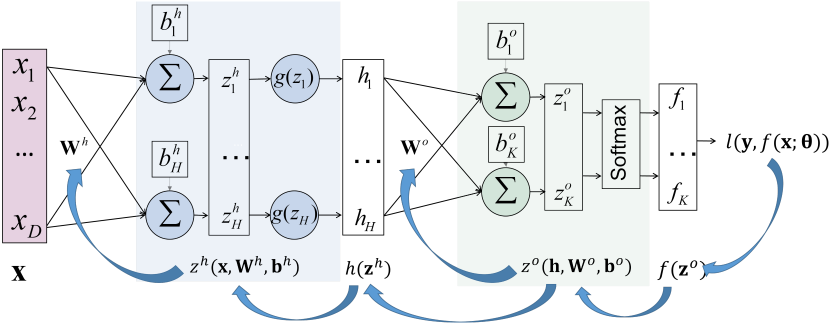
Class 4: Training Issues & Solutions

Instructor: Ruixuan Wang
wangruix5@mail.sysu.edu.cn

School of Computer Science and Engineering
Sun Yat-Sen University

March 21, 2025

Backpropagation: computation of gradient with chain rule



- Gradient of loss function over model parameters can be computed with chain rule.

1. Gradient exploding & vanishing
2. Mini-batch issue
3. ResNet and its extensions

A general model training process

Step 0: Pre-set hyper-parameters

Step 1: Initialize model parameters

Step 2: Repeat over certain number of epochs

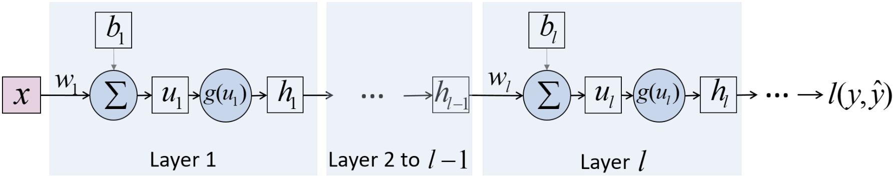
- Shuffle whole training data
- For each mini-batch data
  - load mini-batch data
  - compute gradient of loss over parameters
  - update parameters with gradient descent

Step 3: Save model (structure and parameters)

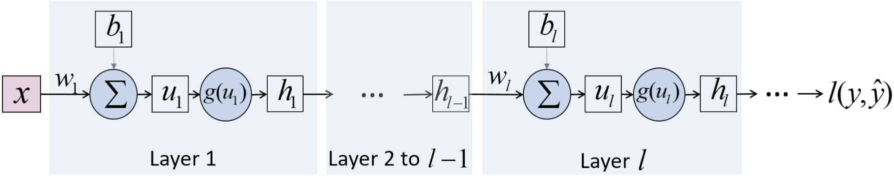
But sometimes...

The training is not working well!

Gradient issues for multi-layer networks

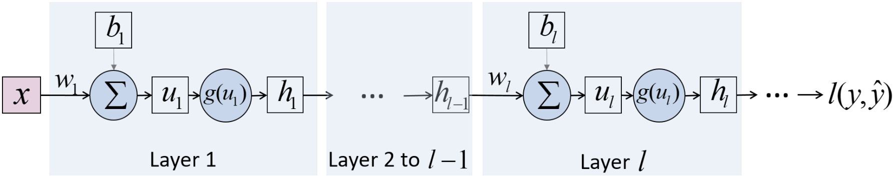


Gradient issues for multi-layer networks



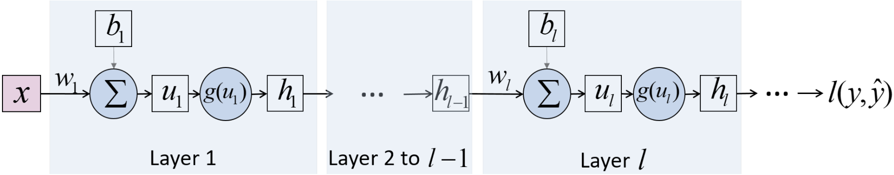
$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial h_l} \cdot \left(\frac{dh_l}{du_l} \cdot \frac{du_l}{dh_{l-1}}\right) \cdot \left(\frac{dh_{l-1}}{du_{l-1}} \cdot \frac{du_{l-1}}{dh_{l-2}}\right) \ldots \left(\frac{dh_1}{du_1} \cdot \frac{du_1}{dw_1}\right)$$

Gradient issues for multi-layer networks



$$
\begin{aligned}
\frac{\partial l}{\partial w_1} &= \frac{\partial l}{\partial h_l} \cdot \left(\frac{dh_l}{du_l} \cdot \frac{du_l}{dh_{l-1}}\right) \cdot \left(\frac{dh_{l-1}}{du_{l-1}} \cdot \frac{du_{l-1}}{dh_{l-2}}\right) \ldots \left(\frac{dh_1}{du_1} \cdot \frac{du_1}{dw_1}\right) \\
&= \frac{\partial l}{\partial h_l} \cdot (g'(u_l) \cdot w_l) \cdot (g'(u_{l-1}) \cdot w_{l-1}) \ldots (g'(u_1) \cdot x)
\end{aligned}
$$

Gradient issues for multi-layer networks

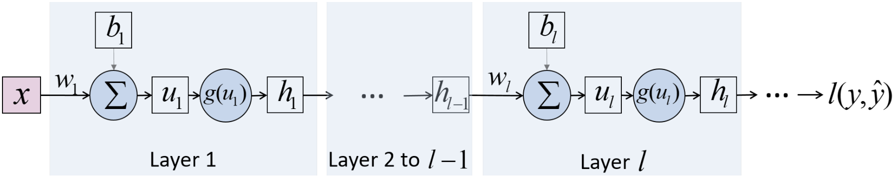


$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial h_l} \cdot \left(\frac{dh_l}{du_l} \cdot \frac{du_l}{dh_{l-1}}\right) \cdot \left(\frac{dh_{l-1}}{du_{l-1}} \cdot \frac{du_{l-1}}{dh_{l-2}}\right) \dots \left(\frac{dh_1}{du_1} \cdot \frac{du_1}{dw_1}\right)$$

$$= \frac{\partial l}{\partial h_l} \cdot (g'(u_l) \cdot w_l) \cdot (g'(u_{l-1}) \cdot w_{l-1}) \dots (g'(u_1) \cdot x)$$

- If each $|g'(u_i)w_i| > 1$, then $|\frac{\partial l}{\partial w_1}| \gg 1$, gradient exploding!

Gradient issues for multi-layer networks



$$
\begin{aligned}
\frac{\partial l}{\partial w_1} &= \frac{\partial l}{\partial h_l} \cdot \left(\frac{dh_l}{du_l} \cdot \frac{du_l}{dh_{l-1}}\right) \cdot \left(\frac{dh_{l-1}}{du_{l-1}} \cdot \frac{du_{l-1}}{dh_{l-2}}\right) \ldots \left(\frac{dh_1}{du_1} \cdot \frac{du_1}{dw_1}\right) \\
&= \frac{\partial l}{\partial h_l} \cdot (g'(u_l) \cdot w_l) \cdot (g'(u_{l-1}) \cdot w_{l-1}) \ldots (g'(u_1) \cdot x)
\end{aligned}
$$

- If each $|g'(u_i)w_i| > 1$, then $|\frac{\partial l}{\partial w_1}| \gg 1$, gradient exploding!
- If each $|g'(u_i)w_i| < 1$, then $|\frac{\partial l}{\partial w_1}| \ll 1$, gradient vanishing!

To avoid gradient exploding

Gradient exploding makes training process not stable!

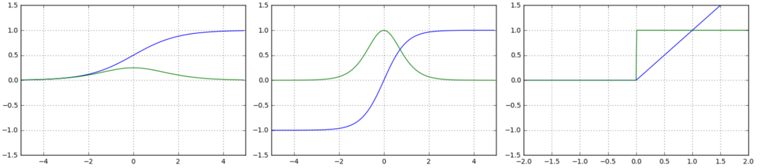
The issue would be gone if $|g'(u_i)| \leq 1$ and $|w_i| \leq 1$:

To avoid gradient exploding

Gradient exploding makes training process not stable!

The issue would be gone if $|g'(u_i)| \leq 1$ and $|w_i| \leq 1$:

- already $|g'(u_i)| \leq 1$



Sigmoid tanh ReLU

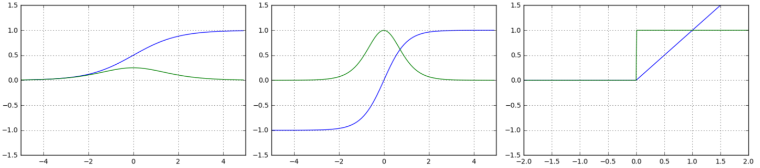
Blue: activation function; Green: derivative of activation

To avoid gradient exploding

Gradient exploding makes training process not stable!

The issue would be gone if $|g'(u_i)| \leq 1$ and $|w_i| \leq 1$:

- already $|g'(u_i)| \leq 1$



Sigmoid tanh ReLU

Blue: activation function; Green: derivative of activation

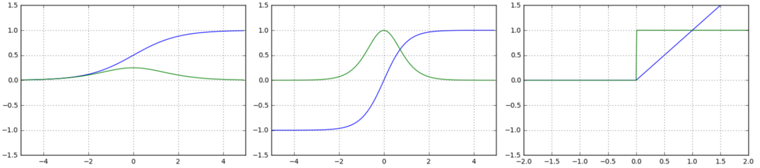
- weight initialization, such that $|w_i| \leq 1$ in general
- weight re-normalization during training

To avoid gradient exploding

Gradient exploding makes training process not stable!

The issue would be gone if $|g'(u_i)| \leq 1$ and $|w_i| \leq 1$:

- already $|g'(u_i)| \leq 1$



Sigmoid tanh ReLU

Blue: activation function; Green: derivative of activation

- weight initialization, such that $|w_i| \leq 1$ in general
- weight re-normalization during training
- rescaling x to $|x| \leq 1$

To reduce gradient vanishing

Gradient vanishing makes training very slow!

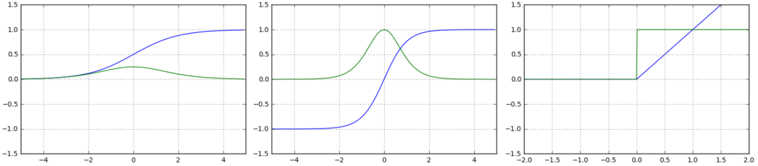
To reduce this issue, should make $|g'(u_i)w_i|$ not that small:

To reduce gradient vanishing

Gradient vanishing makes training very slow!

To reduce this issue, should make $|g'(u_i)w_i|$ not that small:

- choose ReLU activation function: $g'(u_i) = 1$ when $u_i > 0$.
  Sigmoid & tanh: $g'(u_i) \approx 0$ when $|u_i| \gg 1$



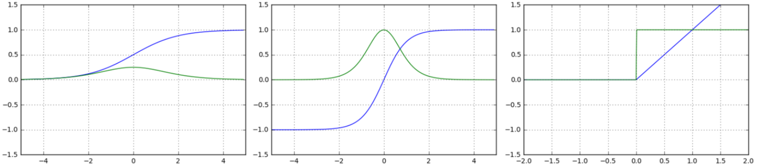
Sigmoid tanh ReLU

To reduce gradient vanishing

Gradient vanishing makes training very slow!

To reduce this issue, should make $|g'(u_i)w_i|$ not that small:

- choose ReLU activation function: $g'(u_i) = 1$ when $u_i > 0$. Sigmoid & tanh: $g'(u_i) \approx 0$ when $|u_i| \gg 1$

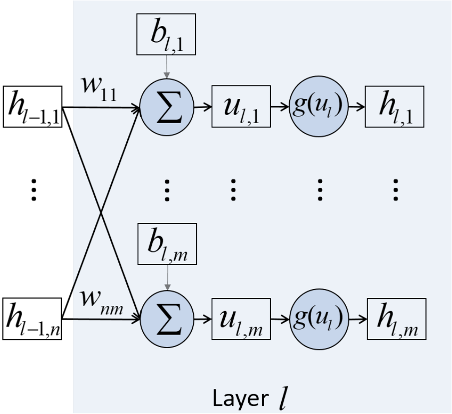


Sigmoid | tanh | ReLU

- many $|w_i|$ not close to 0 if variance of w_i not small!
  - weight initialization, $w_i \sim N(0, \sigma^2)$ or $w_i \sim U(-a, a)$
  - weight re-normalization during training

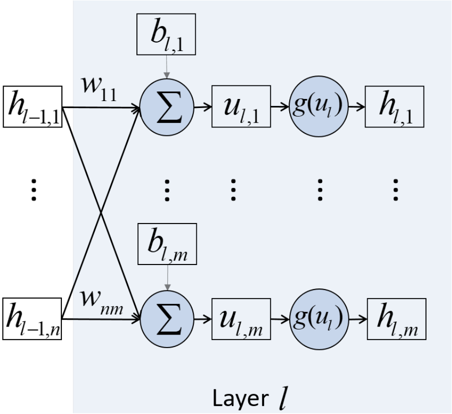
Weight initialization: Xavier's method

Rule: Signals across layers do not shrink and explode!



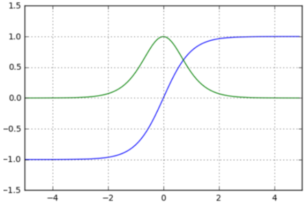
Weight initialization: Xavier's method

Rule: Signals across layers do not shrink and explode!



- Suppose $g(u_{l,k})$ roughly linear with smaller $u_{l,k}$, then

$$h_{l,k} \approx \sum_{j=1}^{n} h_{l-1,j} w_{j,k}$$



tanh

Weight initialization: Xavier's method (cont')

Rule: Signals across layers do not shrink and explode!

Or: Variance of signals across layers does not change!

Weight initialization: Xavier's method (cont')

Rule: Signals across layers do not shrink and explode!

Or: Variance of signals across layers does not change!

- Suppose input signals $\{h_{l-1,j}\}$ are independent and identically distributed, and have zero mean; similarly for $w_{j,k}$. Then

$$
\begin{aligned}
\mathrm{Var}(h_{l,k}) &\approx \sum_{j=1}^{n} \mathrm{Var}(h_{l-1,j})\mathrm{Var}(w_{j,k}) \\
\mathrm{Var}(h_l) &\approx n\mathrm{Var}(h_{l-1})\mathrm{Var}(w)
\end{aligned}
$$

Weight initialization: Xavier's method (cont')

Rule: Signals across layers do not shrink and explode!

Or: Variance of signals across layers does not change!

- Suppose input signals $\{h_{l-1,j}\}$ are independent and identically distributed, and have zero mean; similarly for $w_{j,k}$. Then

$$
\begin{aligned}
\mathrm{Var}(h_{l,k}) &\approx \sum_{j=1}^{n} \mathrm{Var}(h_{l-1,j})\mathrm{Var}(w_{j,k}) \\
\mathrm{Var}(h_l) &\approx n\mathrm{Var}(h_{l-1})\mathrm{Var}(w)
\end{aligned}
$$

- To make $\mathrm{Var}(h_l) \approx \mathrm{Var}(h_{l-1})$:

$$
\begin{aligned}
n\mathrm{Var}(w) &= 1 \\
\mathrm{Var}(w) &= \frac{1}{n}
\end{aligned}
$$

Weight initialization: Xavier's method (cont')

Rule: Signals across layers do not shrink and explode!

Or: Variance of signals across layers does not change!

$$\mathrm{Var}(w) = \frac{1}{n}$$

Weight initialization: Xavier's method (cont')

Rule: Signals across layers do not shrink and explode!

Or: Variance of signals across layers does not change!

$$\mathrm{Var}(w) = \frac{1}{n}$$

Also: Variance of backward gradient signals across layers does not change!

$$\mathrm{Var}(w) = \frac{1}{m}$$

Weight initialization: Xavier's method (cont')

Rule: Signals across layers do not shrink and explode!

Or: Variance of signals across layers does not change!

$$\mathrm{Var}(w) = \frac{1}{n}$$

Also: Variance of backward gradient signals across layers does not change!

$$\mathrm{Var}(w) = \frac{1}{m}$$

- Since the numbers of input and output (n and m) are often different at one layer, a compromise is:

$$\mathrm{Var}(w) = \frac{2}{n+m}$$

Weight initialization: Xavier's method (cont')

Rule: Signals across layers do not shrink and explode!

Or: Variance of signals across layer does not change!

Also: Variance of backward gradient signal across layer does not change!

- Weight initialization by sampling from Gaussian distribution

$$\mathrm{E}(w) = 0 \quad , \quad \mathrm{Var}(w) = \frac{2}{n+m}$$

Weight initialization: Xavier's method (cont')

Rule: Signals across layers do not shrink and explode!

Or: Variance of signals across layer does not change!

Also: Variance of backward gradient signal across layer does not change!

- Weight initialization by sampling from Gaussian distribution

$$\mathrm{E}(w) = 0 \quad , \quad \mathrm{Var}(w) = \frac{2}{n+m}$$

- Weight initialization by sampling from uniform distribution

$$w \quad \sim \quad \mathrm{U}\left[-\frac{\sqrt{6}}{\sqrt{n+m}}, \frac{\sqrt{6}}{\sqrt{n+m}}\right]$$

X. Glorot and Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, 2010.

Weight initialization: He's method

Xavier's method is not appropriate for ReLU activation!

- Xavier's method assumes activation output h_l has zero mean.
- Output from ReLU certainly has non-zero (positive) mean!

Weight initialization: He's method

Xavier's method is not appropriate for ReLU activation!

- Xavier's method assumes activation output h_l has zero mean.
- Output from ReLU certainly has non-zero (positive) mean!

He (Kaiming) proposed a method when activation is ReLU.

- Weight initialization by sampling from Gaussian distribution

$$\mathrm{E}(w) = 0 \quad , \quad \mathrm{Var}(w) = \frac{2}{n}$$

- Weight initialization by sampling from uniform distribution

$$w \sim \mathrm{U}[-\sqrt{\frac{6}{n}}, \sqrt{\frac{6}{n}}]$$

K. He, X. Zhang, S. Ren, and J. Sun, Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, 2015

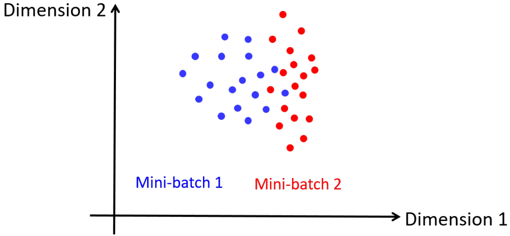
Training is slow

Weight initialization helps at the beginning!

But, training is often slow to converge!

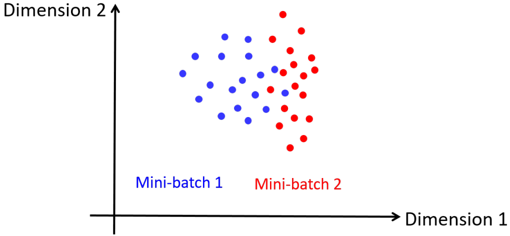
Issue of mini-batch

- Different mini-batch data often have different distributions



Issue of mini-batch

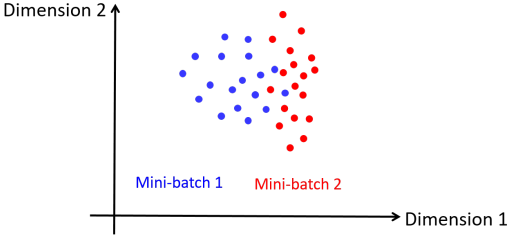
- Different mini-batch data often have different distributions



1. causes different mini-batch input distributions for every layer!
2. distribution of same minibatch changes over time for a layer!
3. each layer needs to continuously adapt to new distributions

Issue of mini-batch

- Different mini-batch data often have different distributions



1. causes different mini-batch input distributions for every layer!
2. distribution of same minibatch changes over time for a layer!
3. each layer needs to continuously adapt to new distributions

So, let's make different mini-batch inputs have similar distributions!

Batch normalization!

Batch normalization (BN)

For a layer with d-dimensional input $\mathbf{x} = (x_1, x_2, \ldots, x_d)^{\mathtt{T}}$,

- For any mini-batch input $\{\mathbf{x}_n\}$, normalize each dimension:

$$\hat{x}_k \quad = \quad \frac{x_k - \mathrm{E}(x_k)}{\sqrt{\mathrm{Var}(x_k) + \epsilon}}$$

$\mathrm{E}(x_k)$ and $\mathrm{Var}(x_k)$ are computed from all x_k's in $\{\mathbf{x}_n\}$.

Batch normalization (BN)

For a layer with d-dimensional input $\mathbf{x} = (x_1, x_2, \dots, x_d)^{\mathtt{T}}$,

- For any mini-batch input $\{\mathbf{x}_n\}$, normalize each dimension:

$$\hat{x}_k = \frac{x_k - \mathrm{E}(x_k)}{\sqrt{\mathrm{Var}(x_k) + \epsilon}}$$

  $\mathrm{E}(x_k)$ and $\mathrm{Var}(x_k)$ are computed from all x_k's in $\{\mathbf{x}_n\}$.
- However, such normalization reduces varieties of neurons' inputs/outputs, i.e., reducing layer's representation power.

Batch normalization (BN)

For a layer with d-dimensional input $\mathbf{x} = (x_1, x_2, \ldots, x_d)^{\mathtt{T}}$,

- For any mini-batch input $\{\mathbf{x}_n\}$, normalize each dimension:

$$\hat{x}_k = \frac{x_k - \mathrm{E}(x_k)}{\sqrt{\mathrm{Var}(x_k) + \epsilon}}$$

  $\mathrm{E}(x_k)$ and $\mathrm{Var}(x_k)$ are computed from all x_k's in $\{\mathbf{x}_n\}$.

- However, such normalization reduces varieties of neurons' inputs/outputs, i.e., reducing layer's representation power.
- To recover neuron's representation variety

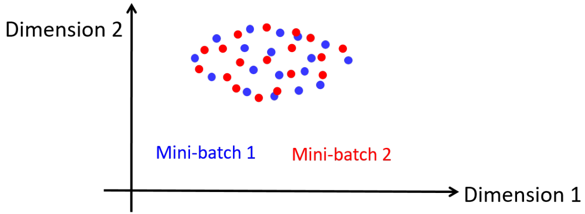
$$y_k = \gamma_k \hat{x}_k + \beta_k \equiv \mathrm{BN}_{\gamma_k, \beta_k}(x_k)$$

  γ_k and β_k are independent of mini-batch data!

S. Ioffe and C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015

Batch normalization (cont')

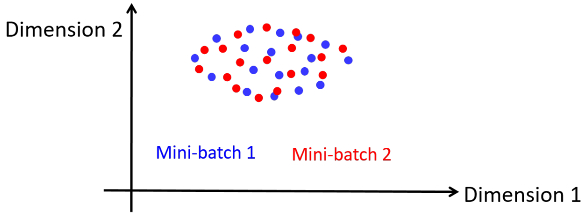
- Now, different mini-batches have similar distributions for a layer



Different input dimensions (neurons) have different γ_k and β_k

Batch normalization (cont')

- Now, different mini-batches have similar distributions for a layer

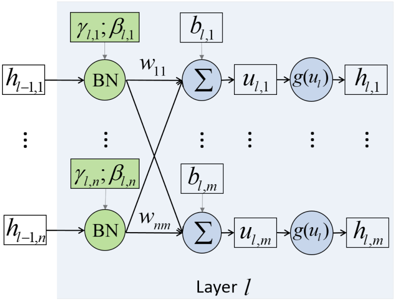


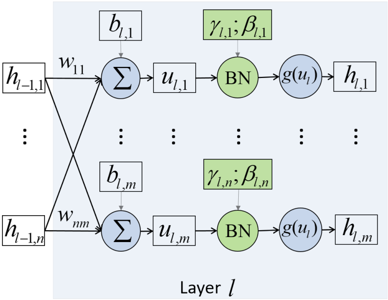
Different input dimensions (neurons) have different γ_k and β_k

- But, how to get γ_k and β_k for each neuron at each layer?

Batch normalization (cont')

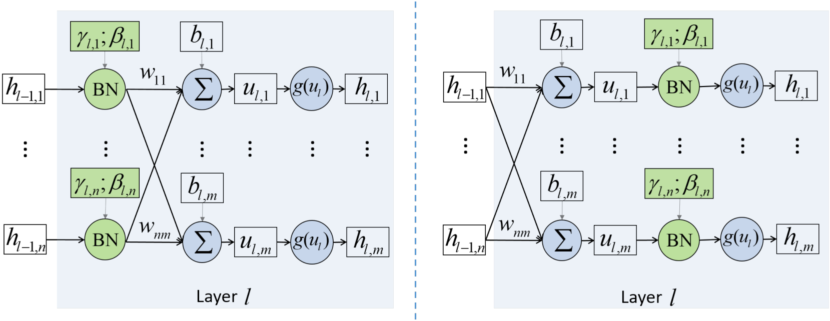
- Solution: consider γ_k and β_k as part of model parameters





Batch normalization (cont')

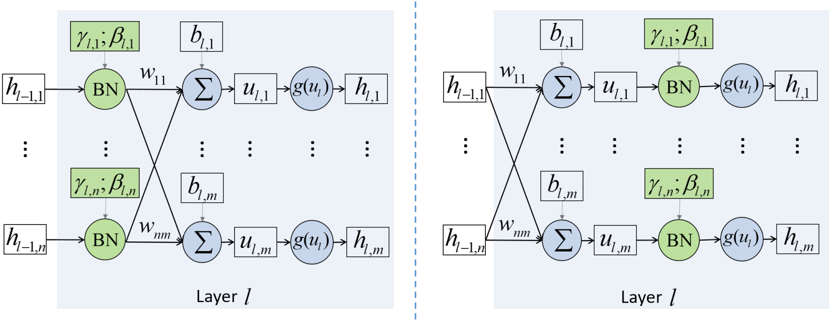
- Solution: consider γ_k and β_k as part of model parameters



- Left: not ideal to normalize input (from non-linear activation)
- Right: BN at pre-activation gives a 'more Gaussian' result

Batch normalization (cont')

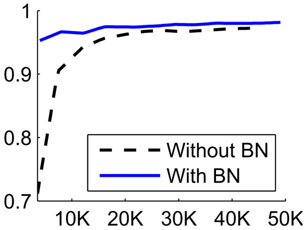
- Solution: consider γ_k and β_k as part of model parameters



- Left: not ideal to normalize input (from non-linear activation)
- Right: BN at pre-activation gives a 'more Gaussian' result
- Q: should perform weight decay on BN parameters?

Batch normalization (cont')

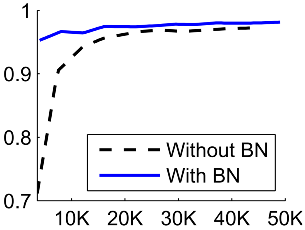
- Effect of BN: helps train faster and achieve higher accuracy (Horizontal axis: training iterations; vertical: testing accuracy)



Wu, Johnson, "Rethinking Batch in BatchNorm", arXiv 2021

Batch normalization (cont')

- Effect of BN: helps train faster and achieve higher accuracy (Horizontal axis: training iterations; vertical: testing accuracy)

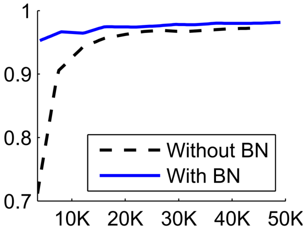


- However, BN not work well when batch size is small (e.g., 4)

Wu, Johnson, "Rethinking Batch in BatchNorm", arXiv 2021

Batch normalization (cont')

- Effect of BN: helps train faster and achieve higher accuracy (Horizontal axis: training iterations; vertical: testing accuracy)

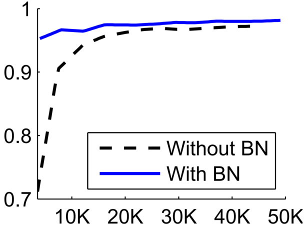


- However, BN not work well when batch size is small (e.g., 4)
- Refinement: use multi mini-batches to estimate mean and var.
- Extension: Group/Layer/Instance normalization

Wu, Johnson, "Rethinking Batch in BatchNorm", arXiv 2021

Batch normalization (cont')

- Effect of BN: helps train faster and achieve higher accuracy (Horizontal axis: training iterations; vertical: testing accuracy)



- However, BN not work well when batch size is small (e.g., 4)
- Refinement: use multi mini-batches to estimate mean and var.
- Extension: Group/Layer/Instance normalization
- Q: during inference, how to get mean $\mathrm{E}(x_k)$ and variance $\mathrm{Var}(x_k)$ for each neuron?

Wu, Johnson, "Rethinking Batch in BatchNorm", arXiv 2021

So far, so good

So far, the network can be trained fast with BN!

But when model is deeper (with more layers)...

Problem of deeper networks

- However, more layers caused larger **training** and test error!

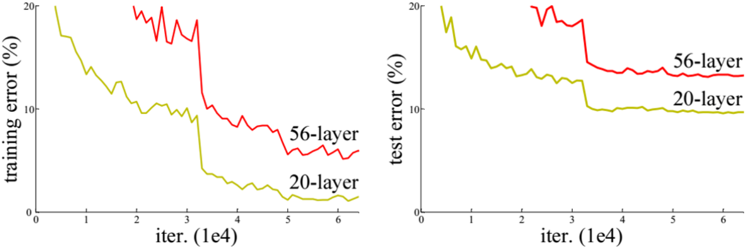
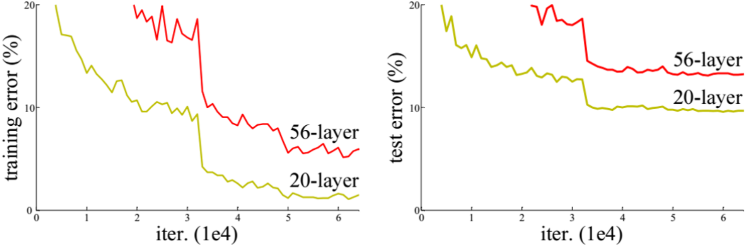


Figure from He, Zhang, Ren, Sun, "Deep residual learning for image recognition", CVPR 2016

Problem of deeper networks

- However, more layers caused larger **training** and test error!



- Deeper network not overfitting, but harder to optimize!

Figure from He, Zhang, Ren, Sun, "Deep residual learning for image recognition", CVPR 2016

ResNet

Solution: use network layer to learn residual mapping rather than directly to learn a desired underlying mapping!

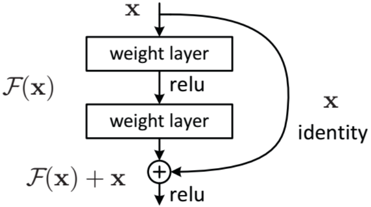


Figure 2. Residual learning: a building block.

ResNet

Solution: use network layer to learn residual mapping rather than directly to learn a desired underlying mapping!

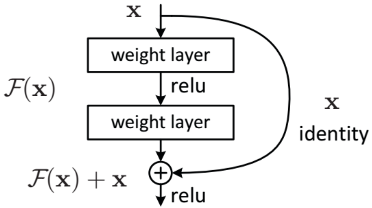


Figure 2. Residual learning: a building block.

- Learning residual between desired mapping $\mathcal{H}(\mathbf{x})$ and input $\mathbf{x}$

$$\begin{aligned} \mathcal{H}(\mathbf{x}) &= \mathcal{F}(\mathbf{x}) + \mathbf{x} \\ \mathcal{F}(\mathbf{x}) &= \mathcal{H}(\mathbf{x}) - \mathbf{x} \end{aligned}$$

ResNet

Solution: use network layer to learn residual mapping rather than directly to learn a desired underlying mapping!

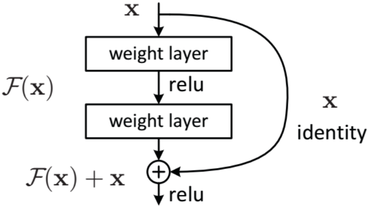
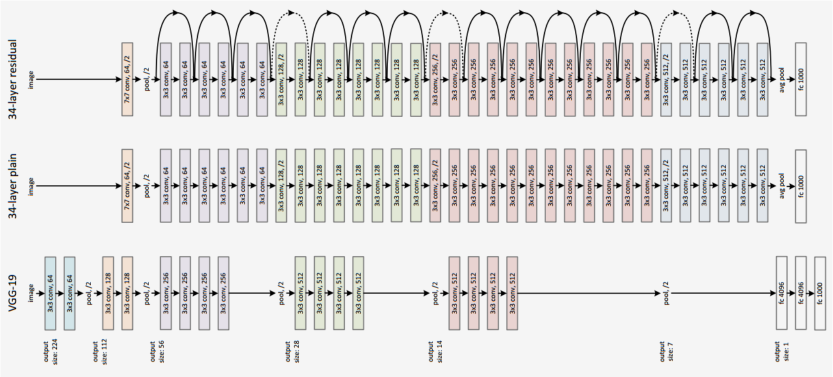


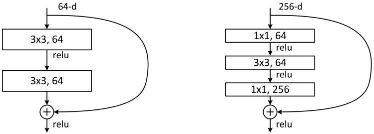
Figure 2. Residual learning: a building block.

- Learning residual between desired mapping $\mathcal{H}(\mathbf{x})$ and input $\mathbf{x}$

$$\begin{aligned}\mathcal{H}(\mathbf{x}) &= \mathcal{F}(\mathbf{x})+\mathbf{x}\\ \mathcal{F}(\mathbf{x}) &= \mathcal{H}(\mathbf{x})-\mathbf{x}\end{aligned}$$

- If $\mathcal{H}(\mathbf{x})$ is identity mapping, it is easier to push residual to zero than to fit an identity mapping by a stack of nonlinear layers





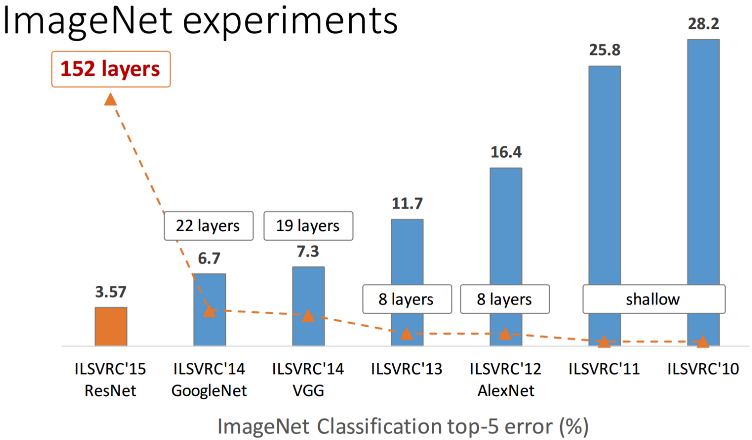
Figures and tables from He, Zhang, Ren, Sun, "Deep residual learning for image recognition", CVPR 2016

ResNet (cont')

ResNets with different number of layers

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ResNet (cont')

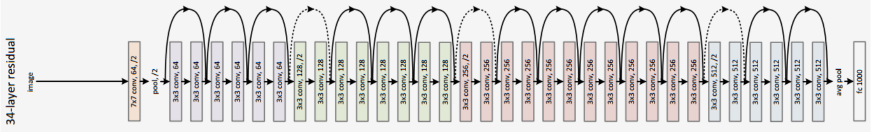


- Better than humans on 1000-class image classification!

figure from Kaiming He, "Deep residual learning for image recognition", tutorial on ICML 2016

Why ResNets work better?

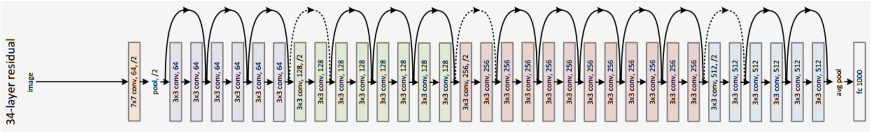
- Train/update each layer easier, by skip connections
- An ensemble of CNN models with different architectures



He, Zhang, Ren, Sun, Identity Mappings in Deep Residual Networks, ECCV 2016; Littwin and Wolf, The Loss Surface of Residual Networks: Ensembles and the Role of Batch Normalization, ICLR 2017.

Why ResNets work better?

- Train/update each layer easier, by skip connections
- An ensemble of CNN models with different architectures



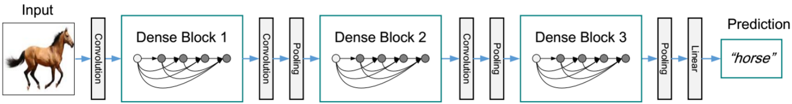
He, Zhang, Ren, Sun, Identity Mappings in Deep Residual Networks, ECCV 2016; Littwin and Wolf, The Loss Surface of Residual Networks: Ensembles and the Role of Batch Normalization, ICLR 2017.

Deep Residual Learning for Image Recognition

Kaiming He   Xiangyu Zhang   Shaoqing Ren   Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

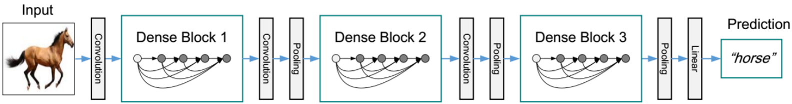
ResNet extensions

- DenseNets: densely skip connections within each block; Fewer kernels (parameters) in each layer, why?



ResNet extensions

- DenseNets: densely skip connections within each block; Fewer kernels (parameters) in each layer, why?



- Wide ResNets, ResNeXt, SENet, EfficientNet, etc.

Huang, Liu, van der Maaten, Weinberger, "Densely connected convolutional networks", CVPR 2017;
Zagoruyko, Komodakis, "Wide residual networks", BMVC 2016;
Xie, Girshick, Dollar, Tu, He, "Aggregated residual transformations for deep neural networks", CVPR 2017;
Hu, Shen, Sun, "Squeeze-and-Excitation Networks", CVPR 2018;
Tan, Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks", ICML 2019;

Summary

- Gradient exploding and vanishing happen during training
- Solved by ReLU, weight initialization, input normalization, etc.
- Batch normalization speeds up training
- ResNet with more layers can be trained well

Further reading:

- Sections 8.7.1, "Deep learning", http://www.deeplearningbook.org/
- BN: https://zhuanlan.zhihu.com/p/437446744
- ResNet: www.bilibili.com/video/BV1P3411y7nn/?spm_id_from=333.999.0.0