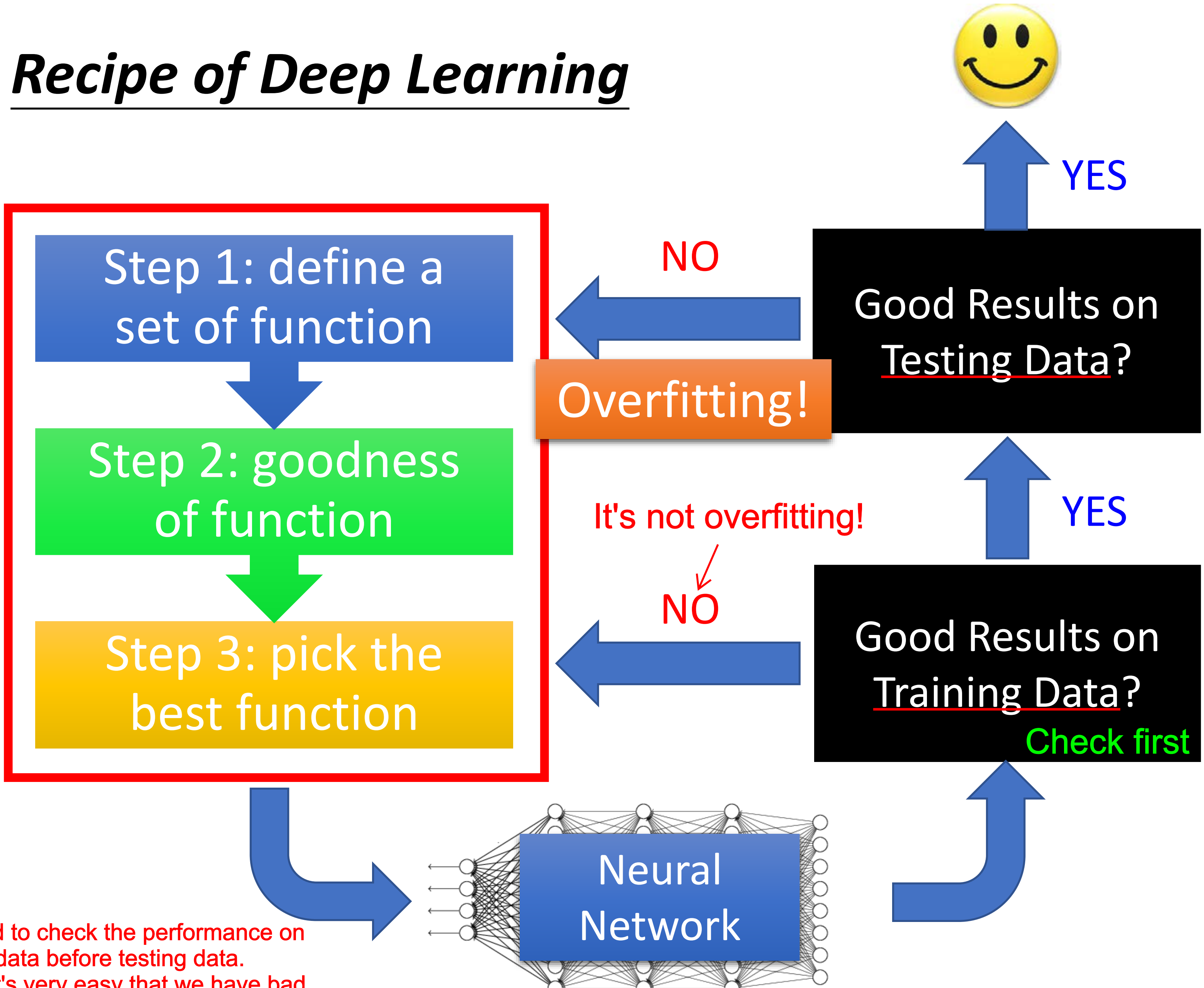
Tips for Deep Learning

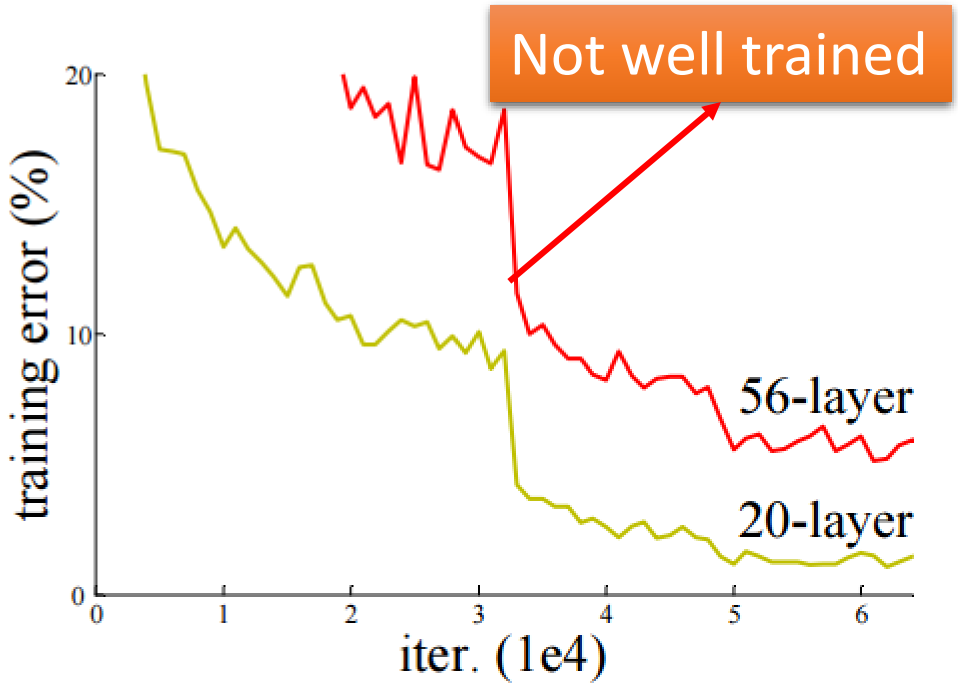
Recipe of Deep Learning



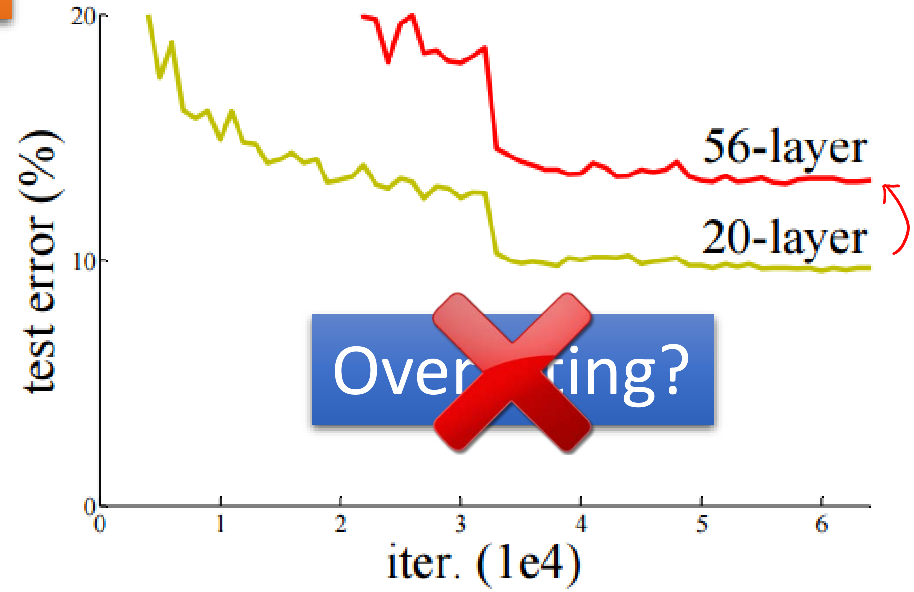
We need to check the performance on training data before testing data.
(In DL, It's very easy that we have bad performance even on training data.)

Do not always blame Overfitting

Didn't find the global minima in training.



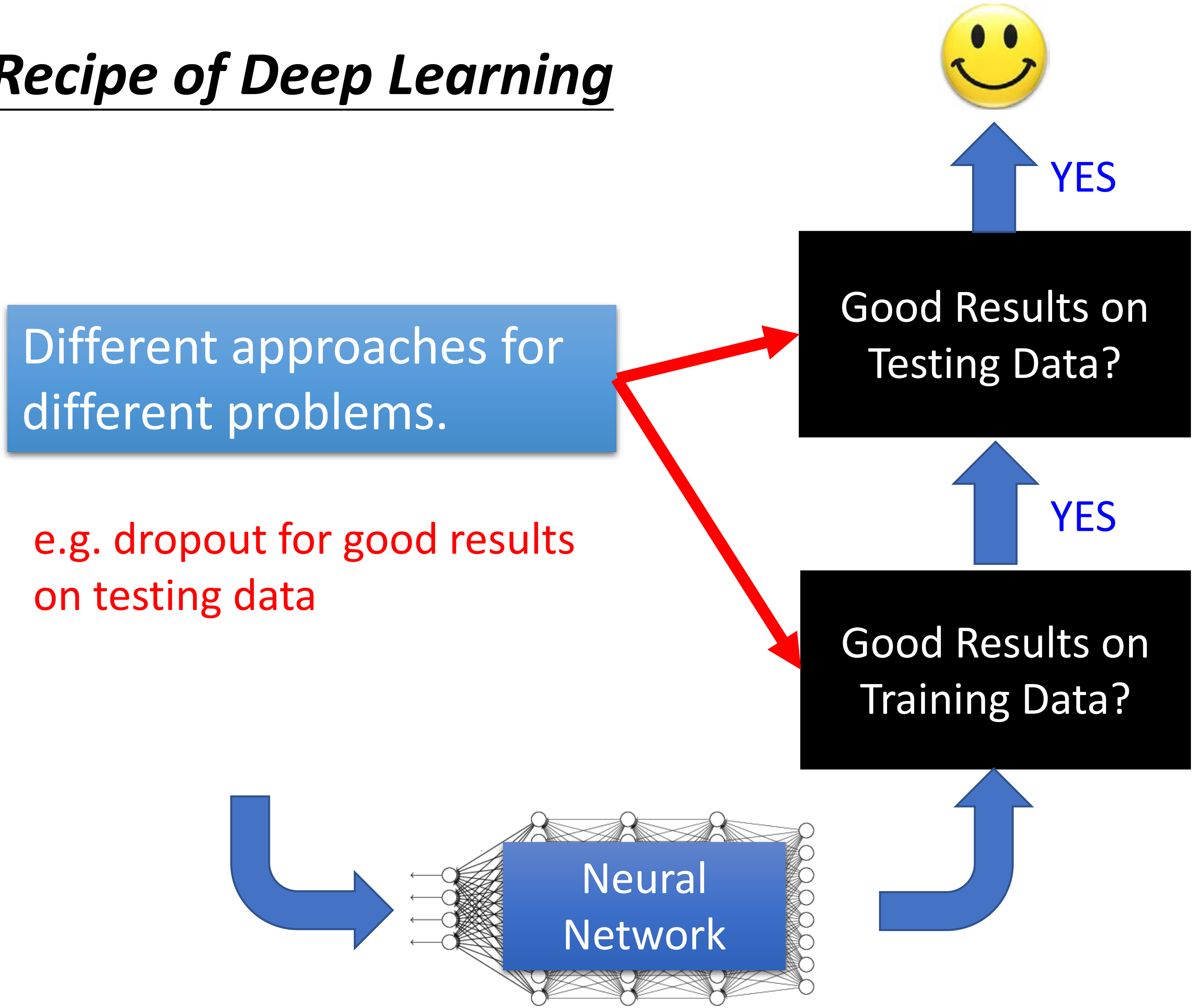
Training Data



Testing Data

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

Recipe of Deep Learning



Recipe of Deep Learning

Already taught in ML, but this chapter will discuss the DL aspects.

- Early Stopping
- Regularization
- Dropout

- New activation function
- Adaptive Learning Rate

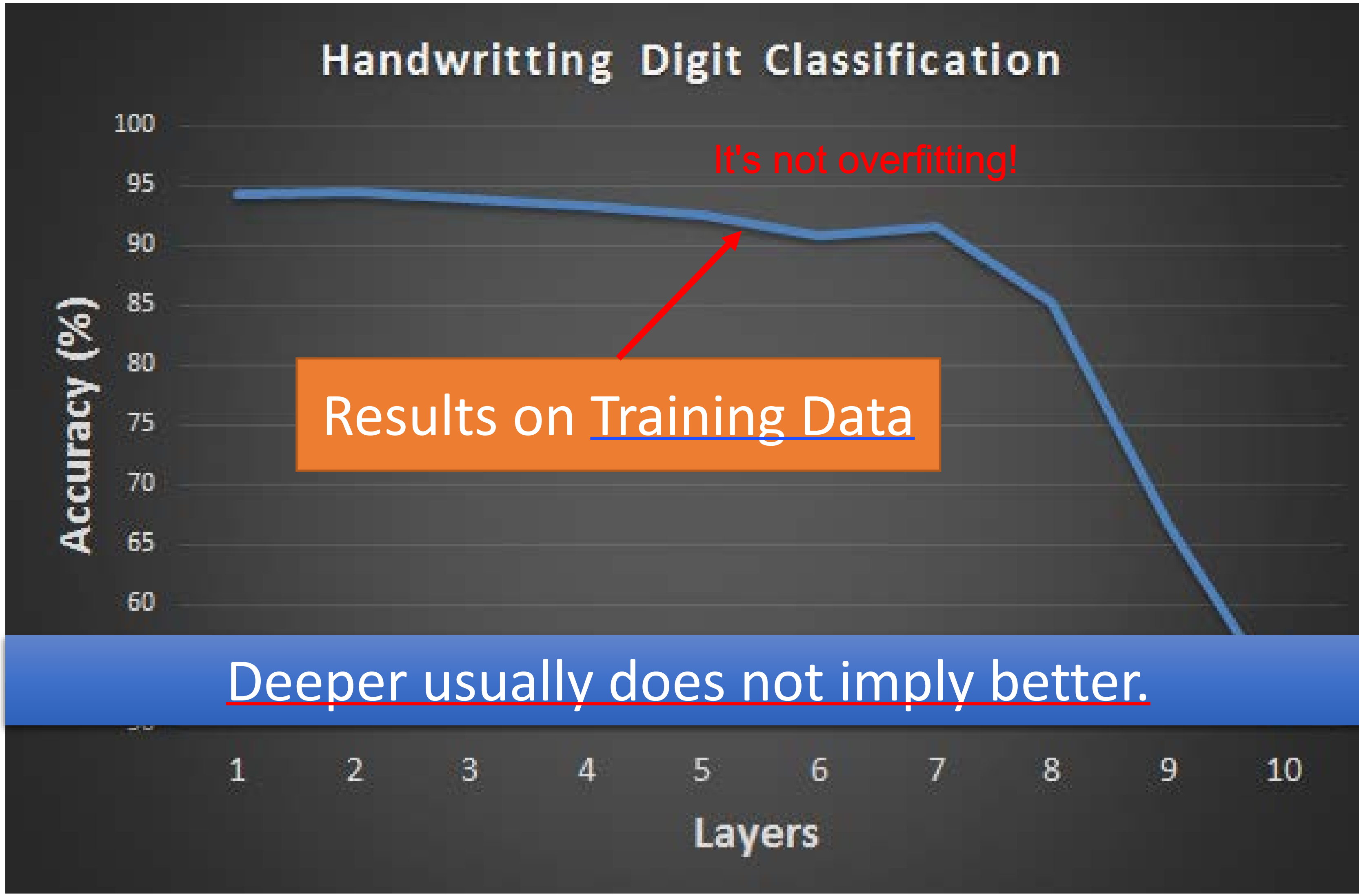
Good Results on Testing Data?

YES

Good Results on Training Data?

YES

Hard to get the power of Deep …



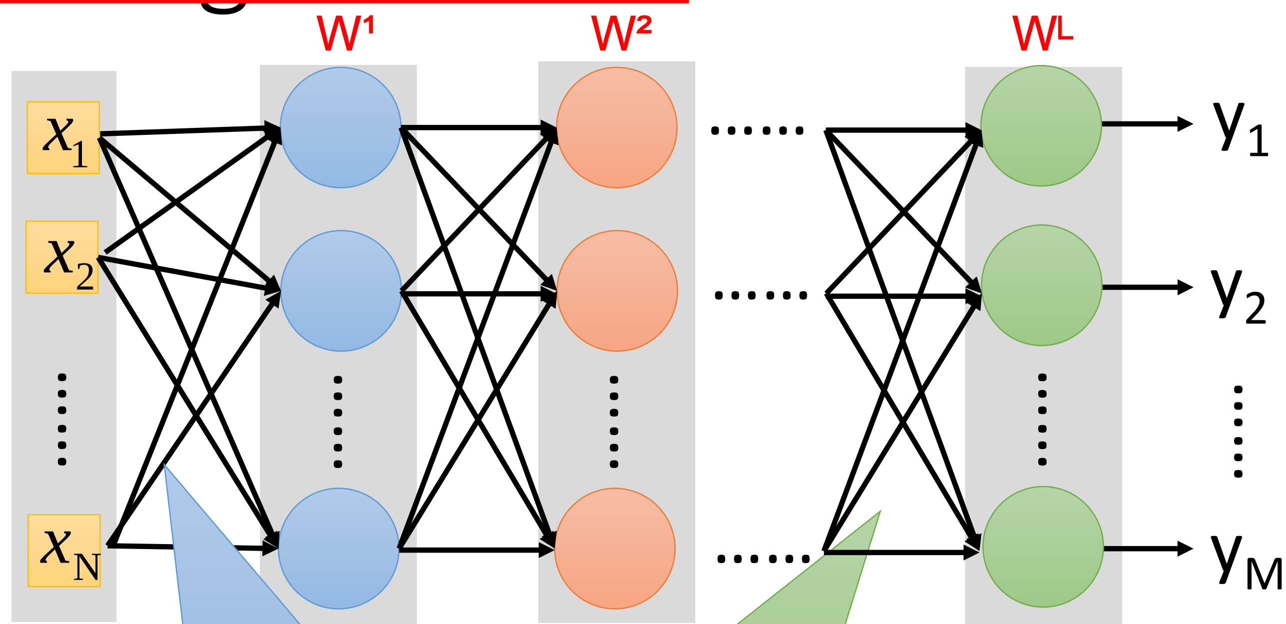
This problem comes from sigmoid function.

$$\sigma(W^L \cdots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \cdots + b^L)$$

Too inside

Vanishing Gradient Problem

Assuming that the learning rates of each layer are the same.



∵ Smaller gradients ← Larger gradients

Why? Next page

∴ Learn very slow | Learn very fast

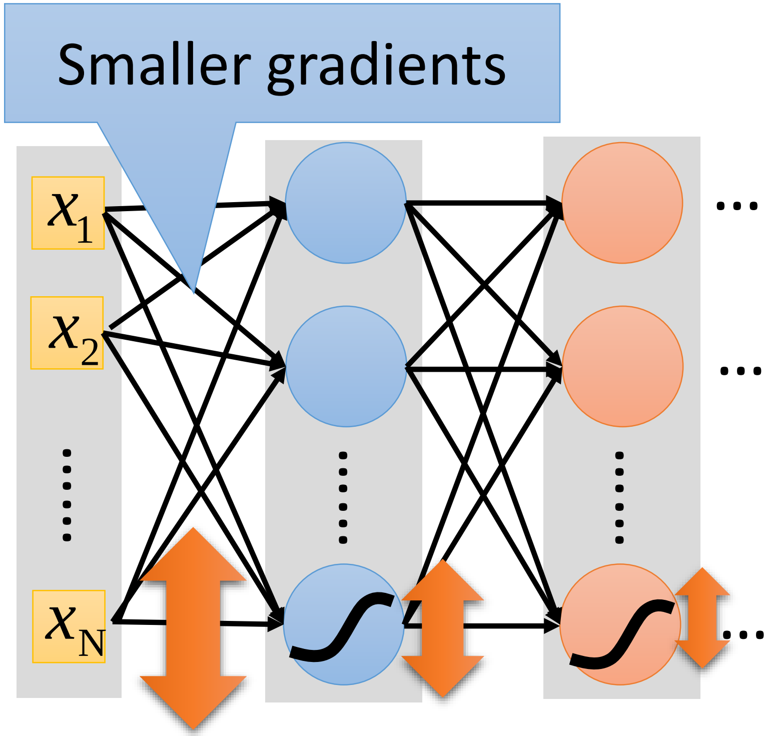
Almost random | Already converge

Not cool

based on random!?

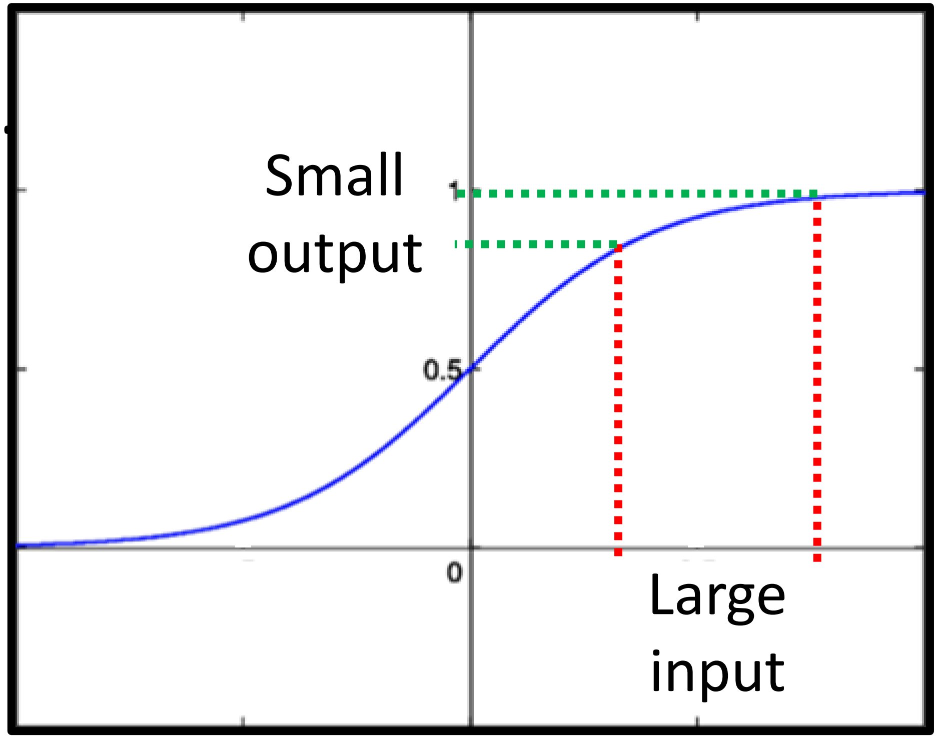
We might be able to solve this problem using dynamic rate,
but it will be much easier that we just change the activation function.
ReLU

Vanishing Gradient Problem



$+\Delta w$ ↘ ⇒ gradient ↗

The change will decay. (ΔW ↘)



Intuitive way to compute the derivatives …

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

↘ ↗

ReLU

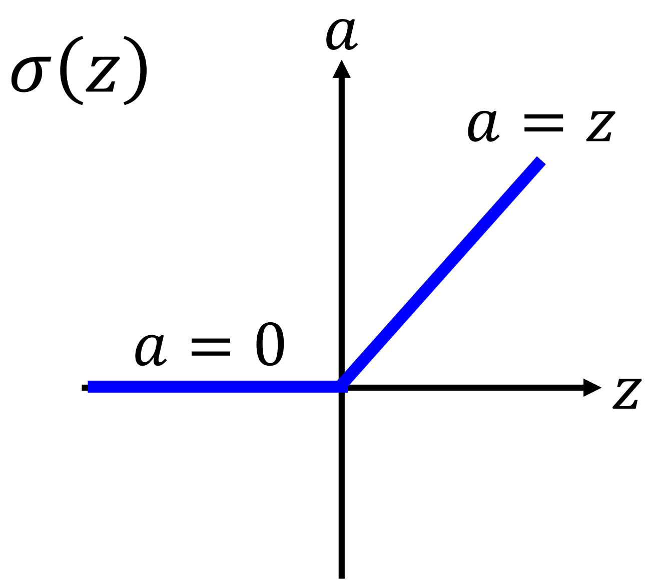
If input < 0, then output = 0

If input ≥ 0, then output = input

Cons:
Dead ReLU problem
(Once we stop a neuron, it will be very hard to reopen it.)

Solution on P.12

- Rectified Linear Unit (ReLU)



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

Reason: Pros:

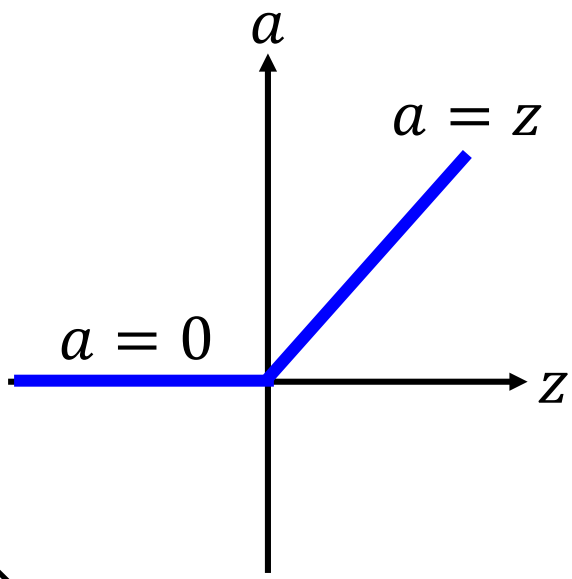
1. Fast to compute
2. Biological reason

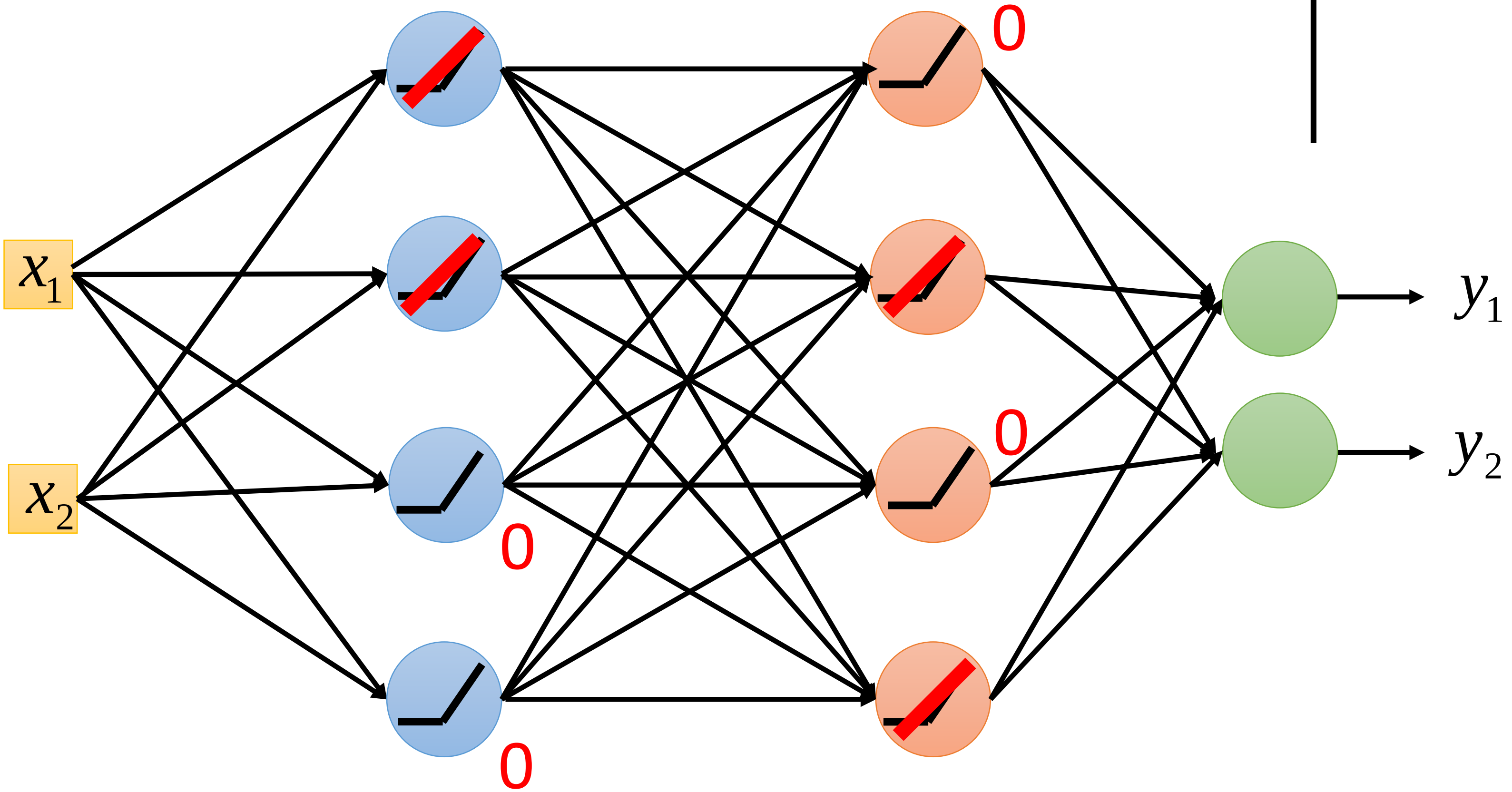
All or none law

≈ Summation of

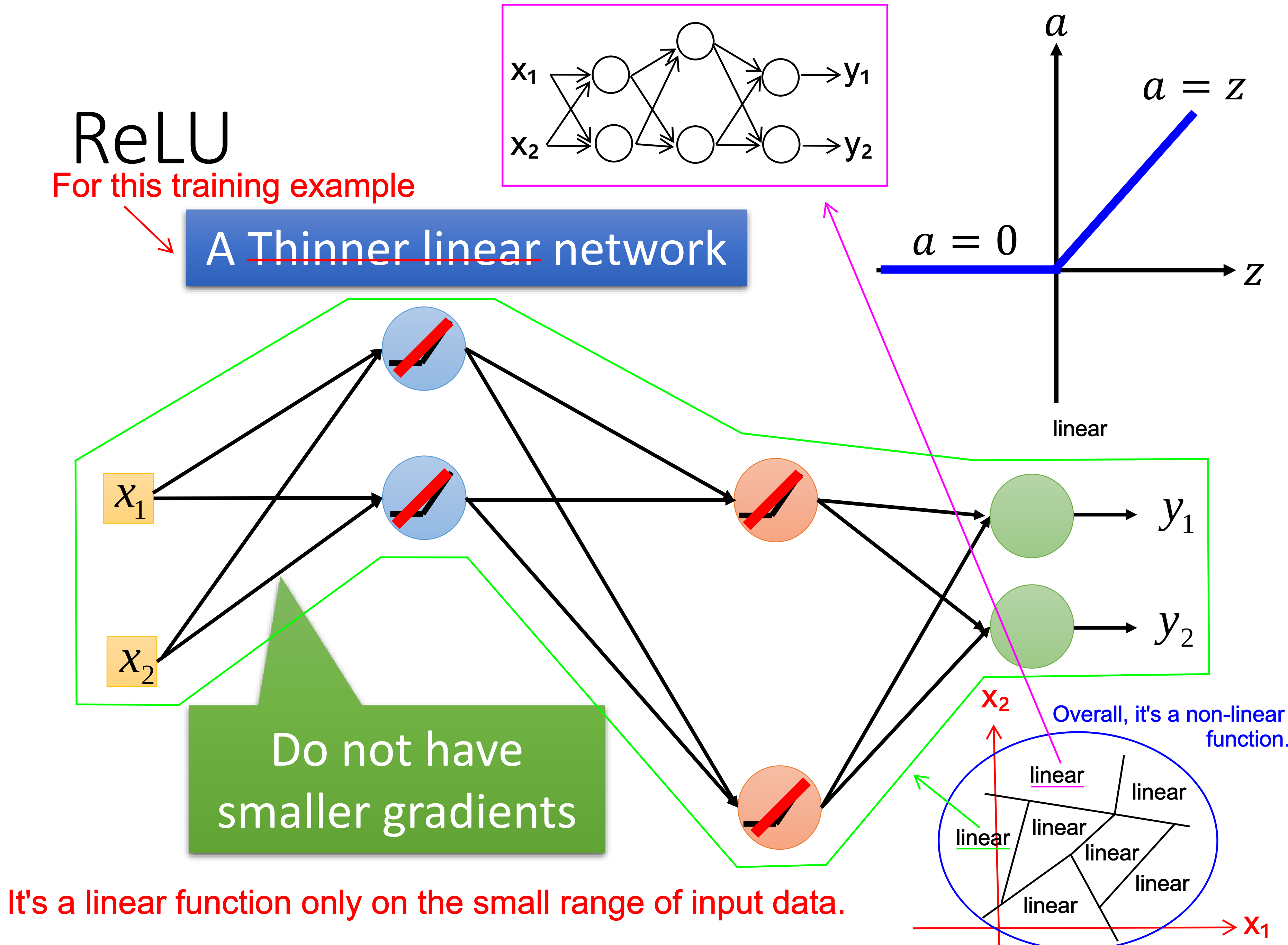
3. Infinite sigmoid with different biases
4. Vanishing gradient problem

ReLU



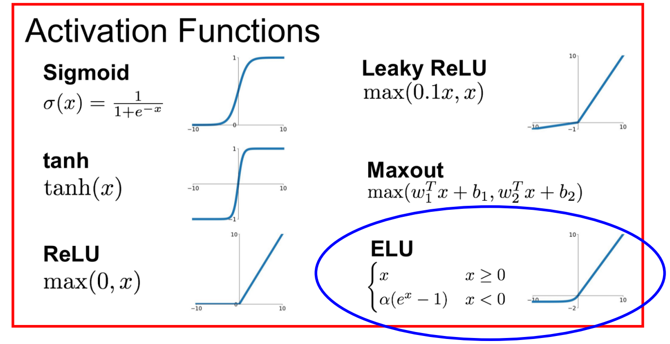


ReLU



For this training example
A ~~Thinner linear~~ network
x_1
x_2
y_1
y_2
Do not have smaller gradients
It's a linear function only on the small range of input data.
a
$a = z$
$a = 0$
z
linear
x_1
x_2
y_1
y_2
Overall, it's a non-linear function.
linear
linear
linear
linear
linear
linear
linear
x_1
x_2

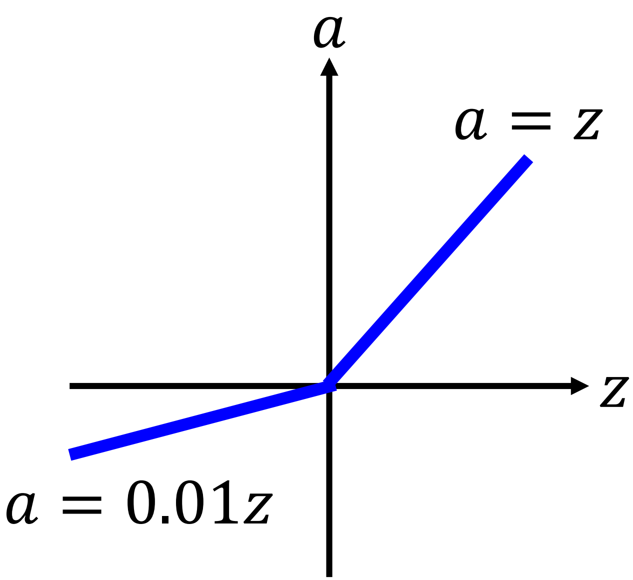
Common activation function:



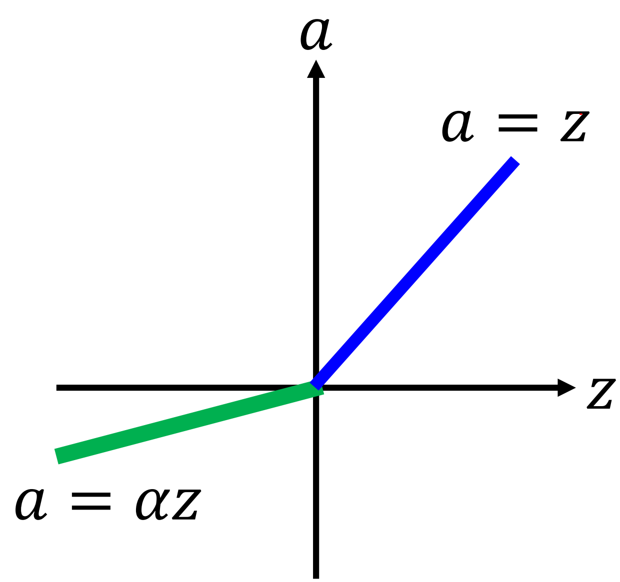
ReLU - variant

Very famous: Exponential Linear Unit (ELU)

$Leaky\ ReLU$



$Parametric\ ReLU$



α also learned by gradient descent

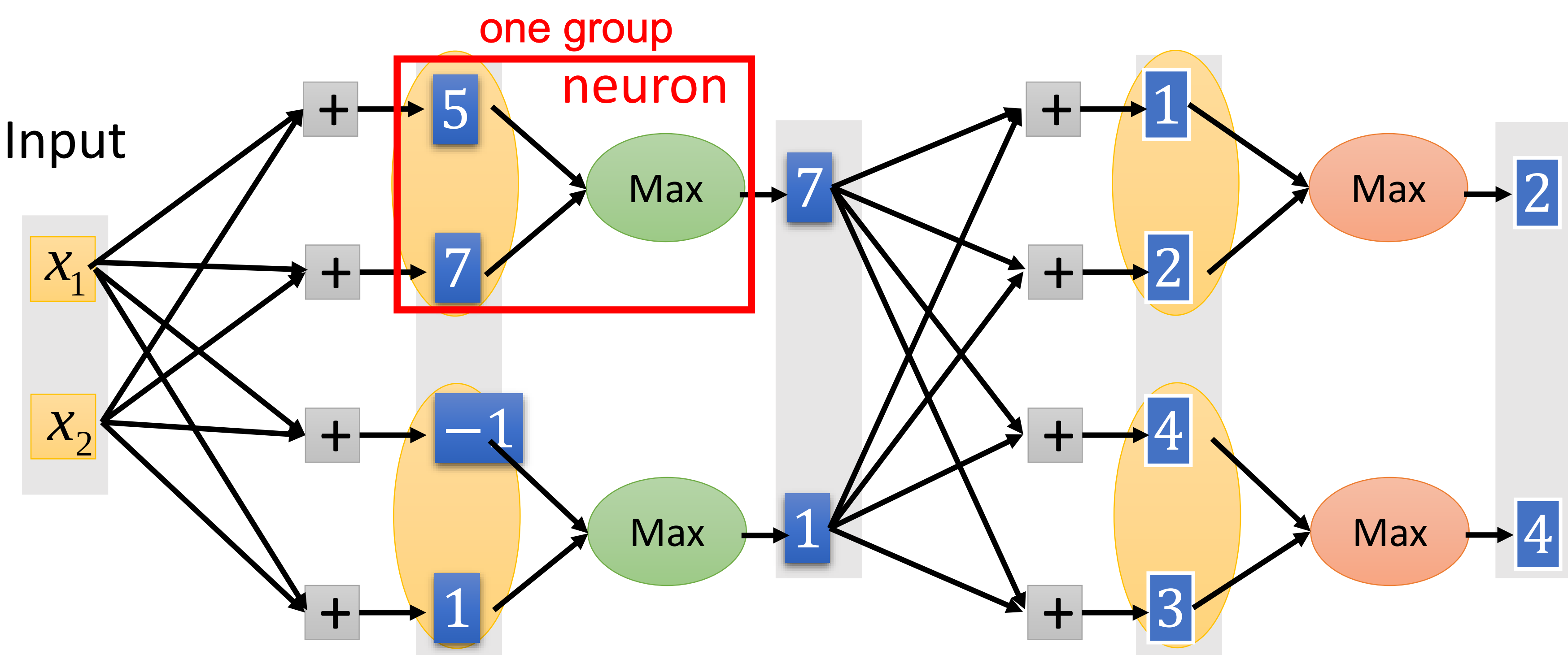
Maxout

Next page

ReLU is a special cases of Maxout

The amount of weights will increase #elements in a group.

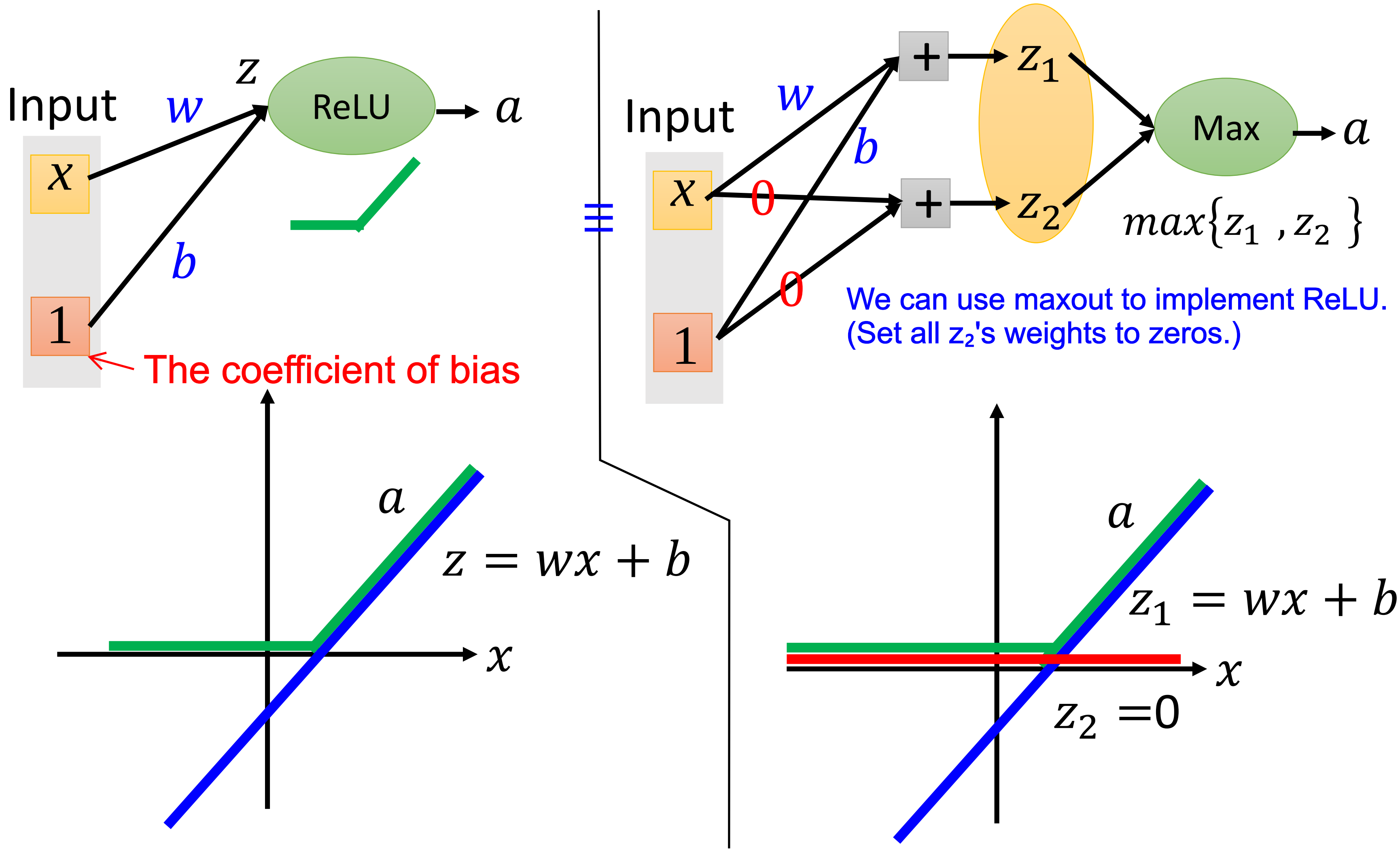
- Learnable activation function [Ian J. Goodfellow, ICML'13]



You can have more than 2 elements in a group.

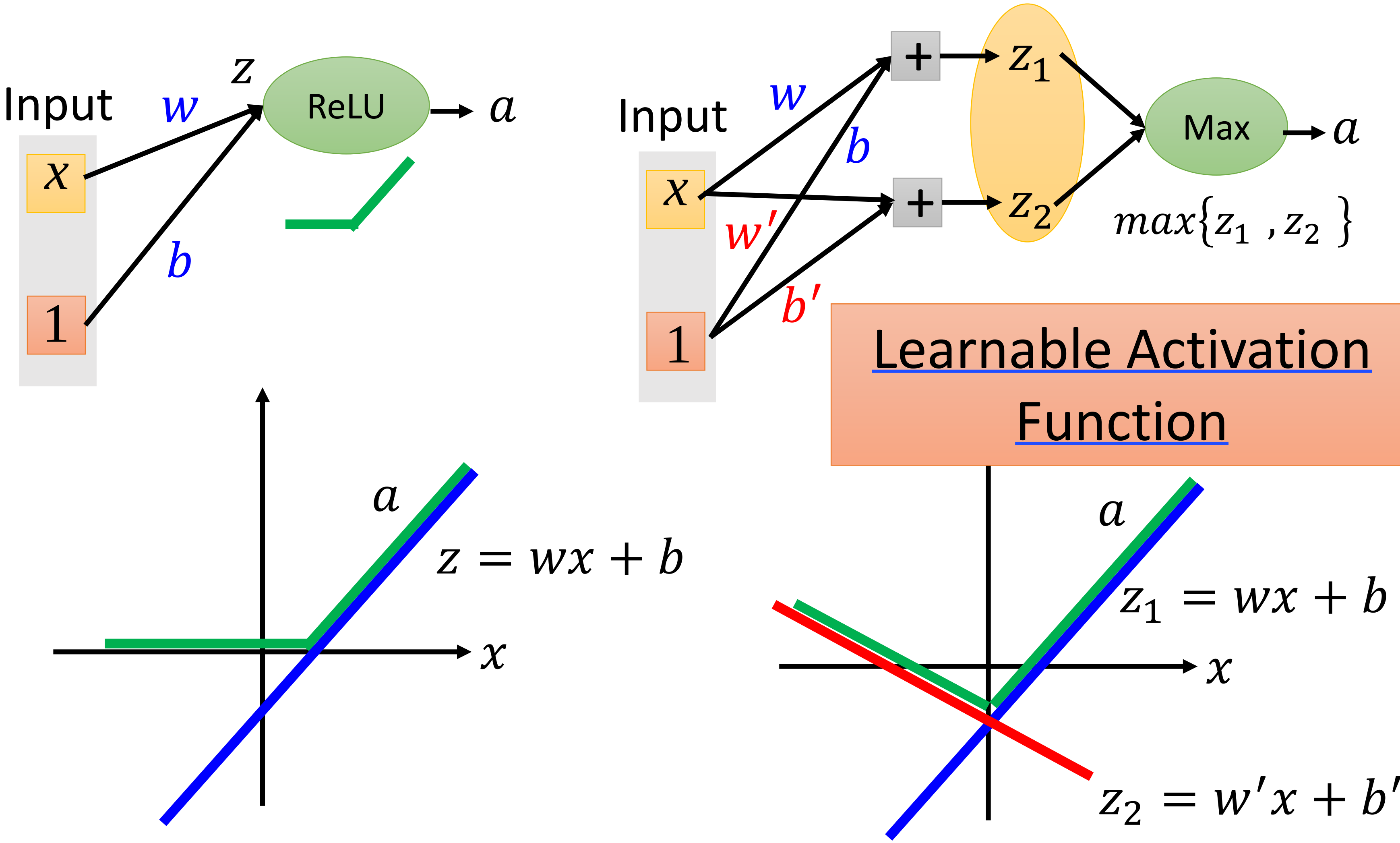
Maxout

ReLU is a special cases of Maxout



Maxout

More than ReLU

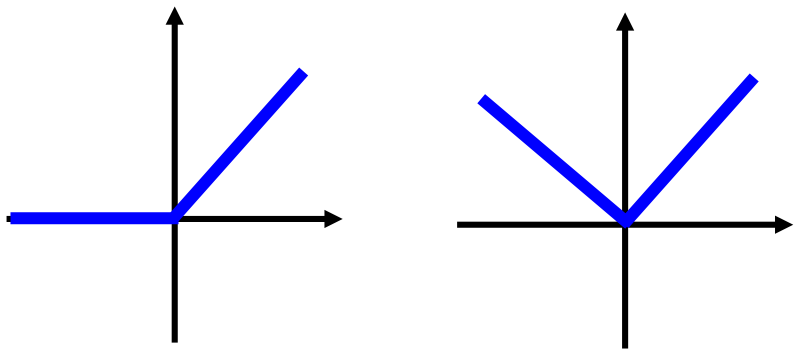


Learnable Activation Function

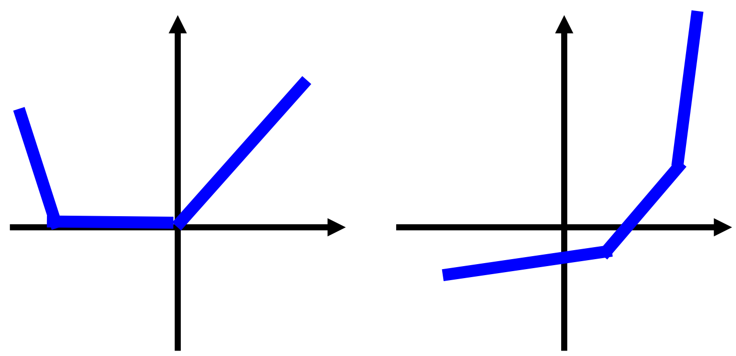
Maxout

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

2 elements in a group



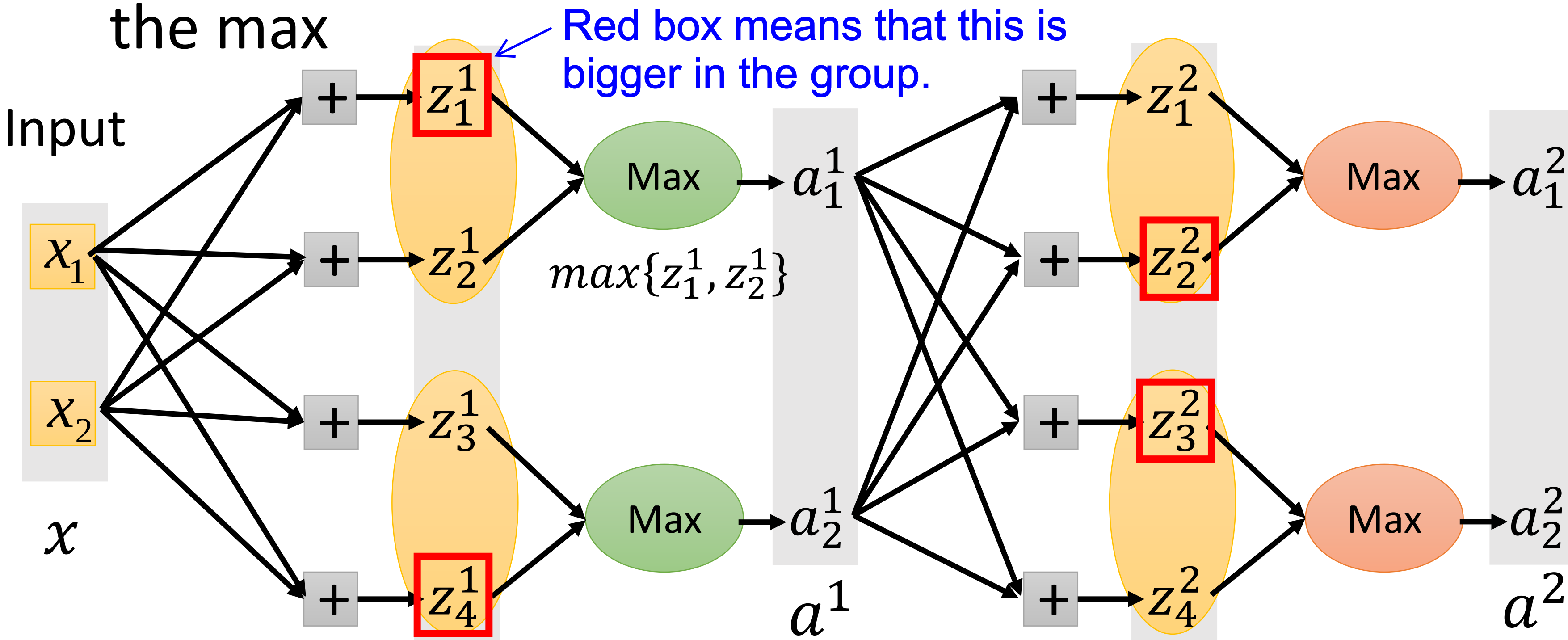
3 elements in a group



Can we still use gradient descent to train when there are "max" operations? (Next page)

Maxout - Training

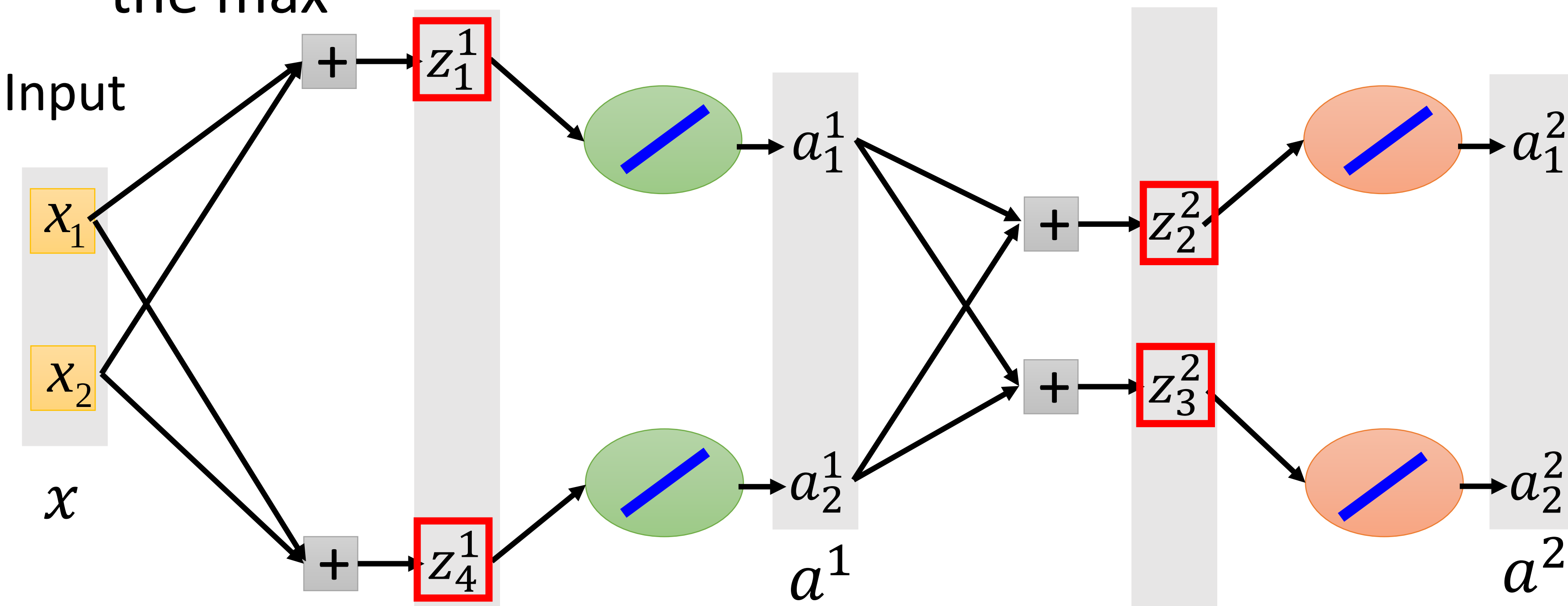
- Given a training data x, we know which z would be the max



Yes, we can. It's just that the neurons that we train each time may be different.

Maxout - Training

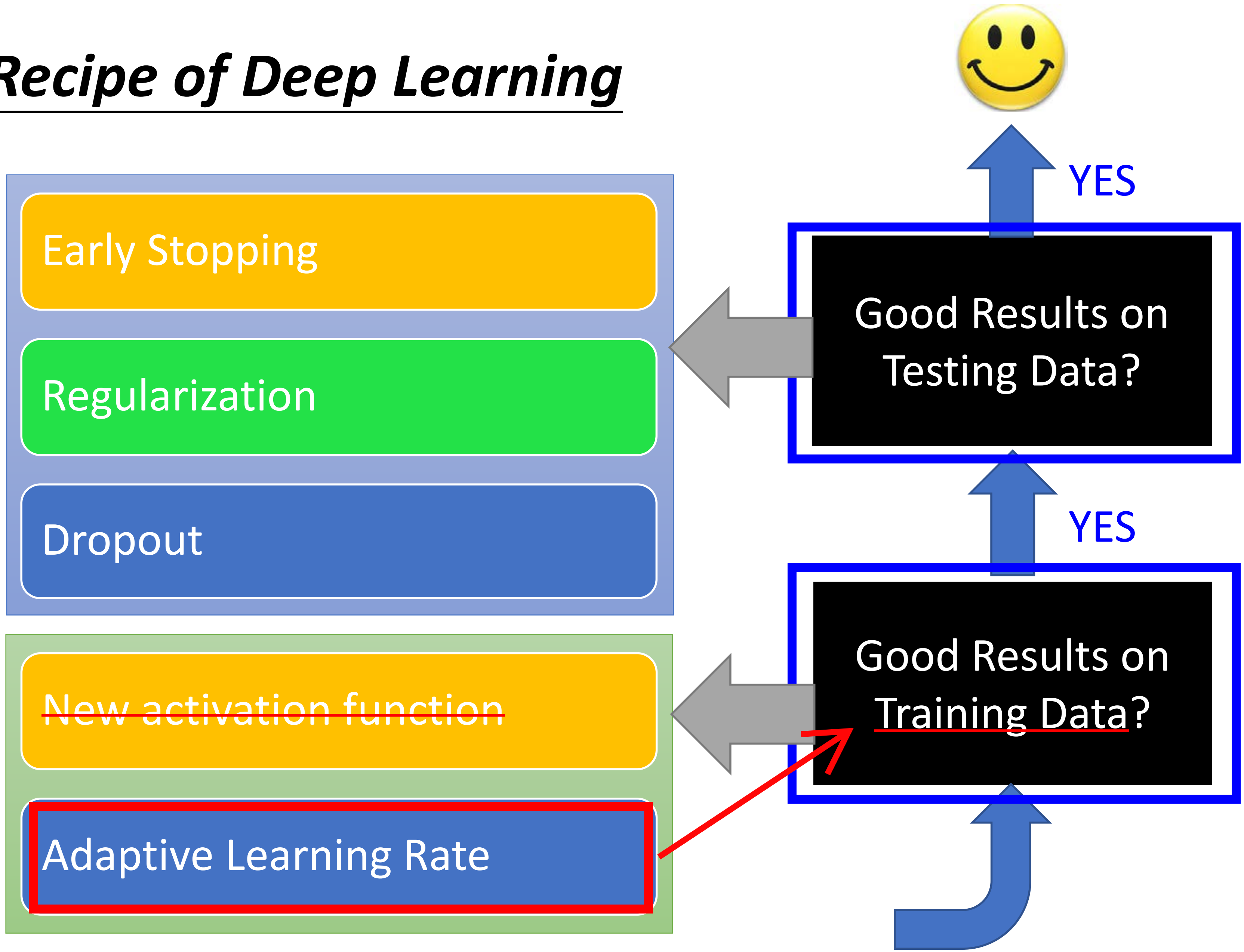
- Given a training data x, we know which z would be the max



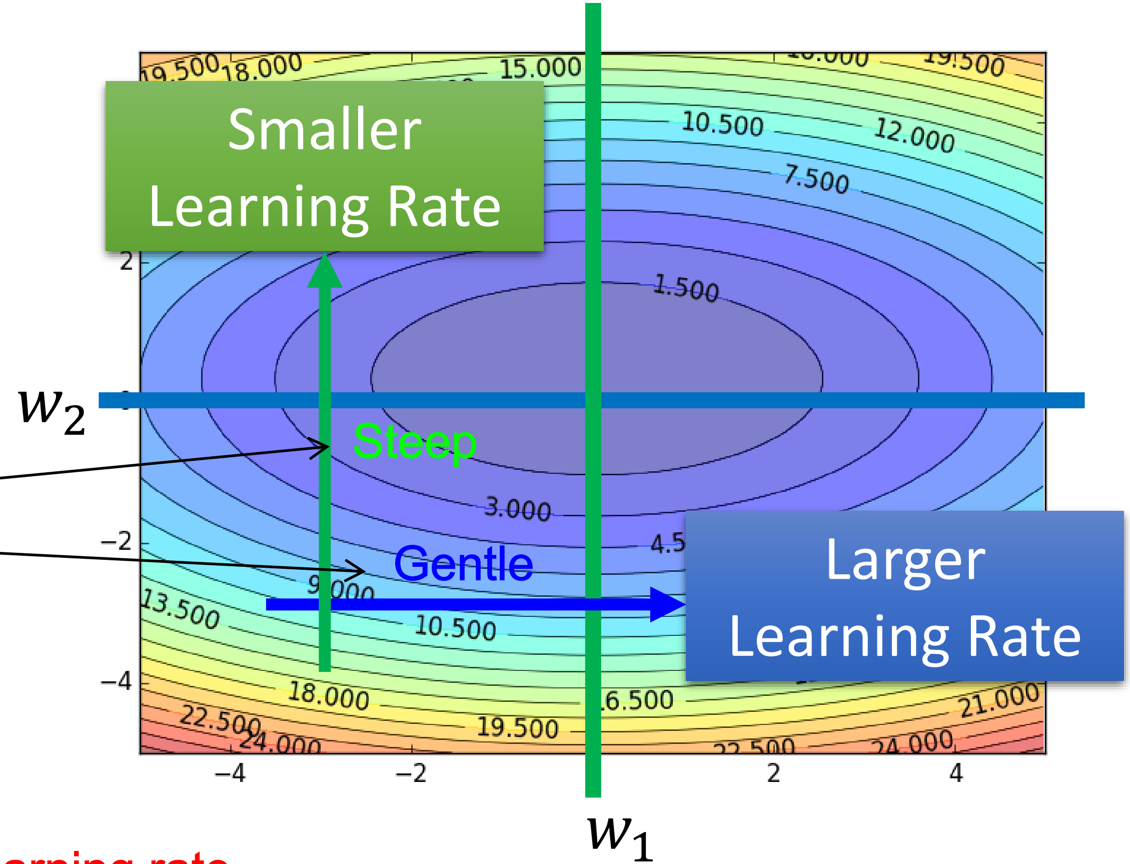
- Train this thin and linear network

Different thin and linear network for different examples

Recipe of Deep Learning



Review



Different directions, different slope

Adagrad

Initial learning rate

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

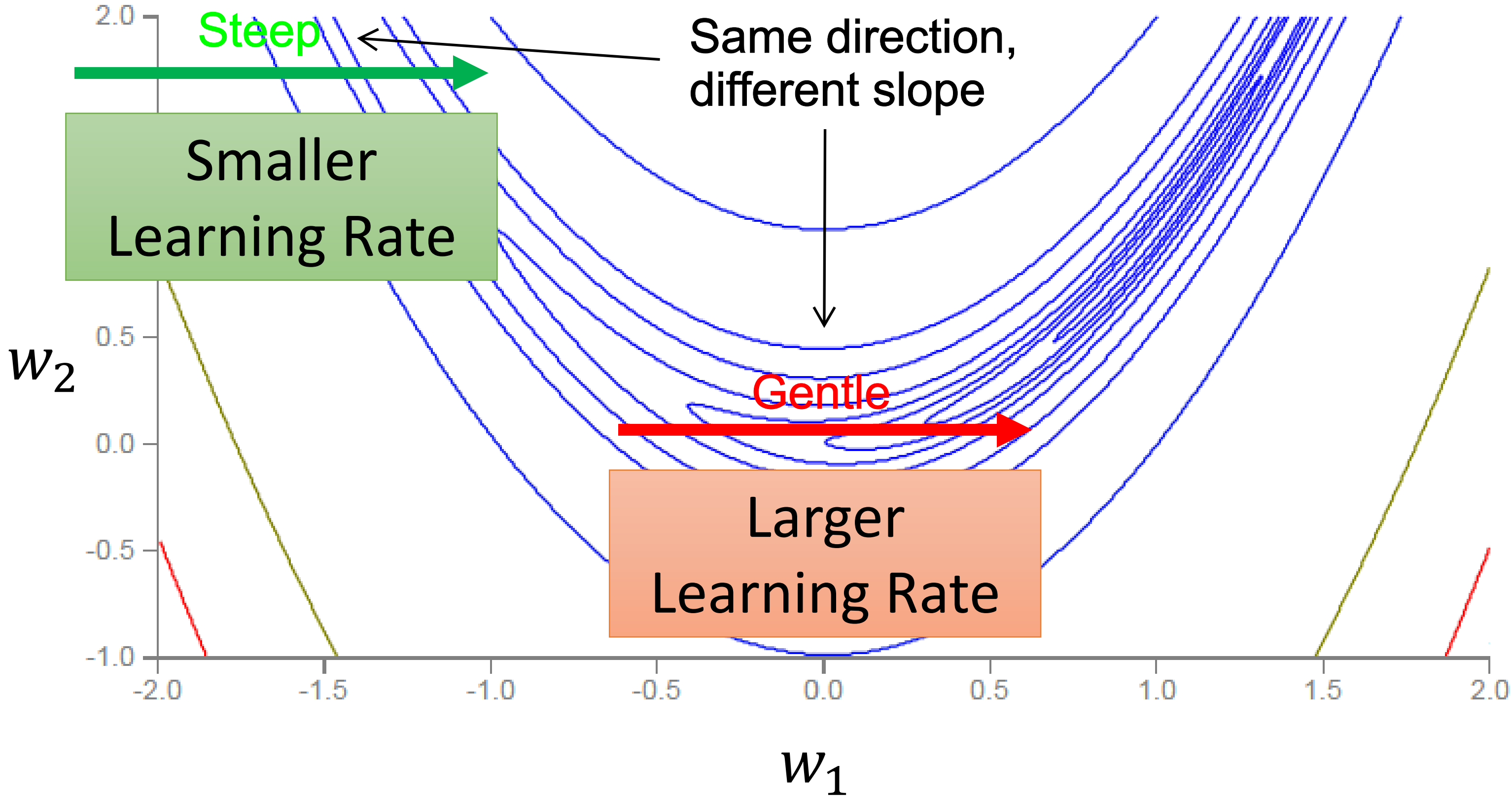
Use first derivative to estimate second derivative

It works when the second derivative is very stable.

RMSProp

Error Surface can be very complex when training NN.

⇒ The second derivative won't be very stable.



RMSProp

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

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

α: decay coefficient

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

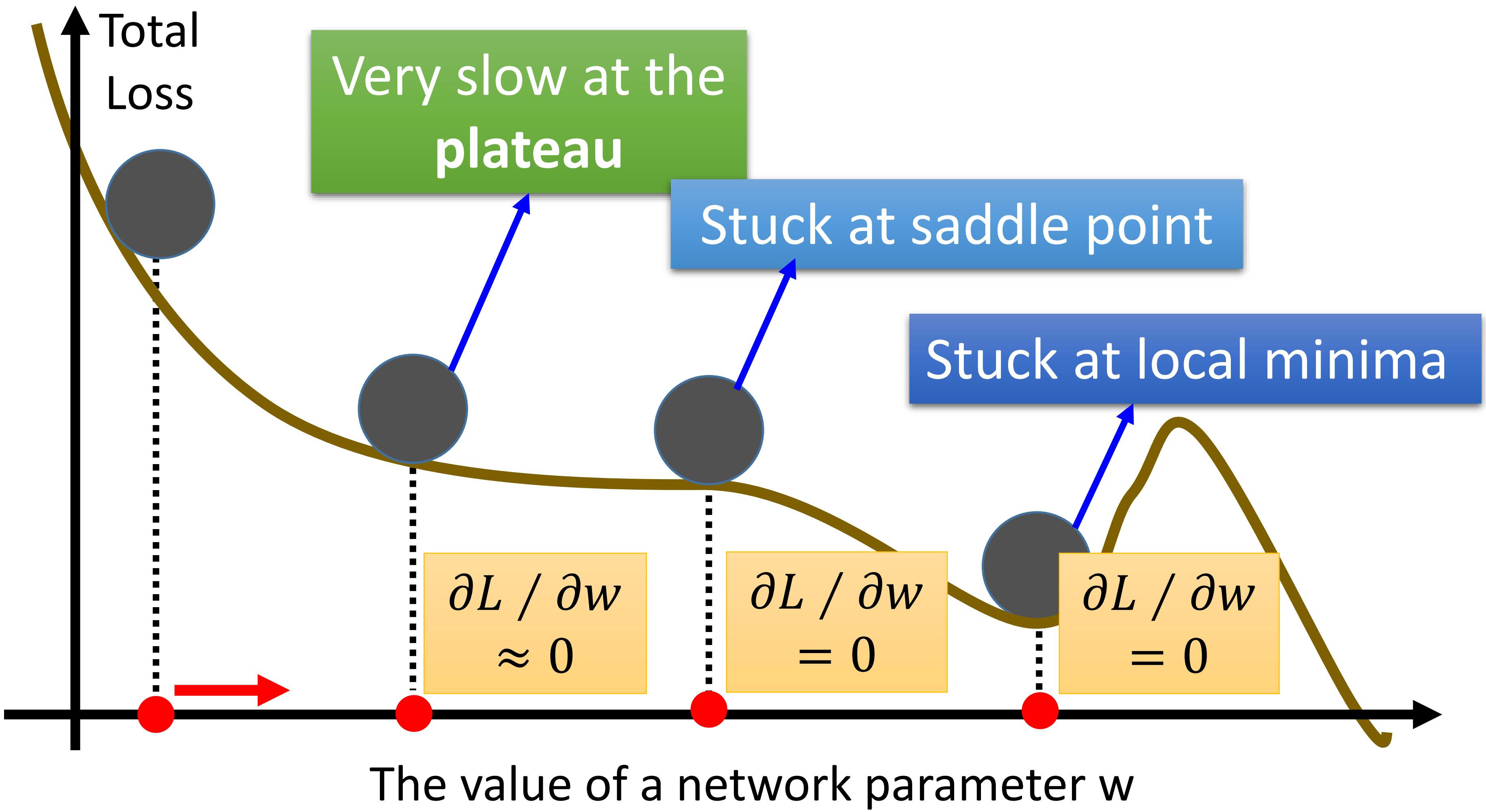
$$w^3 \leftarrow w^2 - \frac{\eta}{\sigma^2} g^2 \qquad \sigma^2 = \sqrt{\alpha(\sigma^1)^2 + (1-\alpha)(g^2)^2}$$

$$\vdots$$

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

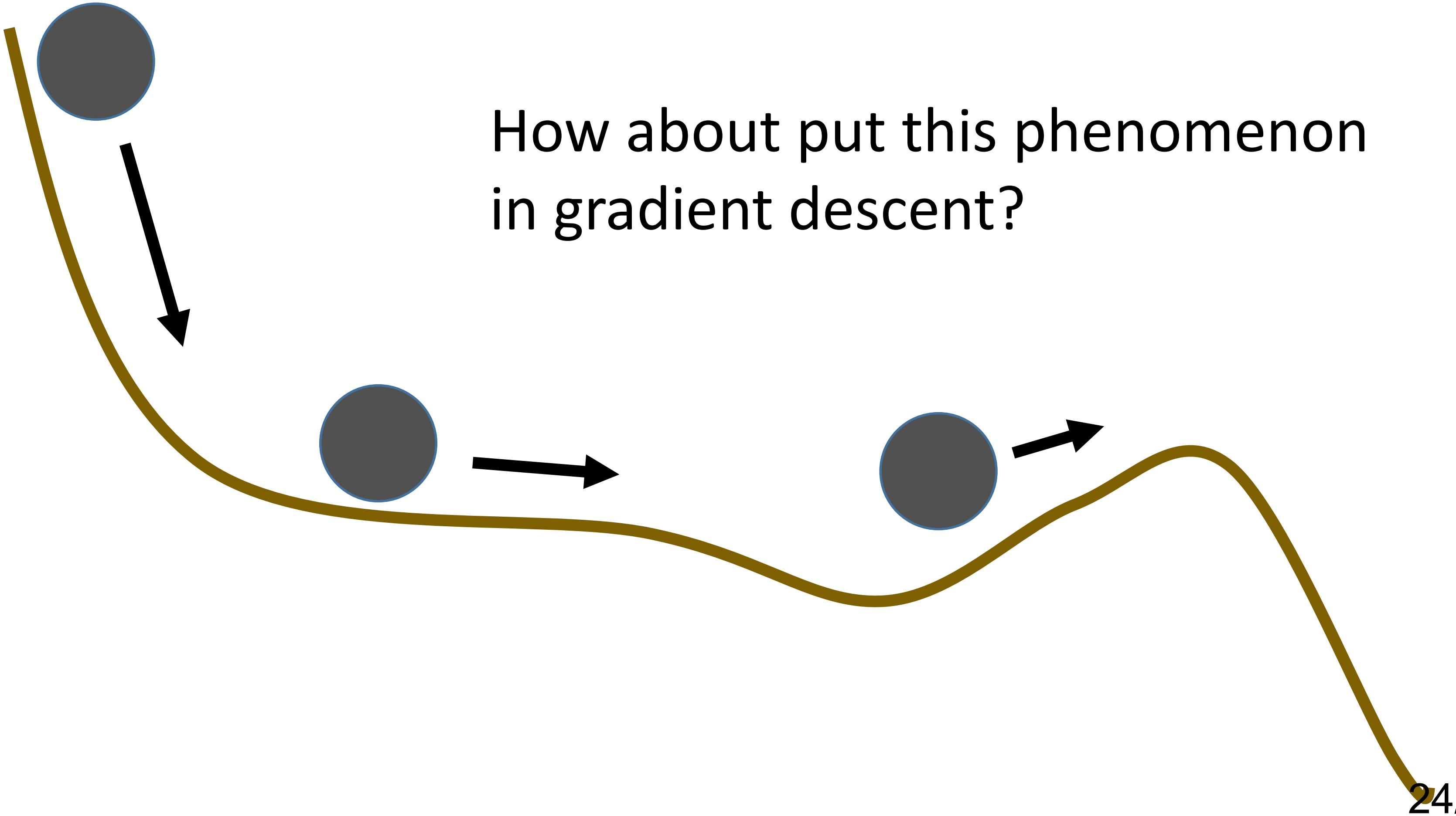
Root Mean Square of the gradients with previous gradients being decayed

Hard to find optimal network parameters



In physical world ……

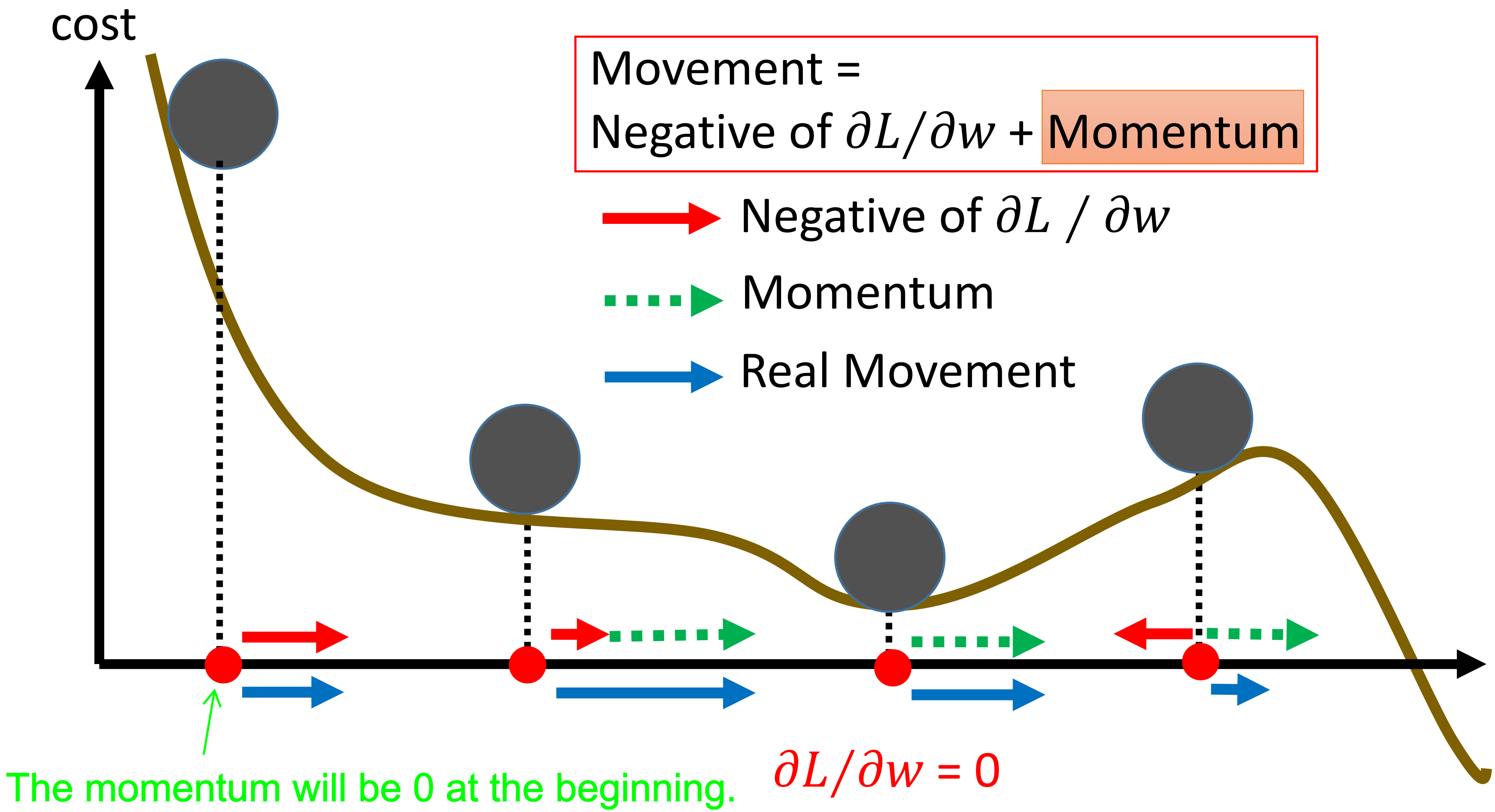
- Momentum There will be inertia.



How about put this phenomenon in gradient descent?

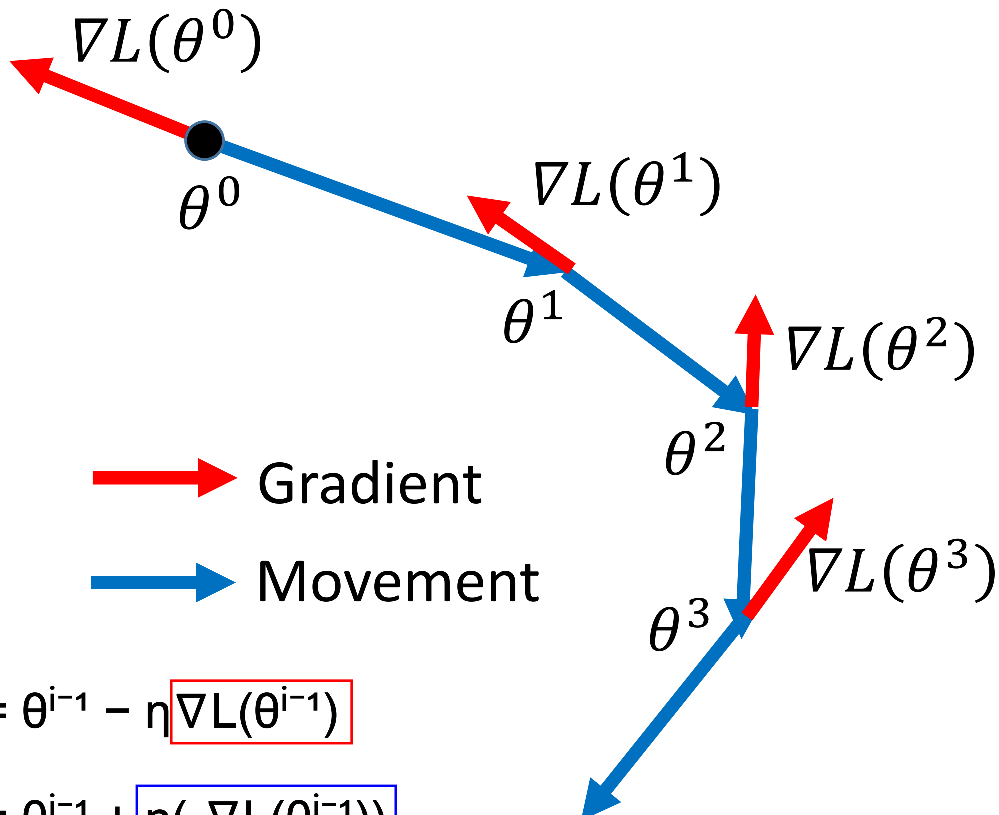
Momentum

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



Review: Vanilla Gradient Descent

Normal



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

$$\theta^i = \theta^{i-1} - \eta\boxed{\nabla L(\theta^{i-1})}$$

$$\theta^i = \theta^{i-1} + \underbrace{\boxed{\eta(-\nabla L(\theta^{i-1}))}}_{V^i}$$

$$\boxed{\text{Movement}^i = -\eta\ \text{Gradient}^i}$$

Movement ^{i} = -η Gradient ^{i} + λ Movement $^{i-1}$

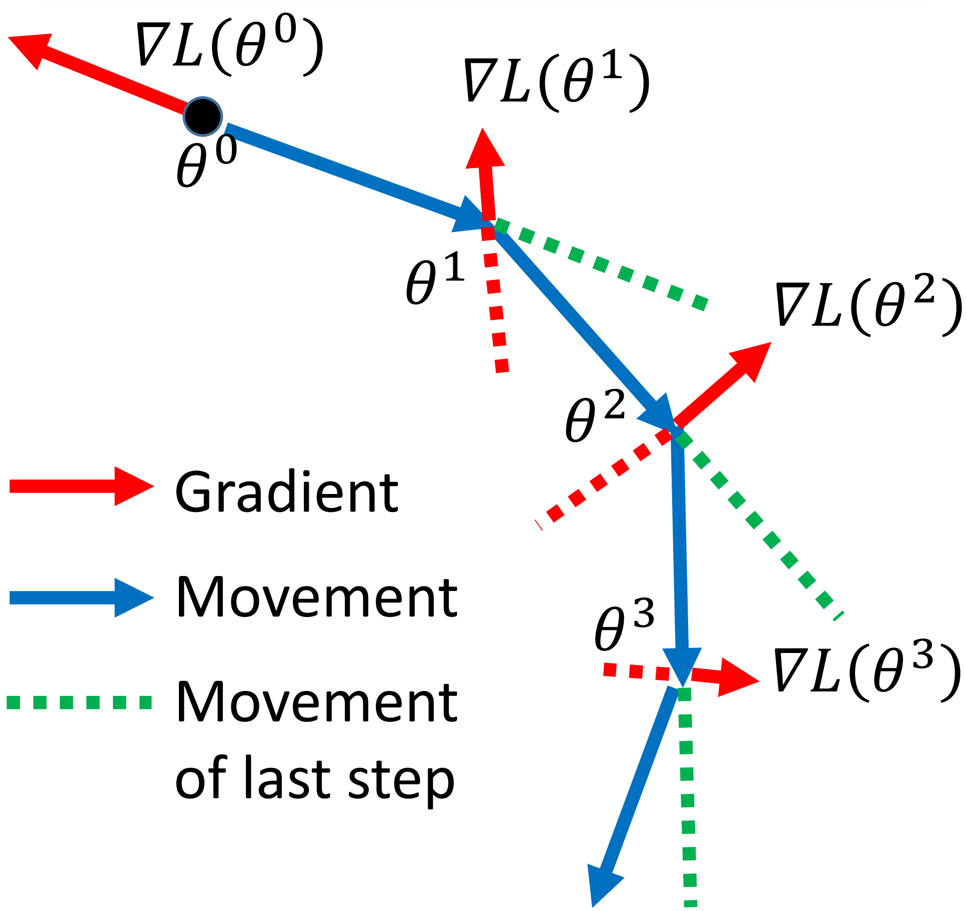
$$\theta^{i} = \theta^{i-1} + \underline{V^{i}}$$

$$= \lambda V^{i-1} - \eta \nabla L(\theta^{i-1})$$

$$\Rightarrow \theta^{i} = \underbrace{\theta^{i-1} - \eta \nabla L(\theta^{i-1})}_{\text{Vanilla}} + \lambda V^{i-1}$$

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.

Why can we only consider the last movement instead of all of the previous movements?
Because considering the last movement IS considering all of the previous movements.

Momentum

Movement: movement of last step minus gradient at present

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

$v^0 = 0$

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

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

⋮

$V^i = -\sum_{k=1}^{i} \lambda^{k-1} \eta \nabla L(\theta^{i-k})$

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

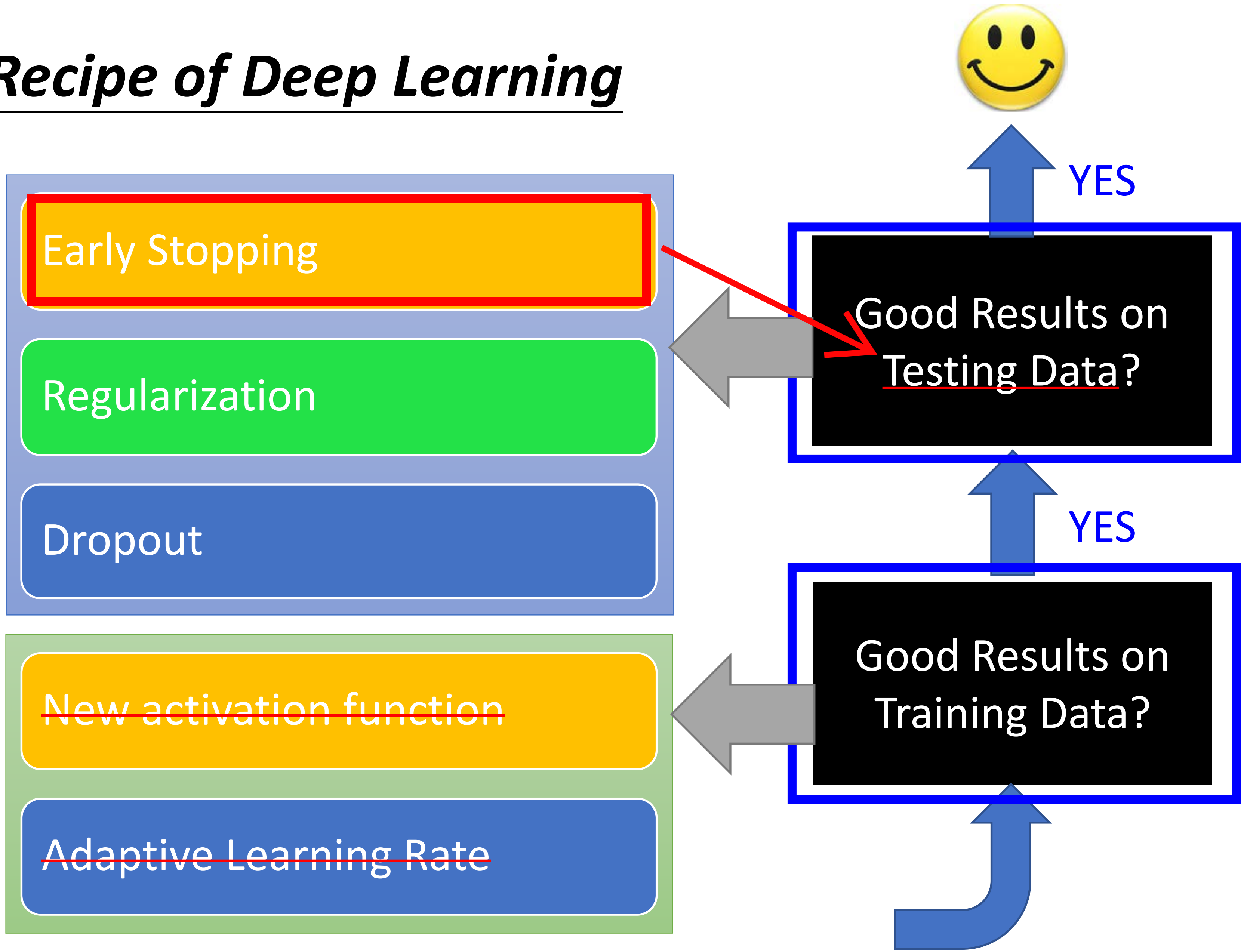
Movement not just based on gradient, but previous movement

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)

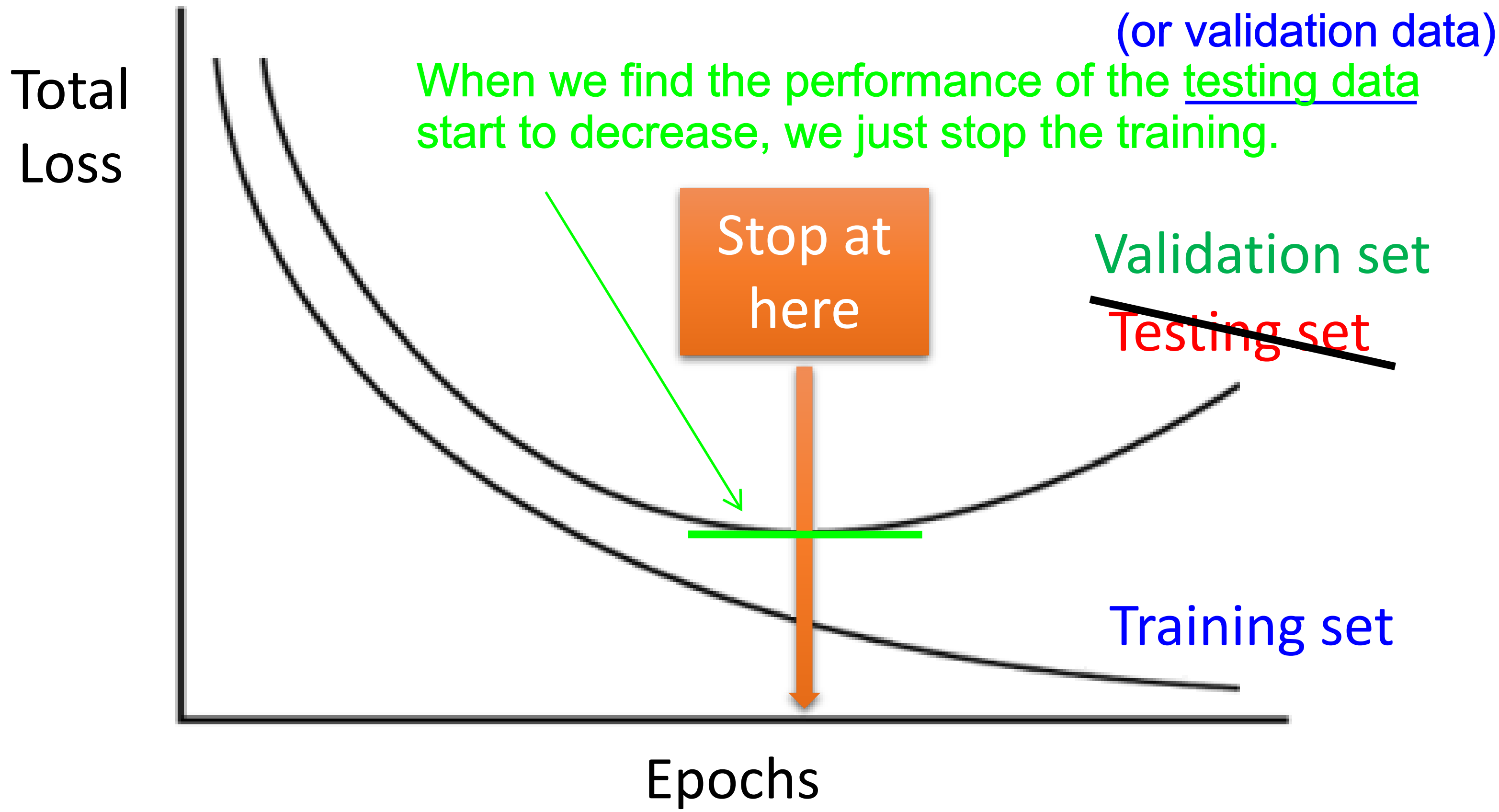
Recipe of Deep Learning



(or validation data)

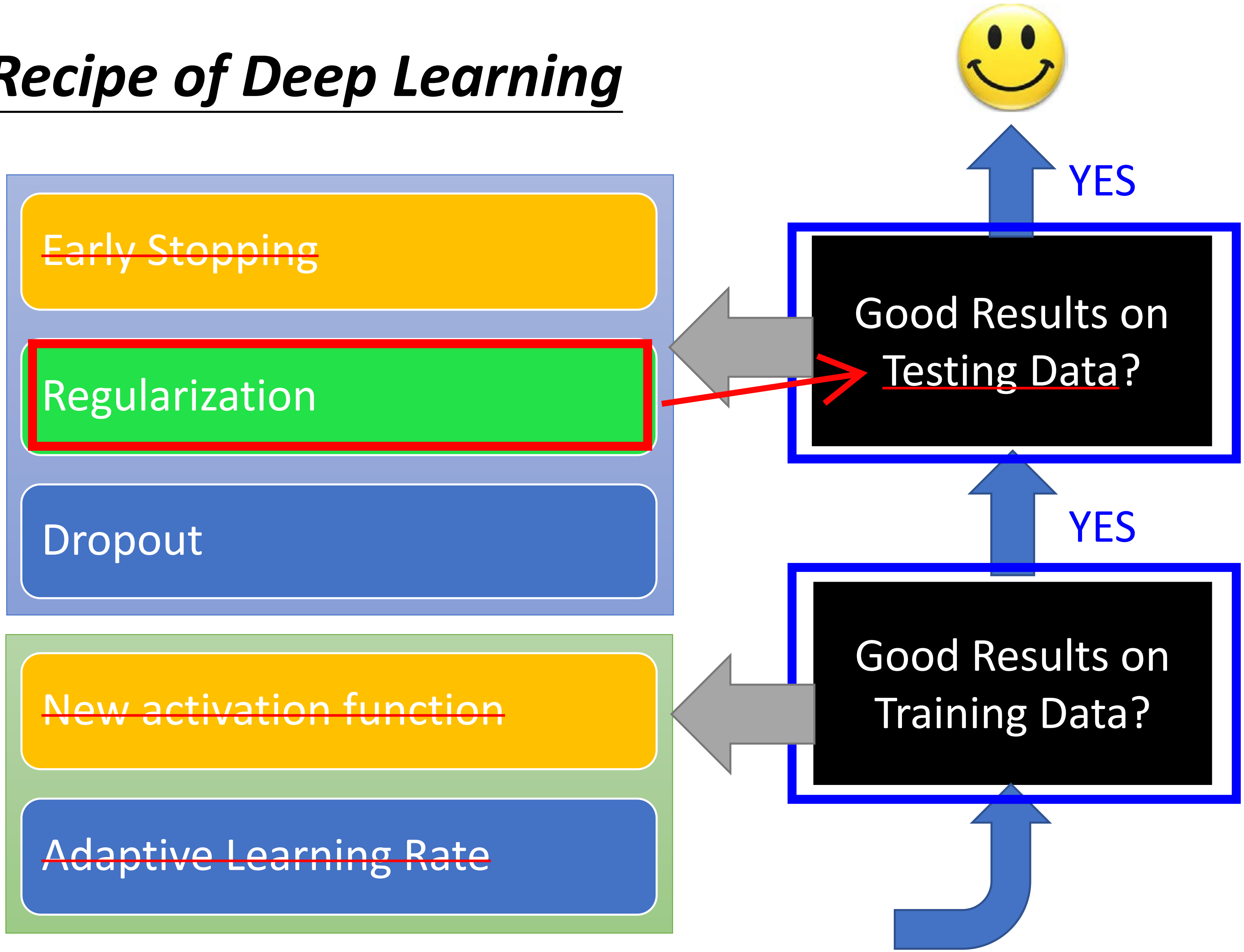
We don't only test the testing data after all of the training is done.
Instead, we test the testing data every epoch of the training data.

Early Stopping



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

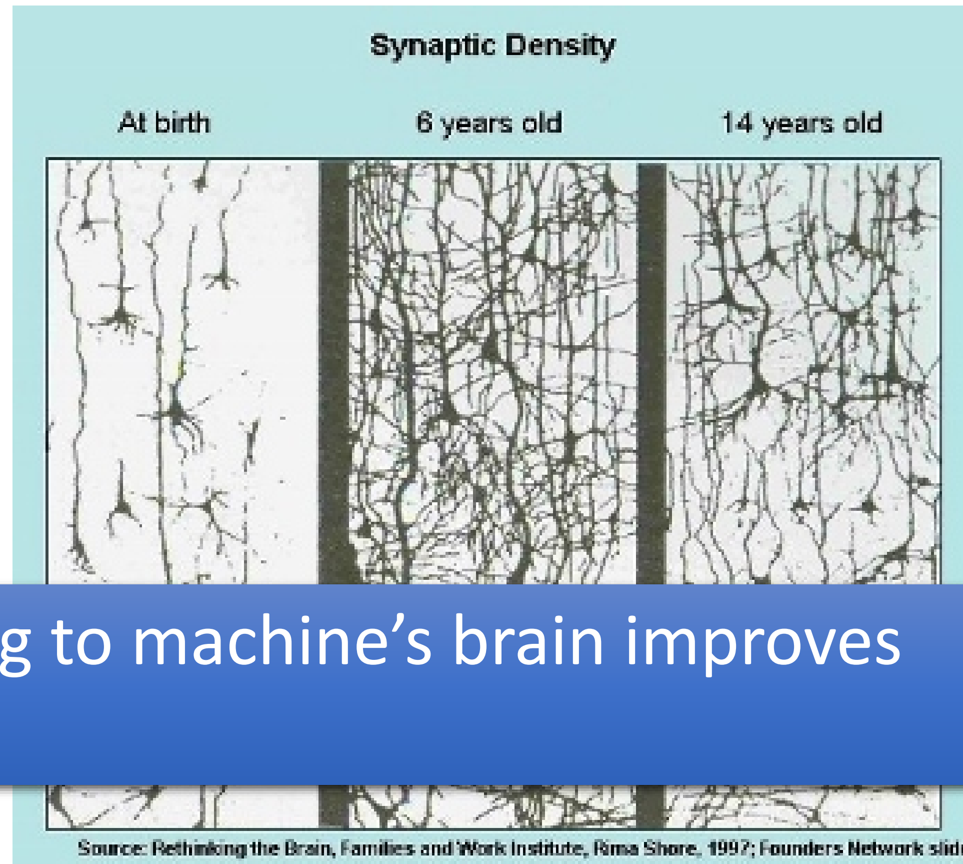
Recipe of Deep Learning



Regularization - Weight Decay

- Our brain prunes out the useless link between neurons.

Decay the links that are not very often to be used.



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

⇒ Weight decay

This page: Conclusion

In DL, regularization has the similar property with early stopping. Because we usually initialize the parameters close to zero, if we use early stopping, the parameters will close to zero normally. (So the regularization is not very essential in DL.)

Regularization

L2: Ridge
L1: Lasso
In-between: Elastic Net

If we didn't add regularization, then when the gradient of this weight is equal to zero (it means that this weight is irrelevant to this training batch), the weight will stay unchanged. But if we use weight decay, the weight will continue to decrease even when its gradient is zero.

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

2-norm: multiply (1−ηλ)

1-norm: subtract $\eta\lambda \cdot sgn(w^t)$

weight decay

$$L'(\theta) = L(\theta) + \lambda \frac{1}{2}\|\theta\|_2 \rightarrow \text{Regularization term}$$

Very easy to let the weight be equal to zero. (Decrease the complexity of the model, but it's pretty unstable compared to 2-norm.)

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

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

L2 regularization:

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

(usually not consider biases)

This page:
2-norm

Regularization

L2 regularization:

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

- New loss function to be minimized

New gradient

Old gradient

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

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

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

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

If we add the regularization, then when we update the weight, we will decay the weight before subtracting the gradient.

Using multiplication

Weight Decay

Closer to zero

It's a number that smaller than 1 but very close to 1.

This page:
1-norm

Regularization

L1 regularization:

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

- New loss function to be minimized

$$L'(\theta) = L(\theta) + \lambda \frac{1}{2} \|\theta\|_1$$

$$\frac{\partial L'}{\partial w} = \frac{\partial L}{\partial w} + \lambda \, \underline{\operatorname{sgn}(w)}$$

$\operatorname{sgn}(w) = \begin{cases} 1, & \text{if } w > 0 \\ 0, & \text{if } w = 0 \\ -1, & \text{if } w < 0 \end{cases}$

Update:

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

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

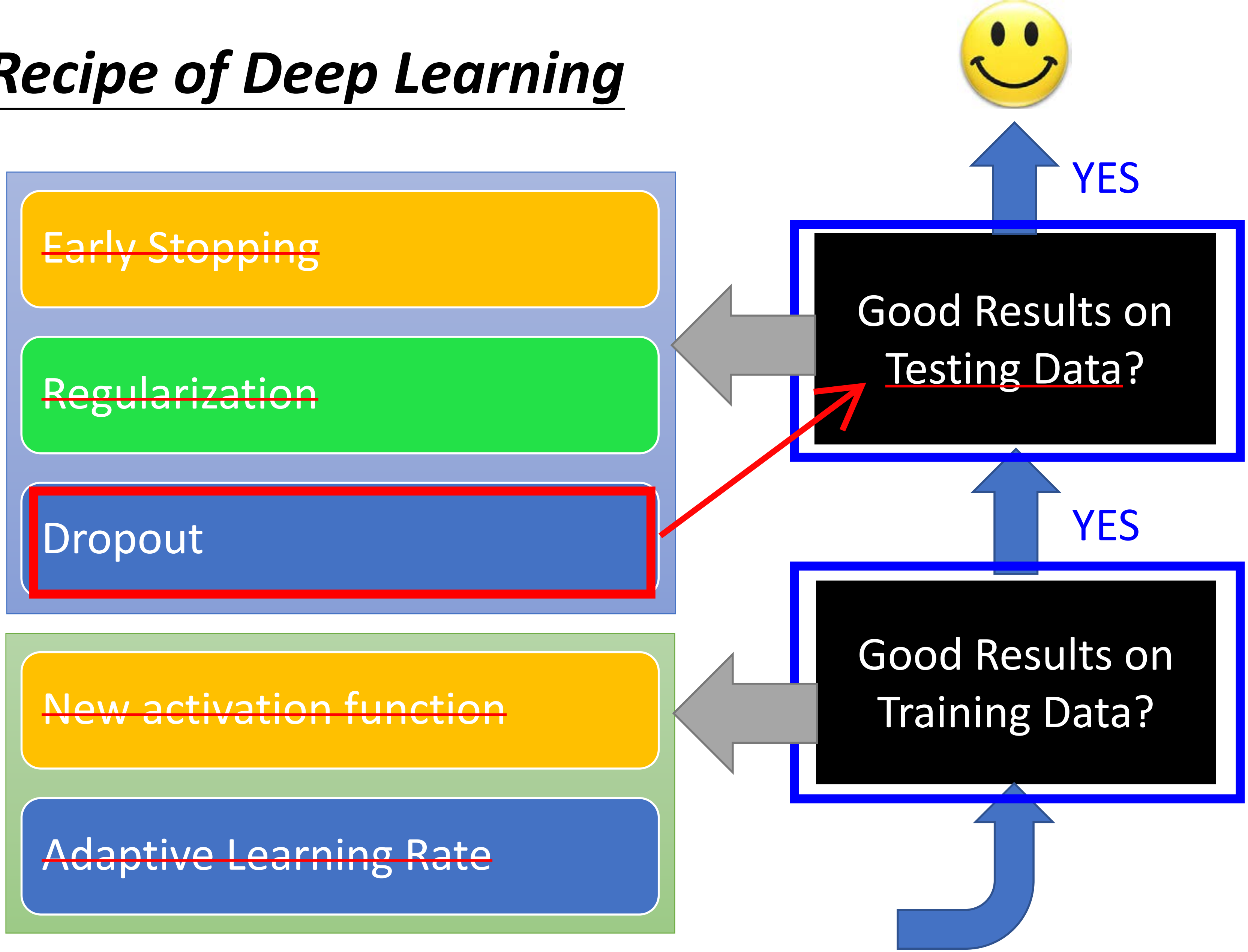
Always <u>delete</u>

Make it close to zero.

Independent with the size of weight itself.
(We just use the sign of it.)

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

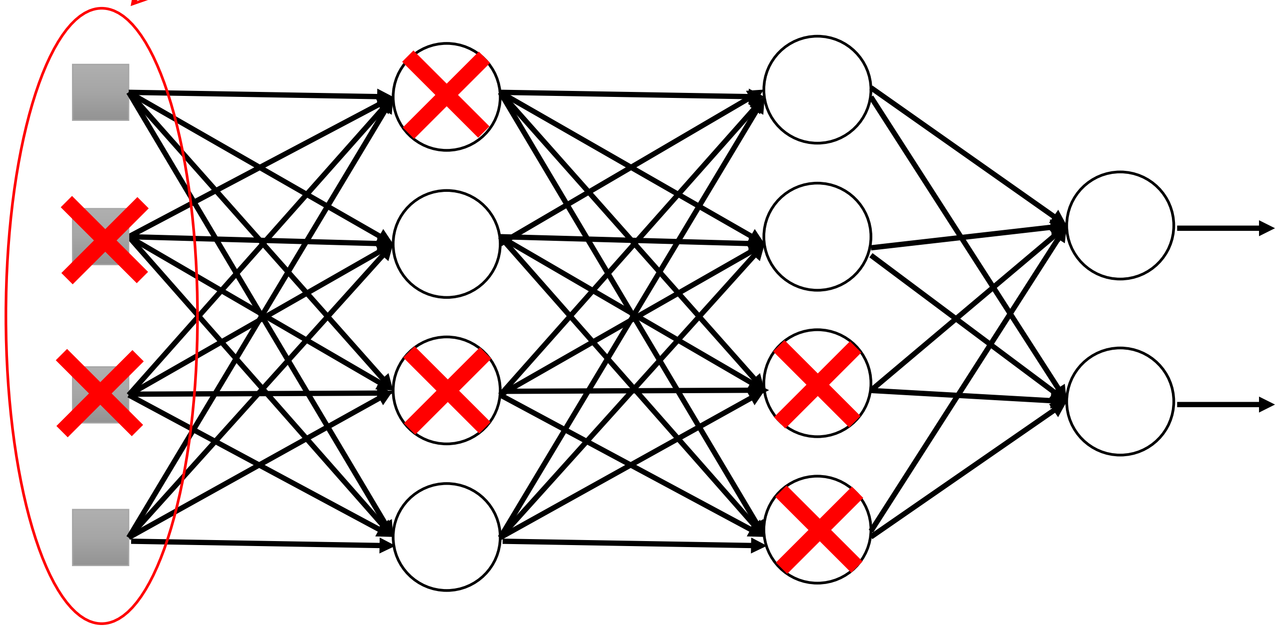
Recipe of Deep Learning



Dropout

We can dropout the input layer as well.

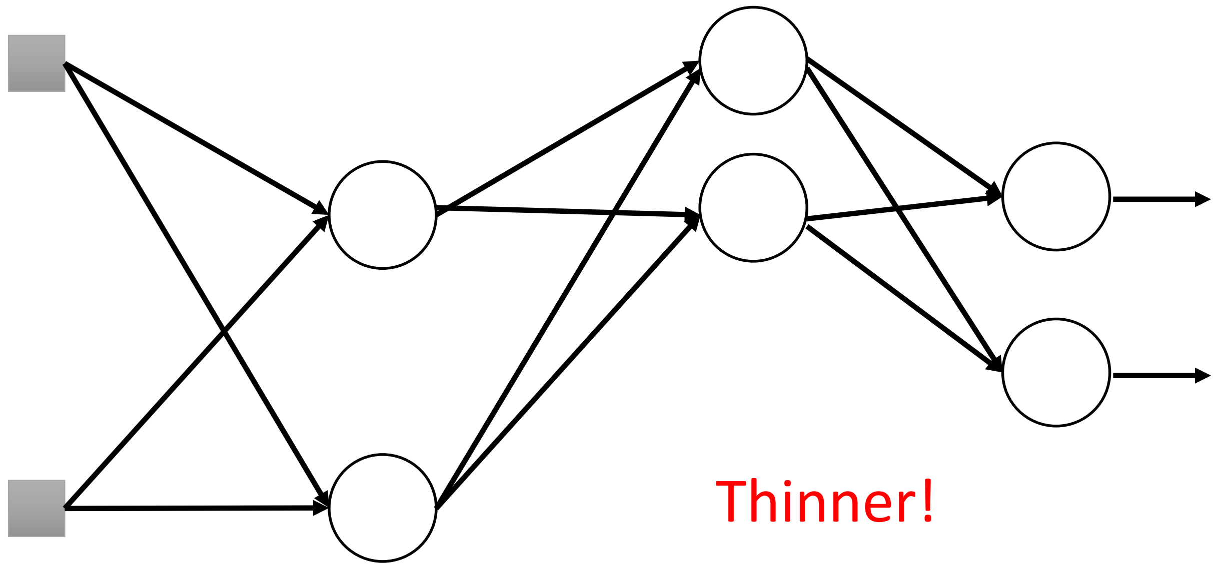
Training:



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

Dropout

Training:



Thinner!

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

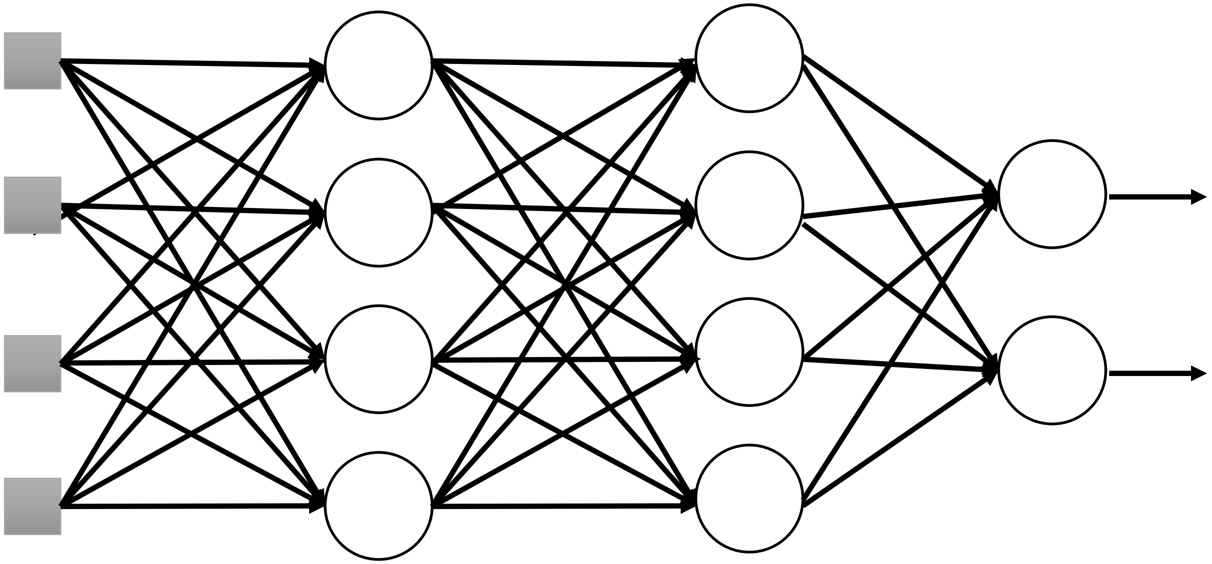
    ➡ **The structure of the network is changed.**
  - Using the new network for training

For each mini-batch, we resample the dropout neurons

Dropout

Dropout on testing:
1. We don't dropout any neurons when we are testing.
2. Instead, we decrease all the weights by multiplying (1-p%).
↳Or, we increase all the weights by dividing 1-p% on training.

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

If we are very strict on training state, we can expect that we will have better results on testing state.

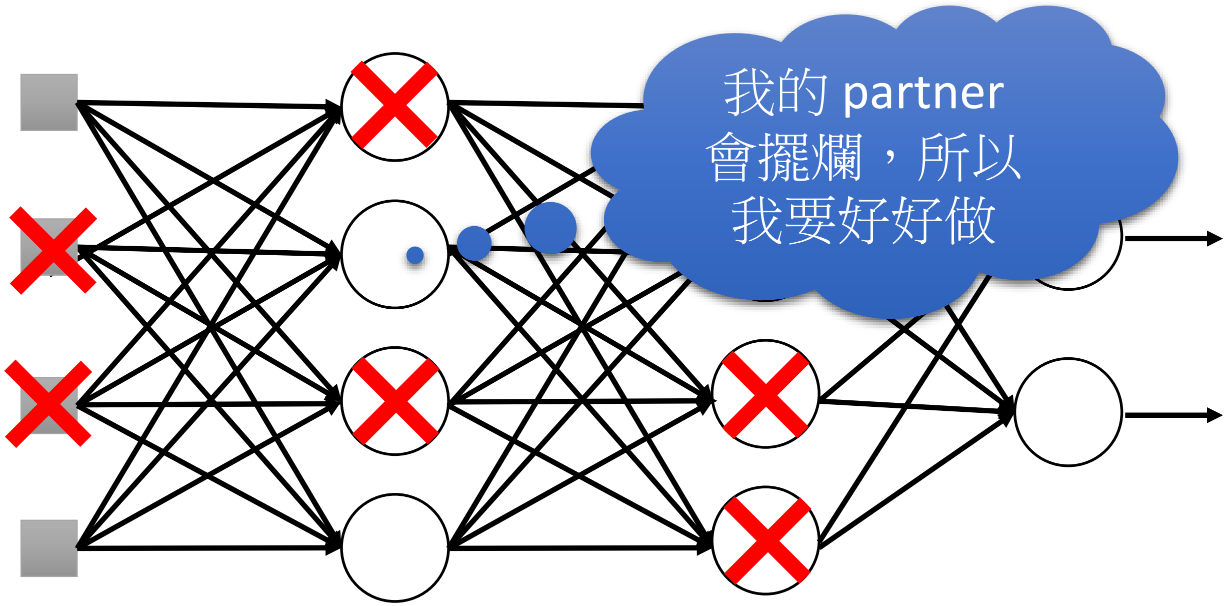
<u>Testing</u>

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.

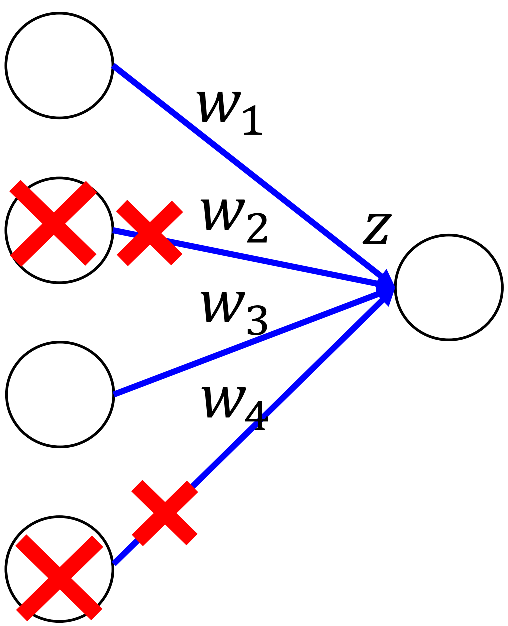
Dropout - Intuitive Reason

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

We want the output of the training state to be equal to the output of the testing state.

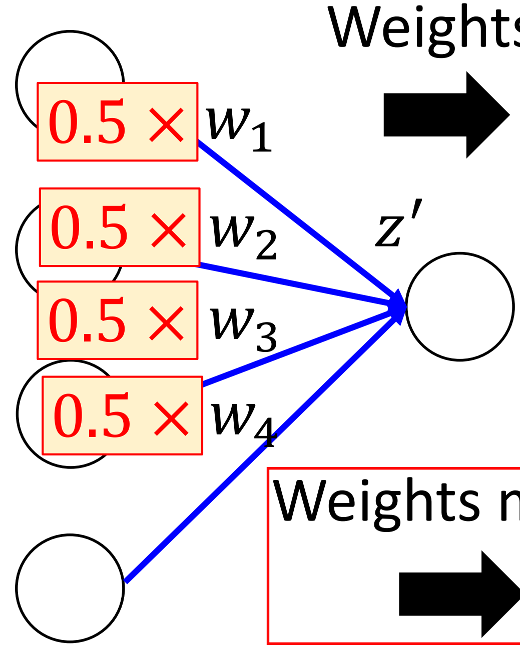
Training of Dropout

Assume dropout rate is 50%



Testing of Dropout

No dropout



Weights from training

$\rightarrow z' \approx 2z$

Weights multiply 1-p%

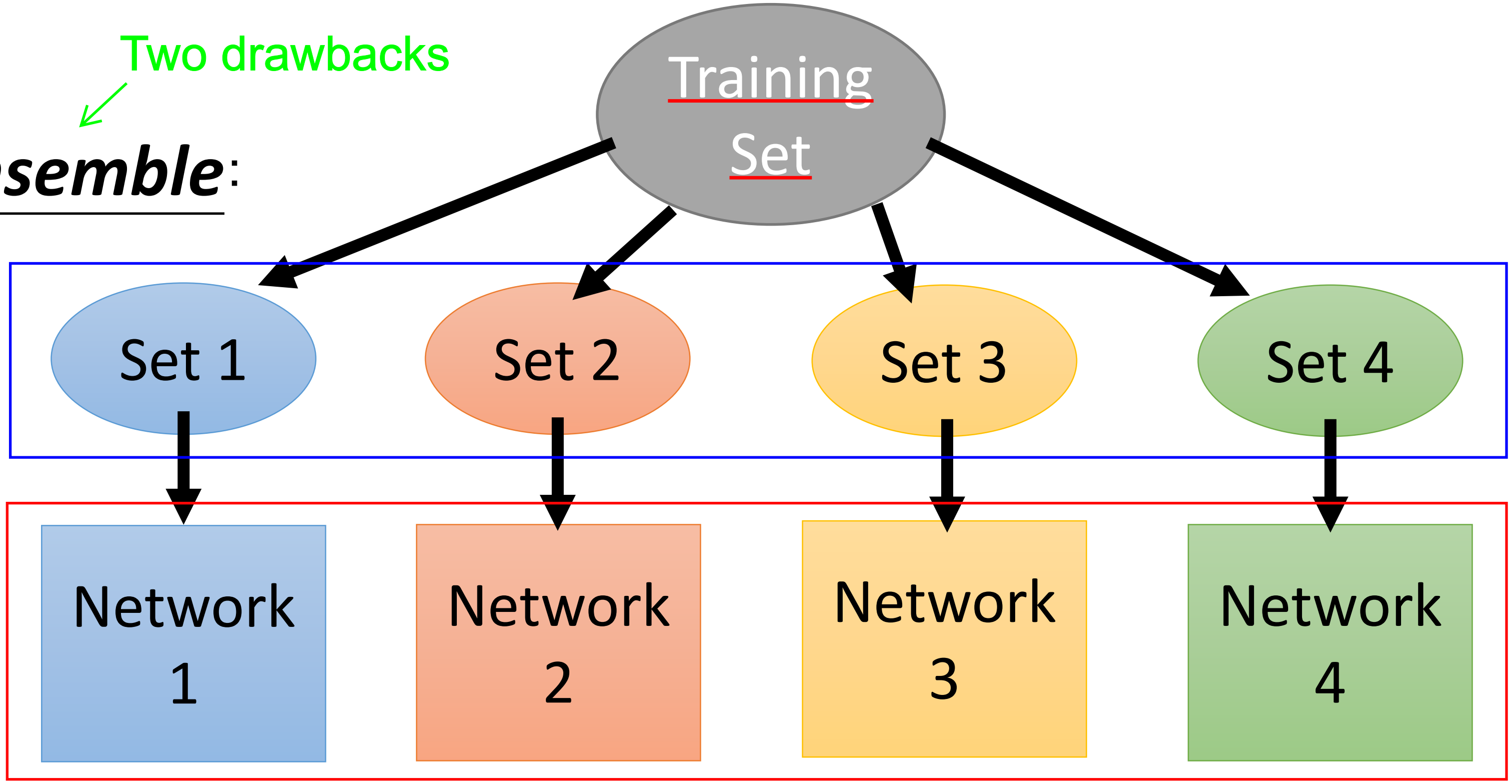
$\rightarrow z' \approx z$

Formal reason:

Dropout is a kind of ensemble.

Two drawbacks

Real ***Ensemble***:

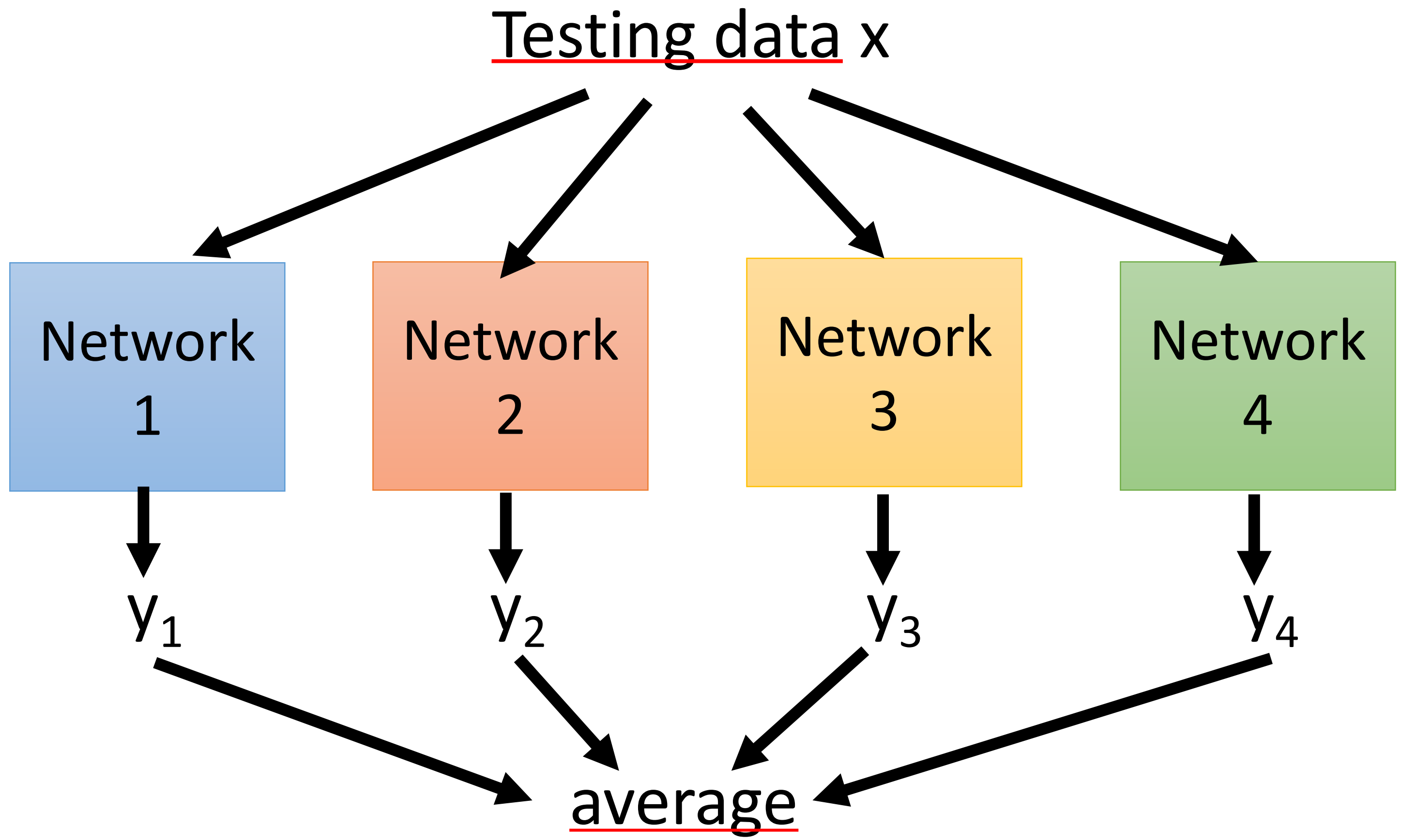


Train a bunch of networks with different structures ①
and different data ②

Didn't share the parameters. $\because$ → $\therefore$ We may encounter the problem of insufficient data.

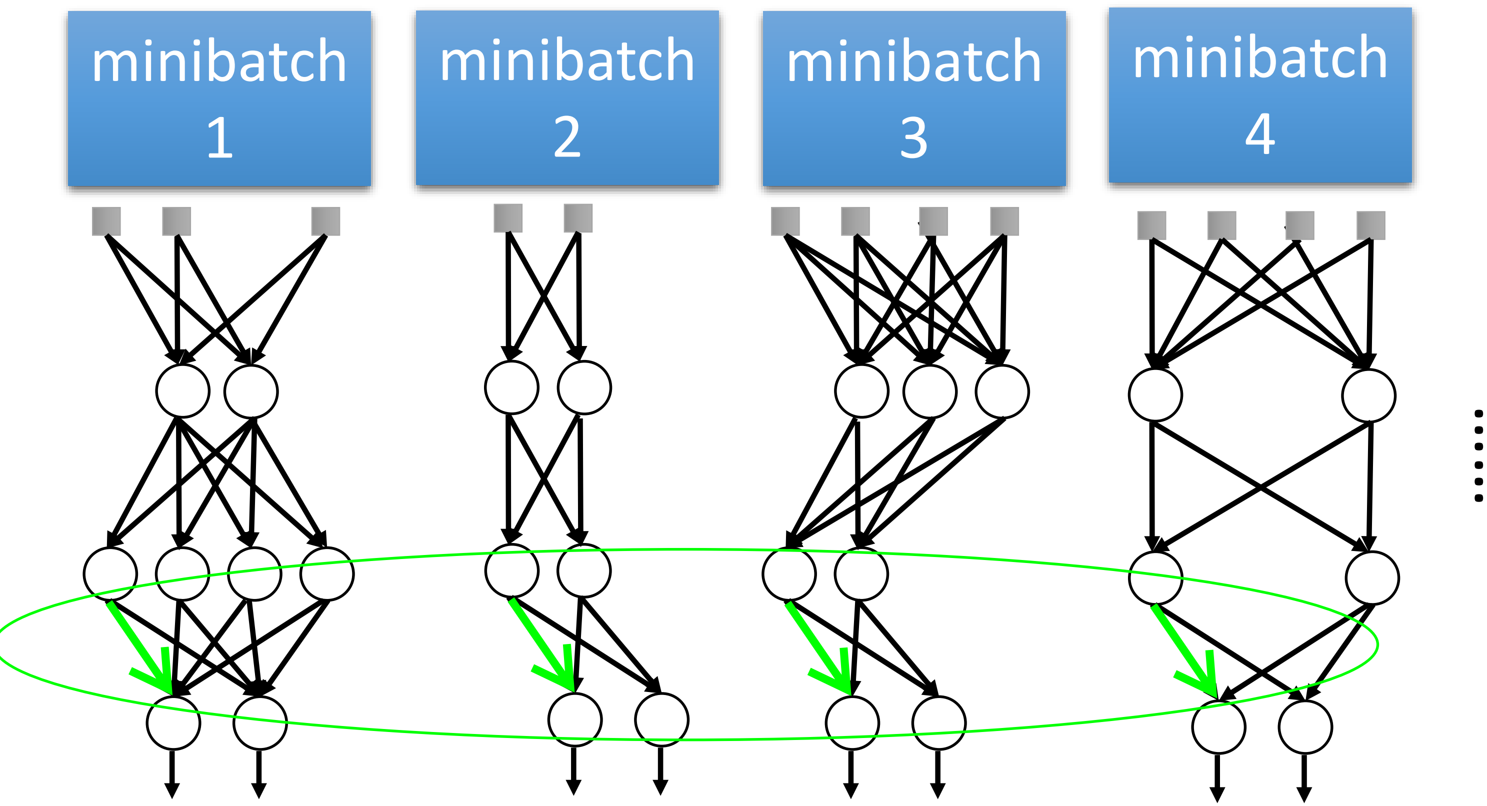
Dropout is a kind of ensemble.

Real ***Ensemble***: When we average a lot of models with big variance, we can get a model which is very similar to the target one.



Dropout is a kind of ensemble.

The ultimate version of ensemble ∵ Dropout shares the parameters and data.



Training of Dropout

M neurons

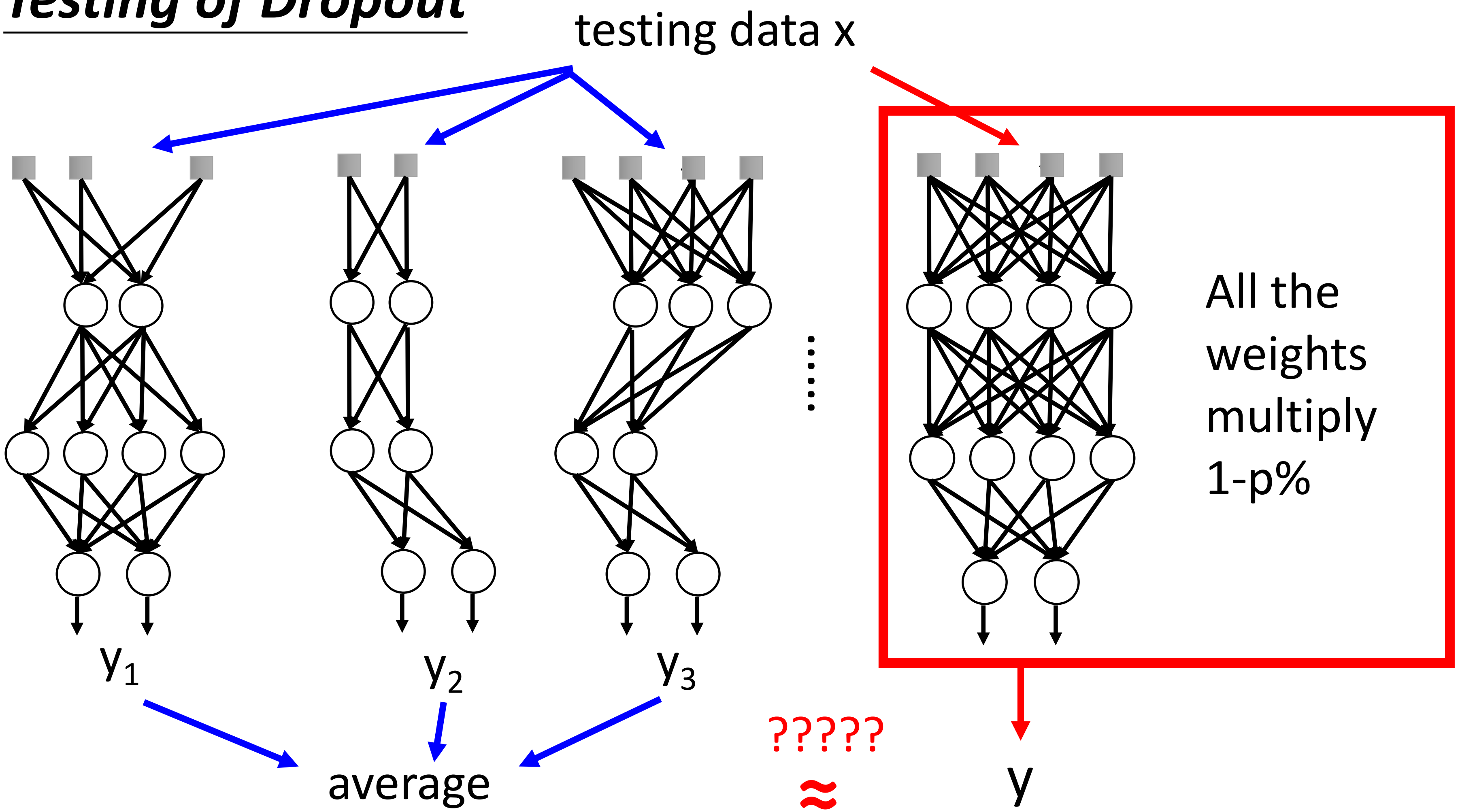
2^M possible networks

For this weight, we use a lot of mini-batch to train it.

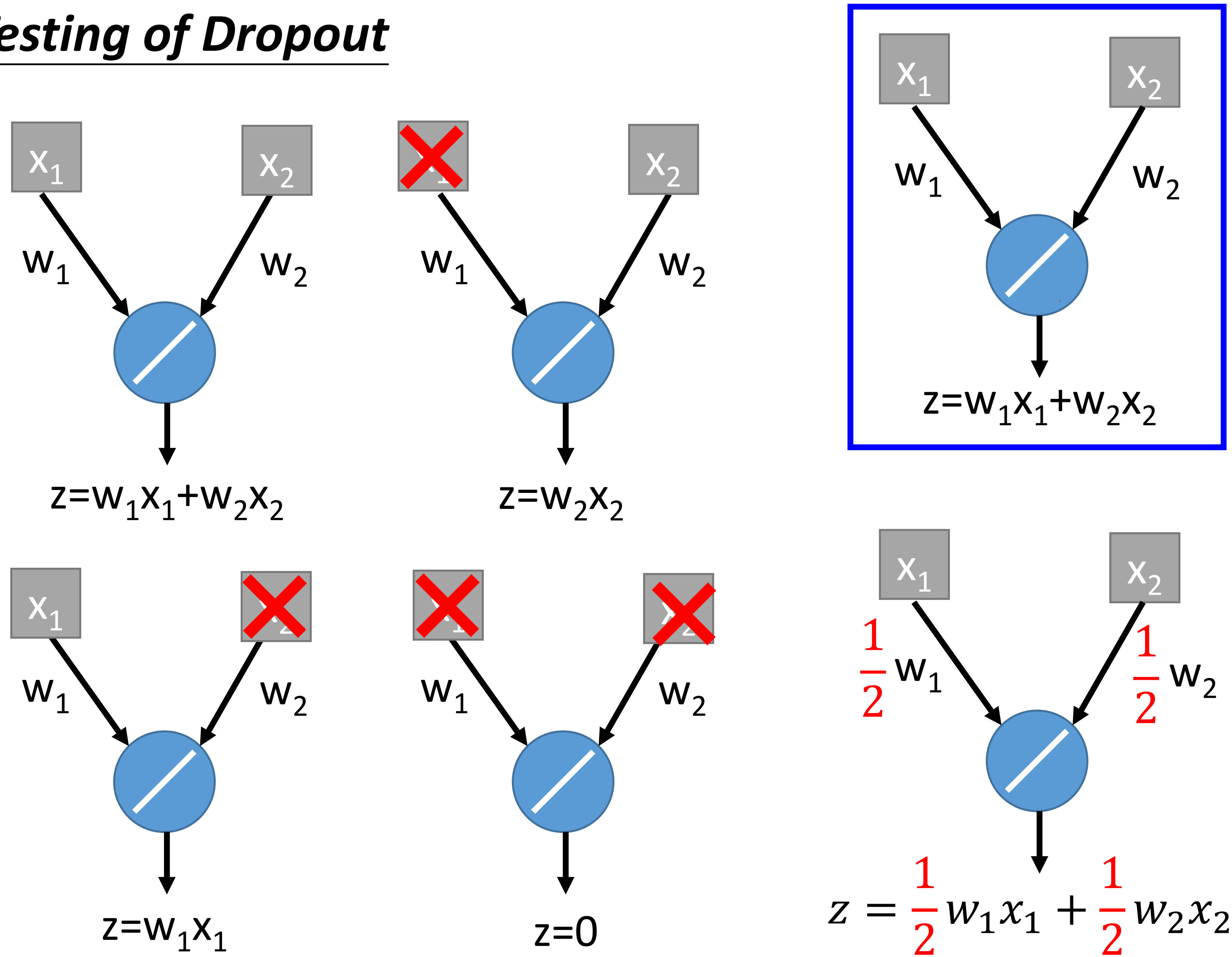
- Using one mini-batch to train one network
- Some parameters in the network are shared

Dropout is a kind of ensemble.

Testing of Dropout



Testing of Dropout



Recipe of Deep Learning