# Tips for Training Deep Neural Network

Hung-yi Lee

# Announcement

- 分組
  - 請確認在 ceiba 上的分組是否正確
- HW1
  - 截止日期: 10/23 2:00 p.m. (下週五上課前)
- HW2
  - 公告日期: 10/23
  - 截止日期: 11/13 2:00 p.m.
    - 比第一堂課公告的提早一週截止

# Outline

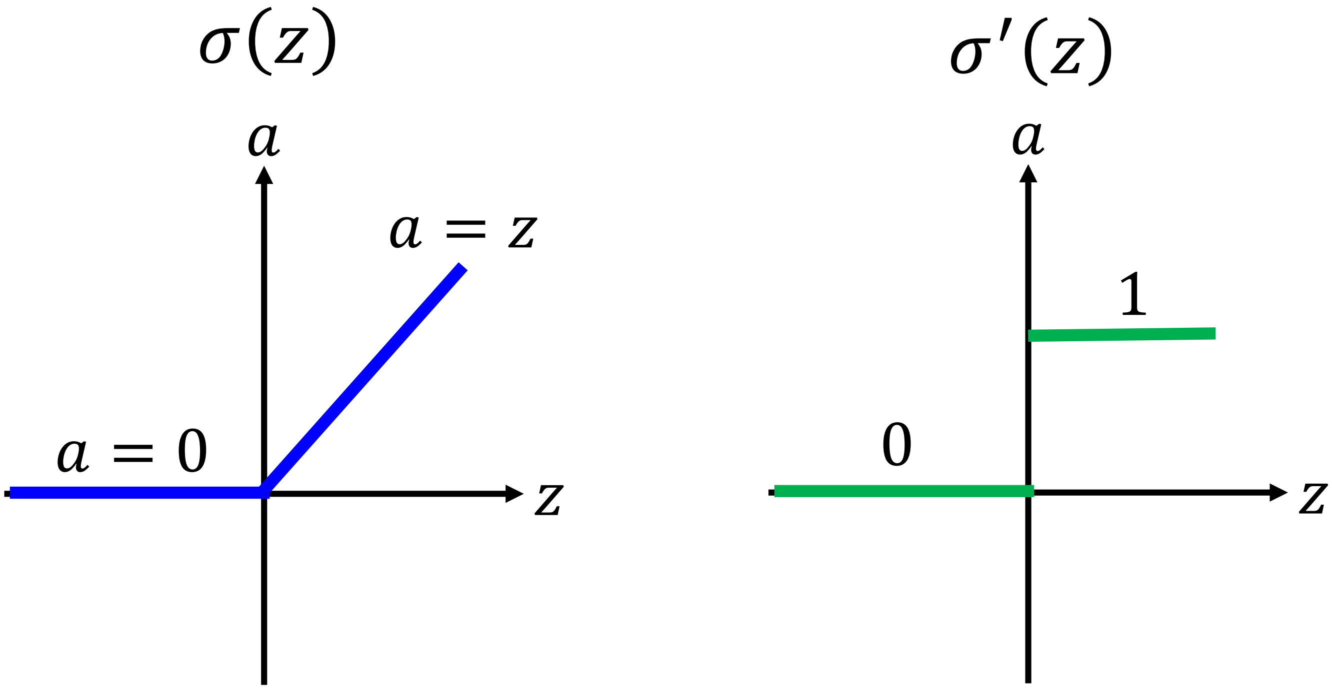
- Activation Function
- Cost Function
- Data Preprocessing
- Optimization
- Generalization

# Outline

- Activation Function
- Cost Function
- Data Preprocessing
- Optimization
- Generalization

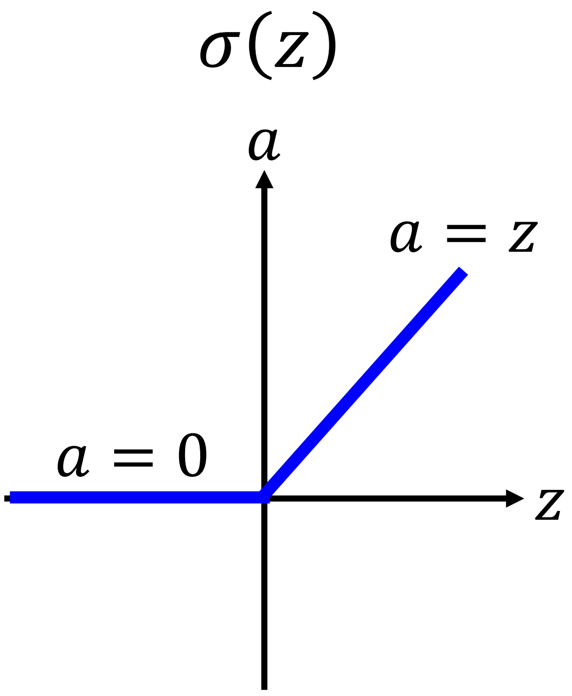
# ReLU

- Rectified Linear Unit (ReLU)



# ReLU

- Rectified Linear Unit (ReLU)



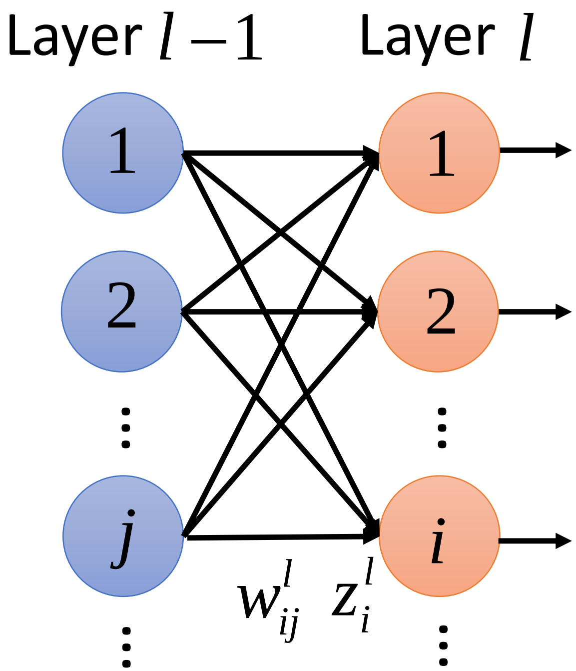

***Reason:***

1. Fast to compute
2. Biological reason
3. Infinite sigmoid with different biases
4. Vanishing gradient problem

# Review: Backpropagation

$$\frac{\partial C_x}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \frac{\partial C_x}{\partial z_i^l}$$

Error signal




$$\frac{\partial z_i^l}{\partial w_{ij}^l} = \begin{cases} a_j^{l-1} & l > 1 \\ x_j & l = 1 \end{cases}$$

$$\frac{\partial C_x}{\partial z_i^l} = \delta_i^l$$

***Forward Pass***

$$z^1 = W^1 x + b^1$$

$$a^1 = \sigma(z^1)$$

……

$$z^{l-1} = W^{l-1} a^{l-2} + b^{l-1}$$

$$a^{l-1} = \sigma(z^{l-1})$$

***Backward Pass***

$$\delta^L = \sigma'(z^L) \bullet \nabla C_x(y)$$

$$\delta^{L-1} = \sigma'(z^{L-1}) \bullet (W^L)^T \delta^L$$

……

$$\delta^l = \sigma'(z^l) \bullet (W^{l+1})^T \delta^{l+1}$$

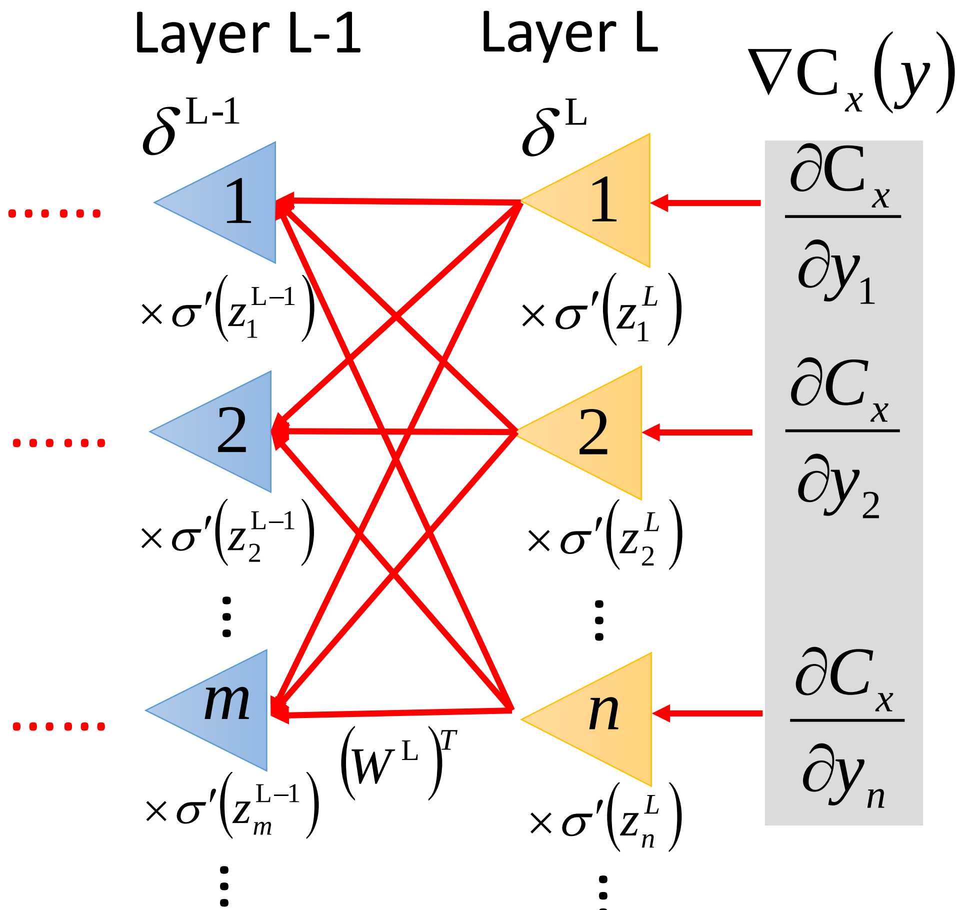
……

# Review: Backpropagation

$$\frac{\partial C_x}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \frac{\partial C_x}{\partial z_i^l}$$

Error signal

$$\delta_i^l$$




***Backward Pass***

$$\delta^L = \sigma'(z^L) \bullet \nabla C_x(y)$$

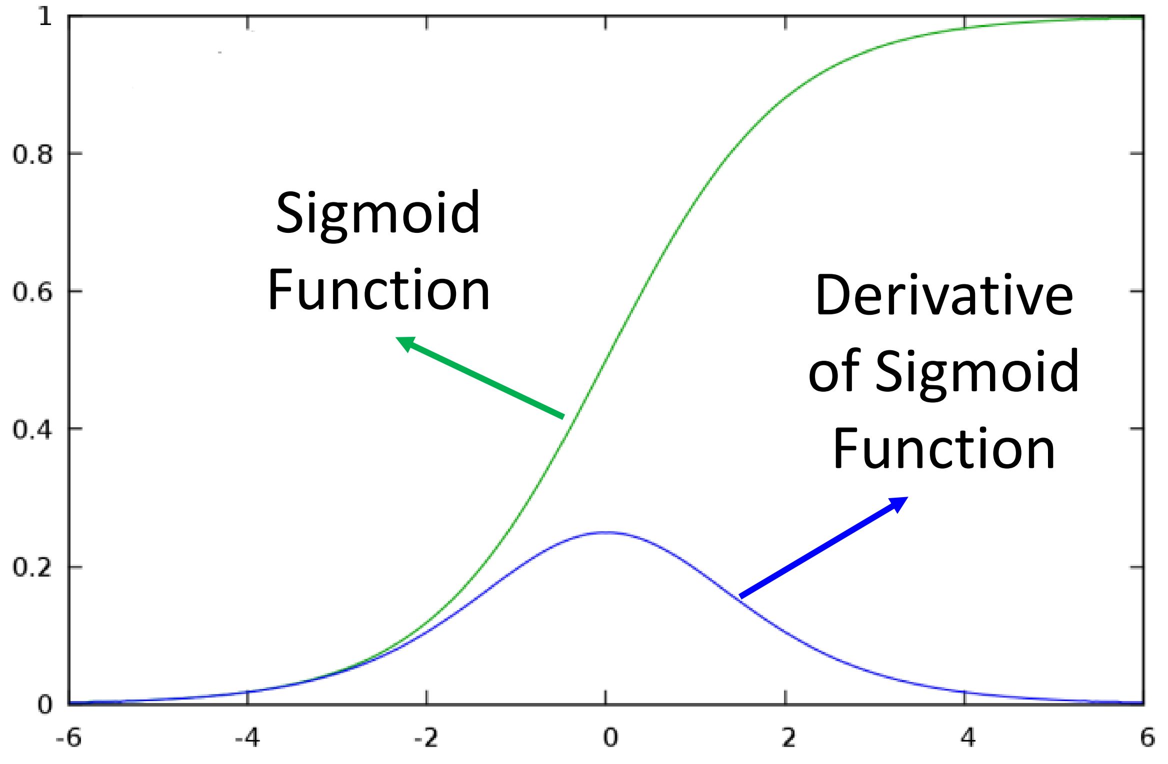
$$\delta^{L-1} = \sigma'(z^{L-1}) \bullet (W^L)^T \delta^L$$

......

$$\delta^l = \sigma'(z^l) \bullet (W^{l+1})^T \delta^{l+1}$$

......

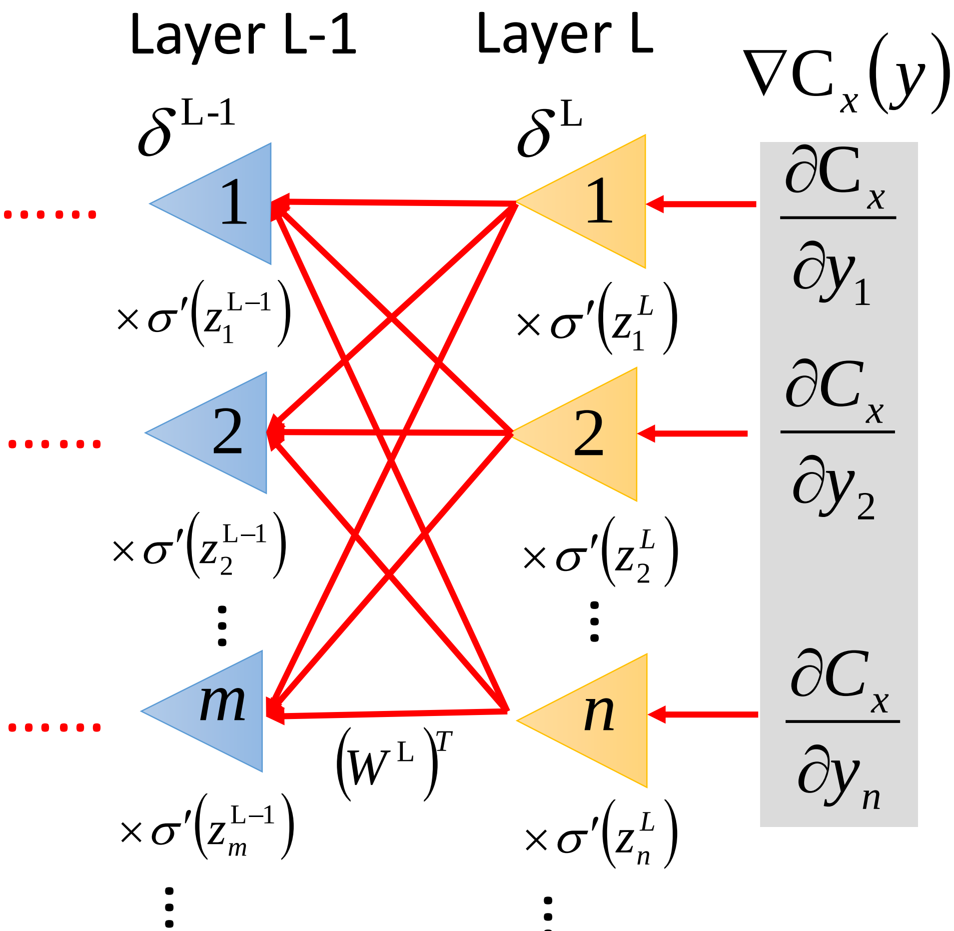
# Problem of Sigmoid




Derivative of Sigmoid Function is always smaller than 1

# Vanishing Gradient Problem

Backward Pass:

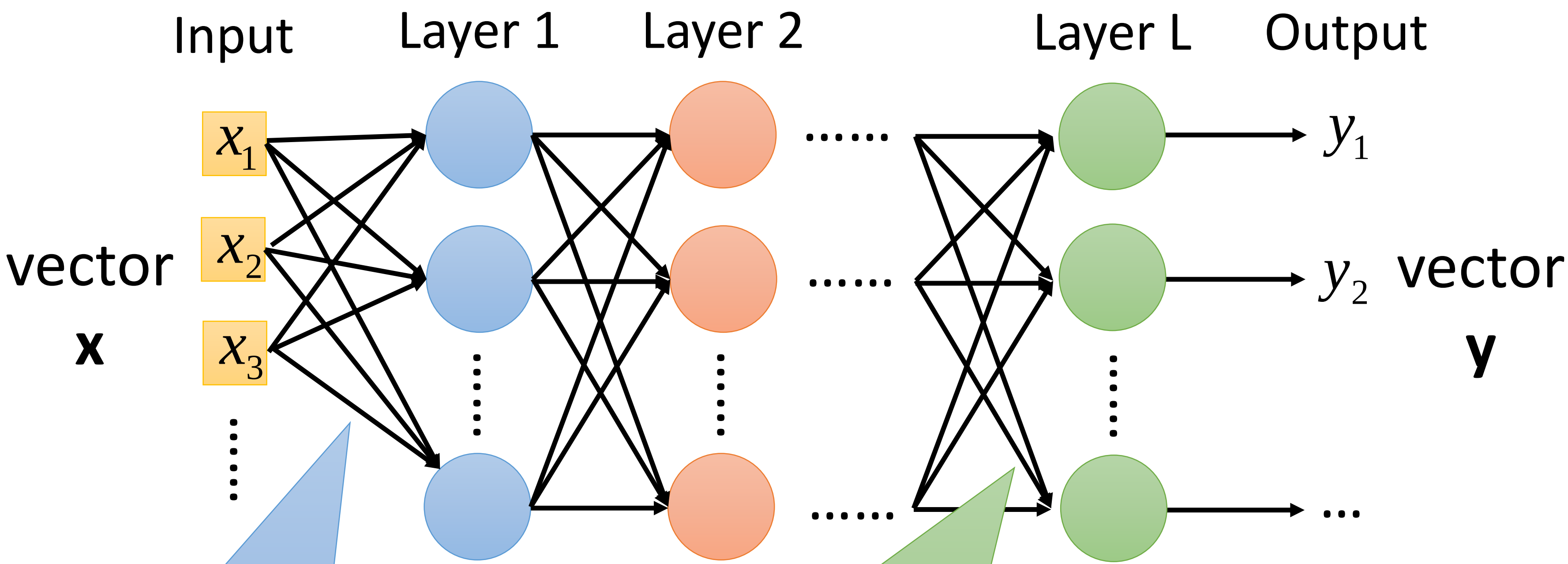



- For sigmoid function, $\sigma'(z)$ always smaller than 1
- Error signal is getting smaller and smaller

Gradient is smaller

$$\frac{\partial C_x}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \boxed{\frac{\partial C_x}{\partial z_i^l}} \rightarrow \boxed{\delta_i^l}$$

# Vanishing Gradient Problem




Learn very slowly

Still random

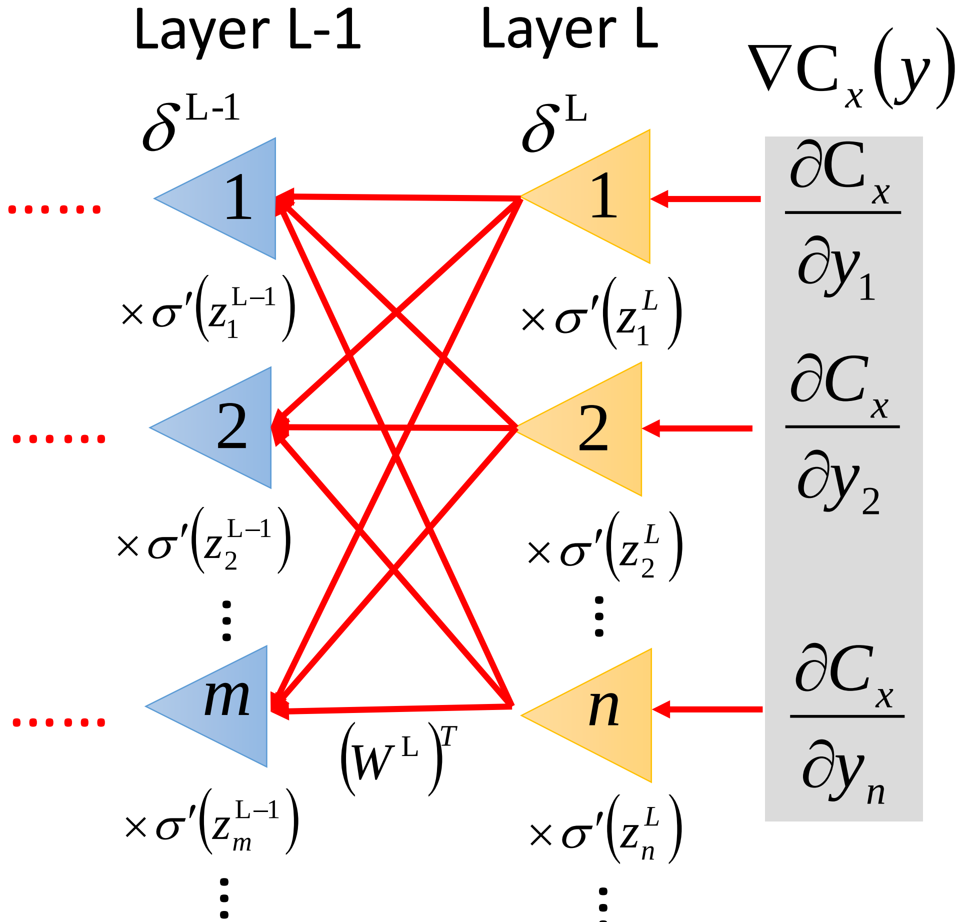
Learn faster

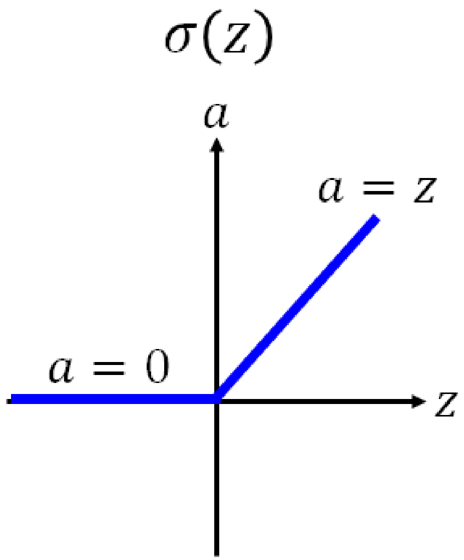
Already converge

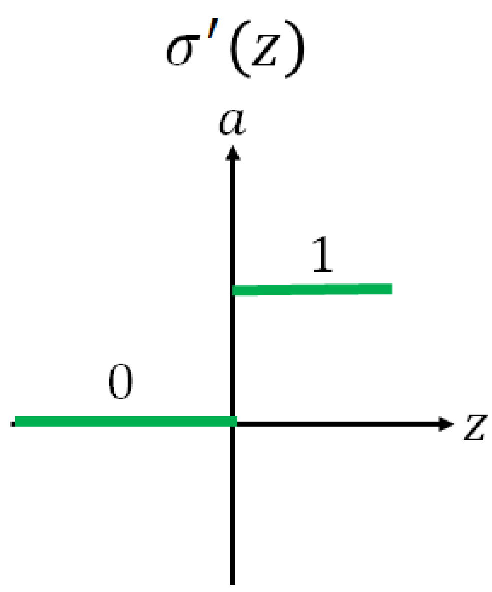
The weights are converged based on random!?

# ReLU

Backward Pass:

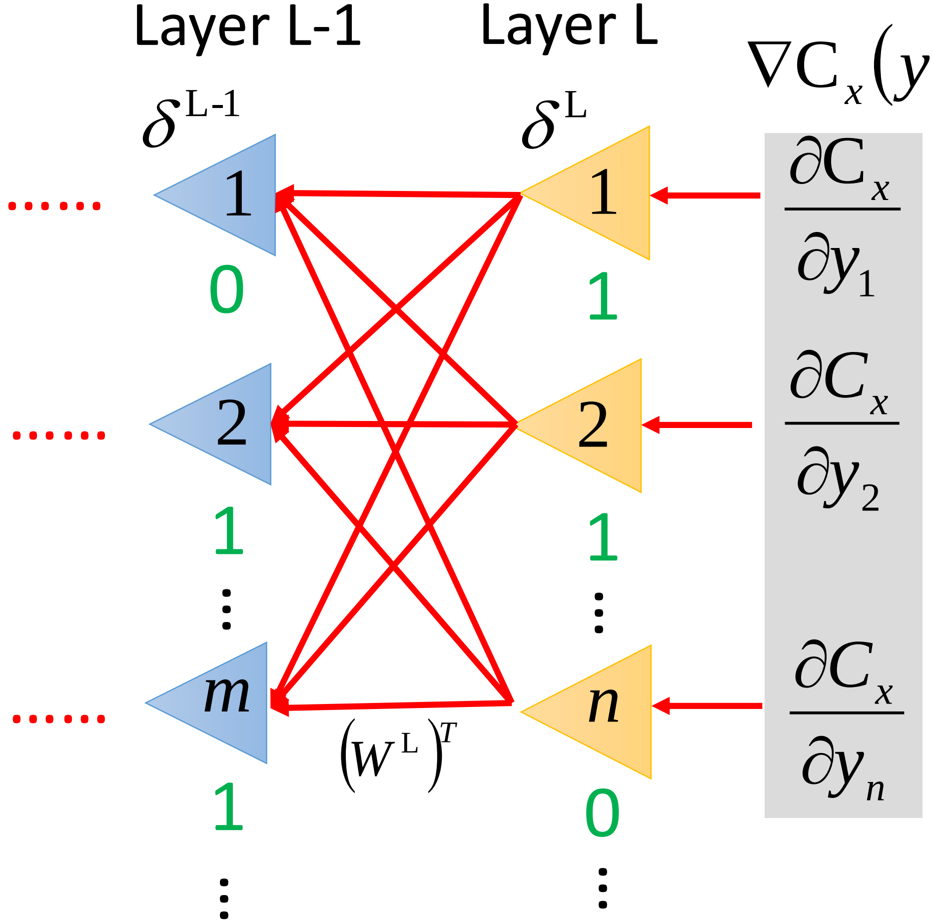


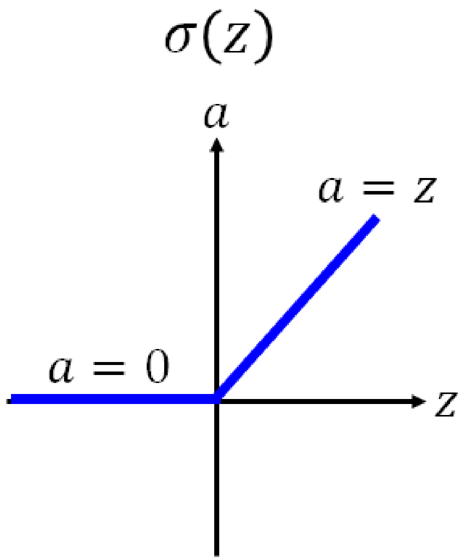



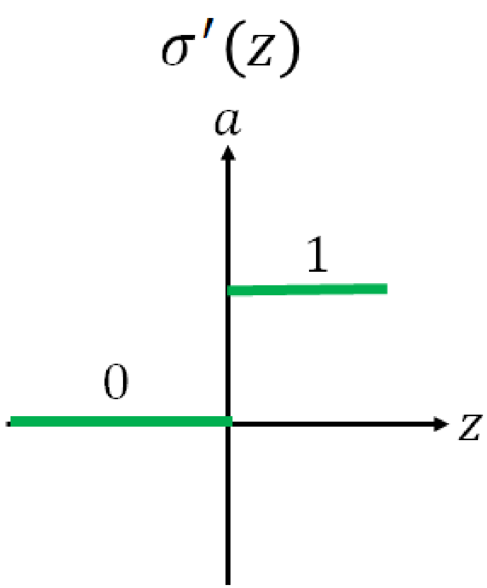



# ReLU

Backward Pass:

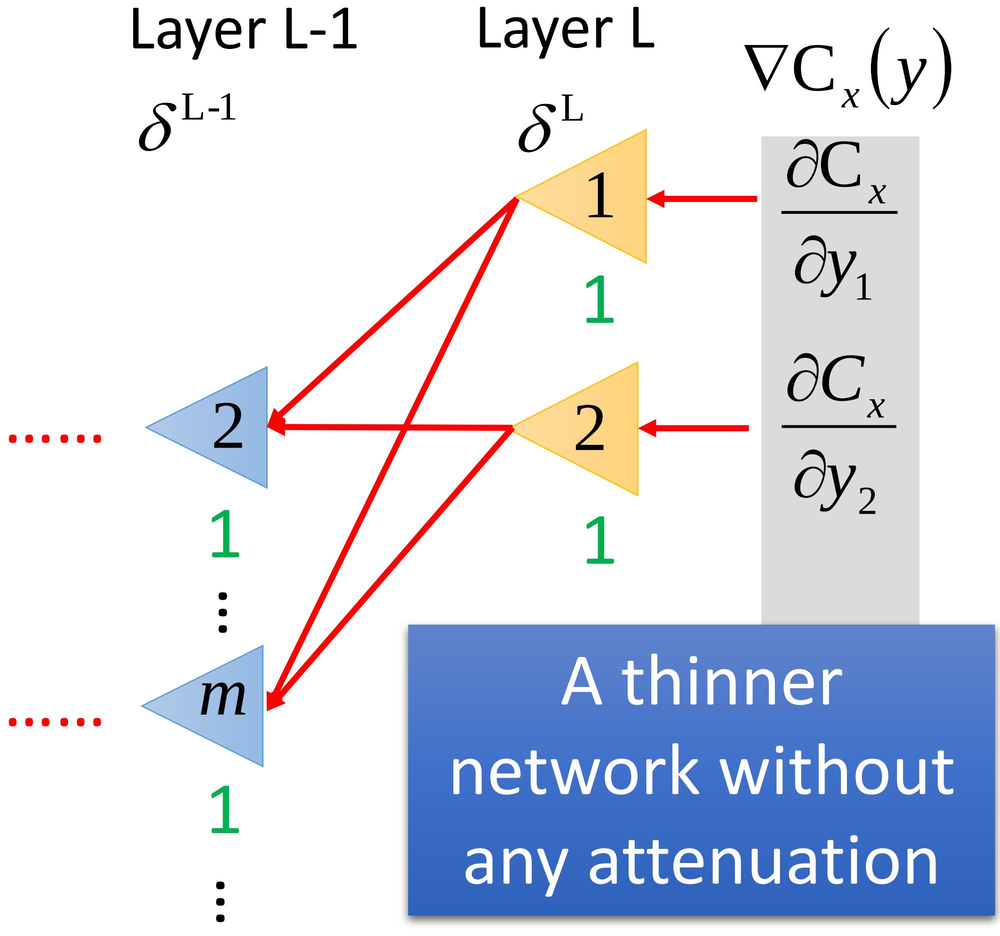


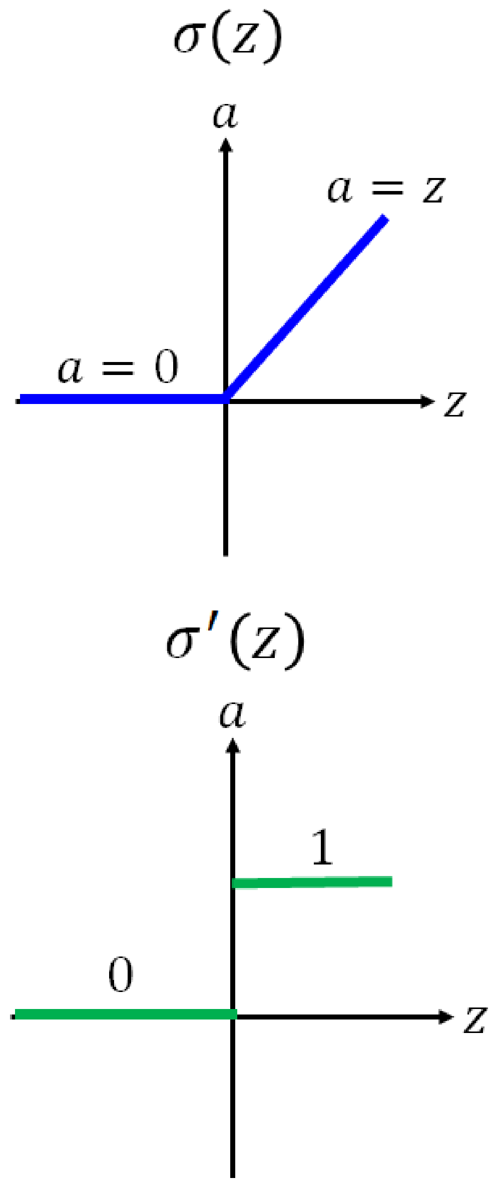



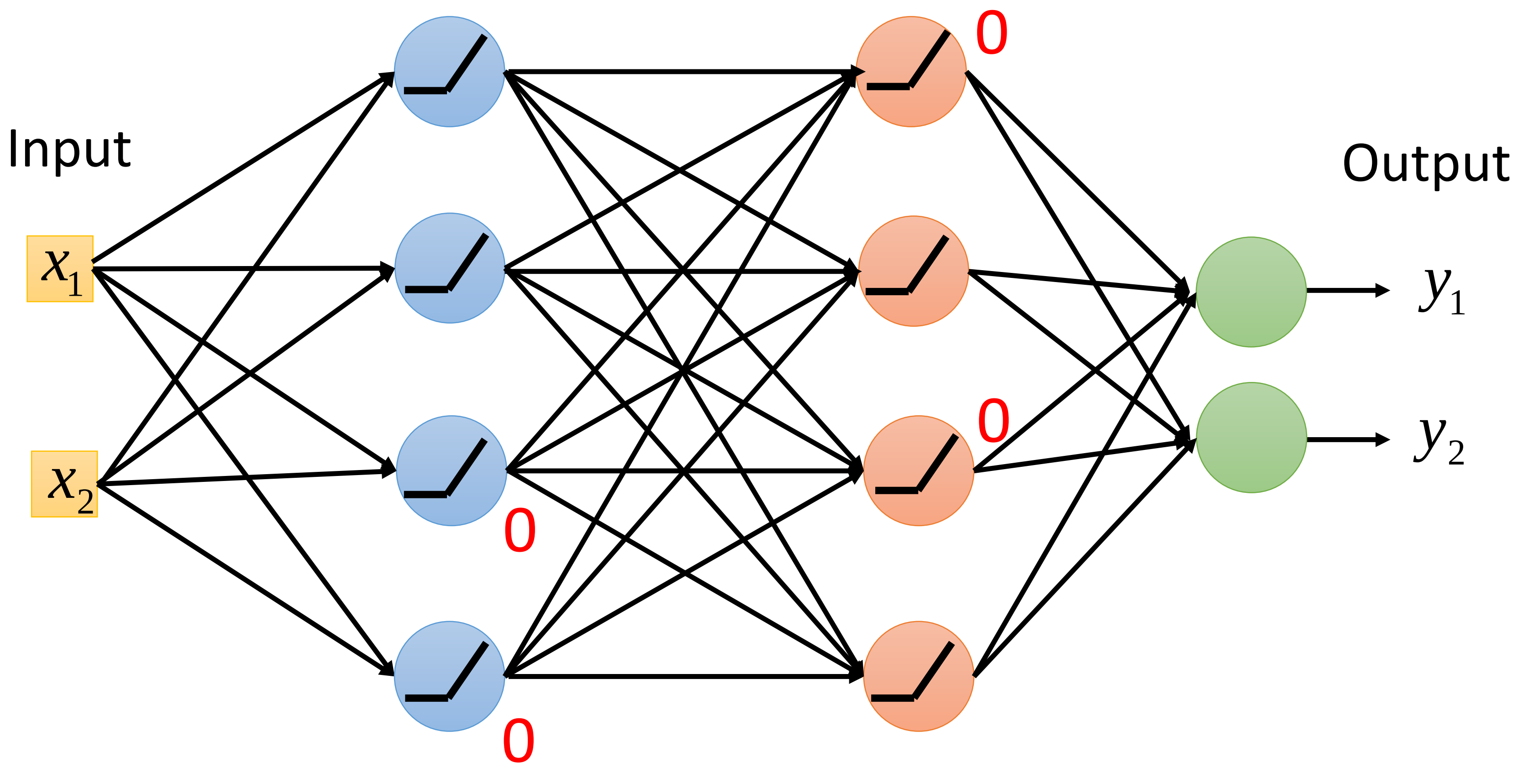
# ReLU

Backward Pass:

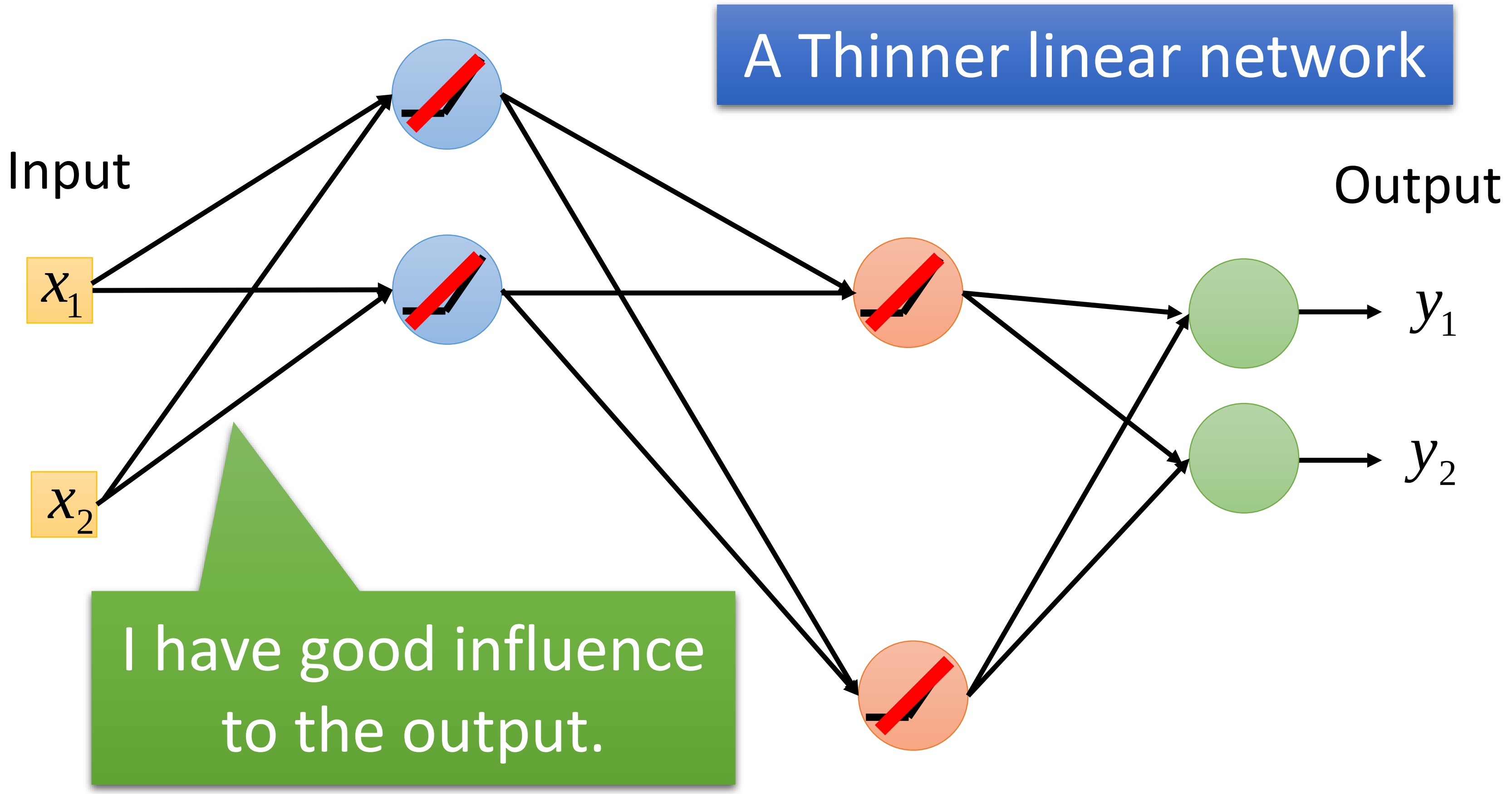


# ReLU

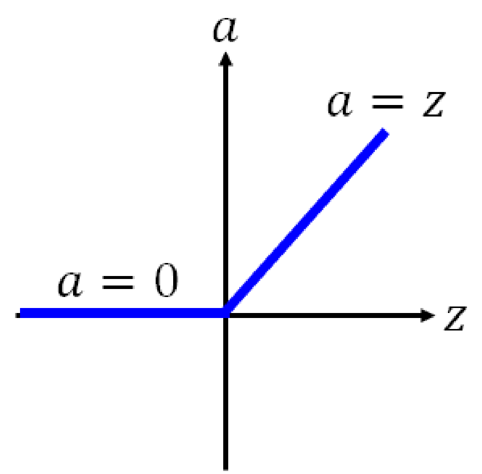


# ReLU


A Thinner linear network
Input
$x_1$
$x_2$
Output
$y_1$
$y_2$
I have good influence to the output.

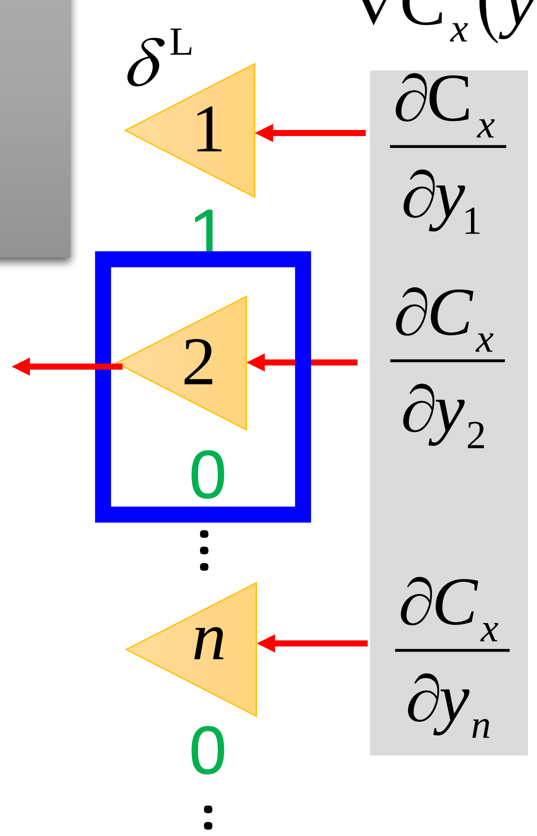
# ReLU

$$\frac{\partial C_x}{\partial w_{nj}^L} = \frac{\partial z_n^L}{\partial w_{nj}^L} \frac{\partial C_x}{\partial z_n^L}$$



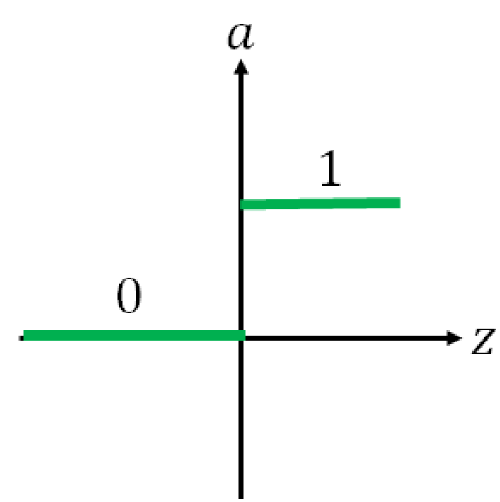

Backward Pass:

All the weights connected to this neuron will not update.




$$\nabla C_x(y) = \left[ \frac{\partial C_x}{\partial y_1}, \frac{\partial C_x}{\partial y_2}, \ldots, \frac{\partial C_x}{\partial y_n} \right]$$

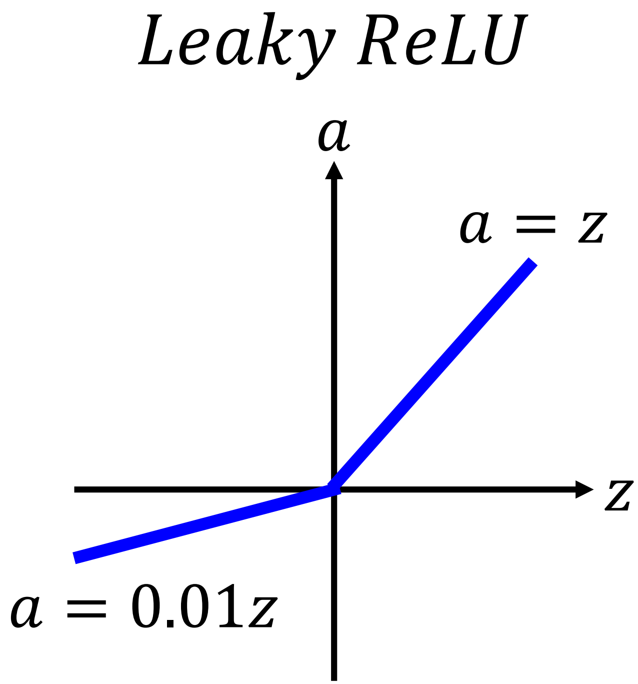
$$\delta_n^L = \frac{\partial C_x}{\partial z_n^L} = 0$$

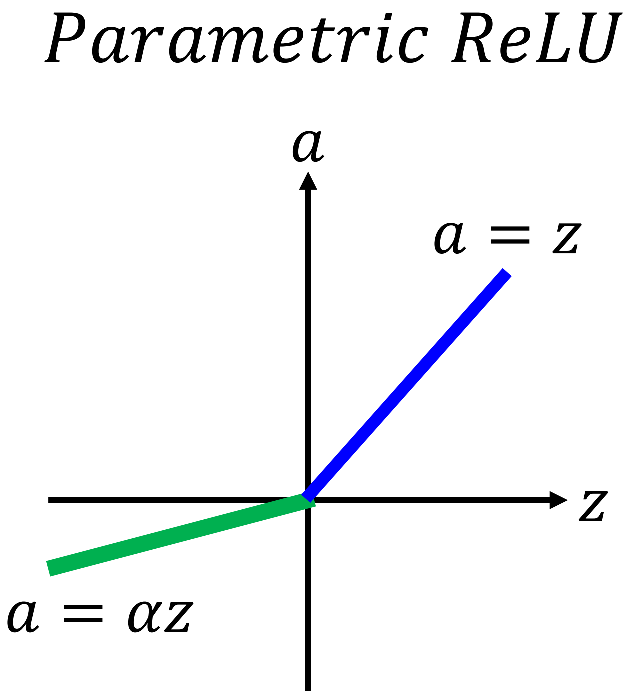



***Possible solution:***

1. softplus
2. Initialize with large bias

# ReLU - variant

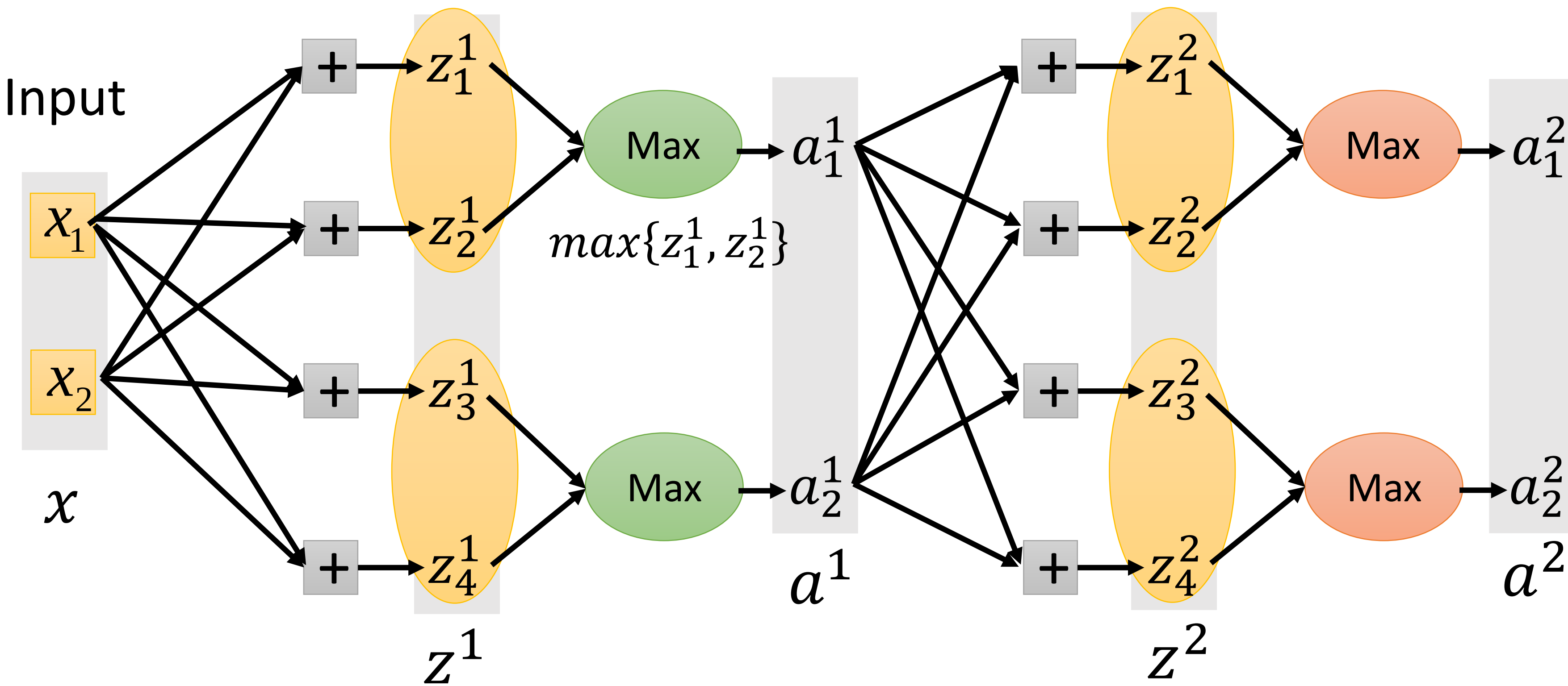



α also learned by gradient descent

# Maxout

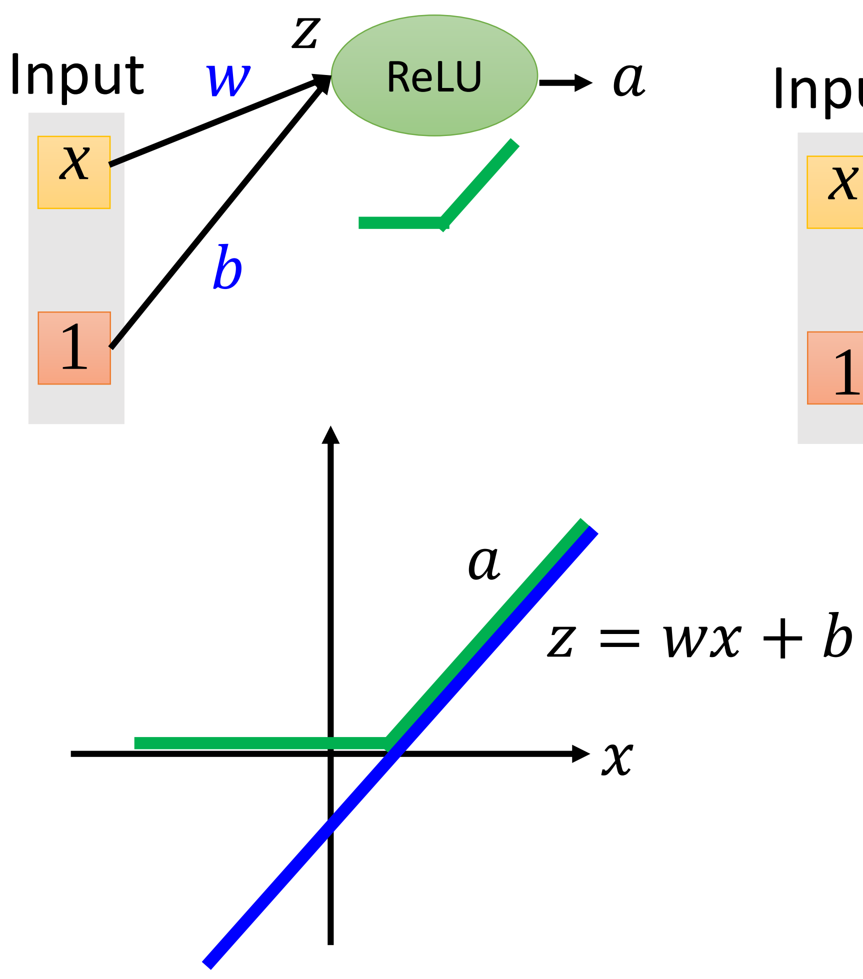
- All ReLU variants are just special cases of Maxout

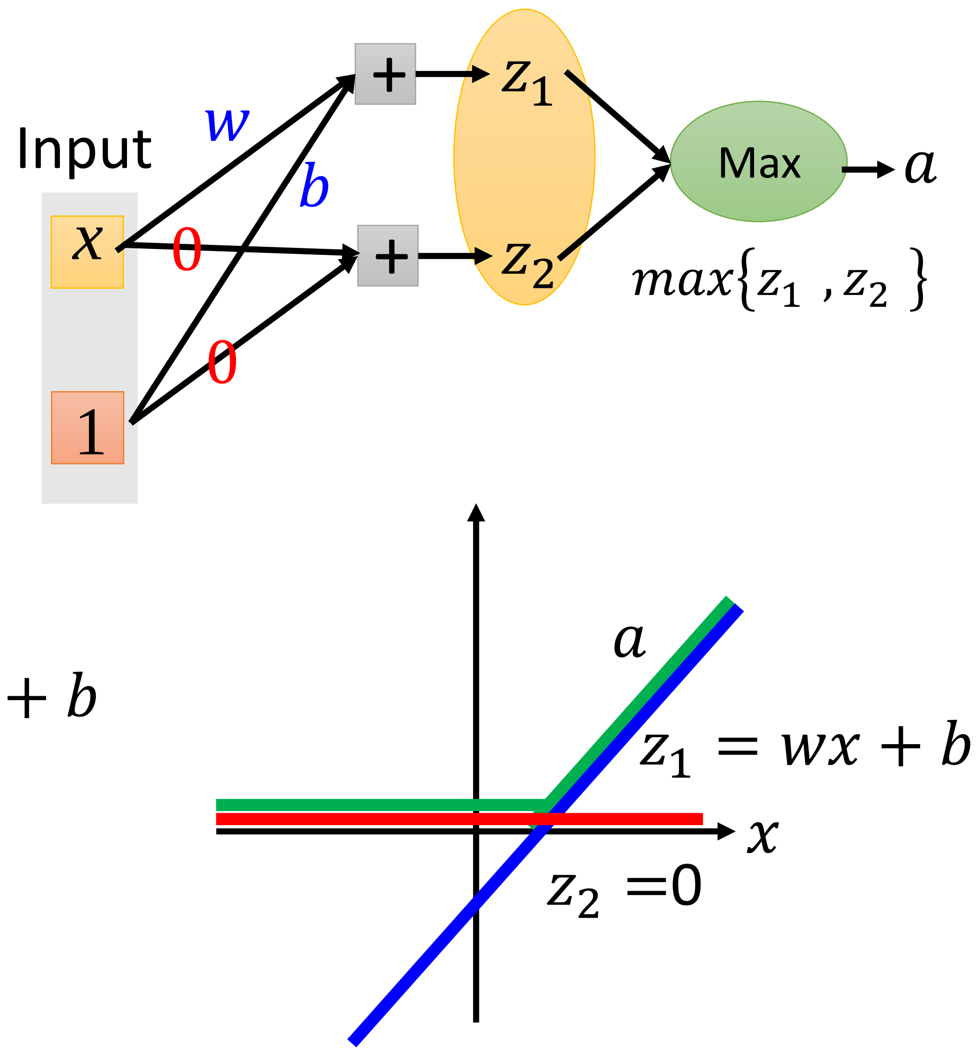



$z^1 = W^1 x$

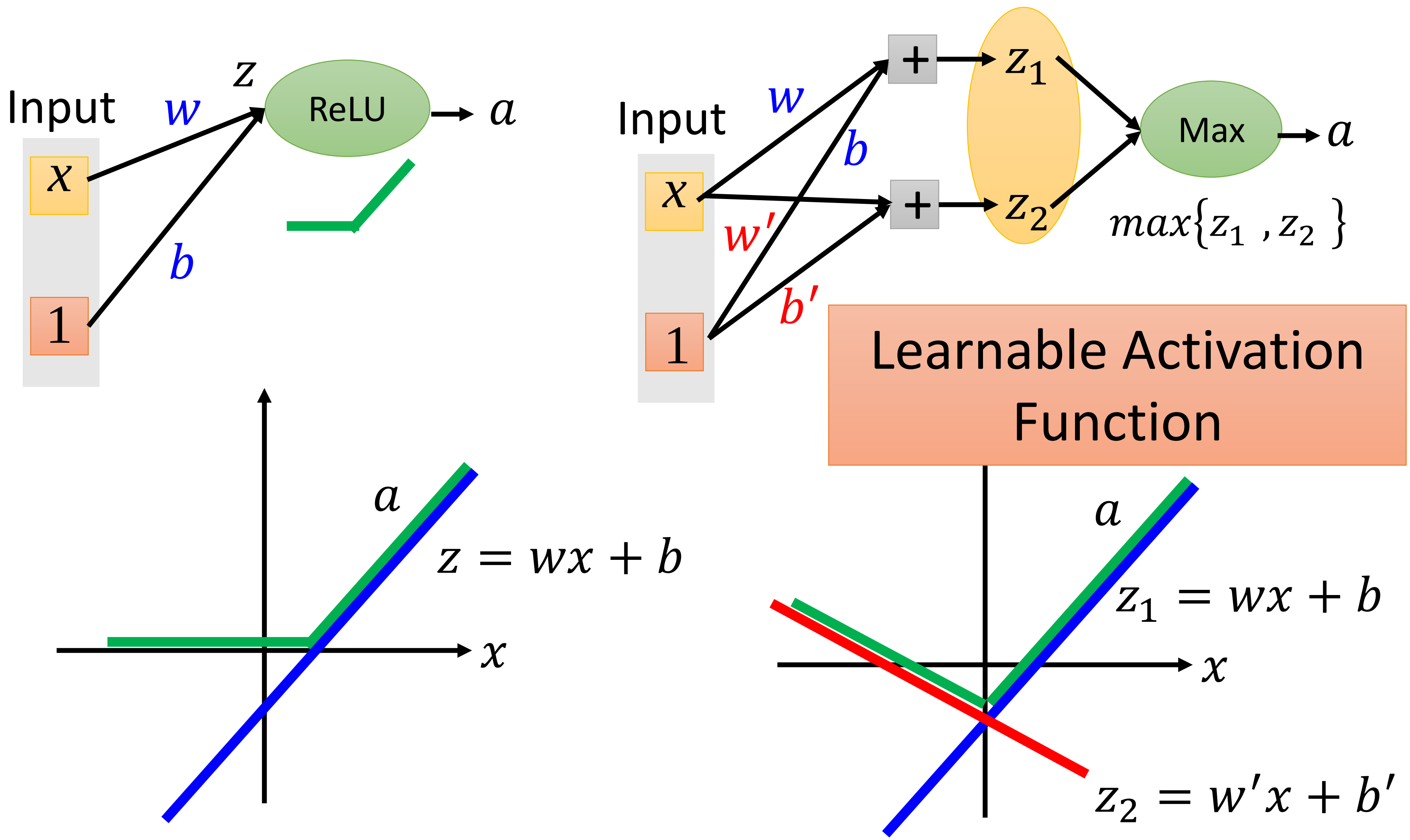
$z^2 = W^2 a^1$

# Maxout – ReLU is special case



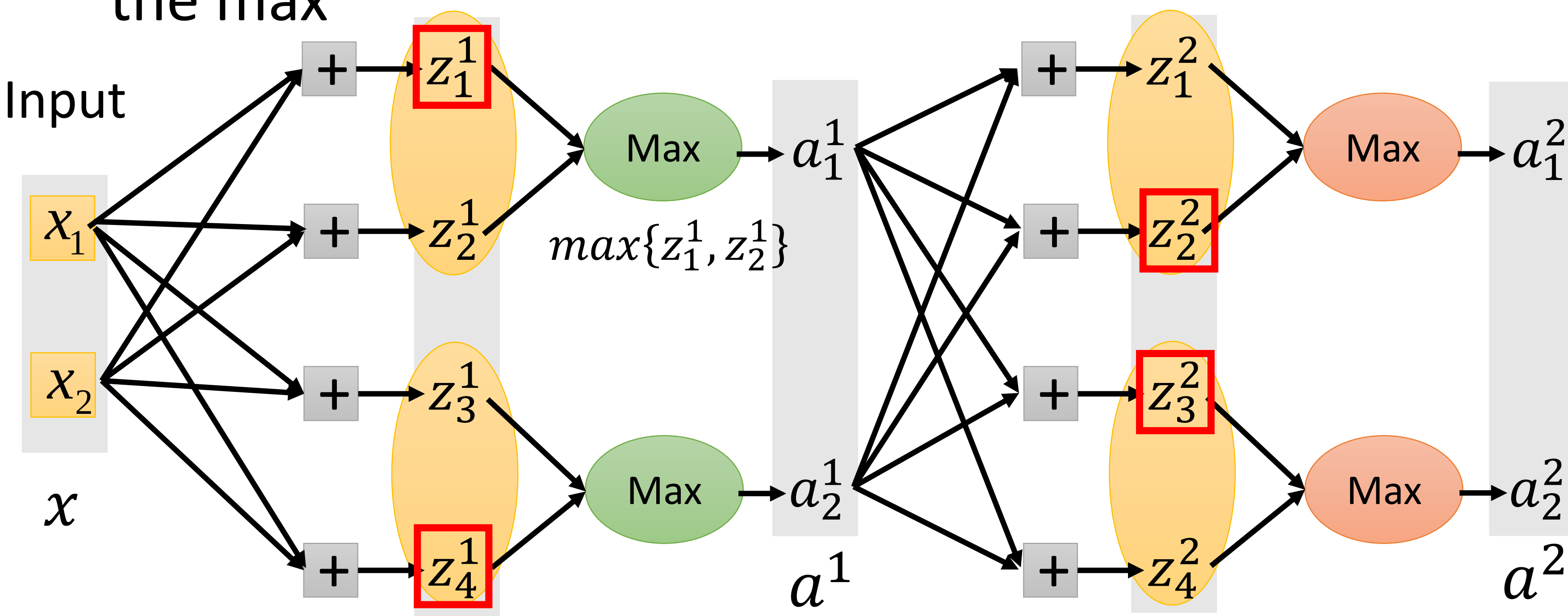



# Maxout – ReLU is special case



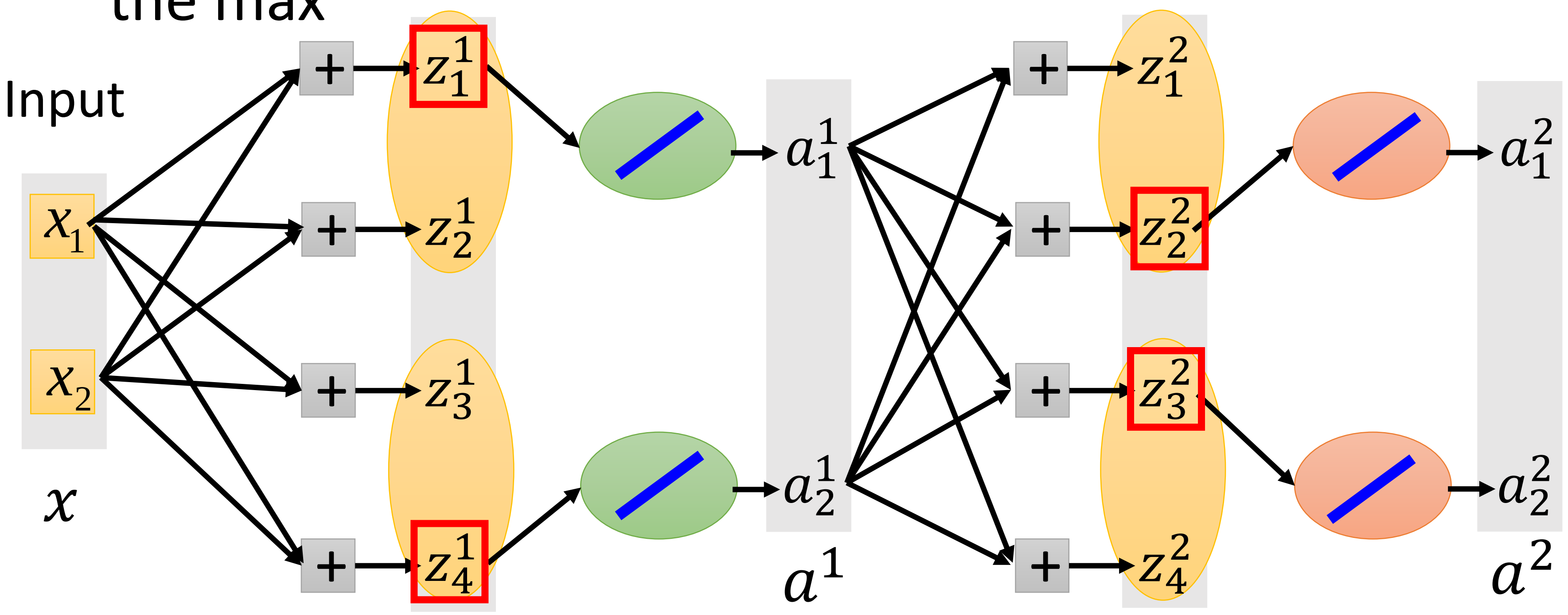
# Maxout - Training

- Given a training data x, we know which z would be the max



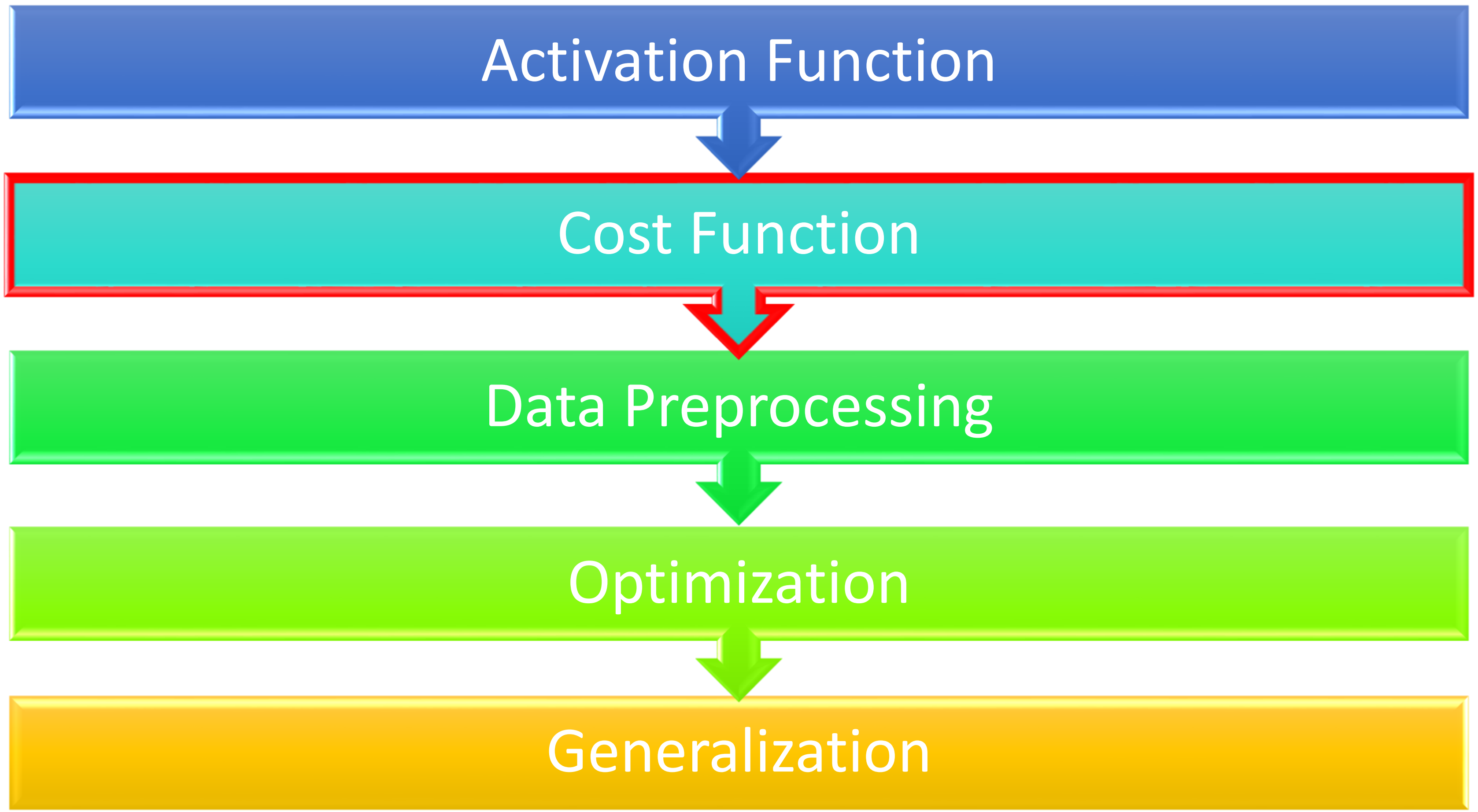
# Maxout - Training

- Given a training data x, we know which z would be the max



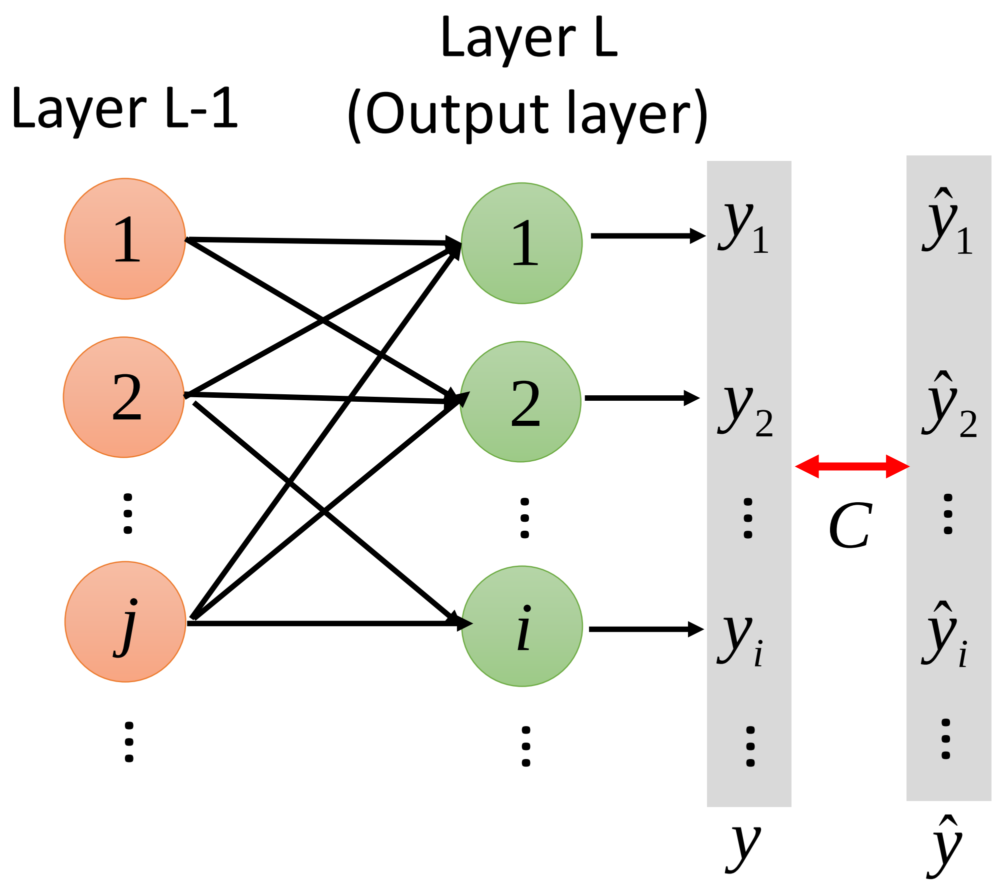

- Train this thin and linear netowrk

# Outline



Activation Function
Cost Function
Data Preprocessing
Optimization
Generalization

# Cost Function




$$C = \frac{1}{2}\|y - \hat{y}\|^2$$

$$= \frac{1}{2}\sum_n \left(y_n - \hat{y}_n\right)^2$$

# Output Layer

$$\hat{y} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \end{bmatrix}$$

More similar?

ReLU $y = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1.2 \\ \vdots \end{bmatrix}$

ReLU $y = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 2 \\ \vdots \end{bmatrix}$

- Larger output means larger confidence

Better?

***Classification Task:***

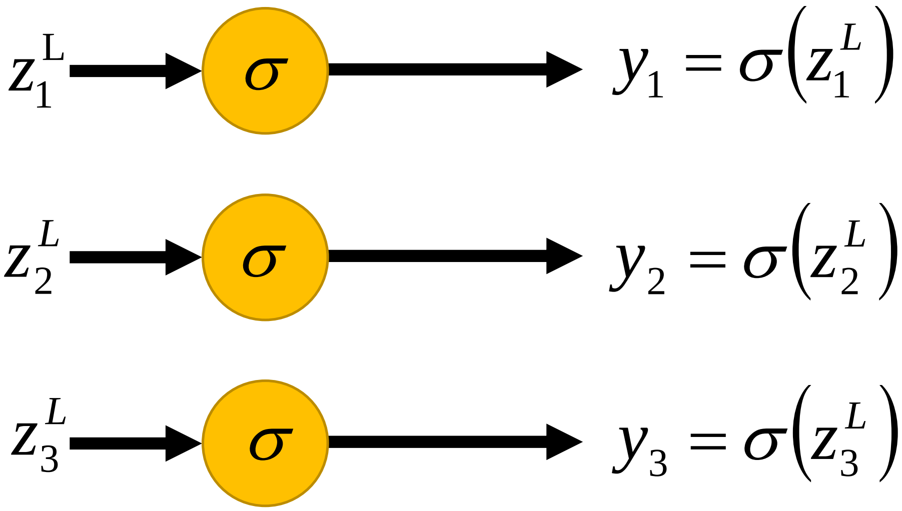
Only one dimension is 1, and others are all 0

It is better to let the output bounded.

# Softmax

- Softmax layer as the output layer

***<u>Ordinary Output layer</u>***



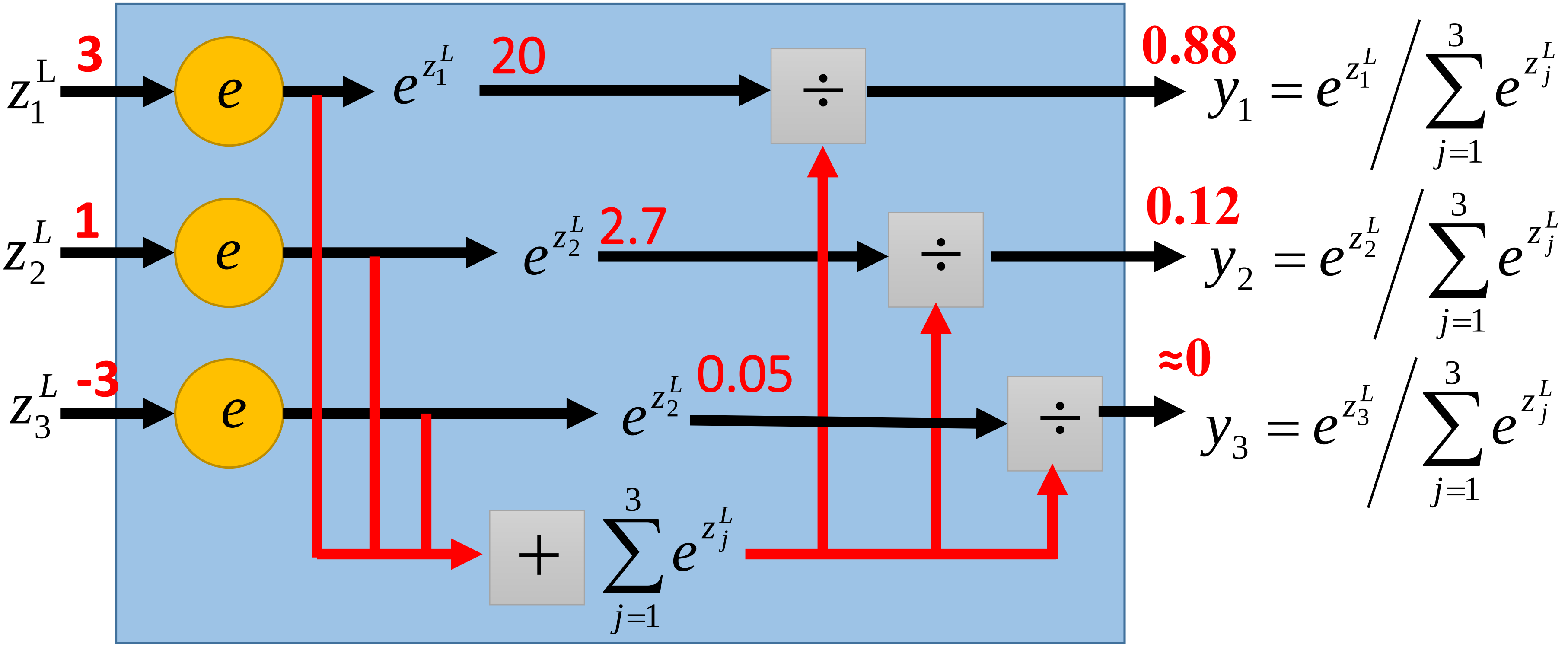
# Softmax

- Softmax layer as the output layer

***Probability***:

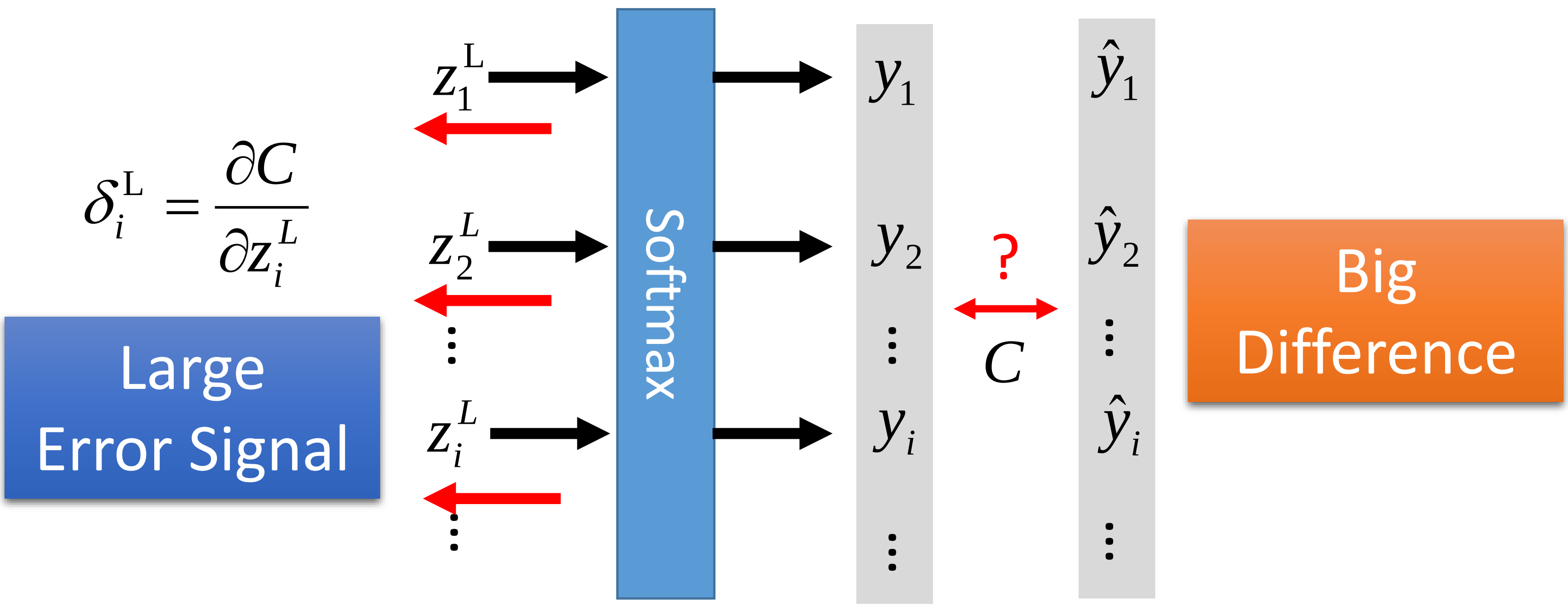
- $1 > y_i > 0$
- $\sum_i y_i = 1$

***Softmax Layer***



# Softmax

- What kind of cost function should we used for softmax layer output?



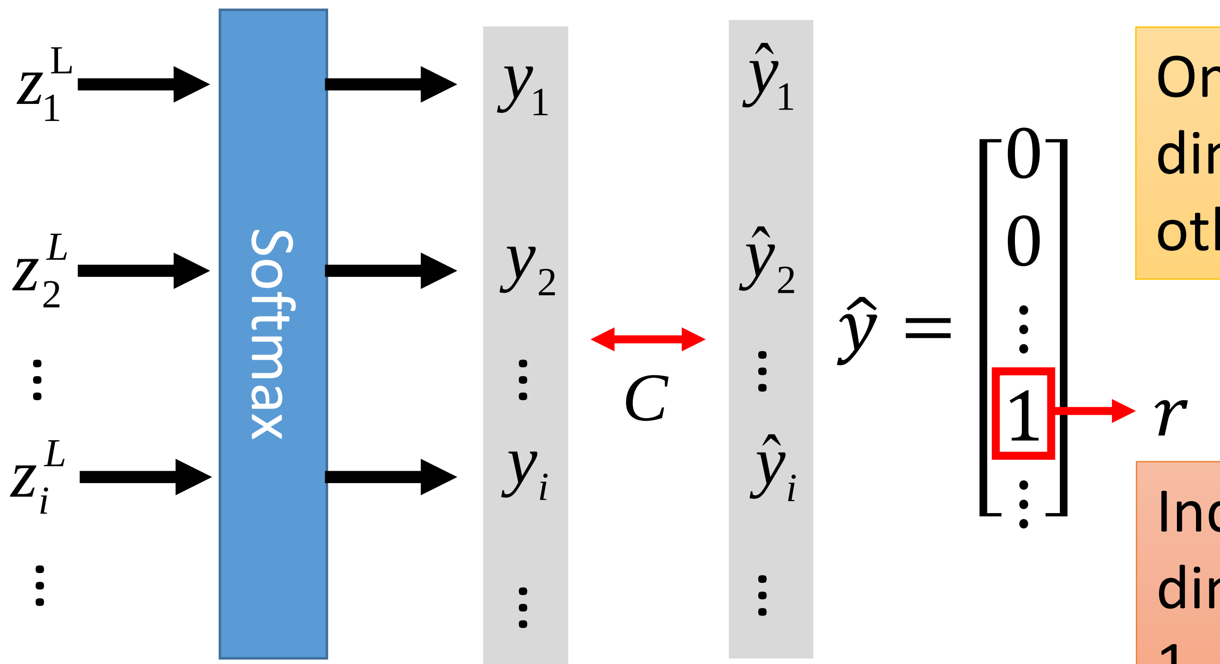
# Softmax

Define cost: $C = -log y_r$

Cross Entropy

$$y_i = \frac{e^{z_i^L}}{\sum_j e^{z_j^L}}$$

**Do we have to consider other dimensions?**



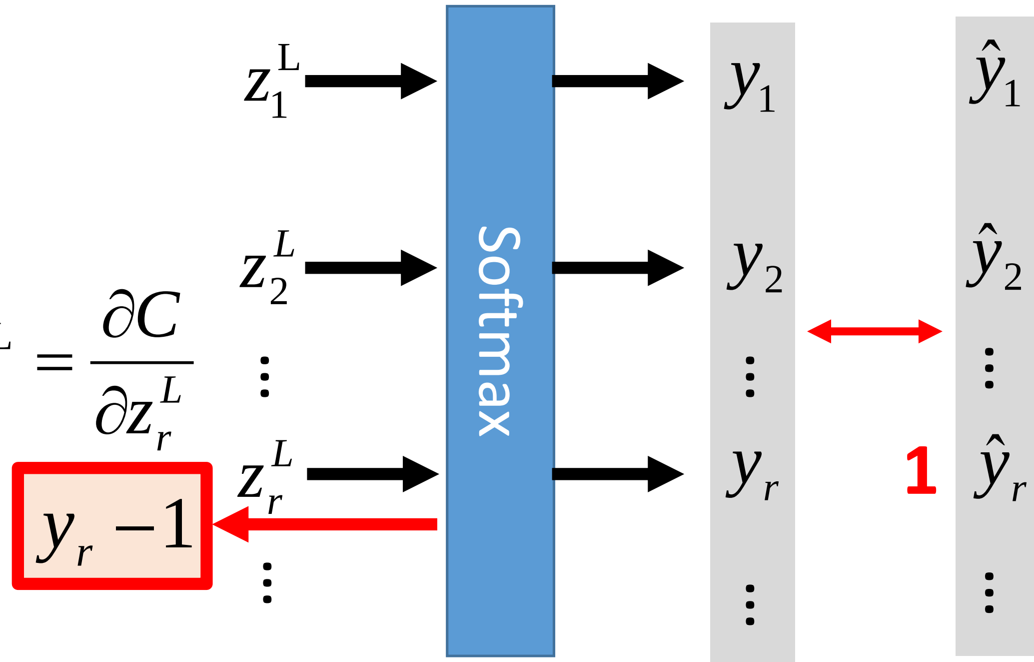

Only one dimension is 1, and others are all 0

Index of the dimension which is 1

$$y_i = \frac{e^{z_i^L}}{\sum_j e^{z_j^L}}$$

$$C = -\log y_r$$

$$\delta_r^L = \frac{\partial C}{\partial z_r^L}$$




$$\frac{\partial C}{\partial y_r} \frac{\partial y_r}{\partial z_r^L}$$

$$\delta_r^L = \frac{\partial C}{\partial z_r^L} = -\frac{1}{y_r} \frac{\partial y_r}{\partial z_r^L} = -\frac{1}{y_r}\left(y_r - y_r^2\right) = y_r - 1$$

The absolute value of $\delta_r^L$ is larger when $y_r$ is far from 1

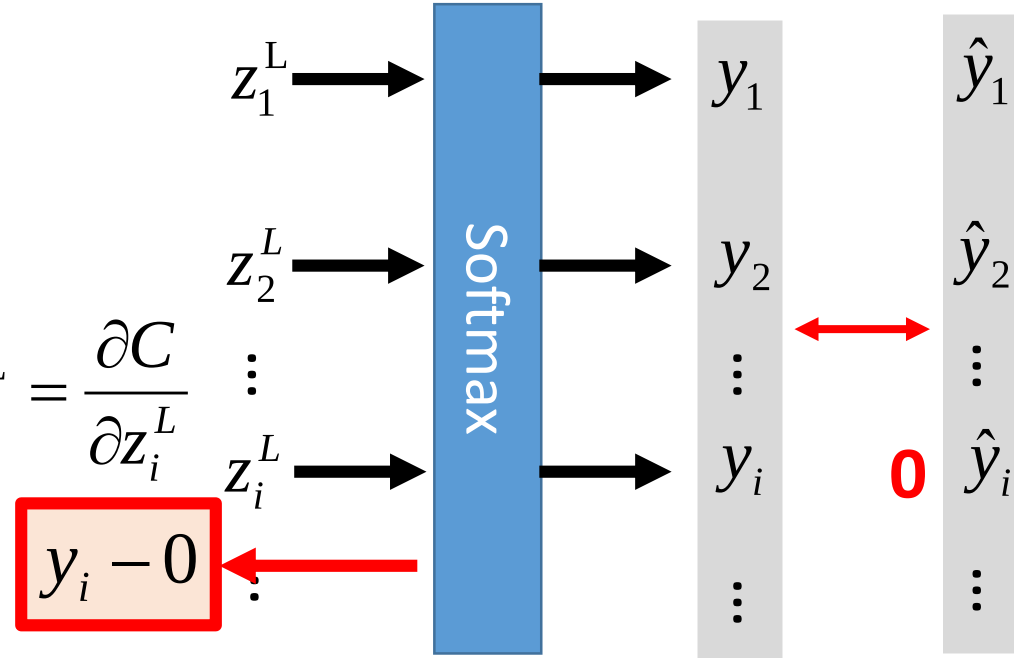
$$y_r = \frac{e^{z_r^L}}{\sum_j e^{z_j^L}}$$

$z_r^L$ appears in both numerator and denominator

$$y_i = \frac{e^{z_i^L}}{\sum_j e^{z_j^L}}$$

$$C = -\log y_r$$

$$\delta_i^L = \frac{\partial C}{\partial z_i^L}$$




$$\frac{\partial C}{\partial y_r} \frac{\partial y_r}{\partial z_i^L}$$

$i \neq r$

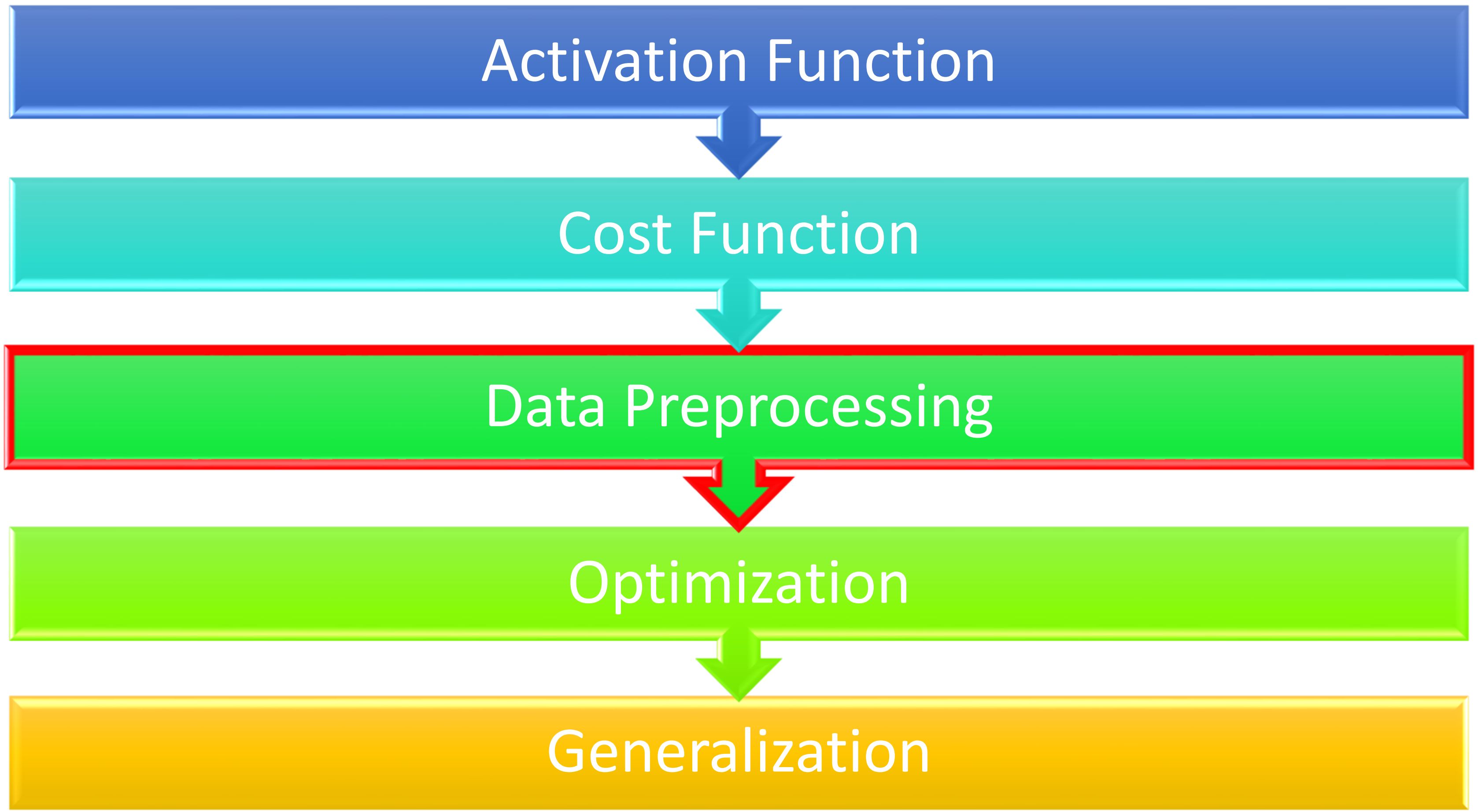
$$\delta_i^L = \frac{\partial C}{\partial z_i^L} = -\frac{1}{y_r} \frac{\partial y_r}{\partial z_i^L} = -\frac{1}{y_r}\left(-y_r y_i\right) = y_i$$

The absolute value of $\delta_i^L$ is larger when $y_i$ is larger

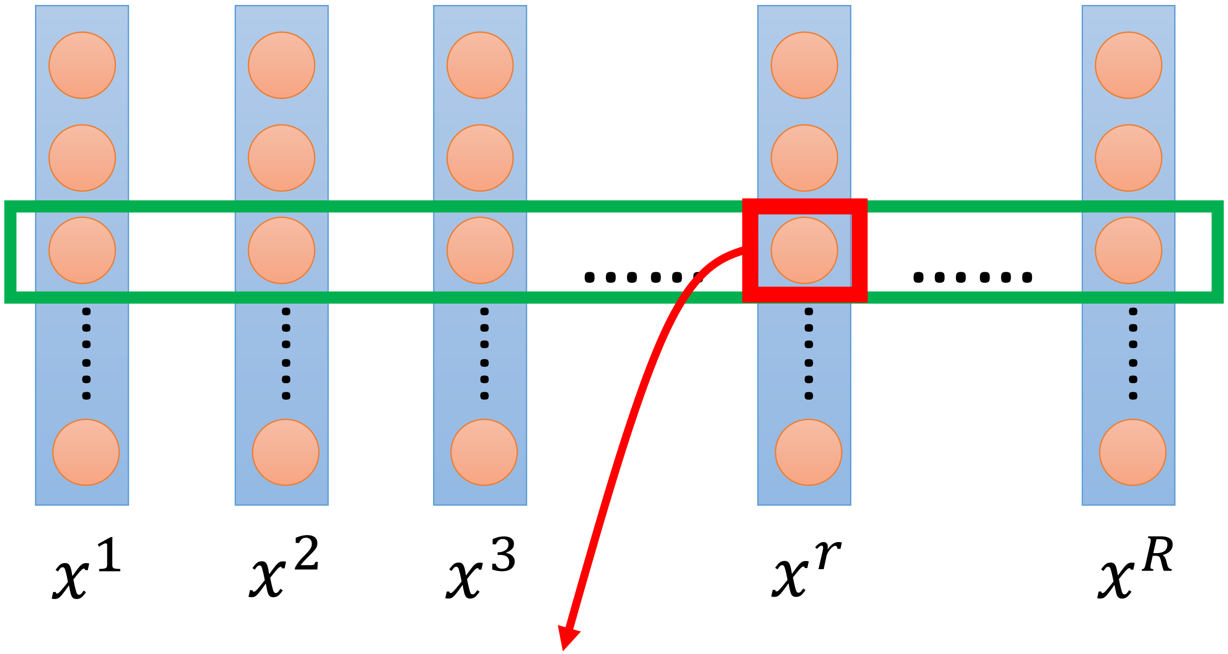
$$y_r = \frac{e^{z_r^L}}{\sum_j e^{z_j^L}}$$

$z_i^L$ appears only in denominator

# Outline


Activation Function
Cost Function
Data Preprocessing
Optimization
Generalization

# Normalizing Input




For each dimension i:

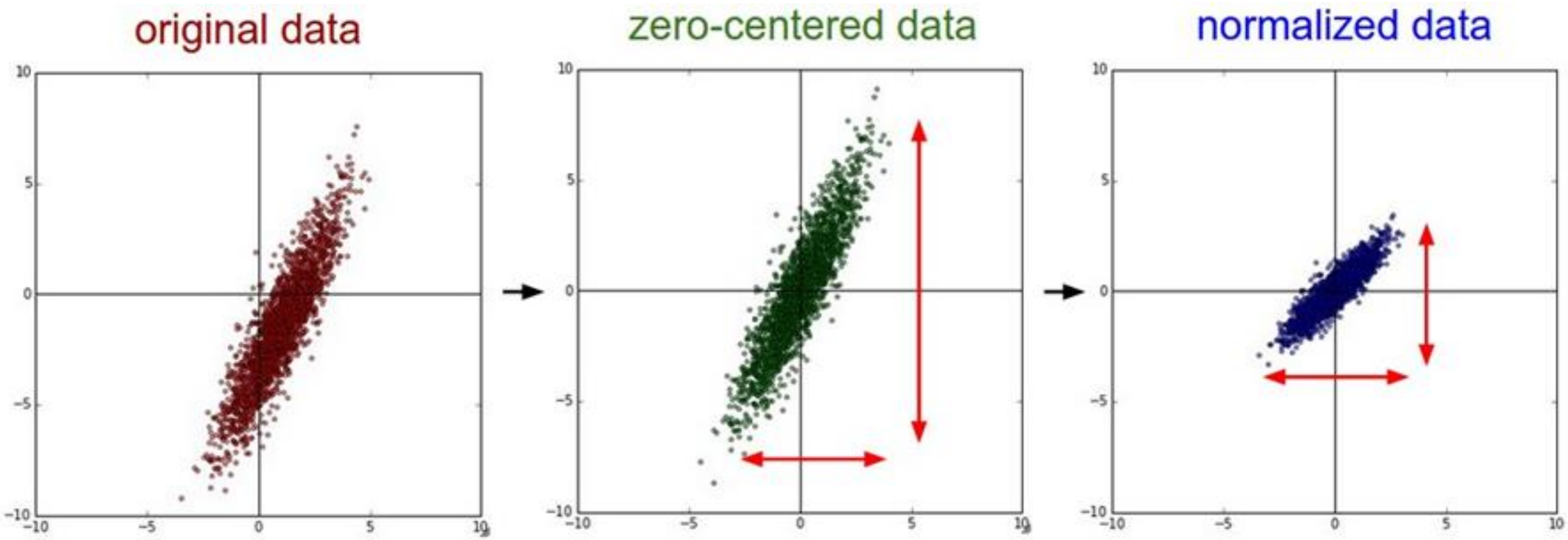
mean: $m_i$

standard deviation: $\sigma_i$

$$x_i^r \leftarrow \frac{x_i^r - m_i}{\sigma_i}$$

The means of all dimensions are 0, and the variances are all 1

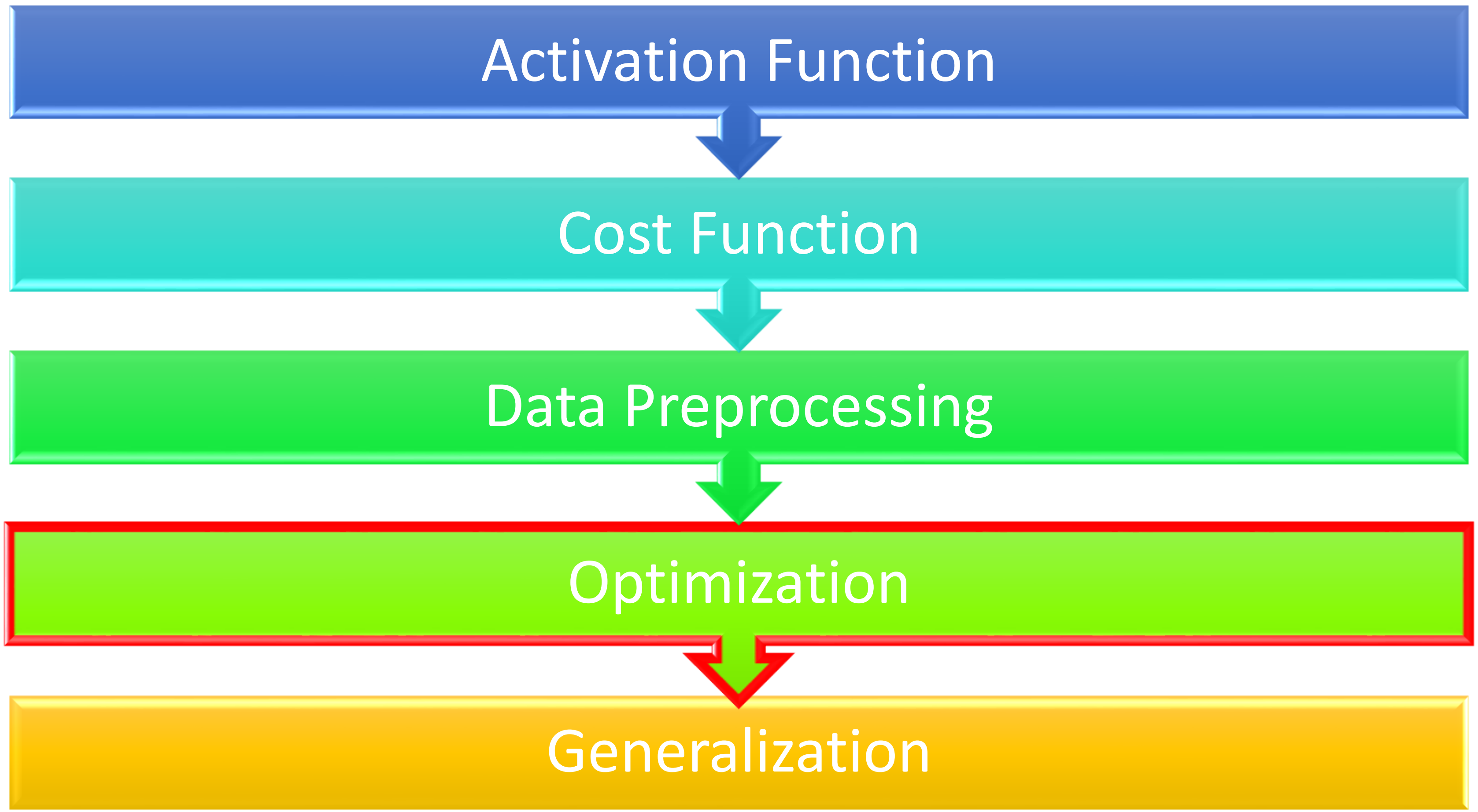
# Normalizing Input



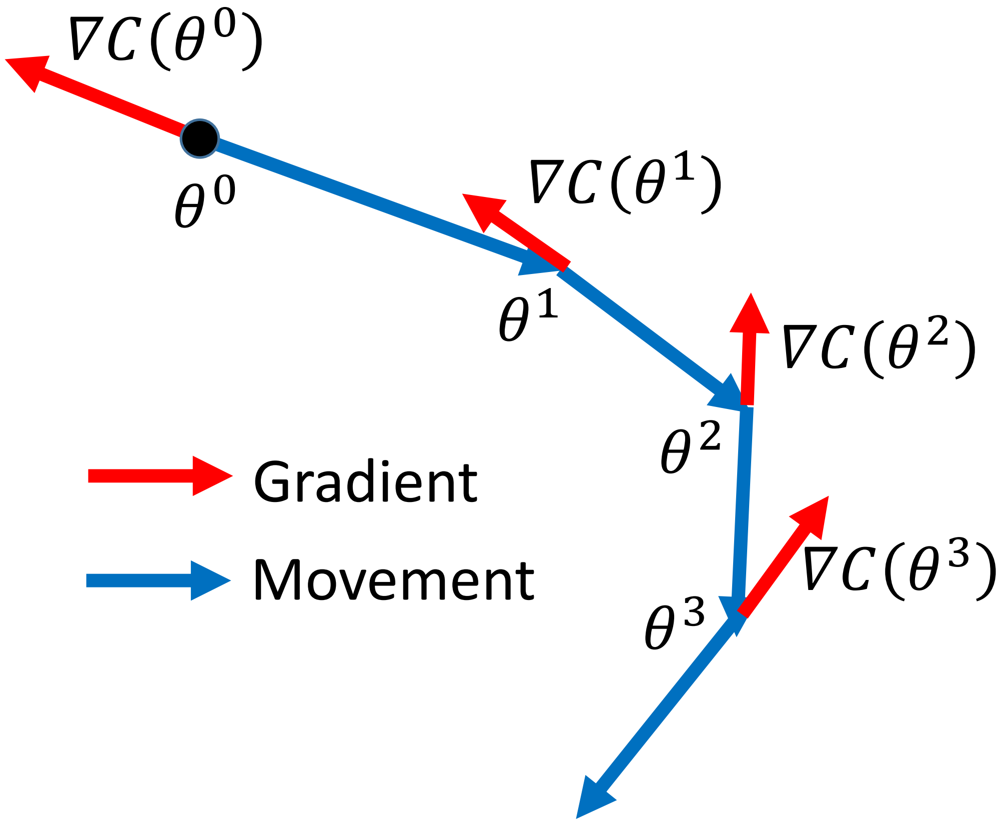

Source of figure: http://cs231n.github.io/neural-networks-2/

Normalizing your training and testing data in the same way.

# Outline


Activation Function
Cost Function
Data Preprocessing
Optimization
Generalization

# Vanilla Gradient Descent




Start at position $\theta^0$

Compute gradient at $\theta^0$

Move to $\theta^1 = \theta^0 - \eta \nabla C(\theta^0)$

Compute gradient at $\theta^1$

Move to $\theta^2 = \theta^1 - \eta \nabla C(\theta^1)$

⋮

Stop until $\nabla C(\theta^t) \approx 0$

1. How to determine the learning rates
2. Stuck at local minima or saddle points

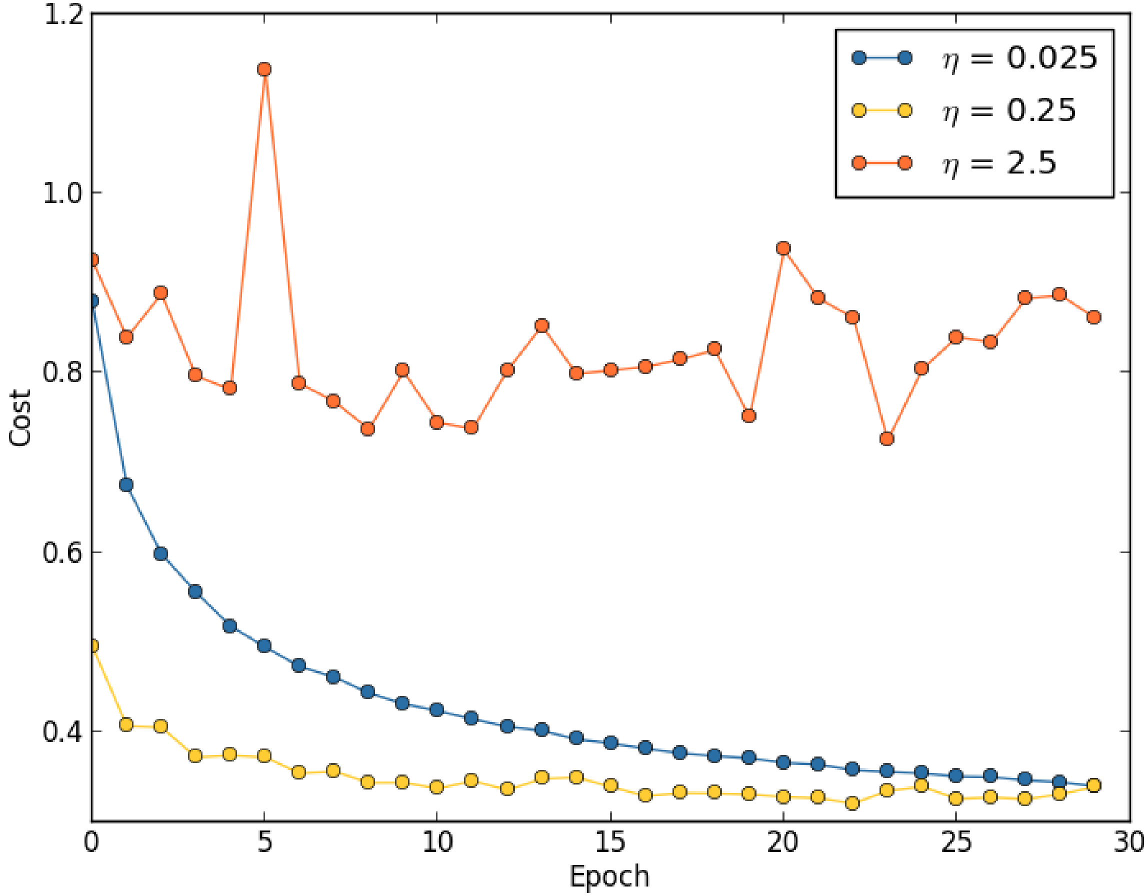
# Outline

- Activation Function
- Cost Function
- Data Preprocessing
- Optimization
  - Learning Rate
  - Momentum
- Generalization

# Outline

- Activation Function
- Cost Function
- Data Preprocessing
- Optimization
  - Learning Rate
  - Momentum
- Generalization

# Learning Rates




Source:
http://neuralnetworksanddeeplearning.com/chap3.html

# Learning Rates

- Popular & Simple Idea: Reduce the learning rate by some factor every few epochs.
  - At the beginning, we are far from the destination, so we use larger learning rate
  - After several epochs, we are close to the destination, so we reduce the learning rate
  - E.g. 1/t decay: $\eta^t = \eta / \sqrt{t+1}$
- Learning rate cannot be one-size-fits-all
  - Give different parameters different learning rates

# Adagrad

$$g^t = \frac{\partial C(\theta^t)}{\partial w} \qquad \eta^t = \frac{\eta}{\sqrt{t+1}}$$

- Divide the learning rate of each parameter by the ***root mean square of its previous derivatives***

***Vanilla Gradient descent***

$$w^{t+1} \leftarrow w^t - \eta^t g^t$$

w is one parameters

***Adagrad***

$$w^{t+1} \leftarrow w^t - \frac{\eta^t}{\sigma^t} g^t$$

$\sigma^t$: ***root mean square*** of the previous derivatives of parameter w

Parameter dependent

# Adagrad

$\sigma^t$: ***root mean square*** of the previous derivatives of parameter w

$$w^1 \leftarrow w^0 - \frac{\eta^0}{\sigma^0} g^0 \qquad \sigma^0 = g^0$$

$$w^2 \leftarrow w^1 - \frac{\eta^1}{\sigma^1} g^1 \qquad \sigma^1 = \sqrt{\frac{1}{2}[(g^0)^2 + (g^1)^2]}$$

$$w^3 \leftarrow w^2 - \frac{\eta^2}{\sigma^2} g^2 \qquad \sigma^2 = \sqrt{\frac{1}{3}[(g^0)^2 + (g^1)^2 + (g^2)^2]}$$

⋮

$$w^{t+1} \leftarrow w^t - \frac{\eta^t}{\sigma^t} g^t \qquad \sigma^t = \sqrt{\frac{1}{t+1}\sum_{i=0}^{t}(g^i)^2}$$

# Adagrad

- Divide the learning rate of each parameter by the ***root mean square of its previous derivatives***

$$w^{t+1} \leftarrow w^t - \frac{\eta^t}{\sigma^t} g^t$$

$$\eta^t = \frac{\eta}{\sqrt{t+1}}$$

1/t decay

$$\sigma^t = \sqrt{\frac{1}{t+1} \sum_{i=0}^{t} (g^i)^2}$$

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^{t} (g^i)^2}} g^t$$

# Contradiction?

$$g^t = \frac{\partial C(\theta^t)}{\partial w} \qquad \eta^t = \frac{\eta}{\sqrt{t+1}}$$

***Vanilla Gradient descent***

$$w^{t+1} \leftarrow w^t - \eta^t \underline{g^t}$$

Larger gradient, larger step

***Adagrad***

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\underline{\sum_{i=0}^{t}(g^i)^2}}} \underline{g^t}$$

Larger gradient, larger step

Larger gradient, smaller step

# Intuitive Reason

- *<u>反差</u>*

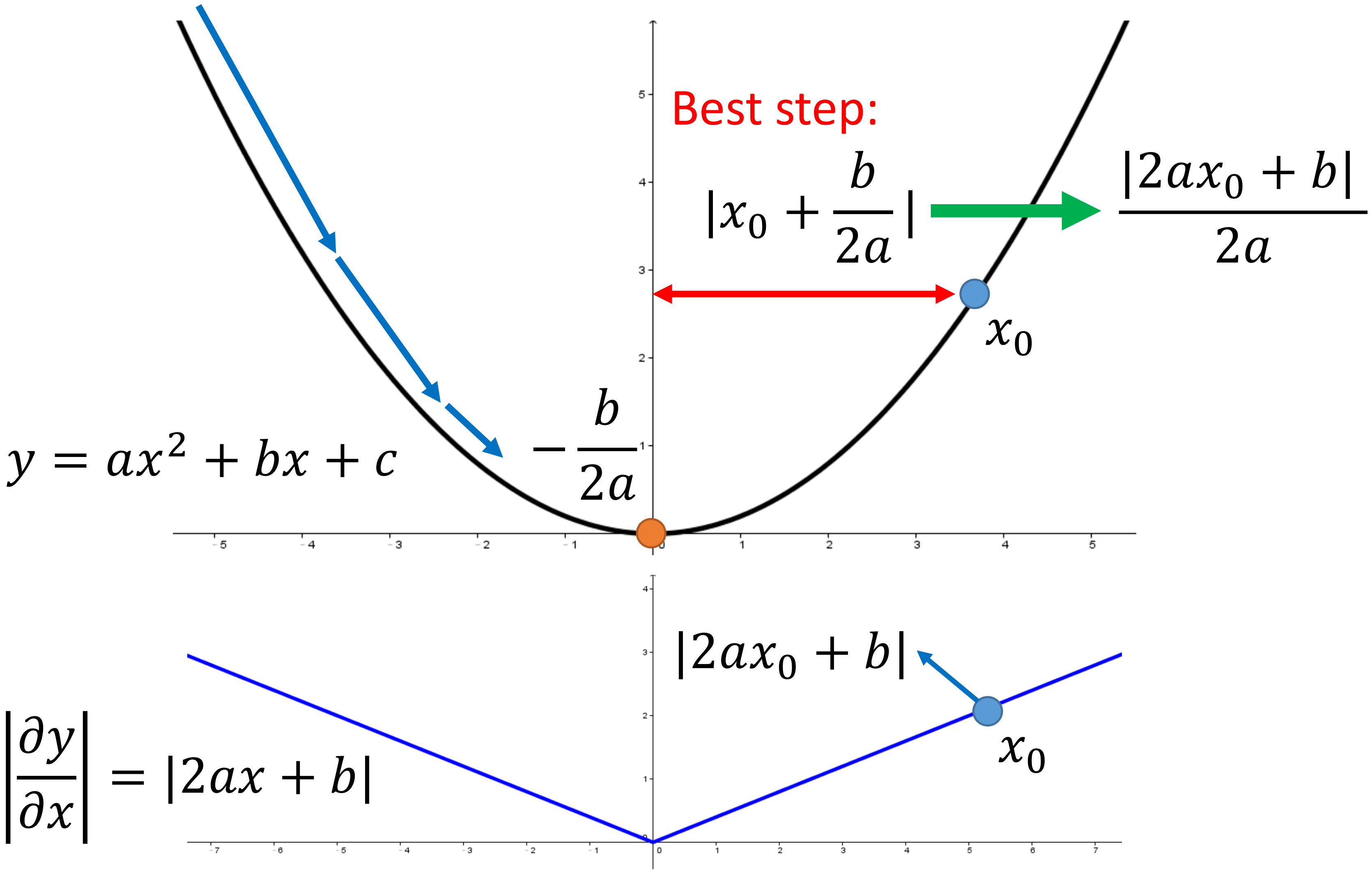
| $g^0$ | $g^1$ | $g^2$ | $g^3$ | $g^4$ | …… |
|---|---|---|---|---|---|
| 0.001 | 0.001 | 0.003 | 0.002 | 0.1 | …… |

| $g^0$ | $g^1$ | $g^2$ | $g^3$ | $g^4$ | …… |
|---|---|---|---|---|---|
| 10.8 | 20.9 | 31.7 | 12.1 | 0.1 | …… |

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^{t}(g^i)^2}} g^t$$

→ 造成反差的效果

# Larger gradient, larger steps?

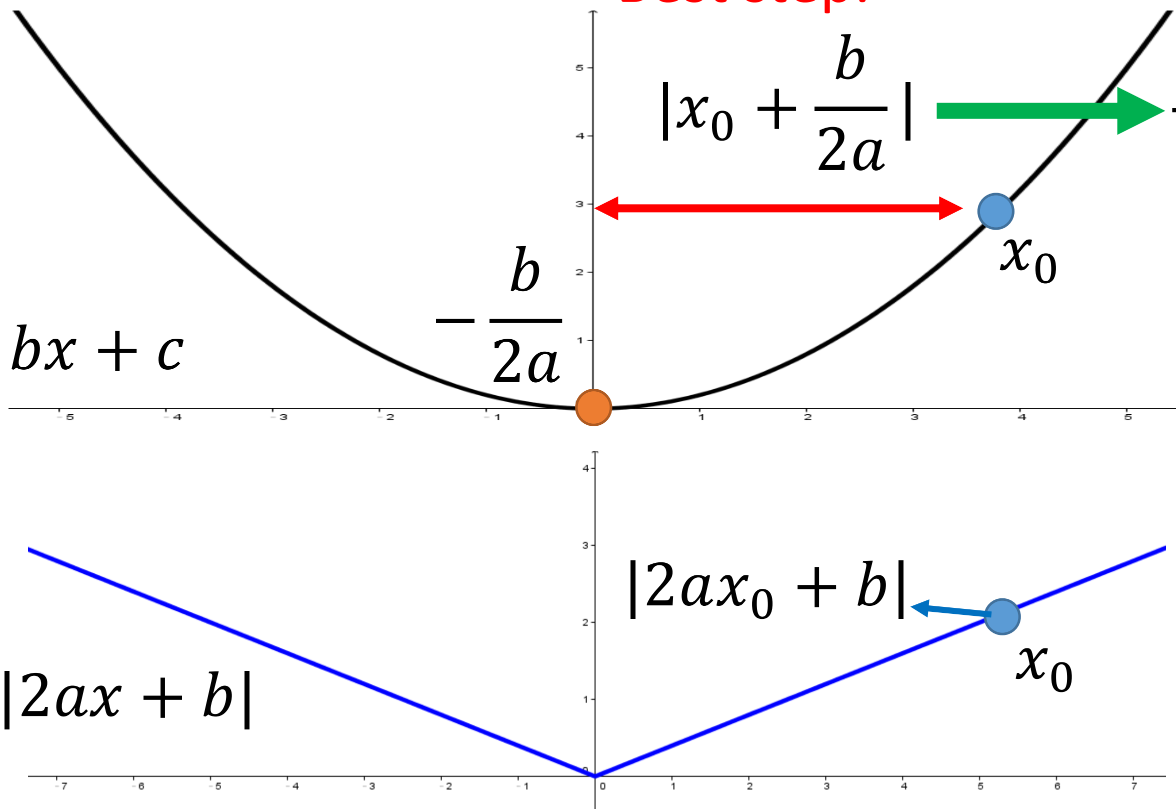


# Second Derivative

Best step:

$$|x_0 + \frac{b}{2a}| \rightarrow \frac{|2ax_0 + b|}{2a}$$

$$y = ax^2 + bx + c$$



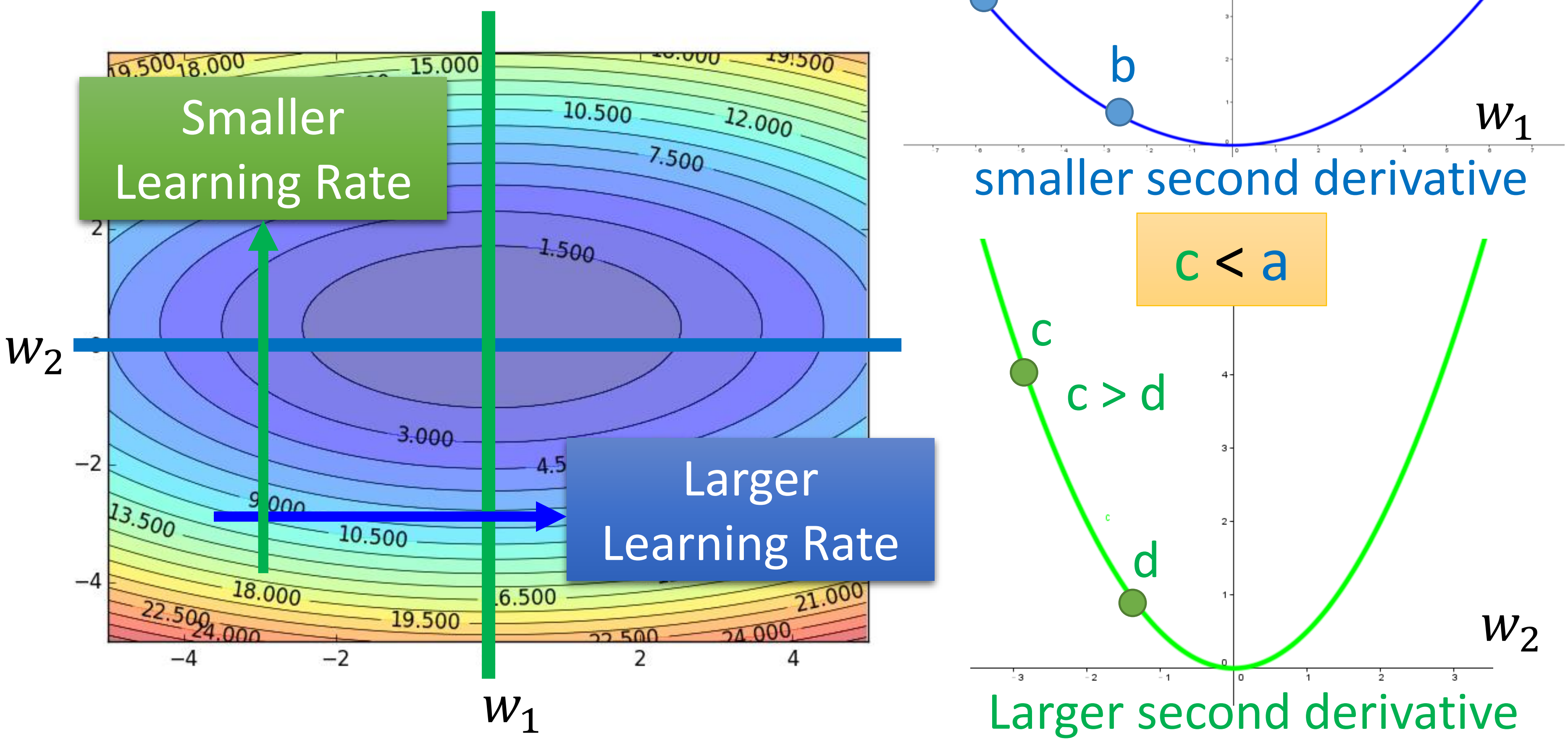

$$\left|\frac{\partial y}{\partial x}\right| = |2ax + b|$$

$$\frac{\partial^2 y}{\partial x^2} = 2a$$

The best step is $\dfrac{|\text{First derivative}|}{\text{Second derivative}}$

# More than one parameters

The best step is $\frac{|\text{First derivative}|}{\text{Second derivative}}$

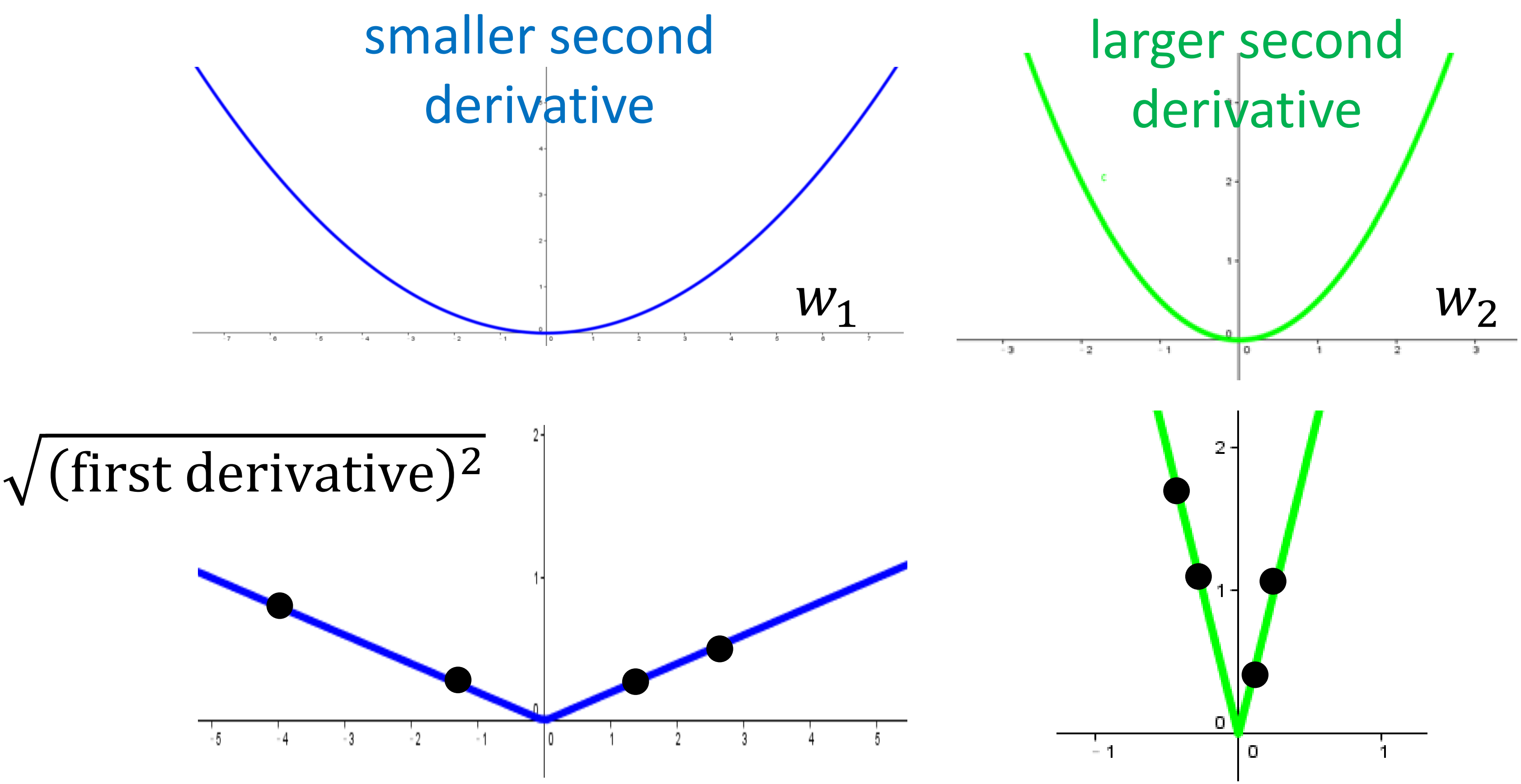


# What to do with Adagrad?

The best step is

$$\frac{|\text{First derivative}|}{\text{Second derivative}}$$

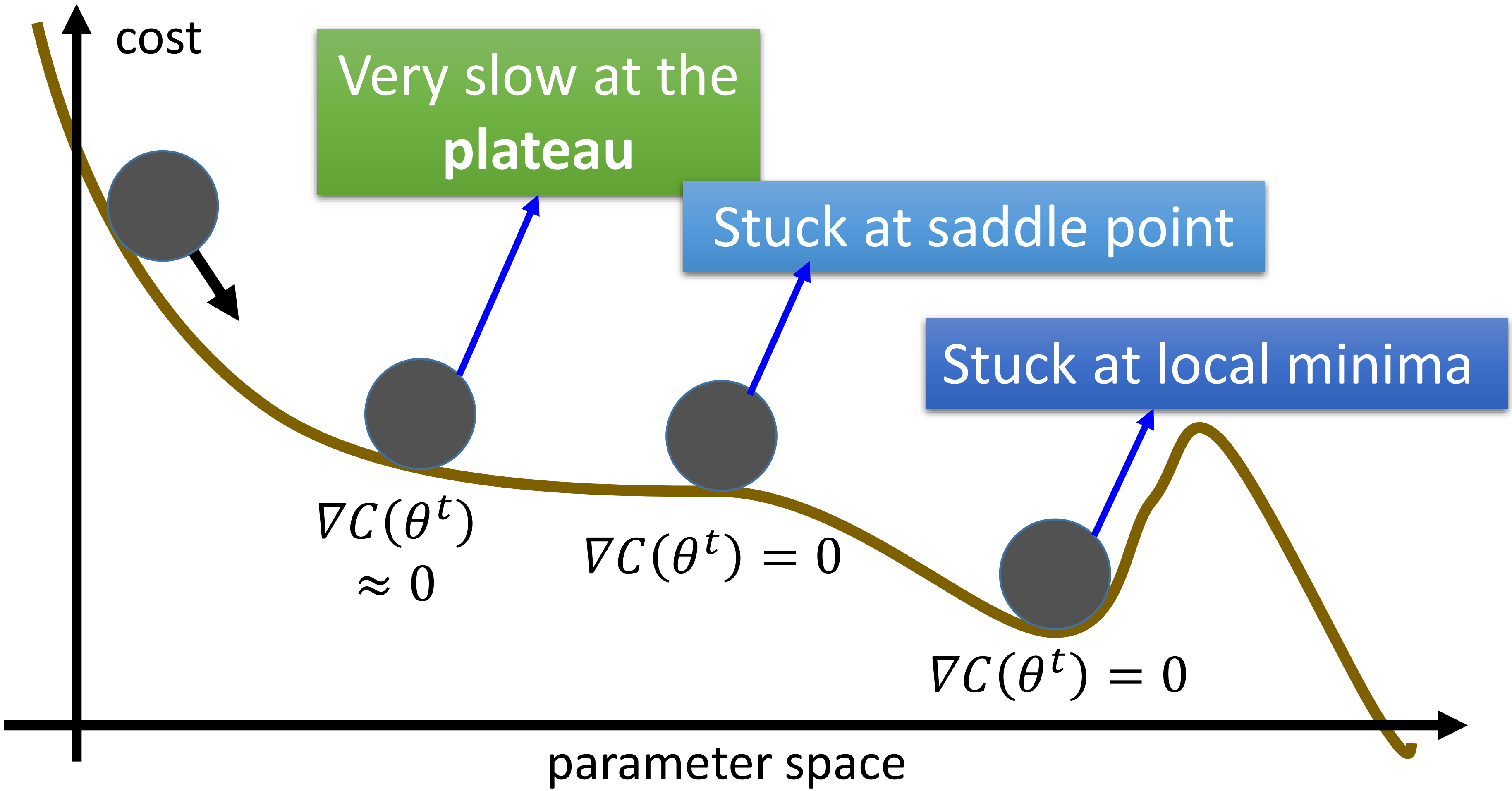
Use *first derivative* to estimate *second derivative*



# Outline

- Activation Function
- Cost Function
- Data Preprocessing
- Optimization
  - Learning Rate
  - Momentum
- Generalization

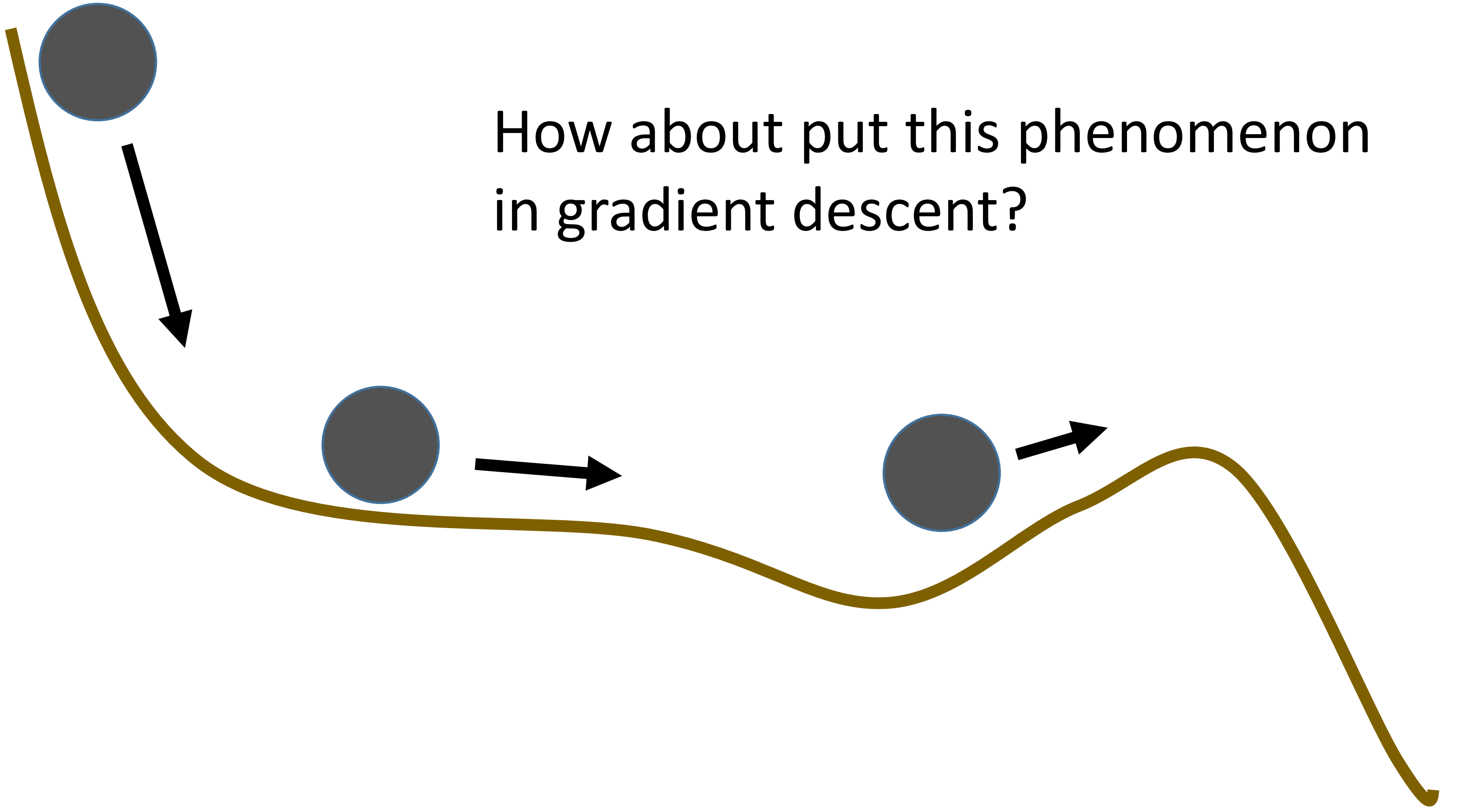
# Easy to stuck



# In physical world ……

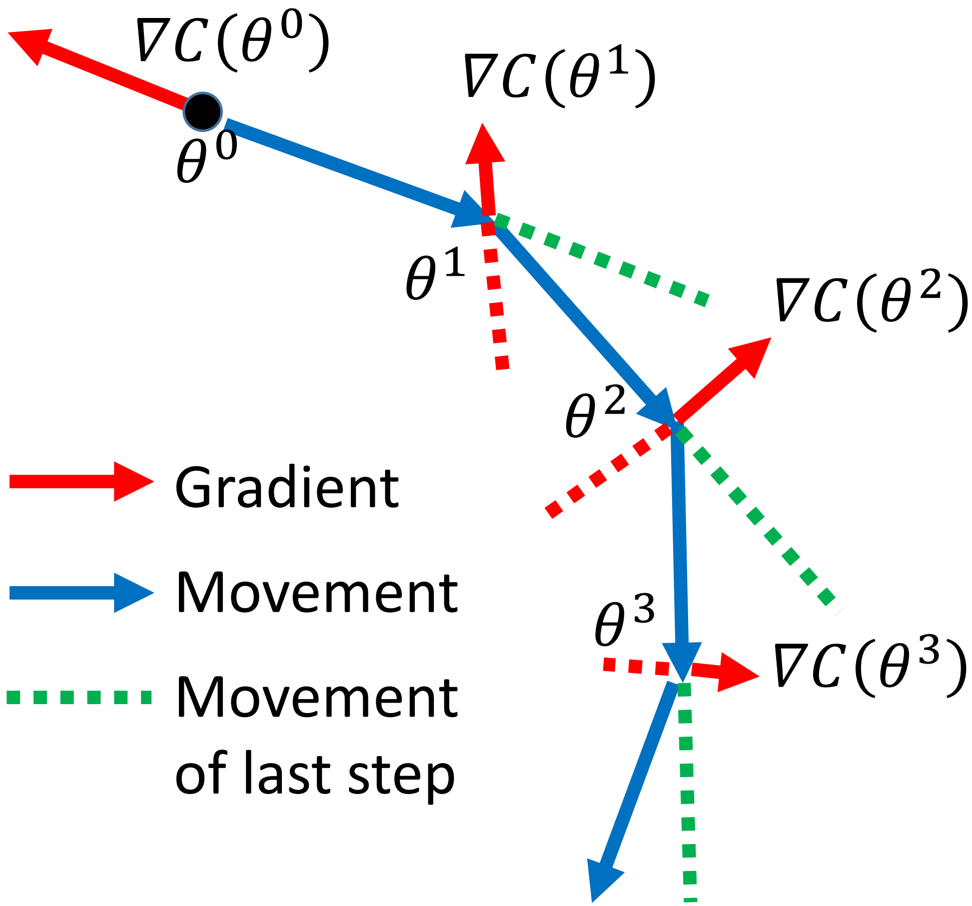
- Momentum

How about put this phenomenon in gradient descent?



# Momentum

Movement: movement of last step minus gradient at present




Start at point $\theta^0$

Movement $v^0=0$

Compute gradient at $\theta^0$

Movement $v^1 = \lambda v^0 - \eta \nabla C(\theta^0)$

Move to $\theta^1 = \theta^0 + v^1$

Compute gradient at $\theta^1$

Movement $v^2 = \lambda v^1 - \eta \nabla C(\theta^1)$

Move to $\theta^2 = \theta^1 + v^2$

Movement not just based on gradient, but previous movement.

# Momentum

Movement: movement of last step minus gradient at present

$v^i$ is actually the weighted sum of all the previous gradient: $\nabla C(\theta^0)$, $\nabla C(\theta^1)$, … $\nabla C(\theta^{i-1})$

$v^0 = 0$

$v^1 = - \eta \nabla C(\theta^0)$

$v^2 = - \lambda \eta \nabla C(\theta^0) - \eta \nabla C(\theta^1)$

⋮

Start at point $\theta^0$

Movement $v^0=0$

Compute gradient at $\theta^0$

Movement $v^1 = \lambda v^0 - \eta \nabla C(\theta^0)$

Move to $\theta^1 = \theta^0 + v^1$

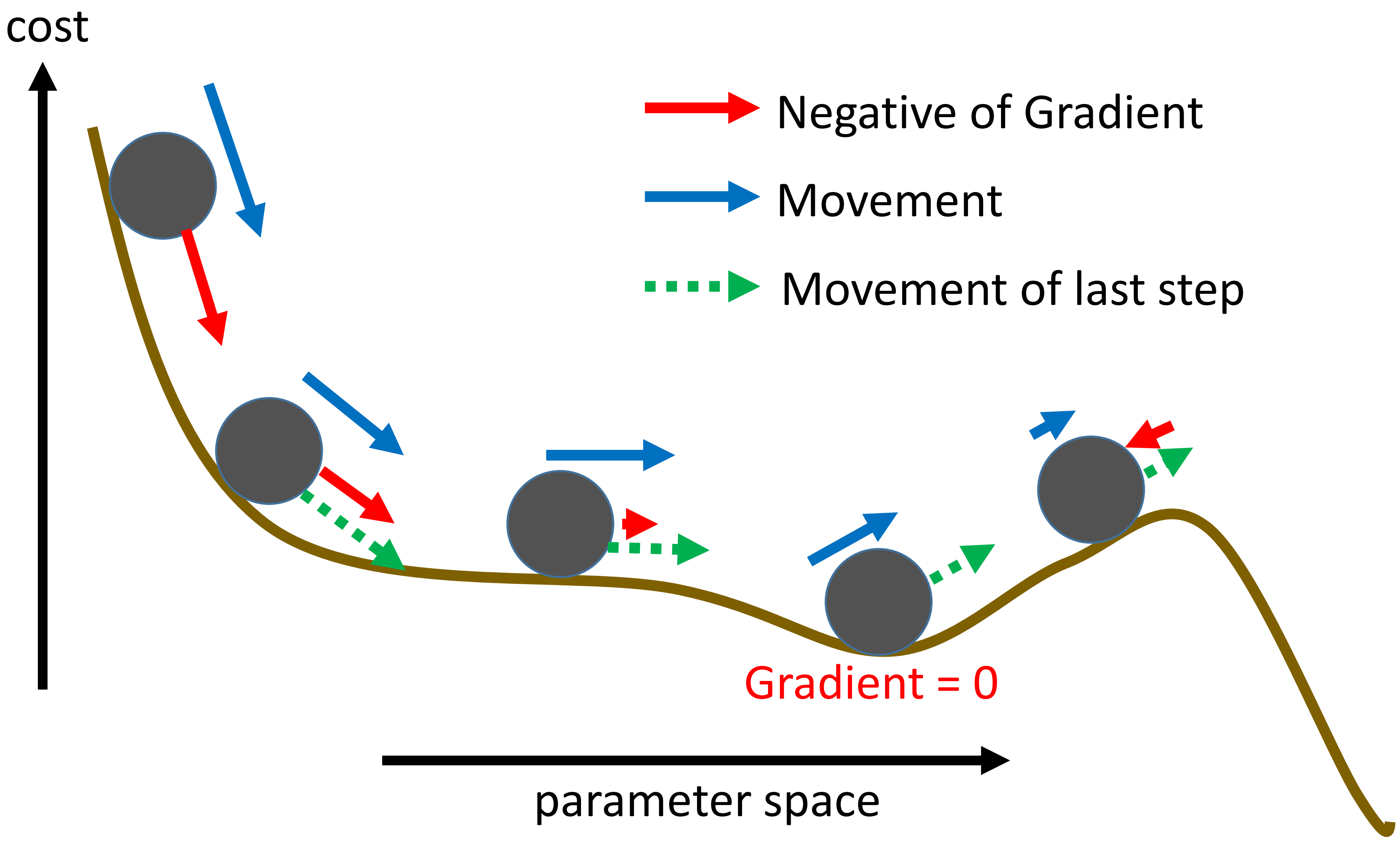
Compute gradient at $\theta^1$

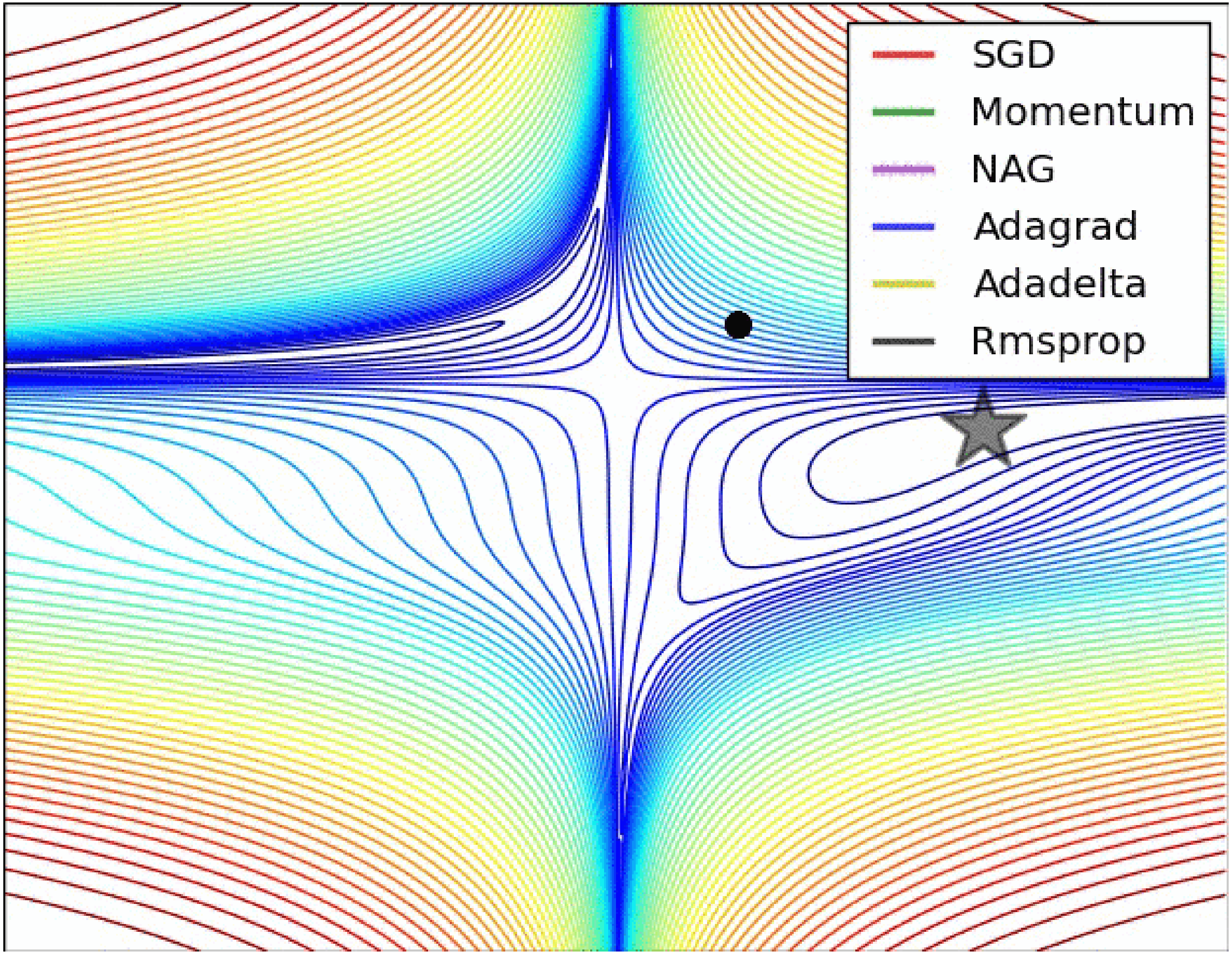
Movement $v^2 = \lambda v^1 - \eta \nabla C(\theta^1)$

Move to $\theta^2 = \theta^1 + v^2$

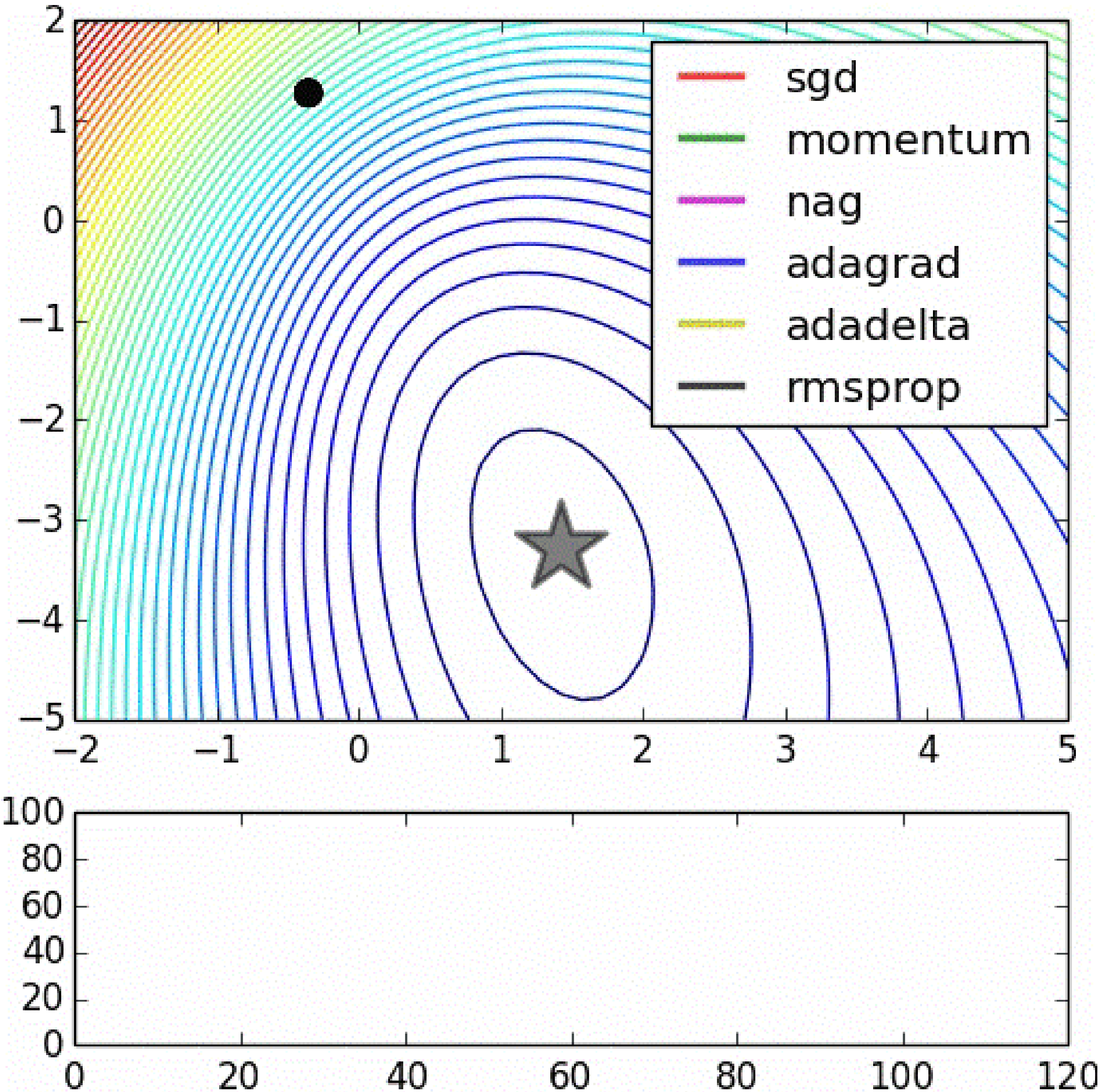
Movement not just based on gradient, but previous movement

# Momentum


cost
Negative of Gradient
Movement
Movement of last step
Gradient = 0
parameter space




http://www.reddit.com/r/MachineLearning/comments/2gopfa/visualizing_gradient_optimization_techniques/cklhott (**By Alec Radford**)




http://www.reddit.com/r/MachineLearning/comments/2gopfa/visualizing_gradient_optimization_techniques/cklhott **(By Alec Radford)**

# Outline

Activation Function

Cost Function

Data Preprocessing

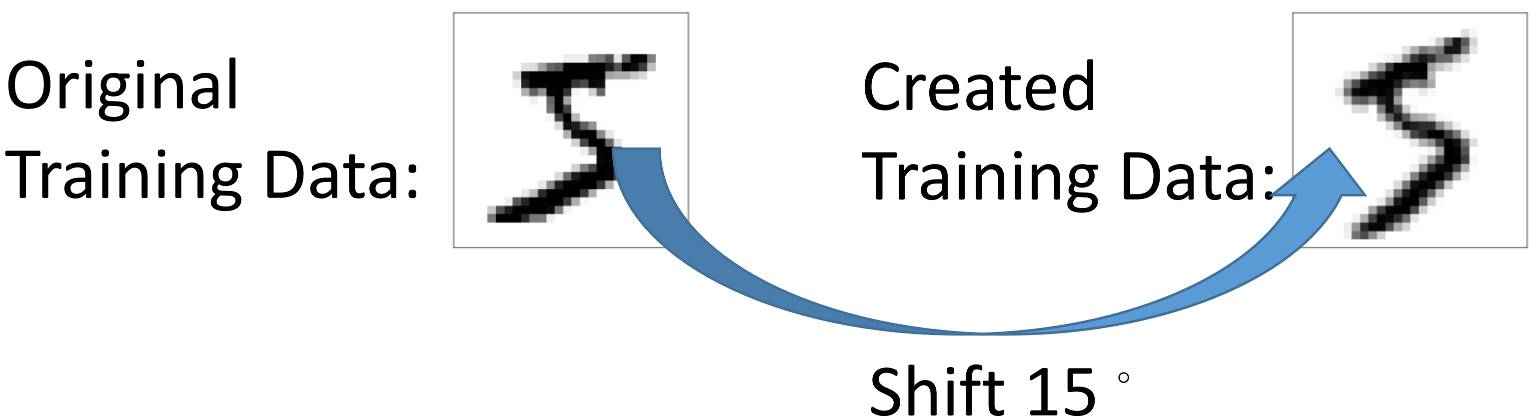
Optimization

Generalization

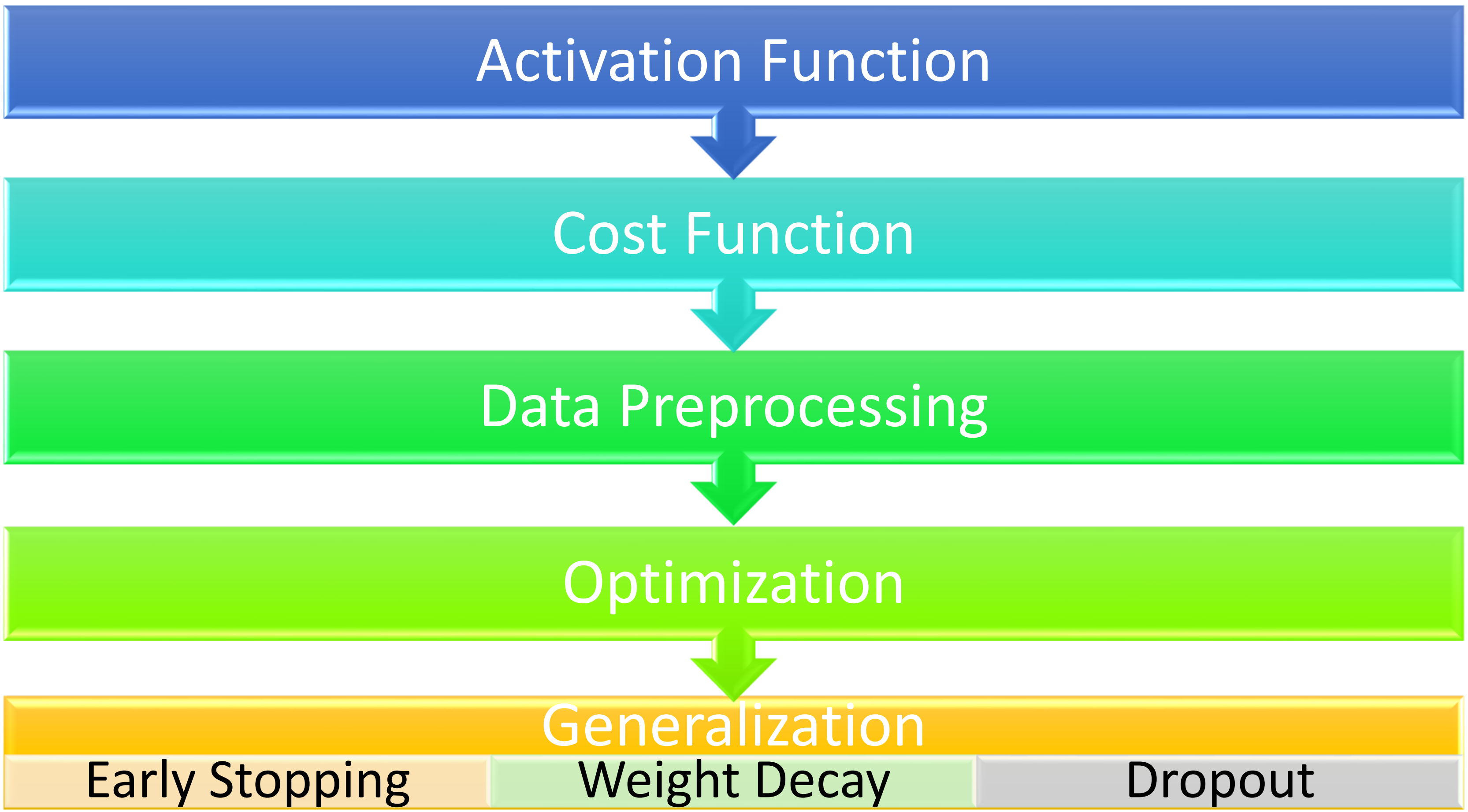
# Panacea

- Have more training data
- ***Create*** more training data (?)

Handwriting recognition:



# Outline

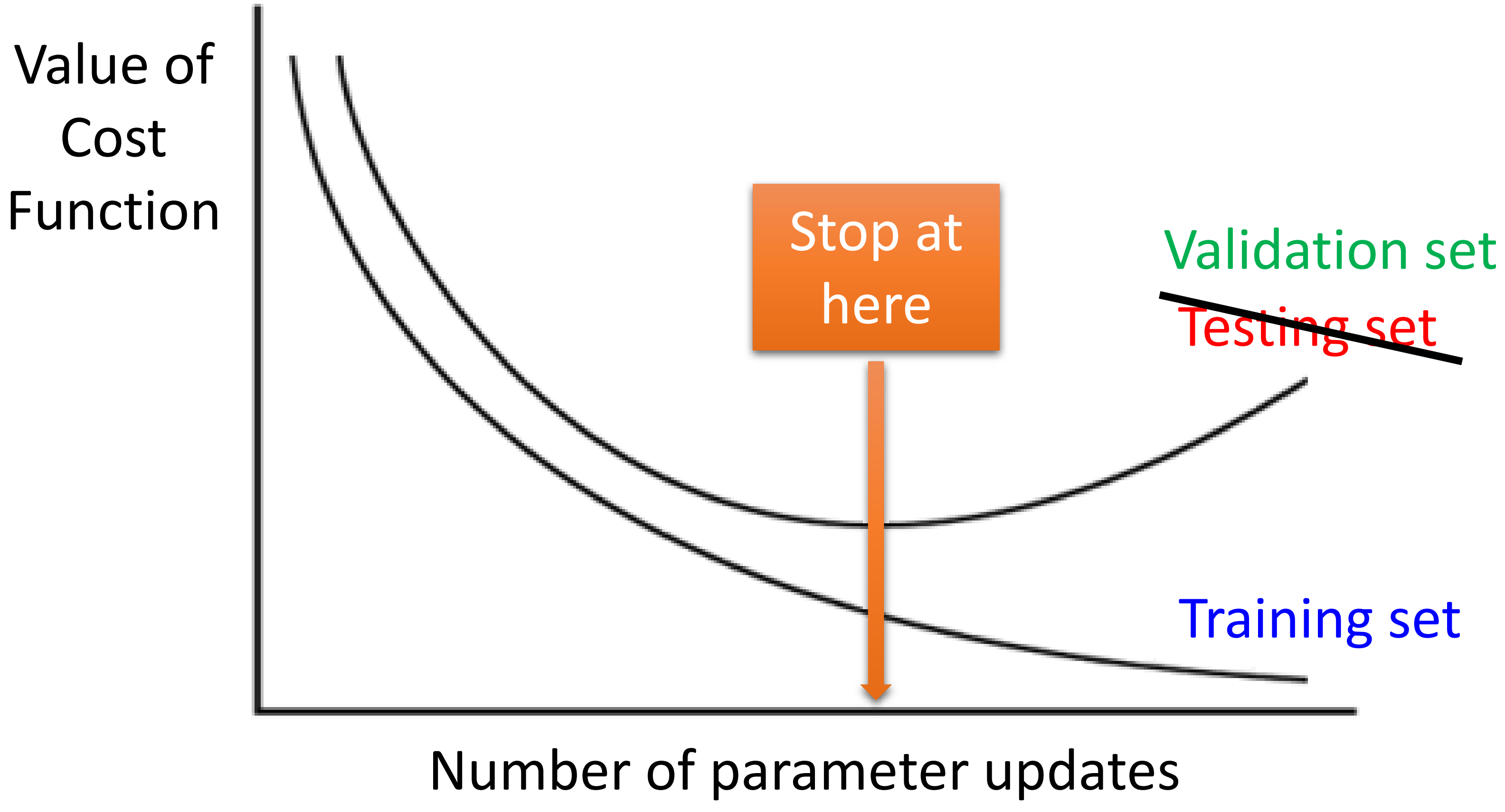

Activation Function
Cost Function
Data Preprocessing
Optimization
Generalization
Early Stopping
Weight Decay
Dropout

# Outline

- Activation Function
- Cost Function
- Data Preprocessing
- Optimization
- Generalization
  - Early Stopping
  - Weight Decay
  - Dropout

# Early Stopping

How many parameter updates do we need?



# Outline

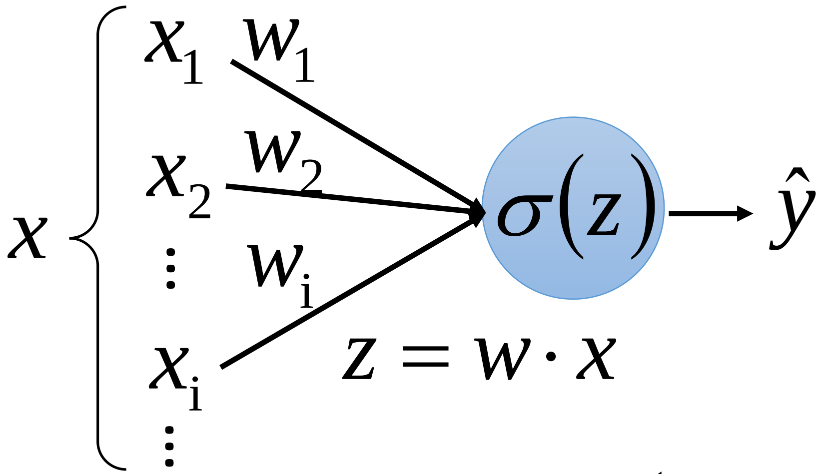
- Activation Function
- Cost Function
- Data Preprocessing
- Optimization
- Generalization
  - Early Stopping
  - Weight Decay
  - Dropout

# Weight Decay

- The parameters closer to zero is preferred.

Training data:

$$\{(x, \hat{y}), \ldots\}$$




$$z = w \cdot x$$

Testing data:

$$\{(x', \hat{y}), \ldots\}$$

$$x' = x + \varepsilon$$

$$z' = w \cdot (x + \varepsilon) = w \cdot x + w \cdot \varepsilon = z + w \cdot \varepsilon$$

To minimize the effect of noise, we want w close to zero.

# Weight Decay

- New cost function to be minimized
  - Find a set of weight not only minimizing original cost but also close to zero

$$C'(\theta) = C(\theta) + \lambda \frac{1}{2} \|\theta\|^2$$

$C(\theta)$ → Original cost (e.g. minimize square error, cross entropy …)

$\frac{1}{2}\|\theta\|^2$ → Regularization term:

$$\theta = \{W^1, W^2, \ldots\}$$

$$\|\theta\|^2 = \left(w_{11}^1\right)^2 + \left(w_{12}^1\right)^2 + \ldots$$
$$+ \left(w_{11}^2\right)^2 + \left(w_{12}^2\right)^2 + \ldots$$

(not consider biases. why?)

# Weight Decay

$$\|\theta\|^2 = \left(w_{11}^1\right)^2 + \left(w_{12}^1\right)^2 + \ldots$$
$$+ \left(w_{11}^2\right)^2 + \left(w_{12}^2\right)^2 + \ldots$$

- New cost function to be minimized

$$\mathrm{C}'(\theta) = C(\theta) + \lambda \frac{1}{2}\|\theta\|^2$$

Gradient: $$\frac{\partial \mathrm{C}'}{\partial w} = \frac{\partial \mathrm{C}}{\partial w} + \lambda w$$

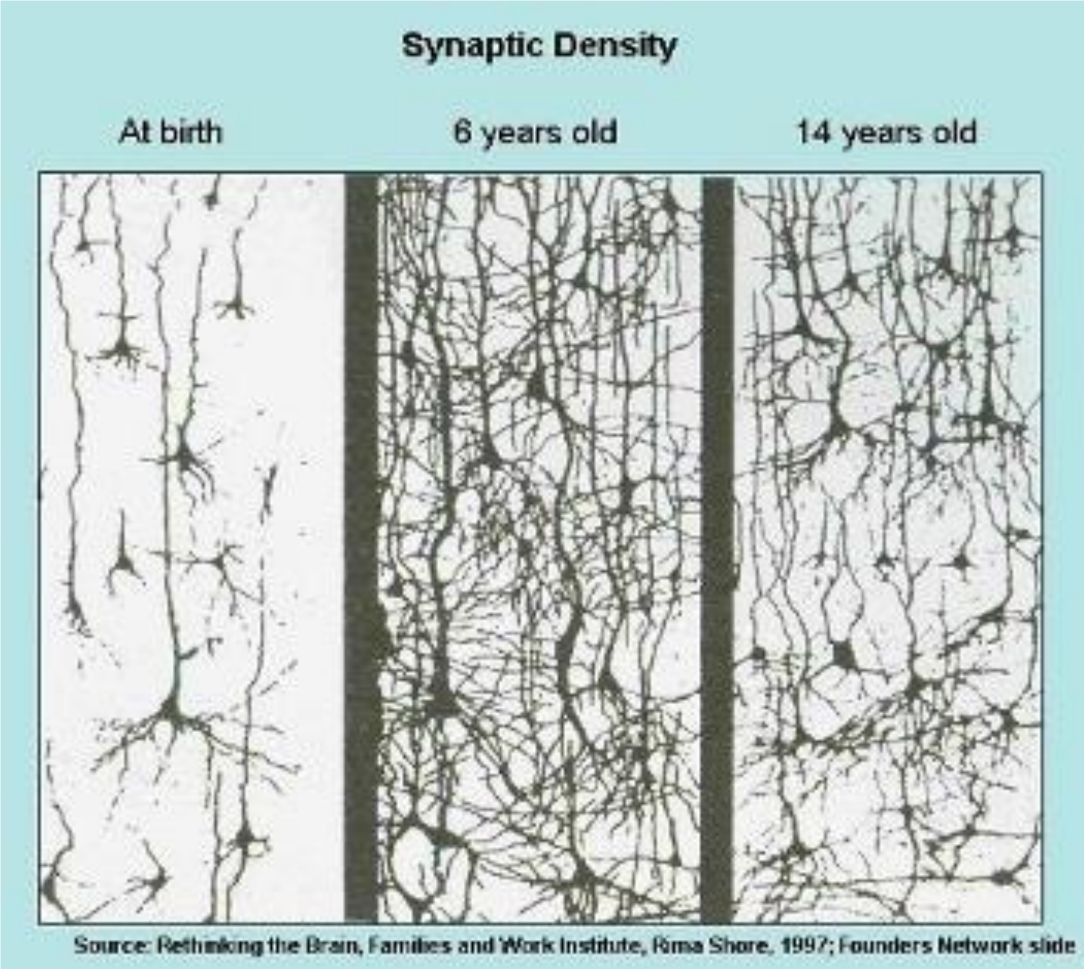
Update: $$w^{t+1} \rightarrow w^t - \eta \frac{\partial \mathrm{C}'}{\partial w} = w^t - \eta\left(\frac{\partial \mathrm{C}}{\partial w} + \lambda w^t\right)$$

$$= \underline{(1 - \eta\lambda) w^t} - \eta \underline{\frac{\partial \mathrm{C}}{\partial w}}$$

Smaller and smaller

# Weight Decay

- Our Brain



# Outline

- Activation Function
- Cost Function
- Data Preprocessing
- Optimization
- Generalization
  - Early Stopping
  - Weight Decay
  - Dropout

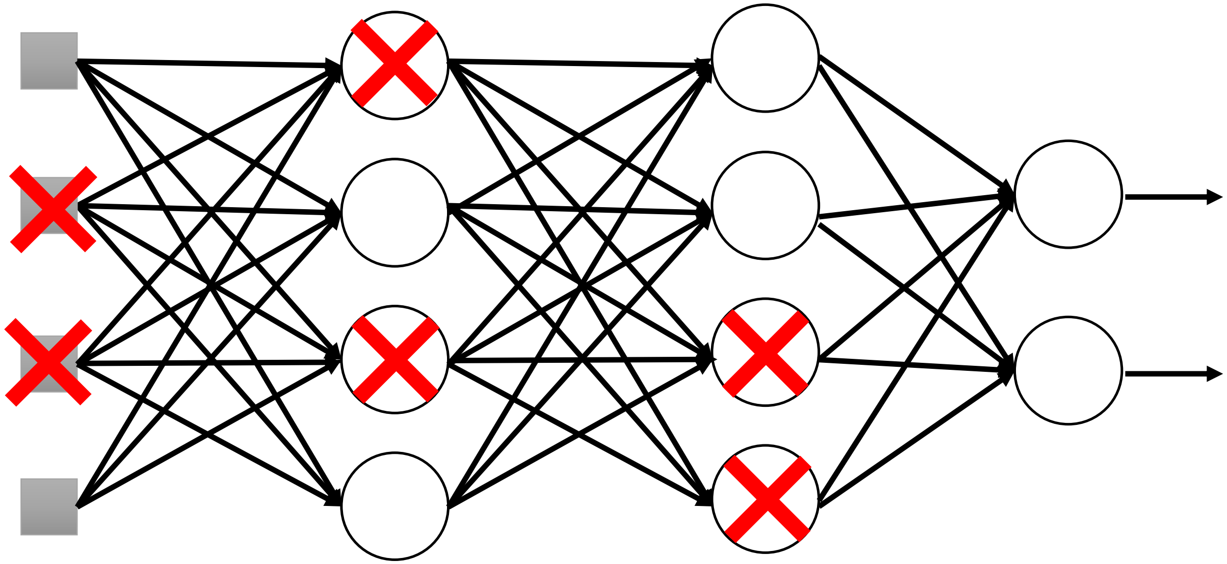
# Dropout

$$\theta^t \leftarrow \theta^{t-1} - \eta \nabla C_x(\theta^{t-1})$$

**Training:**

$x$

(stochastic gradient descent)



- In each ***iteration***
  - Each neuron has p% to dropout

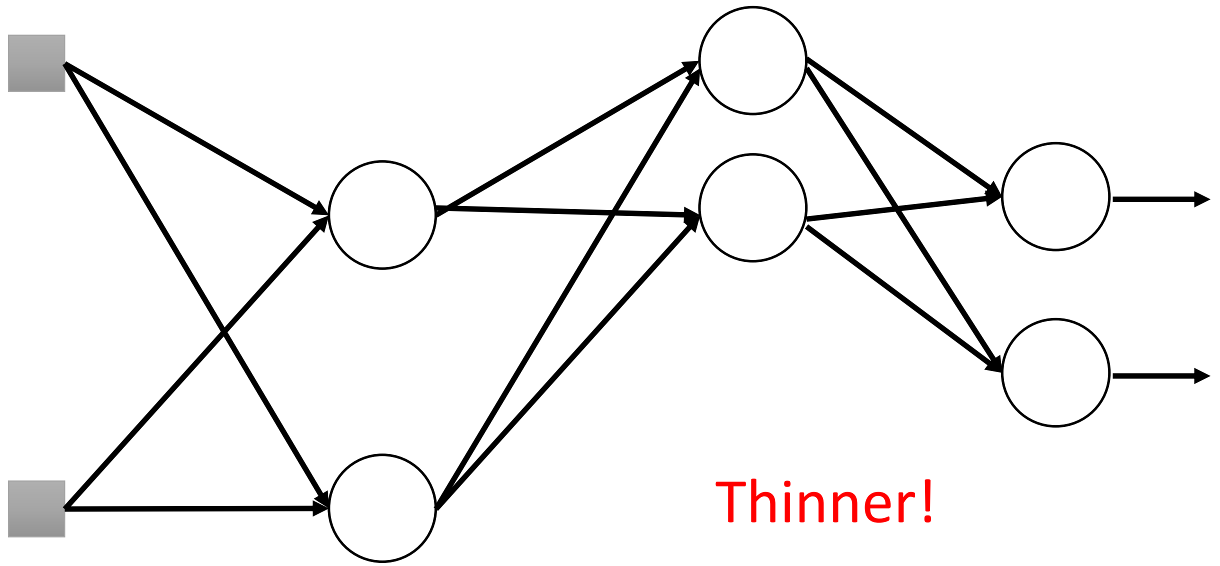
# Dropout

$$\theta^t \leftarrow \theta^{t-1} - \eta \nabla C_x(\theta^{t-1})$$

**Training:**

$x$

(stochastic gradient descent)

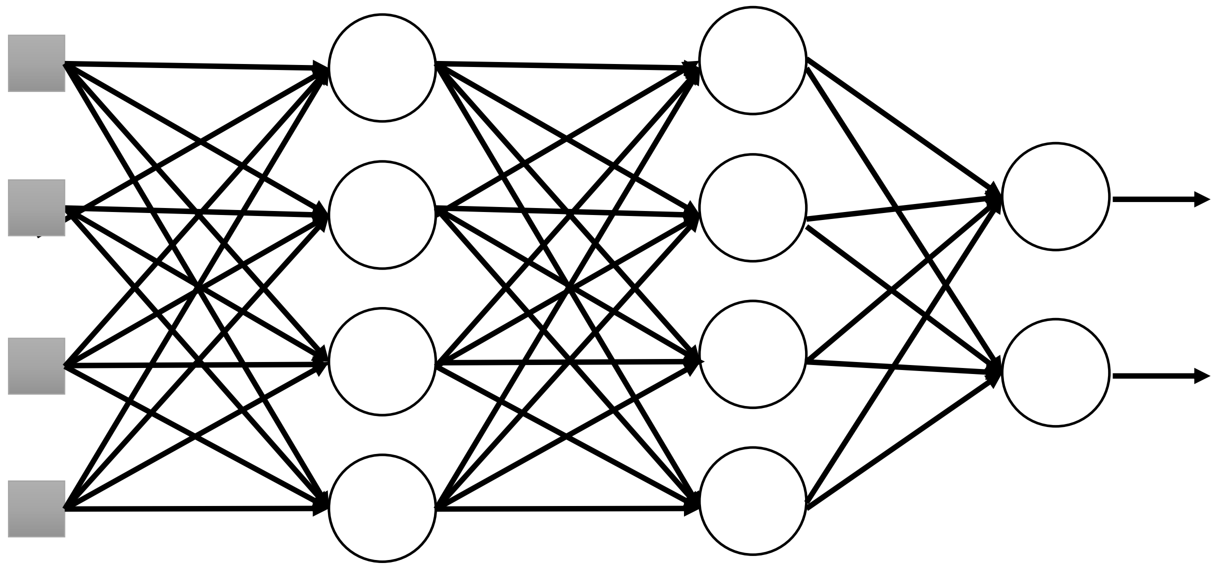



- In each ***iteration***
  - Each neuron has p% to dropout
    
    ⇨ **The structured of the network is changed.**
  - Using the new network for training

For each iteration, we resample the dropout neurons

# Dropout

**Testing:**



- **No dropout**
  - If the dropout rate at training is p%, all the weights times (1-p)%
  - Assume that the dropout rate is 50%. If $w_{ij}^{l} = 1$ from training, set $w_{ij}^{l} = 0.5$ for testing.

# Dropout
# - Intuitive Reason

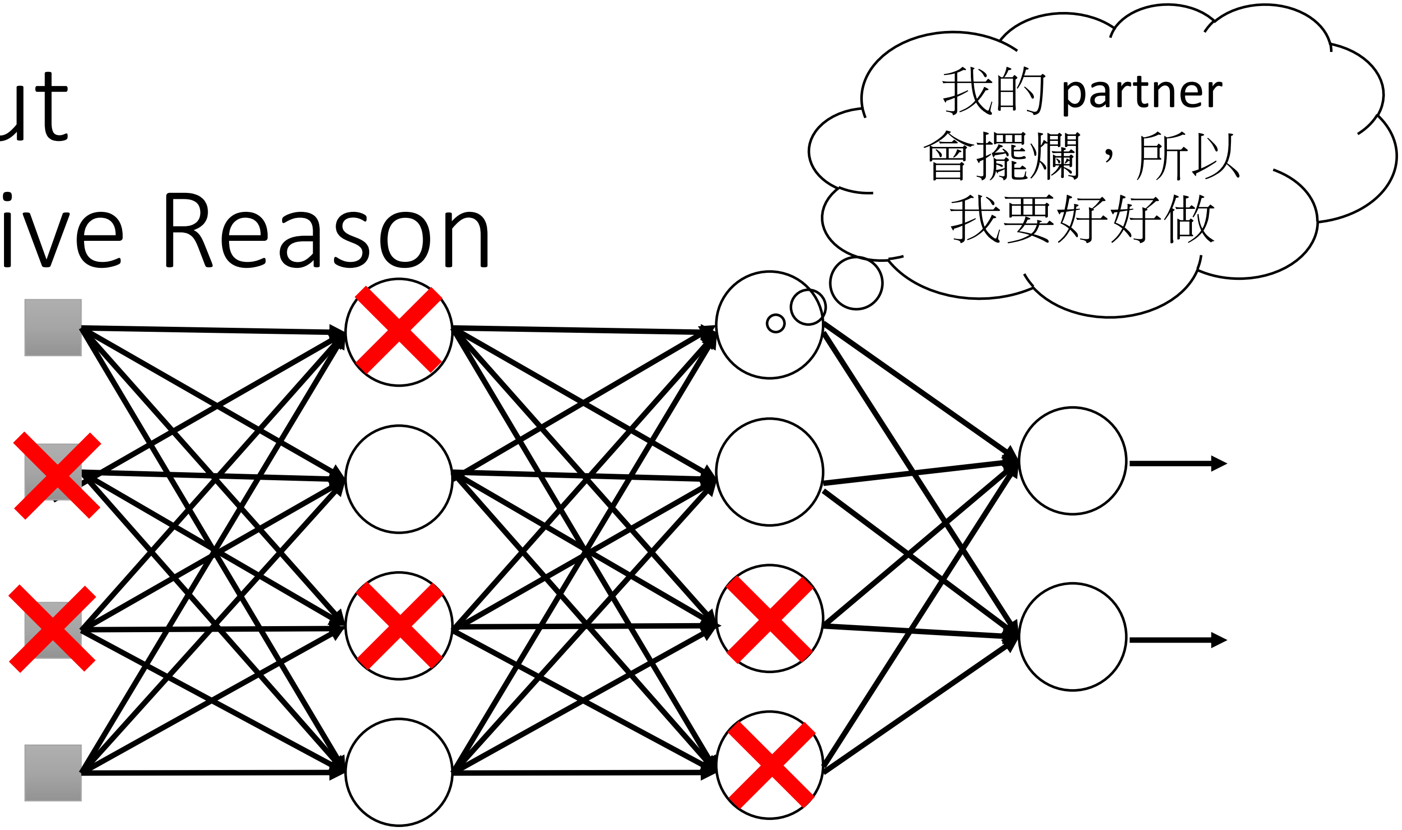
***Testing***

No dropout
(拿下重物後就變很強)

***Training***

Dropout (腳上綁重物)


# Dropout
# - Intuitive Reason



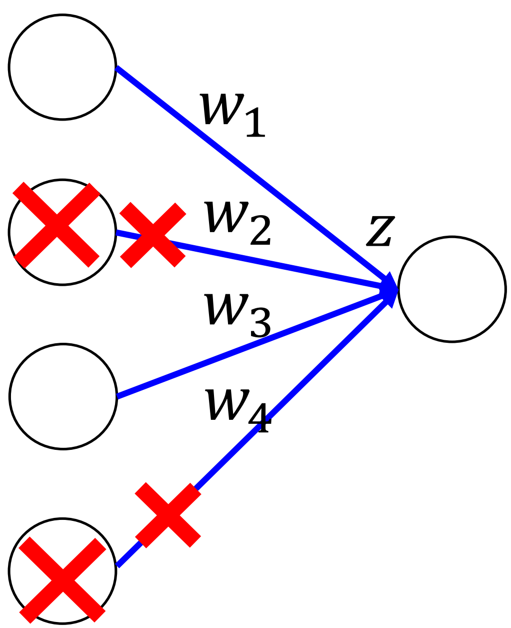

- When teams up, if everyone expect the partner will do the work, nothing will be done finally.
- However, if you know your partner will dropout, you will do better.
- When testing, no one dropout actually, so obtaining good results eventually.

# Dropout
# - Intuitive Reason

- Why the weights should multiply (1-p)% (dropout rate) when testing?

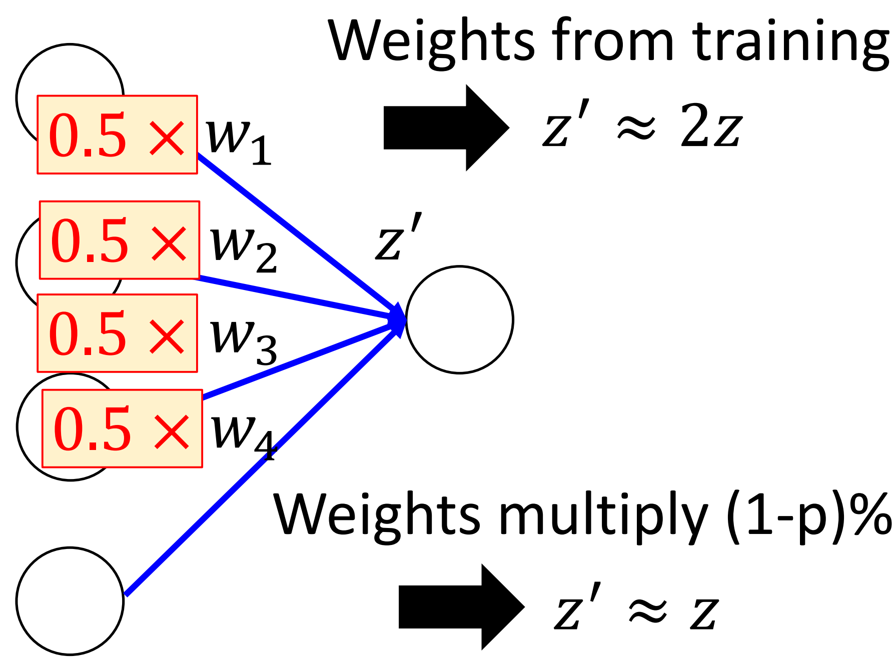
***Training of Dropout***

Assume dropout rate is 50%



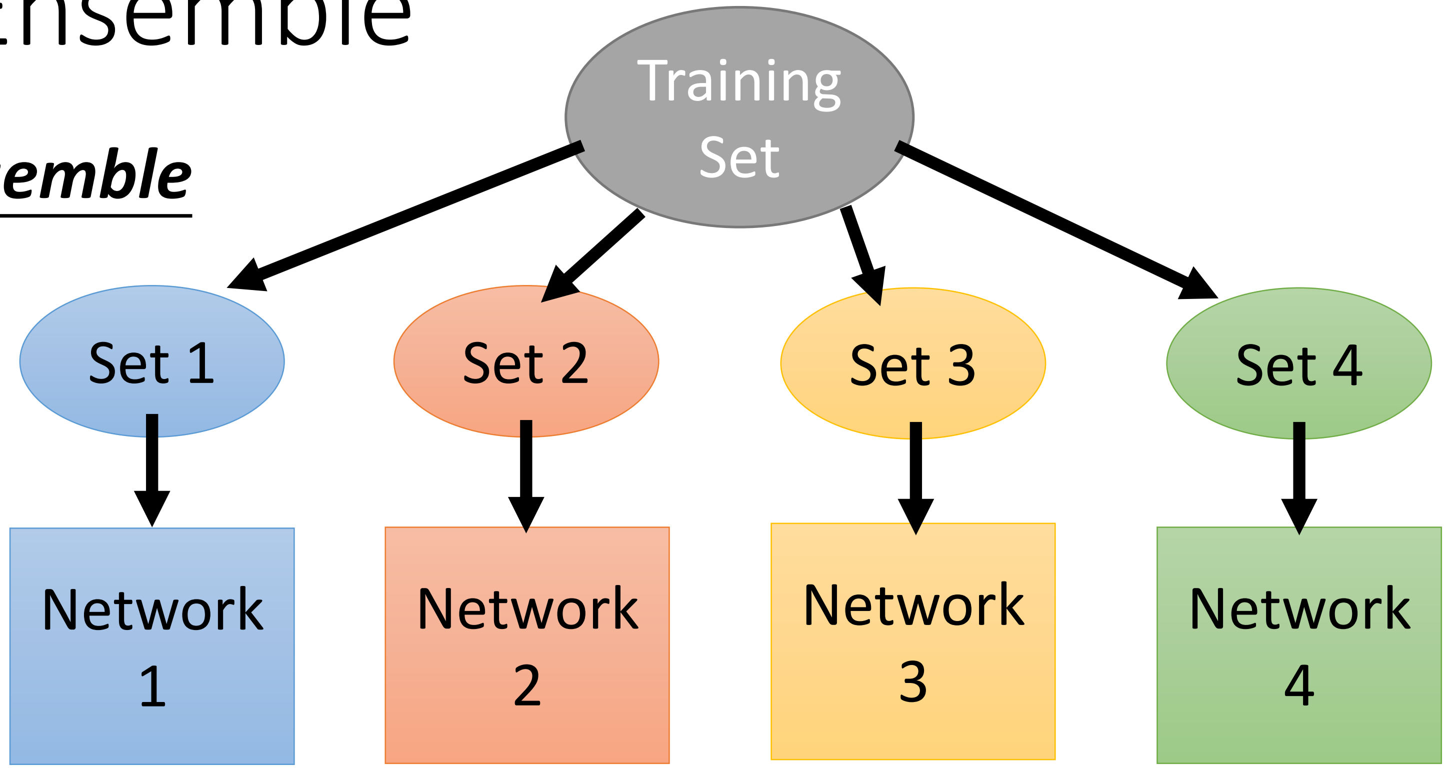

***Testing of Dropout***

No dropout



# Dropout
# - Ensemble

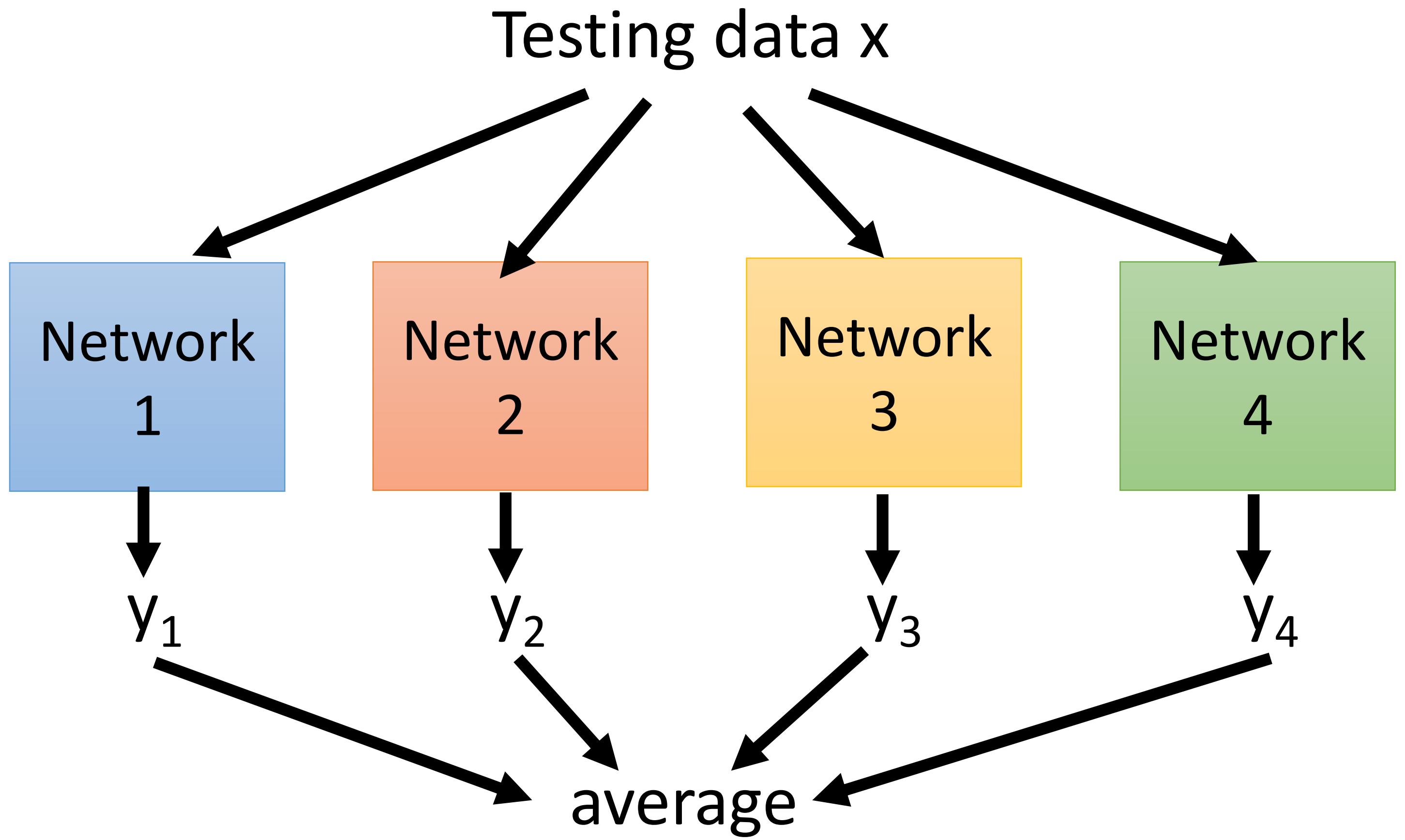
***Ensemble***




Train a bunch of networks with different structures

# Dropout
# - Ensemble

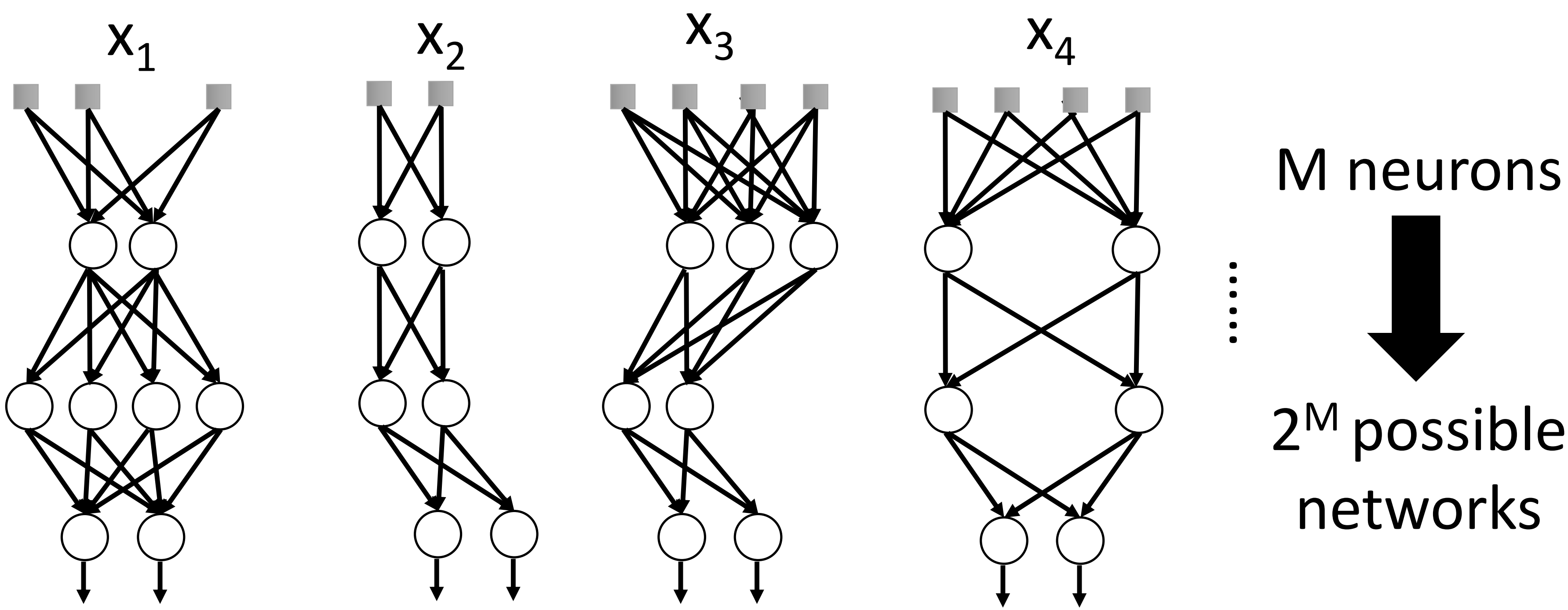
***Ensemble***



# Dropout
# - Ensemble

***Dropout*** ≈ ***Ensemble***.

***Training of Dropout***

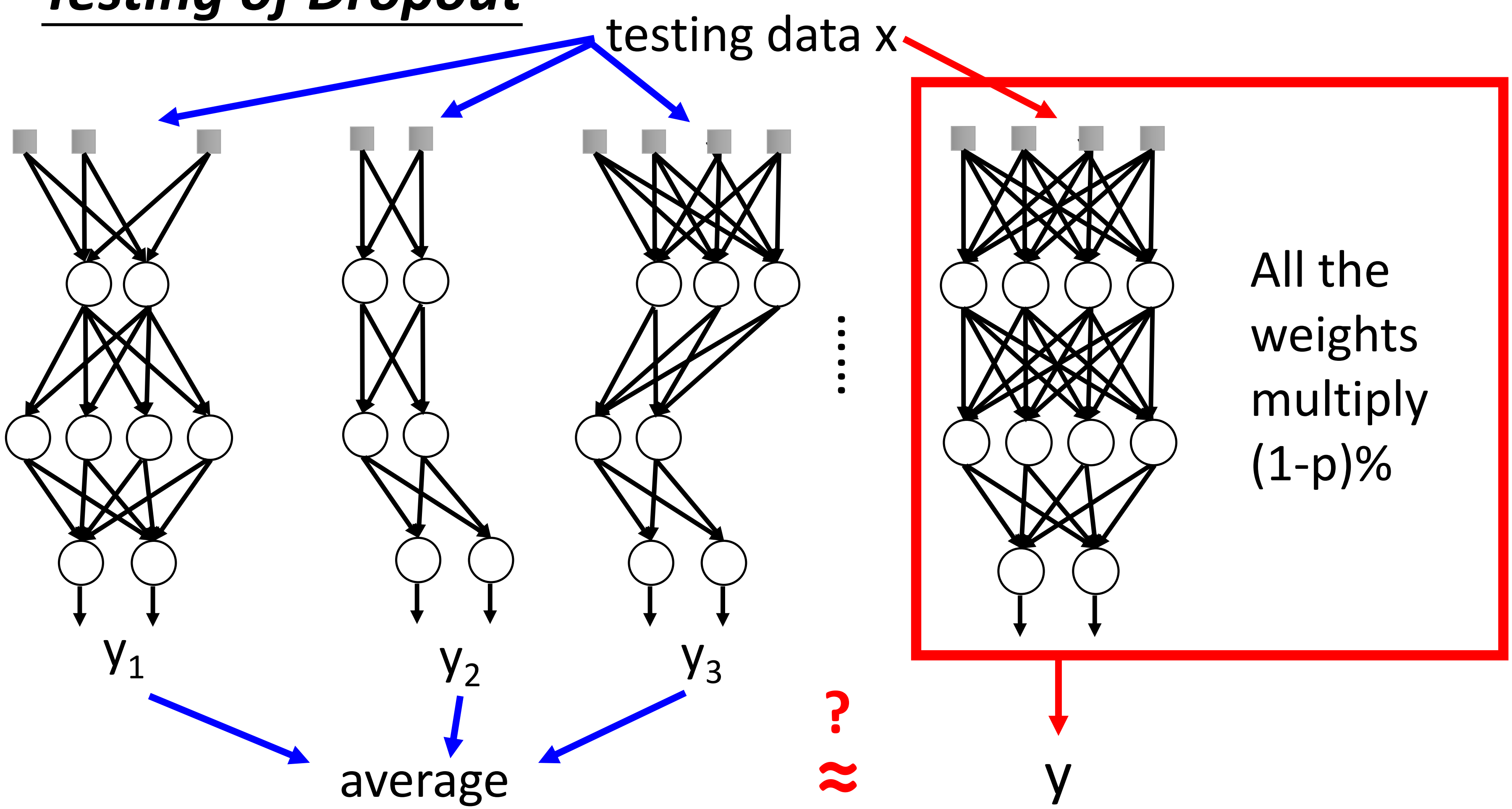



- Using one data to train one network
- Some parameters in the network are shared

# Dropout - Ensemble

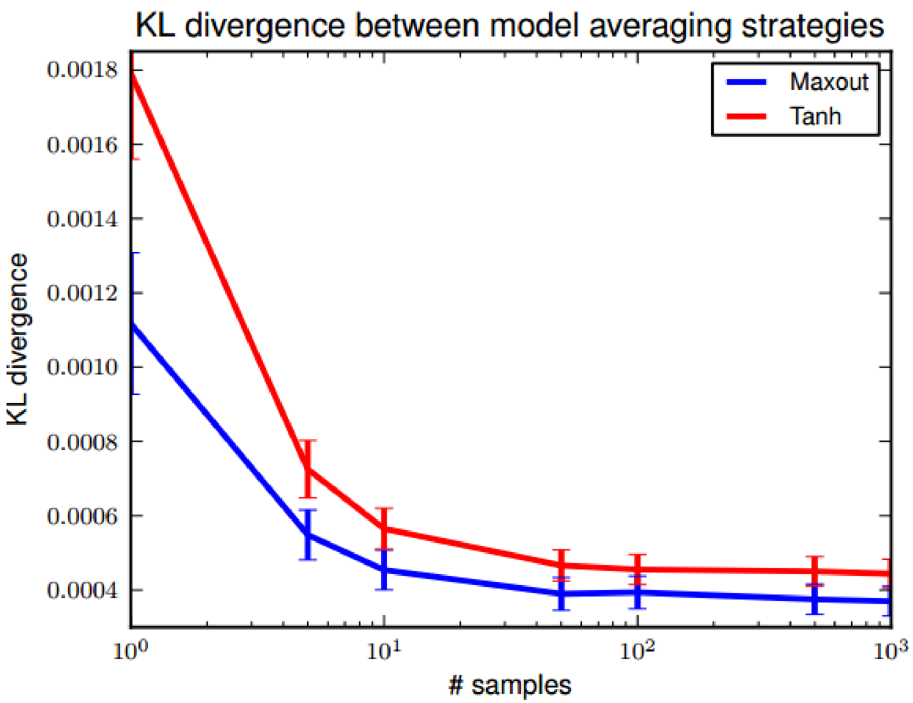
***Dropout* ≈ *Ensemble*.**

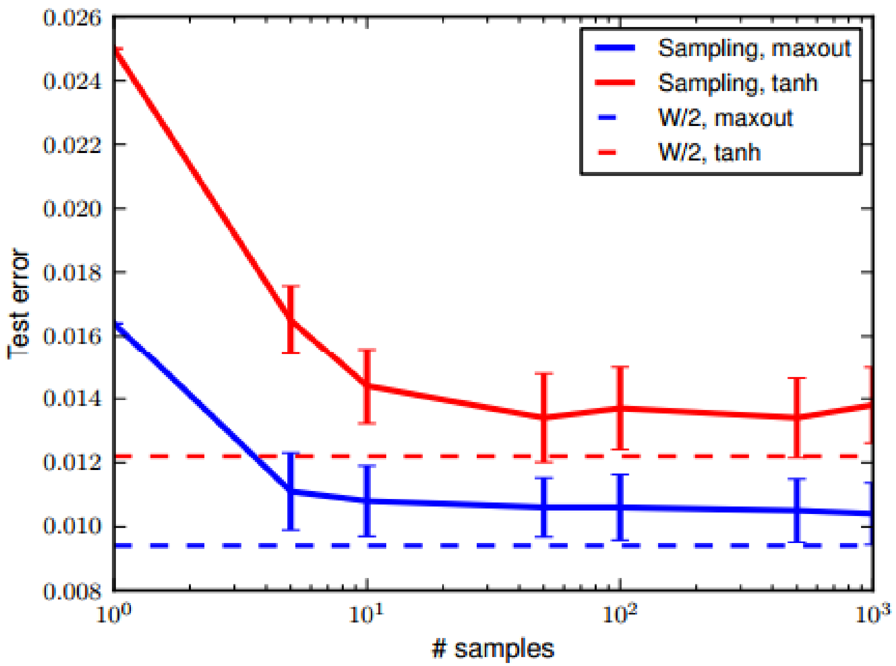
***Testing of Dropout***



# Dropout
# - Ensemble

- Experiments on hand writing digital classification




Ref: http://arxiv.org/pdf/1302.4389.pdf

# Practical Suggestion for Dropout

- Larger network
  - If you know your task need ***n*** neurons, for dropout rate ***p***, your network need ***n/(1-p)*** neurons.
- Longer training time
- Higher learning rate
- Larger momentum

# Concluding Remarks

Not covered today:
Parameters Initialization

http://neuralnetworksanddeeplearning.com/chap3.html#weight_initialization

- **Activation Function** — ReLU, Maxout
- **Cost Function** — Softmax + Cross Entropy
- **Data Preprocessing** — Mean=0, Variance=1
- **Optimization** — Adagrad, Momentum
- **Generalization** — Early Stopping, Weight decay, Dropout

# Acknowledgement


- 感謝 李朋軒 同學糾正投影片上的錯誤
  - 很多地方 p 應該改為 1-p
- 感謝 Ryan Sun 來信指出投影片上的錯誤