NN Practical Tips

Nov 10th, 2016

Applied Deep Learning

Yun-Nung (Vivian) Chen

www.csie.ntu.edu.tw/~yvchen/f105-adl

National Taiwan University

Slide credit from Hung-Yi Lee & Richard Socher

Outline

Data Preprocessing

Activation Function

Loss Function

Optimization
- Adagrad
- Momentum

Generalization
- Early Stopping
- Regularization
- Dropout

Outline

Data Preprocessing

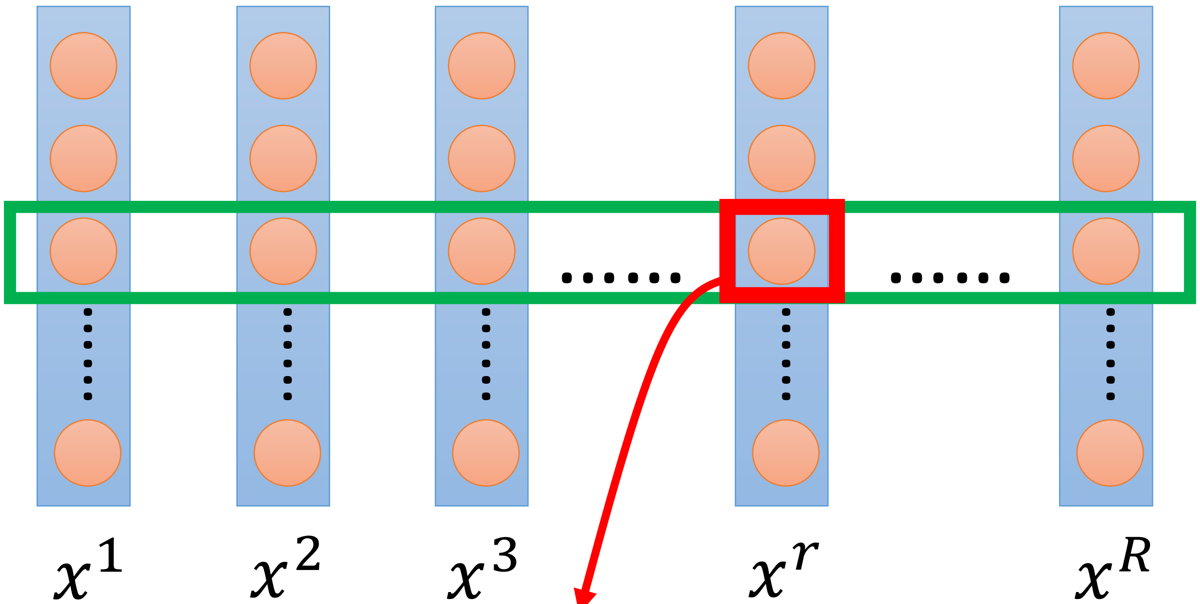
Activation Function

Loss Function

Optimization
- Adagrad
- Momentum

Generalization
- Early Stopping
- Regularization
- Dropout

Input Normalization



For each dimension i:

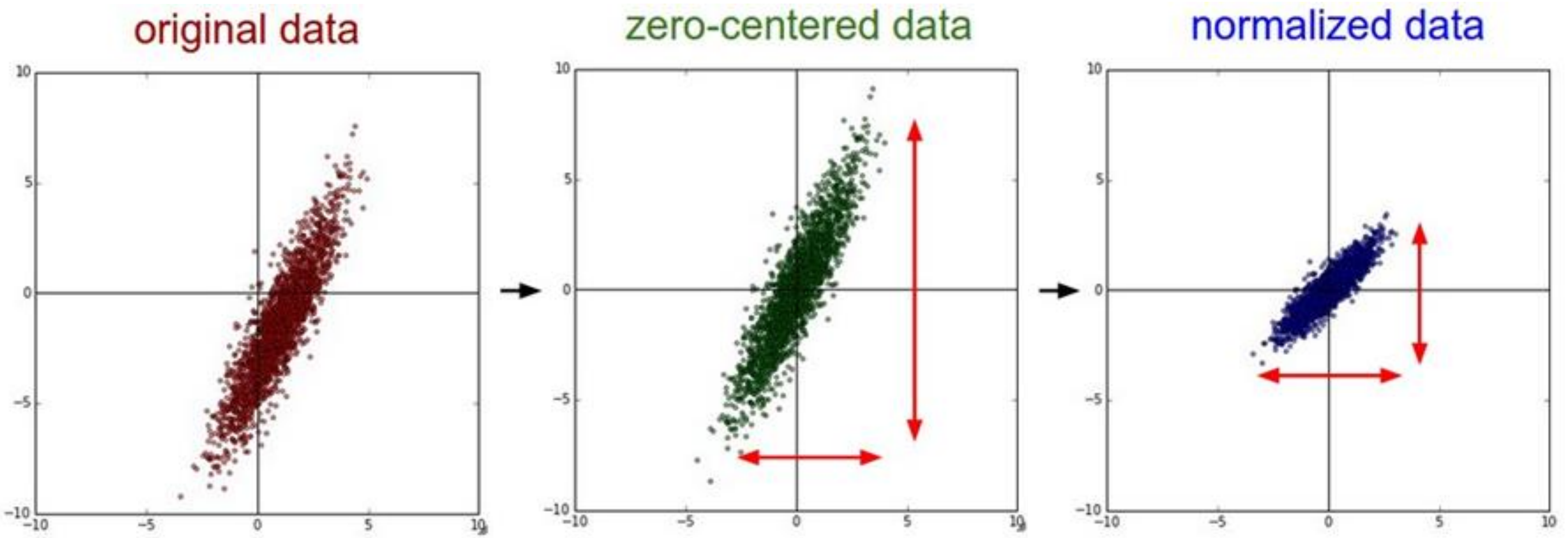
mean: m_i

standard deviation: σ_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

Input Normalization



Normalizing training and testing data in the same way

Outline

Data Preprocessing

Activation Function

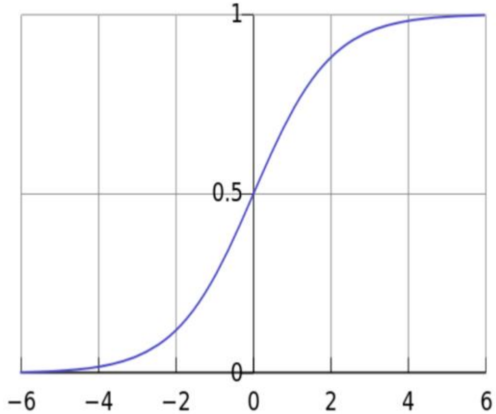
Loss Function

Optimization
- Adagrad
- Momentum

Generalization
- Early Stopping
- Regularization
- Dropout

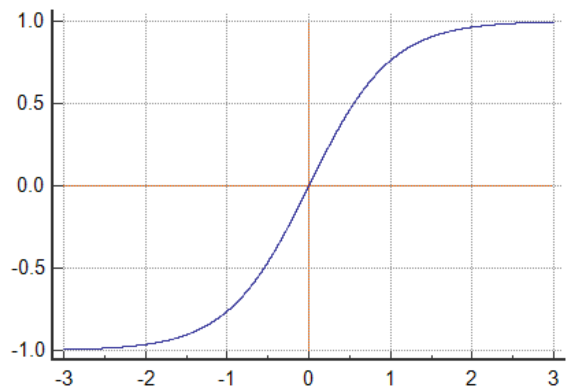
Activation Function

Sigmoid $f(x) = \frac{1}{1+e^{-x}}$



$$f'(x) = f(x)(1 - f(x))$$

Tanh $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$



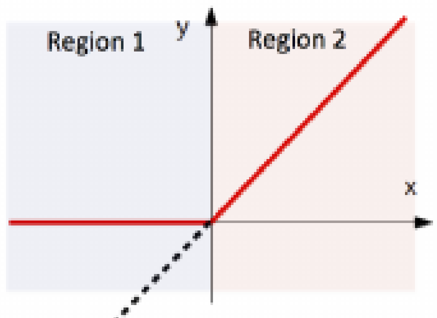
$$f'(x) = 1 - f(x)^2$$

tanh is just a rescaled and shifted sigmoid, but better for many models

- Initialization: values close to 0
- Convergence: faster in practice
- Nice derivative (similar to sigmoid)

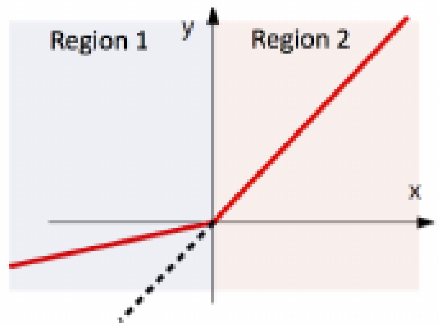
Variants

ReLU



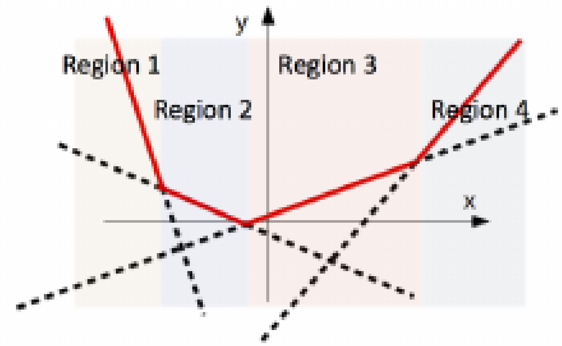
ReLU

LReLU & PReLU



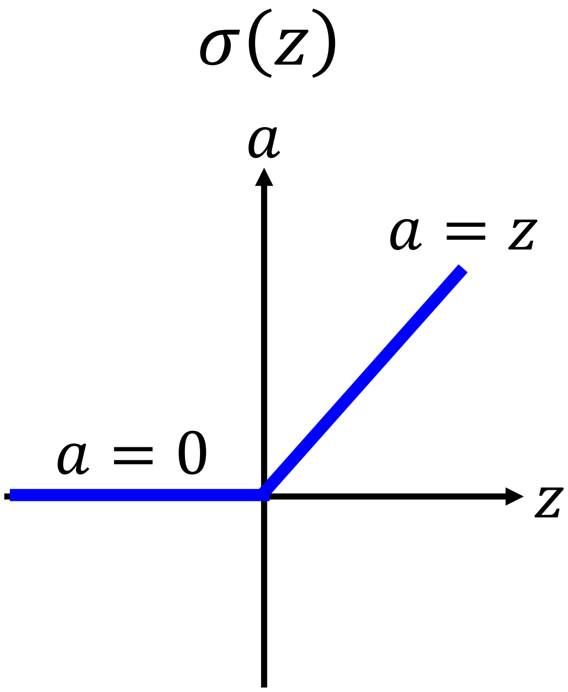
LReLU and PReLU

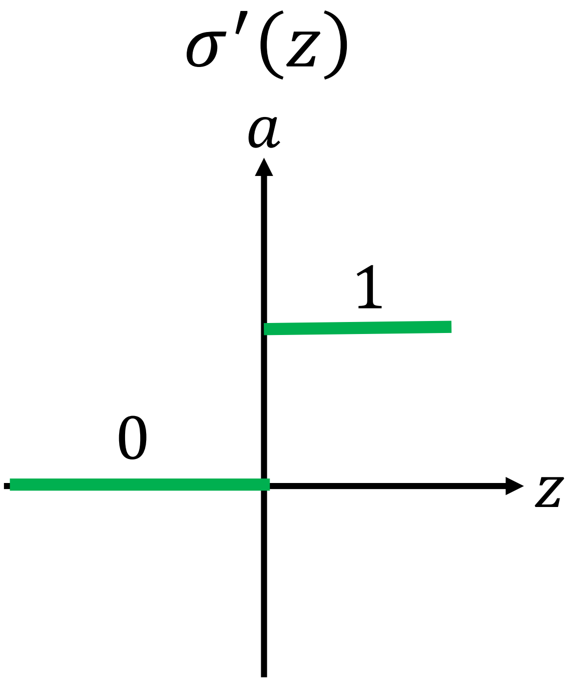
Maxout



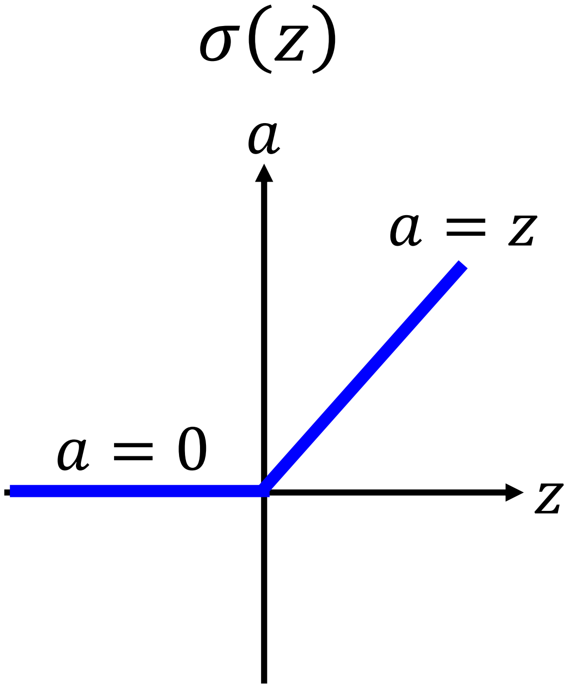
Maxout (k=4)

Rectified Linear Unit (ReLU)





Rectified Linear Unit (ReLU)

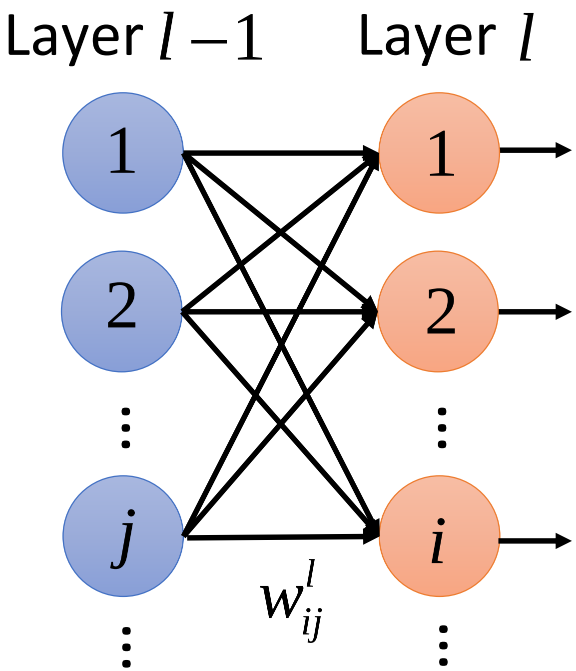


Reason

1. Fast to compute
2. Biological reason
3. Infinite sigmoid with different biases
4. Solution for vanishing gradient

Backpropagation

$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \frac{\partial C(\theta)}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l}$$



δ_i^l Error signal

$$\begin{cases} a_j^{l-1} & l > 1 \\ x_j & l = 1 \end{cases}$$

Backward Pass

$$\delta^L = \sigma'(z^L) \odot \nabla C(y)$$
$$\delta^{L-1} = \sigma'(z^{L-1}) \odot (W^L)^T \delta^L$$
$$\vdots$$
$$\delta^l = \sigma'(z^l) \odot (W^{l+1})^T \delta^{l+1}$$
$$\vdots$$

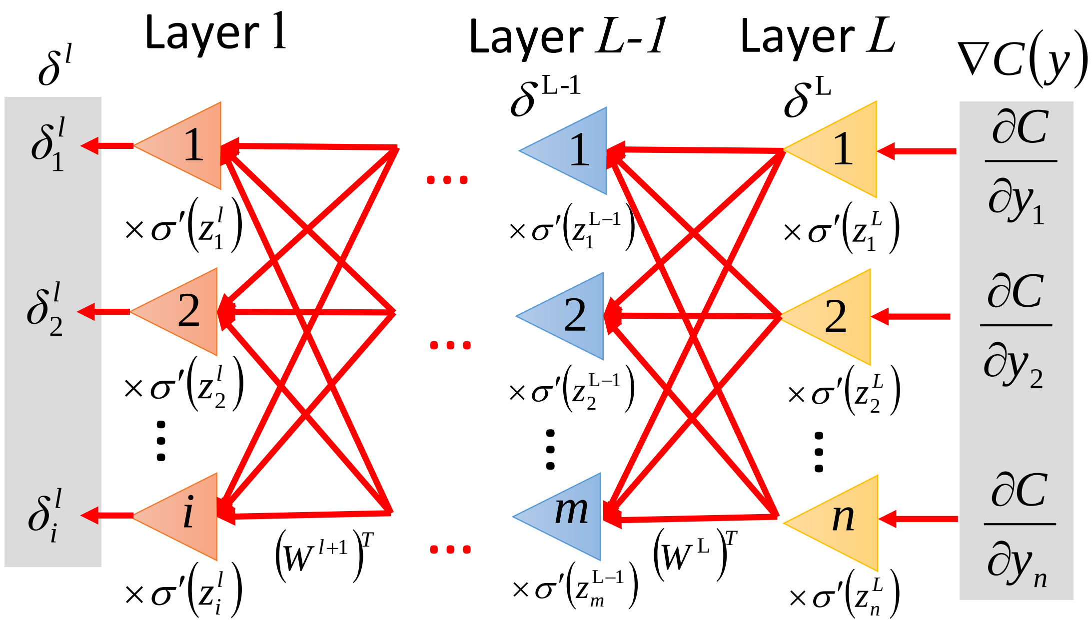
Forward Pass

$$z^1 = W^1 x + b^1$$
$$a^1 = \sigma(z^1)$$
$$\vdots$$
$$z^l = W^l a^{l-1} + b^l$$
$$a^l = \sigma(z^l)$$
$$\vdots$$

Backpropagation

$$\frac{\partial C(\theta)}{\partial w_{ij}^{l}} = \frac{\partial C(\theta)}{\partial z_i^{l}} \frac{\partial z_i^{l}}{\partial w_{ij}^{l}}$$

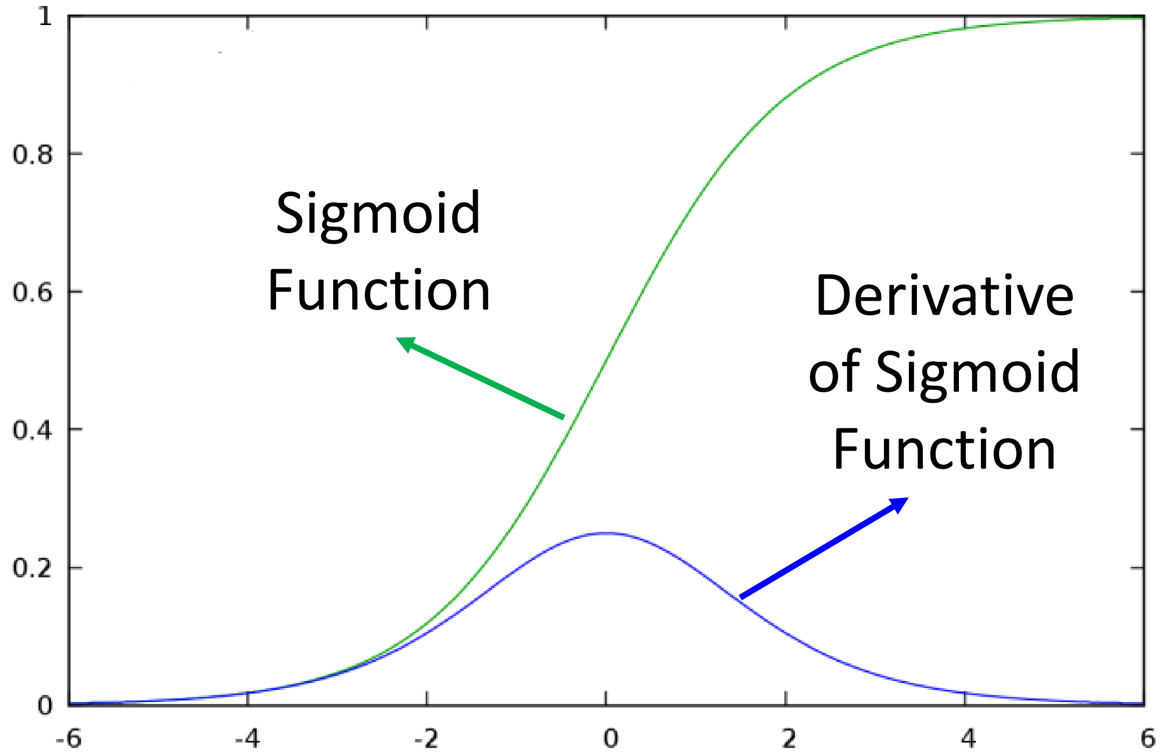
δ_i^l Error signal



Backward Pass

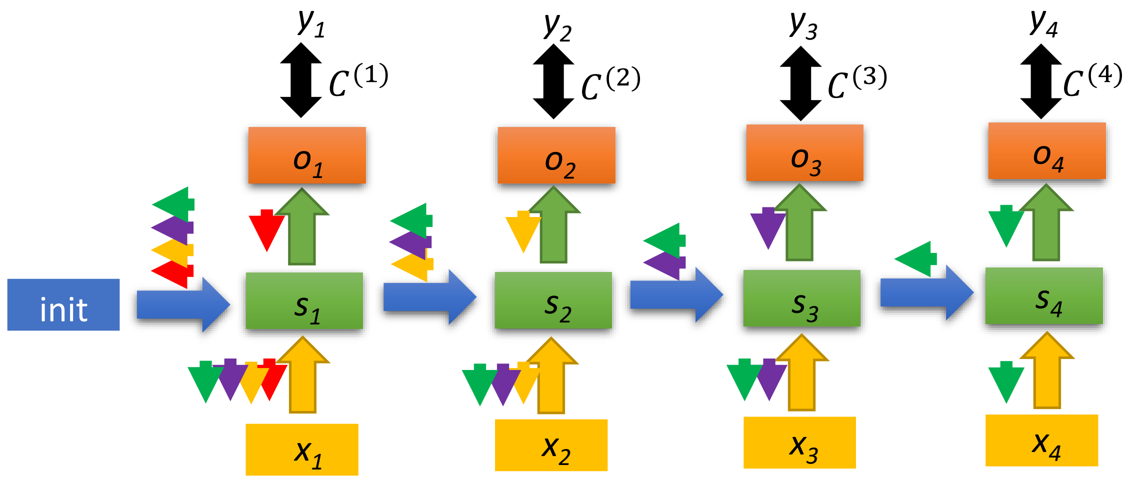
$$\delta^L = \sigma'(z^L) \odot \nabla C(y)$$
$$\delta^{L-1} = \sigma'(z^{L-1}) \odot (W^L)^T \delta^L$$
$$\vdots$$
$$\delta^l = \sigma'(z^l) \odot (W^{l+1})^T \delta^{l+1}$$
$$\vdots$$

Sigmoid Issue



Derivative of the sigmoid function is always smaller than 1

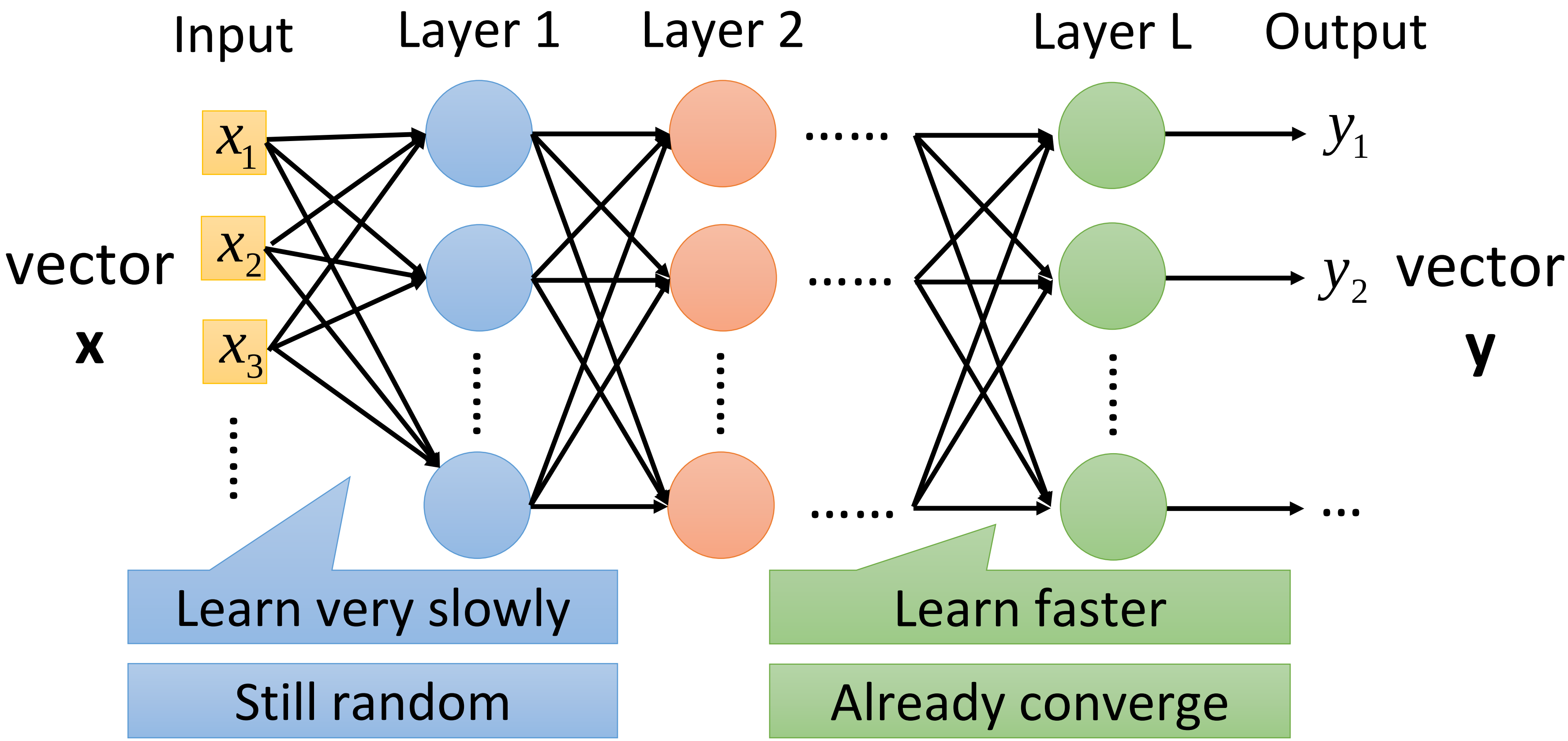
Vanishing Gradient Problem



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

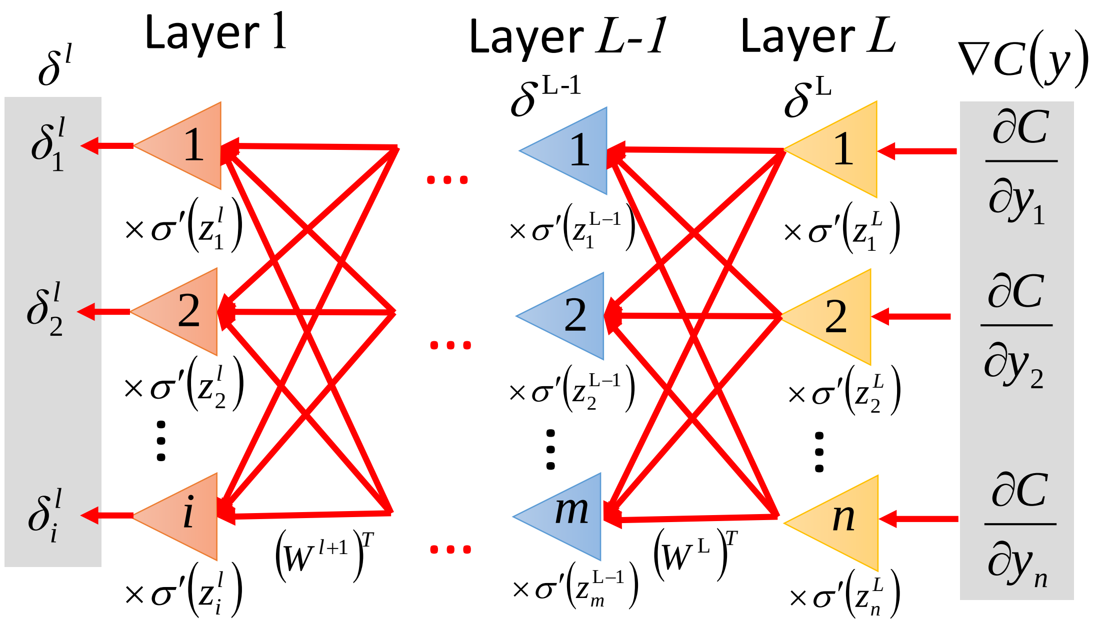
The error signal is getting smaller and smaller due to $\sigma'(z) < 1$
→ **vanishing gradient**

Vanishing Gradient Problem

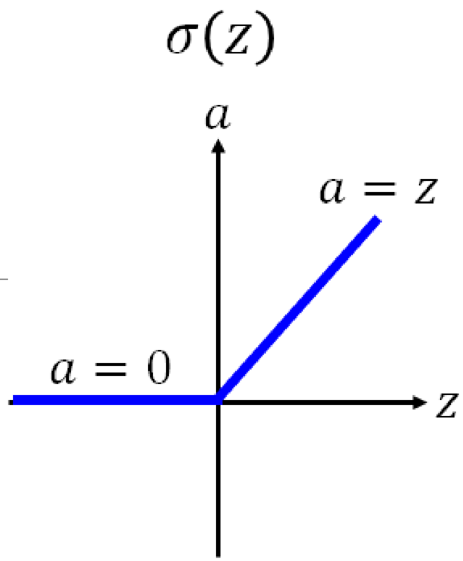


The weights are converged based on random!?

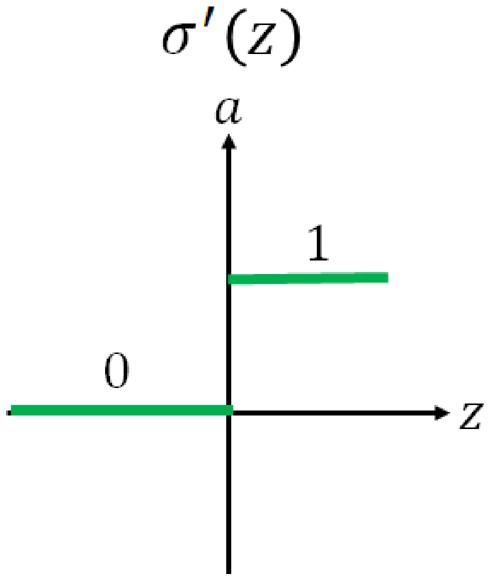
ReLU



Layer l
Layer L-1
Layer L
δ^l
δ_1^l
δ_2^l
δ_i^l
δ^{L-1}
δ^L
$\nabla C(y)$
$\frac{\partial C}{\partial y_1}$
$\frac{\partial C}{\partial y_2}$
$\frac{\partial C}{\partial y_n}$
$\times\sigma'(z_1^l)$
$\times\sigma'(z_2^l)$
$\times\sigma'(z_i^l)$
$\times\sigma'(z_1^{L-1})$
$\times\sigma'(z_2^{L-1})$
$\times\sigma'(z_m^{L-1})$
$\times\sigma'(z_1^L)$
$\times\sigma'(z_2^L)$
$\times\sigma'(z_n^L)$
$(W^{l+1})^T$
$(W^L)^T$

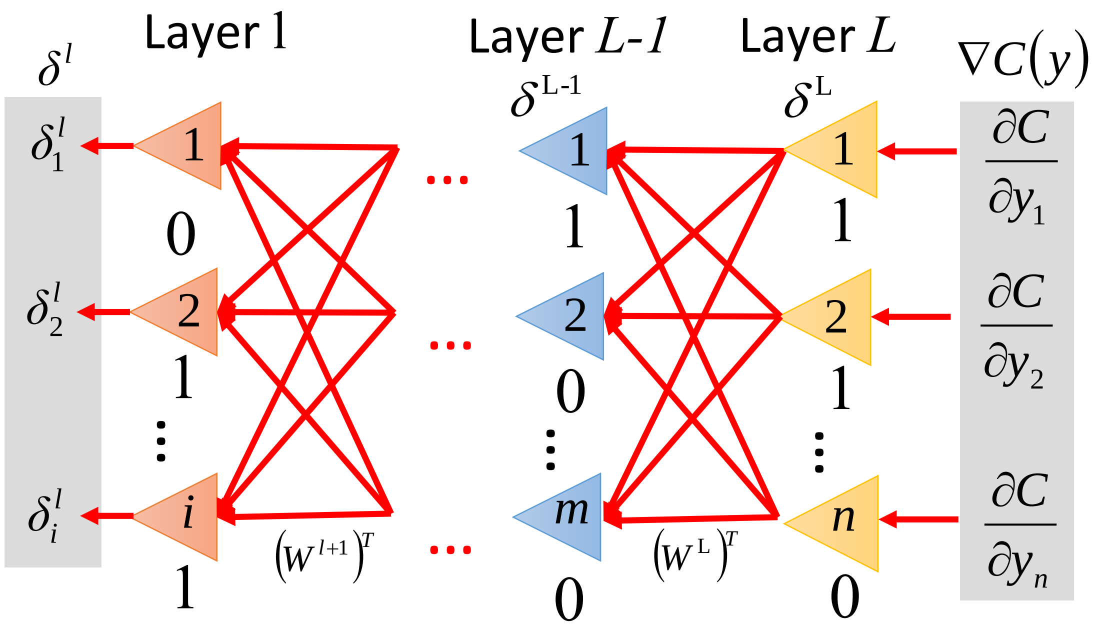


$\sigma(z)$
a
$a = z$
$a = 0$
z

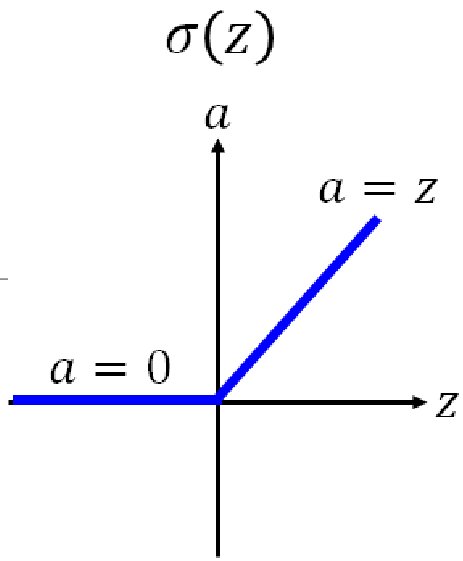


$\sigma'(z)$
a
1
0
z

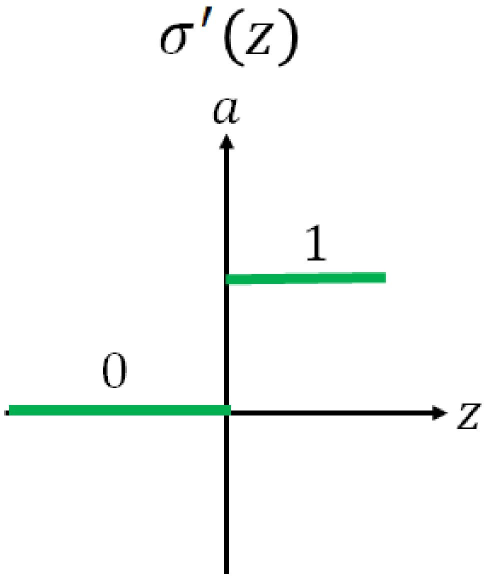
ReLU



Layer l
Layer L-1
Layer L
$\nabla C(y)$
δ^l
δ^{L-1}
δ^L
δ_1^l
δ_2^l
δ_i^l
$\frac{\partial C}{\partial y_1}$
$\frac{\partial C}{\partial y_2}$
$\frac{\partial C}{\partial y_n}$
$(W^{l+1})^T$
$(W^L)^T$

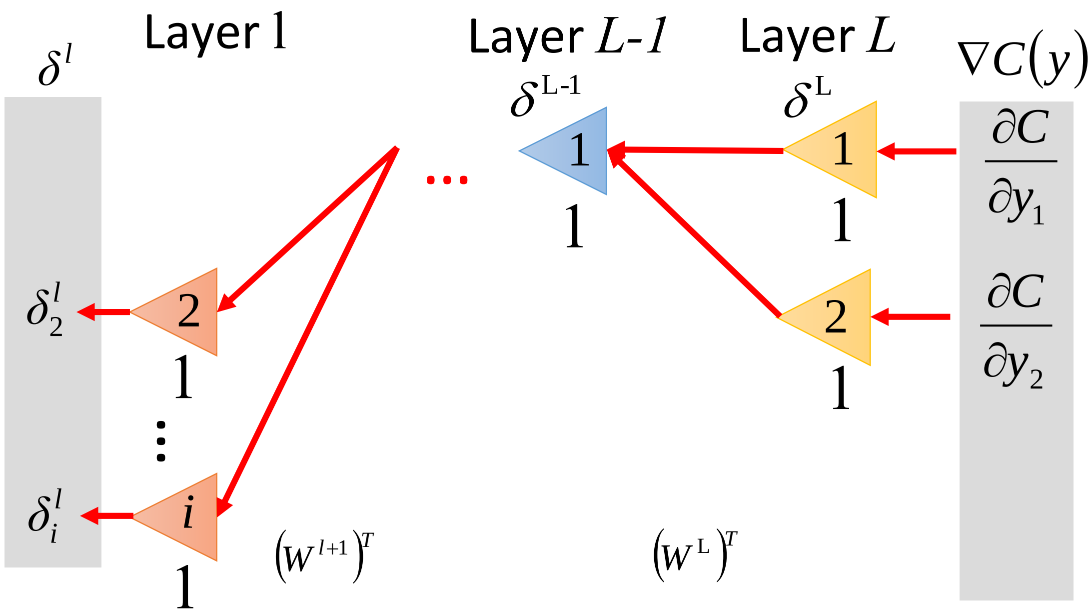


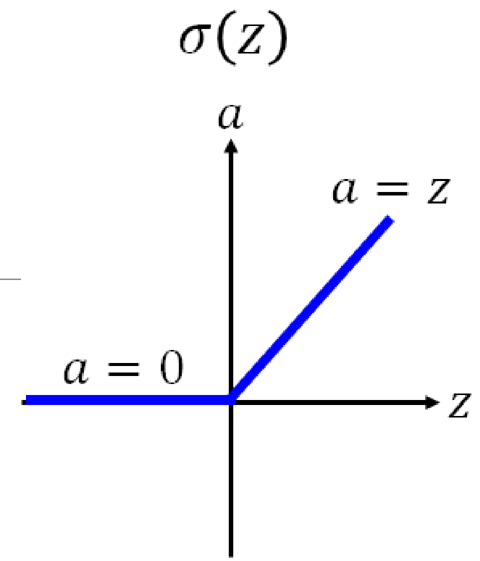
$\sigma(z)$
a
$a = z$
$a = 0$
z

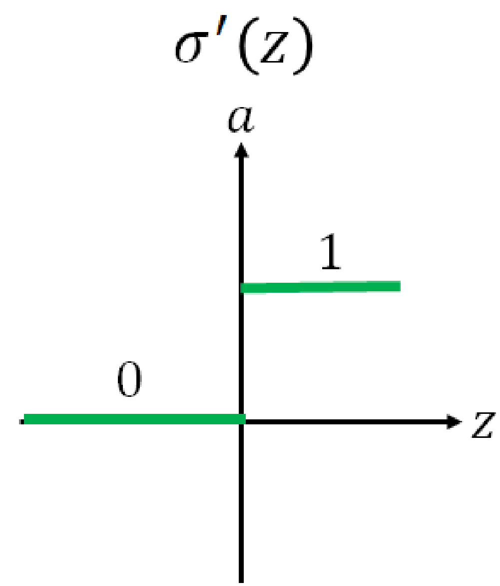


$\sigma'(z)$
a
1
0
z

ReLU

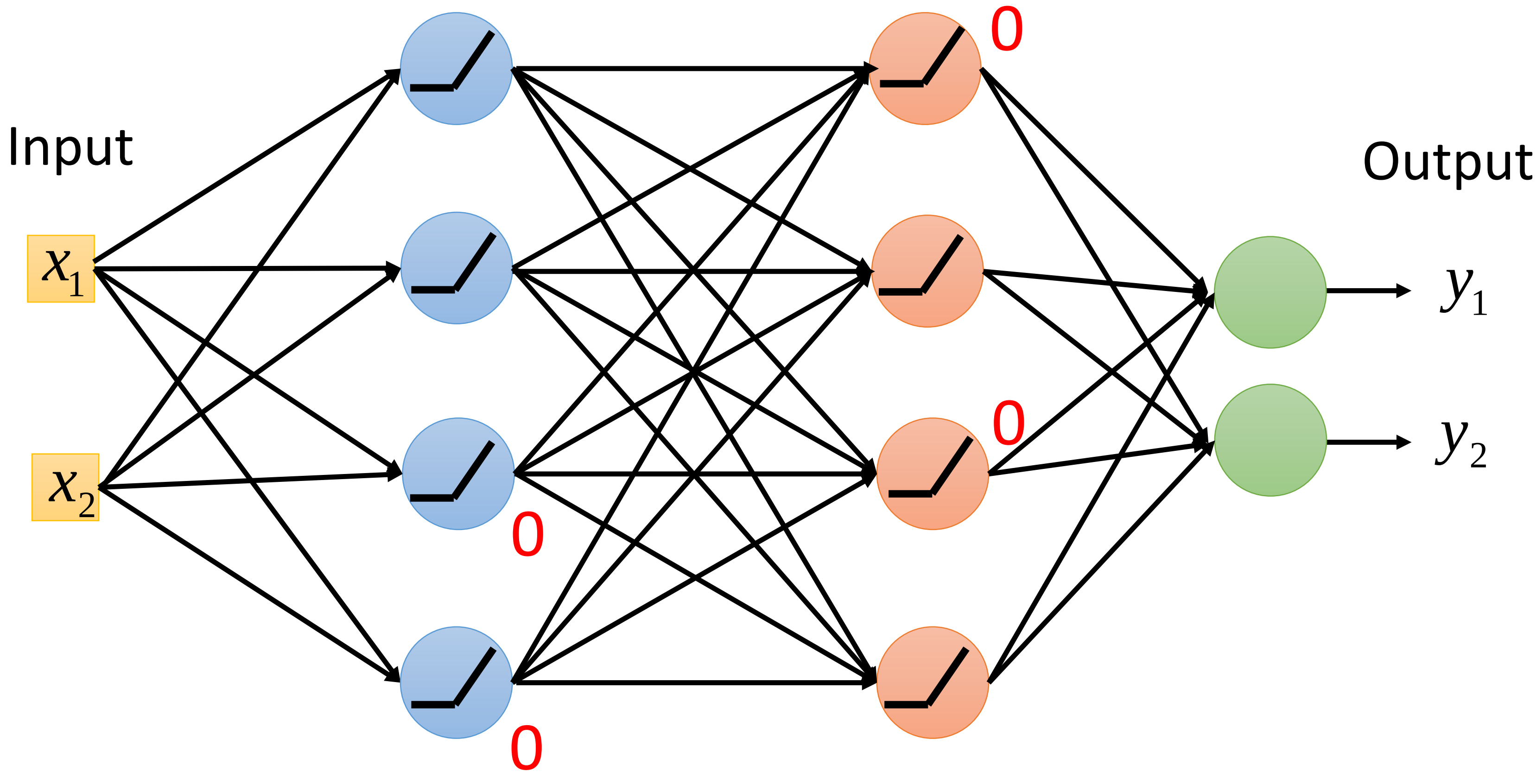




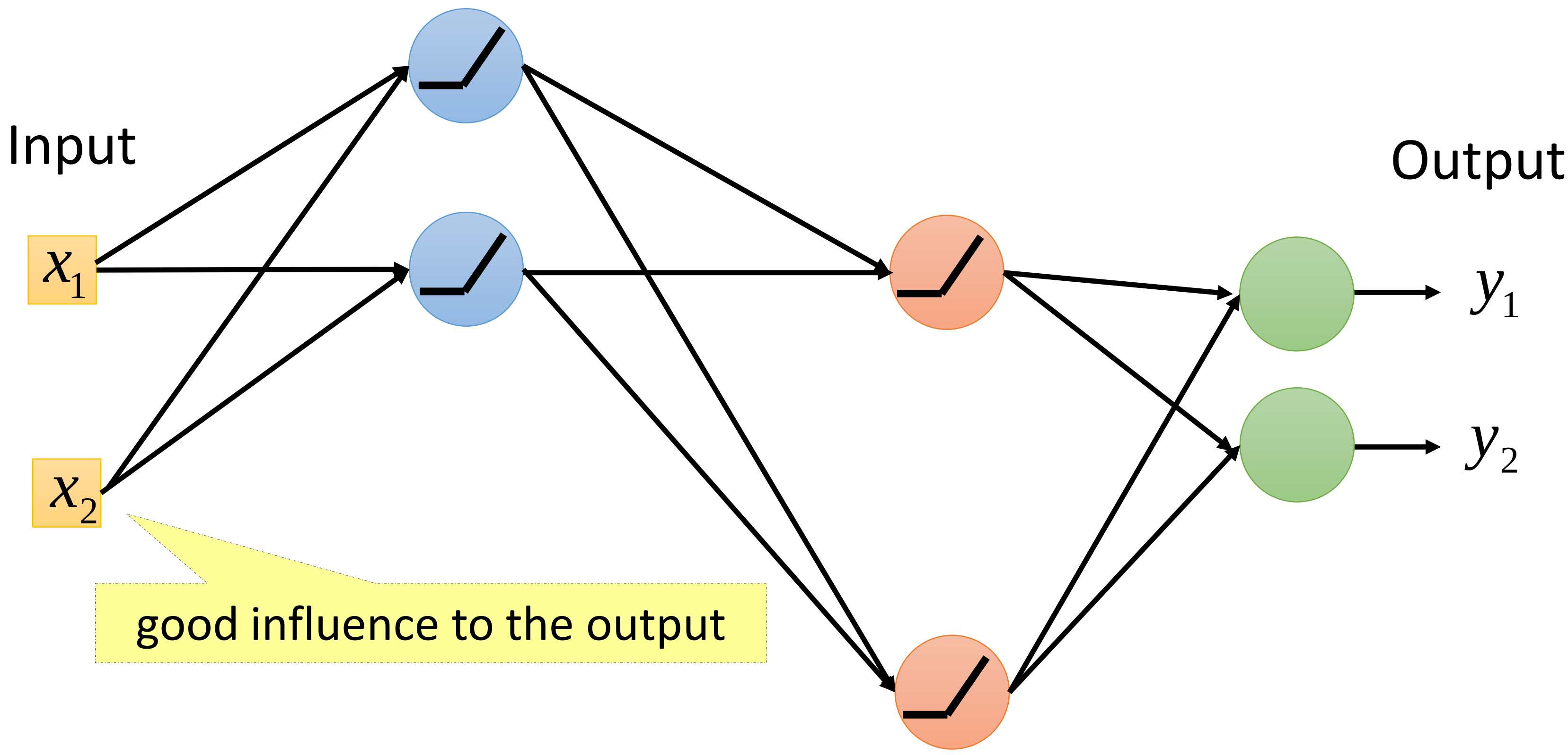


A thinner network without any attenuation

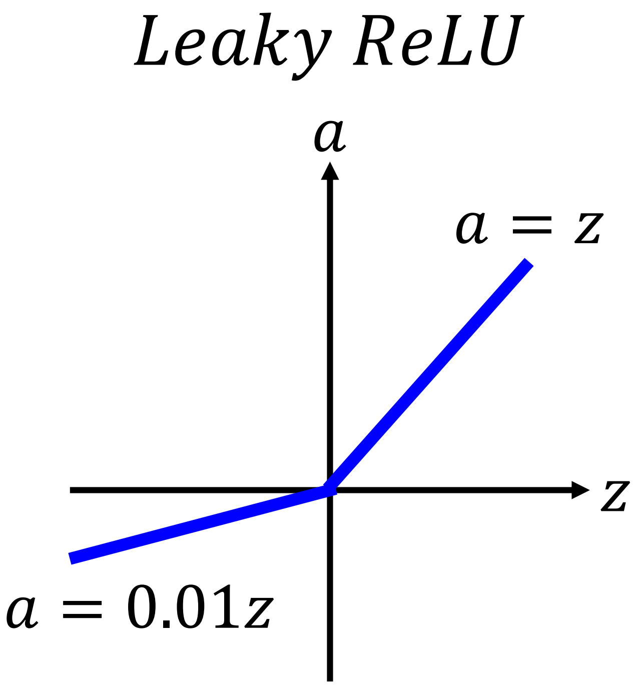
ReLU – Forward Pass

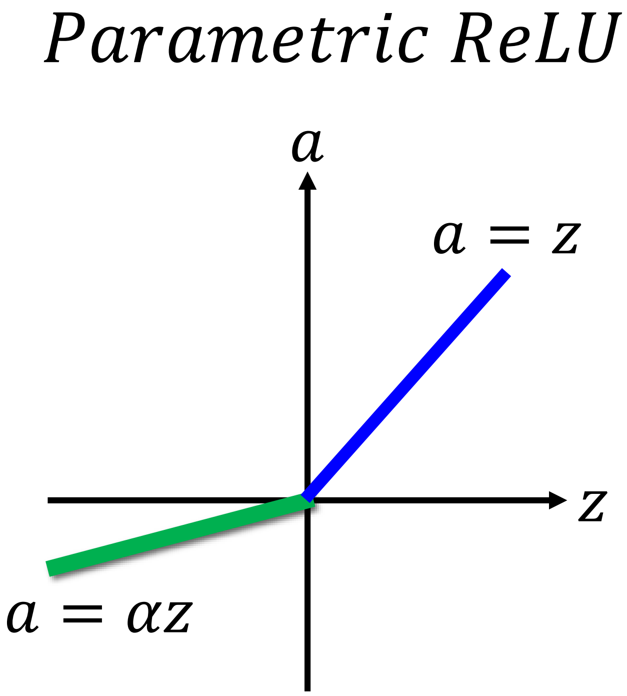


ReLU – Backward Pass



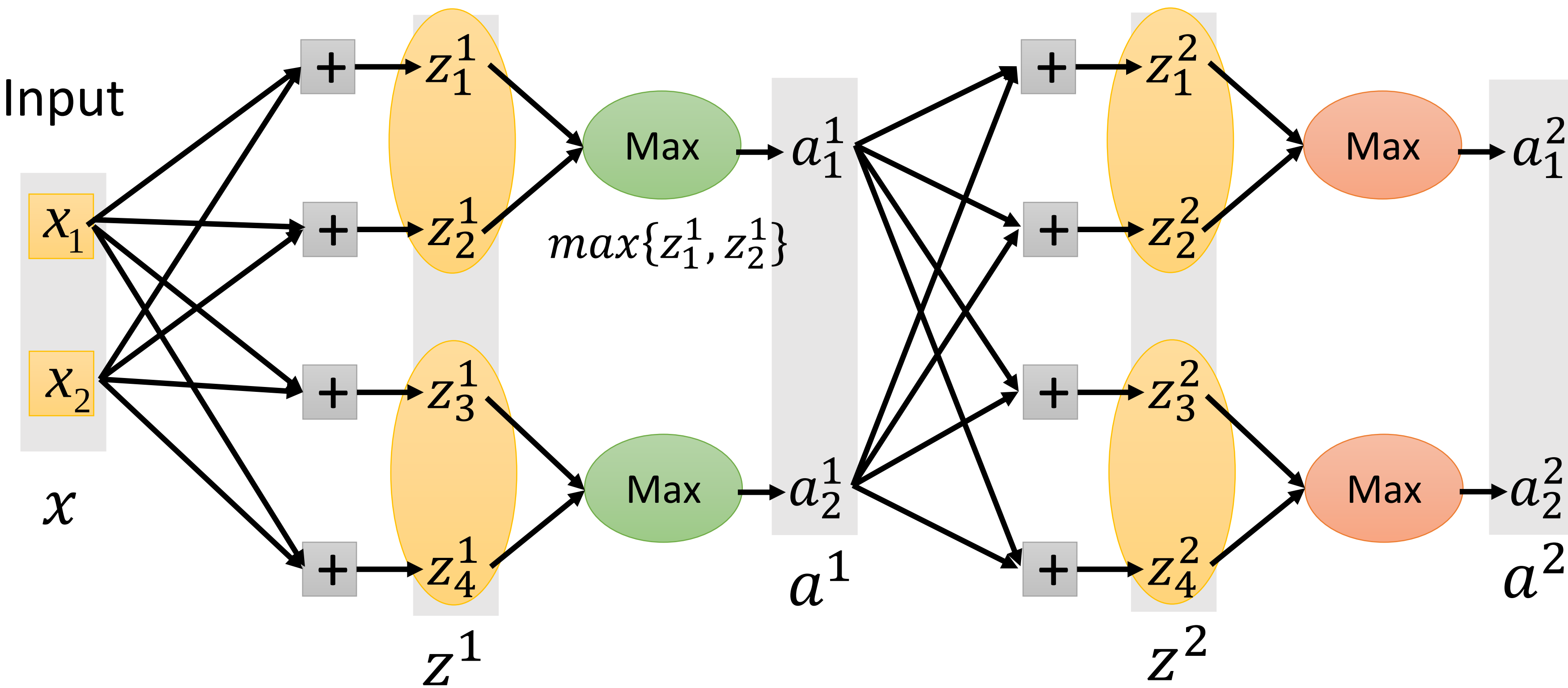
Variant ReLU





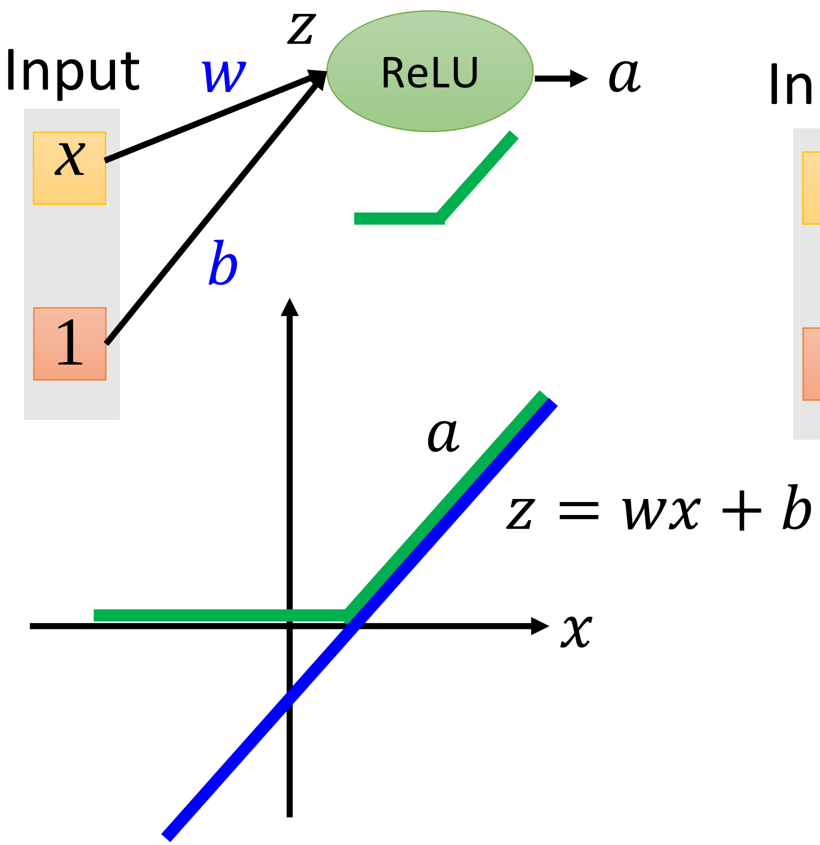
α is also learned by gradient descent

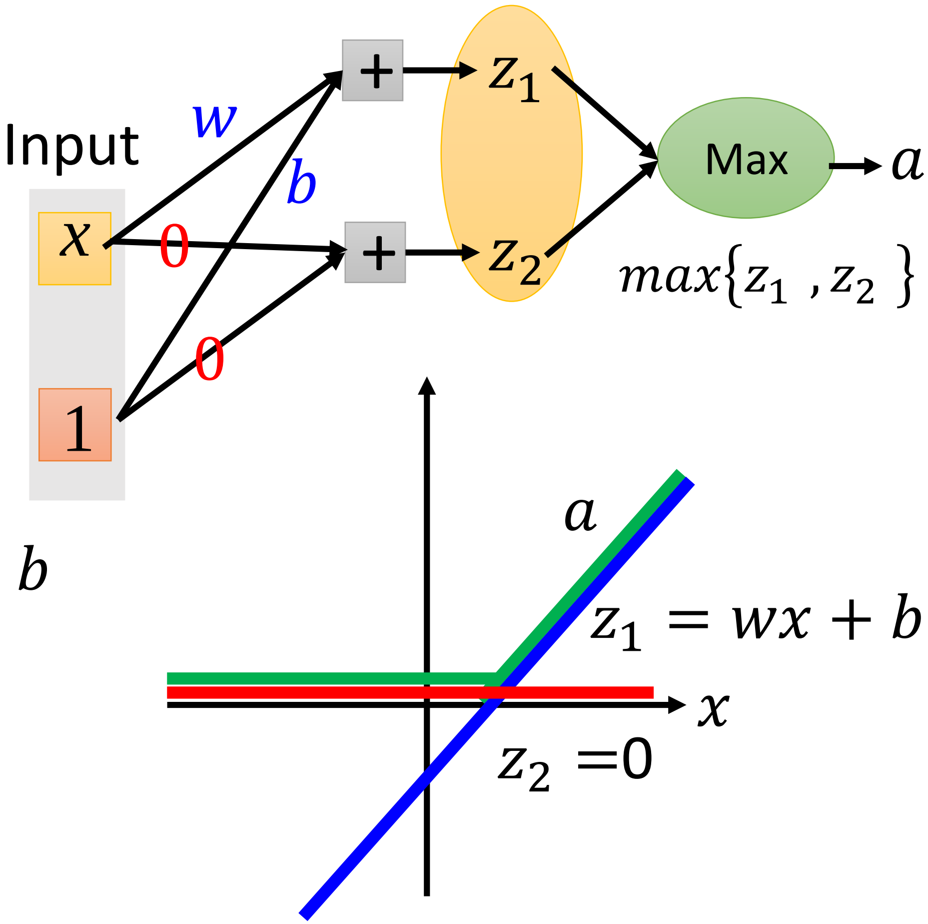
Maxout



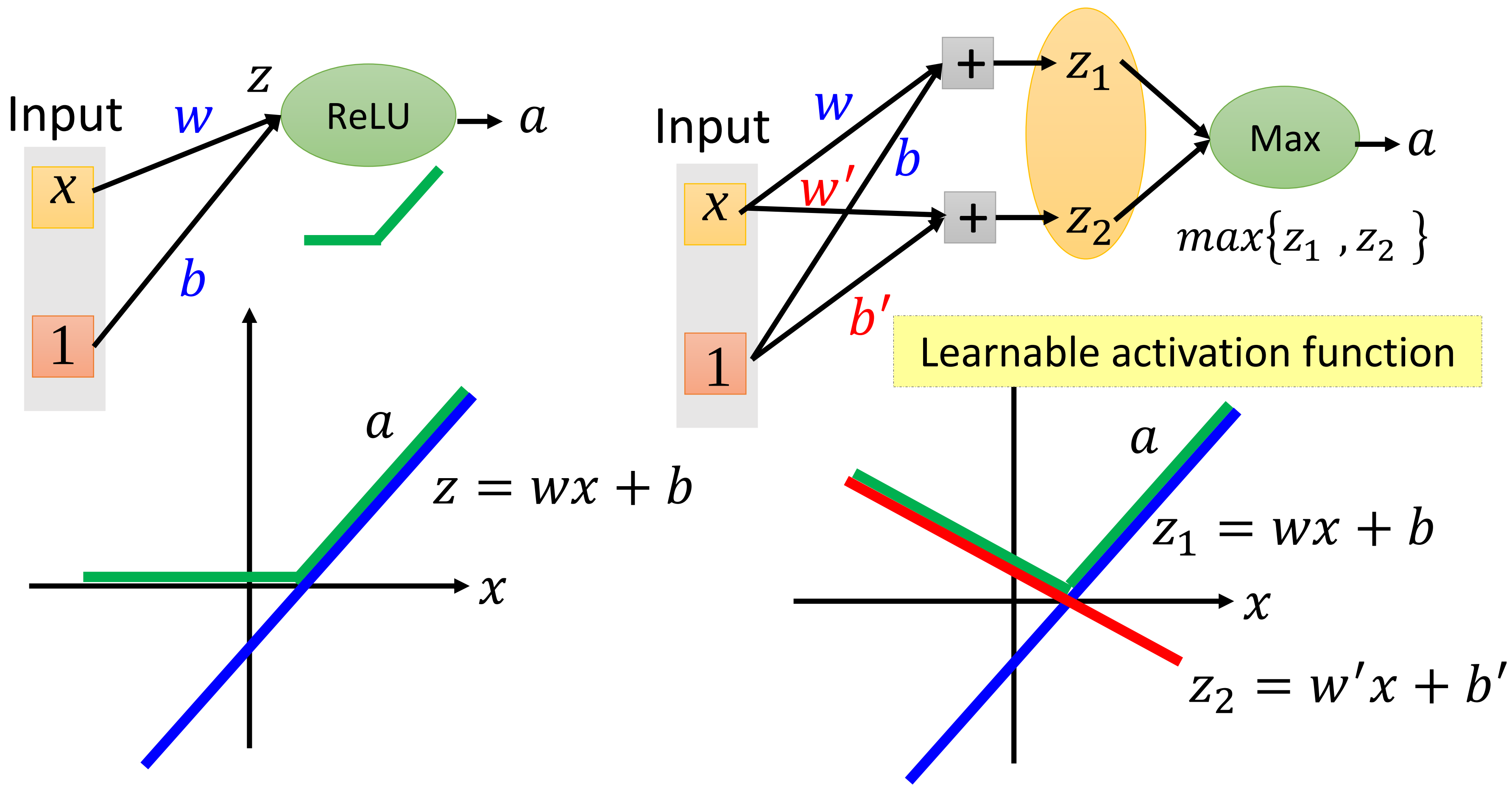
$z^1 = W^1 x$ $\quad$ $z^2 = W^2 a^1$

Maxout – ReLU is a special case



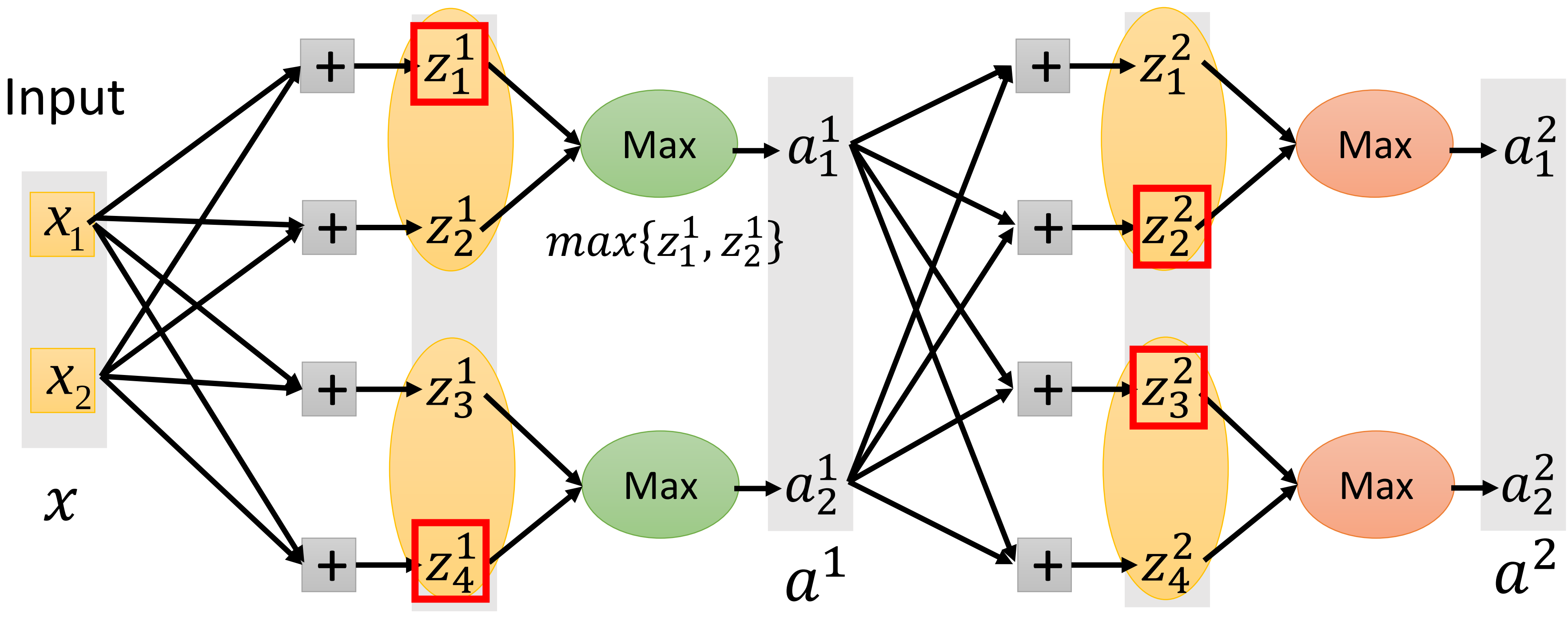


Maxout – ReLU is a special case



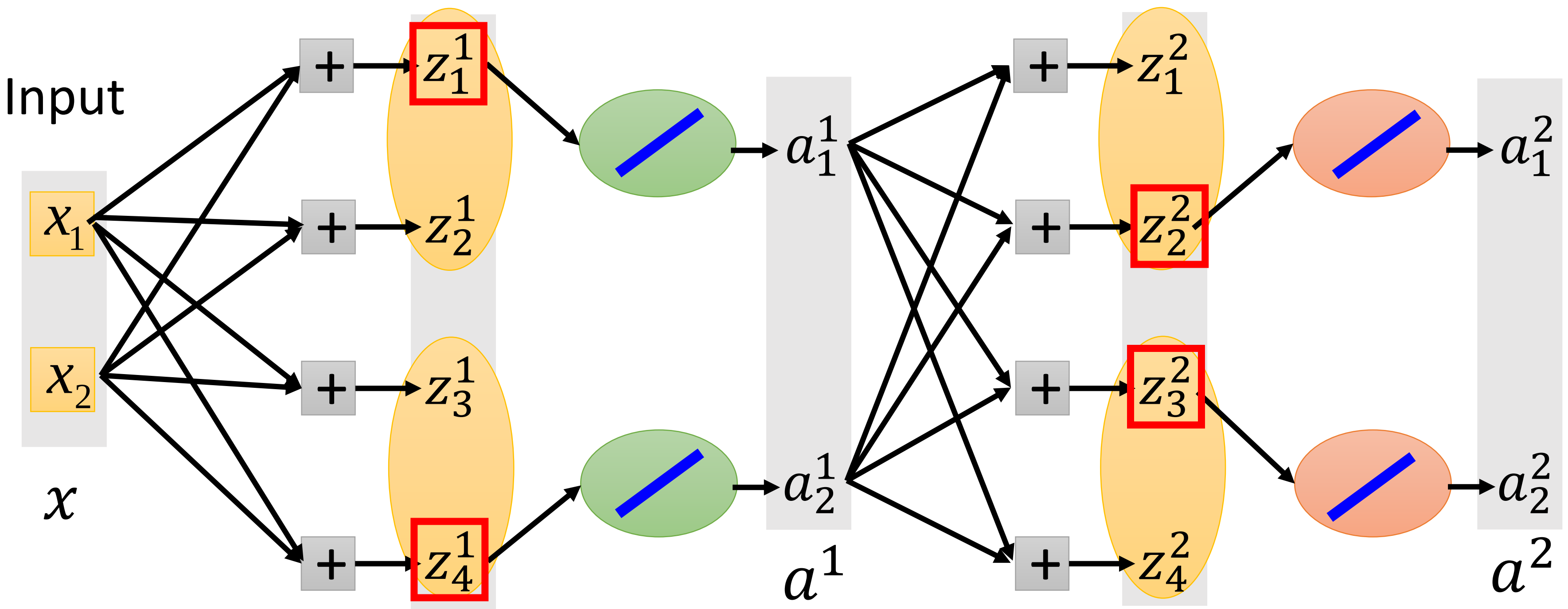
Maxout - Training

Given training data x, we decide z for maxout



Maxout - Training

Given training data x, we decide z for maxout



Training this thin and linear network

Outline

Data Preprocessing

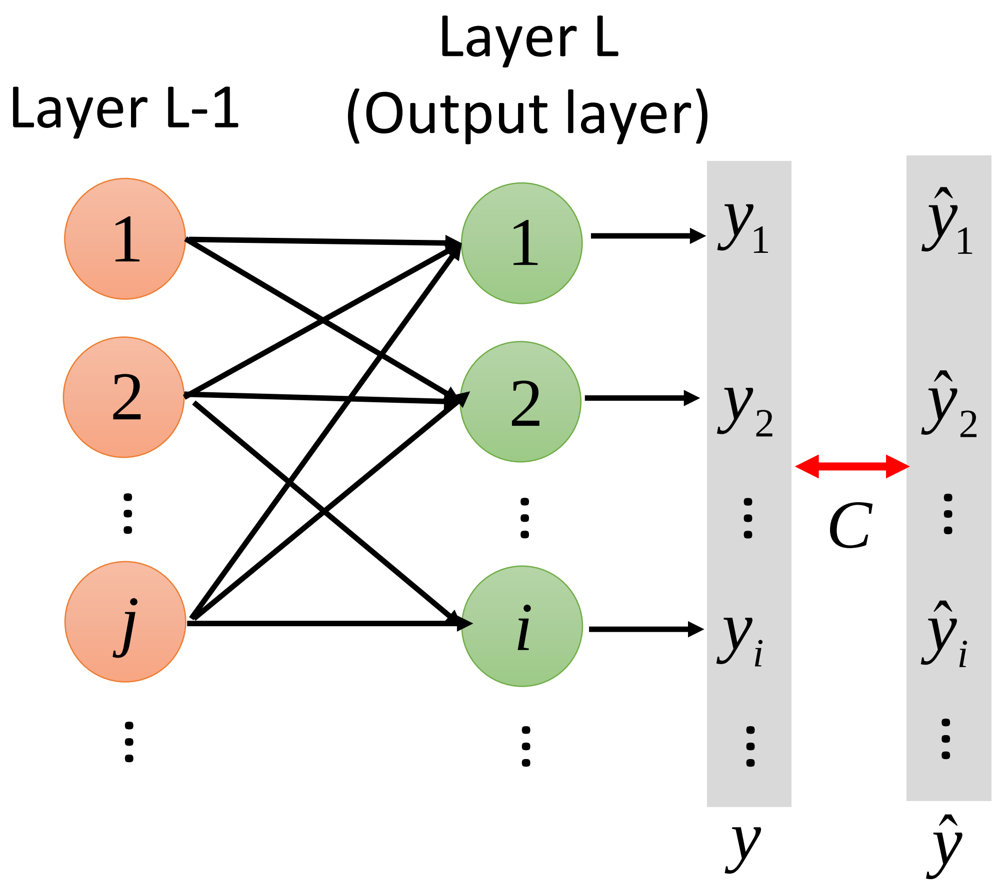
Activation Function

Loss Function

Optimization
- Adagrad
- Momentum

Generalization
- Early Stopping
- Regularization
- Dropout

Loss Function – Square Error



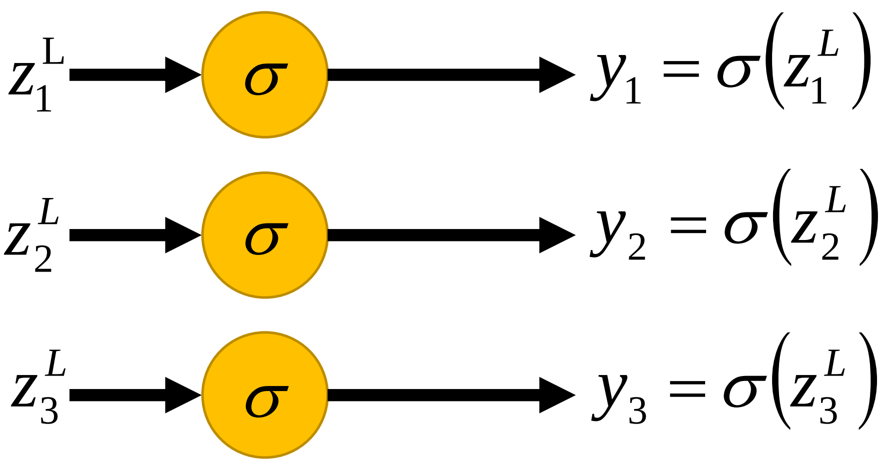
$$C = \frac{1}{2}\left\|y - \hat{y}\right\|^2$$

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

Softmax

Softmax layer as the output layer

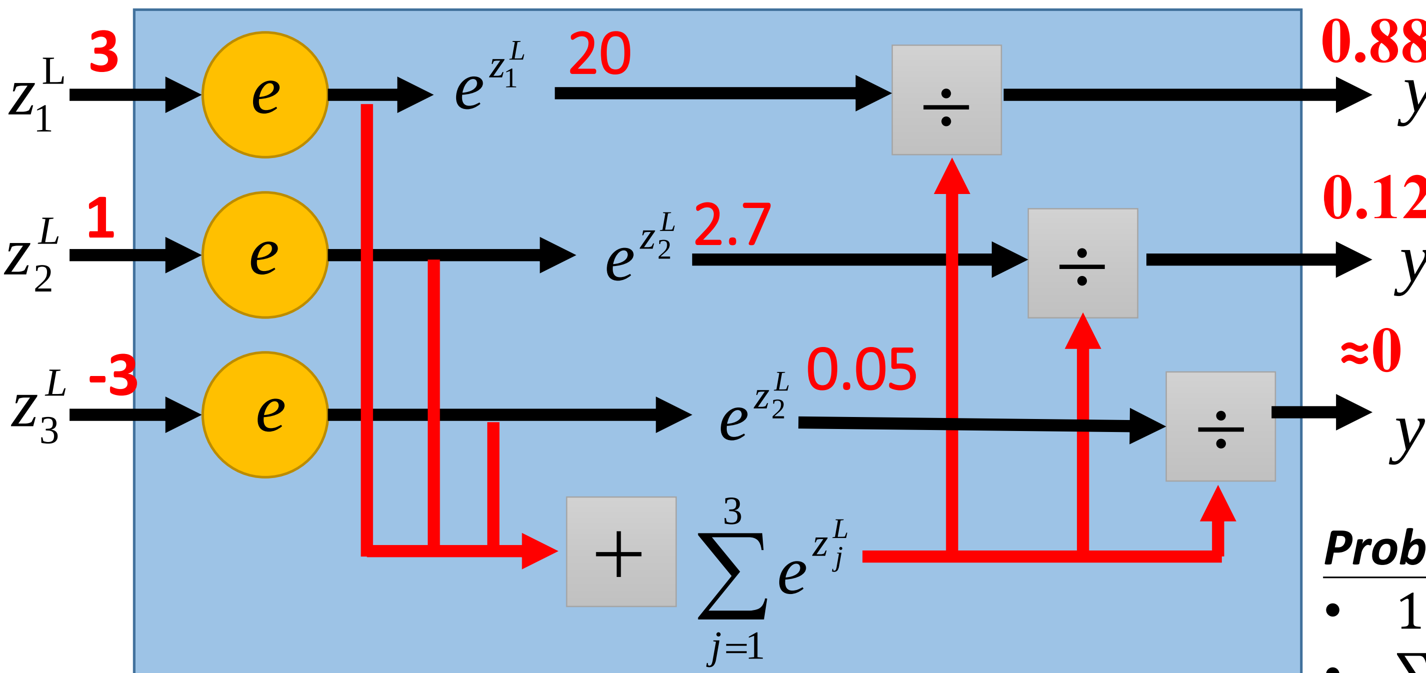
<u>Ordinary Output layer</u>



Softmax

Softmax layer as the output layer

Softmax Layer



$$y_1 = e^{z_1^L} \Big/ \sum_{j=1}^{3} e^{z_j^L}$$

$$y_2 = e^{z_2^L} \Big/ \sum_{j=1}^{3} e^{z_j^L}$$

$$y_3 = e^{z_3^L} \Big/ \sum_{j=1}^{3} e^{z_j^L}$$

Probability:

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

Training labels indicate positive and negative samples in stead of the actual values

Outline

Data Preprocessing

Activation Function

Loss Function

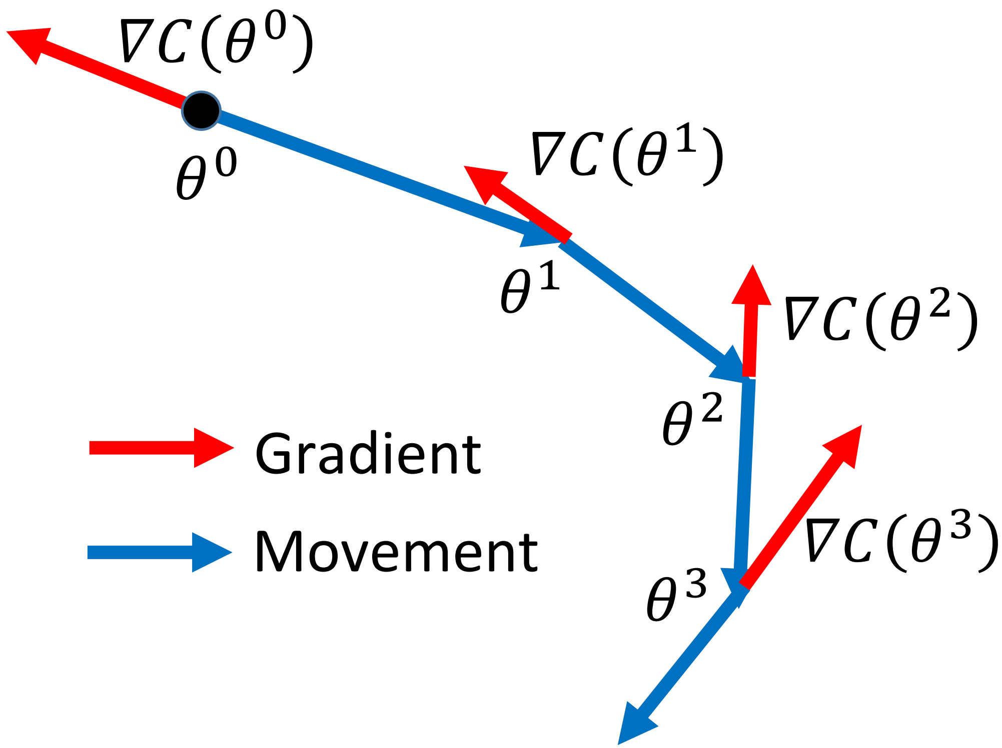
Optimization

- Adagrad
- Momentum

Generalization

- Early Stopping
- Regularization
- Dropout

Gradient Descent for Optimization



Algorithm

Initialization: start at θ^0

while($\theta^{(i+1)} \neq \theta^i$)

{

compute gradient at θ^i

update parameters

$$\theta^{i+1} \leftarrow \theta^i - \eta \nabla_\theta C(\theta^i)$$

}

1) How to determine the learning rates → learning rate
2) How to avoid stucking at local minima or saddle points → learning direction

Outline

Data Preprocessing

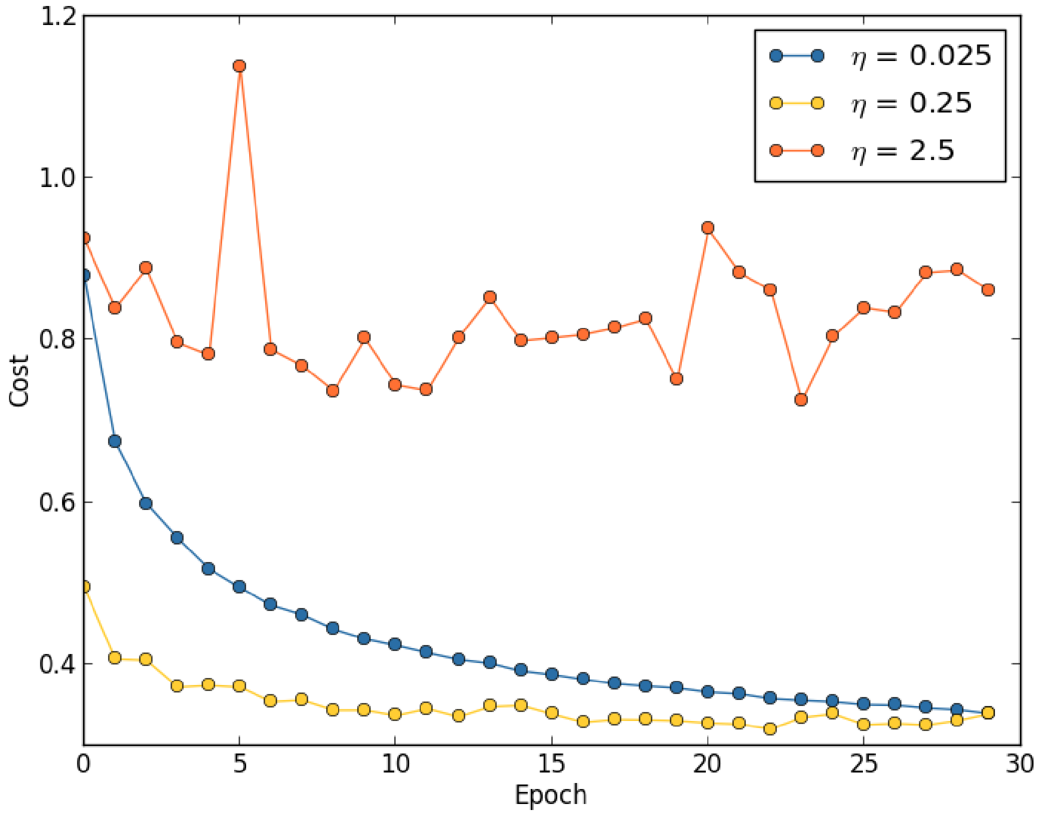
Activation Function

Loss Function

Optimization
- **Adagrad**
- Momentum

Generalization
- Early Stopping
- Regularization
- Dropout

Learning Rate



The proper learning rate is important to find the optimal point

Learning Rate

Idea: reduce the learning rate every few epochs

- At the beginning, we are far from the destination, so we use a larger learning rate
- After several epochs, we are close to the destination, so we reduce the learning rate

Manually set learning rate

1) Reduce by 0.5 when validation error stops improving
2) 1/t decay: $\eta^t = \eta / \sqrt{t+1}$ due to theoretical convergence guarantees

Learning rate cannot be one-size-fits-all
→ different parameters have different learning rates

Adagrad

Idea: adaptive learning rates for each parameter

Approach: divide the learning rate of each parameter by the ***root mean square of its previous derivatives***

w is a parameter

Vanilla Gradient Descent

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

1/t decay

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

Adagrad

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

σ^t: root mean square of the previous derivatives of parameter w
→ Parameter dependent

σ^t: root mean square of the previous derivatives of parameter w
→ Parameter dependent

Adagrad

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

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

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

⋮

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

Adagrad

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

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

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

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

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

Learning rate is adapting differently for each parameter and rare parameters get larger updates than frequently occurring parameters

Outline

Data Preprocessing

Activation Function

Loss Function

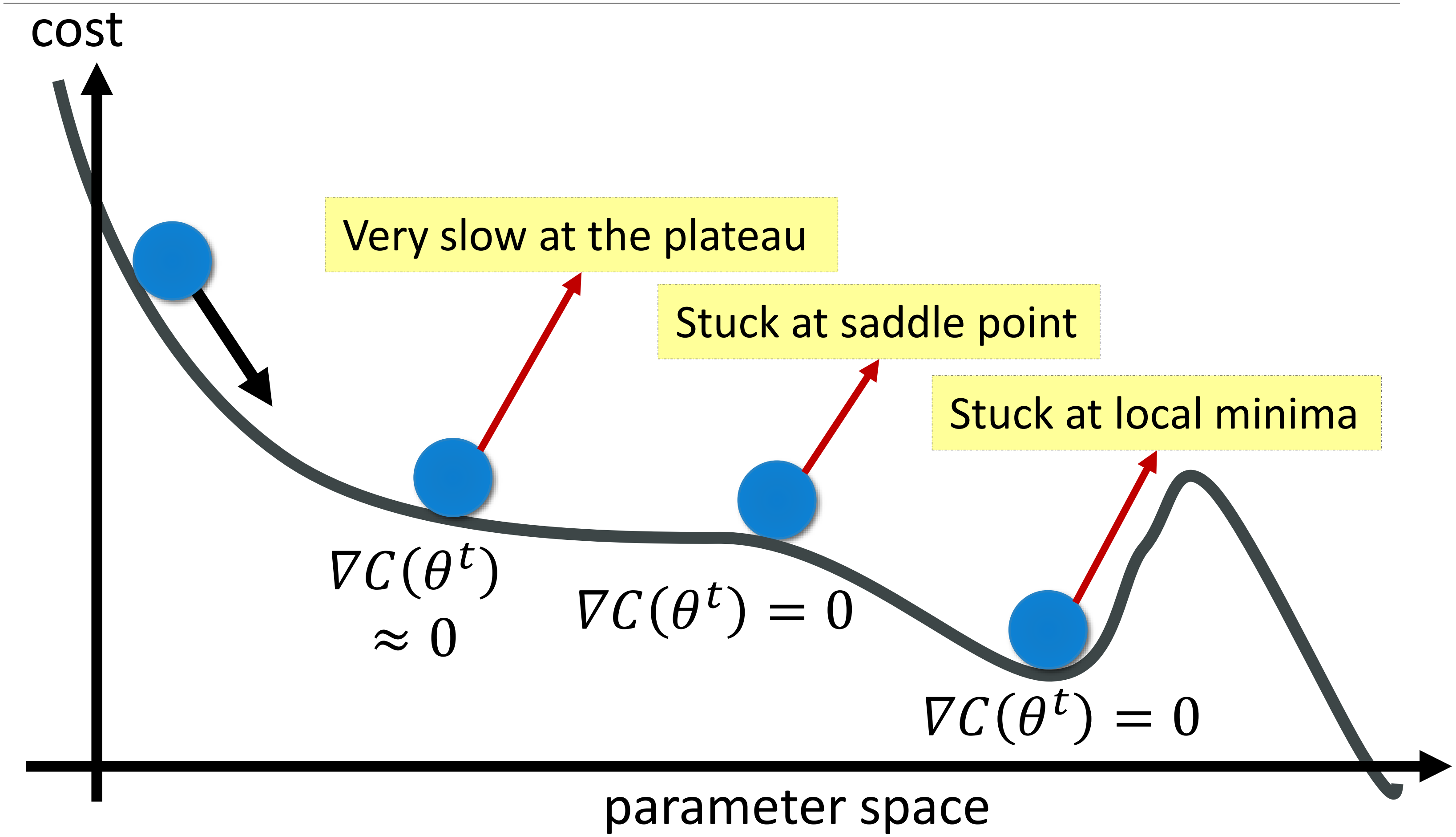
Optimization

- Adagrad
- **Momentum**

Generalization

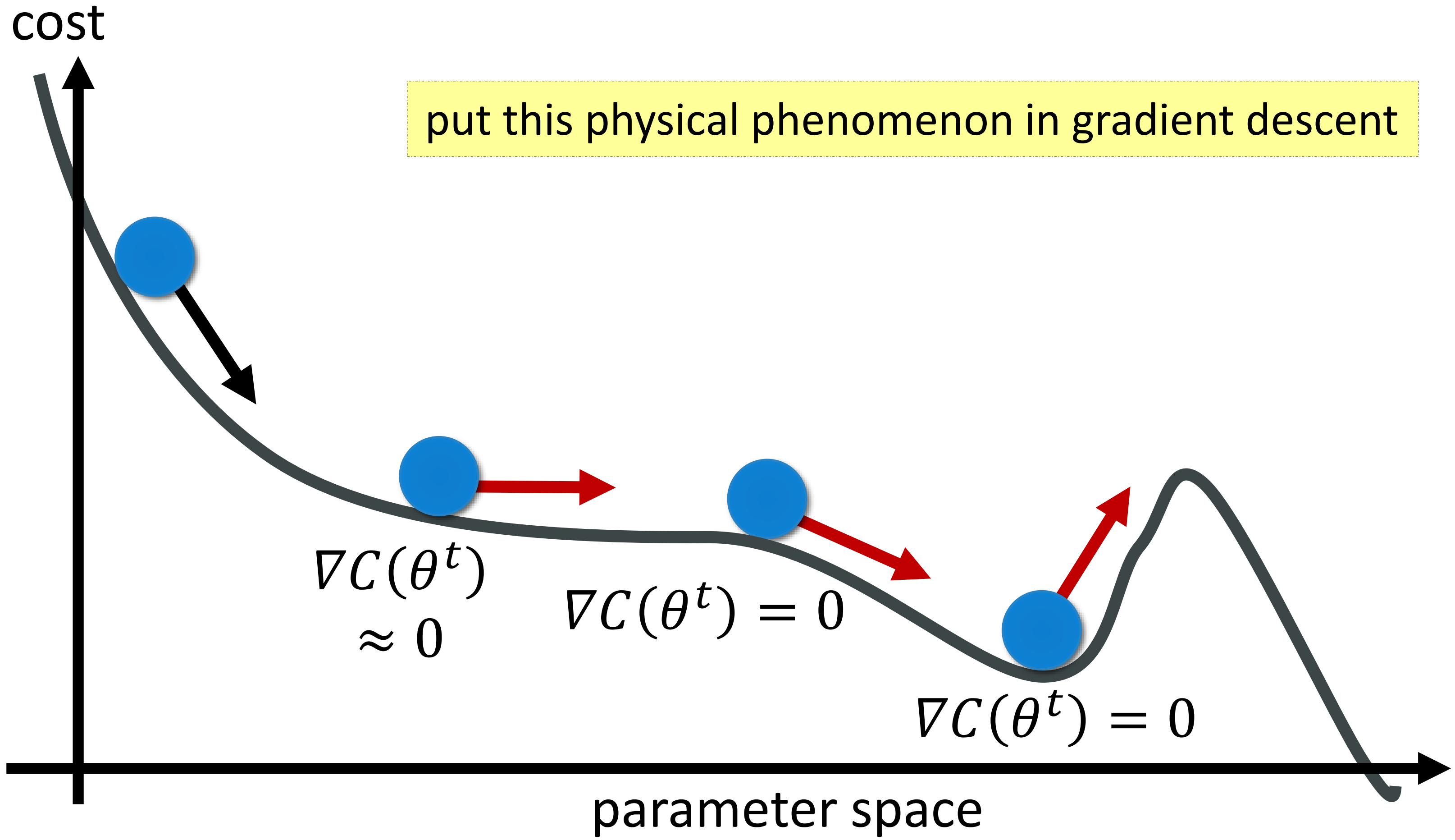
- Early Stopping
- Regularization
- Dropout

Gradient Descent Stuck Issue



Momentum

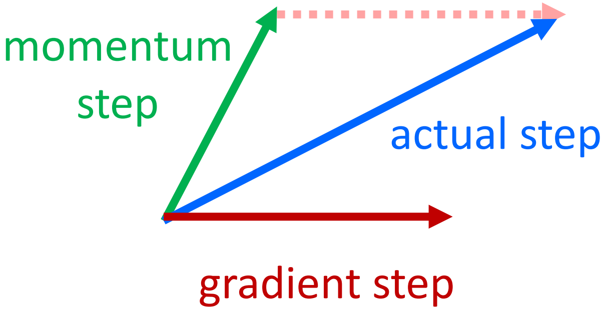
put this physical phenomenon in gradient descent



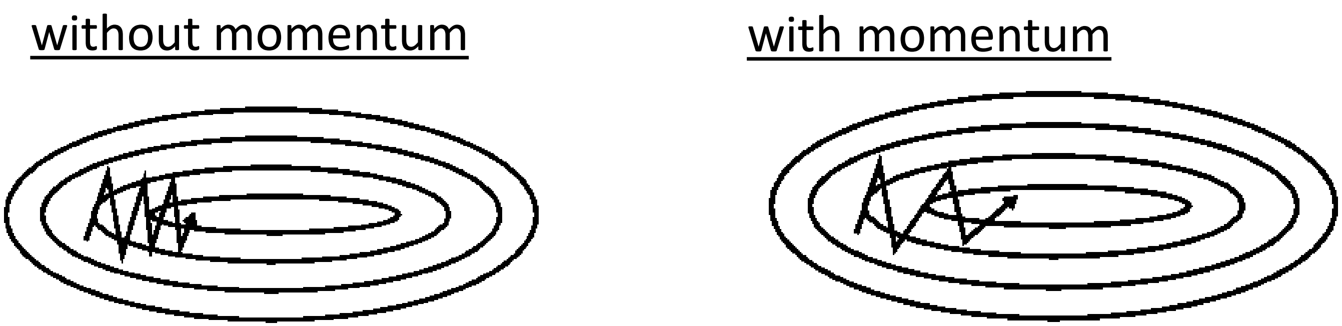
Momentum

Parameters build up velocity in direction of consistent gradient

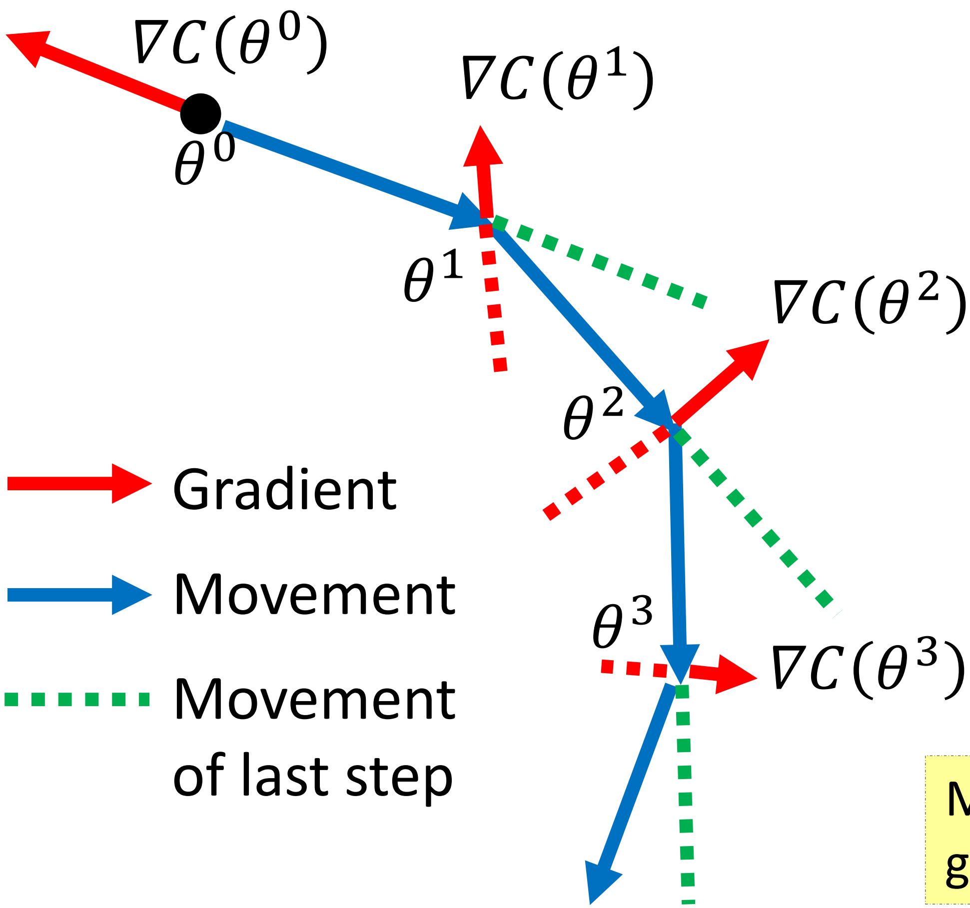
$$v^{i+1} = \lambda v^i - \eta \nabla C_\theta(\theta^i)$$
$$\theta^{i+1} = \theta^i + v^{i+1}$$



E.g. convex function optimization dynamics



Momentum



Algorithm

Initialization: model parameters start at θ^0, movement $v^0 = 0$

while($\theta^{(i+1)} \neq \theta^i$)

{

compute gradient at θ^i

update parameters

$$v^{i+1} = \lambda v^i - \eta \nabla C_\theta(\theta^i)$$

$$\theta^{i+1} = \theta^i + v^{i+1}$$

}

Movement is not only based on gradient, but also previous movement

Momentum

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

$$v^0 = 0$$

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

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

Algorithm

Initialization: model parameters start at θ^0, movement $v^0 = 0$

while($\theta^{(i+1)} \neq \theta^i$)

{

compute gradient at θ^i

update parameters

$$v^{i+1} = \lambda v^i - \eta \nabla C_\theta(\theta^i)$$

$$\theta^{i+1} = \theta^i + v^{i+1}$$

}

Movement is not only based on gradient, but also previous movement

Outline

Data Preprocessing

Activation Function

Loss Function

Optimization
- Adagrad
- Momentum

Generalization
- Early Stopping
- Regularization
- Dropout

Generalization

Practical tricks

1) find the right network structure and implement and optimize it properly
2) make the model overfit on training data
3) prevent overfitting
    - Reduce the model size by lowering the number of units and layers/hyperparameters
    - Early stopping
    - Standard L1 or L2 regularization
    - Sparsity constraint

Outline

Data Preprocessing

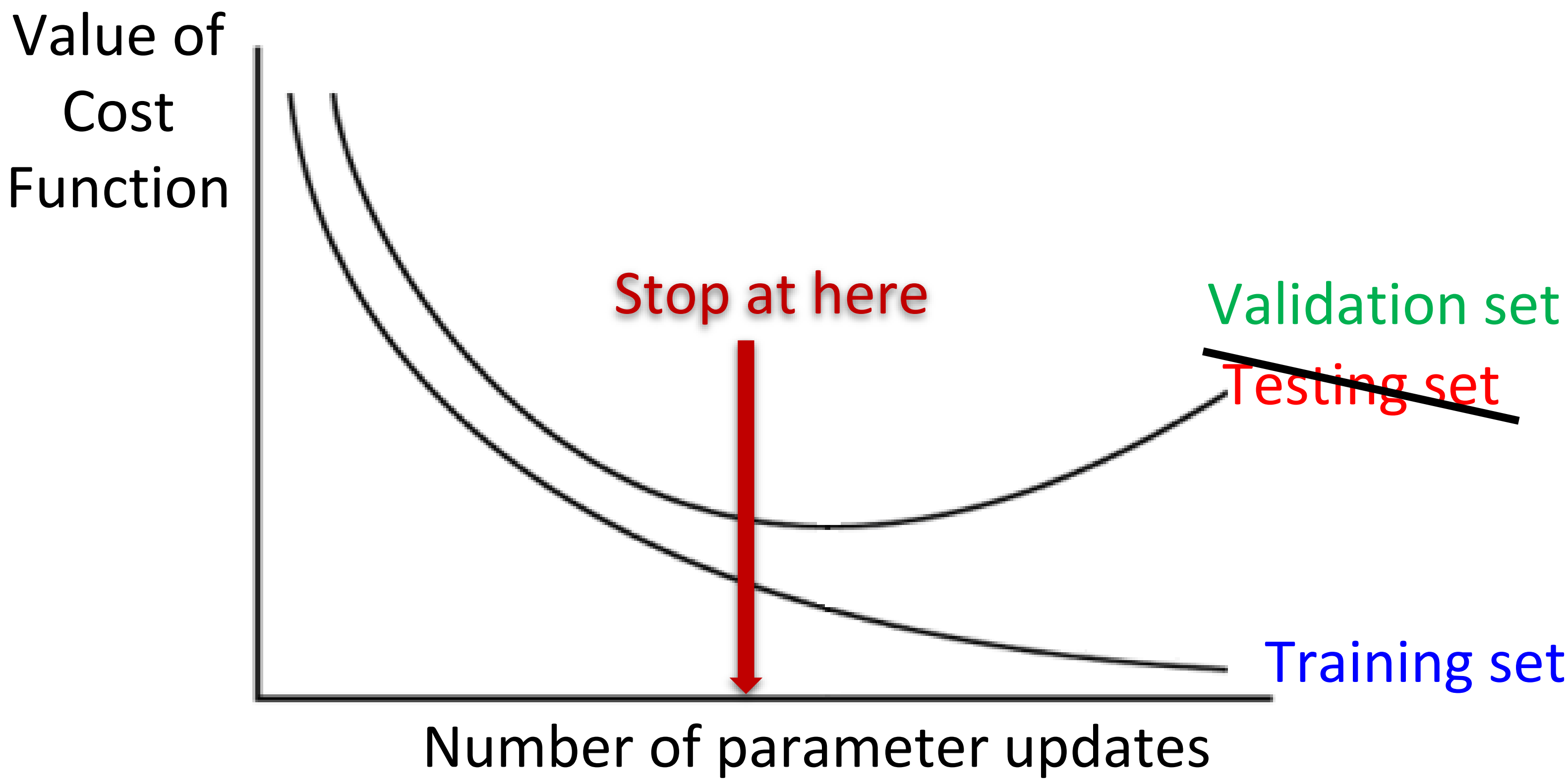
Activation Function

Loss Function

Optimization
- Adagrad
- Momentum

Generalization
- **Early Stopping**
- Regularization
- Dropout

Early Stopping



Check performance on validation set to prevent training too many iterations

Outline

Data Preprocessing

Activation Function

Loss Function

Optimization

- Adagrad
- Momentum

Generalization

- Early Stopping
- **Regularization**
- Dropout

Regularization

Idea: the parameters closer to zero are preferred

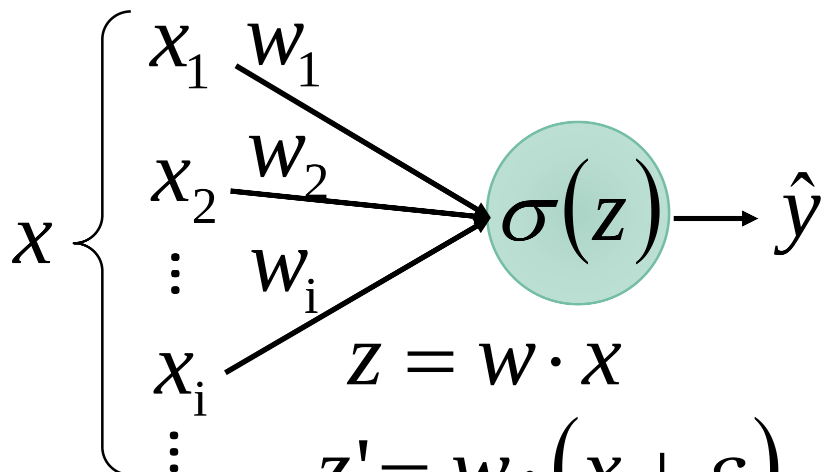
Training data:

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

Testing data:

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

$$x' = x + \varepsilon$$



$$z = w \cdot x$$

$$\begin{aligned} z' &= w \cdot \left(x + \varepsilon\right) \\ &= w \cdot x + w \cdot \varepsilon \\ &= z + w \cdot \varepsilon \end{aligned}$$

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

Regularization

Idea: optimize a new cost function to find a set of weight that 1) minimizes original cost and 2) is close to zero

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

→ regularization term

original cost

(e.g. minimize square error, cross entropy …)

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

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

→ not consider biases

Regularization

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

Idea: optimize a new cost function to find a set of weight that 1) minimizes original cost and 2) is close to zero

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

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

Update:

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

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

Smaller and smaller

Outline

Data Preprocessing

Activation Function

Loss Function

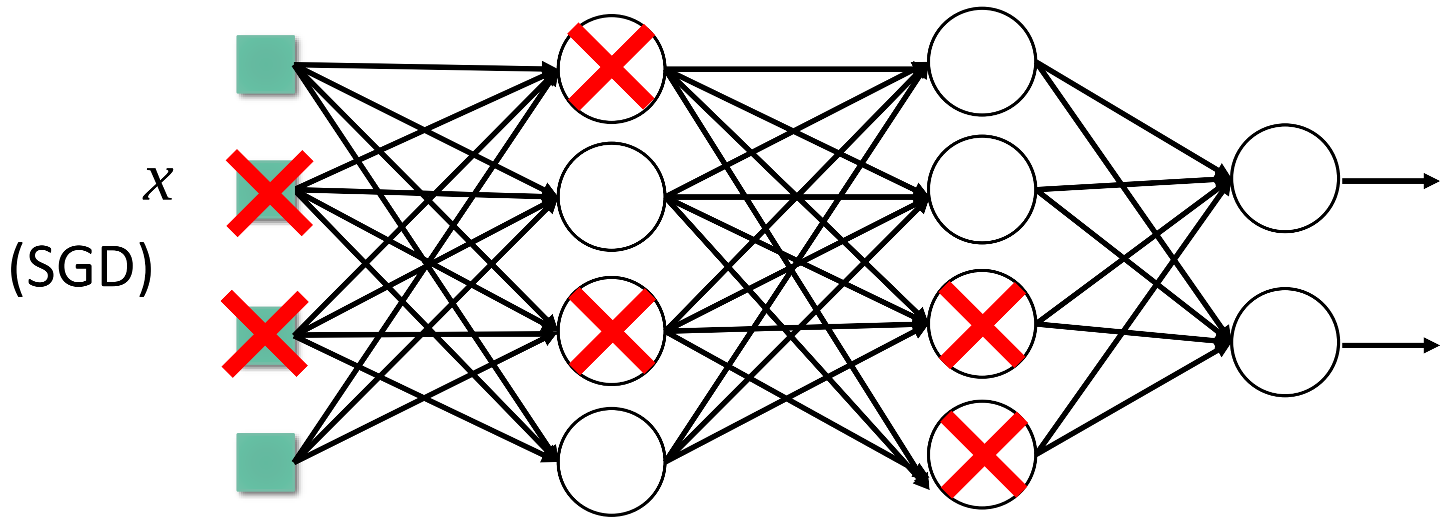
Optimization
- Adagrad
- Momentum

Generalization
- Early Stopping
- Regularization
- **Dropout**

Dropout

For each iteration of training,

- each neuron has $p\%$ to dropout

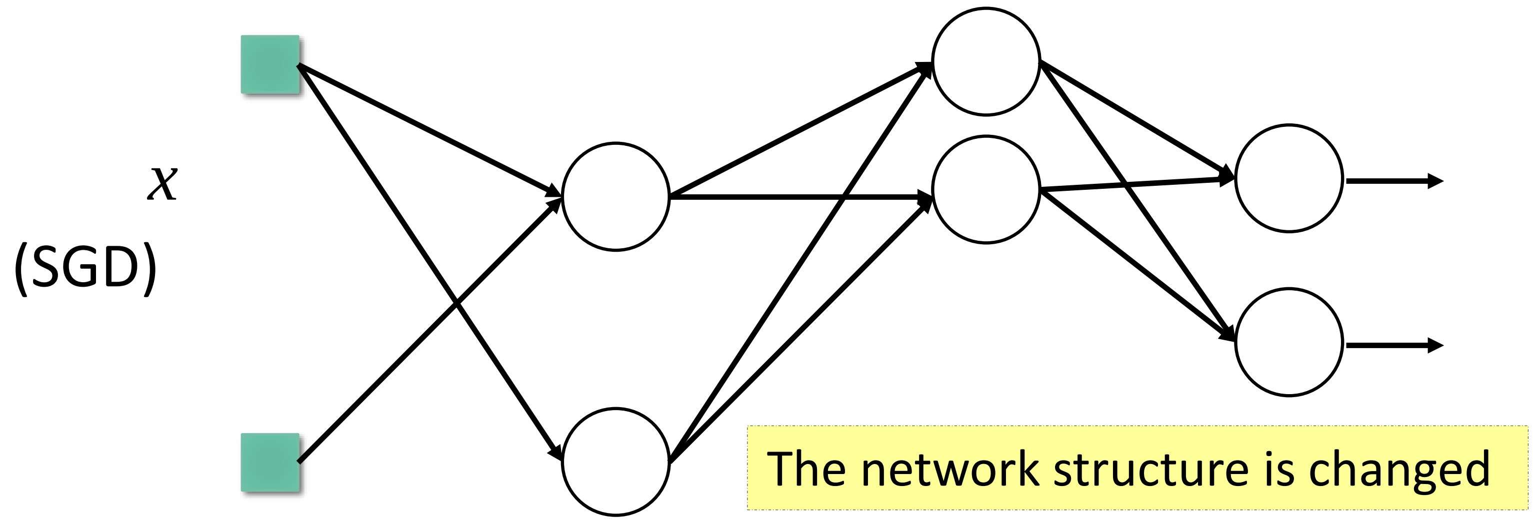


Dropout

For each iteration of training,

- each neuron has $p\%$ to dropout
- training using the new network

Training: $\theta^{i+1} \leftarrow \theta^{i} - \eta \nabla_{\theta} C(\theta^{i})$



Dropout

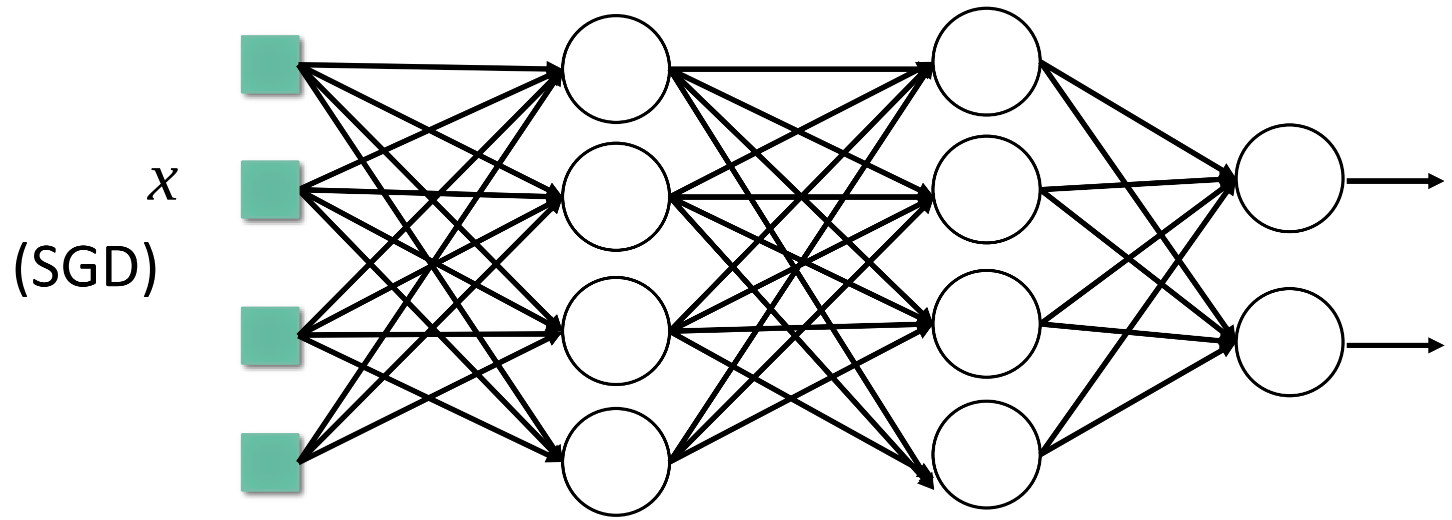
For each iteration of training,

- each neuron has p% to dropout
- training using the new network

Training: $\theta^{i+1} \leftarrow \theta^{i} - \eta \nabla_{\theta} C(\theta^{i})$

For testing,

- no dropout
- if the dropout rate at training is p%, all the weights times (1-p)%
- e.g. the dropout rate is 50%, $w_{ij}^{l} = 1$ from training → $w_{ij}^{l} = 0.5$ for testing



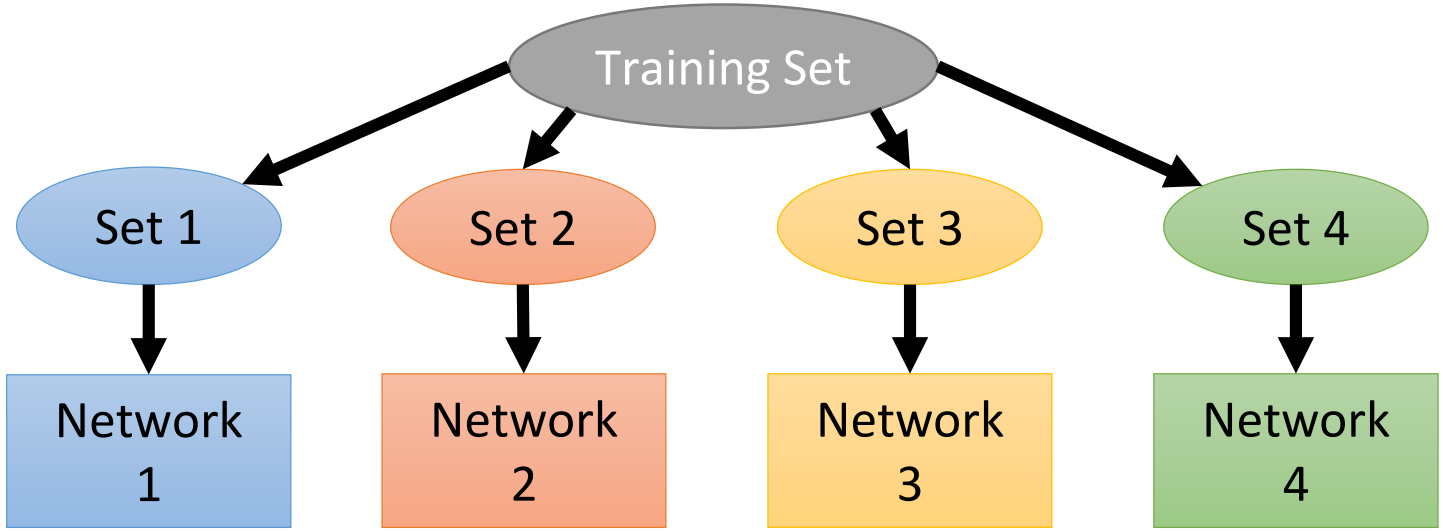
Dropout – Intuitive Reason

Training: dropout

Testing: no dropout

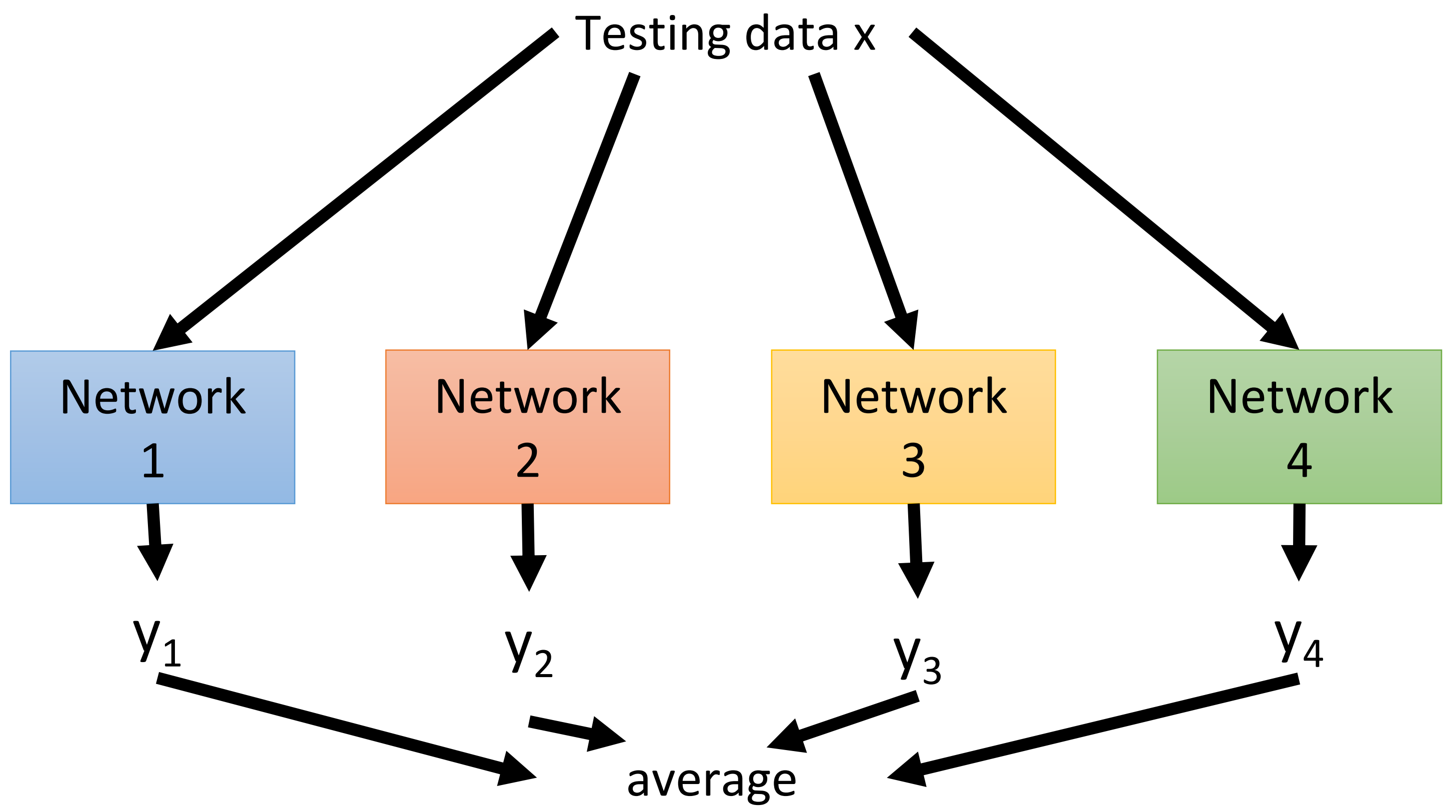
Dropout

Train a bunch of networks with different structures



Dropout – Ensemble

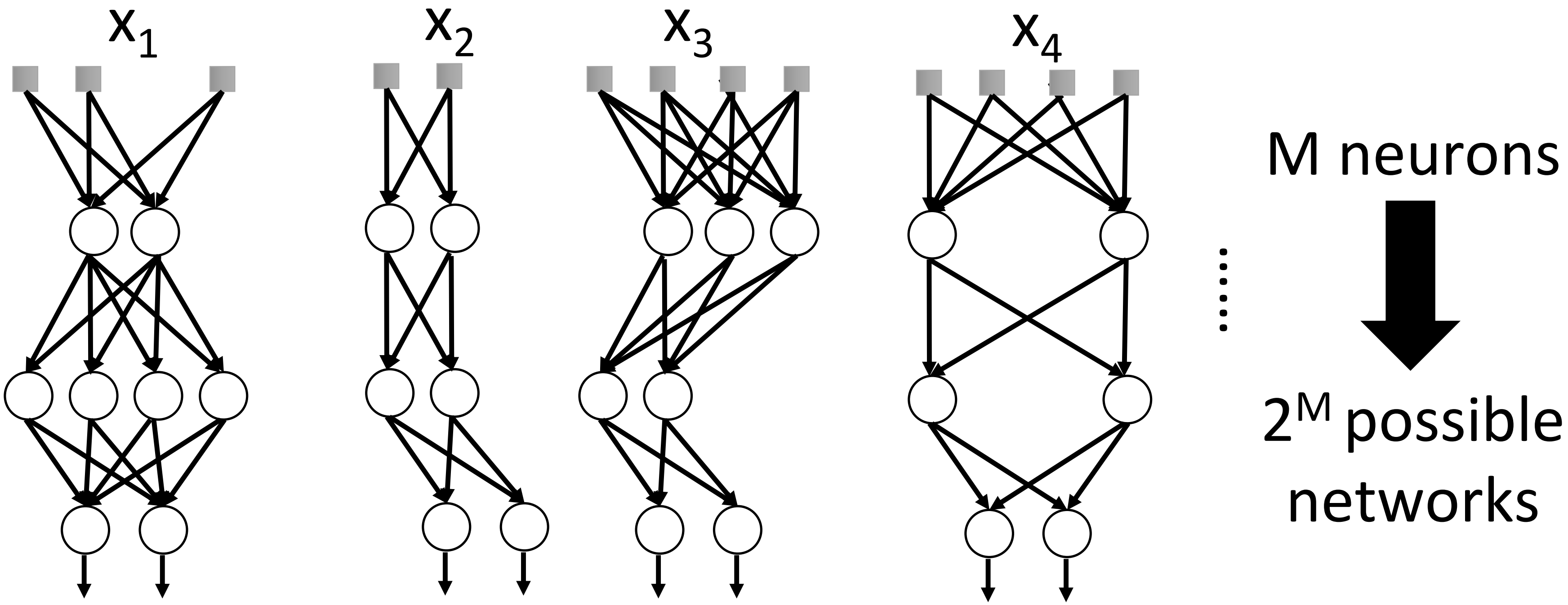
Ensemble



Dropout – Ensemble

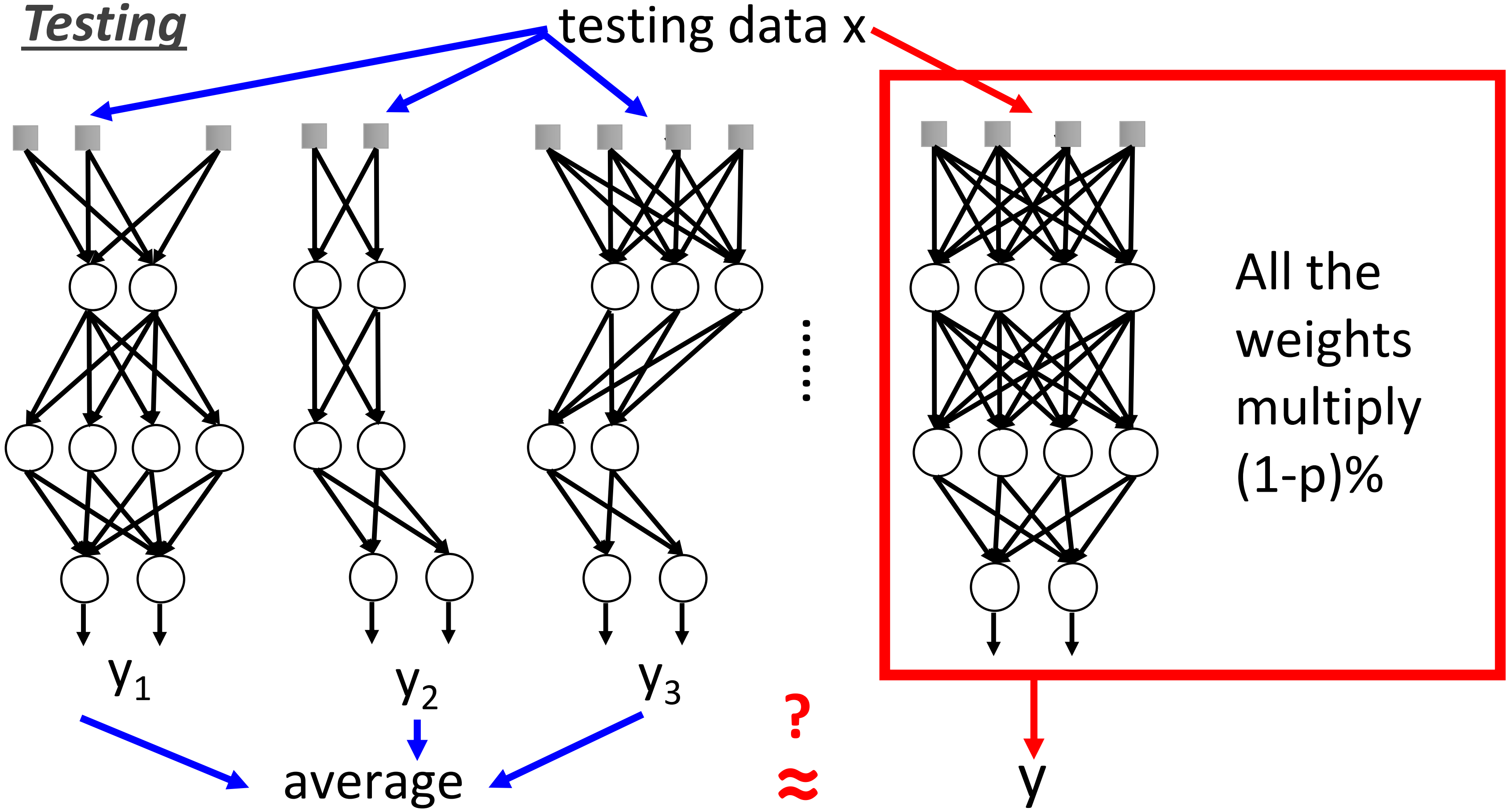
Training

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



Dropout – Ensemble

Testing



Dropout Tips

Larger network

- If you know that your task needs ***n*** neurons, for dropout rate ***p***, your network need ***n/(1-p)*** neurons.

Longer training time

Higher learning rate

Larger momentum

Concluding Remarks

Data Preprocessing: Input Normalization

Activation Function: ReLU, Maxout

Loss Function: Softmax

Optimization

- Adagrad: Learning Rate Adaptation
- Momentum: Learning Direction Adaptation

Generalization

- Early Stopping: avoid too many iterations from overfitting
- Regularization: minimize the effect of noise
- Dropout: leverage the benefit of ensemble