训练神经网络的技巧

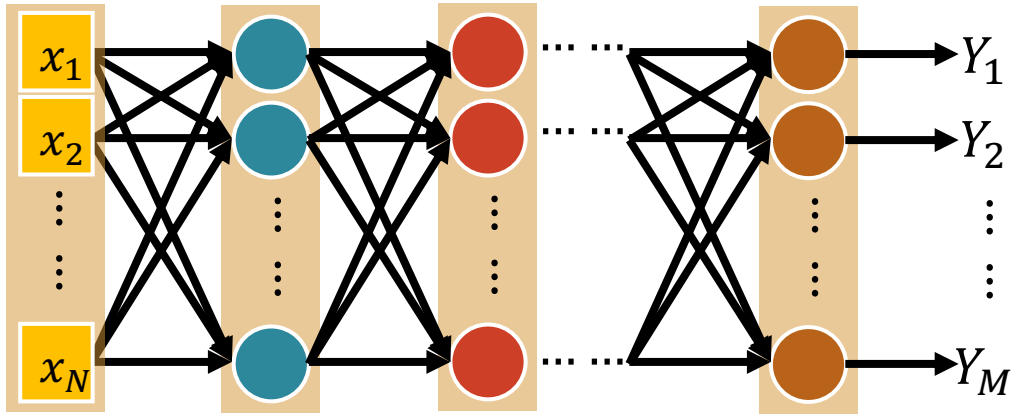
Tricks for Training Artificial Neural Networks

郝　奇
南京大学 天文与空间科学学院

Application of Machine Learning in Astronomy

机器学习在天文中的应用

Contents



Cite: Hung-yi Lee, Machine Learning

Recipe of Artificial Neural Networks

01

1. Recipe of Artificial Neural Networks

Step 1
define a set of function

Step 2
goodness of function

Step 3
pick the best function

NO

Overfitting

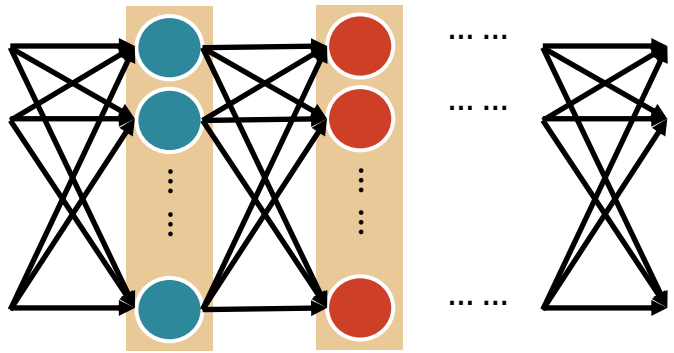
Good results on testing data ?

YES

YES

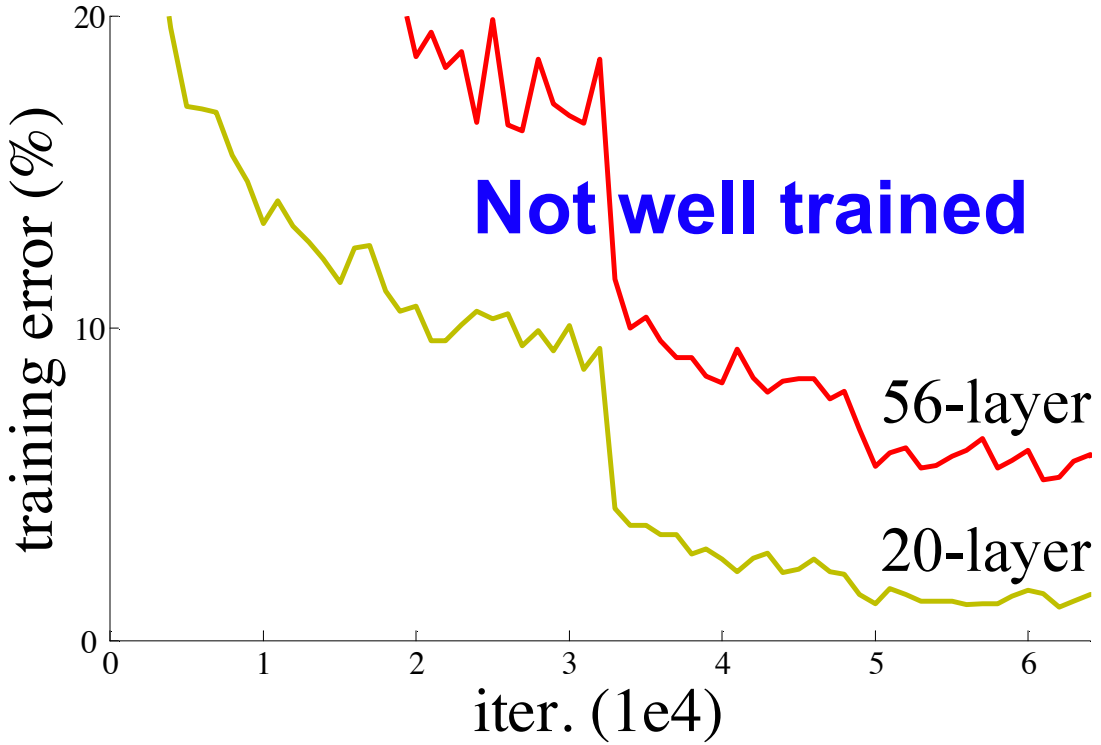
NO

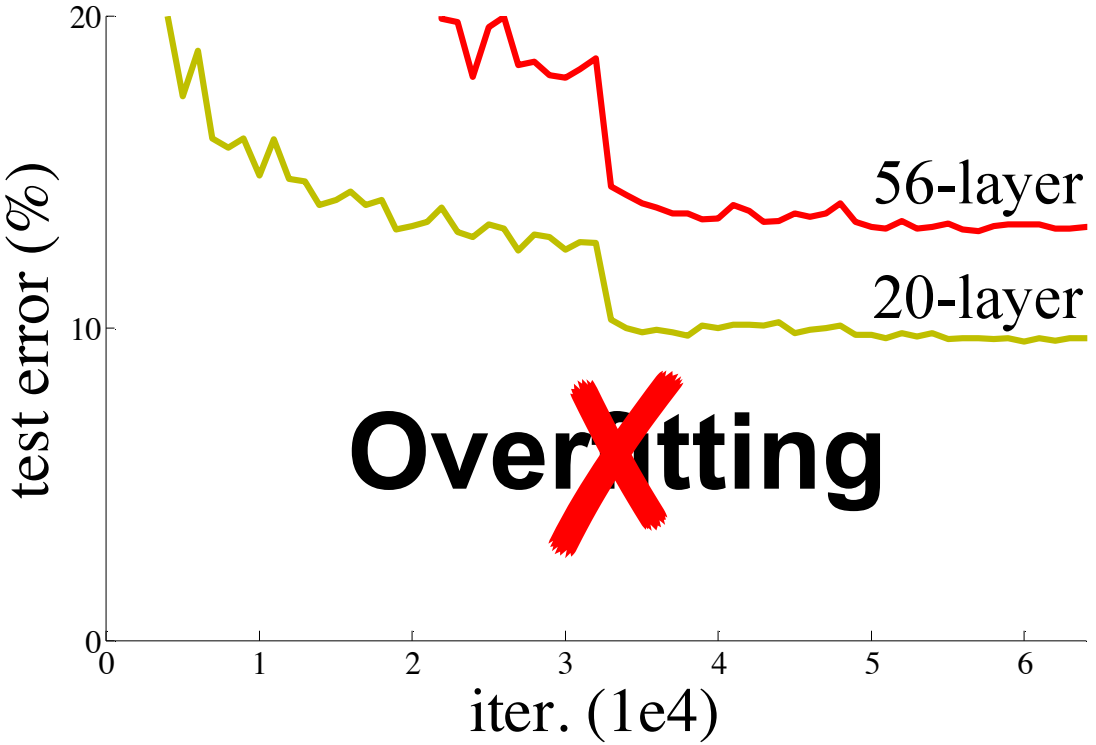
Good results on training data ?



1. Recipe of Artificial Neural Networks

Do not always blame overfitting

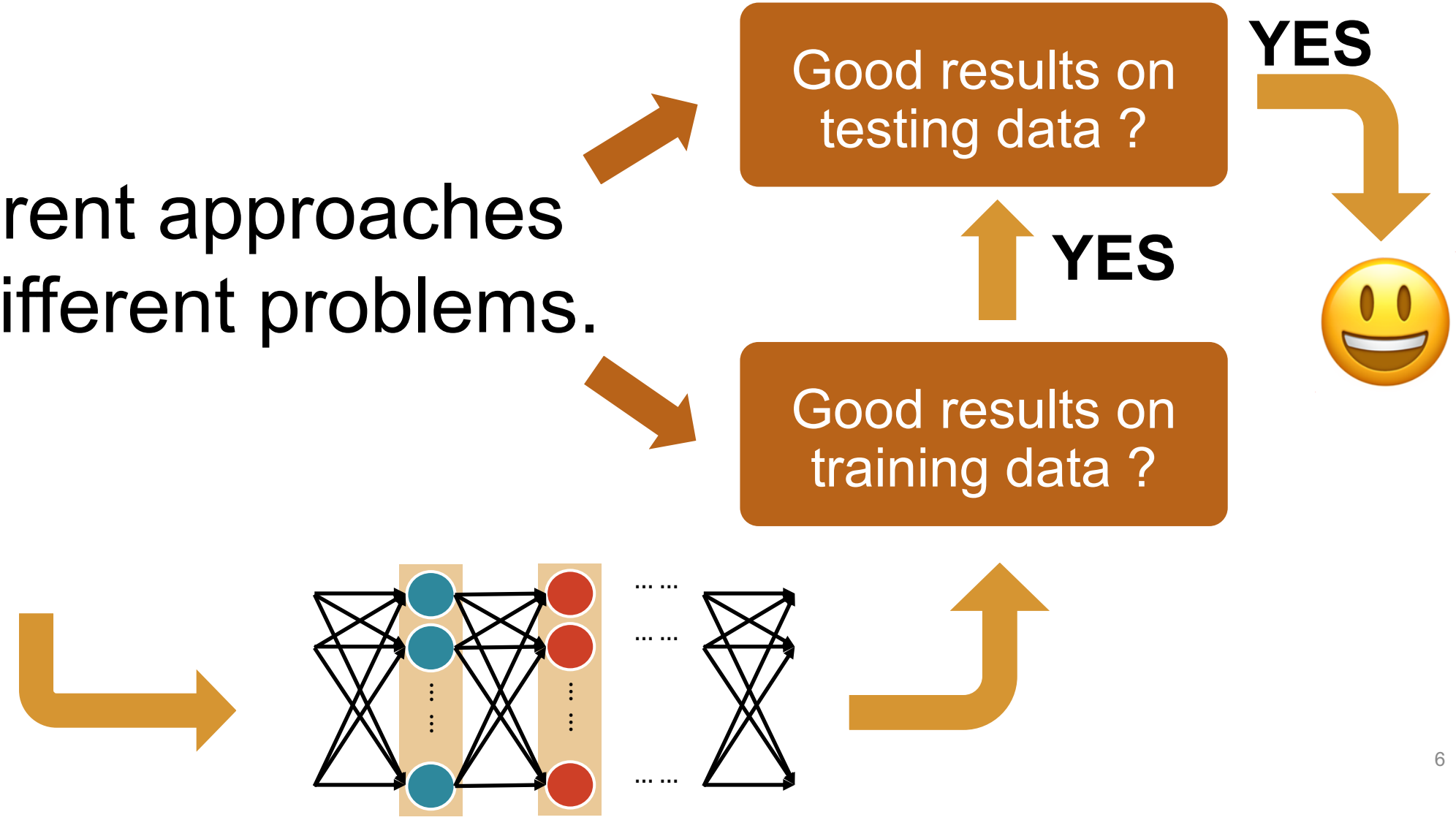




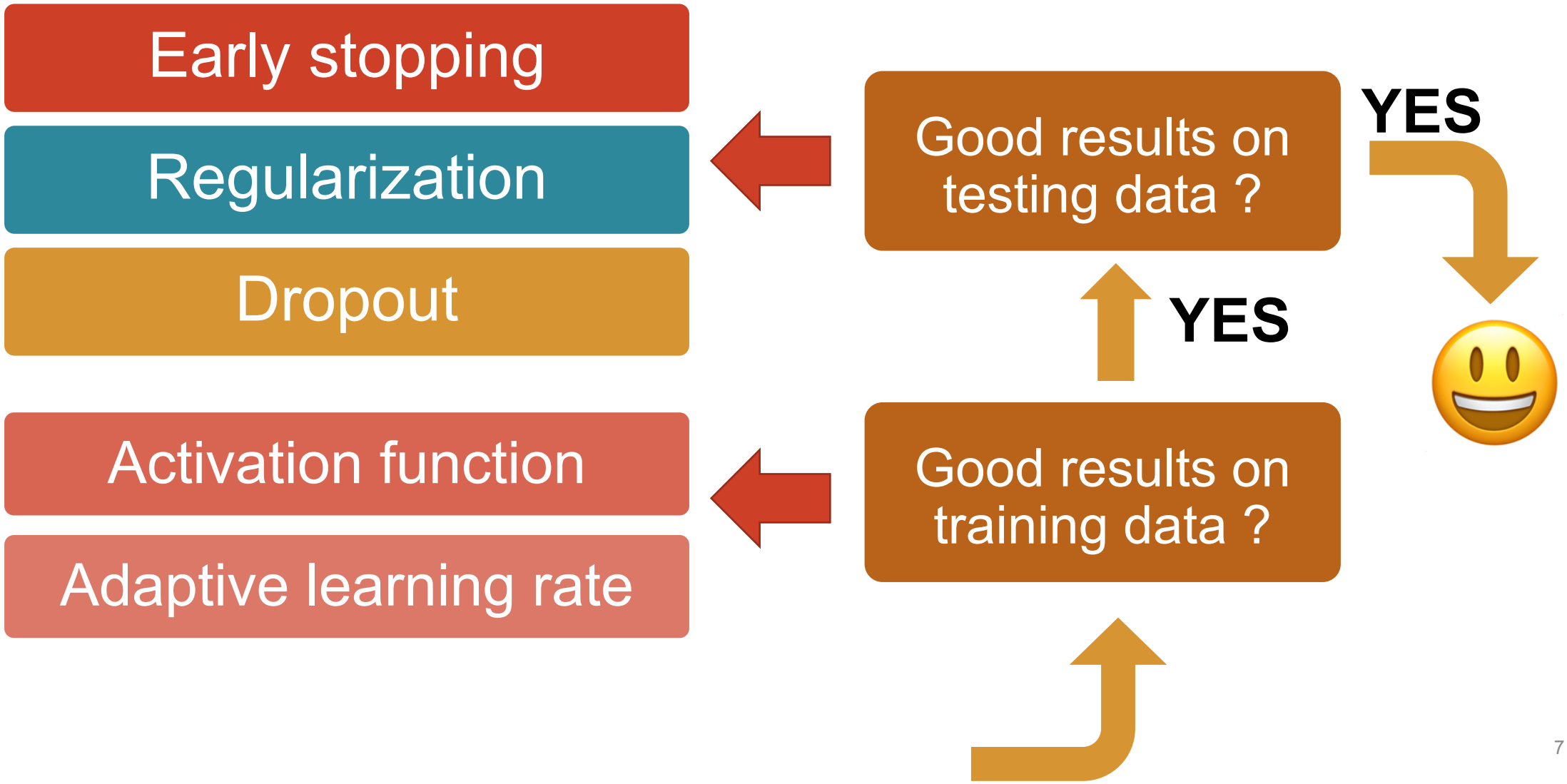
Cite: https://arxiv.org/abs/1512.03385

1. Recipe of Artificial Neural Networks

Different approaches for different problems.

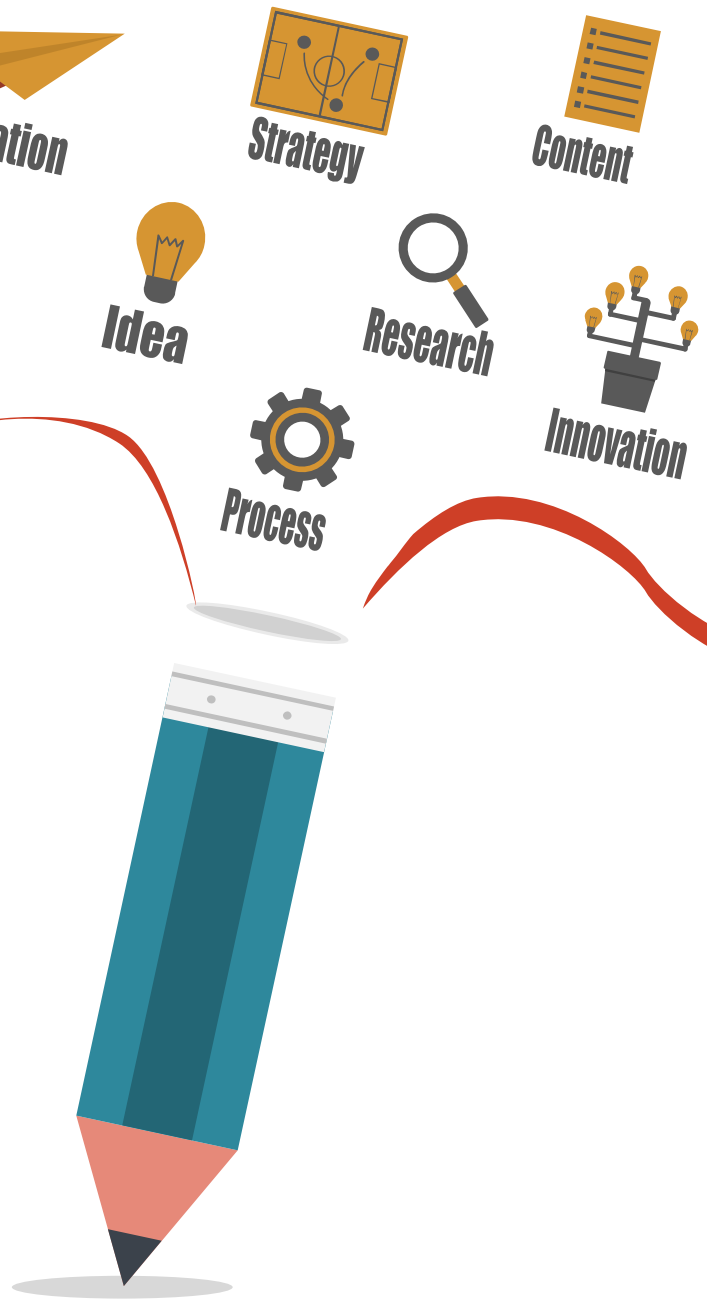


1. Recipe of Artificial Neural Networks



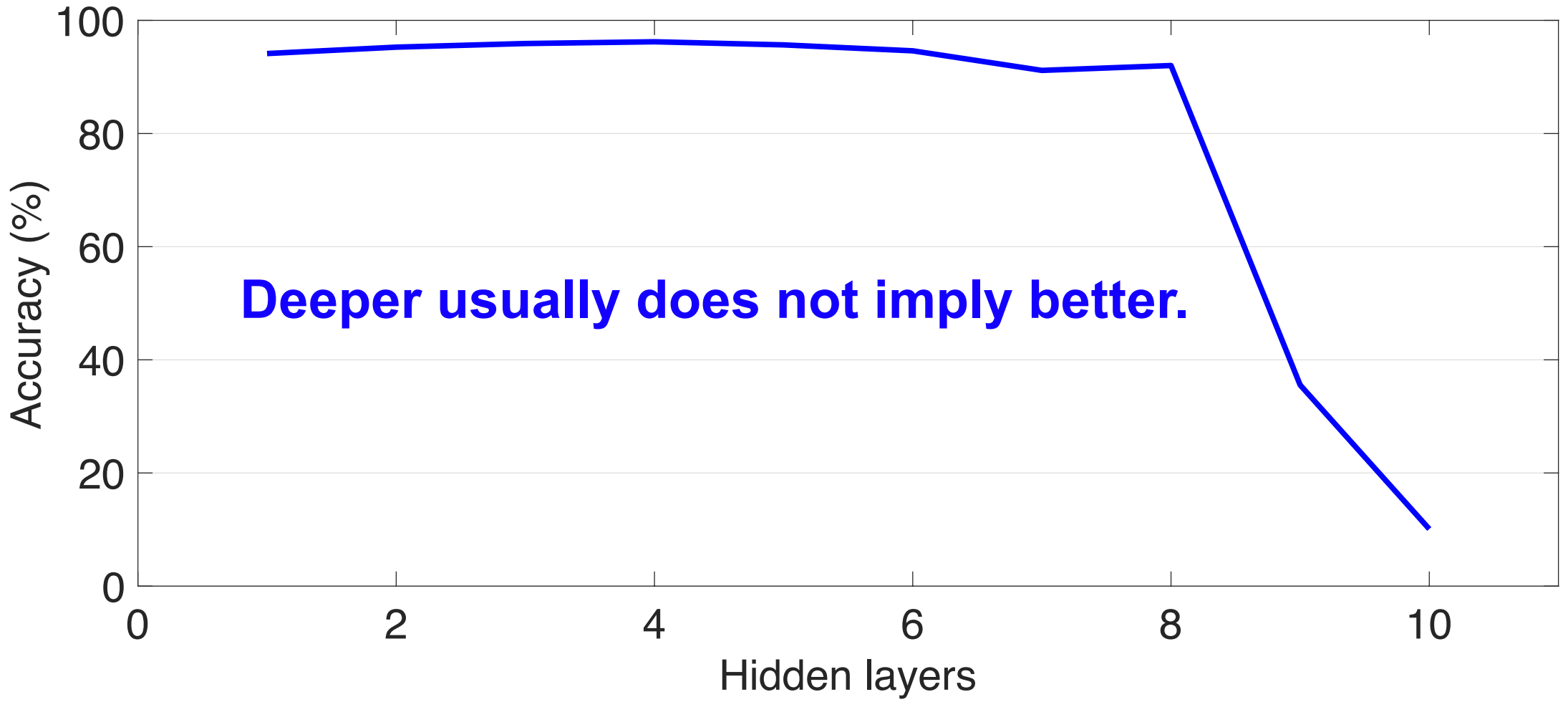
Tricks from training process

02



2. Tricks from training process

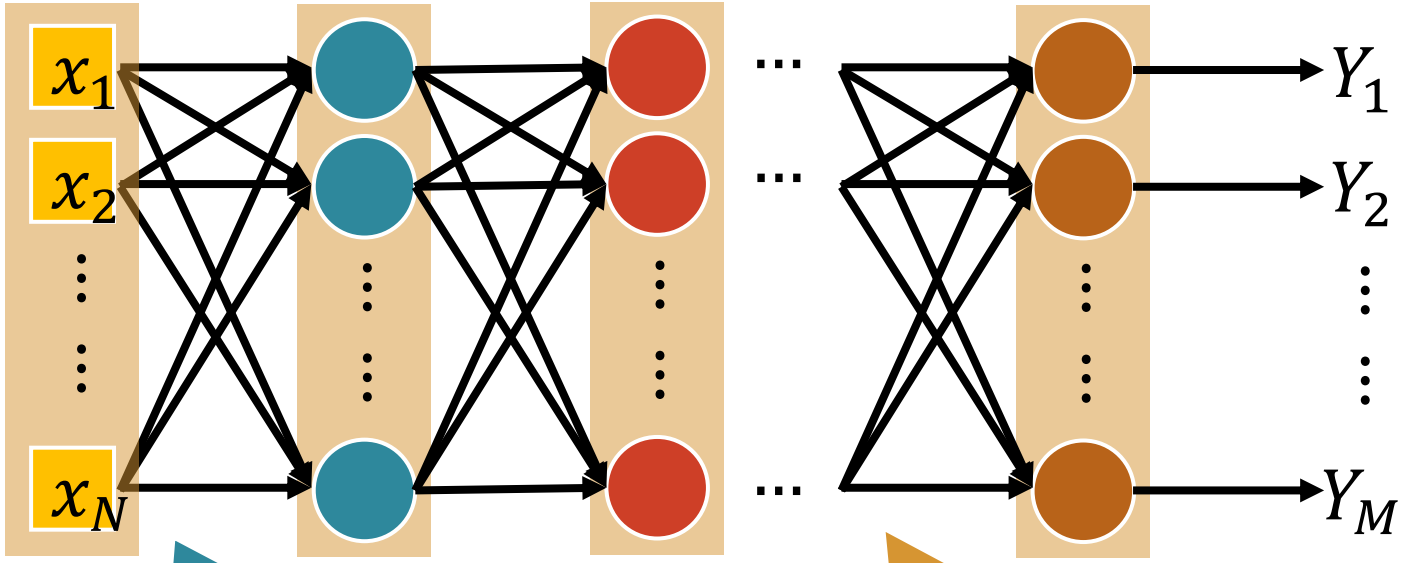
2.1 Activation function



2. Tricks from training process

2.1 Activation function

Vanishing gradient problem



Smaller gradients

Learn very slow

Almost random

Larger gradients

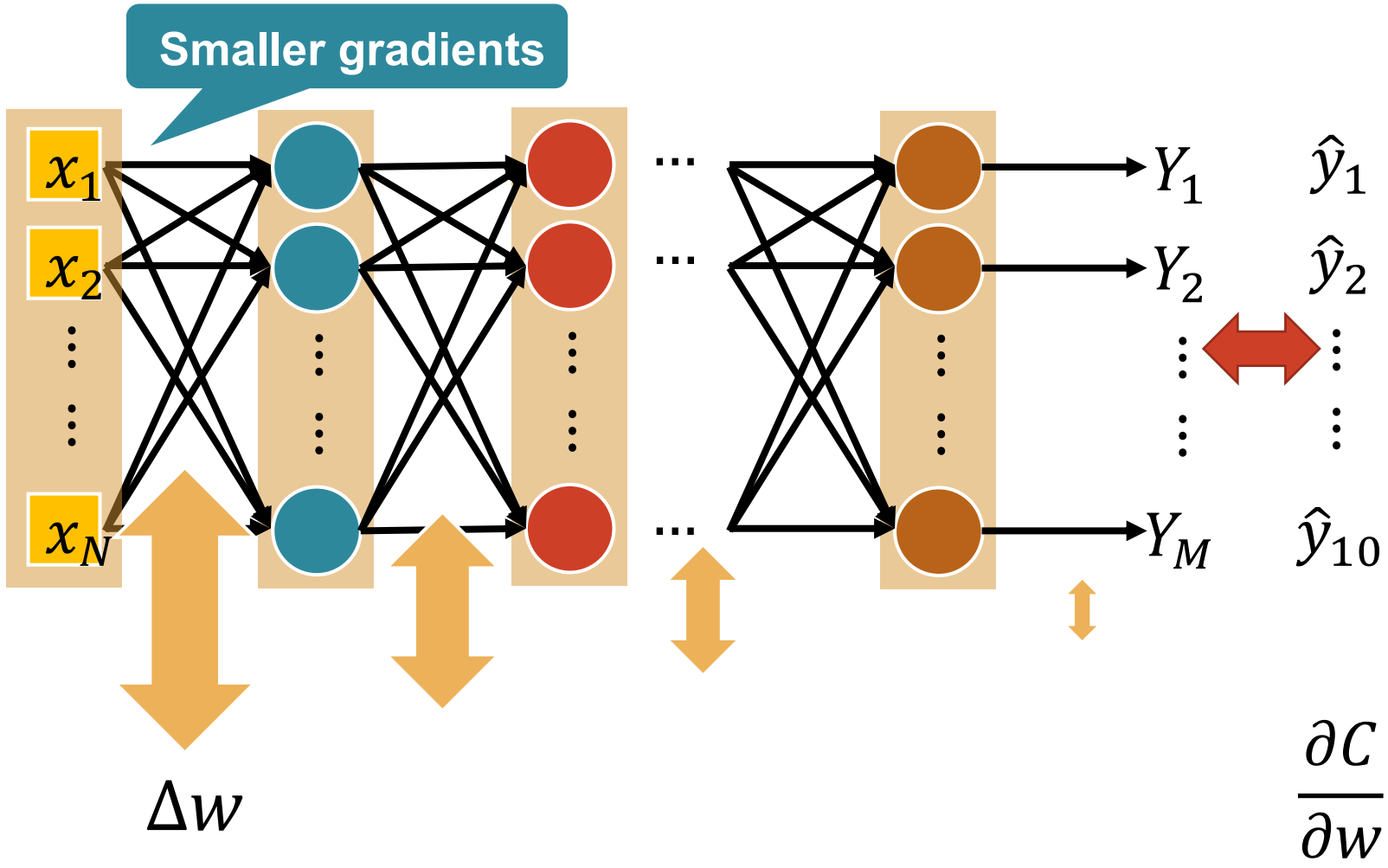
Learn very fast

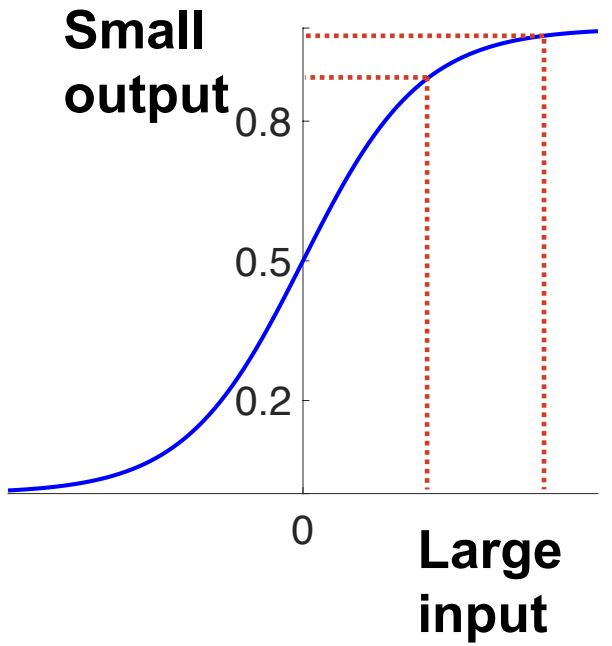
Already converge

2. Tricks from training process

2.1 Activation function

Vanishing gradient problem





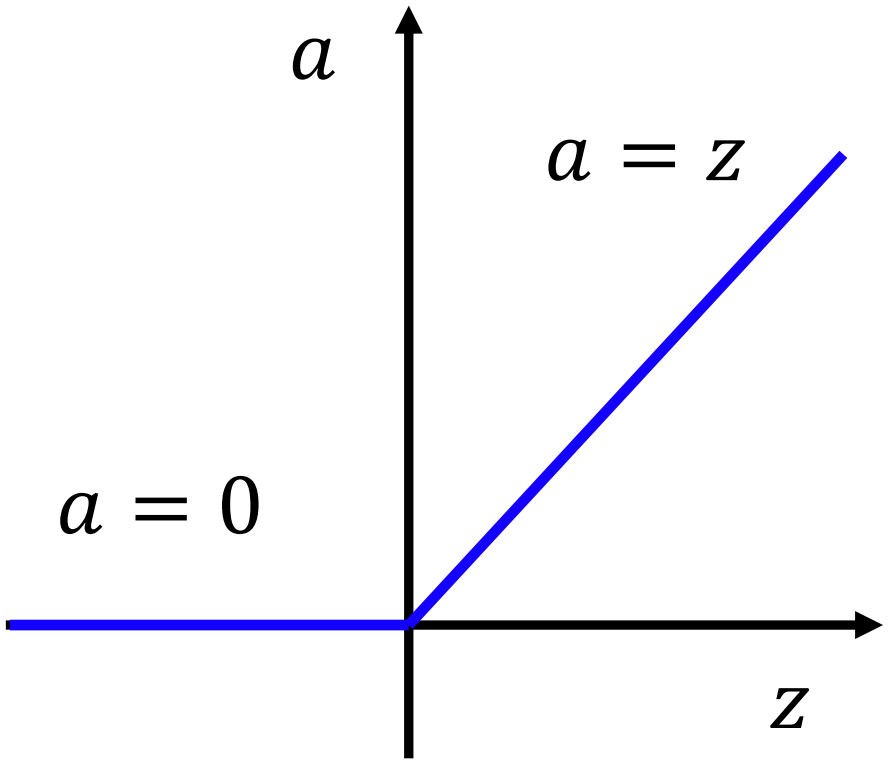
$$\frac{\partial C}{\partial w} =? \frac{\Delta C}{\Delta w}$$

2. Tricks from training process

2.1 Activation function

ReLU

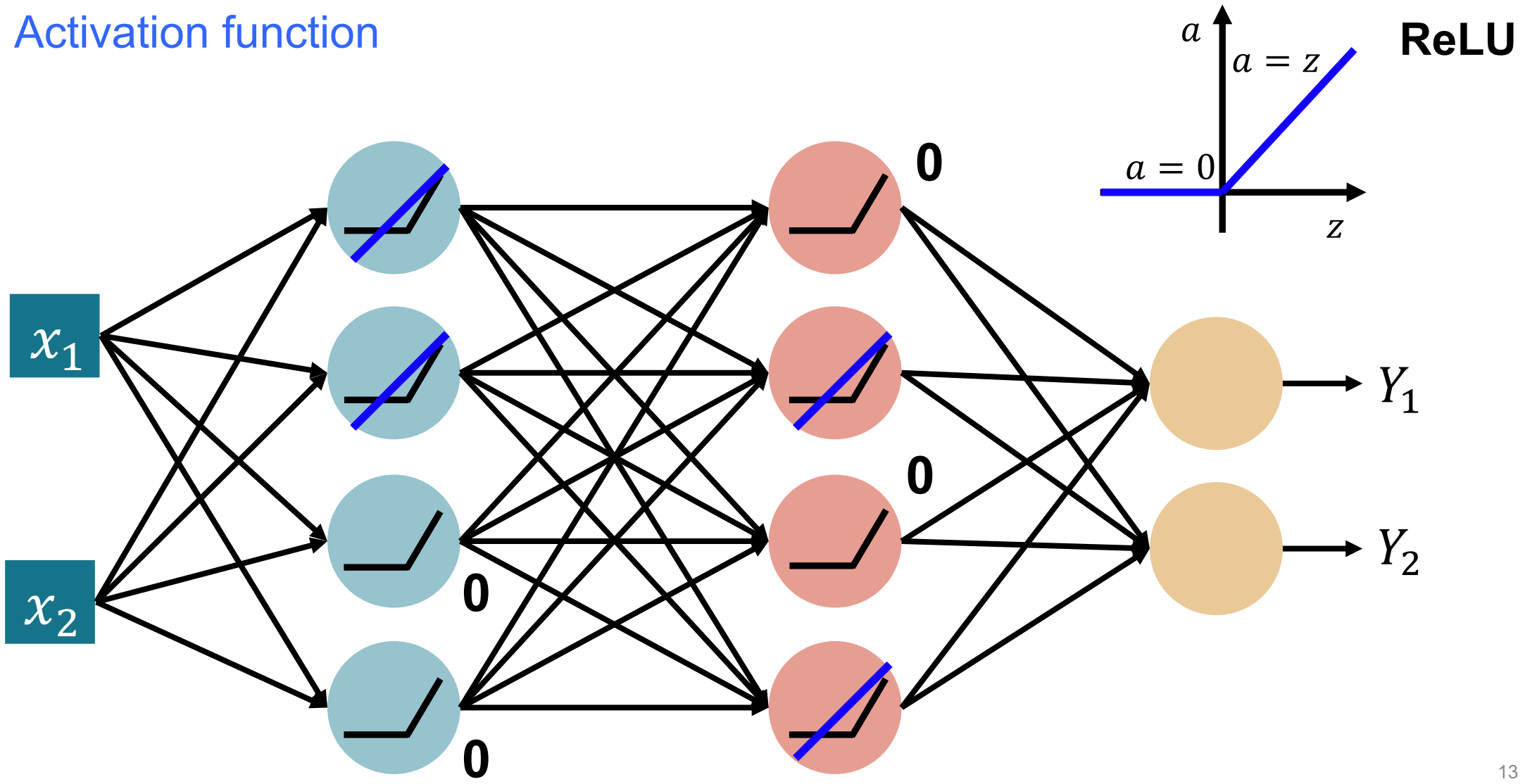
Rectified Linear Unit (ReLU)



- Fast to compute;
- Biological reason;
- Infinite sigmoid with different biases;
- Solve vanishing gradient problem.

2. Tricks from training process

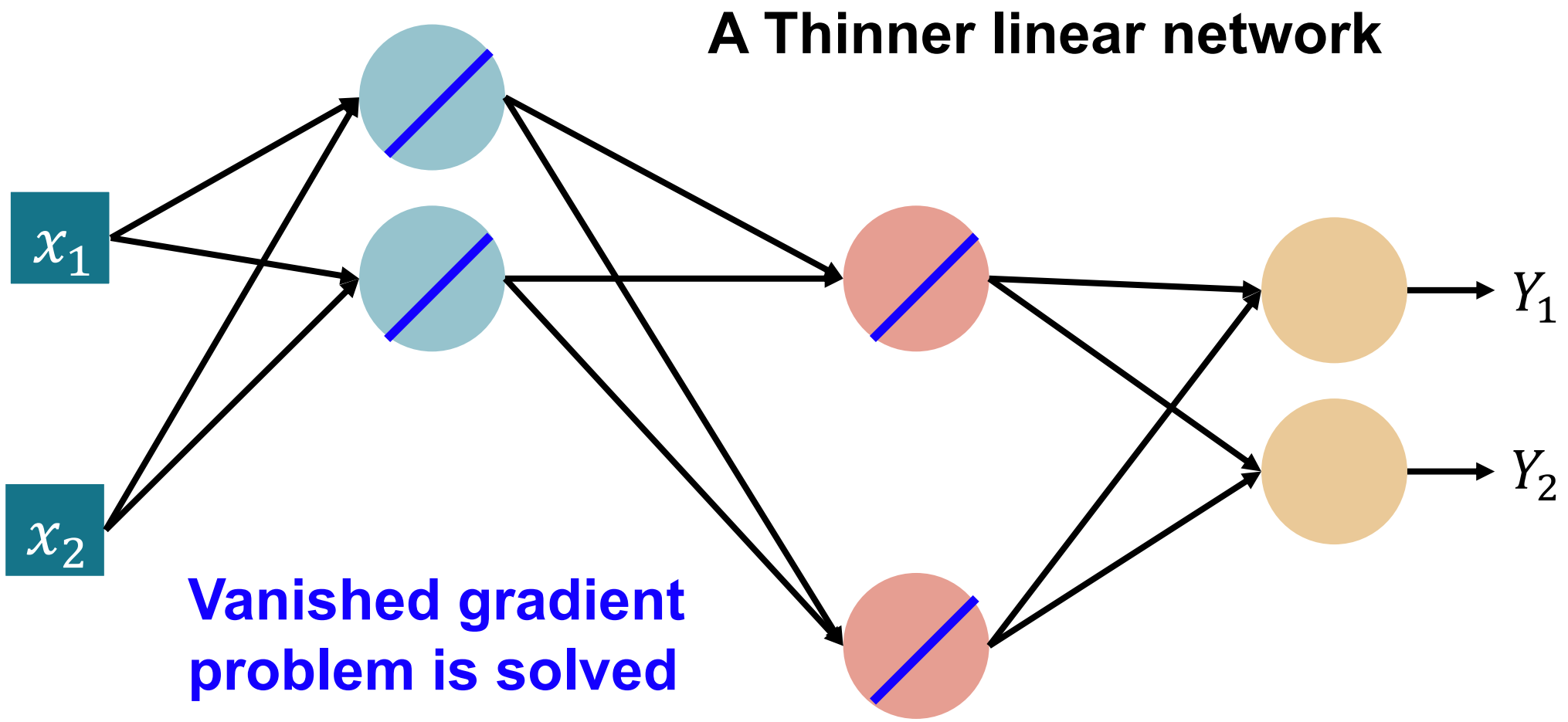
2.1 Activation function



2. Tricks from training process

2.1 Activation function

ReLU

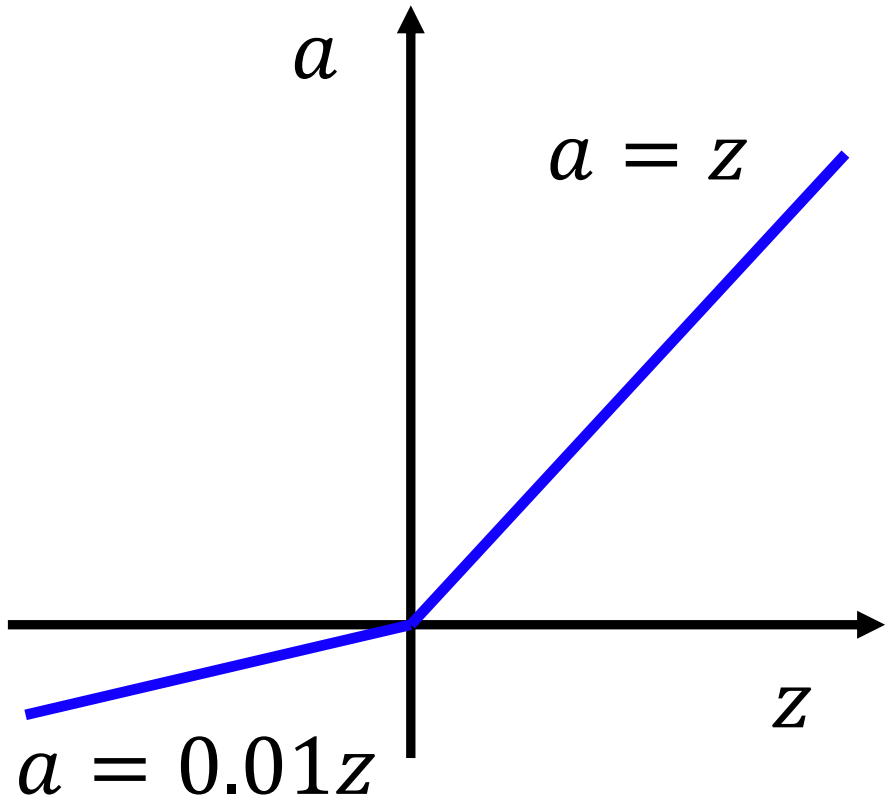


2. Tricks from training process

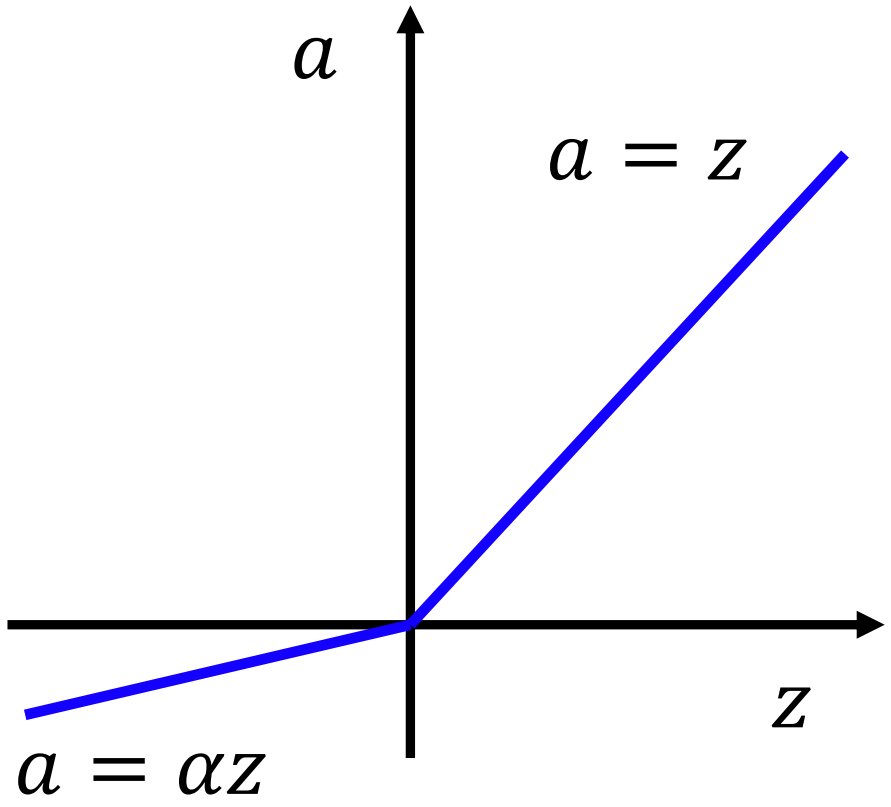
2.1 Activation function

ReLU variant

Leaky ReLU



Parametric ReLU

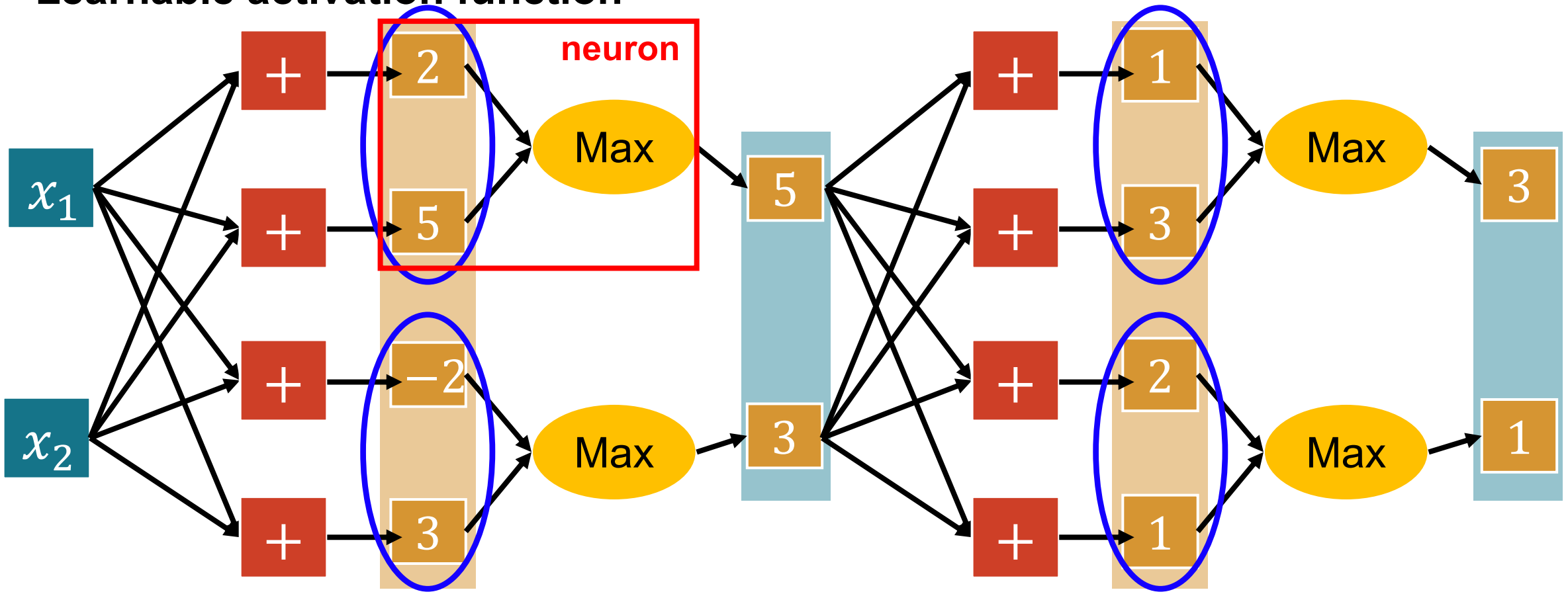


2. Tricks from training process

2.1 Activation function

Maxout

Learnable activation function

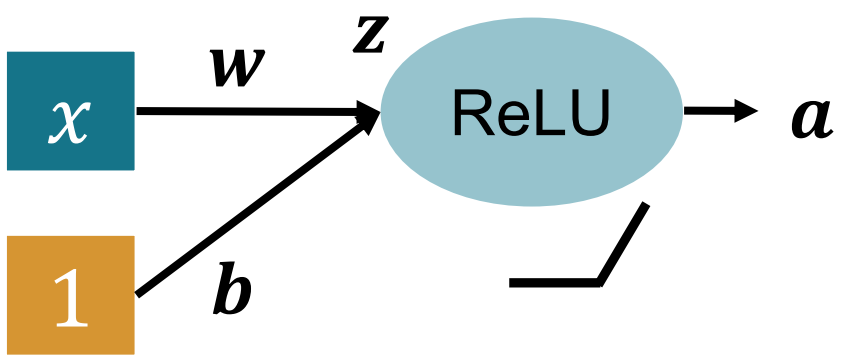


You can have more than 2 elements in a group

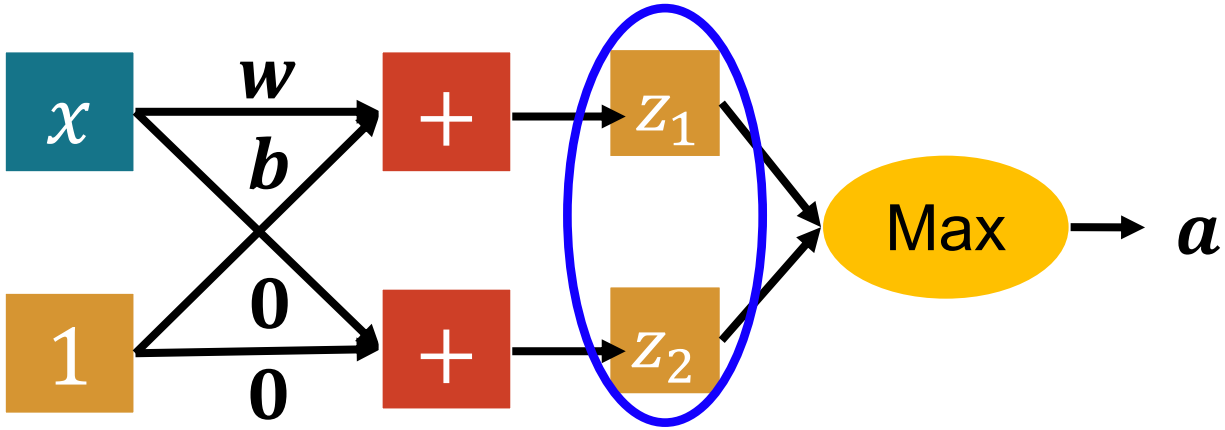
2. Tricks from training process

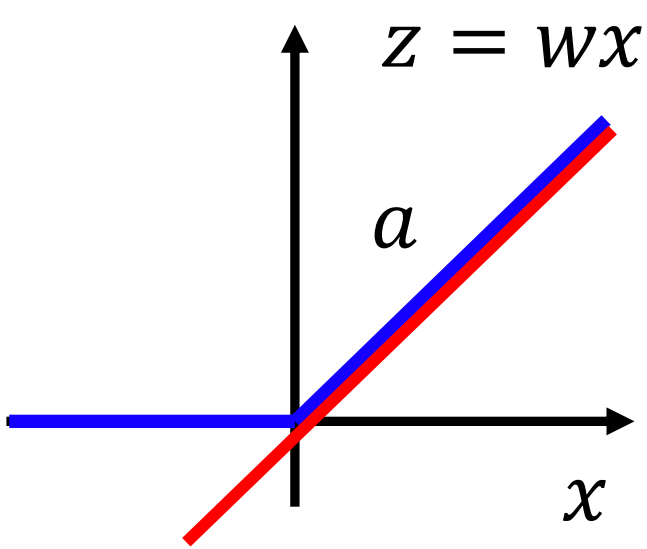
2.1 Activation function

Learnable activation function

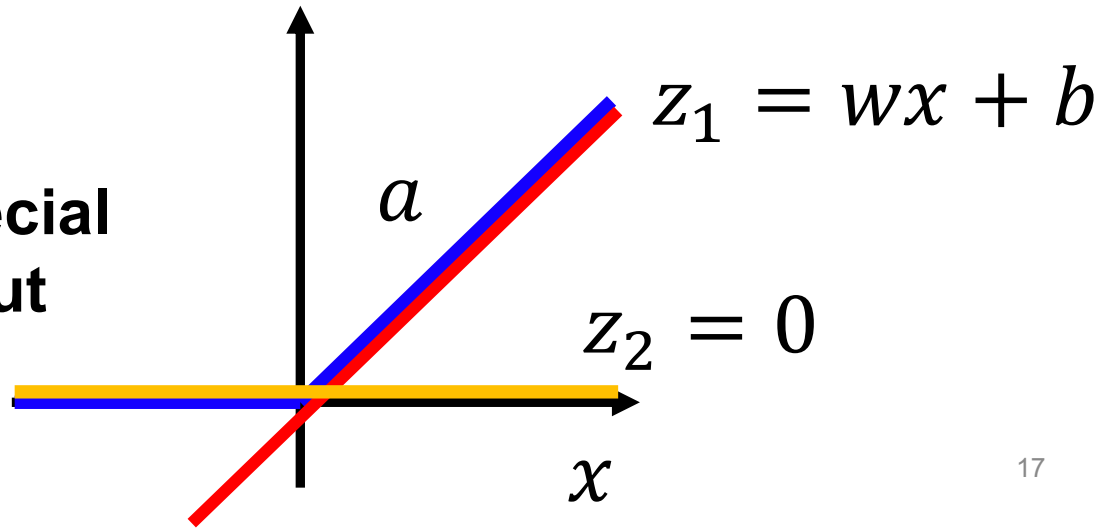


Maxout





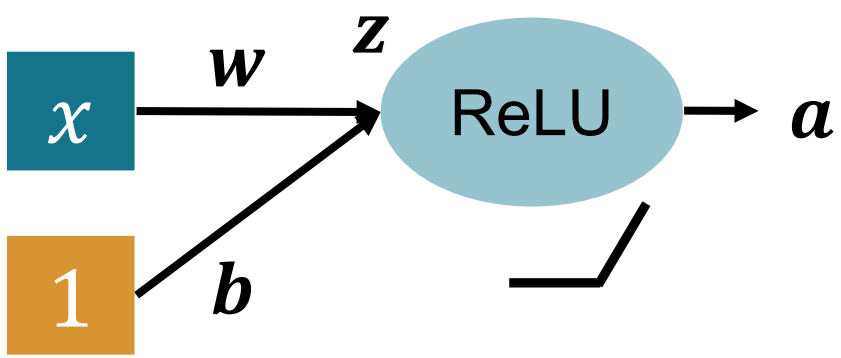
ReLU is a special case of Maxout



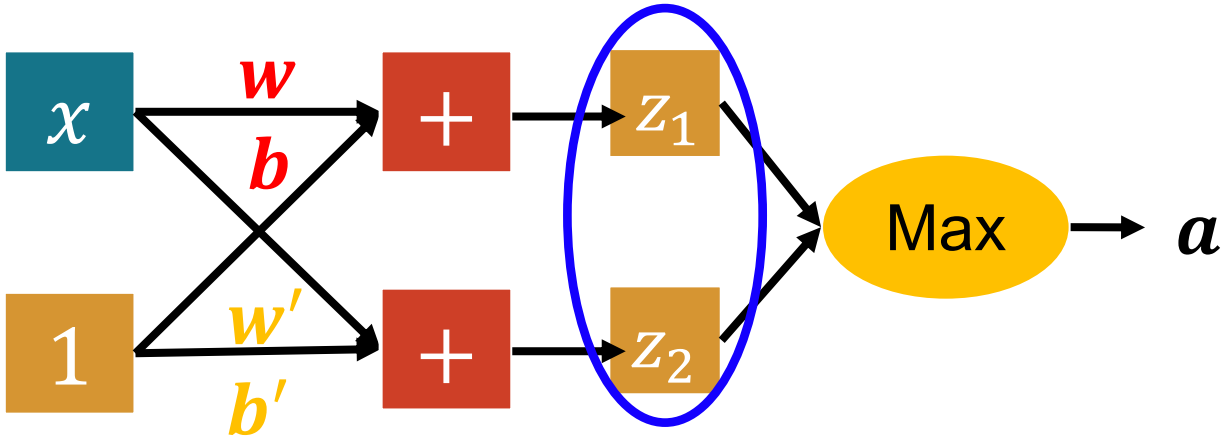
2. Tricks from training process

2.1 Activation function

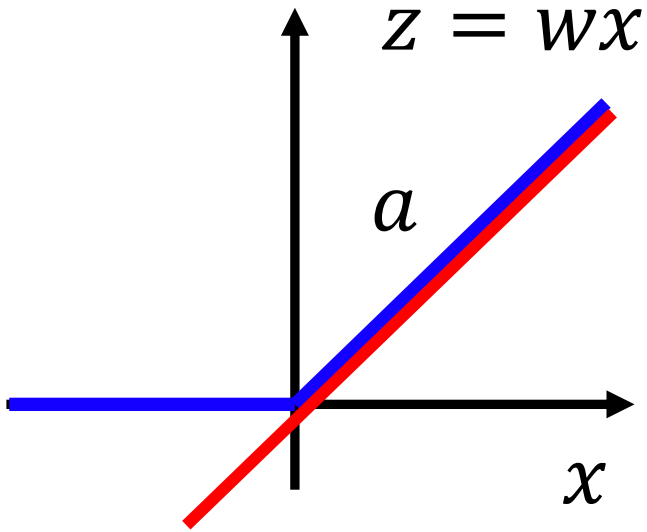
Learnable activation function



Maxout

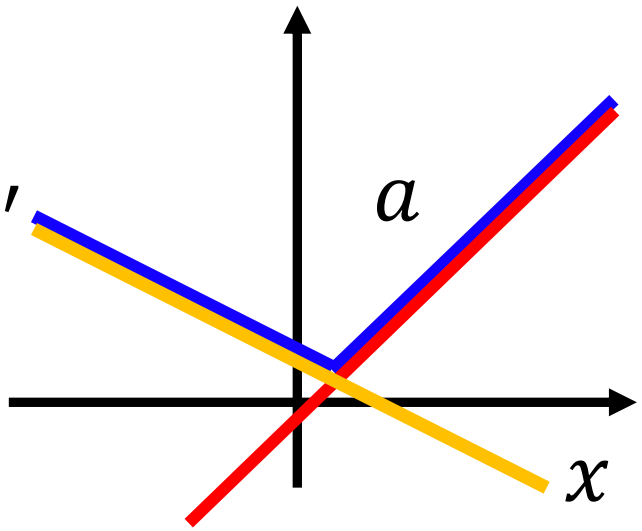


$z = wx + b$



$z_2 = w'x + b'$

$z_1 = wx + b$



2. Tricks from training process

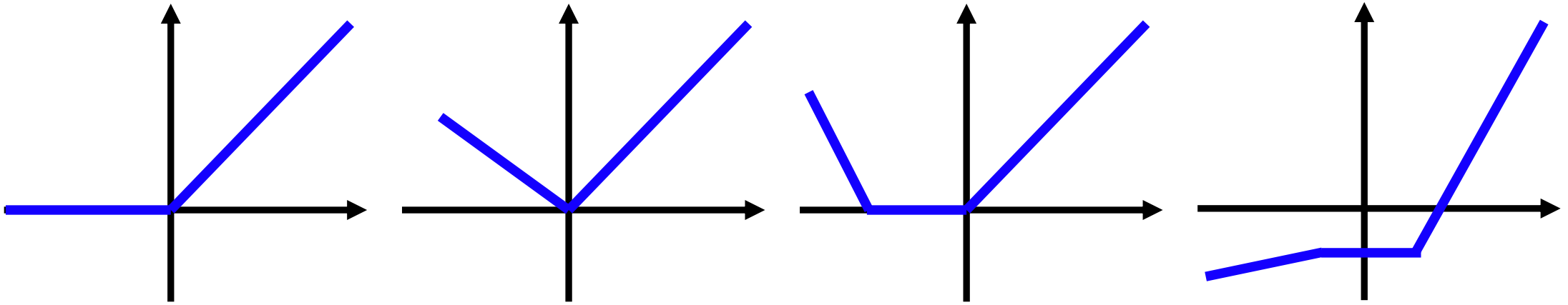
2.1 Activation function

Maxout

Learnable activation function

- Activation function in maxout network can be any piecewise linear convex function;
- How many pieces depending on how many elements in a group.

2 element in a group **3 element in a group**

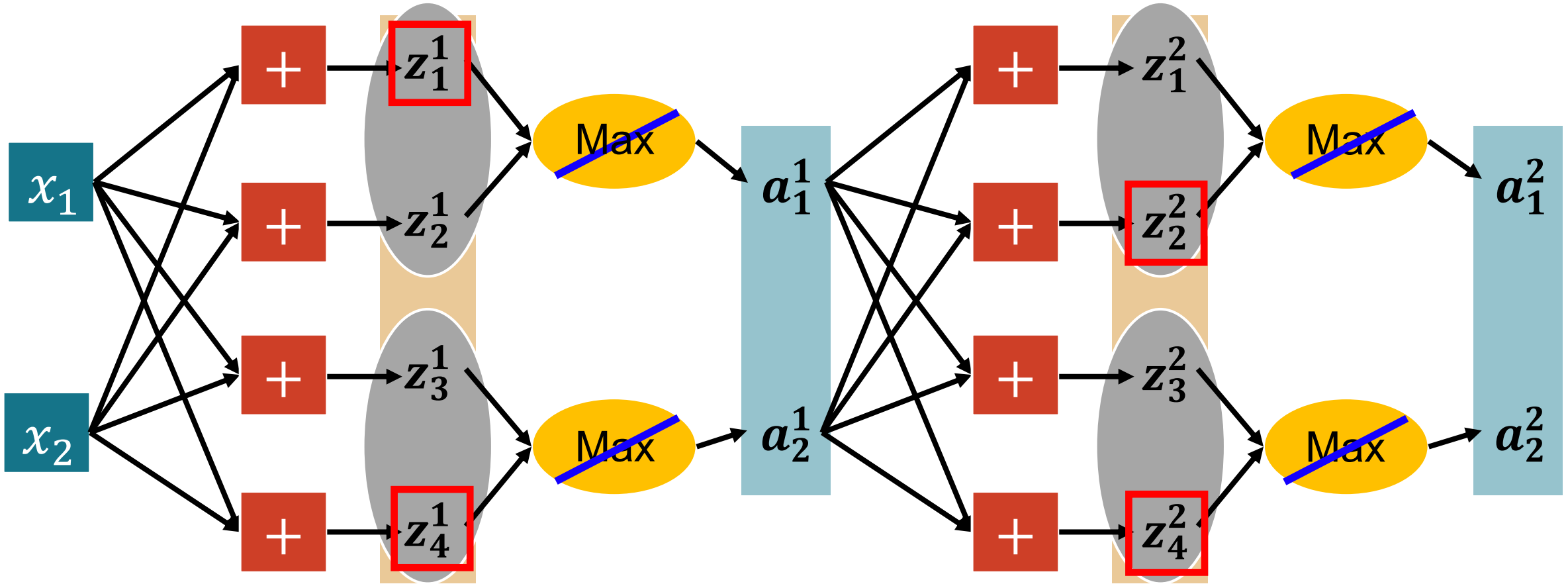


2. Tricks from training process

2.1 Activation function

Maxout

Training **Given a training data x, the max of z is determined**



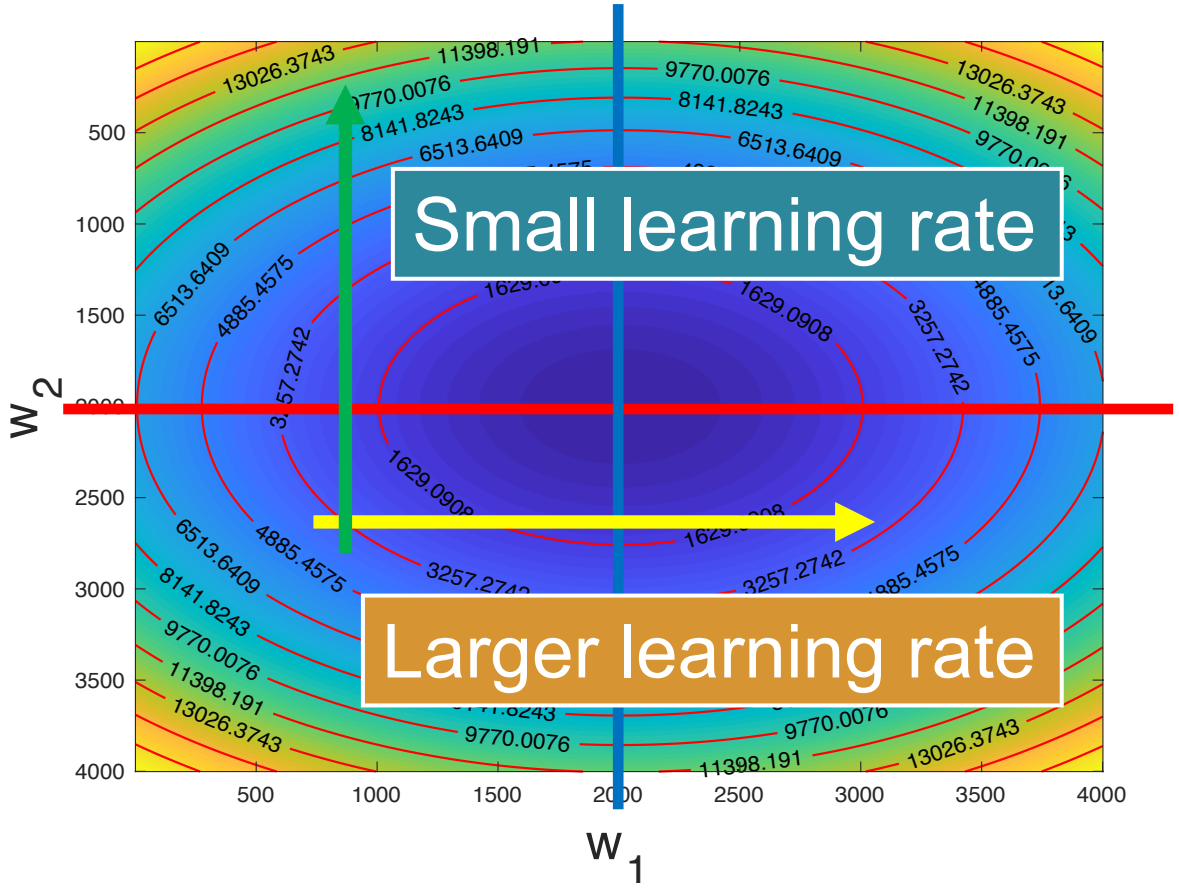
Train this thin and linear network

2. Tricks from training process

2.2 Adaptive learning rate

Previous Lecture

Adagrad



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

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

2. Tricks from training process

2.2 Adaptive learning rate

RMSProp

Root mean square of the gradients with previous gradients being decayed.

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

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

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

$$\vdots$$

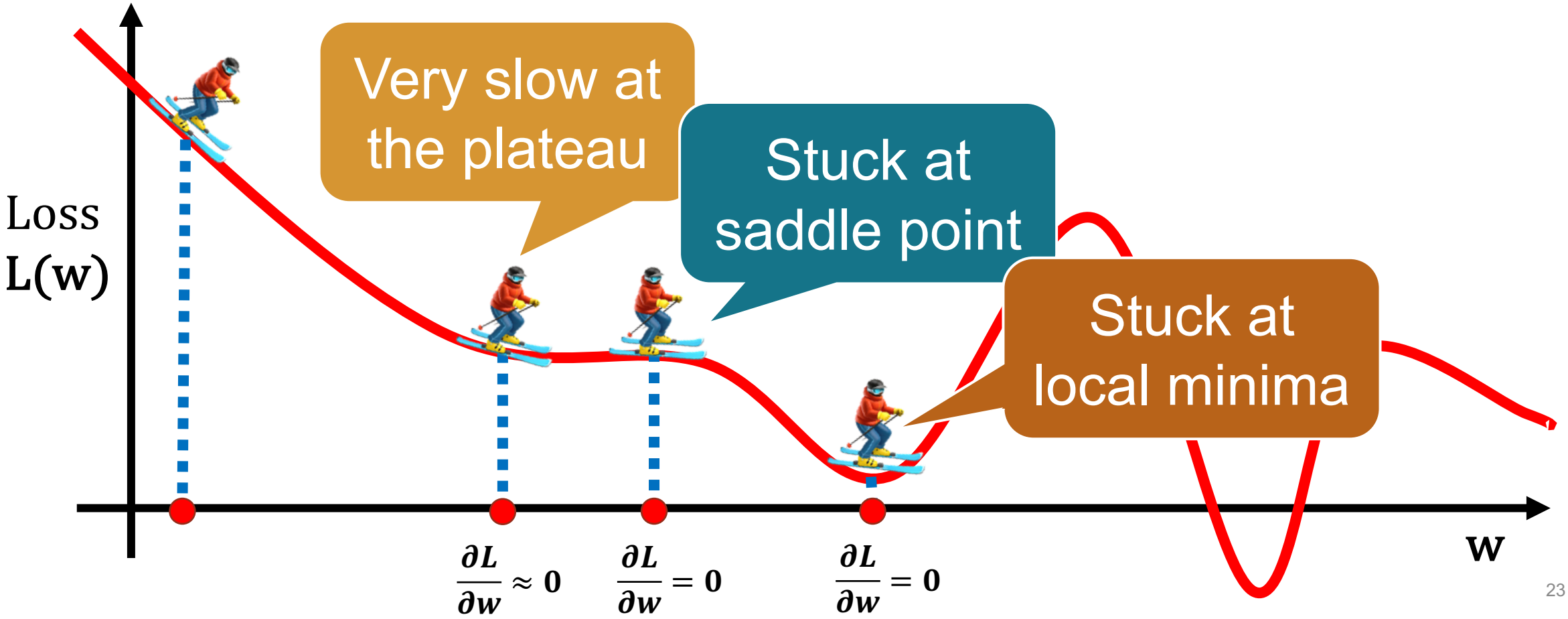
$$w^{t+1} = w^t - \frac{\eta}{\sigma^t} g^t \qquad \sigma^t = \sqrt{\alpha(\sigma^{t-1})^2 + (1-\alpha)(g^t)^2}$$

2. Tricks from training process

2.2 Adaptive learning rate

Previous Lecture

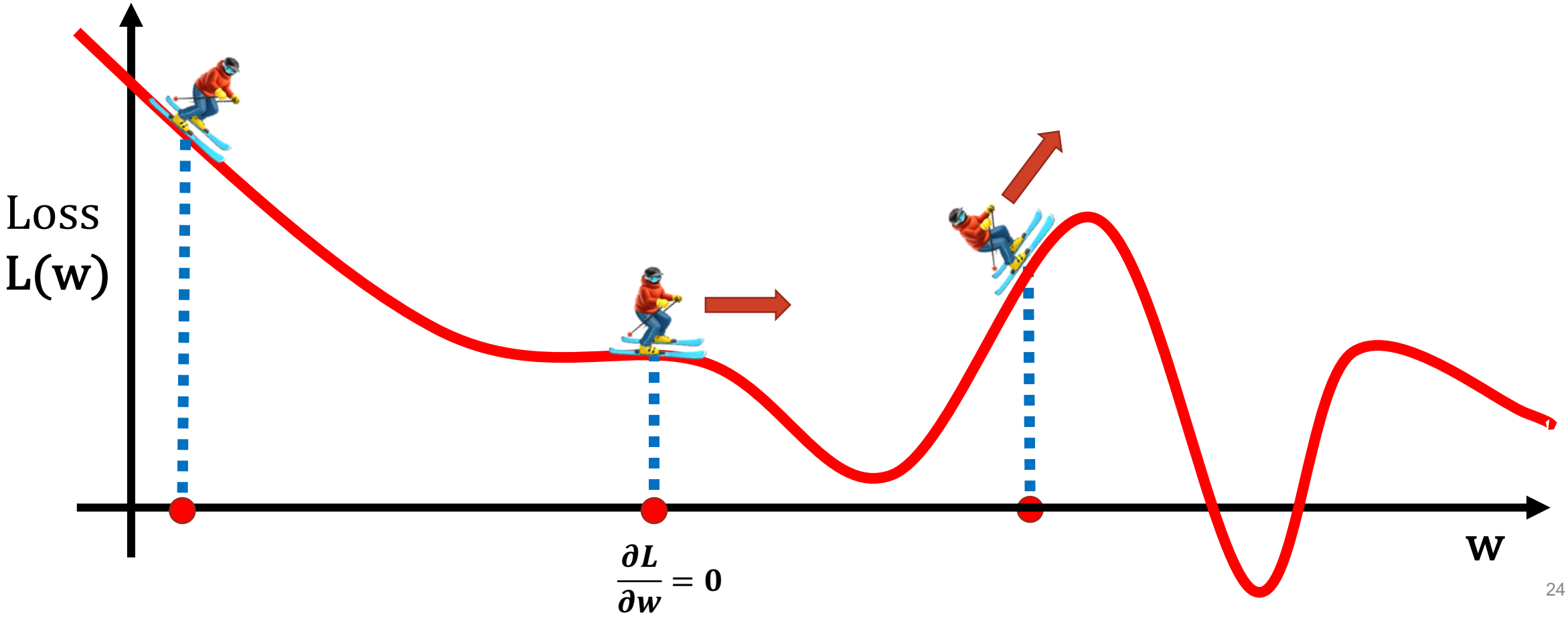
Momentum



2. Tricks from training process

2.2 Adaptive learning rate

Momentum

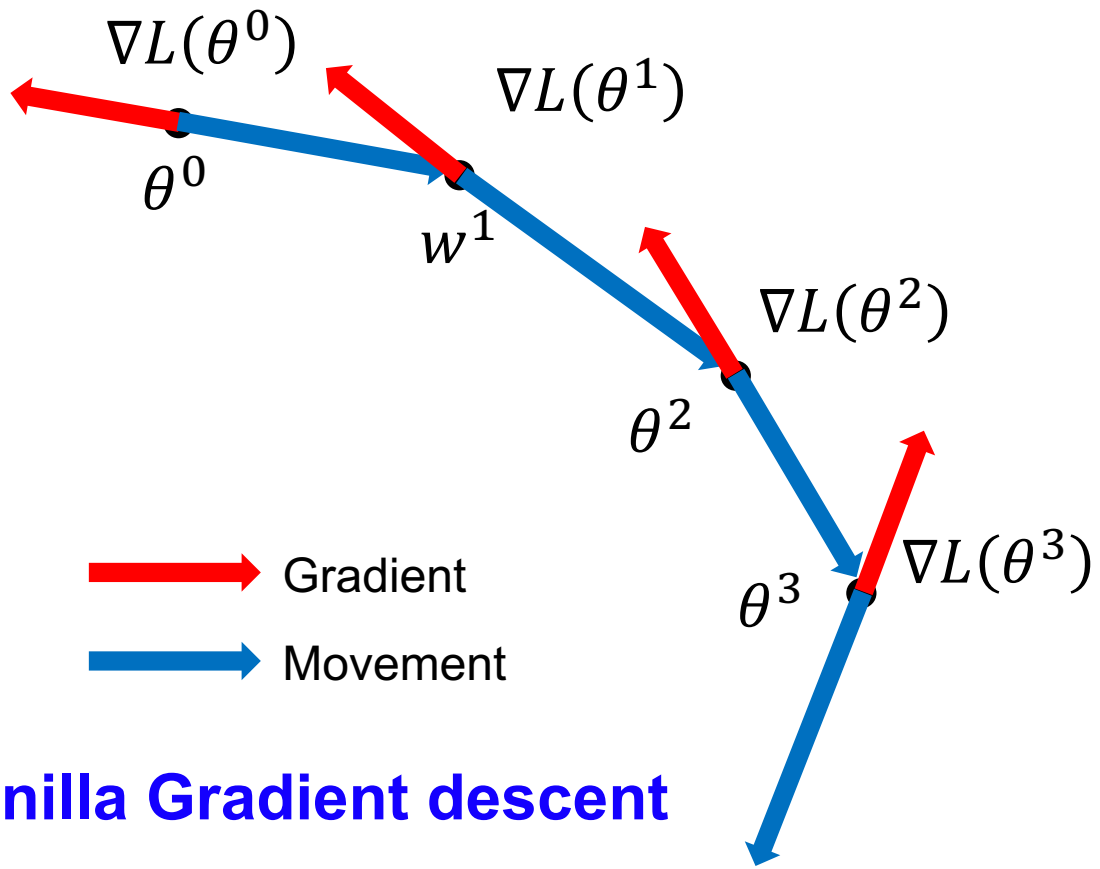


2. Tricks from training process

2.2 Adaptive learning rate

Previous Lecture

Momentum



Vanilla Gradient descent

Start at position θ^0

Calculate gradient at θ^0: $\nabla L(\theta^0)$

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

Calculate gradient at θ^1: $\nabla L(\theta^1)$

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

⋮

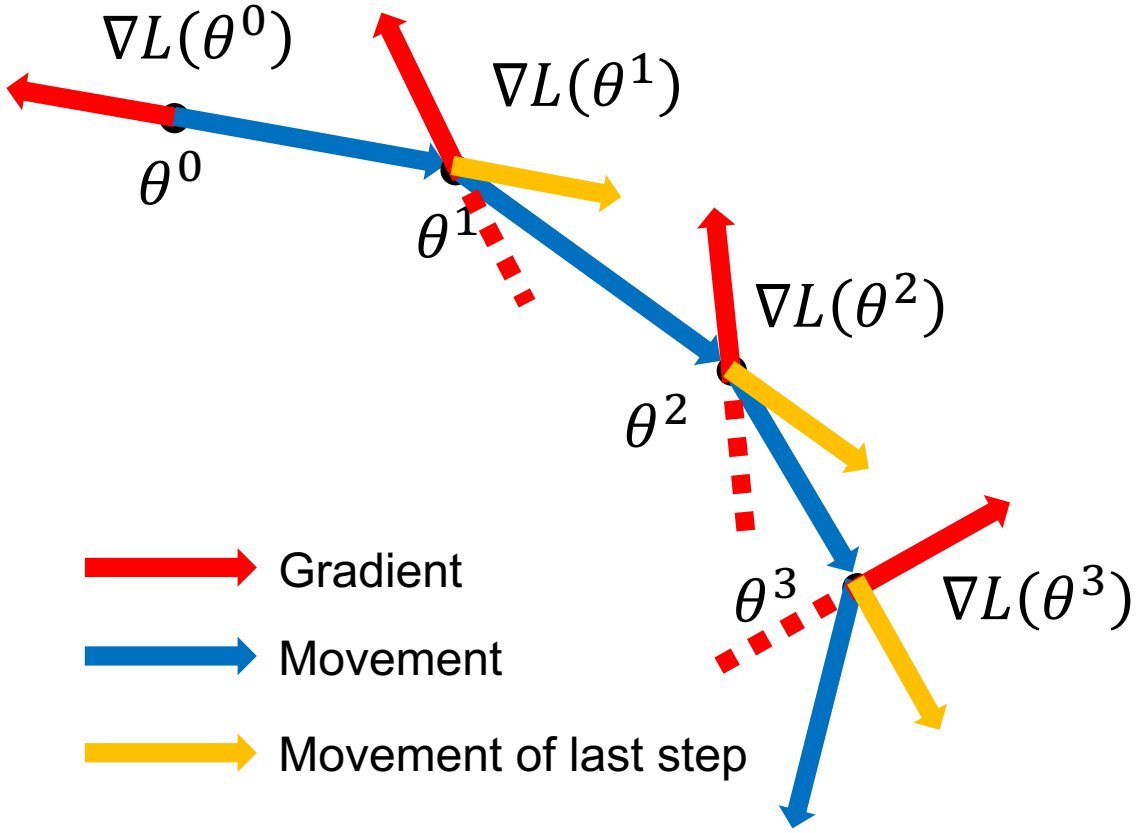
⋮

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

2. Tricks from training process

2.2 Adaptive learning rate

Momentum **Movement**: movement of last step minus gradient at present.



Start at position θ^0

Movement $v^0 = 0$

Calculate gradient at θ^0: $\nabla L(\theta^0)$

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

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

Calculate gradient at θ^1: $\nabla L(\theta^1)$

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

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

⋮

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

2. Tricks from training process

2.2 Adaptive learning rate

Momentum **Movement**: movement of last step minus gradient at present.

v^i is the weighted sum of all the previous gradient:

$$\nabla L(\theta^0), \nabla L(\theta^1), \ldots \nabla L\left(\theta^{i-1}\right)$$

$v^0 = 0$

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

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

Start at position θ^0

Movement $v^0 = 0$

Calculate gradient at θ^0: $\nabla L(\theta^0)$

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

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

Calculate gradient at θ^1: $\nabla L(\theta^1)$

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

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

⋮

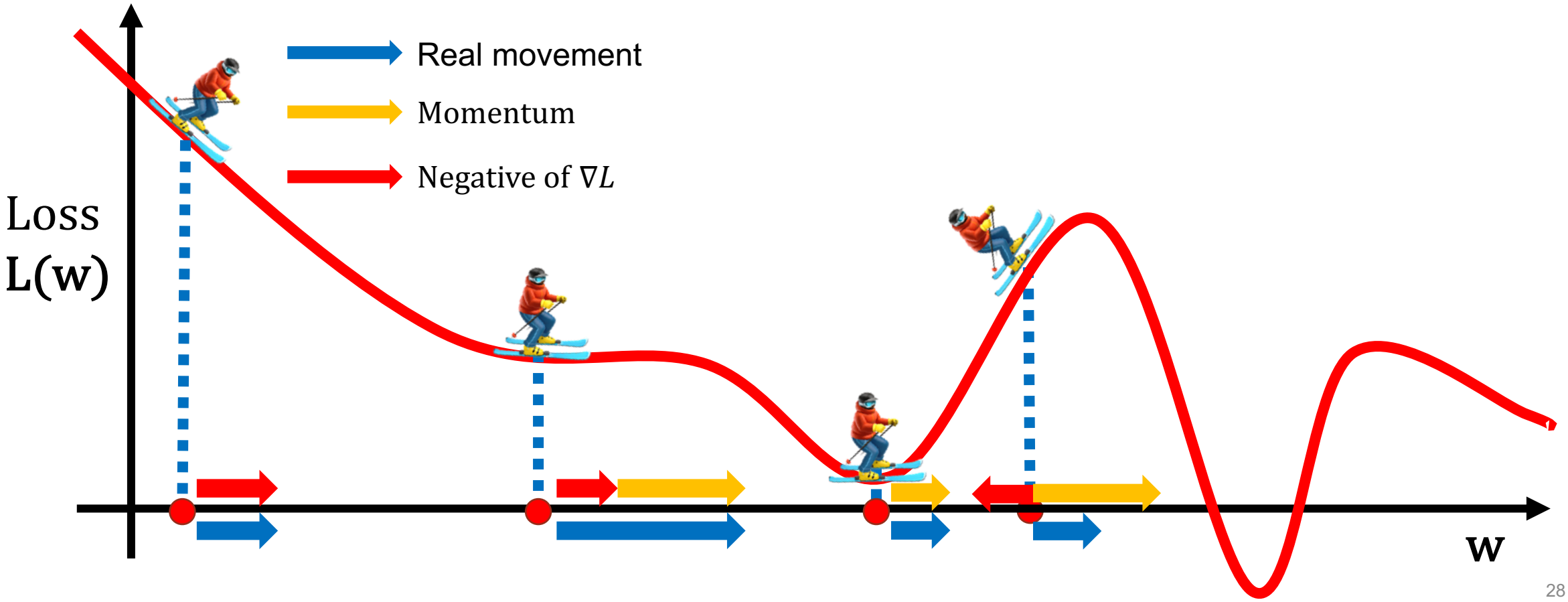
Movement is not only based on gradient, but also consider the previous movement.

2. Tricks from training process

2.2 Adaptive learning rate

Momentum

Movement = Negative of ∇L + Momentum



2. Tricks from training process

2.2 Adaptive learning rate

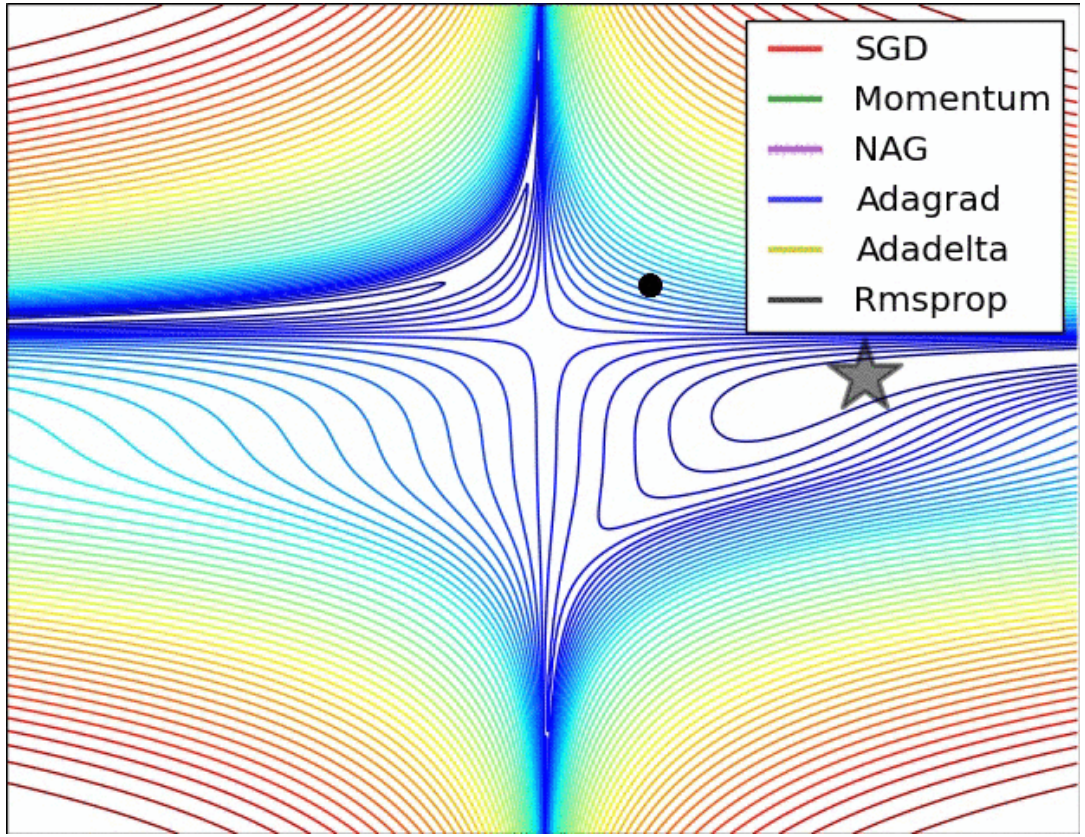
Momentum

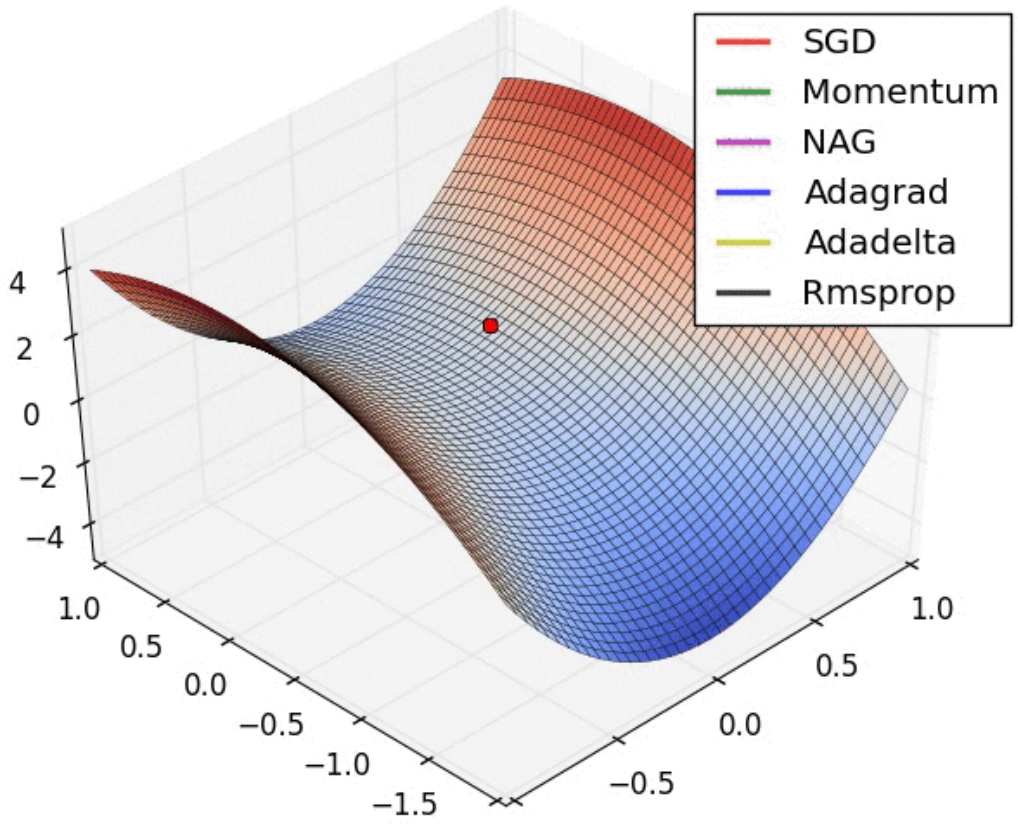
Adam =

RMSProp + **Momentum**

2. Tricks from training process

2.2 Adaptive learning rate

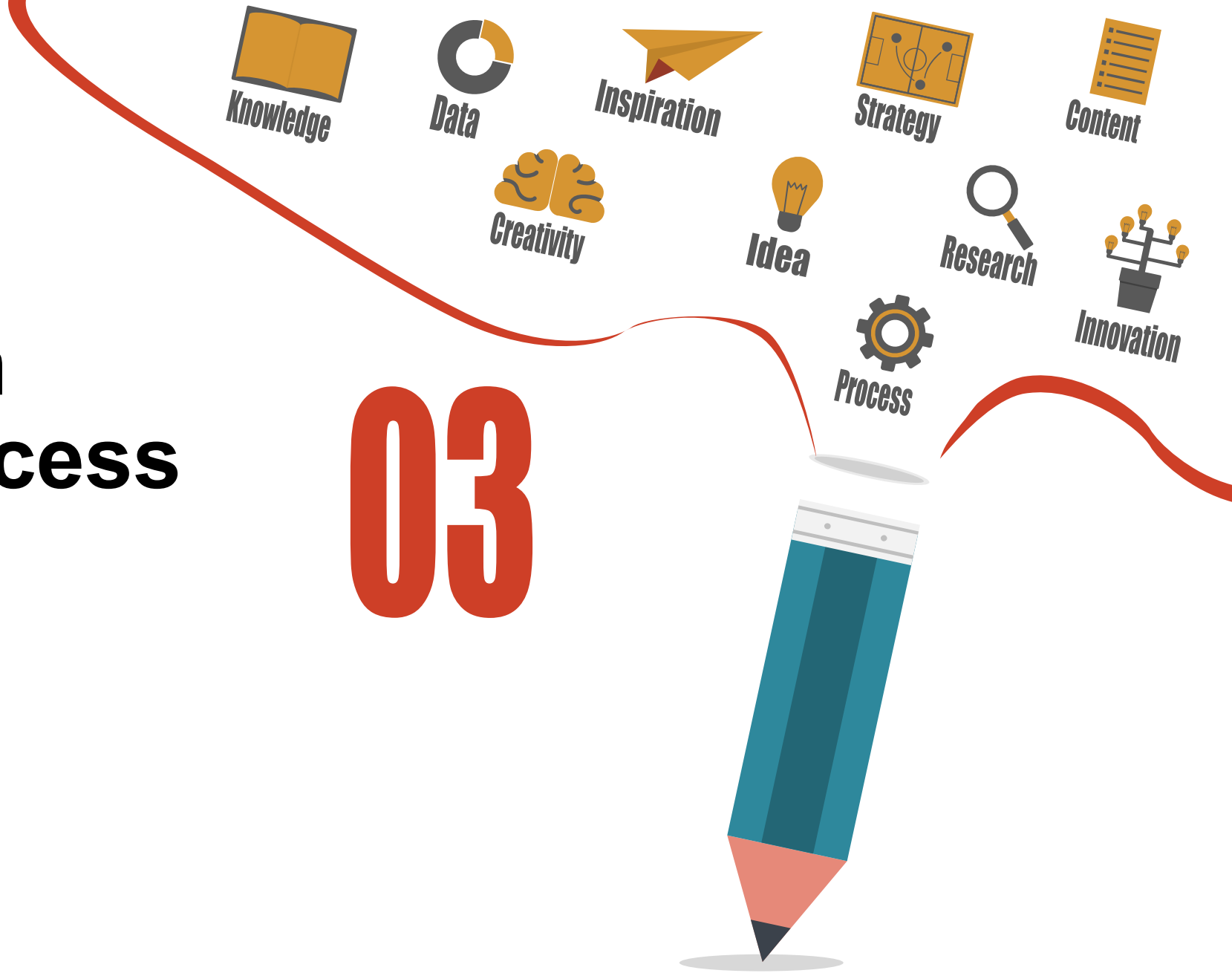




Cite: https://ruder.io/optimizing-gradient-descent/

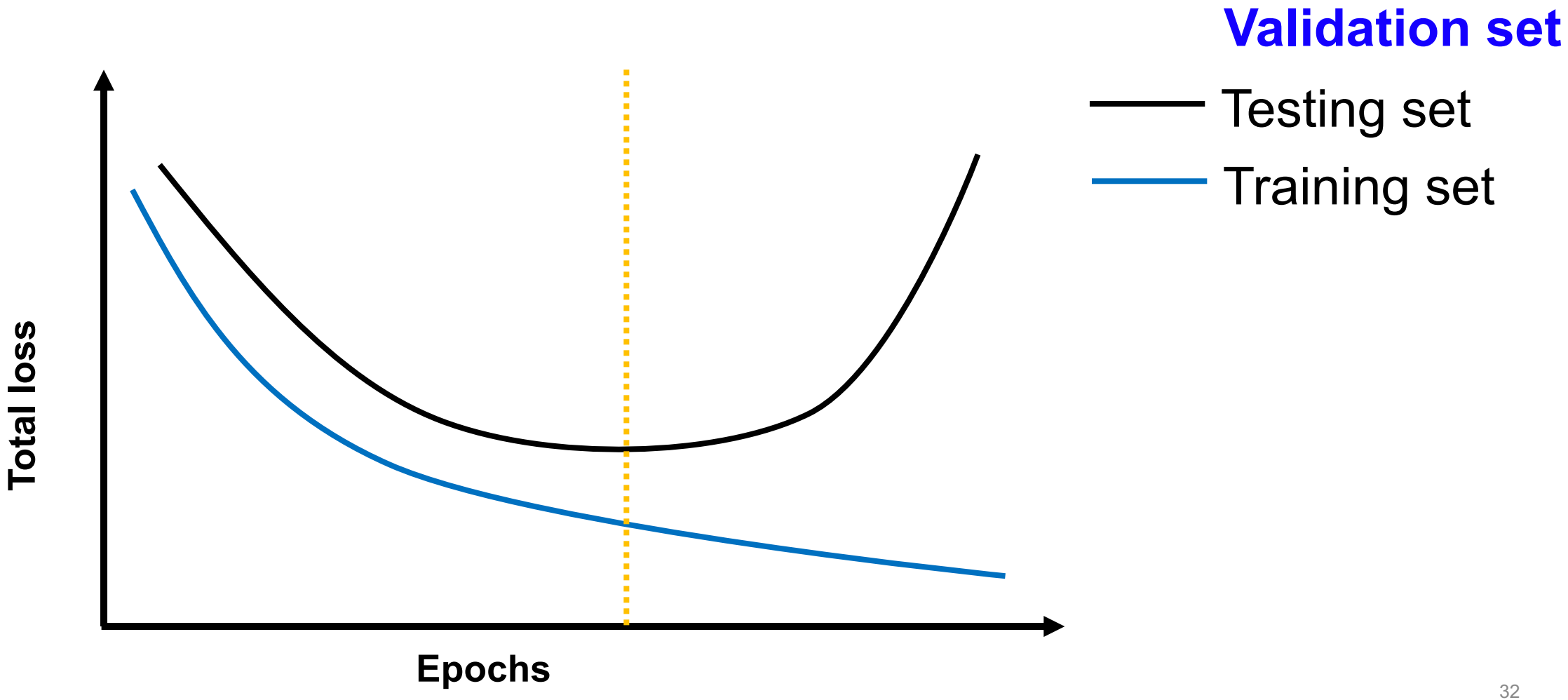
Tricks from testing process

03



3. Tricks from testing process

3.1 Early stopping



Keras tip: https://keras.io/getting_started/faq/#how-can-i-interrupt-training-when-the-validation-loss-isnt-decreasing-anymore

3. Tricks from testing process

3.2 Regularization

Previous Lecture

Regularization ——Adding a penalty on model complexity

$$L = \sum_{n} \left(\hat{y}^n - \left(b + \sum w_i x_i \right) \right)^2 + \lambda \sum (w_i)^2$$

L2-norm

Ridge regression

$$L = \sum_{n} \left(\hat{y}^n - \left(b + \sum w_i x_i \right) \right)^2 + \lambda \sum |w_i|$$

L1-norm

Lasso regression

Least **a**bsolute **s**hrinkage and **s**election **o**perator

3. Tricks from testing process

3.2 Regularization

New loss function to be minimized: not only minimizing original cost but also let it be close to zero

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

Regularization term

Original loss

e.g. minimize square error, cross entropy …

$$\theta = \{w_1, w_2, \dots\dots\}$$

L2 regularization: $\|\theta\|^2 = (w_1)^2 + (w_2)^2 + \cdots$

3. Tricks from testing process

3.2 Regularization

L2 regularization:

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

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

Update:

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

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

Close to zero

Weight decay

3. Tricks from testing process

3.2 Regularization

L1 regularization: $\|\theta\|^1 = |w_1| + |w_2| + \cdots$

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

Gradient: $\frac{\partial \mathrm{L}'}{\partial w} = \frac{\partial L}{\partial w} + \lambda sign(w)$

Update:

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

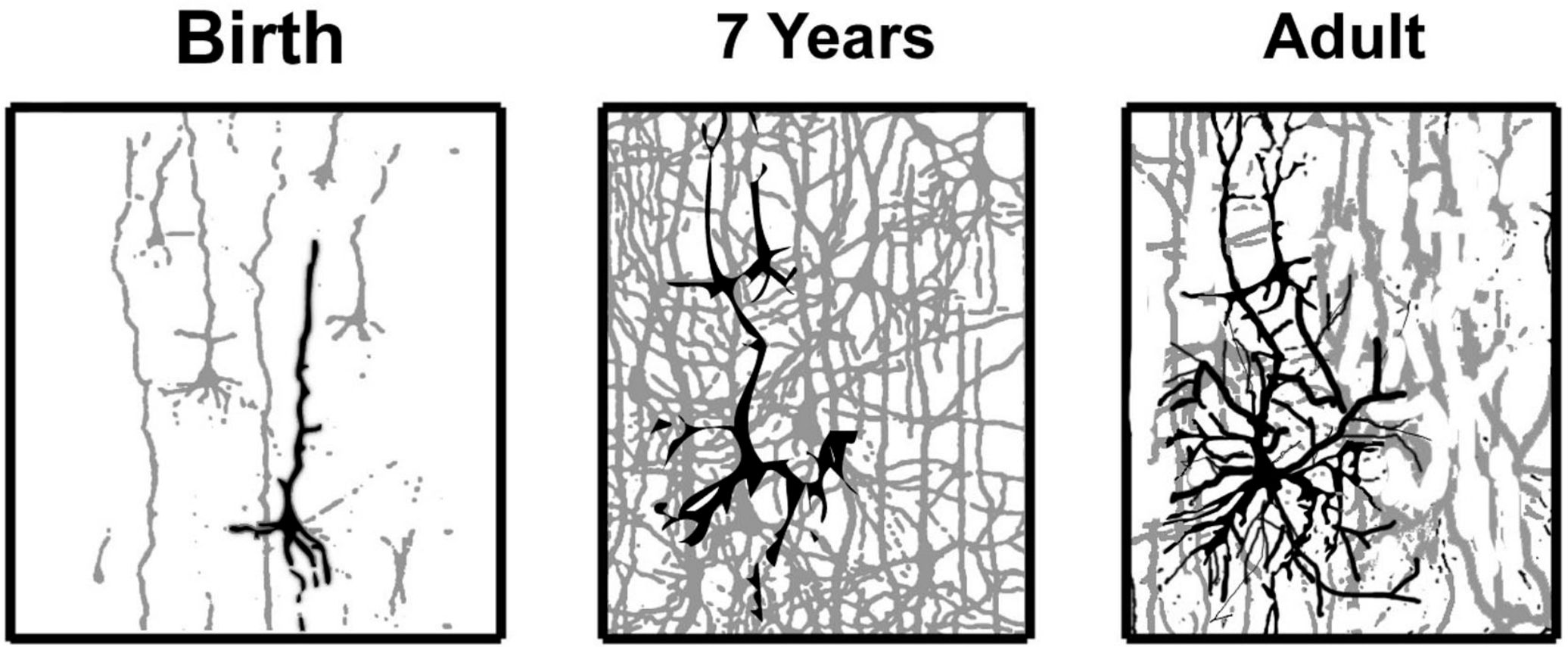
$$= w^t - \eta \frac{\partial L}{\partial w} - \eta\lambda sign(w^t)$$ **Always delete**

$$(1 - \eta\lambda) w^t - \eta \frac{\partial L}{\partial w}$$

3. Tricks from testing process

3.2 Regularization

Weight decay



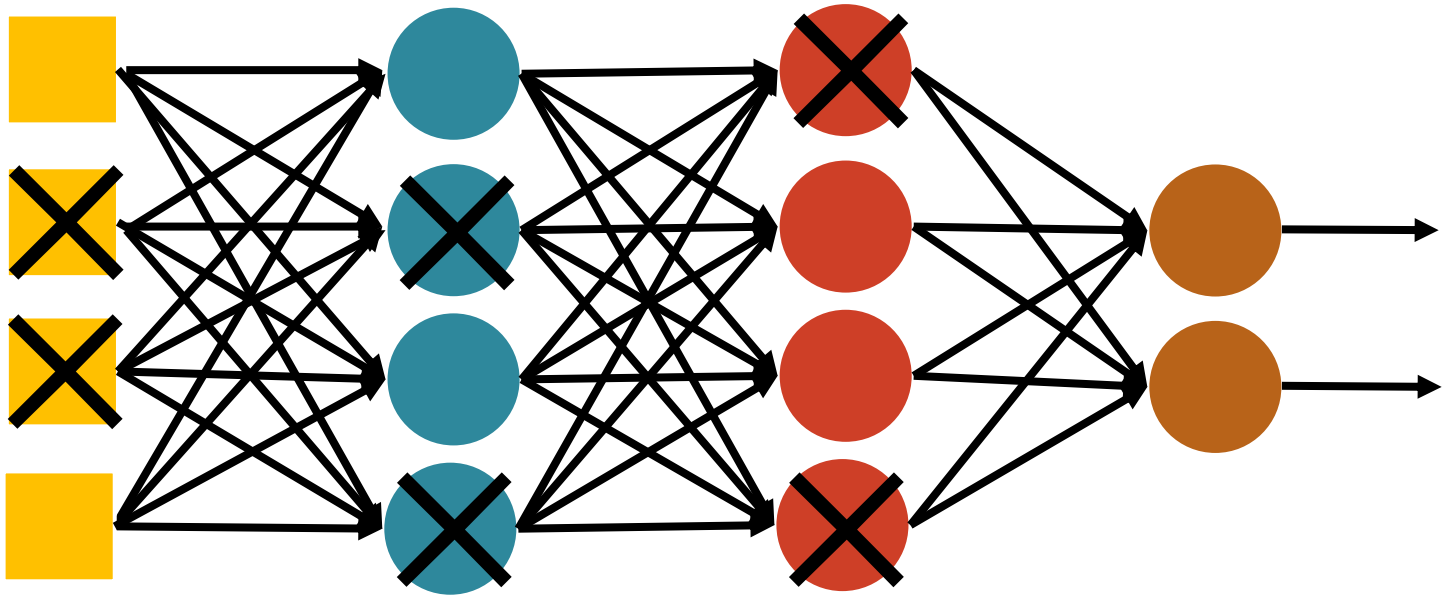
cite: https://www.psywww.com/gst/creativity.html

3. Tricks from testing process

3.3 Dropout

For each mini-batch, we resample the dropout neurons

Training

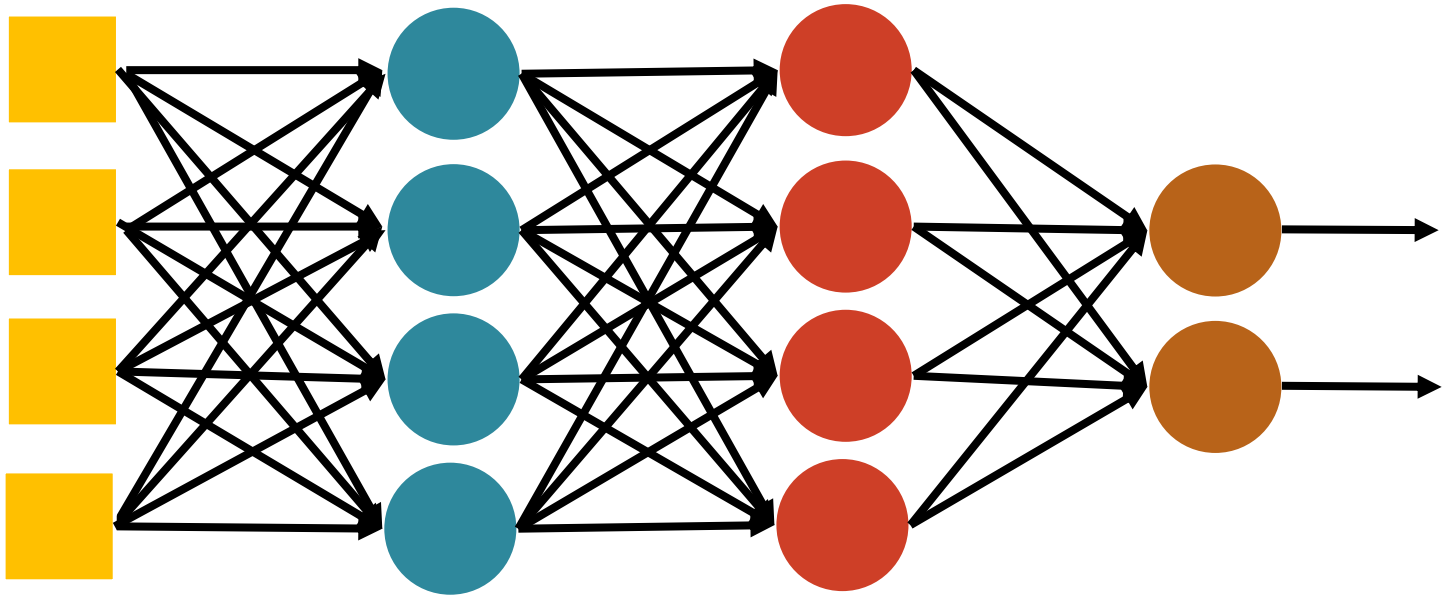


- Each time before updating the parameters：
  - Each neuron has probability p to dropout, then the structure of the network is changed;
  - Using the new network for training.

3. Tricks from testing process

3.3 Dropout

Testing



No dropout

- For each neuron has probability p to dropout in training process, the weight times $(1-p)$;
- E.g., Assume the dropout rate is 40%, if a weight $w = 1$ by training, set $w = 0.6$ for testing.

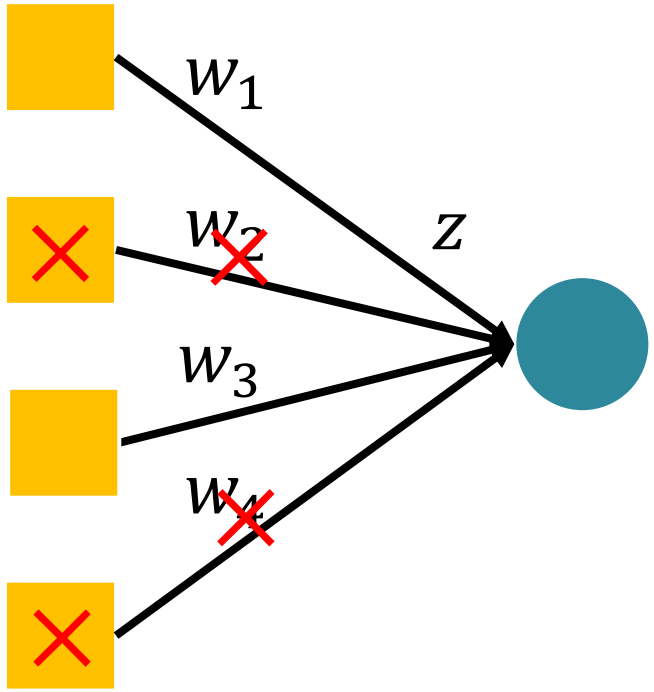
3. Tricks from testing process

3.3 Dropout

Why need the weights multiply $(1-p)$ when testing?

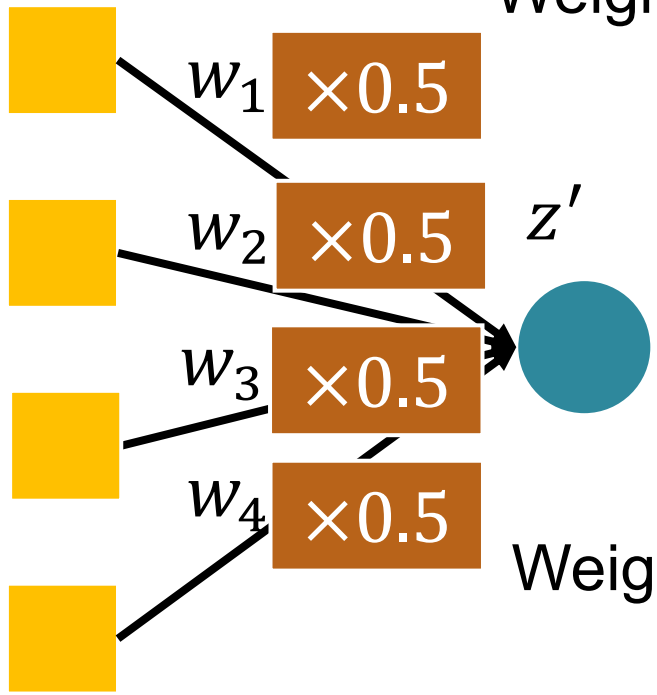
Training processing

Assume the dropout rate is 50%



Testing processing

No dropout



Weights from training

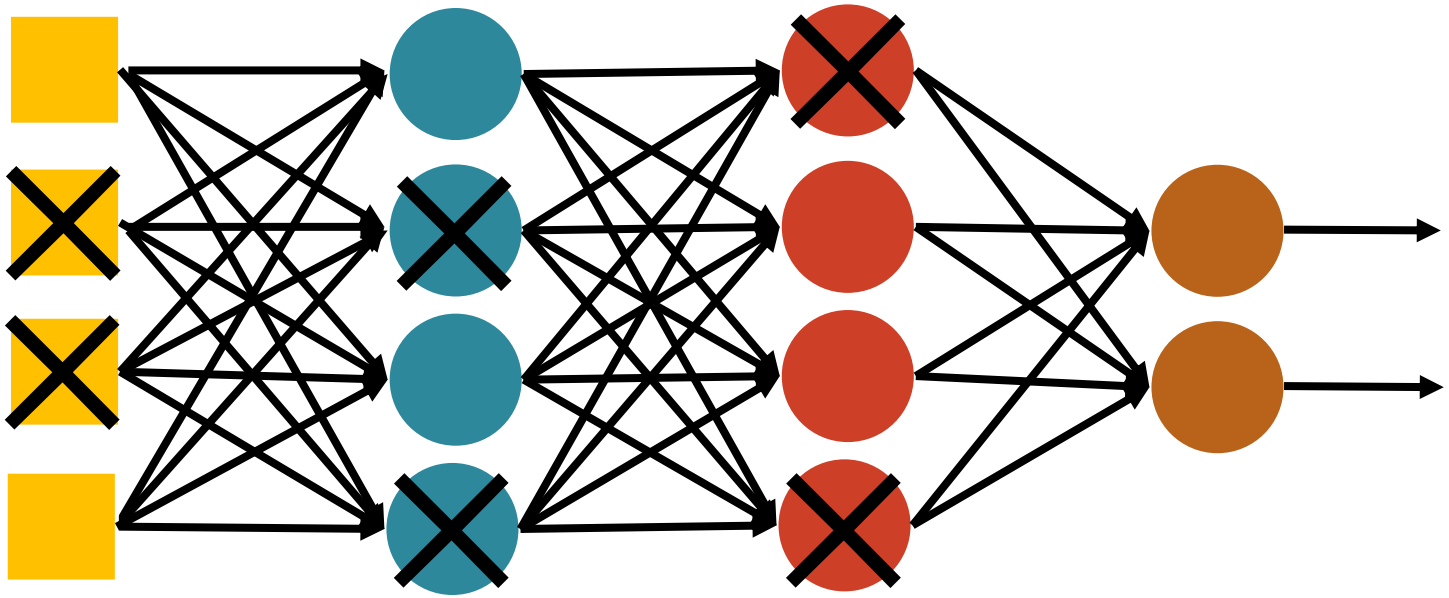
$z' \approx 2z$

Weights $\times(1-p)$

$z' \approx z$

3. Tricks from testing process

3.3 Dropout

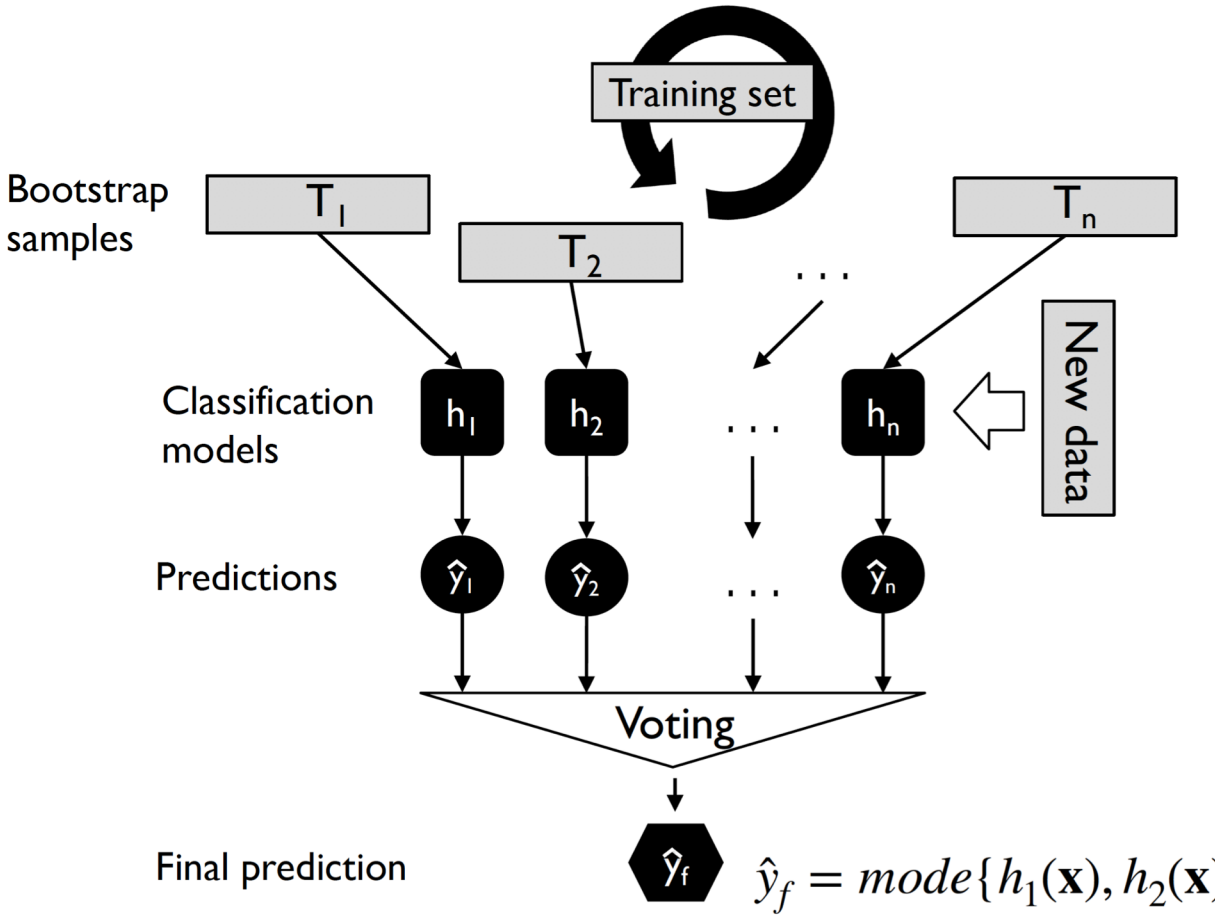


- **Considering a teamwork**
- If everyone expect the partner will do the work but do nothing himself, nothing will be done finally;
- However, if everyone know some partner will dropout, you need to work harder to finish the task;
- When submitting (testing), no one dropout actually, the results would be good eventually.

3. Tricks from testing process

3.3 Dropout -- A kind of ensemble

Previous Lecture

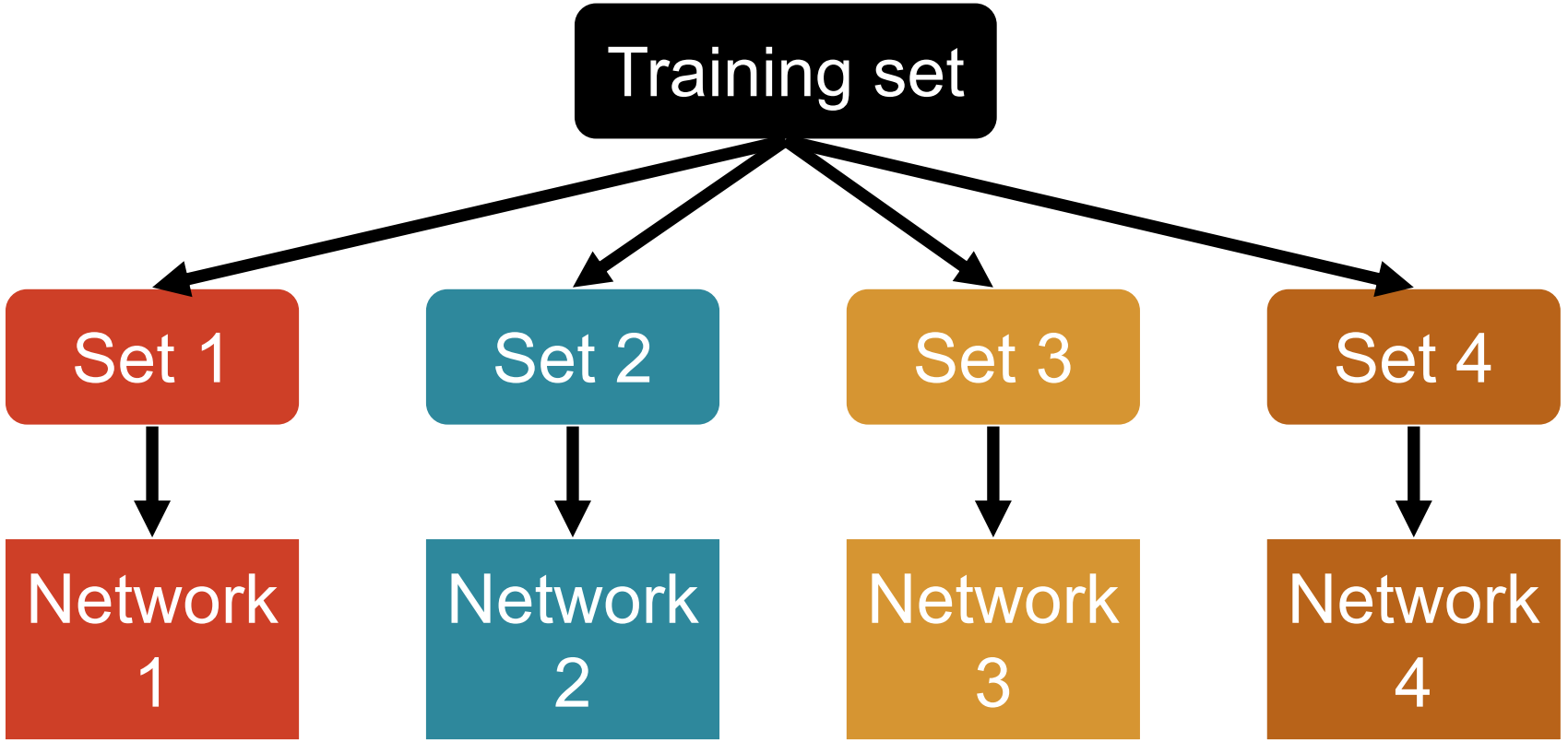


$$\hat{y}_f = mode\{h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots h_n(\mathbf{x})\}$$

where $h_i(\mathbf{x}) = \hat{y}_i$

3. Tricks from testing process

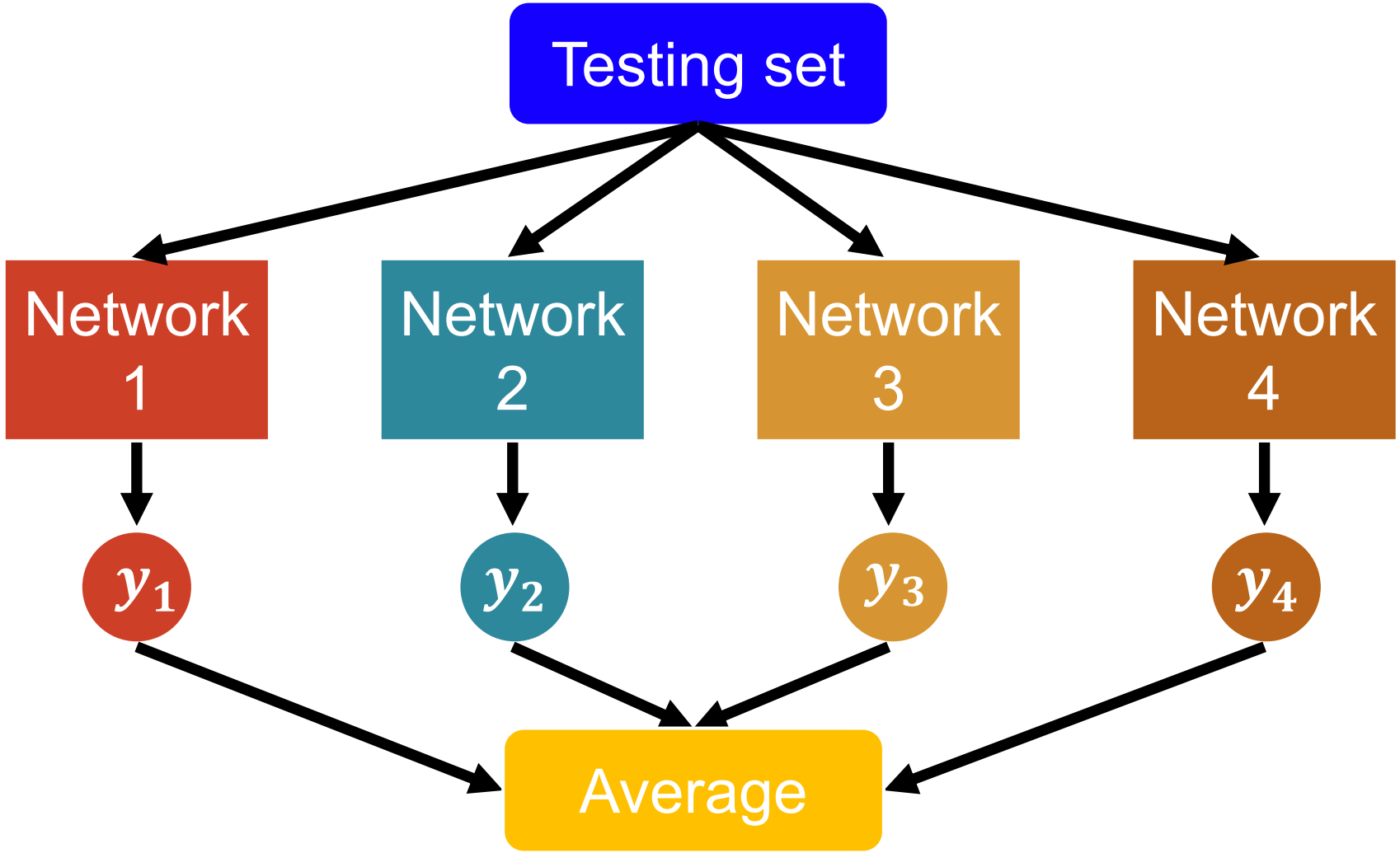
3.3 Dropout -- A kind of ensemble



Train a bunch of networks with different structures

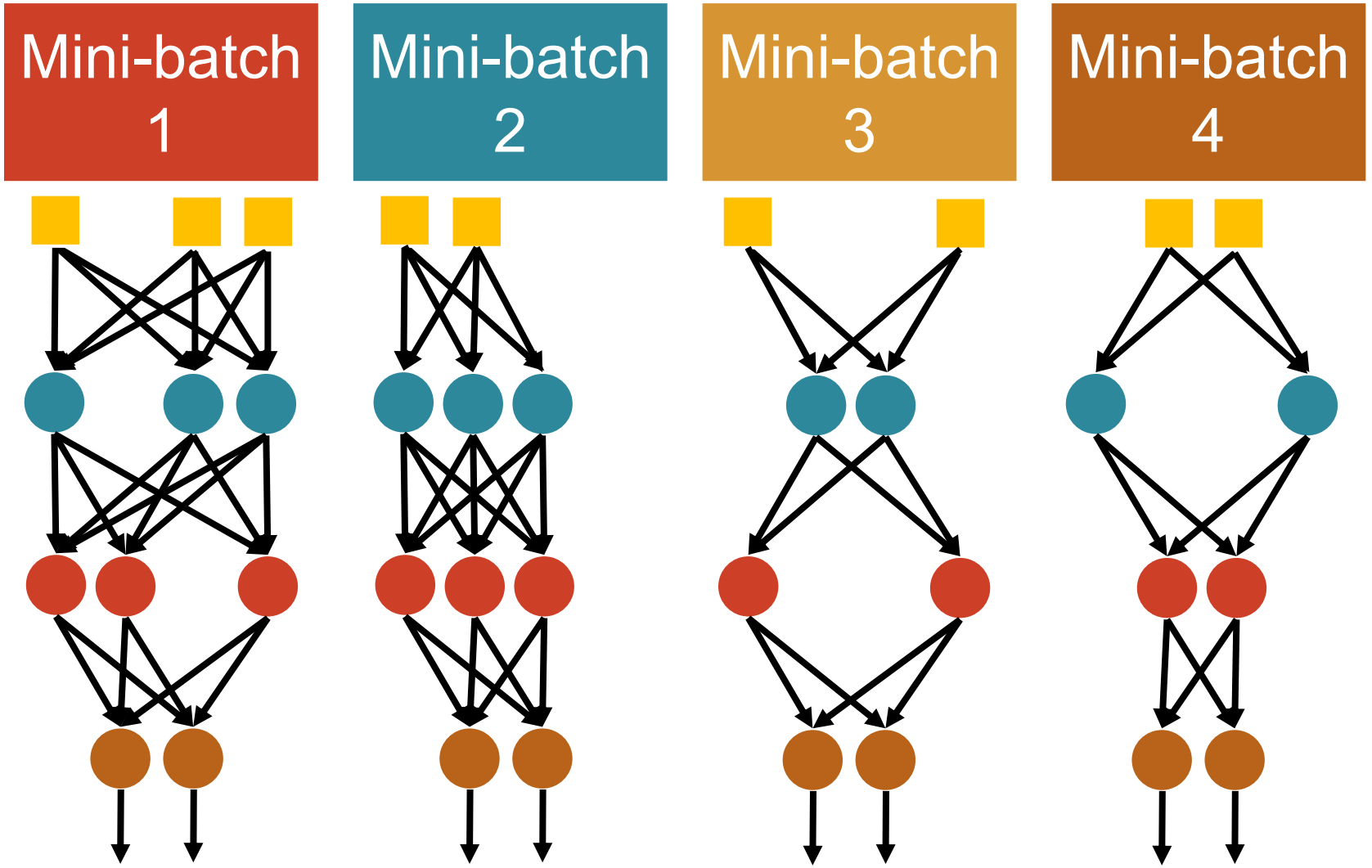
3. Tricks from testing process

3.3 Dropout -- A kind of ensemble



3. Tricks from testing process

3.3 Dropout -- A kind of ensemble



Training of dropout

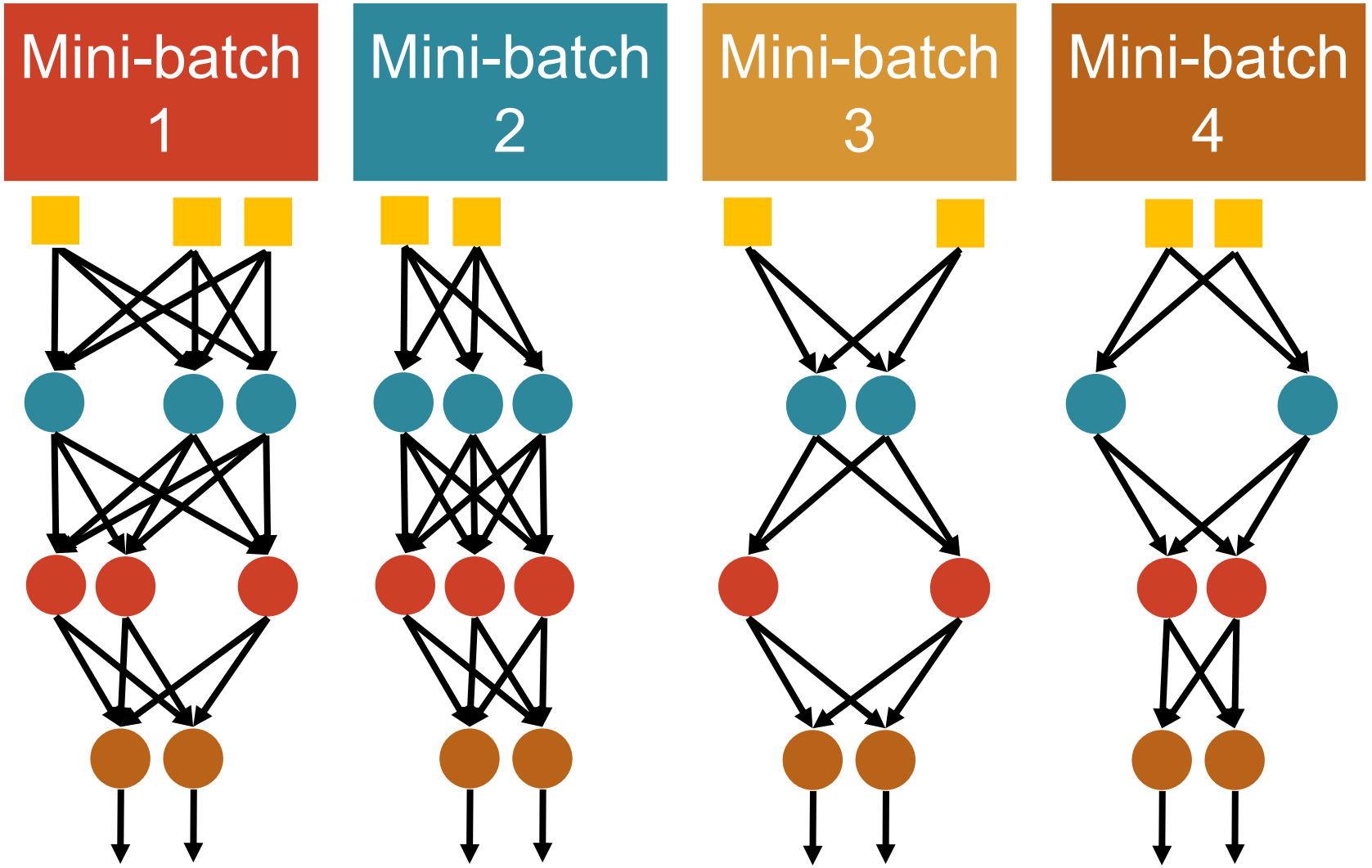
Assume M neurons;

Can dropout or not;

2^M possible networks.

3. Tricks from testing process

3.3 Dropout -- A kind of ensemble

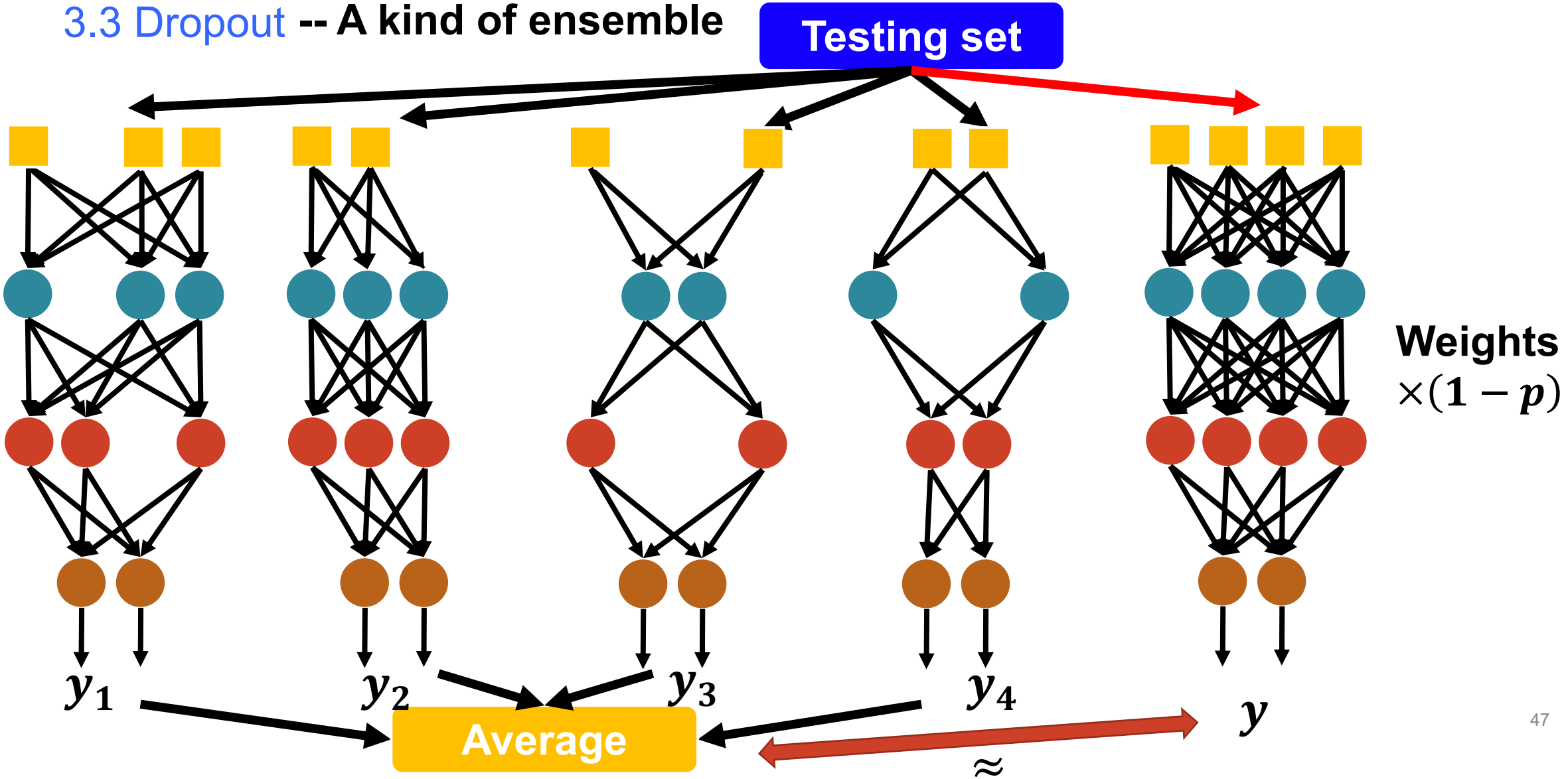


Training of dropout

- Using one mini-batch to train one network;
- Some parameters in the network are shared.

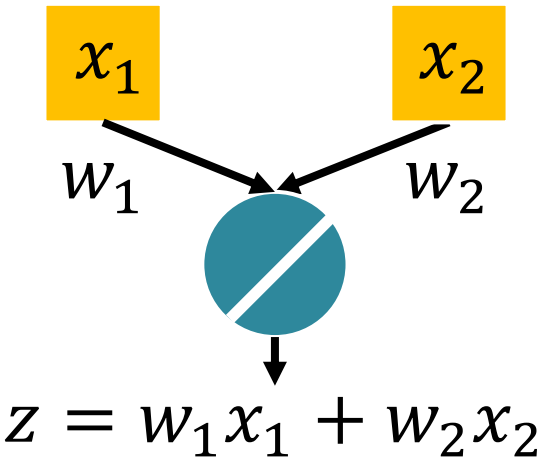
3. Tricks from testing process

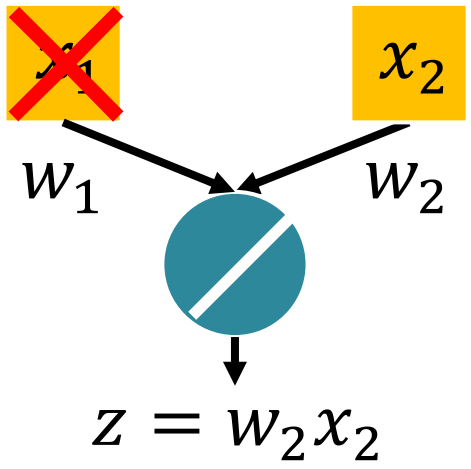
3.3 Dropout -- A kind of ensemble

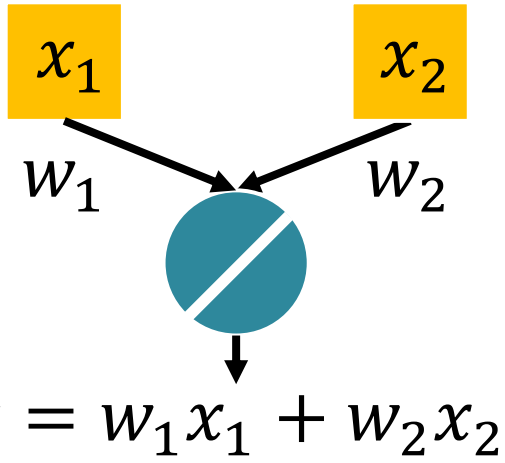


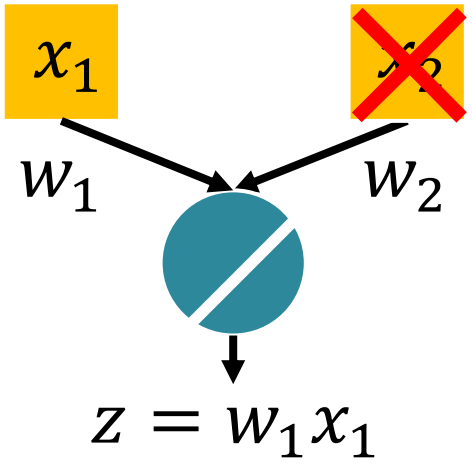
3. Tricks from testing process

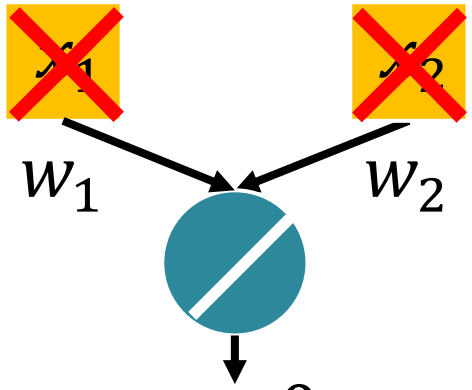
3.3 Dropout -- A kind of ensemble

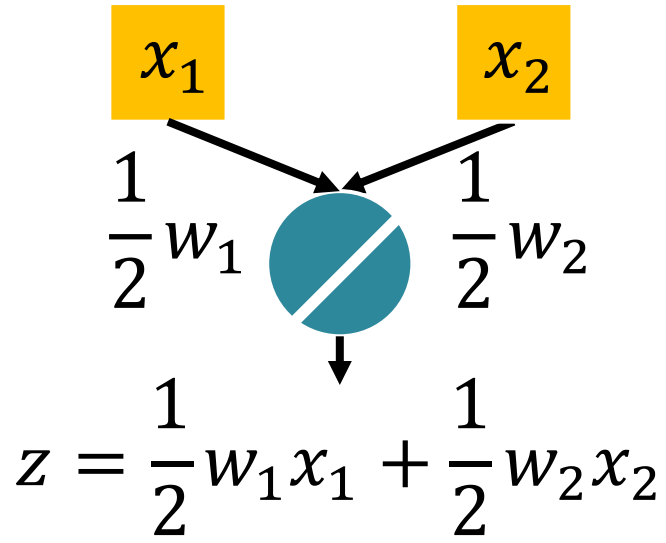












Tricks for Training Artificial Neural Networks

Hao, Qi

School of Astronomy and Space Science

THANKS