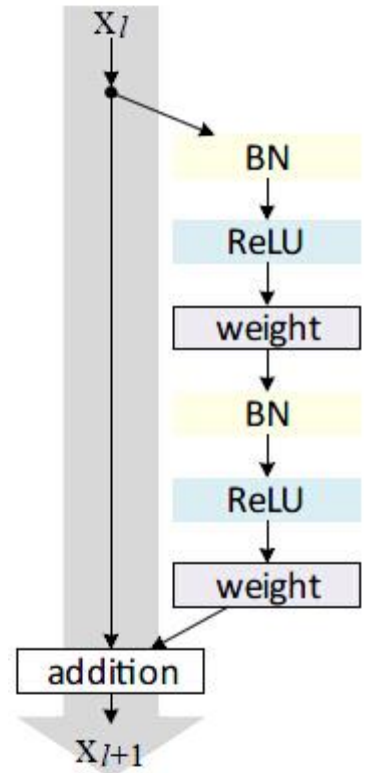


Batch Normalization—how to use

- Example:



Residual block (CVPR 2015)



Pre-activation Residual block (ECCV 2016)

Batch Normalization—characteristics

- For accelerating training：
  - Weight scale invariant: Not sensitive for weight initialization

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

  - Adjustable learning rate

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

  - Large learning rate
    - Better conditioning, (1998 Lecun)

- For generalization
  - Stochastic, works like Dropout

Batch Normalization

- Routine in deep feed forward neural networks, especially for CNNs.

- Weakness
  - Can not be used for online learning
  - Unstable for small mini batch size
  - Used in RNN with caution

Batch Normalization– for RNN

- The extra problems need be considered:
  - Where BN should put?
  - Sequence data

$$\mathbf{h}_t = \phi(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t)$$

- 2016,ICASSP, Batch Normalized Recurrent Neural Networks
  - How to put BN module
  - Sequence data problem
    - Frame-wise normalization
    - Sequence-wise normalization

$$\mathbf{h}_t = \phi(BN(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t)).$$

$$\mathbf{h}_t = \phi(\mathbf{W}_h \mathbf{h}_{t-1} + BN(\mathbf{W}_x \mathbf{x}_t)).$$

Batch Normalization for RNN

- 2017, ICLR, Recurrent Batch Normalization
  - How to put BN module

$$\begin{pmatrix} \tilde{\mathbf{f}}_t \\ \tilde{\mathbf{i}}_t \\ \tilde{\mathbf{o}}_t \\ \tilde{\mathbf{g}}_t \end{pmatrix} = \mathrm{BN}(\mathbf{W}_h \mathbf{h}_{t-1}; \gamma_h, \beta_h) + \mathrm{BN}(\mathbf{W}_x \mathbf{x}_t; \gamma_x, \beta_x) + \mathbf{b}$$

$$\mathbf{c}_t = \sigma(\tilde{\mathbf{f}}_t) \odot \mathbf{c}_{t-1} + \sigma(\tilde{\mathbf{i}}_t) \odot \tanh(\tilde{\mathbf{g}}_t)$$

$$\mathbf{h}_t = \sigma(\tilde{\mathbf{o}}_t) \odot \tanh(\mathrm{BN}(\mathbf{c}_t; \gamma_c, \beta_c))$$

  - Sequence data problem
    - T_max Frame-wise normalization
    - It depends…..

Outline

- Introduction to Deep Neural Networks (DNNs)
- Training DNNs: Optimization
- Batch Normalization
- Other Normalization Techniques
- Centered Weight Normalization

Norm-propagation (2016, ICML)

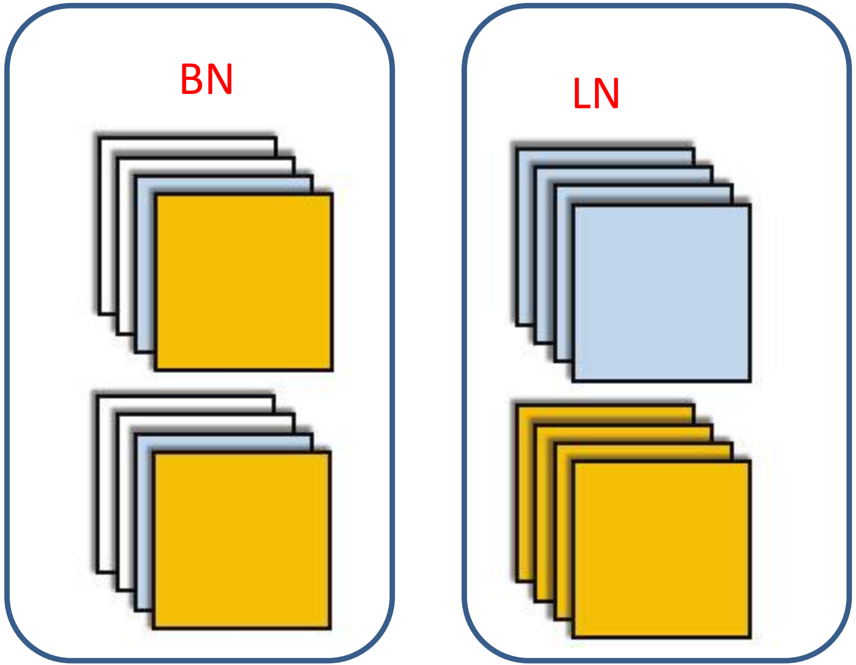
- Target BN's drawback:
  - Can not be used for online learning
  - Unstable for small mini batch size.
- Data independent parametric estimate of mean and variance
  - Normalize input: 0-mean and unit variance
  - Assuming W is orthogonal
  - Derivate the nonlinear dynamic
    - Relu:

Remark 1. *(Post-ReLU distribution) Let* $X \sim \mathcal{N}(0,1)$ *and* $Y = \max(0, X)$. *Then* $\mathbb{E}[Y] = \frac{1}{\sqrt{2\pi}}$ *and* $\operatorname{var}(Y) = \frac{1}{2}\left(1 - \frac{1}{\pi}\right)$

Layer Normalization (2016, Arxiv)

- Target BN's drawback:
  - Can not be used for online learning
  - Unstable for small mini batch size
  - RNN
- Normalizing each example, over dimensions

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$$



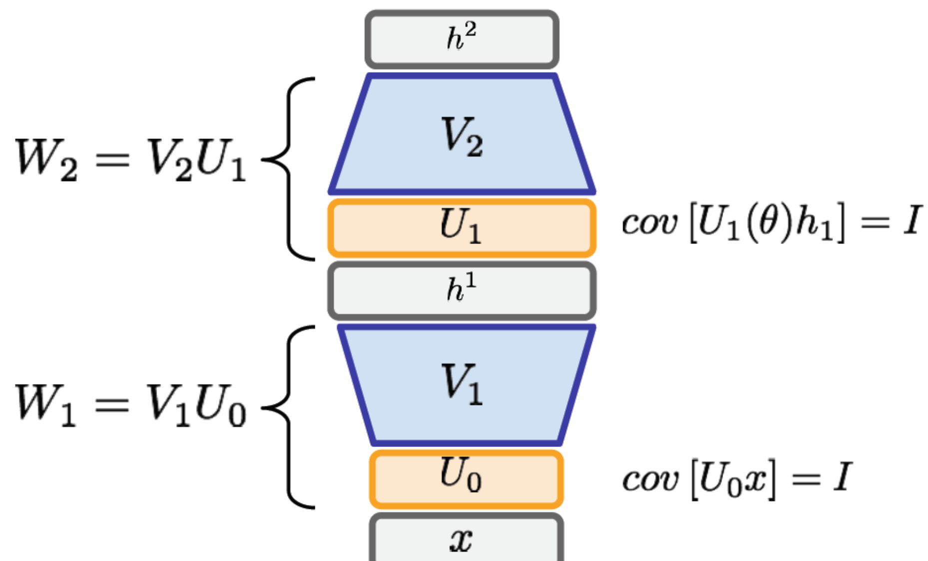
Natural Neural Network (2015, NIPS)

- How about decorrelate the activations?
- Canonical model(MLP): $h_i = f_i(W_i h_{i-1} + b_i)$
- Natural neural network

$$h_i \quad = \quad f_i\left(V_i U_{i-1}\left(h_{i-1} - c_i\right) + d_i\right)$$

- Model parameters:
  $\Omega = \{V_1, d_1 \ldots, V_L, d_L\}$

- Whitening coefficients :
  $\Phi = \{U_0, c_0 \ldots, U_{L-1}, c_{L-1}\}$



Weight Normalization (2016, NIPS)

- Target BN's drawback:
  - Can not be used for online learning
  - Unstable for small mini batch size
  - RNN

- Express weight as new parameters

$$\mathbf{w} = \frac{g}{||\mathbf{v}||}\mathbf{v} \qquad\qquad y = \phi(\mathbf{w} \cdot \mathbf{x} + b)$$

- Decouple direction and length of vectors

Reference

- batch normalization accelerating deep network training by reducing internal covariate shift, ICML 2015 (Batch Normalization)
- Normalization Propagation A Parametric Technique for Removing Internal Covariate Shift in Deep Networks, ICML, 2016
- Weight Normalization A Simple Reparameterization to Accelerate Training of Deep Neural Networks, NIPS, 2016
- Layer Normalization, Arxiv:1607.06450, 2016
- Recurrent Batch Normalization, ICLR,2017
- Batch Normalized Recurrent Neural Networks, ICASSP, 2016
- Natural Neural Networks, NIPS, 2015
- Normalizing the normaliziers-comparing and extending network normalization schemes, ICLR, 2017
- Batch Renormalization, Arxiv:1702.03275, 2017
- mean-normalized stochastic gradient for large-scale deep learning, ICASSP 2014
- deep learning made easier by linear transformations in perceptrons, AISTATS 2012

Outline

- Introduction to Deep Neural Networks (DNNs)
- Training DNNs: Optimization
- Batch Normalization
- Other Normalization Techniques
- Centered Weight Normalization

Centered Weight Normalization in Accelerating Training of Deep Neural Networks

Lei Huang, Xianglong Liu, Yang Liu, Bo Lang, Dacheng Tao

International Conference on Computer Vision
(ICCV) 2017

Motivation

- Stable distribution in hidden layer
- Initialization method
  - Random Init (1998, YanLecun)
    - Zero-mean, stable-var
  - Xavier Init (2010, Xavier)

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$$

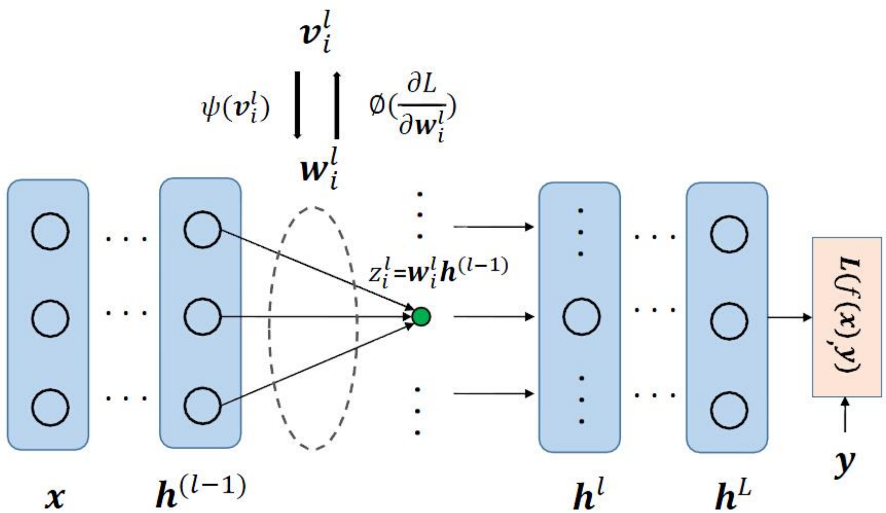
  - He Init (2015, He Kaiming)
    - $W \sim N\left(0, \sqrt{\frac{2}{n}}\right), n = out * H * W$
- Keep desired characters during training

Method

- Formulation: Constrained optimization problem:

$$\theta^* \quad = \arg\min_\theta \mathbb{E}_{(\mathbf{x},\mathbf{y})\in D}[\mathcal{L}(\mathbf{y}, f(\mathbf{x};\theta))]$$
$$s.t. \quad \mathbf{w}^T\mathbf{1} = 0 \; and \; \|\mathbf{w}\| = 1$$

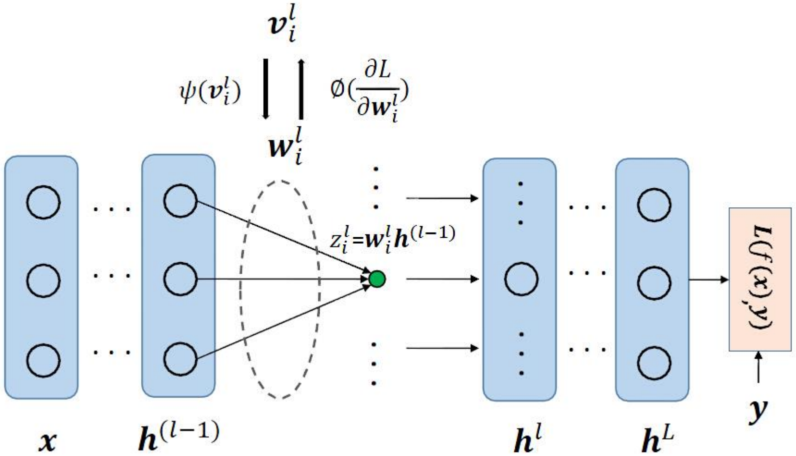
- Solution by re-parameterization



Method

- Using proxy parameter v:

$$\mathbf{w} = \frac{\mathbf{v} - \frac{1}{d}\mathbf{1}(\mathbf{1}^T\mathbf{v})}{\|\mathbf{v} - \frac{1}{d}\mathbf{1}(\mathbf{1}^T\mathbf{v})\|}$$



- Gradient Information:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{v}} = \frac{1}{\|\hat{\mathbf{v}}\|}\left[\frac{\partial \mathcal{L}}{\partial \mathbf{w}} - \left(\frac{\partial \mathcal{L}}{\partial \mathbf{w}}\mathbf{w}\right)\mathbf{w}^T - \frac{1}{d}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{w}}\mathbf{1}\right)\mathbf{1}^T\right]$$

- Adjustable scale:

$$z = g\left(\frac{\mathbf{v} - \frac{1}{d}\mathbf{1}(\mathbf{1}^T\mathbf{v})}{\|\mathbf{v} - \frac{1}{d}\mathbf{1}(\mathbf{1}^T\mathbf{v})\|}\right)^T \mathbf{h} + b.$$

Method

- Wrapped as module for practitioner:
  - Forward

Algorithm 1 Forward pass of linear mapping with centered weight normalization.

1: **Input**: the mini-batch input data $\mathbf{X} \in \mathbb{R}^{d \times m}$ and parameters to be learned: $\mathbf{g} \in \mathbb{R}^{n \times 1}$, $\mathbf{b} \in \mathbb{R}^{n \times 1}$, $\mathbf{V} \in \mathbb{R}^{d \times n}$.
2: **Output**: pre-activation $\mathbf{Z} \in \mathbb{R}^{n \times m}$.
3: compute centered weight: $\hat{\mathbf{V}} = \mathbf{V} - \frac{1}{d}\mathbf{1}_d(\mathbf{1}_d^T \mathbf{V})$.
4: **for** $i = 1$ to n **do**
5:     calculate normalized weight with respect to the i-th neuron: $\mathbf{w}_i = \frac{\hat{\mathbf{v}}_i}{\|\hat{\mathbf{v}}_i\|}$
6: **end for**
7: calculate: $\hat{\mathbf{Z}} = \mathbf{W}^T \mathbf{X}$.
8: calculate pre-activation: $\mathbf{Z} = (\mathbf{g}\mathbf{1}_m^T) \odot \hat{\mathbf{Z}} + \mathbf{b}\mathbf{1}_m^T$.

Method

- Wrapped as module for practitioner:
  - Backward

Algorithm 2 Back-propagation pass of linear mapping with centered weight normalization.

Input: pre-activation derivative $\{\frac{\partial \mathcal{L}}{\partial \mathbf{Z}} \in \mathbb{R}^{n \times m}\}$. Other auxiliary variables from respective forward pass: $\hat{\mathbf{V}}$, $\mathbf{W}$, $\hat{\mathbf{Z}}$, $\mathbf{X}$, $\mathbf{g}$.

Output: the gradients with respect to the inputs $\{\frac{\partial \mathcal{L}}{\partial \mathbf{X}} \in \mathbb{R}^{d \times m}\}$ and learnable parameters: $\frac{\partial \mathcal{L}}{\partial \mathbf{g}} \in \mathbb{R}^{1 \times n}$, $\frac{\partial \mathcal{L}}{\partial \mathbf{b}} \in \mathbb{R}^{1 \times n}$, $\frac{\partial \mathcal{L}}{\partial \mathbf{V}} \in \mathbb{R}^{d \times n}$.

$\frac{\partial \mathcal{L}}{\partial \mathbf{g}} = \mathbf{1}_m^T (\frac{\partial \mathcal{L}}{\partial \mathbf{Z}} \odot \hat{\mathbf{Z}})^T$

$\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \mathbf{1}_m^T \frac{\partial \mathcal{L}}{\partial \mathbf{Z}}^T$

$\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{Z}}} = \frac{\partial \mathcal{L}}{\partial \mathbf{Z}} \odot (\mathbf{g} \mathbf{1}_m^T)$

$\frac{\partial \mathcal{L}}{\partial \mathbf{X}} = \mathbf{W} \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{Z}}}$

$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \mathbf{X} \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{Z}}}^T$

for $i = 1$ to n **do**

$\quad \frac{\partial \mathcal{L}}{\partial \mathbf{v}_i} = \frac{1}{\|\hat{\mathbf{v}}_i\|} (\frac{\partial \mathcal{L}}{\partial \mathbf{w}_i} - (\frac{\partial \mathcal{L}}{\partial \mathbf{w}_i} \mathbf{w}_i) \mathbf{w}_i^T - \frac{1}{d} (\frac{\partial \mathcal{L}}{\partial \mathbf{w}_i} \mathbf{1}_d) \mathbf{1}_d^T)$

end for

Discussion

- Beneficial Characters for training
  - Stabilize the distributions

Proposition 1. *Let* $z = \mathbf{w}^T\mathbf{h}$, *where* $\mathbf{w}^T\mathbf{1} = 0$ *and* $\|\mathbf{w}\| = 1$. *Assume* $\mathbf{h}$ *has Gaussian distribution with the mean:* $\mathbb{E}_{\mathbf{h}}[\mathbf{h}] = \mu\mathbf{1}$, *and covariance matrix:* $cov(\mathbf{h}) = \sigma^2\mathbf{I}$, *where* $\mu \in \mathbb{R}$ *and* $\sigma^2 \in \mathbb{R}$. *We have* $\mathbb{E}_z[z] = 0$, $var(z) = \sigma^2$.

  - Better Conditioning of Hessian

Proposition 2. *Regarding to the proxy parameter* $\mathbf{v}$, *centered weight normalization makes that the gradient* $\frac{\partial \mathcal{L}}{\partial \mathbf{v}}$ *has following properties: (1)* zero-mean, i.e. $\frac{\partial \mathcal{L}}{\partial \mathbf{v}} \cdot \mathbf{1} = 0$; *(2) orthogonal to the parameters* $\mathbf{w}$, i.e. $\frac{\partial \mathcal{L}}{\partial \mathbf{v}} \cdot \mathbf{w} = 0$.

- Regularization in improving performance

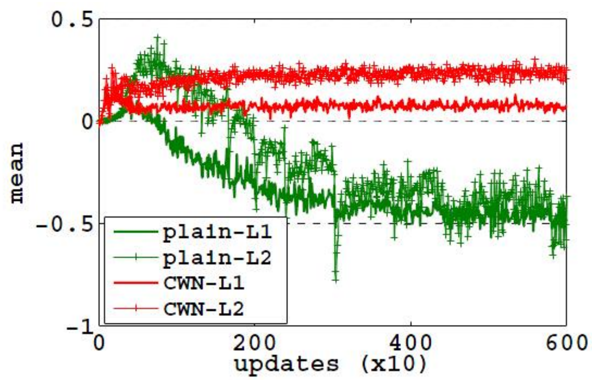
Experiments

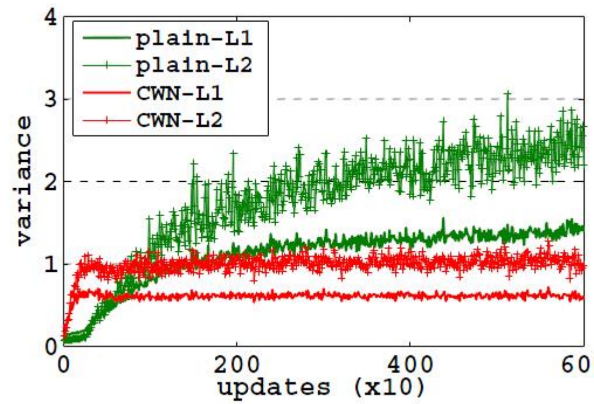
- Data set
  - Yale-B
  - SVHN
  - Cifar10, Cifar100
  - ImageNet

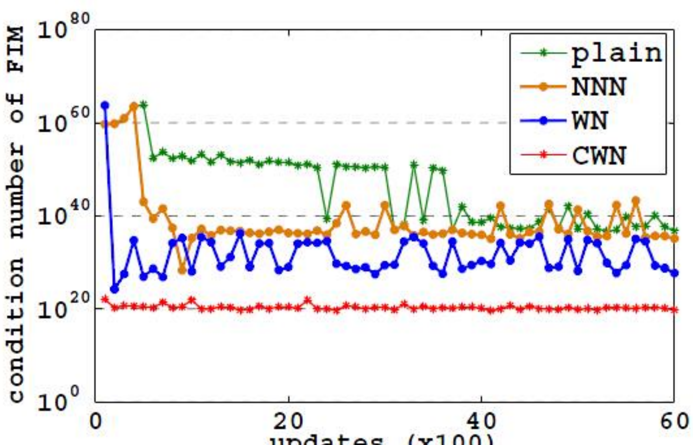
- Reproducible experiments and Code: https://github.com/huangleiBuaa/CenteredWN

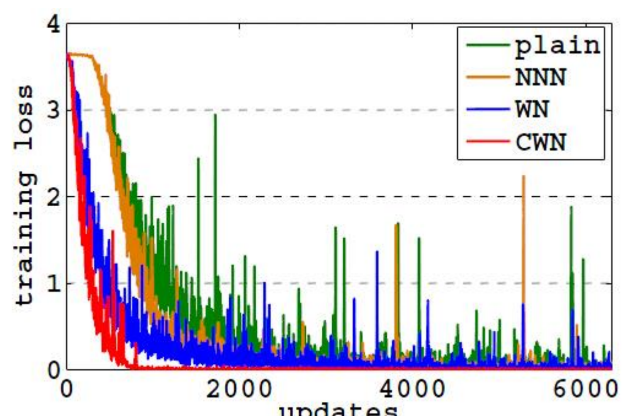
Experiments

- Ablation study
  - YaleB, MLP{128,64,48,48}





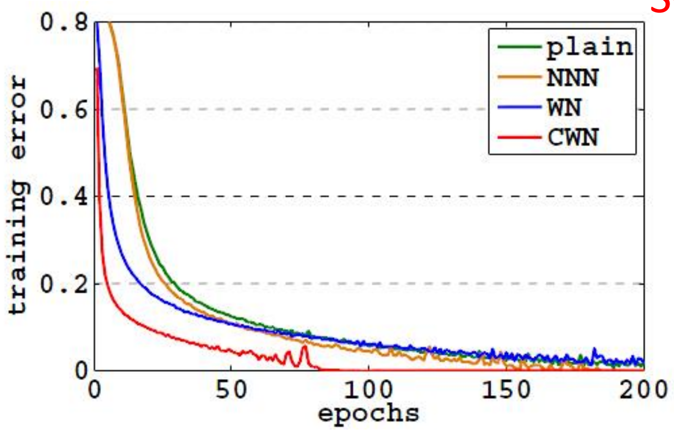


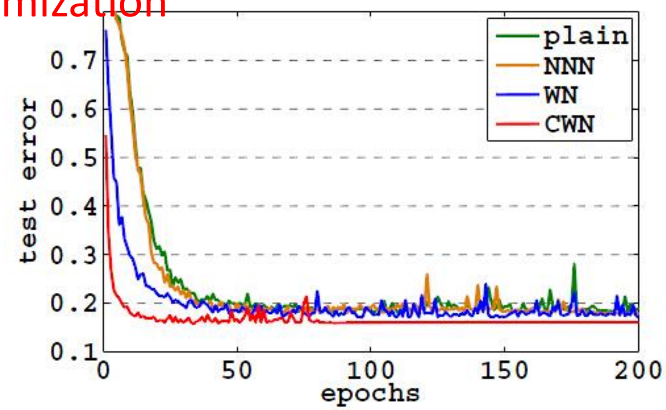


Experiments

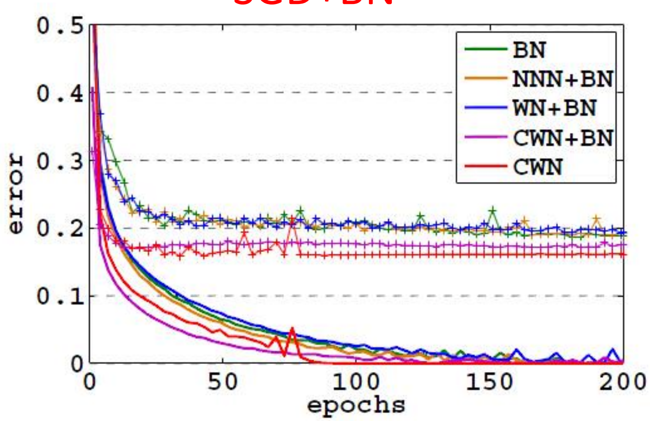
- MLP
  - SVHN, {128,128,128,128,128}

SGD optimization

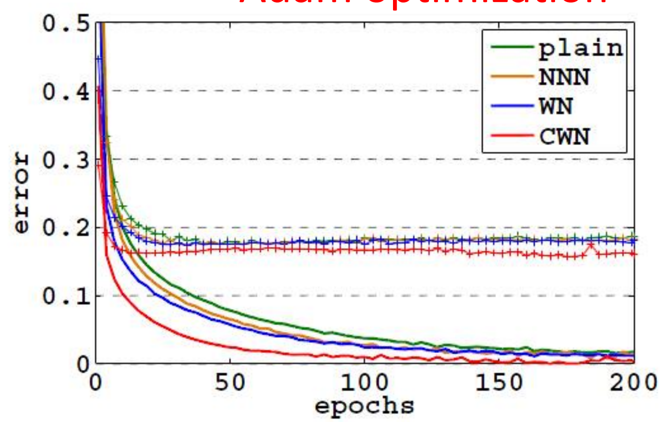




SGD+BN



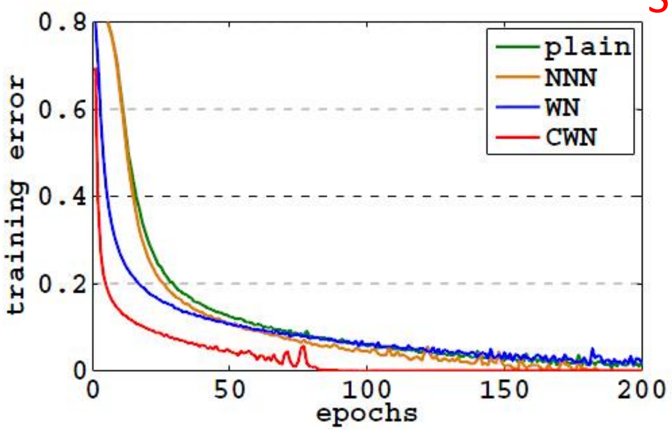
Adam optimization

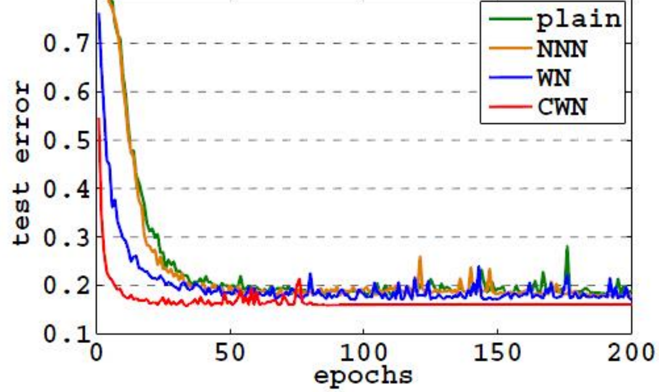


Experiments

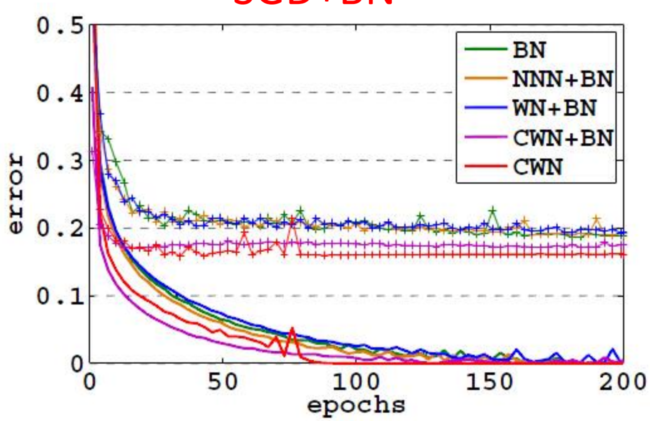
- MLP
  - SVHN, {128,128,128,128,128}

SGD optimization

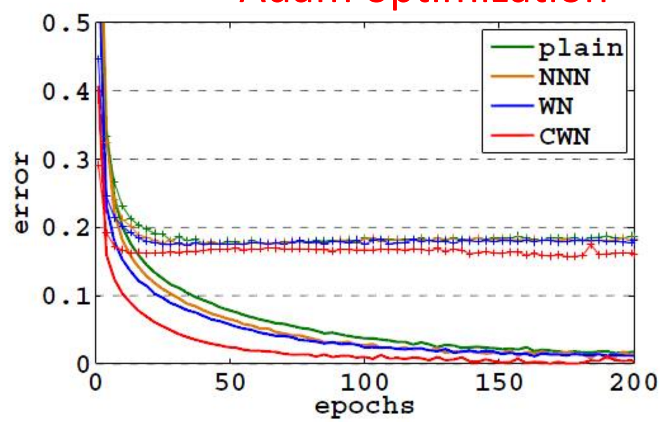




SGD+BN



Adam optimization



Experiments

- Cifar10 & Cifar100
  - BN-Inception

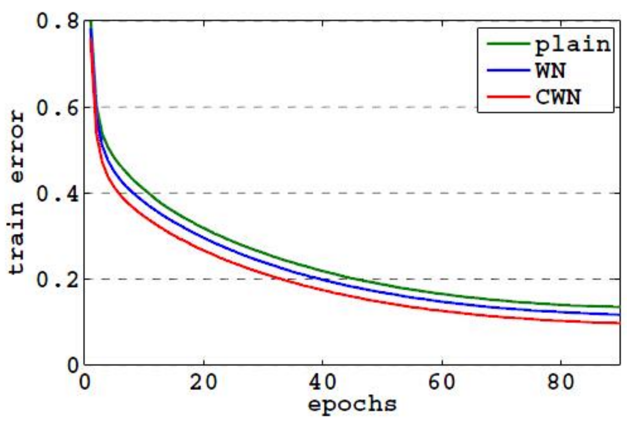
	Cifar-10	Cifar-100
Plain	6.14 ± 0.04	25.52 ± 0.15
WN	6.18 ± 0.34	25.49 ± 0.35
WCBN	**6.01 ± 0.16**	**24.45 ± 0.54**

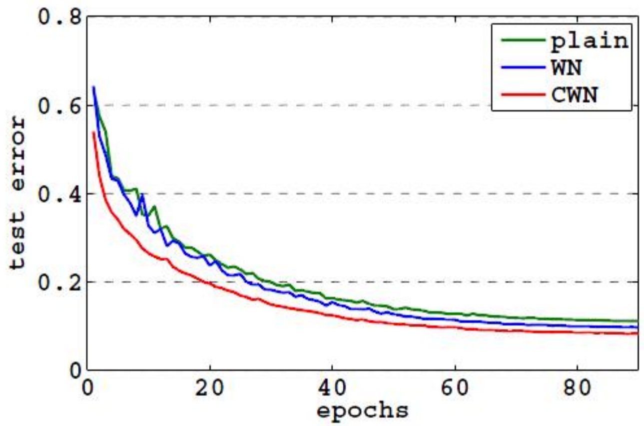
  - Residual Network-56 layers.

	Cifar-10	Cifar-100
Plain	7.34 ± 0.52	29.38 ± 0.14
WN	7.58 ± 0.40	29.85 ± 0.66
WCBN	**6.85 ± 0.25**	**29.23 ± 0.14**

Experiments

- ImageNet
  - BN-Inception





Methods	Top-1 error	Top-5 error
plain	30.78	11.14
WN	28.64	9.7
CWN	**26.1**	**8.35**

Conclusion and Feature work

- Conclusion:
  - CWN shows the advantages in accelerating training and better generalization
  - CWN module as a optimal module to replace linear module

- Apply CWN module
  - RNNs
  - Reinforcement learning scene

Thanks !