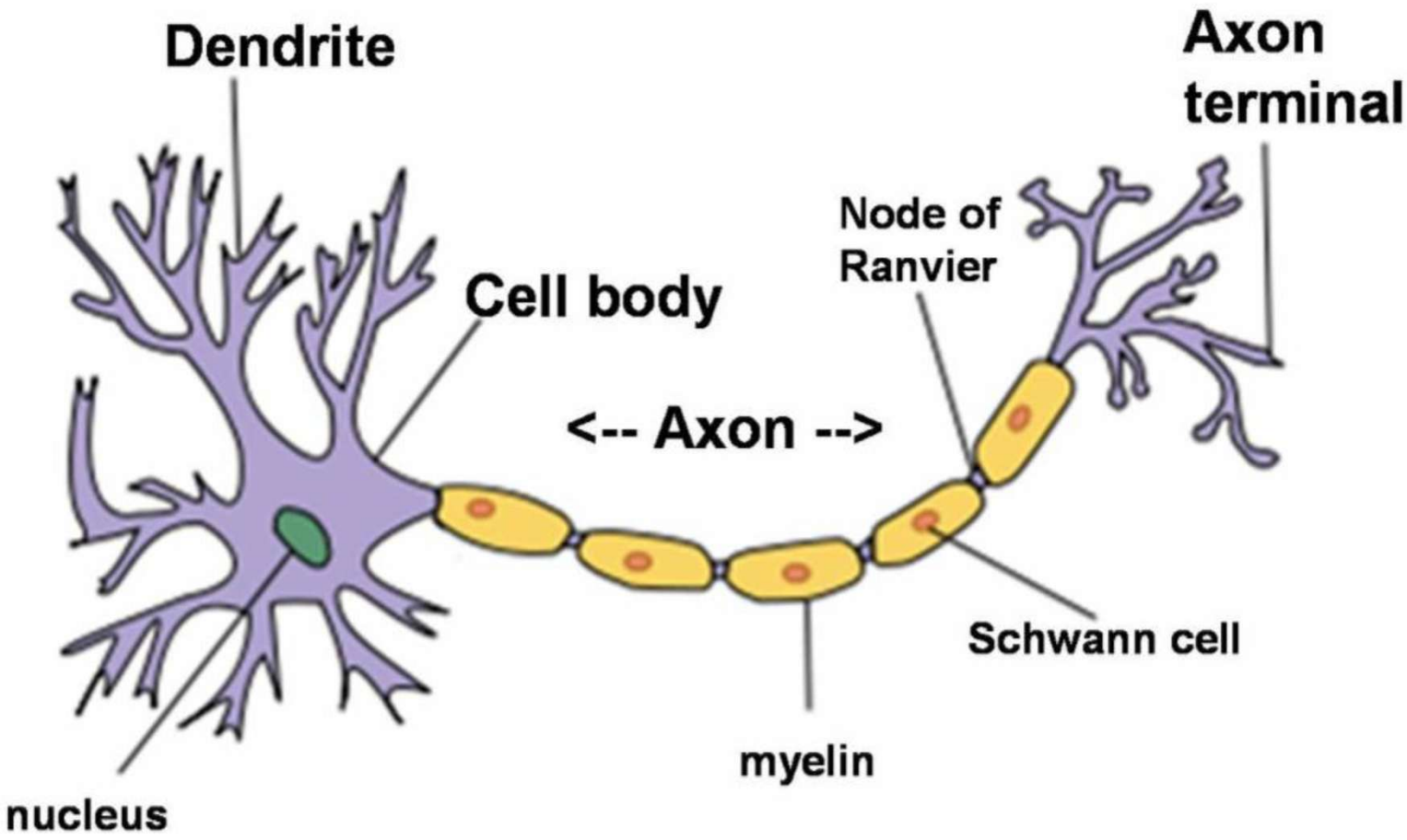
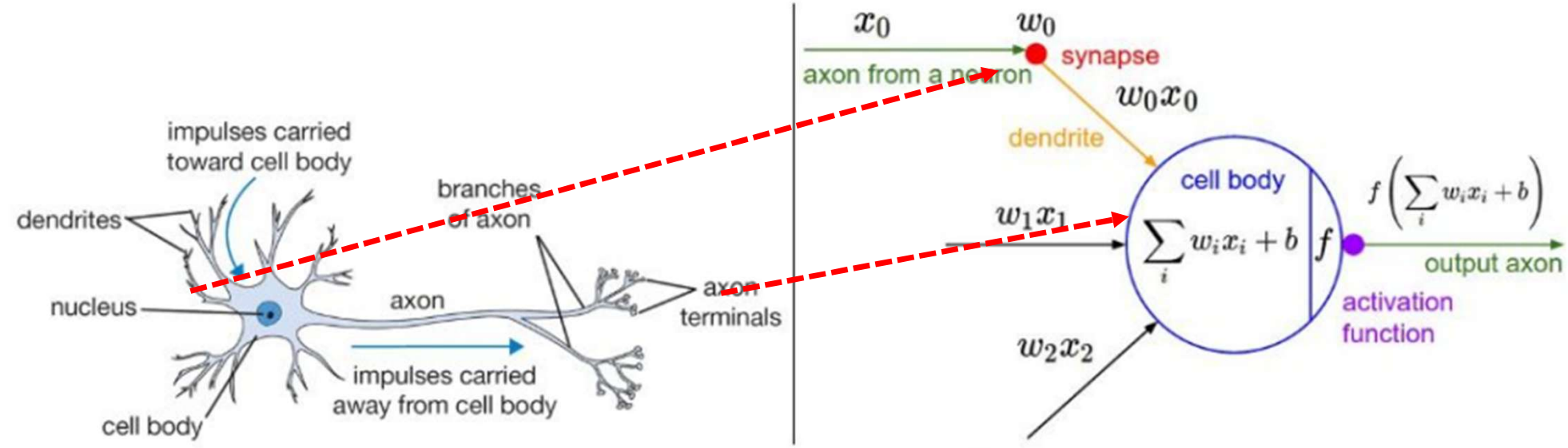
Multi Layer Perceptron

Structure of Neuron



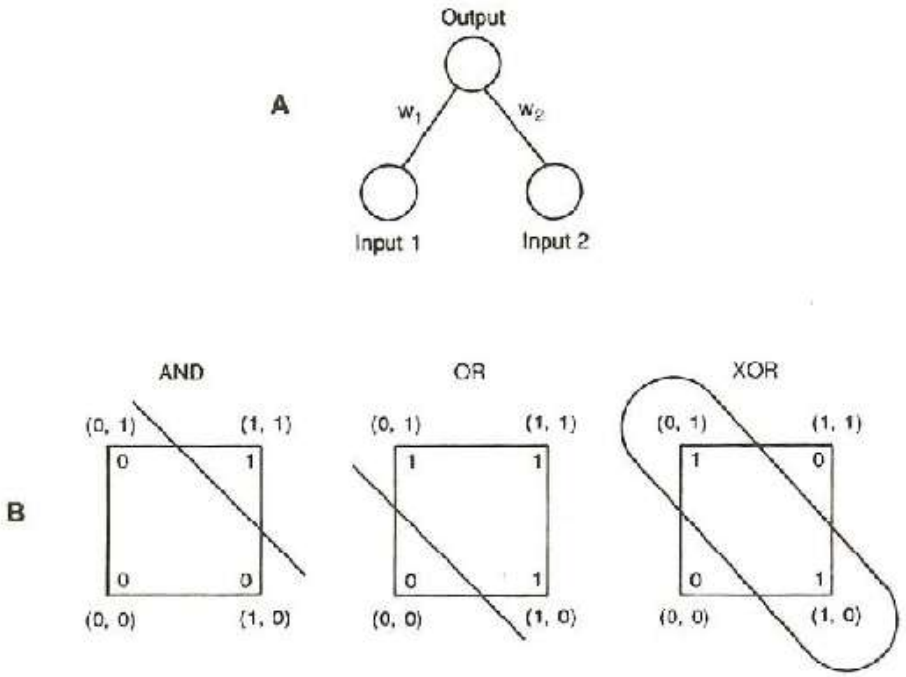
Modeling Neuron work flow



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

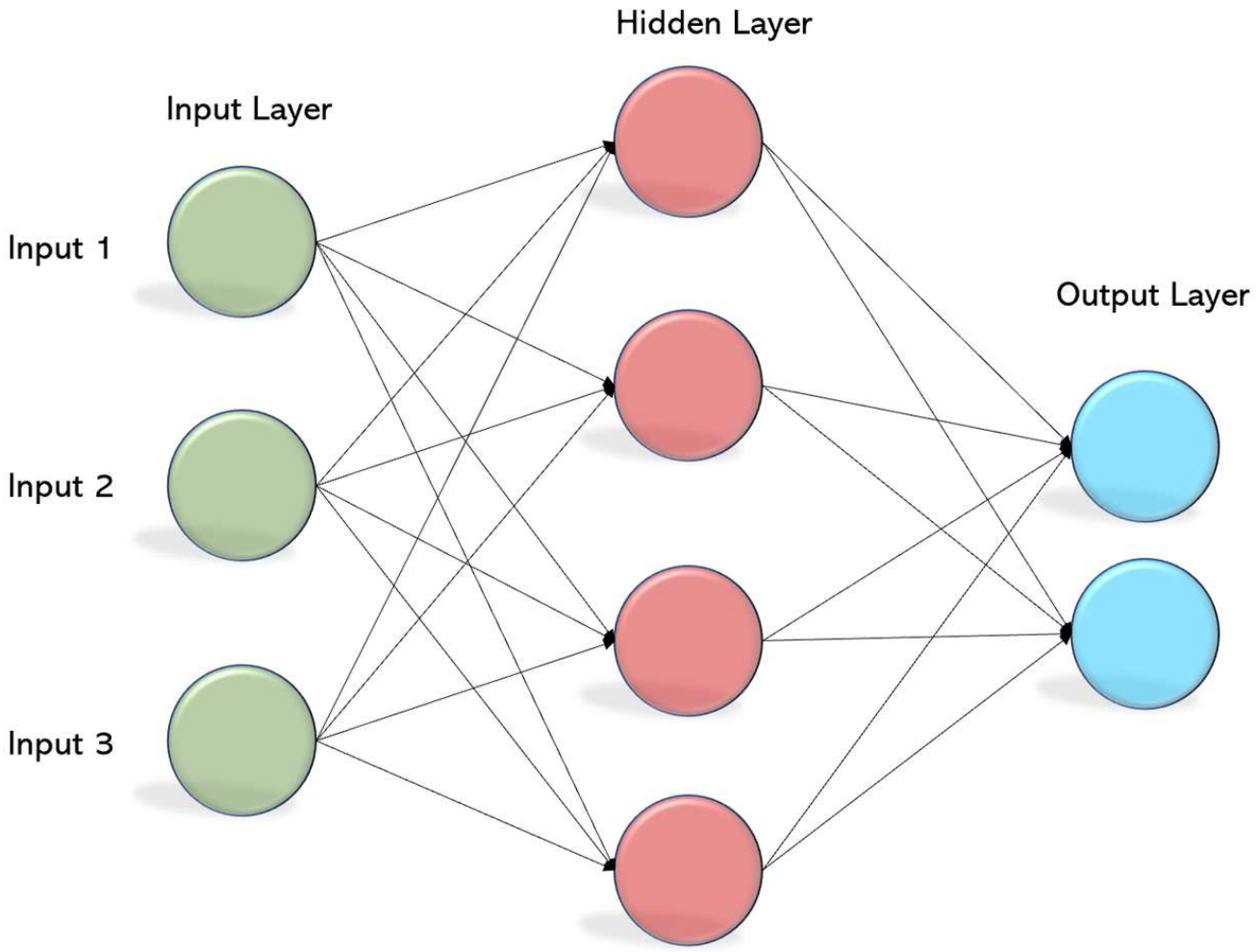
And/Or problem

Input Patterns		Output Patterns
00	→	0
01	→	1
10	→	1
11	→	0

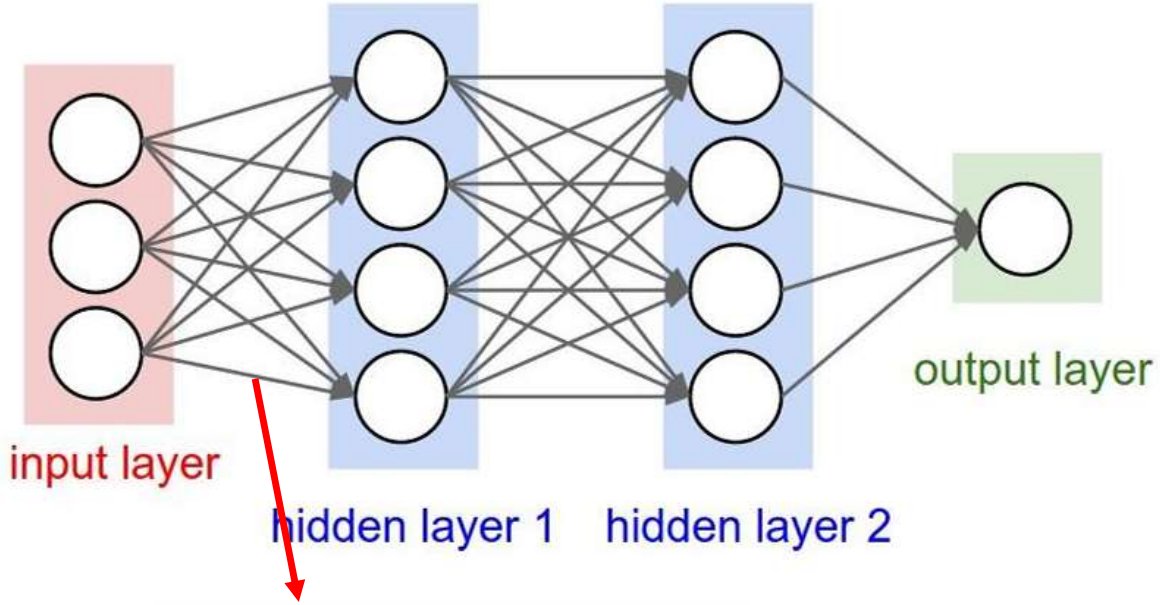


XOR can be solved by Multilayer perceptron: at least one hidden layer (proved mathematically)

Multilayer Perceptron



MLP



$$H(X) = G(XW + b)$$

-Linear regression

$$\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_1 w_1 + x_2 w_2 + x_3 w_3 \end{pmatrix}$$

$$H(X) = XW$$

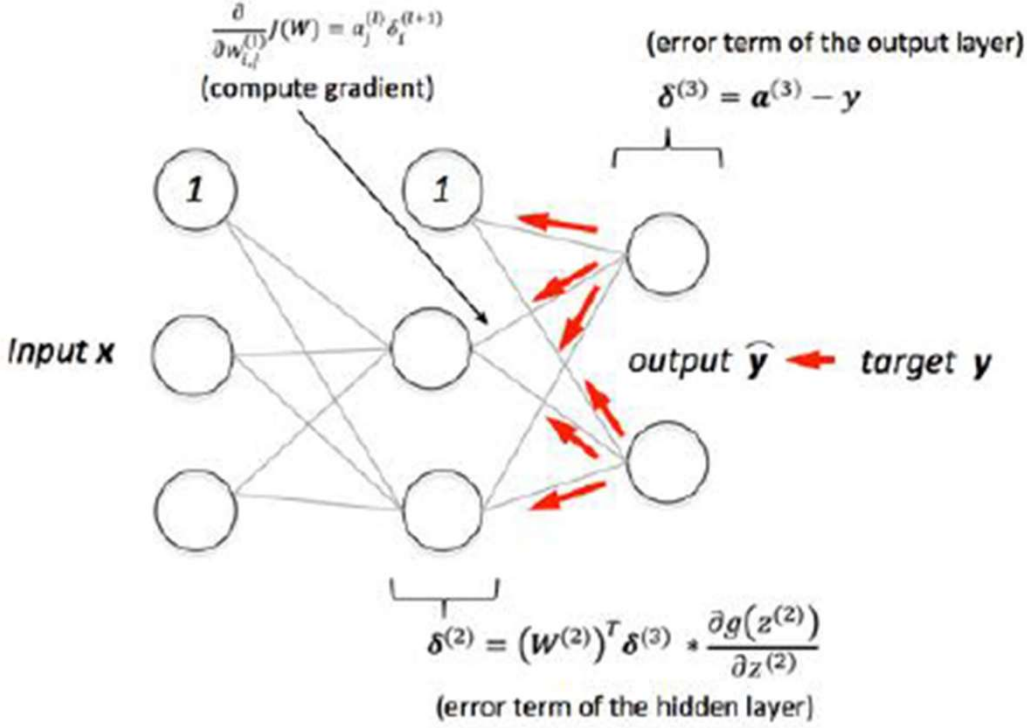
-Logistic regression

	Binary	Multinomial
Hypothesis	$Sigmoid(WX)$	$Softmax(WX)$

Neural network & Deep learning

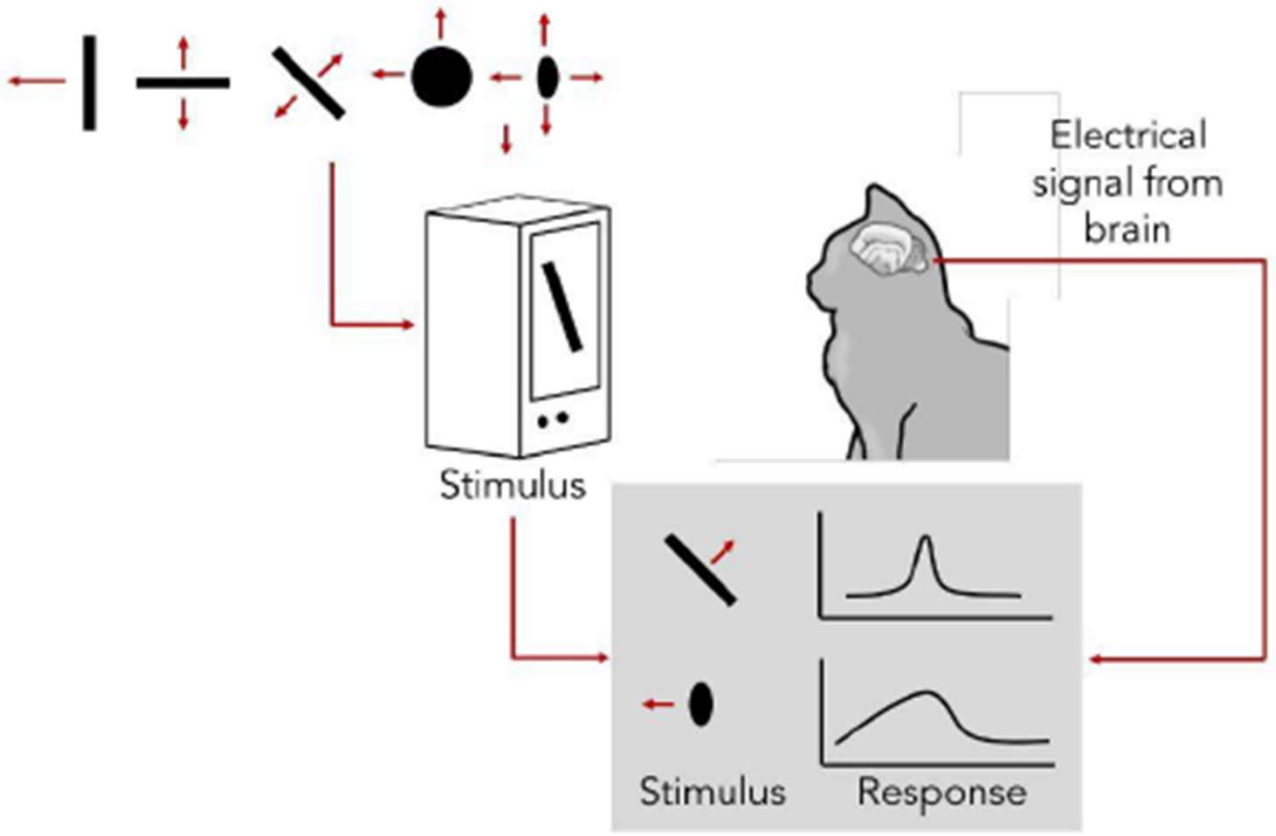
- Neural network: input → one hidden layer → output
(only feed forward)
- Deep learning: more than 2 hidden layers + back propagation

Back propagation

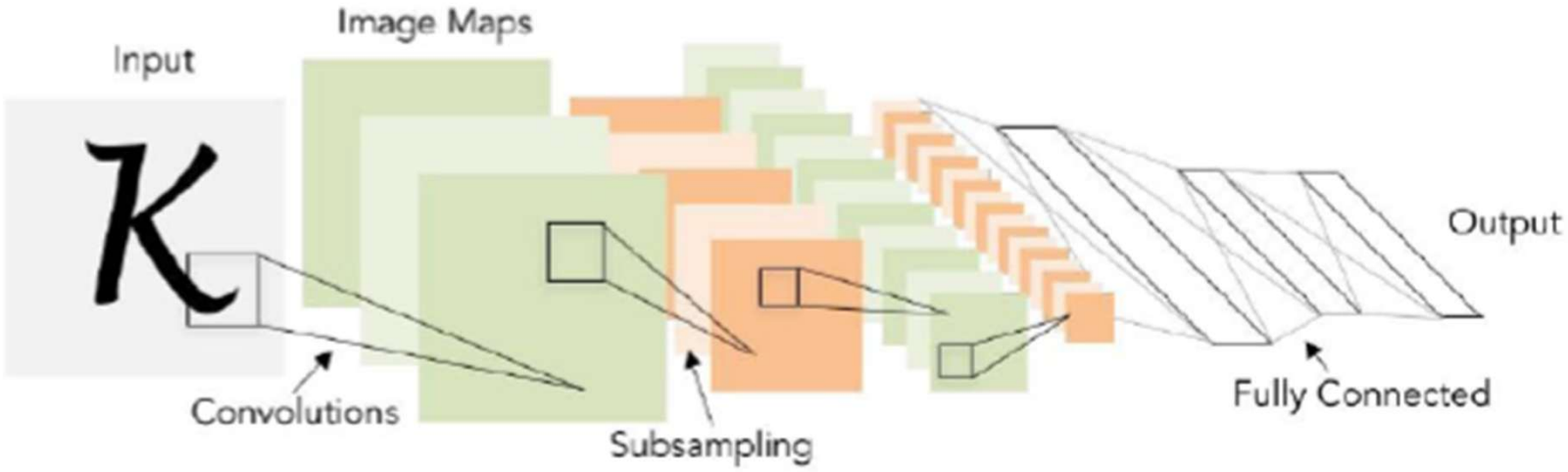


Back propagation → measure error → gradient descent algorithm
→ update parameters (training !)

Example (Image training)

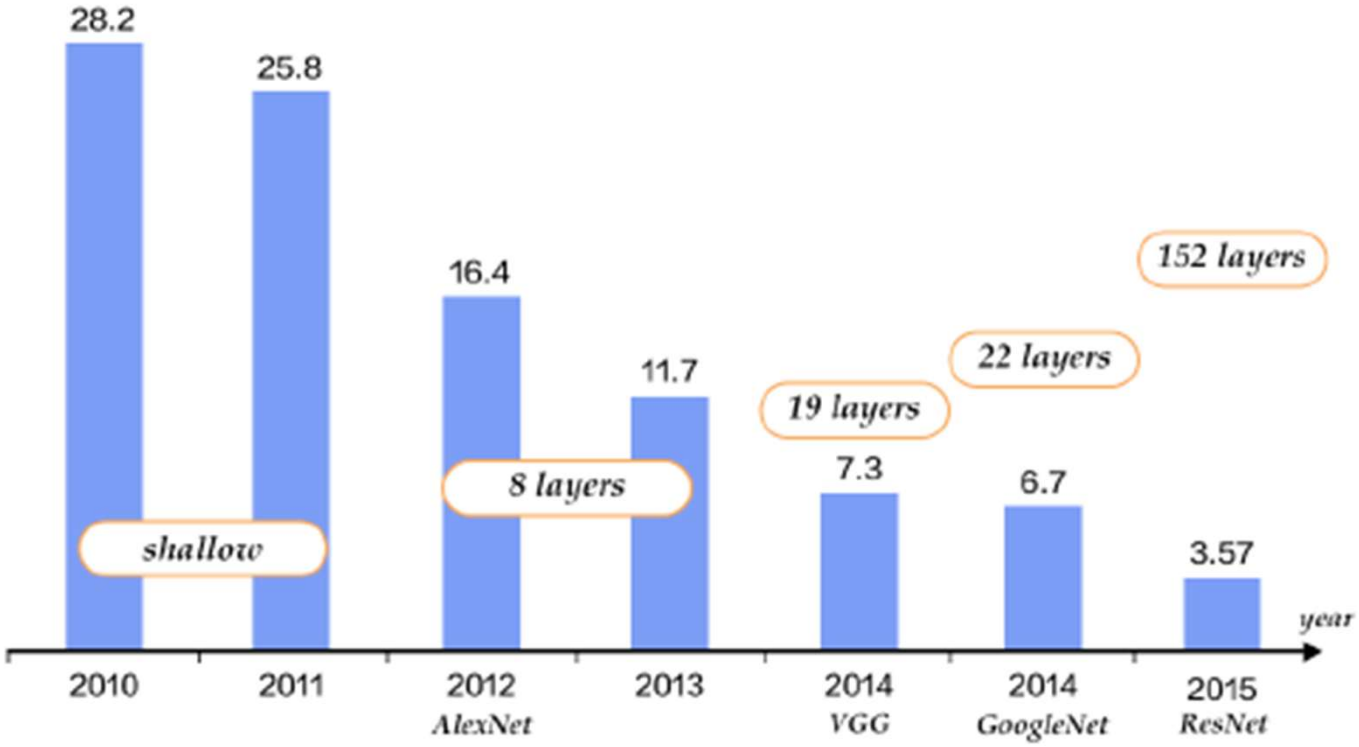


Example (Convolutional neural networks)



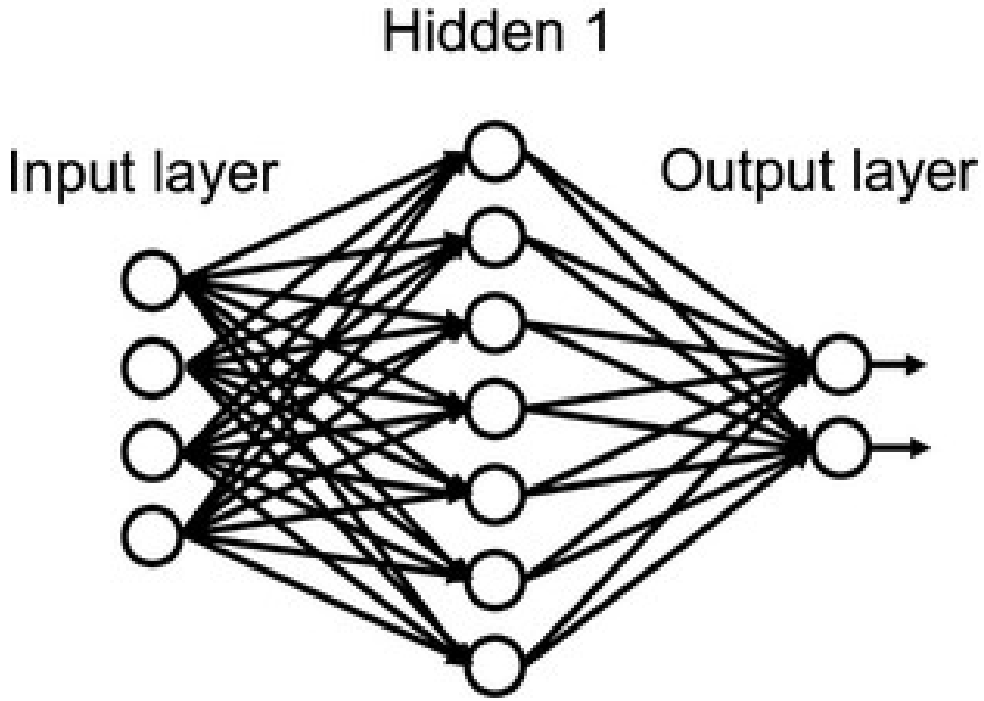
LeNet-5

ImageNet

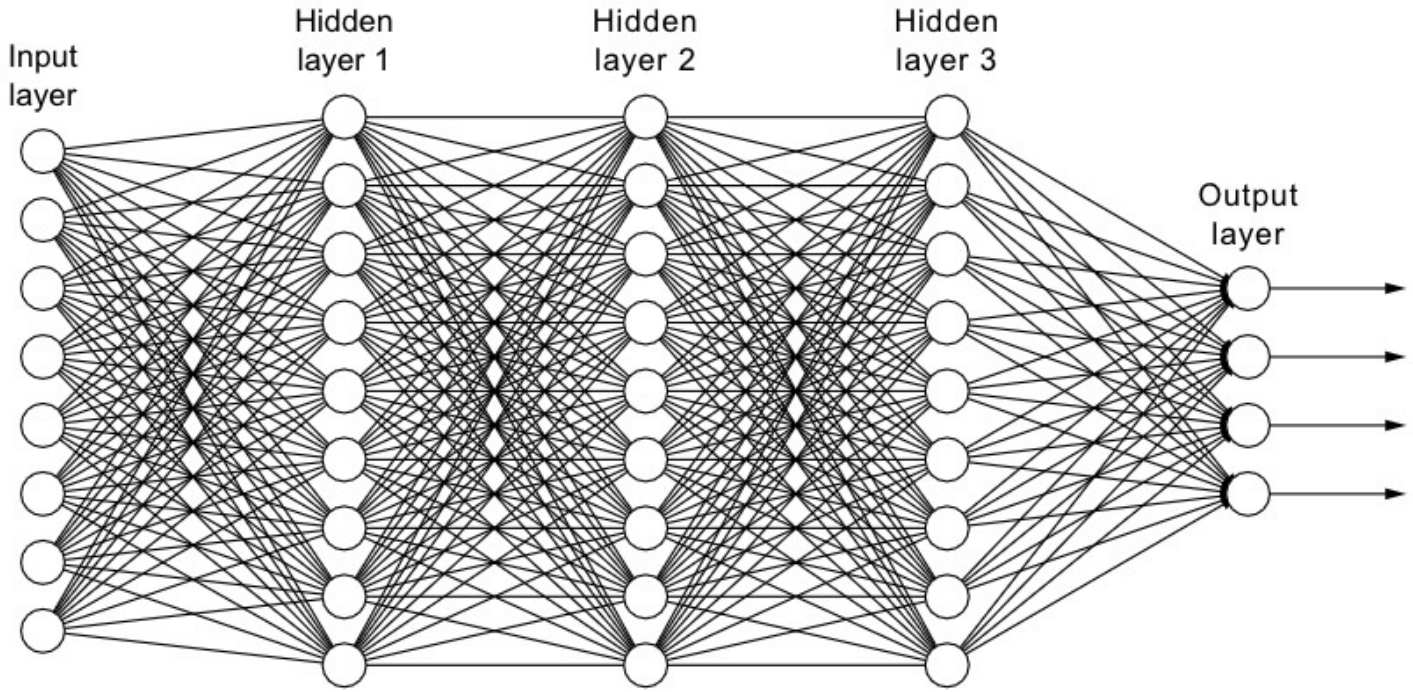


More layers → better performance
(better than increasing the node number

Deep learning architecture



Deep learning architecture



More layers + Many parameters

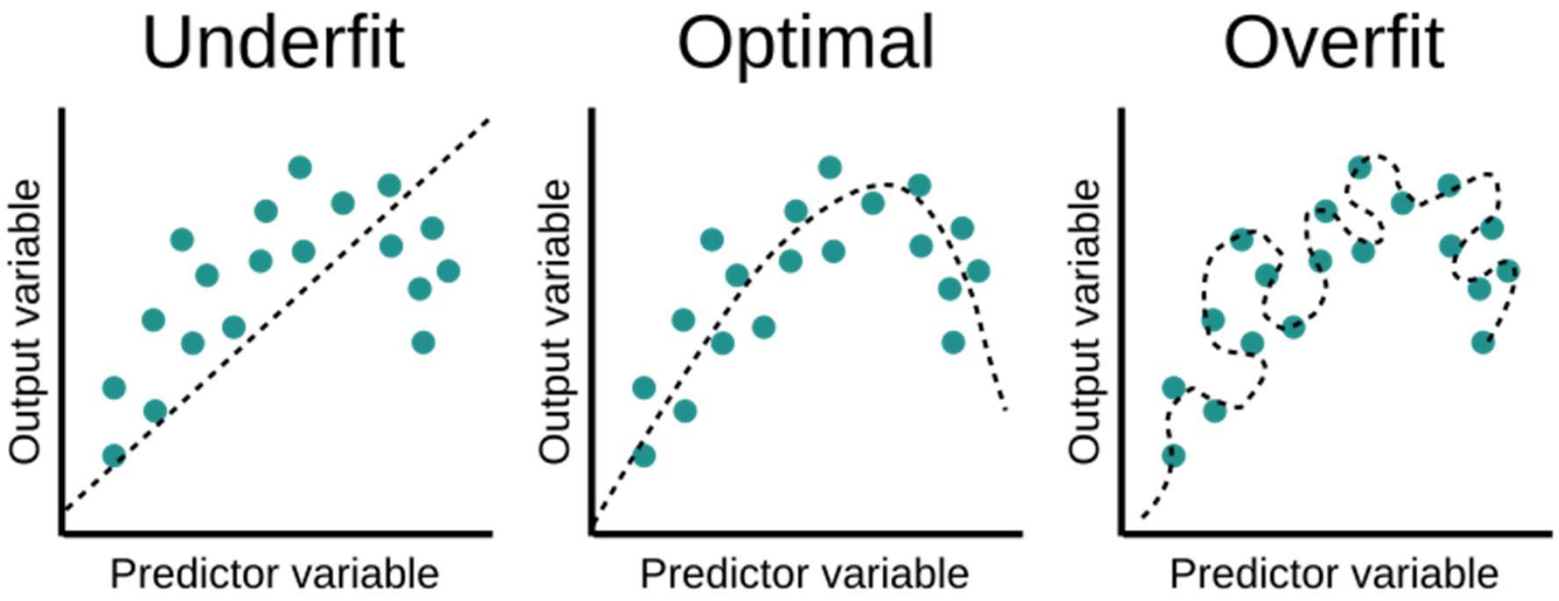
CPU vs GPU

CPU: best processing unit
GPU: moderate processing unit → parallelism → boost parallel matrix operation
*require enough memory to process multiple processing

https://www.youtube.com/watch?v=1BAZf3PsjWA

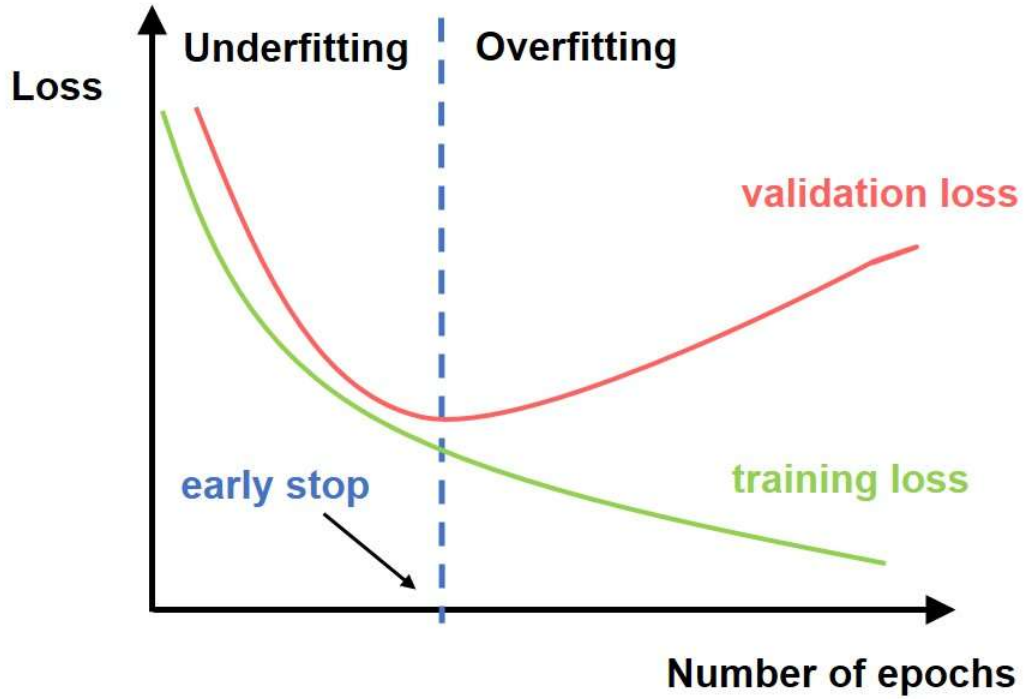
Model capacity and parameters

DL → trying to find the best model (lowest Loss function) → Prone to over-fitting



Model capacity and parameters

DL → trying to find the best model (lowest Loss function) → Prone to over-fitting
Model capacity: number of hidden layers, parameters, epoch …



(model capacity)

-Training loss: Loss during training
-Validation (Test) loss: Loss with validation (test) set: test your model with new data

Regularization

-Penalty for loss function
-ex: Loss = MSE + regularization

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

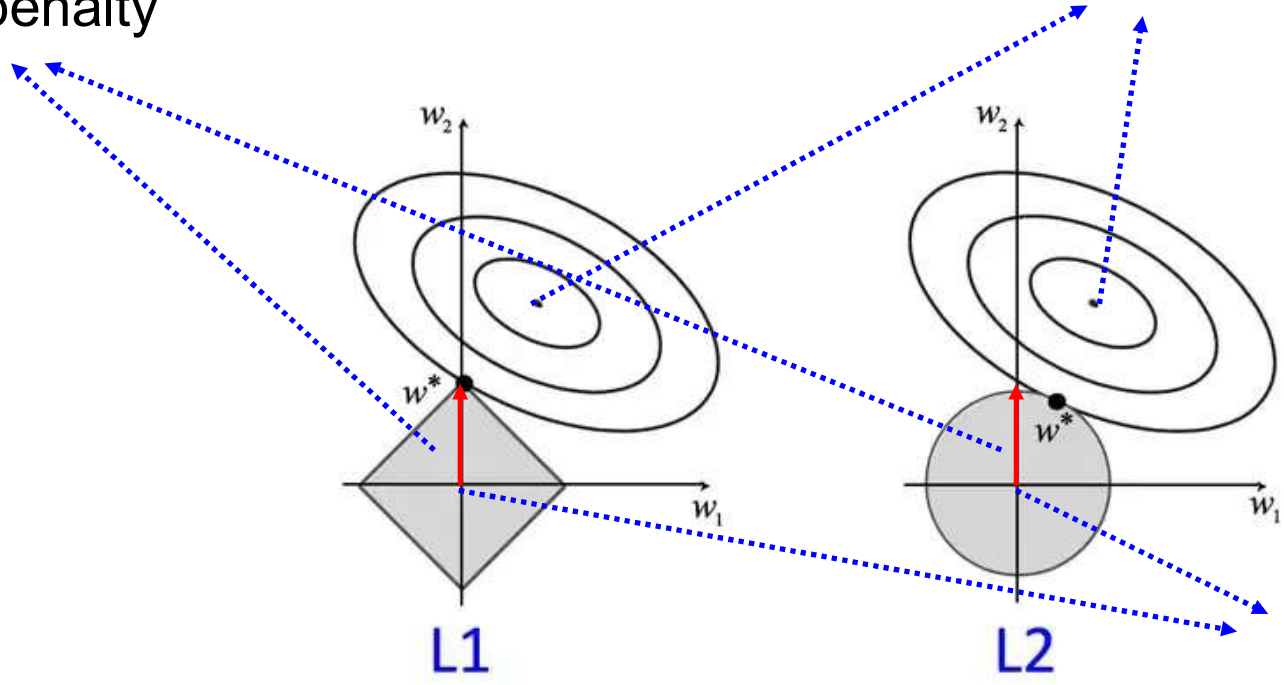
$$\text{Cost} = \underbrace{\sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2}_{\text{Loss function}} + \underbrace{\lambda \sum_{j=0}^{M} W_j^2}_{\text{Regularization Term}}$$

-Penalty from each weight (parameters)
-L1 (absolute value) or L2 (square) → always positive → constraint for loss function
-λ: penalty

Regularization

Minimum loss for the test set
→ Can lead to over-fitting

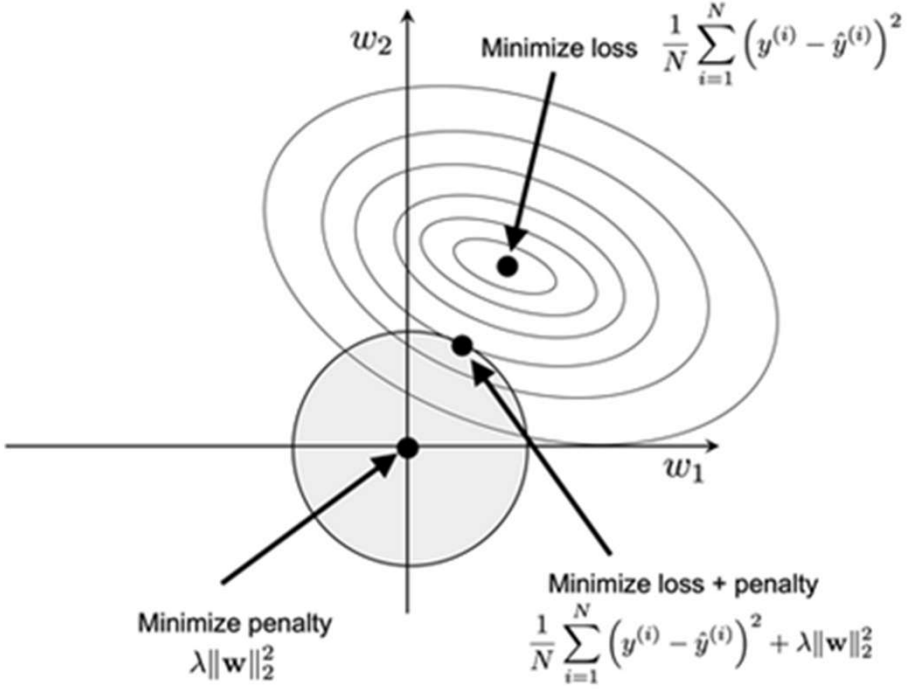
-λ: penalty



Minimum regularization

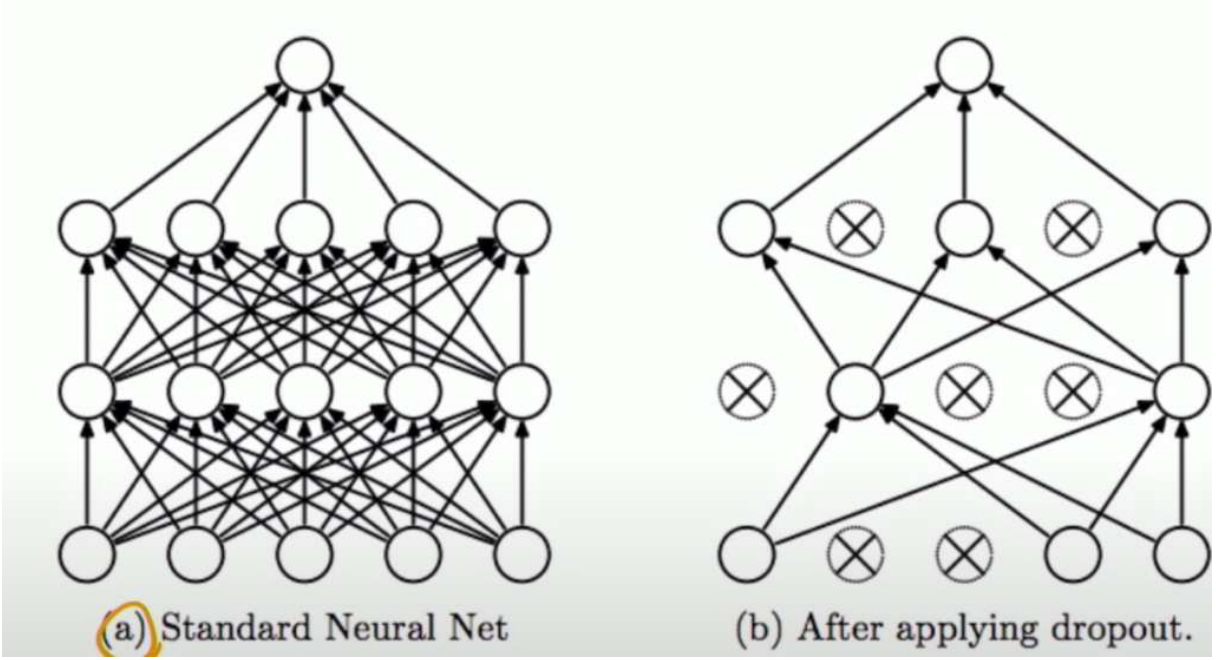
-Regularization prevents the “weight” from increasing too much

Regularization



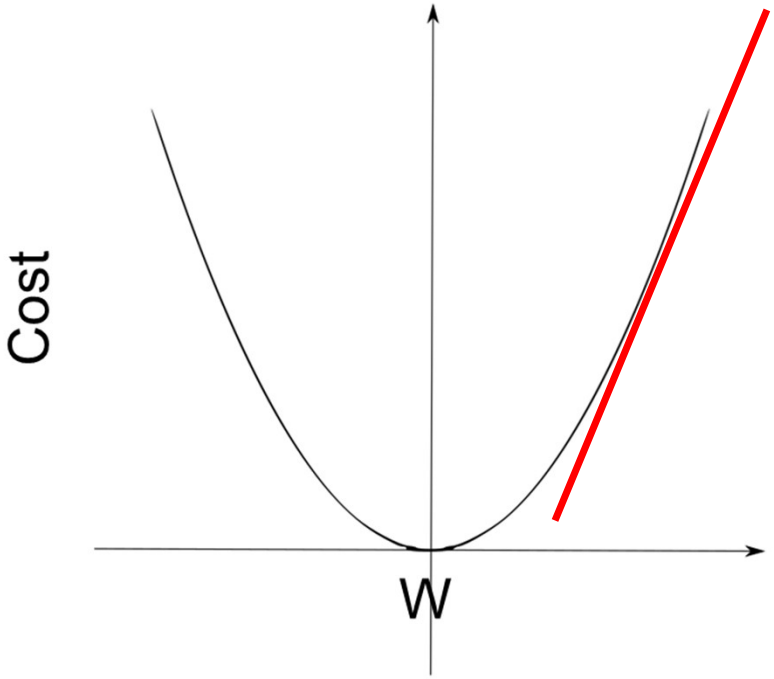
-Regularization prevents the “weight” from increasing too much
→ Now this model can also predict the data from “left-bottom”

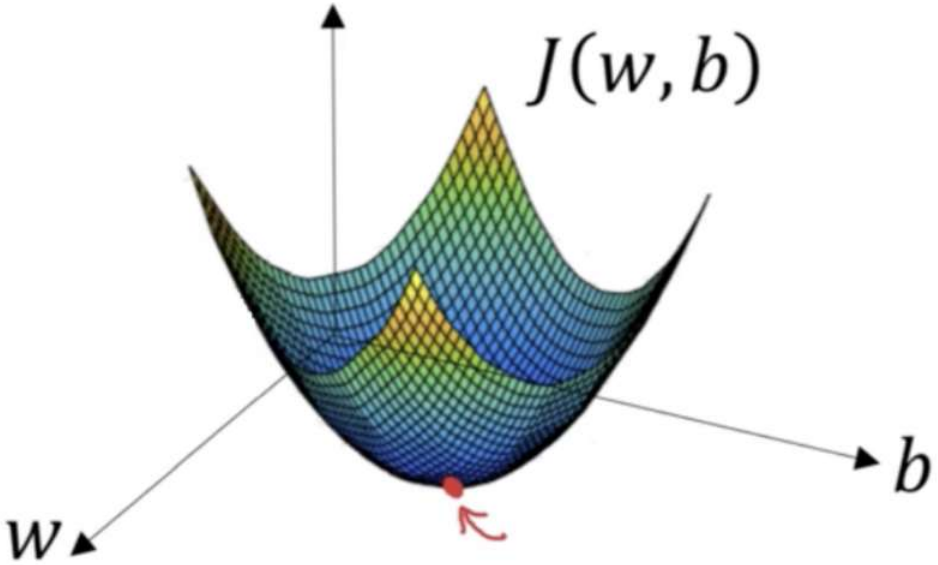
Dropout



-Weight → 0 (for probability *p*) → lessen model capacity

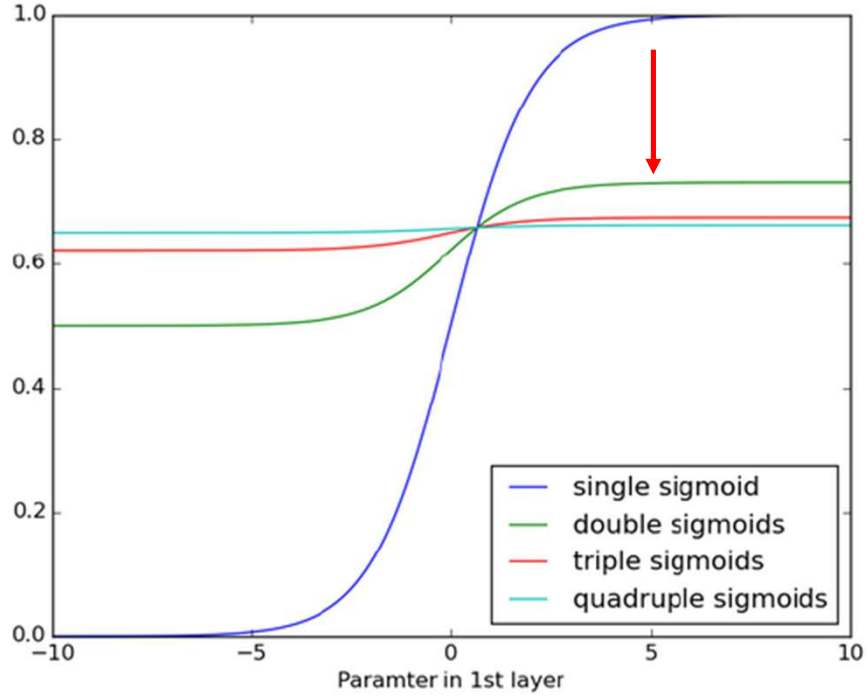
Gradient vanishing





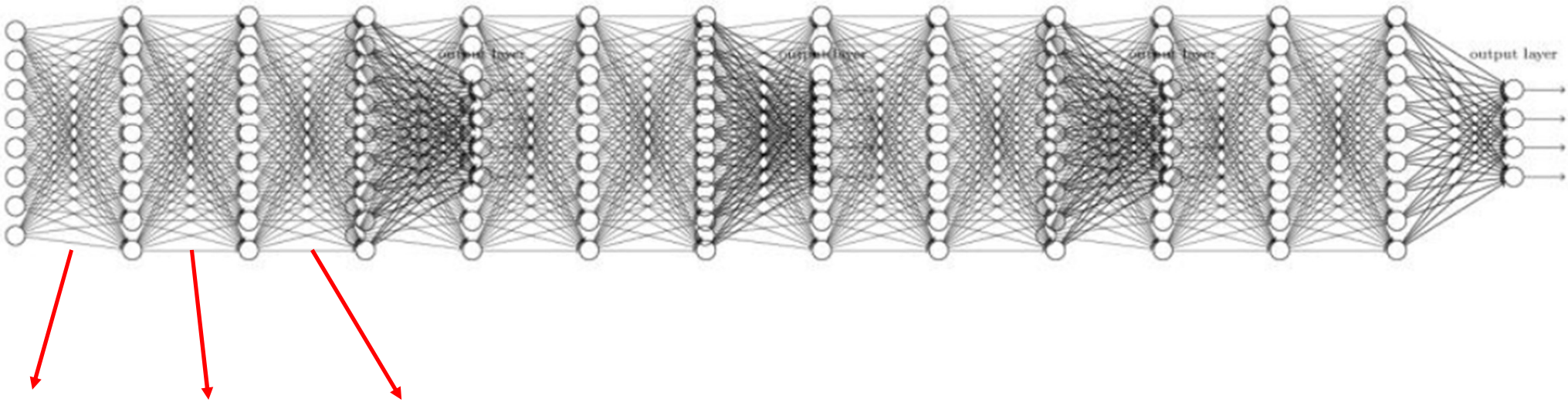
-Gradient descent method for finding the minimum loss

Gradient vanishing



-Sigmoid * Sigmoid * Sigmoid … → gradient of “multiple” sigmoid → ~0

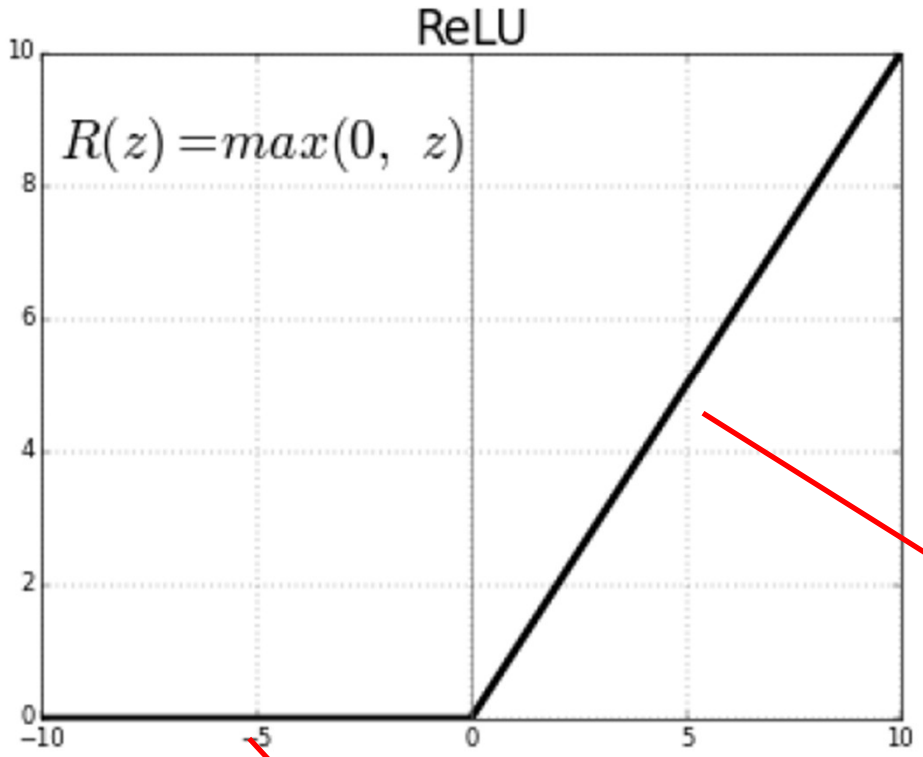
Gradient vanishing



-Sigmoid * Sigmoid * Sigmoid … → gradient of “multiple” sigmoid → ~0
-For early layers, it cannot learn from backpropagation

Activation function

Rectified Linear Unit (ReLU) Activation

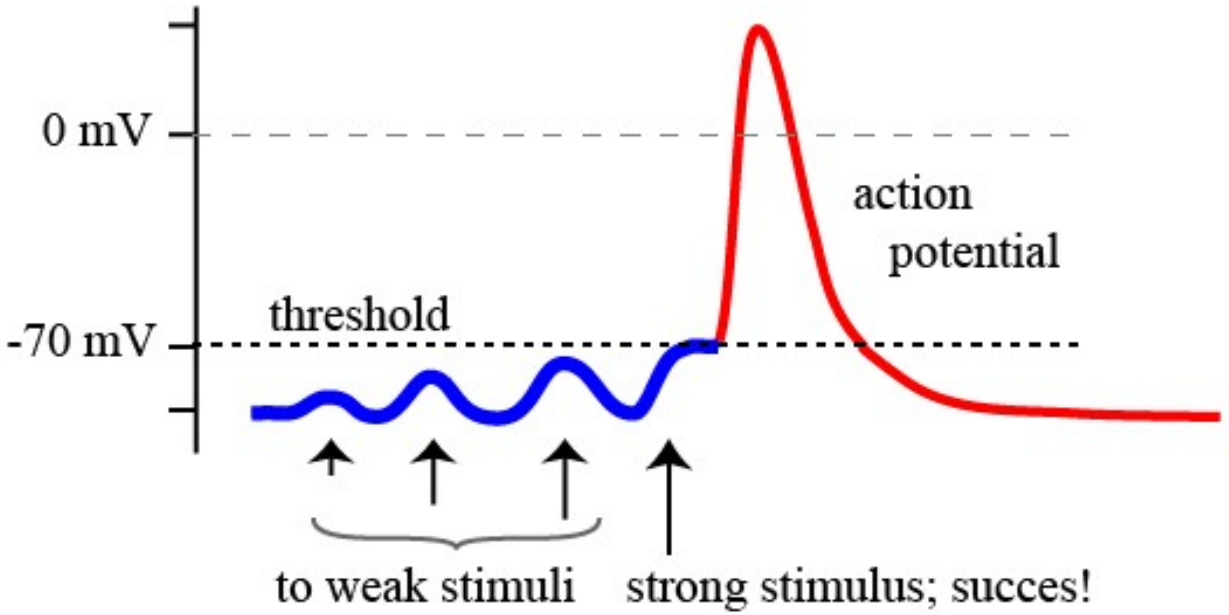


-Keep the gradient

-Kill the gradient → no learning

Activation function

-Only pass the signal if the stimulus is higher than the threshold

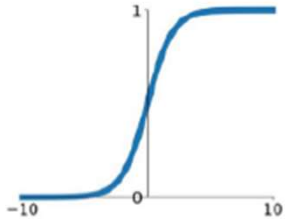


Activation function

Activation Functions

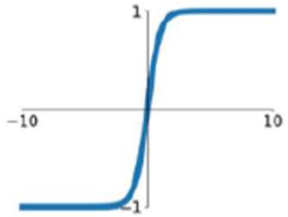
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



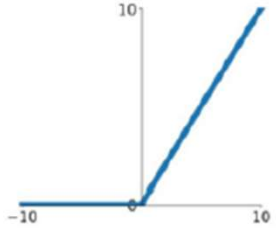
tanh

$$\tanh(x)$$



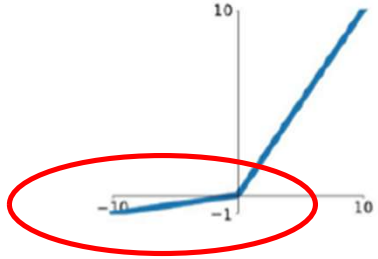
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

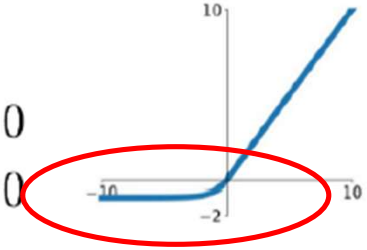


Maxout

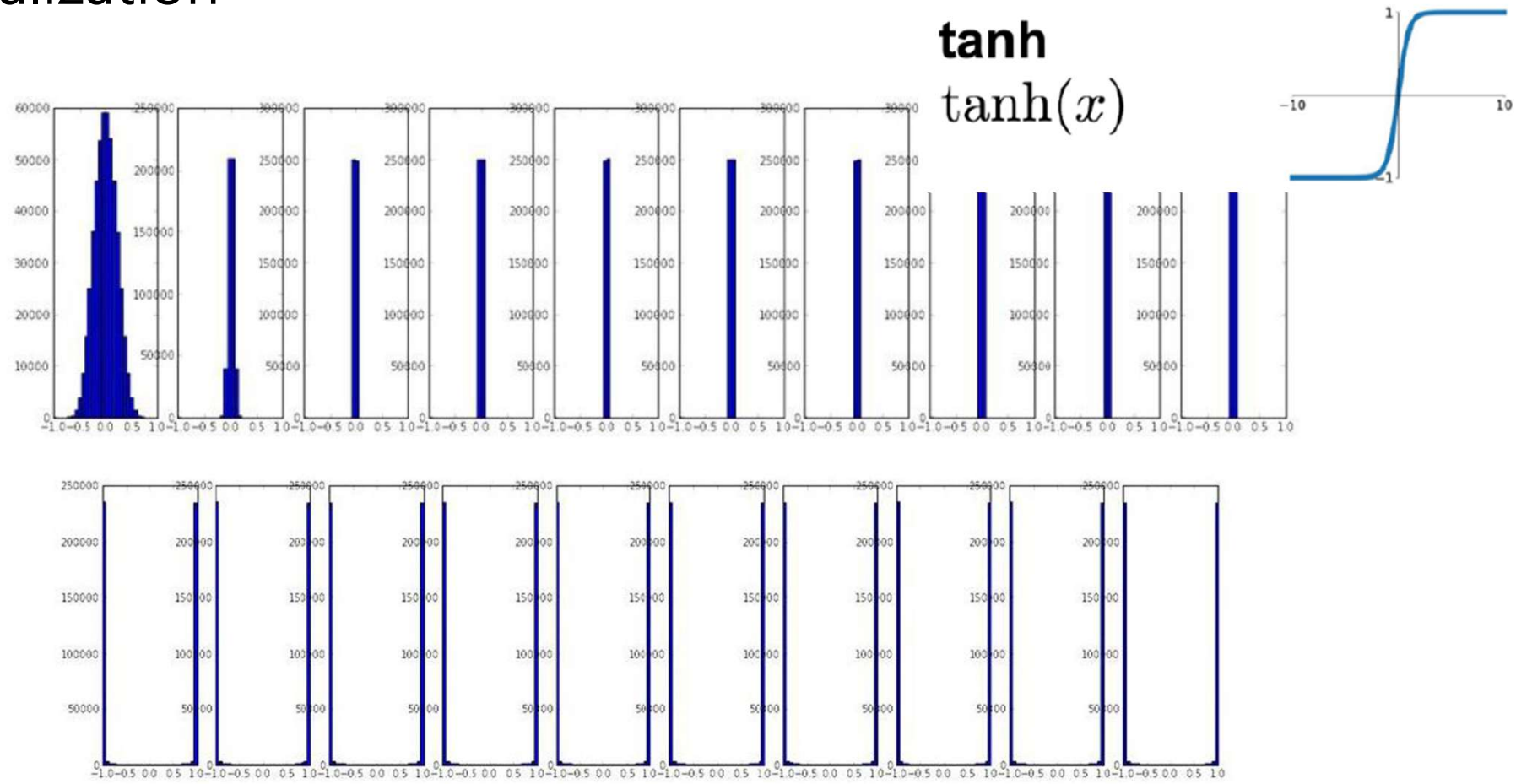
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Xavier initialization



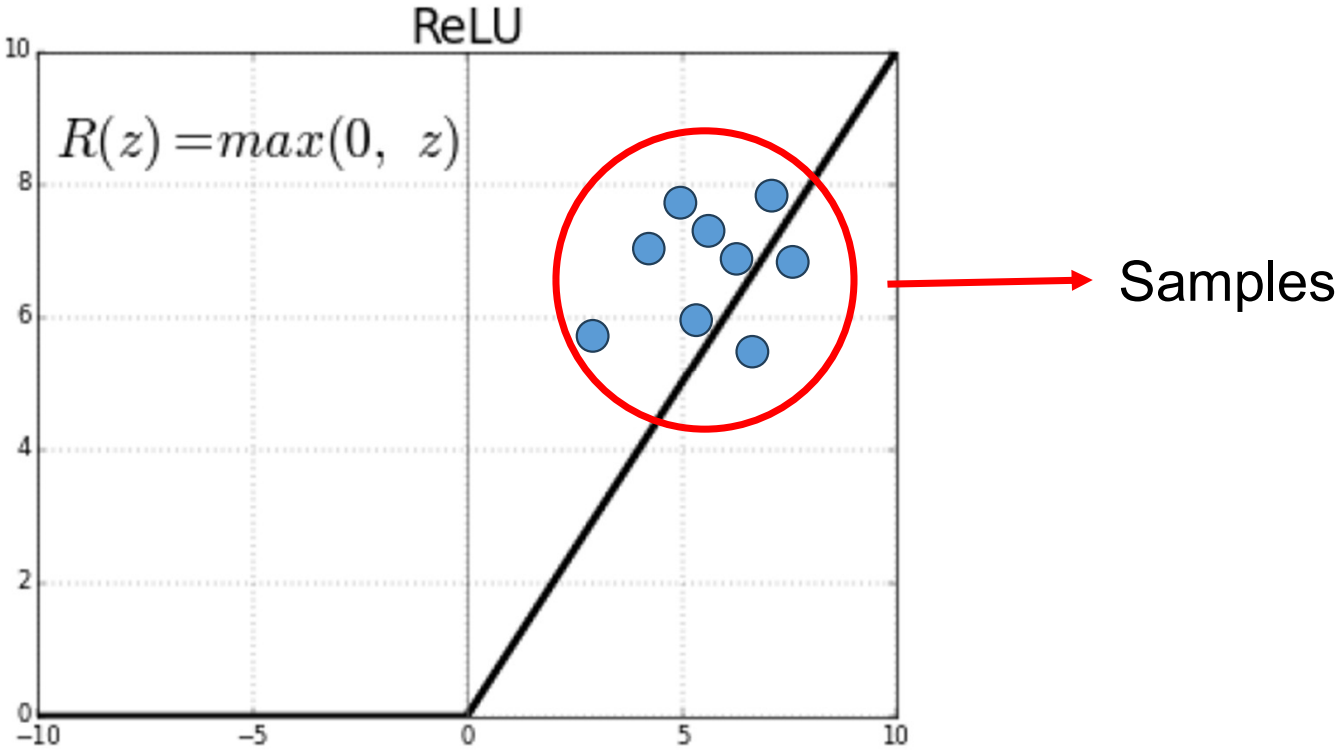
-Distribution of weight: Gaussian → Tanh activation function → converge into -1 or 1

Xavier initialization

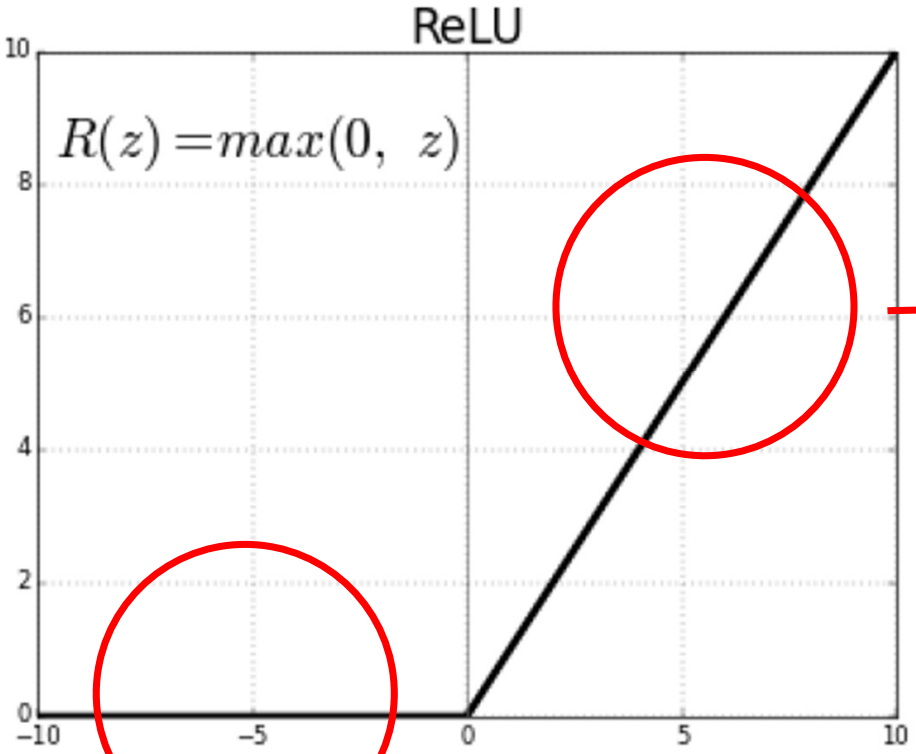
-Initialize weight distribution into gaussian distribution
→ Prevent convergence during training across multiple layers

-n: number of nodes in the next hidden layer
-m: number of nodes in the current layer
-S.D.: 2 / $\sqrt{n+m}$: Gaussian distribution → weight initialization

Batch normalization



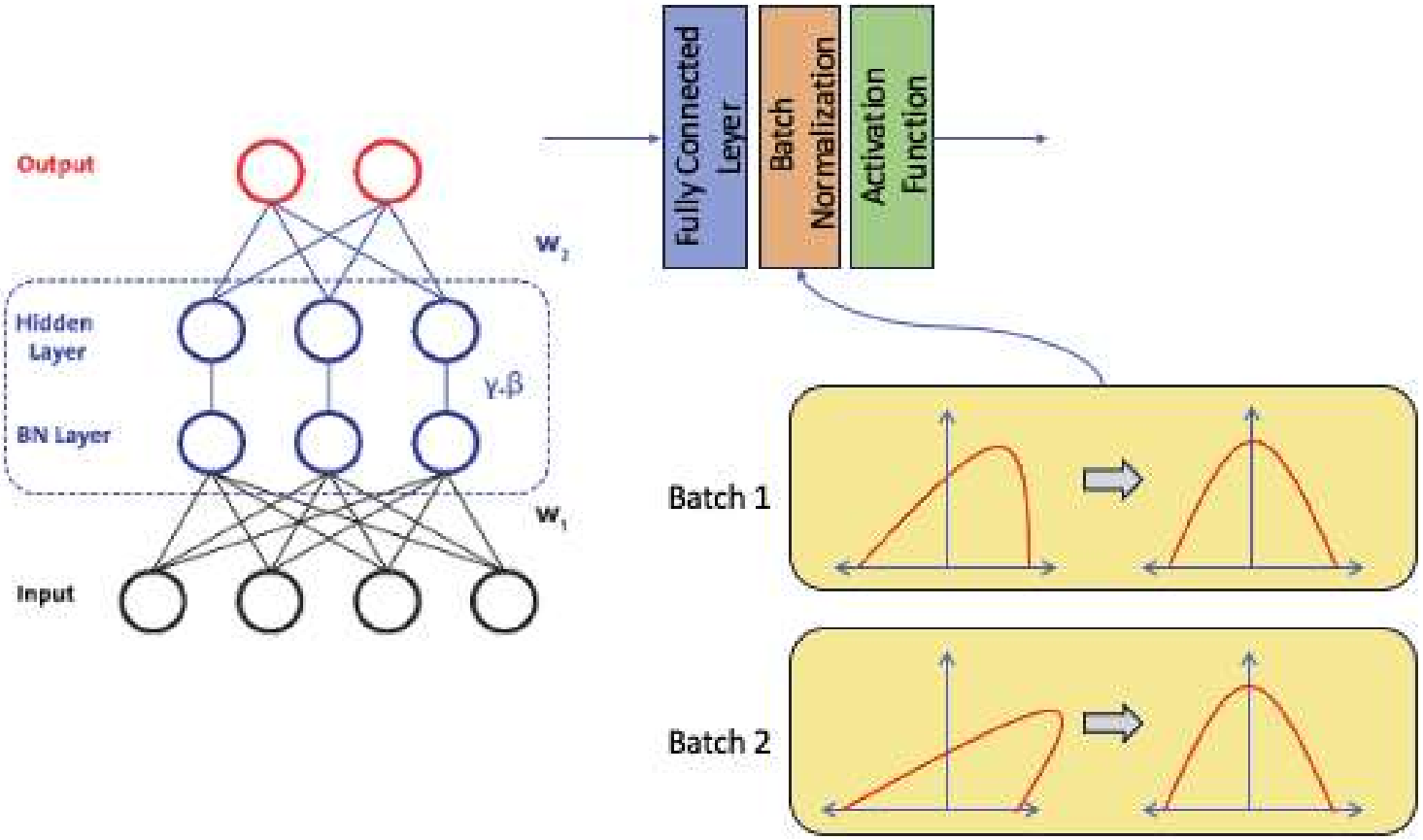
Batch normalization



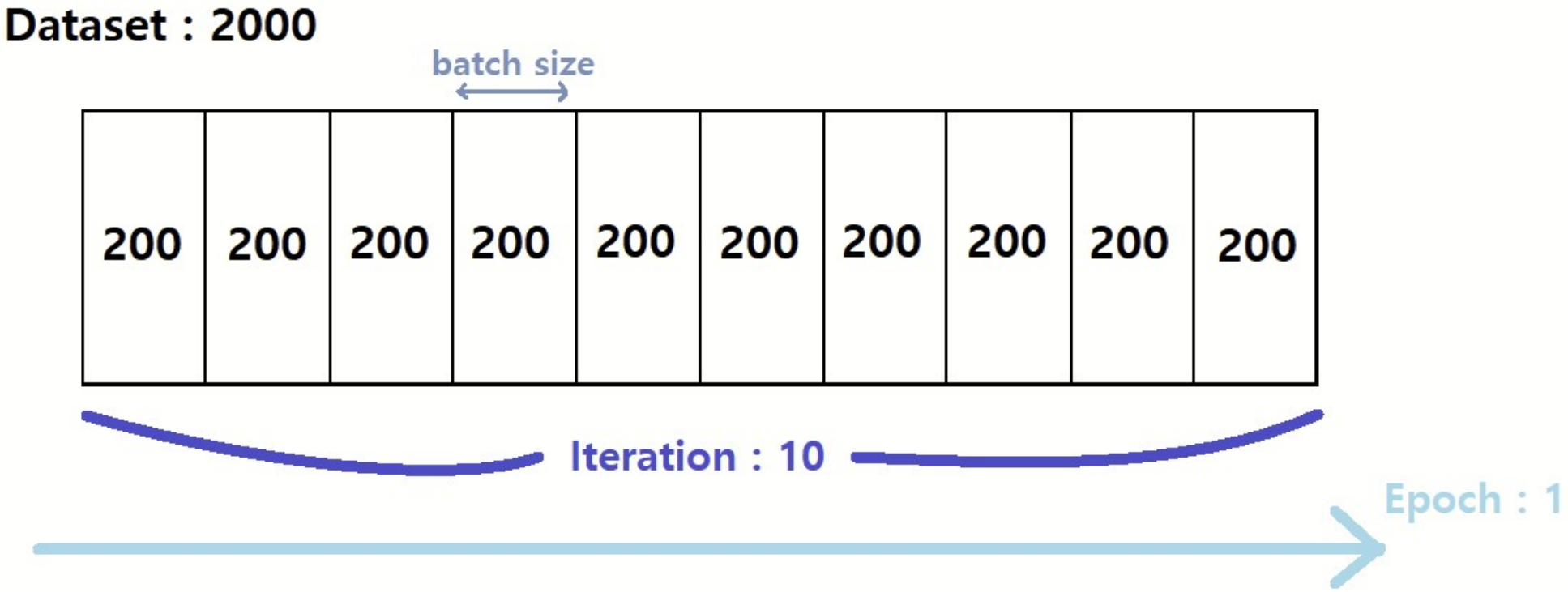
- Weight 1 → next layer → 1 → 1 → 1

- Weight 0 → next layer: cannot learn

Batch normalization



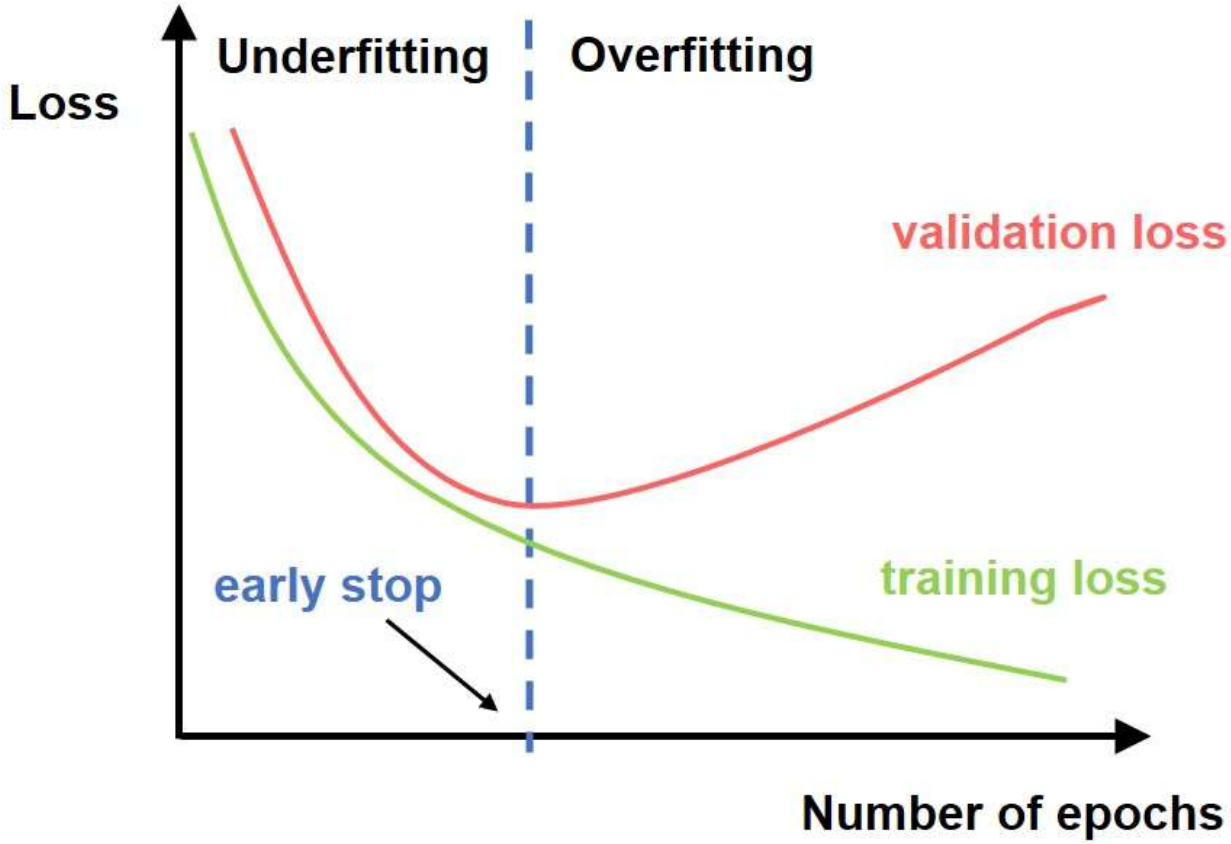
Hyper parameter tuning



-Batch: one set of samples (from input data) → for single iteration (learning)
-1 epoch: whole dataset

-Too big: memory limitation
-Too small: poor training, sample distribution bias

Hyper parameter tuning



-Epoch: early stopping for preventing over-fitting

Hyper parameter tuning

Purpose of Hyperparameter Tuning?

↓

Increase Model Performance

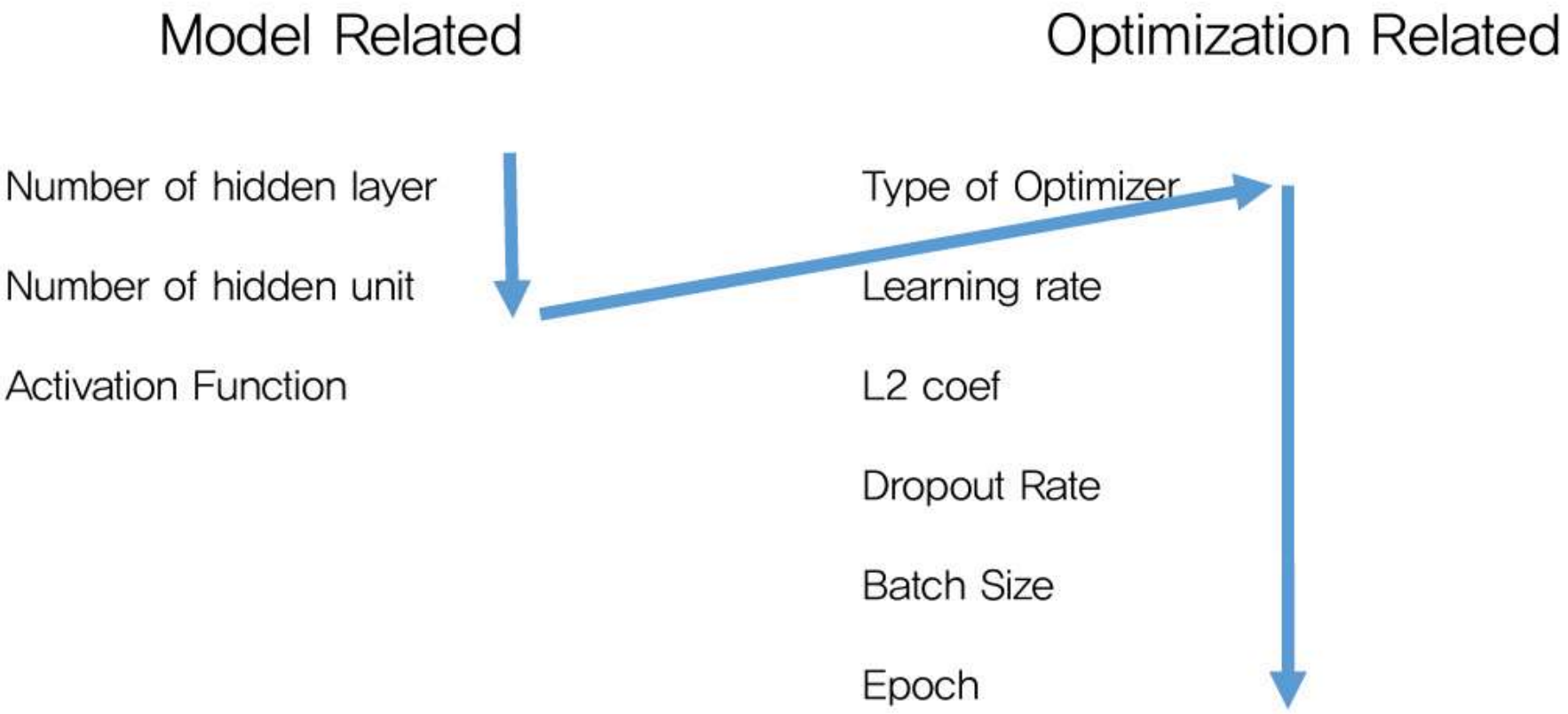
↓

Reduce True Risk(Generalization Error) of Model

↓

Reduce True Risk on Validation Set, approximately

Hyper parameter tuning



Hyper parameter tuning

-Hidden layer

Very Flexible
(1~10 layers for MLP)

(~152, ~1000 layers for Recent CNN)

-Hidden unit

Very Flexible
(10~1024 unit/layer)

-Activation function

Sigmoid (x)
tanh (x)
ReLu (o)
LeakyReLu(o)
GeLu(o)
Elu…

-Type of optimizer

GD(X)
SGD(soso)
RMSProp(O)
ADAM(O)
AdaDelta(O)

-Learning rate

1e-5 ~ 1e-1
Log scale

0.00001
0.0001
0.001
0.01
0.1

0.001
0.003
0.006

Hyper parameter tuning

-L2 coef

1e-5 ~ 1e5
Log scale

0.00001
0.0001
0.001
0.01
0.1
1.0
10
100
1000
10000
100000

-Dropout rate

0.1~0.5(~0.7)

-Epoch

Regularly Measure Train Loss & Val Loss

Early Stopping
e.g. if validation loss is not reduced
more than N epoch, then stop training