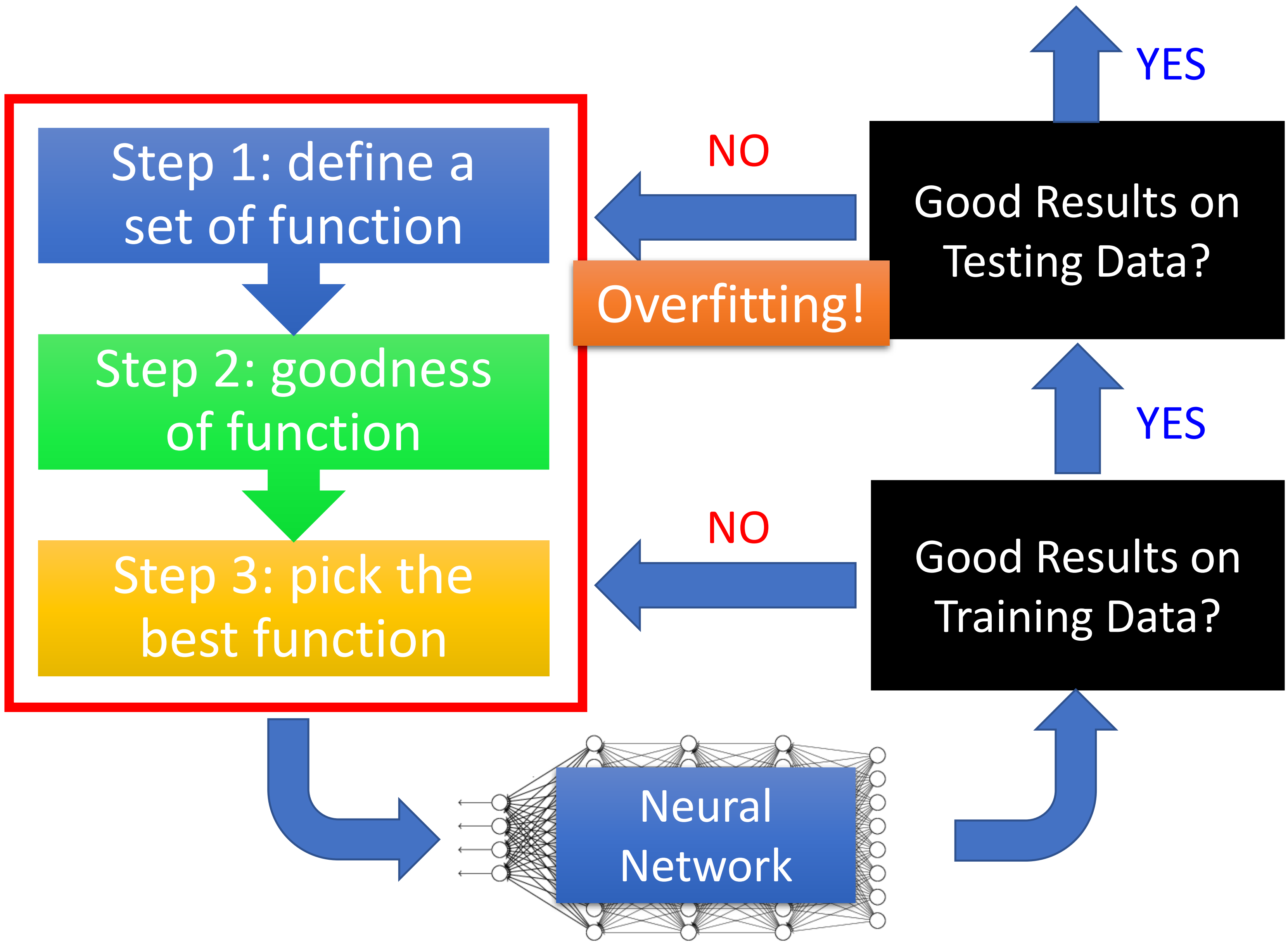
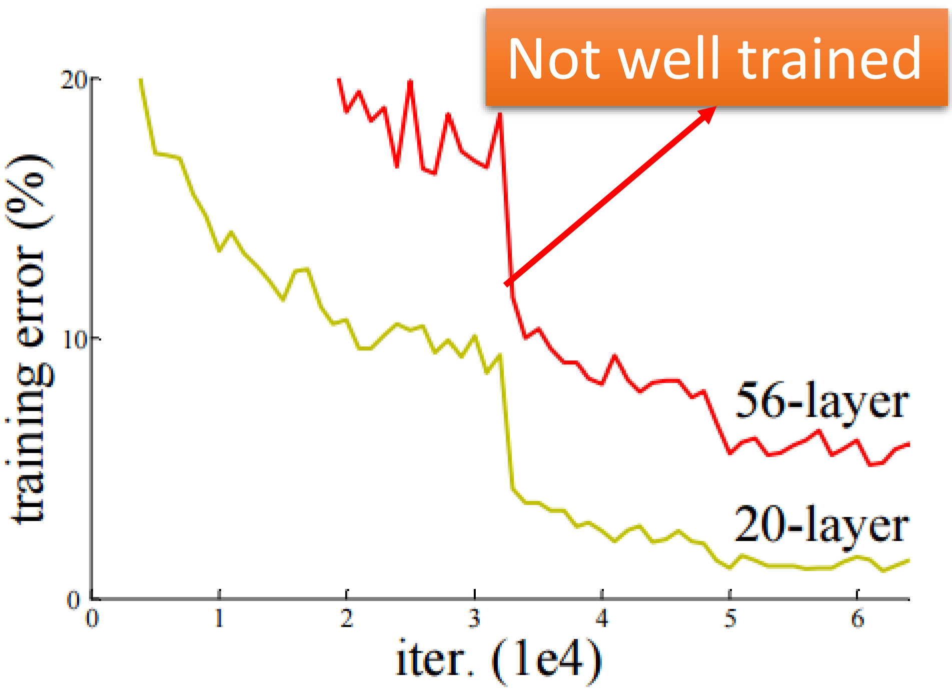
Tips for Deep Learning

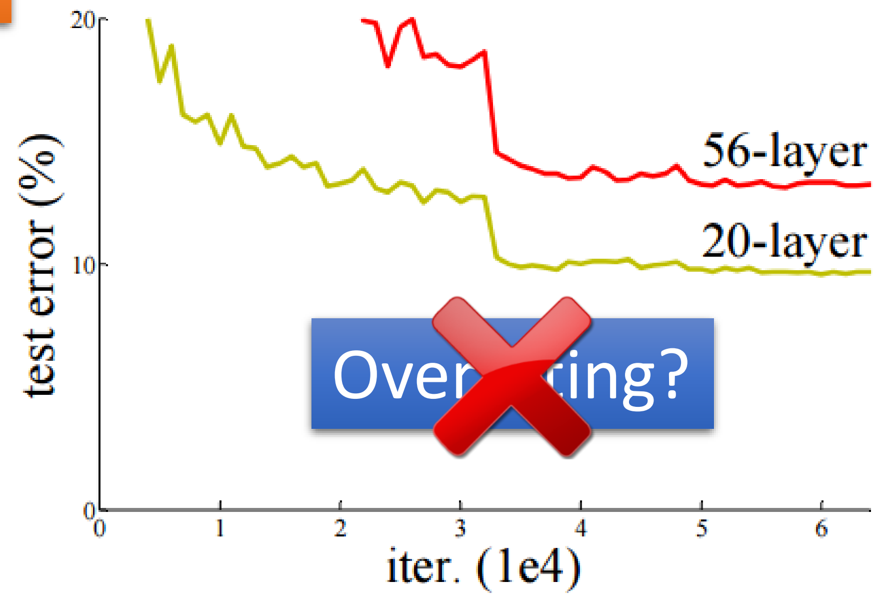
Recipe of Deep Learning



Do not always blame Overfitting



Training Data

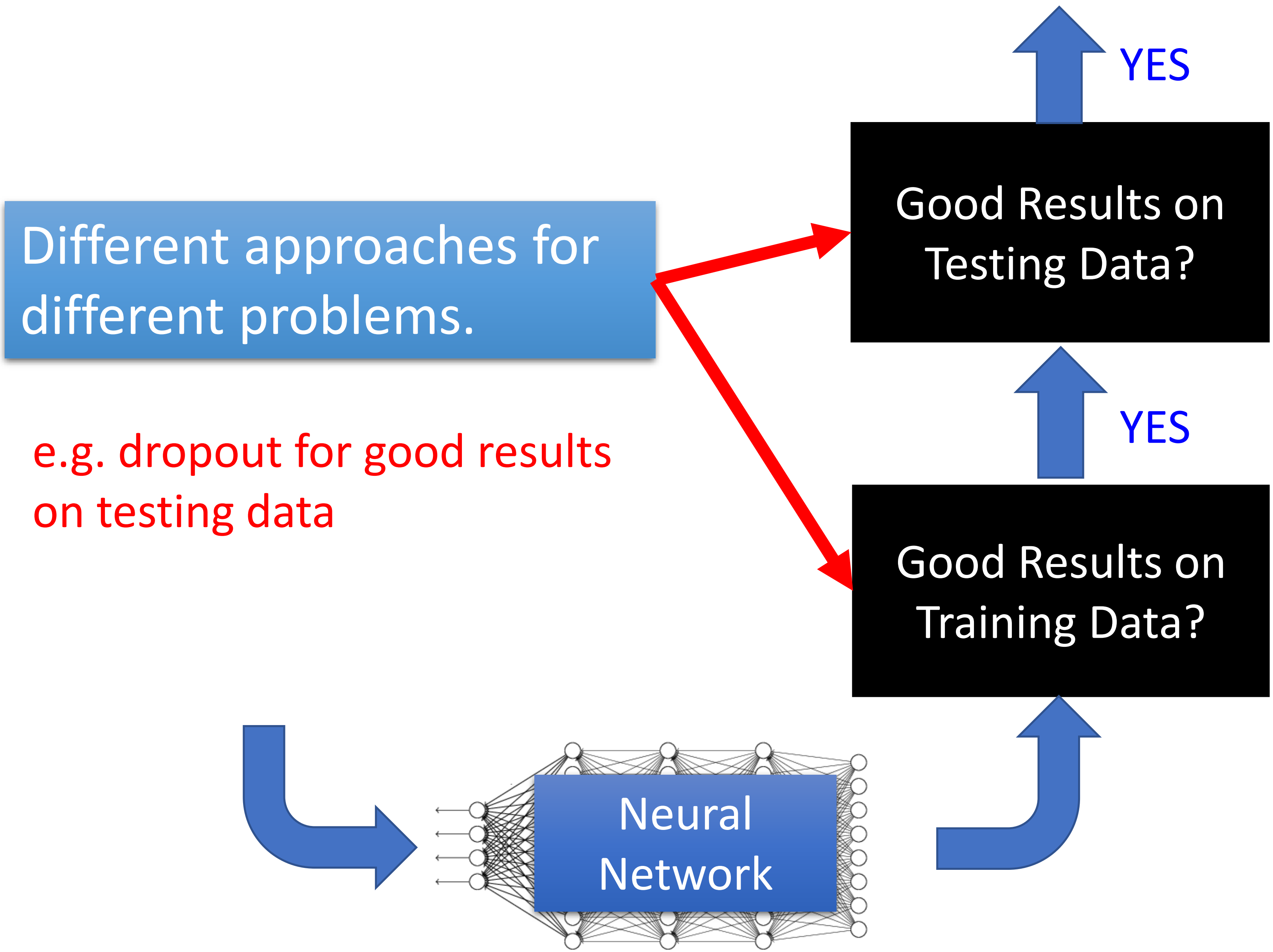


Testing Data

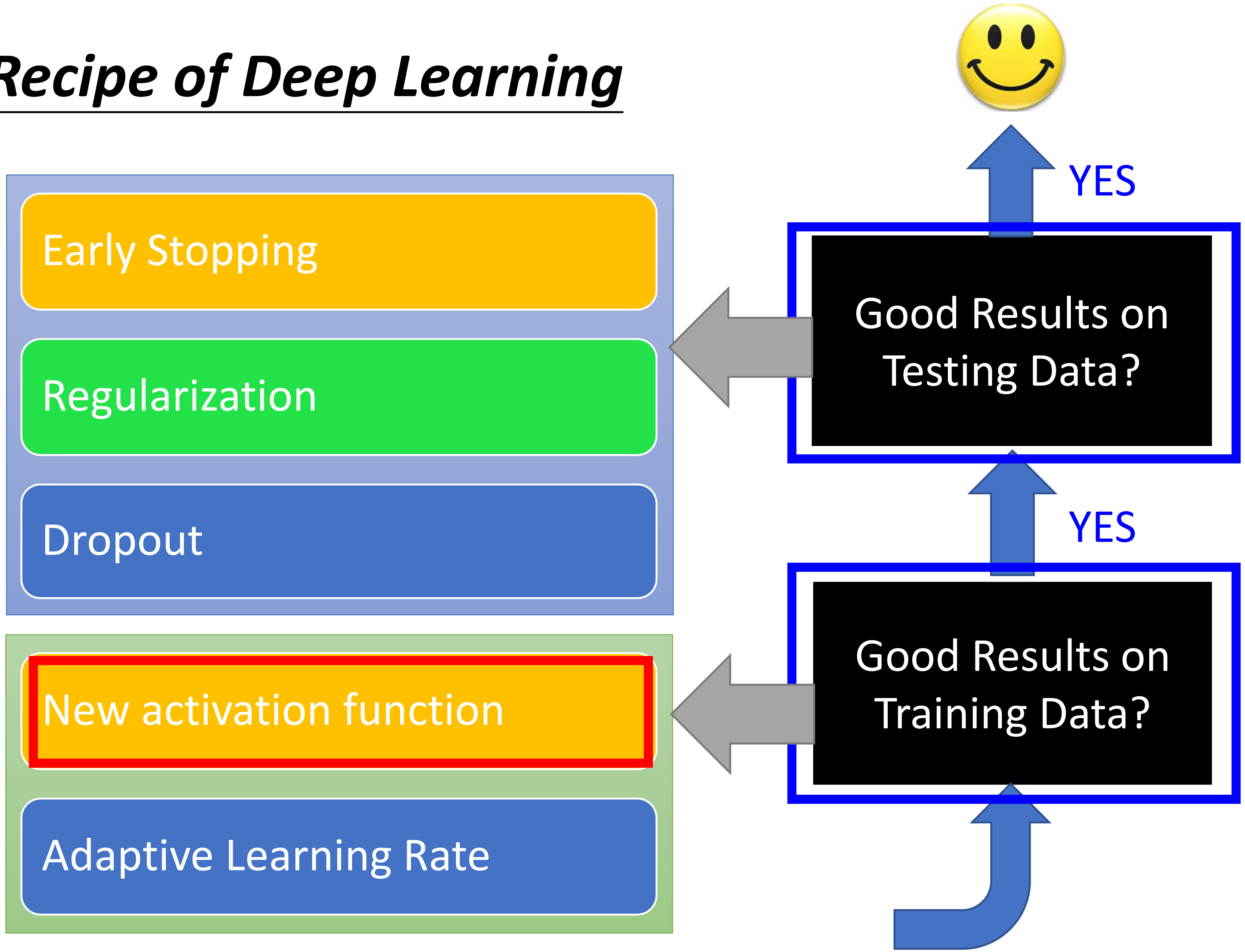
Deep Residual Learning for Image Recognition

http://arxiv.org/abs/1512.03385

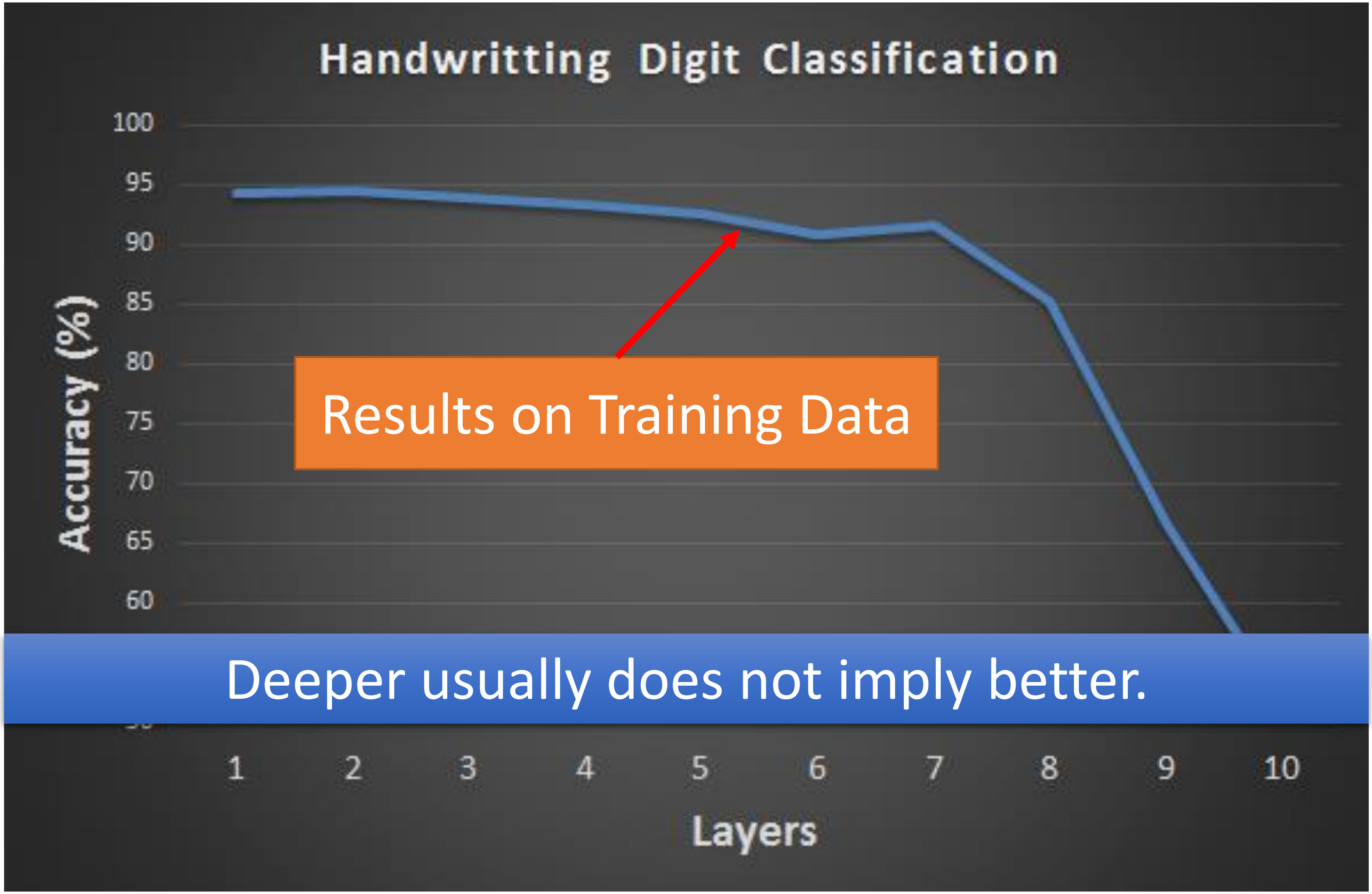
Recipe of Deep Learning



Recipe of Deep Learning

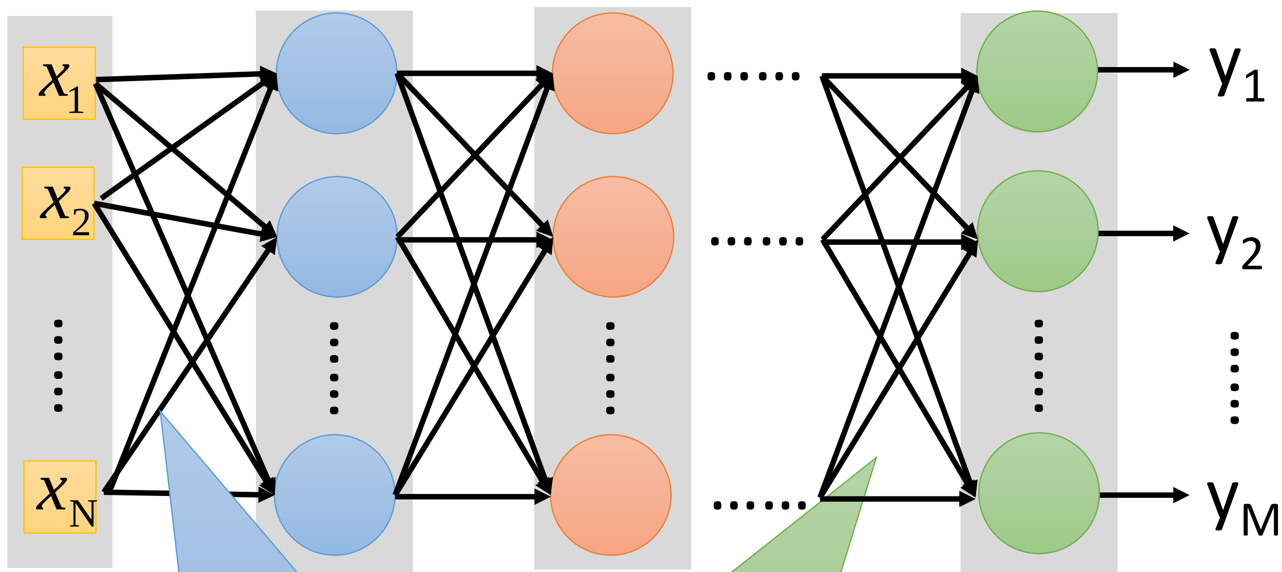


Hard to get the power of Deep …



Deeper usually does not imply better.

Vanishing Gradient Problem



Smaller gradients

Learn very slow

Almost random

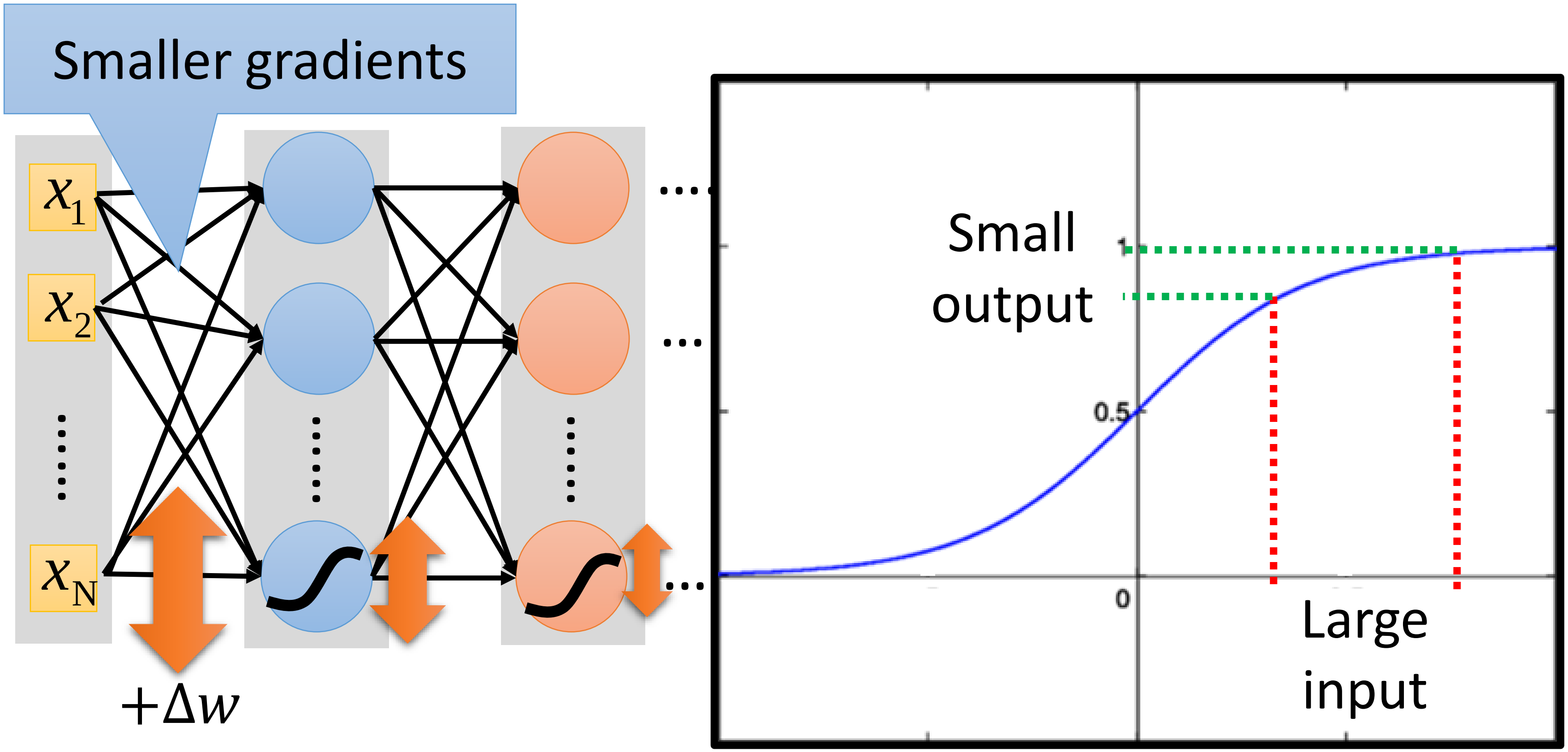
Larger gradients

Learn very fast

Already converge

based on random!?

Vanishing Gradient Problem

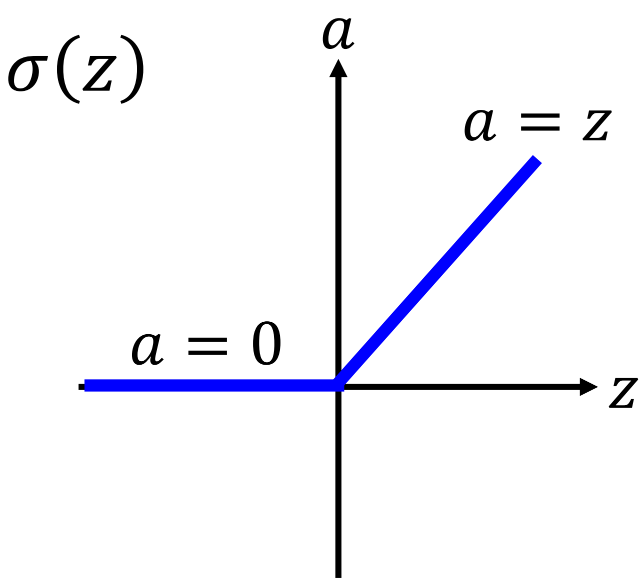


Intuitive way to compute the derivatives …

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

ReLU

- Rectified Linear Unit (ReLU)

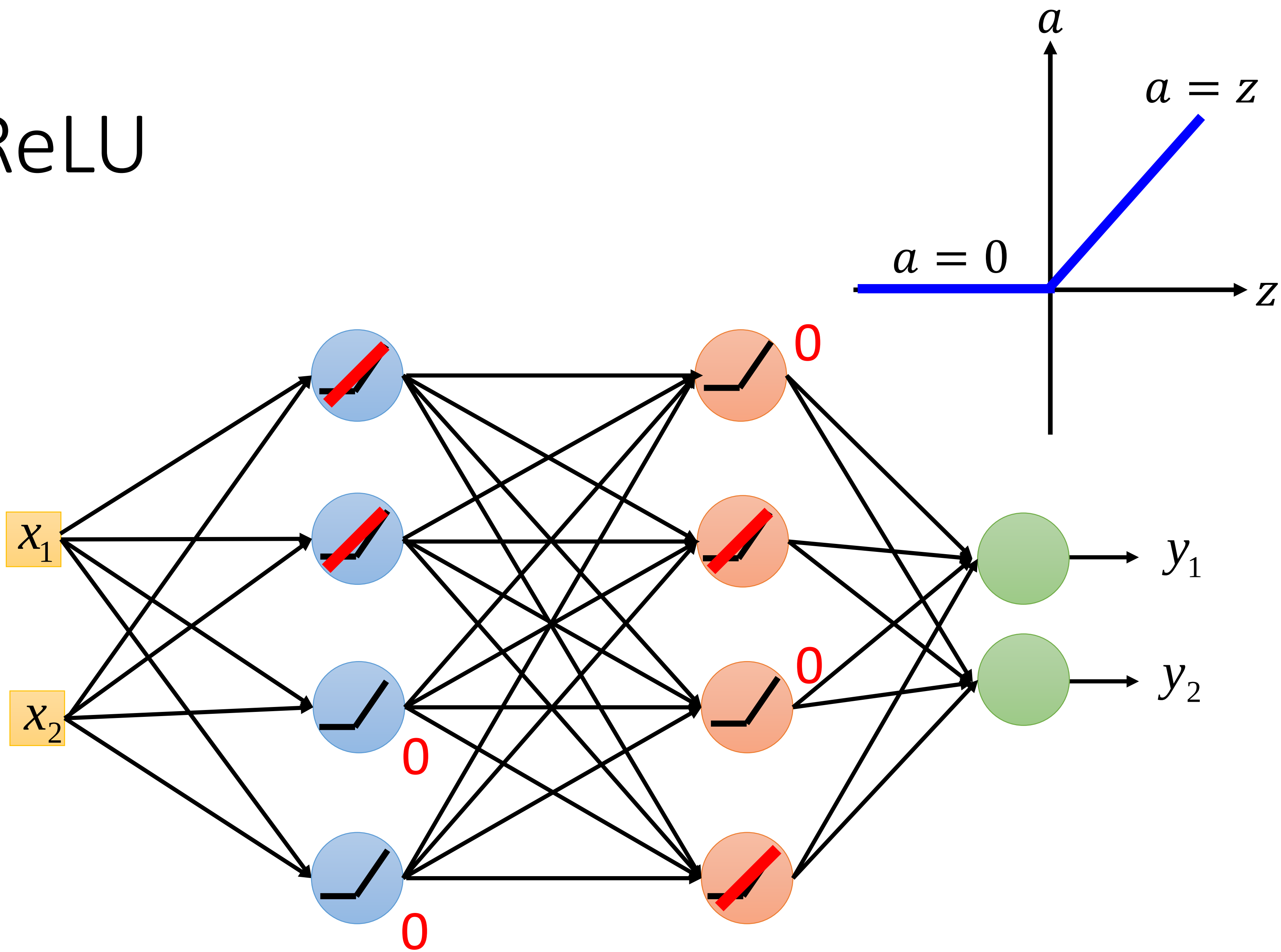


[Xavier Glorot, AISTATS'11]
[Andrew L. Maas, ICML'13]
[Kaiming He, arXiv'15]

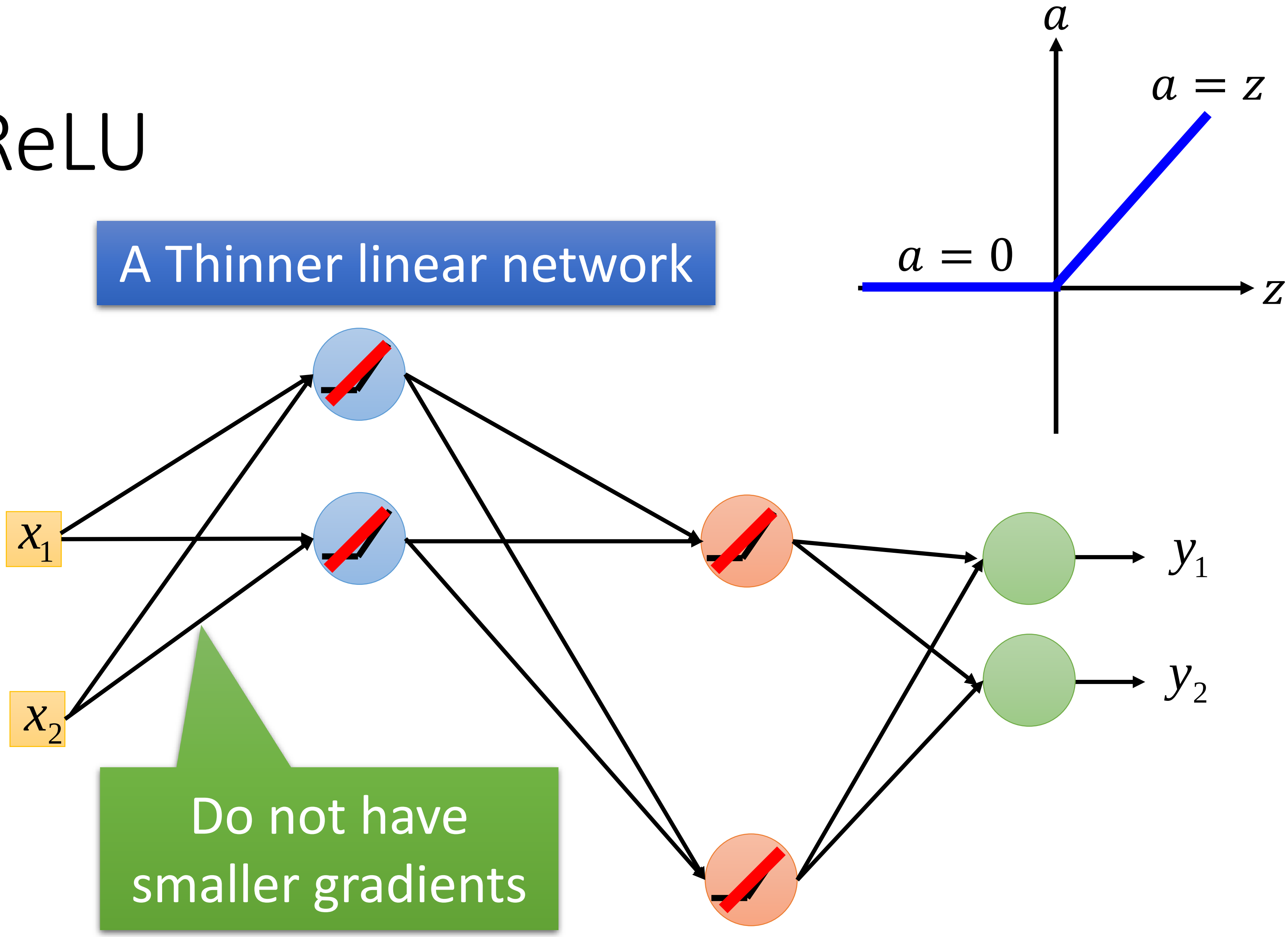
Reason:

- Fast to compute
- Biological reason
  - Only 1~4% neurons active in brain
- Infinite sigmoid with different biases
  - $\int_{-\infty}^{0} \sigma(z+\xi)d\xi = \log(1+e^{z}) \approx ReLU(z)$
- Vanishing gradient problem

ReLU

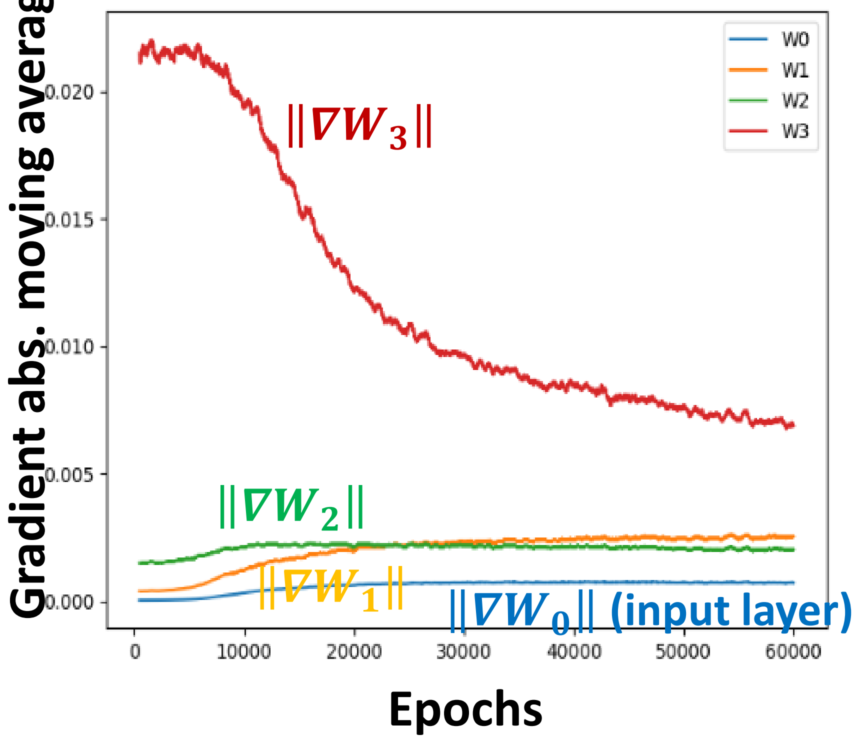


ReLU

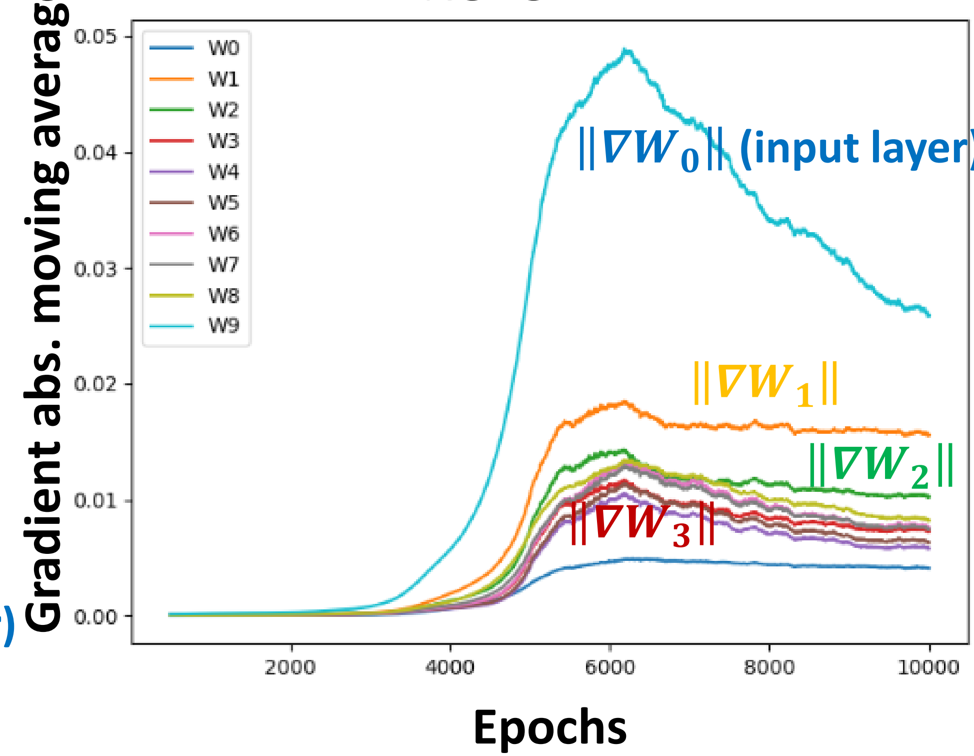

A Thinner linear network
a
$a = z$
$a = 0$
z
x_1
x_2
y_1
y_2
Do not have smaller gradients

Activation Function Comparison

Sigmoid



ReLU



W_n: Weights for neurons in the n'th layer

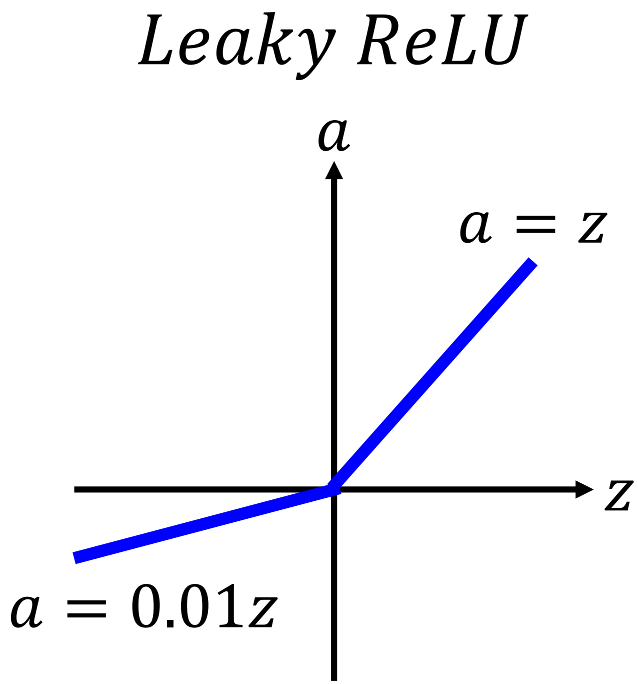
MNIST dataset

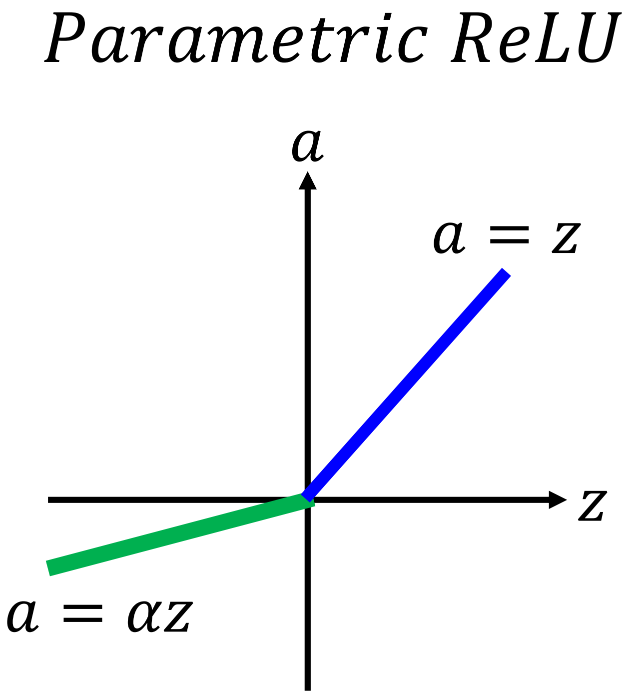
4 layers feedforward NN, 100 nodes for each hidden layer

SGD with learning rate 0.01, batch size 32

Courtesy of 李維道同學

ReLU - variant



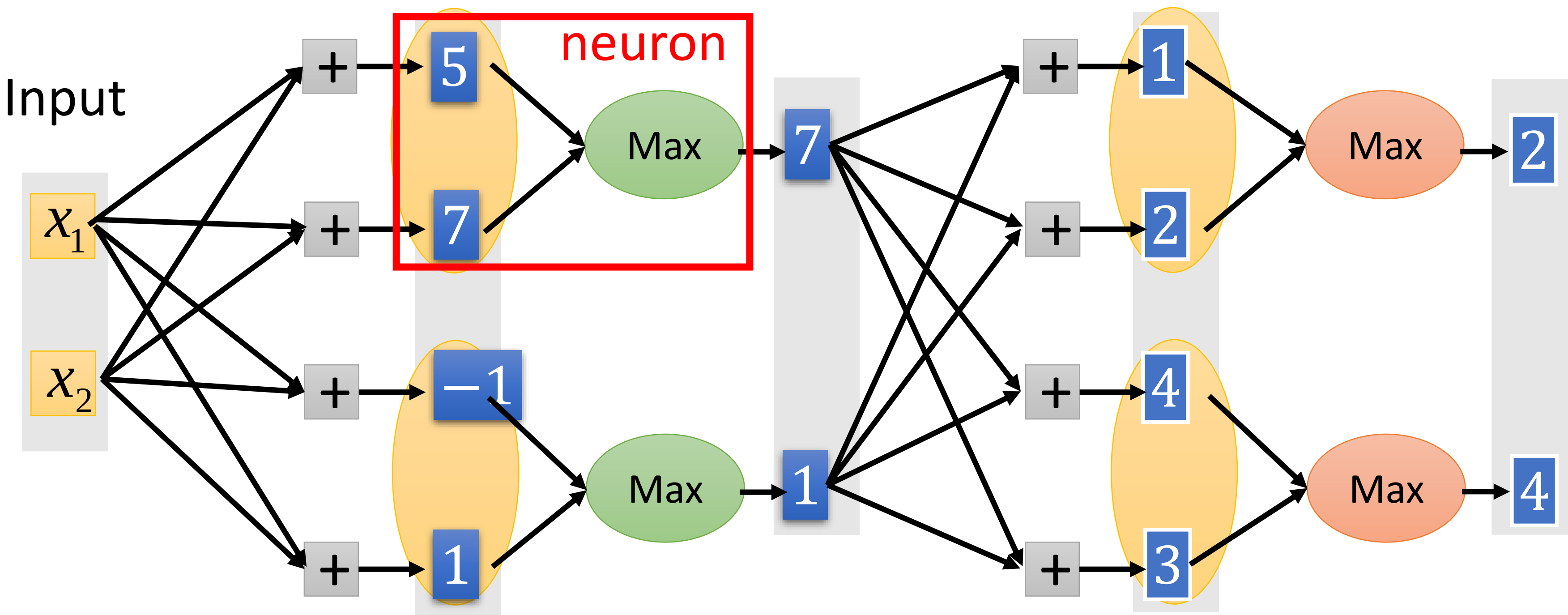


α also learned by gradient descent

Maxout

ReLU is a special cases of Maxout

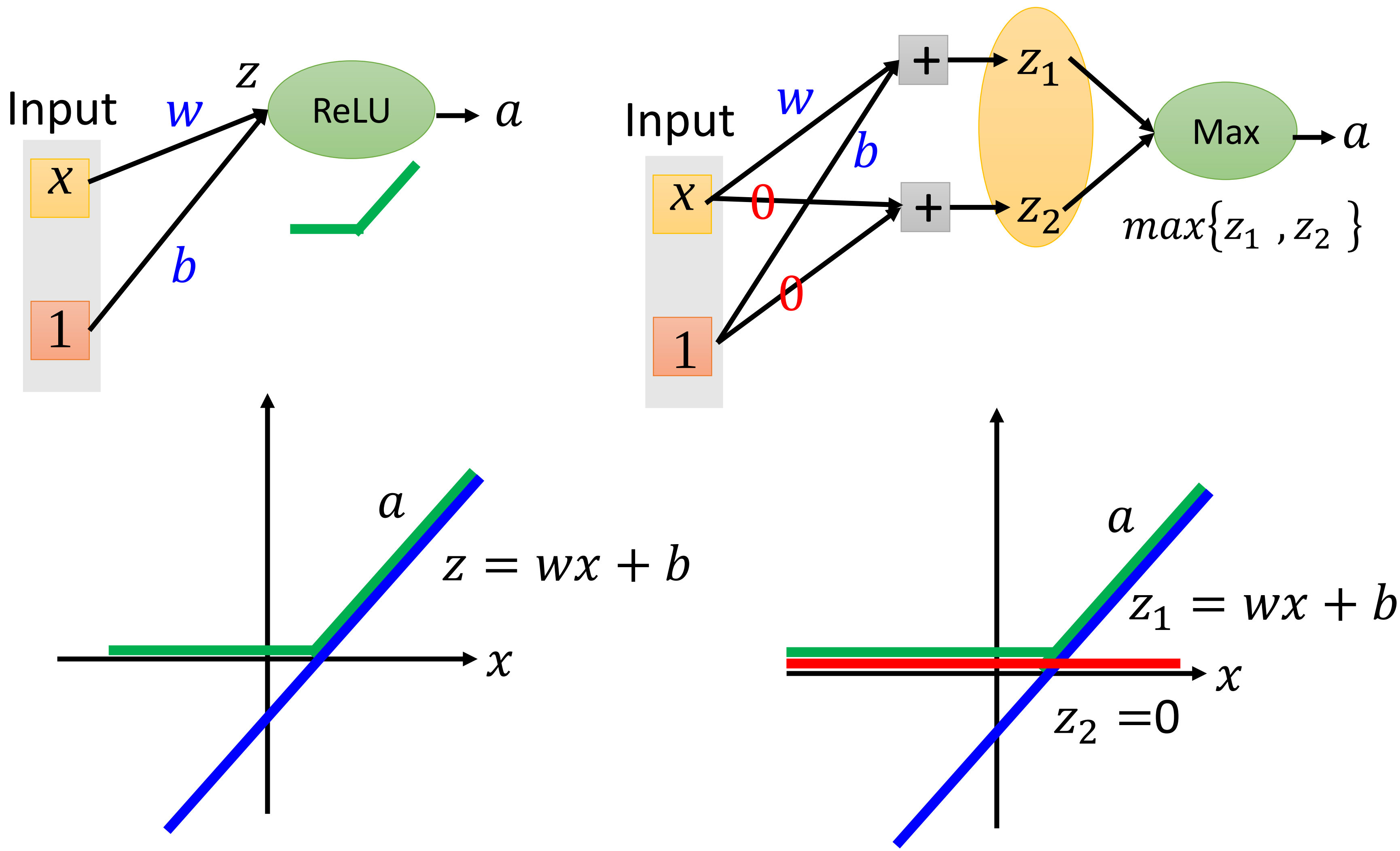
- Learnable activation function [Ian J. Goodfellow, ICML'13]



You can have more than 2 elements in a group.

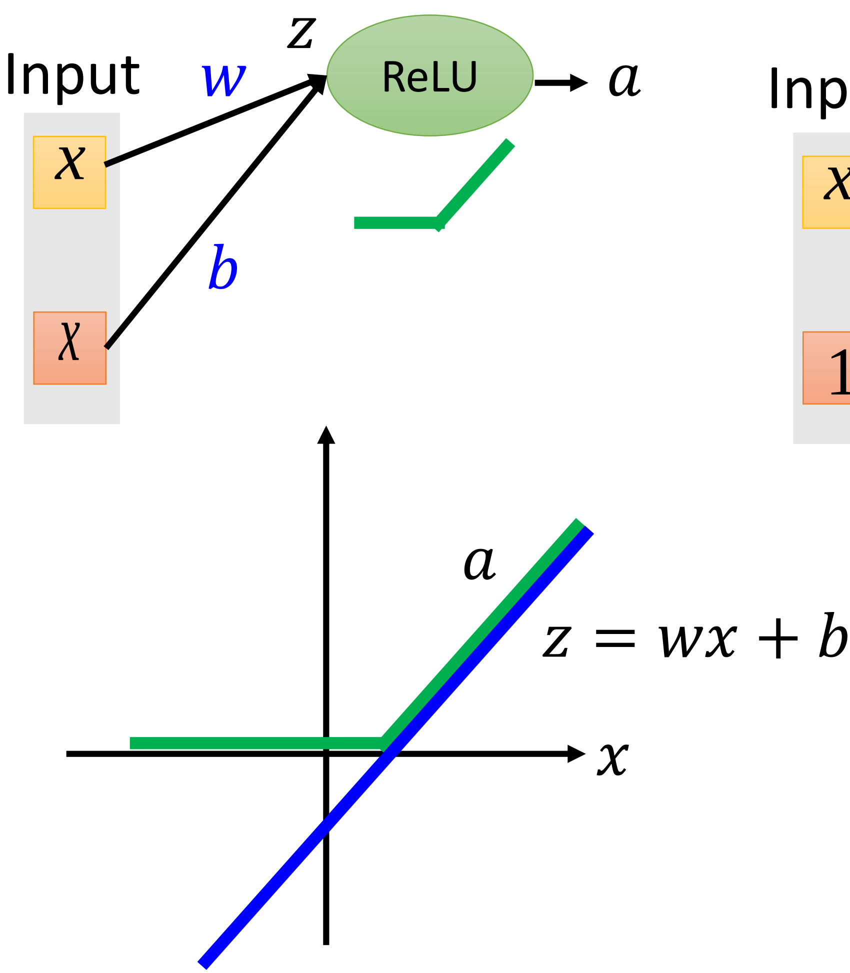
Maxout

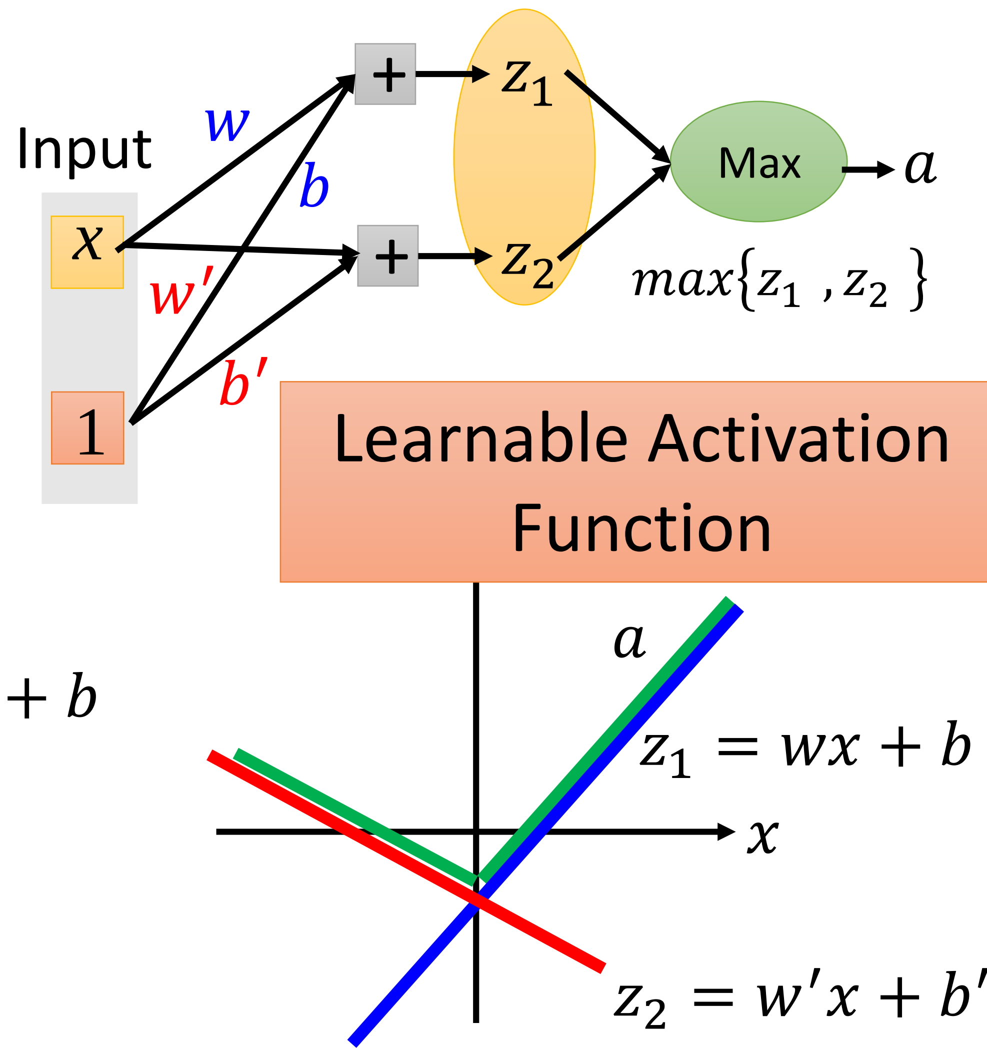
ReLU is a special cases of Maxout



Maxout

More than ReLU



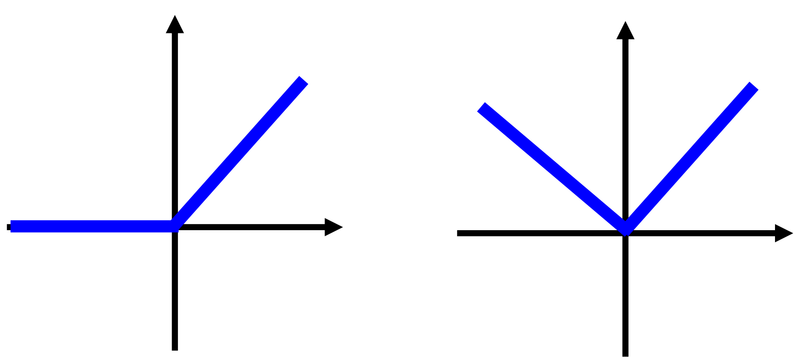


Learnable Activation Function

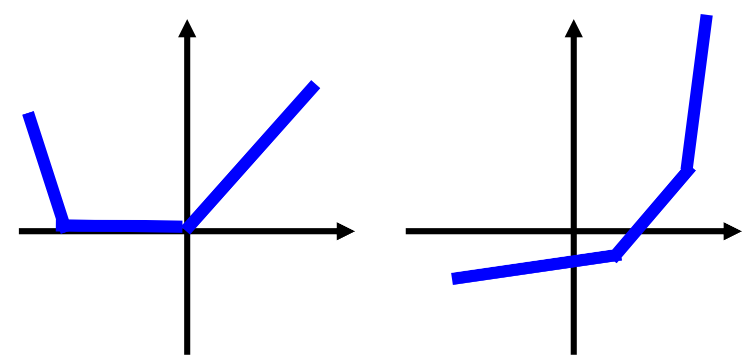
Maxout

- 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

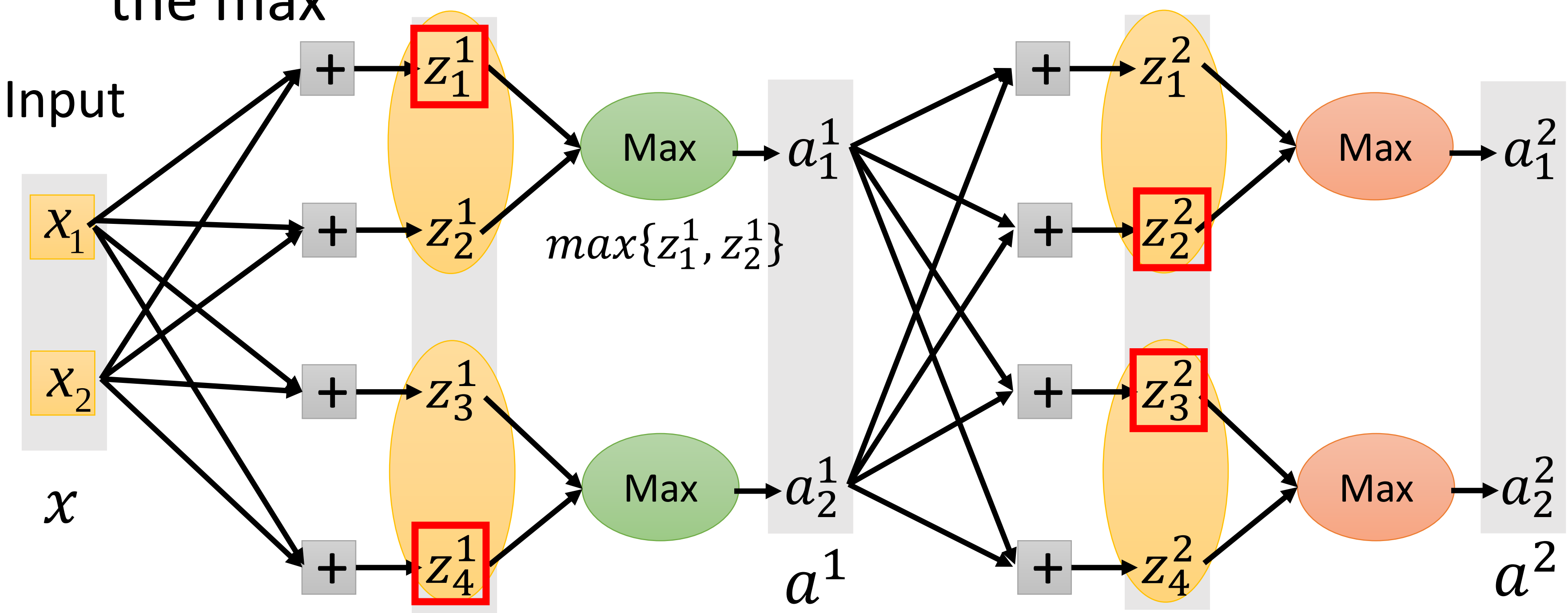


3 elements in a group



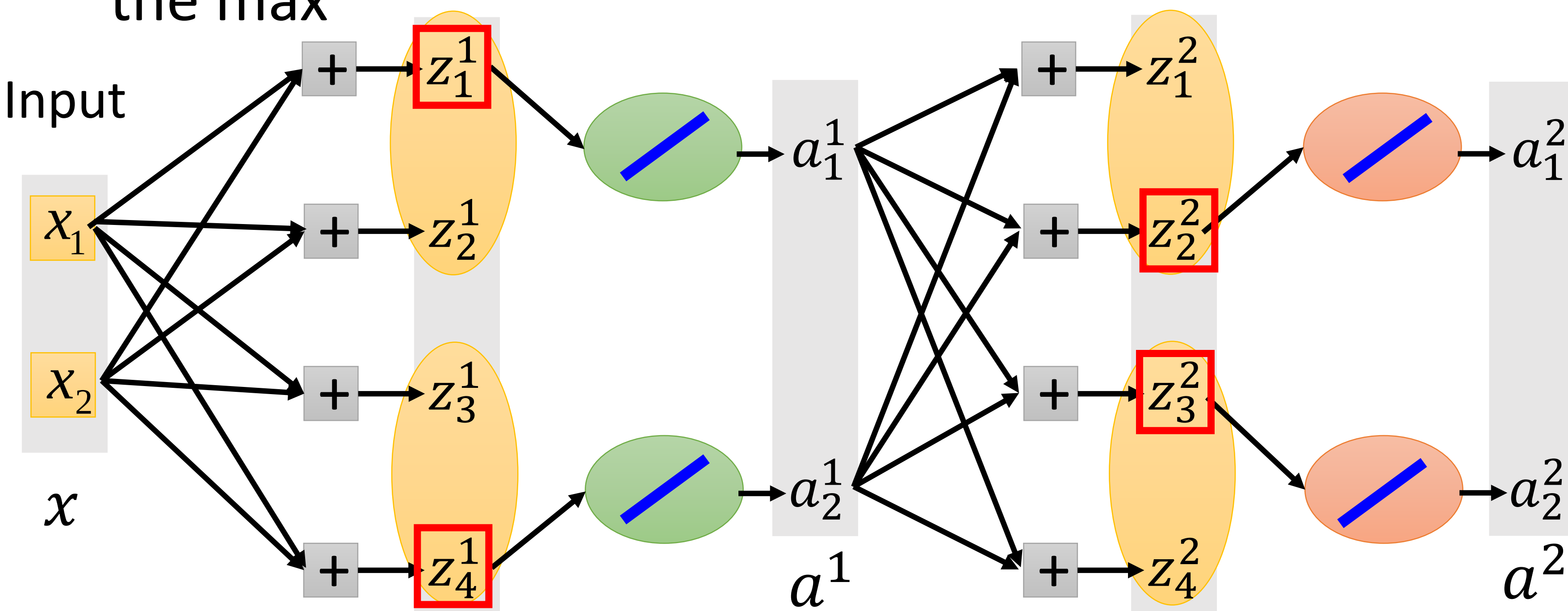
Maxout - Training

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



Maxout - Training

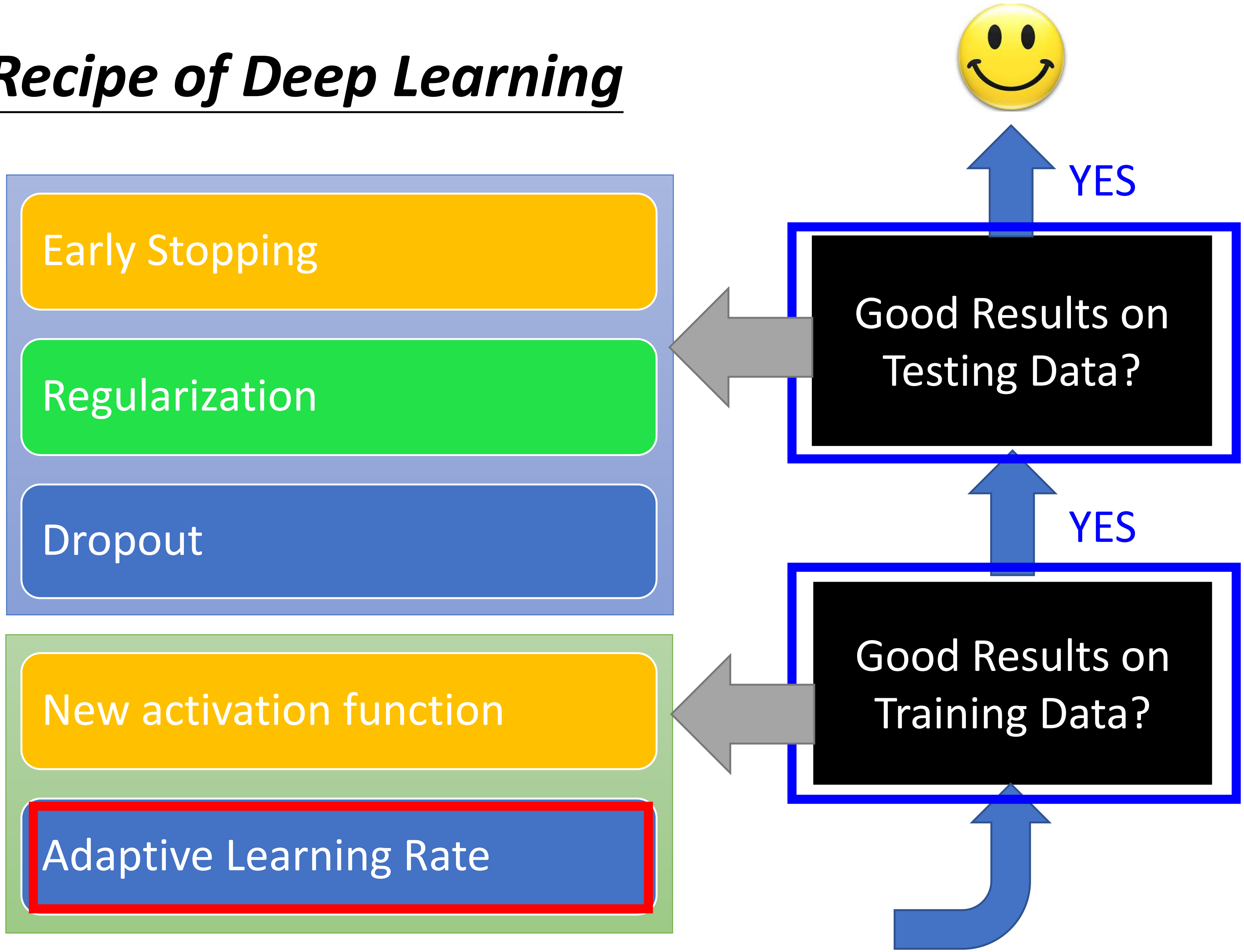
- Given a training data x, we know which z would be the max



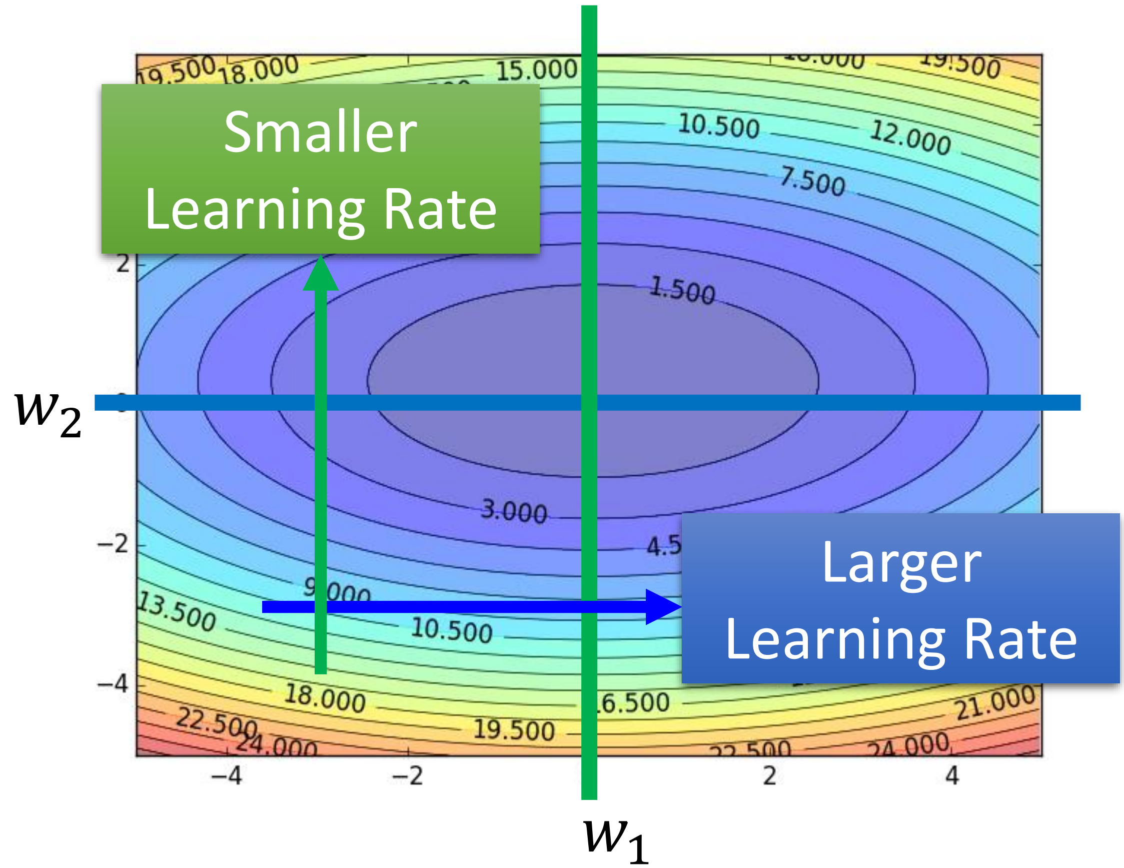
- Train this thin and linear network

Different thin and linear network for different examples

Recipe of Deep Learning



Review



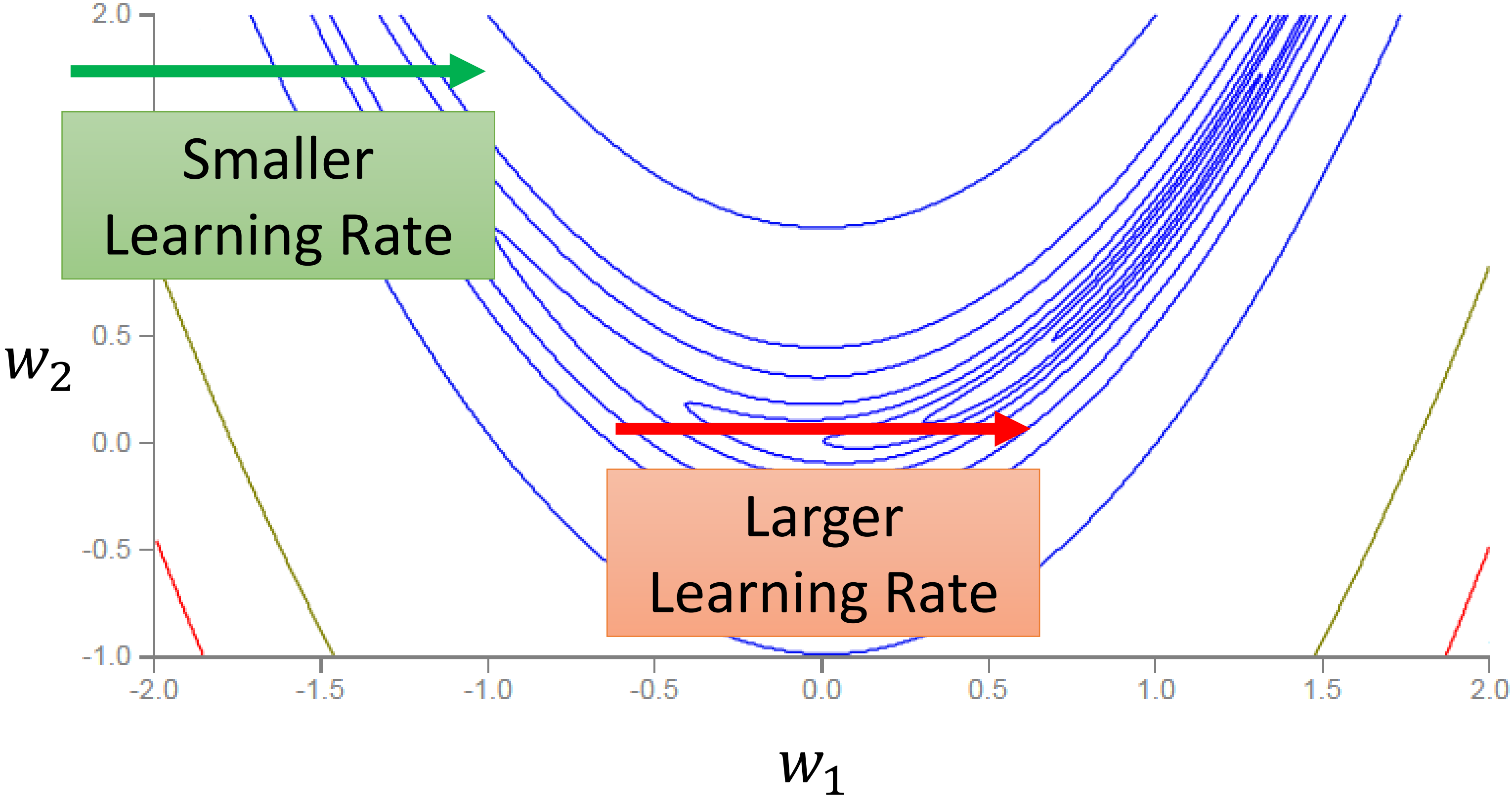
Adagrad

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

Use first derivative to estimate second derivative

RMSProp

Error Surface can be very complex when training NN.



RMSProp

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

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

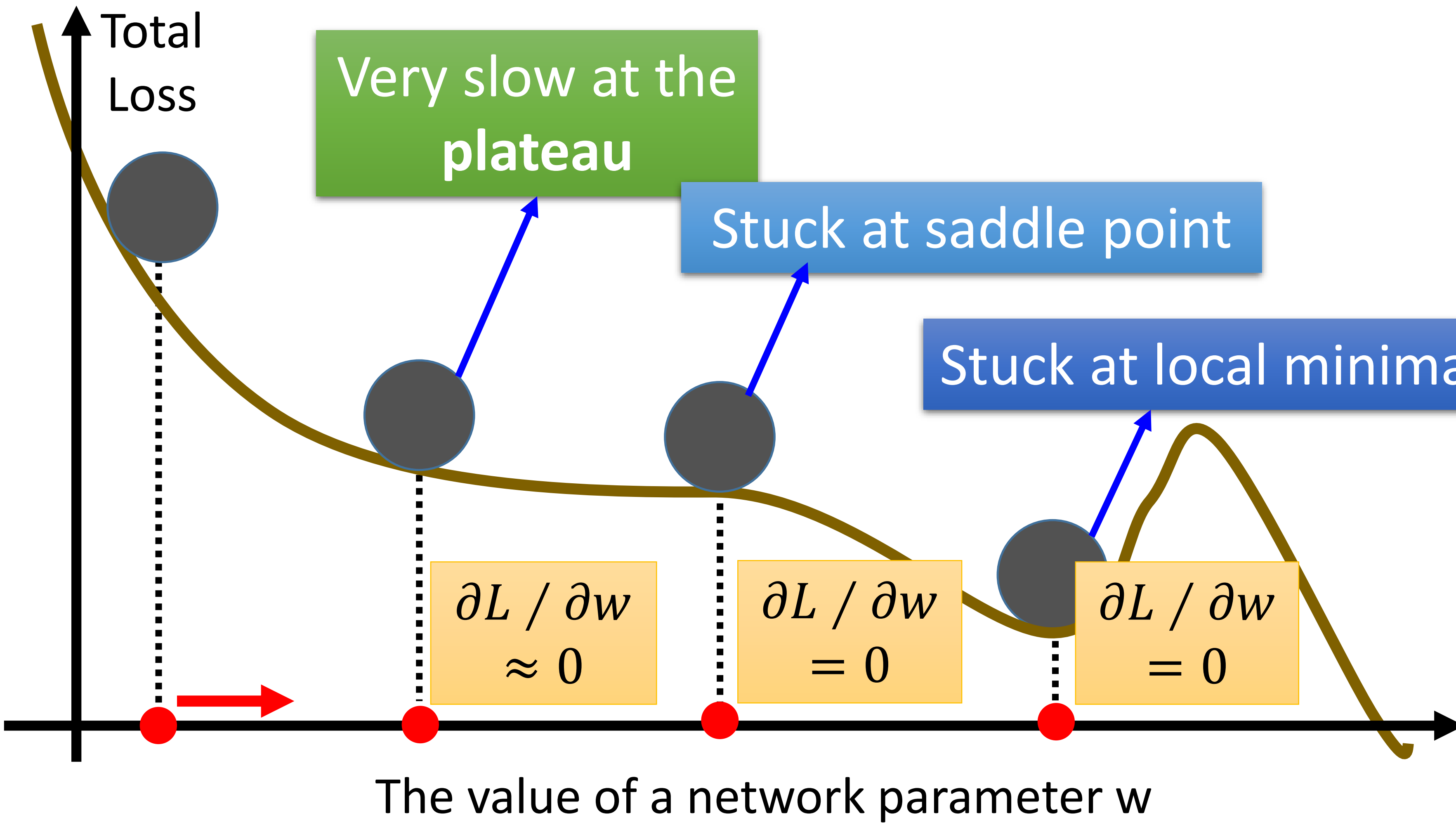
$$w^3 \leftarrow w^2 - \frac{\eta}{\sigma^2} g^2 \qquad \sigma^2 = \sqrt{\alpha(\sigma^1)^2 + (1-\alpha)(g^2)^2}$$

$$\vdots$$

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

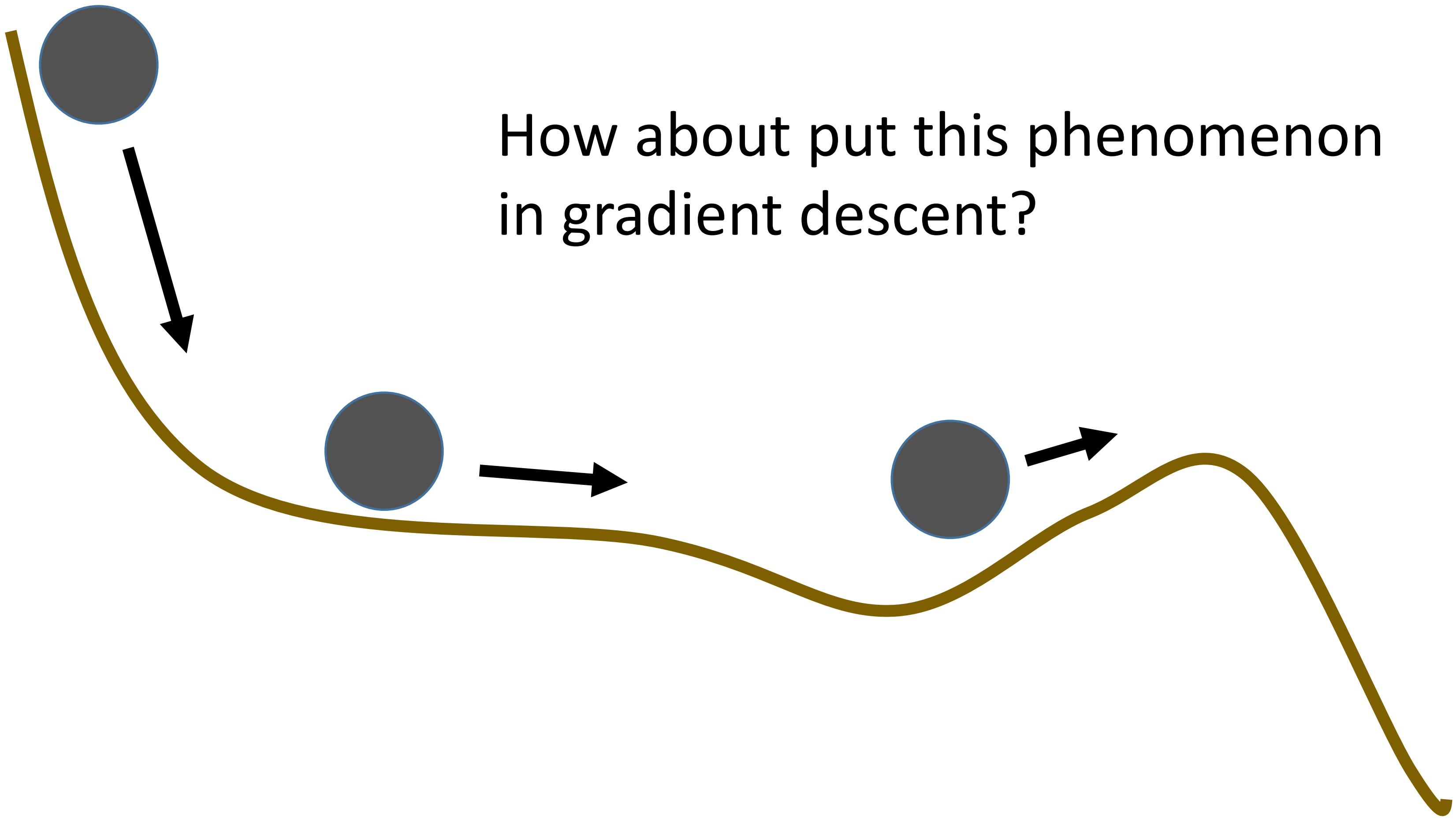
Root Mean Square of the gradients with previous gradients being decayed

Hard to find optimal network parameters



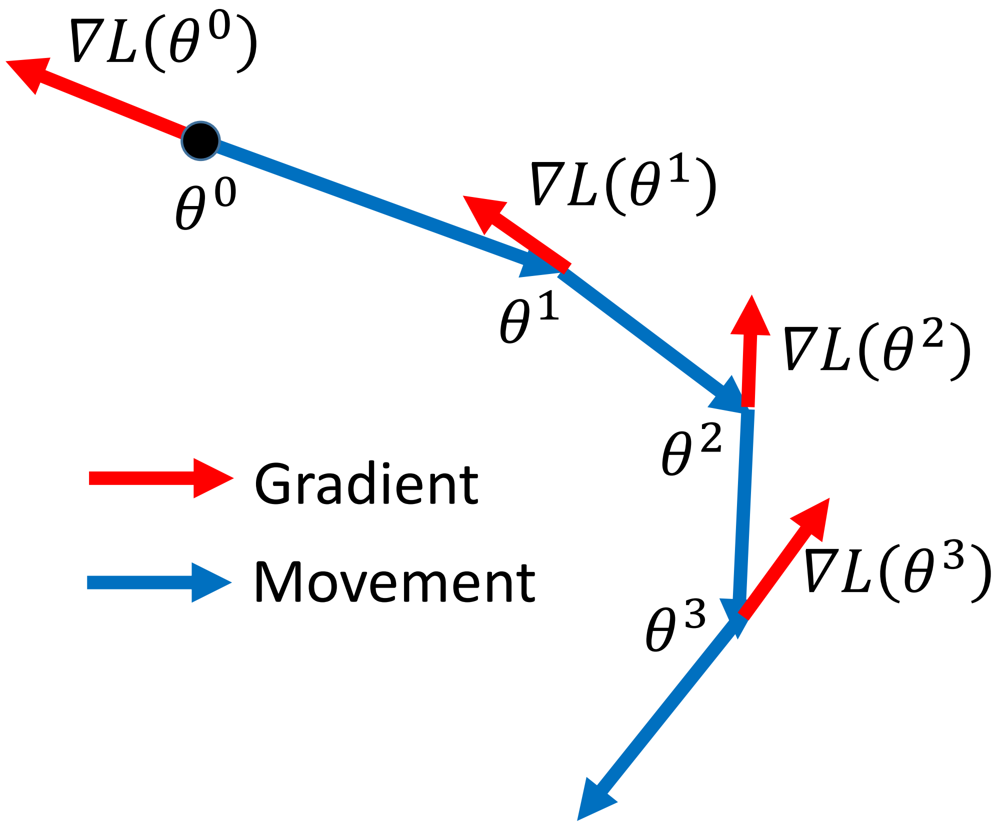
In physical world ……

- 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

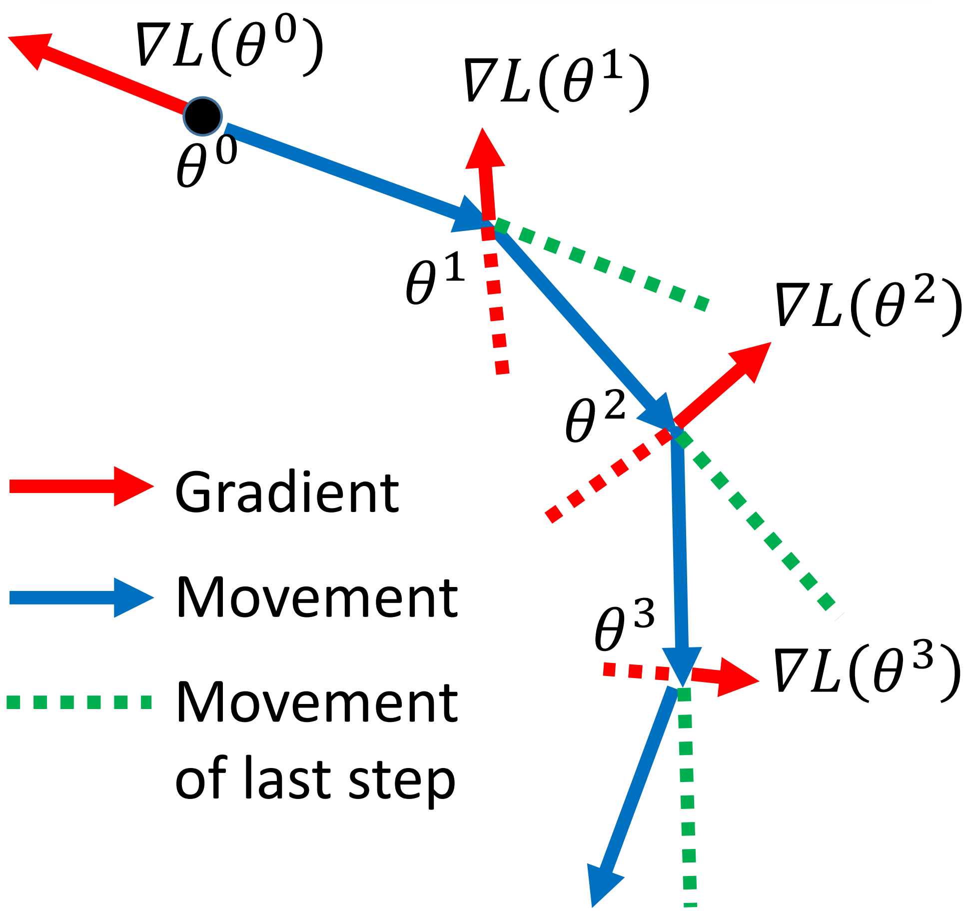
Move to $\theta^2 = \theta^1 - \eta \nabla L(\theta^1)$

⋮

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

$v^0 = 0$

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

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

⋮

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

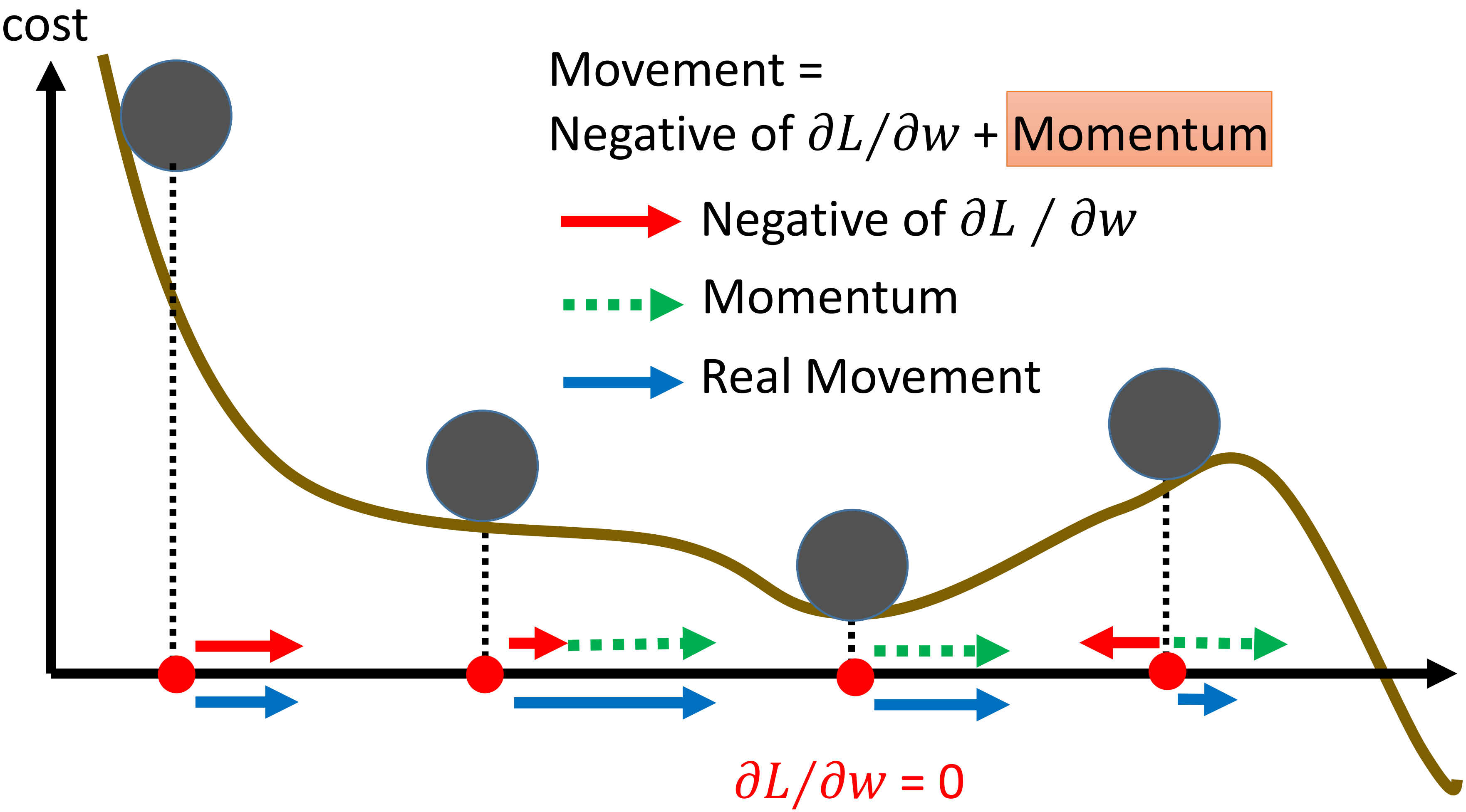
Movement $v^2 = \lambda v^1 - \eta \nabla L(\theta^1)$

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

Movement not just based on gradient, but previous movement

Momentum

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



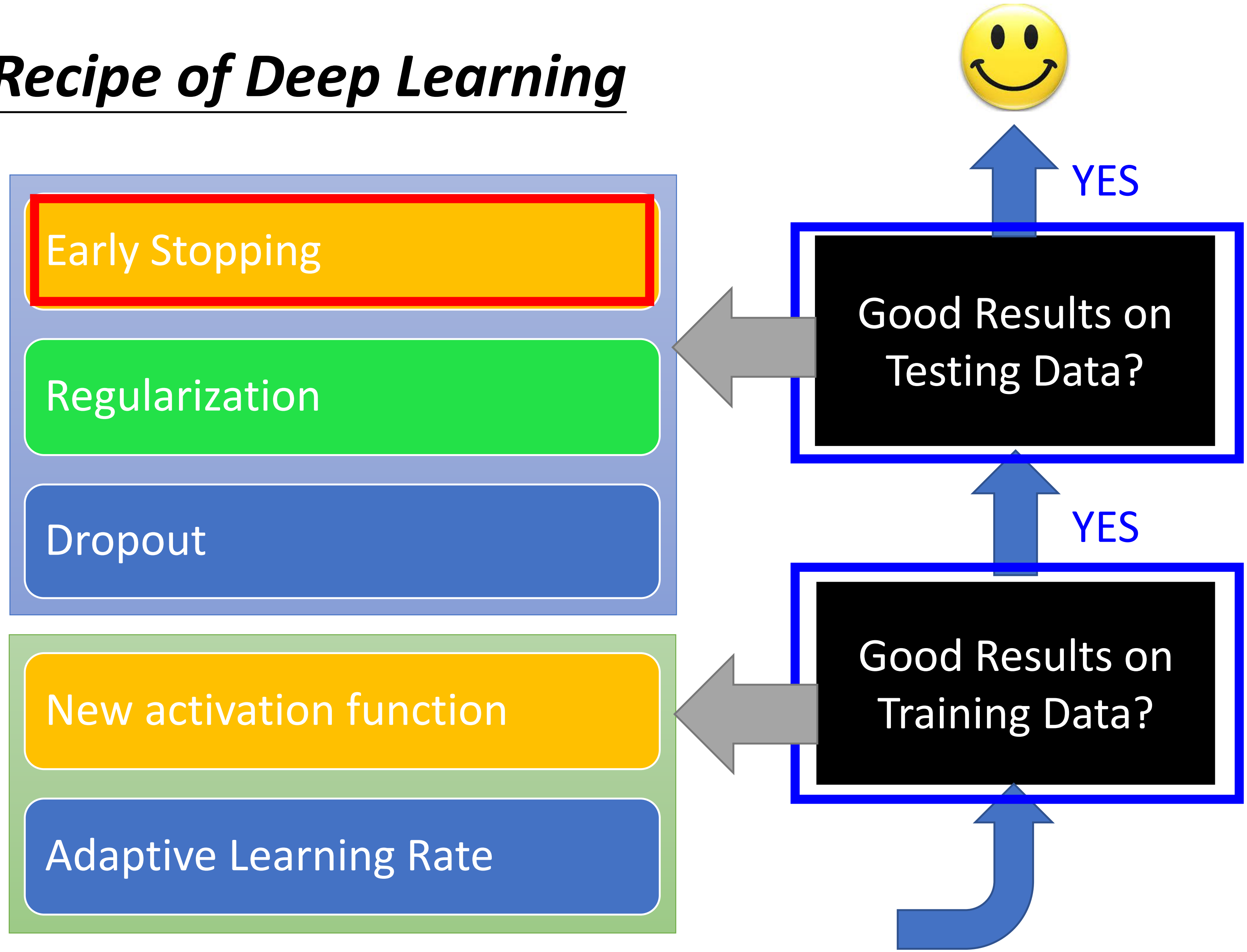
Adam

RMSProp + Momentum

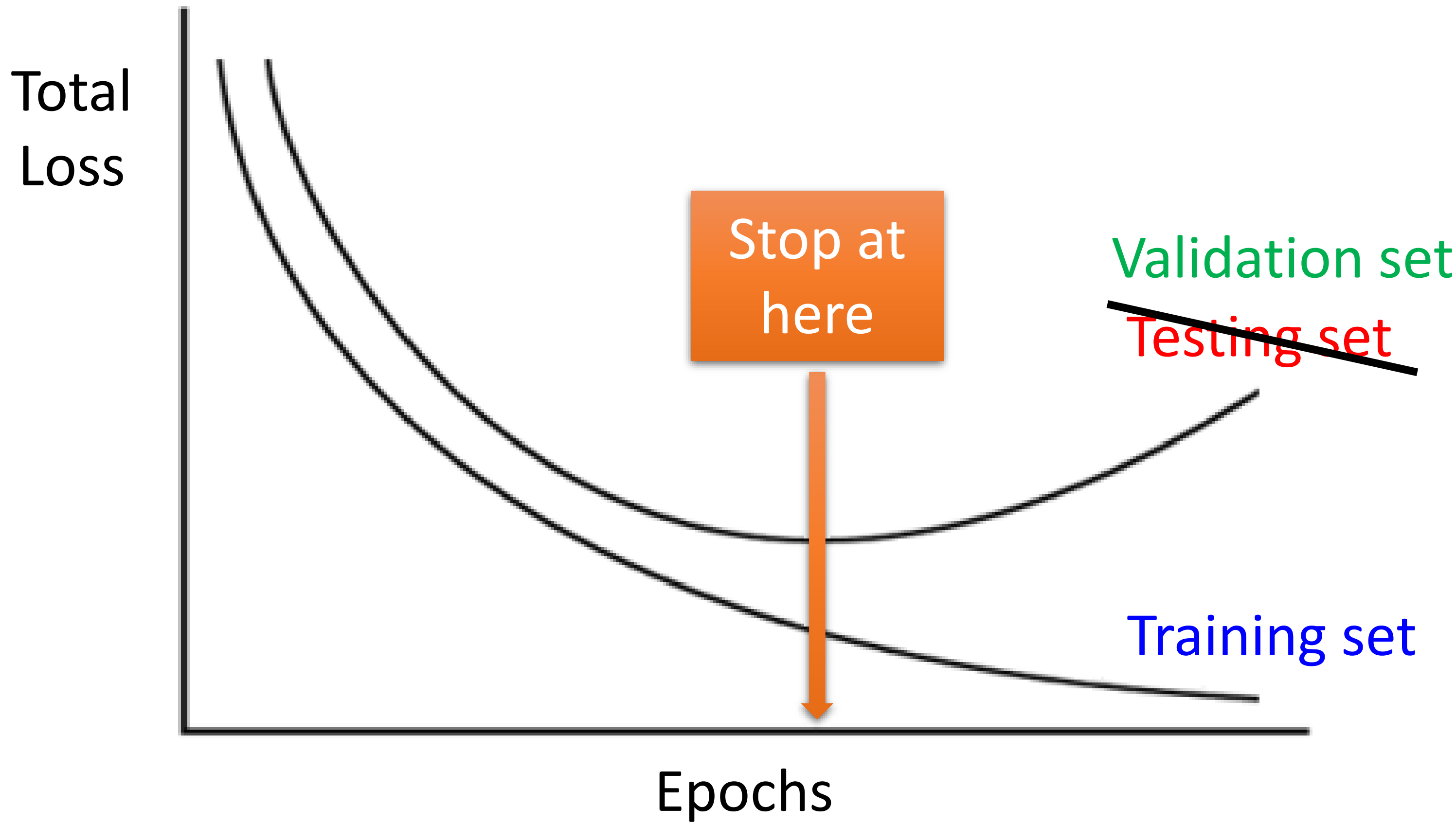
Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ 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

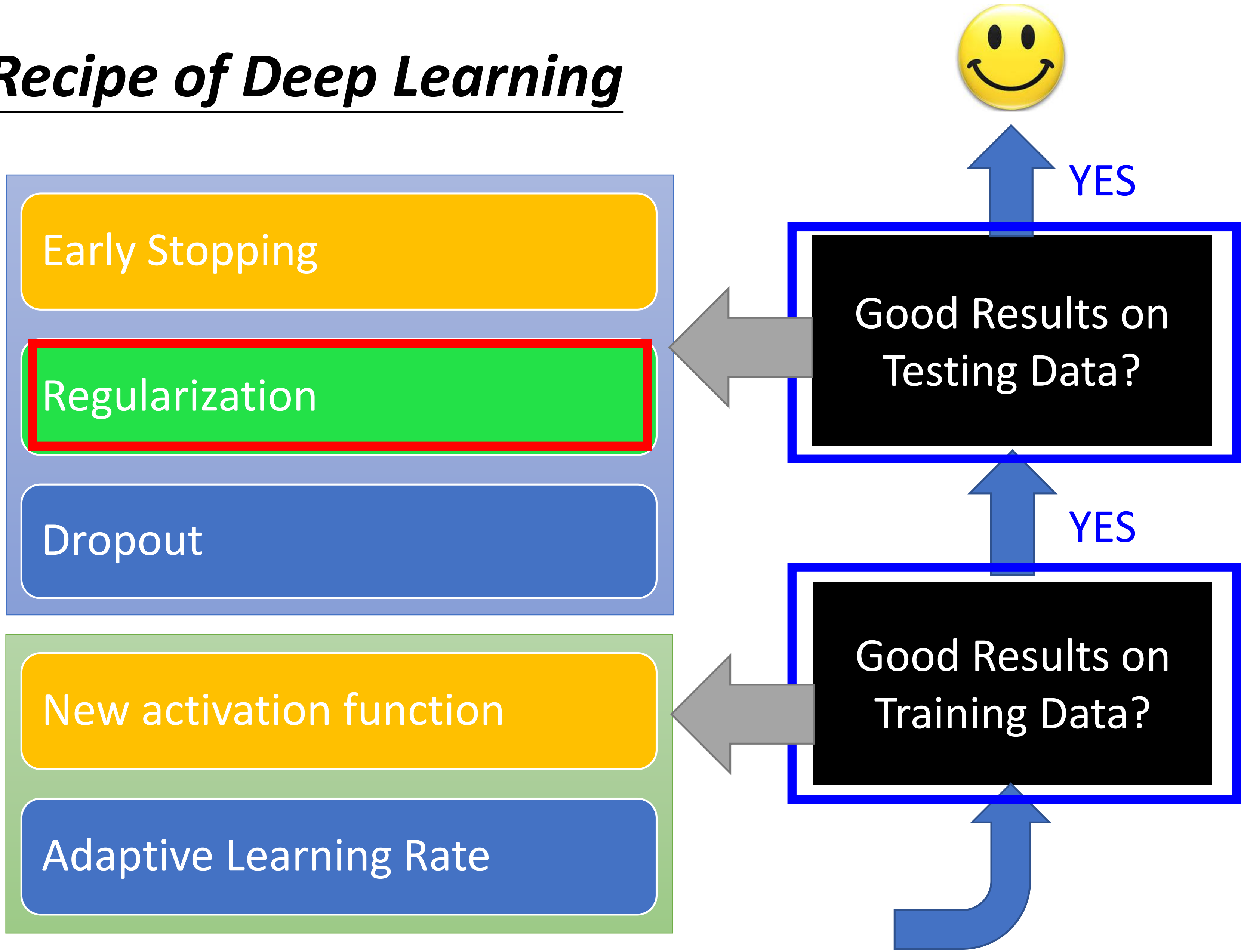


Early Stopping



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

Recipe of Deep Learning



Regularization

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

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

→ Regularization term

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

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

L2 regularization:

$$\|\theta\|_2 = \sqrt{(w_1)^2 + (w_2)^2 + \cdots}$$

(usually not consider biases)

Regularization

L2 regularization:

$$\|\theta\|_2 = \sqrt{(w_1)^2 + (w_2)^2 + \cdots}$$

- New loss function to be minimized

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

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

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

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

Weight Decay

Closer to zero

Regularization

L1 regularization (LASSO):

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

- New loss function to be minimized

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

Update:

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

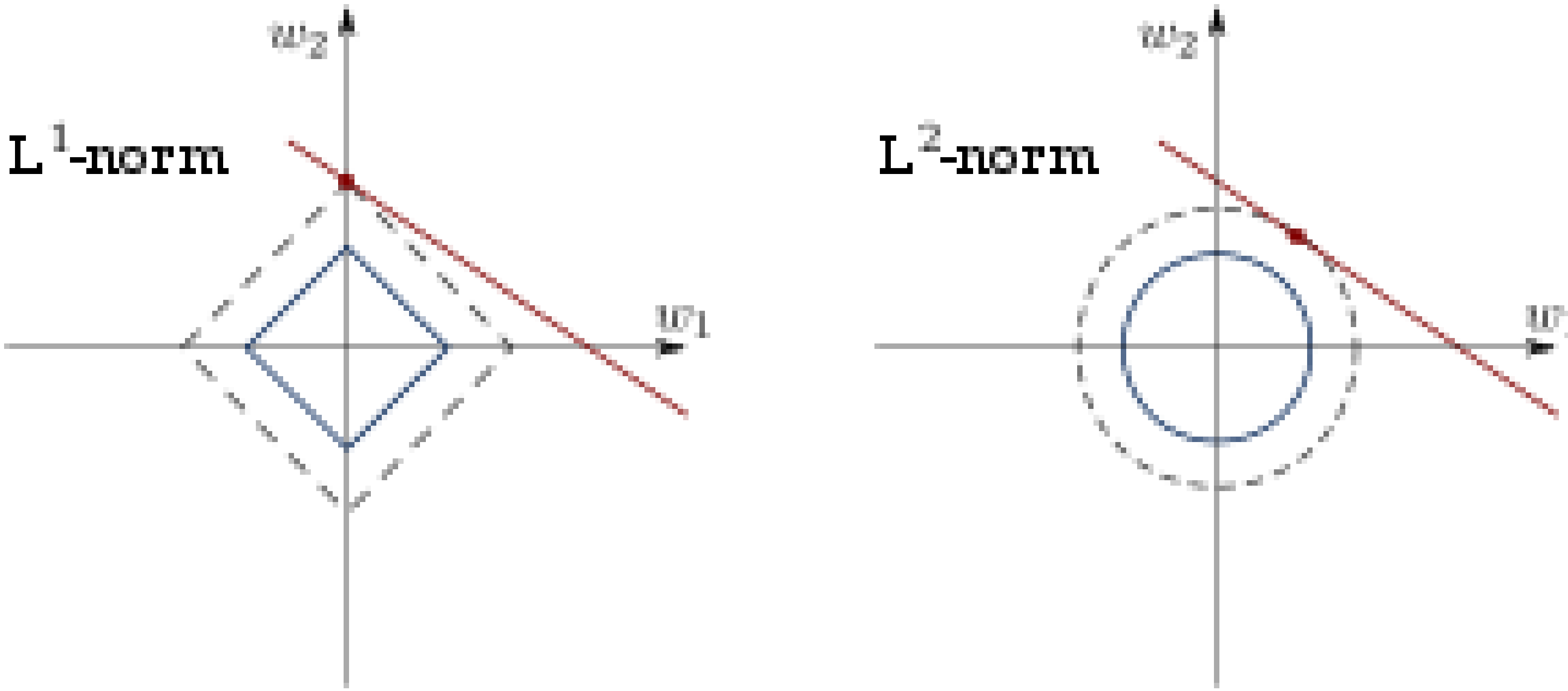
Magnitude decreases by a constant

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

Compare to L2

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

Magnitude decay by a factor



L1 regularization forces some attributes to be EXACTLY zero
➔ Less attributes and corresponding coefficients.

https://en.wikipedia.org/wiki/Lasso_(statistics)

L1-Regularization and Exact Recovery

Let $\bar{\mathbf{x}} \in \mathbb{R}^n$ have support S, $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{y} = \mathbf{A}\bar{\mathbf{x}}$. By Lemma.14.8, if

$$\begin{array}{c} Null(\mathbf{A}_S) = \{\mathbf{0}\} \\ \exists \mathbf{v} \in \mathbb{R}^m : (\mathbf{A}^T\mathbf{v})_S = \operatorname{sign}(\bar{\mathbf{x}}_S), \quad \|(\mathbf{A}^T\mathbf{v})_{\neg S}\|_\infty < 1 \end{array} \tag{12}$$

Then $\bar{\mathbf{x}}$ is the unique optimal solution to the following optimization problem

$$\begin{array}{ll} \text{minimize} & \|\mathbf{x}\|_1 \\ \text{subject to} & \mathbf{y} = \mathbf{A}\mathbf{x} \\ \text{variables} & \mathbf{x} \in \mathbb{R}^n \end{array} \tag{13}$$

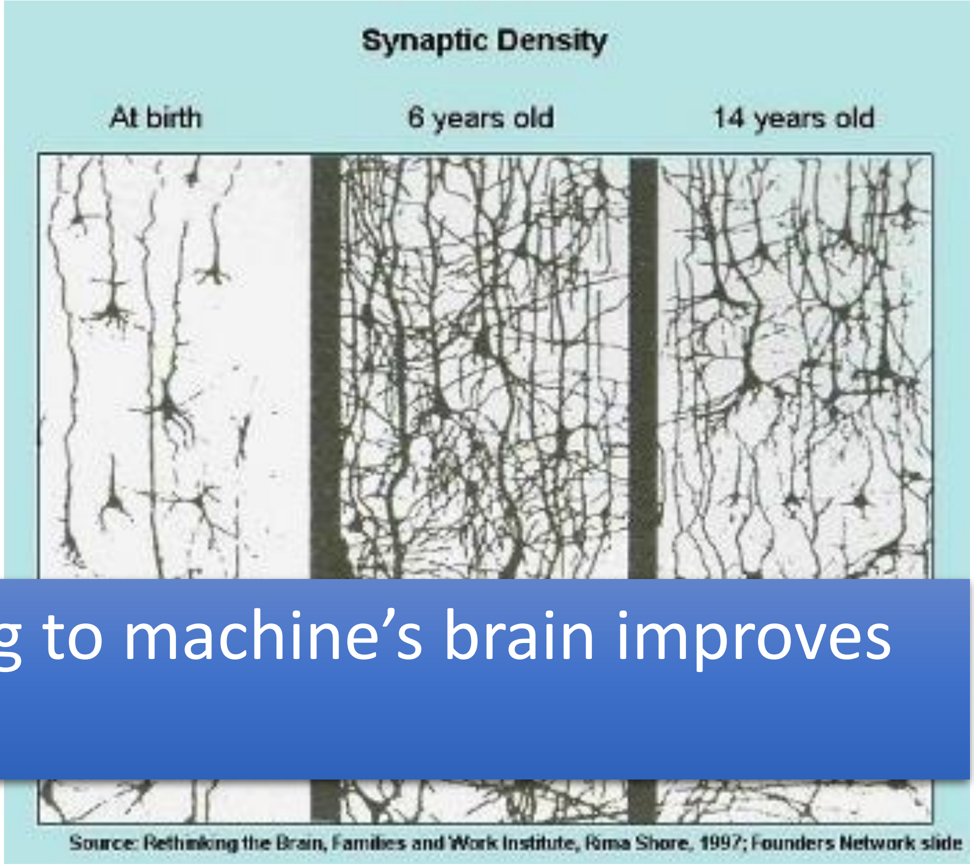
In other words, exact recovery of $\bar{\mathbf{x}}$ is achieved by solving the ℓ_1-minimization problem (13), provided the condition (12) is met.

Lemma 14.9. *Let $\bar{\mathbf{x}} \in \mathbb{R}^n$ have support S, where $|S| = s$, and let $\mathbf{A} \in \mathbb{R}^{m\times n}$ be a random matrix where each element $a_{ij} \in \mathcal{N}(0,1)$ is i.i.d. Then condition (12) holds with probability at least*

$$1 - 2 \inf_{\epsilon_1 \in (0,1)} \left(\left((9/\epsilon_1)^s \, e^{-0.2017 m \epsilon_1^2} + n \exp\left(-\frac{m(1-\epsilon_1)^2}{2s} \right) \right) \right) \tag{14}$$

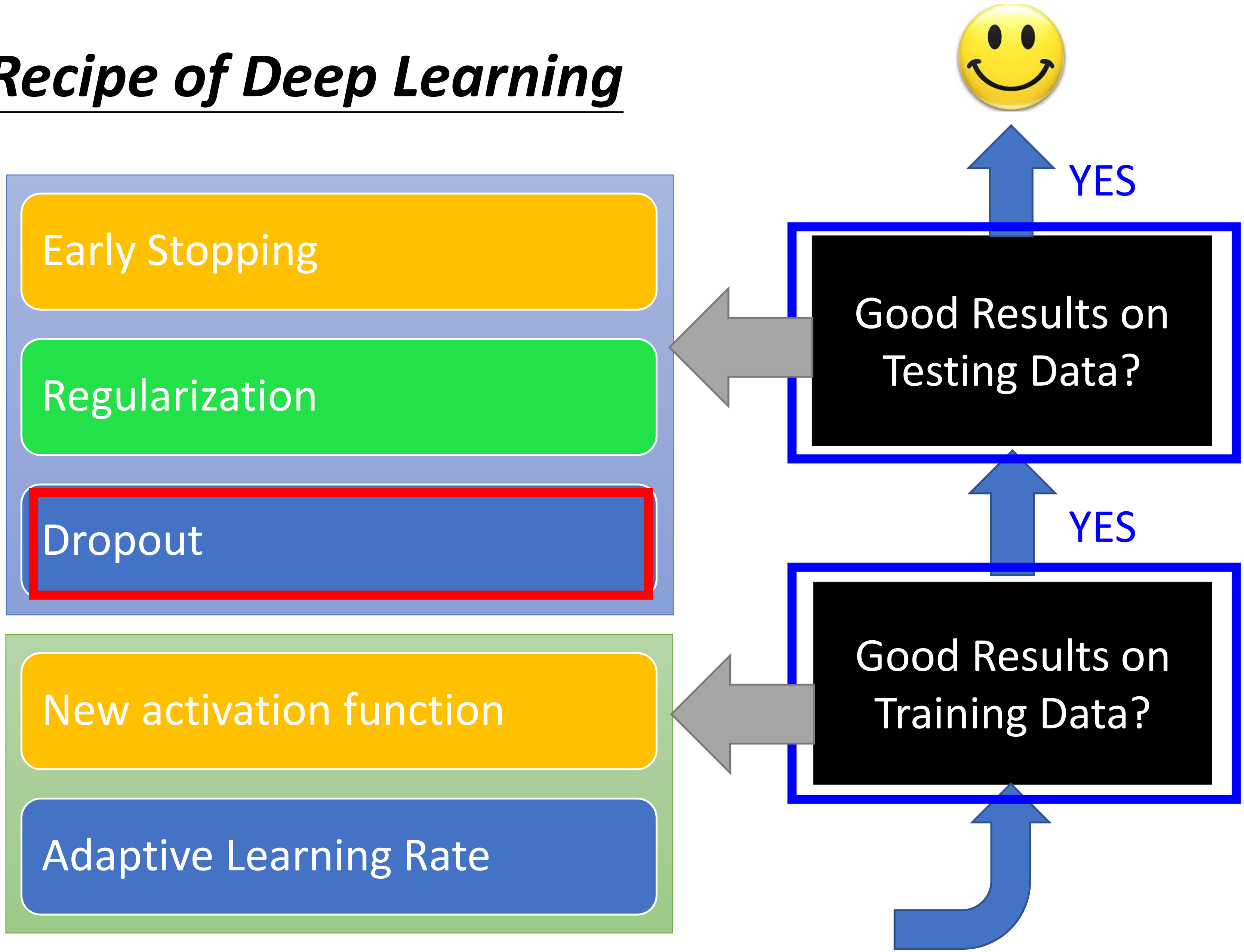
Regularization - Weight Decay

- Our brain prunes out the useless link between neurons.



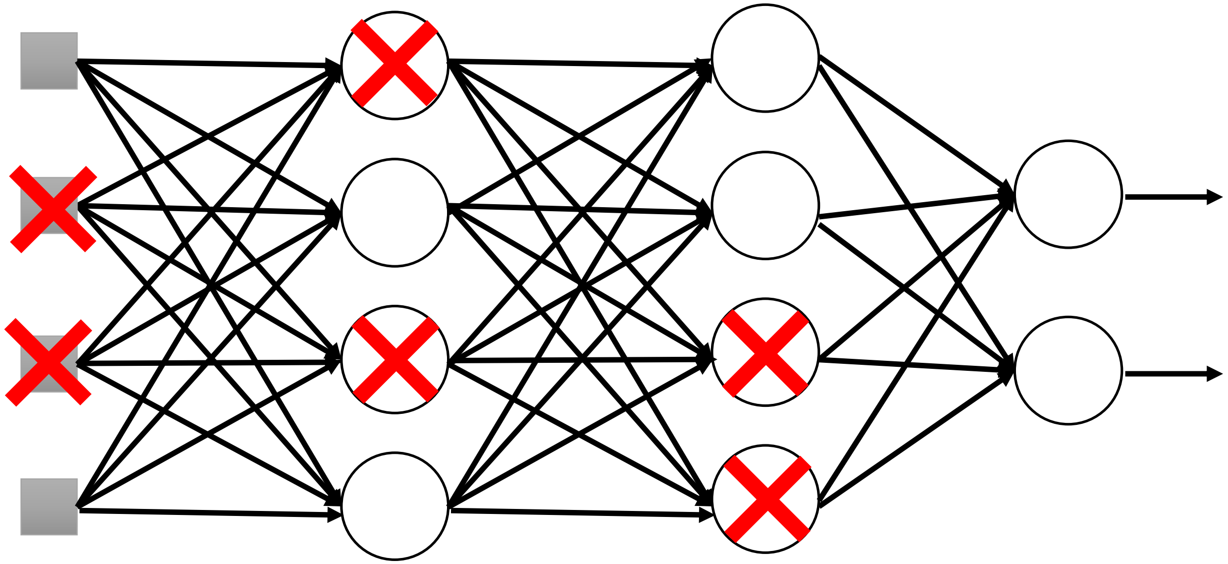
Doing the same thing to machine’s brain improves the performance.

Recipe of Deep Learning



Dropout

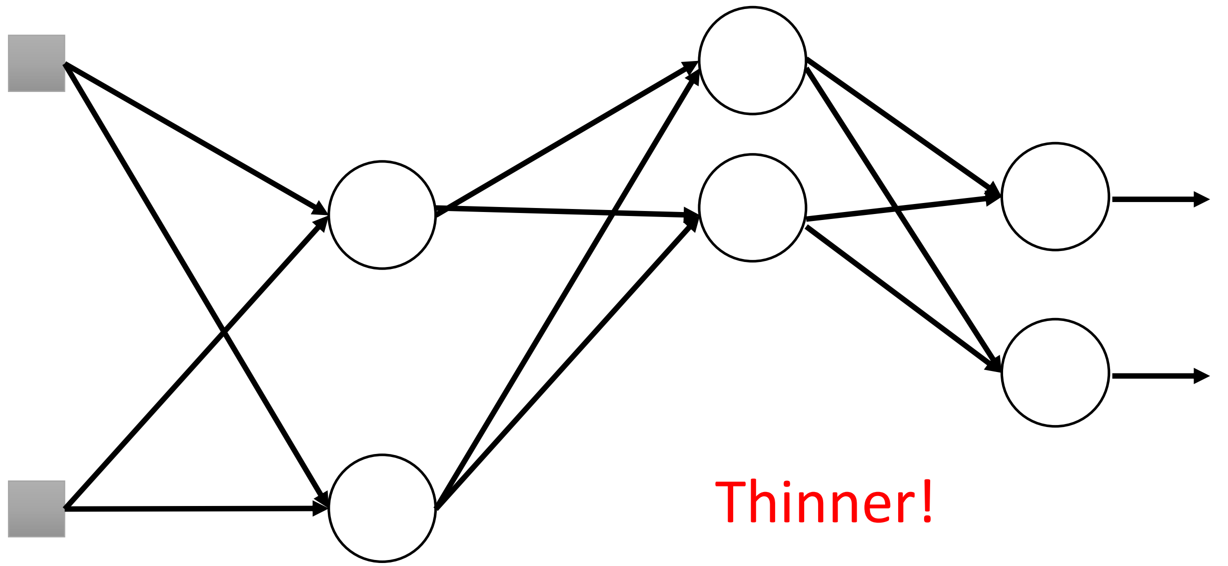
Training:



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

Dropout

Training:

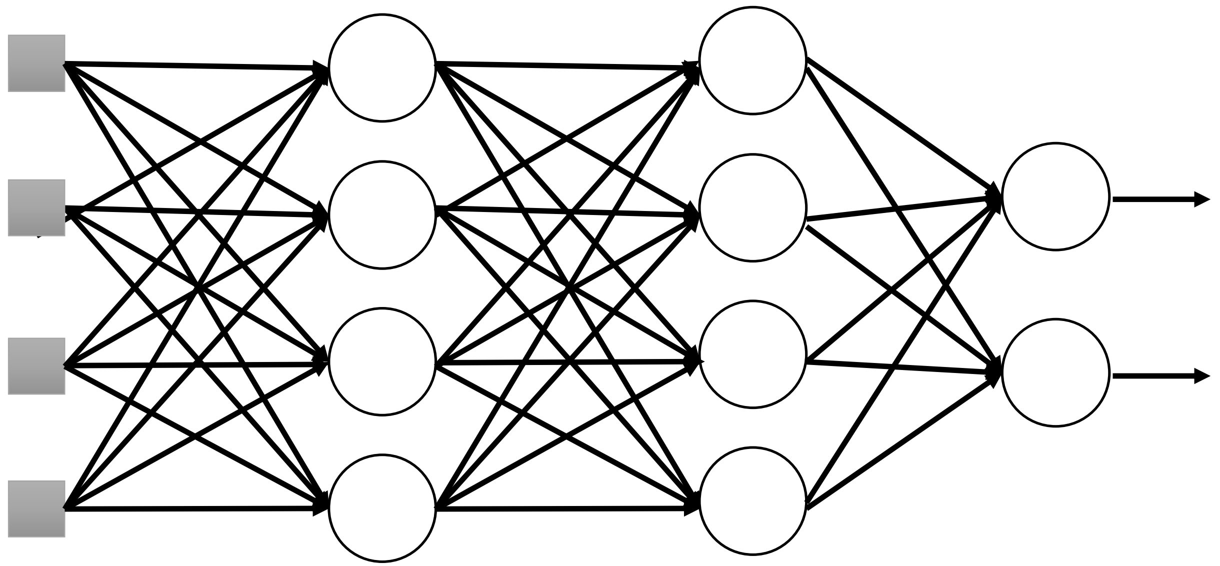


- **Each time before updating the parameters**
  - Each neuron has p% to dropout
    - ➡ **The structure of the network is changed.**
  - Using the new network for training

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

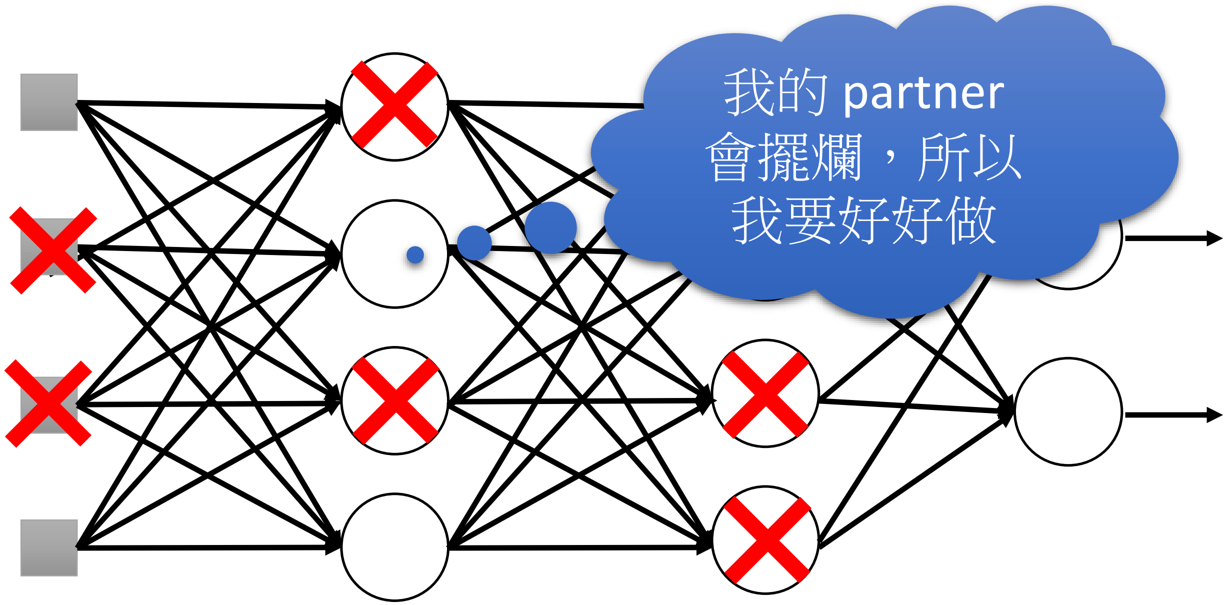
Testing

No dropout
(拿下重物後就變很強)

Training

Dropout (腳上綁重物)

Dropout - Intuitive Reason



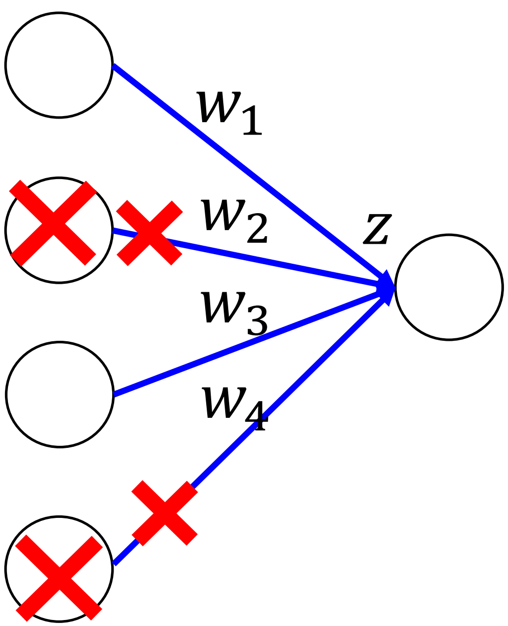
- When teams up, if everyone expect the partner will do the work, nothing will be done finally.
- However, if you know your partner will dropout, you will do better.
- When testing, no one dropout actually, so obtaining good results eventually.

Dropout - Intuitive Reason

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

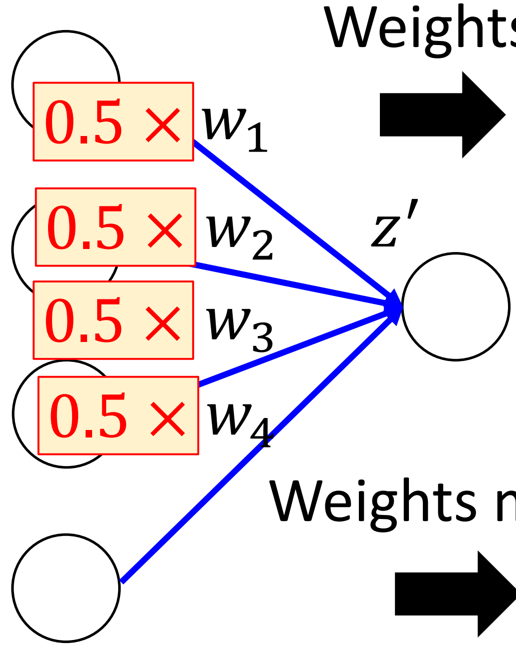
Training of Dropout

Assume dropout rate is 50%



Testing of Dropout

No dropout



Weights from training

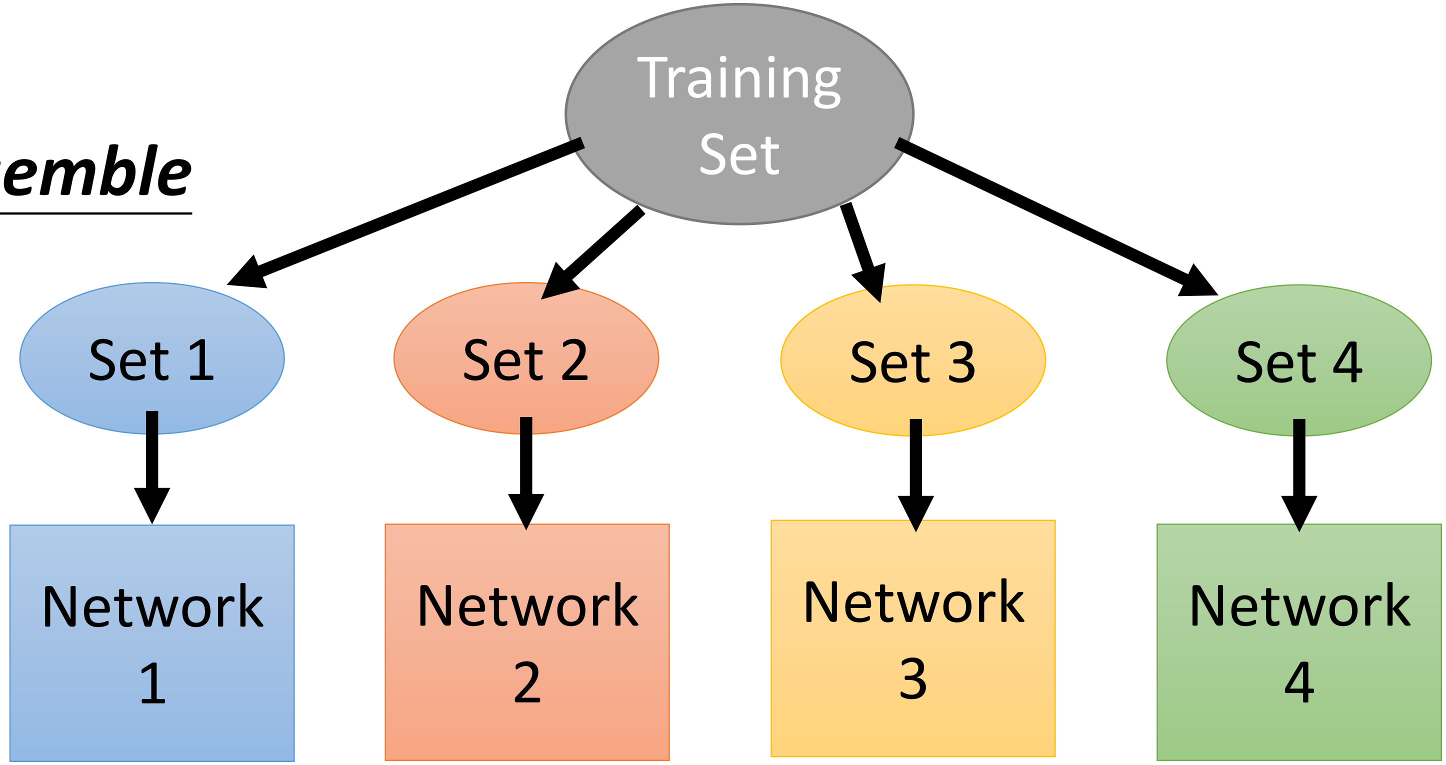
$\Rightarrow z' \approx 2z$

Weights multiply 1-p%

$\Rightarrow z' \approx z$

Dropout is a kind of ensemble.

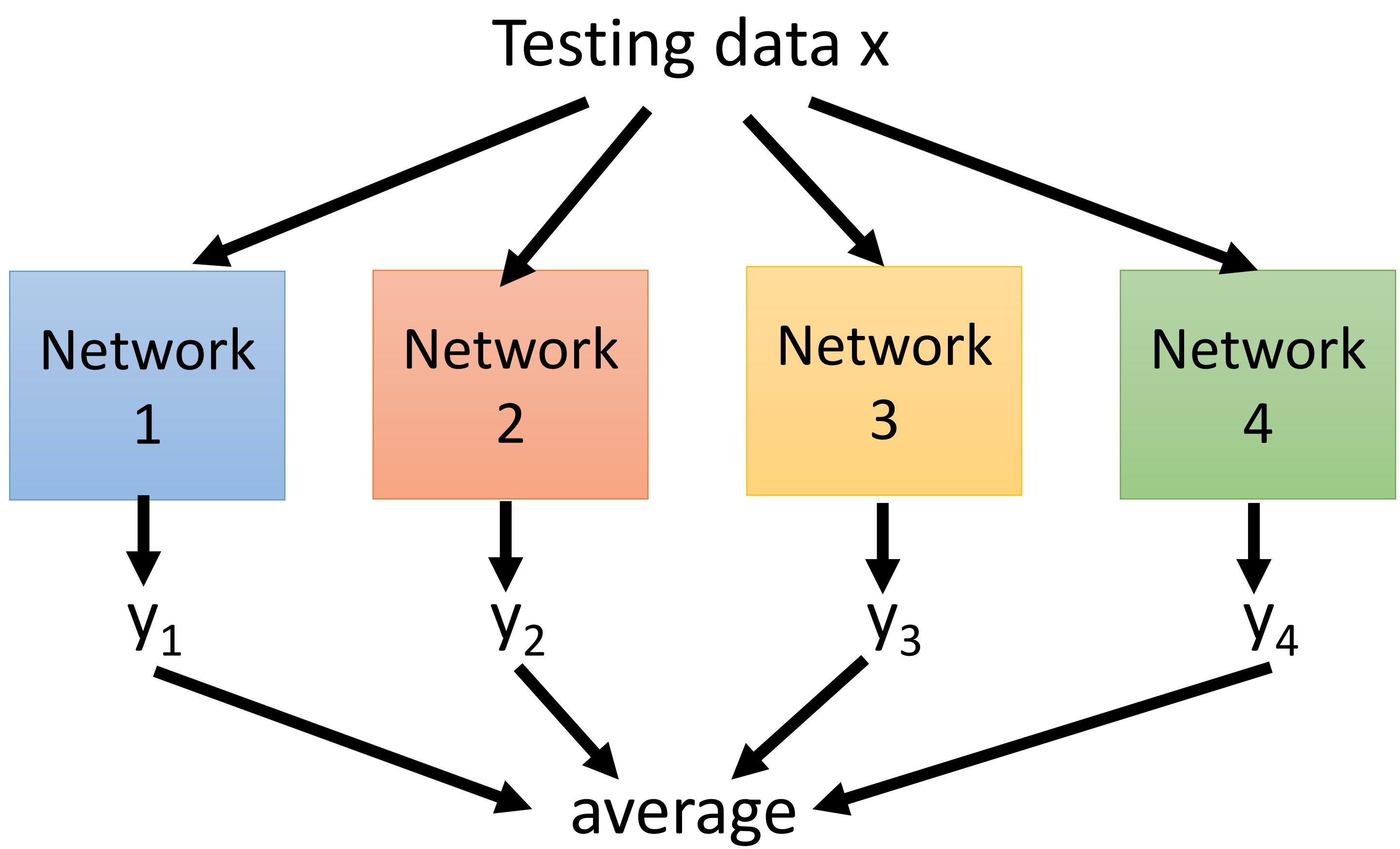
Ensemble



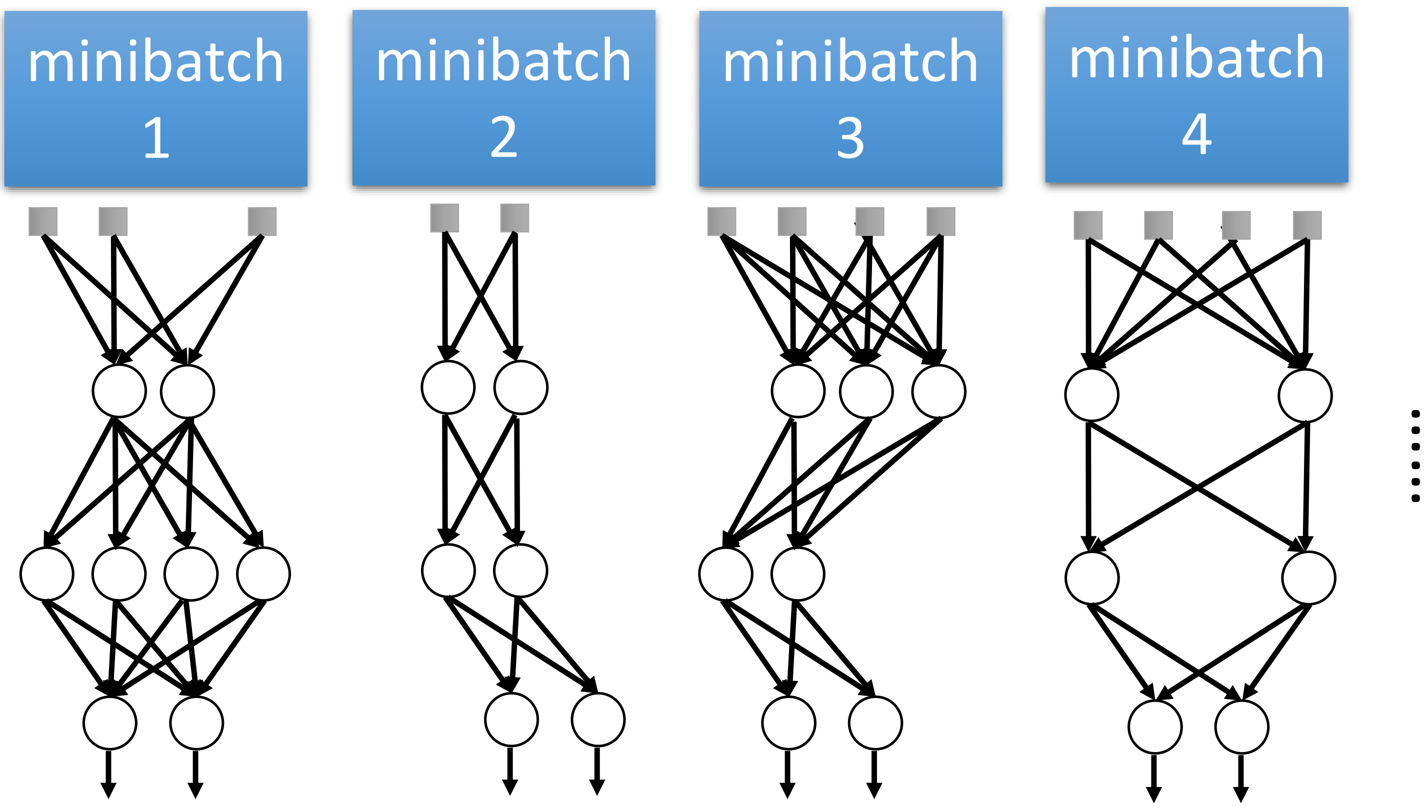
Train a bunch of networks with different structures

Dropout is a kind of ensemble.

Ensemble



Dropout is a kind of ensemble.



Training of Dropout

M neurons

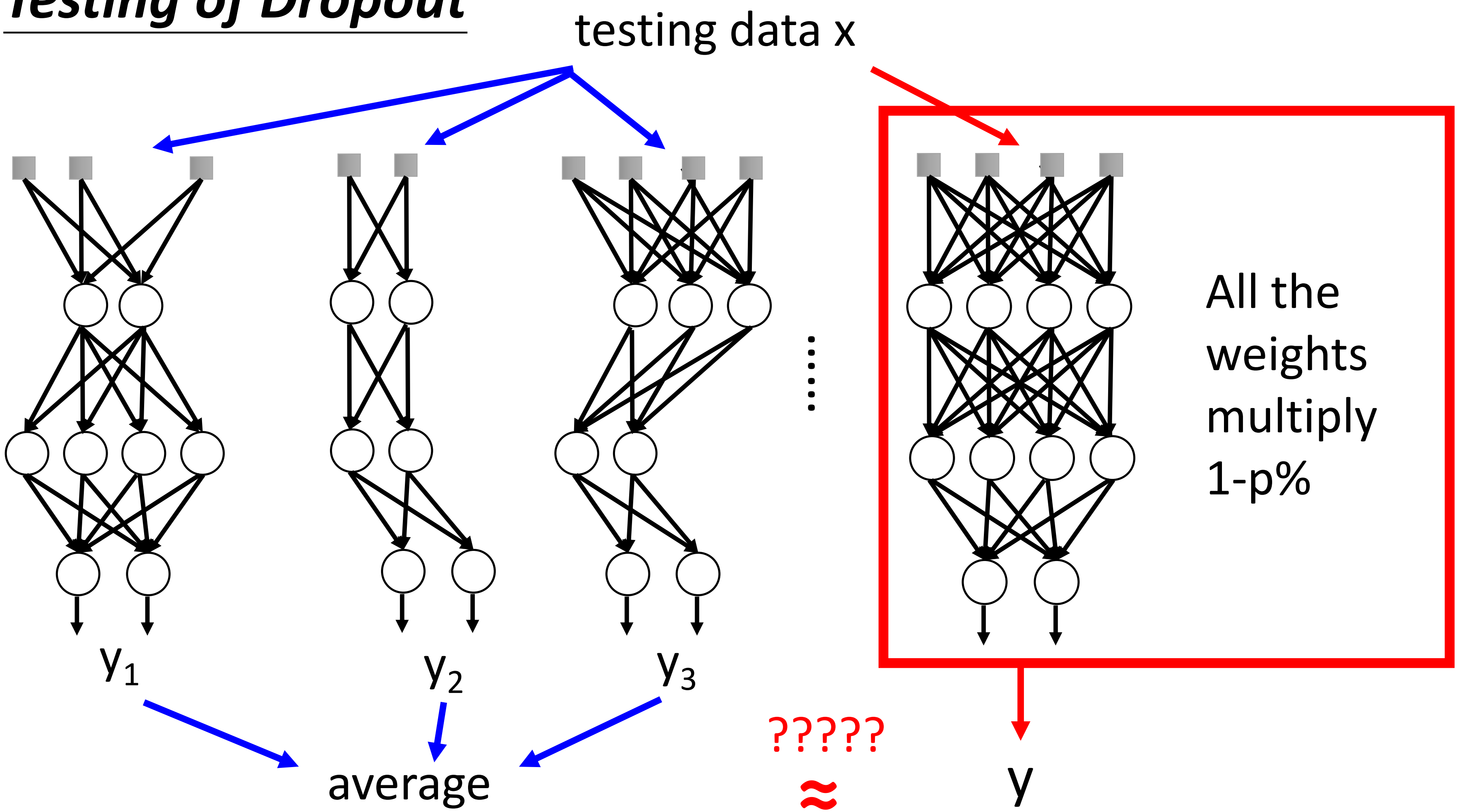
2^M possible networks

➢Using one mini-batch to train one network

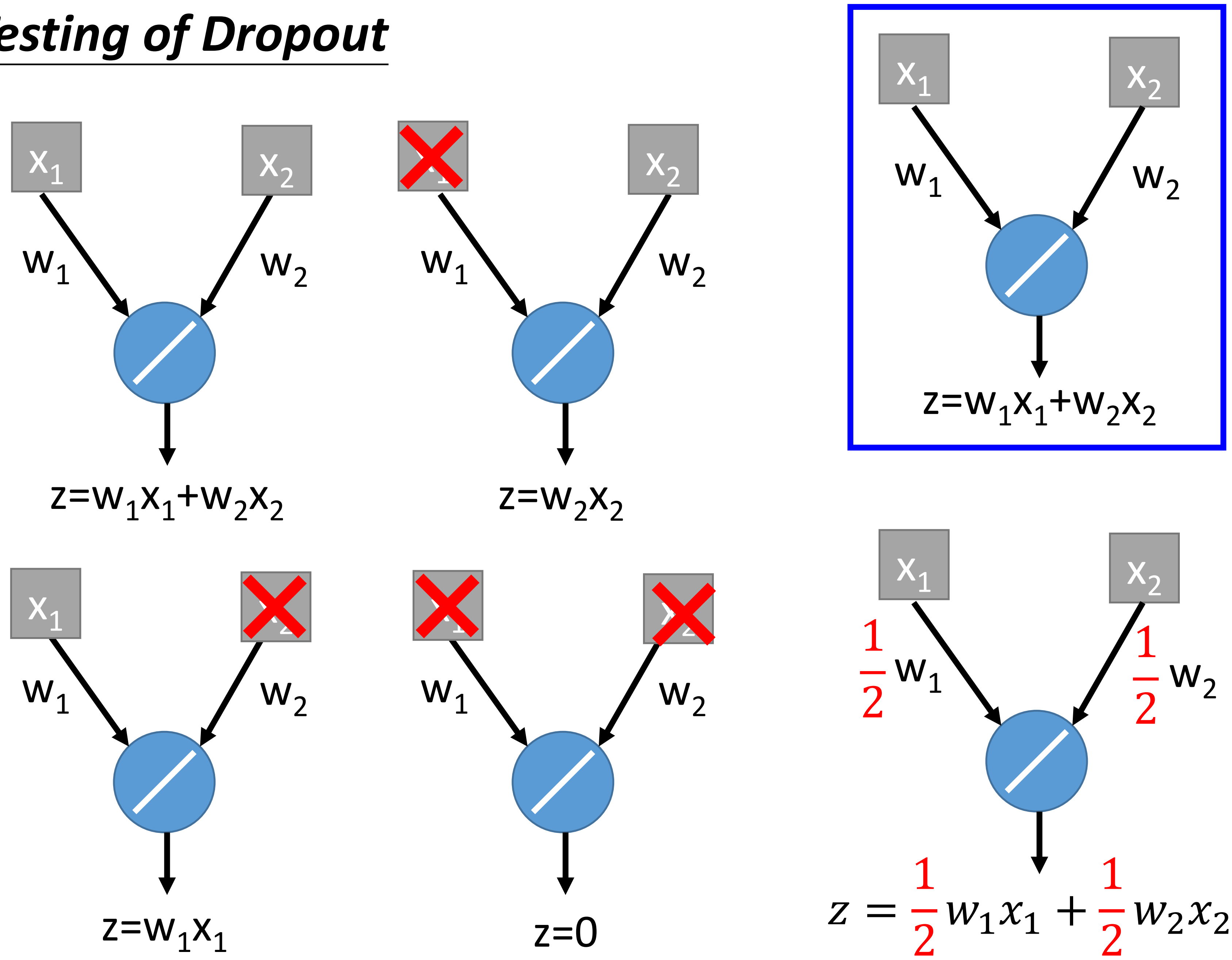
➢Some parameters in the network are shared

Dropout is a kind of ensemble.

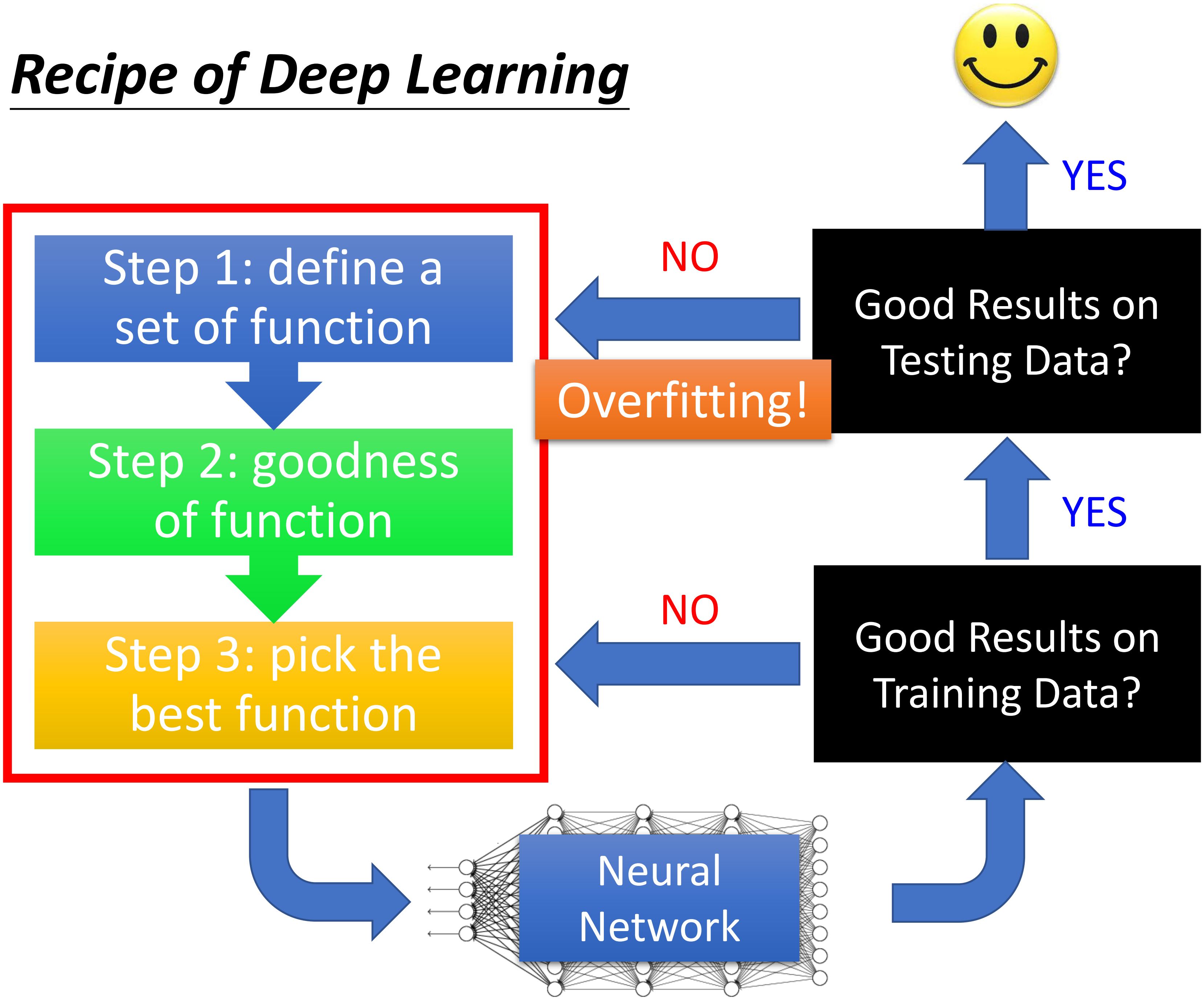
Testing of Dropout



Testing of Dropout



Recipe of Deep Learning



Live Demo