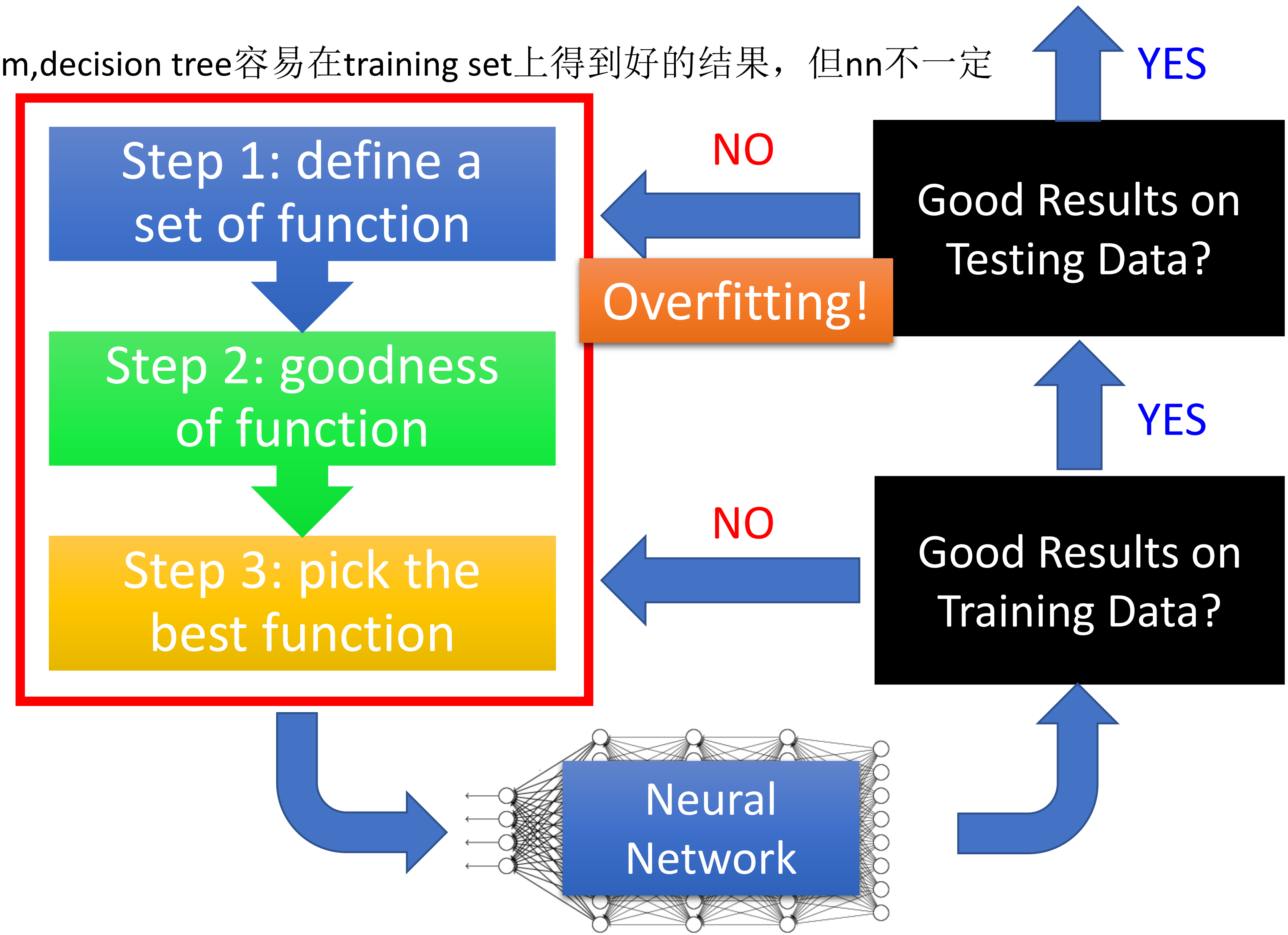
Tips for Deep Learning

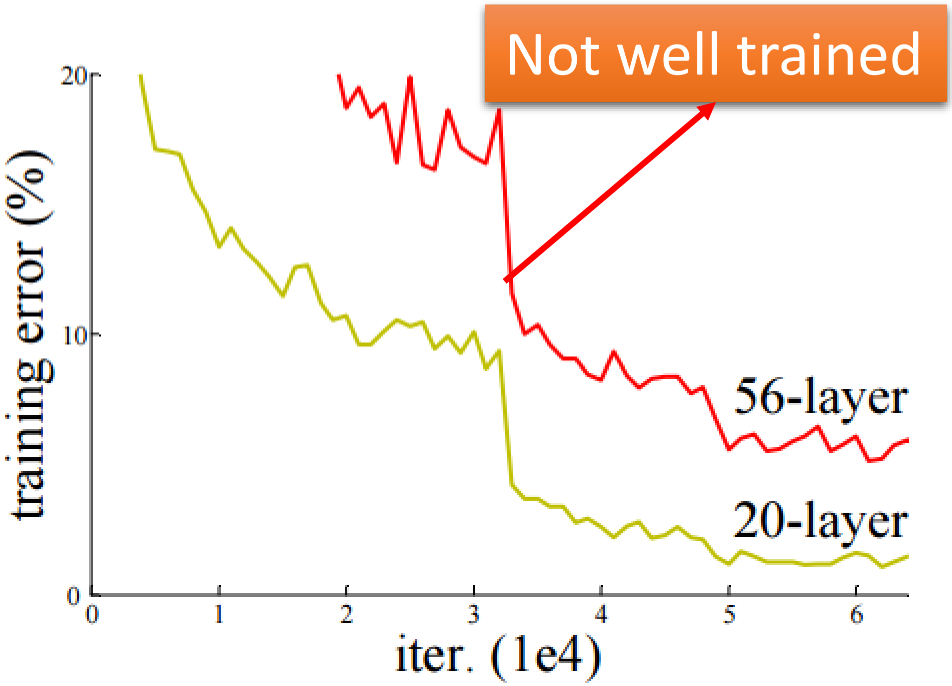
Recipe of Deep Learning

svm,decision tree容易在training set上得到好的结果，但nn不一定

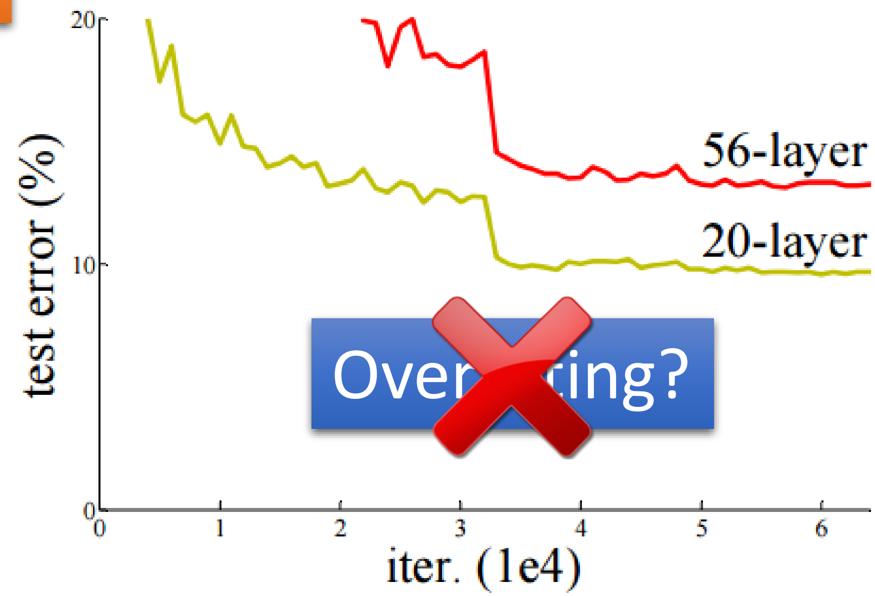


Do not always blame Overfitting

不能仅仅看testing error，要先关注trainging error.



Training Data



Testing Data

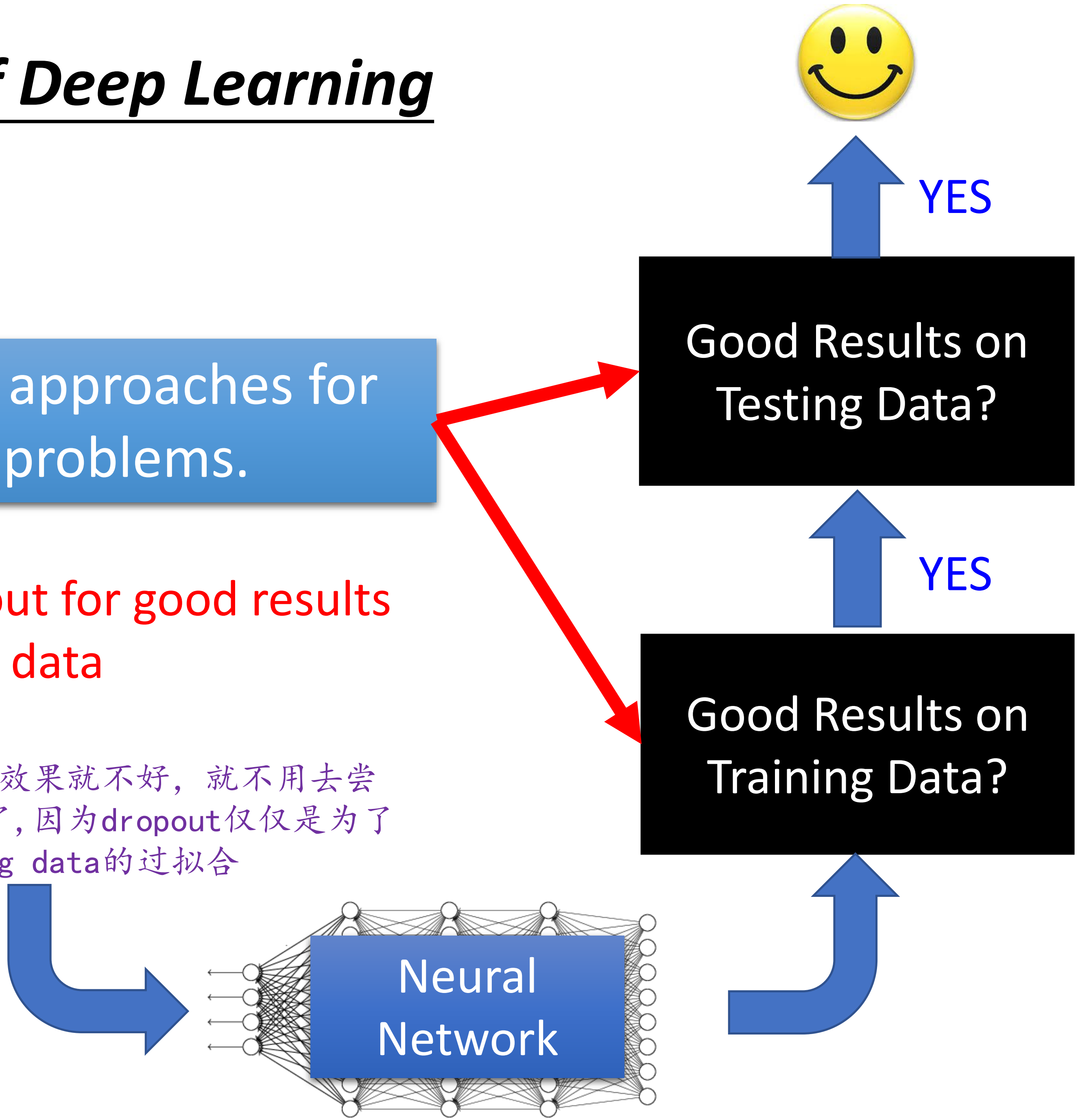
复杂的模型不见得就一定会比简单的，层数少的模型
得到更小的training error, 很可能在一个局部极植上

Recipe of Deep Learning

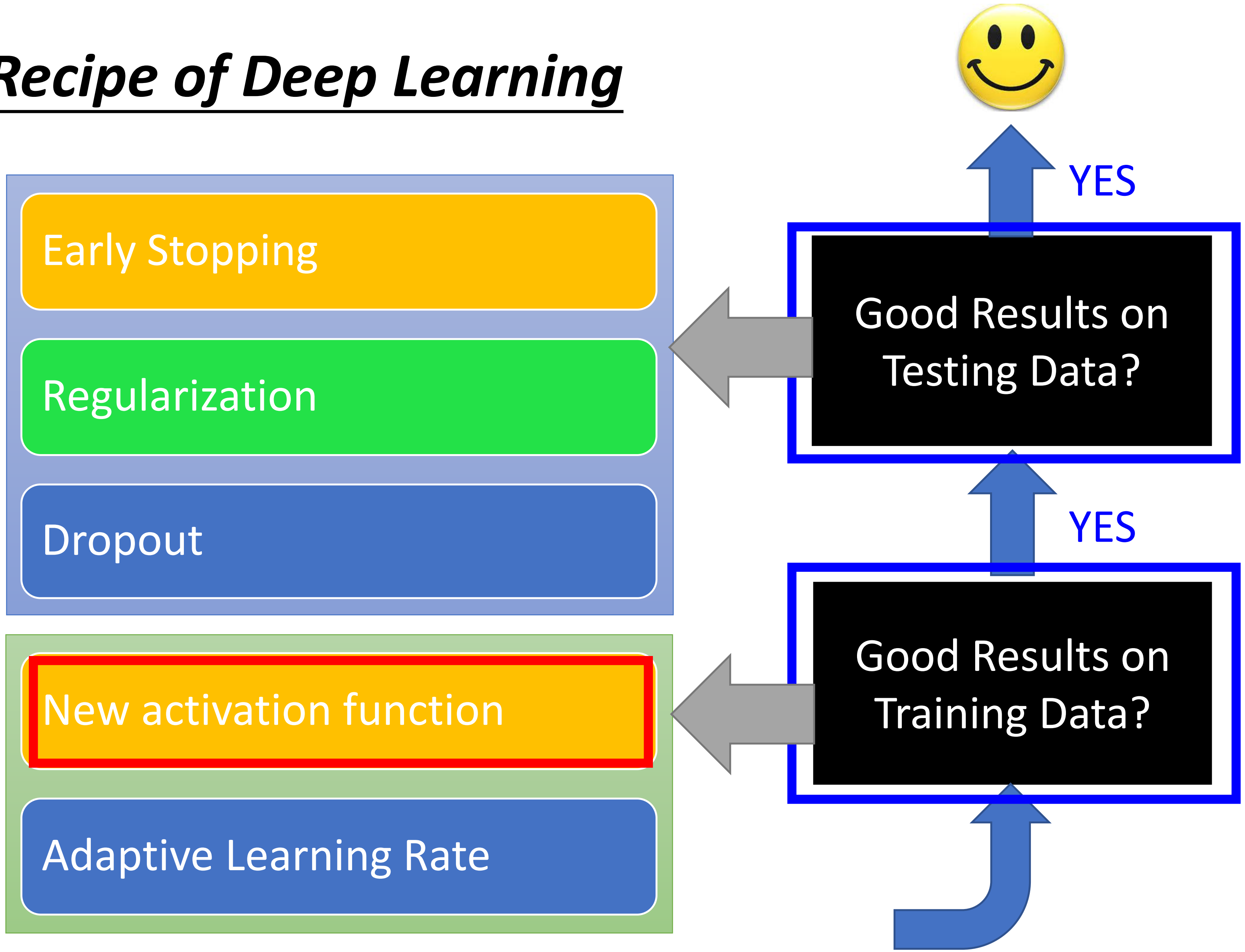
Different approaches for different problems.

e.g. dropout for good results on testing data

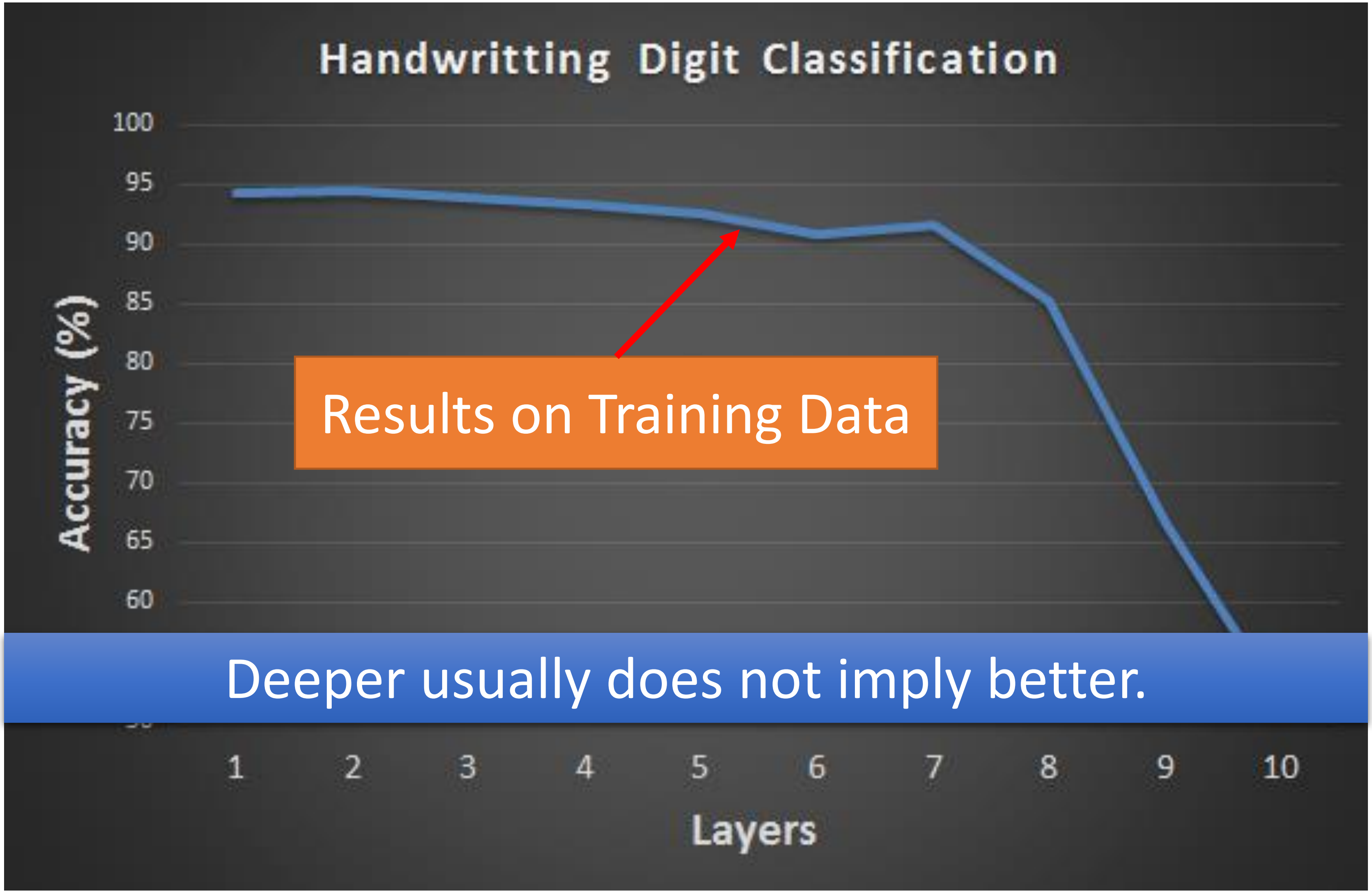
如果训练集上效果就不好，就不用去尝试加dropout了，因为dropout仅仅是为了防止在testing data的过拟合



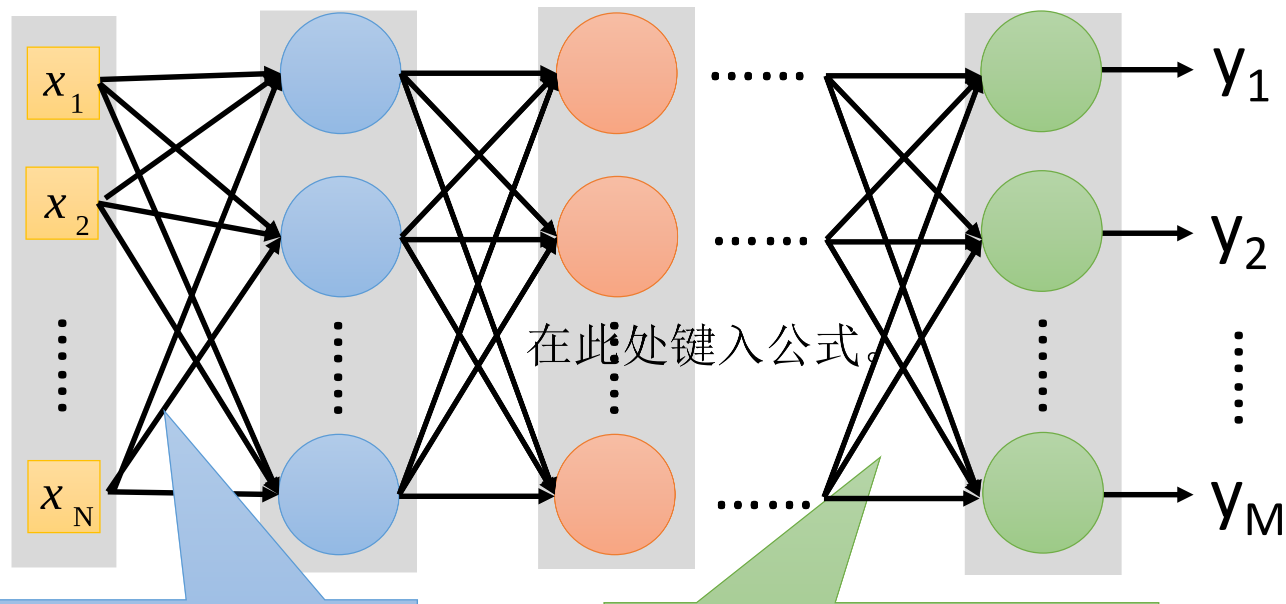
Recipe of Deep Learning



Hard to get the power of Deep …



Vanishing Gradient Problem



Smaller gradients

Learn very slow

Almost random

Larger gradients

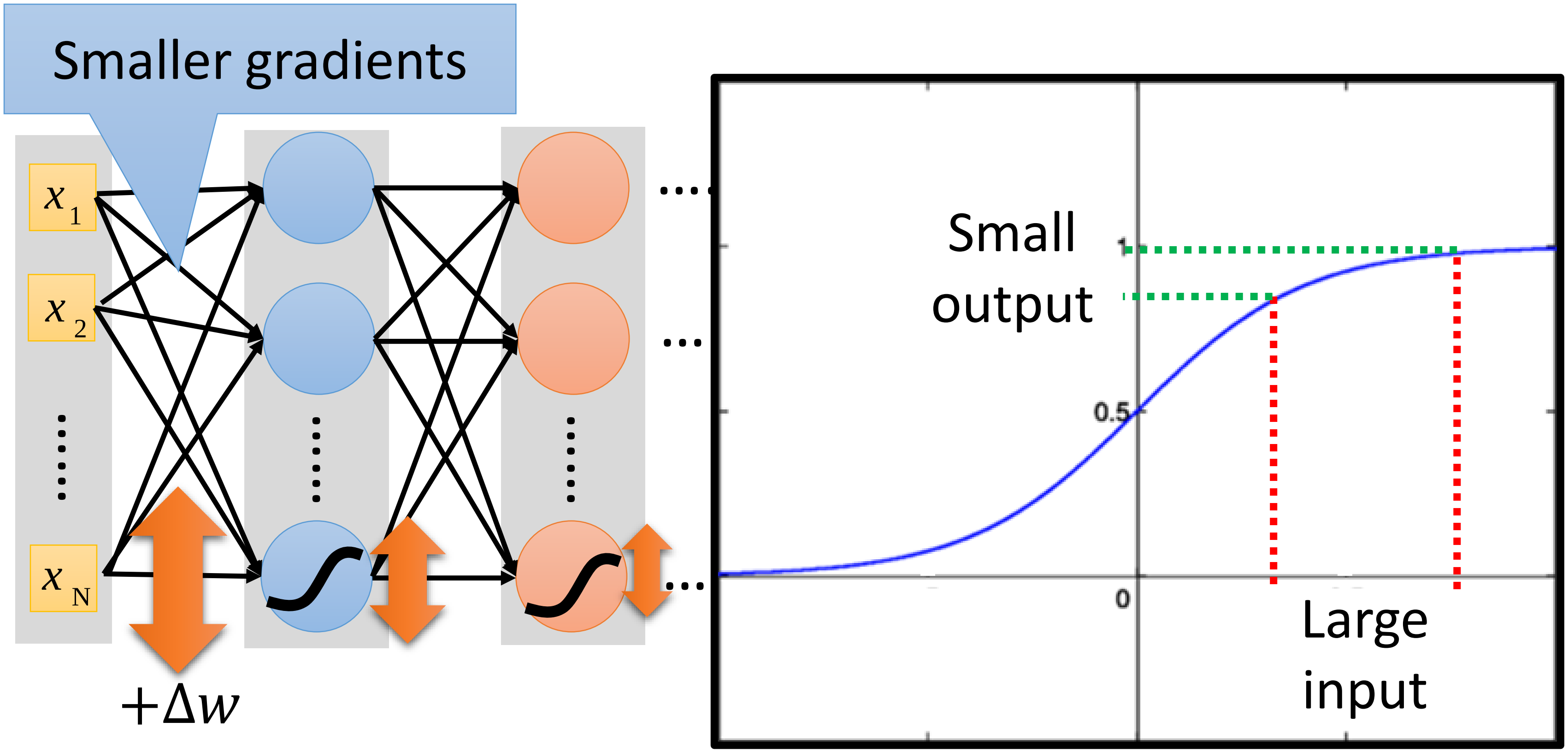
Learn very fast

Already converge

based on random!?

网络最后也会
layer收敛了，
随机的Value

Vanishing Gradient Problem



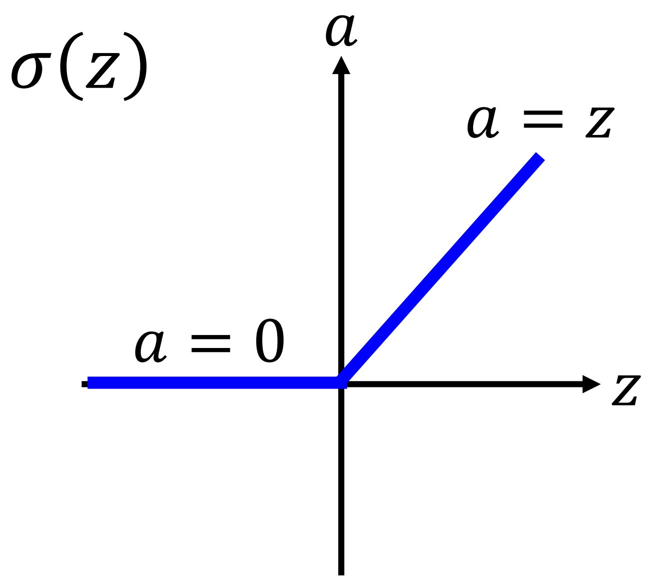
Intuitive way to compute the derivatives …

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

既然越靠近Input的layer，梯度越难传播过来，那是否可以考虑，
越靠近input的layer，使用越大的learning rata

ReLU

- Rectified Linear Unit (ReLU)

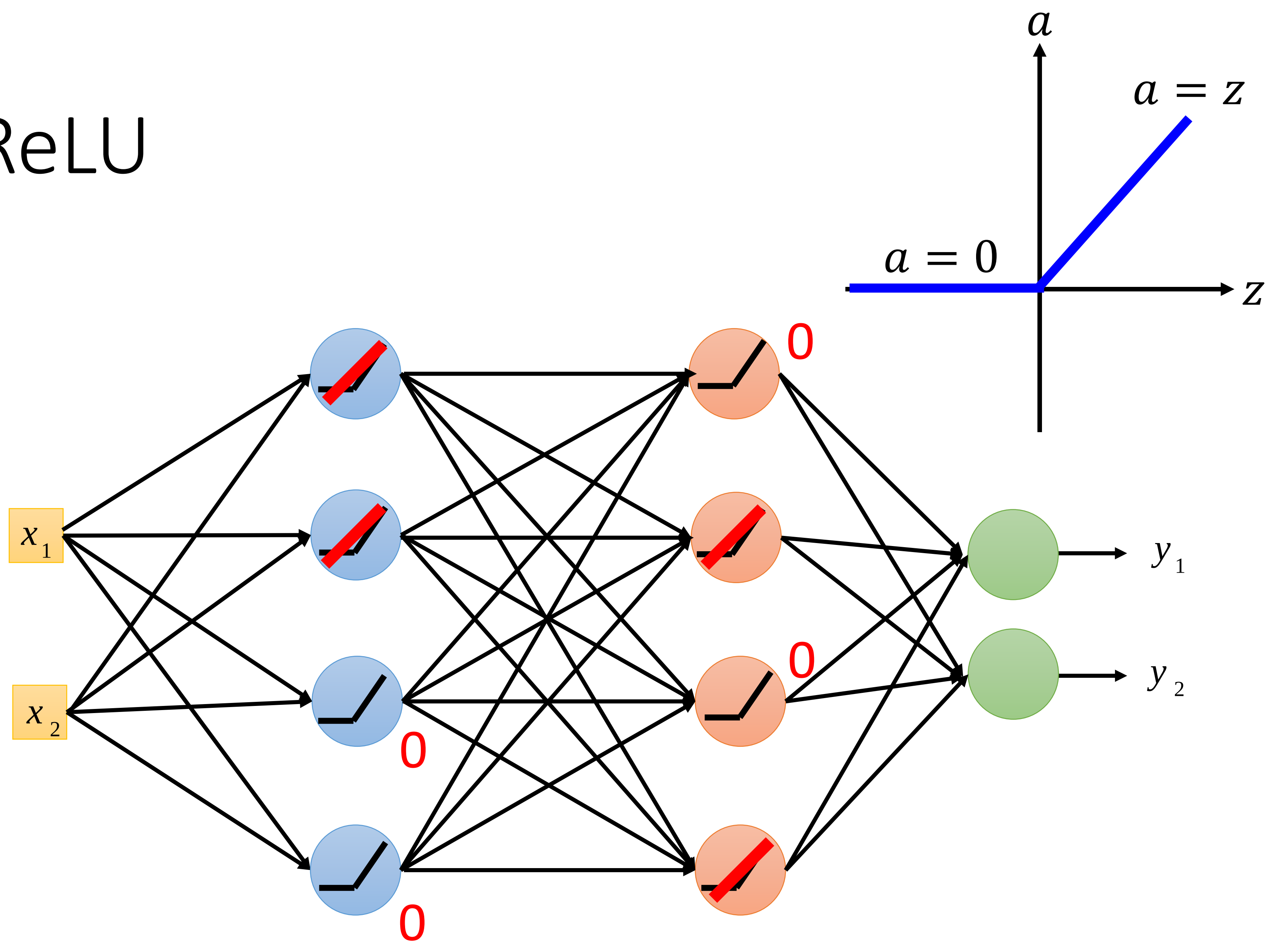


[Xavier Glorot, AISTATS'11]
[Andrew L. Maas, ICML'13]
[Kaiming He, arXiv'15]

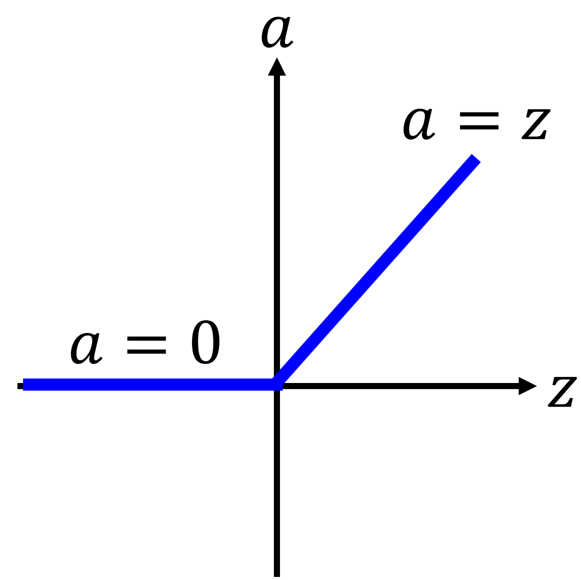
Reason:

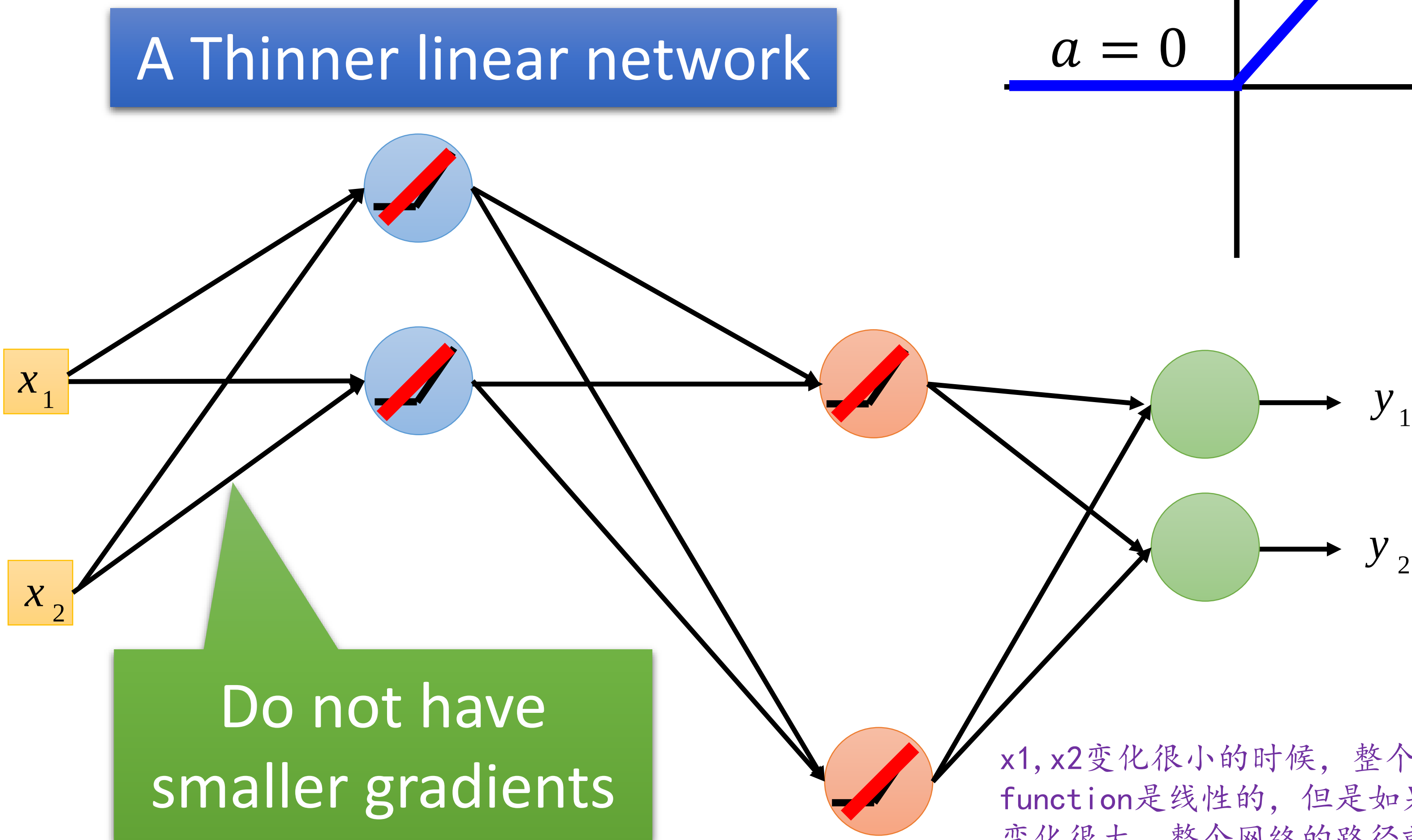
1. Fast to compute
2. Biological reason
3. Infinite sigmoid with different biases
4. Vanishing gradient problem

ReLU



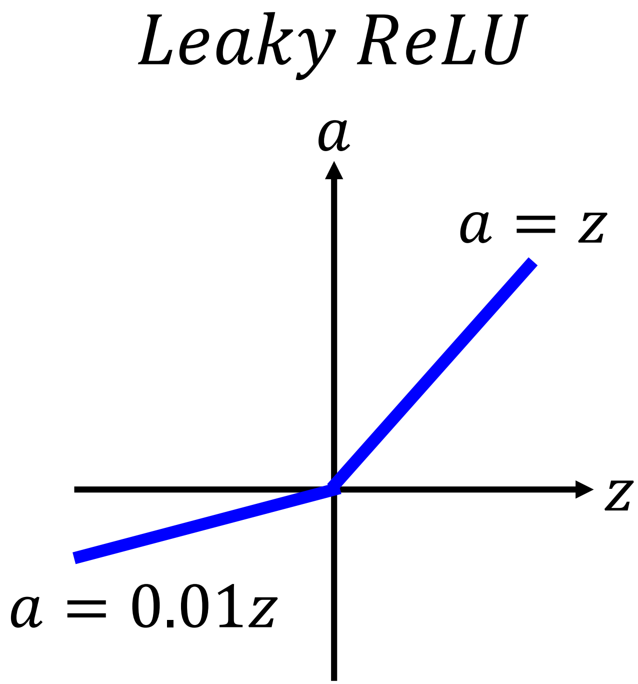
ReLU

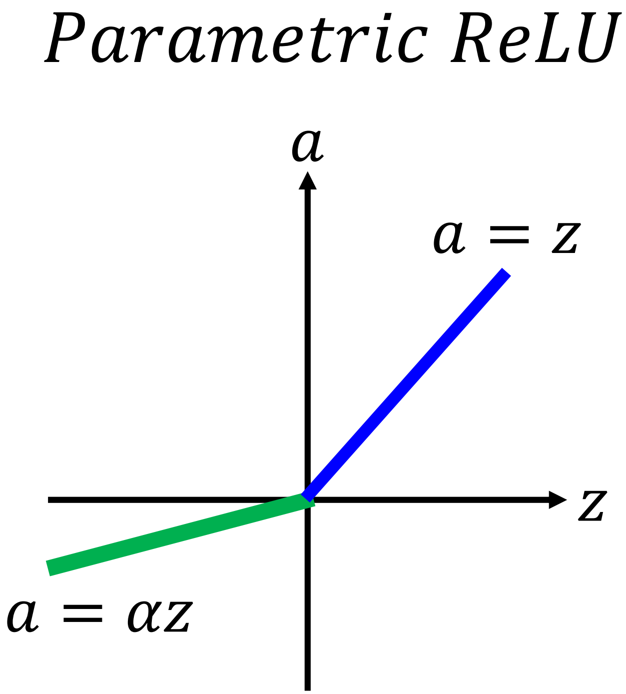




x1,x2变化很小的时候，整个function是线性的，但是如果x1,x2变化很大，整个网络的路径就变了，就变为另外一个线性结构了，所以整体来看，用relu并不是说整个网络退化为了线性结构

ReLU - variant



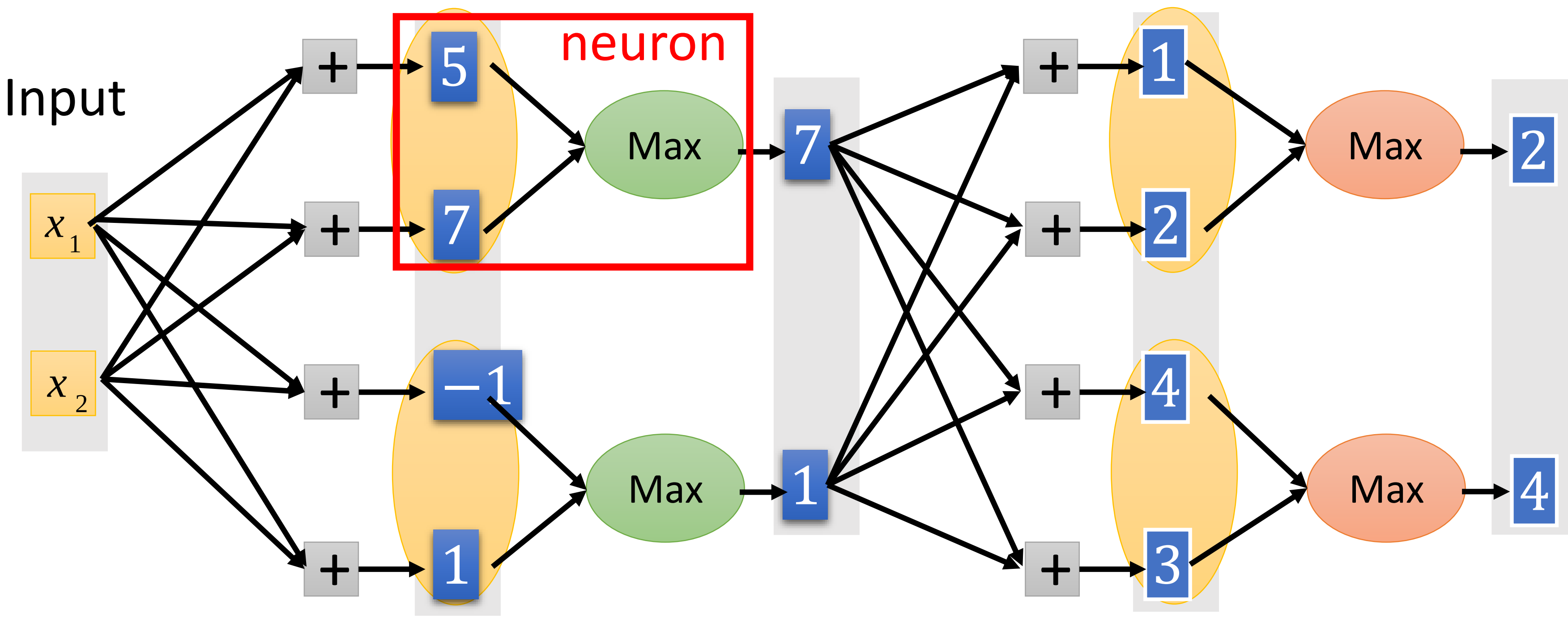


α also learned by gradient descent

Maxout

ReLU is a special cases of Maxout

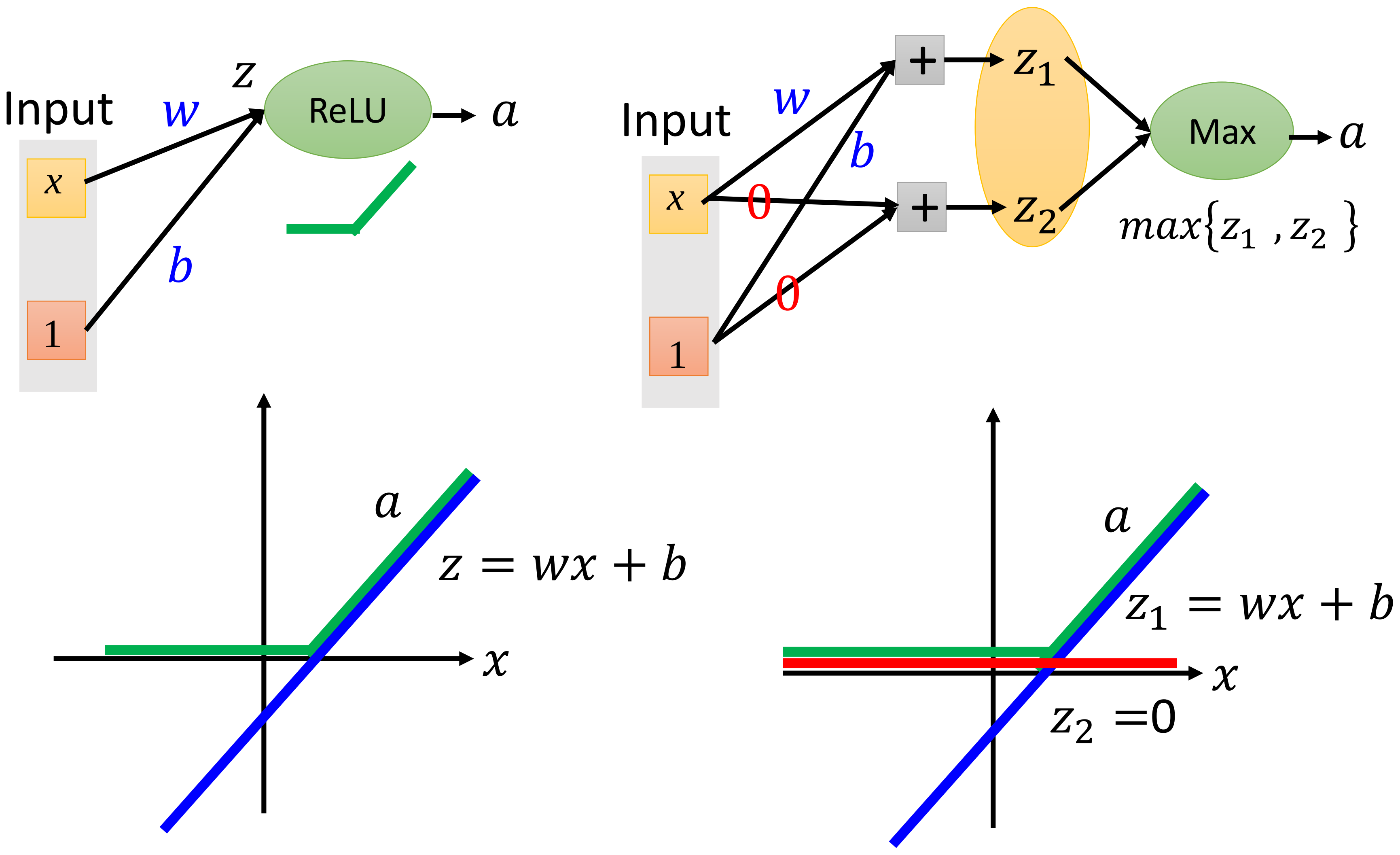
- Learnable activation function [Ian J. Goodfellow, ICML'13]



You can have more than 2 elements in a group.

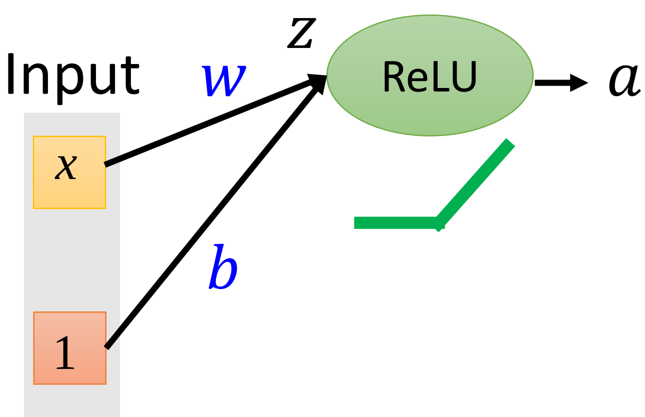
Maxout

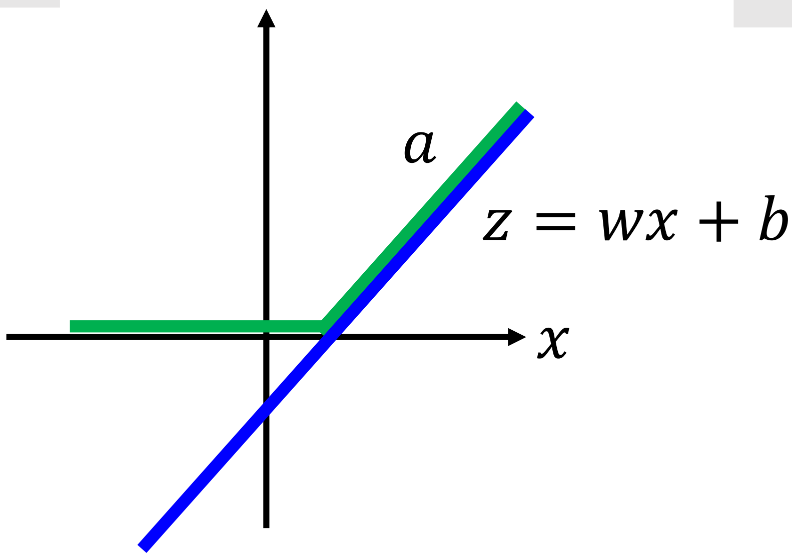
ReLU is a special cases of Maxout

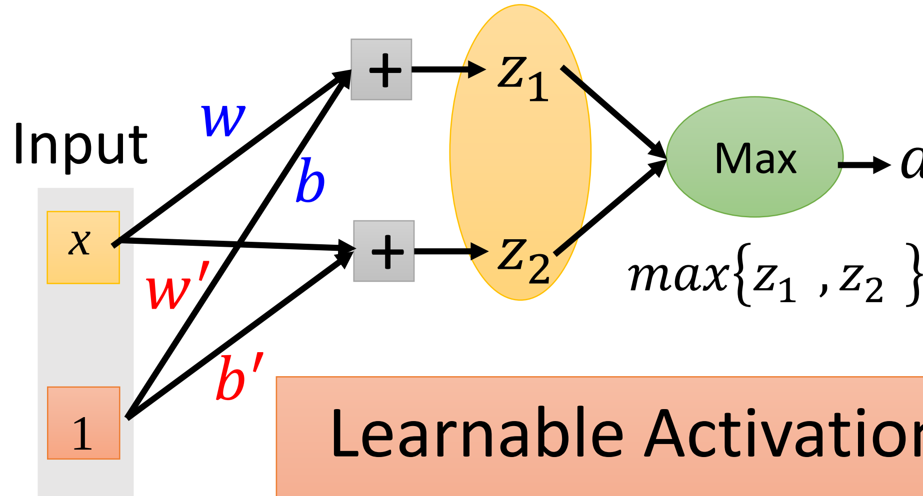


Maxout

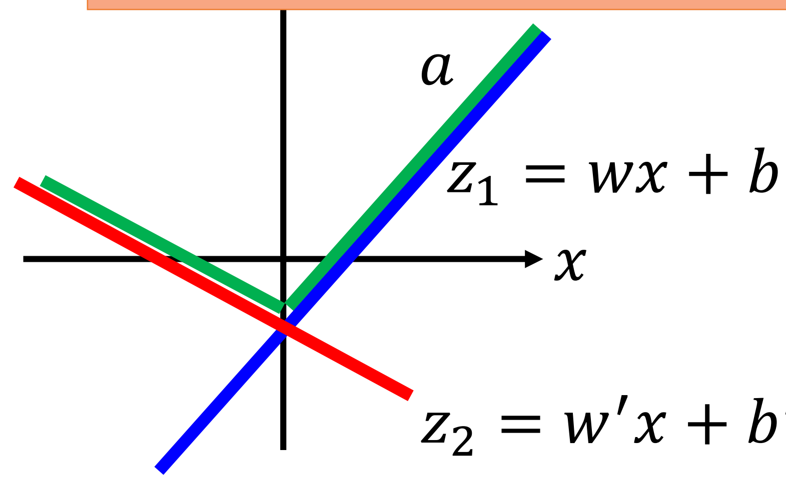
More than ReLU







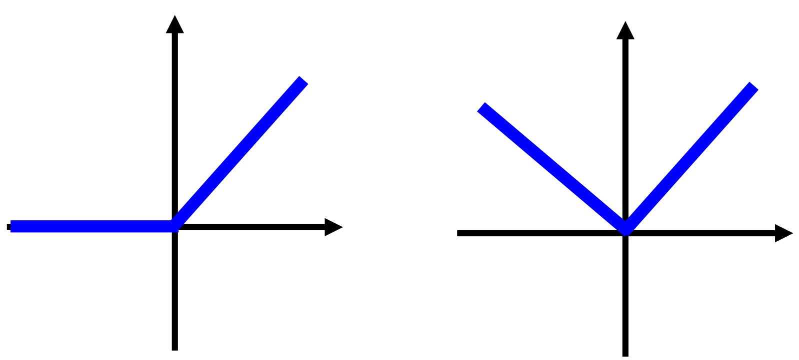
Learnable Activation Function



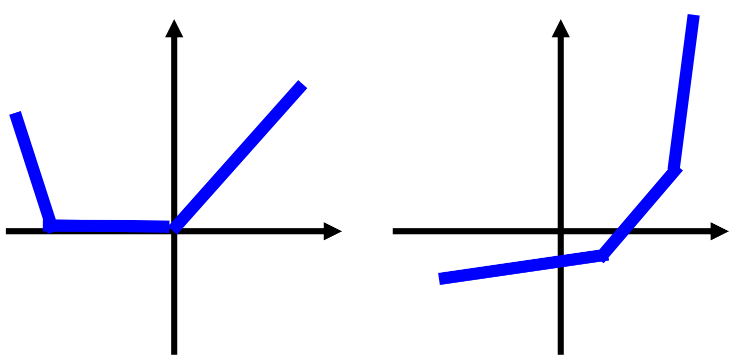
Maxout

- Learnable activation function [Ian J. Goodfellow, ICML'13]
  - Activation function in maxout network can be any piecewise linear convex function
  - How many pieces depending on how many elements in a group

2 elements in a group

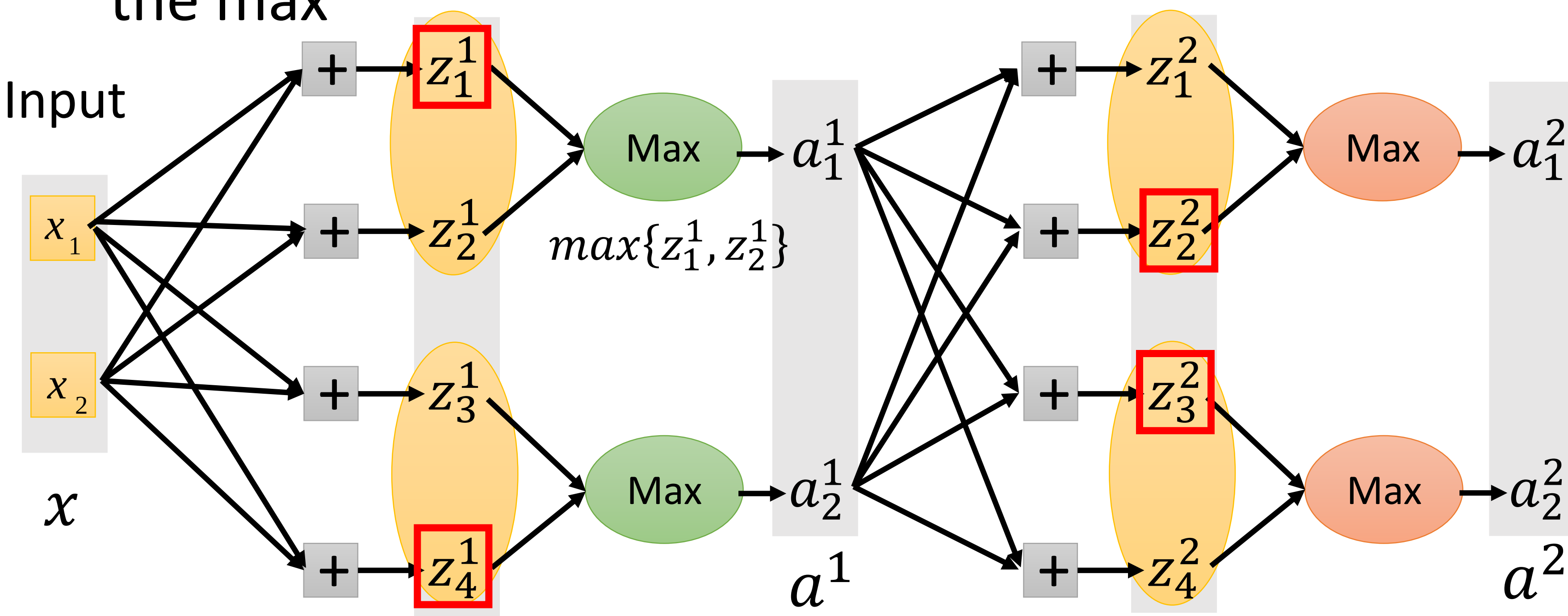


3 elements in a group



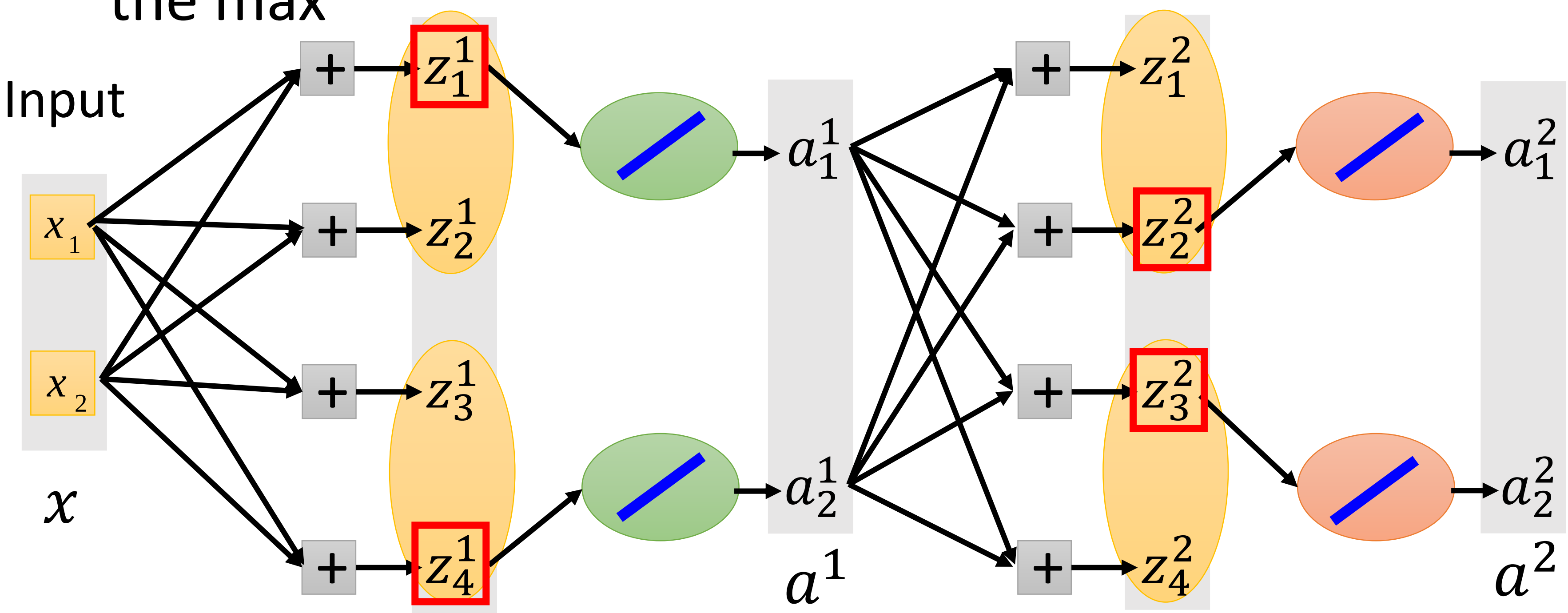
Maxout - Training

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



Maxout - Training

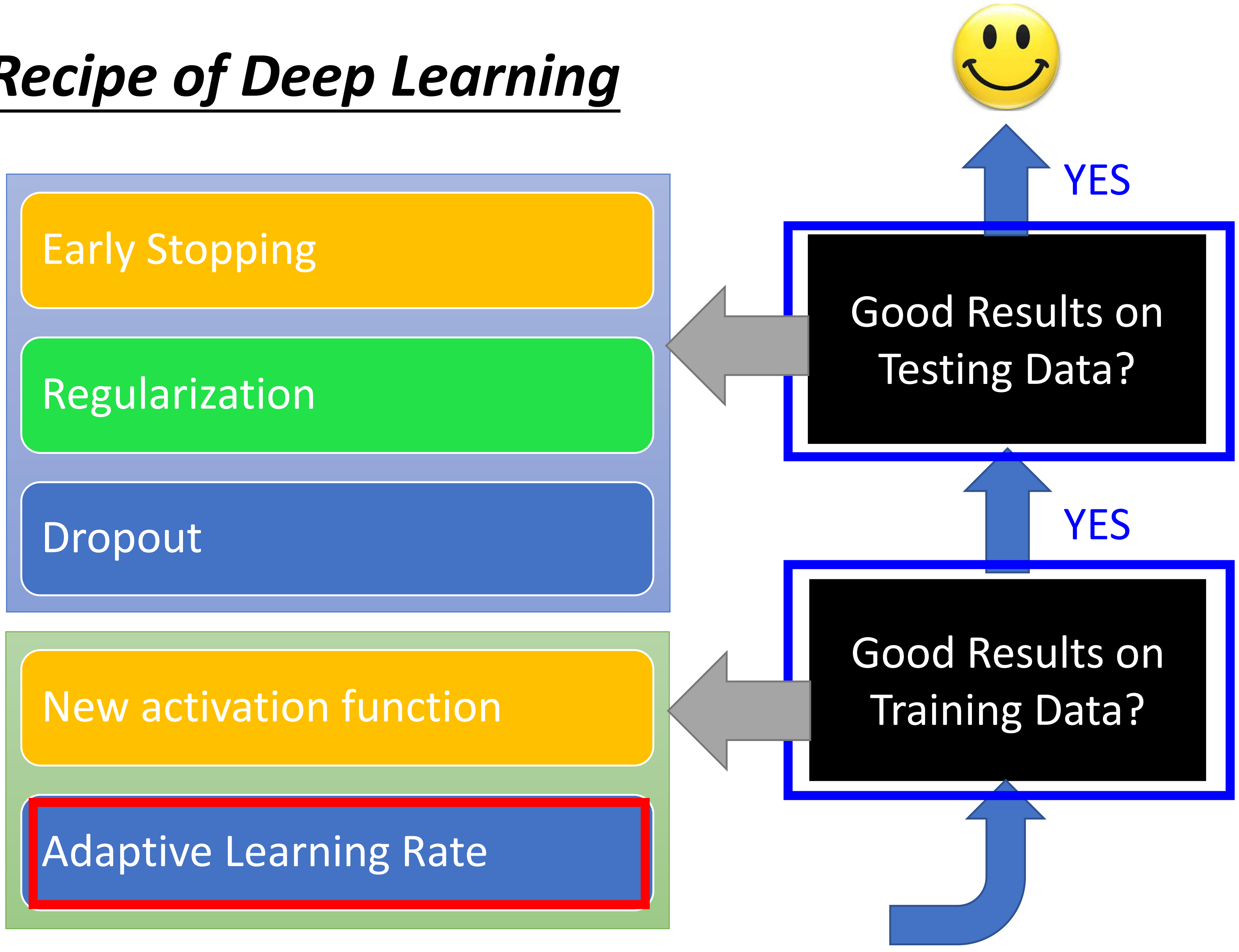
- Given a training data x, we know which z would be the max



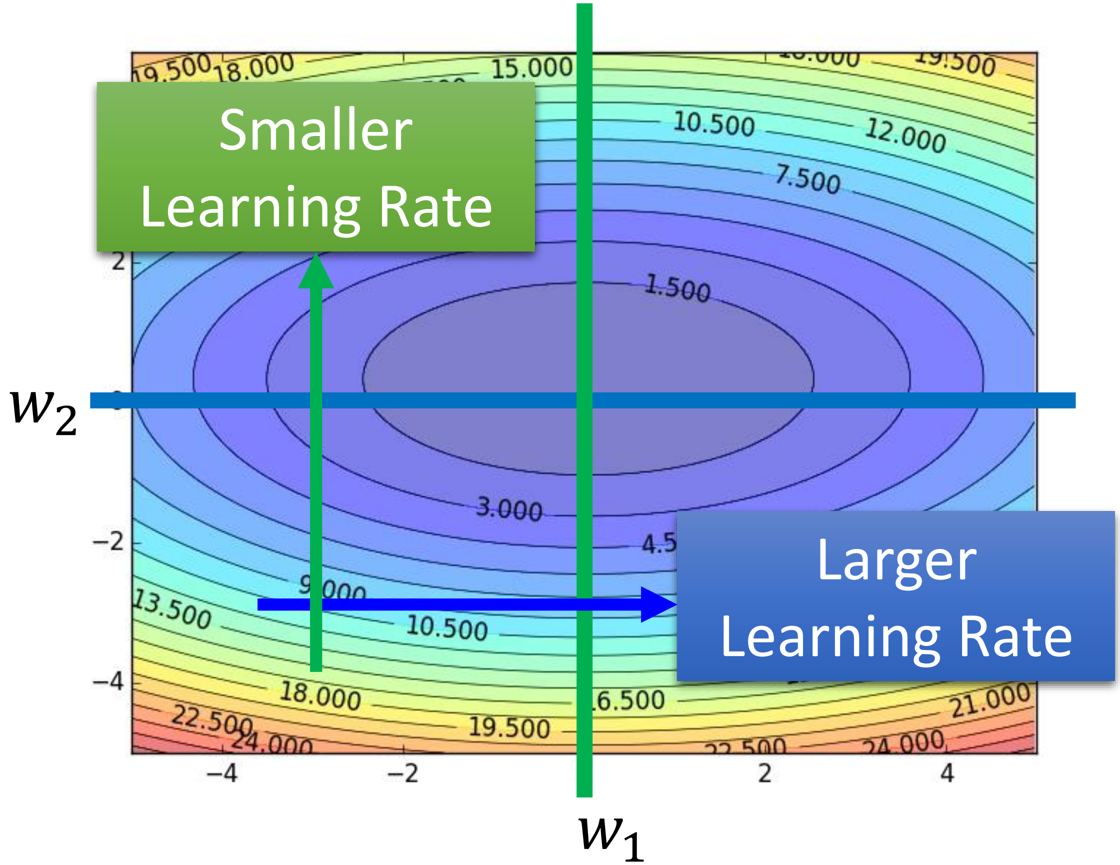
- Train this thin and linear network

Different thin and linear network for different examples

Recipe of Deep Learning



Review



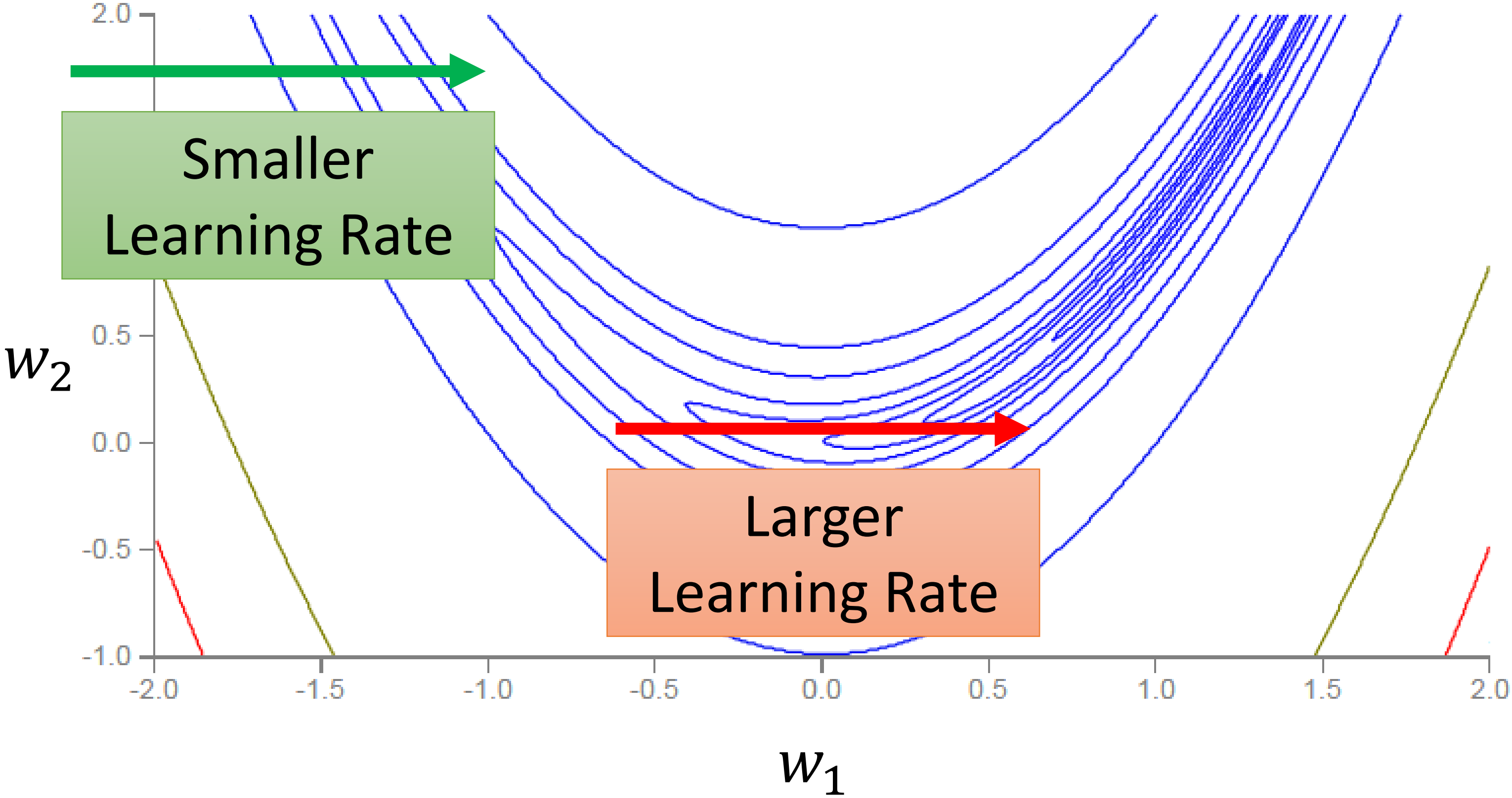
<u>Adagrad</u>

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

Use first derivative to estimate second derivative

RMSProp

Error Surface can be very complex when training NN.



RMSProp

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

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

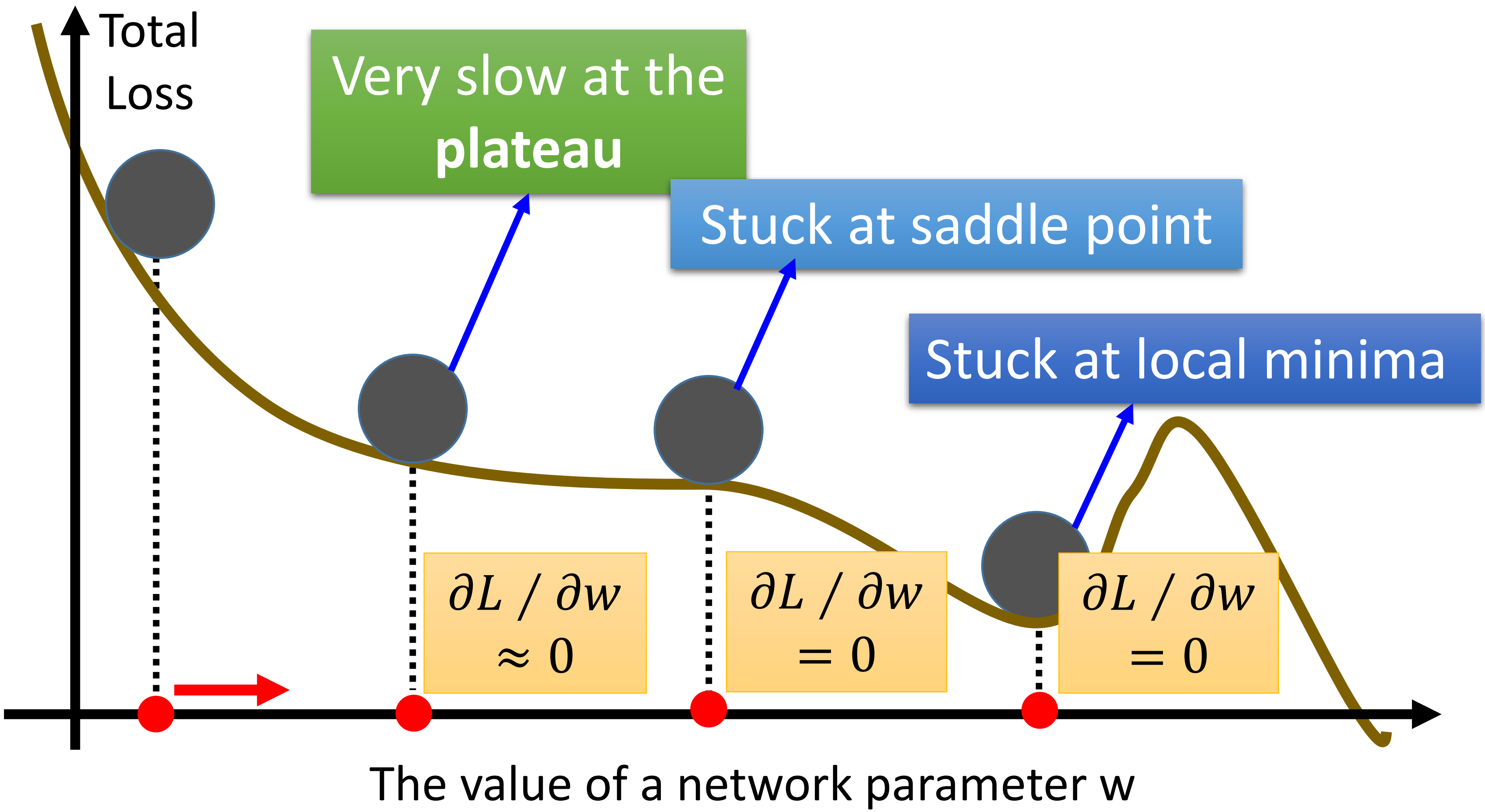
$$w^3 \leftarrow w^2 - \frac{\eta}{\sigma^2} g^2 \qquad \sigma^2 = \sqrt{\alpha(\sigma^1)^2 + (1-\alpha)(g^2)^2}$$

$$\vdots$$

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

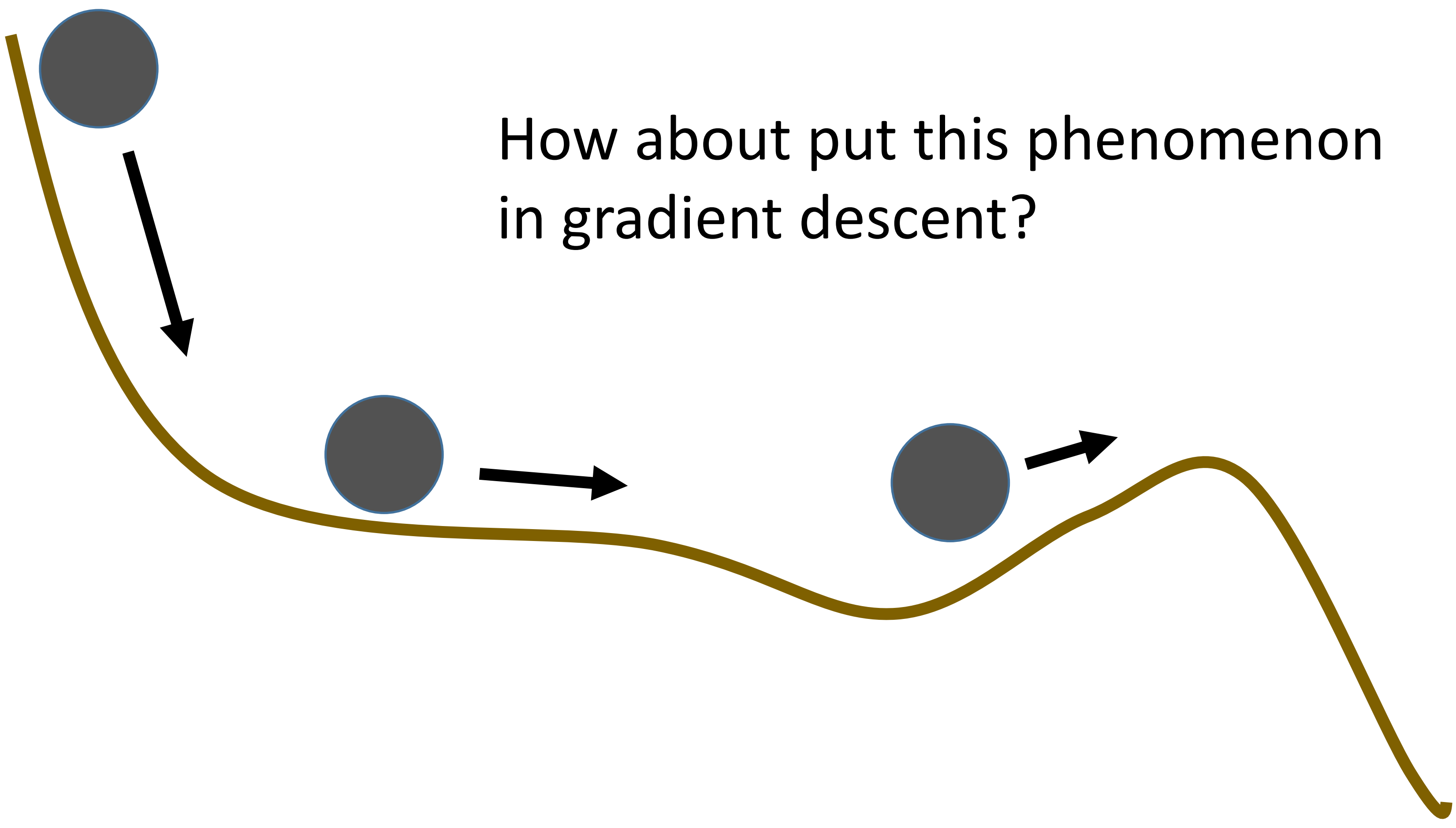
Root Mean Square of the gradients with previous gradients being decayed

Hard to find optimal network parameters



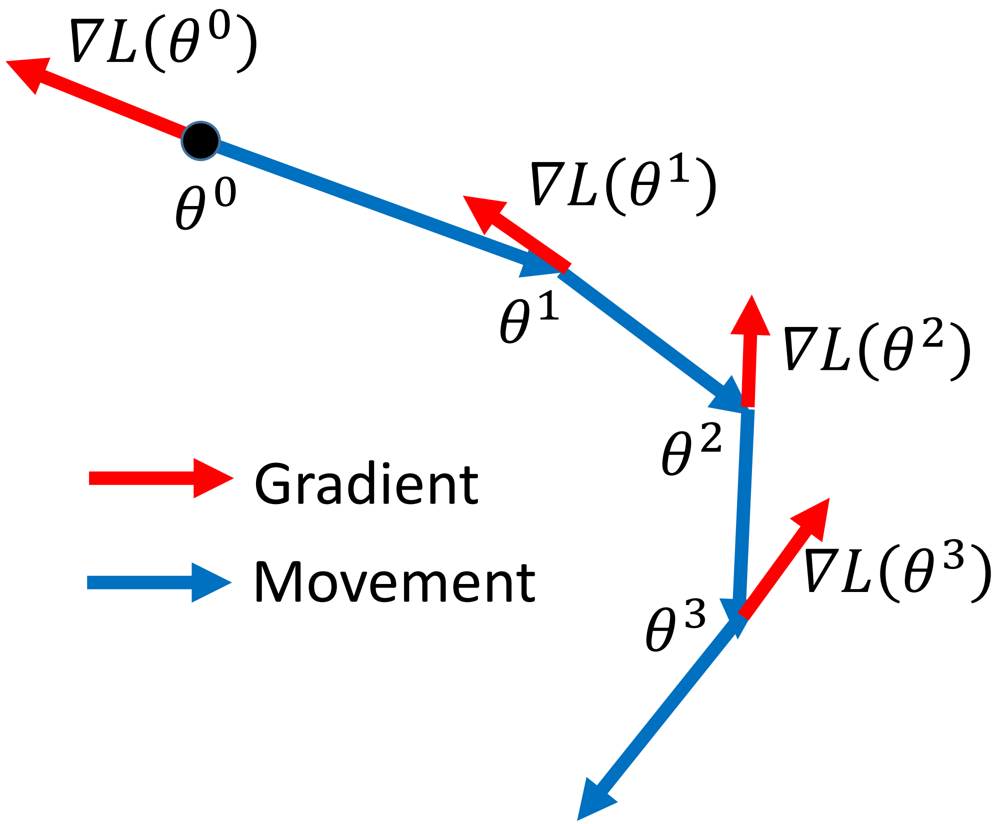
In physical world ……

- Momentum



How about put this phenomenon in gradient descent?

Review: Vanilla Gradient Descent



Start at position θ^0

Compute gradient at θ^0

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

Compute gradient at θ^1

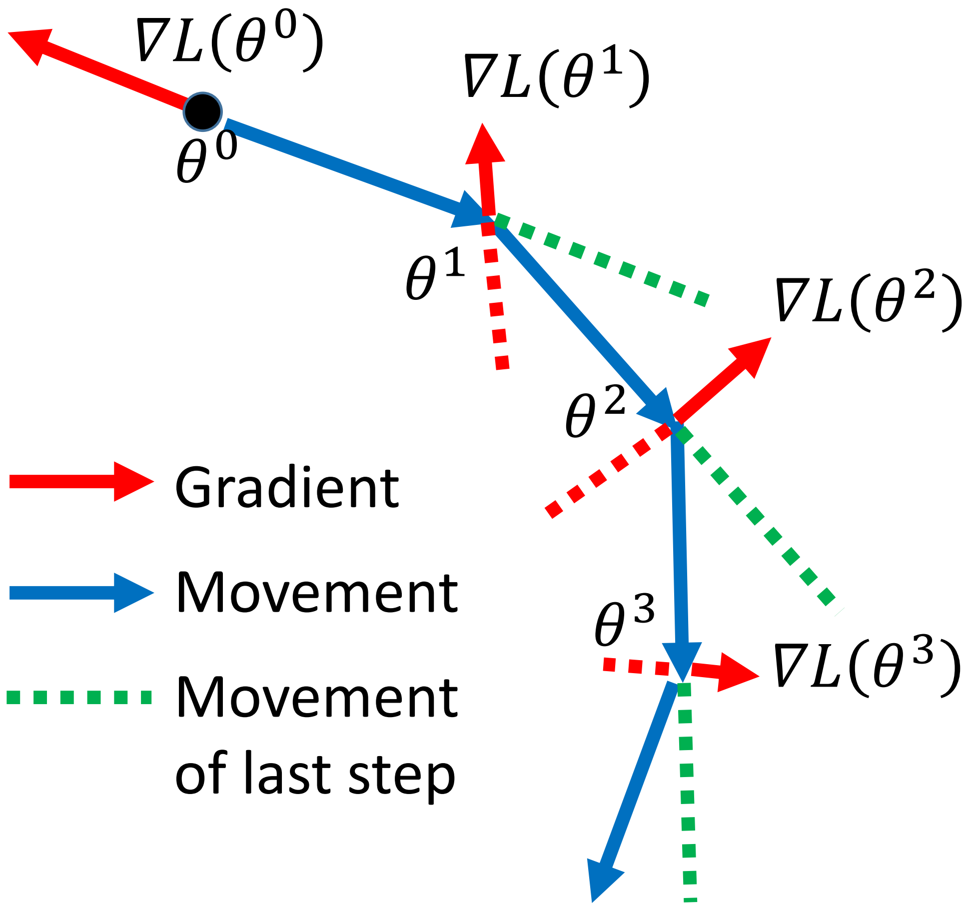
Move to $\theta^2 = \theta^1 - \eta \nabla L(\theta^1)$

⋮

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

Momentum

Movement: movement of last step minus gradient at present



Start at point θ^0

Movement $v^0=0$

Compute gradient at θ^0

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

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

Compute gradient at θ^1

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

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

$v^0 = 0$

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

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

$\vdots$

Start at point θ^0

Movement v^0=0

Compute gradient at θ^0

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

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

Compute gradient at θ^1

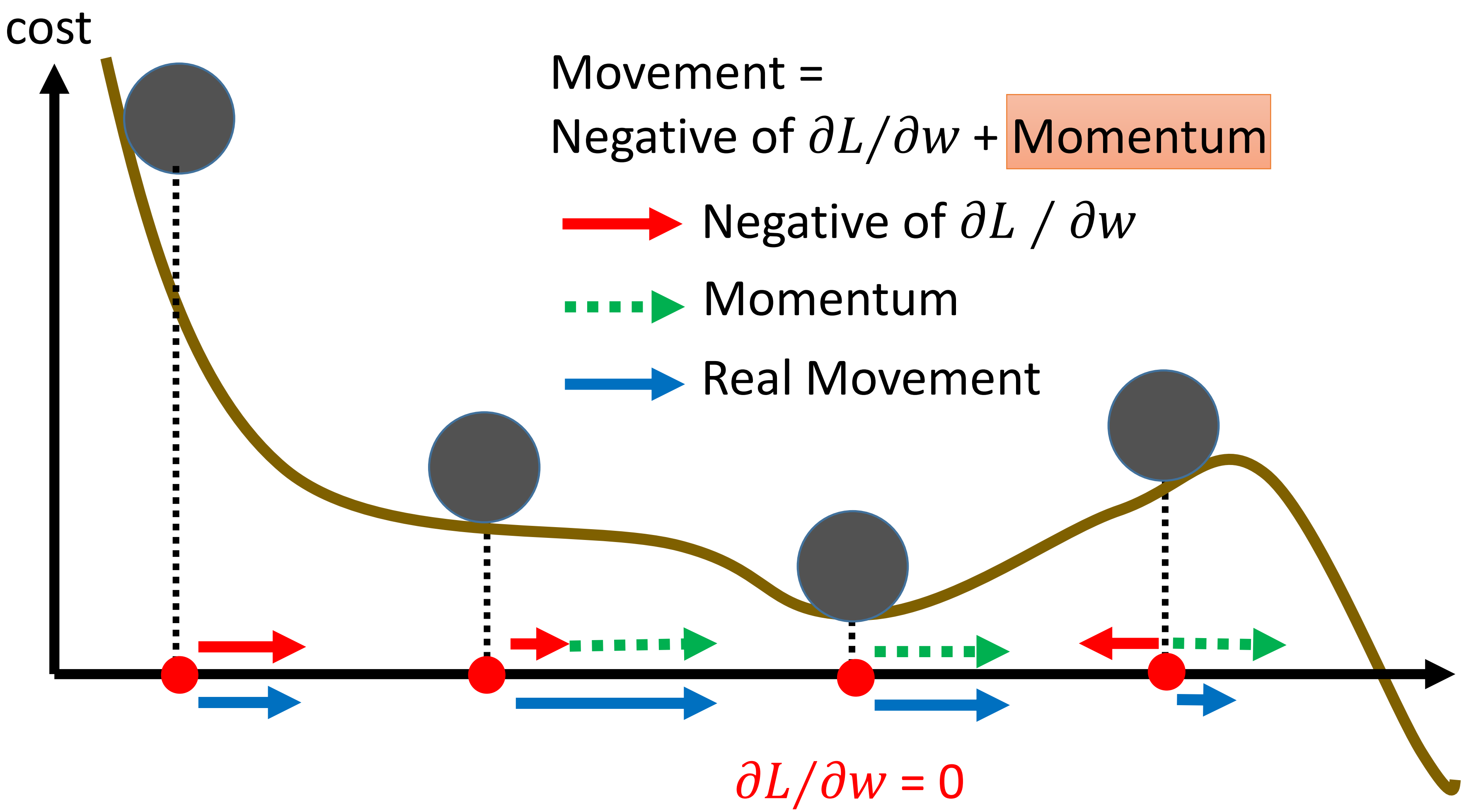
Movement $v^2 = \lambda v^1 - \eta\nabla L(\theta^1)$

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

Movement not just based on gradient, but previous movement

Momentum

Still not guarantee reaching global minima, but give some hope ……



Adam

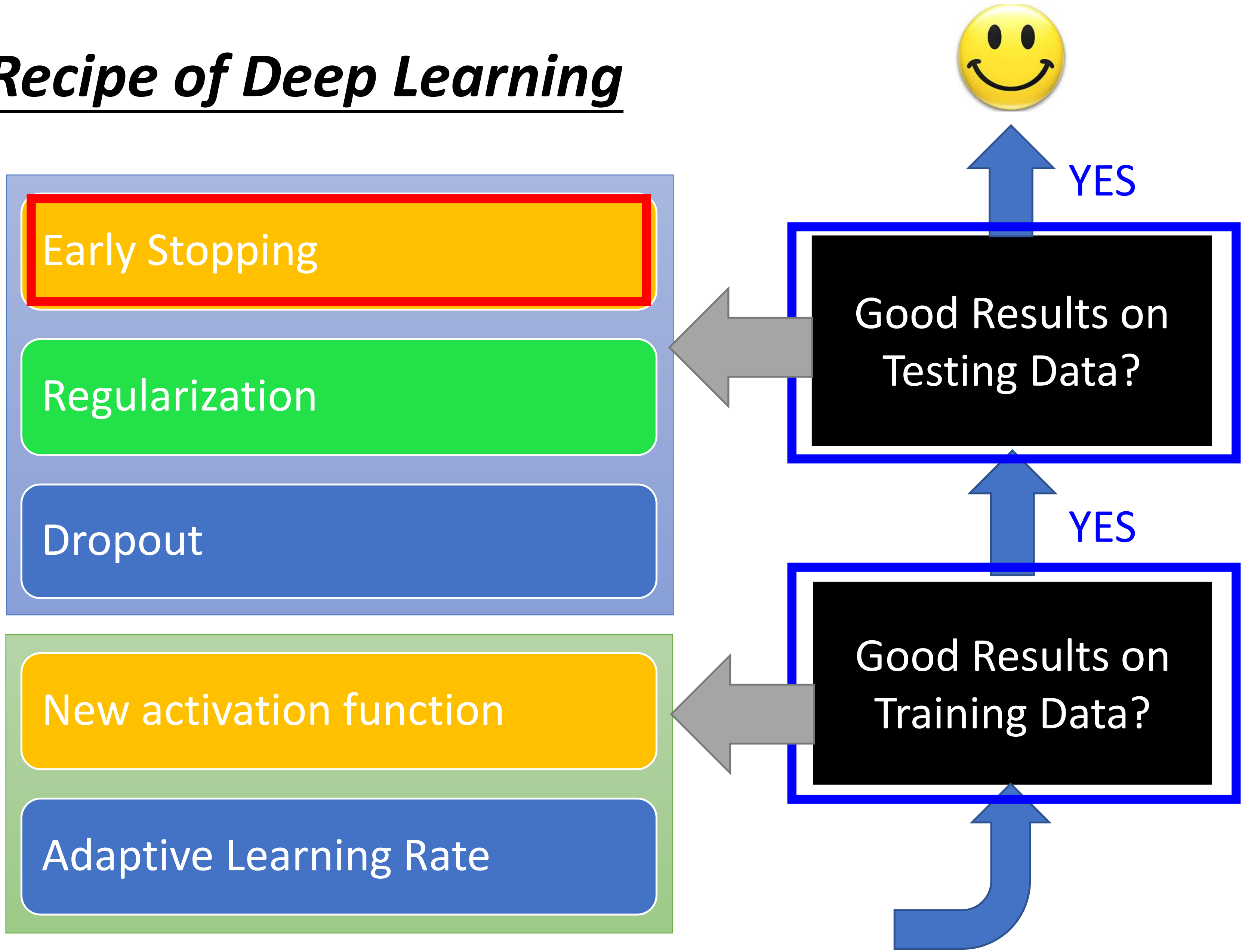
RMSProp + Momentum

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector

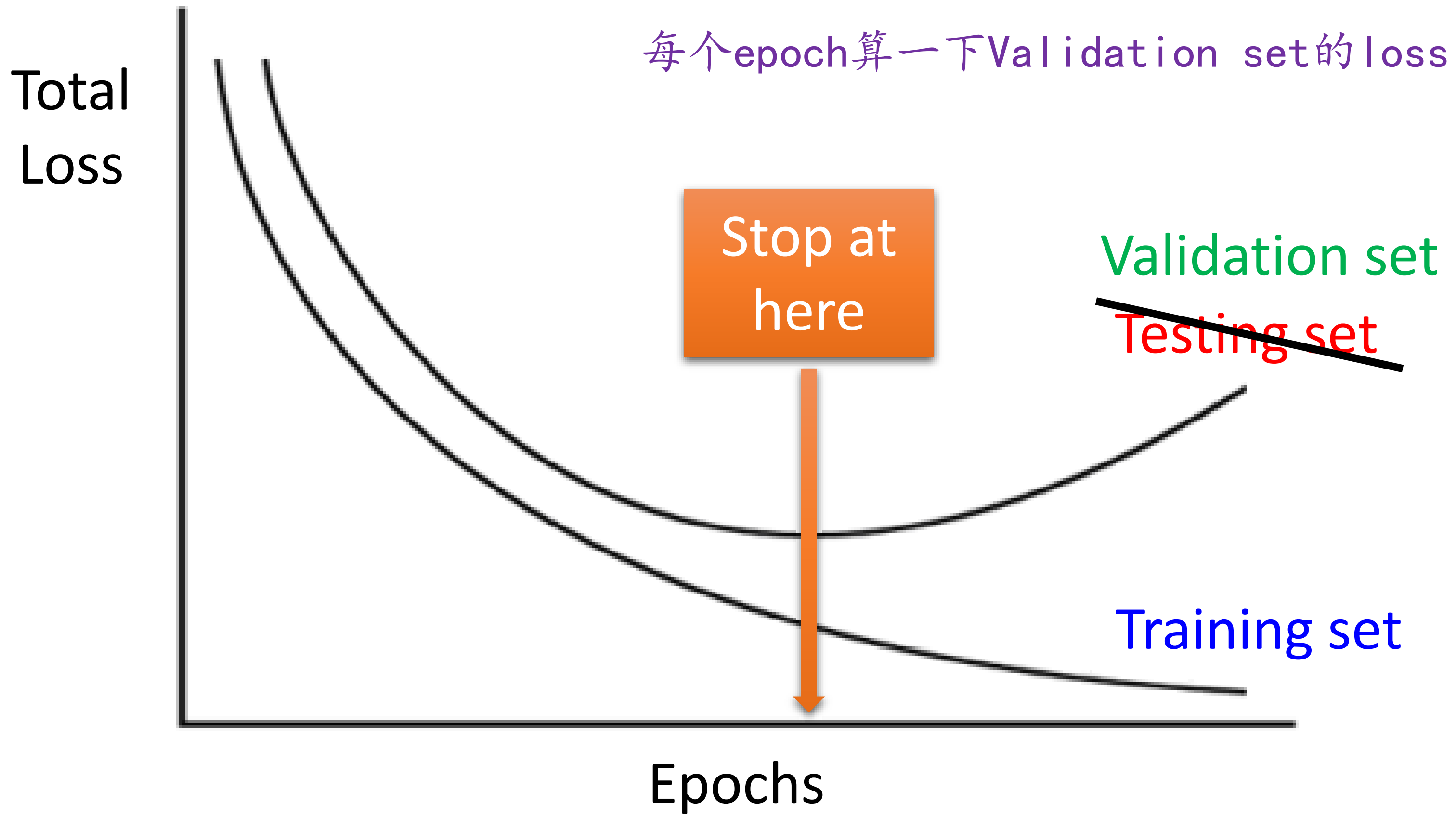
- $m_0 \leftarrow 0$ (Initialize 1st moment vector) → for momentum
- $v_0 \leftarrow 0$ (Initialize 2nd moment vector) → for RMSprop
- $t \leftarrow 0$ (Initialize timestep)
- **while** θ_t not converged **do**
  - $t \leftarrow t + 1$
  - $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
  - $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
  - $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
  - $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
  - $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
  - $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
- **end while**
- **return** θ_t (Resulting parameters)

Recipe of Deep Learning



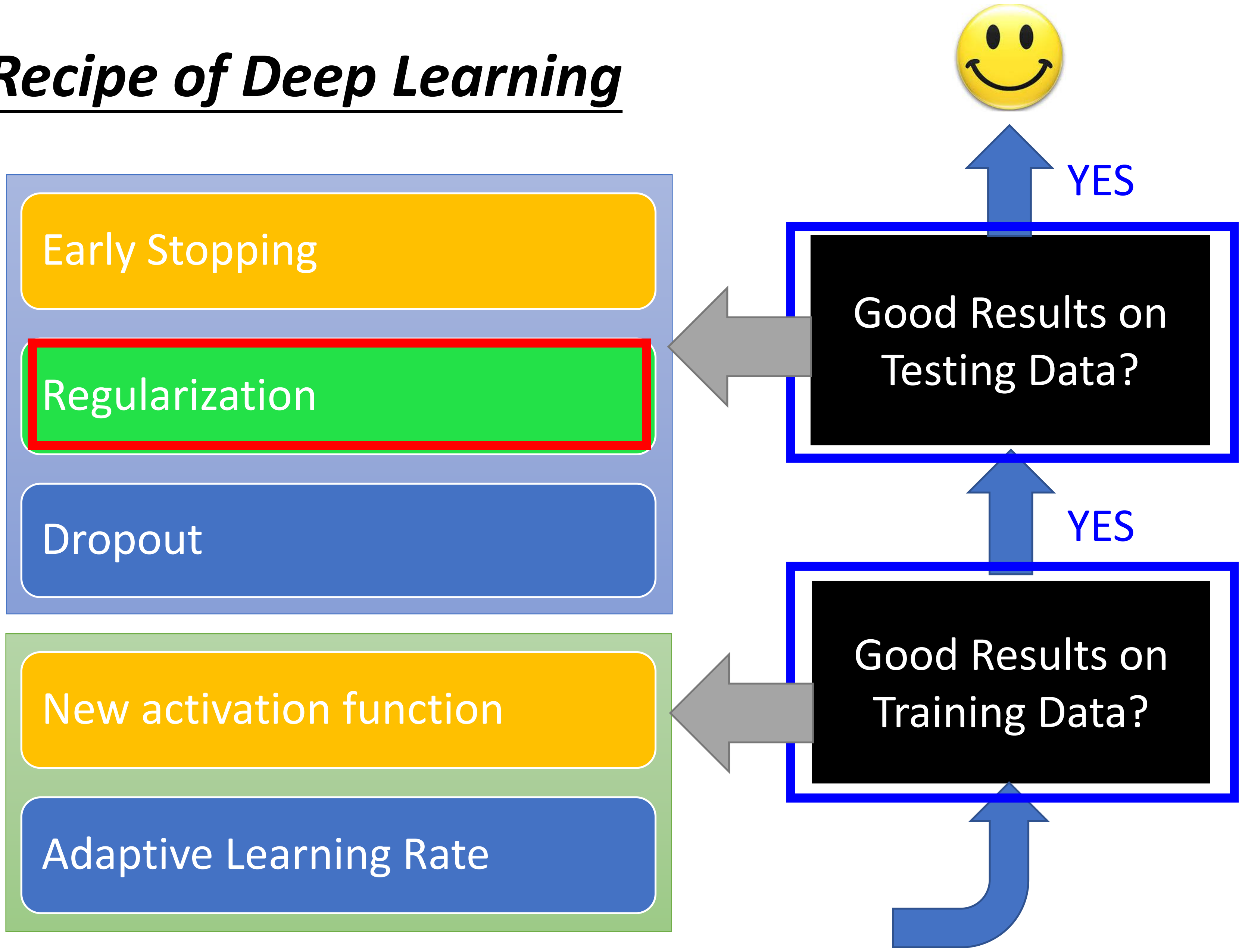
如何把ealy stopping理解为一个种regularization

Early Stopping



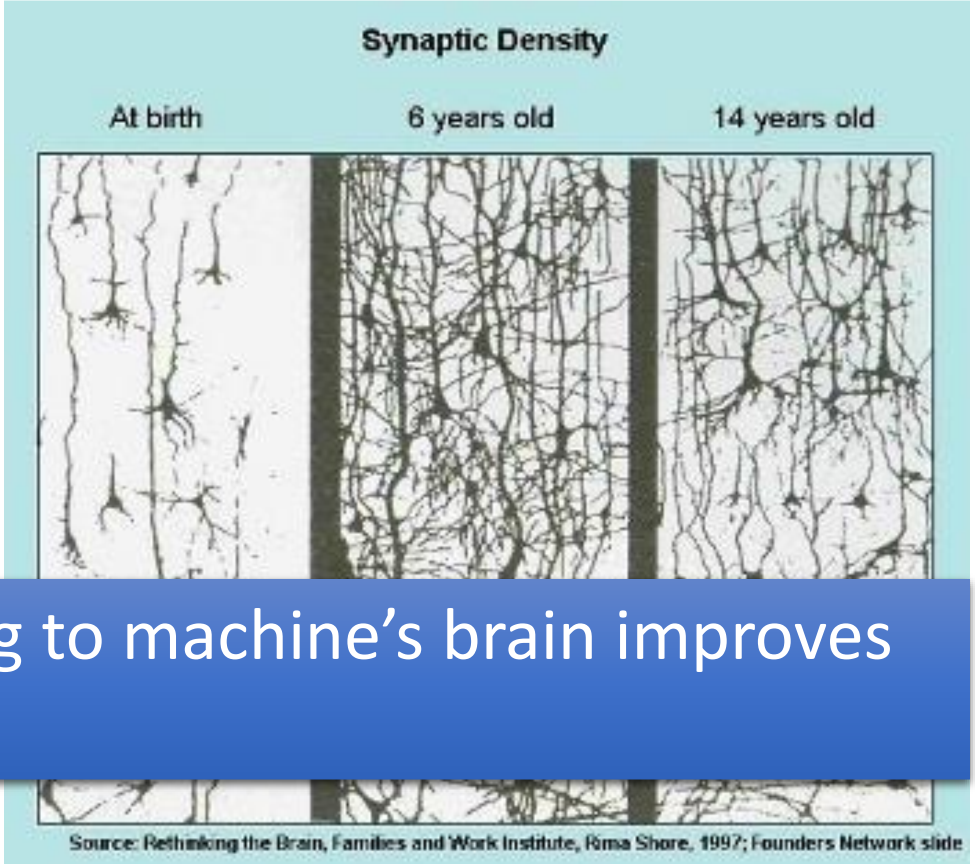
Keras: http://keras.io/getting-started/faq/#how-can-i-interrupt-training-when-the-validation-loss-isnt-decreasing-anymore

Recipe of Deep Learning



Regularization - Weight Decay

- Our brain prunes out the useless link between neurons.



Doing the same thing to machine's brain improves the performance.

Regularization

在一定程度上early stopping和regularization在做同样一件事，就是使|w|保持很小。
w一般初始化为很小的值，随着训练会越来越大，所以early stopping会让w来不及变大。

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

$$\mathrm{L}'(\theta) = \underbrace{L(\theta)}_{\text{Original loss}} + \lambda \frac{1}{2}\underline{\|\theta\|_2} \rightarrow \text{Regularization term}$$

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

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

L2 regularization:

$$\|\theta\|_2 = (w_1)^2 + (w_2)^2 + \ldots$$

(usually not consider biases)

Regularization

L2 regularization:

$$\|\theta\|_2 = (w_1)^2 + (w_2)^2 + \ldots$$

- New loss function to be minimized

$$\mathrm{L}'(\theta) = L(\theta) + \lambda \frac{1}{2}\|\theta\|_2$$

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

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

$$= \underbrace{(1 - \eta\lambda)w^t}_{\text{Closer to zero}} - \eta \underline{\frac{\partial \mathrm{L}}{\partial w}}$$

Weight Decay

当接近局部最小值时，后一项基本为0，那w的更新就在不断的decay

Regularization

L1 regularization:

$$\|\theta\|_1 = |w_1| + |w_2| + \ldots$$

- New loss function to be minimized

$$\mathrm{L}'(\theta) = L(\theta) + \lambda \frac{1}{2}\|\theta\|_1 \qquad \frac{\partial \mathrm{L}'}{\partial w} = \frac{\partial \mathrm{L}}{\partial w} + \lambda\ \mathrm{sgn}(w)$$

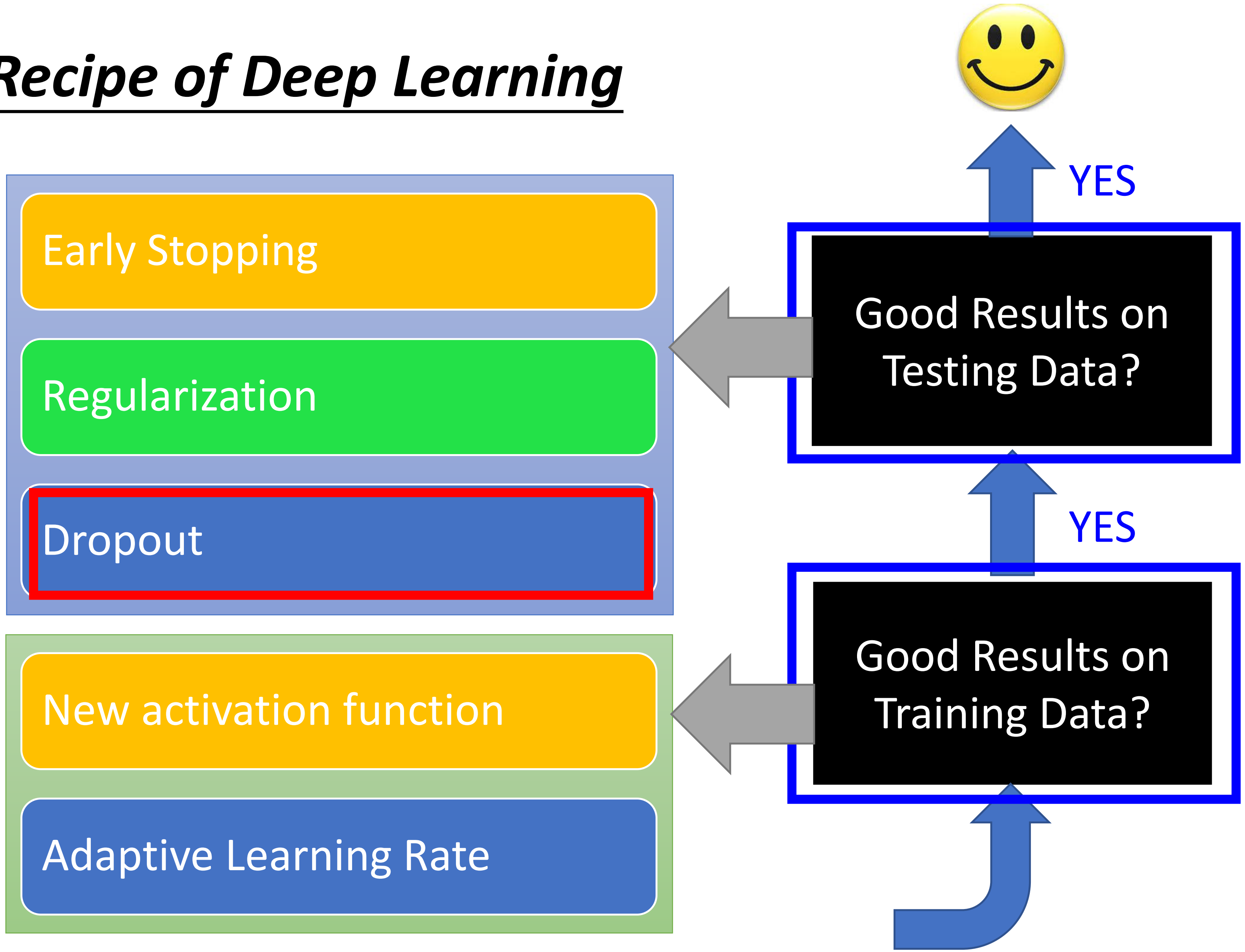
Update:

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

$$= w^t - \eta \frac{\partial \mathrm{L}}{\partial w} - \underline{\eta\lambda\ \mathrm{sgn}(w^t)}$$ Always delete

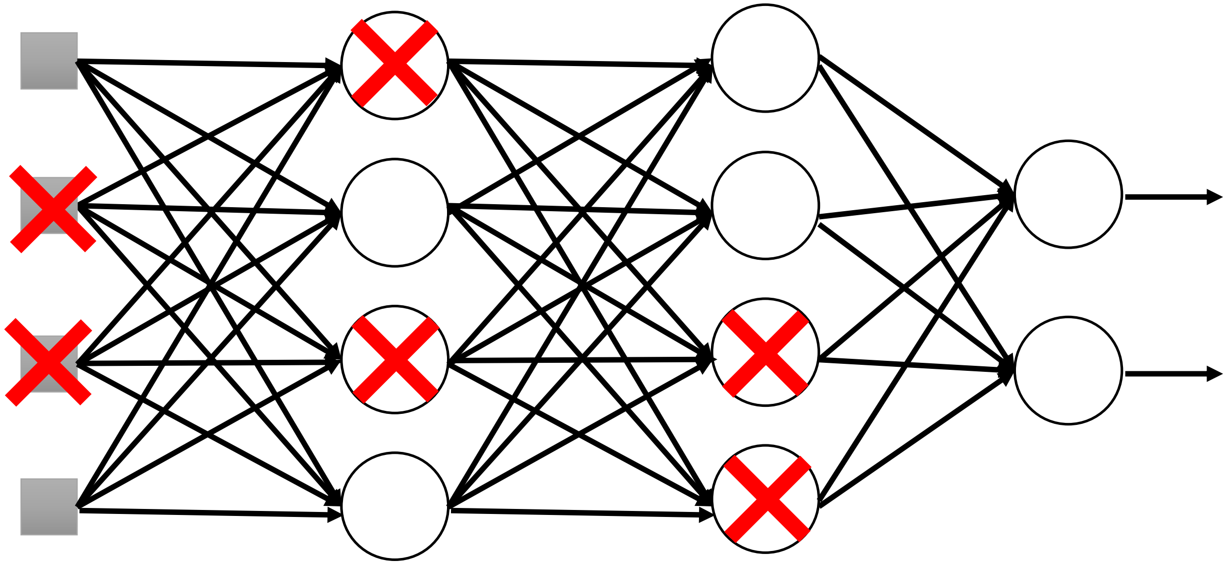
$$= (1 - \eta\lambda)w^t - \eta \frac{\partial \mathrm{L}}{\partial w}$$ …… L2

Recipe of Deep Learning



Dropout

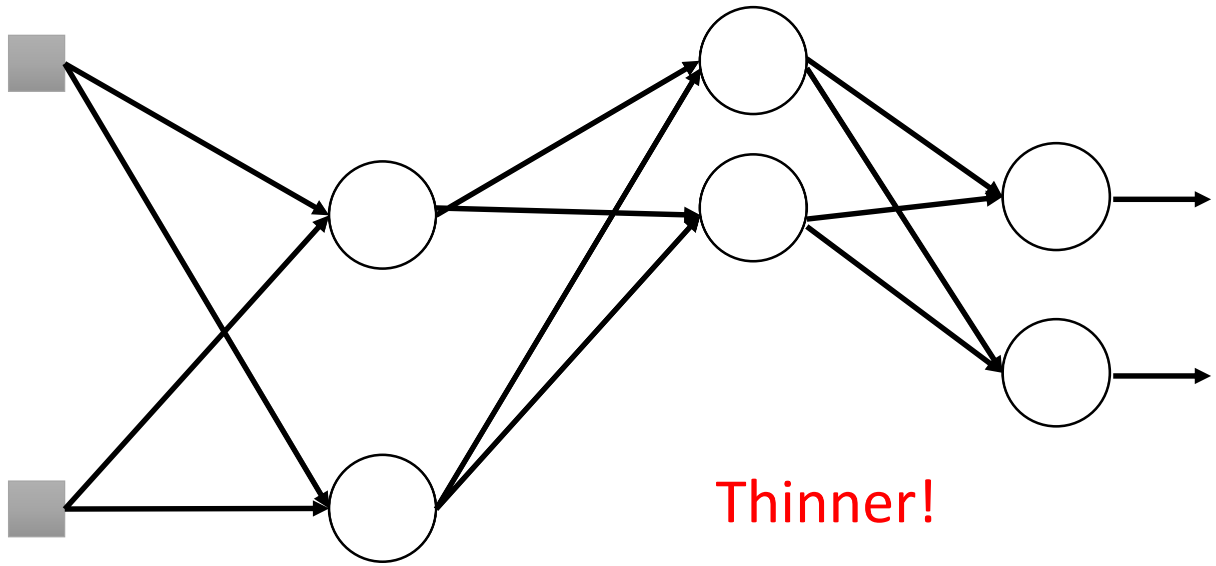
Training:



- **Each time before updating the parameters**
  - Each neuron has p% to dropout

Dropout

Training:

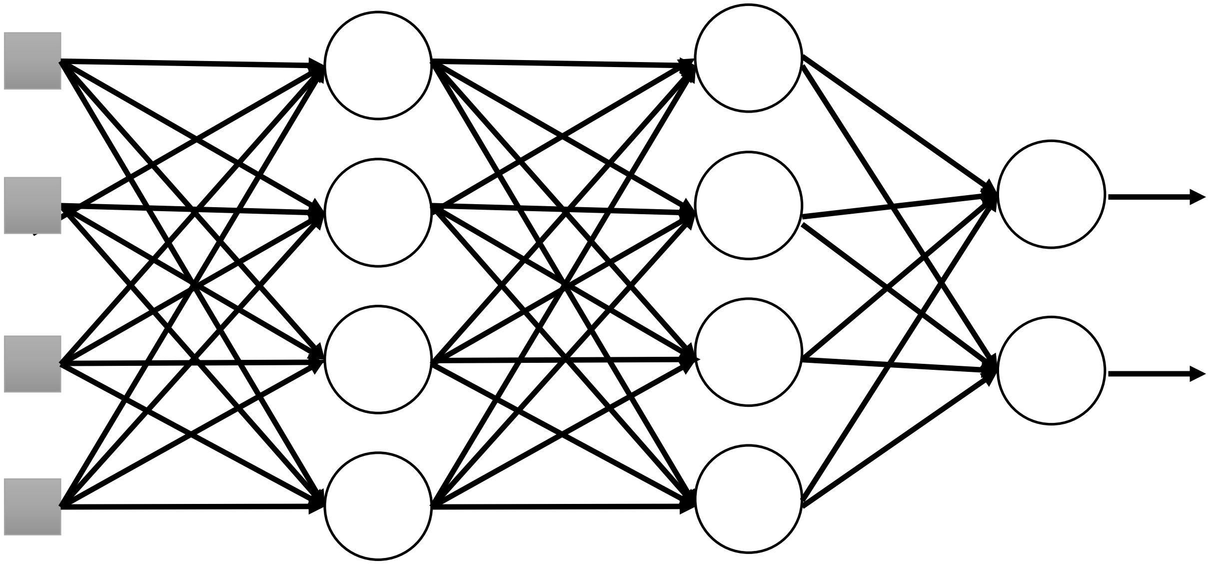


- **Each time before updating the parameters**
  - Each neuron has p% to dropout
    - **The structure of the network is changed.**
  - Using the new network for training

For each mini-batch, 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 a weight $w = 1$ by training, set $w = 0.5$ for testing.

Dropout
- Intuitive Reason

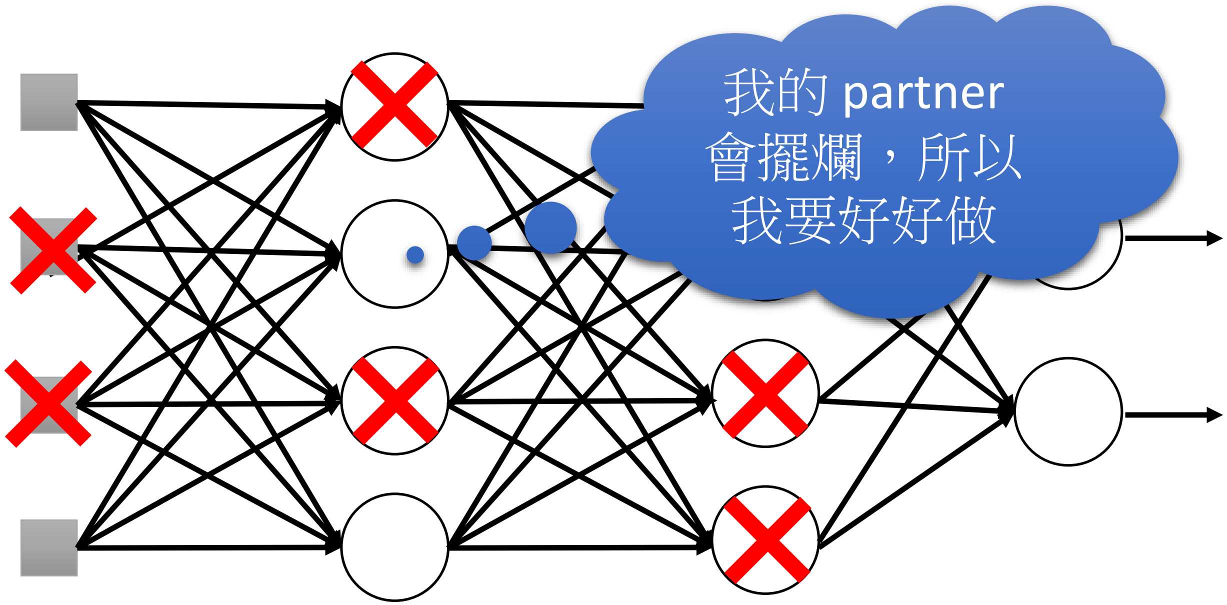
<u>Testing</u>

No dropout
(拿下重物後就變很強)

<u>Training</u>

Dropout (腳上綁重物)

Dropout - Intuitive Reason



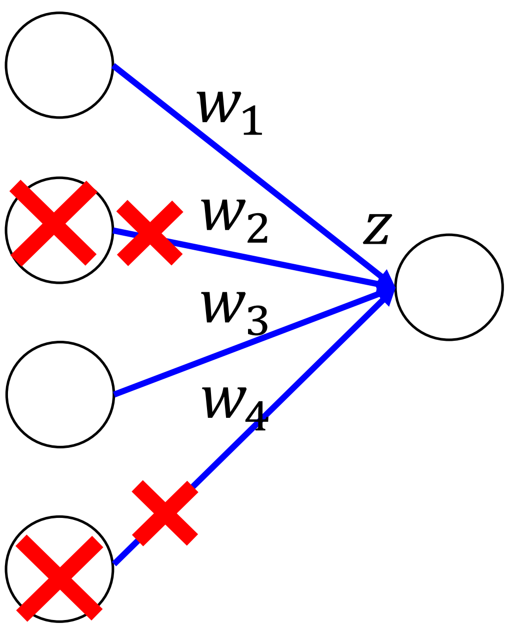
- When teams up, if everyone expect the partner will do the work, nothing will be done finally.
- However, if you know your partner will dropout, you will do better.
- When testing, no one dropout actually, so obtaining good results eventually.

Dropout - Intuitive Reason

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

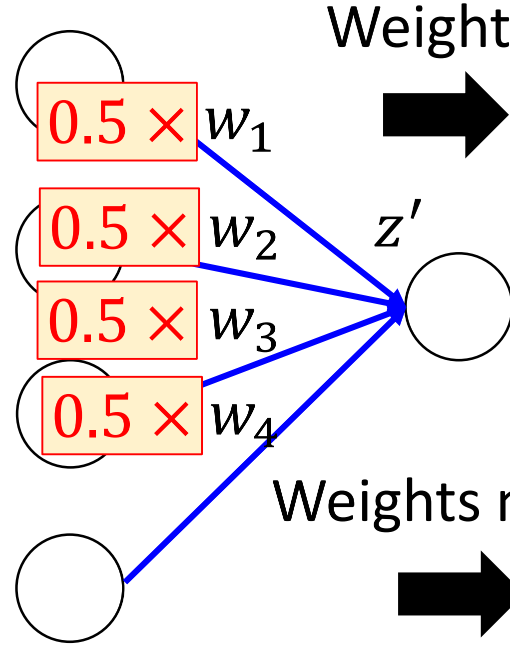
<u>Training of Dropout</u>

Assume dropout rate is 50%



<u>Testing of Dropout</u>

No dropout



Weights from training

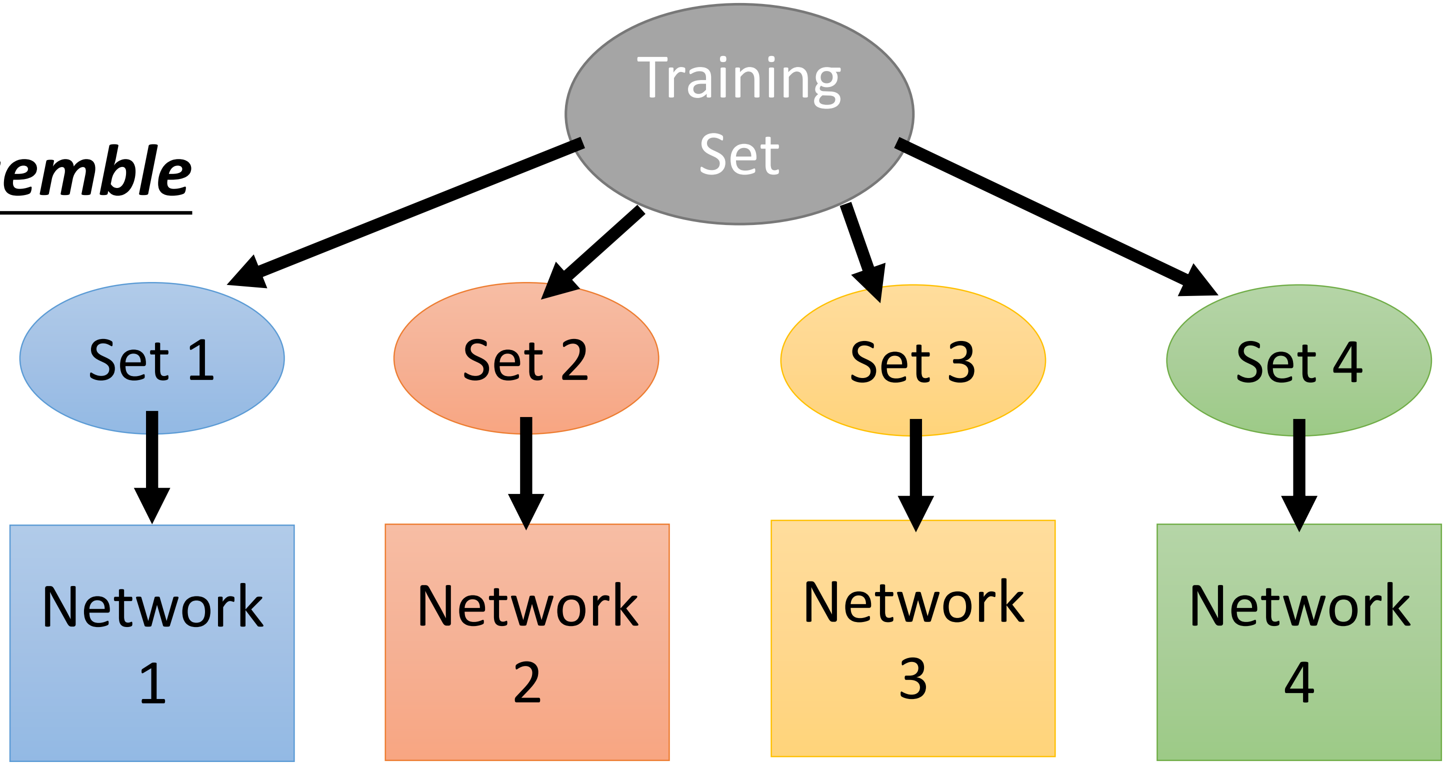
$\Rightarrow z' \approx 2z$

Weights multiply 1-p%

$\Rightarrow z' \approx z$

Dropout is a kind of ensemble.

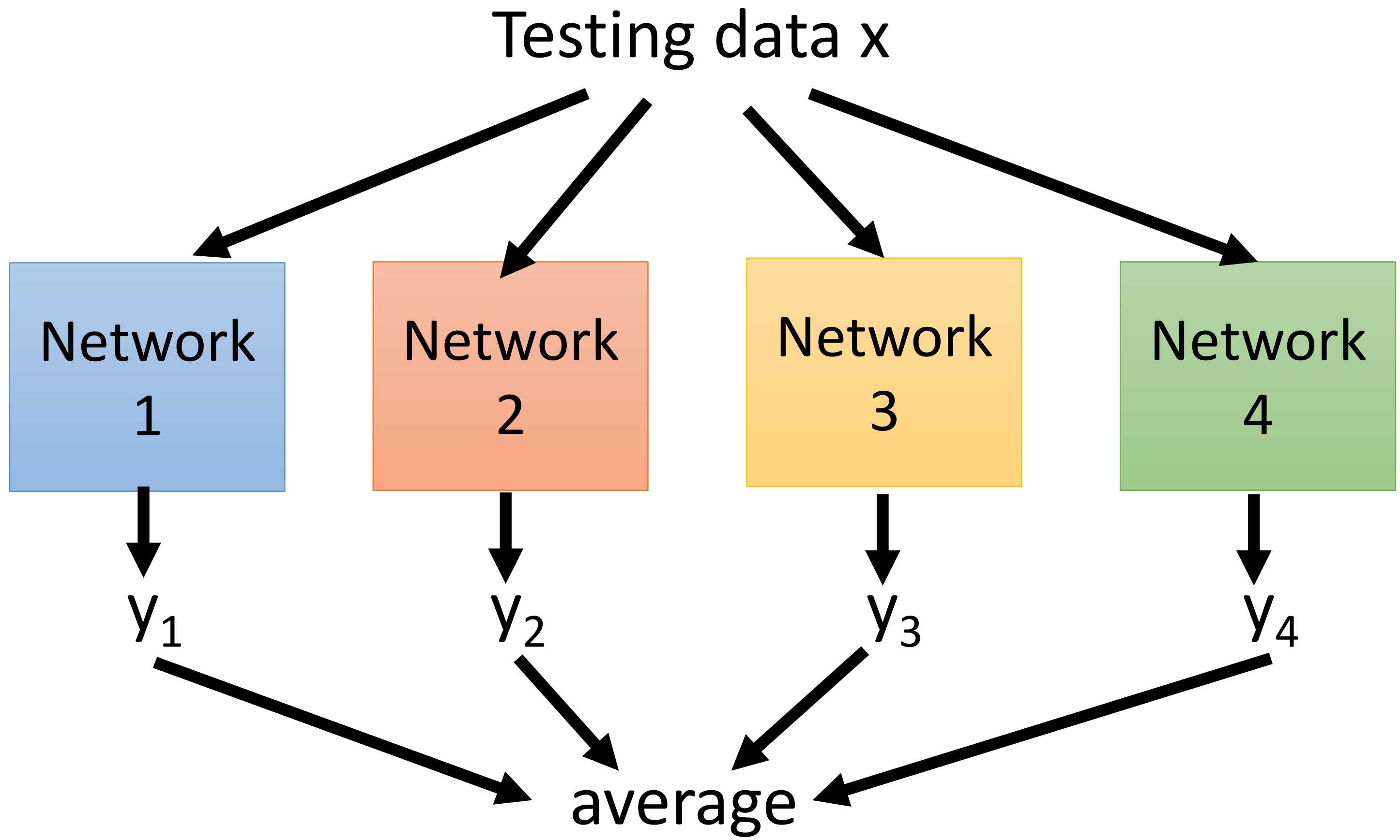
Ensemble



Train a bunch of networks with different structures

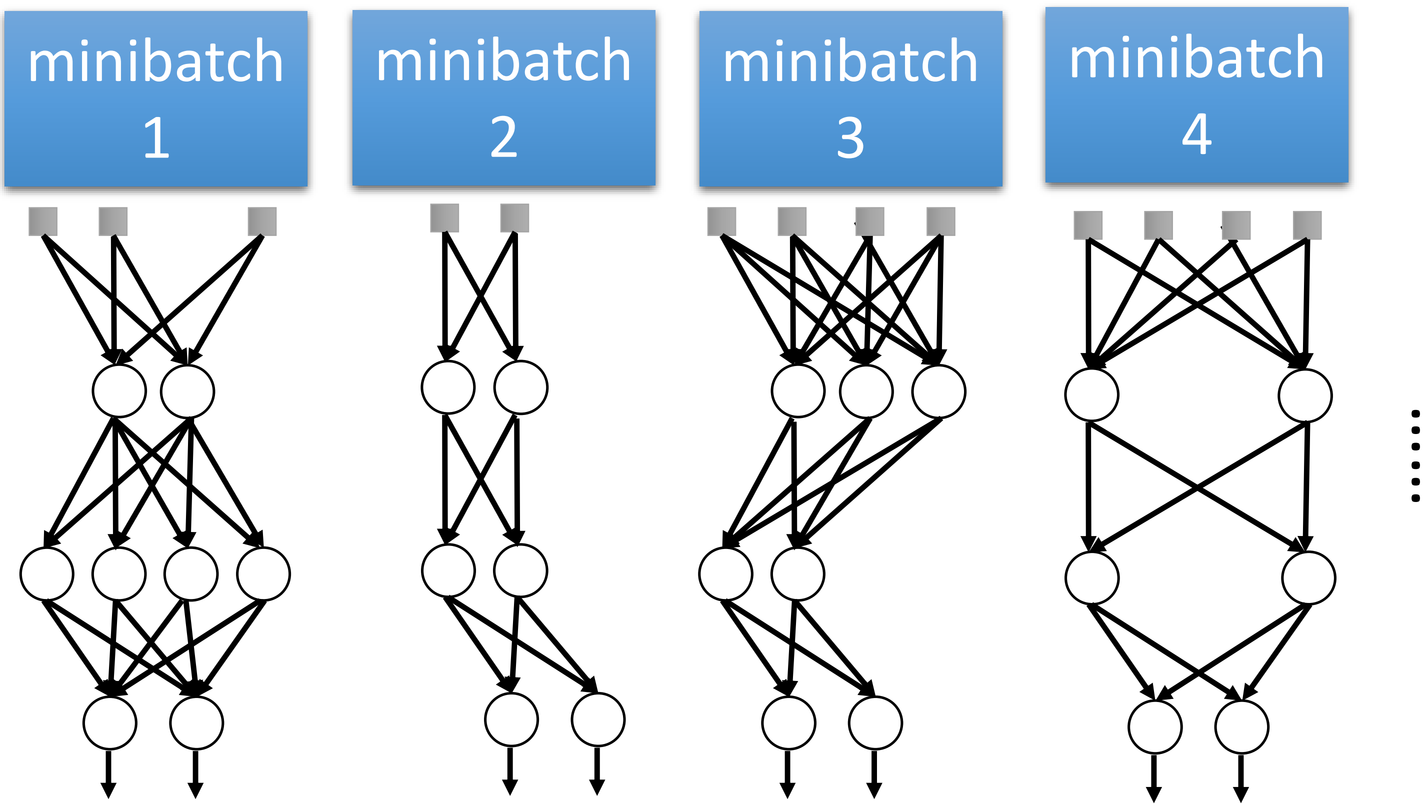
Dropout is a kind of ensemble.

Ensemble



Dropout is a kind of ensemble.

因为参数是shared的，所以用一个minibatch训练一个结构也没有问题



<u>Training of Dropout</u>

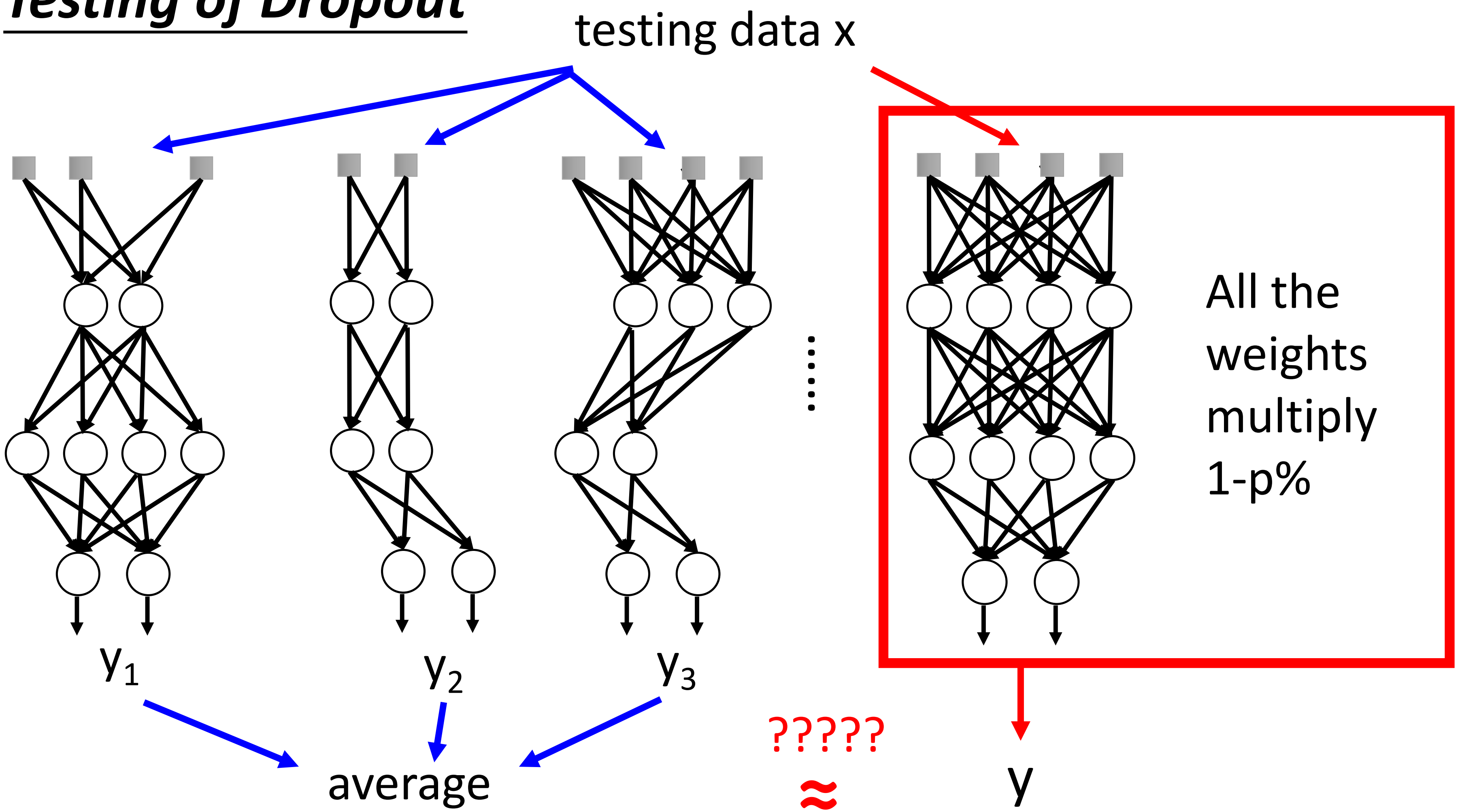
M neurons

2^M possible networks

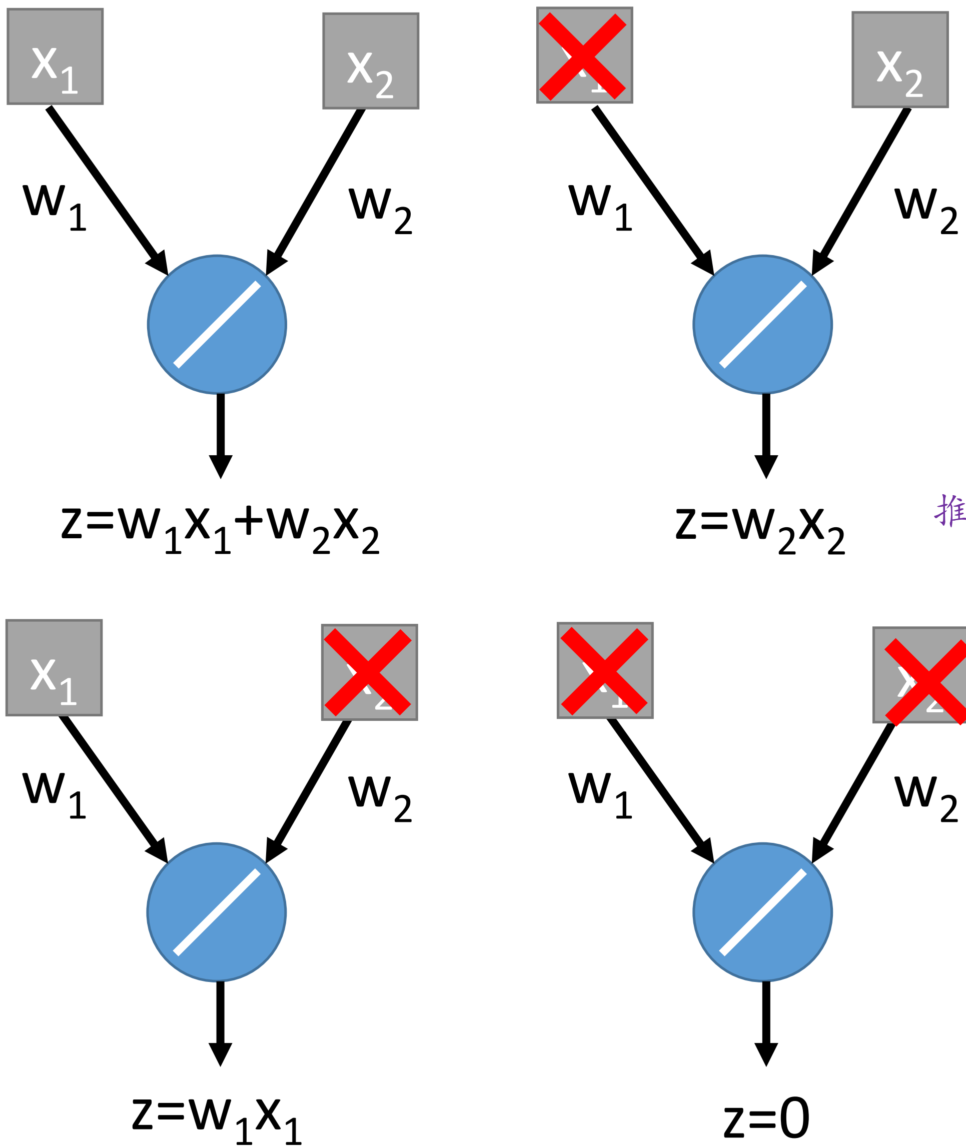
- ➢Using one mini-batch to train one network
- ➢Some parameters in the network are shared

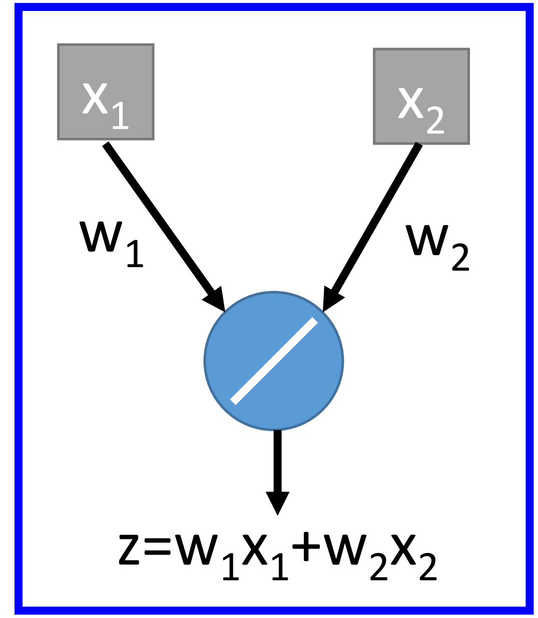
Dropout is a kind of ensemble.

Testing of Dropout

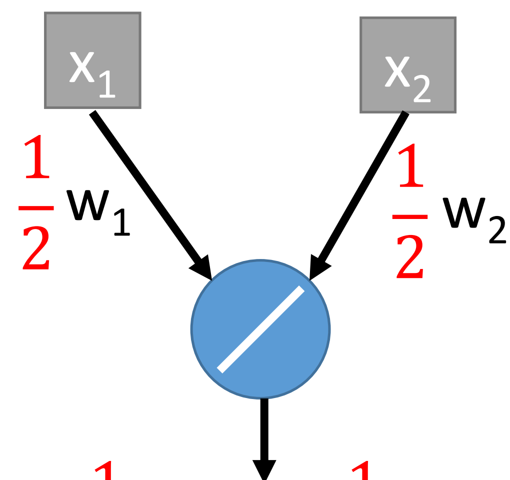


Testing of Dropout



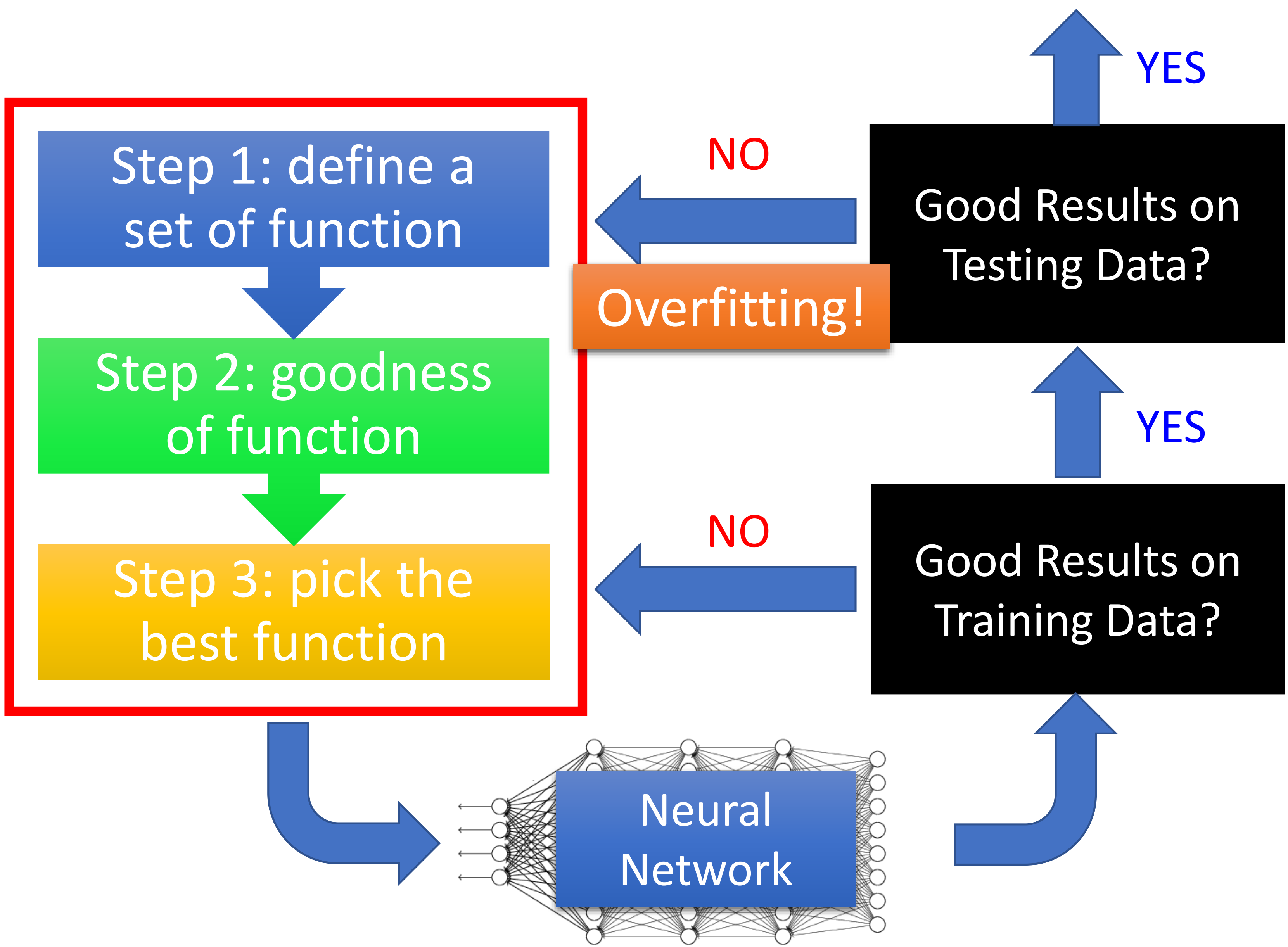


推导成立的前提是NN是一个线性的结构。

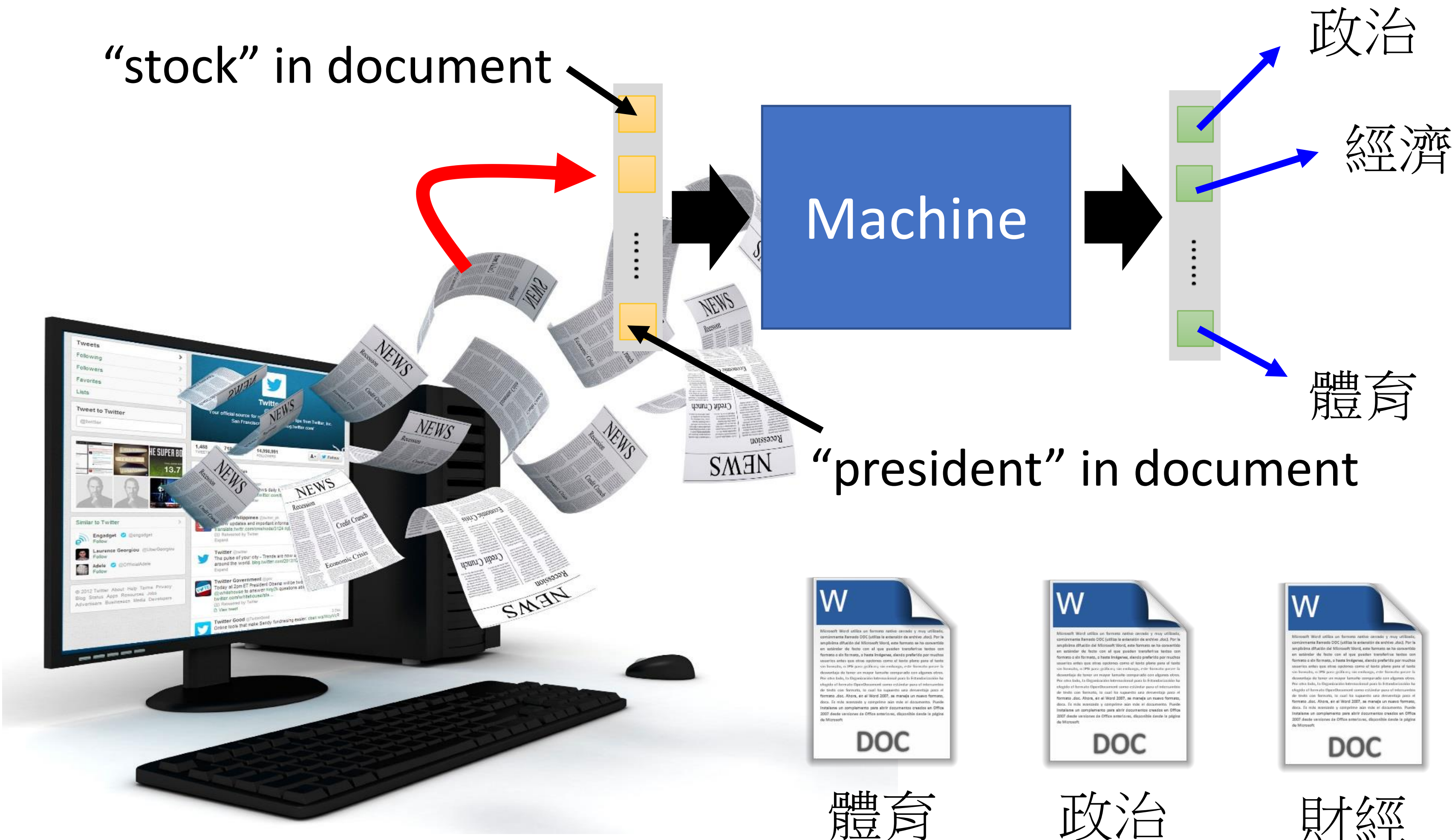


$$z=\frac{1}{2}w_1x_1+\frac{1}{2}w_2x_2$$

Recipe of Deep Learning



Try another task



http://top-breaking-news.com/

Try another task

```
In [8]: x_train.shape
Out[8]: (8982, 1000)

In [9]: y_train.shape
Out[9]: (8982, 46)

In [10]: x_test.shape
Out[10]: (2246, 1000)

In [11]: y_test.shape
Out[11]: (2246, 46)
```

```
In [12]: x_train[0]
Out[12]:
array([ 0.,   1.,   1.,   0.,   1.,   1.,   1.,   1.,   1
        0.,   0.,   1.,   1.,   1.,   0.,   1.,   0.,   0
        1.,   0.,   0.,   1.,   1.,   0.,   1.,   0.,   0
        1.,   0.,   0.,   0.,   1.,   1.,   0.,   0.,   0
        1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0
        0.,   0.,   1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
        0.,   0.,   0.,   0.,   0.,   1.,   1.,   0.,   0.,   0.,   0.,   1.,   1.,
        0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   1.,   0.,
        0.,   0.,   0.,   0.,   0.,   1.,   0.,   1.,   0.,   0.,   0.,   0.,   0.,
        0.,   0.,   0.,   0.,   0.,   0.,   0.,   1.,   0.,   0.,   0.,   0.,   0.,
        0.,   0.,   1.,   0.,   1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
        0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   1.,   0.,   0.,   1.,   1.,
        0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
        0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
        0.,   0.,   0.,   0.,   1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
        0.,   0.,   1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   1.,
        0.,   1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
        0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
```

```
In [13]: y_train[0]
Out[13]:
array([ 0.,   0.,   0.,   1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
        0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
        0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
        0.,   0.,   0.,   0.,   0.,   0.,   0.])
```

Live Demo