

Introduction to Machine Learning

Lecture 14: Neural Networks

Nov 30, 2023

Jie Wang

Machine Intelligence Research and Applications Lab

Department of Electronic Engineering and Information Science (EEIS)

<http://staff.ustc.edu.cn/~jwangx/>

jiawangx@ustc.edu.cn



Machine Intelligence Research and Applications Lab



Contents

- **Introduction**
- **Multi-Layer Perception**
- **Tips**

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

Microsoft | The AI Blog The Official Microsoft Blog Microsoft On the Issues Transform

Microsoft reaches a historic milestone, using AI to match human performance in translating news from Chinese to English

March 14, 2018 | [Allison Linn](#)



Google

Translate

Turn off instant translation



English Spanish French English - detected



English Spanish Chinese (Simplified)

Translate

Deep feedforward networks, also called feedforward neural networks, or multilayer perceptrons (MLPs), are the quintessential deep learning models.

深度前馈网络，也称为前馈神经网络，或多层感知器（MLP），是典型的深度学习模型。



143/5000



Suggest an edit

Shēndù qián kuī wǎngluò, yě chēng wéi qián kuī shénjīng wǎngluò, huò duō céng gǎnzī qì (MLP), shì diǎnxíng de shēndù xuéxí móxíng.

Breakthroughs by Deep Learning

Speech recognition

Microsoft AI Beats Humans at Speech Recognition

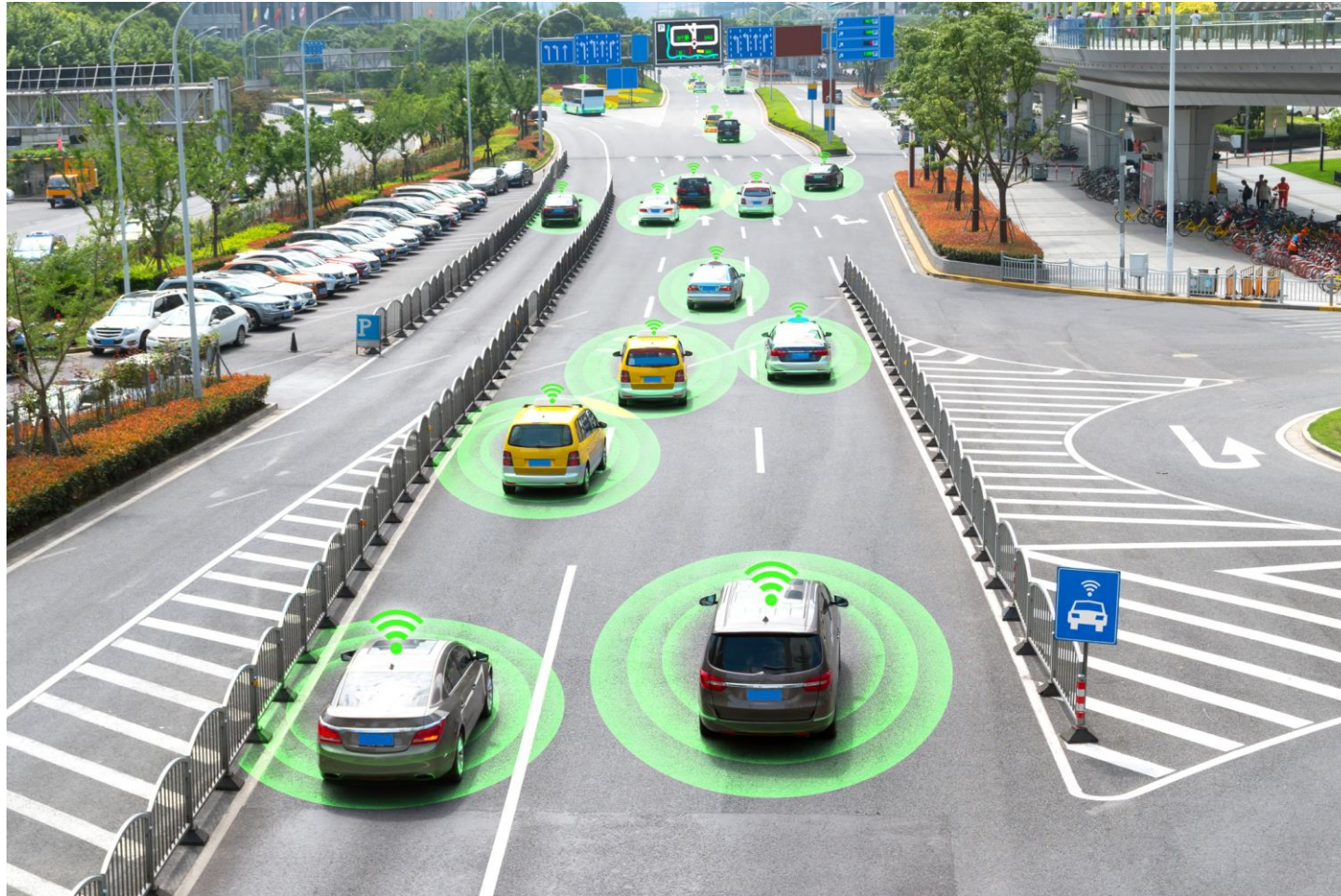
By Richard Adhikari
Oct 20, 2016 11:40 AM PT

 Print
 Email



Breakthroughs by Deep Learning

Self-driving

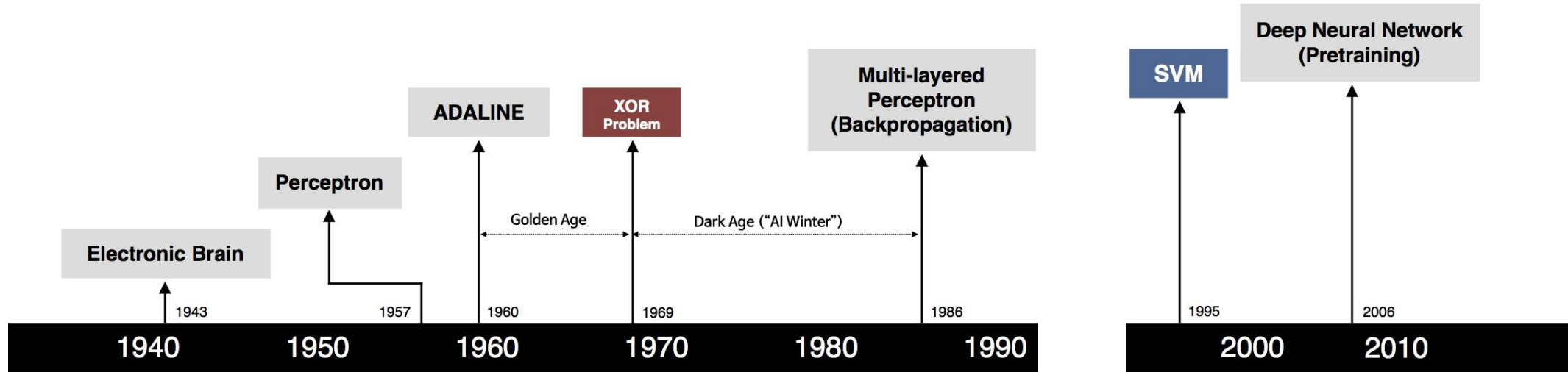


Breakthroughs by Deep Learning

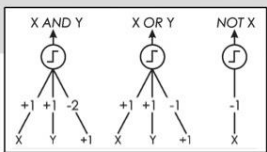
Machine reading comprehension

SQuAD		Home	
<h3>SQuAD1.1 Leaderboard</h3>			
<p>Since the release of SQuAD1.0, the community has made rapid progress, with the best models now rivaling human performance on the task. Here are the ExactMatch (EM) and F1 scores evaluated on the test set of v1.1.</p>			
Rank	Model	EM	F1
	Human Performance Stanford University (Rajpurkar et al. '16)	82.304	91.221
1 Sep 09, 2018	nlnet (ensemble) Microsoft Research Asia	85.356	91.202
2 Jul 11, 2018	QANet (ensemble) Google Brain & CMU	84.454	90.490
3 Jul 08, 2018	r-net (ensemble) Microsoft Research Asia	84.003	90.147

Milestones of Deep Learning



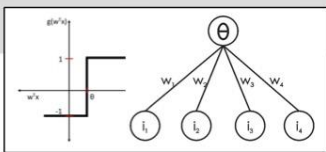
S. McCulloch - W. Pitts



- Adjustable Weights
- Weights are not Learned



F. Rosenblatt



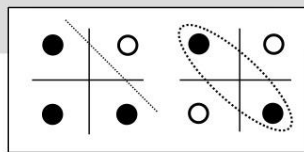
- Learnable Weights and Threshold



B. Widrow - M. Hoff



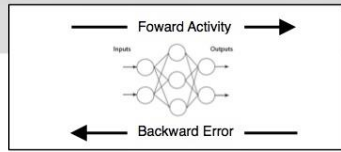
M. Minsky - S. Papert



- XOR Problem



D. Rumelhart - G. Hinton - R. Williams



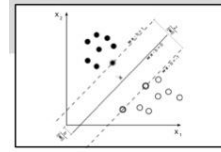
- Solution to nonlinearly separable problems
- Big computation, local optima and overfitting



Y. Lecun



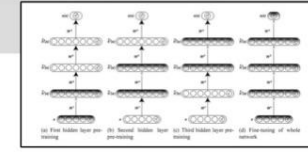
V. Vapnik - C. Cortes



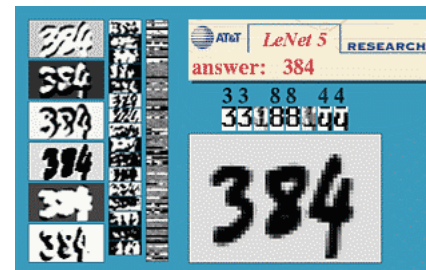
- Limitations of learning prior knowledge
- Kernel function: Human Intervention



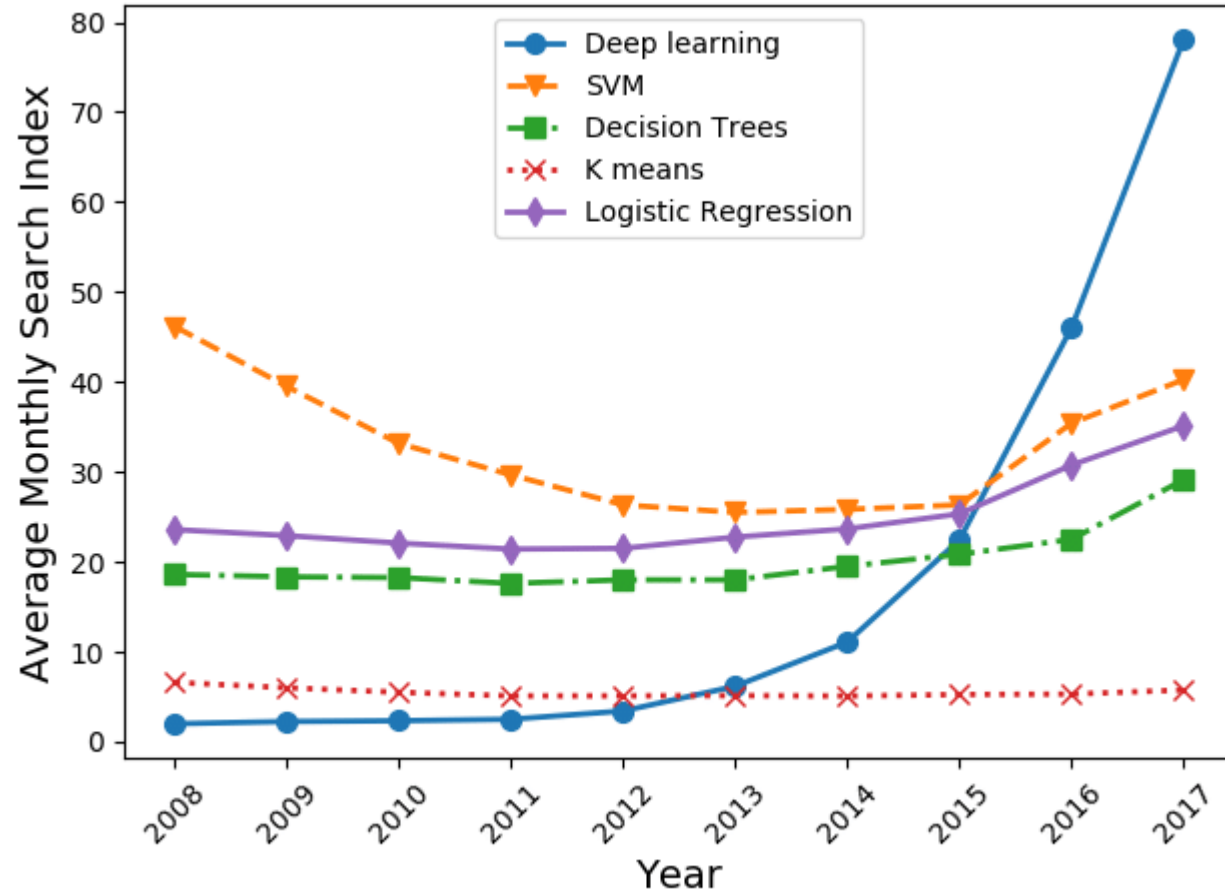
G. Hinton - S. Ruslan



- Hierarchical feature Learning



Google Trend of Deep Learning



Motivation of Neural Networks

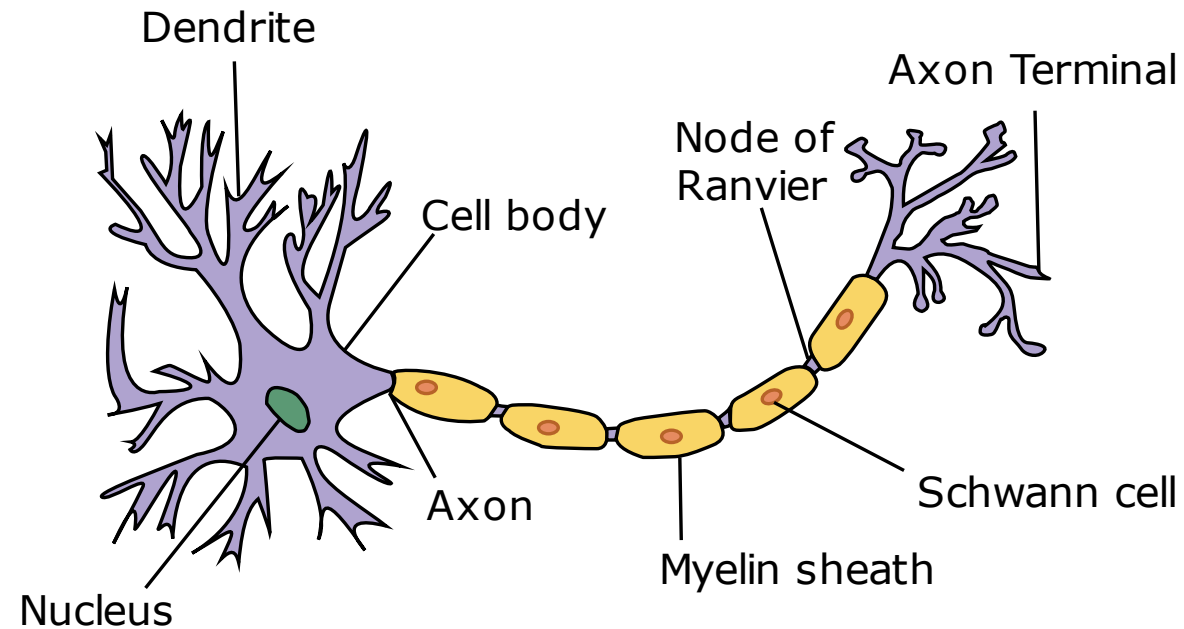
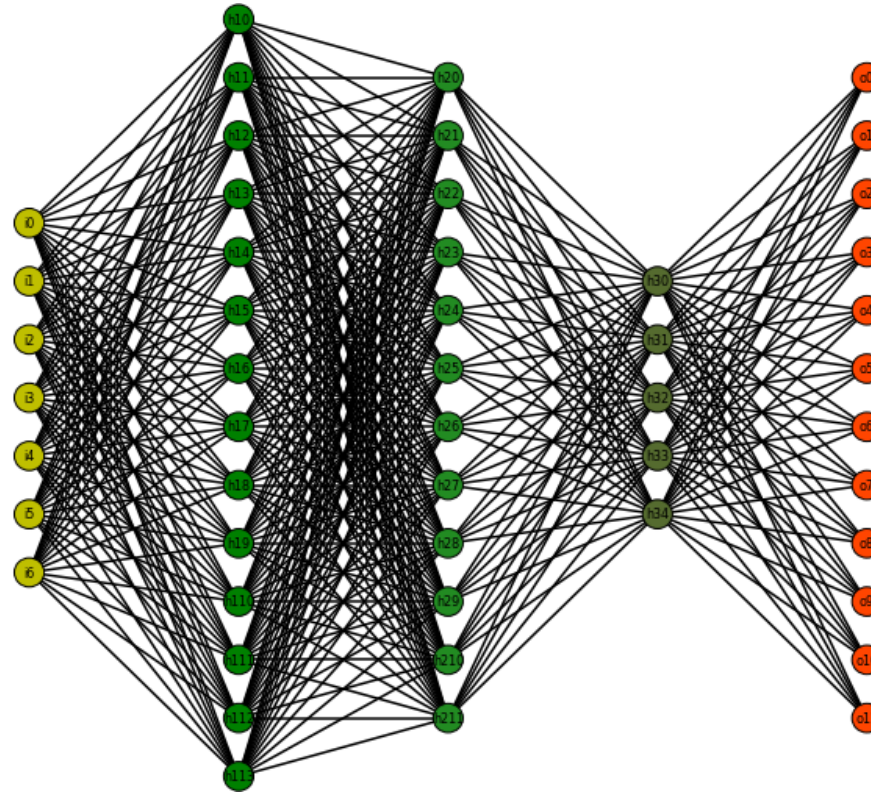
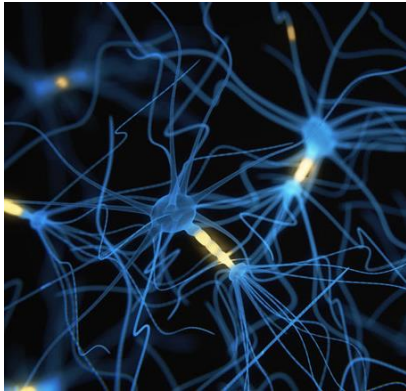


Diagram of neuron

Motivation of Neural Networks



What is Neural Network?



Biological Neural Network

Artificial Neural Network

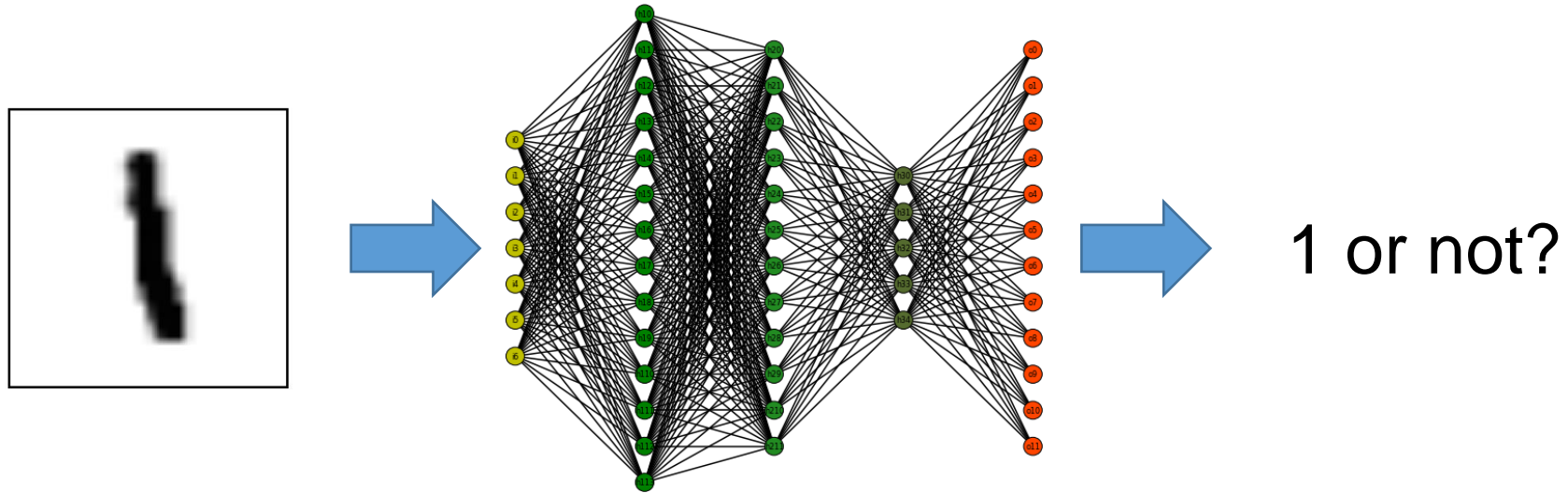
Multi-Layer Perceptron

Hand-written Digits Recognition

The MNIST dataset

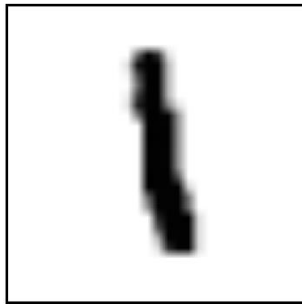


Hand-written Digits Recognition



Vector representation

x : image



28 × 28 pixels

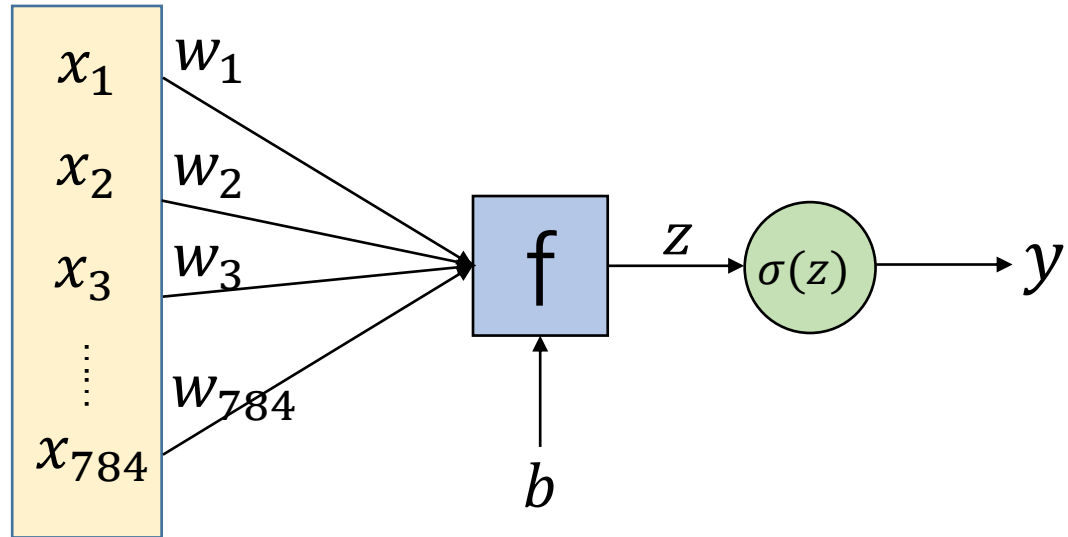


$$\begin{bmatrix} 0 \\ 1 \\ \dots \\ \dots \\ 0 \end{bmatrix} \in R^{784}$$

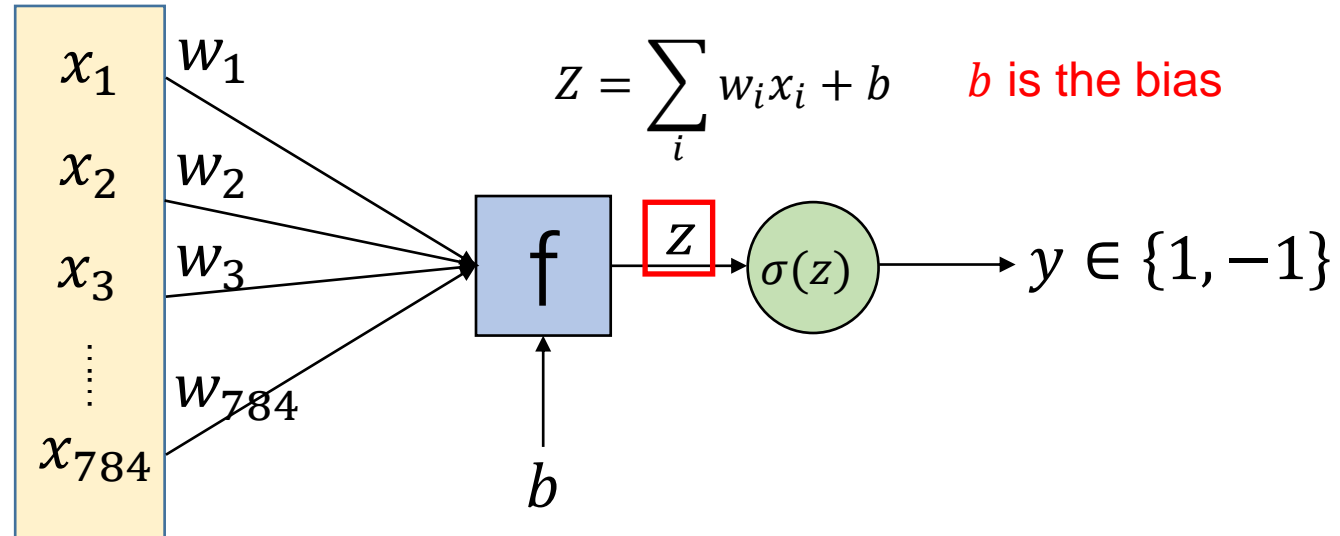
1: for ink
0: otherwise

Input domain

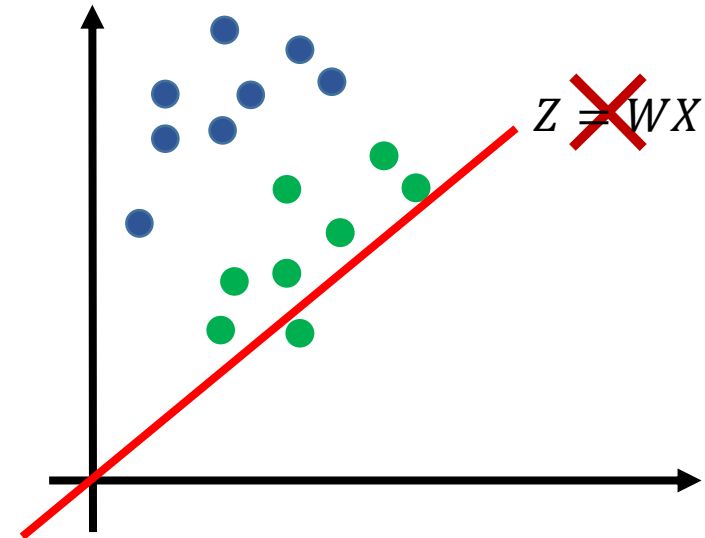
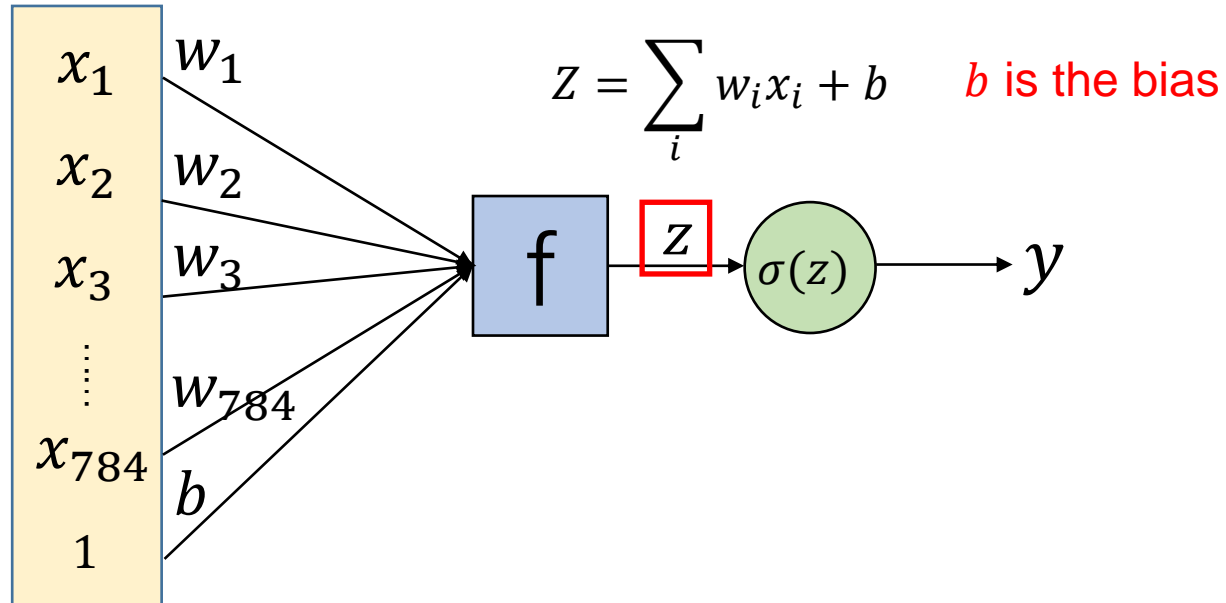
Single Neuron



Single Neuron

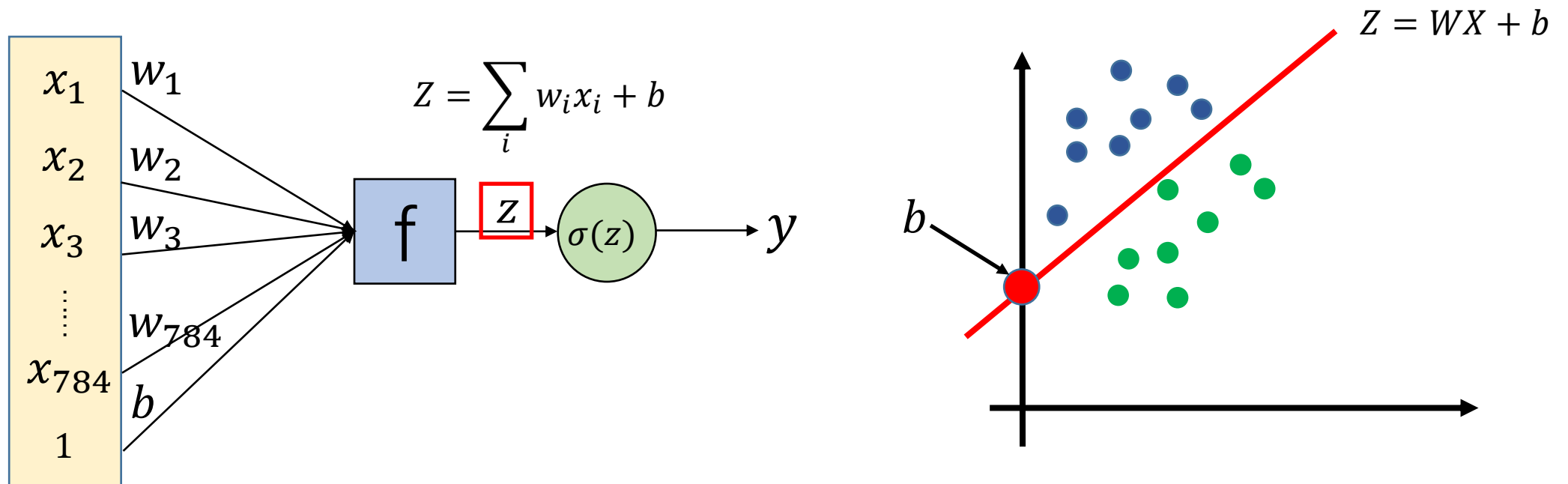


Single Neuron



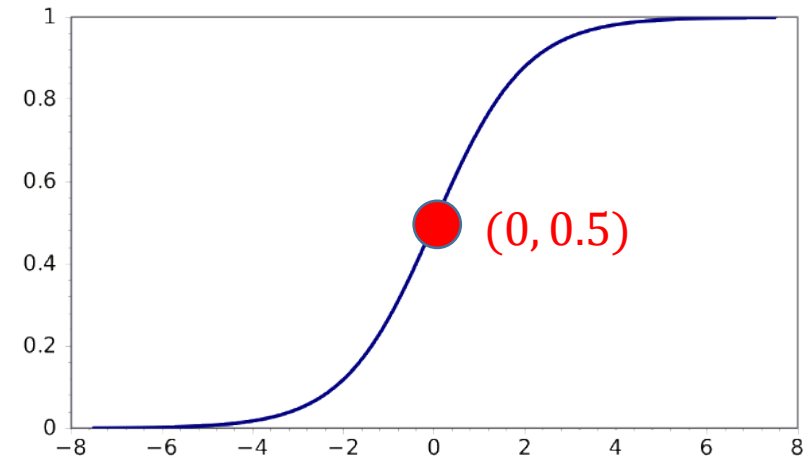
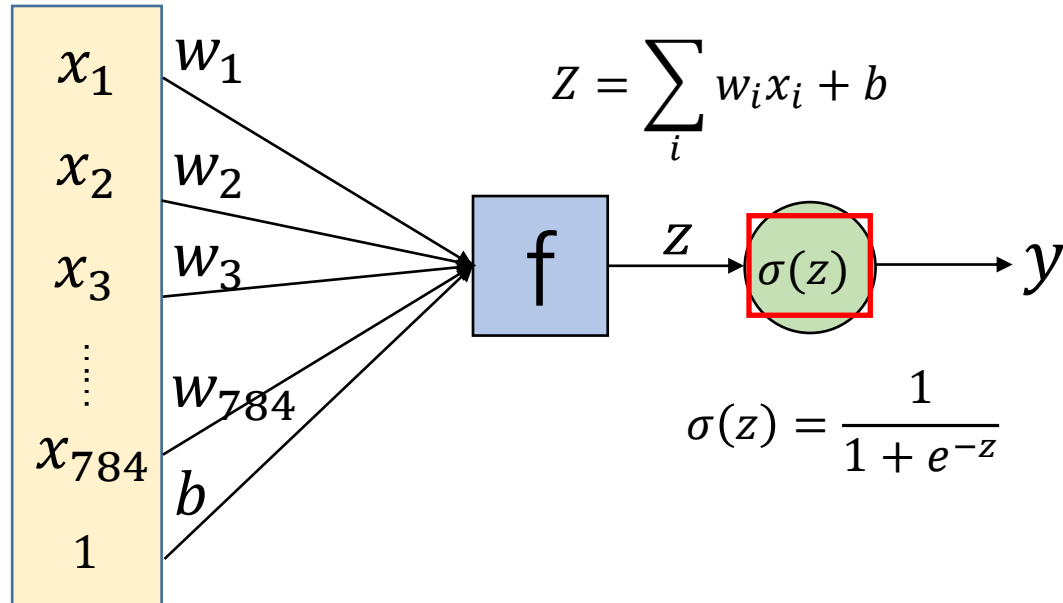
Why do we need a bias b ?

Single Neuron



Why do we need a bias b ?

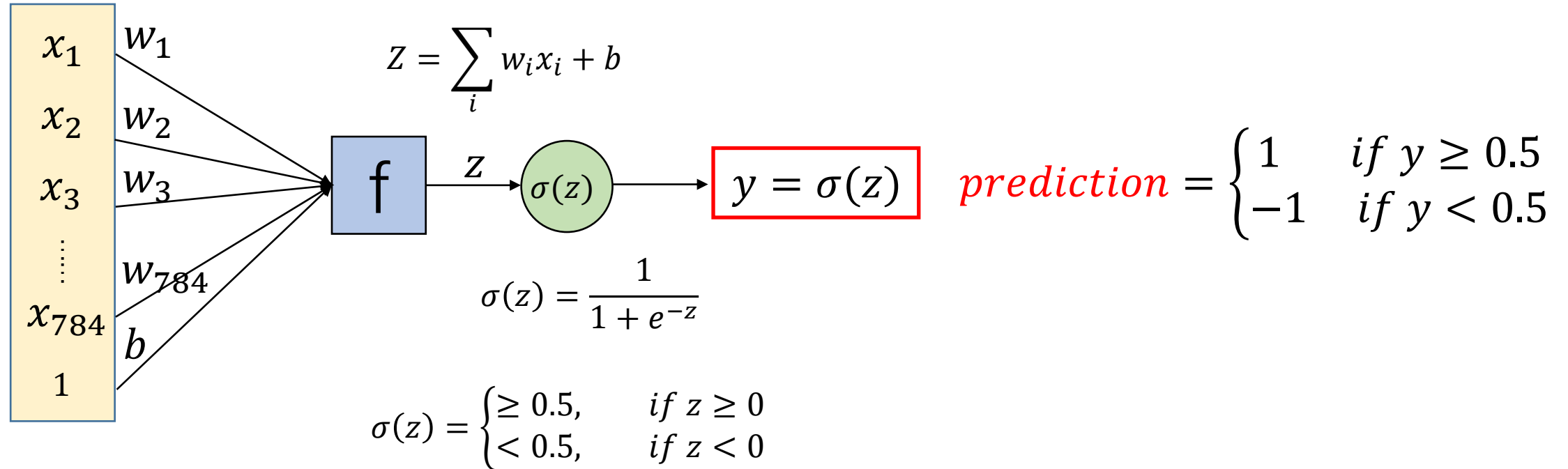
Single Neuron



Sigmoid Function

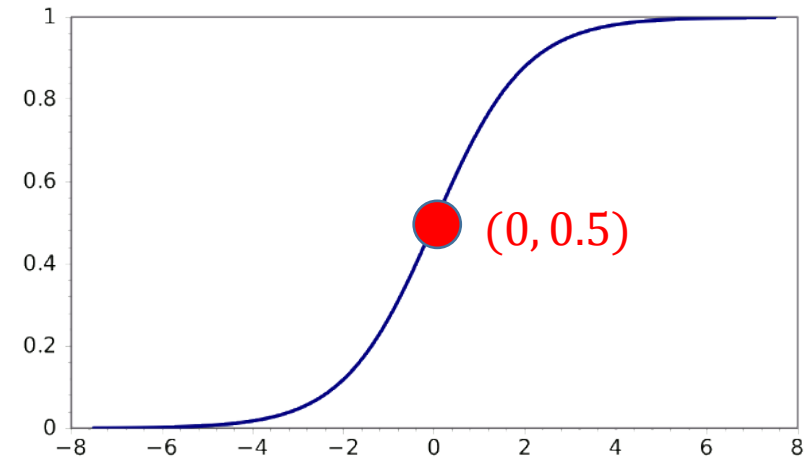
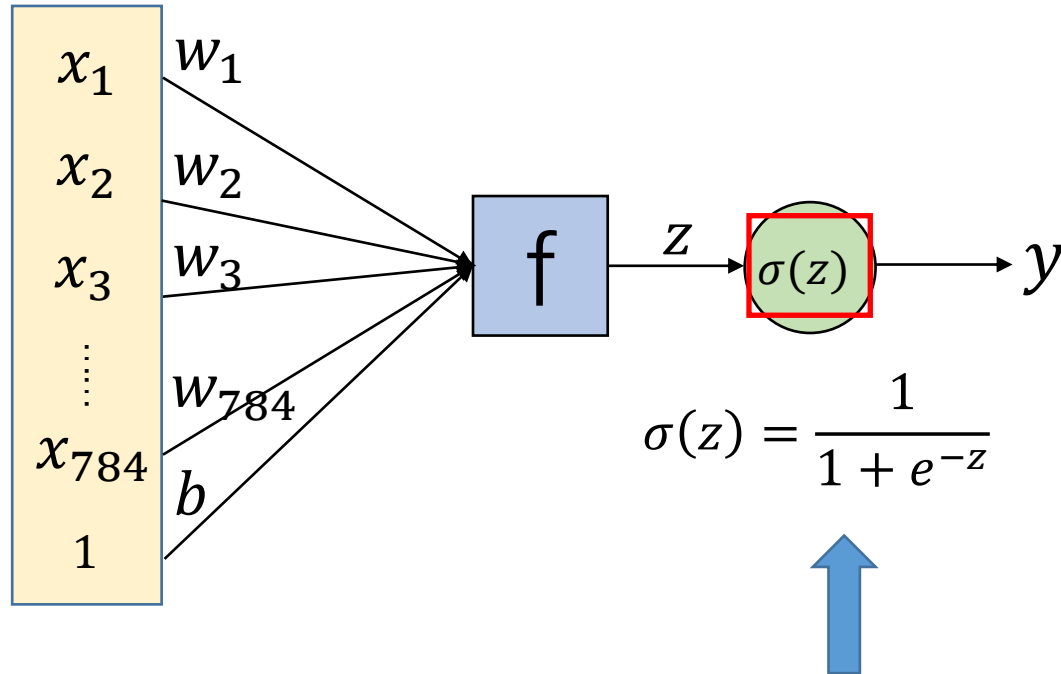
$$\sigma(z) = \begin{cases} \geq 0.5, & \text{if } z \geq 0 \\ < 0.5, & \text{if } z < 0 \end{cases}$$

Single Neuron



This is a **linear** classifier.

Activation Function

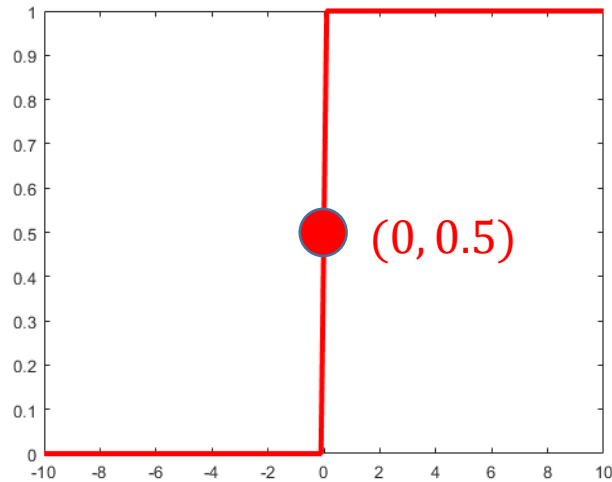


Activation function: The function that acts on the weighted combination of inputs.

We also have other activation function.

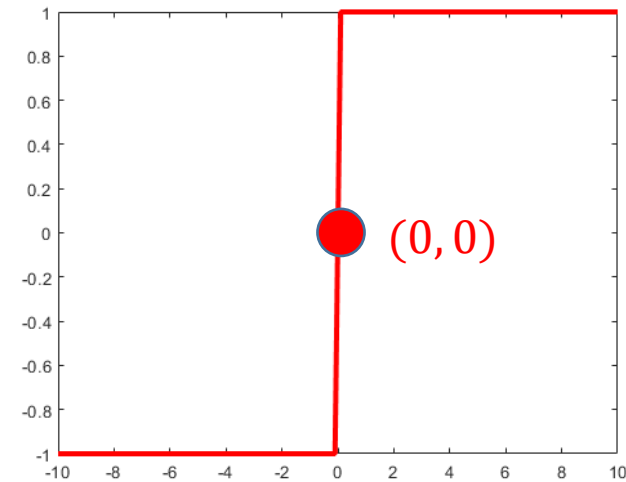
Activation Function

Boolean



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

Unit step function

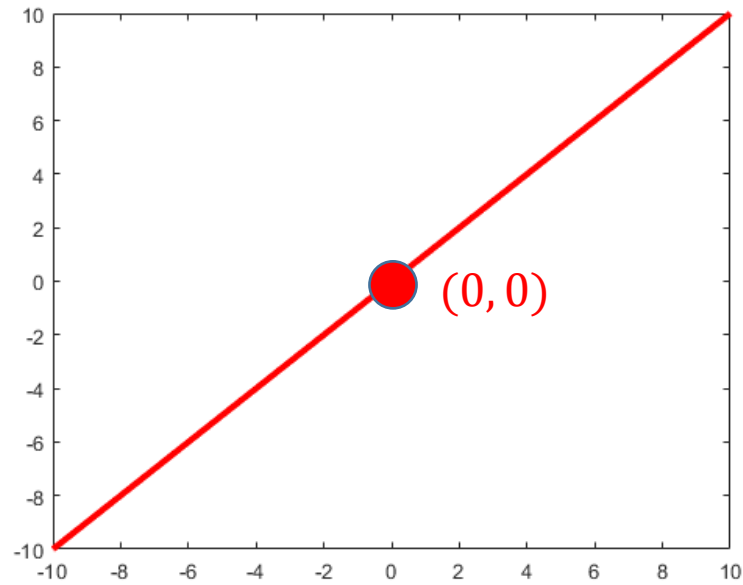


$$\sigma(z) = \begin{cases} 1 & z > 0 \\ 0 & z = 0 \\ -1 & z < 0 \end{cases}$$

Sign function

Activation Function

Linear

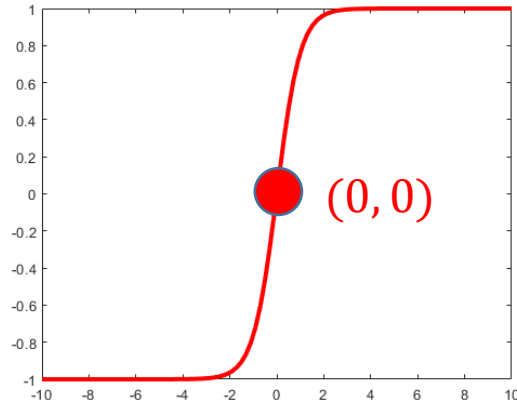


$$\sigma(z) = z$$

Linear function

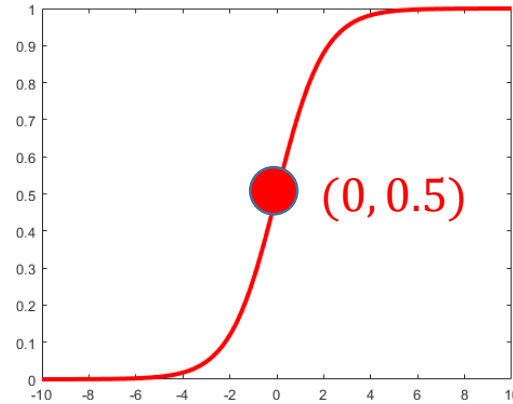
Activation Function

Non-linear



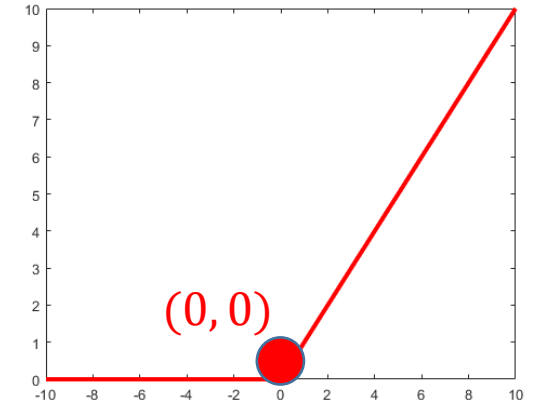
$$\sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

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.

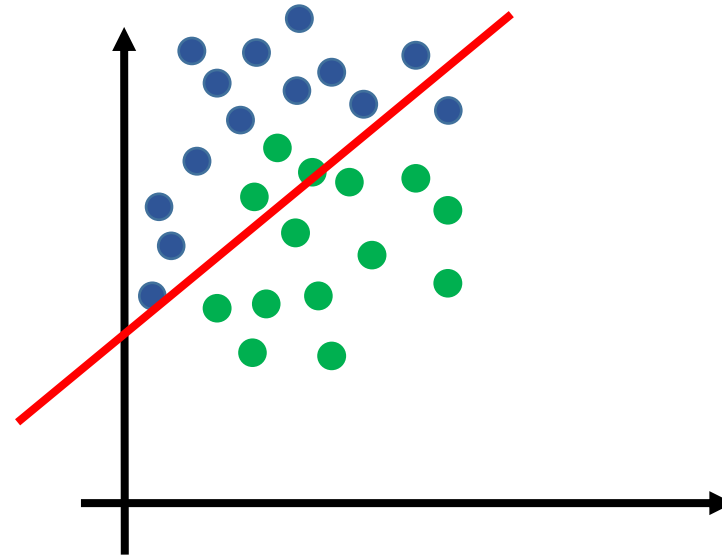
Why?

Why Non-Linearity?

Without non-linearity

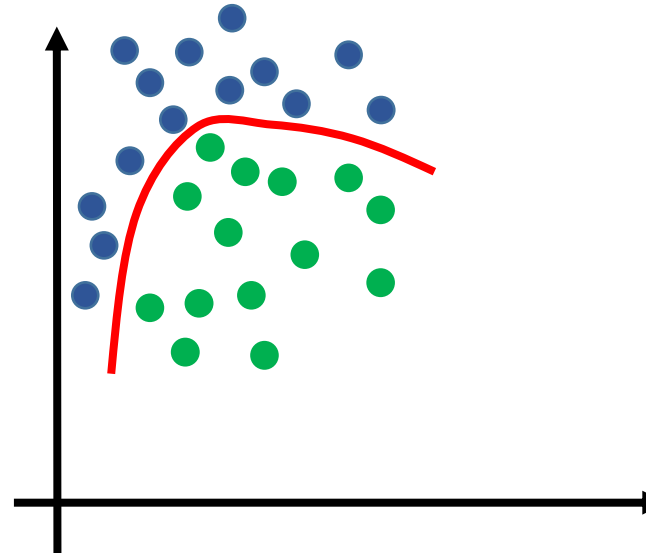
- Deep neural networks are equivalent to linear transforms.

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

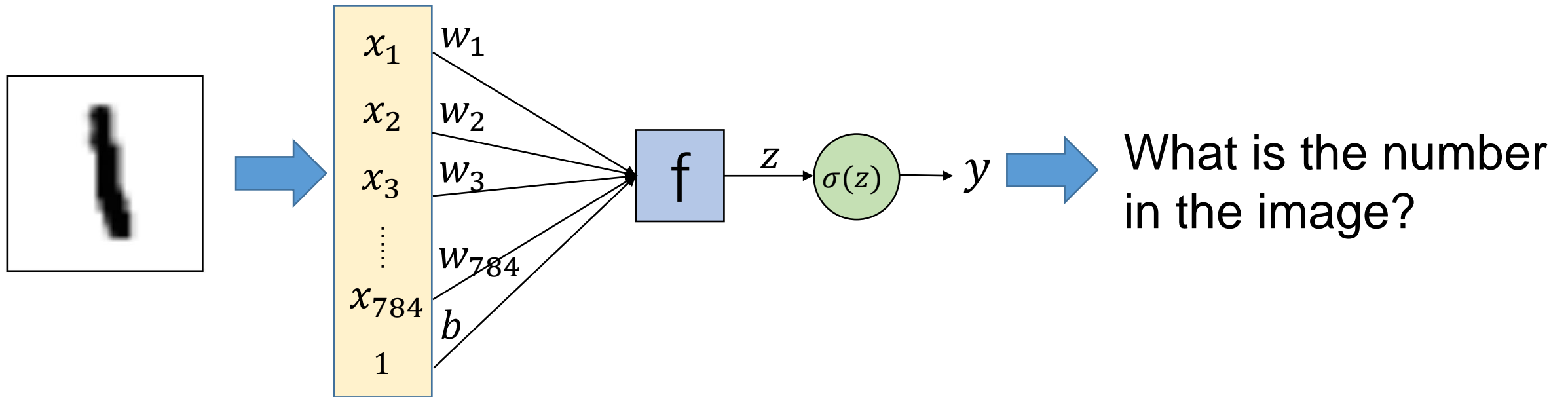


With non-linearity

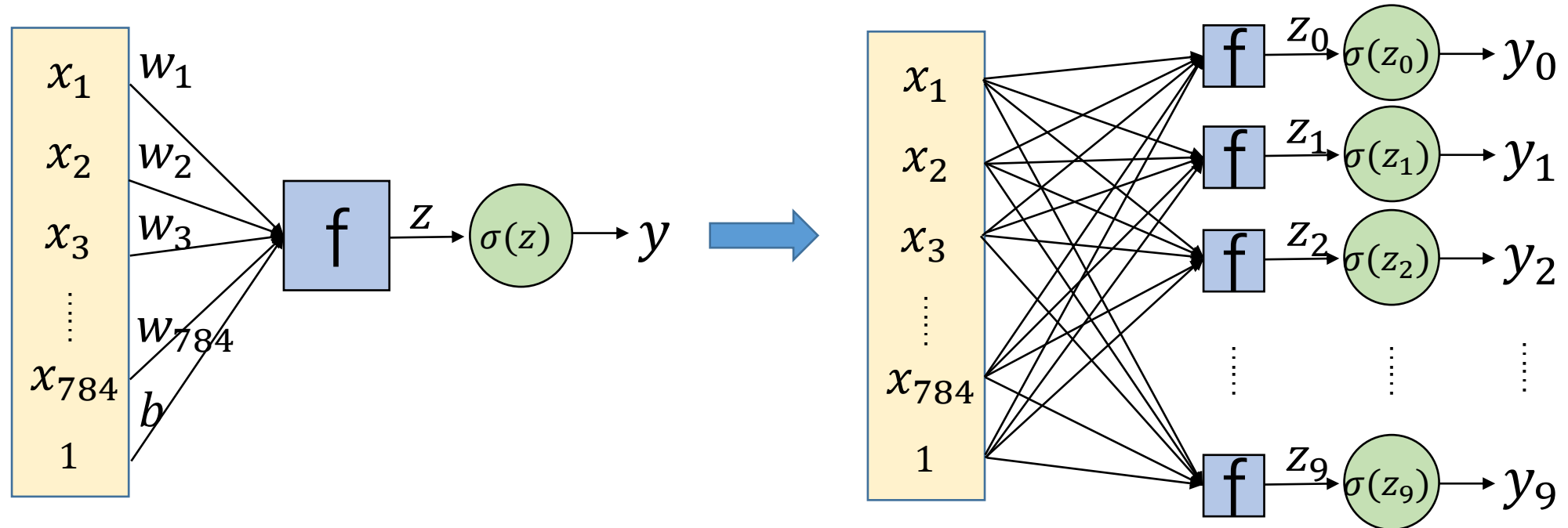
- The neural networks can approximate complicated functions.



A More Complicated Task



Multiple Outputs

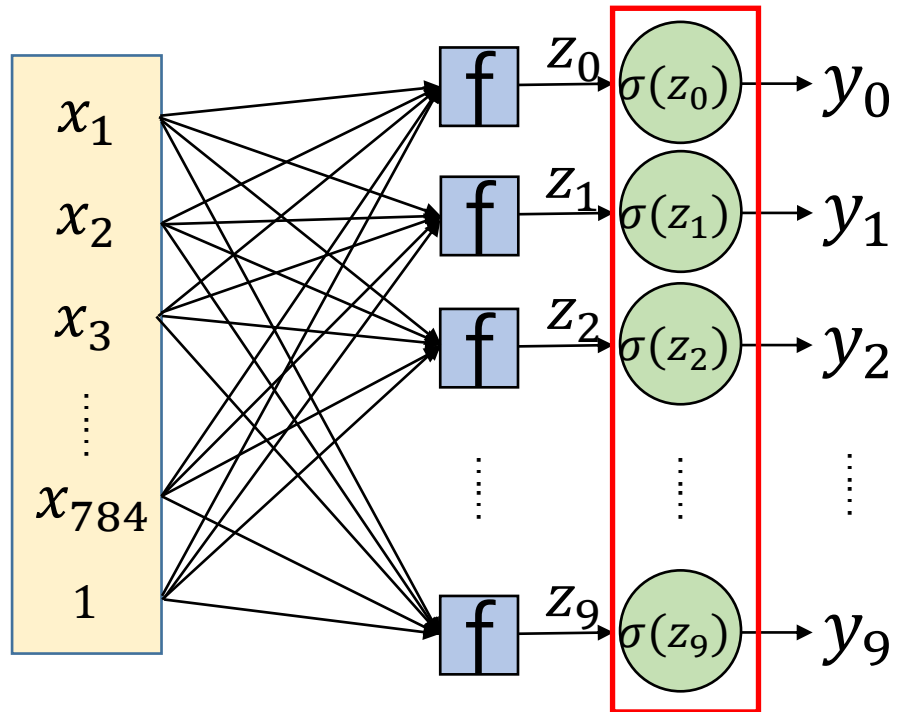


$$W \in R^{784 \times 10}$$

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

Multiple Outputs

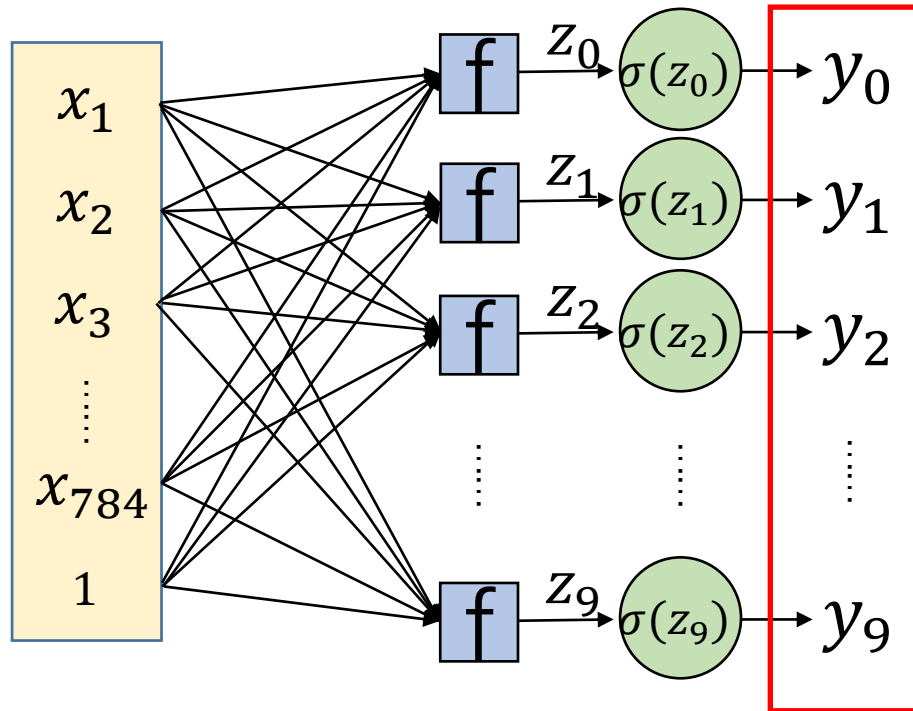
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



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

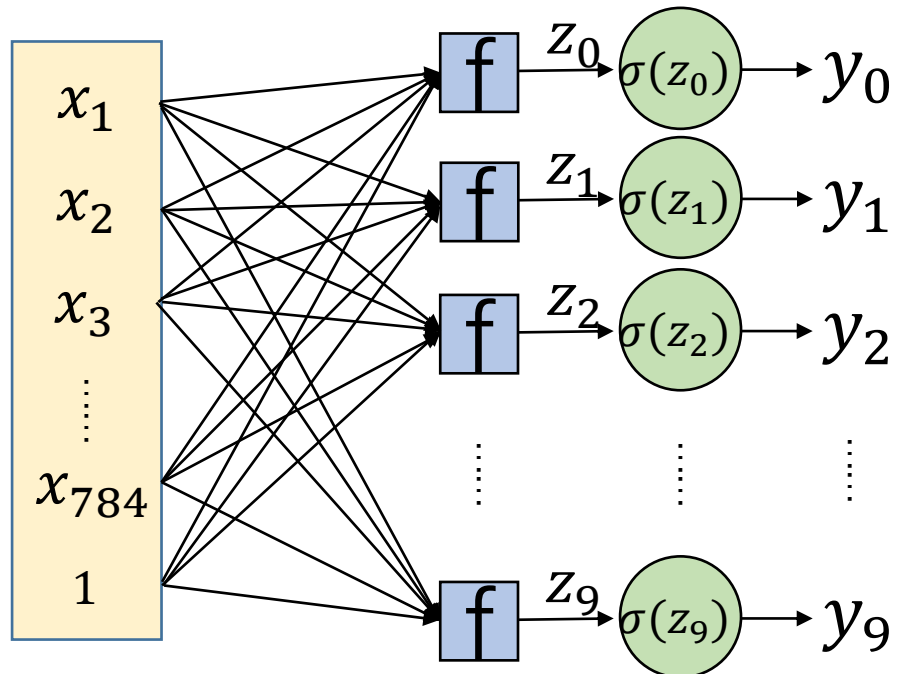
$$b \in R^{1 \times 10}$$

Multiple Outputs



We choose label corresponding to the maximum value of y_i .

Multiple Outputs



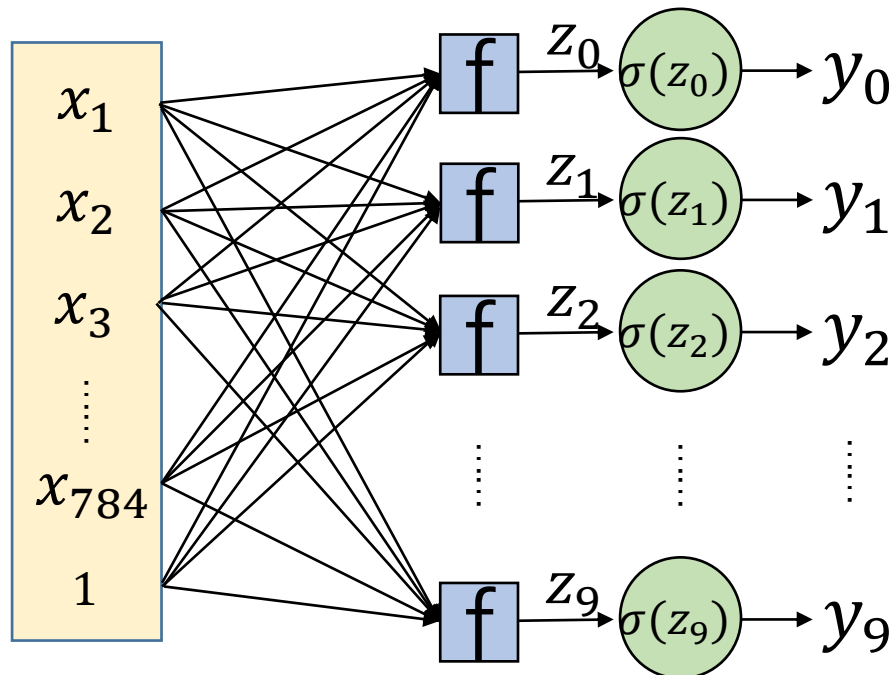
$$W \in R^{784 \times 10}$$

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

Question:

How do we evaluate the performance of the model?

Loss Function



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

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

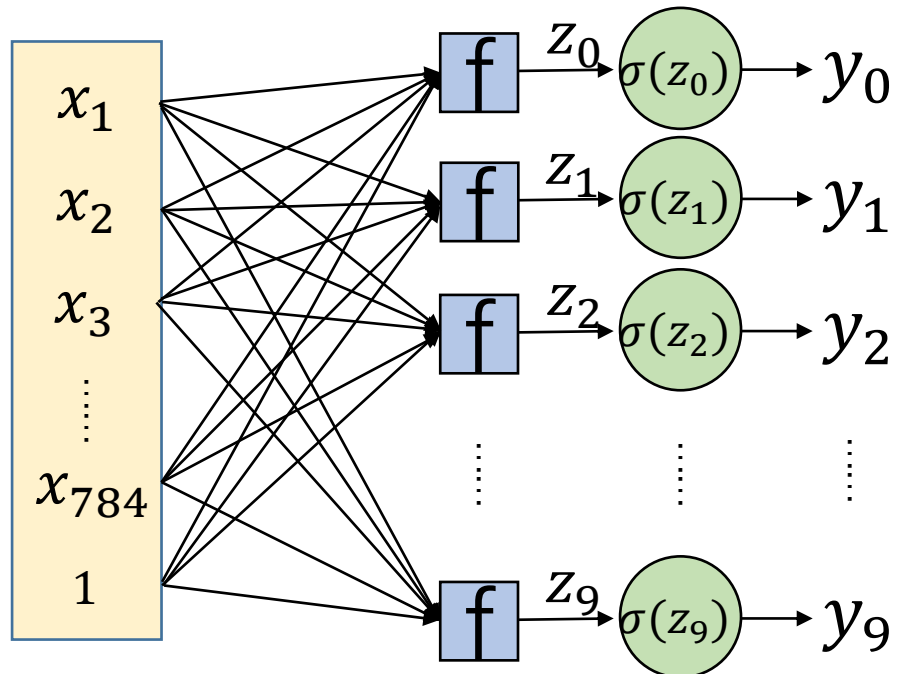
Ground truth: $Q = \begin{bmatrix} 0 \\ 1 \\ \dots \\ \dots \\ 0 \end{bmatrix} \in R^{10}$ One hot vector
The component corresponding to the true label is "1".

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

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

The goal is to minimize the loss!

Model Parameters



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

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

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

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



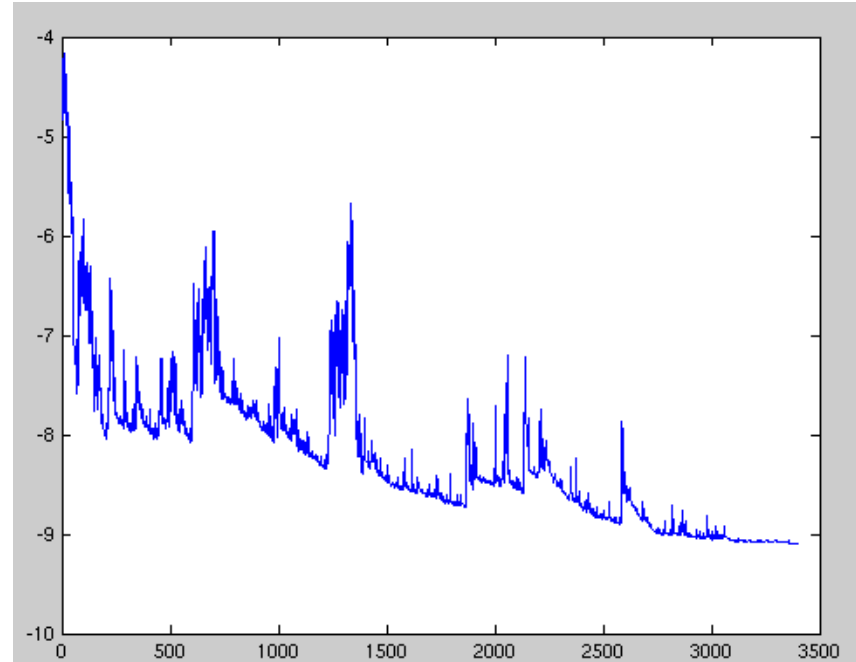
Backpropagation

Stochastic Gradient Descent

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

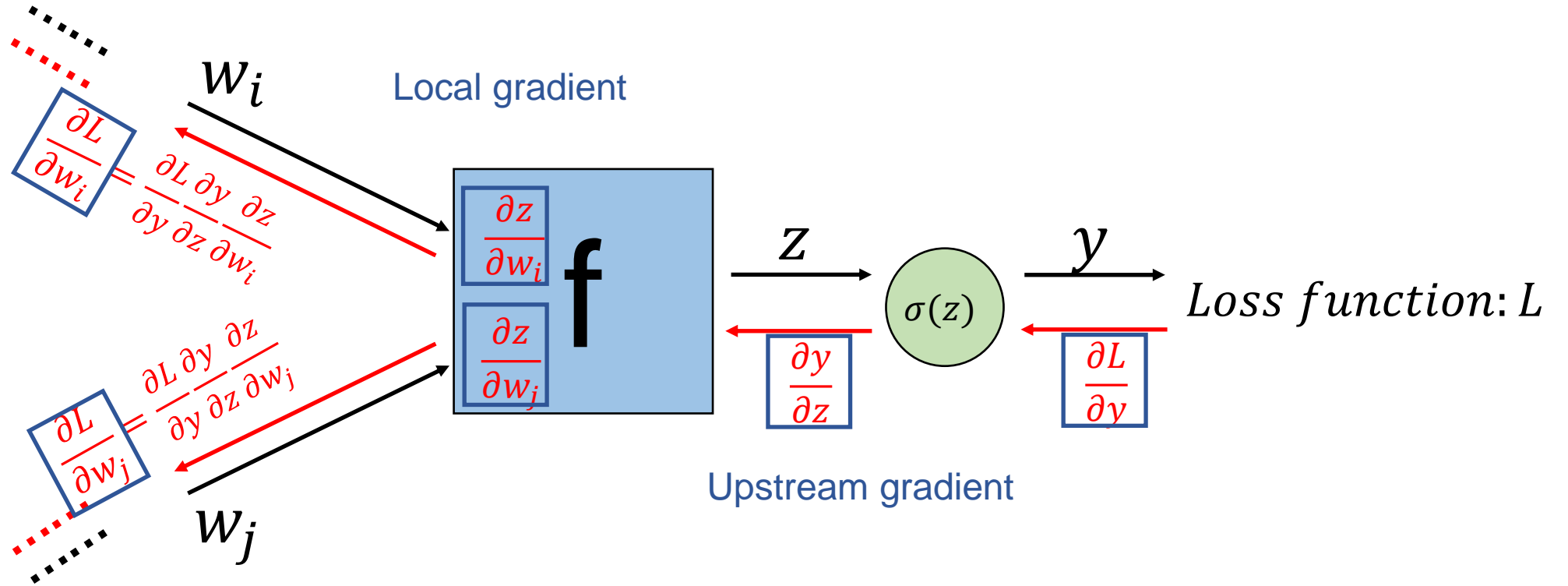
- Initialize the parameter x and learning rate η
- Repeat until the termination condition is met
 - Randomly shuffle examples in the training set
 - For $i = 1, \dots, n$

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



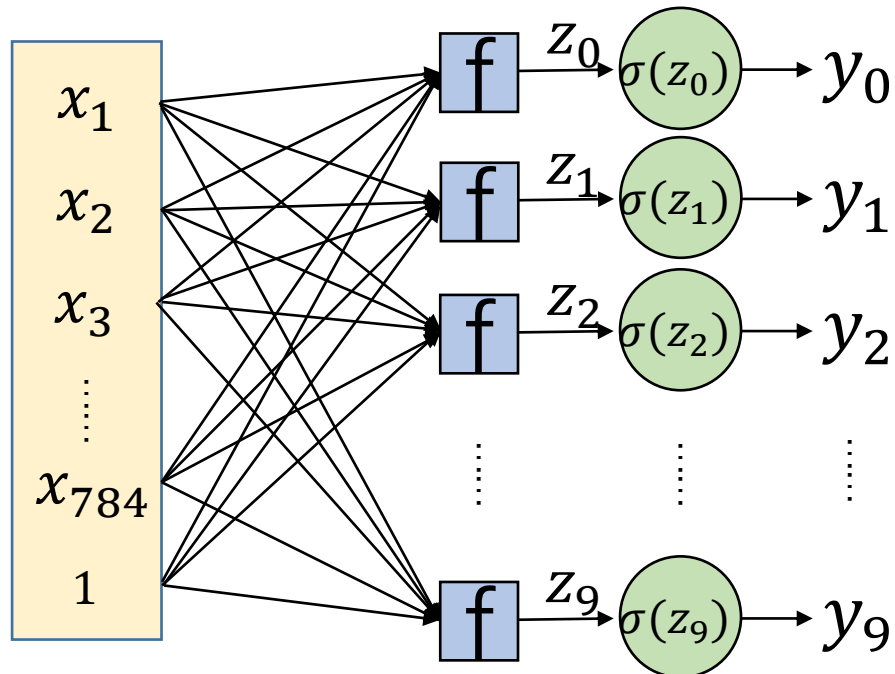
By Joe pharos at the English language Wikipedia, CC BY-SA 3.0,
<https://commons.wikimedia.org/w/index.php?curid=42498187>

Backpropagation



Upstream gradient * Local gradient

Backpropagation



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

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

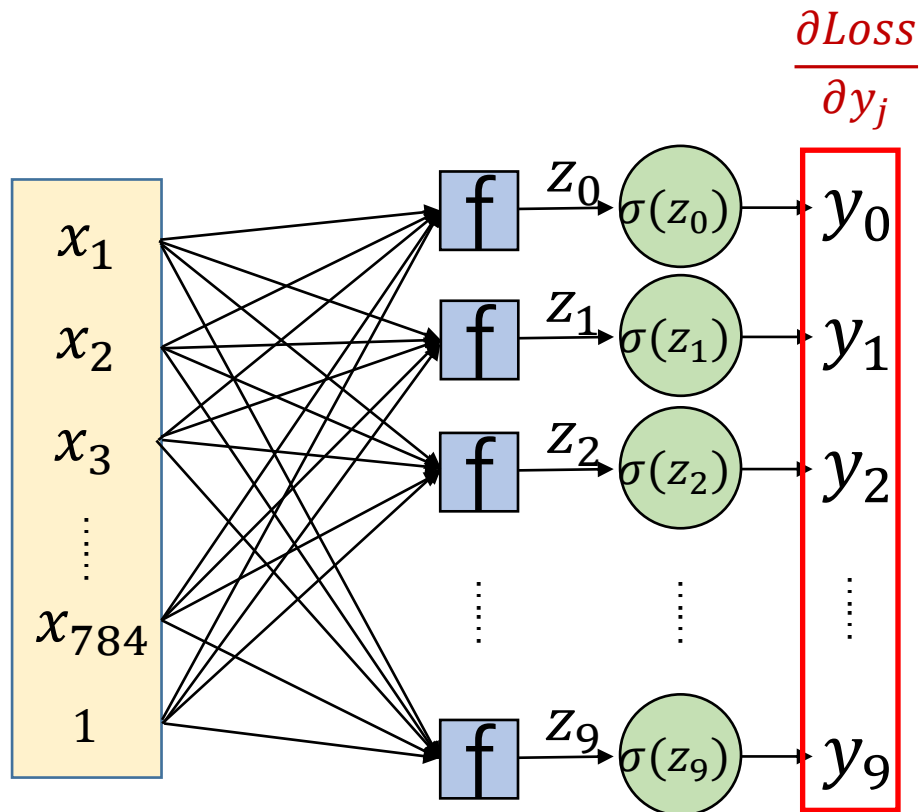
$$\text{Ground truth: } Q = \begin{bmatrix} 0 \\ 1 \\ \dots \\ 0 \end{bmatrix} \in R^{10}$$

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

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

Suppose that, the true label of a given data instance is k . Then
 $\text{Loss} = \text{cross entropy} = - \sum_i q_i \log(p_i) = - \log(p_k)$

Backpropagation



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

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

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

$$\text{Loss} = \text{cross entropy} = -\log(p_k)$$

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

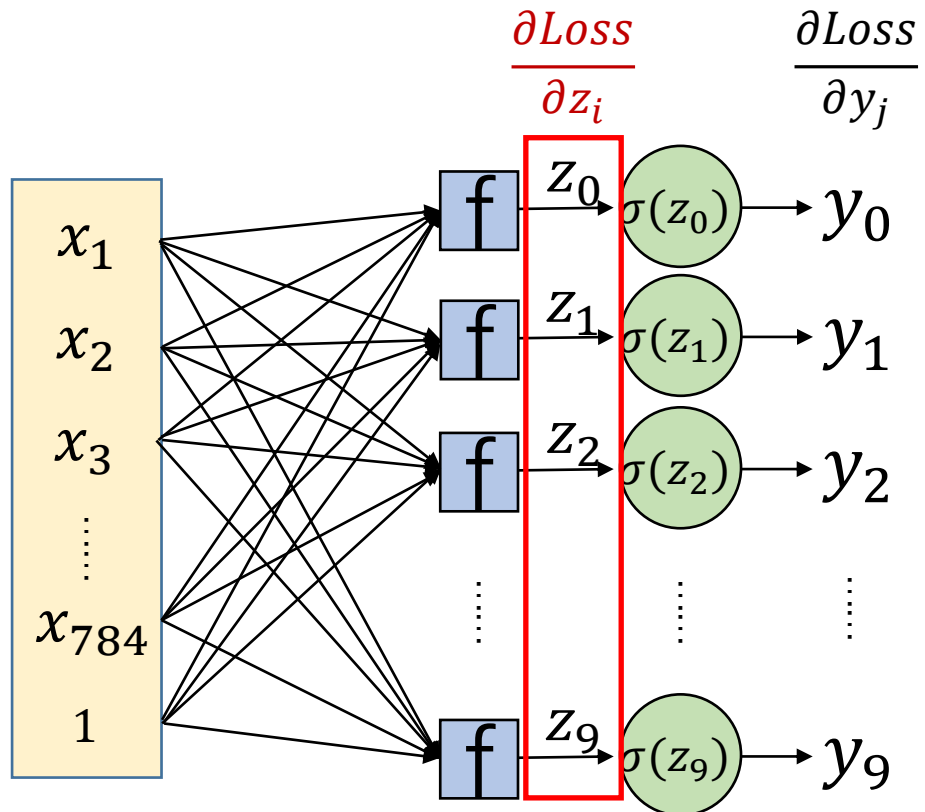
$$\frac{\partial \text{Loss}}{\partial y_j} = \frac{\partial \text{Loss}}{\partial p_k} \frac{\partial p_k}{\partial y_j}$$

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



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

Backpropagation



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

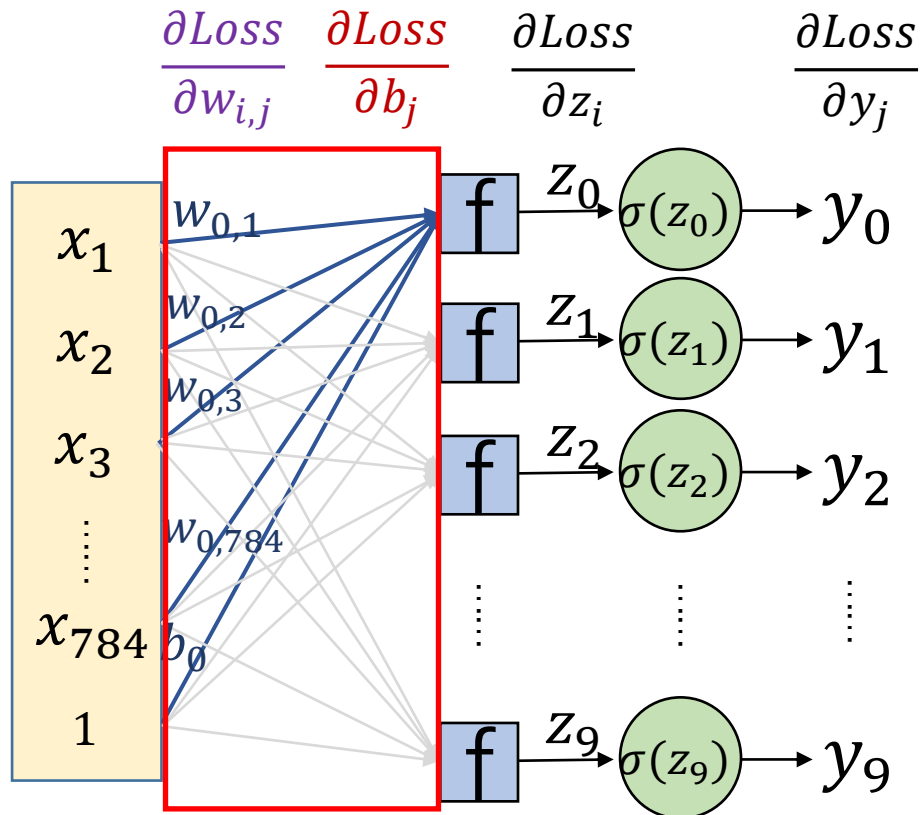
$$b \in R^{1 \times 10}$$

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

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

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

Backpropagation



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

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

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

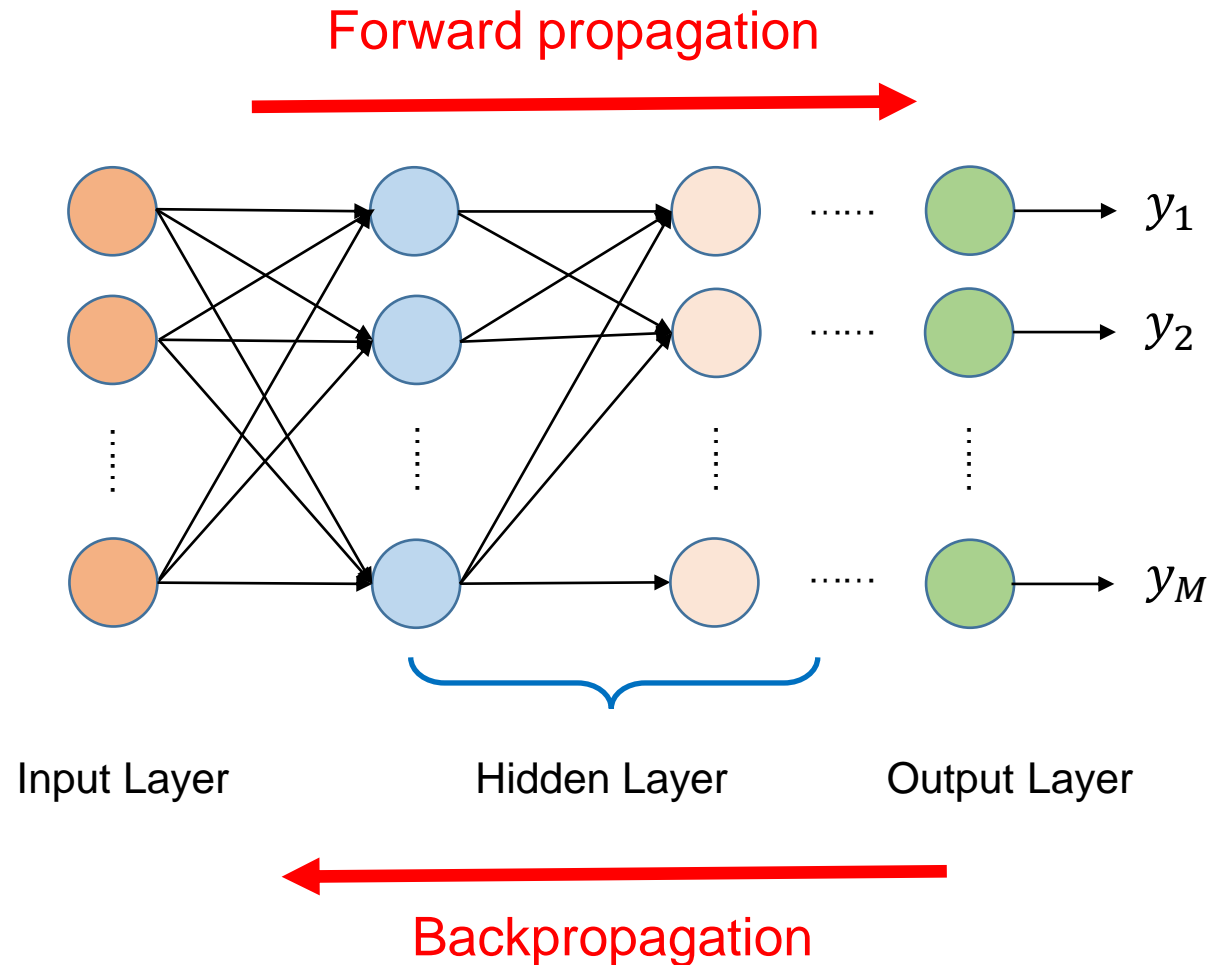
$$\frac{\partial \text{Loss}}{\partial w_{i,j}} = \frac{\partial \text{Loss}}{\partial z_i} \frac{\partial z_i}{\partial w_{i,j}} \quad \frac{\partial z_i}{\partial w_{i,j}} = x_j$$

$$\frac{\partial \text{Loss}}{\partial b_i} = \frac{\partial \text{Loss}}{\partial z_i} \frac{\partial z_i}{\partial b_i} \quad \frac{\partial z_i}{\partial b_i} = 1$$

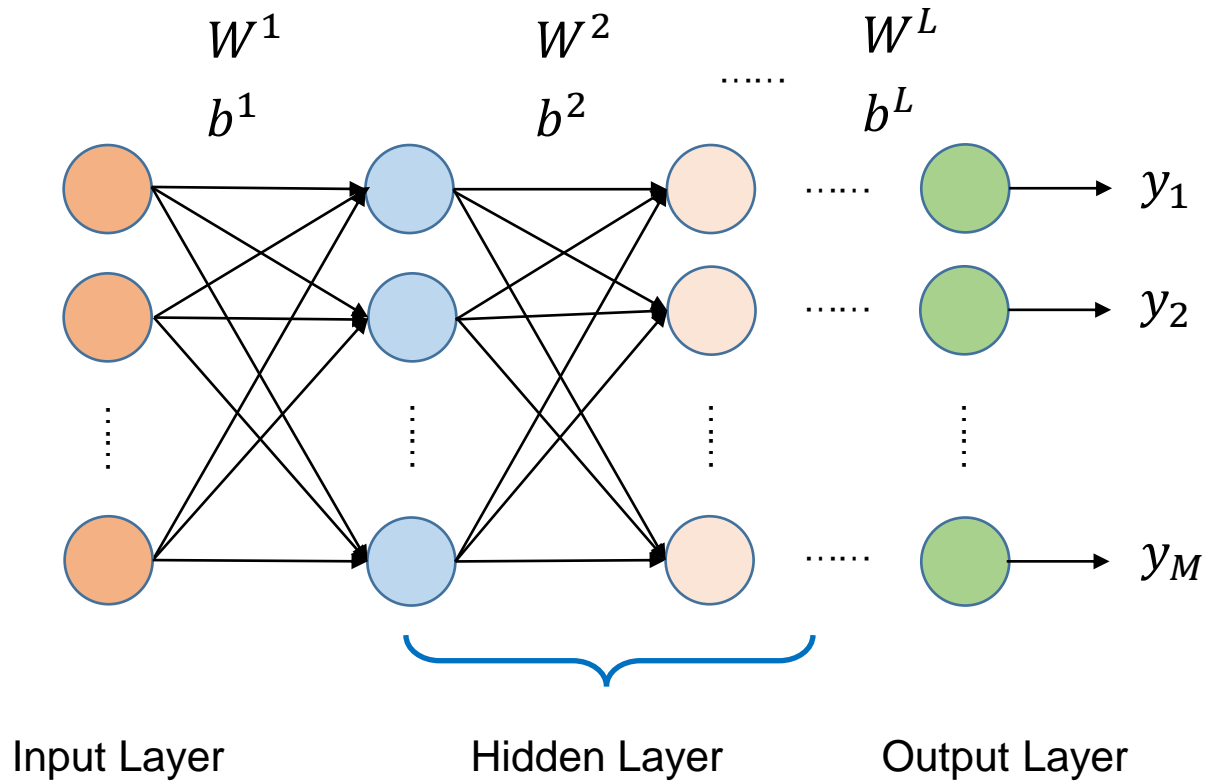
$$W = W - \eta \frac{\partial \text{Loss}}{\partial W}$$

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

Backpropagation: Multi-Layer Perceptron



Backpropagation: Multi-Layer Perceptron



Backpropagation

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

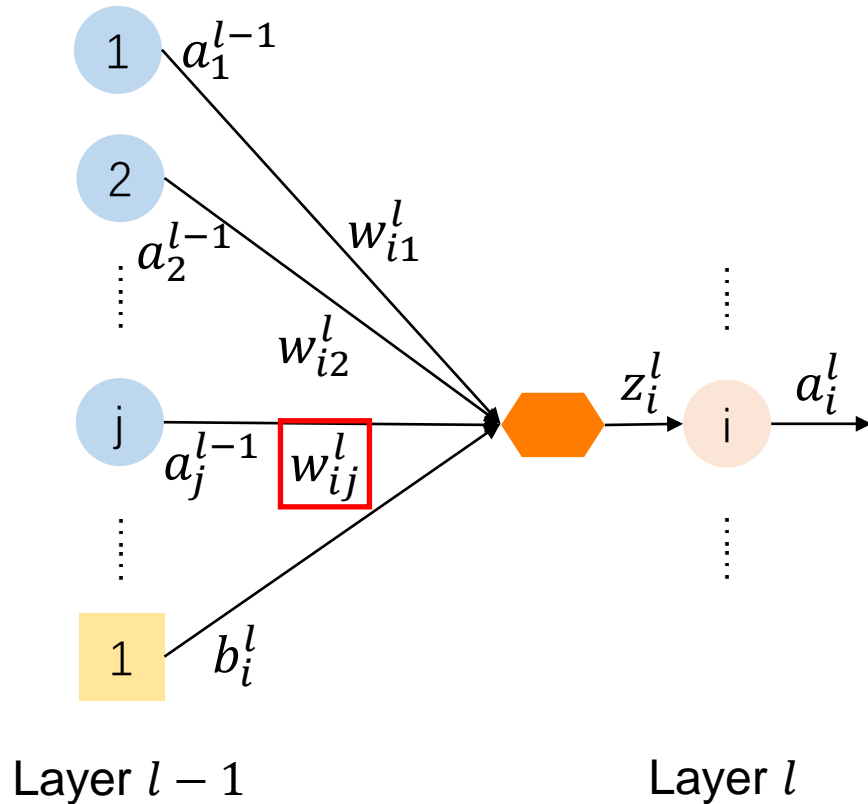
$$W^l = \begin{bmatrix} w_{11}^l & w_{12}^l & \dots \\ w_{21}^l & w_{22}^l & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \quad b^l = \begin{bmatrix} \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

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

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

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

Backpropagation: Multi-Layer Perceptron



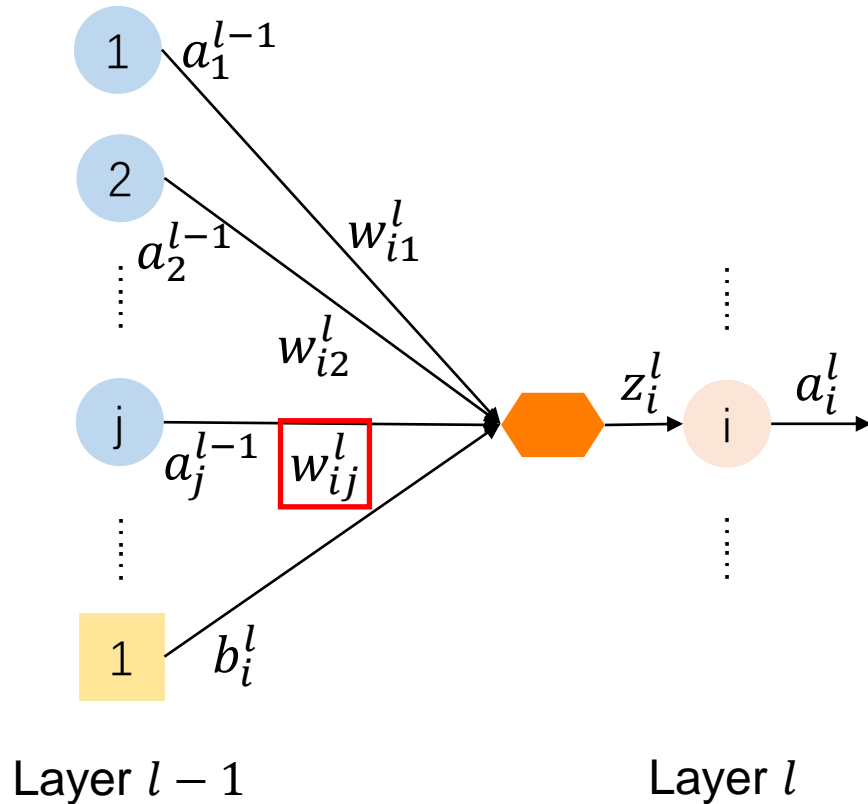
a_i^l : output of a neuron w_{ij}^l : a weight of layer l

b_i^l : a bias of layer l 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



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

If $l > 1$:

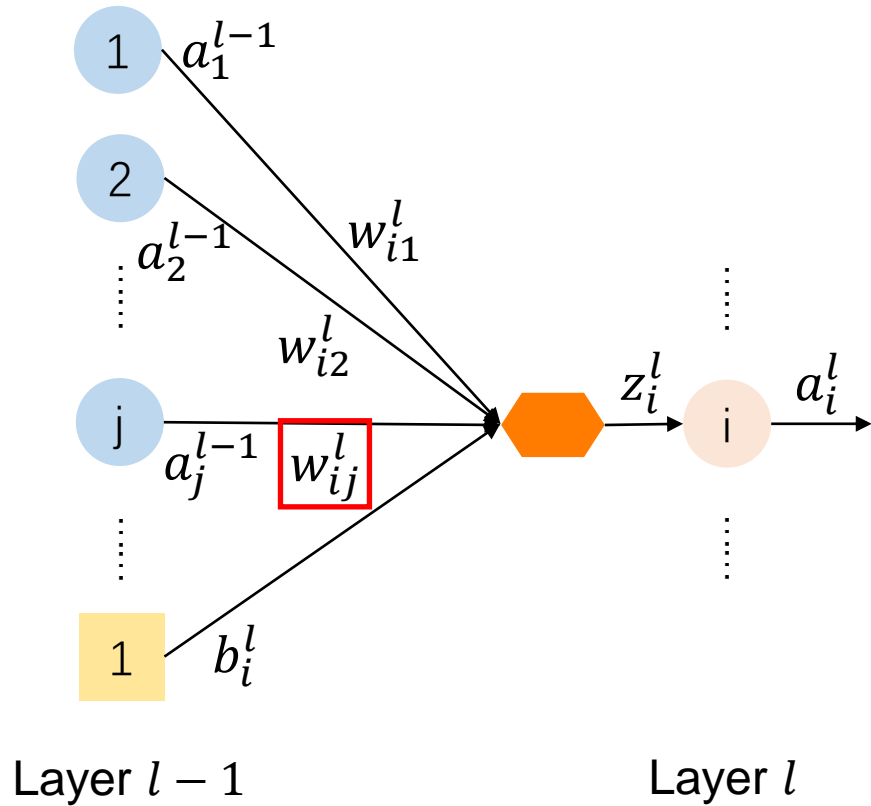
$$z_i^l = \sum_j w_{ij}^l a_j^{l-1} + b_i^l$$

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

If $l=1$:

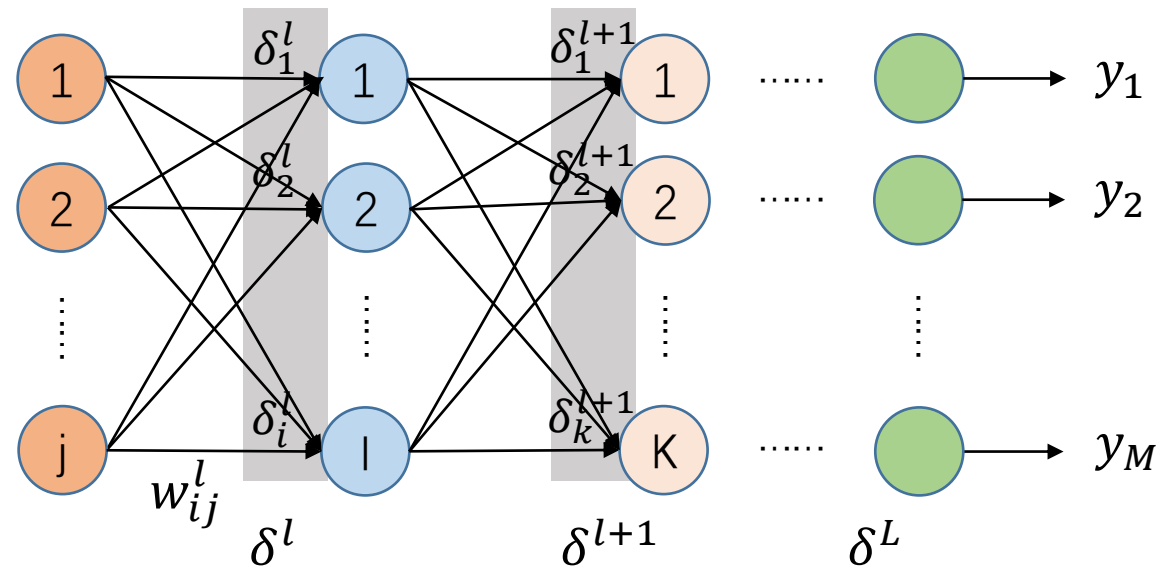
$$\frac{\partial z_i^l}{\partial w_{ij}^l} = x_j$$

Backpropagation: Multi-Layer Perception

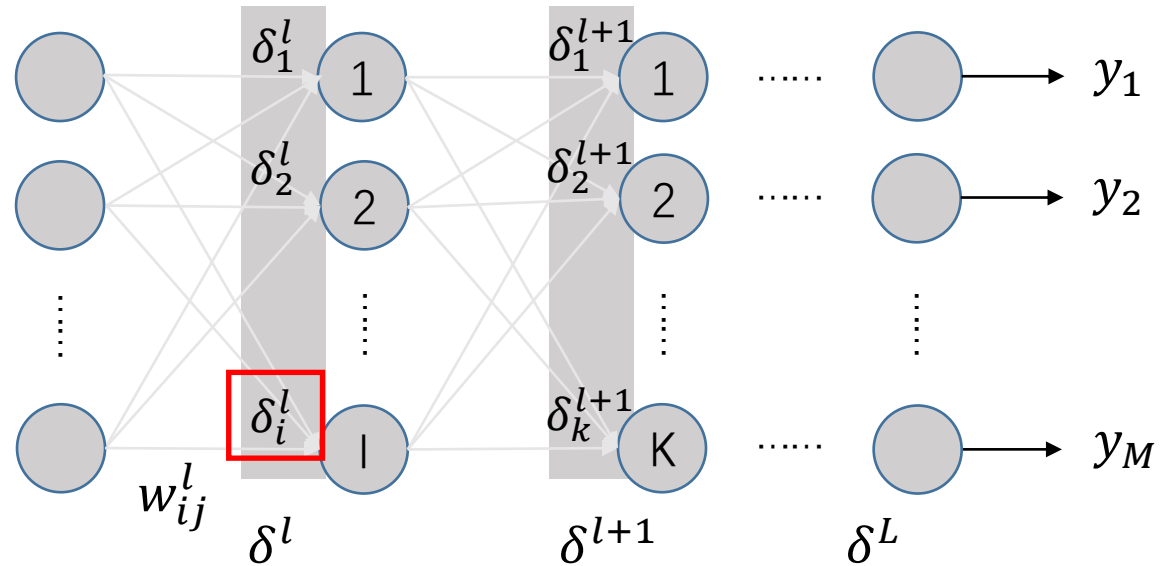
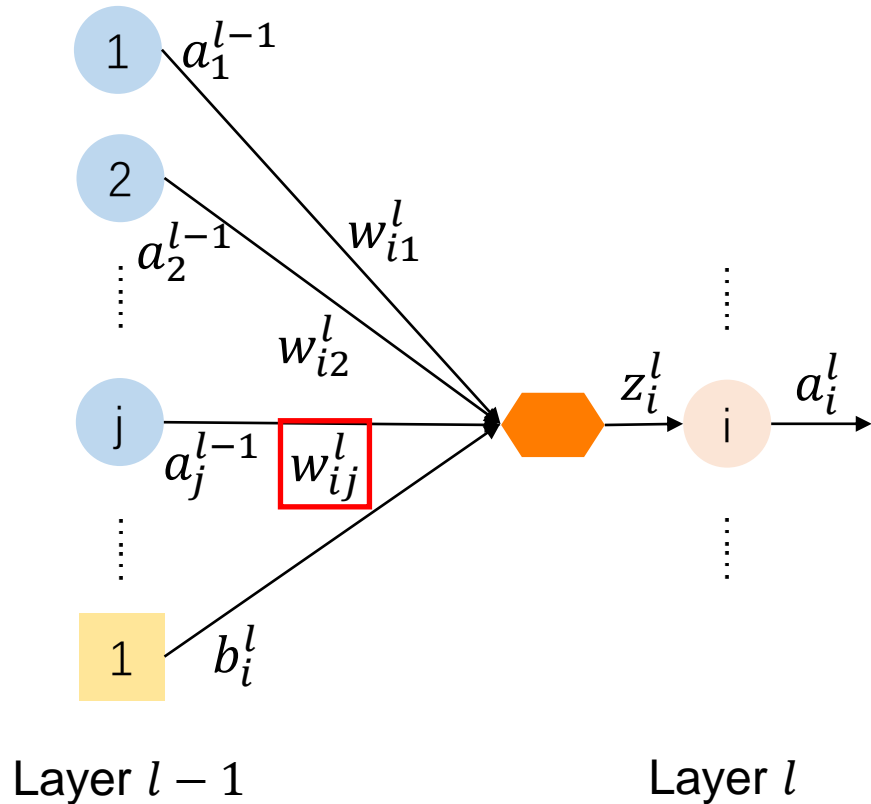


$$\frac{\partial \text{Loss}(\theta)}{\partial w_{ij}^l} = \frac{\partial \text{Loss}(\theta)}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l}$$

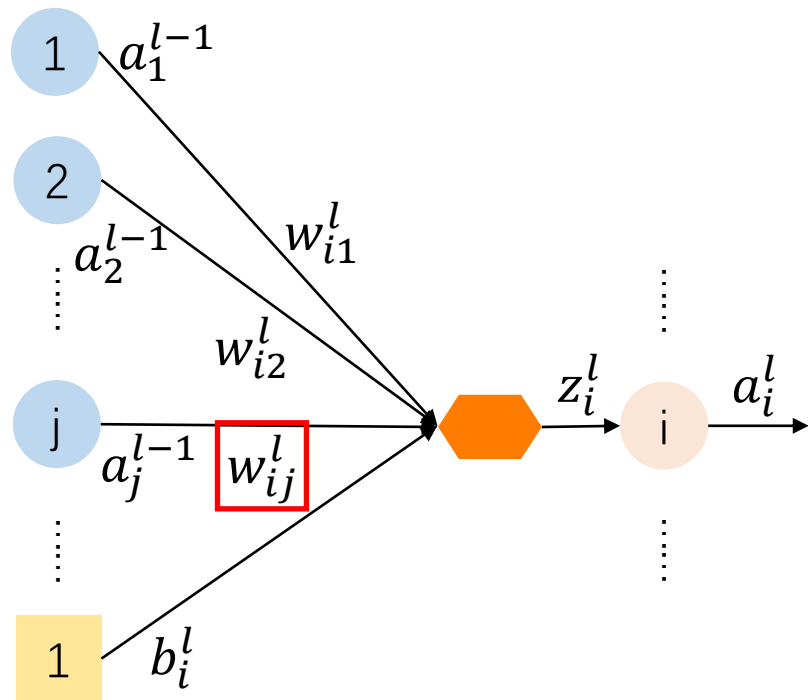
$$\delta_i^l = \frac{\partial \text{Loss}(\theta)}{\partial z_i^l}$$



Backpropagation: Multi-Layer Perception

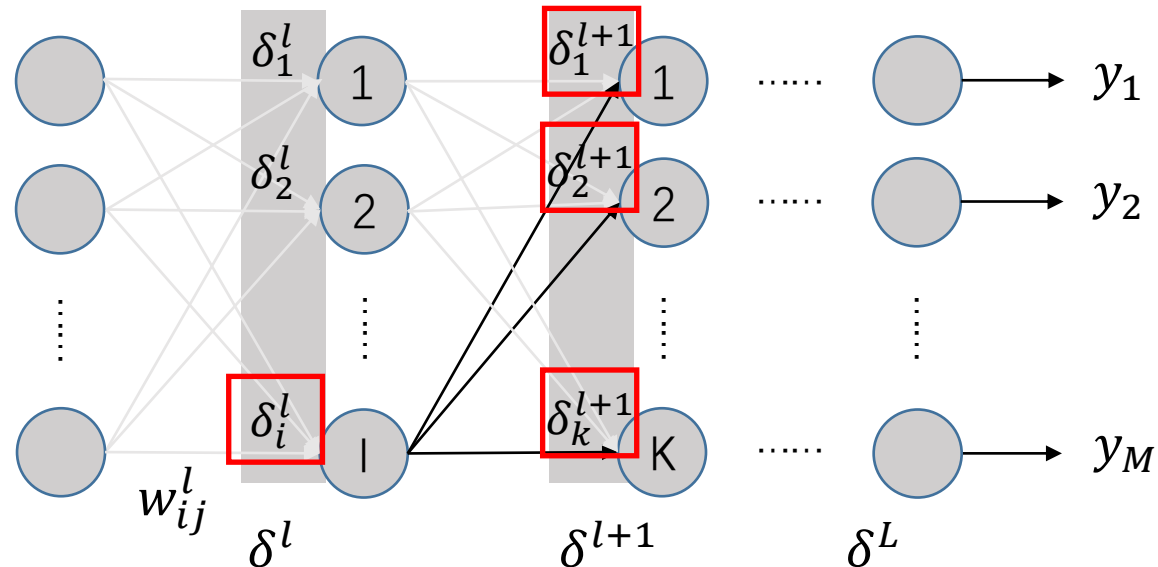


Backpropagation: Multi-Layer Perception



Layer $l - 1$

Layer l



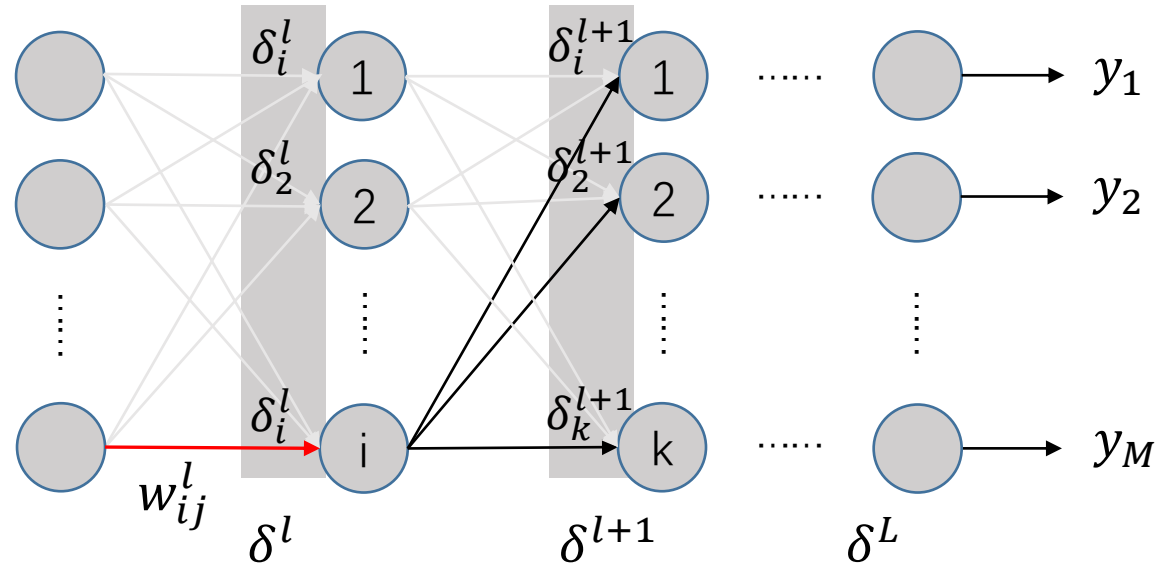
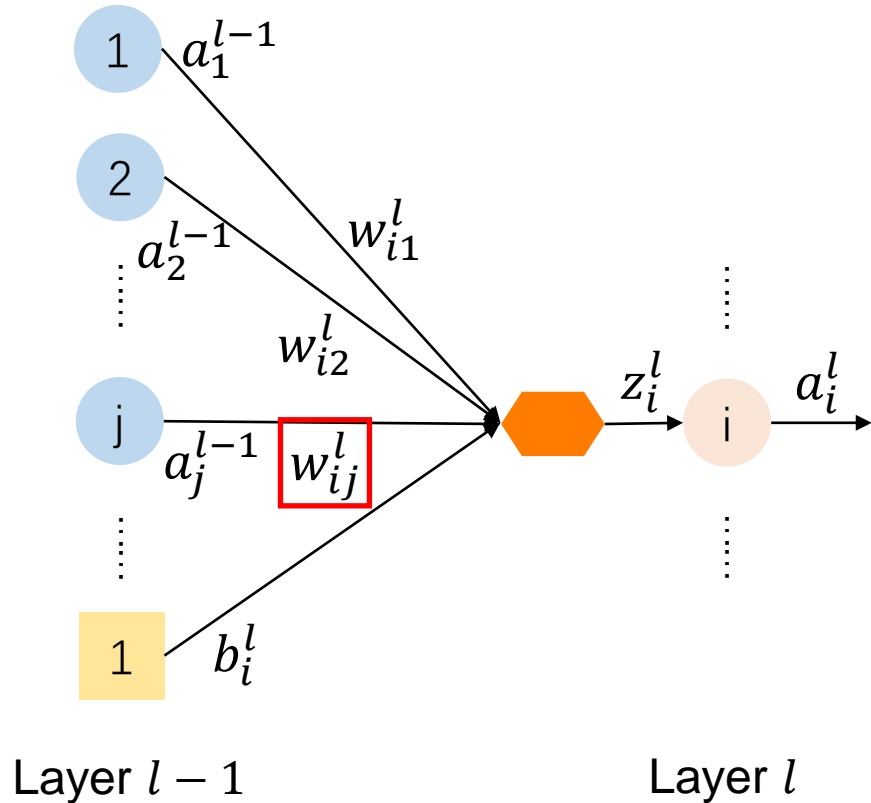
$$\delta_i^l = \frac{\partial \text{Loss}(\theta)}{\partial z_i^l} = \frac{\partial \text{Loss}(\theta)}{\partial z_1^{l+1}} \frac{\partial z_1^{l+1}}{\partial a_i^l} \frac{\partial a_i^l}{\partial z_i^l} + \dots + \frac{\partial \text{Loss}(\theta)}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial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

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

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

$$\delta^l = \sigma'(z^l) \odot \left((W^{l+1})^T \delta^{l+1} \right) \leftarrow \sigma'(z_i^l) \sum_k w_{ki}^{l+1} \delta_k^{l+1}$$

Backpropagation: Multi-Layer Perception

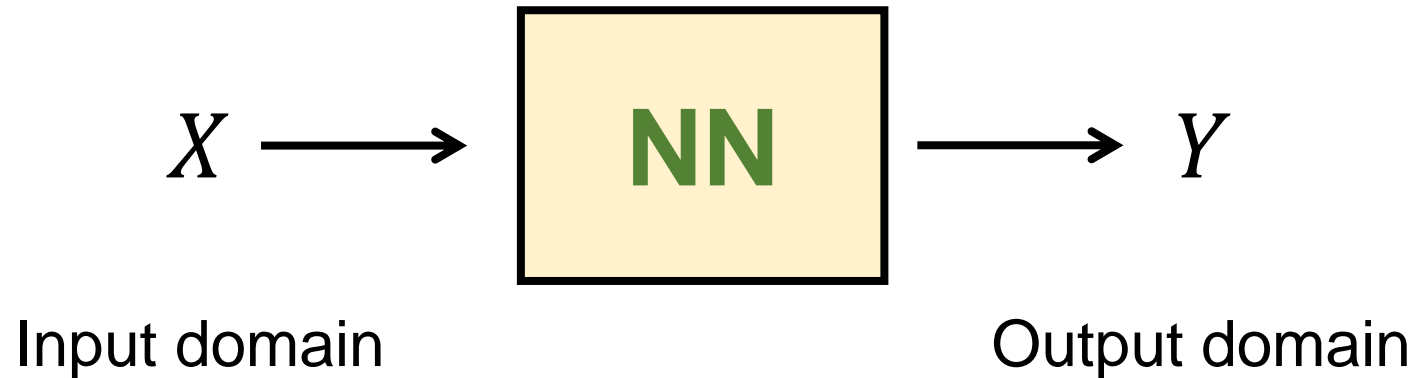


$$\frac{\partial \text{Loss}(\theta)}{\partial w_{ij}^l} = \frac{\partial \text{Loss}(\theta)}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l} = \delta_i^l \frac{\partial z_i^l}{\partial w_{ij}^l}$$

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

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

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

$$f(\text{1}) = \text{"1"}$$

- Speech Recognition

$$f(\text{audio waveform}) = \text{"Hello, MIRA"}$$

In fact, the neural networks are universal
function approximators!

Universal Function Approximator

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

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

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

- Hinge loss

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

- Logistic loss

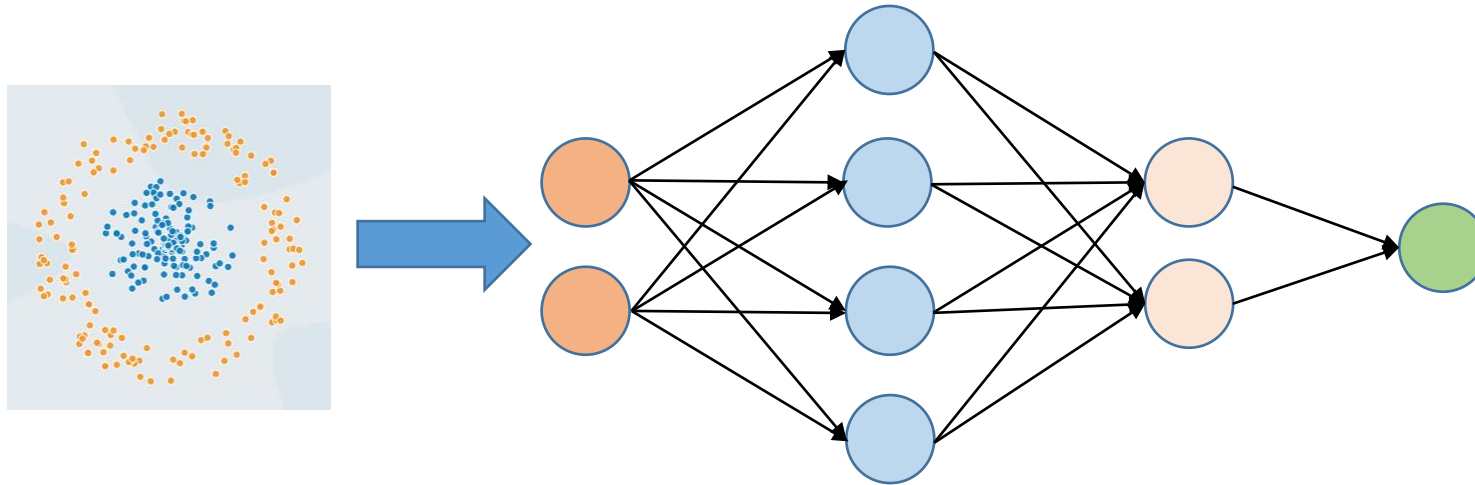
$$Loss = -y \log(f(x; \theta))$$

- Cross entropy loss

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

Demonstration

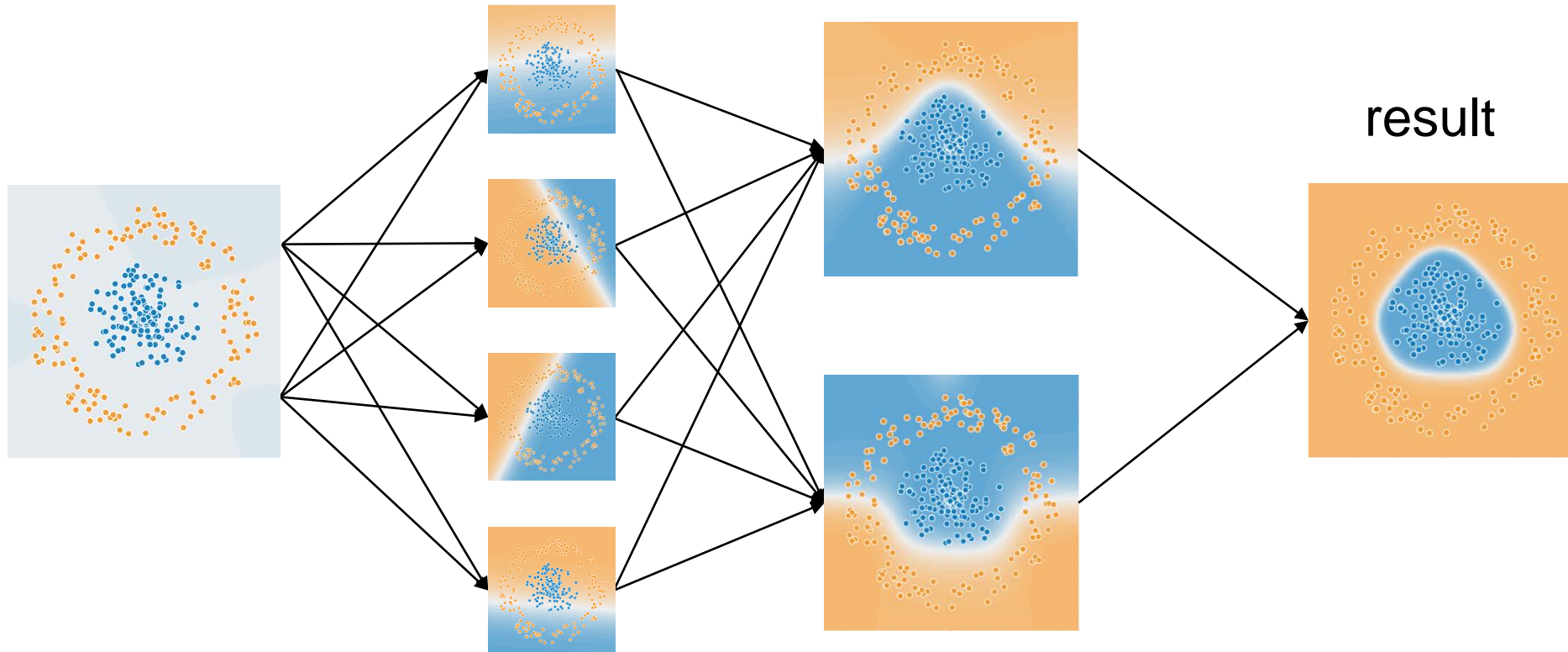
Classification Problem



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

Tips

Deeper is Better?

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



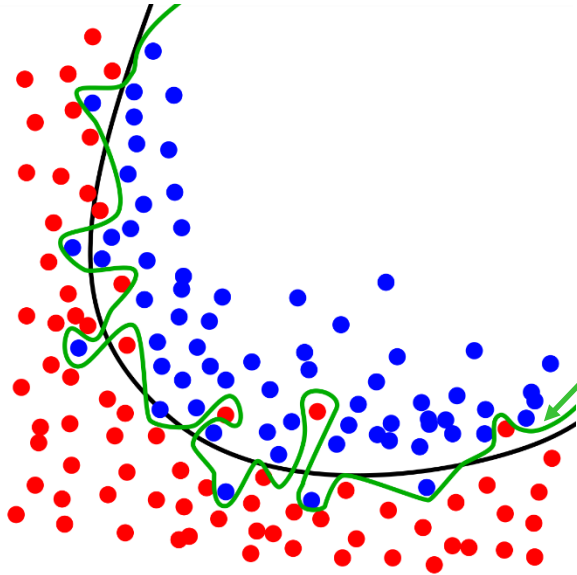
Deeper is Better?

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

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

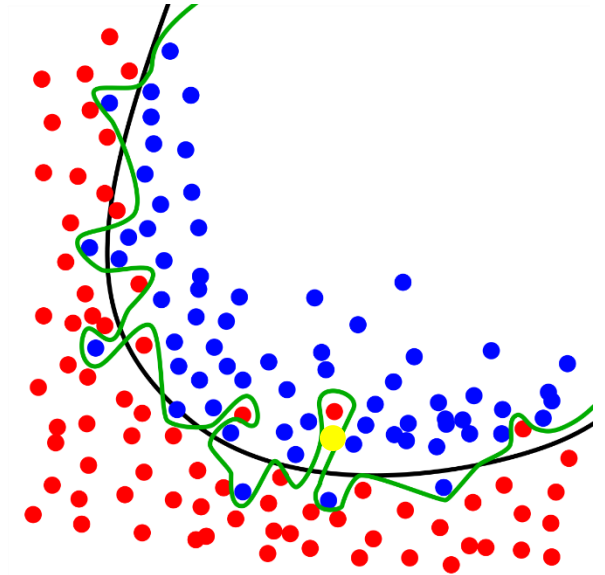
Deep structure can capture complex patterns more efficiently than the shallow one.

Overfitting



The generalization performance of this model can be poor.

Which one is better?

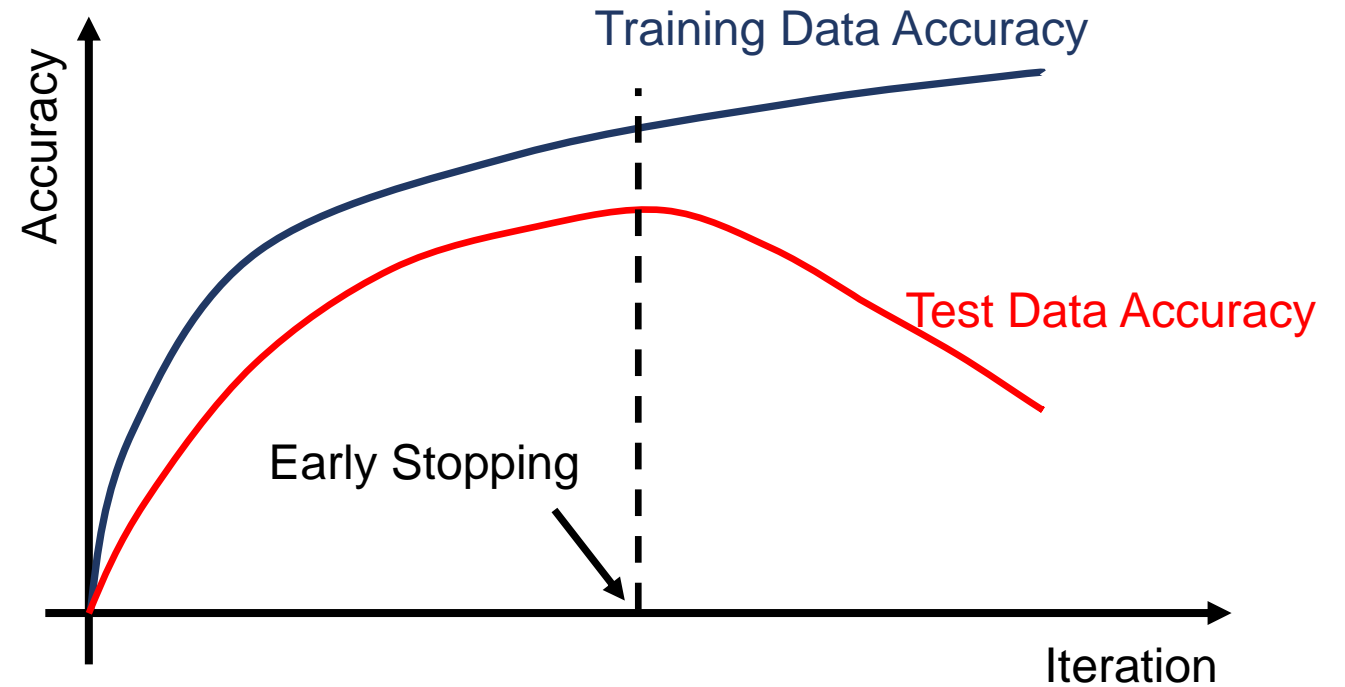


The predicted label is ~~red!~~

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
- ...

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

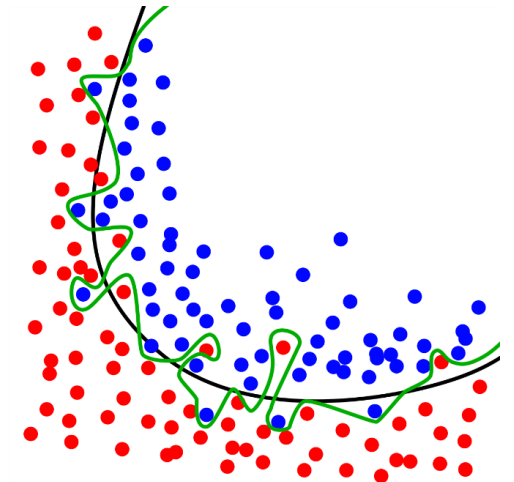
regularization term

➤ ℓ_2 norm

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

➤ ℓ_1 norm

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



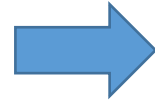
Small weights usually imply smooth decision boundary.

L2 Regularization

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

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

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



$$\begin{aligned} \theta^{t+1} &:= \theta^t - \eta \frac{\partial Loss'}{\partial \theta^t} \\ &= \theta^t - \eta \left(\frac{\partial Loss}{\partial \theta^t} + \lambda \theta^t \right) \\ &= (1 - \eta \lambda) \theta^t - \eta \frac{\partial Loss}{\partial \theta^t} \end{aligned}$$

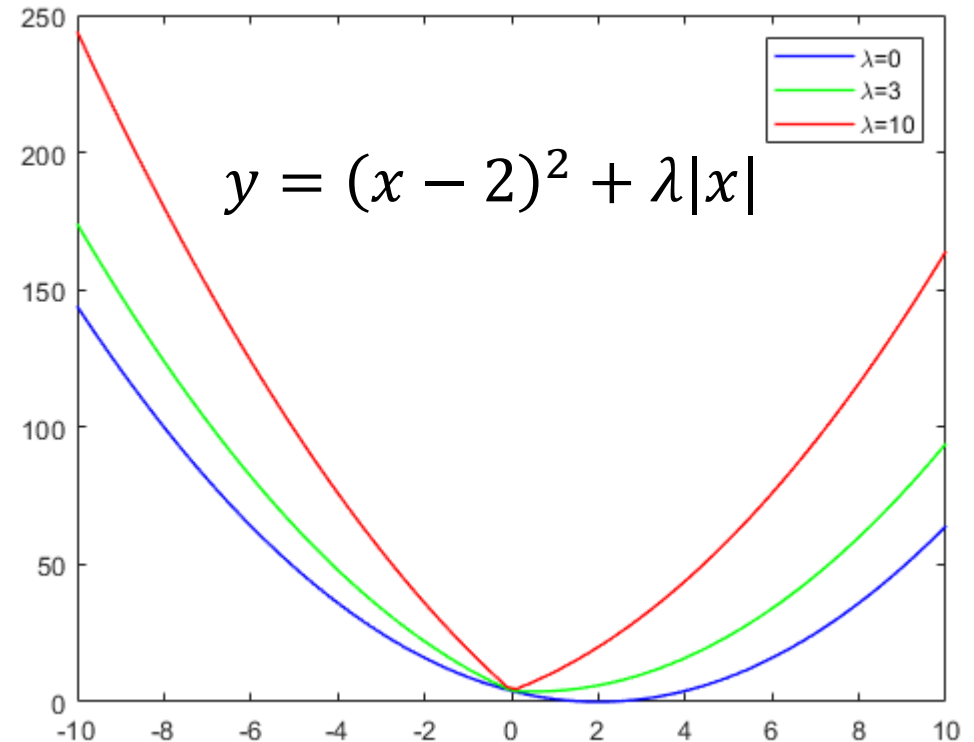
L1 Regularization

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

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

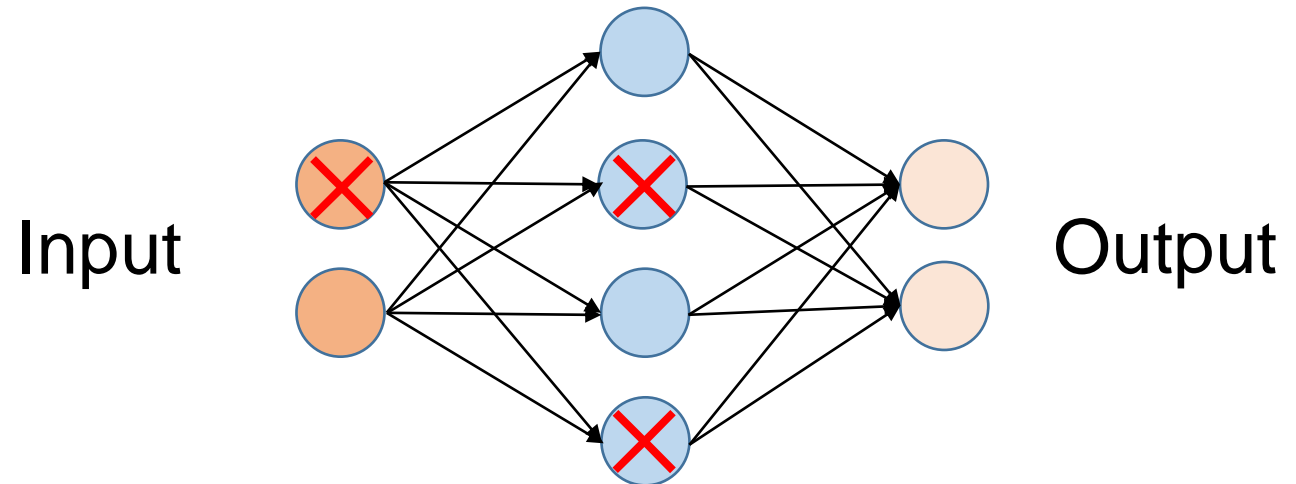
$$\frac{\partial Loss'}{\partial \theta} = \frac{\partial Loss}{\partial \theta} + \lambda * sgn(\theta)$$

$$\begin{aligned}\theta^{t+1} &:= \theta^t - \eta \frac{\partial Loss'}{\partial \theta^t} \\ &= \theta^t - \eta \left(\frac{\partial Loss}{\partial \theta^t} + \lambda sgn(\theta^t) \right) \\ &= \theta^t - \eta \lambda sgn(\theta^t) - \eta \frac{\partial Loss}{\partial \theta^t}\end{aligned}$$



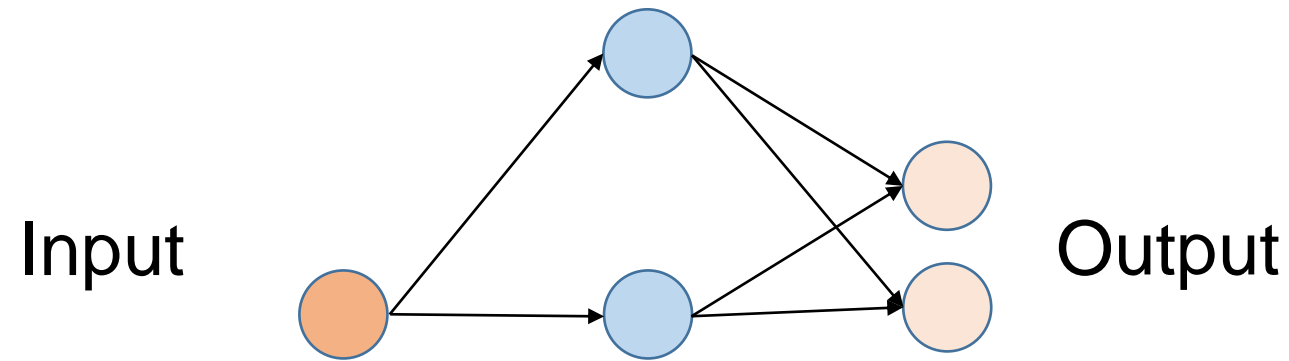
Preventing Overfitting in DNN

- Early Stopping
- Regularization
- **Dropout**
- ...



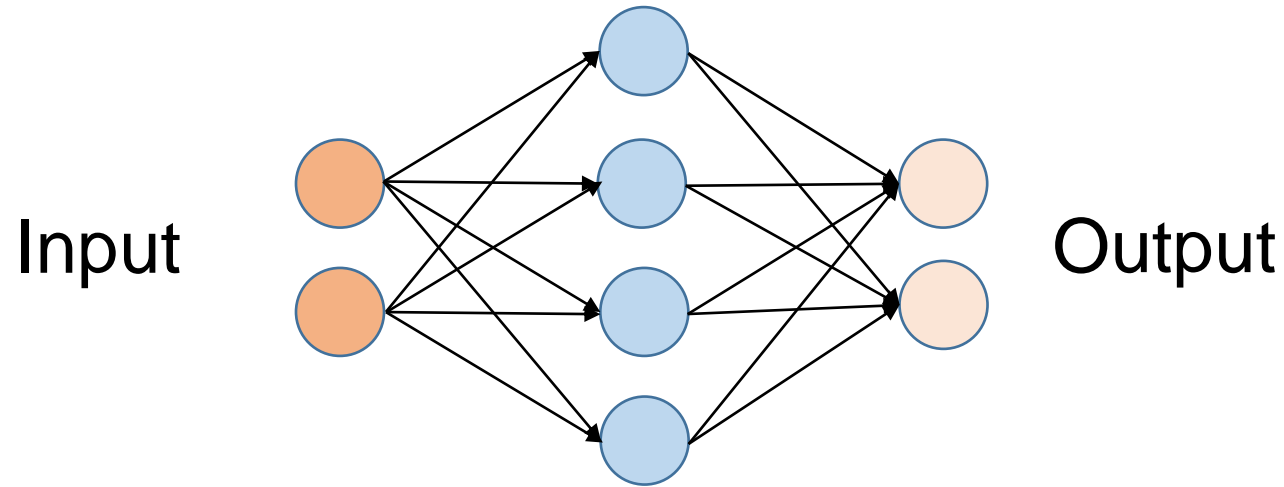
Training: We drop each neuron with probability p

Dropout

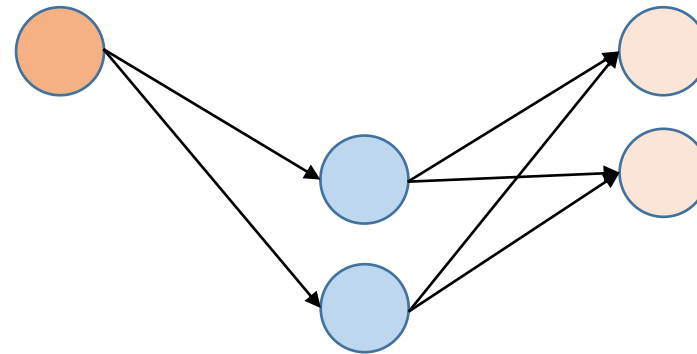
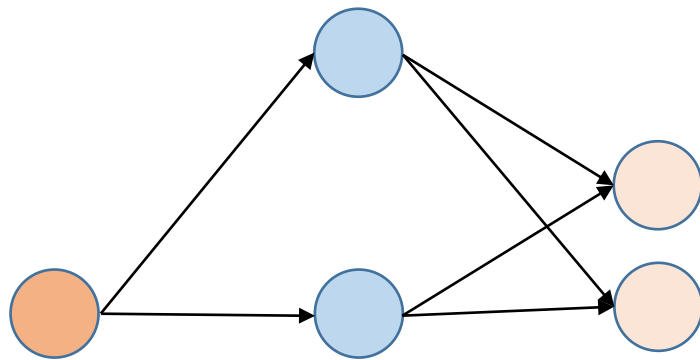


Training: We dropout each neuron with probability p . Then, we train the resulting network for one iteration.

Dropout



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



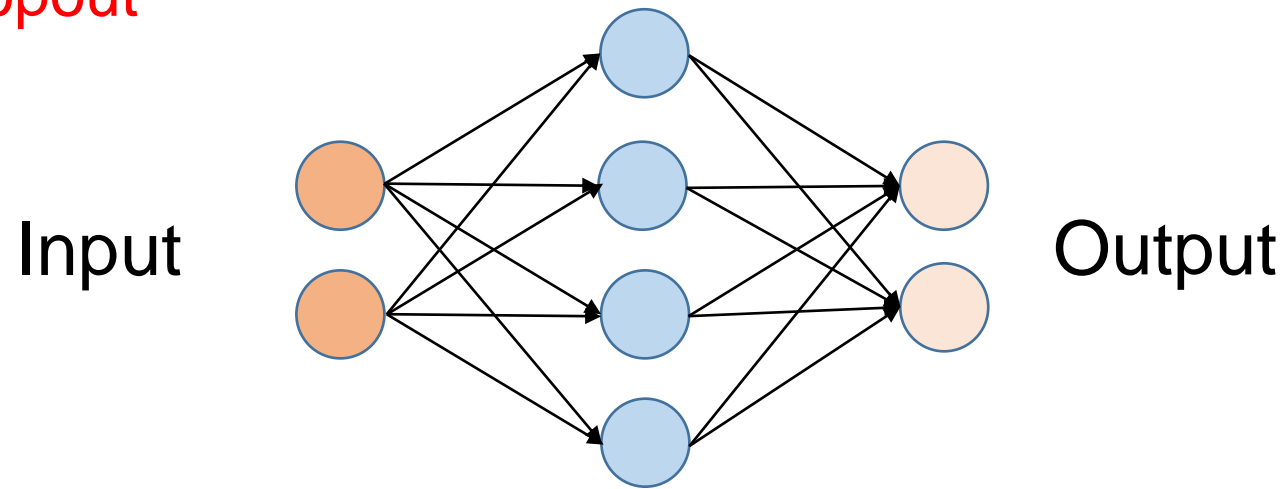
.....

.....

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

Dropout

Testing: No dropout

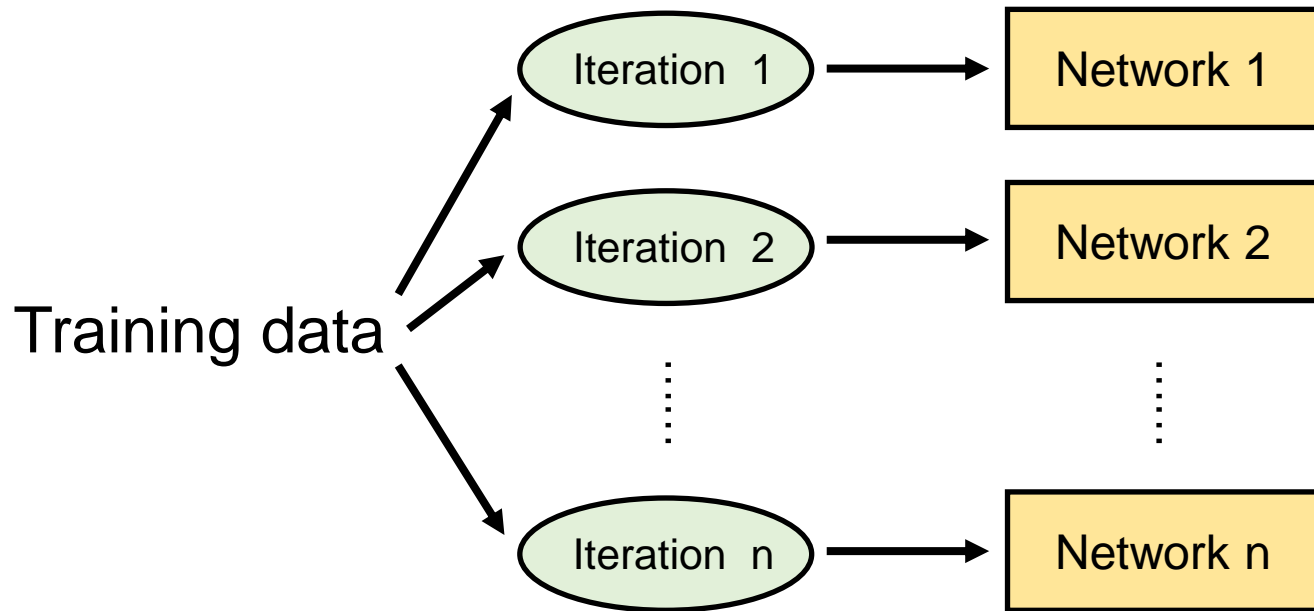


$$W_{test} = (1 - p)W_{train}$$

Why?

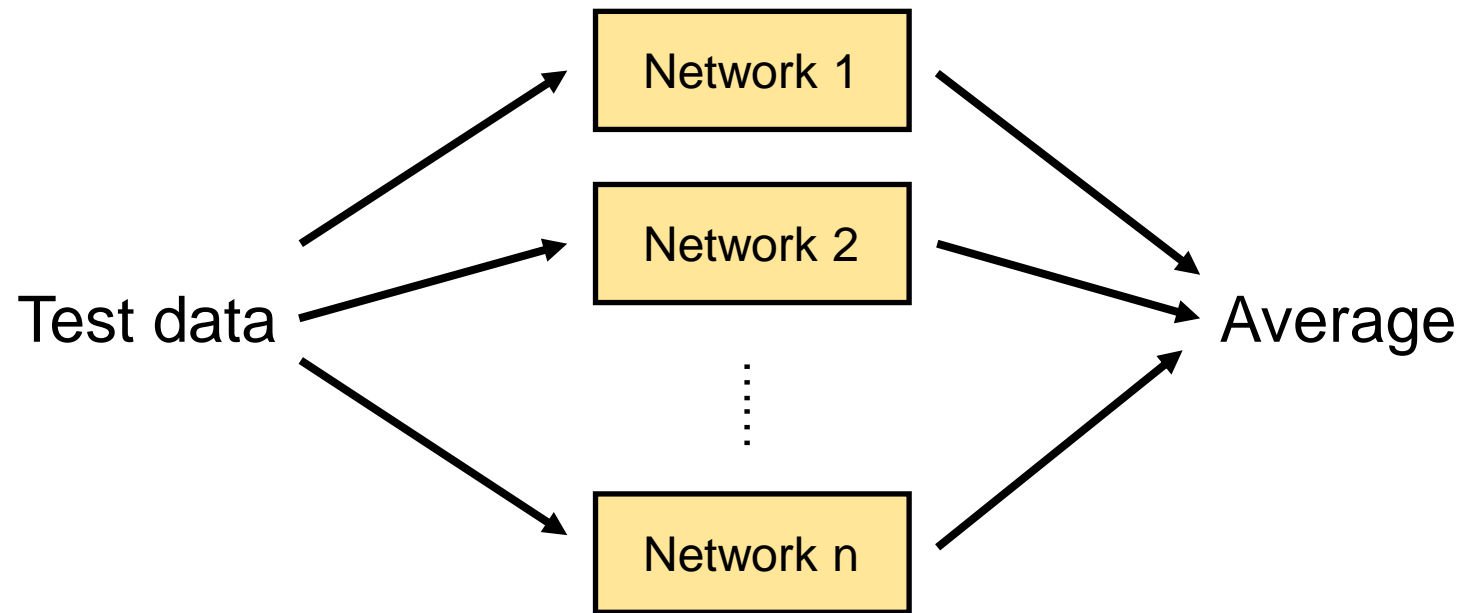
Why Dropout

Dropout is a kind of ensemble



Why Dropout

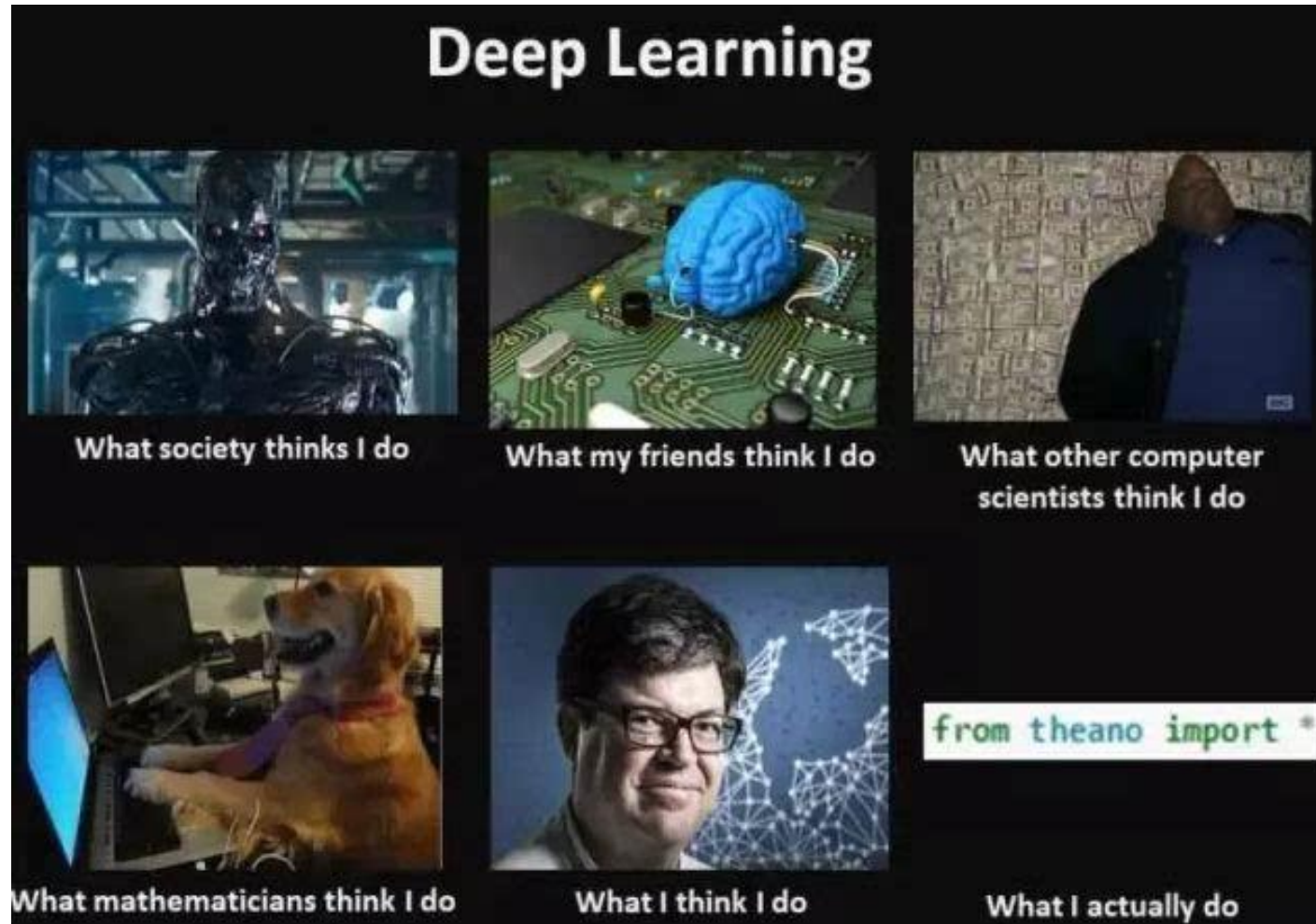
Dropout is a kind of ensemble



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

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

Design Deep model



Questions

