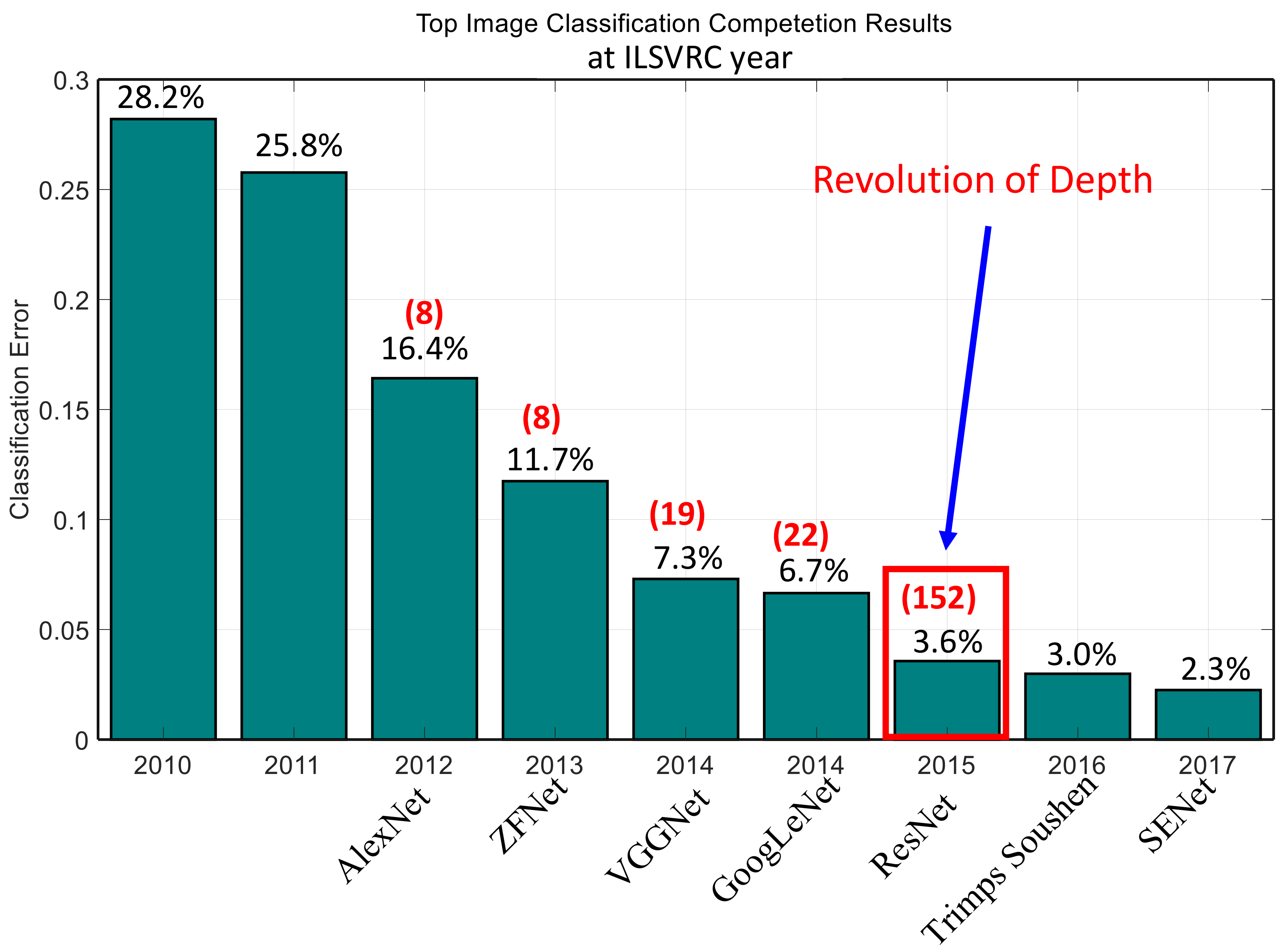
My Last Lecture
Typical CNN Architectures

Case Studies
- AlexNet
- VGG
- GoogLeNet
- ResNet

about other architecture…

- NIN (Network in Network)
- Wide ResNet
- ResNeXT
- Stochastic Depth
- DenseNet
- SENet
- FractalNet
- SqueezeNet



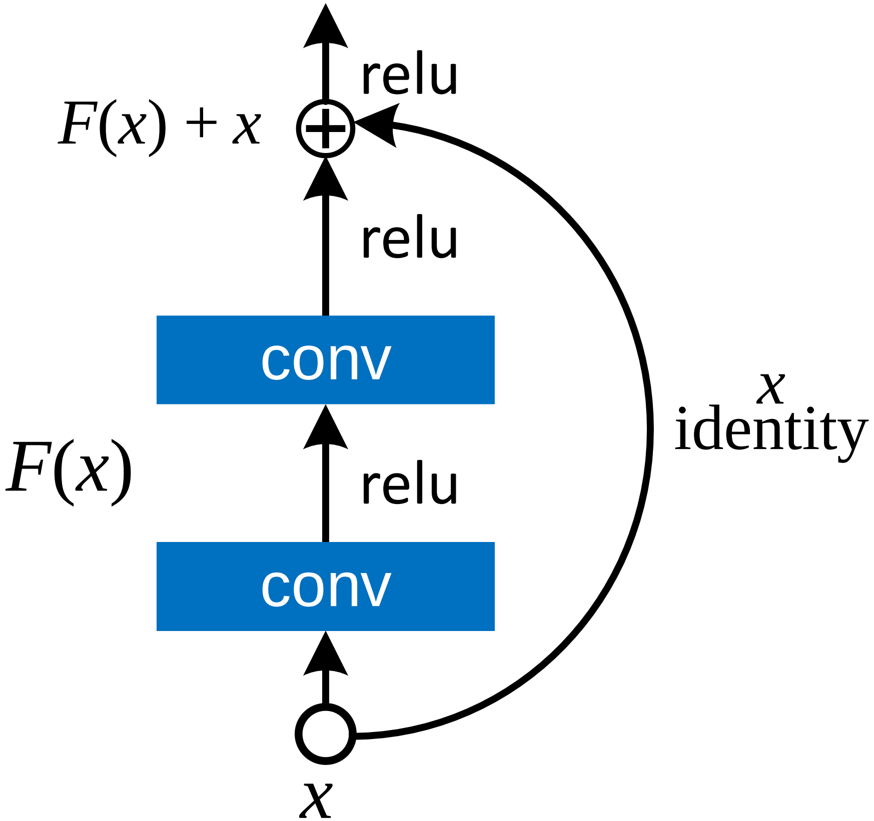
Li Liu et al., Deep Learning for Generic Object Detection: A Survey, IJCV, 2019.

Case Study: ResNet

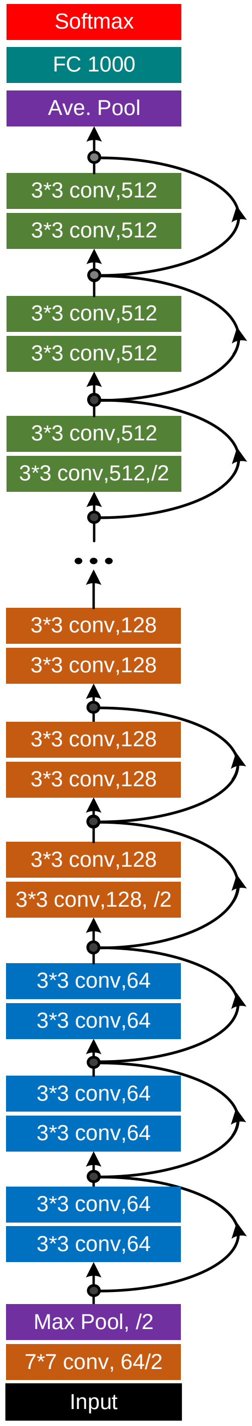
[He et al., 2015]

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!



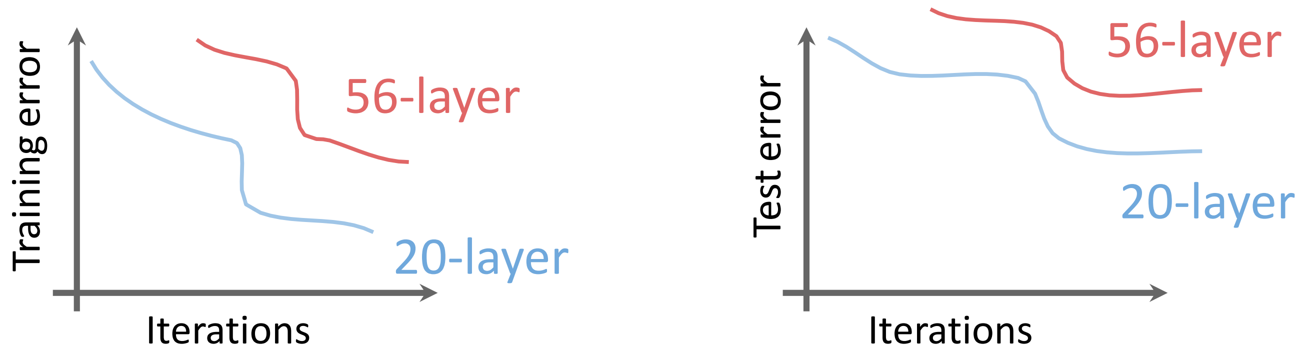
Residual block



Case Study: ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



Q: What’s strange about these training and test curves?
[Hint: look at the order of the curves]

56-layer model performs worse on both training and test error.
→The deeper model performs worse, but it’s not caused by overfitting!

Case Study: ResNet

[He et al., 2015]

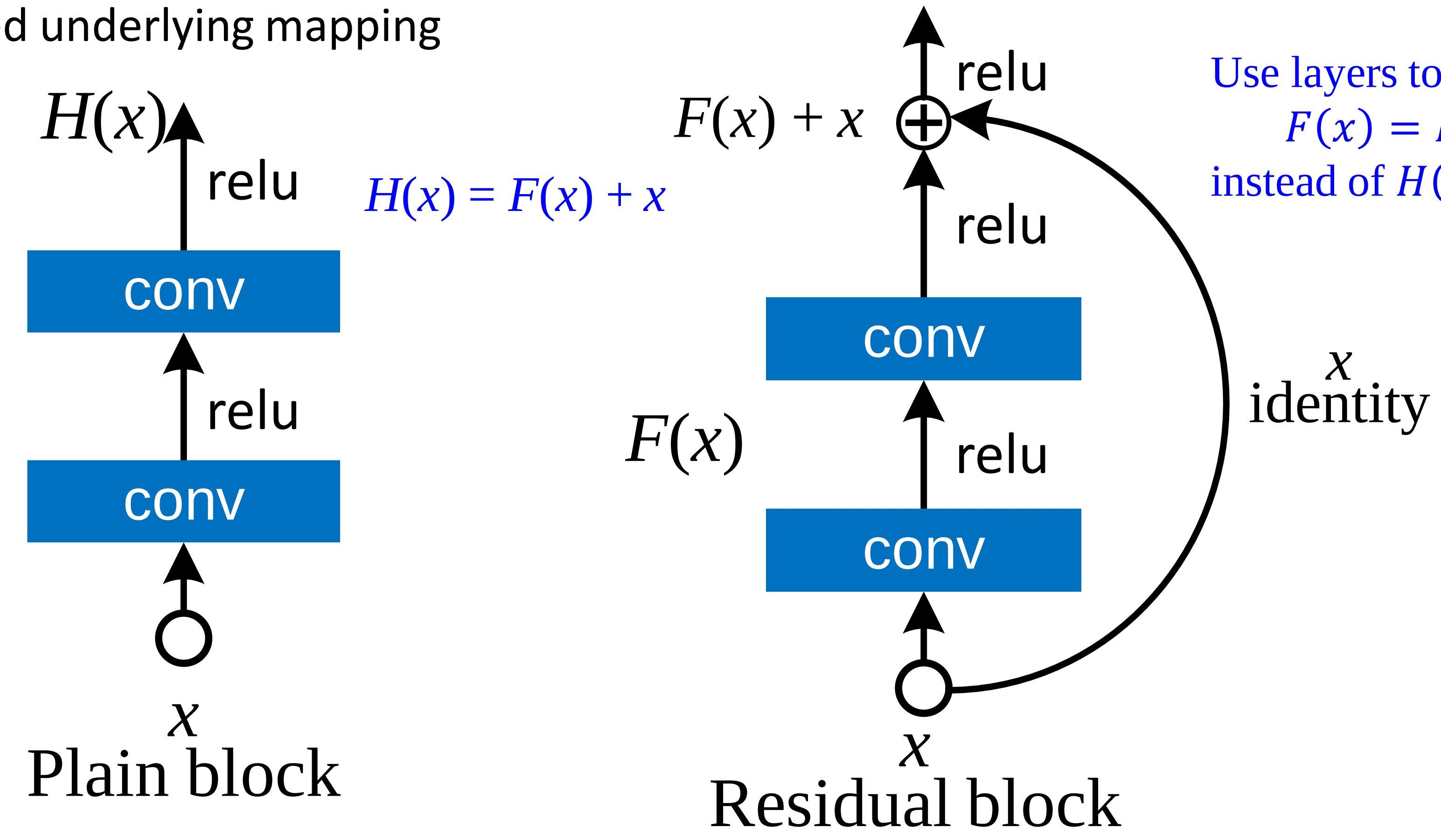
Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

- The deeper model should be able to perform at least as well as the shallower model.
- A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

Case Study: ResNet

[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



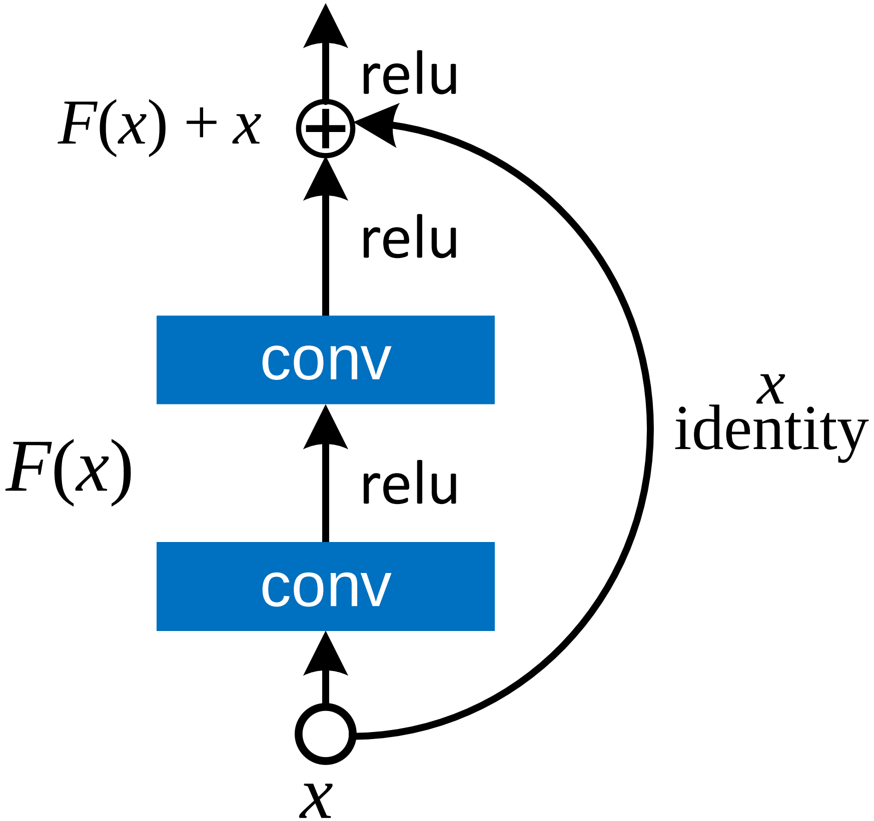
Use layers to fit residual
$F(x) = H(x) - x$
instead of $H(x)$ directly

Case Study: ResNet

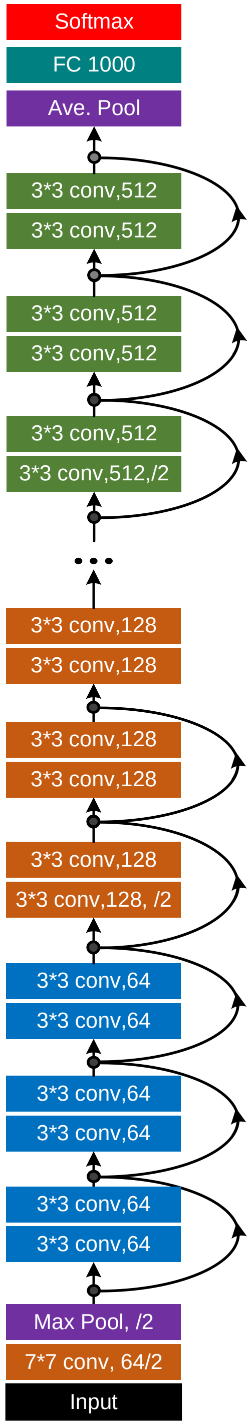
[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double the number of filters and subsample spatially using stride 2
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)



Residual block



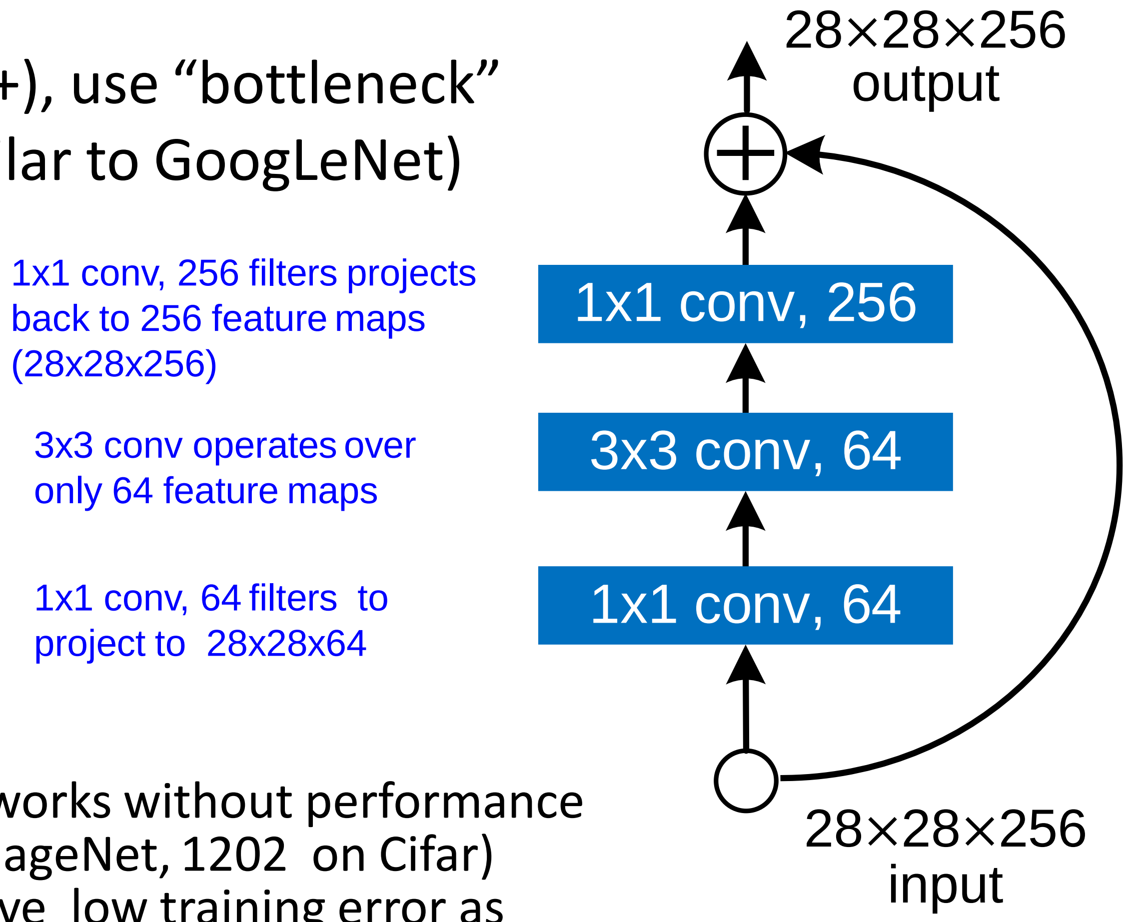
Total depths of 34, 50, 101, or 152 layers for ImageNet

Case Study: ResNet

[He et al., 2015]

For deeper networks (ResNet50+), use "bottleneck" layer to improve efficiency (similar to GoogLeNet)

ILSVRC 2015 classification winner (3.6% top 5 error)→better than "human performance"!

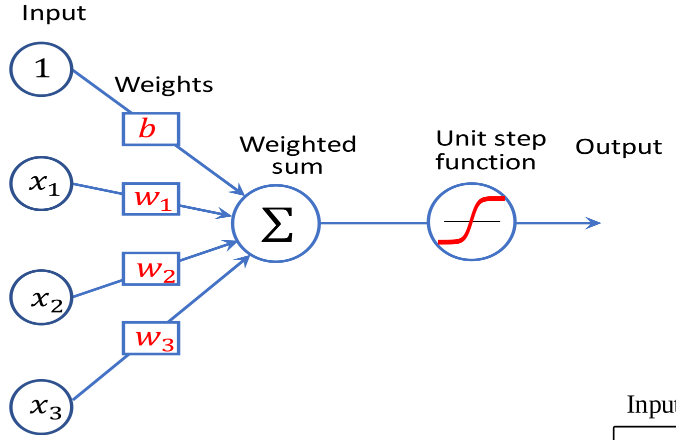


Experimental Results

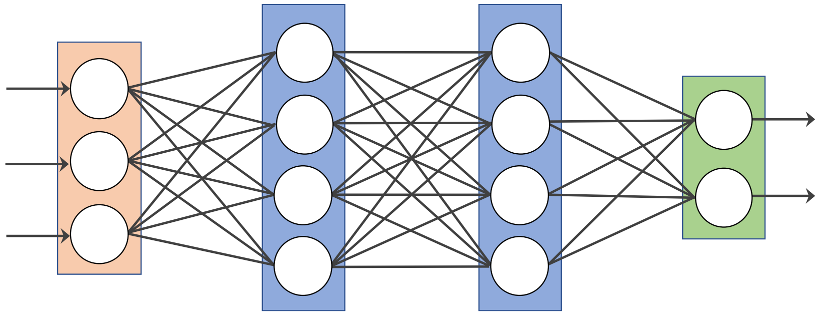
- Able to train very deep networks without performance degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve low training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

In this Course

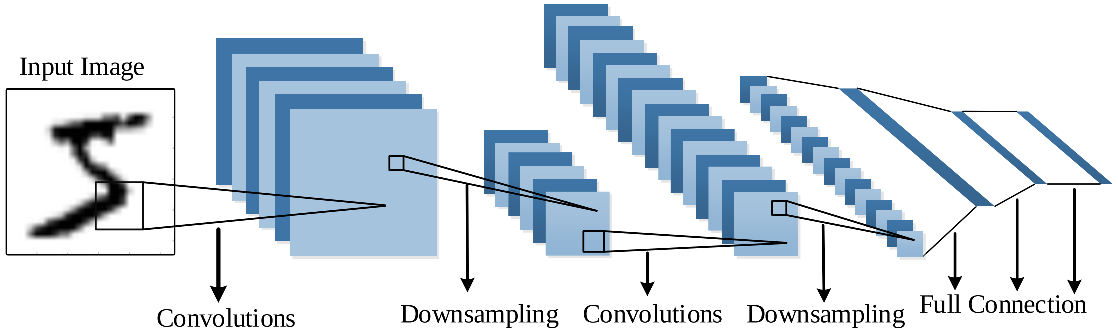
1. DL basics, linear regression, logistic regression etc.



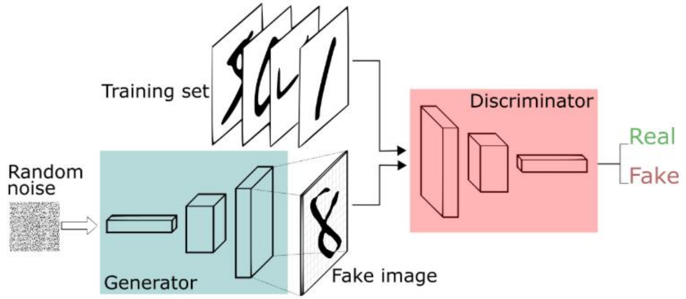
2. Multilayer neural networks, backpropagation



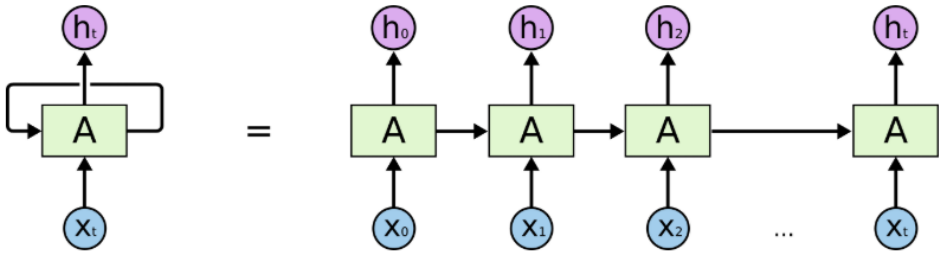
3. Convolutional Neural Networks and Applications



4. Generative Adversarial Networks



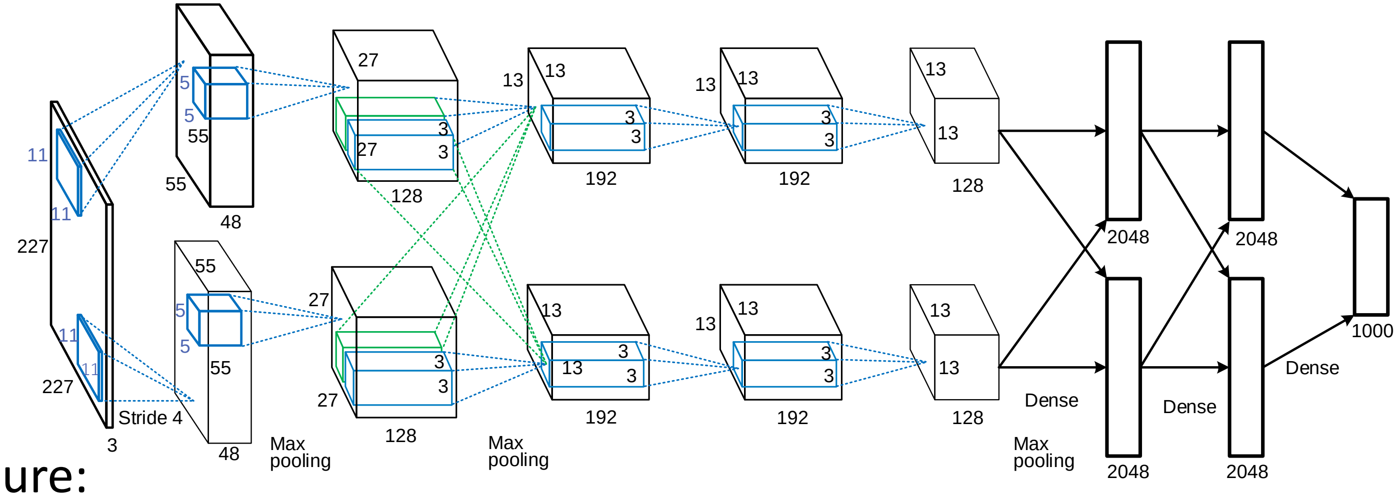
5. Recurrent networks and applications



For this Final Lecture Today:
Tips for Training Deep Learning

Case Study: AlexNet

[Krizhevsky *et al.* 2012]



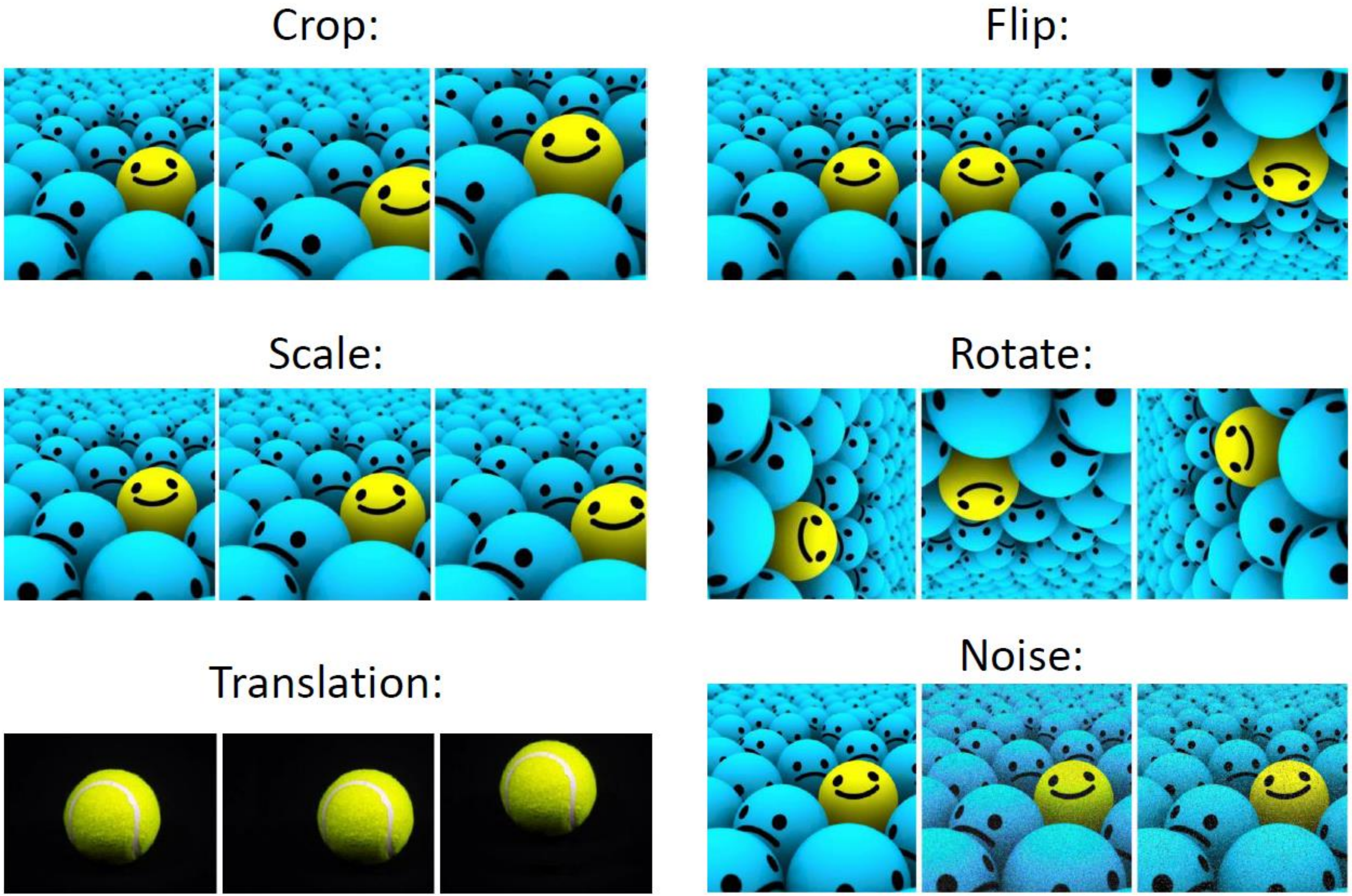
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details:

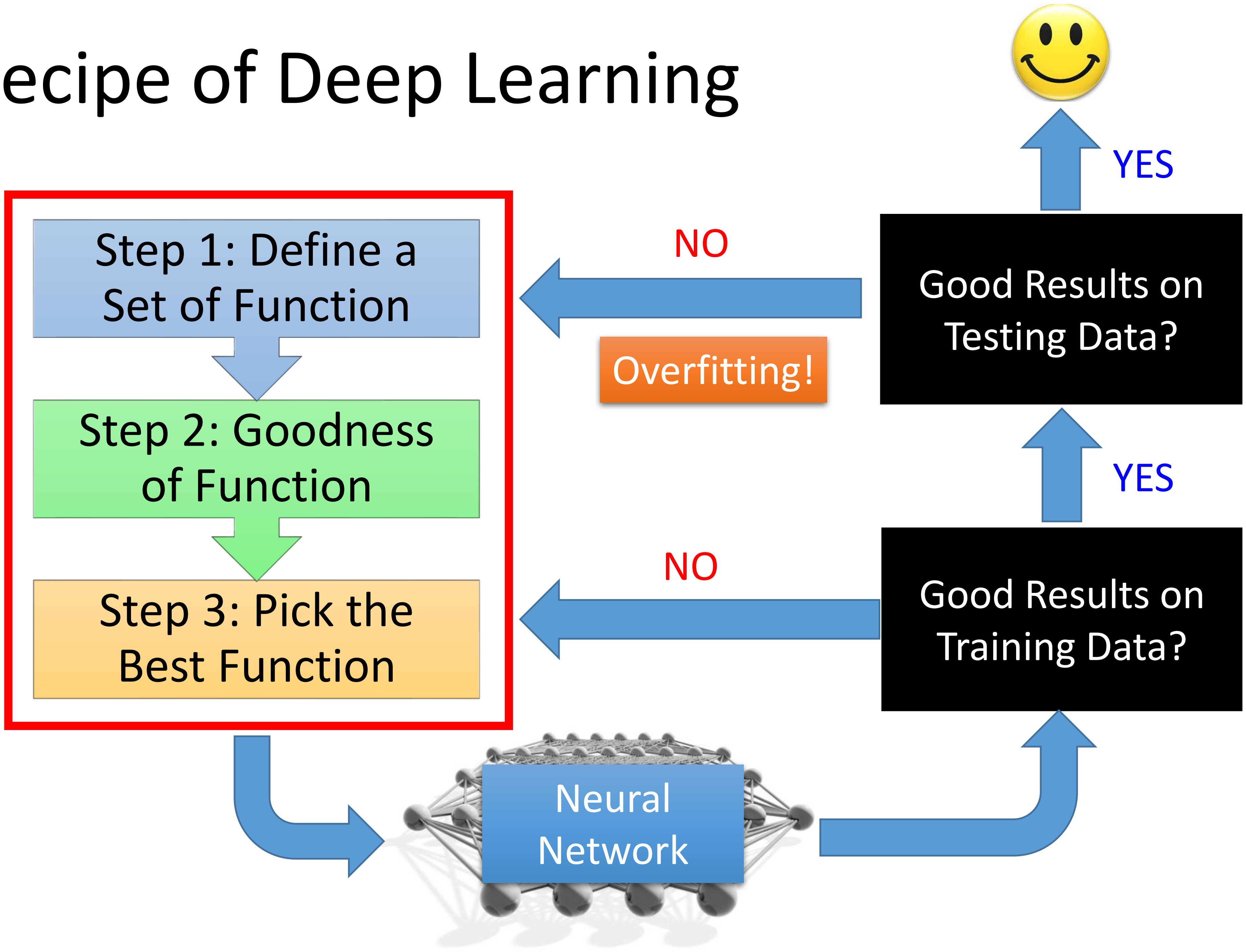
- heavy data augmentation
- first use of ReLU
- used Norm layers (not common anymore)
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 0.01, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 0.0005

7 CNN ensemble: 18.2%→ 15.4%

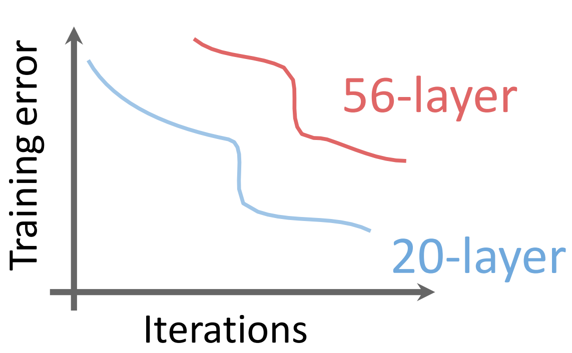
Data Augmentation

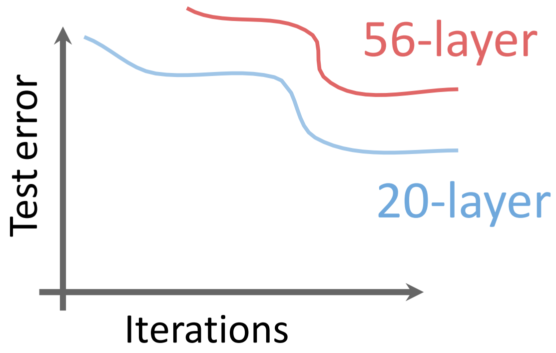


Recipe of Deep Learning



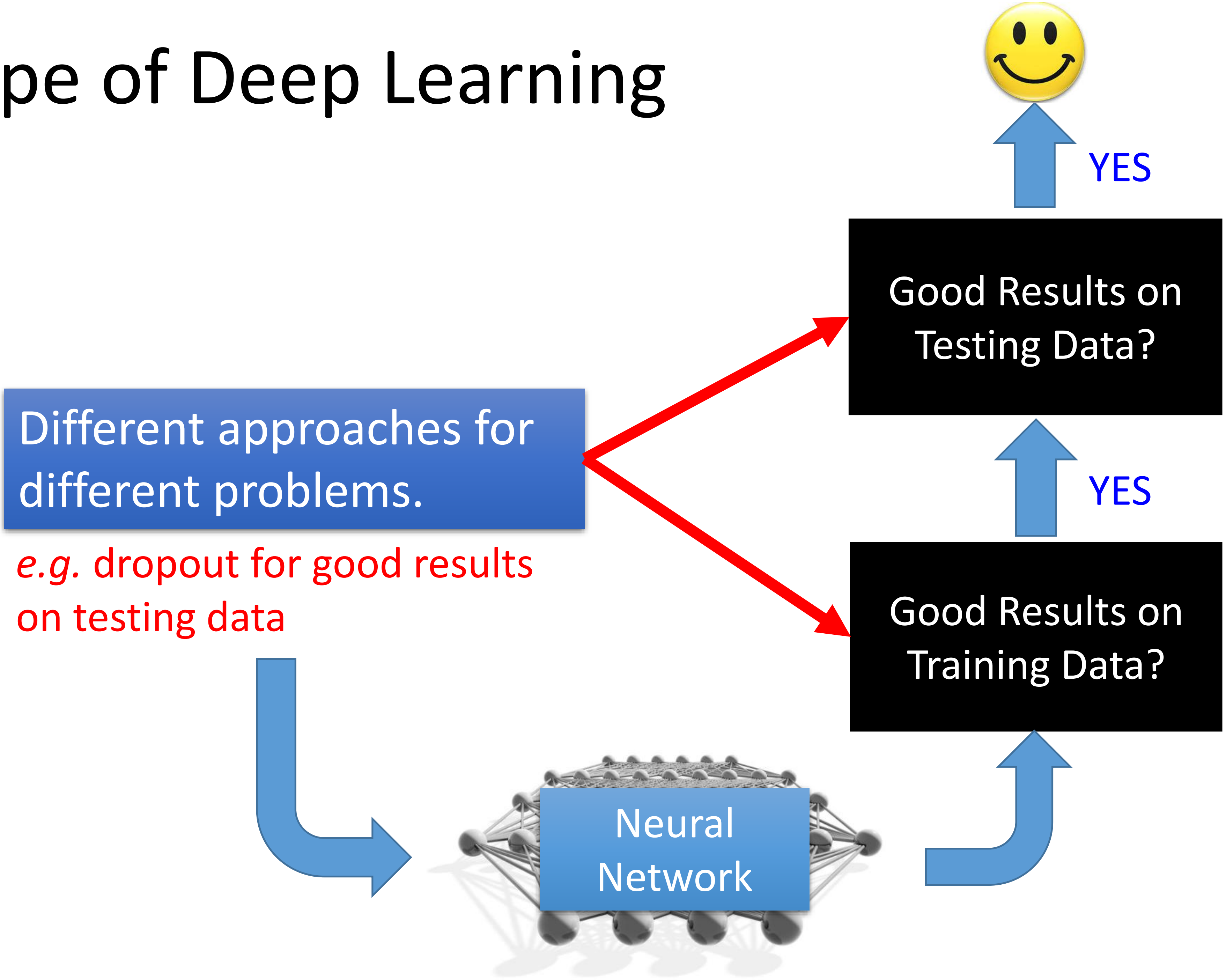
Do not always blame Overfitting



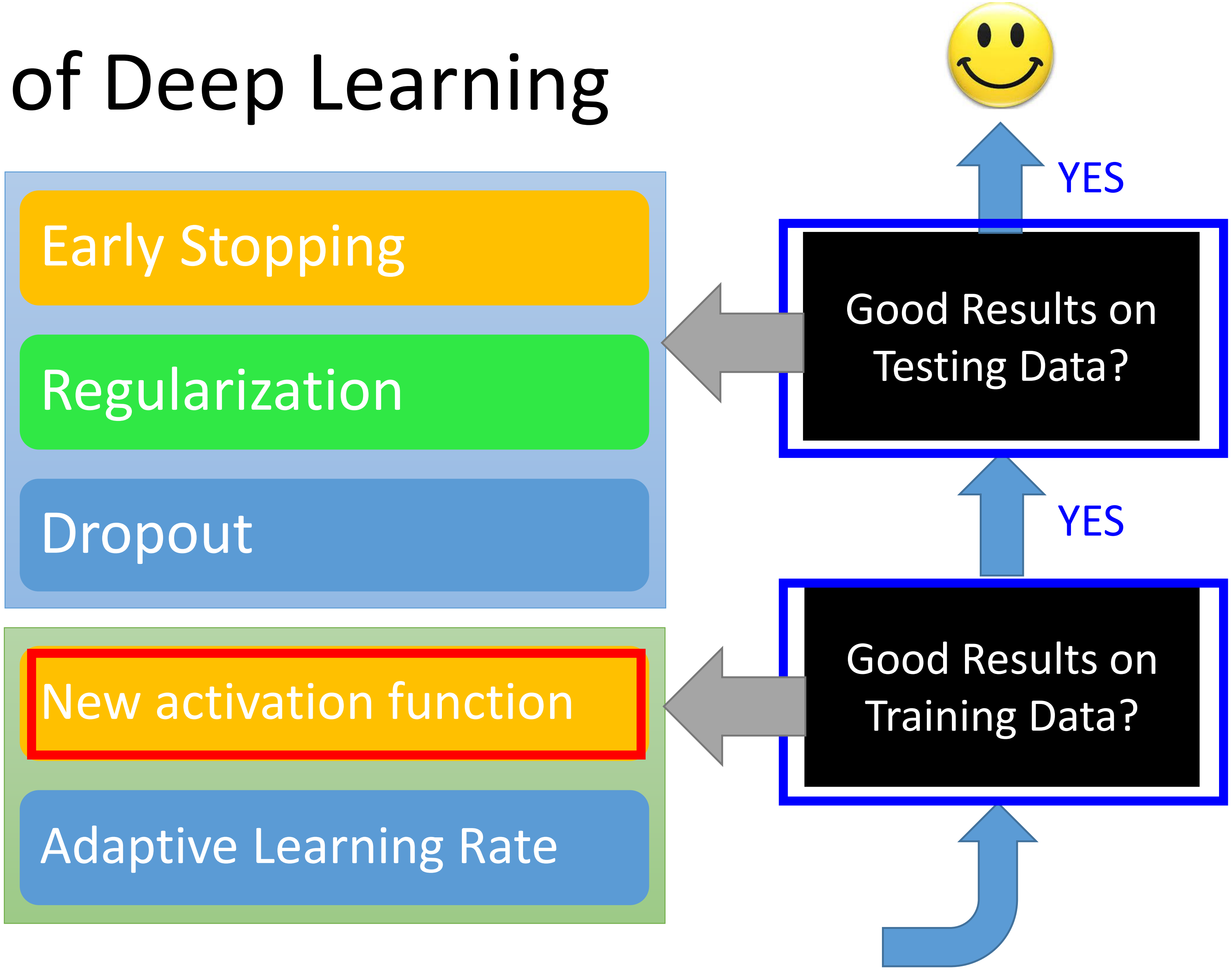


Please refer to:
Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep Residual Learning for Image Recognition, CVPR, 2016.

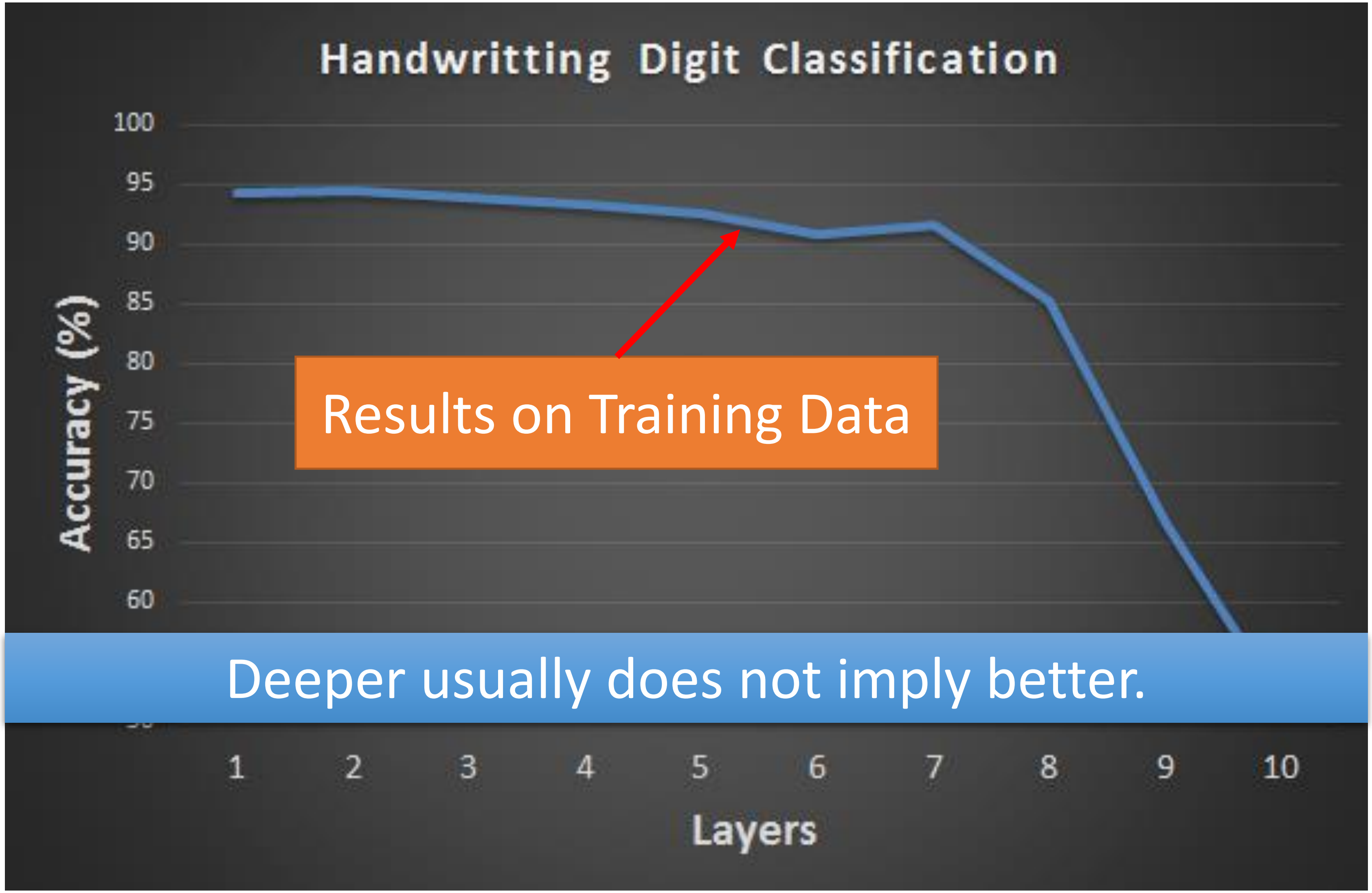
Recipe of Deep Learning



Recipe of Deep Learning

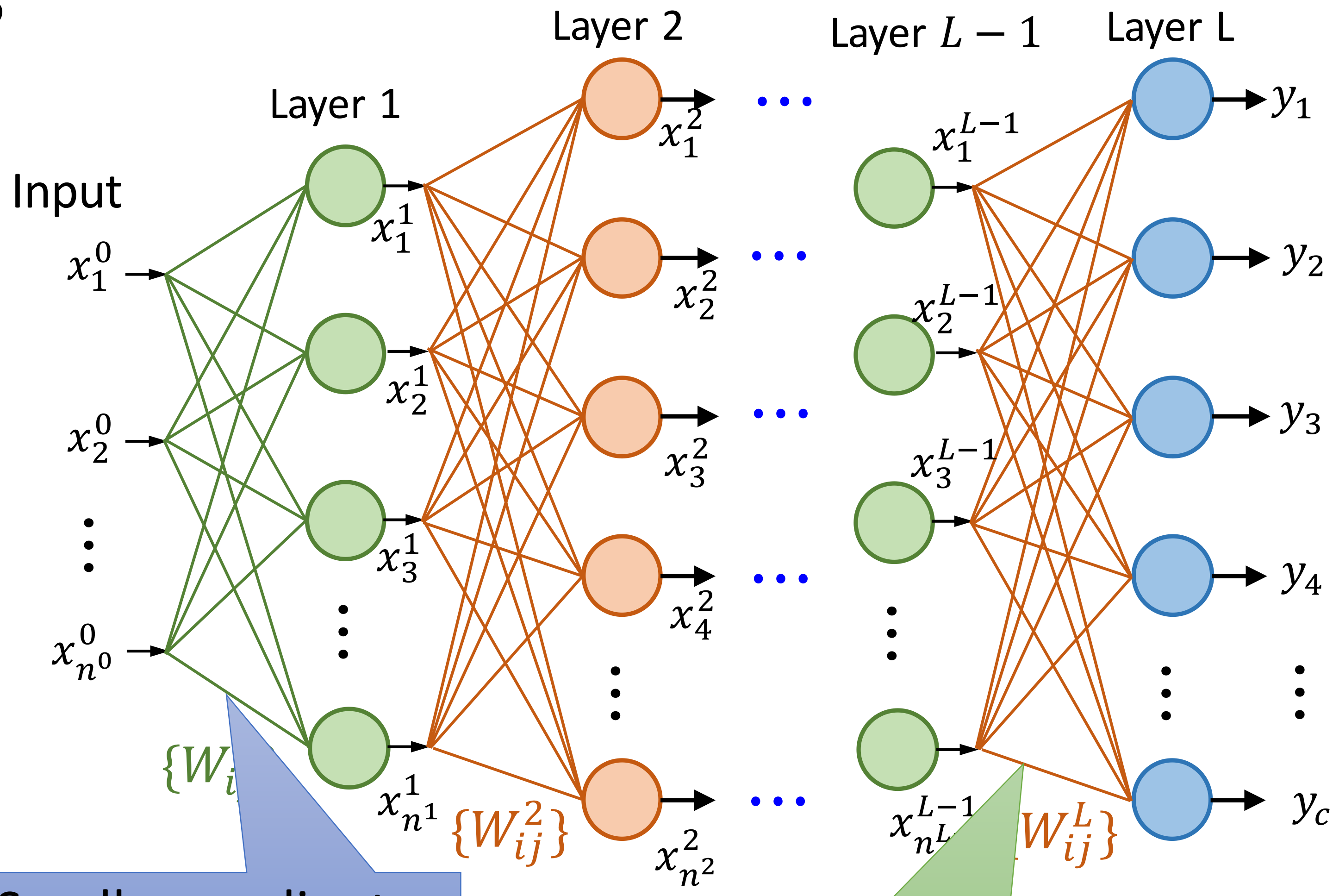


Hard to get the power of Deep …



Vanishing Gradient Problem

See Gradient flow in recurrent nets: the difficulty of learning long term dependencies, by Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber (2001).



Smaller gradients

Learn very slow

Almost random

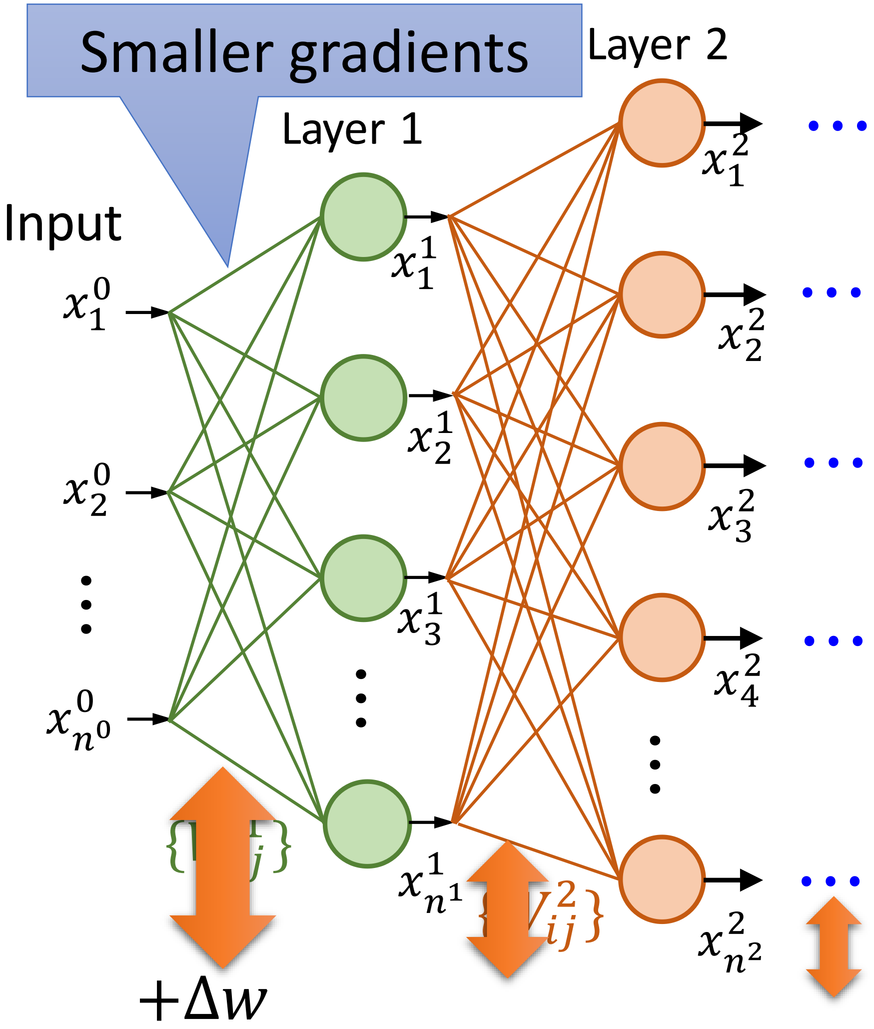
Larger gradients

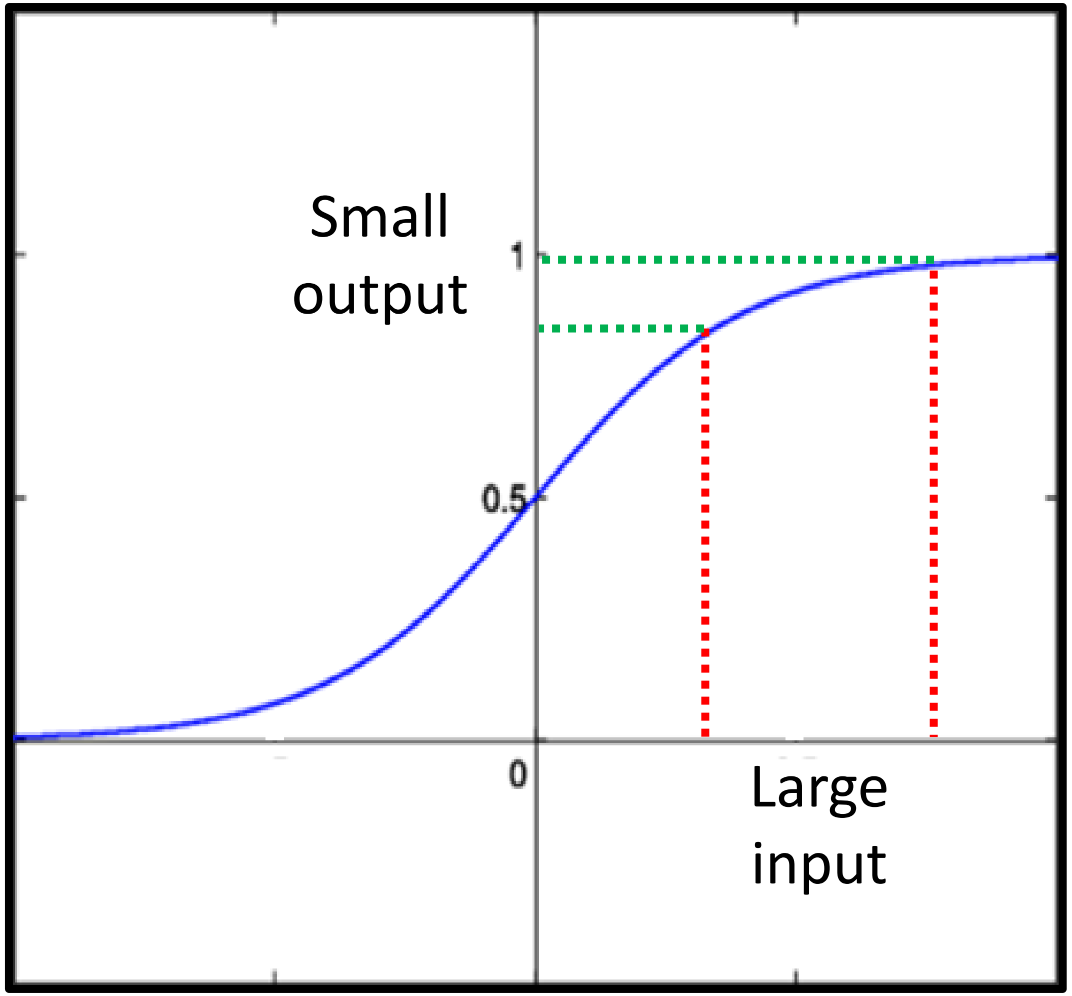
Learn very fast

Already converge

converge based on random!?

Vanishing Gradient Problem



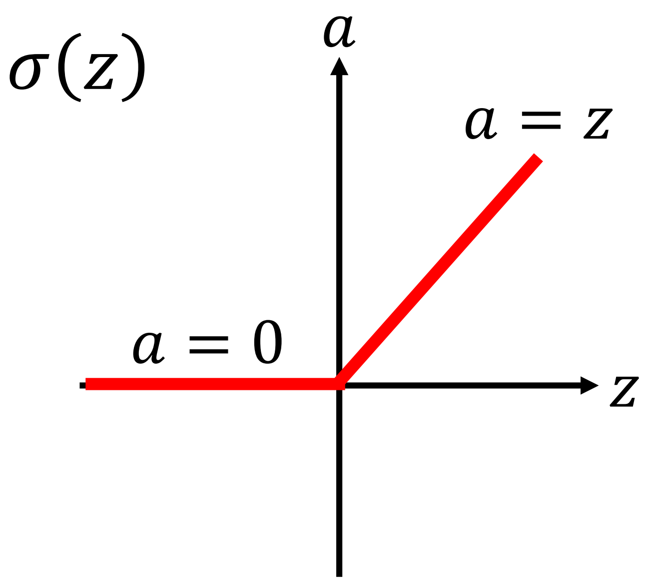


Intuitive way to compute the derivatives ...

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

ReLU

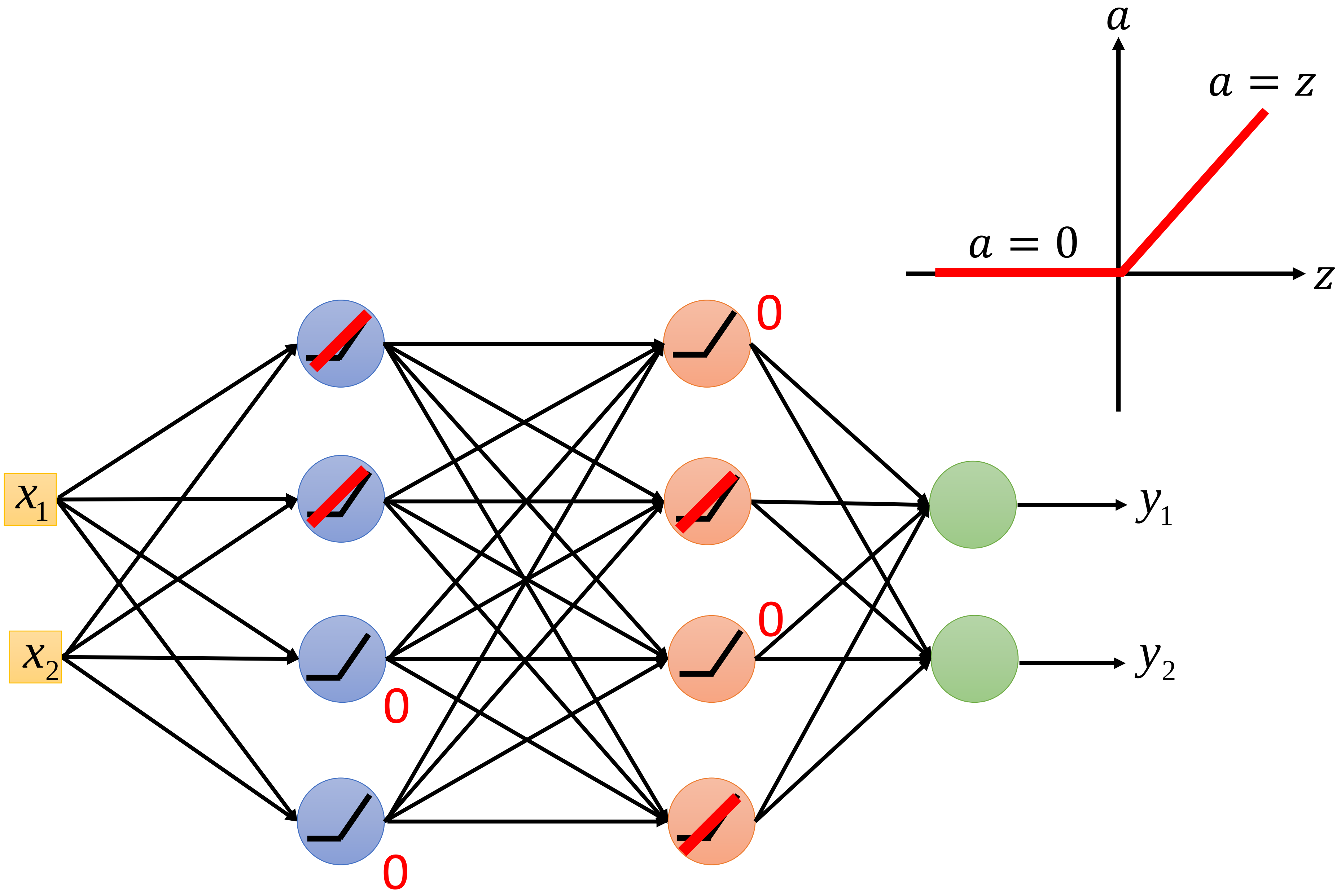
- Rectified Linear Unit (ReLU)



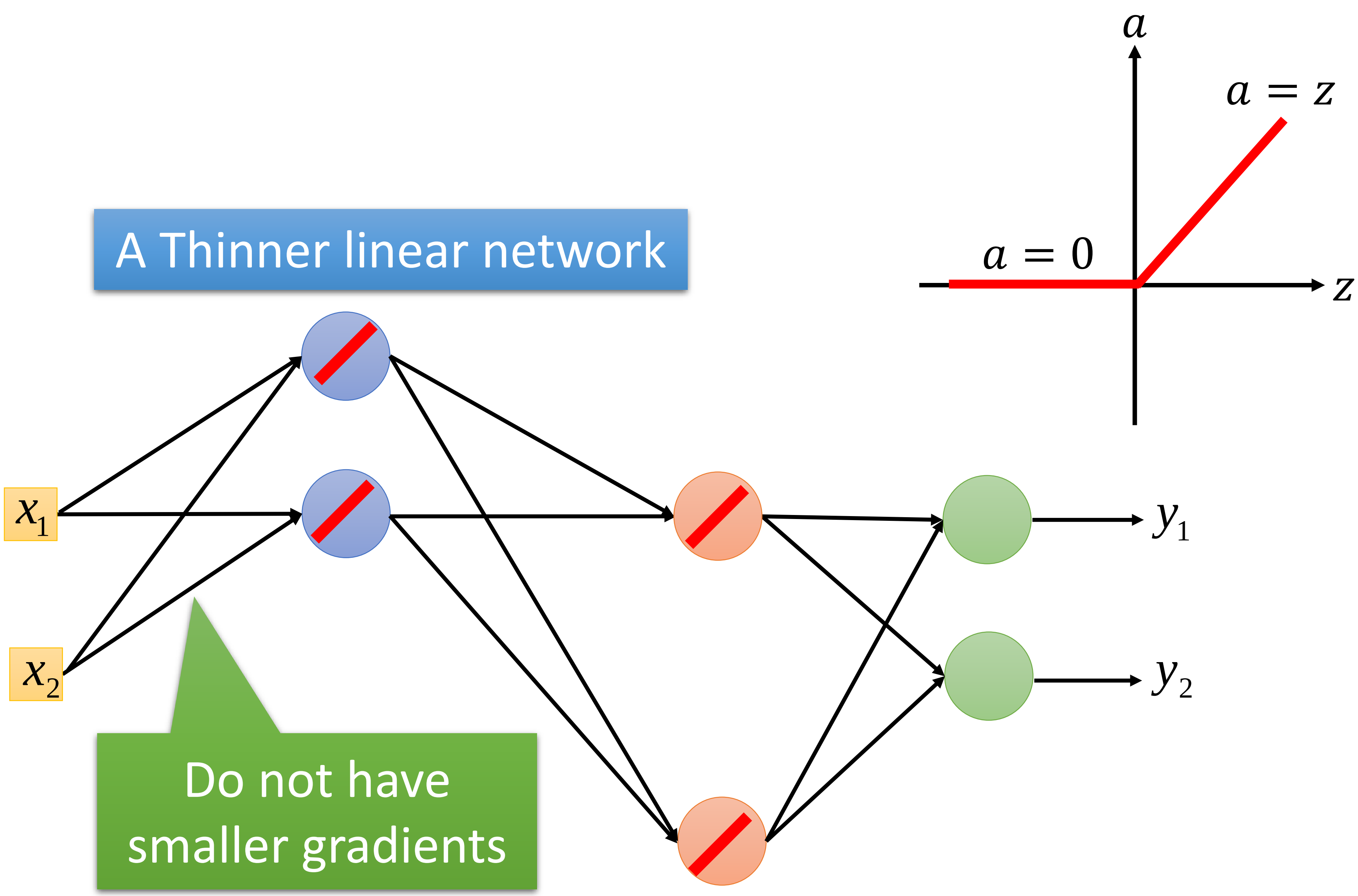
Reason:

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

ReLU



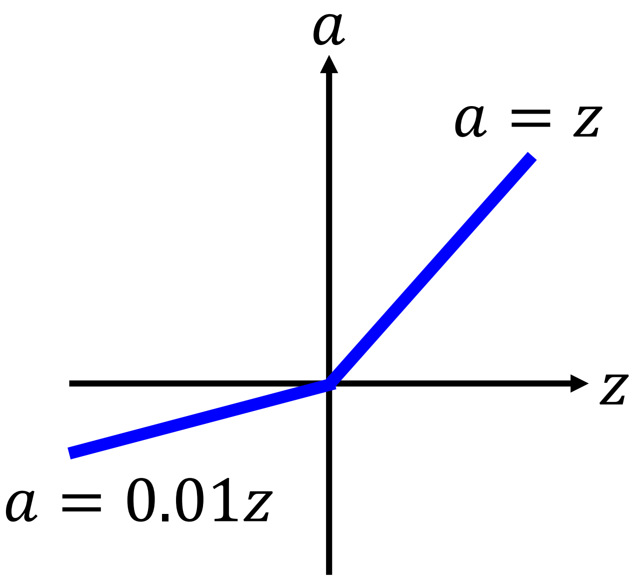
ReLU



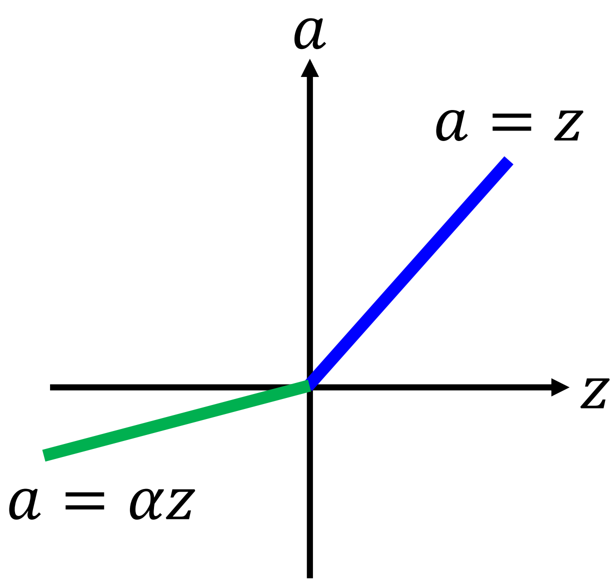
A Thinner linear network
Do not have
smaller gradients
x_1
x_2
y_1
y_2
a
$a = z$
$a = 0$
z

Variants of ReLU

Leaky ReLU



Parametric ReLU



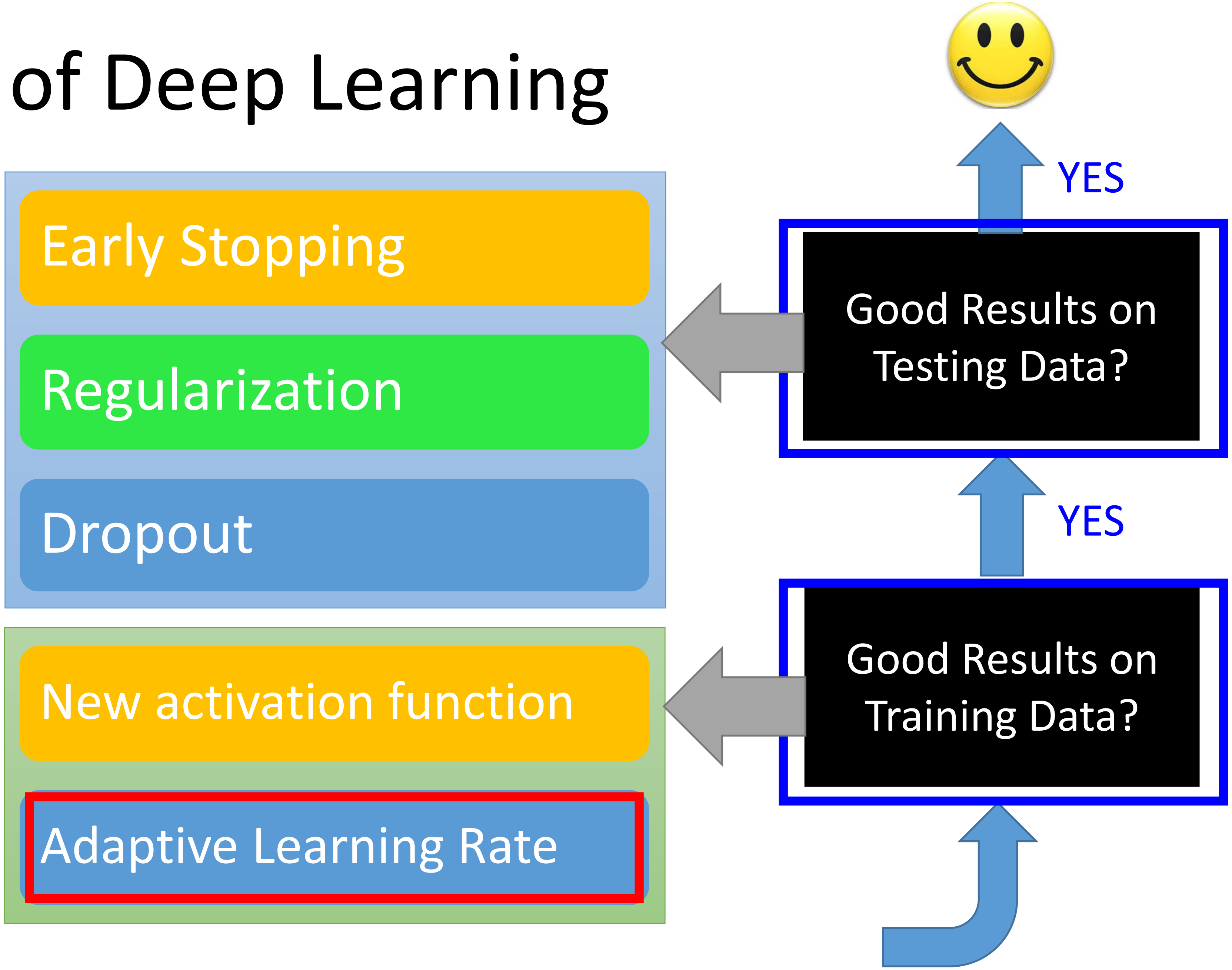
α also learned by gradient descent

Maxout

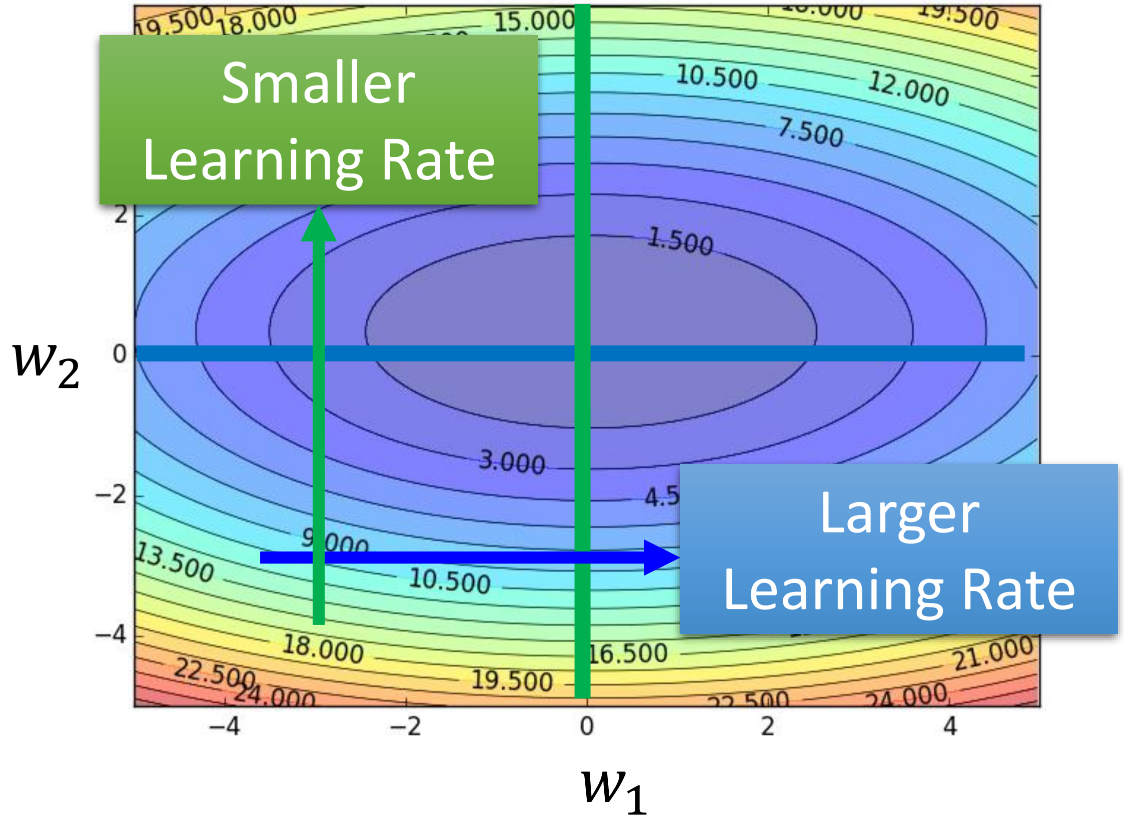
ReLU is a special case of Maxout

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

Recipe of Deep Learning



Review



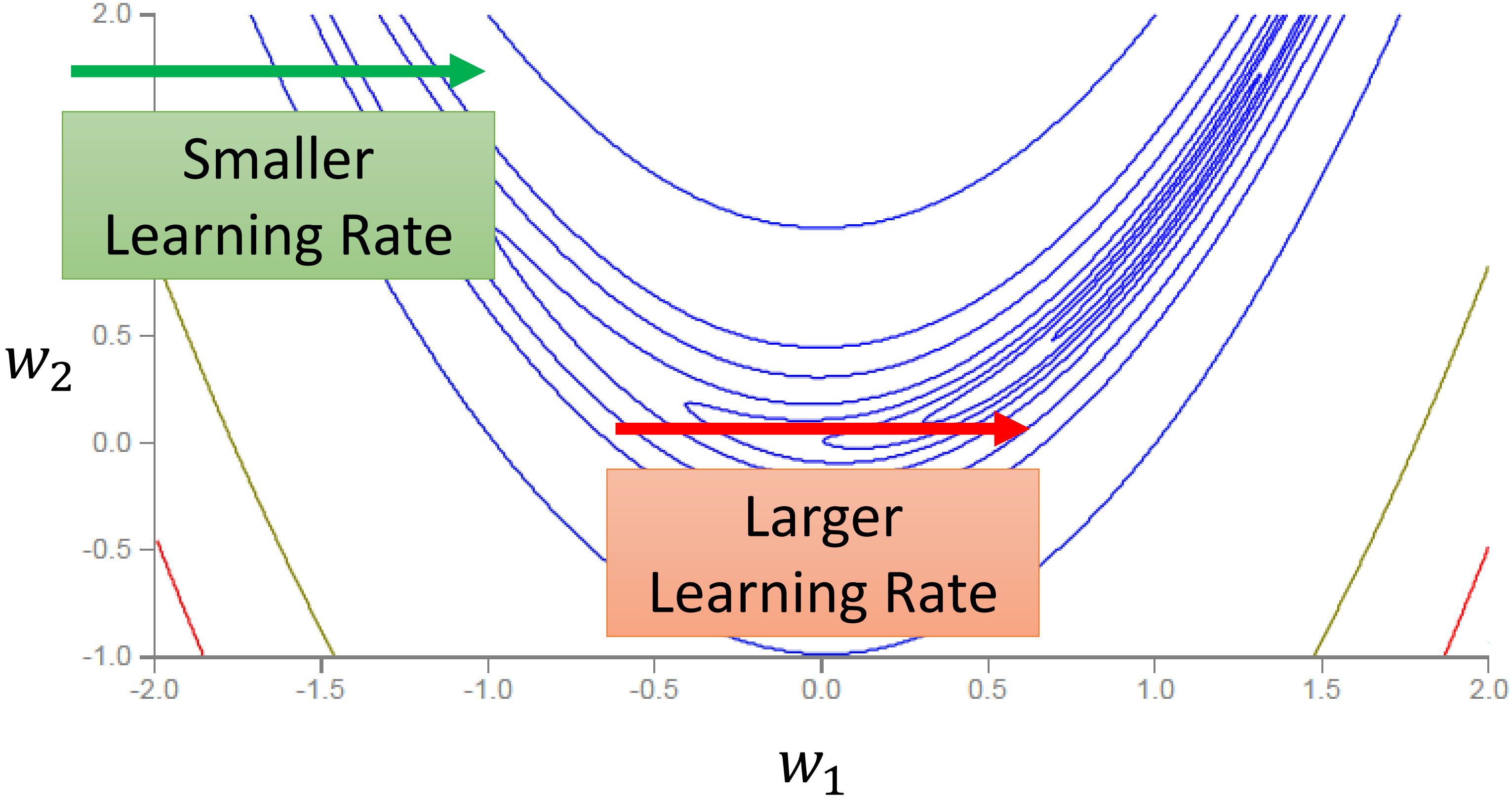
AdaGrad

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

Use first derivative to estimate second derivative

RMSProp

Error Surface can be very complex when training NN.



RMSProp

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

AdaGrad

$$w^{(1)} \leftarrow w^{(0)} - \frac{\eta}{\sigma^{(0)}} g^{(0)}$$

$$\sigma^{(0)} = g^{(0)}$$

$$w^{(2)} \leftarrow w^{(1)} - \frac{\eta}{\sigma^{(1)}} g^{(1)}$$

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

$$w^{(3)} \leftarrow w^{(2)} - \frac{\eta}{\sigma^{(2)}} g^{(2)}$$

$$\sigma^{(2)} = \sqrt{\alpha(\sigma^{(1)})^2 + (1-\alpha)(g^{(2)})^2}$$

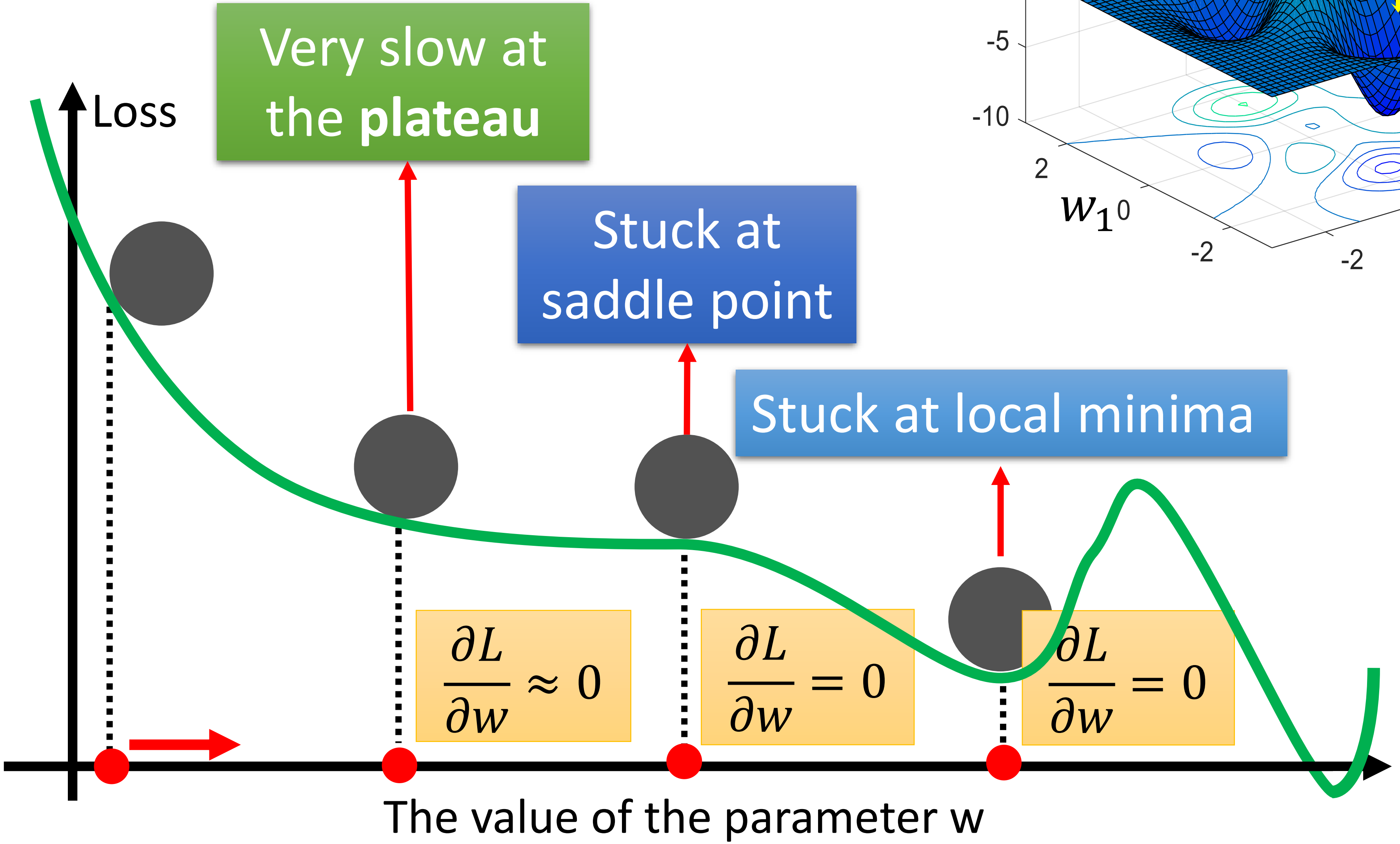
$$\vdots$$

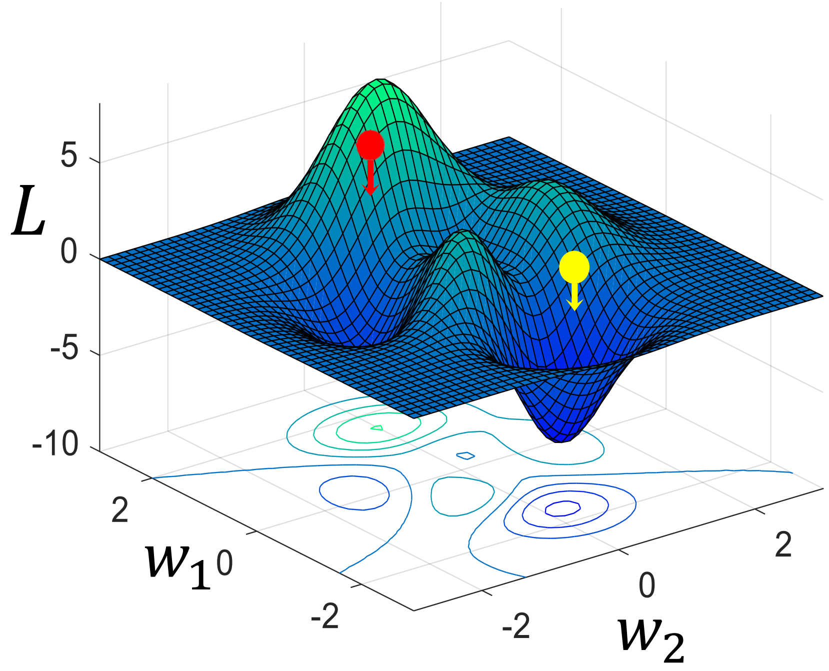
$$w^{(t+1)} \leftarrow w^{(t)} - \frac{\eta}{\sigma^{(t)}} g^{(t)}$$

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

Root Mean Square of the gradients
with previous gradients being decayed

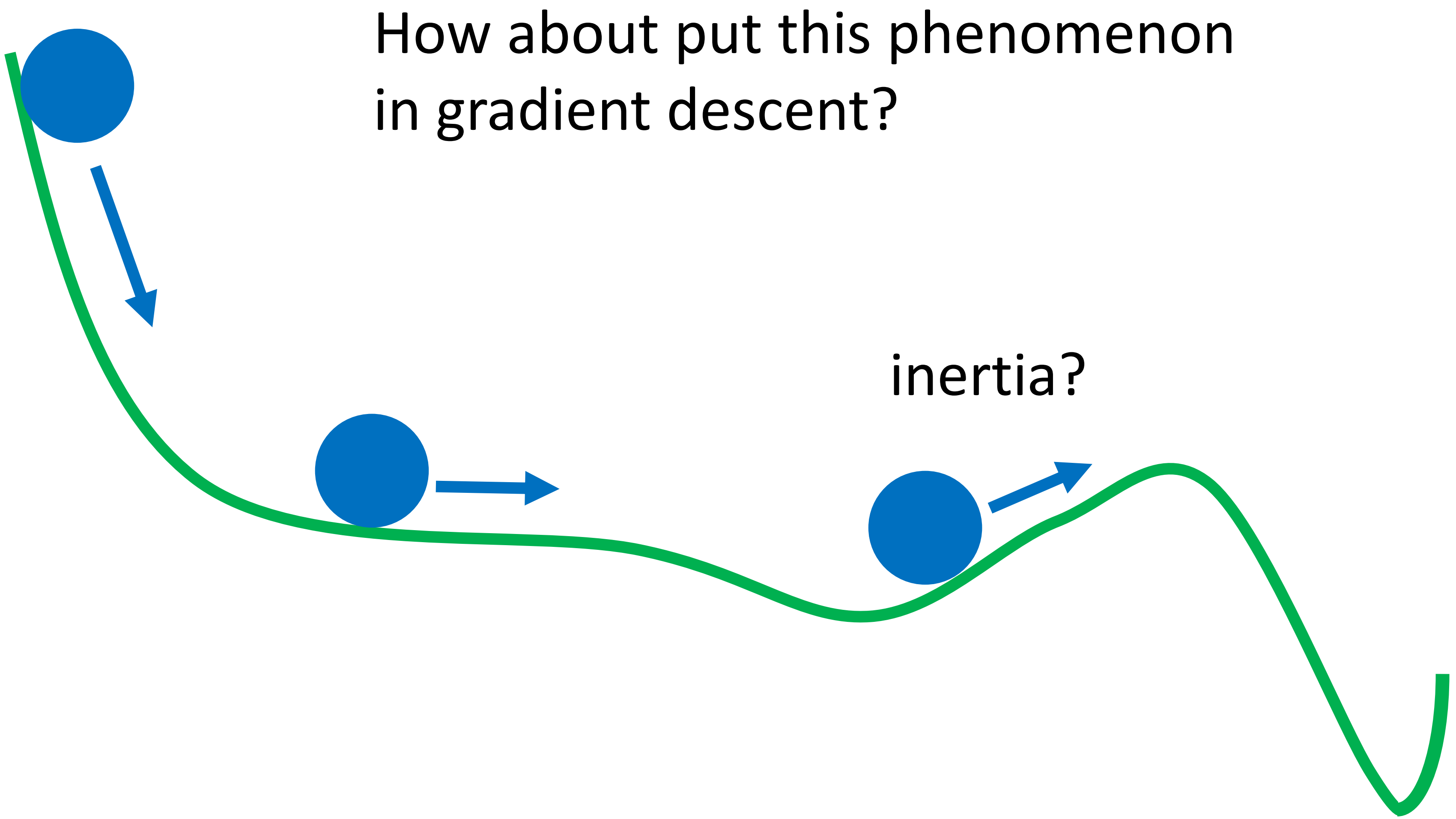
Hard to find optimal network parameters





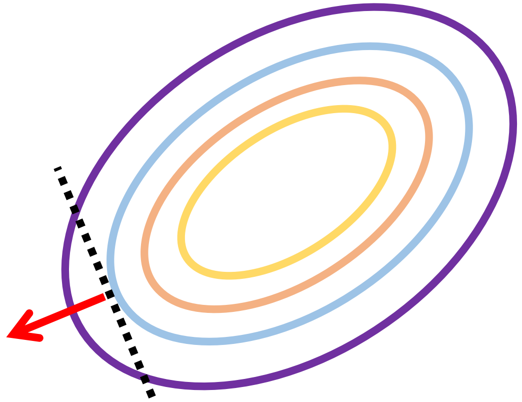
In physical world ……

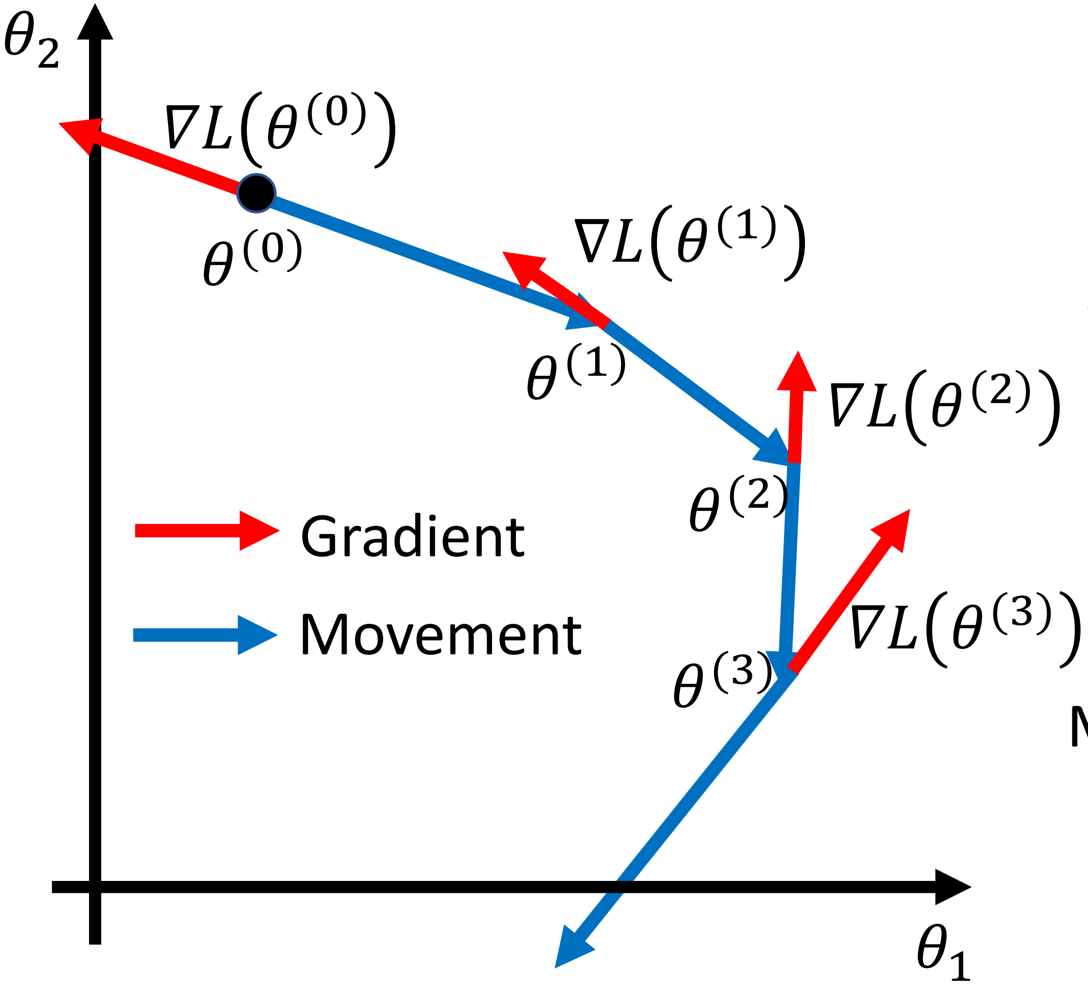
- Momentum



Review: Vanilla Gradient Descent

Gradient: the normal direction of the contour of loss function





Start at position $\theta^{(0)}$

Compute gradient at $\theta^{(0)}$

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

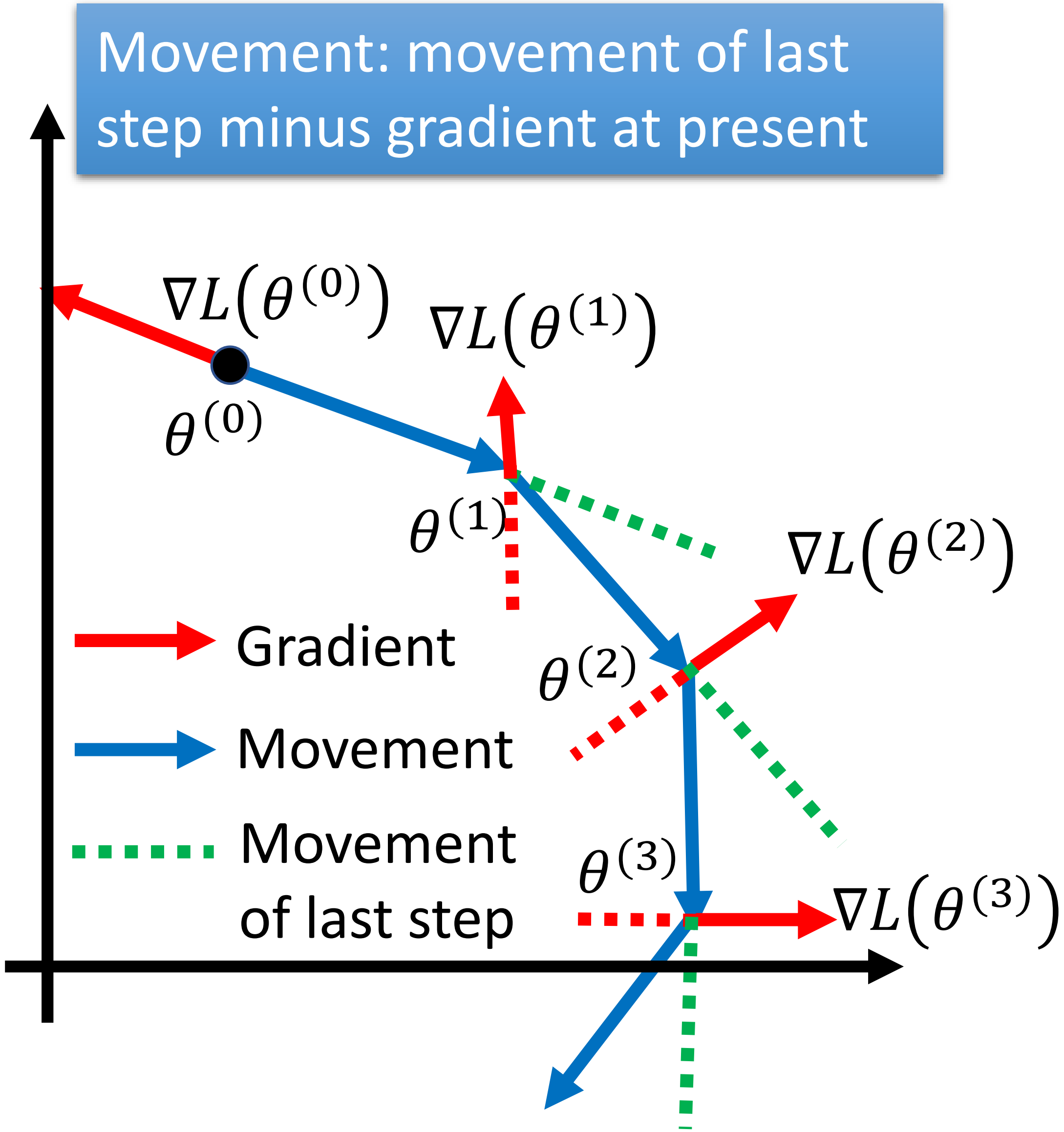
Compute gradient at $\theta^{(1)}$

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

⋮

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

Momentum



Start at point $\theta^{(0)}$

Movement $v^{(0)} = 0$

Compute gradient at $\theta^{(0)}$

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

Move to $\theta^{(1)} = \theta^{(0)} + v^{(1)}$

Compute gradient at $\theta^{(1)}$

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

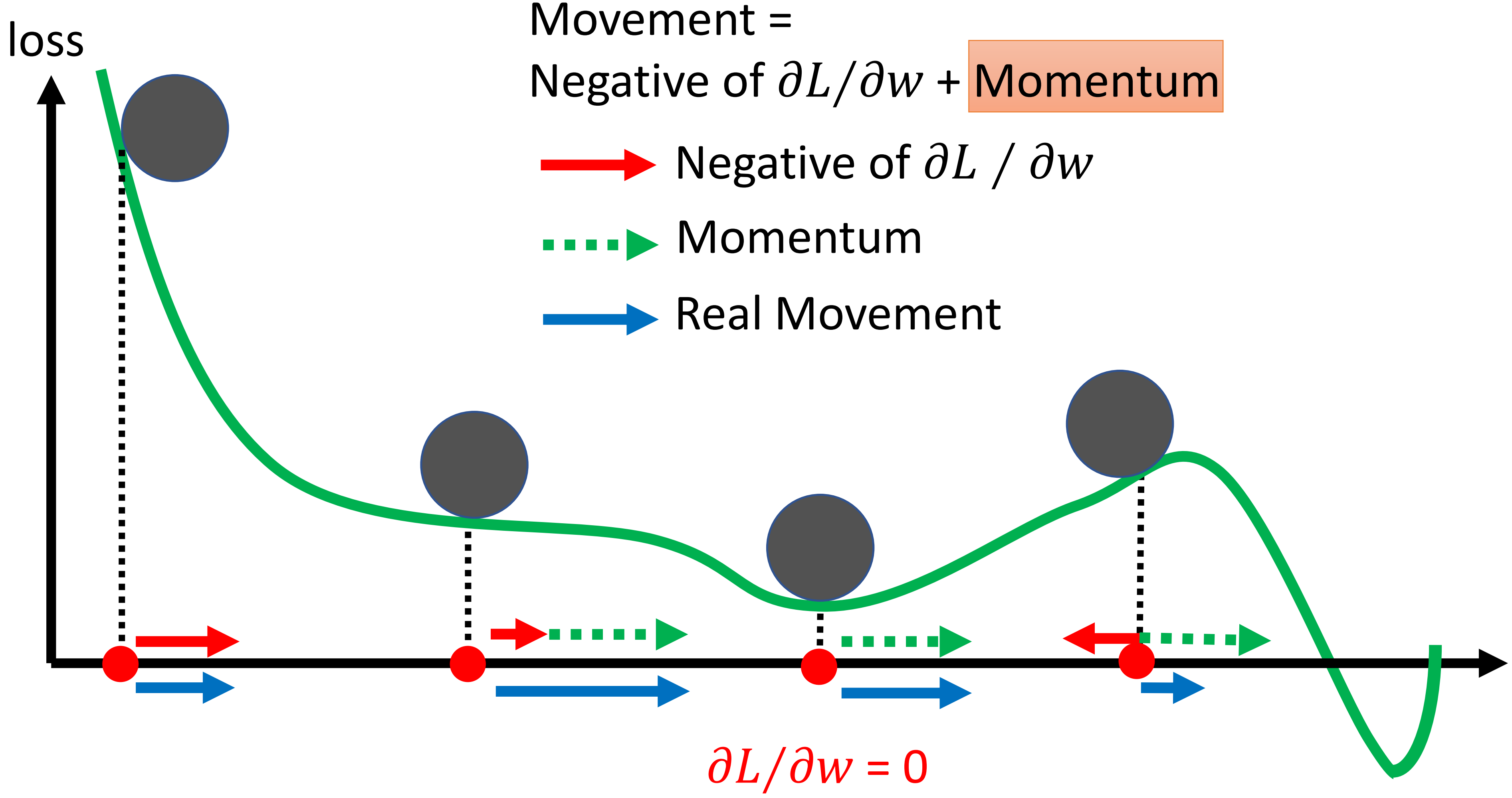
Move to $\theta^{(2)} = \theta^{(1)} + v^{(2)}$

Movement not just based on gradient, but previous movement.

Momentum

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

Movement =
Negative of $\partial L / \partial w$ + Momentum



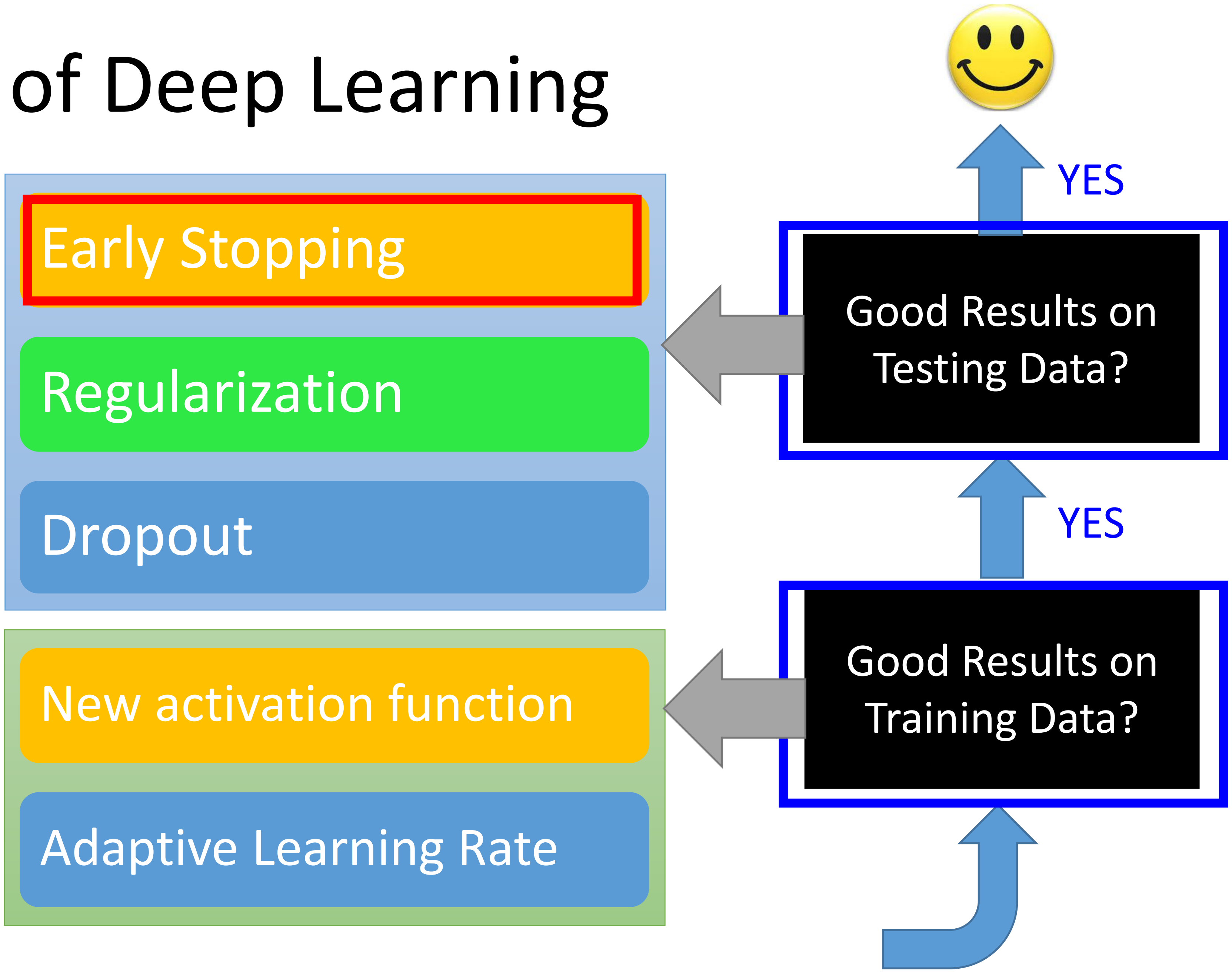
Adam

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.

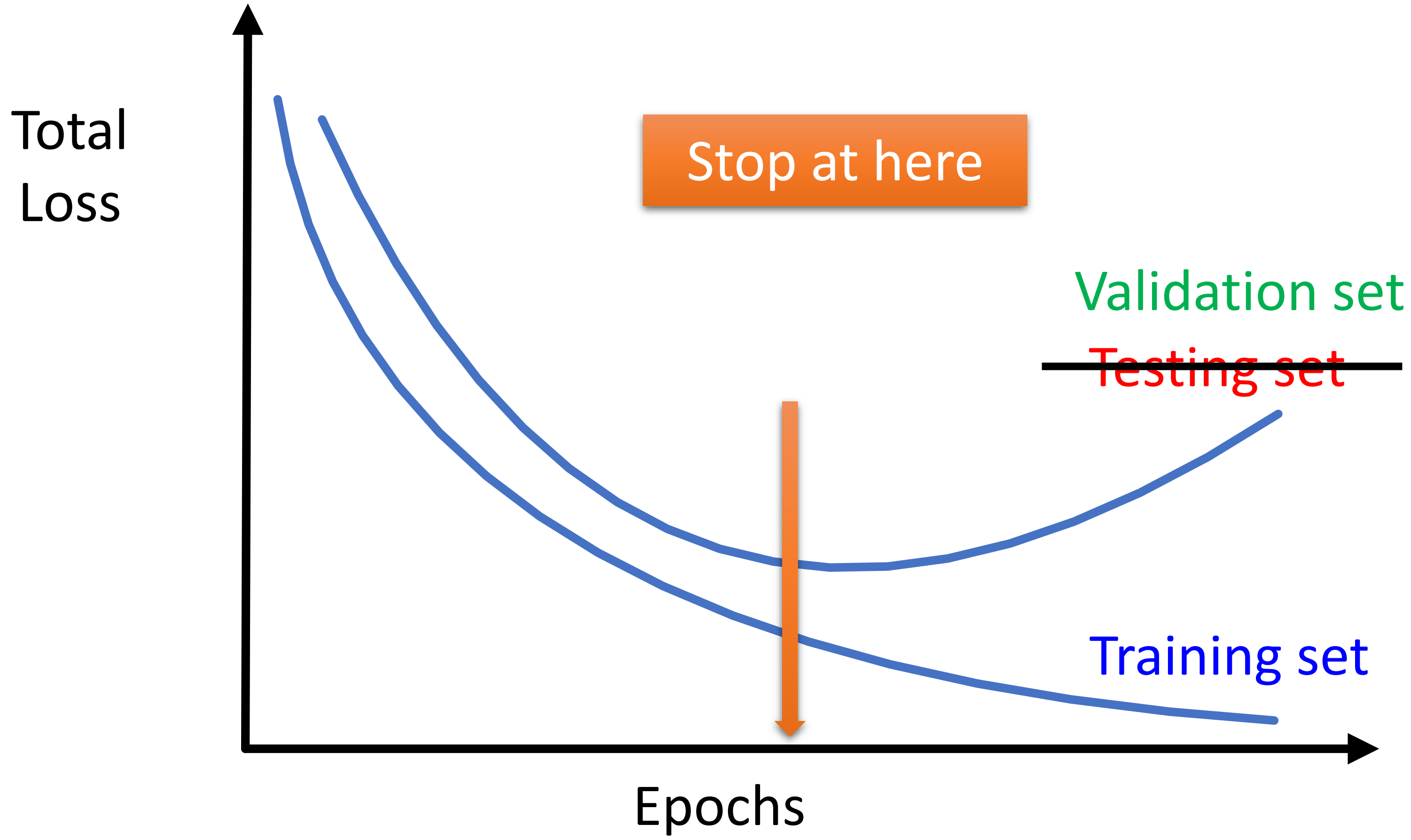
Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector) → for momentum
  $v_0 \leftarrow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

RMSProp + Momentum

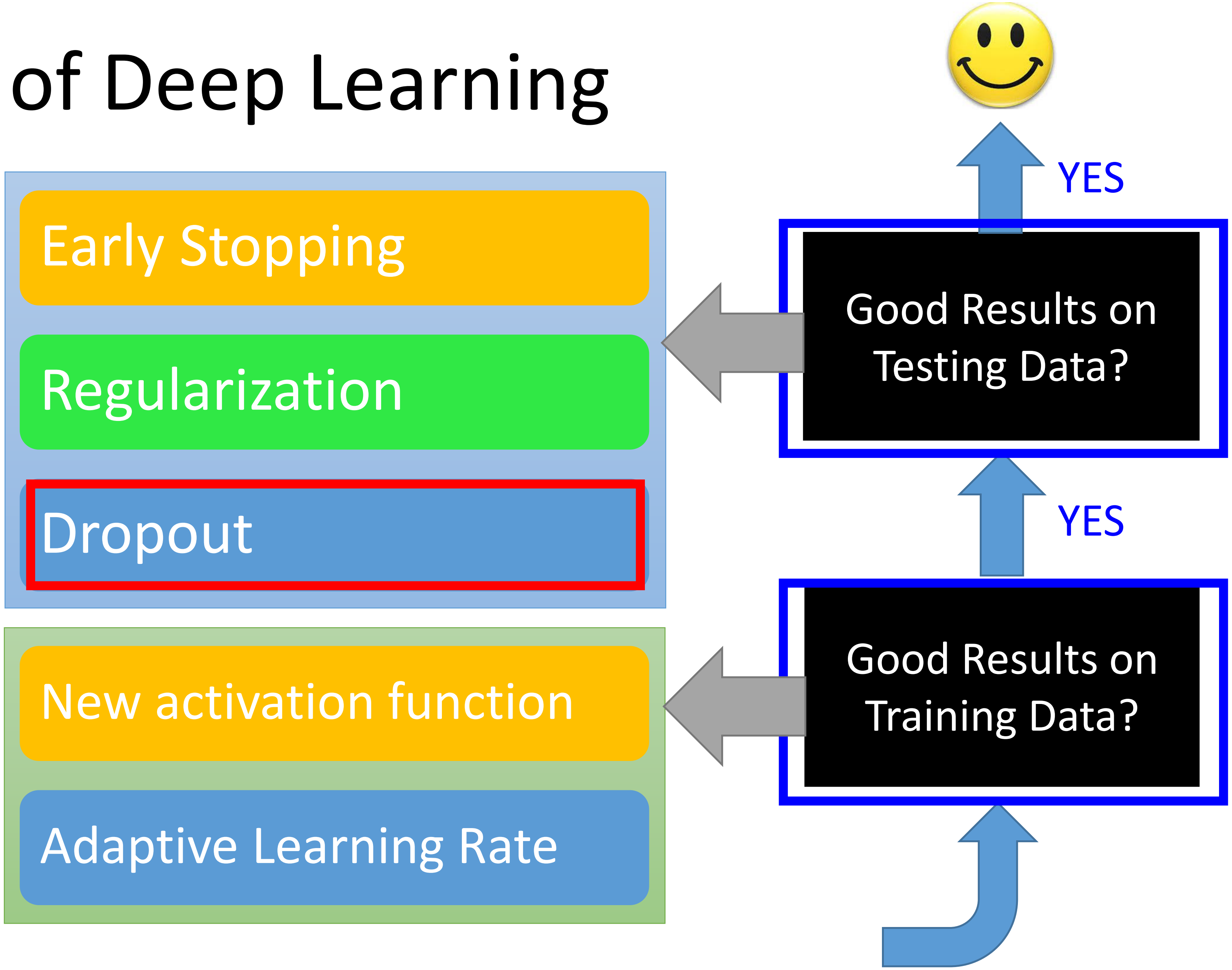
Recipe of Deep Learning



Early Stopping

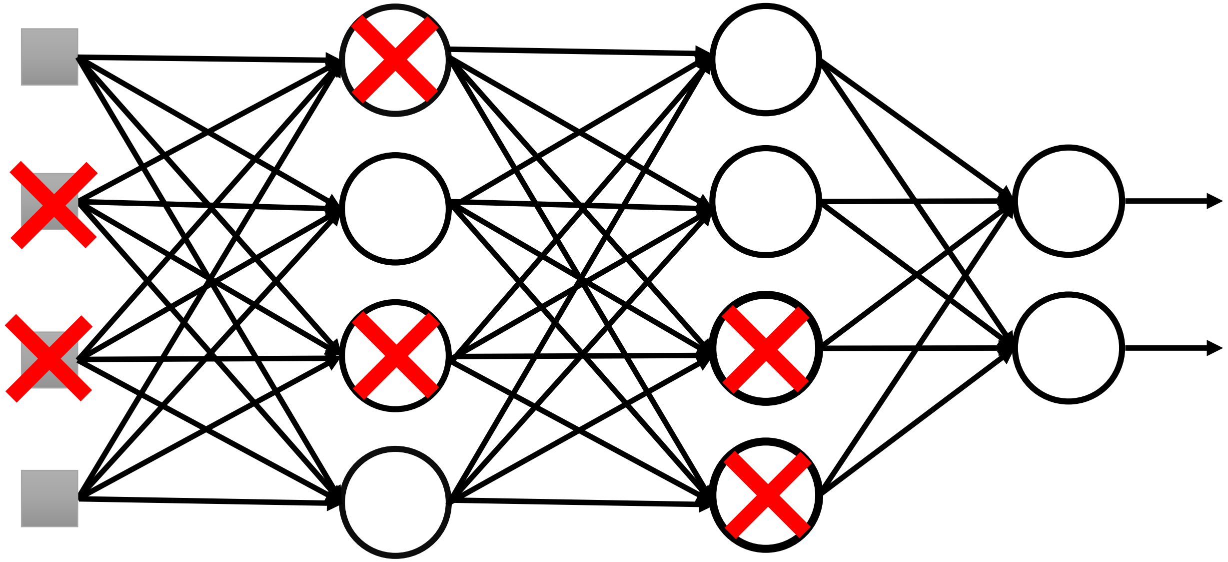


Recipe of Deep Learning



Dropout

Training:

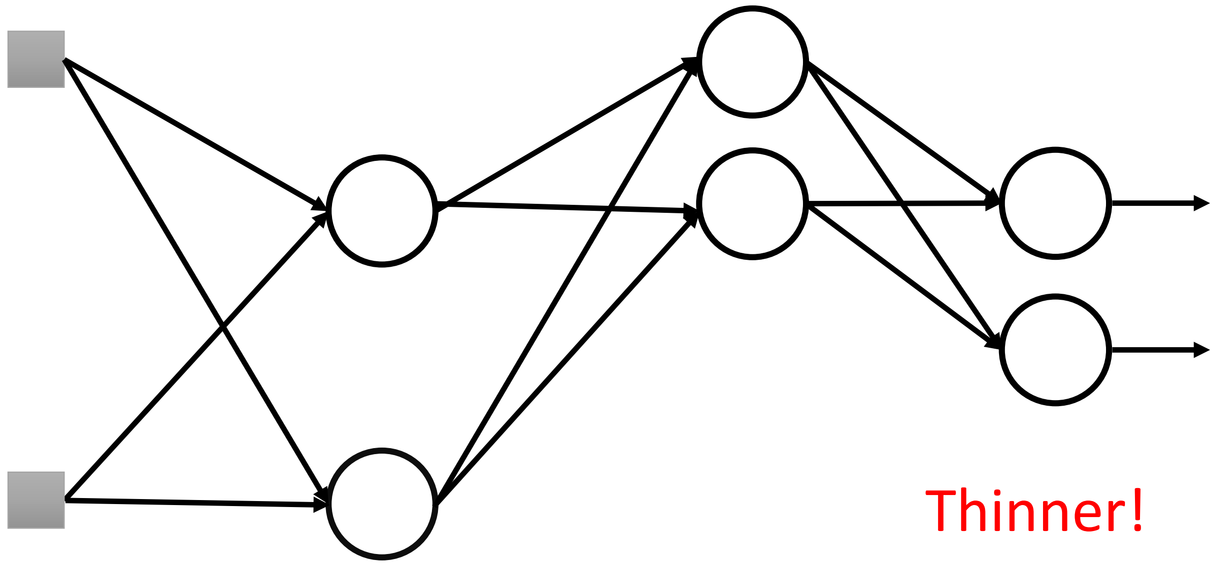


- **Each time before updating the parameters**

- Each neuron has $p\%$ to be preserved, *i.e.* $1 - p\%$ to dropout

Dropout

Training:



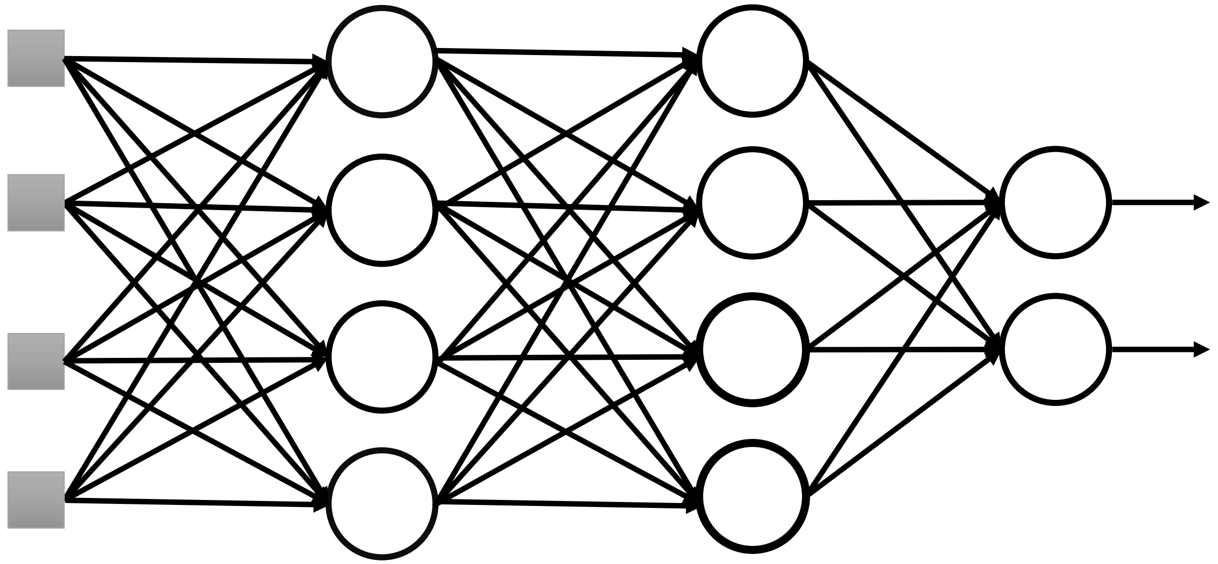
- **Each time before updating the parameters**
  - Each neuron has $p\%$ to be preserved

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

For each minibatch, we resample the dropout neurons

Dropout

Testing:



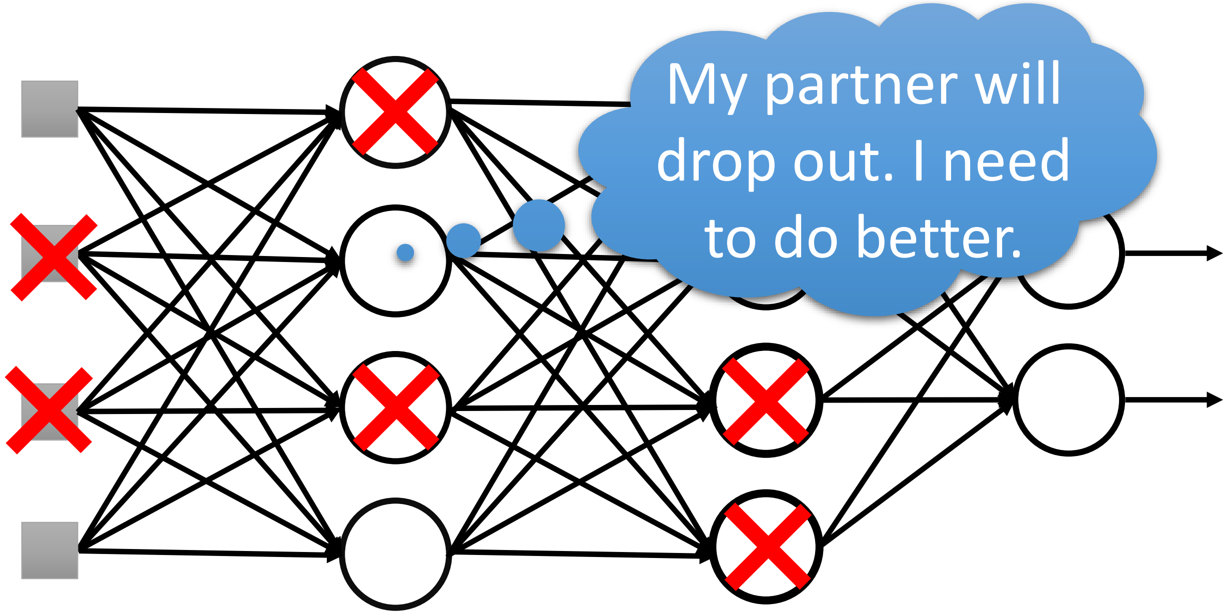
No dropout

- If the keep rate at training is $p\%$, all the weights at testing times $p\%$
- Assume that the keep rate is 50%.
- If a weight $w = 1$ by training, set $w = 0.5$ for testing.

Dropout: Intuitive Reason

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Nitish Srivastava NITISH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Alex Krizhevsky KRIZ@CS.TORONTO.EDU
Ilya Sutskever ILYA@CS.TORONTO.EDU
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
10 Kings College Road, Rm 3302
Toronto, Ontario, M5S 3G4, Canada.



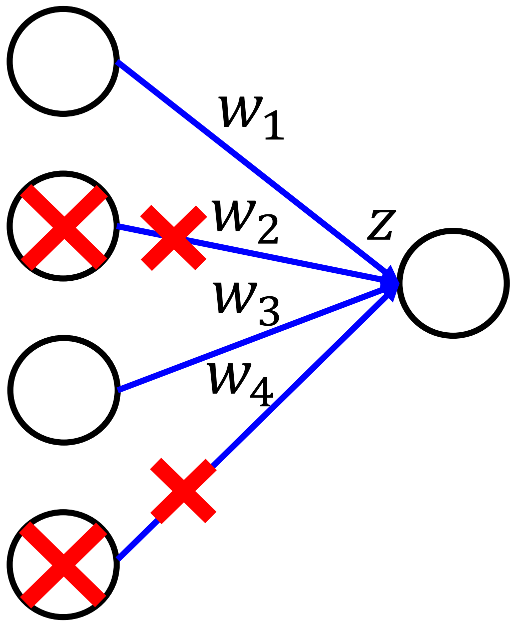
- When teams up, if everyone expect the partner will do the work, nothing will be done finally.
- However, if you know your partner will dropout, you will do better.
- When testing, no one dropout actually, so obtaining good results eventually.

Dropout: Intuitive Reason

- Why the weights should multiply $p\%$ ($p\%$ is the keep rate) when testing?

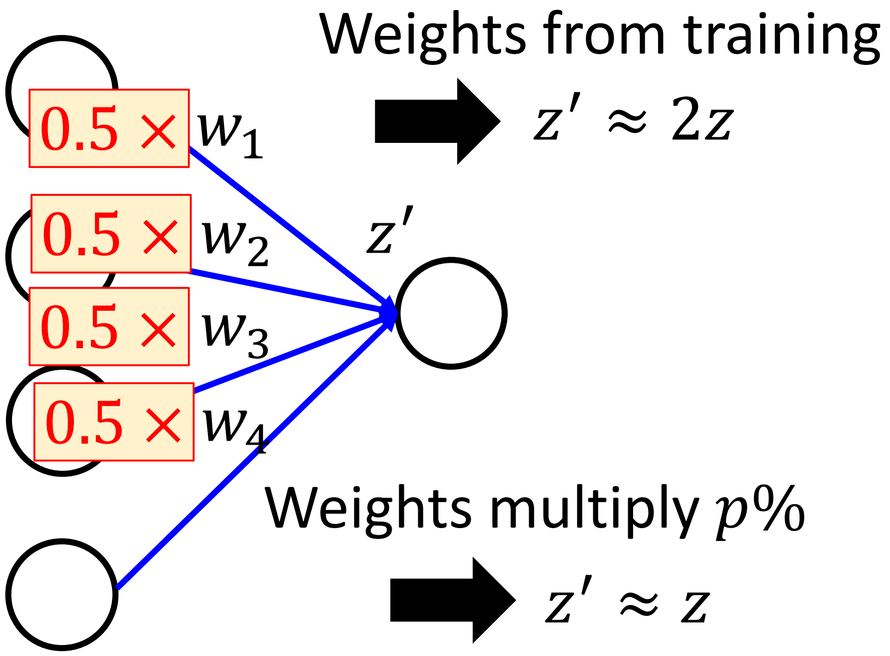
Training of Dropout

Assume keep rate is 50%

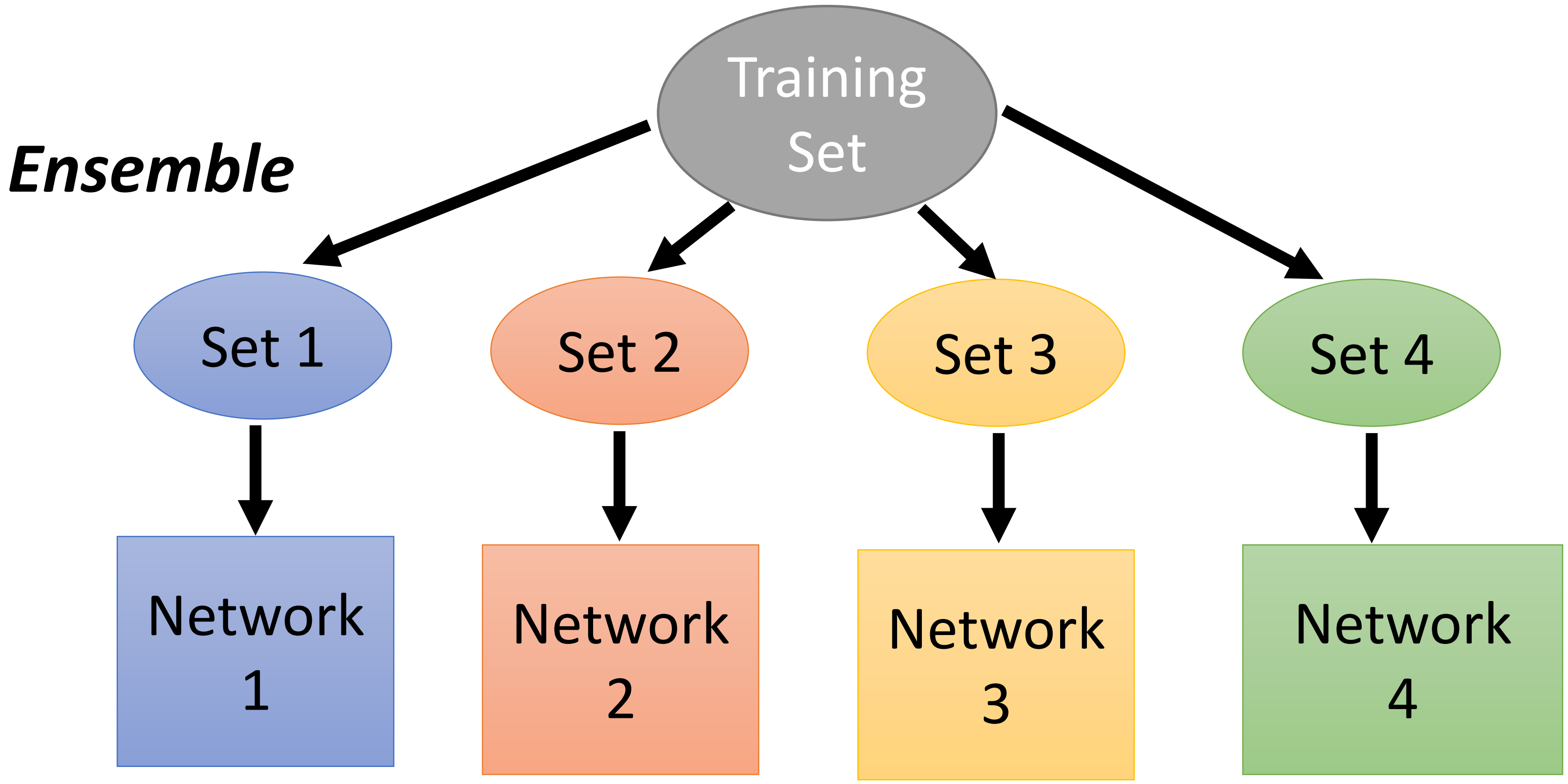


Testing of Dropout

No dropout



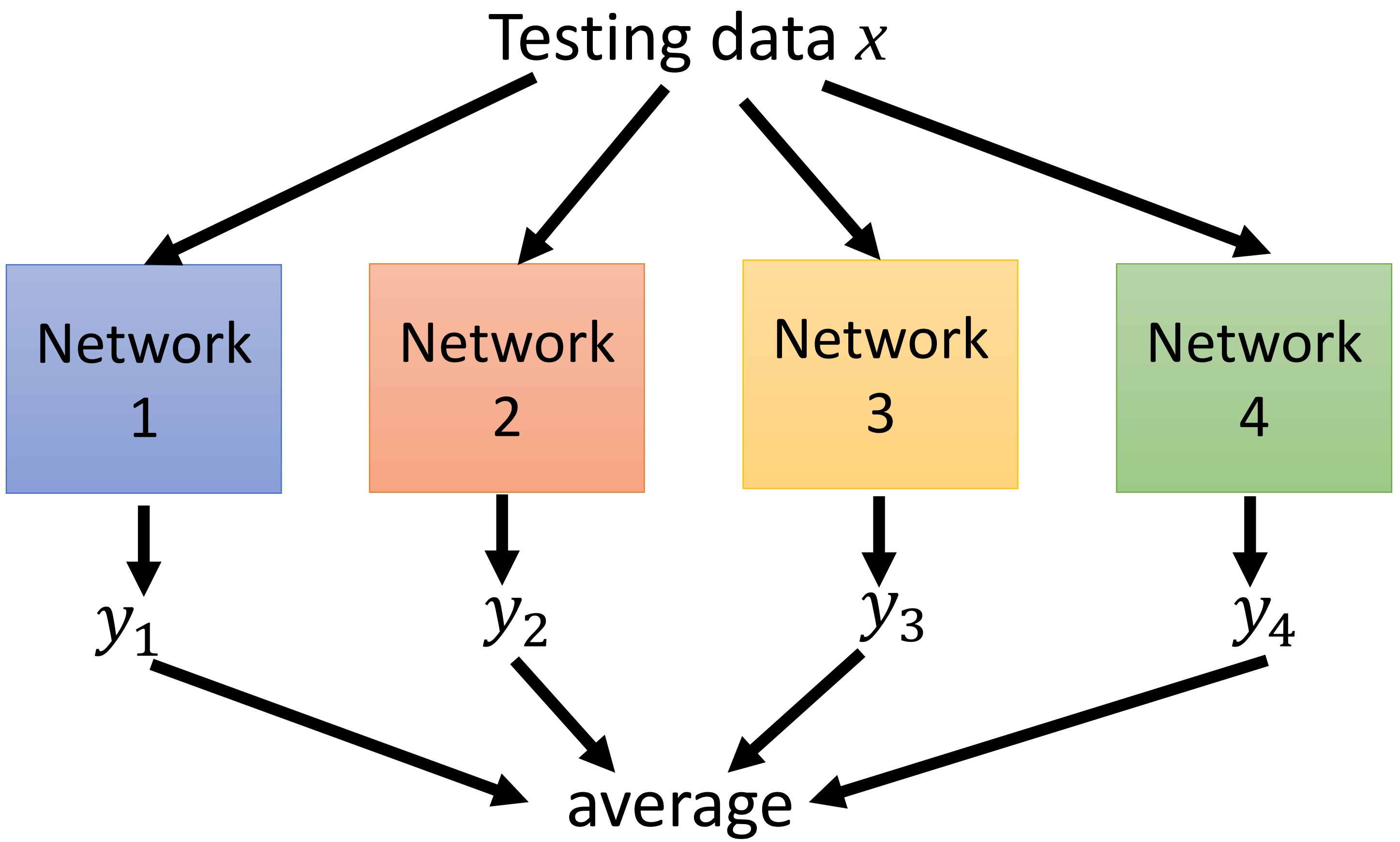
Dropout is a kind of ensemble.



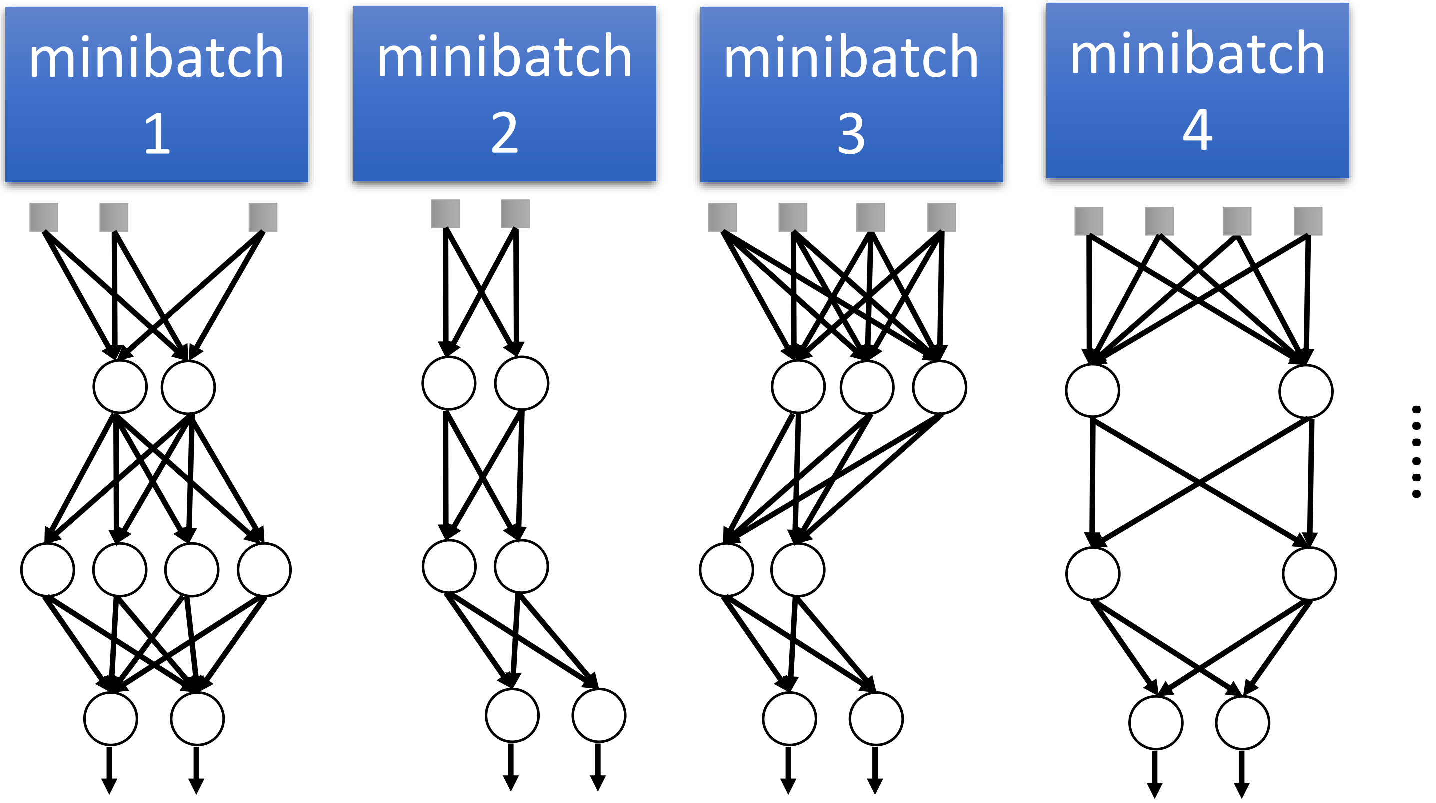
Train a bunch of networks with different structures

Dropout is a kind of ensemble.

Ensemble



Dropout is a kind of ensemble.



Training of Dropout

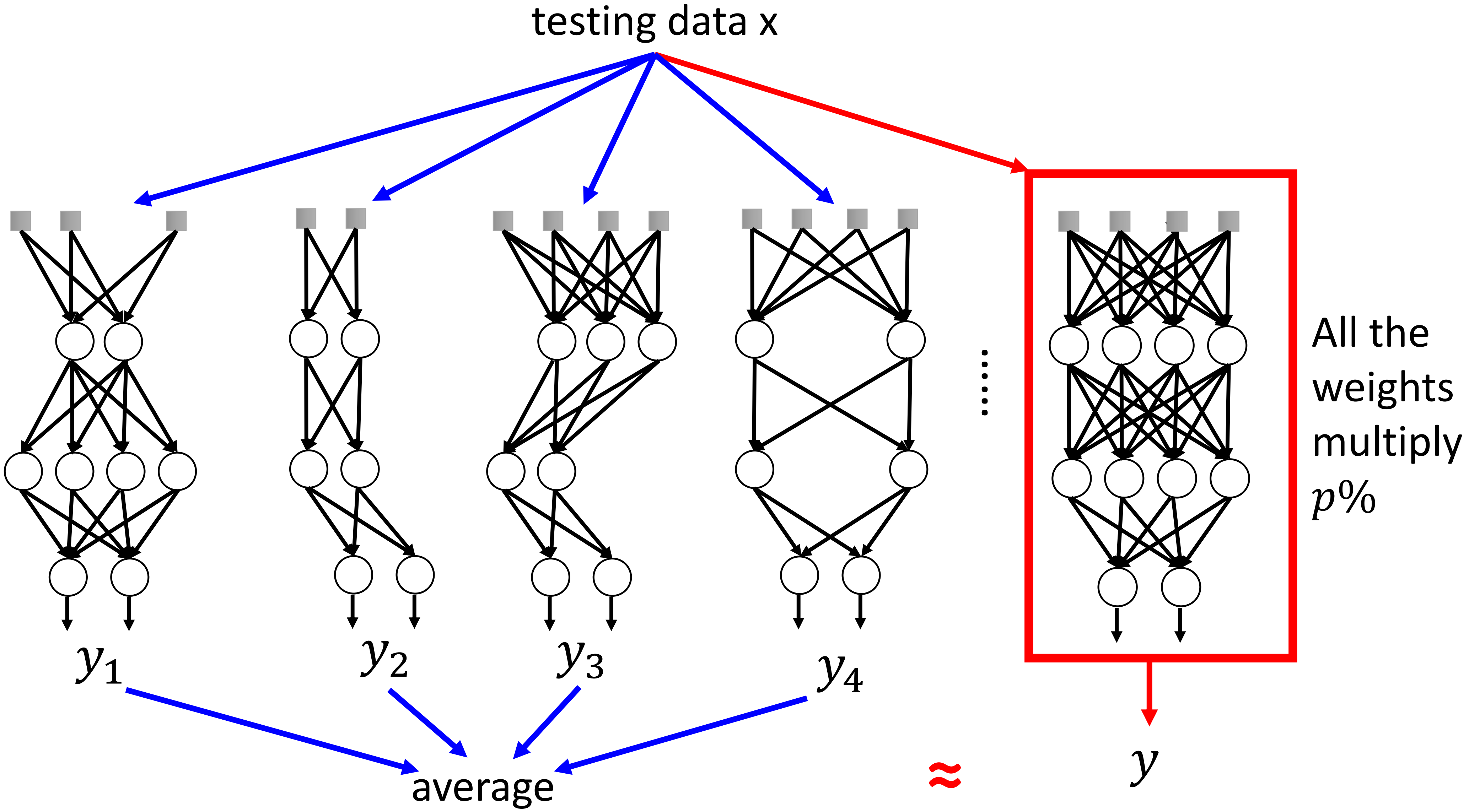
M neurons

2^M possible networks

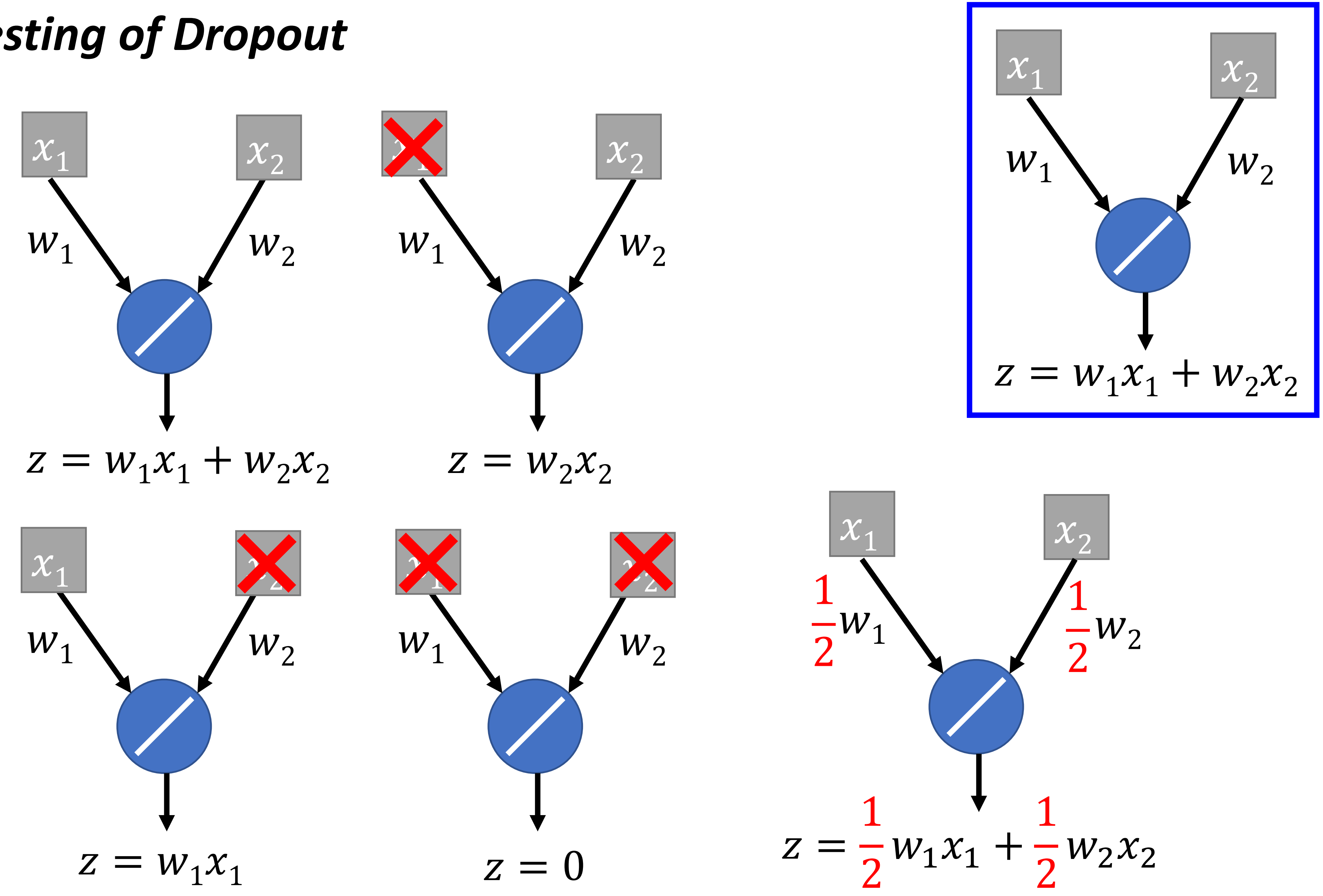
- Using one minibatch to train one network
- Some parameters in the network are shared

Dropout is a kind of ensemble.

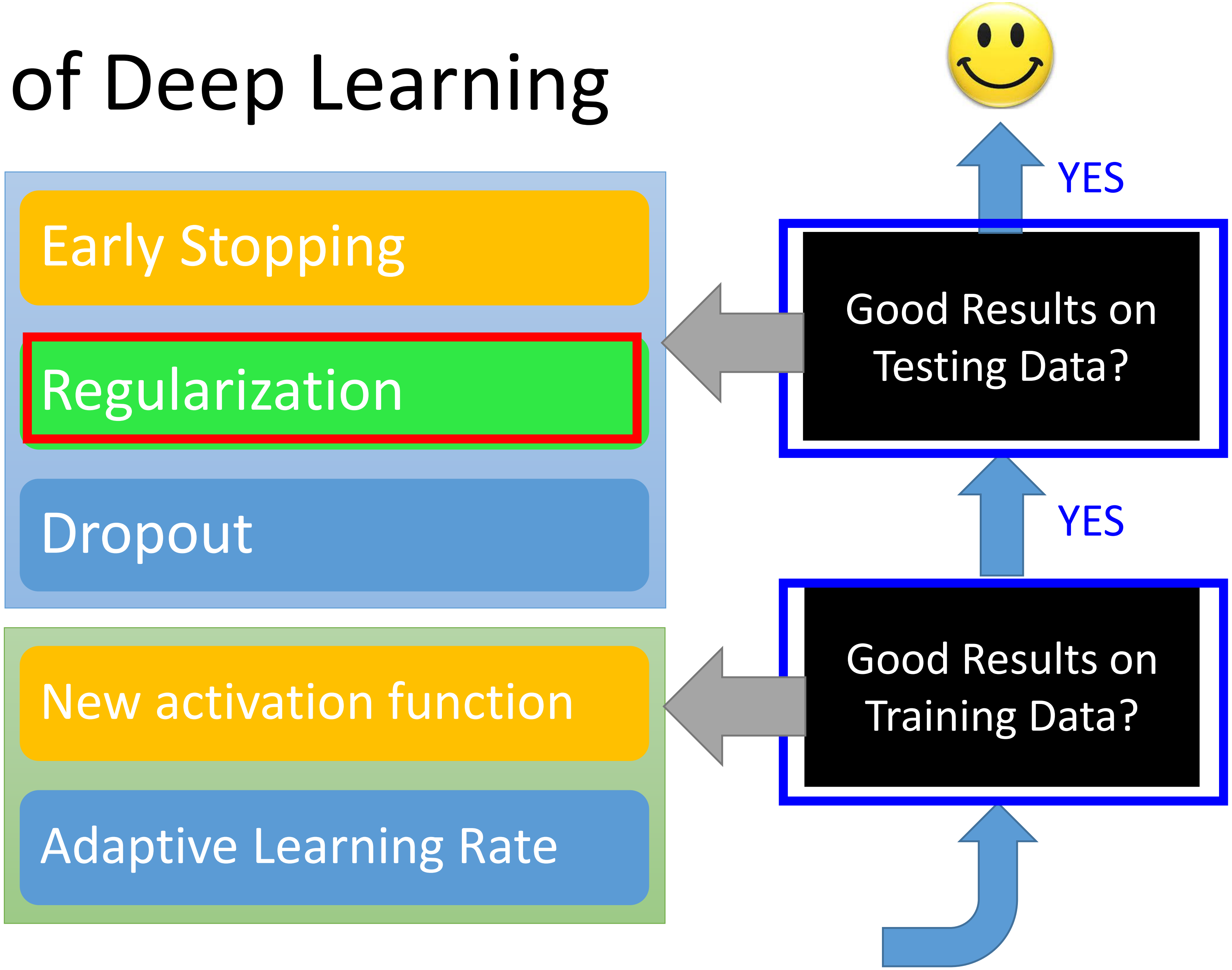
Testing of Dropout



Testing of Dropout



Recipe of Deep Learning



Regularization

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

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

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

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

L2 regularization:

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

(usually not consider biases)

Regularization

L2 regularization:
$\|\theta\|_2 = (w_1)^2 + (w_2)^2 + \cdots$

- New loss function to be minimized

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

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

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

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

Weight Decay

Closer to zero

Regularization

L1 regularization:

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

- 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:

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

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

---

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

Regularization: L1 vs. L2

L1 regularization: $\|\theta\|_1 = |w_1| + |w_2| + \cdots$

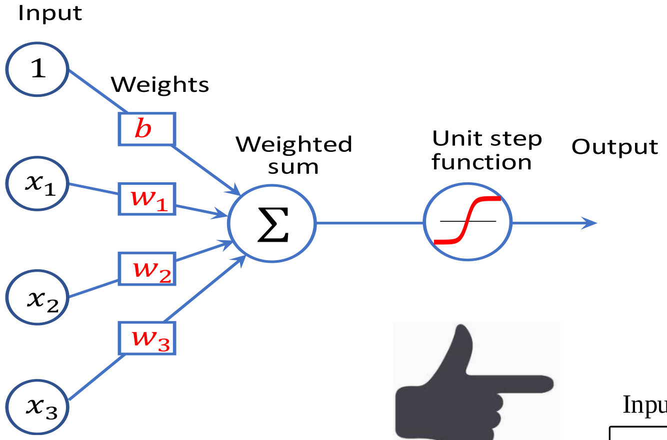
$$w^{(t+1)} = w^{(t)} - \eta \frac{\partial \mathrm{L}}{\partial w} - \eta\lambda \,\mathrm{sgn}\left(w^{(t)}\right)$$

L2 regularization: $\|\theta\|_2 = (w_1)^2 + (w_2)^2 + \cdots$

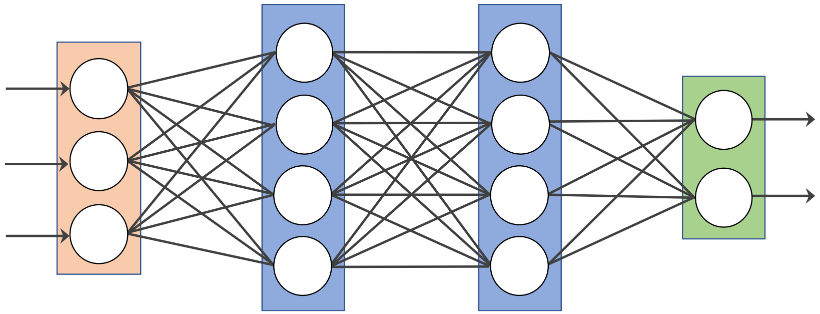
$$w^{(t+1)} = (1 - \eta\lambda) w^{(t)} - \eta \frac{\partial \mathrm{L}}{\partial w}$$

In this Course

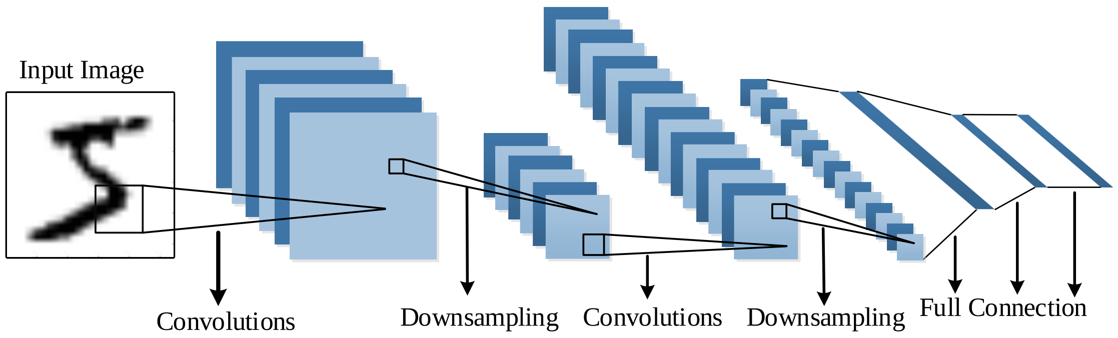
1. DL basics, linear regression, logistic regression etc.



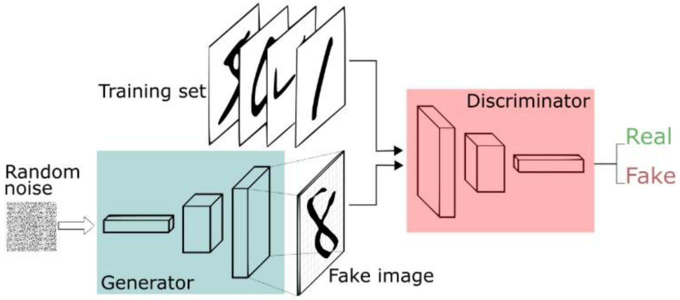
2. Multilayer neural networks, backpropagation



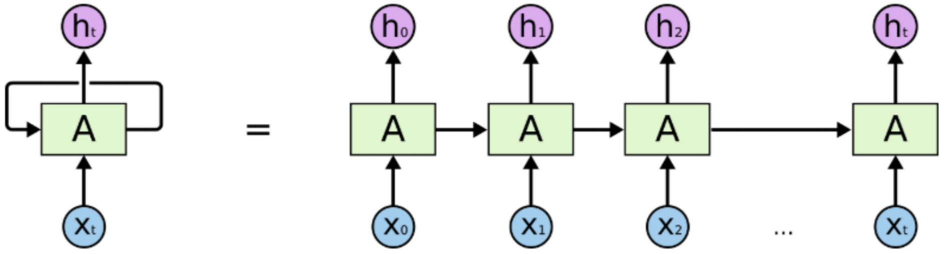
3. Convolutional Neural Networks and Applications



4. Generative Adversarial Networks



5. Recurrent networks and applications




THANK
YOU