Week 3: Issues in Training

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

School of Data and Computer Science
Sun Yat-Sen University

14 March, 2019

1. Gradient exploding & vanishing
2. Mini-batch issue
3. Overfitting issue

A general model training process

Step 0: Pre-set hyper-parameters

Step 1: Initialize model parameters

Step 2: Repeat over certain number of epochs

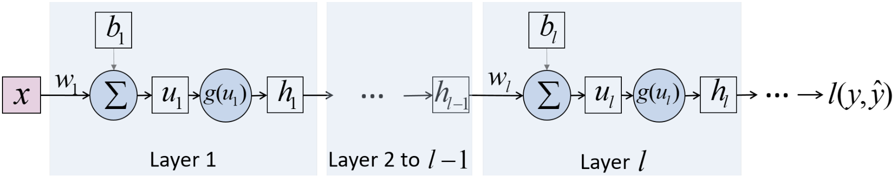
- Shuffle whole training data
- For each mini-batch data
  - load mini-batch data
  - compute gradient of loss over parameters
  - update parameters with gradient descent

Step 3: Save model (structure and parameters)

But sometimes...

The training is not working well!

Gradient issues for multi-layer networks

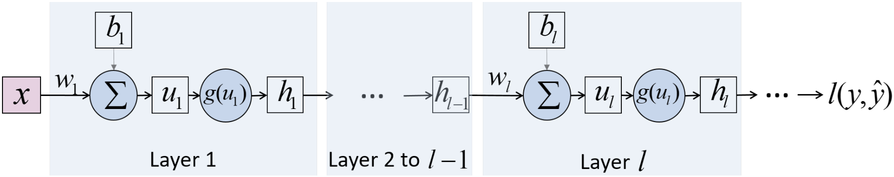


$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial h_l} \cdot \left(\frac{dh_l}{du_l} \cdot \frac{du_l}{dh_{l-1}}\right) \cdot \left(\frac{dh_{l-1}}{du_{l-1}} \cdot \frac{du_{l-1}}{dh_{l-2}}\right) \dots \left(\frac{dh_1}{du_1} \cdot \frac{du_1}{dw_1}\right)$$

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

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

Gradient issues for multi-layer networks

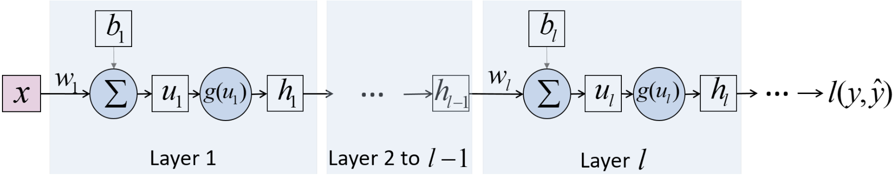


$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial h_l} \cdot \left(\frac{dh_l}{du_l} \cdot \frac{du_l}{dh_{l-1}}\right) \cdot \left(\frac{dh_{l-1}}{du_{l-1}} \cdot \frac{du_{l-1}}{dh_{l-2}}\right) \dots \left(\frac{dh_1}{du_1} \cdot \frac{du_1}{dw_1}\right)$$

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

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

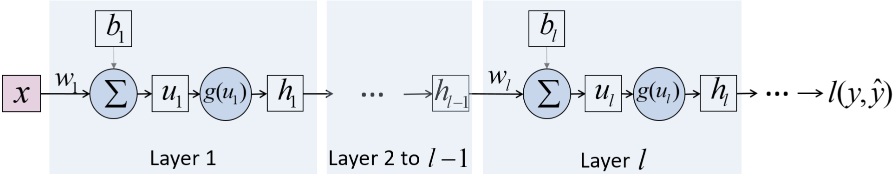
Gradient issues for multi-layer networks



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

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

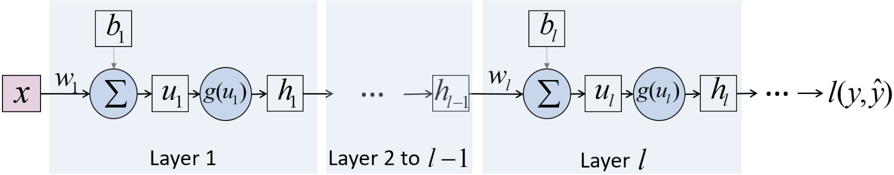
Gradient issues for multi-layer networks



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

- 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!

Gradient issues for multi-layer networks



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

- 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 $|w_i| \leq 1$:

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

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 $|x| \leq 1$

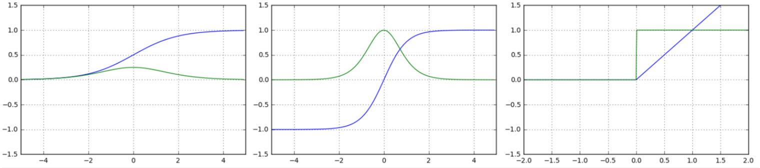
weight re-normalization: https://arxiv.org/abs/1602.07868

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 $|x| \leq 1$

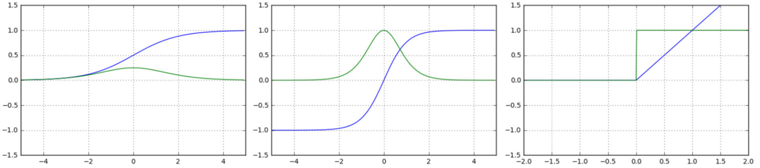
weight re-normalization: https://arxiv.org/abs/1602.07868

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

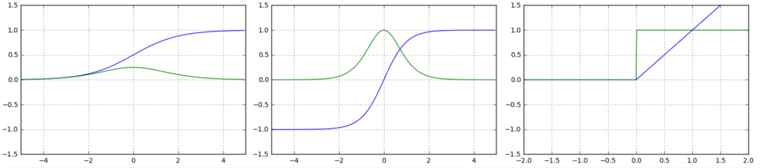
- weight initialization, such that $|w_i| \leq 1$ in general
- weight re-normalization during training
- rescaling x to $|x| \leq 1$

To 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 $|x| \leq 1$

weight re-normalization: https://arxiv.org/abs/1602.07868

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 $|u_i| \gg 1$

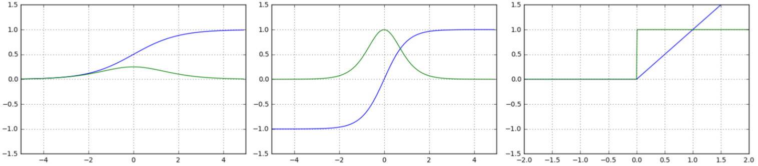
- most $|w_i|$ not close to 0 if variance of w_i not small!
  - weight initialization, $w_i \sim N(0, \sigma^2)$ or $w_i \sim U(-a, a)$
  - weight re-normalization during training

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

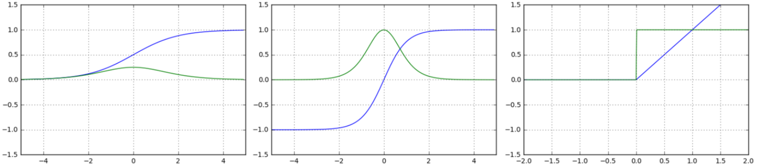
- most $|w_i|$ not close to 0 if variance of w_i not small!
  - weight initialization, $w_i \sim N(0, \sigma^2)$ or $w_i \sim U(-a, a)$
  - weight re-normalization during training

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 $|u_i| \gg 1$

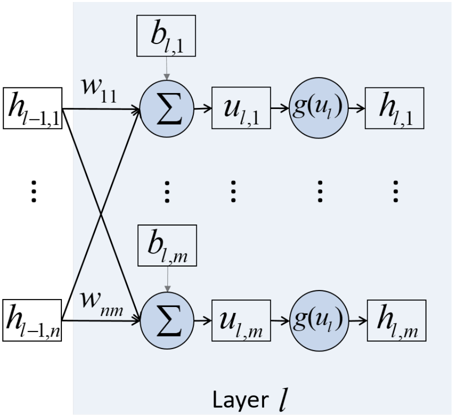


Sigmoid tanh ReLU

- most $|w_i|$ 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: Signal across layer does not shrink and explode!

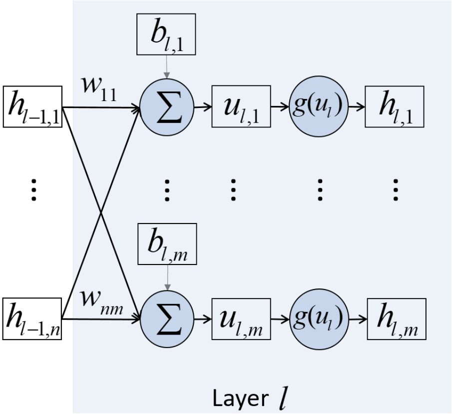


- Suppose $g(u_{l,k})$ roughly linear with smaller $u_{l,k}$, then

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

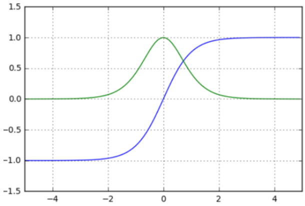
Weight initialization: Xavier's method

Rule: Signal across layer does 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: Signal across layer does not shrink and explode!

Or: Variance of signal across layer 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: Signal across layer does not shrink and explode!

Or: Variance of signal across layer 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: Signal across layer does not shrink and explode!

Or: Variance of signal across layer 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: Signal across layer does not shrink and explode!

Or: Variance of signal across layer does not change!

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

Also: Variance of backward gradient signal across layer 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: Signal across layer does not shrink and explode!

Or: Variance of signal across layer does not change!

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

Also: Variance of backward gradient signal across layer 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: Signal across layer does not shrink and explode!

Or: Variance of signal across layer does not change!

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

Also: Variance of backward gradient signal across layer 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: Signal across layer does not shrink and explode!

Or: Variance of signal 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: Xavier's method (cont')

Rule: Signal across layer does not shrink and explode!

Or: Variance of signal 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

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

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!

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

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

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

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

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

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

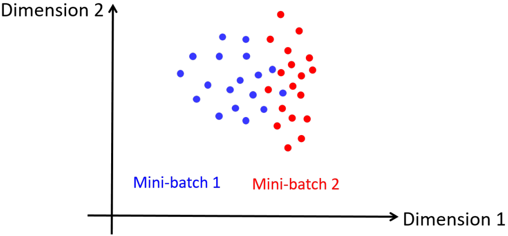
Training is slow

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



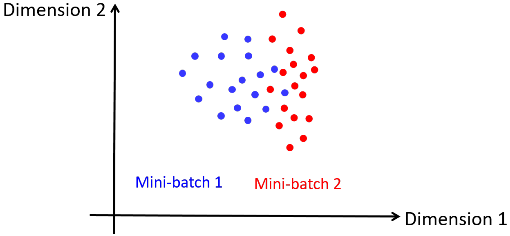
- Caused different mini-batch input distributions for every layer!
- Distribution of one minibatch changes over time for a layer!
- Each layer needs to continuously adapt to new distributions

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

Batch normalization!

Issue of mini-batch

- Different mini-batch data often have different distributions



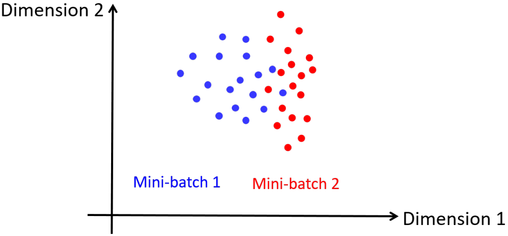
- Caused different mini-batch input distributions for every layer!
- Distribution of one minibatch changes over time for a layer!
- Each layer needs to continuously adapt to new distributions

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

Batch normalization!

Issue of mini-batch

- Different mini-batch data often have different distributions



- Caused different mini-batch input distributions for every layer!
- Distribution of one minibatch changes over time for a layer!
- 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:

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

$\mathrm{E}(x_k)$ 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 (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 (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

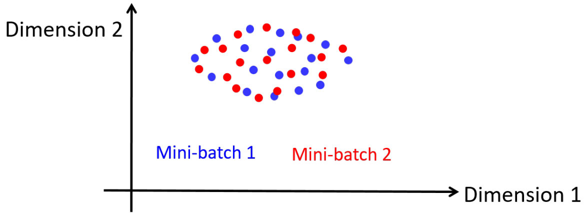
$$y_k = \gamma_k \hat{x}_k + \beta_k \equiv \mathrm{BN}_{\gamma_k, \beta_k}(x_k)$$

  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

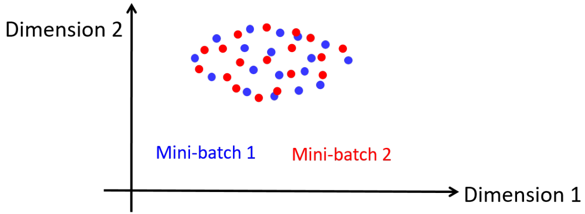


  Different input dimensions may have different γ_k and β_k

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

Batch normalization (cont')

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

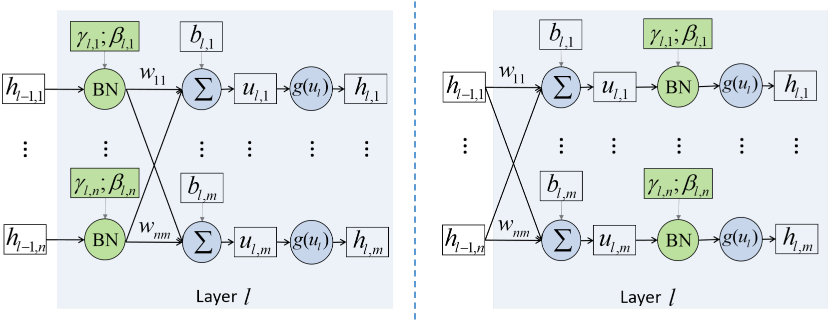


  Different input dimensions may have different γ_k and β_k

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

Batch normalization (cont')

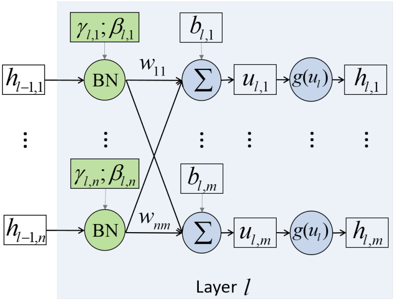
- Solution: consider γ_k and β_k as part of model parameters

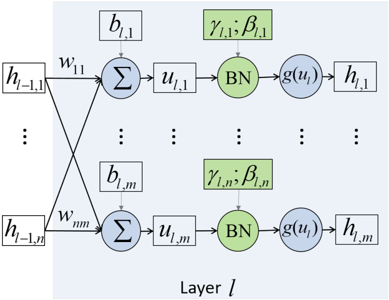


- Left: Not ideal to normalize input (from non-linear activation)
- Right: BN at pre-activation gives a 'more Gaussian' result

Batch normalization (cont')

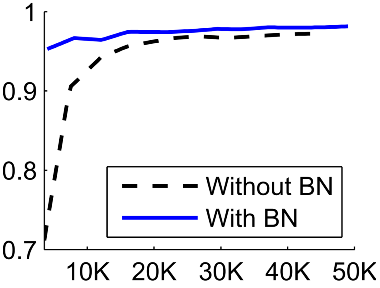
- Solution: consider γ_k and β_k as part of model parameters





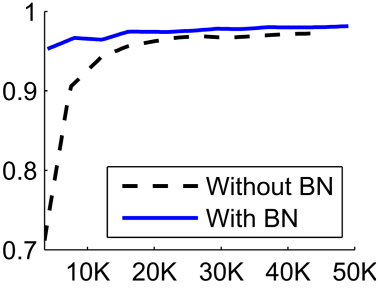
- Left: Not ideal to normalize input (from non-linear activation)
- Right: BN at pre-activation gives a 'more Gaussian' result

Batch normalization (cont')



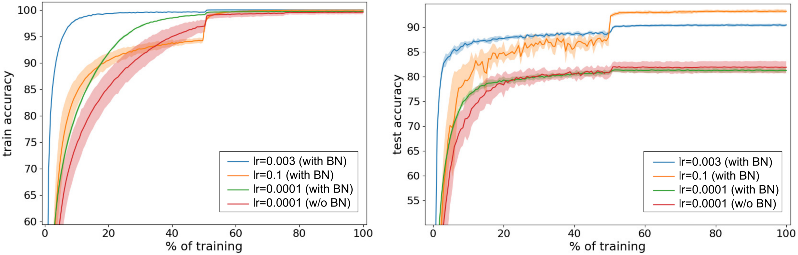
- Horizontal axis: training iterations; vertical: testing accuracy
- BN helps train faster and achieve higher accuracy.
- However, BN not work well when batch size is small (e.g., 4)

Batch normalization (cont')



- Horizontal axis: training iterations; vertical: testing accuracy
- BN helps train faster and achieve higher accuracy.
- However, BN not work well when batch size is small (e.g., 4)

Why BN works?



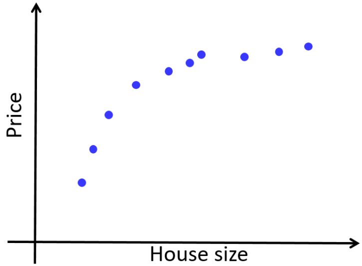
- Small learning rate ($lr = 0.0001$): networks with and w/t BN perform similarly in testing accuracy with .
- Larger learning rate: higher testing accuracy with BN networks (blue & orange); diverge without BN (not shown).

So far, so good

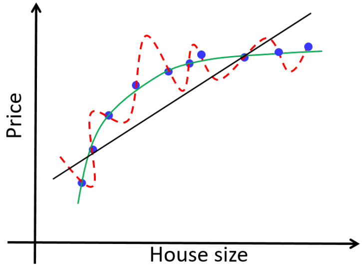
So far, the network can be trained fast with BN!

But when to stop training?

Overfitting issue

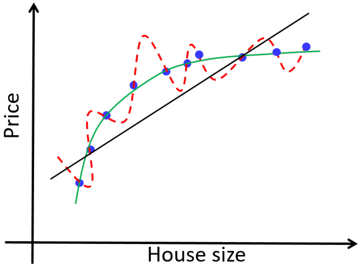


Overfitting issue



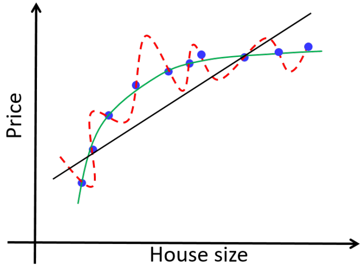
- Overfitting (red curve): trained to predict training data too accurate to be generalizable!

Overfitting issue



- Overfitting (red curve): trained to predict training data too accurate to be generalizable!

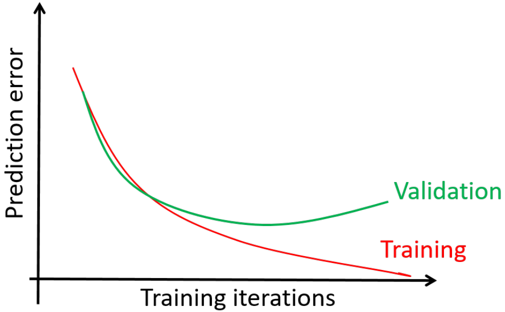
Overfitting issue



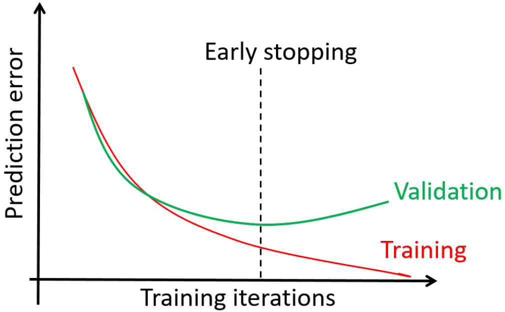
- Overfitting (red curve): trained to predict training data too accurate to be generalizable!

死记硬背 ➡ 过犹不及

Prevent overfitting: early stopping

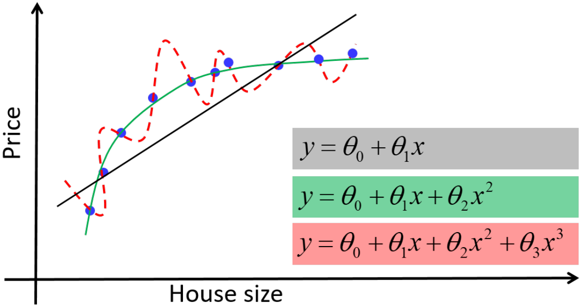


Prevent overfitting: early stopping



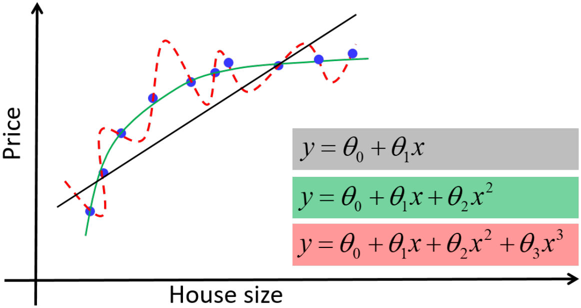
- Early stopping: stop training when prediction error on validation set does not decrease.

Regularization: L_p norm



- More model parameters, more likely to be overfitting
- Fewer model parameters, more likely to have larger loss
- So: need trade-off between loss and number of working parameters.

Regularization: L_p norm



- More model parameters, more likely to be overfitting
- Fewer model parameters, more likely to have larger loss
- So: need trade-off between loss and number of working parameters.

Regularization: L_p norm (cont')

L_p regularization

Adding a penalty on large parameter values with L_p norm in the loss function to reduce overfitting:

$$L(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N} l(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n; \boldsymbol{\theta})) + \lambda \|\boldsymbol{\theta}\|_p$$

- L_p norm $\|\boldsymbol{\theta}\|_p \equiv (\sum_i |\theta_i|^p)^{1/p}$
- λ: a hyper-parameter to balance two terms
- $p = 2$: "weight decay", causing smaller weight values
- $p = 1$: causing fewer non-zero weight parameters

Regularization: L_p norm (cont')

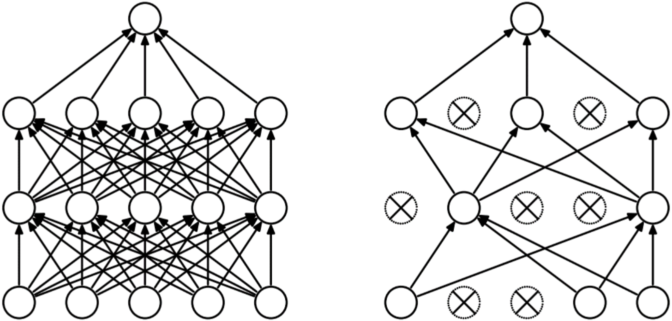
L_p regularization

Adding a penalty on large parameter values with L_p norm in the loss function to reduce overfitting:

$$L(\boldsymbol{\theta}) \quad = \quad \frac{1}{N}\sum_{n=1}^{N} l(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n;\boldsymbol{\theta})) + \lambda\|\boldsymbol{\theta}\|_p$$

- L_p norm $\|\boldsymbol{\theta}\|_p \equiv (\sum_i |\theta_i|^p)^{1/p}$
- λ: a hyper-parameter to balance two terms
- $p = 2$: "weight decay", causing smaller weight values
- $p = 1$: causing fewer non-zero weight parameters

Regularization: Dropout



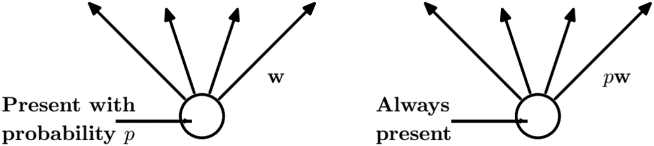
(a) Standard Neural Net

(b) After applying dropout.

- At training, each hidden neuron is present (not dropped out) with probability p
- So, each mini-batch is to train a different random structure

Srivastava et al., Dropout: A Simple Way to Prevent Neural Networks from Overfitting, 2014

Regularization: Dropout (cont')

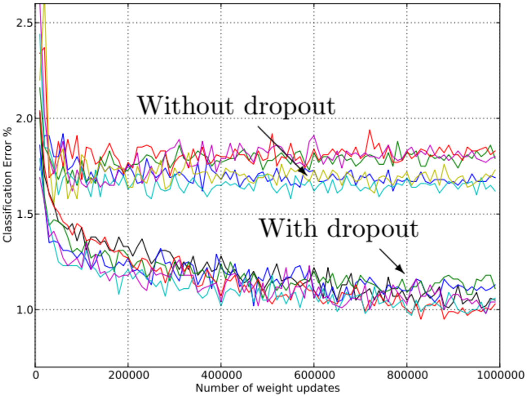


(a) At training time

(b) At test time

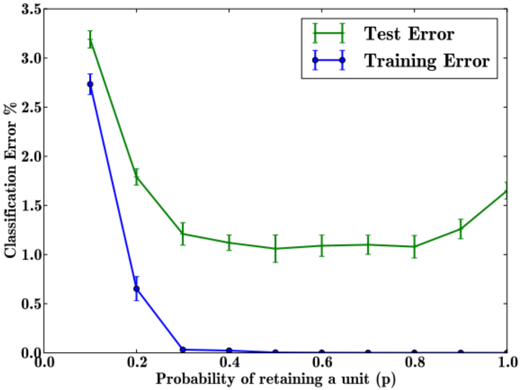
- At test, every neuron is always present. Weights are (down-) scaled by p, such that output at test time is same as expected output at training time.

Regularization: Dropout (cont')



- Dropout reduces test errors on different model architectures (each architecture with a unique color)

Regularization: Dropout (cont')



- Dropout works well at large range of rate p.

Regularization: Dropout (cont')

Why does dropout work?

- At each training, every retaining neuron is forced to finish the task with less help from other neurons.
- At test time, the whole network approximates the average over many 'thinned' (with some neurons dropped) networks.

Drawback of dropout:

- It takes 2-3 times longer in training

Regularization: Dropout (cont')

Why does dropout work?

- At each training, every retaining neuron is forced to finish the task with less help from other neurons.
- At test time, the whole network approximates the average over many 'thinned' (with some neurons dropped) networks.

Drawback of dropout:

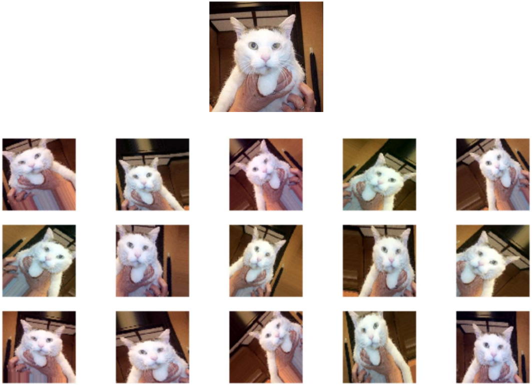
- It takes 2-3 times longer in training

More generalization ideas

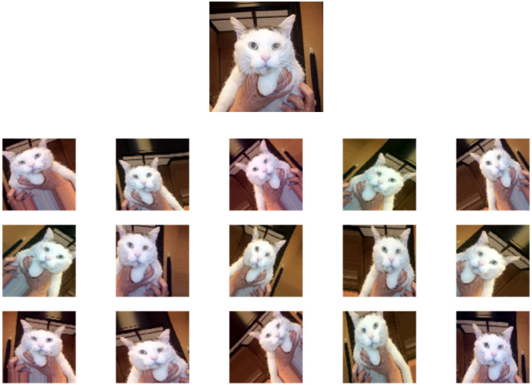
Besides above regularization techniques, there are other effective ways to improve model's generalization ability!

Data augmentation

Data augmentation



Data augmentation



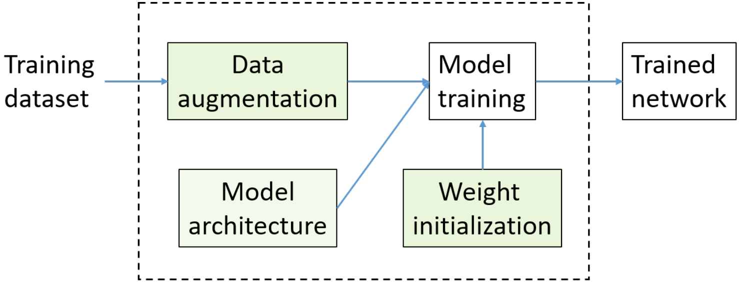
- Augmentation ways: rotate, scale, translate, flip, shear, deform, color and illumination change, etc
- Data augmentation produced more training data

Ensemble model

- Use a **group** of models (experts) to predict result!
- First, train multiple slightly different networks
- Networks are different due to different weight initialization, augmented data, and possibly different model architectures.

Ensemble model

- Use a **group** of models (experts) to predict result!
- First, train multiple slightly different networks

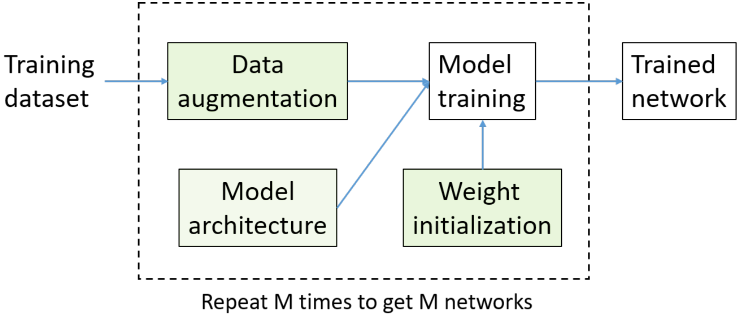


Repeat M times to get M networks

- Networks are different due to different weight initialization, augmented data, and possibly different model architectures.

Ensemble model

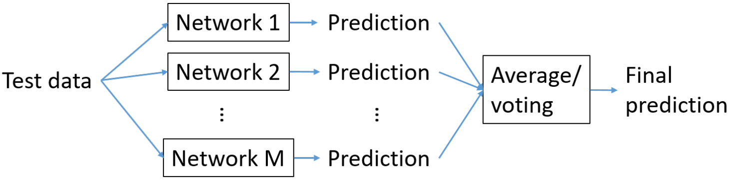
- Use a **group** of models (experts) to predict result!
- First, train multiple slightly different networks



- Networks are different due to different weight initialization, augmented data, and possibly different model architectures.

Ensemble model (cont')

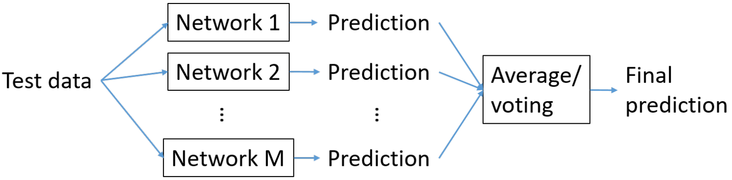
- Then, collect predictions of all experts for final prediction



- Ensemble model generalizes better (lower test error)

Ensemble model (cont')

- Then, collect predictions of all experts for final prediction



- Ensemble model generalizes better (lower test error)

Summary

- Gradient issues solved by ReLU, weight initialization, input normalization, etc.
- Batch normalization speeds up training.
- Generalization improved by early stopping, L_p regularization, dropout, data augmentation, and ensemble model, etc.

Further reading:

- Sections 7.1, 7.2, 7.4, 7.8, 7.11, 7.12, 8.7.1, in textbook "Deep learning", http://www.deeplearningbook.org/

About projects

Course project deadlines:

- Team established: 17 March, 2019
- Contest selected and summarized: 31 March, 2019
- Mid-term report: 21 April, 1 method+result
- Final report: 30 June, 2019

Lab project deadlines:

- Paper selected: 21 April, 2019
- Mid-term report: 12 May, method+first result
- Final report: 23 June, 2019