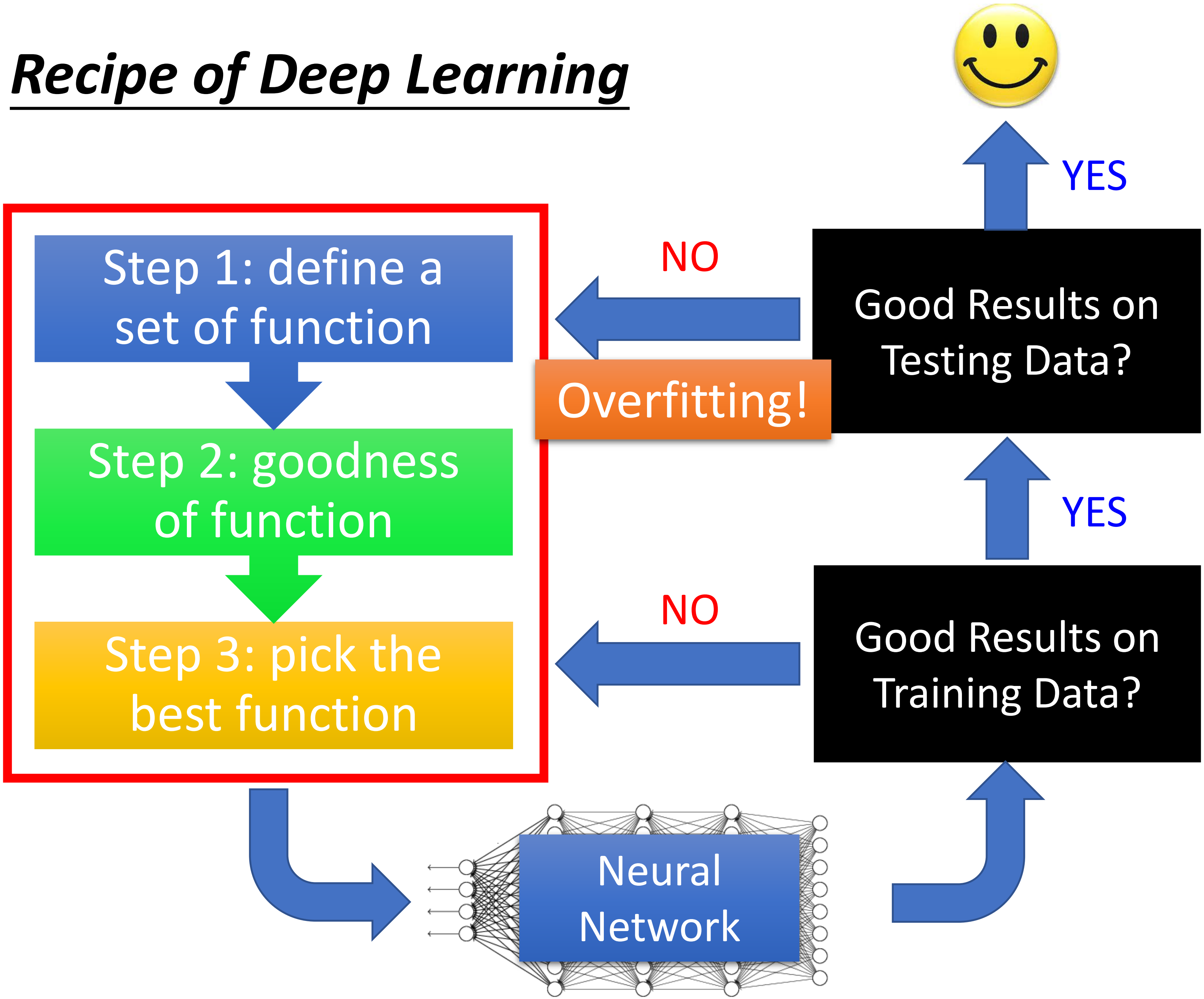
Tips for Deep Learning

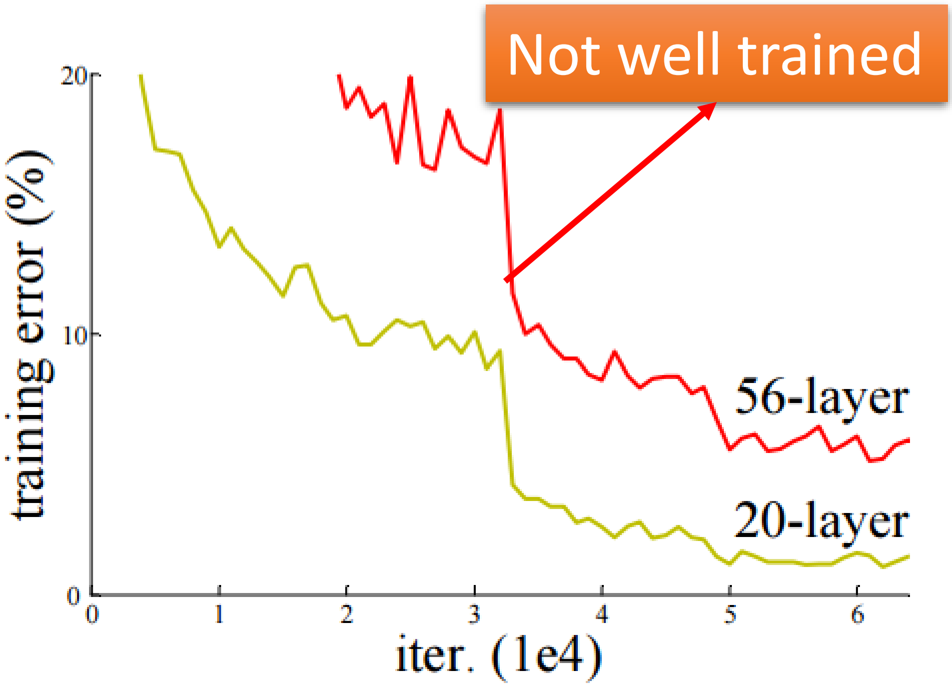
Recipe of Deep Learning



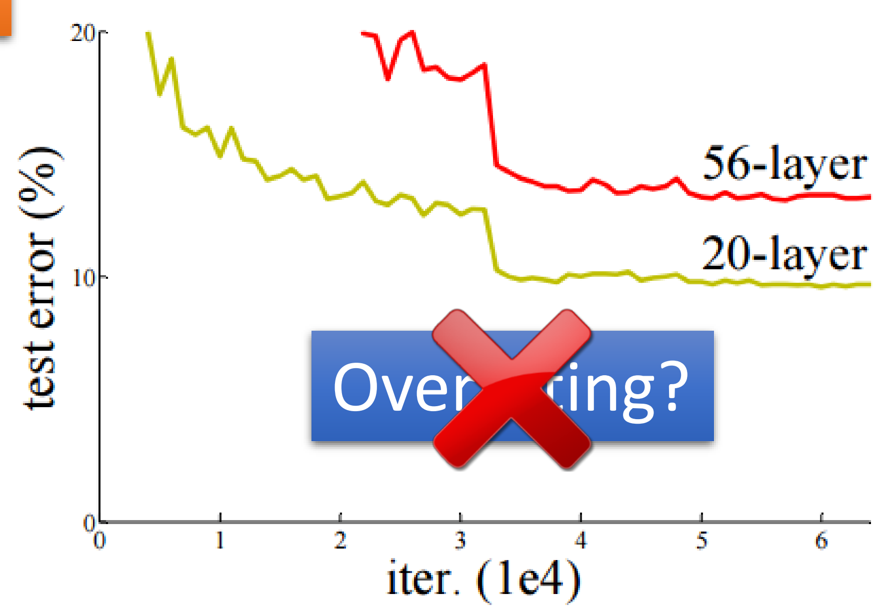
Do not always blame Overfitting

有時候不見得是overfitting，可能是在train的時候就沒有到達minimum

有時候可能把loss function從MSE改成CrossEntropy即可



Training Data



Testing Data

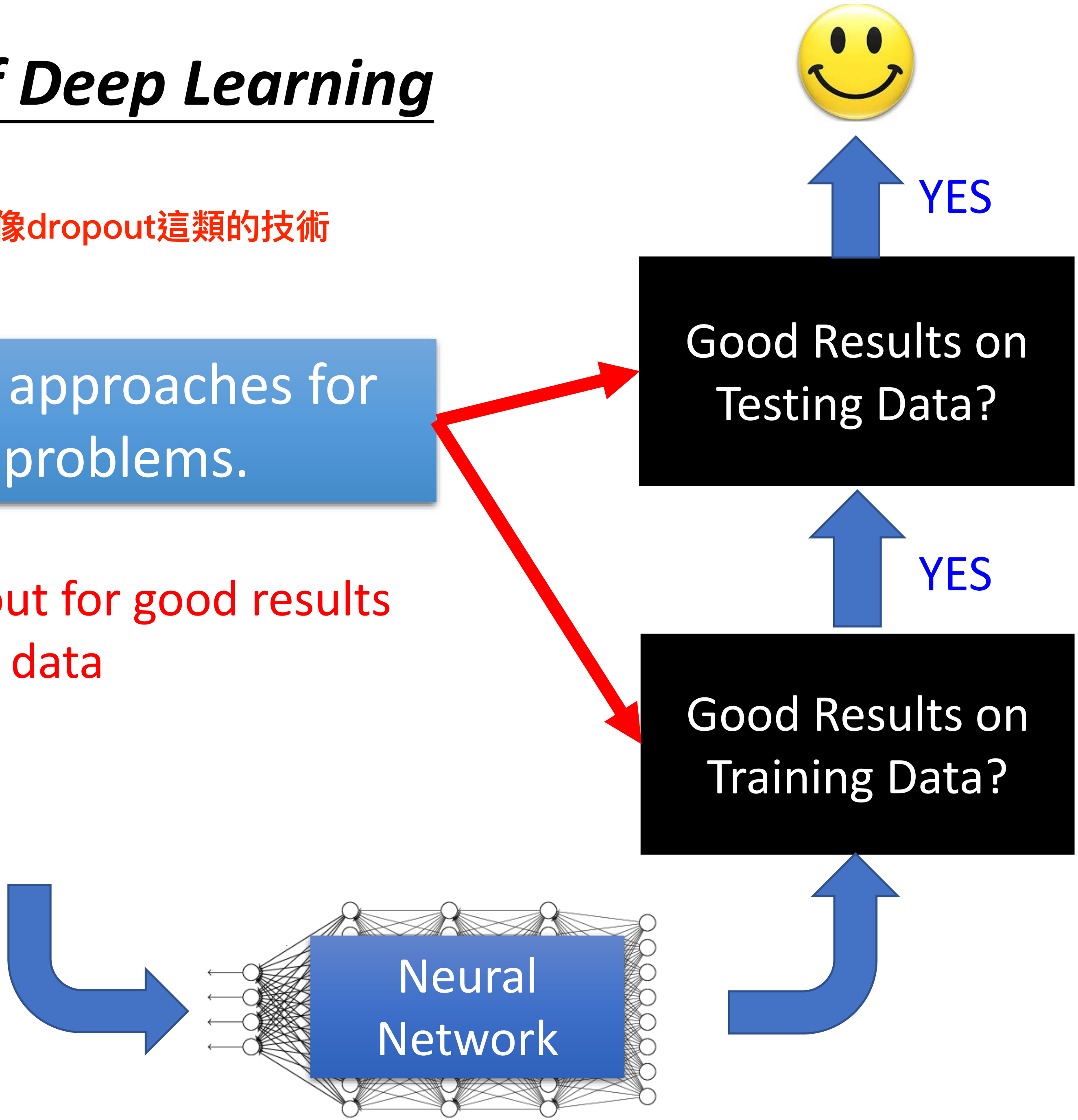
Deep Residual Learning for Image Recognition
http://arxiv.org/abs/1512.03385

Recipe of Deep Learning

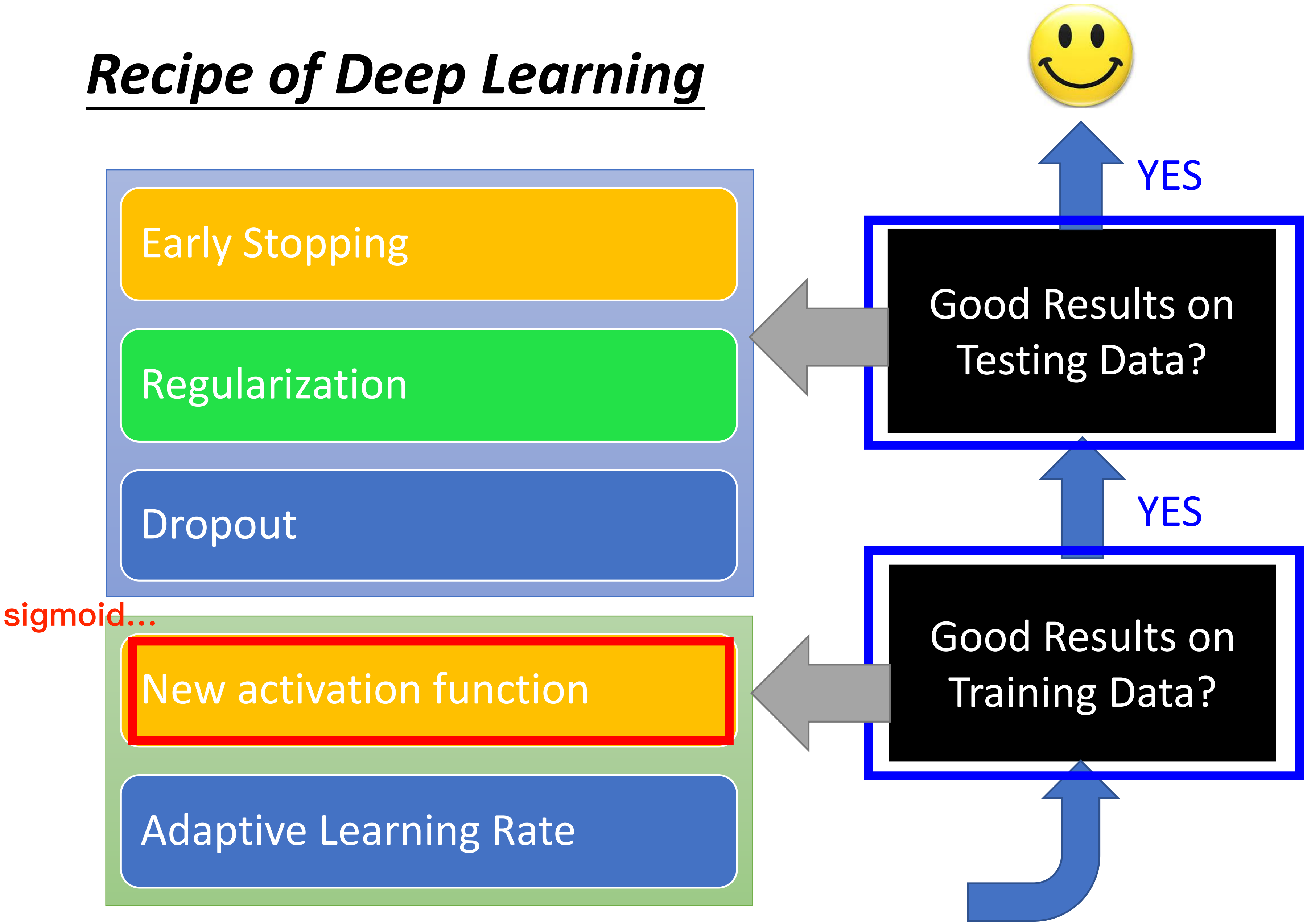
可利用像dropout這類的技術

Different approaches for different problems.

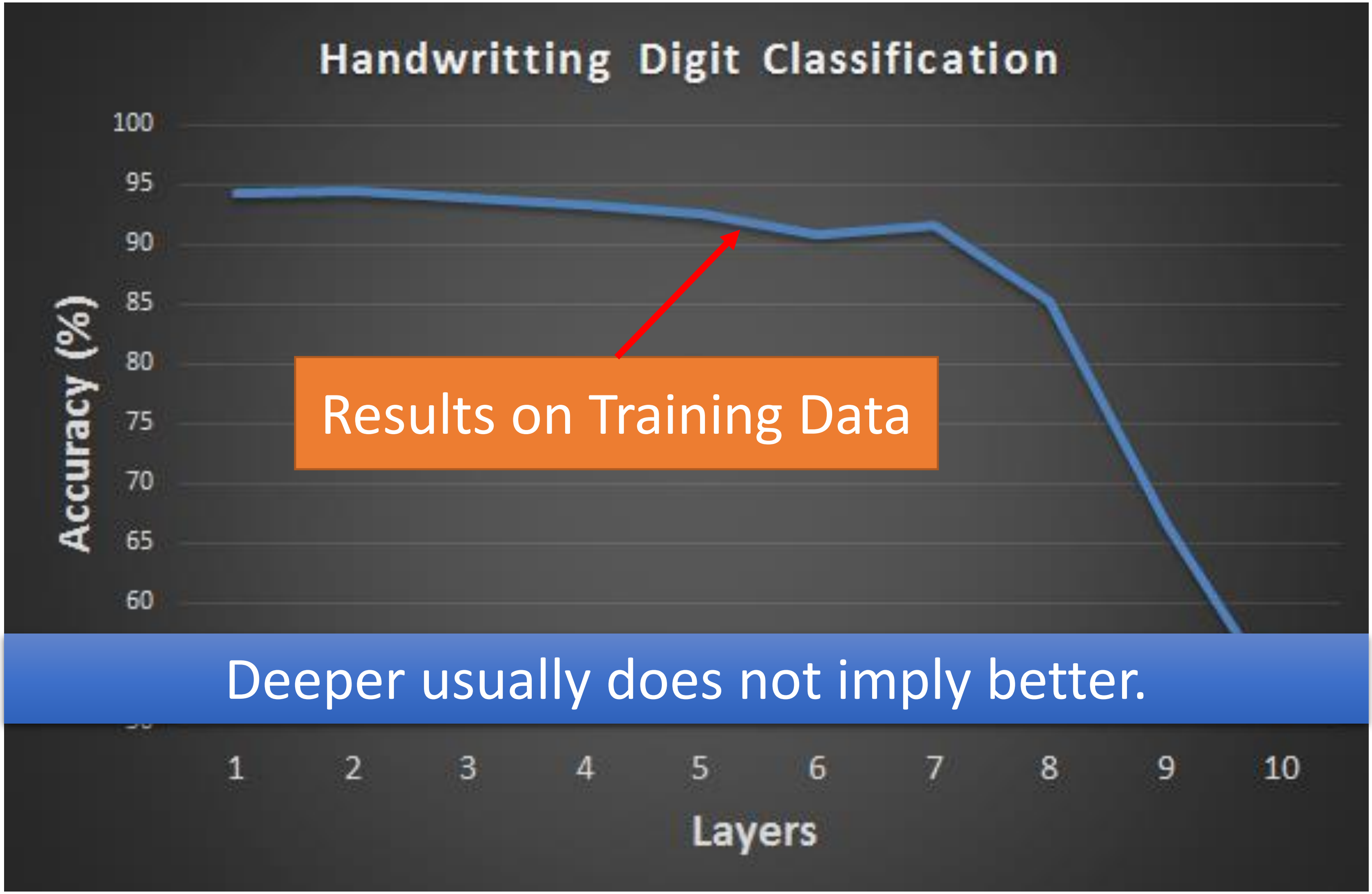
e.g. dropout for good results on testing data



Recipe of Deep Learning

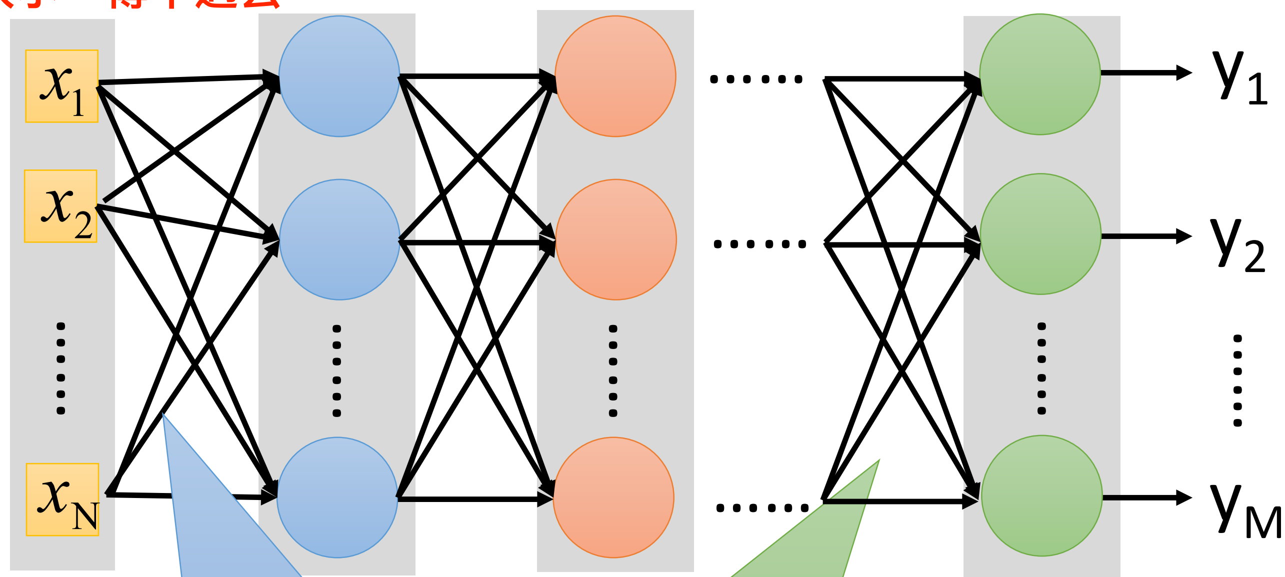


Hard to get the power of Deep …



Vanishing Gradient Problem

gradient太小，傳不過去



Smaller gradients

Learn very slow

Almost random

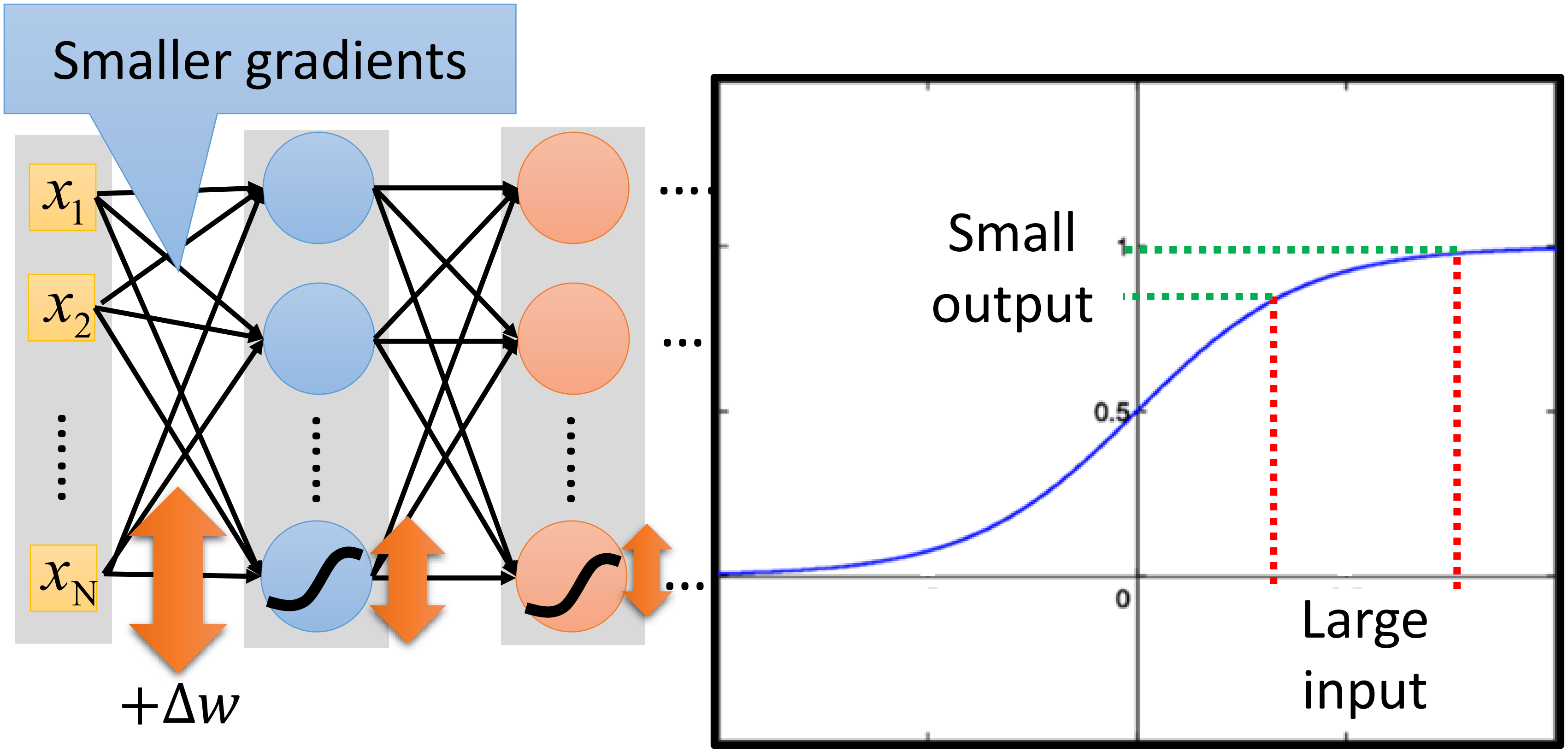
Larger gradients

Learn very fast

Already converge

based on random!?

Vanishing Gradient Problem



雖然可以透過adagram動態調整learning rate，但是直接修改activation function更直觀

Intuitive way to compute the derivatives ...

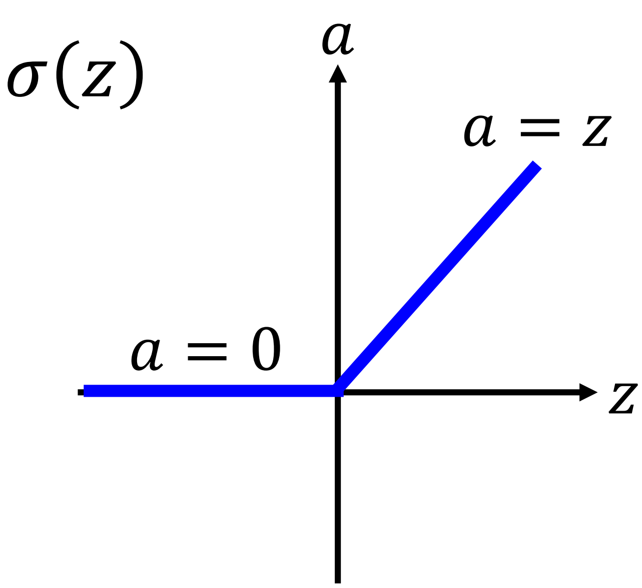
雖然在前面調整w，但是透過sigmoid　function會壓縮變化量

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

ReLU

取代sigmoid funcion

- Rectified Linear Unit (ReLU)



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

Reason:

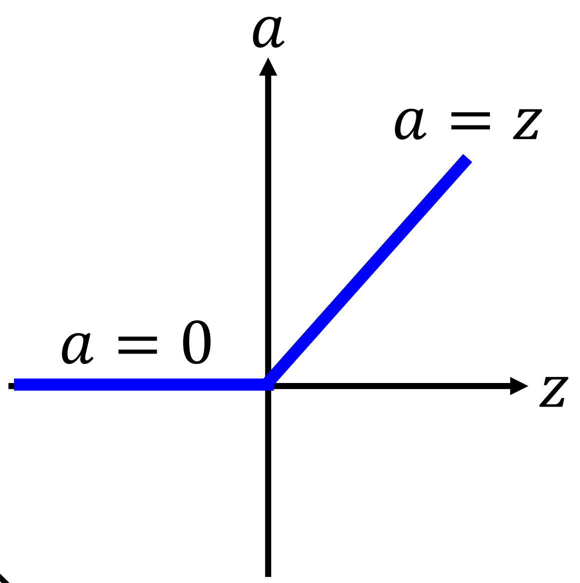
1. Fast to compute
2. Biological reason
3. Infinite sigmoid with different biases
4. Vanishing gradient problem

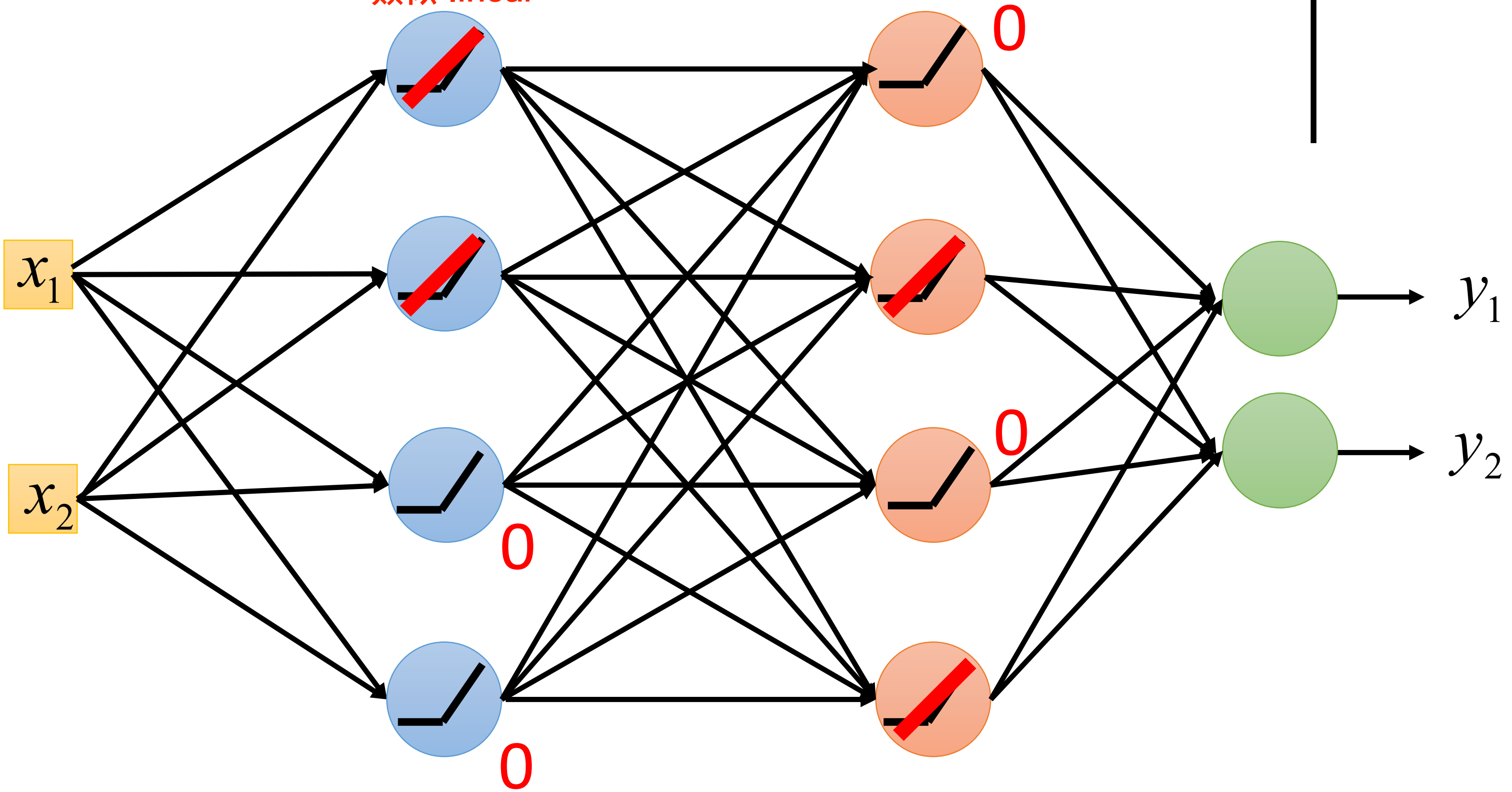
可以解决vanish gradient

ReLU

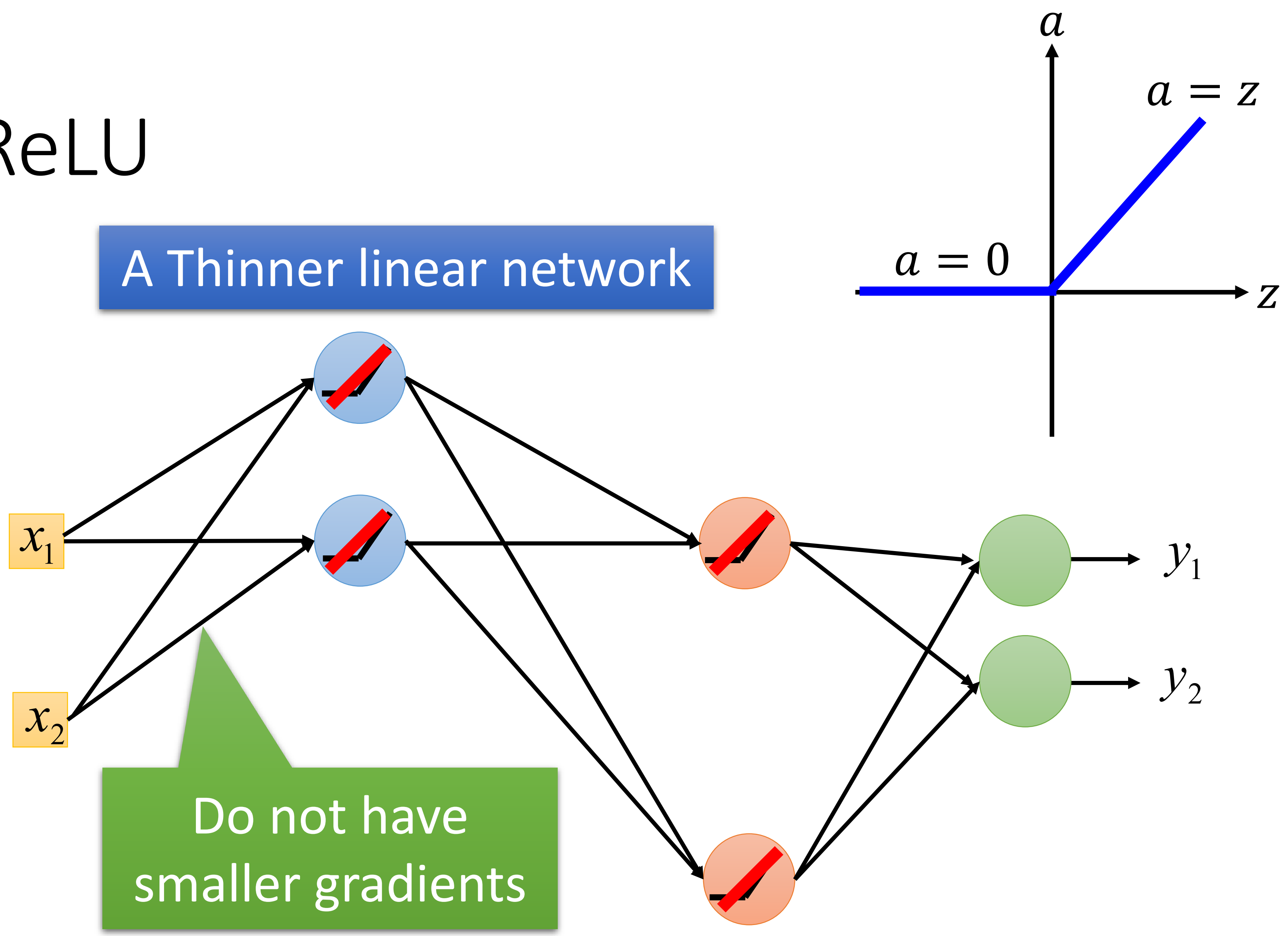
input < 0，output = 0

input >= 0，output = input
類似 linear





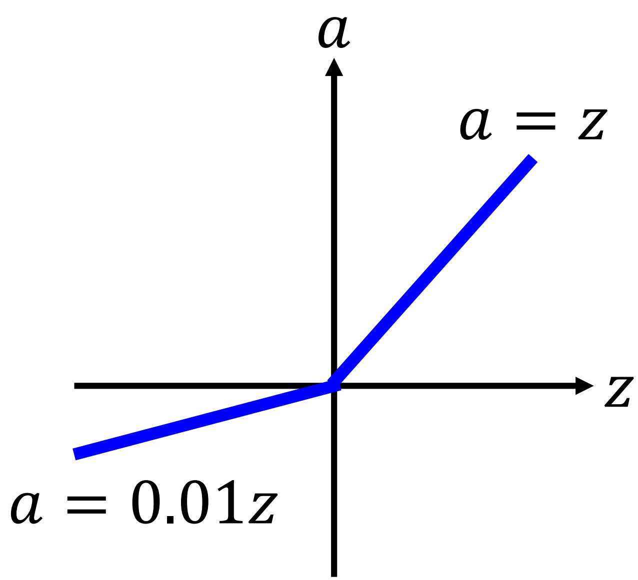
ReLU



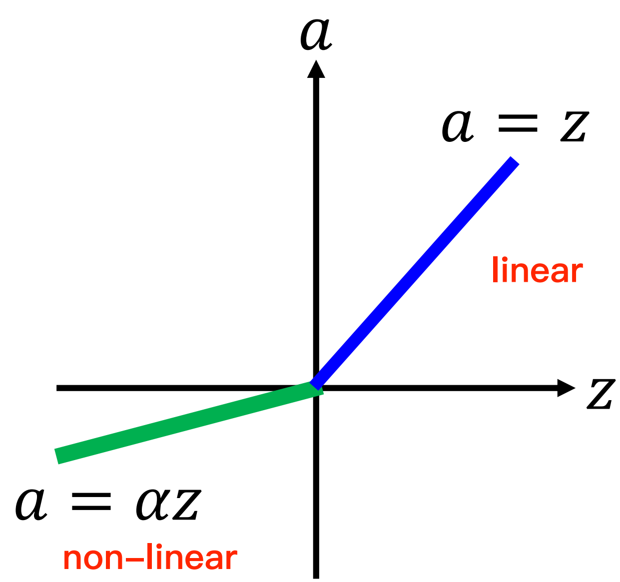
A Thinner linear network
a
$a = z$
$a = 0$
z
x_1
x_2
y_1
y_2
Do not have smaller gradients

ReLU - variant

Leaky ReLU



Parametric ReLU



α also learned by gradient descent

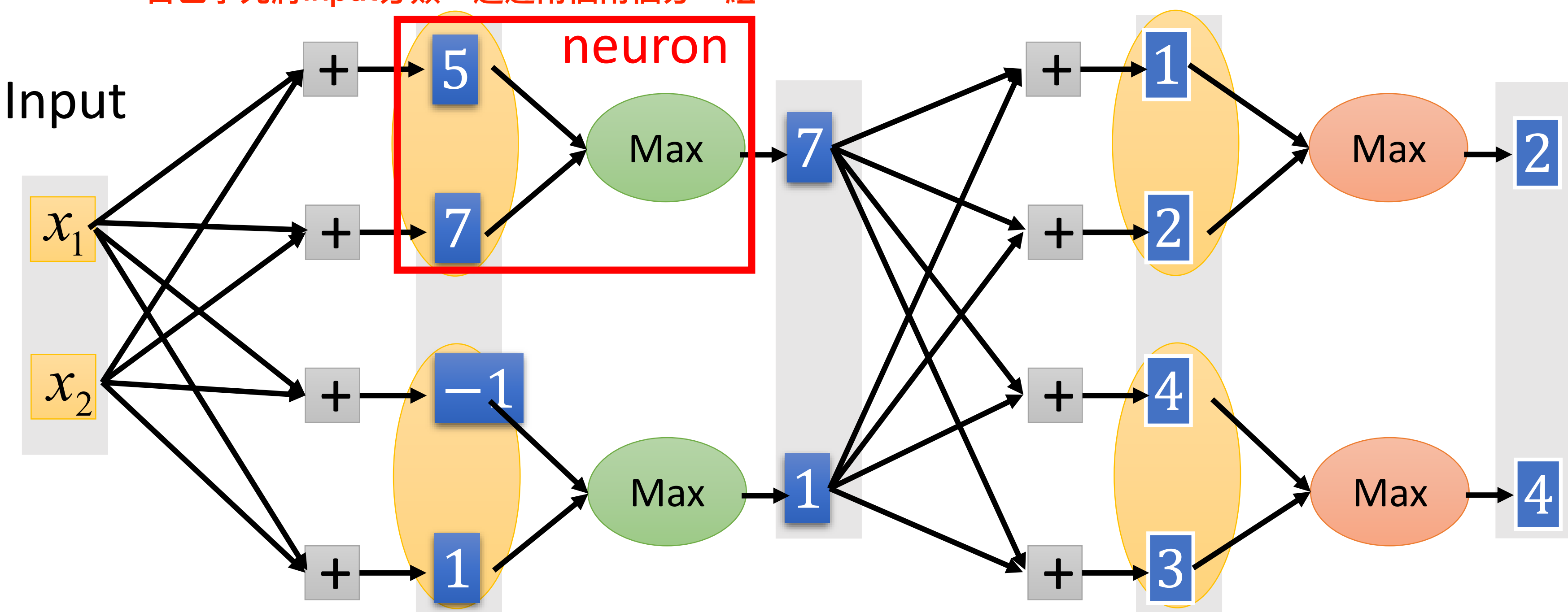
Maxout

自己學習選擇activation function

ReLU is a special cases of Maxout

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

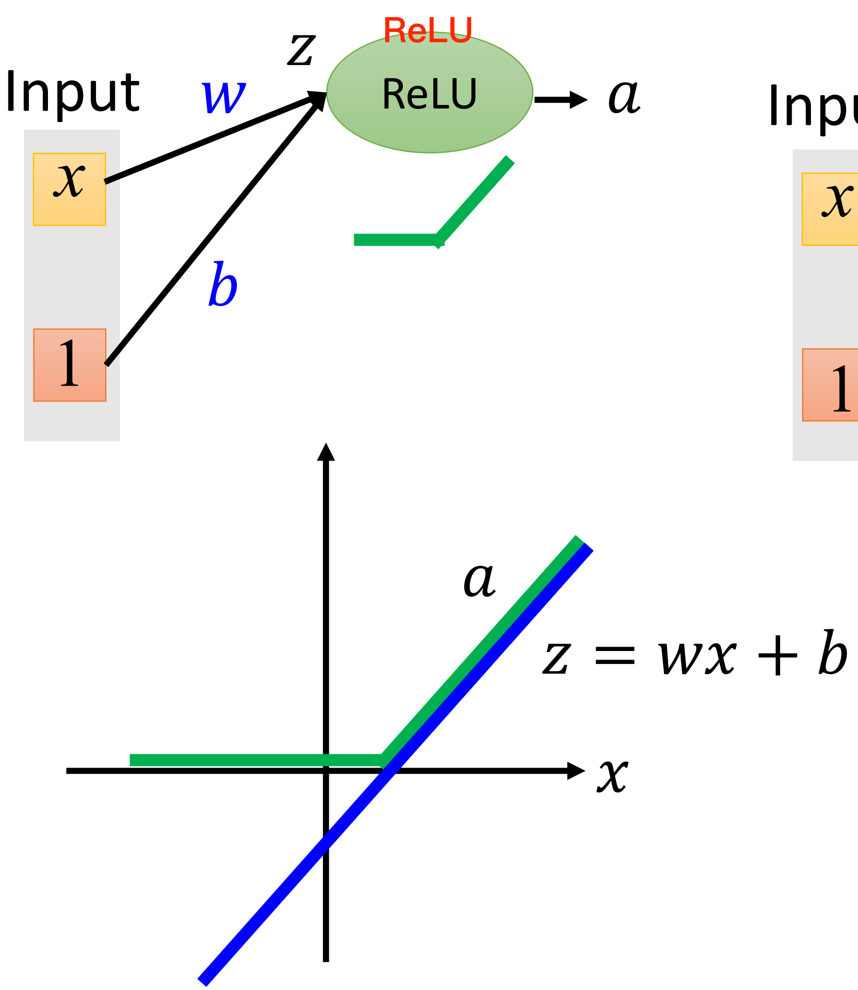
自己事先將input分類，這邊兩個兩個分一組

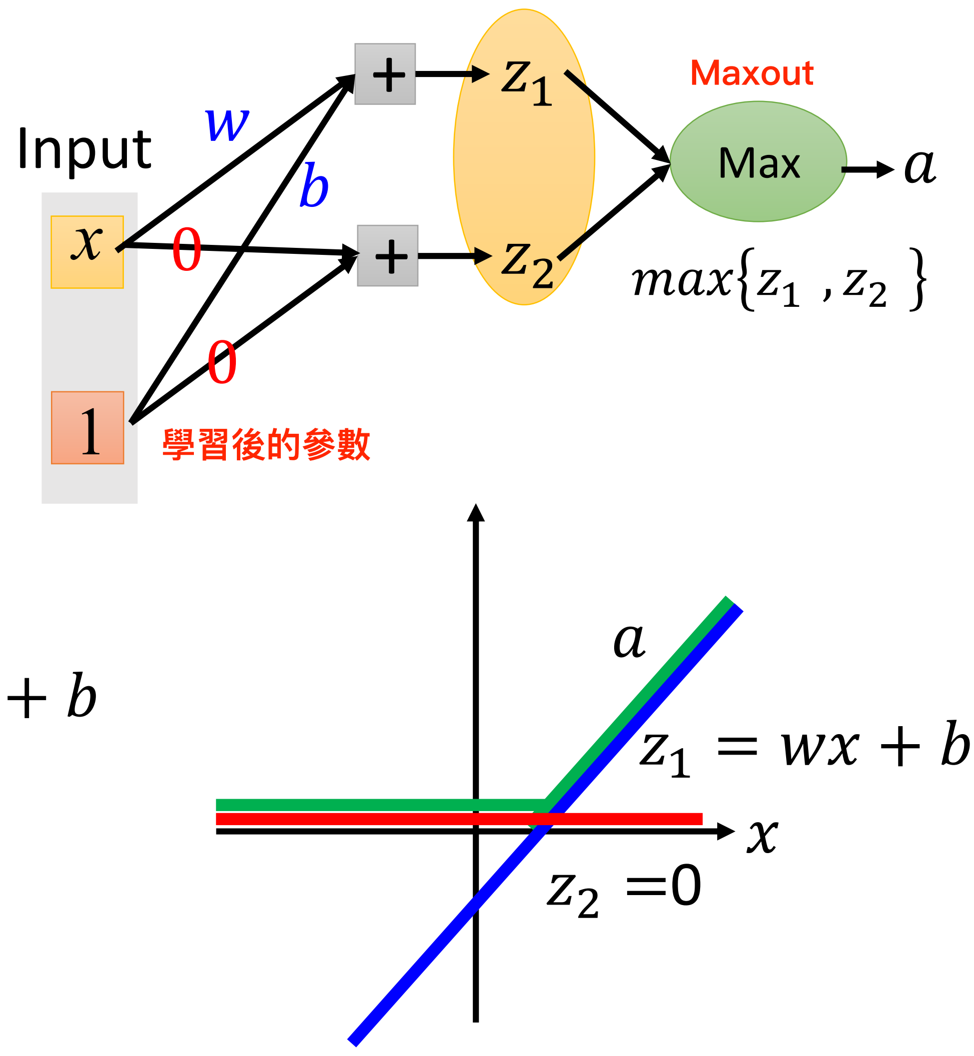


You can have more than 2 elements in a group.

Maxout

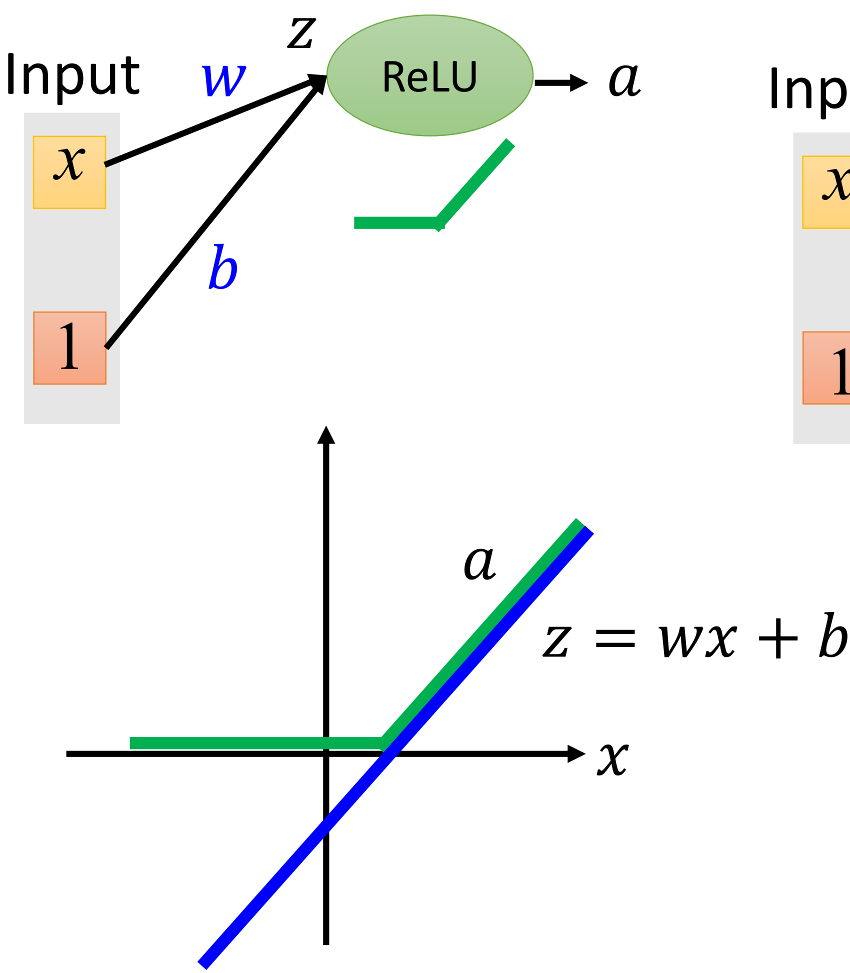
ReLU is a special cases of Maxout

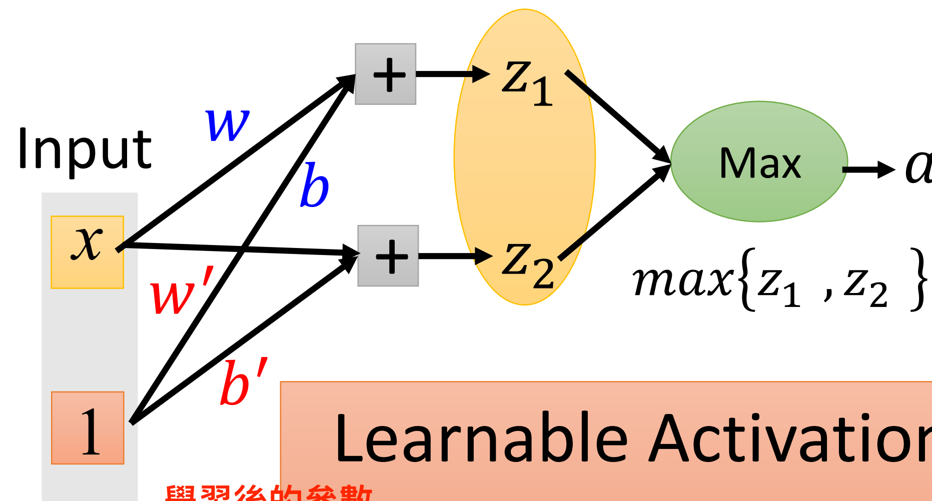




Maxout

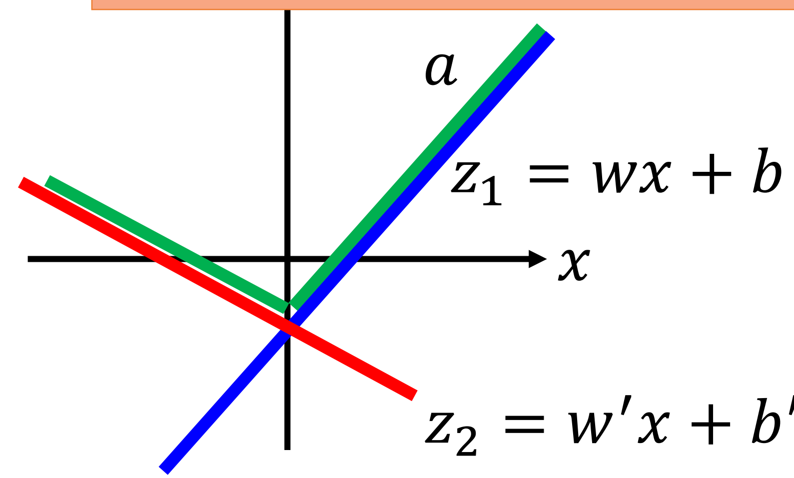
More than ReLU





學習後的參數

Learnable Activation Function

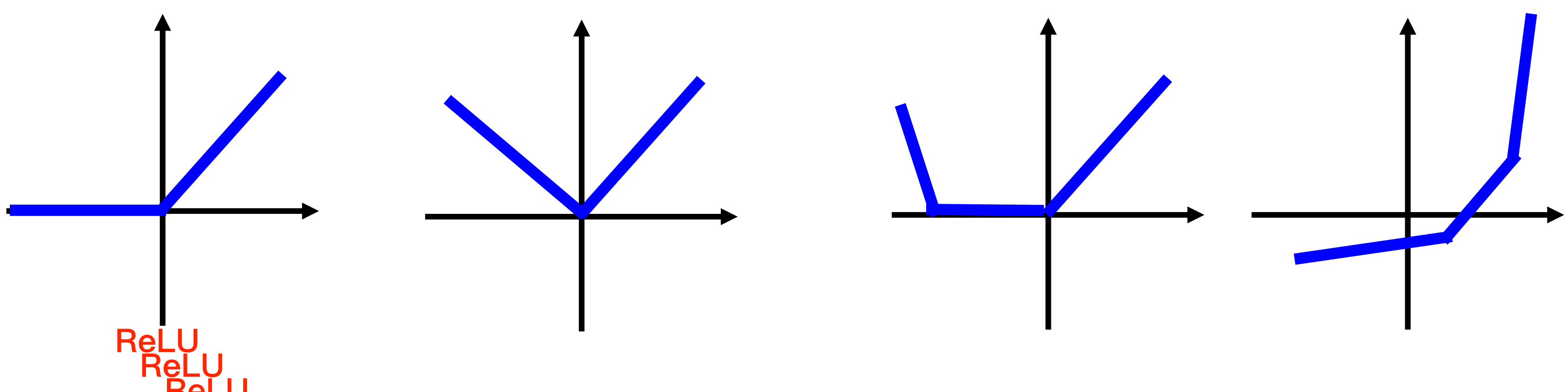


Maxout

- 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

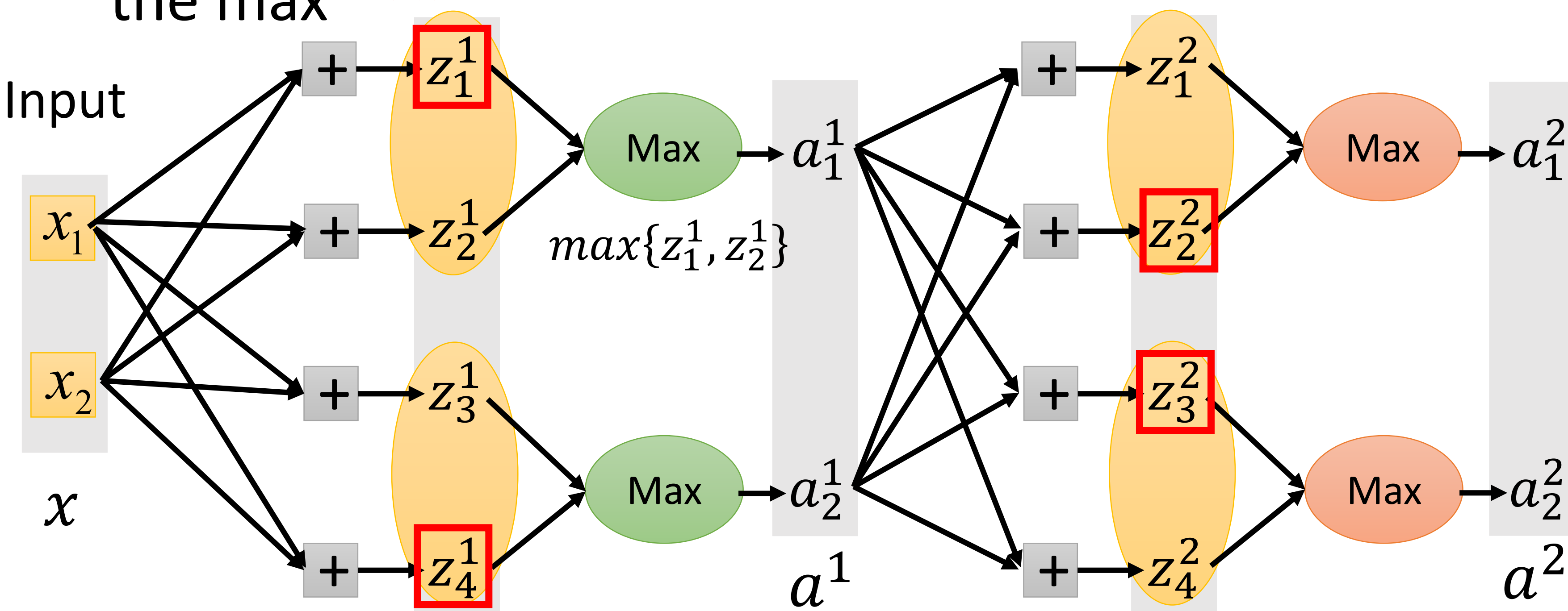
2 elements in a group

3 elements in a group



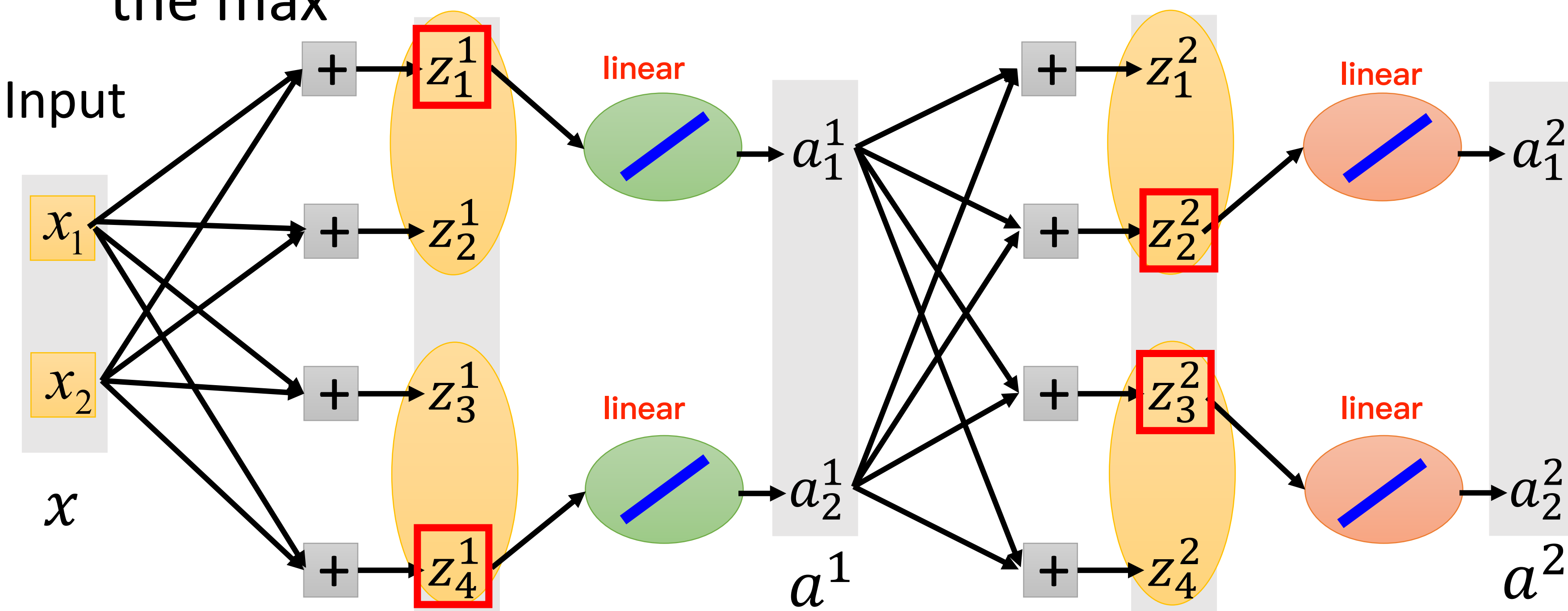
Maxout - Training 可微分？

- Given a training data x, we know which z would be the max 假設紅色框框較大



Maxout - Training

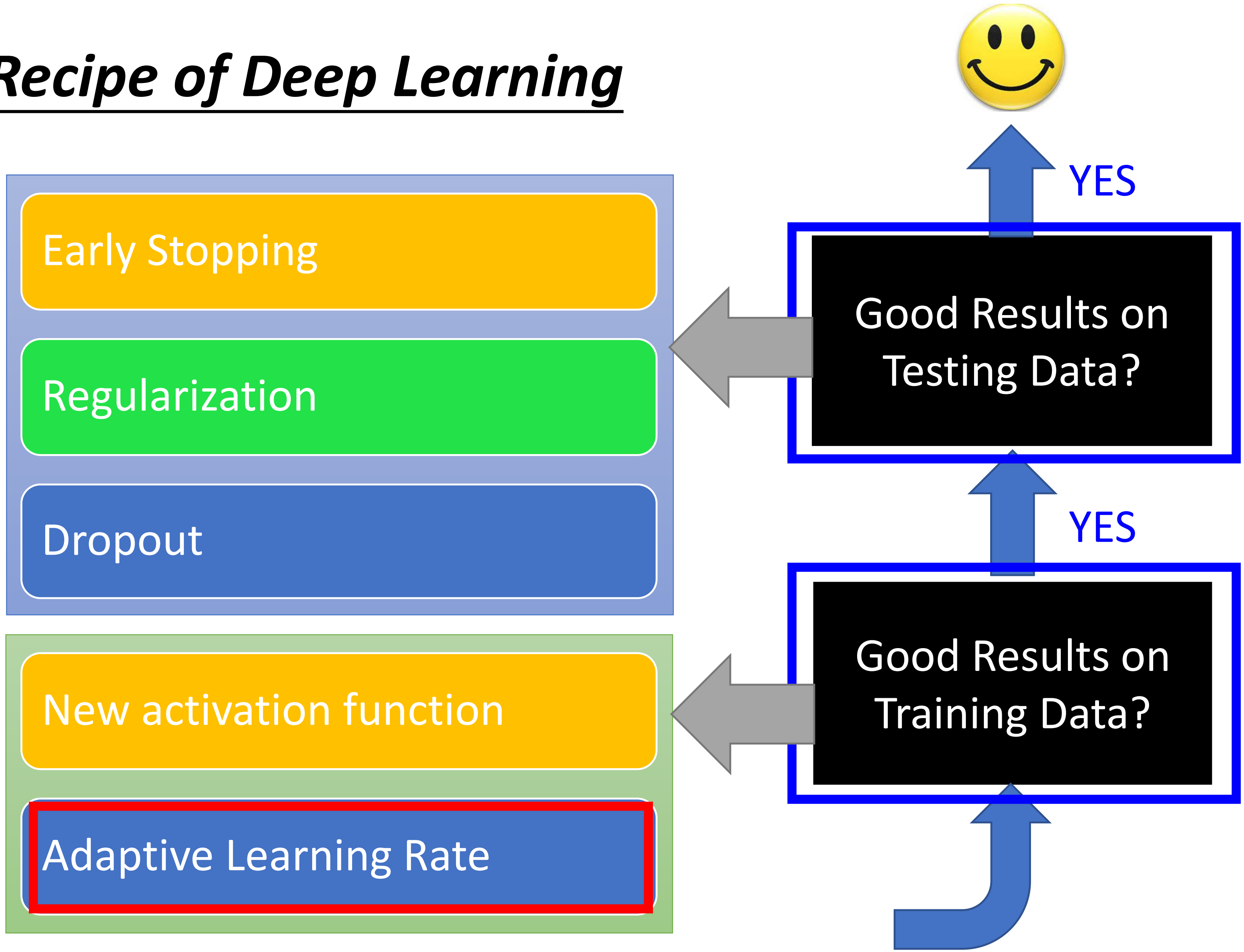
- Given a training data x, we know which z would be the max



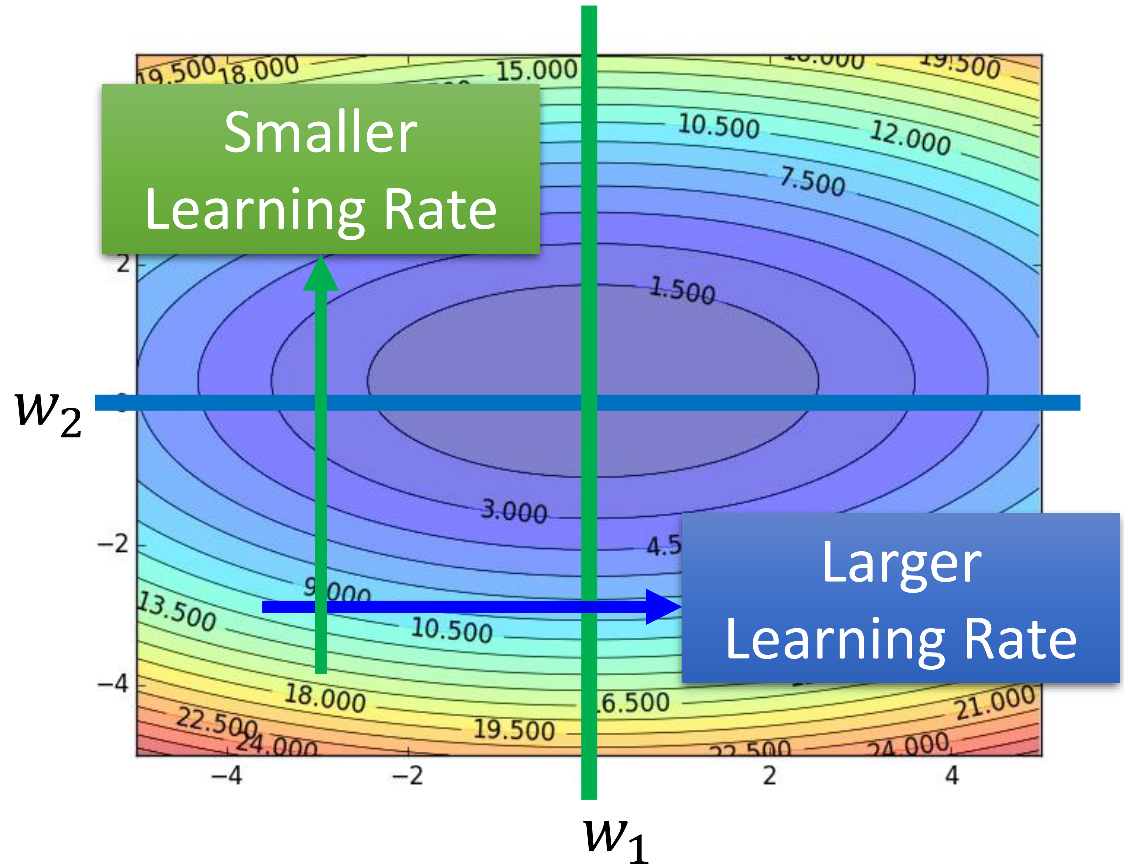
- Train this thin and linear network 每次分類每筆資料都有可能是max

Different thin and linear network for different examples

Recipe of Deep Learning



Review



Adagrad

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

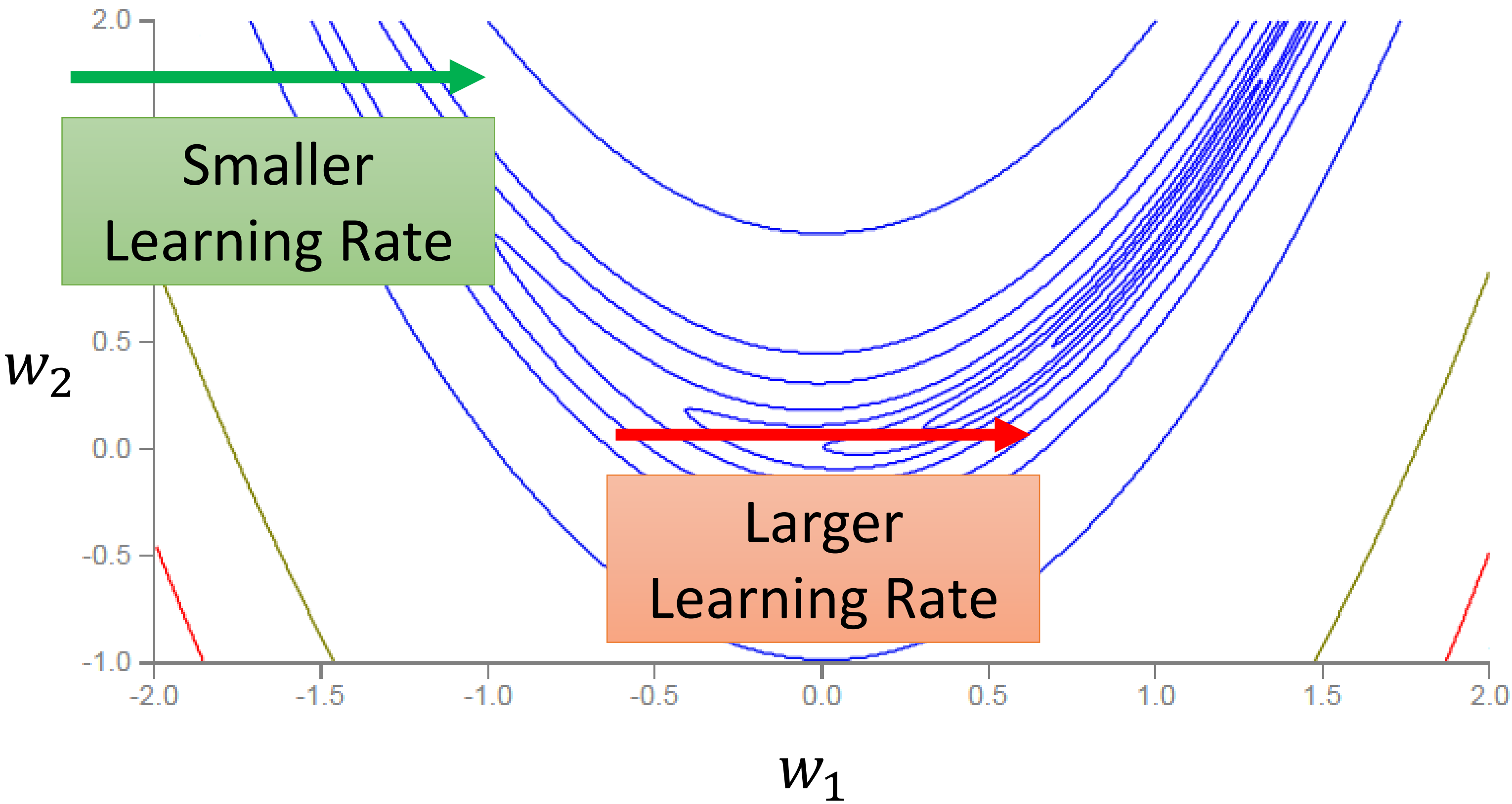
Use first derivative to estimate second derivative

RMSProp

用來train Neural Network

比Adagram複雜

Error Surface can be very complex when training NN.



RMSProp

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

乘上一個weight(α)

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

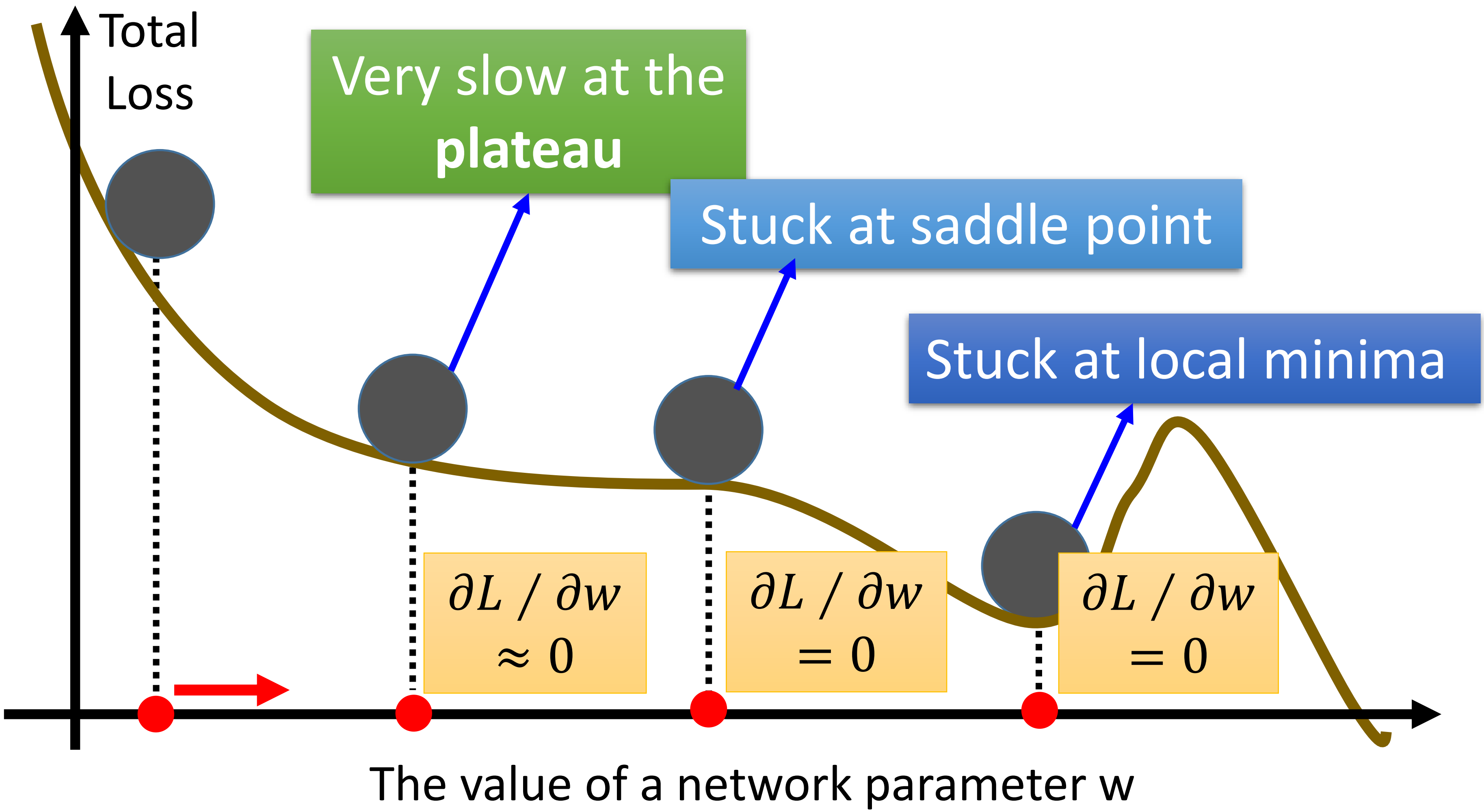
$$w^3 \leftarrow w^2 - \frac{\eta}{\sigma^2} g^2 \qquad \sigma^2 = \sqrt{\alpha(\sigma^1)^2 + (1-\alpha)(g^2)^2}$$

$$\vdots$$

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

Root Mean Square of the gradients with previous gradients being decayed

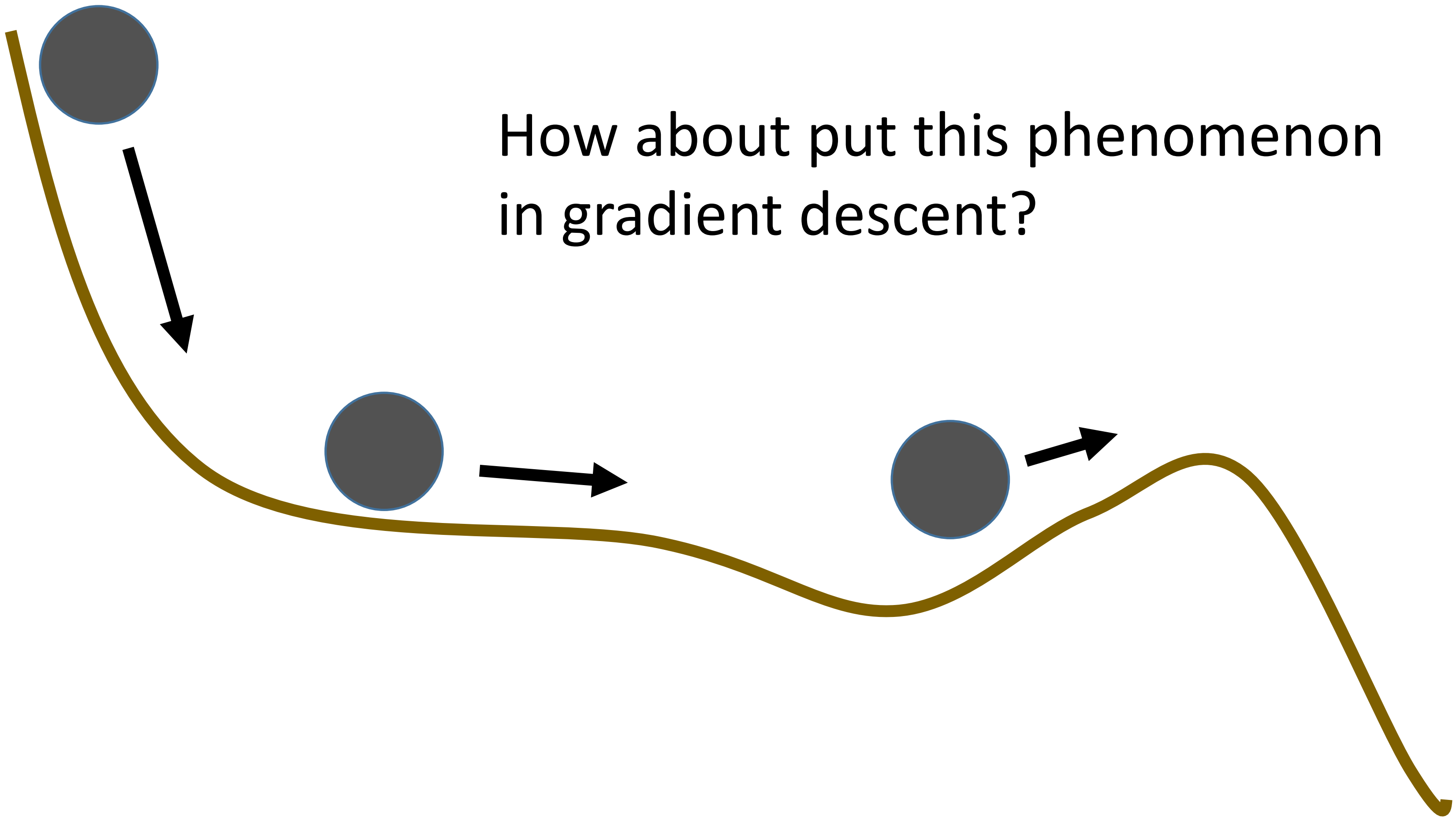
Hard to find optimal network parameters



In physical world ……

- Momentum

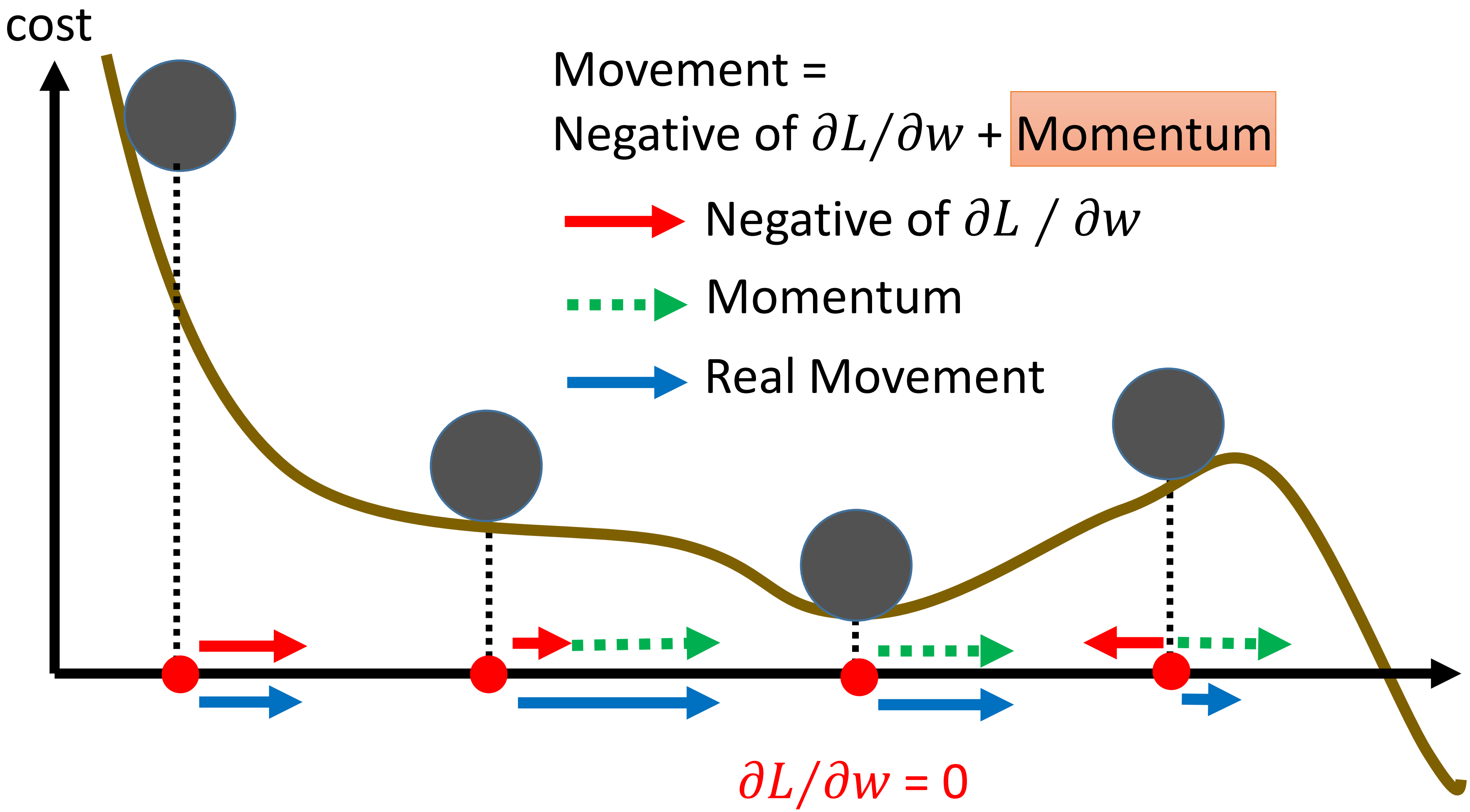
How about put this phenomenon in gradient descent?



Momentum

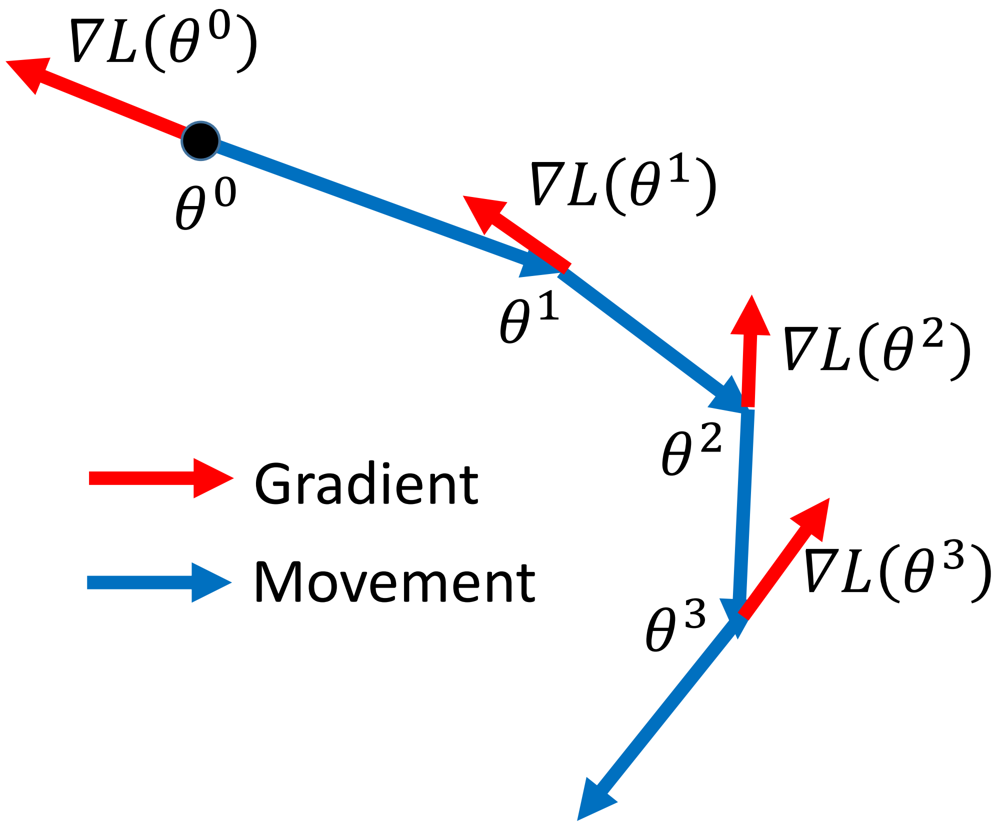
考量到參數變化慣性

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



香草;簡單;平凡

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

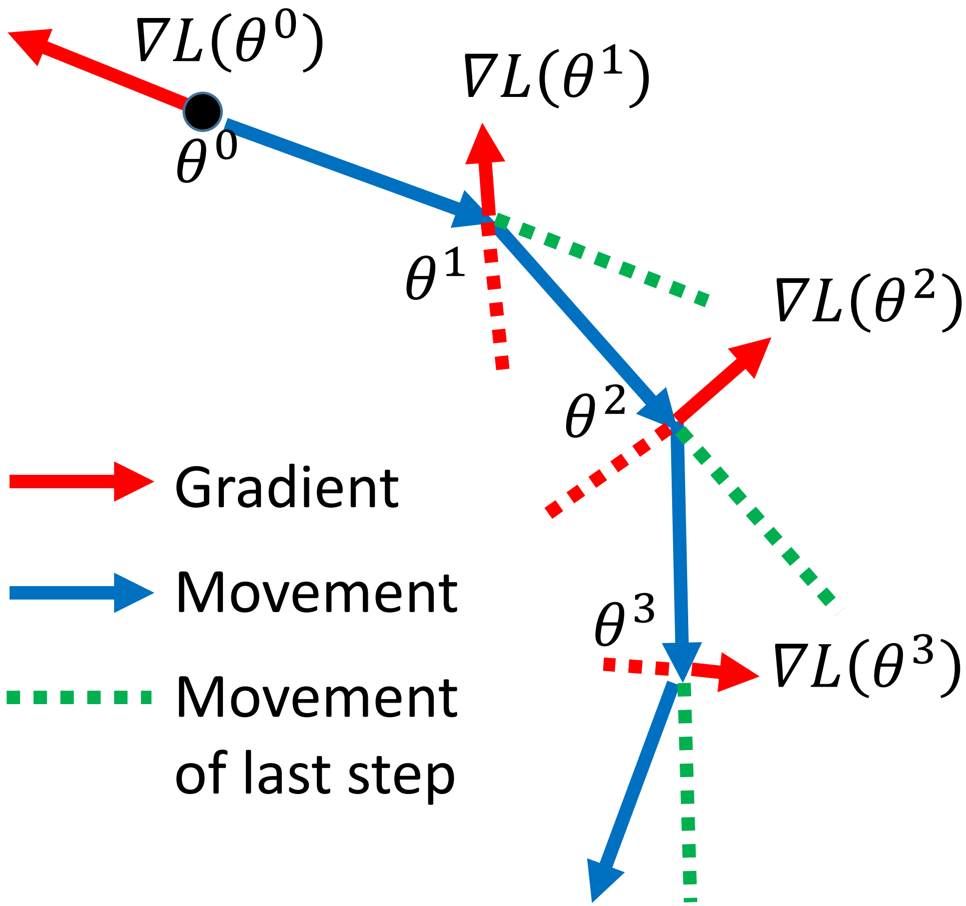
Move to $\theta^2 = \theta^1 - \eta \nabla L(\theta^1)$

⋮

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

$v^0 = 0$

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

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

⋮

Start at 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

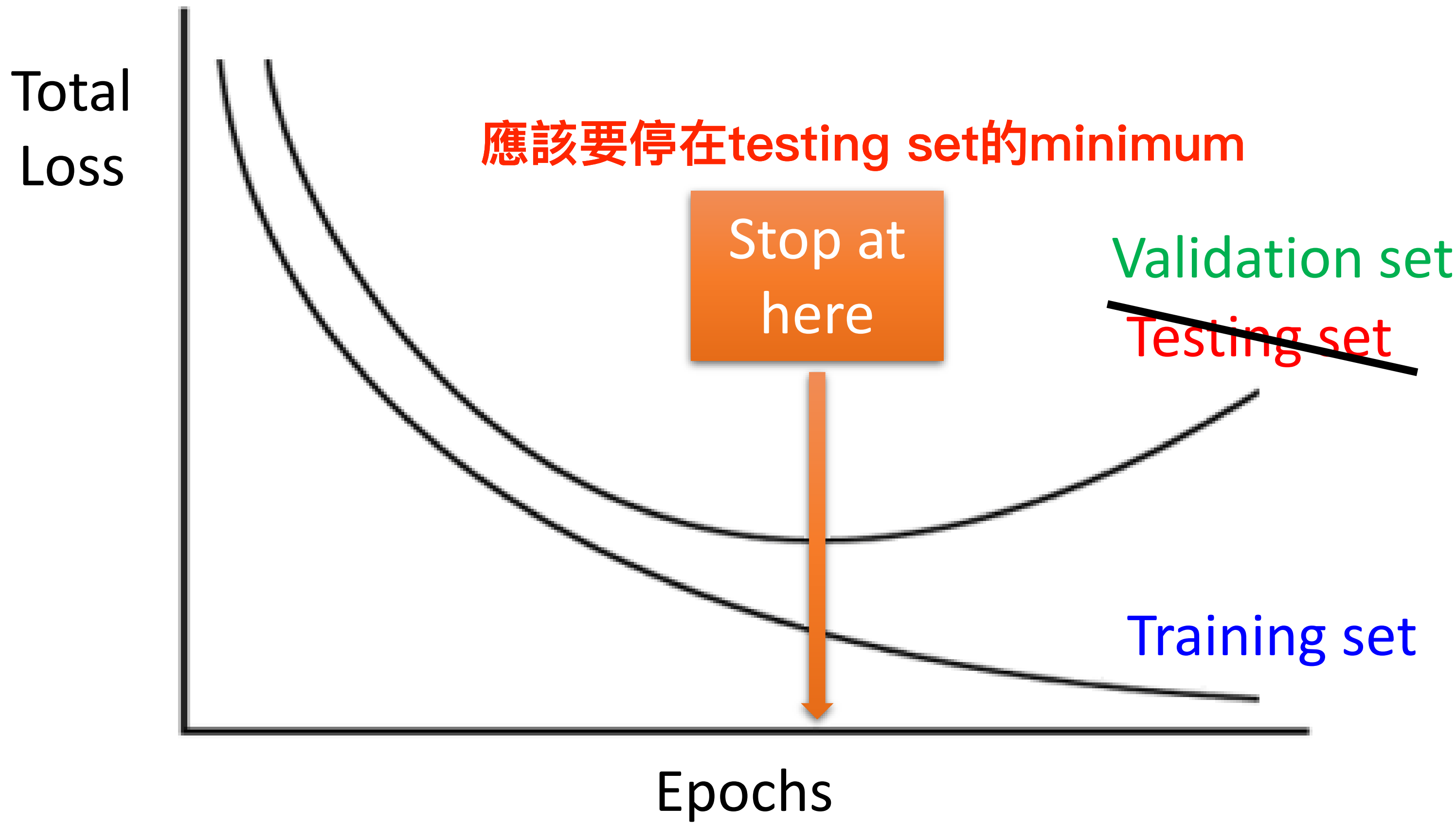
Adam

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.

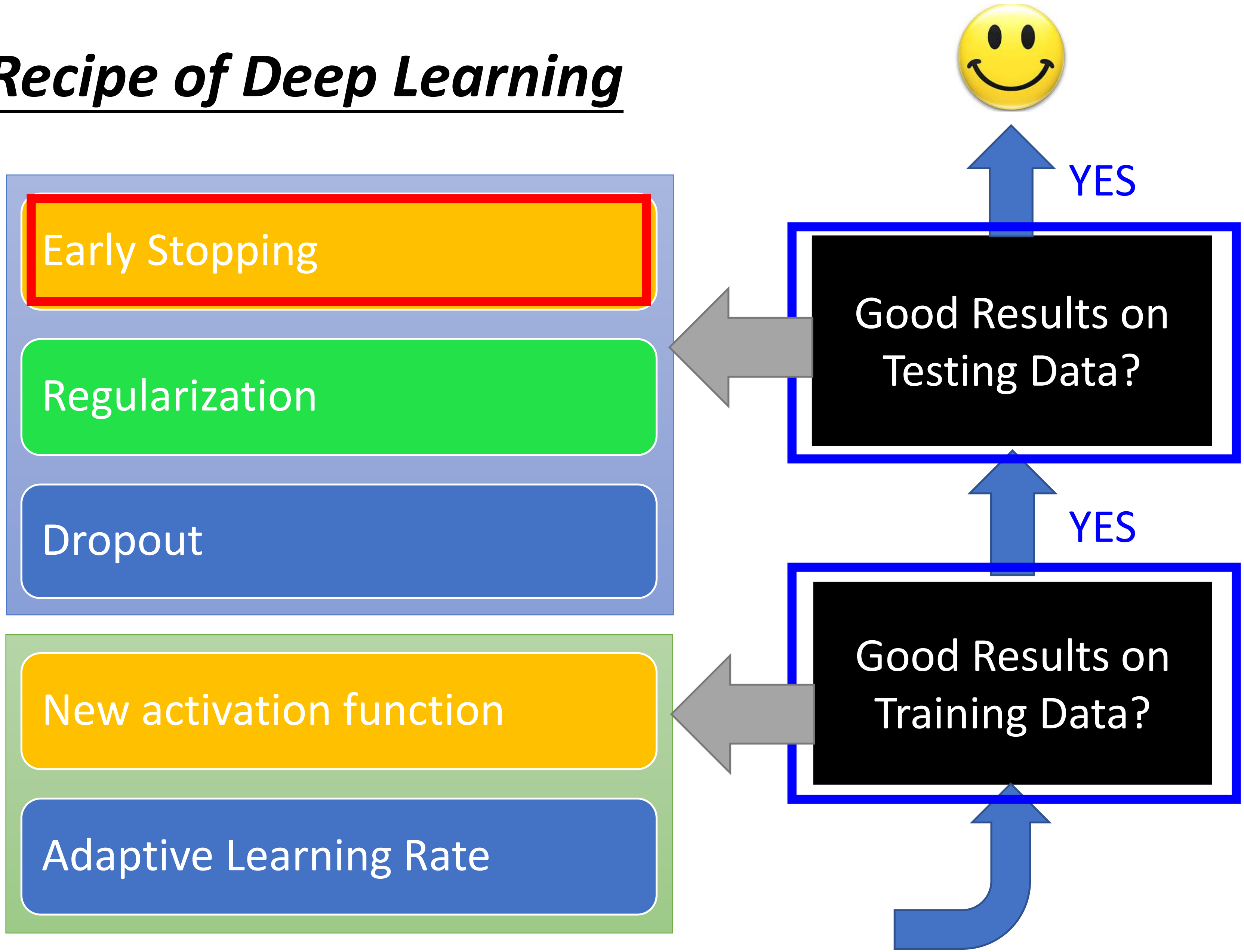
Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector) → for momentum
  $v_0 \leftarrow 0$ (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)

Early Stopping

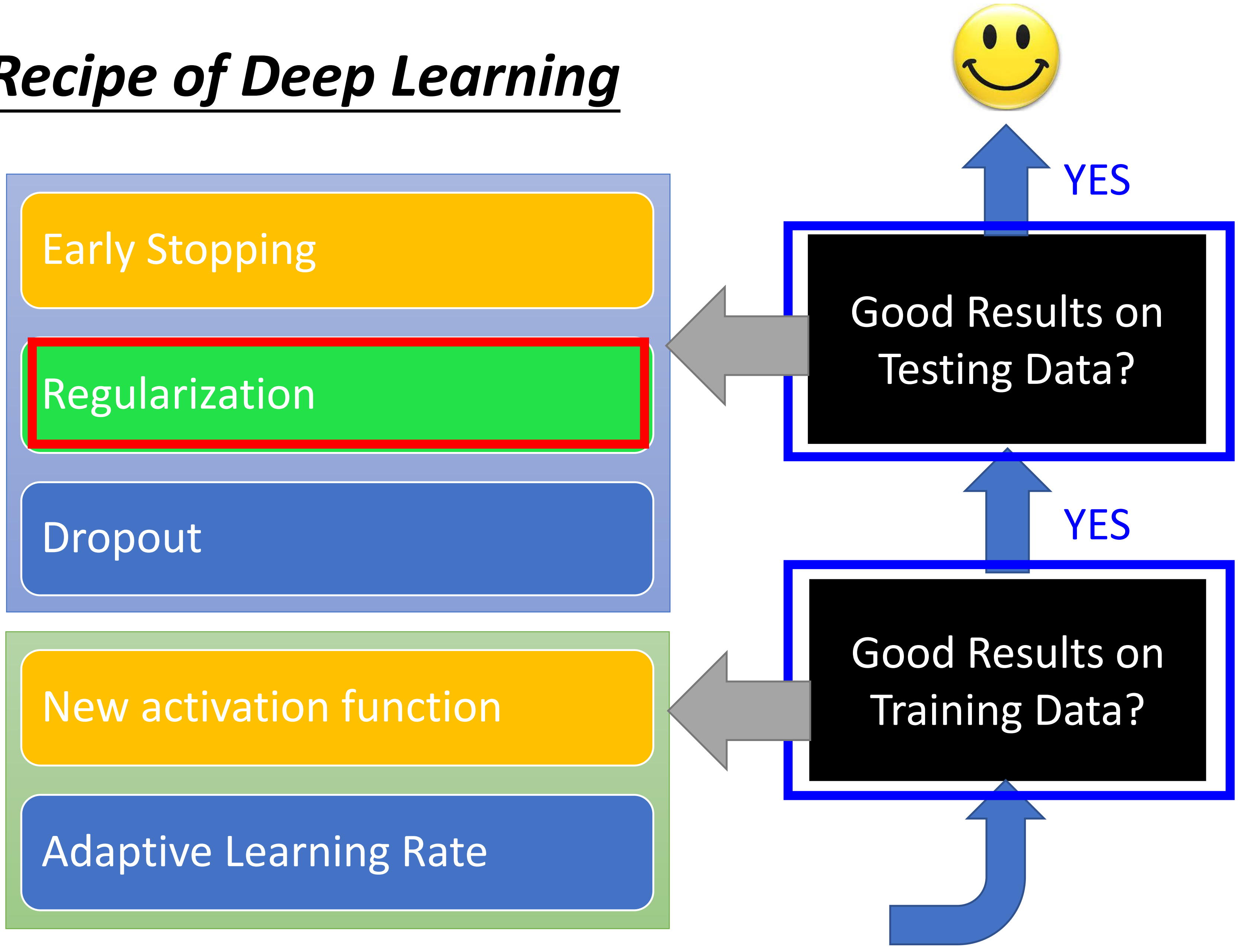


Keras: http://keras.io/getting-started/faq/#how-can-i-interrupt-training-when-the-validation-loss-isnt-decreasing-anymore

Recipe of Deep Learning

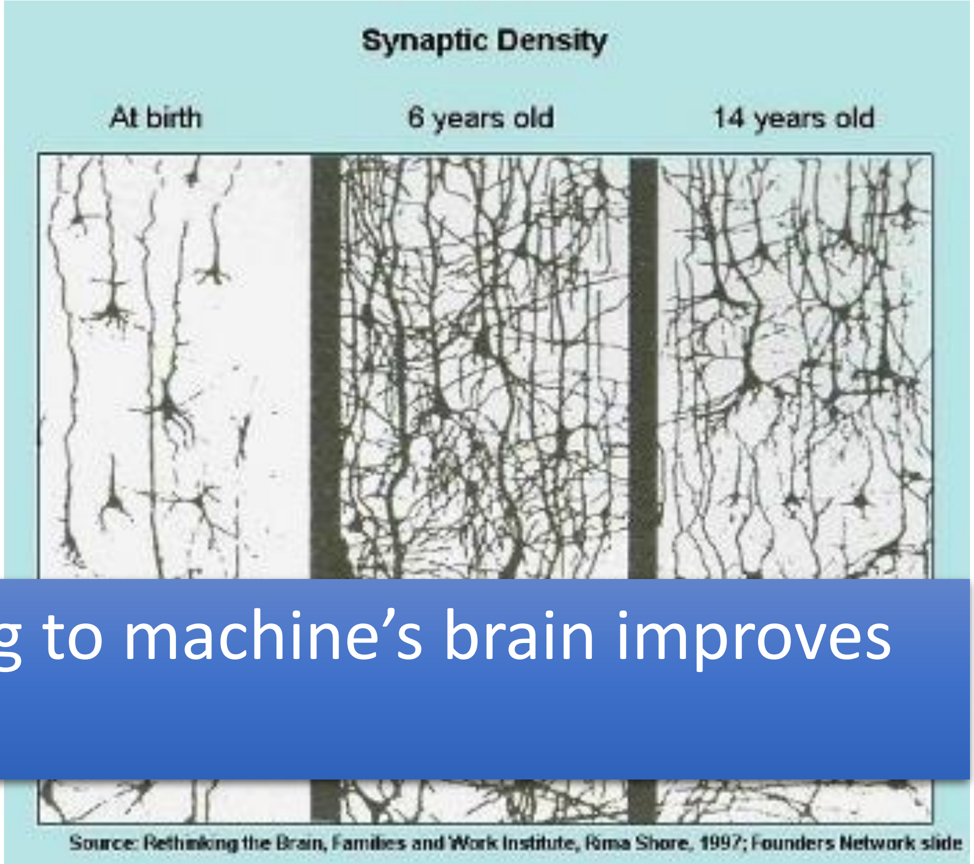


Recipe of Deep Learning



Regularization - Weight Decay

- Our brain prunes out the useless link between neurons.



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

Regularization

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

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

→ Regularization term

希望2-norm值越小越好

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

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

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

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

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

Weight Decay

有 用 的 w 參 數 會 跟 gradient取得平衡

L1 vs L2

Regularization

L1 regularization:

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

- New loss function to be minimized

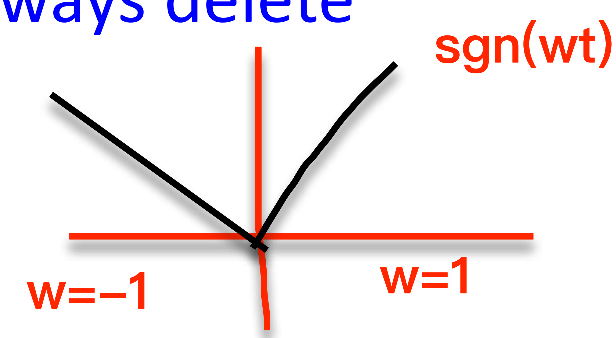
$$L'(\theta) = L(\theta) + \lambda \frac{1}{2}\|\theta\|_1 \qquad \frac{\partial L'}{\partial w} = \frac{\partial L}{\partial w} + \lambda \operatorname{sgn}(w)$$

Update:

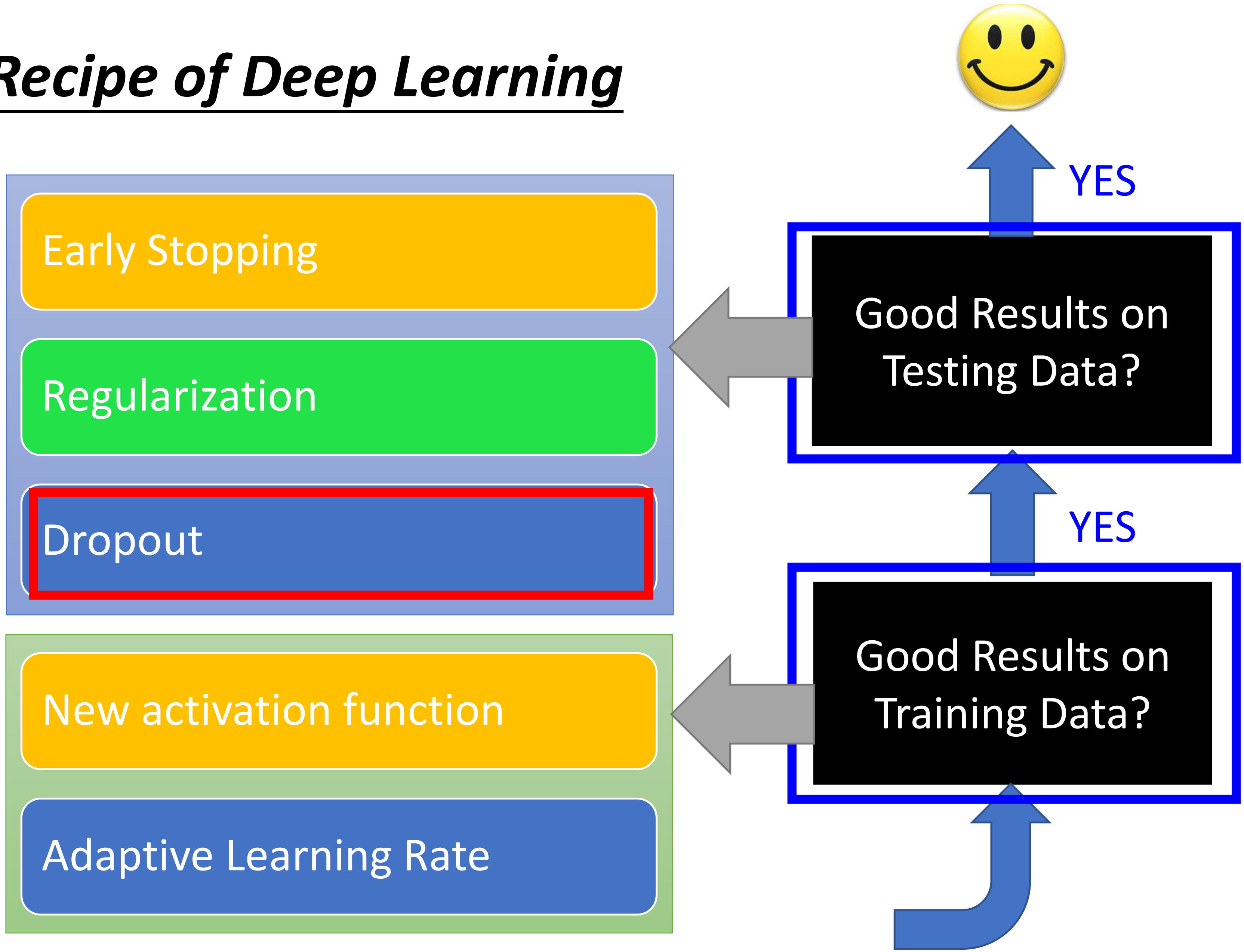
$$w^{t+1} \rightarrow w^t - \eta \frac{\partial L'}{\partial w} = w^t - \eta\left(\frac{\partial L}{\partial w} + \lambda \operatorname{sgn}(w^t)\right)$$

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

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

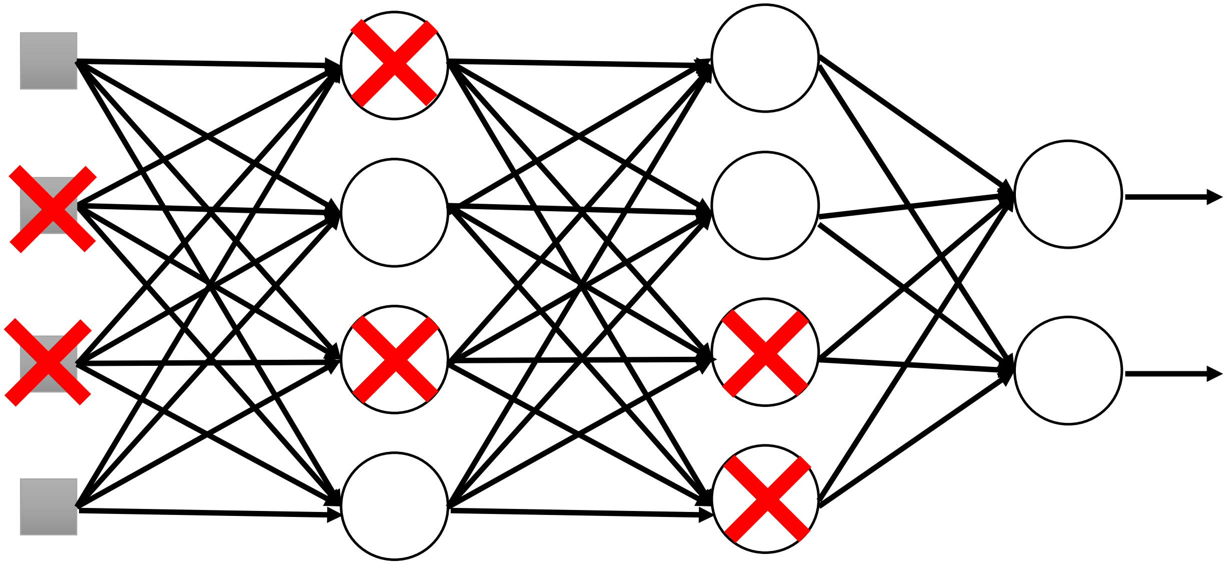


Recipe of Deep Learning



Dropout

Training:

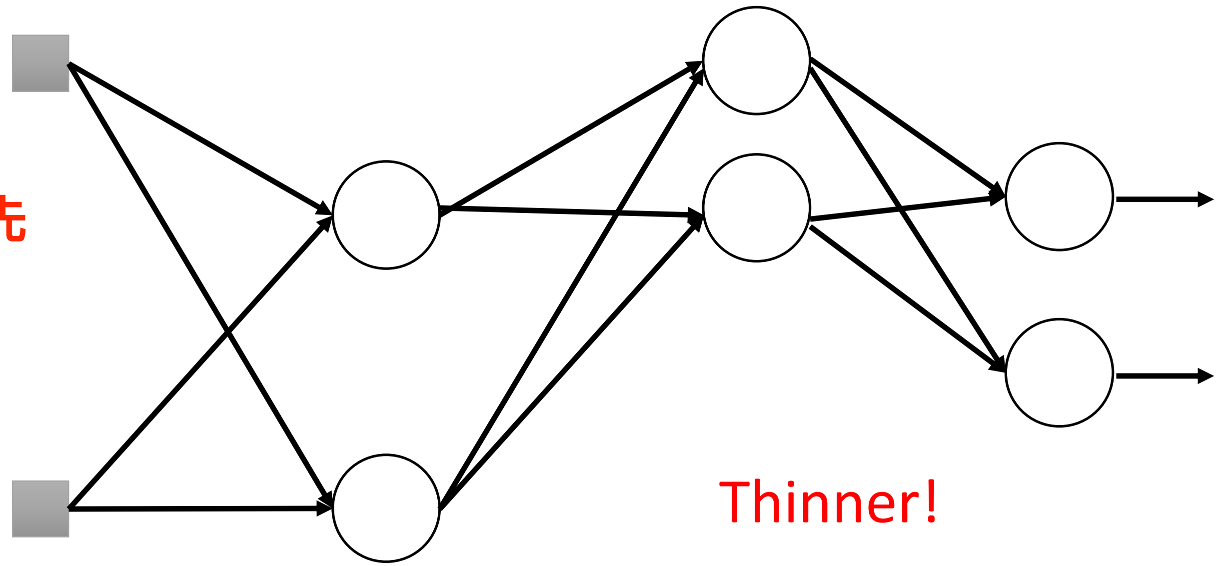


- **Each time before updating the parameters**
  - Each neuron has p% to dropout

Dropout

Training:

每次train之前會先 sample部份node



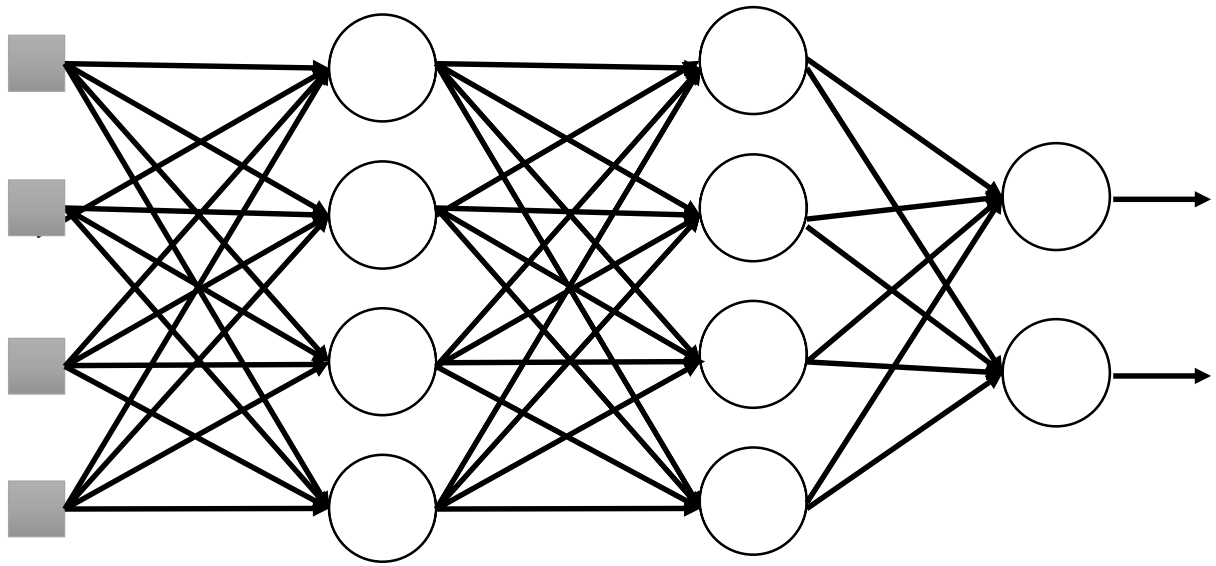
- **Each time before updating the parameters**
  - Each neuron has p% to dropout
    - **The structure of the network is changed.**
  - Using the new network for training

For each mini-batch, we resample the dropout neurons

Dropout

在dropout時train中的weight與test中的weight是不同的

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

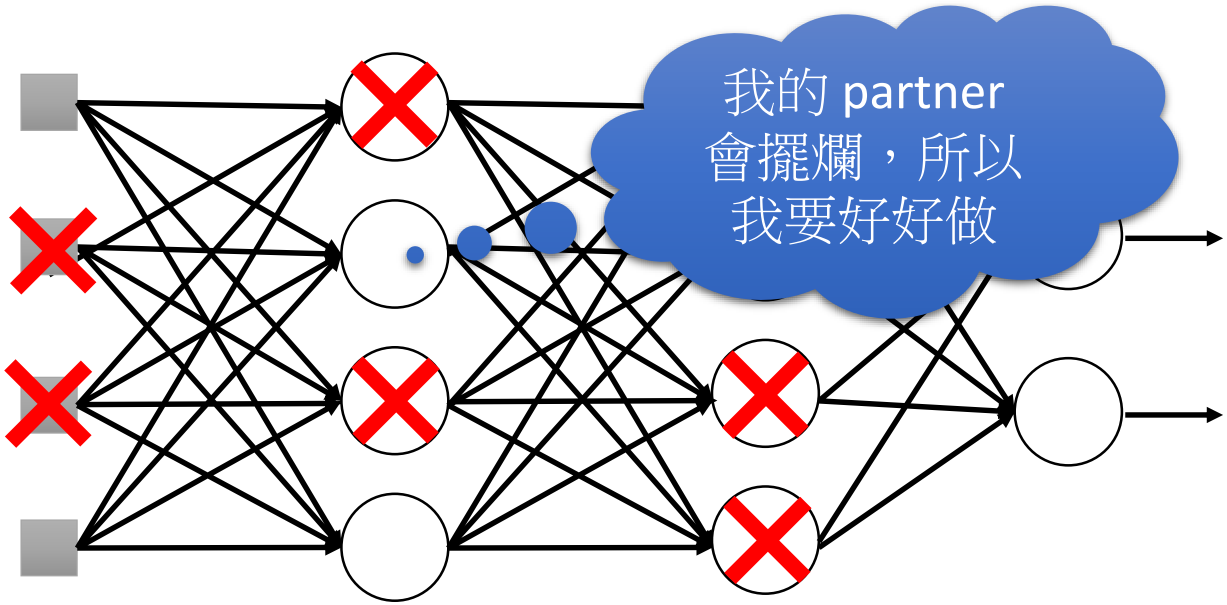
Testing

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.

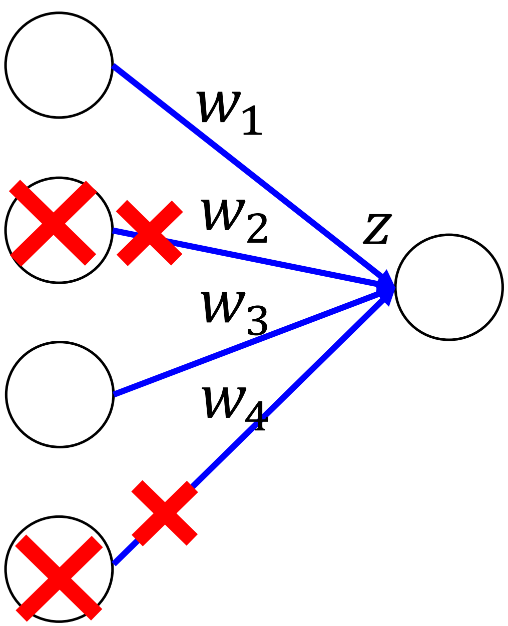
Dropout - Intuitive Reason

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

雖然non-linear較不準確但是仍然work

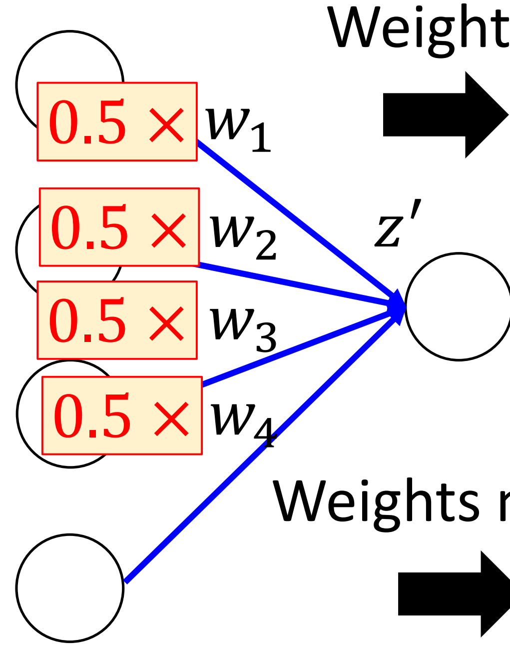
Training of Dropout

Assume dropout rate is 50%



Testing of Dropout

No dropout



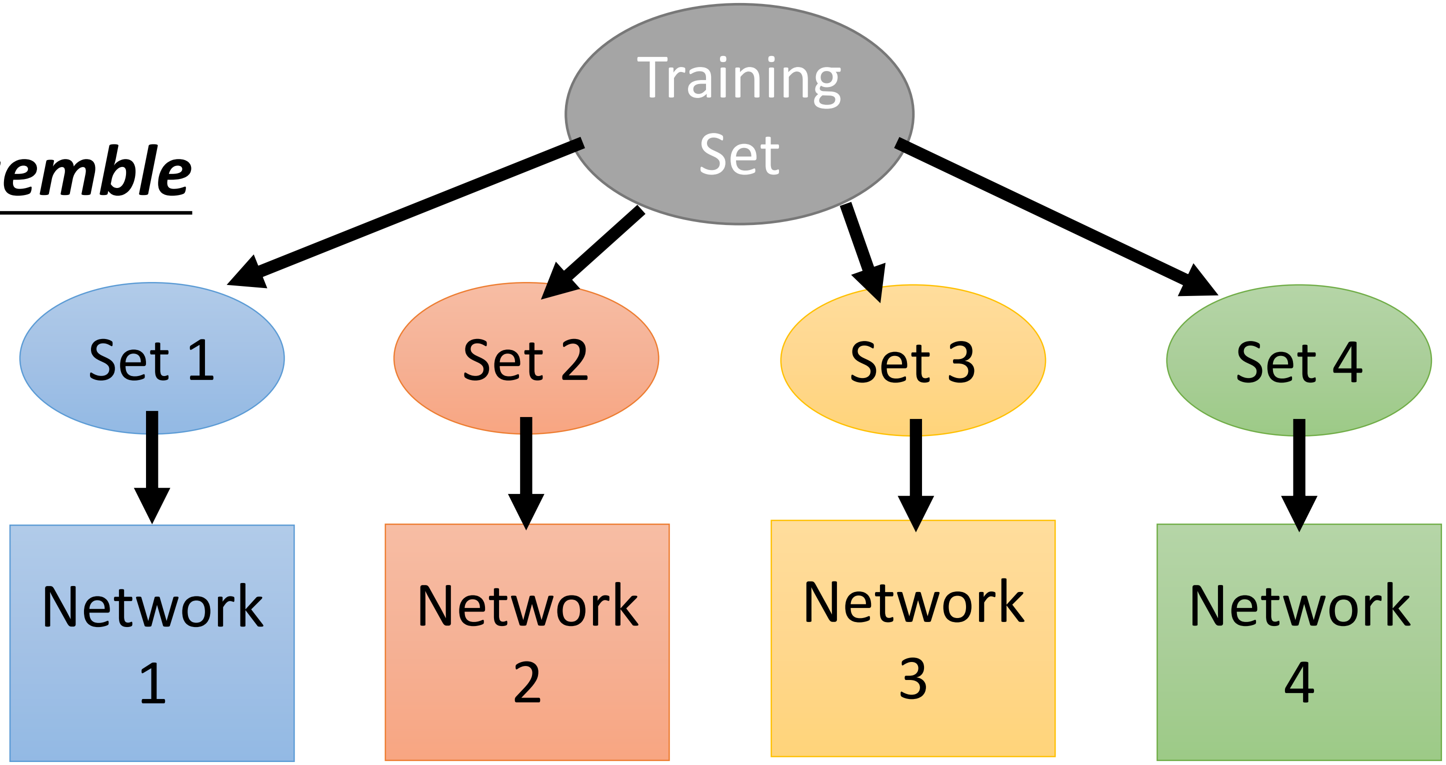
Weights from training $\Rightarrow$ $z' \approx 2z$

使training與testing的z一致

Weights multiply 1-p% $\Rightarrow$ $z' \approx z$

Dropout is a kind of ensemble.

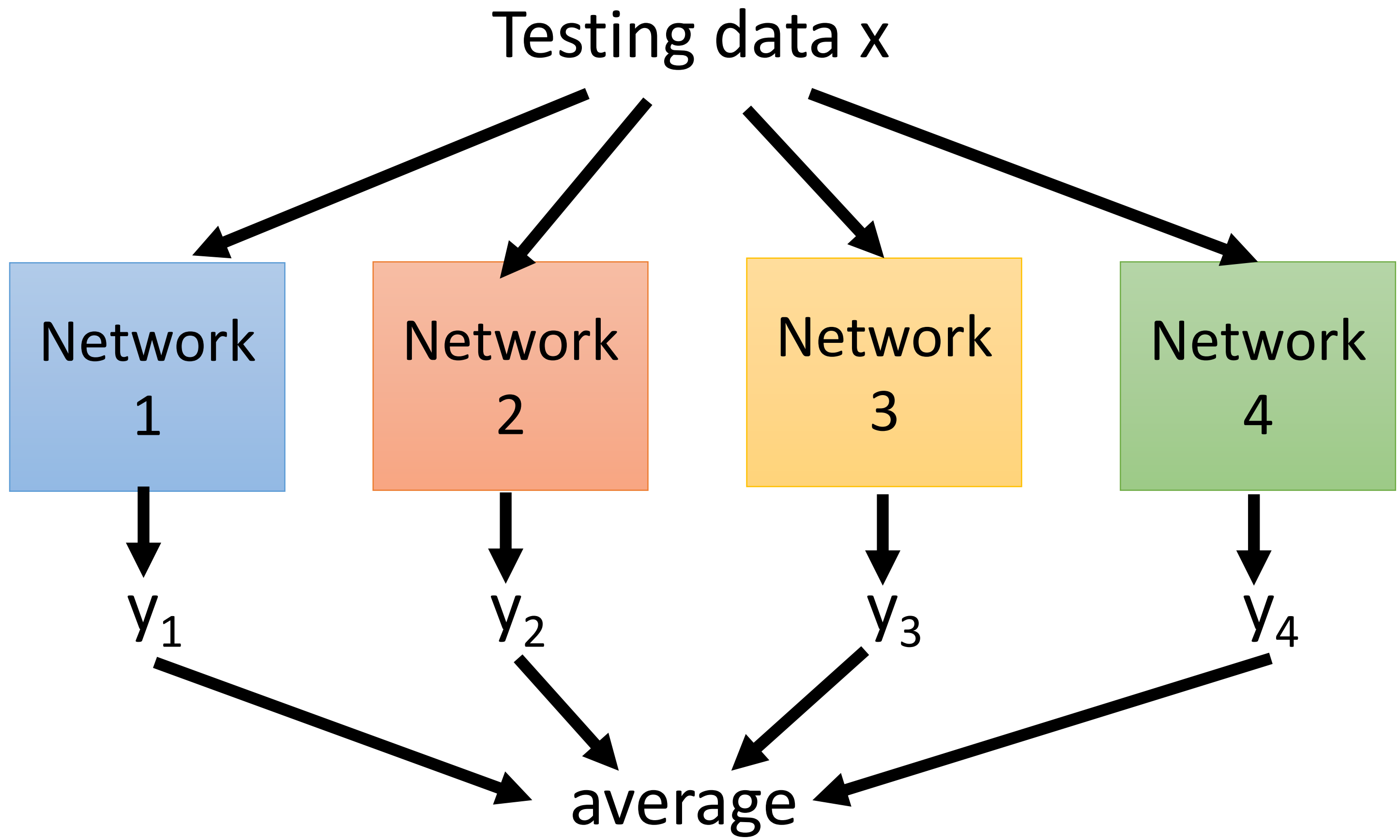
Ensemble



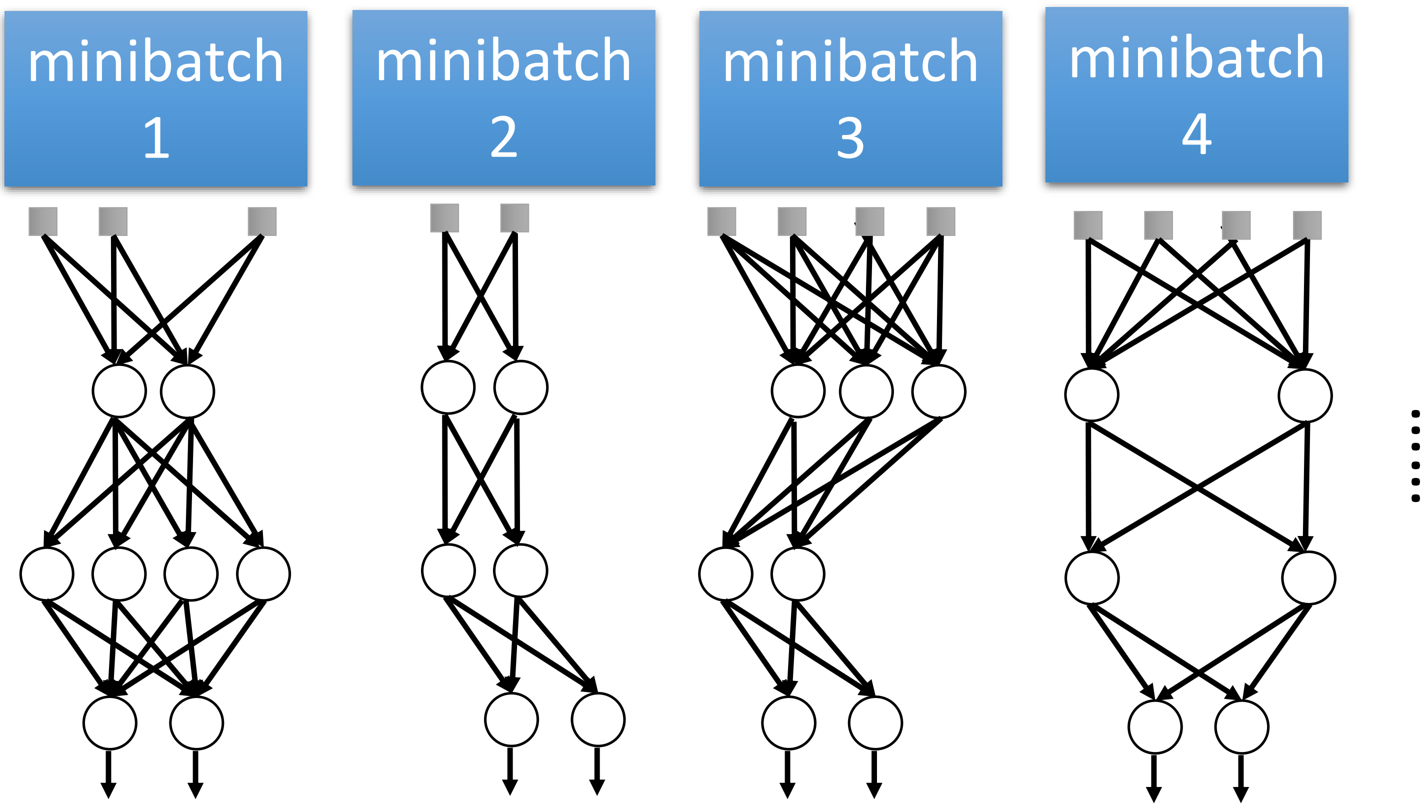
Train a bunch of networks with different structures

Dropout is a kind of ensemble.

Ensemble



Dropout is a kind of ensemble.



Training of Dropout

M neurons

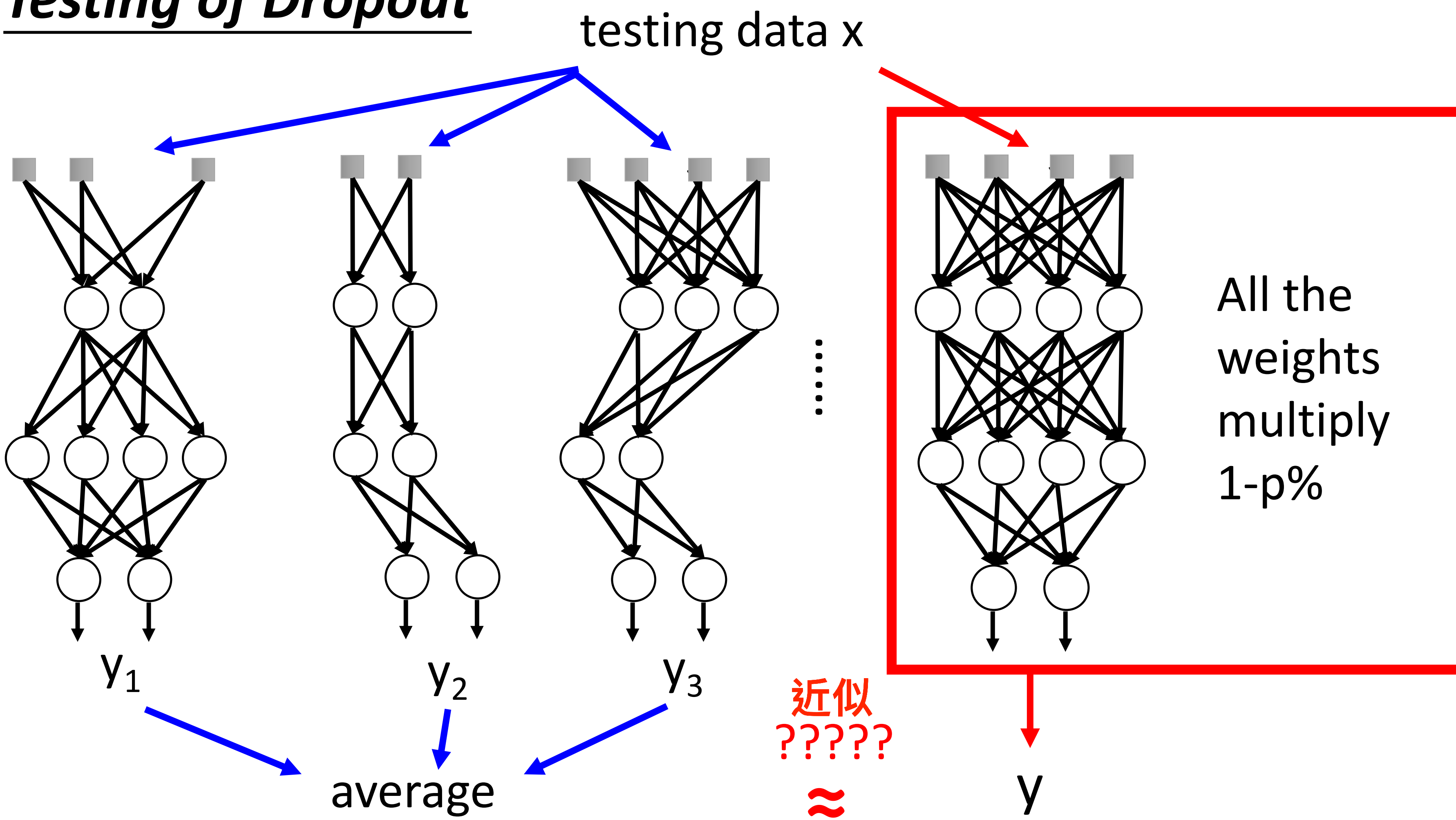
2^M possible networks

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

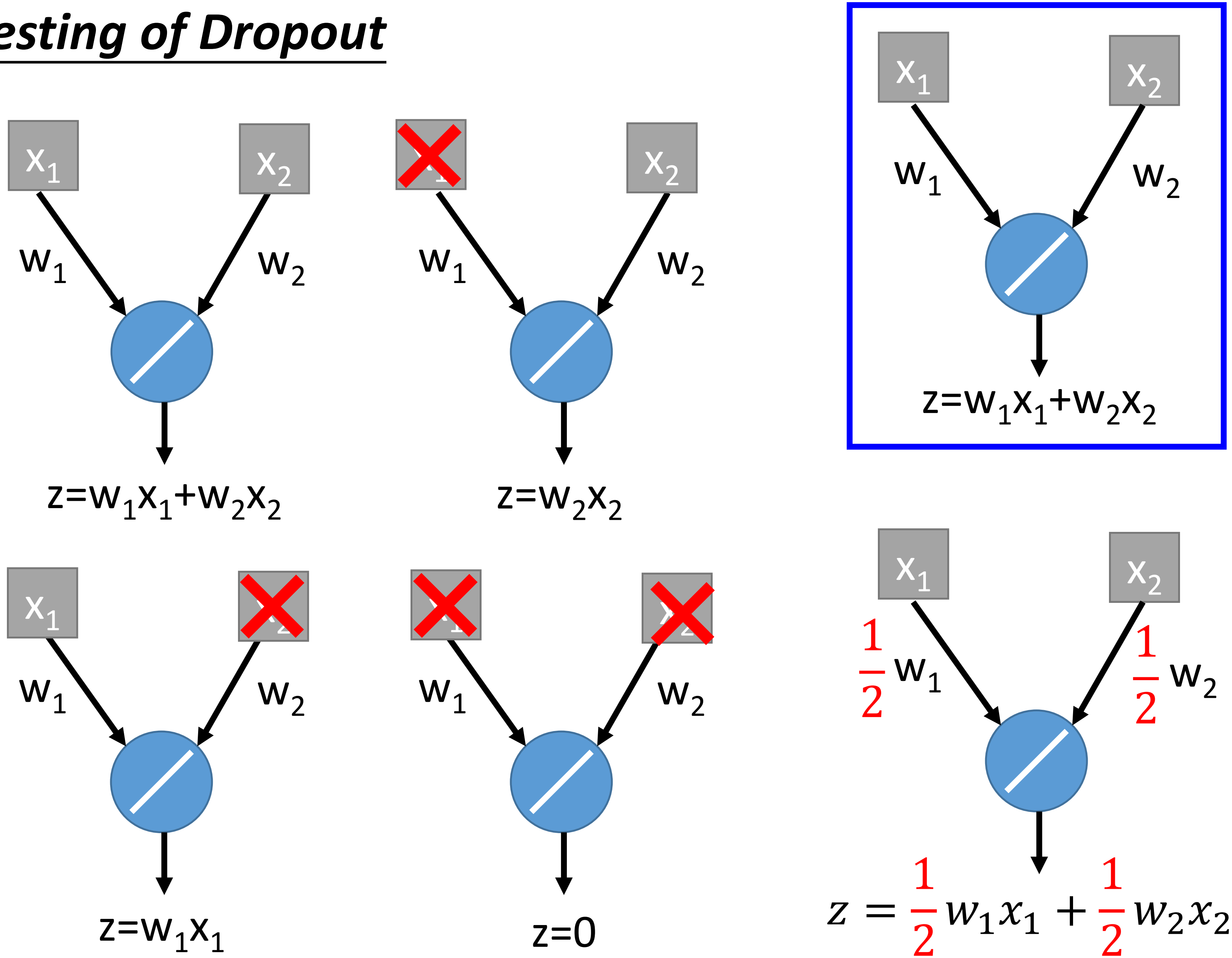
Dropout is a kind of ensemble.

標準作法，M種Network，2的m次方個可能取平均

Testing of Dropout

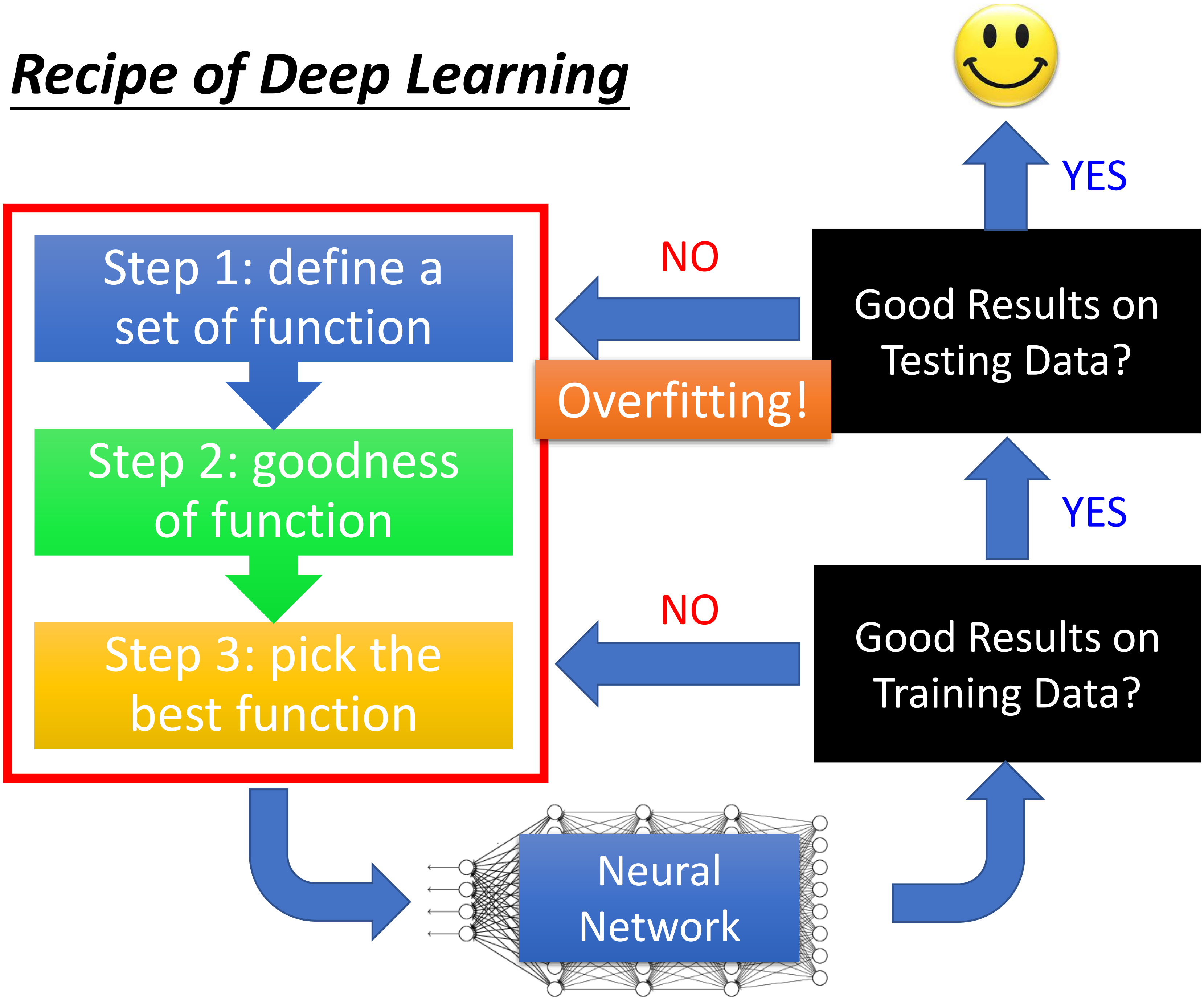


Testing of Dropout

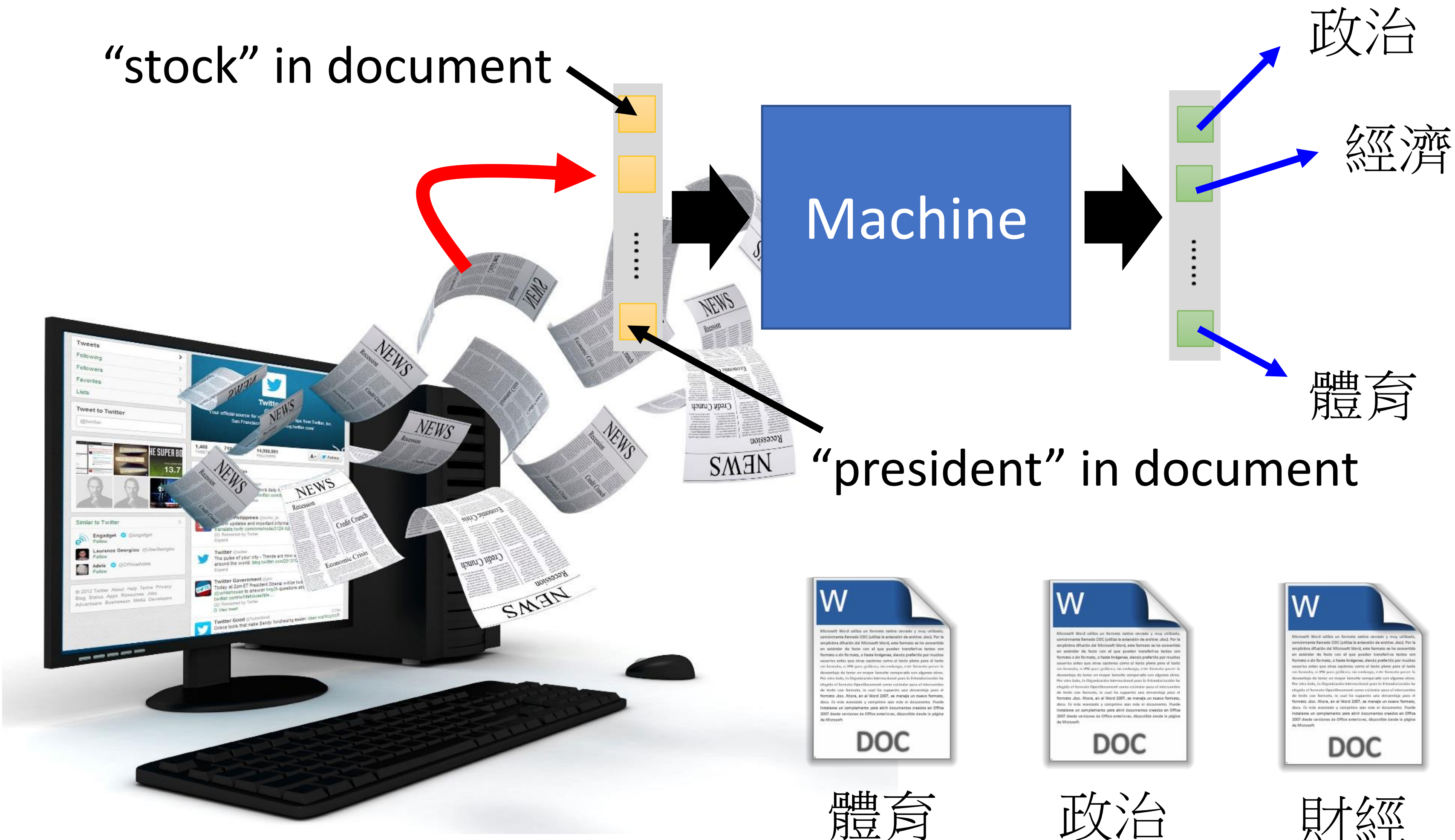


只有在linear的activation function中能夠準確逼近

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