Introduction to Machine Learning

Lecture 14: Neural Networks

Nov 21, 2024

Jie Wang
Machine Intelligence Research and Applications Lab
Department of Electronic Engineering and Information Science (EEIS)
http://staff.ustc.edu.cn/~jwangx/
jiewangx@ustc.edu.cn

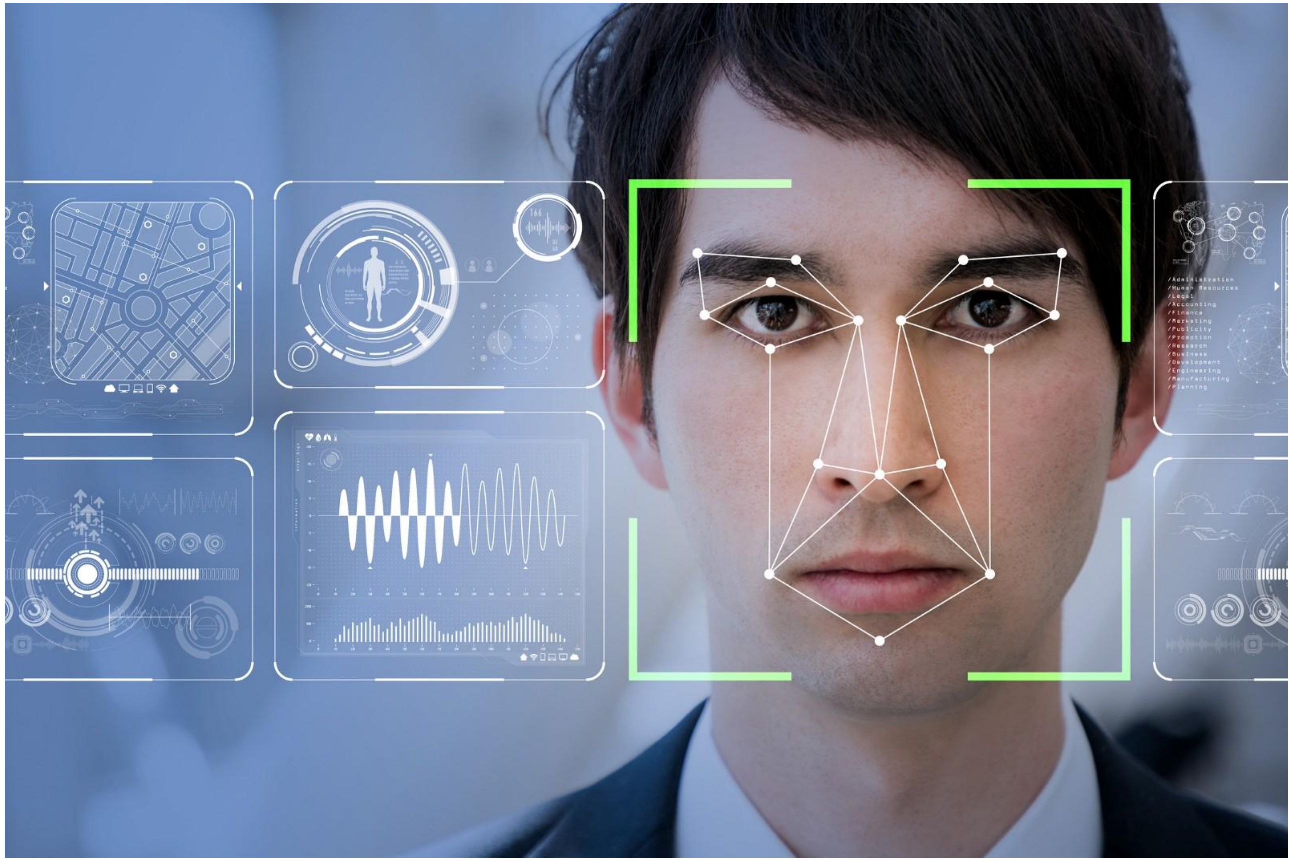
Machine Intelligence Research and Applications Lab

Contents

Introduction

Breakthroughs by Deep Learning

Face recognition



SENSETIME
商 汤 科 技

Face++ 旷视

云从科技
CLOUDWALK

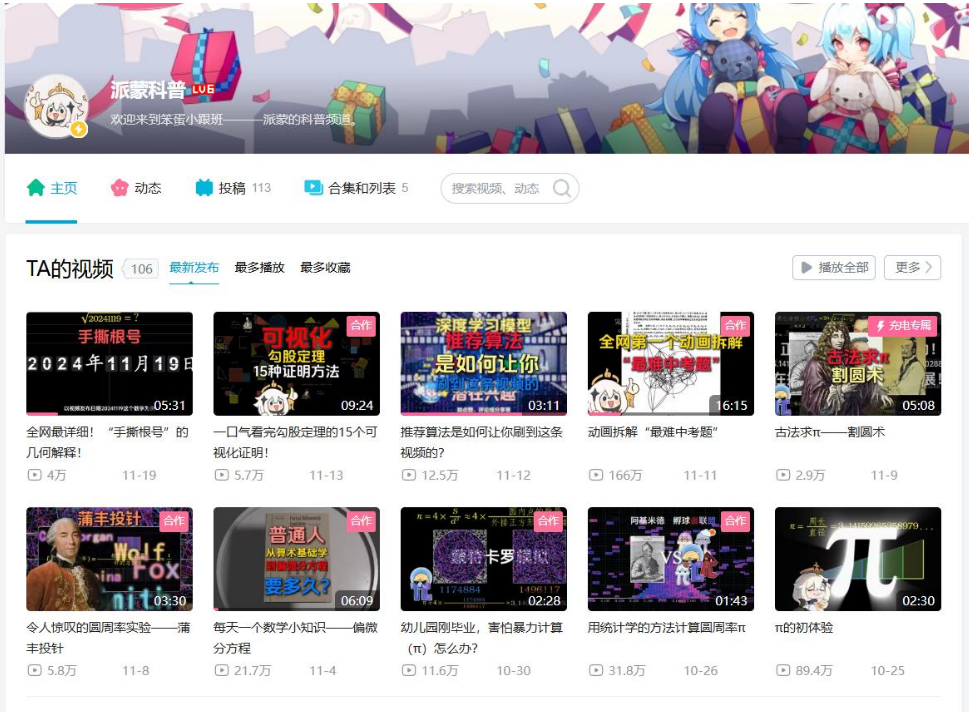
阿里云
aliyun.com

Breakthroughs by Deep Learning

Machine translation

Breakthroughs by Deep Learning

Speech synthesis



Breakthroughs by Deep Learning

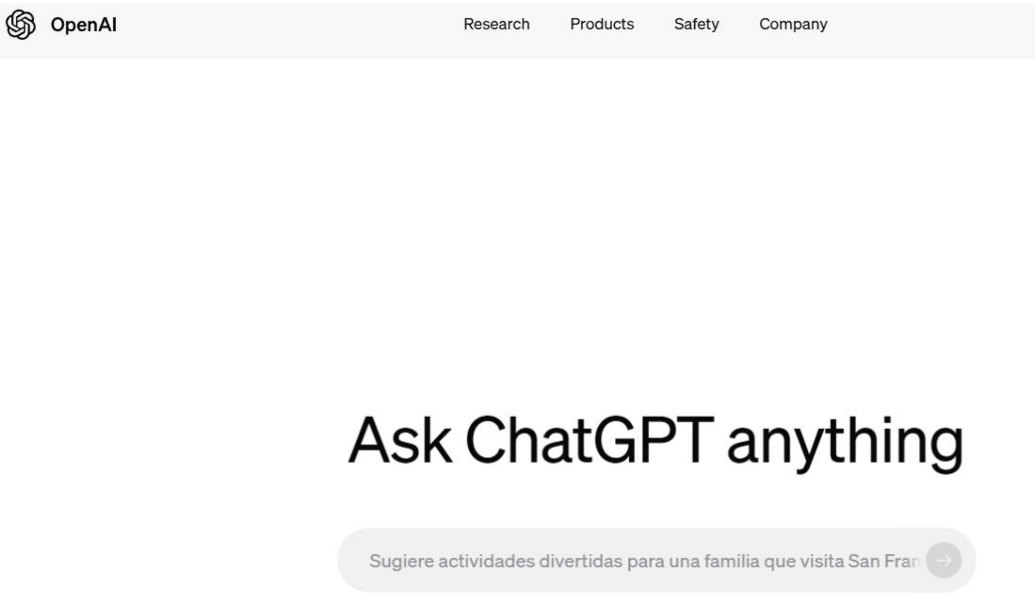
Self-driving

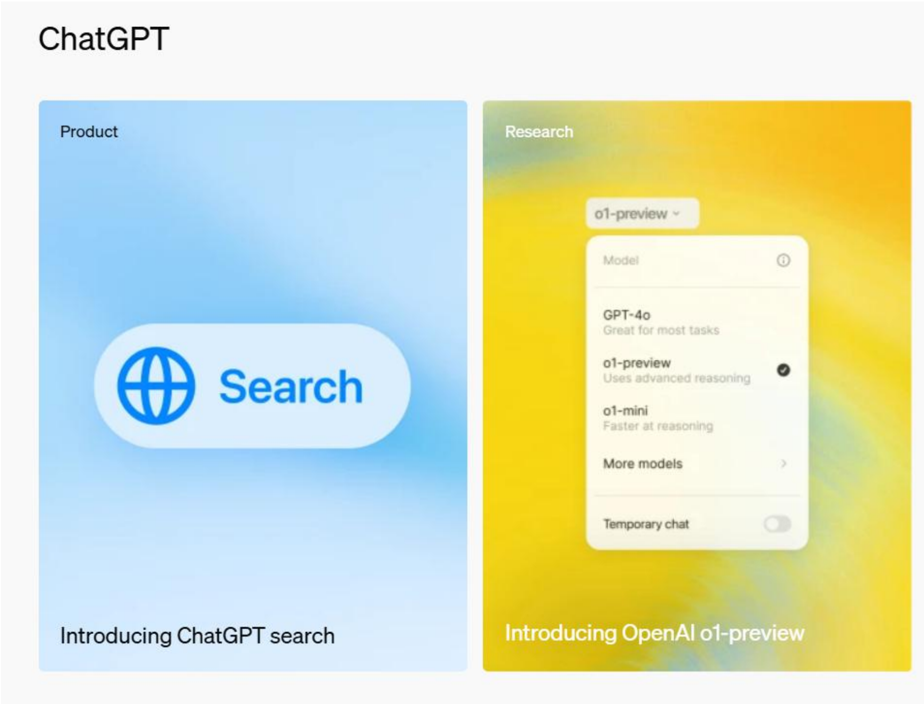
Breakthroughs by Deep Learning

AI Generated Paintings

Breakthroughs by Deep Learning

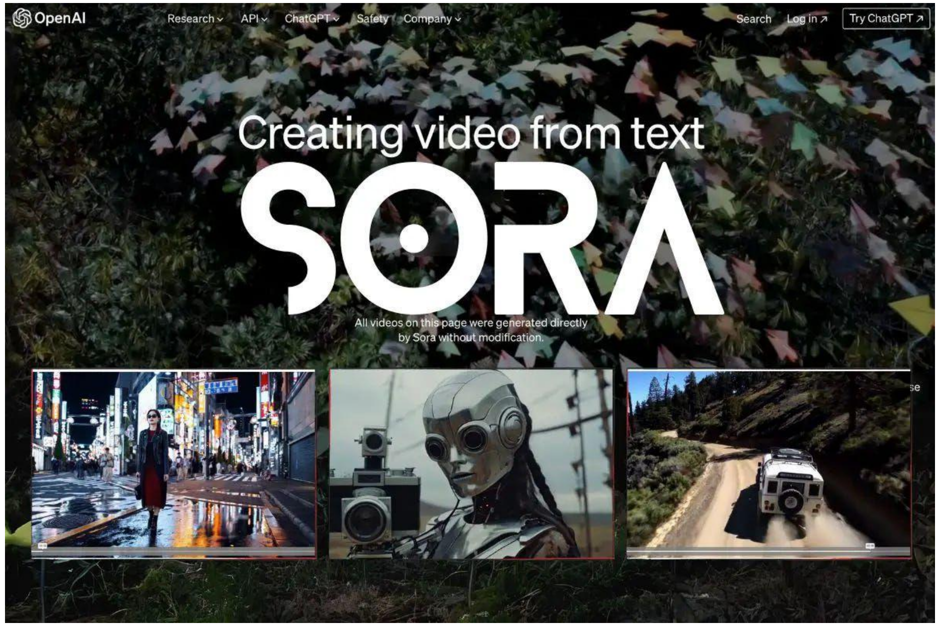
Large Language Model



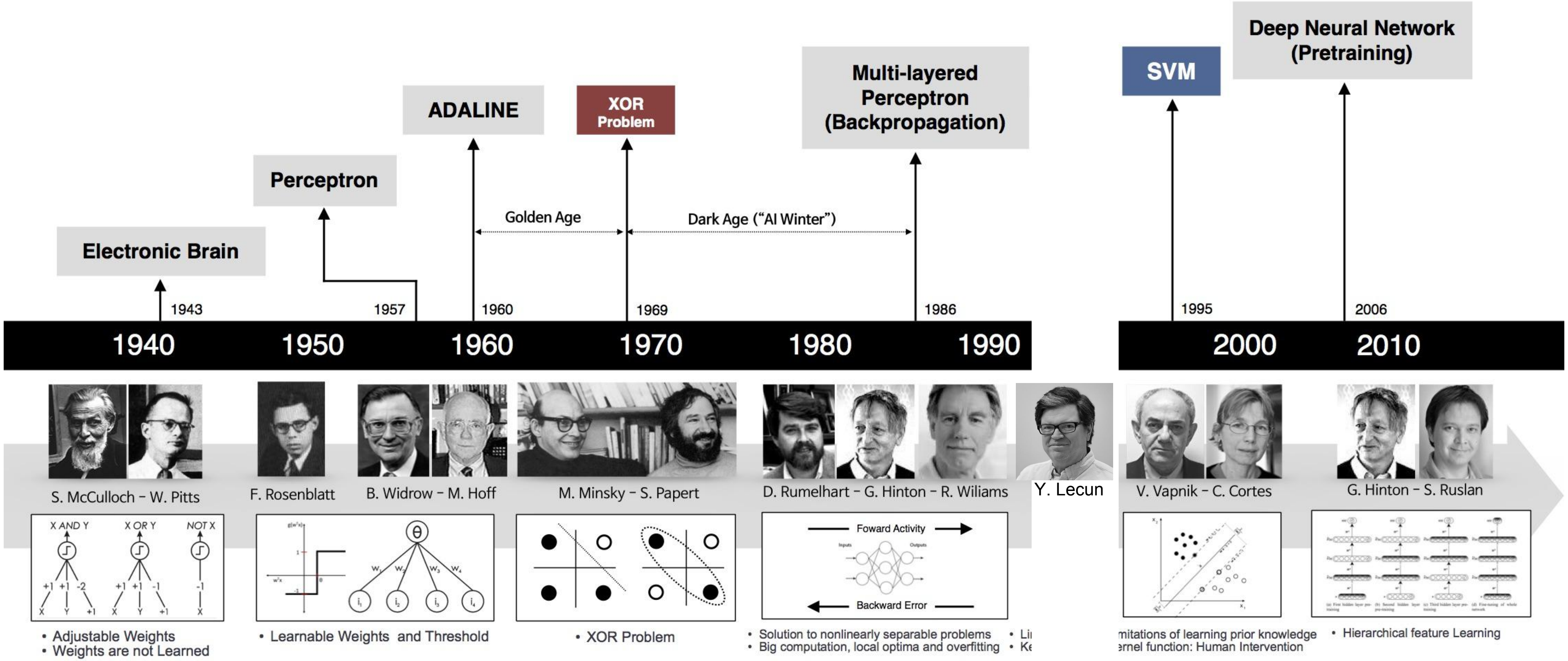


Breakthroughs by Deep Learning

AI Generated Videos

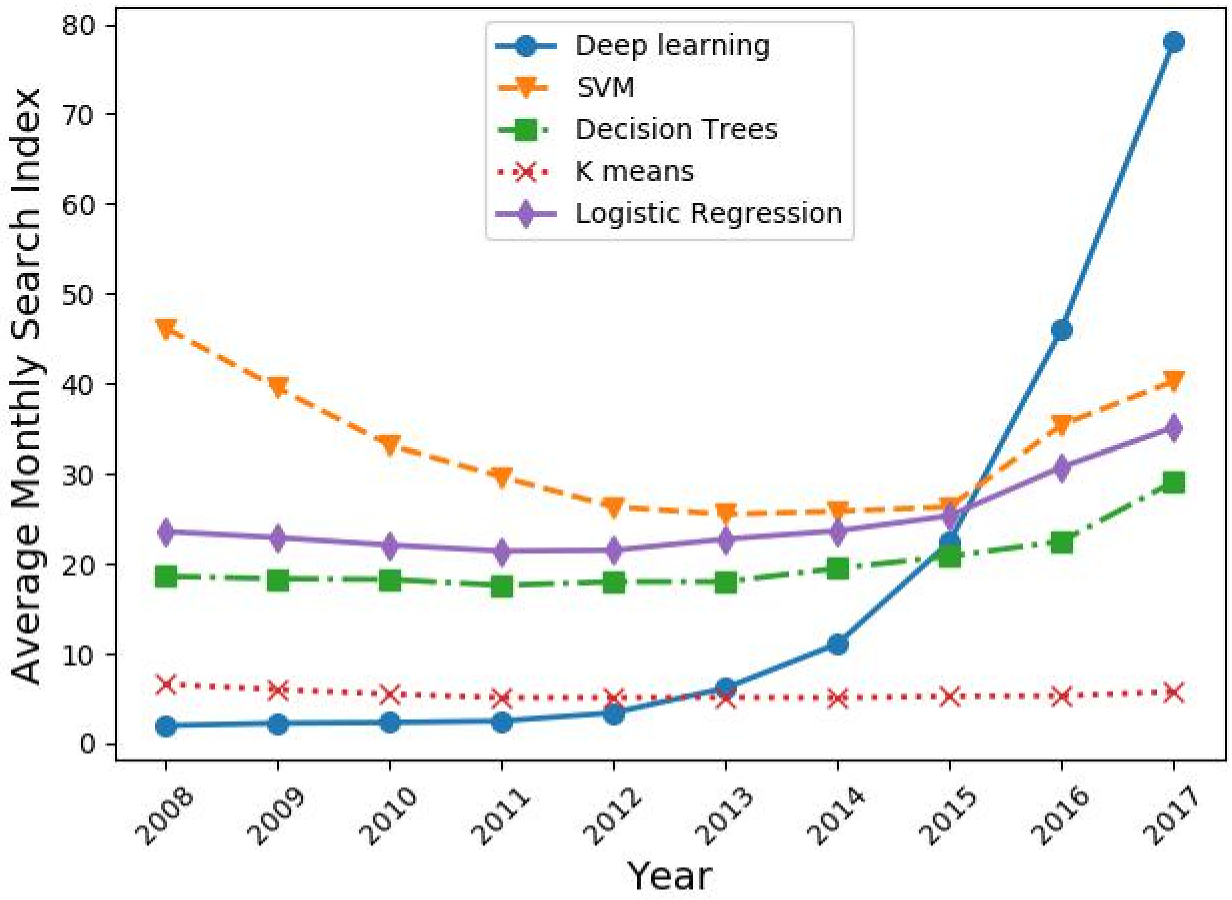


Milestones of Deep Learning



https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

Google Trend of Deep Learning



Mehdi Mohammadi, at.al. Deep Learning for IoT Big Data and Streaming Analytics: A Survey.
IEEE Communications Surveys and Tutorials Journal, 2018.

Motivation of Neural Networks

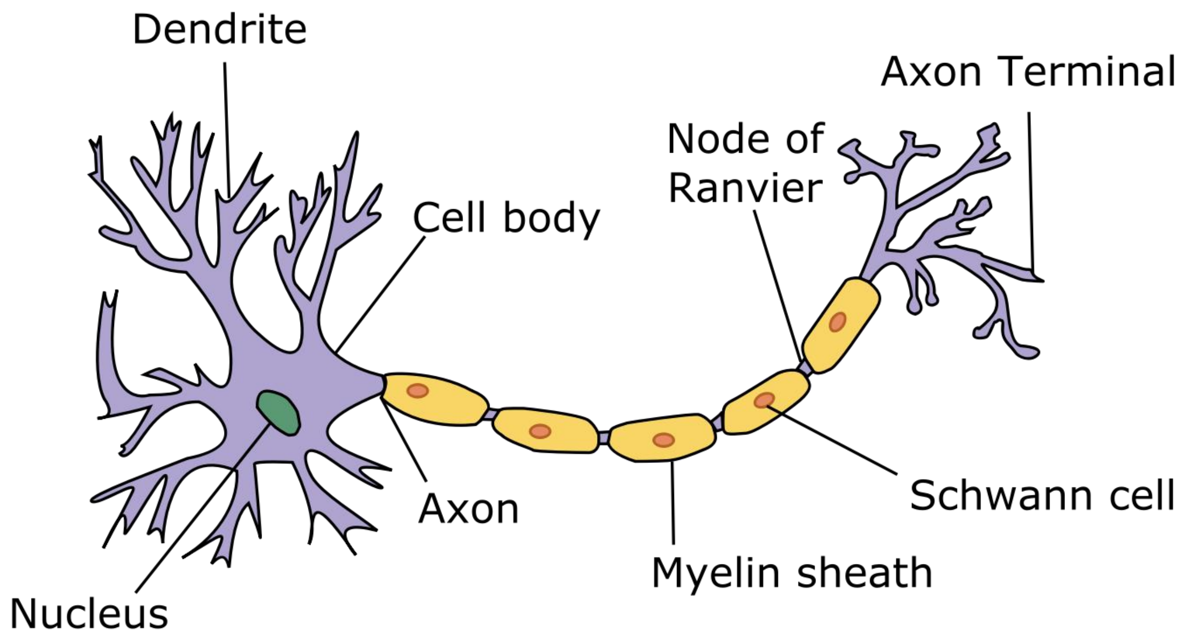
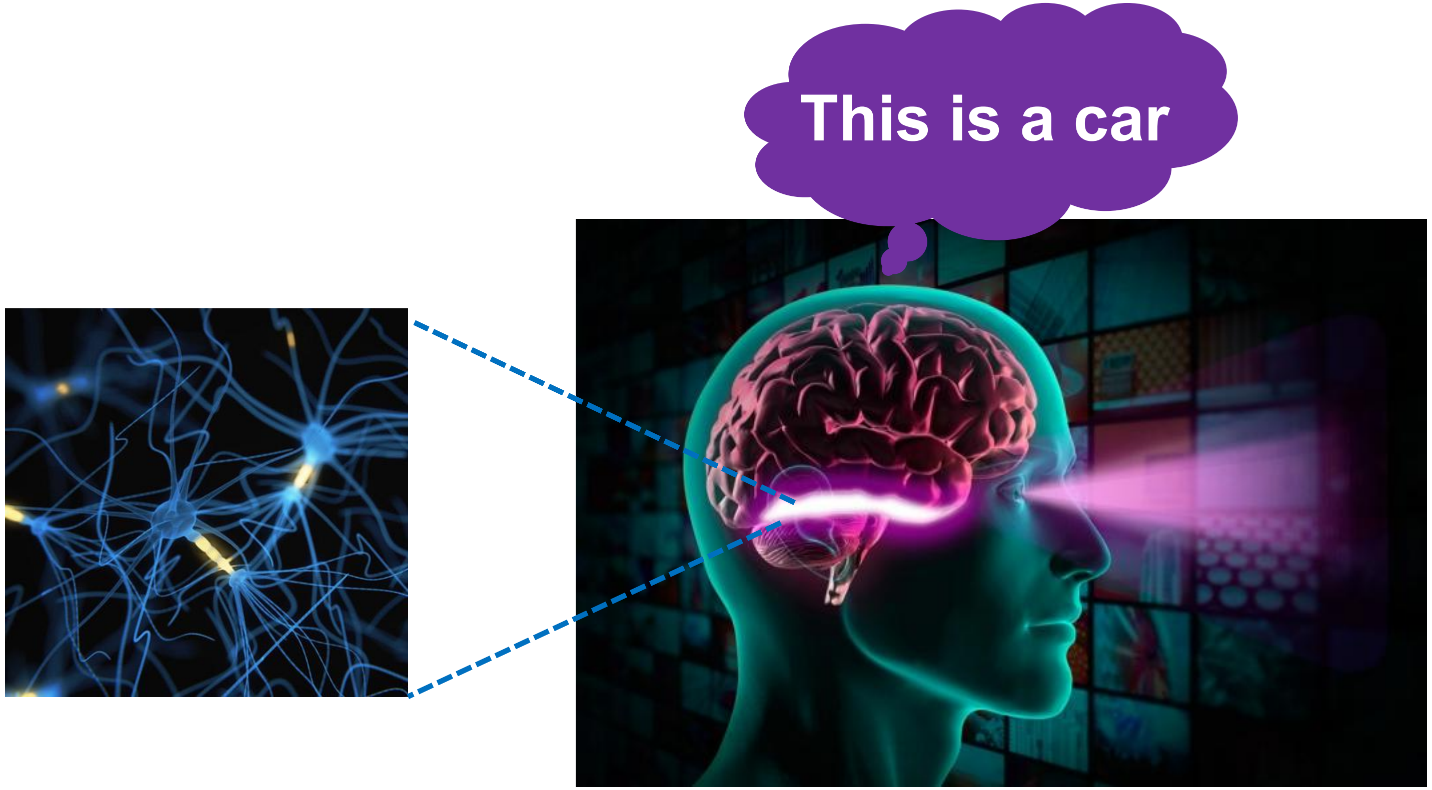


Diagram of neuron

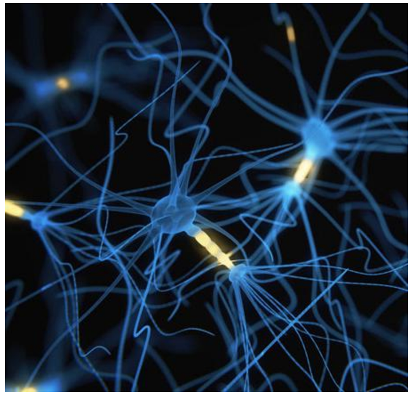
https://simple.wikipedia.org/wiki/Neuron

Motivation of Neural Networks

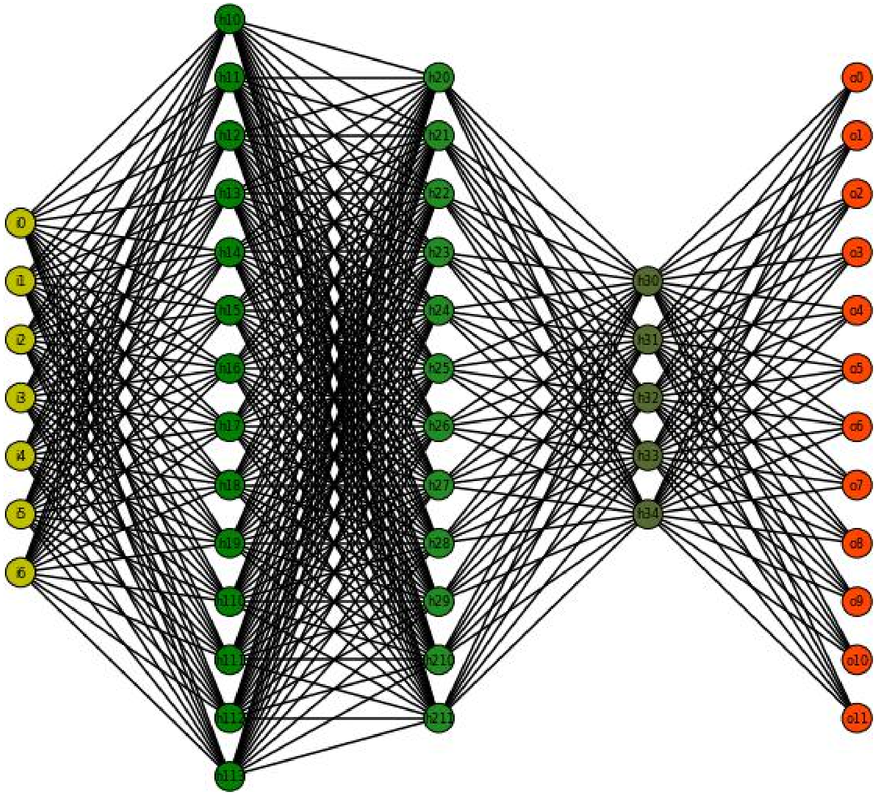


http://news.mit.edu/2015/how-brain-recognizes-objects-1005

What is Neural Network?



Biological Neural Network

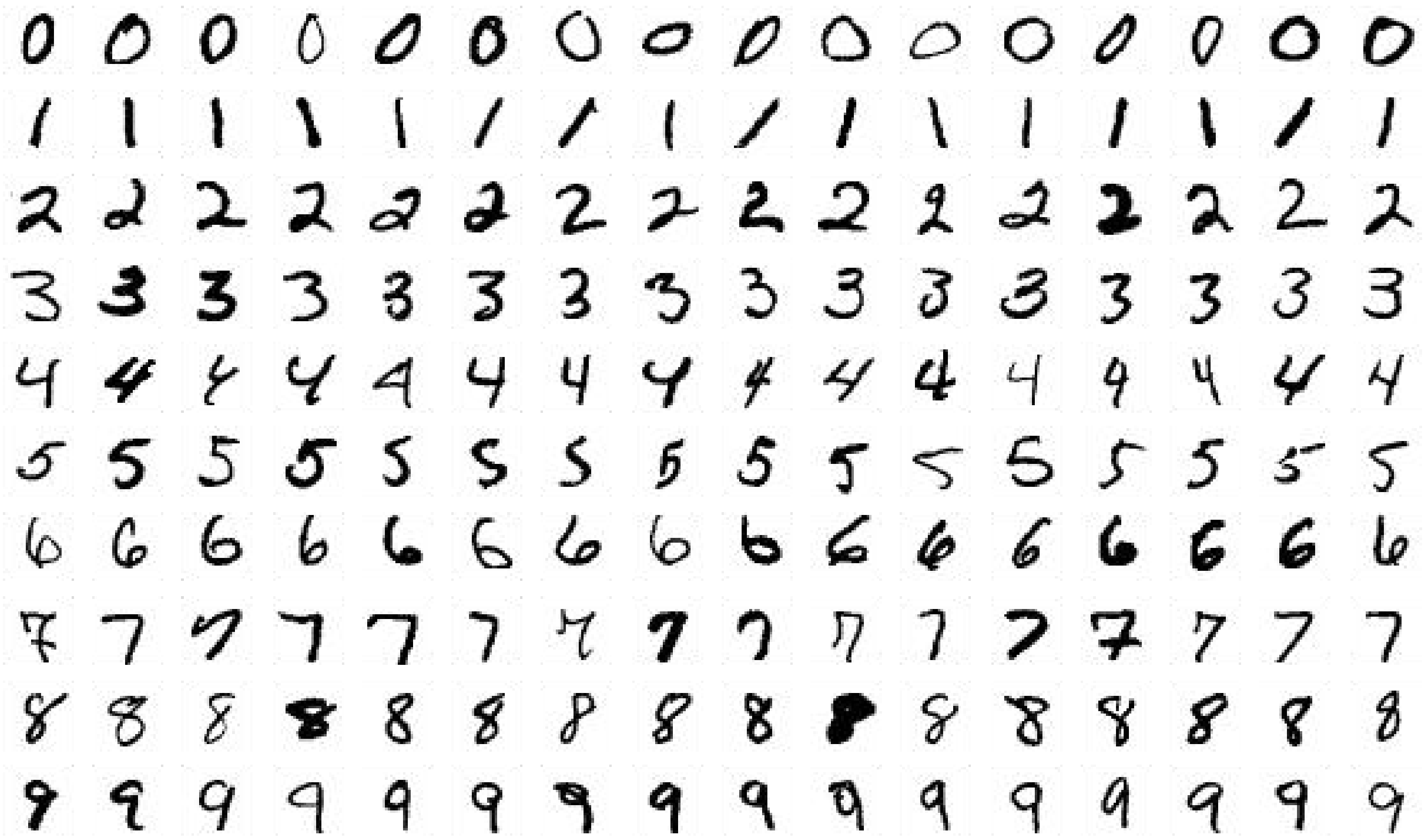


Artifical Neural Network

Multi-Layer Perceptron

Hand-written Digits Recognition

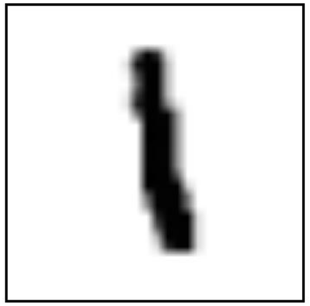
The MNIST dataset



By Josef Steppan - Own work, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=64810040

Vector representation

x: image



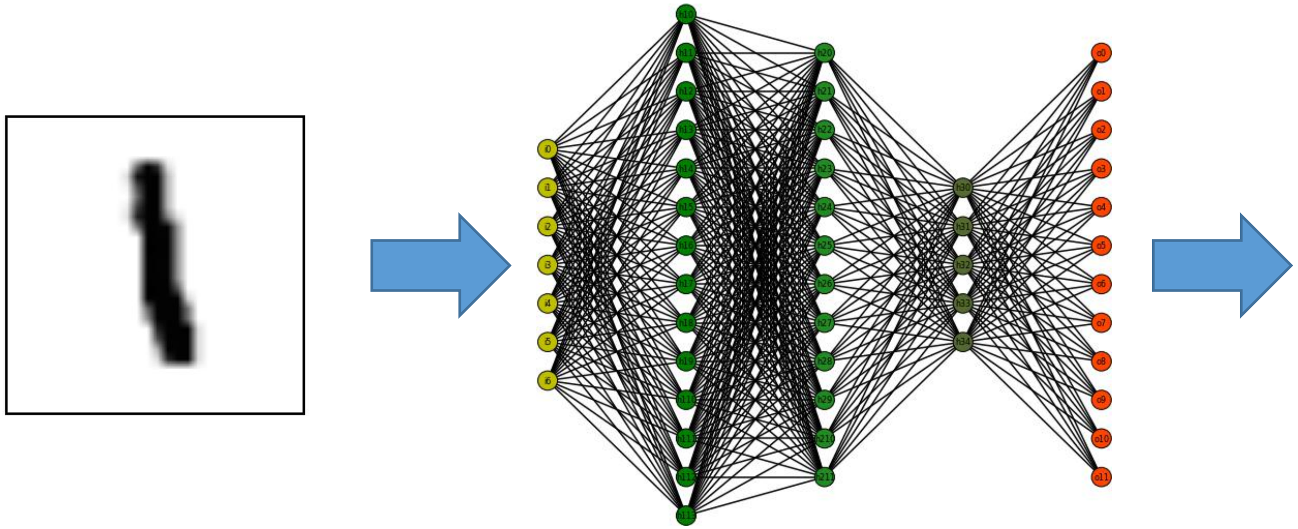
28×28 pixels

$$\begin{bmatrix} 0 \\ 1 \\ \dots \\ \dots \\ 0 \end{bmatrix} \in R^{784}$$

1: for ink
0: otherwise

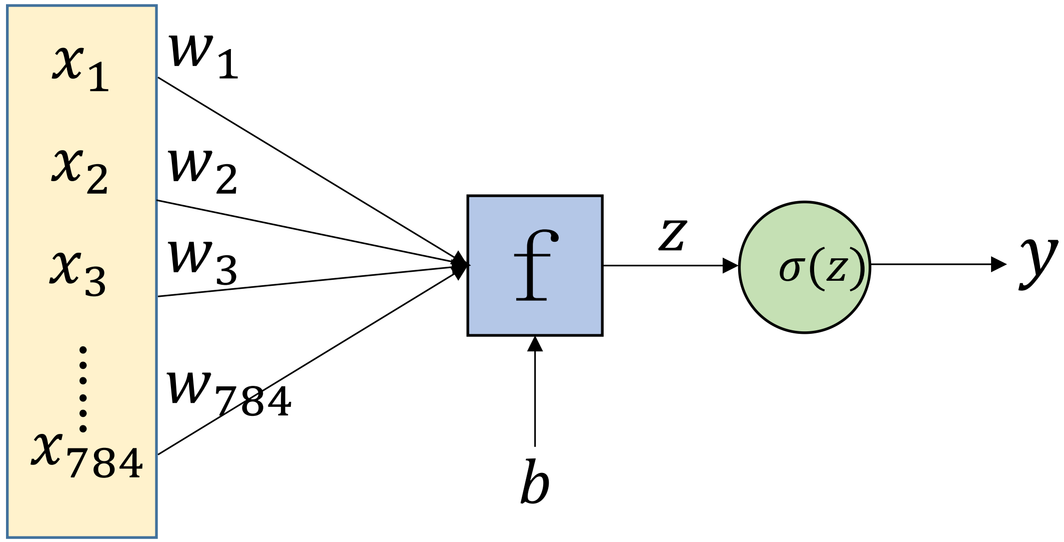
Input domain

Hand-written Digits Recognition

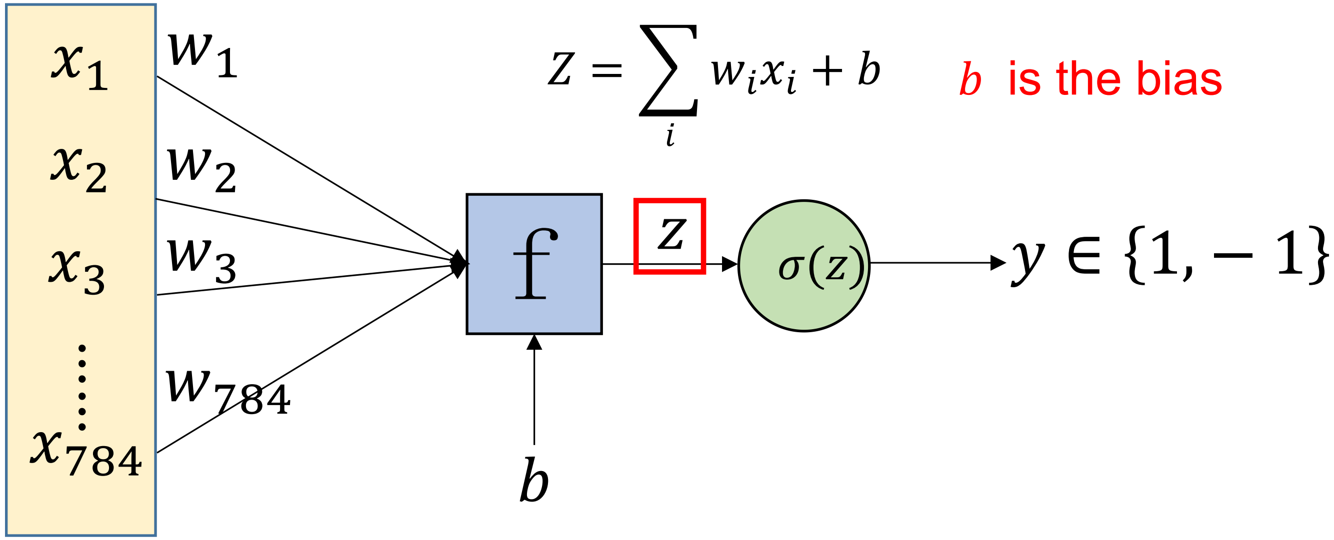


1 or not?

Single Neuron



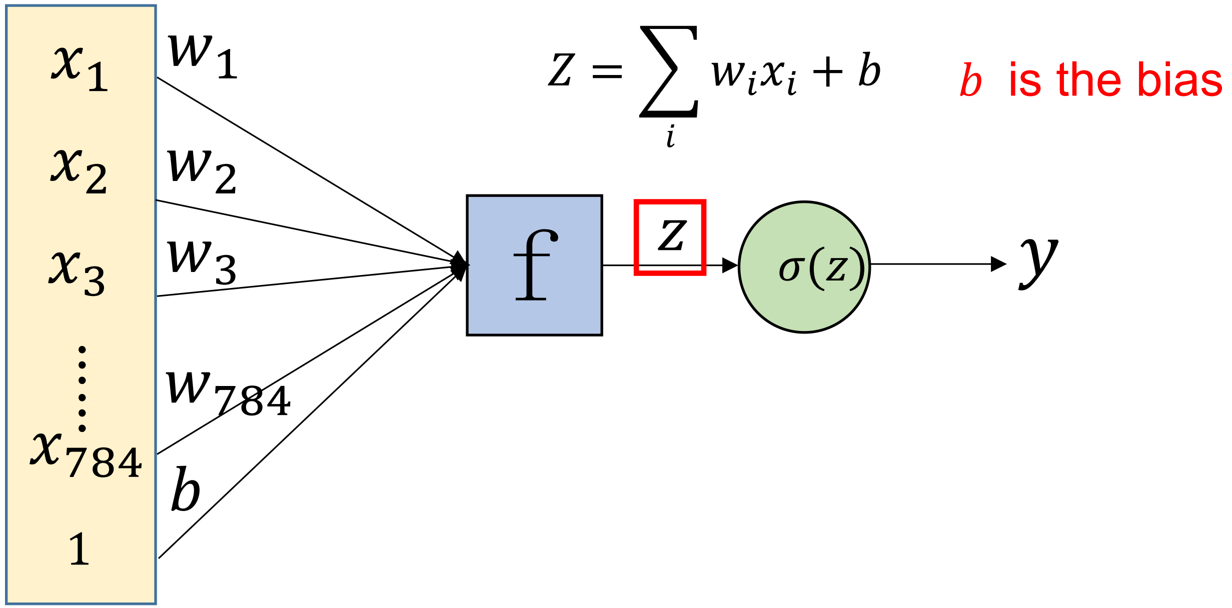
Single Neuron

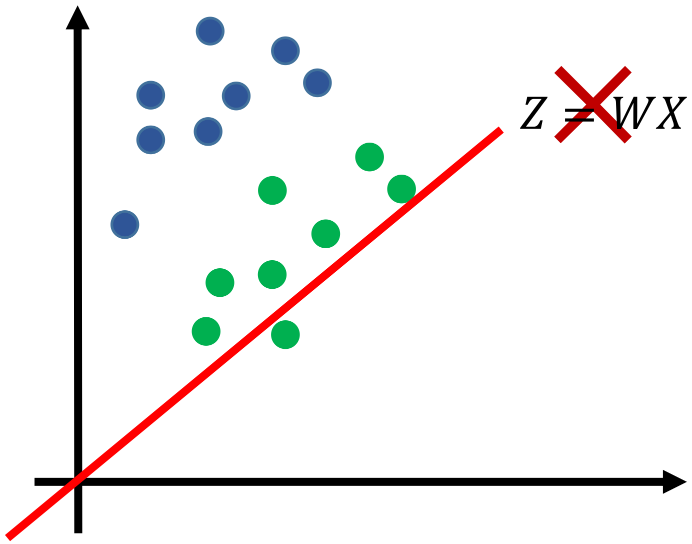


$$Z = \sum_i w_i x_i + b$$

b is the bias

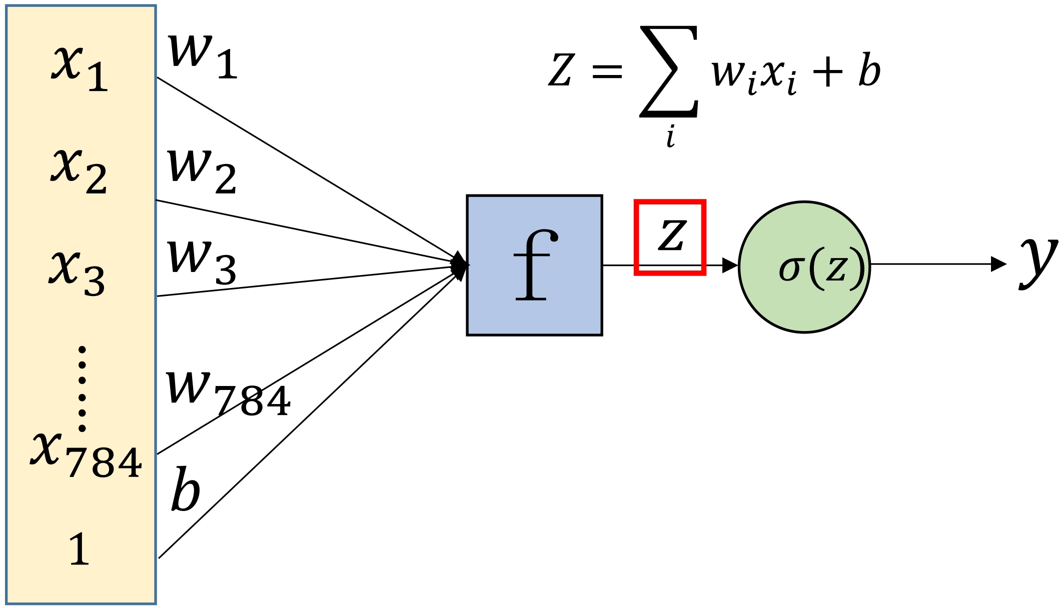
Single Neuron

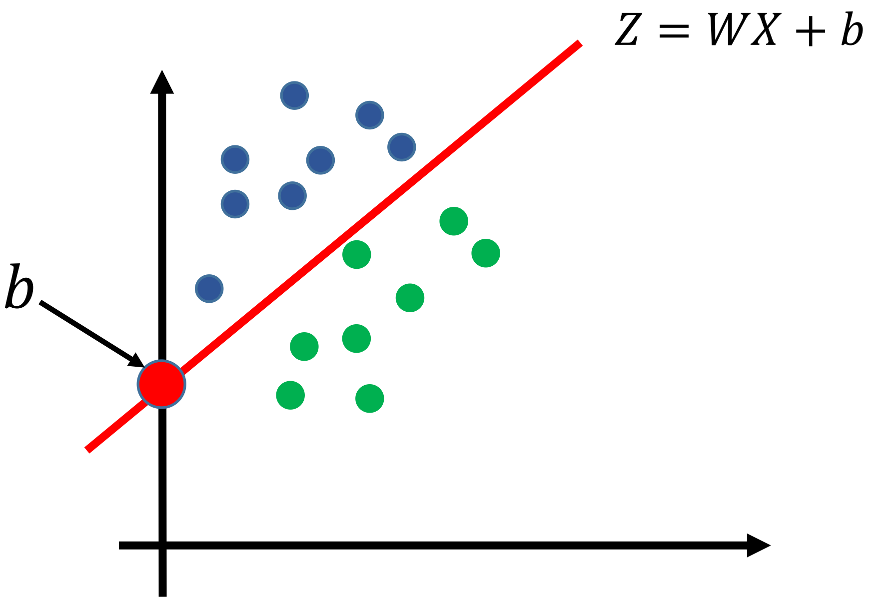




Why do we need a bias ***b***?

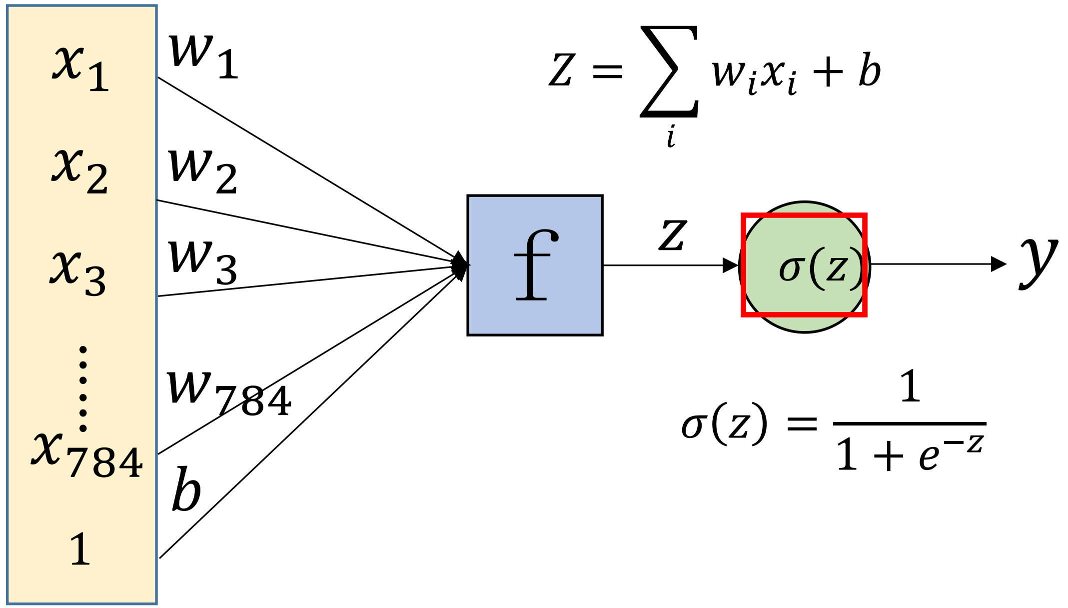
Single Neuron





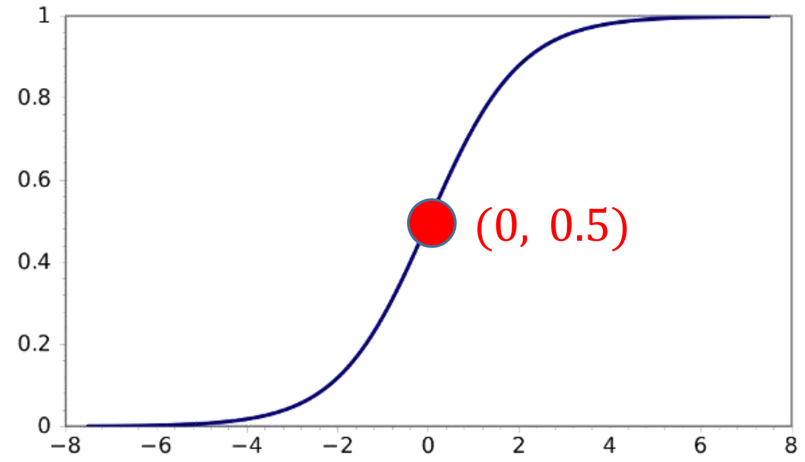
Why do we need a bias $\boldsymbol{b}$?

Single Neuron



$$Z = \sum_i w_i x_i + b$$

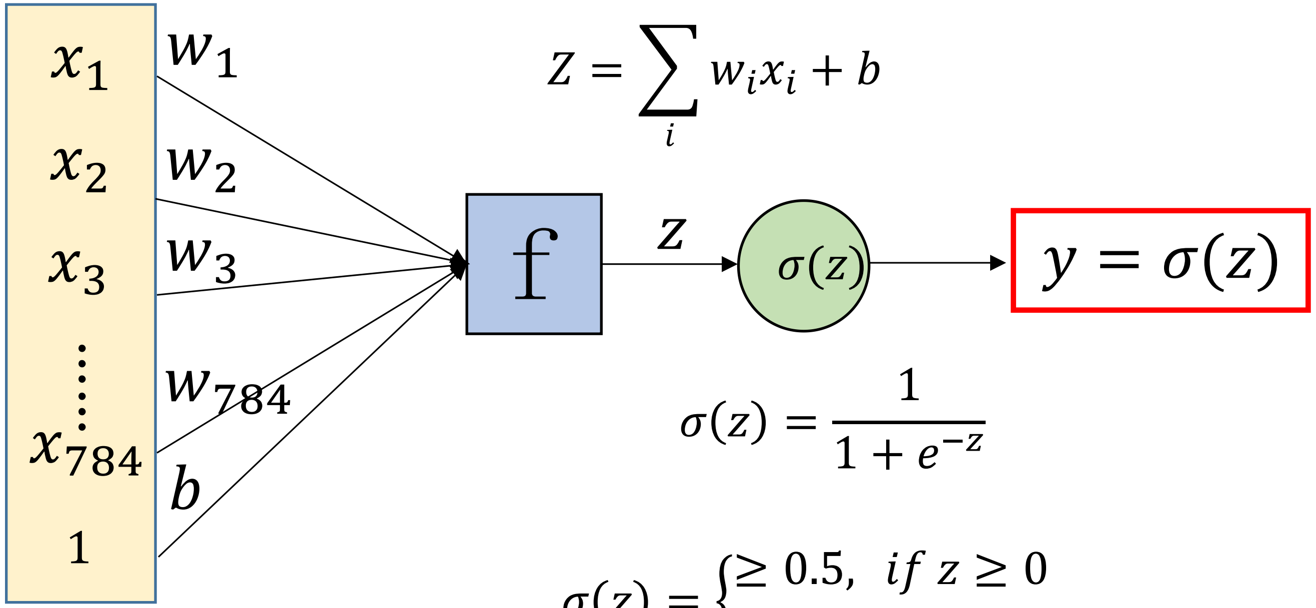
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Sigmoid Function

$$\sigma(z) = \begin{cases} \geq 0.5, & if\ z \geq 0 \\ < 0.5, & if\ z < 0 \end{cases}$$

Single Neuron



$$Z = \sum_i w_i x_i + b$$

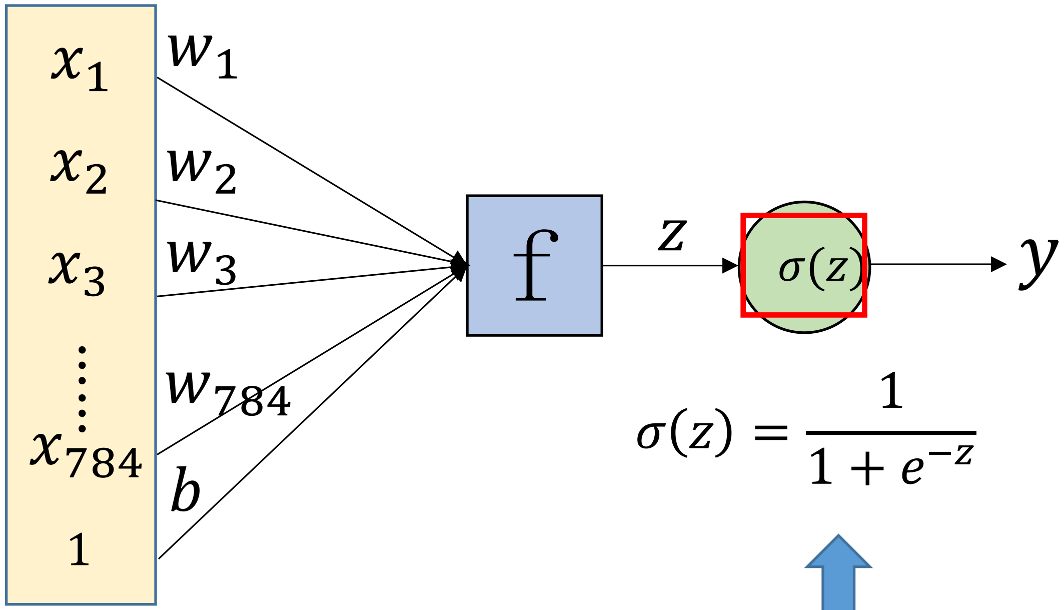
$$prediction = \begin{cases} 1 & if\ y \geq 0.5 \\ -1 & if\ y < 0.5 \end{cases}$$

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

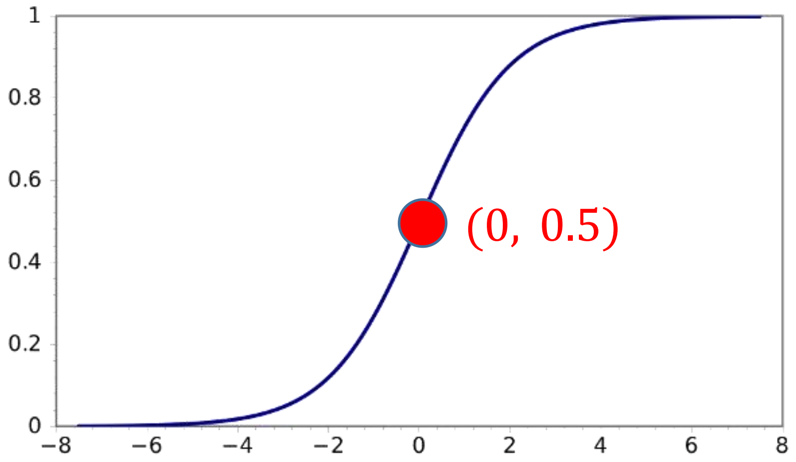
$$\sigma(z) = \begin{cases} \geq 0.5, & if\ z \geq 0 \\ < 0.5, & if\ z < 0 \end{cases}$$

This is a linear classifier.

Activation Function



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

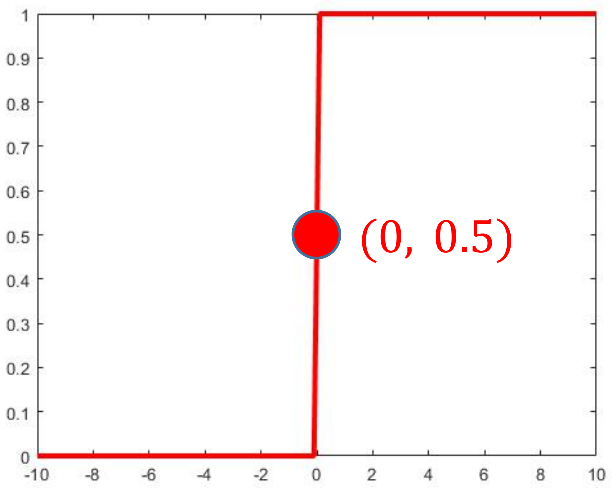


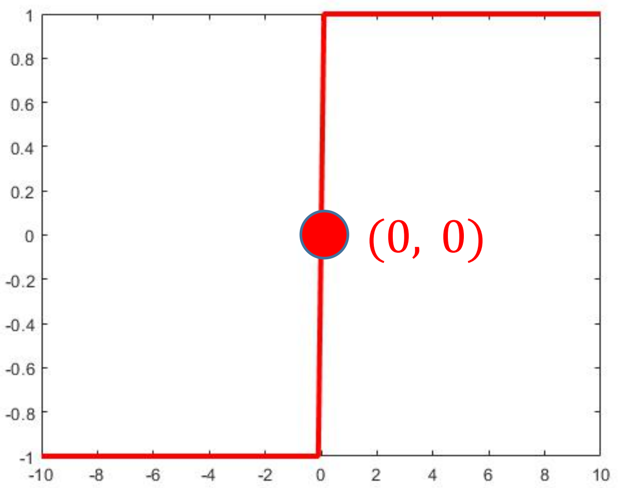
Activation function: The function that acts on the weighted combination of inputs.

We also have other activation function.

Activation Function

Boolean





$$\sigma(z) = \begin{cases} 1 & z > 0 \\ 0.5 & z = 0 \\ 0 & z < 0 \end{cases}$$

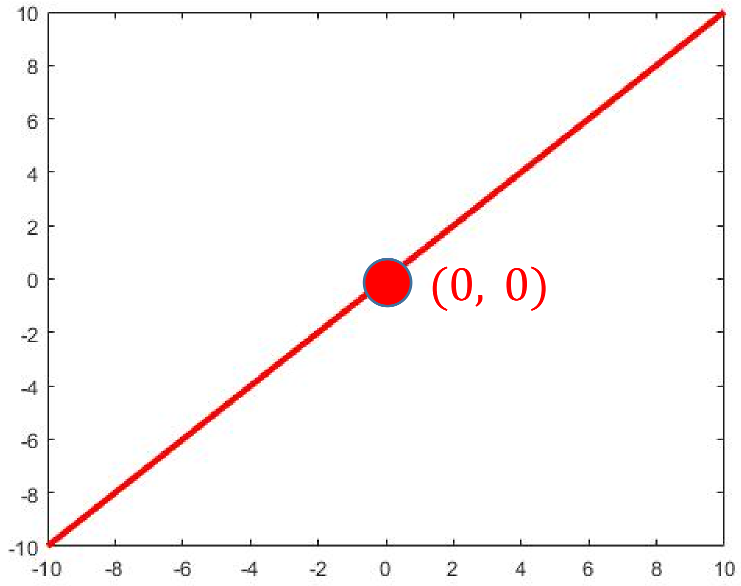
$$\sigma(z) = \begin{cases} 1 & z > 0 \\ 0 & z = 0 \\ -1 & z < 0 \end{cases}$$

Unit step function

Sign function

Activation Function

Linear

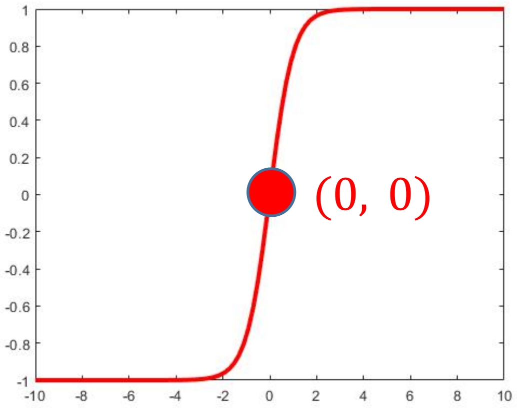


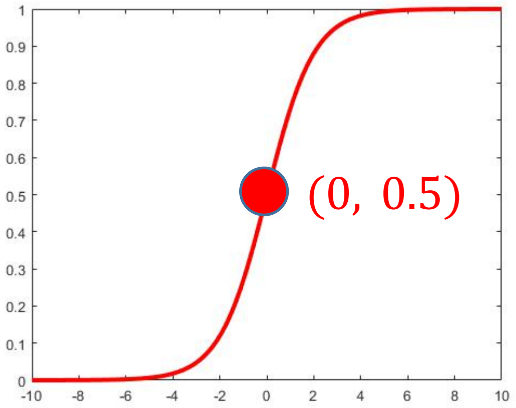
$$\sigma(z) = z$$

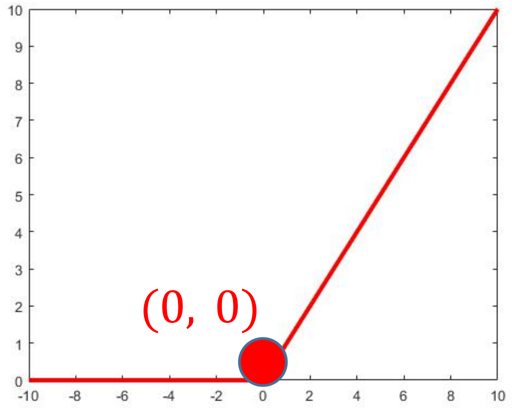
Linear function

Activation Function

Non-linear







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

Tanh function

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

Sigmoid function

$$\sigma(z) = \max(0, z)$$

ReLU function

Non-linear activation functions are frequently used in neural networks.

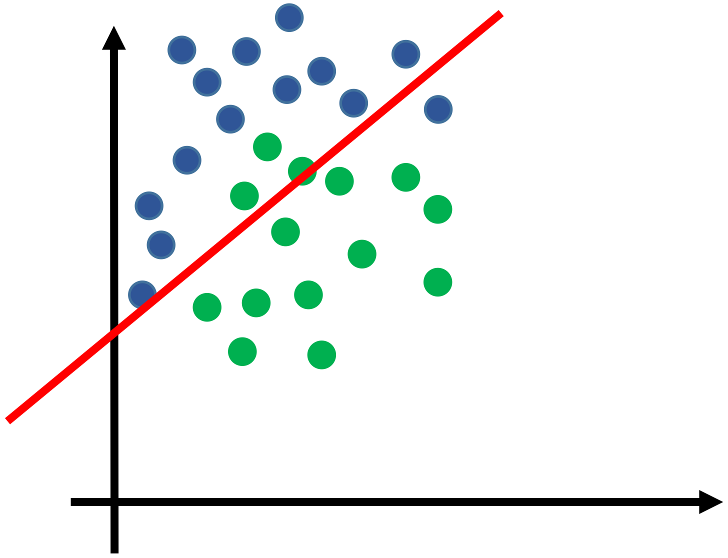
Why?

Why Non-Linearity?

Without non-linearity

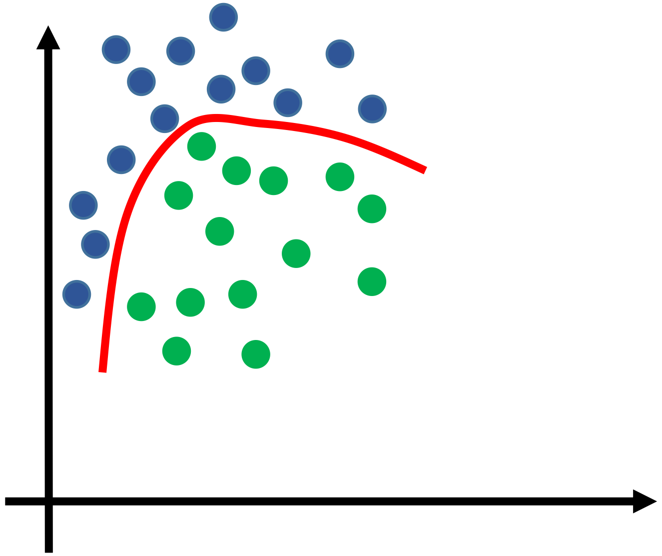
- ➢ Deep neural networks are equivalent to linear transforms.

$$W_1\big(W_2(W_3 \cdot x)\big) = Wx$$



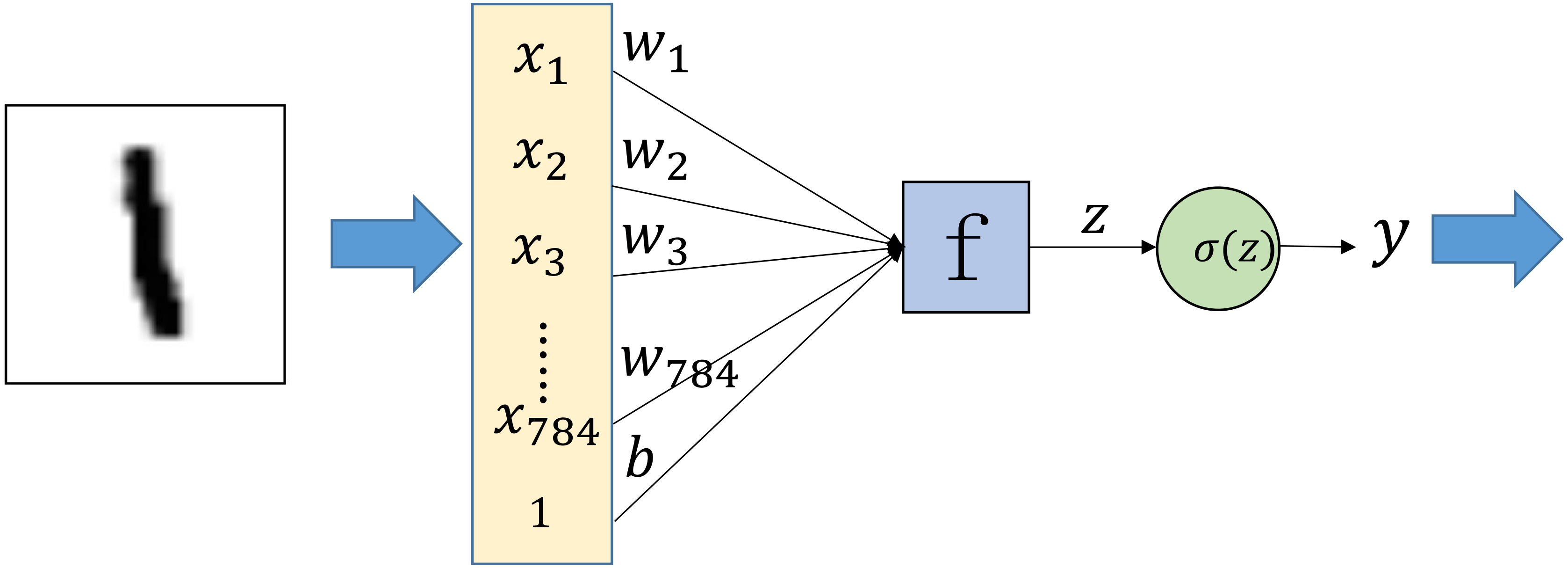
With non-linearity

- ➢ The neural networks can approximate complicated functions.



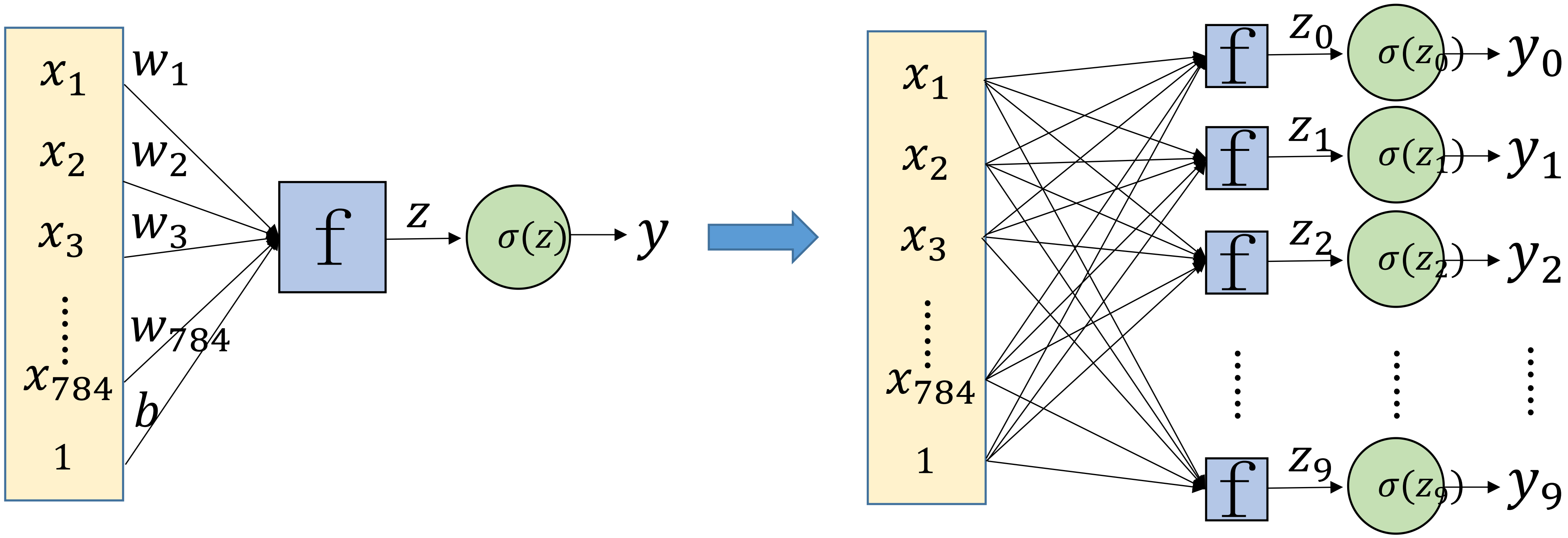
http://cs224d.stanford.edu/lectures/CS224d-Lecture4.pdf

A More Complicated Task



What is the number in the image?

Multiple Outputs

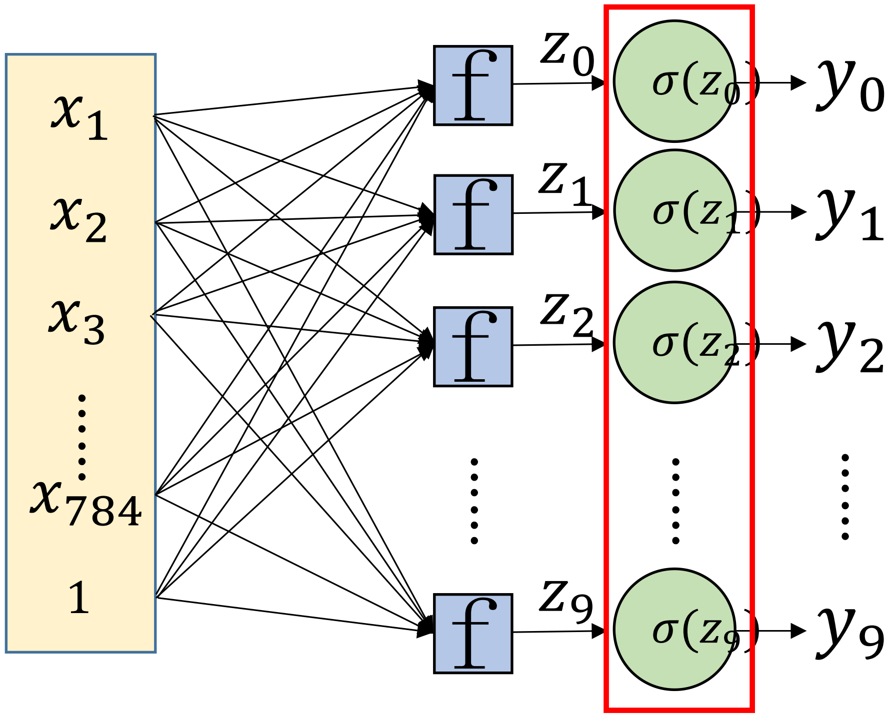


$W \in R^{784\times10}$

$b \in R^{1\times10}$

Multiple Outputs

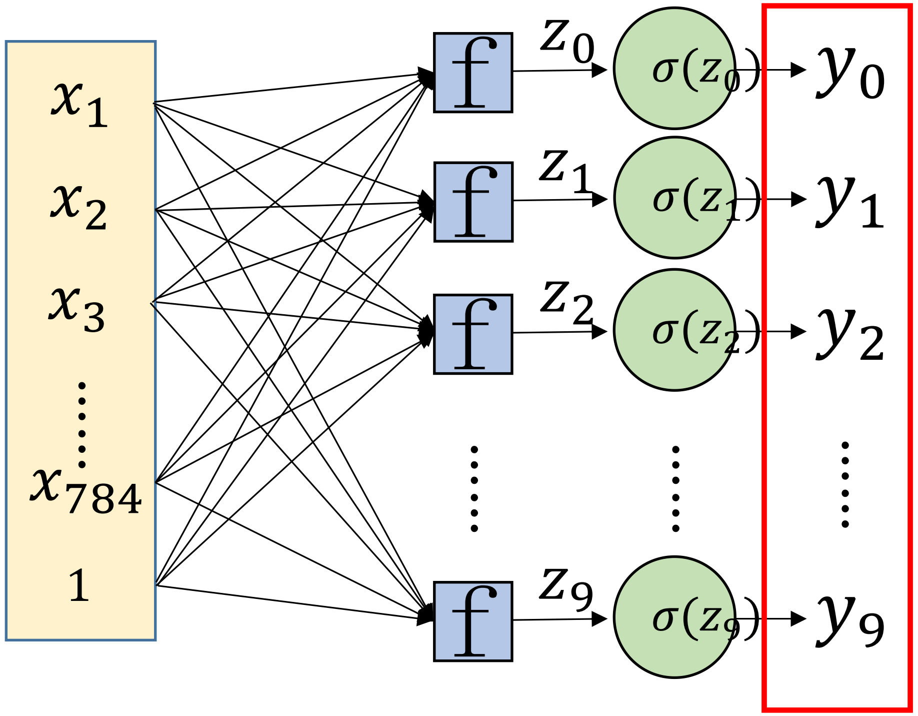
$$\sigma(z) = \frac{1}{1+e^{-z}}$$



$W \in R^{784\times10}$

$b \in R^{1\times10}$

Multiple Outputs

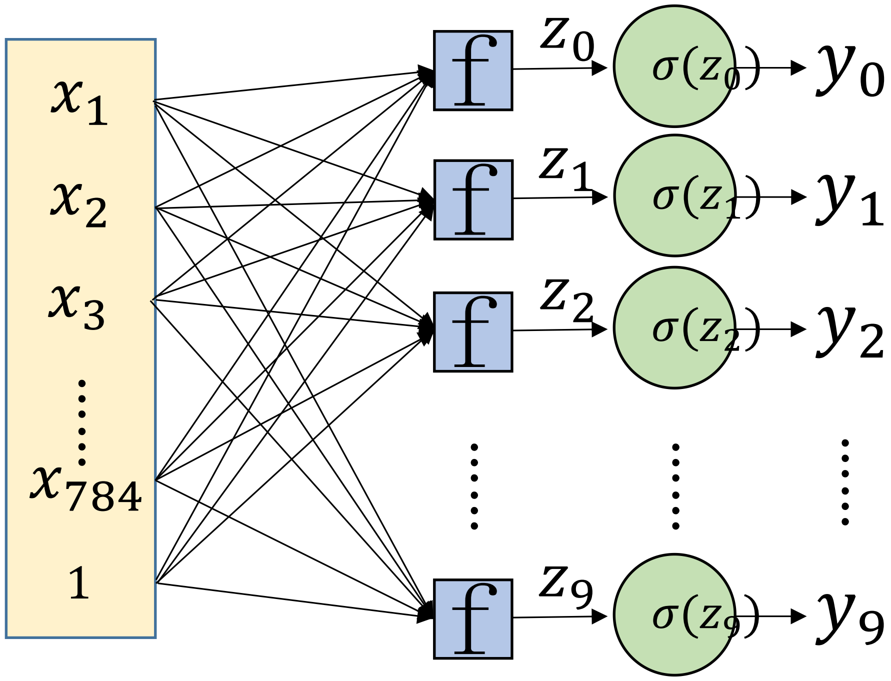


We choose label corresponding to the maximum value of y_i.

$W \in R^{784 \times 10}$

$b \in R^{1 \times 10}$

Multiple Outputs



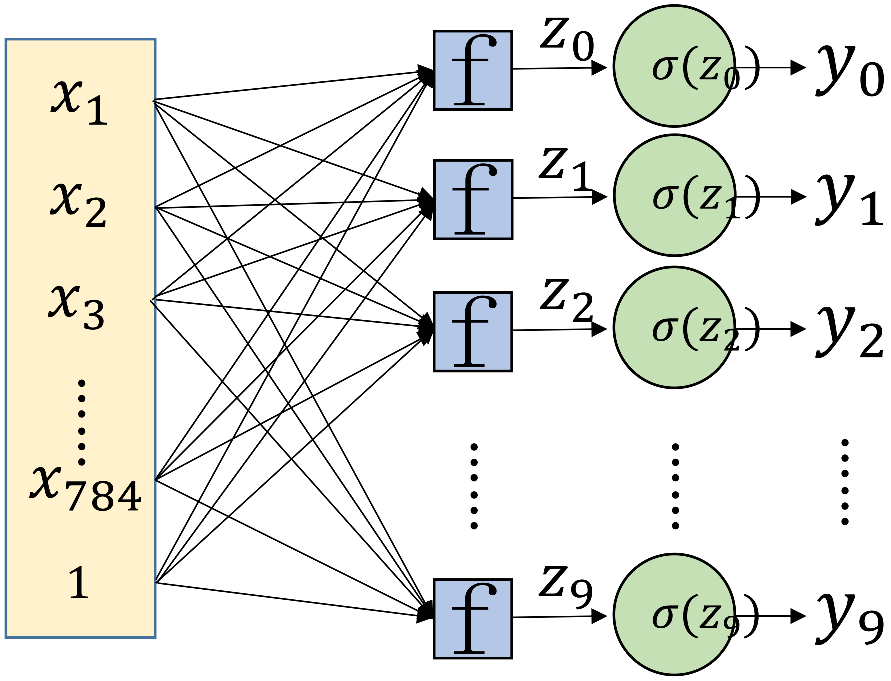
$W \in R^{784\times 10}$

$b \in R^{1\times 10}$

Question：

- How do we evaluate the performance of the model?
- How we model the optimization problem?

Loss Function



$W \in R^{784\times 10}$

$b \in R^{1\times 10}$

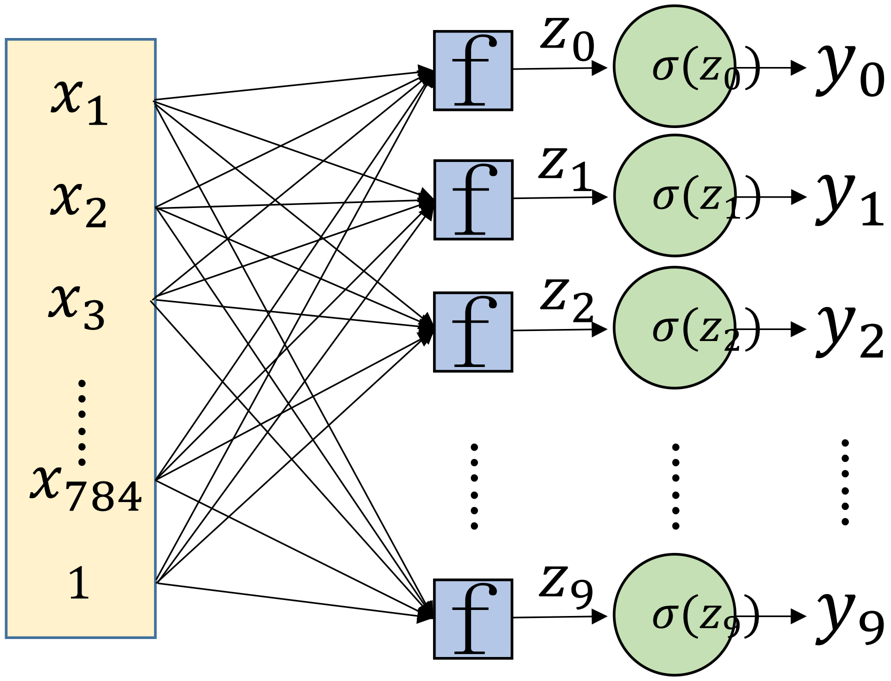
Ground truth: $Q = \begin{bmatrix} 0 \\ 1 \\ \dots \\ \dots \\ 0 \end{bmatrix} \in R^{10}$ One hot vector The component corresponding to the true label is "1".

$$p_i = softmax(y_i) = \frac{e^{y_i}}{\sum_i e^{y_i}}$$

$$Loss = cross\ entropy = -\sum_i q_i \log(p_i)$$

The goal is to minimize the loss!

Model Parameters



$W \in R^{784\times10}$

$b \in R^{1\times10}$

$$y = f(x) = \sigma(Wx + b)$$

Model parameter set $\theta = \{W, b\}$

Minimize the loss = Pick the best θ

Optimization

Any idea to pick the optimal parameter values ?

Optimization

Any idea to pick the optimal parameter values ?

↓

(Stochastic) Gradient Descent

↓

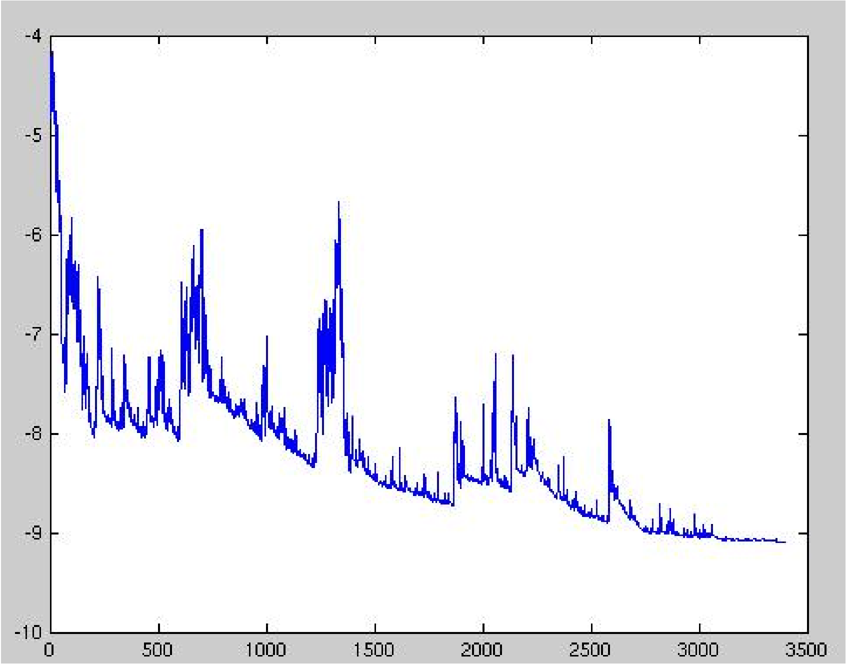
Backpropagation

Stochastic Gradient Descent

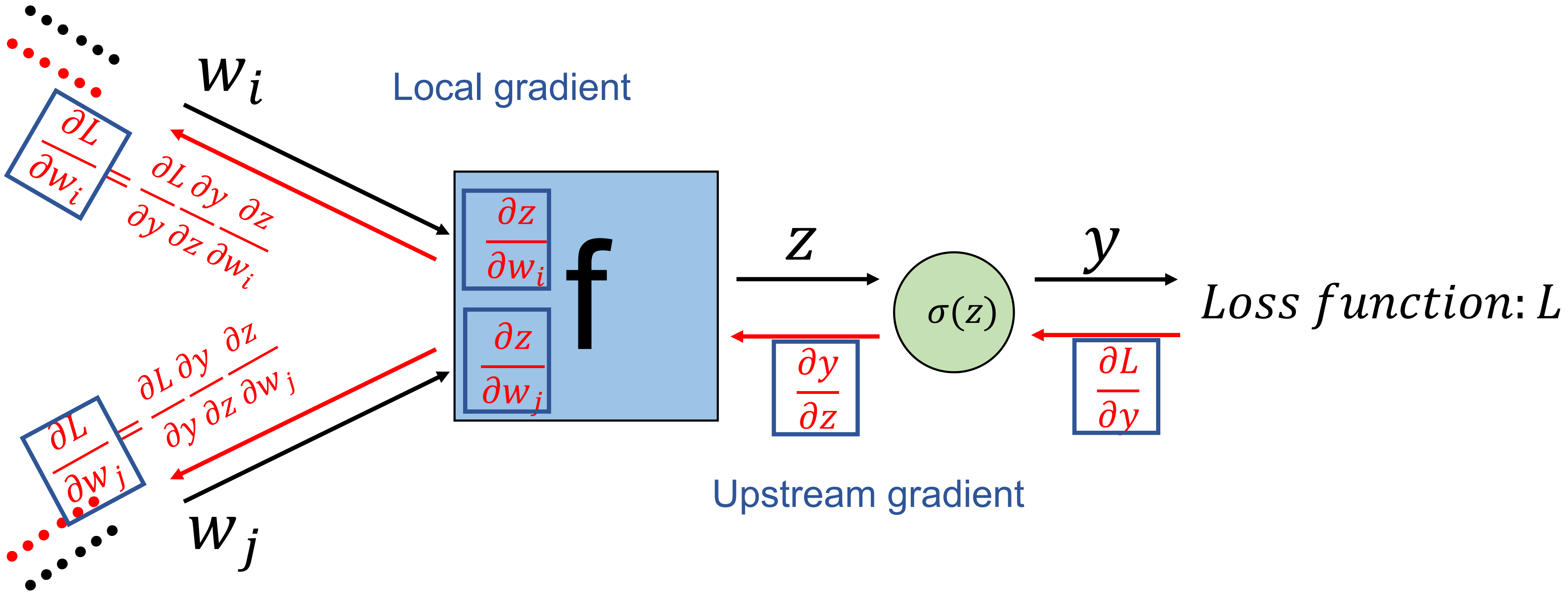
$$\min_x F(x) = \sum_{i=1}^{n} f_i(x)$$

- Initialize the parameter x and learning rate η
- Repeat until the termination condition is met
  - Randomly shuffle examples in the training set
  - For $i = 1, \ldots, n$

$$x_{k+1} \leftarrow x_k - \eta \nabla f_i(x_k)$$

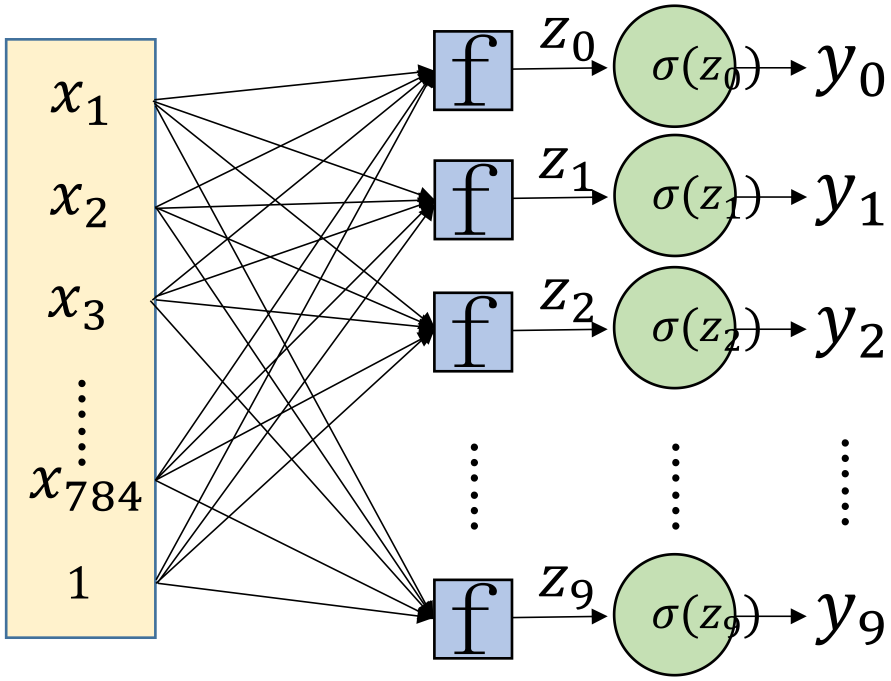


Backpropagation



Upstream gradient * Local gradient

Backpropagation



$W \in R^{784 \times 10}$

$b \in R^{1 \times 10}$

Ground truth: $Q = \begin{bmatrix} 0 \\ 1 \\ \dots \\ \dots \\ 0 \end{bmatrix} \in R^{10}$

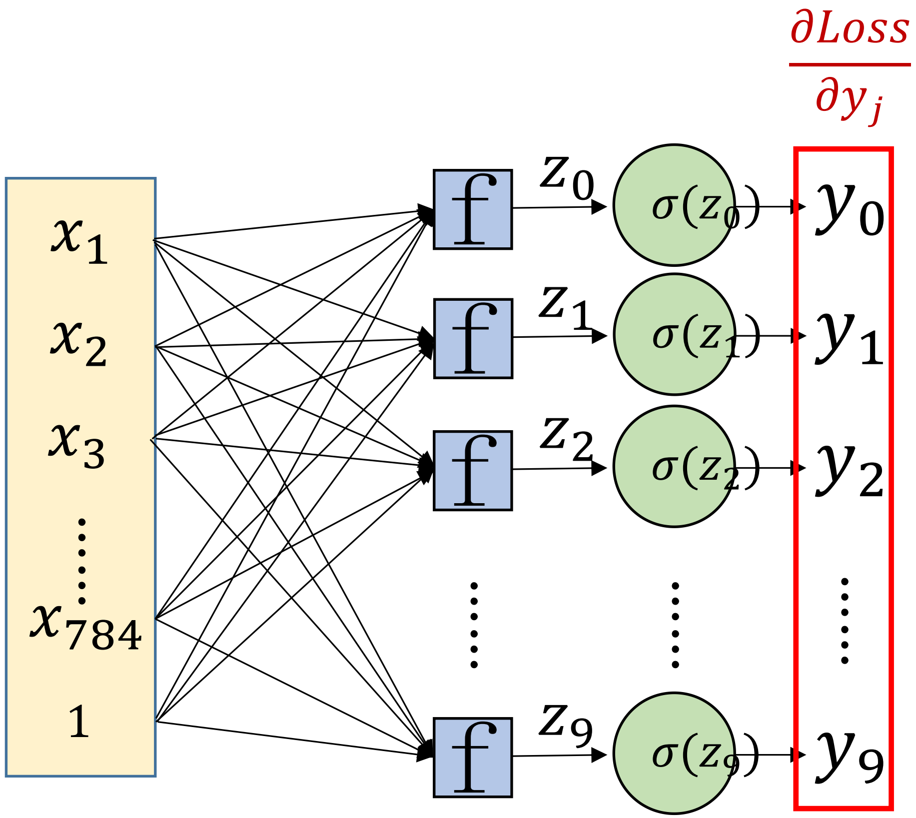
One hot vector: the component corresponding to the true label is “1”.

$$p_i = softmax(y_i) = \frac{e^{y_i}}{\sum_i e^{y_i}}$$

Suppose that, the true label of a given data instance is k. Then

$$Loss = cross\ entropy = -\sum_i q_i \log(p_i) = -\log(p_k)$$

Backpropagation



$W \in R^{784\times10}$

$b \in R^{1\times10}$

Suppose that, the true label of a given data instance is k.
Then

$$Loss = cross\ entropy = -\log(p_k)$$

$$p_k = softmax(y_k) = \frac{e^{y_k}}{\sum_i e^{y_i}}$$

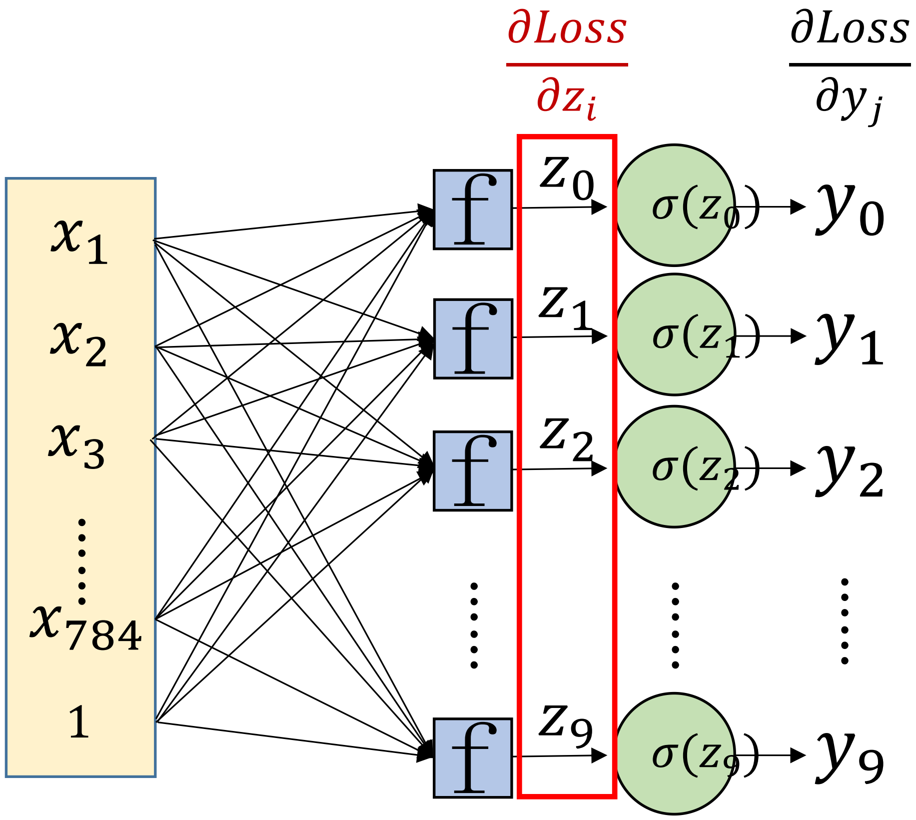
$$\frac{\partial Loss}{\partial y_j} = \frac{\partial Loss}{\partial p_k}\frac{\partial p_k}{\partial y_j}$$

$$\frac{\partial Loss}{\partial p_k} = -\frac{1}{p_k}$$

$$\frac{\partial p_k}{\partial y_j} = \begin{cases} p_k(1-p_k) & k=j \\ -p_k p_j & k \neq j \end{cases}$$

$$\frac{\partial Loss}{\partial y_j} = \frac{\partial Loss}{\partial p_k}\frac{\partial p_k}{\partial y_j} = \begin{cases} p_j - 1 & k=j \\ p_j & k \neq j \end{cases}$$

Backpropagation



$W \in R^{784\times 10}$

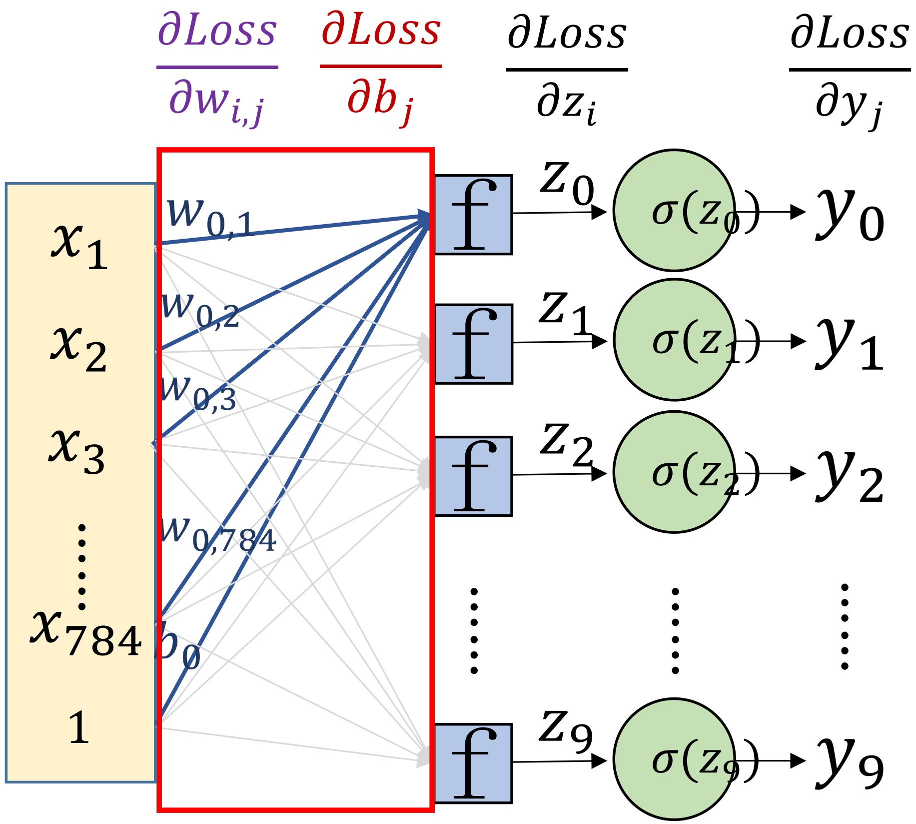
$b \in R^{1\times 10}$

$$y_i = \frac{1}{1+e^{-z_i}}$$

$$\frac{\partial Loss}{\partial z_i} = \frac{\partial Loss}{\partial y_i}\frac{\partial y_i}{\partial z_i}$$

$$\frac{\partial y_i}{\partial z_i} = y_i(1-y_i)$$

Backpropagation



$W \in R^{784\times 10}$

$b \in R^{1\times 10}$

$$z_i = w_{i,1}x_1 + w_{i,2}x_2 + \dots + w_{i,784}x_{784} + b_i$$

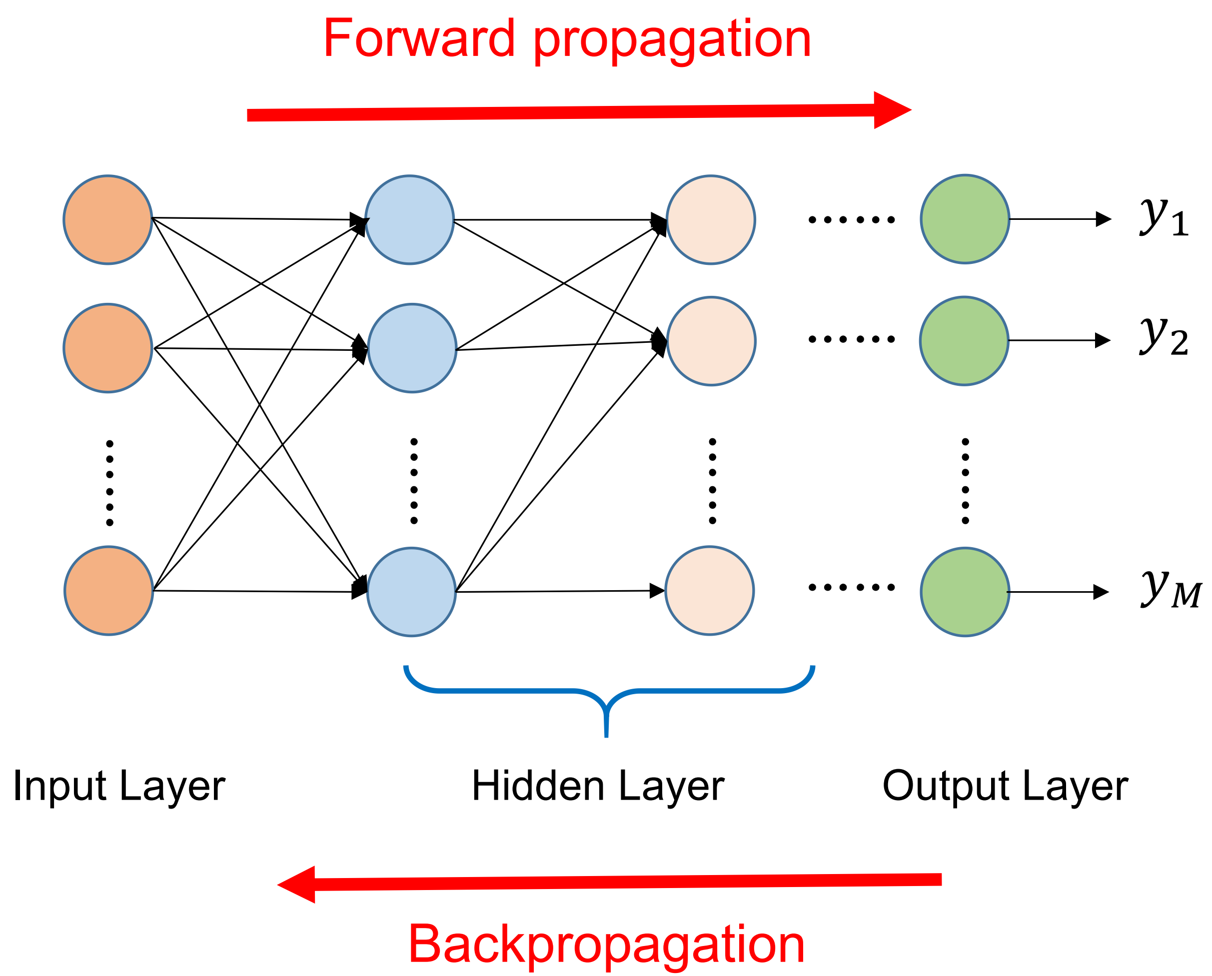
$$\frac{\partial Loss}{\partial w_{i,j}} = \frac{\partial Loss}{\partial z_i}\frac{\partial z_i}{\partial w_{i,j}} \qquad \frac{\partial z_i}{\partial w_{i,j}} = x_i$$

$$\frac{\partial Loss}{\partial b_i} = \frac{\partial Loss}{\partial z_i}\frac{\partial z_i}{\partial b_i} \qquad \frac{\partial z_i}{\partial b_i} = 1$$

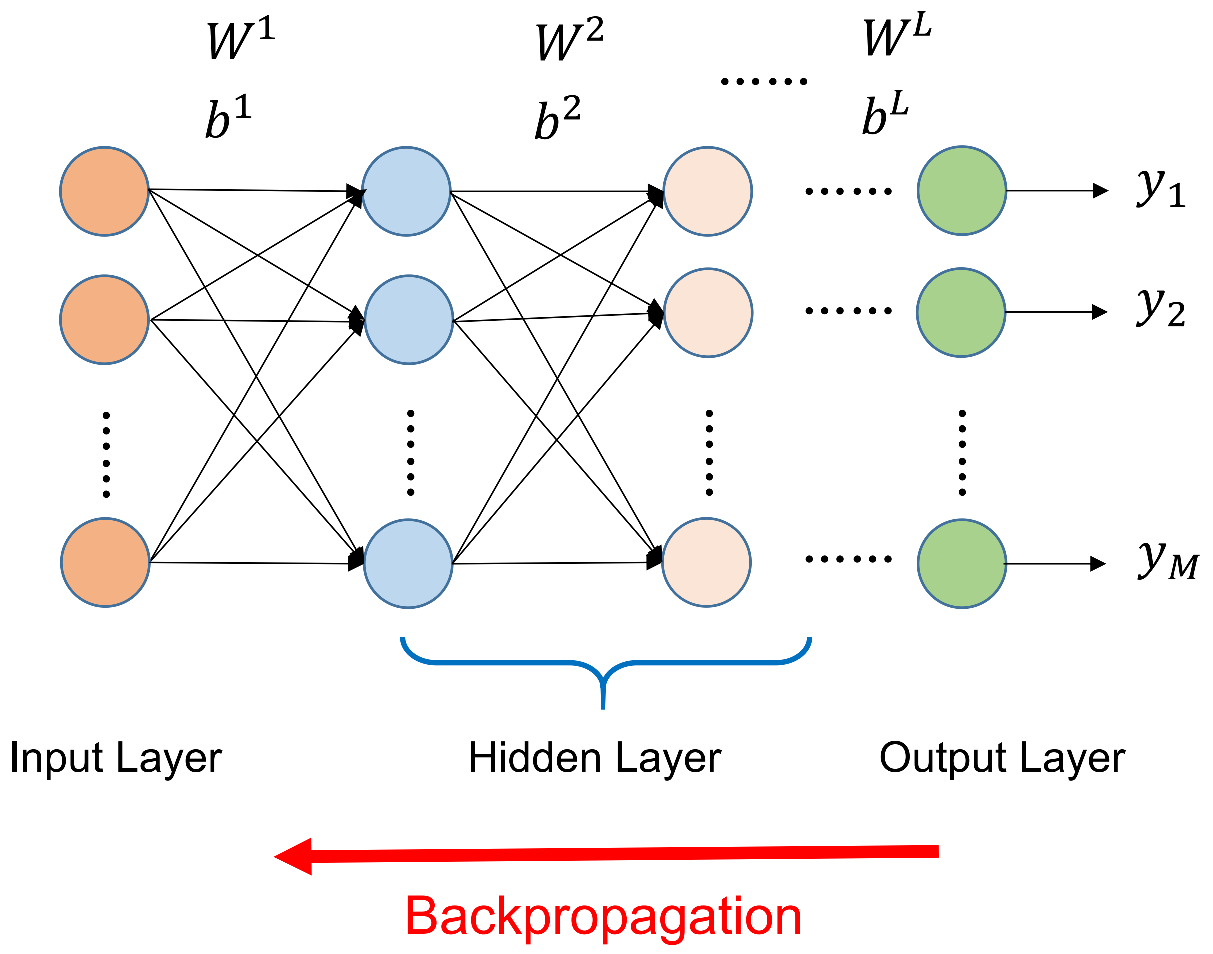
$$W = W - \eta\frac{\partial Loss}{\partial W}$$

$$b = b - \eta\frac{\partial Loss}{\partial b}$$

Backpropagation：Multi-Layer Perceptron



Backpropagation：Multi-Layer Perceptron



$$\theta = \{W^1, b^1, W^2, b^2, ..., W^L, b^L\}$$

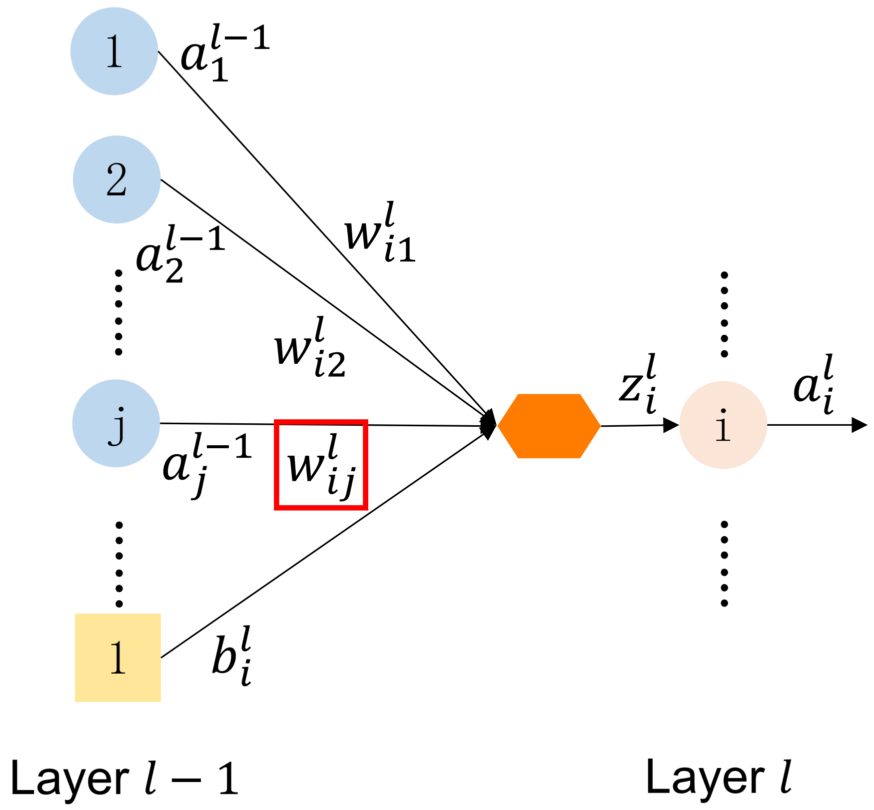
$$W^l = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \qquad b^l = \begin{bmatrix} \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

$$\frac{\partial Loss(\theta)}{\partial W^l} = \begin{bmatrix} \frac{\partial Loss(\theta)}{\partial W_{11}^l} & \frac{\partial Loss(\theta)}{\partial W_{12}^l} & \cdots \\ \frac{\partial Loss(\theta)}{\partial W_{21}^l} & \frac{\partial Loss(\theta)}{\partial W_{22}^l} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

$$\frac{\partial Loss(\theta)}{\partial b^l} = \begin{bmatrix} \vdots \\ \frac{\partial Loss(\theta)}{\partial b_i^l} \\ \vdots \end{bmatrix}$$

$$W = W - \eta \frac{\partial Loss}{\partial W} \qquad b = b - \eta \frac{\partial Loss}{\partial b}$$

Backpropagation：Multi-Layer Perceptron



$\boldsymbol{a_i^l}$: output of a neuron

$\boldsymbol{w_{ij}^l}$: a weight of layer $\boldsymbol{l}$

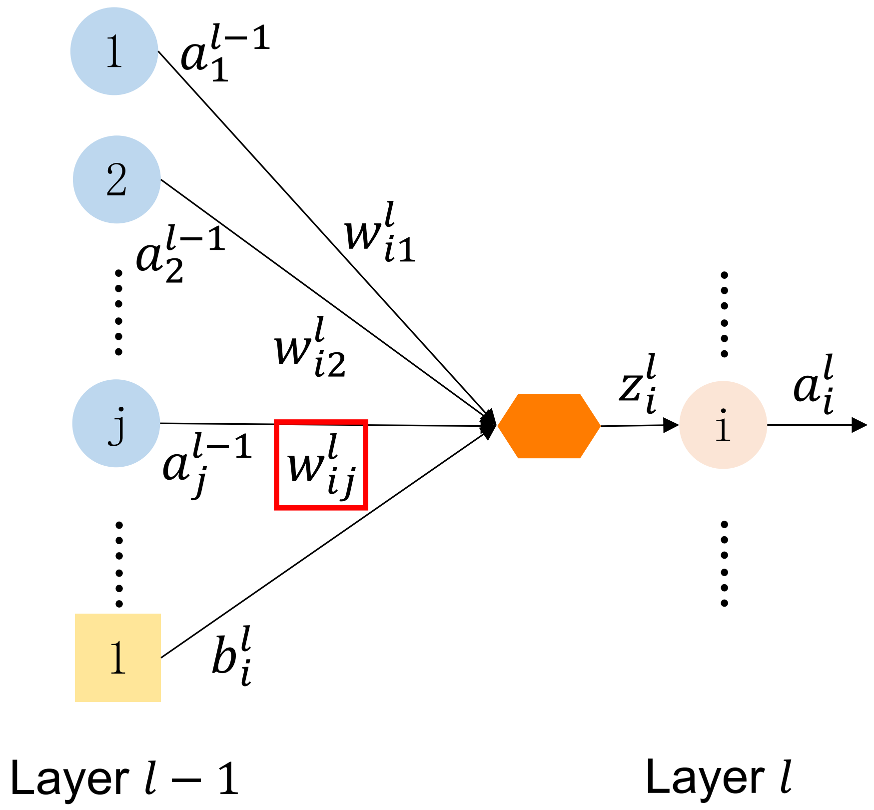
$\boldsymbol{b_i^l}$: a bias of layer $\boldsymbol{l}$

$\boldsymbol{z_i^l}$: input of an activation function

$$z^l = W^l a^{l-1} + b^l$$

$$a^l = \sigma(z^l)$$

Backpropagation：Multi-Layer Perception



$$\frac{\partial Loss(\theta)}{\partial w_{ij}^l} = \frac{\partial Loss(\theta)}{\partial z_i^l}\frac{\partial z_i^l}{\partial w_{ij}^l}$$

If l >1:

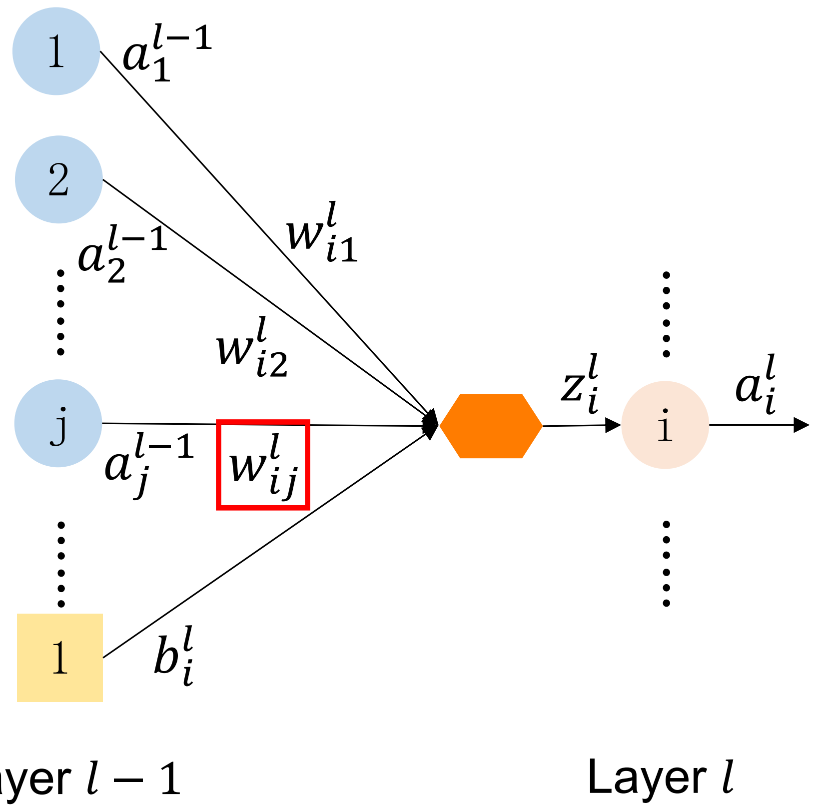
$$z_i^l = \sum_j w_{ij}^l a_j^{l-1} + b_i^l$$

$$\frac{\partial z_i^l}{\partial w_{ij}^l} = a_j^{l-1}$$

If l=1:

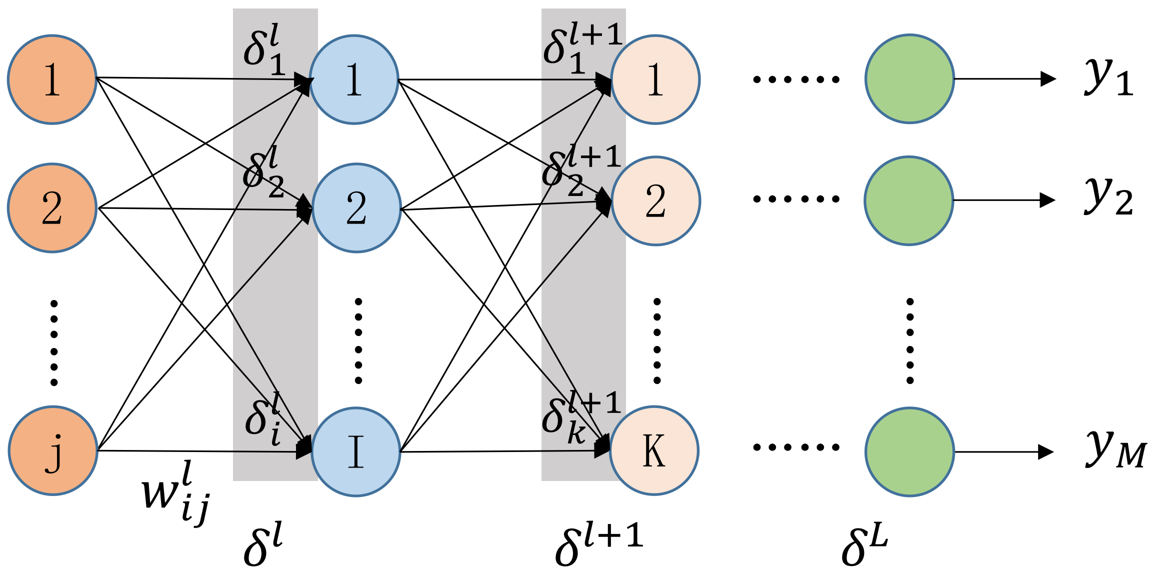
$$\frac{\partial z_i^l}{\partial w_{ij}^l} = x_j$$

Backpropagation：Multi-Layer Perception

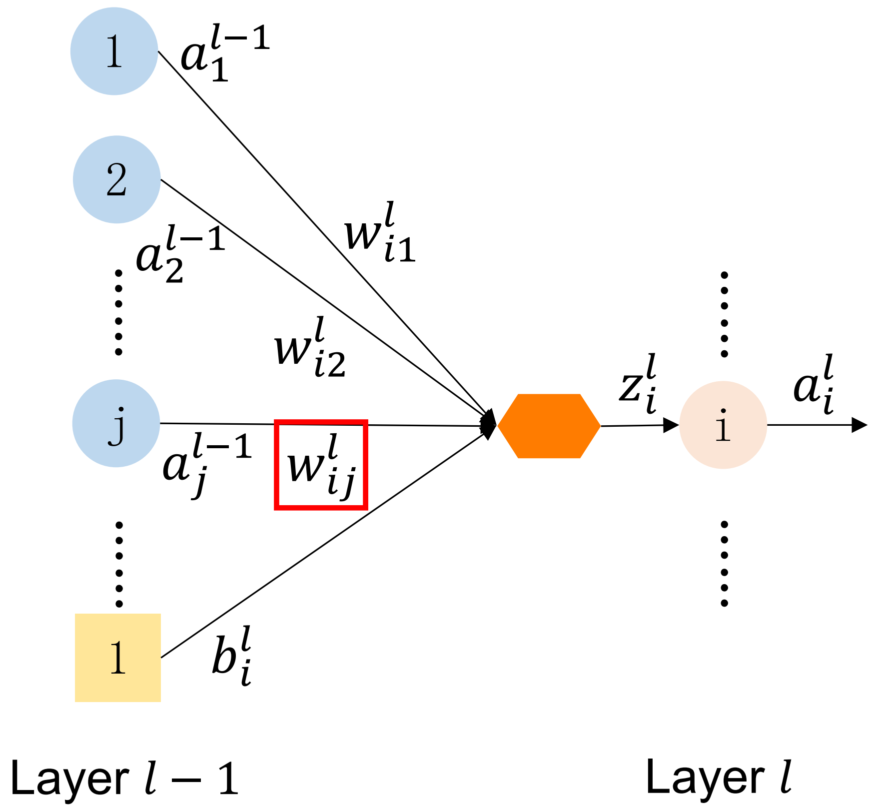


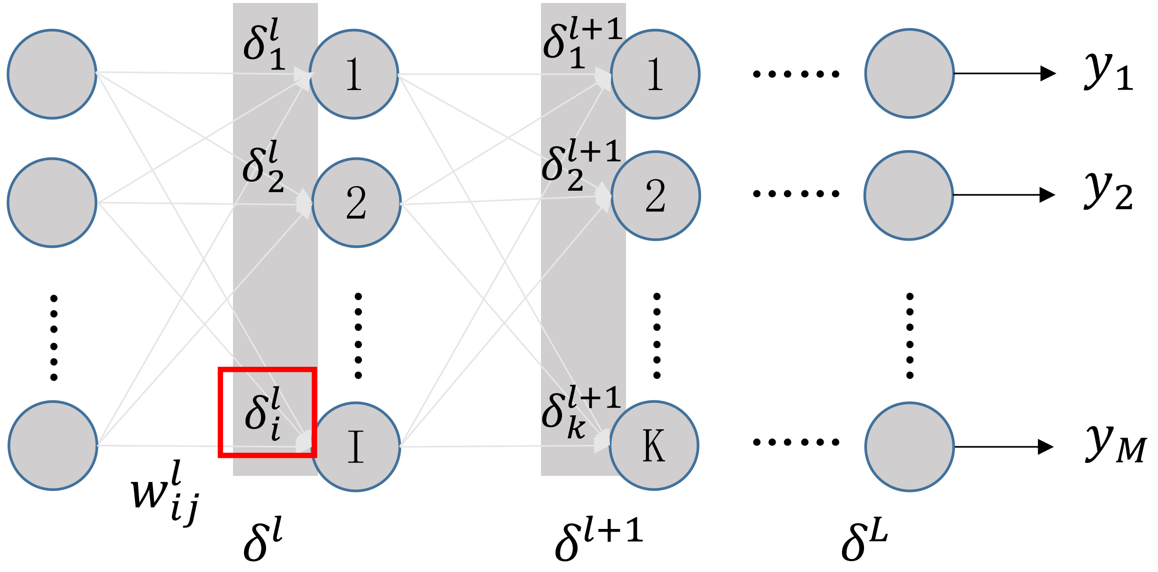
$$\frac{\partial Loss(\theta)}{\partial w_{ij}^l} = \frac{\partial Loss(\theta)}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l}$$

$$\delta_i^l = \frac{\partial Loss(\theta)}{\partial z_i^l}$$

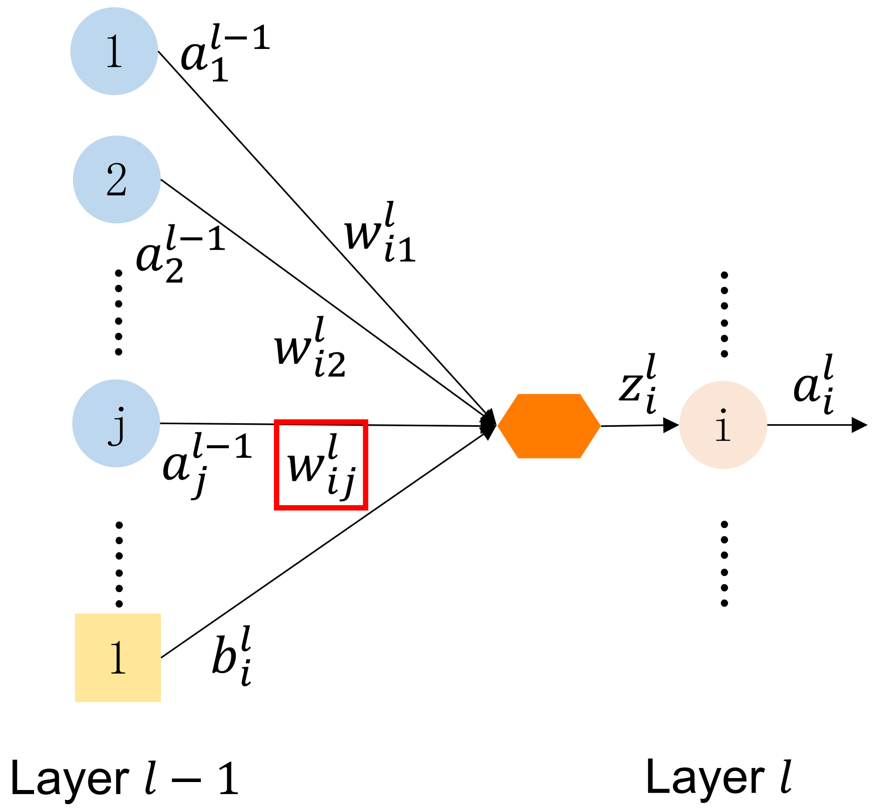


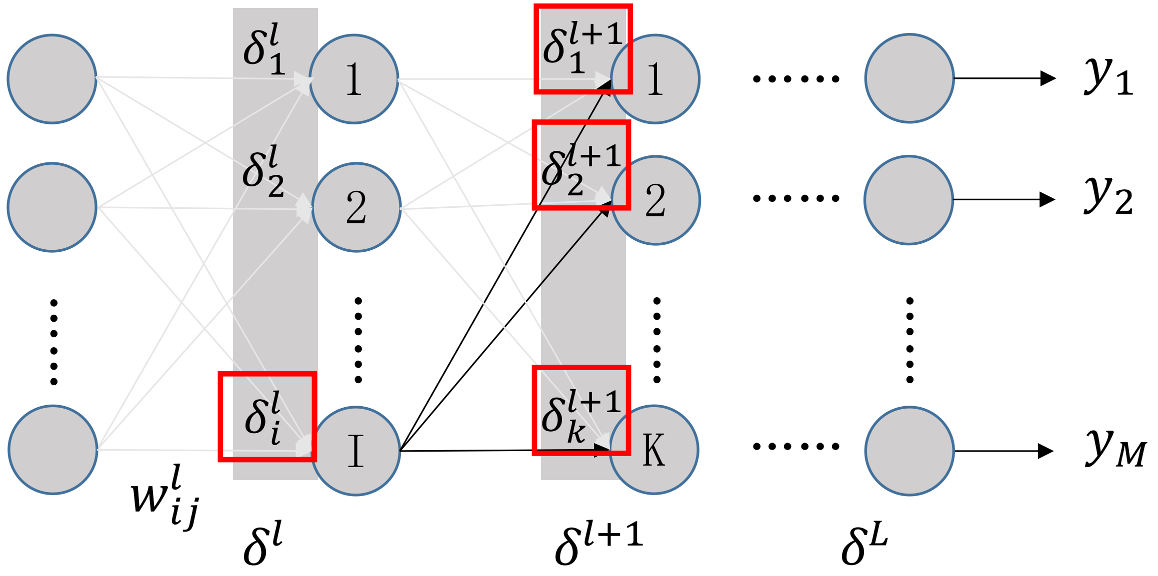
Backpropagation：Multi-Layer Perception





Backpropagation：Multi-Layer Perception





$$\delta_i^l = \frac{\partial Loss(\theta)}{\partial z_i^l} = \frac{\partial Loss(\theta)}{\partial z_1^{l+1}} \frac{\partial z_1^{l+1}}{\partial a_i^l} \frac{\partial a_i^l}{\partial z_i^l} + \ldots + \frac{\partial Loss(\theta)}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial a_i^l} \frac{\partial a_i^l}{\partial z_i^l}$$

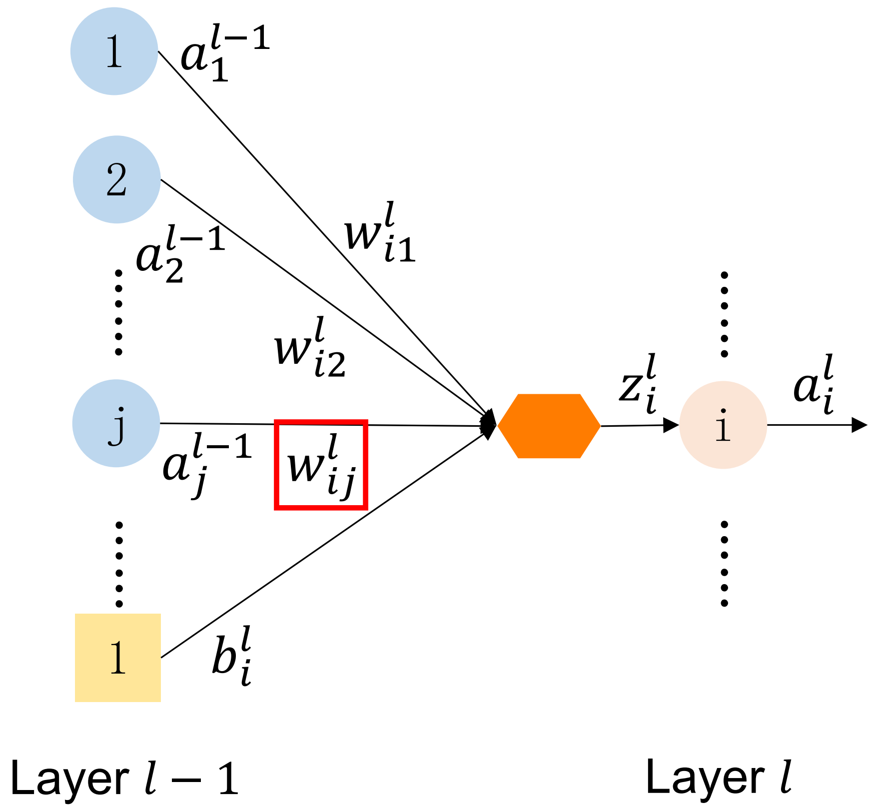
$$= \frac{\partial a_i^l}{\partial z_i^l} \sum_k \frac{\partial Loss(\theta)}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial a_i^l} = \frac{\partial a_i^l}{\partial z_i^l} \sum_k \frac{\partial z_k^{l+1}}{\partial a_i^l} \delta_k^{l+1}$$

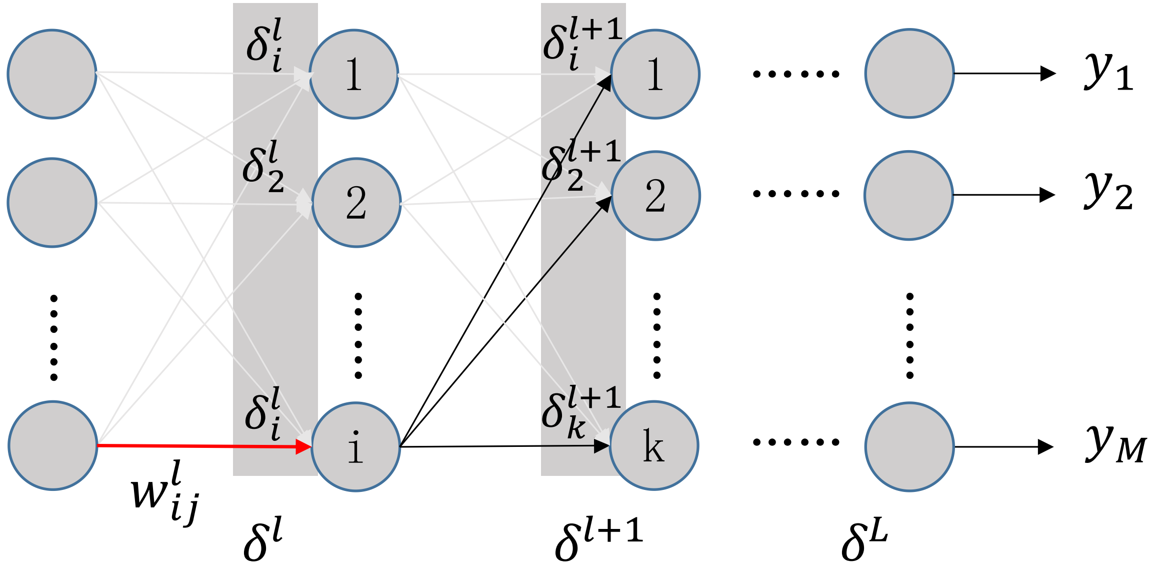
$$z_k^{l+1} = \sum_i w_{ki}^{l+1} a_i^l + b_k^{l+1}$$

$$= \sigma'(z_i^l) \sum_k w_{ki}^{l+1} \delta_k^{l+1}$$

$$\delta^l = \sigma'(z^l) \odot \left(\left(W^{l+1}\right)^T \delta^{l+1} \right)$$

Backpropagation：Multi-Layer Perception



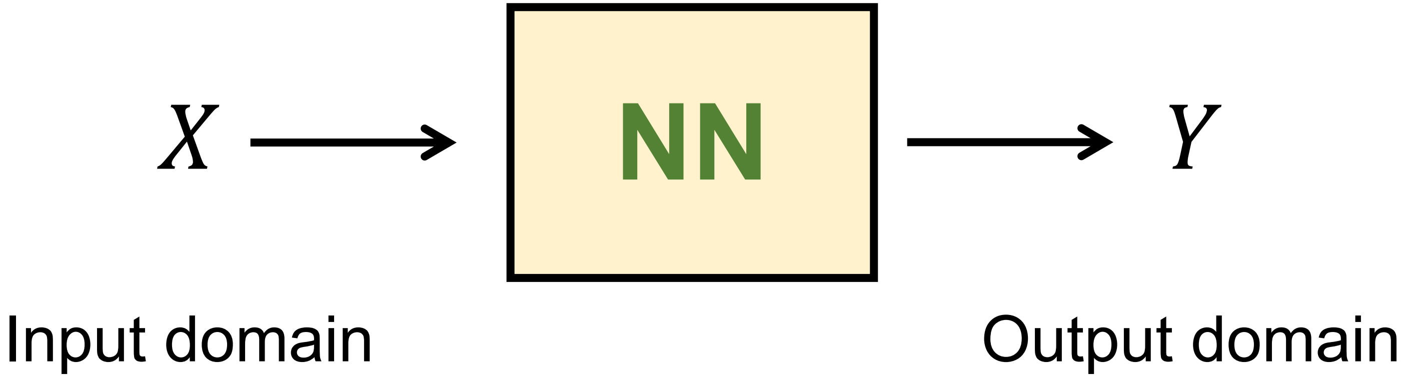


$$\frac{\partial Loss(\theta)}{\partial w_{ij}^l} = \frac{\partial Loss(\theta)}{\partial z_i^l}\frac{\partial z_i^l}{\partial w_{ij}^l} = \delta_i^l \frac{\partial z_i^l}{\partial w_{ij}^l}$$

$$\delta_i^l = \sigma'(z_i^l)\sum_k w_{ki}^{l+1}\delta_k^{l+1}$$

$$\frac{\partial z_i^l}{\partial w_{ij}^l} = \begin{cases} a_j^{l-1} & l>1 \\ x_j & l=1 \end{cases}$$

Universal Function Approximator



- Input domain: document, word, image, voice, etc.
- Output domain: probability distribution, single label, etc.

Universal Function Approximator

The learning algorithm is to map the input domain X into the output domain Y

$$f : X \longrightarrow Y$$

- Handwriting Recognition

$f($  $) =$ "1"

- Speech Recognition

$f($  $) =$ "Hello, MIRA"

In fact, the neural networks are universal function approximators!

Universal Function Approximator

$$y = f(x;\theta) = \sigma(W^L ... \sigma(W^2\sigma(W^1 x + b^1) + b^2) ... + b^L)$$

Different model parameters W and b determine different mappings.

Standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy.

------*‘ Multilayer feedforward networks are universal approximators’*

Pick a function f = pick a set of model parameters θ

Universal Function Approximator

- A good function: The output of the function is close to the label.

$$f(x;\theta) \sim y$$

- An example loss function:

$$Loss = \sum_k ||y_k - f(x_k;\theta)||^2$$

where k is the number of training examples

Commonly Used Loss Functions

- Square loss

$$Loss = \left(1 - f(x;\theta)\right)^2$$

- Hinge loss

$$Loss = \max(0,1 - yf(x;\theta))$$

- Logistic loss

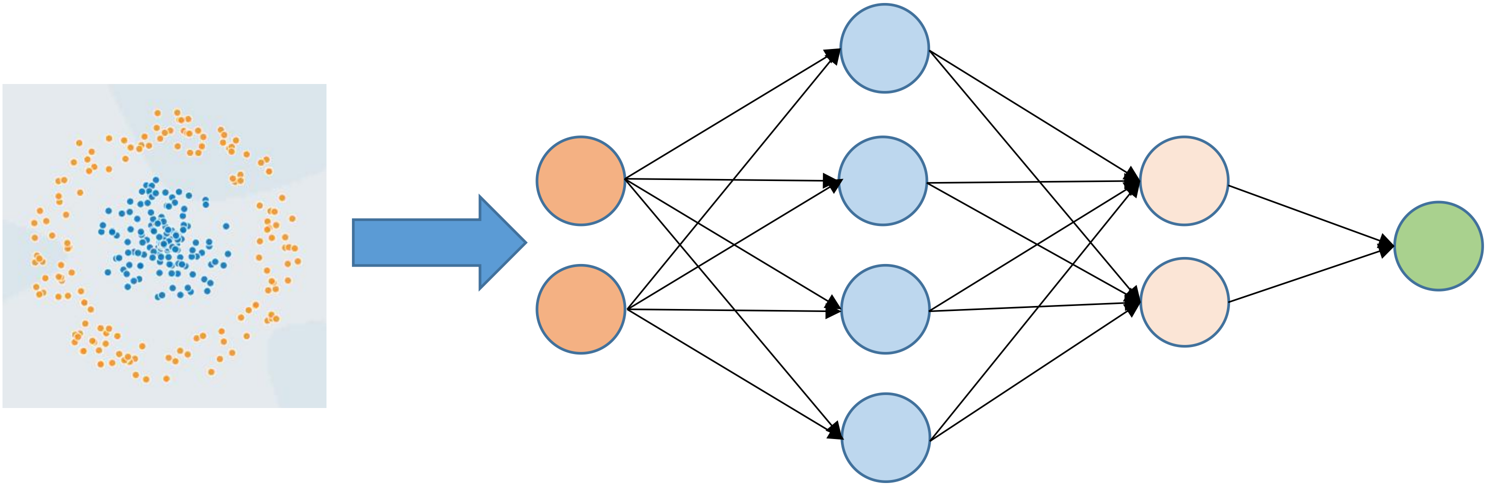
$$Loss =- y\log(f(x;\theta))$$

- Cross entropy loss

$$Loss =- \sum y\log(f(x;\theta))$$

Demonstration

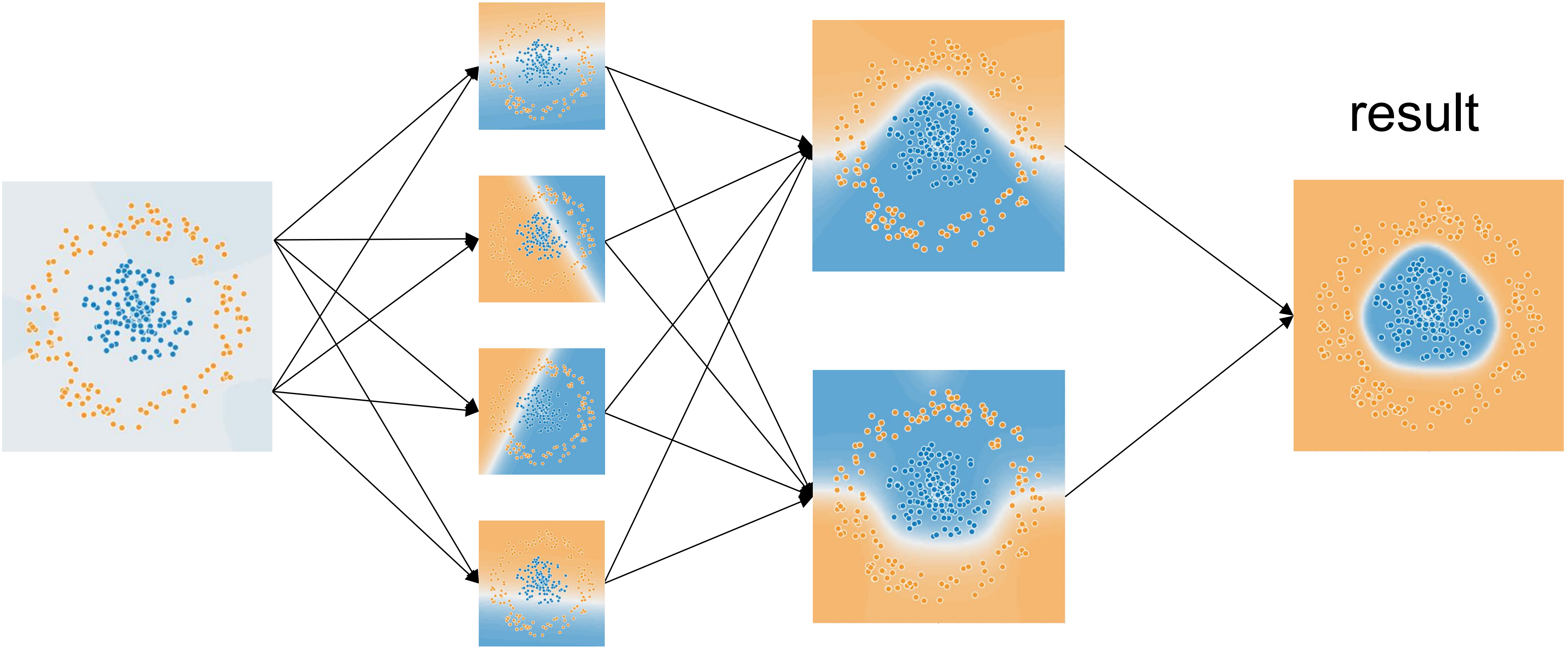
Classification Problem



The input is the coordinates of the points.

Demonstration

Classification Problem: 500 Epoches



An epoch= one forward pass and one backward pass of all the training examples

http://playground.tensorflow.o

Tips

Deeper is Better?

Deeper $\stackrel{?}{=}$ Better performance

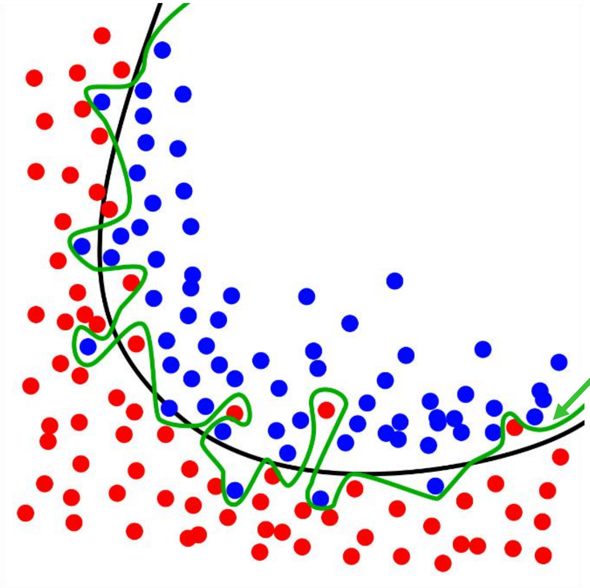
Deeper is Better?

Model	Depth(layers)	Performance(error rate)
AlexNet[Hinton, at. al. 2012]	8	16.4%
GooLeNet[Simonyan, at. al. 2014]	22	6.7%
ResNet[Kaiming He, at. al. 2015]	152	3.57%

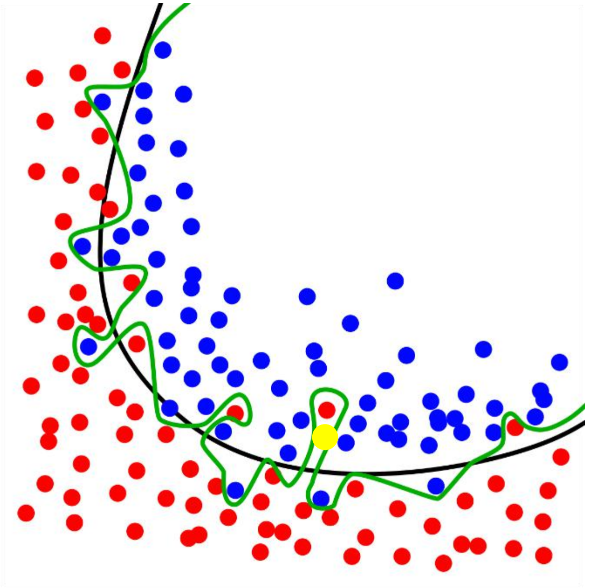
Dataset: ImageNet, which is a benchmark dataset for image classification.

Deep structure can capture complex patterns more efficiently than the shallow one.

Overfitting



The generalization performance of this model can be poor.



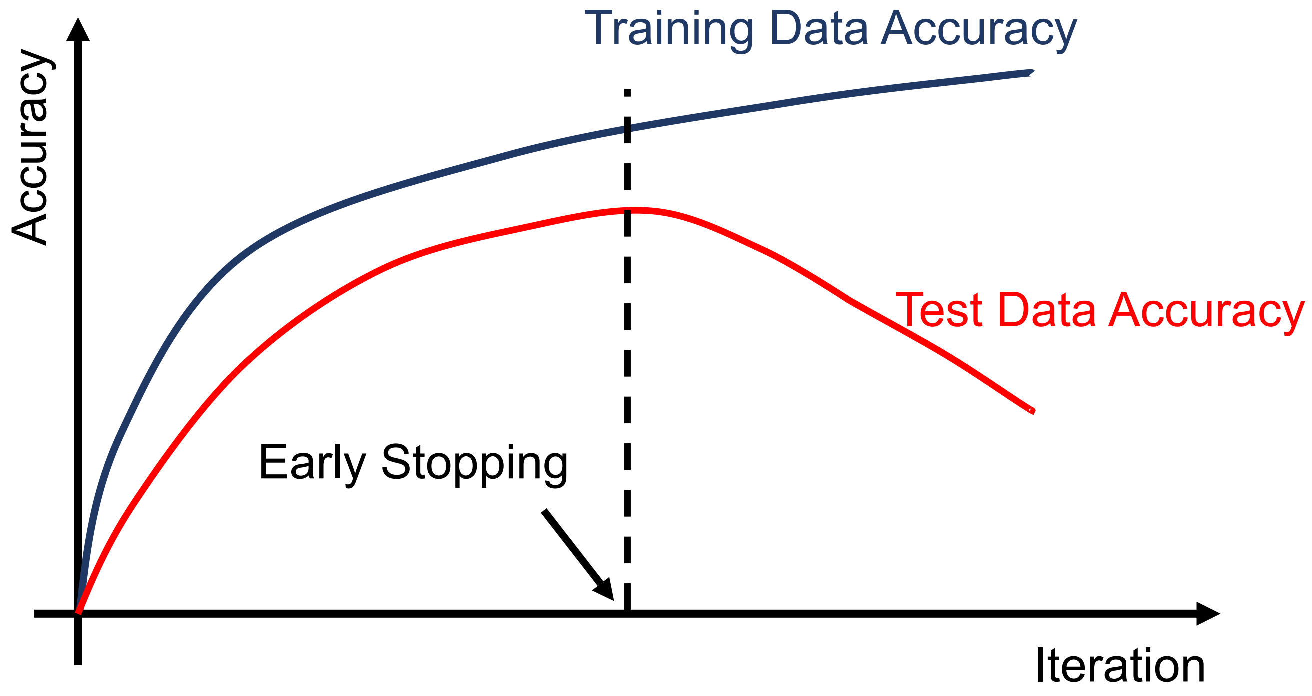
Which one is better？

The predicted label is red!

A good model is the one that generalizes well on the unseen data.

Preventing Overfitting in DNN

- Early Stopping
- Regularization
- Dropout
- …



Preventing Overfitting in DNN

- Early Stopping
- Regularization
- Dropout
- …

$$Loss'(\theta) = Loss(\theta) + \lambda||\theta||_p$$

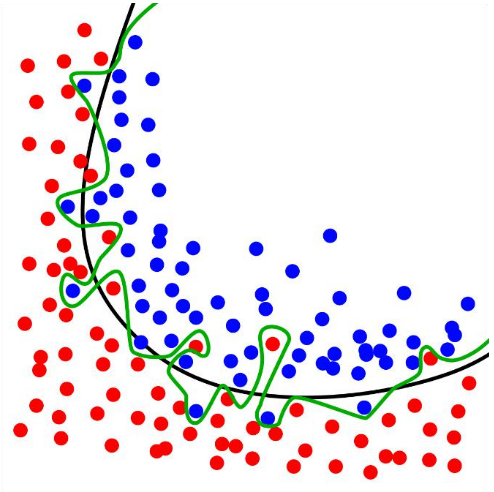
regularization term

➢ ℓ_2 norm

$$||\theta||_2^2 = (\theta_1)^2 + (\theta_2)^2 + \ldots$$

➢ ℓ_1 norm

$$||\theta||_1 = |\theta_1| + |\theta_2| + \ldots$$



Small weights usually imply smooth decision boundary.

L2 Regularization

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

$$||\theta||_2 = (\theta_1)^2 + (\theta_2)^2 + \ldots$$

$$\frac{\partial Loss'}{\partial \theta} = \frac{\partial Loss}{\partial \theta} + \lambda\theta$$

$$\theta^{t+1} := \theta^t - \eta \frac{\partial Loss'}{\partial \theta^t}$$

$$= \theta^t - \eta\left(\frac{\partial Loss}{\partial \theta^t} + \lambda\theta^t\right)$$

$$= (1 - \eta\lambda)\theta^t - \eta \frac{\partial Loss}{\partial \theta^t}$$

L1 Regularization

$$Loss'(\theta) = Loss(\theta) + \lambda||\theta||_1$$

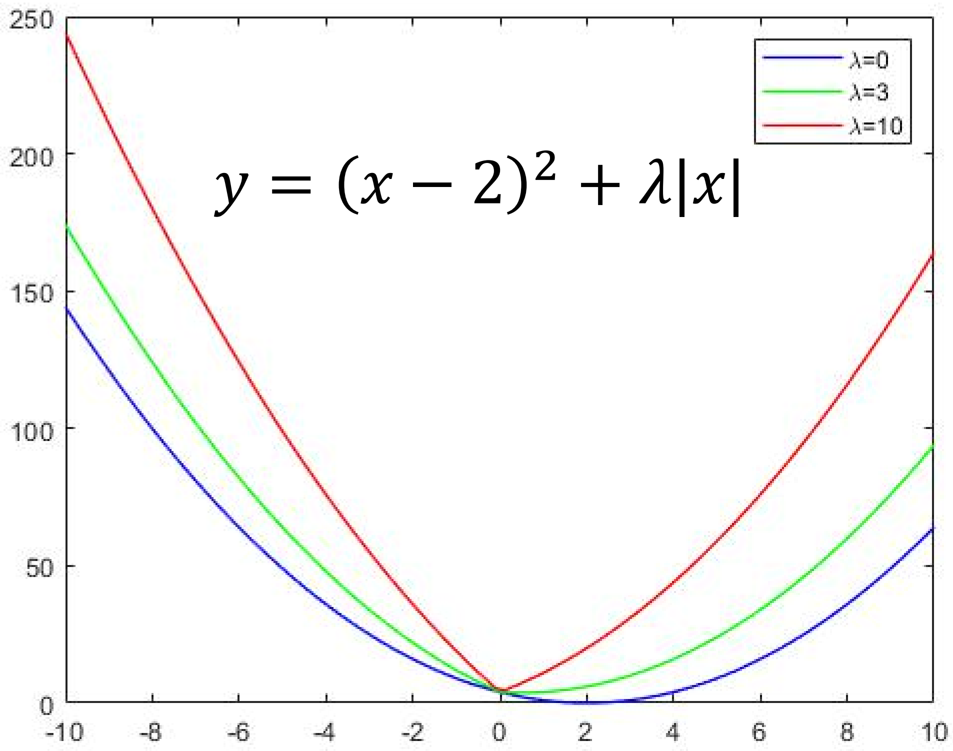
$$||\theta||_1 = |\theta_1| + |\theta_2| + \dots$$

$$\frac{\partial Loss'}{\partial \theta} = \frac{\partial Loss}{\partial \theta} + \lambda * sgn(\theta)$$

$$\theta^{t+1} := \theta^t - \eta \frac{\partial Loss'}{\partial \theta^t}$$

$$= \theta^t - \eta\left(\frac{\partial Loss}{\partial \theta^t} + \lambda sgn(\theta^t)\right)$$

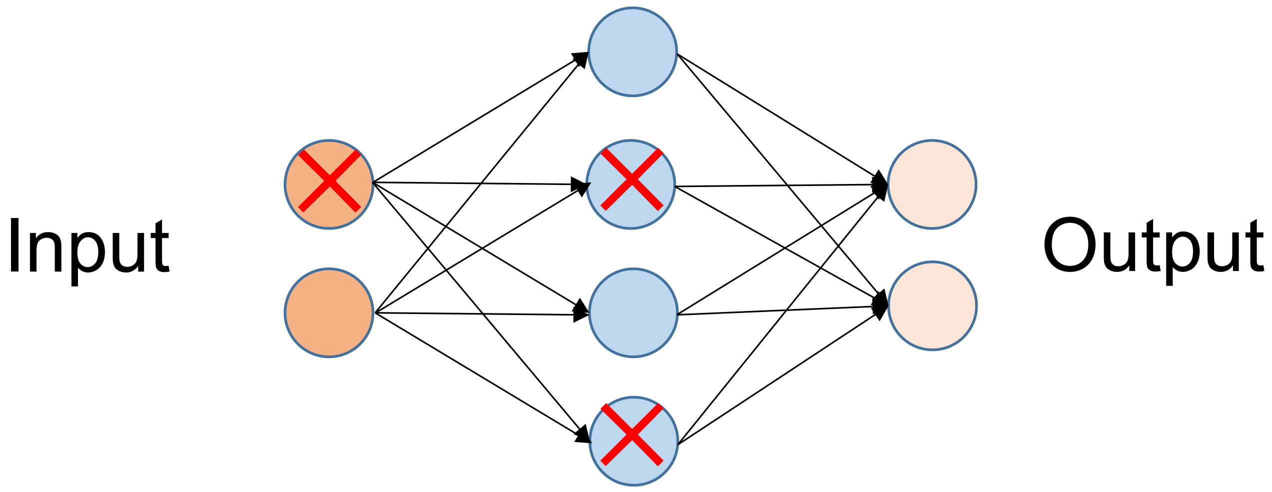
$$= \theta^t - \eta\lambda sgn(\theta^t) - \eta \frac{\partial Loss}{\partial \theta^t}$$



$$y = (x-2)^2 + \lambda|x|$$

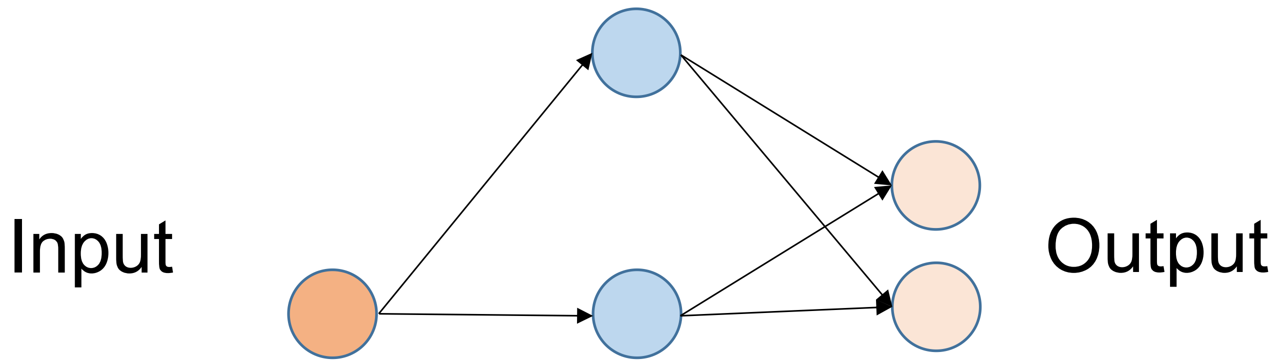
Preventing Overfitting in DNN

- Early Stopping
- Regularization
- Dropout
- …



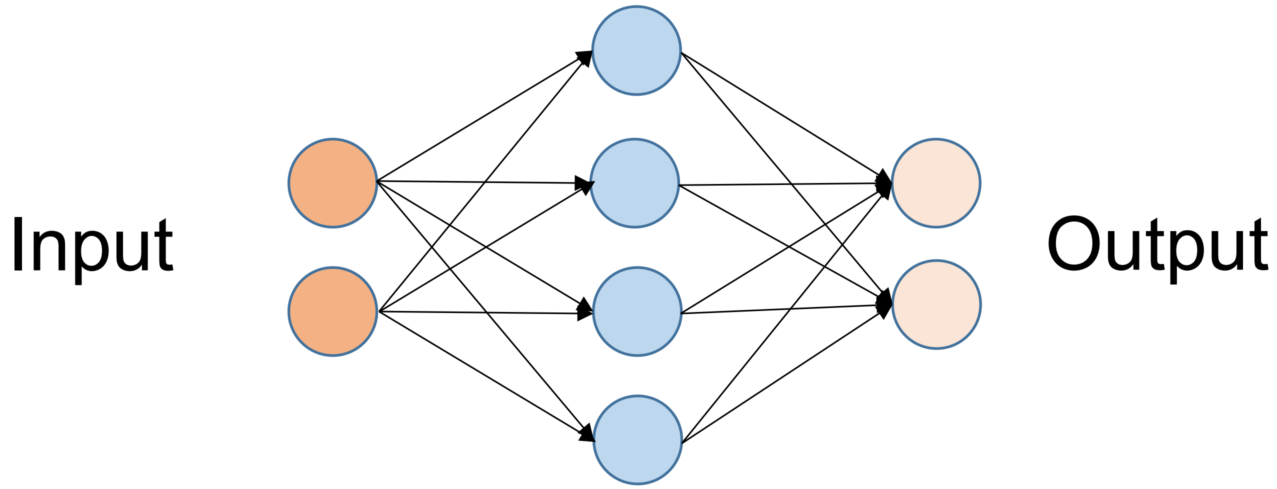
Training: We drop each neuron with probability p

Dropout

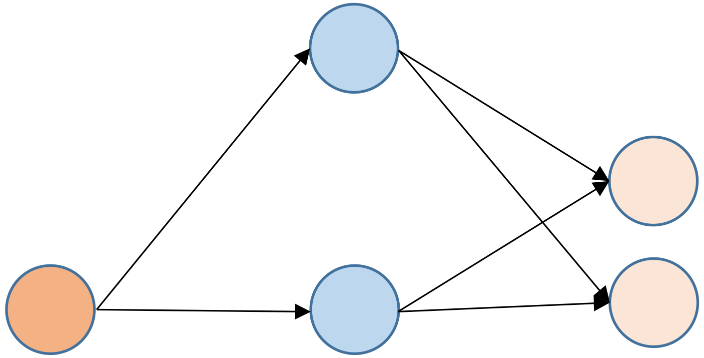


Training: We dropout each neuron with probability p. Then, we train the resulting network for one iteration.

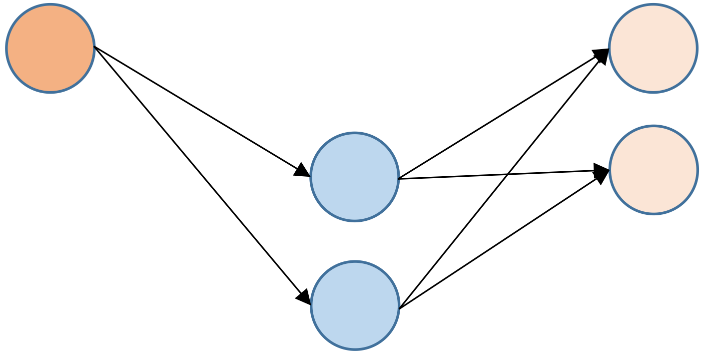
Dropout



In each iteration, we will not update the weights of the links connecting to the dropped neurons.



Iteration 1



Iteration 2

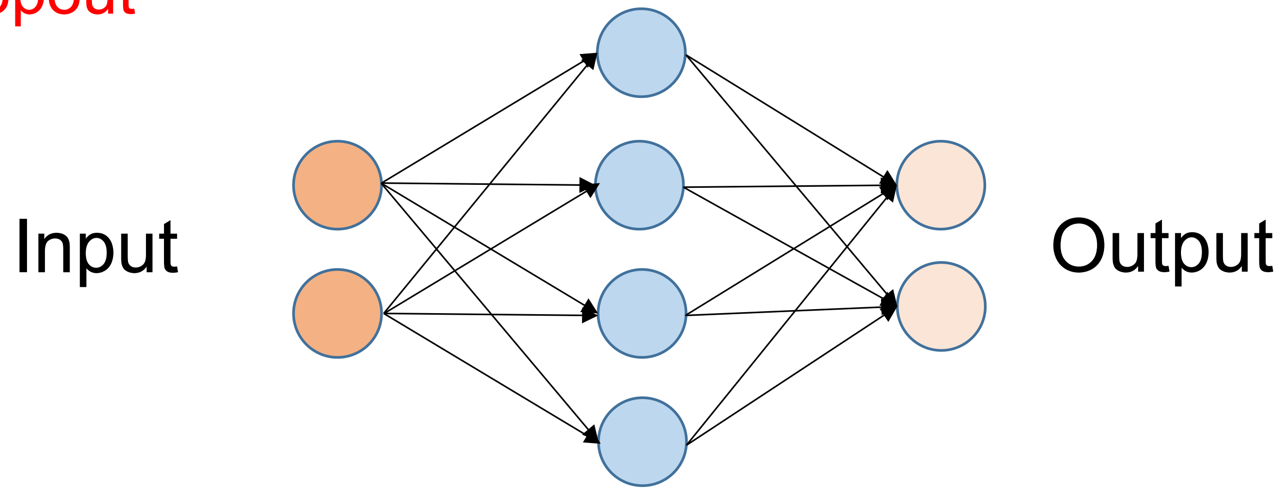
……

……

An iteration = a *batch* of training data passing through the network

Dropout

Testing: No dropout

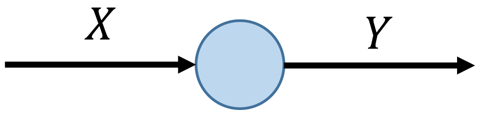


$$W_{test} = (1 - p)W_{train}$$

Why?

Dropout

The dropout rate at training is P



Training: dropout

$$E_{train}(Y|X) = (1 - P)X$$

Testing: no dropout

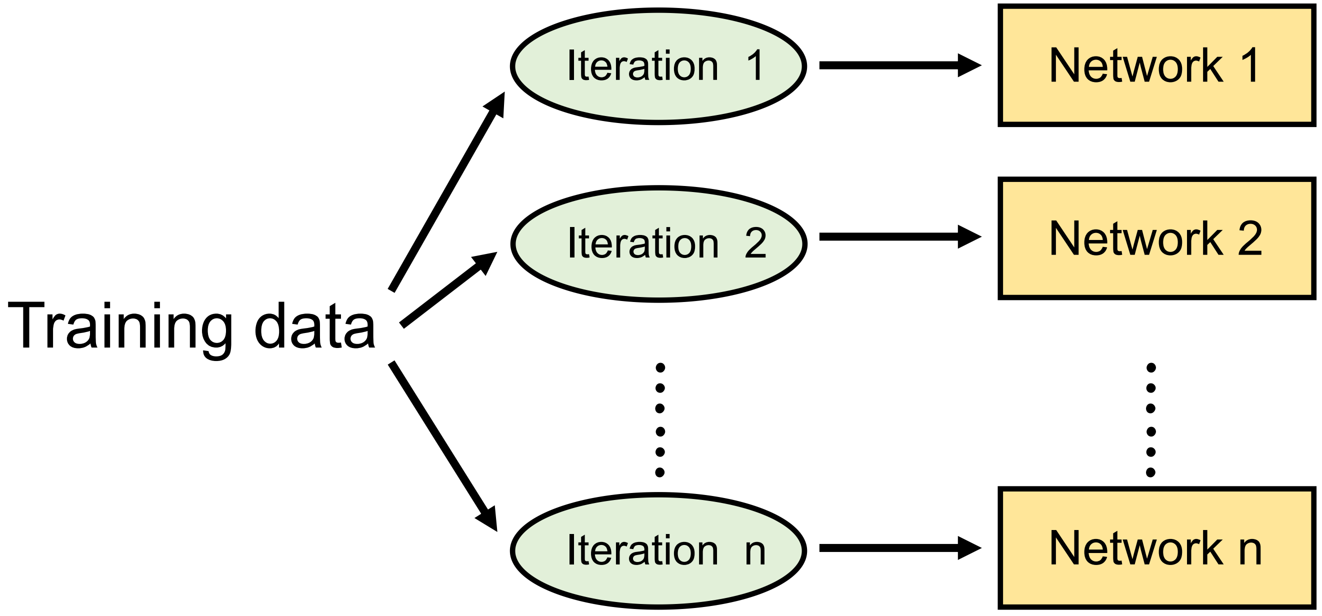
$$E_{test}(Y|X) = X$$

$$E_{test}(Y) \longrightarrow E_{train}(Y)$$

$$W_{test} = (1 - P)W_{train}$$

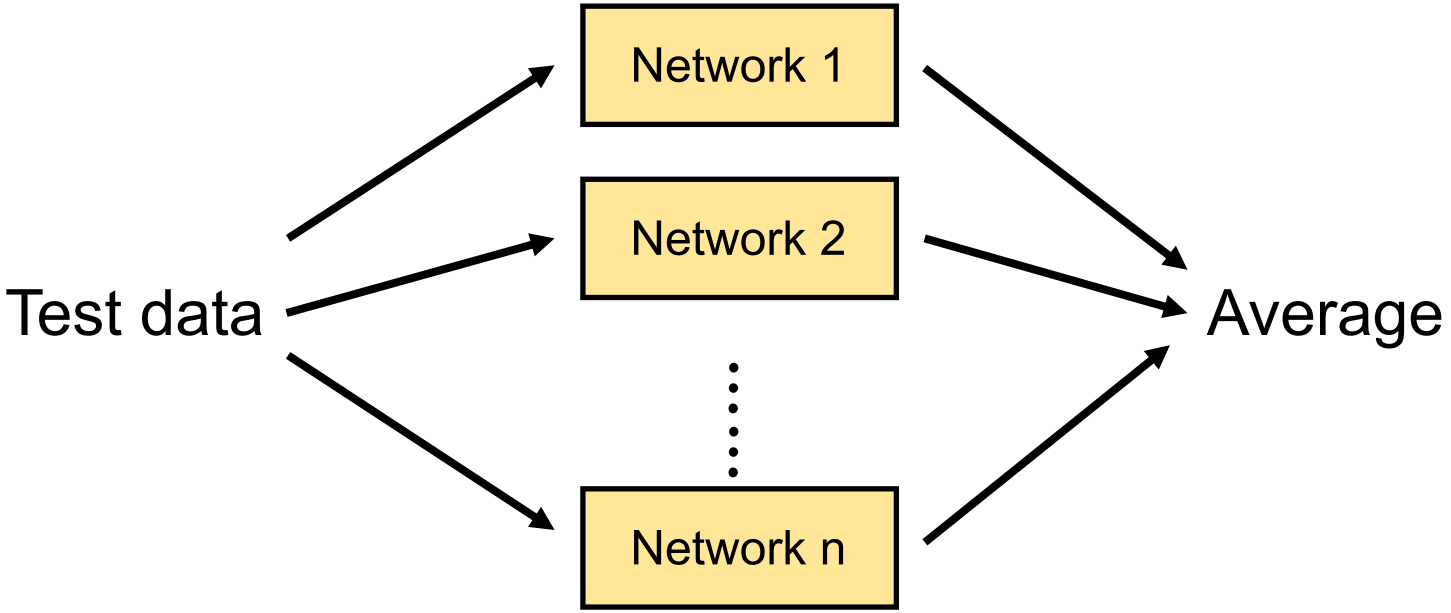
Why Dropout

Dropout is a kind of ensemble



Why Dropout

Dropout is a kind of ensemble



With N neurons, there are 2^N possible sub-networks.

- The average can relieve overfitting
- Dropout can learn more robust patterns

http://deeplearning.cs.cmu.edu/slides/lec6.stochastic_gradient.pdf

Design Deep model

http://www.sohu.com/a/100823811_114877

Questions