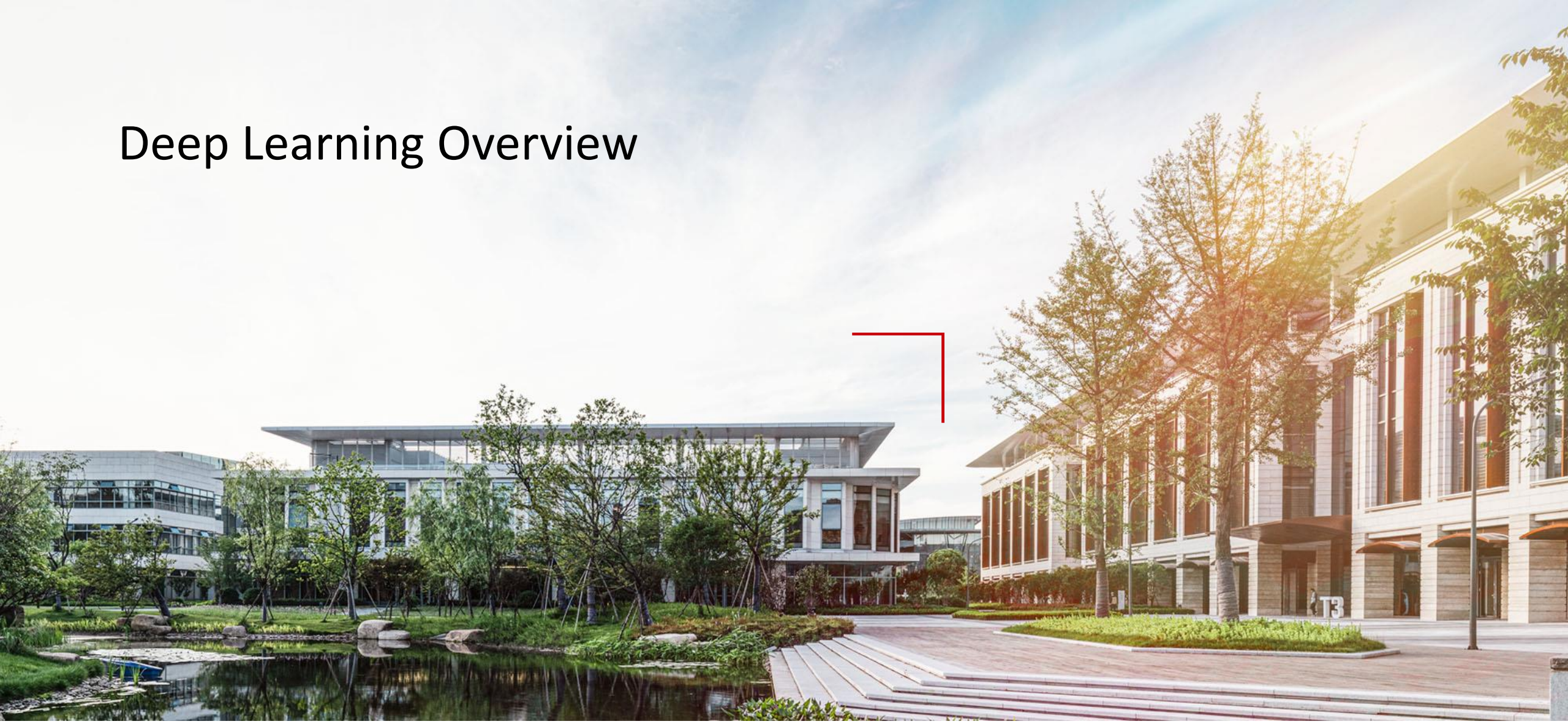
# Deep Learning Overview



HUAWEI

# Foreword

- This chapter describes the basic knowledge related to deep learning, including the development history, components and types of neural networks for deep learning, and common problems in deep learning engineering.

# Objectives

- Upon completion of this course, you will be able to:
  - Describe the definition and development of neural networks.
  - Be familiar with the components of neural networks for deep learning.
  - Be familiar with the training and optimization of neural networks.
  - Describe common problems in deep learning.

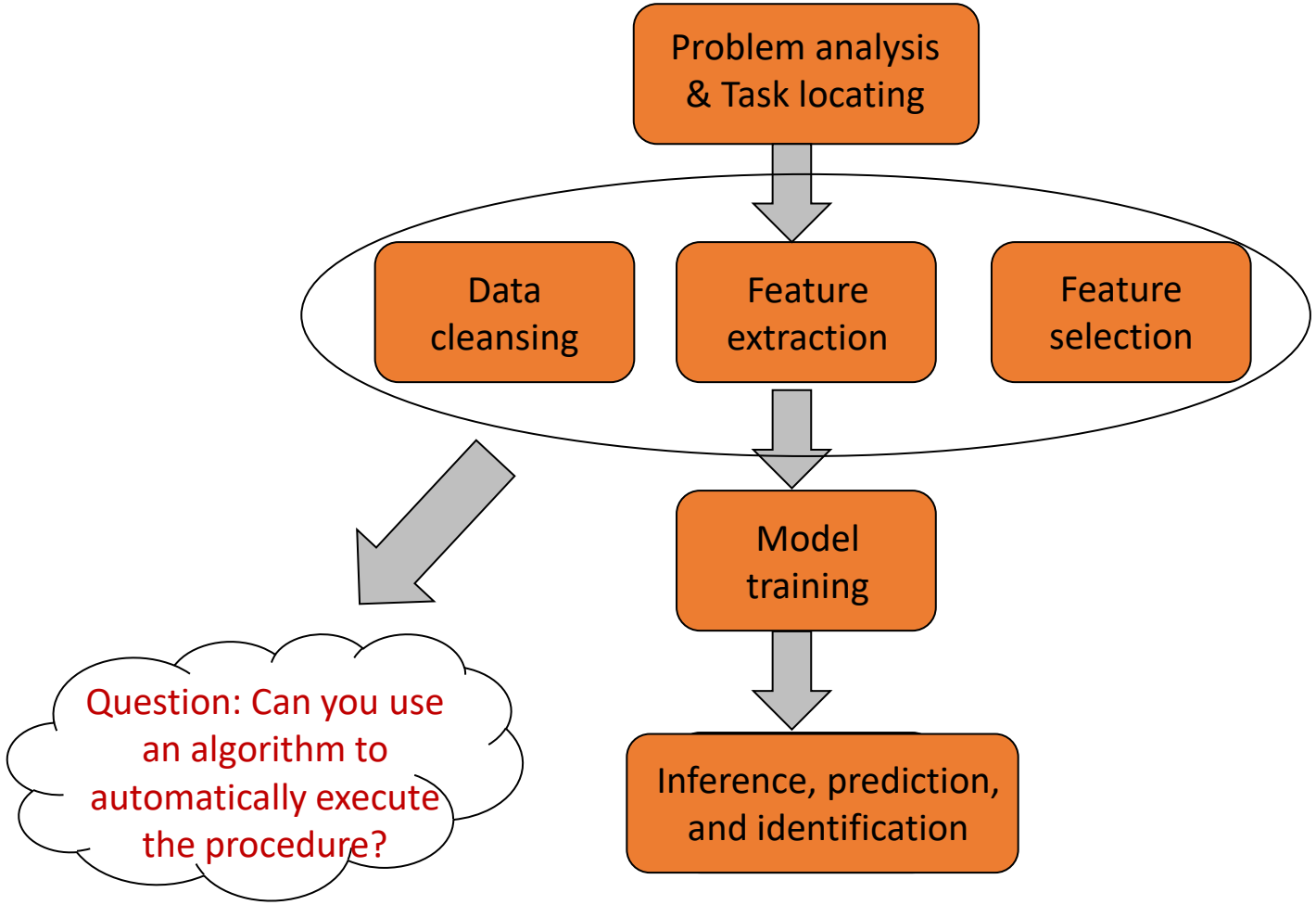
# Contents

# Traditional Machine Learning vs. Deep Learning

- Deep learning is a learning model based on unsupervised feature learning and the feature hierarchical structure. It has great advantages in computer vision, speech recognition, and natural language processing (NLP).

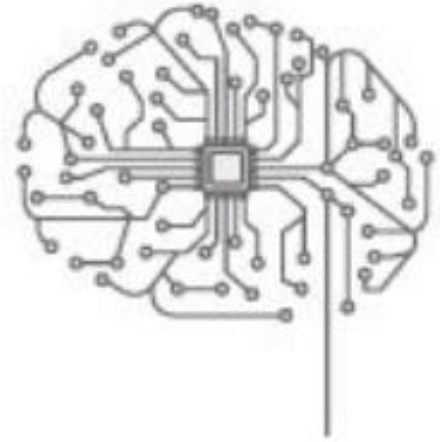
| Traditional Machine Learning | Deep Learning |
| --- | --- |
| It has low requirements for hardware because the amount of computation is limited, and GPUs are not required for parallel computing. | It has certain hardware requirements because numerous matrix operations are needed, and also requires GPUs for parallel computing. |
| It is suitable for training with a small amount of data, but its performance cannot improve as the amount increases. | It achieves high performance when high-dimensional weight parameters and massive training data are provided. |
| Problems are located level by level. | It is an end-to-end learning method. |
| Features are manually selected. | Algorithms are used to automatically extract features. |
| Feature interpretability is strong. | Feature interpretability is weak. |

# Traditional Machine Learning

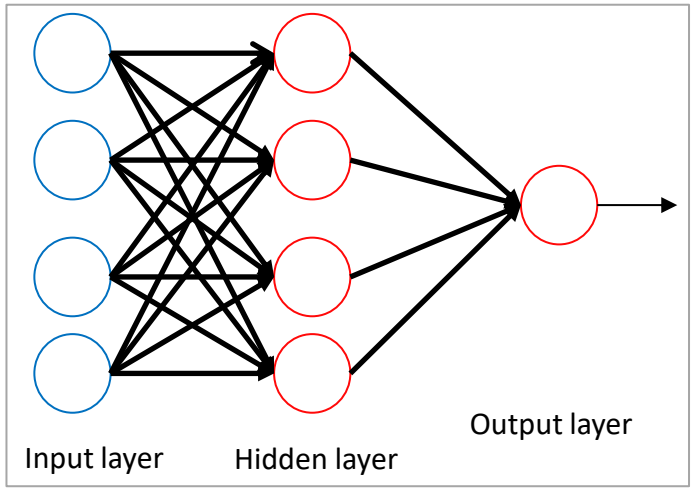


# Deep Learning

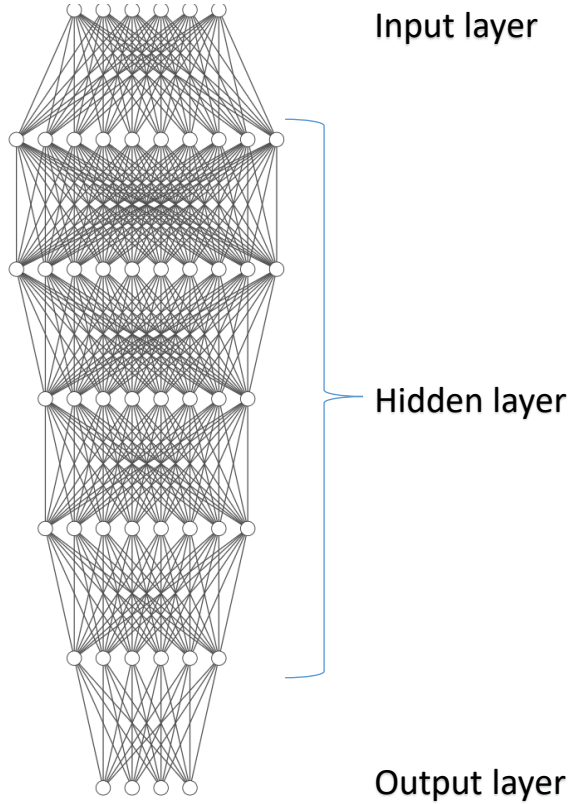
- Generally, the deep learning architecture is a deep neural network. "Deep" in "deep learning" refers to the number of layers of the neural network.



Human neural network




Perceptron

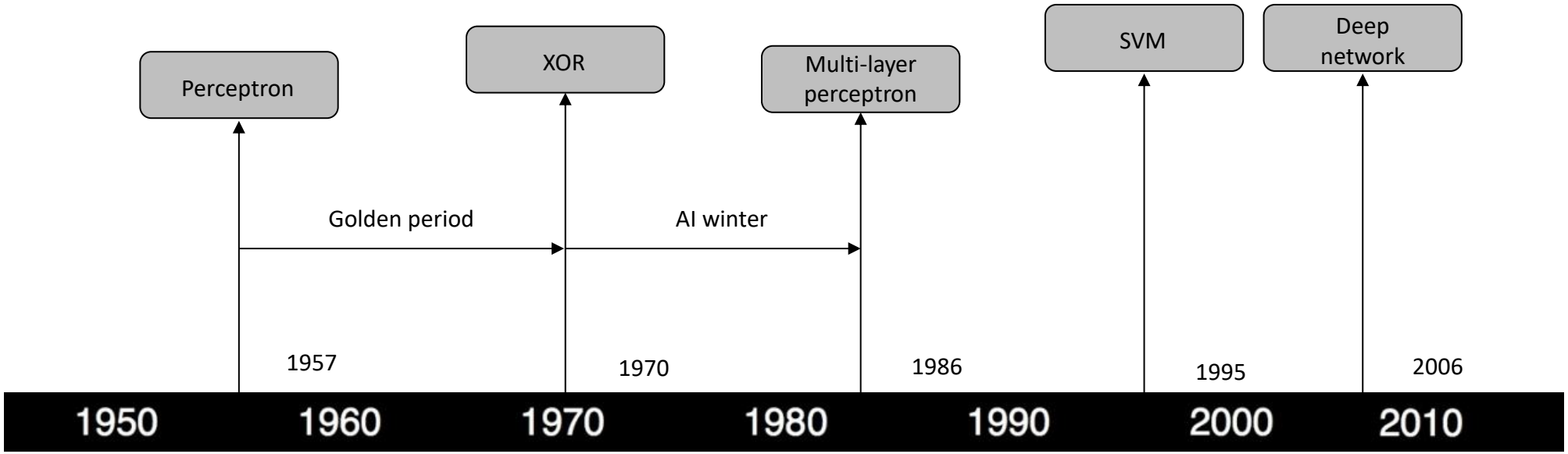



Deep neural network

# Neural Network

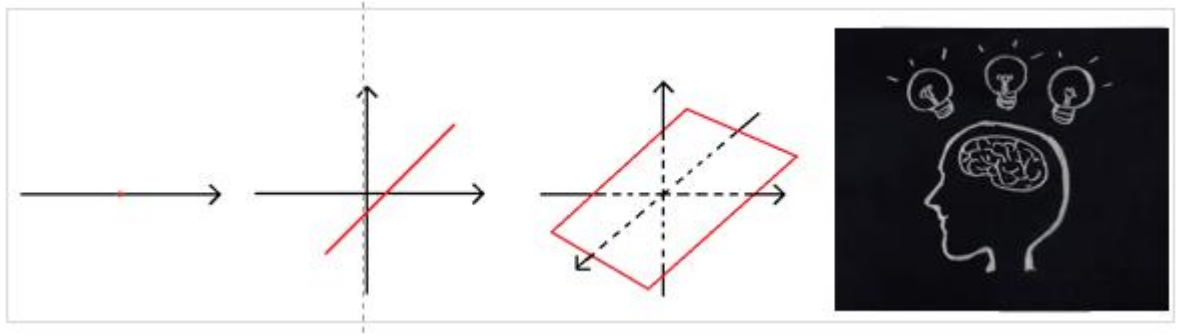
- Currently, definitions of the neural network are not unified. According to Hecht Nielsen, an American neural network scientist, a neural network is a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.
- Based on the source, features, and explanations of the neural network, **the artificial neural network (ANN) can be simply expressed as an information processing system designed to imitate the human brain structure and functions.**
- **ANN** is a network formed by artificial neurons connected to each other. It extracts and simplifies the human brain's microstructure and functions, and is an important approach to simulating human intelligence. It reflects several basic features of human brain functions, such as concurrent information processing, learning, association, model classification, and memory.

# Development History of Neural Networks



# Single-layer Perceptron

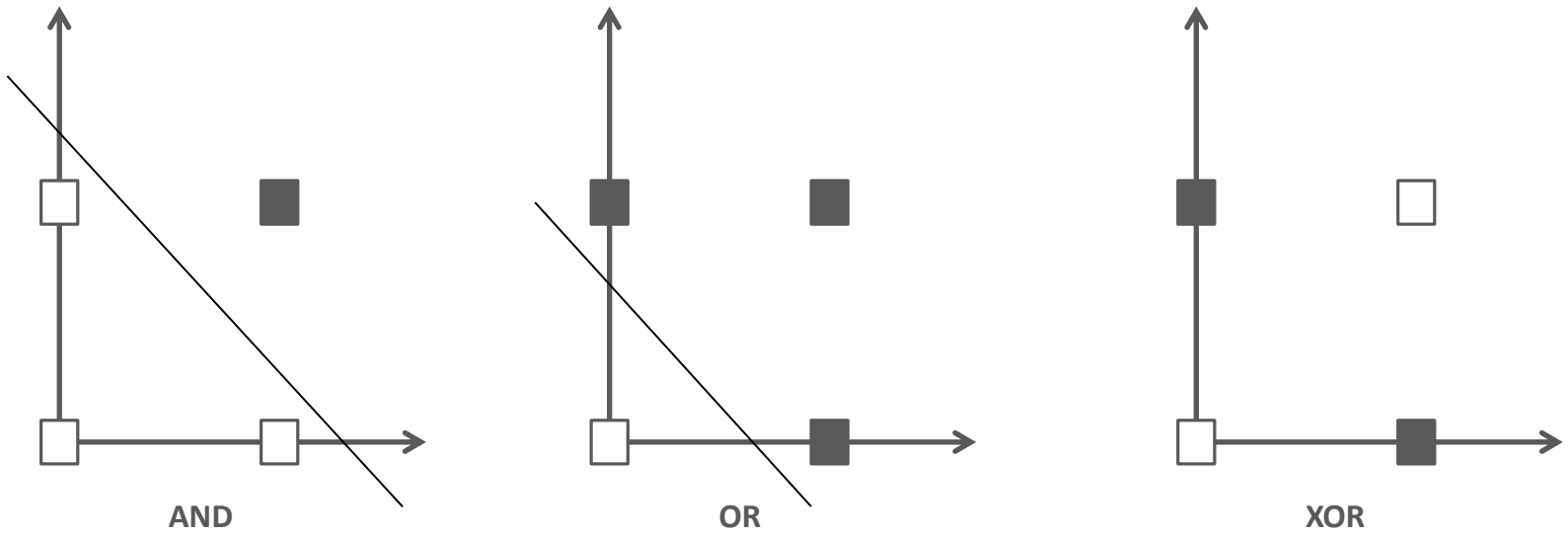
- **Input vector:** $X = [x_0, x_1, \dots, x_n]^T$.
- **Weight:** $W = [\omega_0, \omega_1, \dots, \omega_n]^T$, where $\omega_0$ indicates an offset.
- **Activation function:** $O = sign(net) = \begin{cases} 1, net > 0, \\ -1, otherwise. \end{cases}$
- The preceding perceptron is equivalent to a classifier. It uses the high-dimensional $X$ vector as the input and performs binary classification on input samples in the high-dimensional space. When $\boldsymbol{W^T X} > 0$, $O = 1$. In this case, the samples are classified into a type. Otherwise, $O = -1$. In this case, the samples are classified into the other type. The boundary of these two types is $\boldsymbol{W^T X} = 0$, which is a high-dimensional hyperplane.



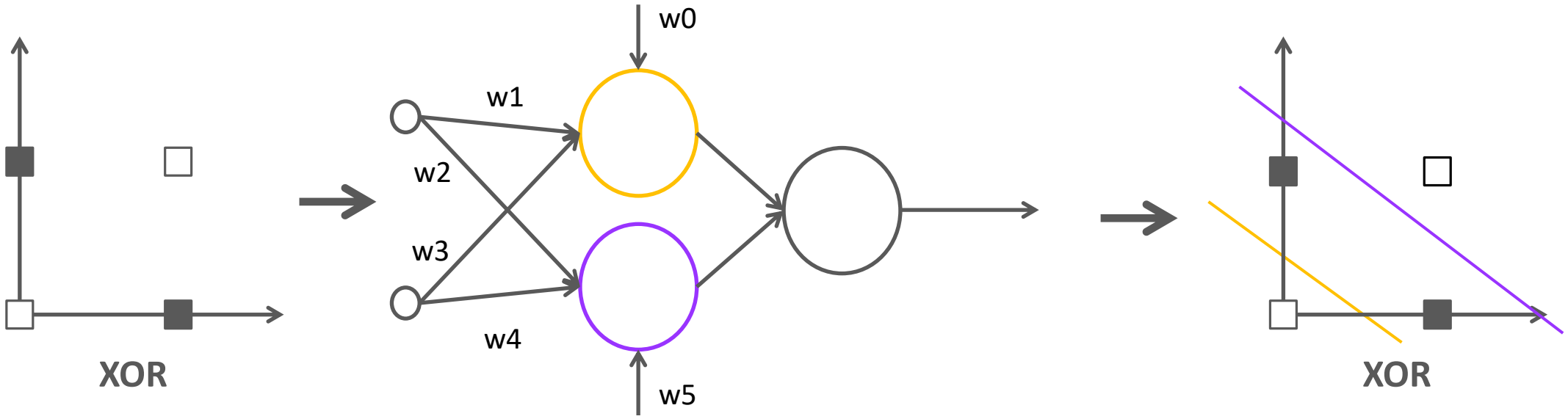
Classification point $Ax + B = 0$ | Classification line $Ax + By + C = 0$ | Classification plane $Ax + By + Cz + D = 0$ | Classification hyperplane $W^T X + b = 0$

# XOR Problem

- In 1969, Marvin Minsky, an American mathematician and AI pioneer, proved that perceptrons are essentially a type of linear model capable of processing only linear classification.

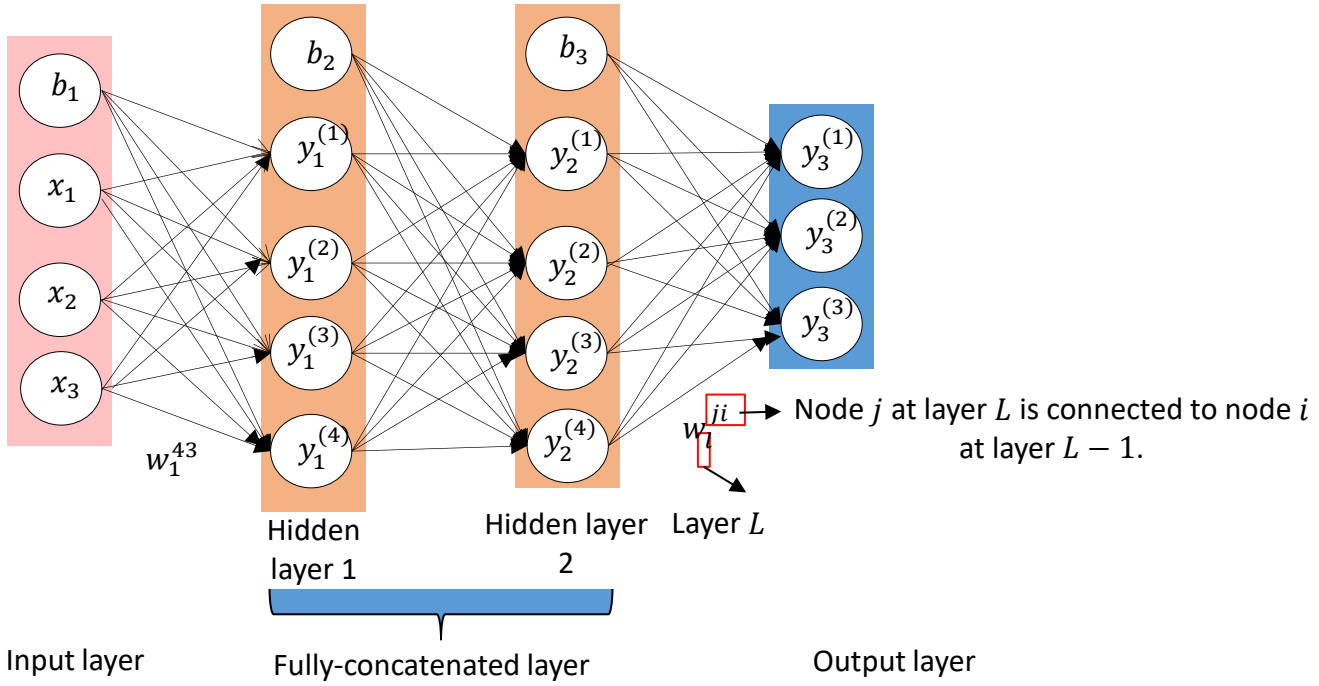


# Solving the XOR Problem

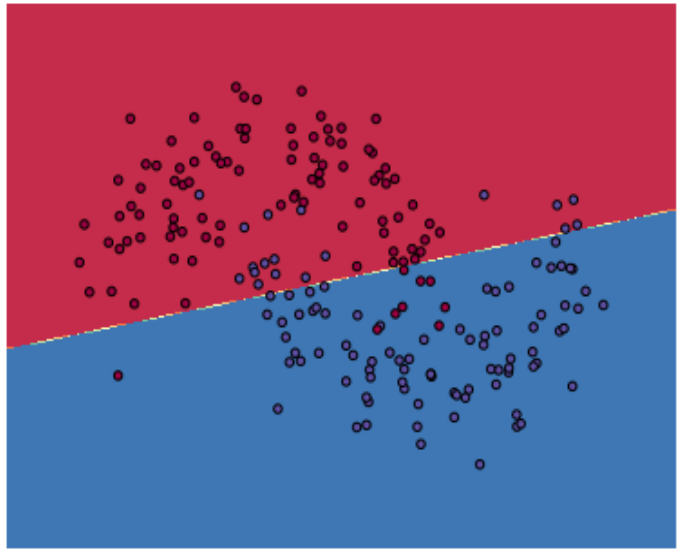


# Feedforward Neural Network (FNN)

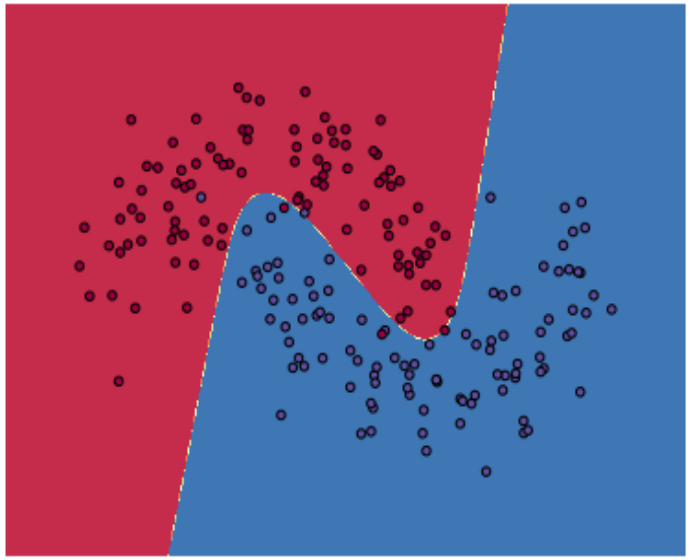
- An FNN is a typical deep learning model. It has the following features:
  - Input nodes do not provide the computing function and are used to represent only the element values of an input vector.
  - Each neuron is connected only to a neuron at the previous layer, receives an output of the previous layer as its input, and outputs the computing result to the next layer. A unidirectional multi-layer structure is used. There is no feedback in the entire network. Signals are transmitted unidirectionally from the input layer to the output layer. The network can be represented by a directed acyclic graph (DAG).



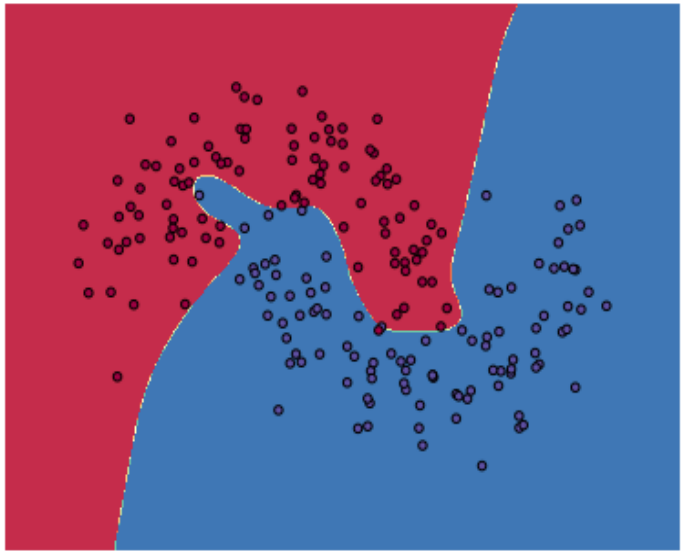
# Influence of Hidden Layers on Neural Networks



**0 hidden layers**



**3 hidden layers**



**20 hidden layers**

# Contents

# Common Loss Functions in Deep Learning

- When training a deep learning network, we need to parameterize the error of target classification. A **loss function (error function)** is used, which reflects the error between the target output and the actual output of the perceptron.
  - For **regression tasks**, the commonly used loss function is the **quadratic cost function**. For a separate training sample $x$, the quadratic cost function can be written as follows:

$$C(W) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$$

    $d$ indicates a neuron at the output layer, $D$ indicates all neurons at the output layer, $t_d$ indicates the target output, and $o_d$ indicates the actual output.
  - For **classification tasks**, the commonly used loss function is the **cross-entropy cost function**.

$$C(W) = -\frac{1}{n}\sum_{x}\sum_{d \in D}[t_d \ln o_d]$$

    The cross-entropy cost function depicts the distance between two probability distributions, which is a widely used loss function for classification problems.
- Generally, the quadratic cost function is used for regression problems, and the cross-entropy cost function is used for classification problems.

# Extremum of a Loss Function

- Purpose: A loss function can iteratively search for and update parameter $\boldsymbol{W}$ in the negative gradient direction to minimize the loss function.
- **Limitation:** There is no effective method for solving the extremum in mathematics on the complex high-dimensional surface of $C(W) = \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$.
- **Approach:** The core approach of the gradient descent method is as follows: The negative gradient direction is the fastest descent direction of the function. Therefore, the minimum point of $C(W)$ is expected to exist along the $-C()$ direction.

# Gradient Descent and Loss Function

- The gradient of a multivariable function $C(W) = f(w_0, w_1, \dots, w_n)$ at $W' = [w_0', w_1', \dots, w_n']^T$ is as follows:

$$\nabla f(w_0', w_1', \dots, w_n') = [\frac{\partial f}{\partial w_0}, \frac{\partial f}{\partial w_1}, \dots, \frac{\partial f}{\partial w_n}]^T|_{W=W'},$$

Direction of the gradient vector points to the direction in which the function grows fastest. Therefore, the direction of the negative gradient vector $-\nabla C$ points to the direction in which the function decreases fastest.

- The learning rate (LR) is a coefficient for adjusting weights according to an error gradient, and is usually denoted as $\eta$.

$$W_{t+1} = W_t - \eta \frac{dC}{dW}$$

Based on the learning rate and gradient value, updating all parameter values will reduce the network loss.

# Batch Gradient Descent (BGD) Algorithm

- $< X, t >$ indicates a sample in the training set. $X$ is an input value vector, $t$ is a target output, $o$ is an actual output, $\eta$ is a learning rate, and **C** is a loss function.
  - Initialize each $w_i$ to a random value with a smaller absolute value.
  - Before the termination condition is met, do as follows:
    - Initialize each $\Delta w_i$ to zero.
    - For each $< X, t >$ in the training set, do as follows:
      - Input $X$ to this unit and compute the output $o$.
      - For each $w_i$ in this unit: $\Delta w_i$+= -$\eta \frac{1}{n} \sum_x \sum_{d \in D} \frac{\partial C(t_d, o_d)}{\partial w_i}$
    - For each $w_i$ in this unit: $w_i$+= $\Delta w_i$
- The gradient descent algorithm of this version is not commonly used because it has the following defect:
  - The convergence process is very slow because all training samples need to be computed every time the weight is updated.

# Stochastic Gradient Descent (SGD) Algorithm

- To address the defect of the BGD algorithm, a common variant is developed, which is called incremental gradient descent or stochastic gradient descent. One implementation is called online learning, which updates the gradient based on each sample:

$$\Delta w_i = -\eta \frac{1}{n} \sum_{\mathrm{x}} \sum_{d \in D} \frac{\partial \mathrm{C}(t_d, o_d)}{\partial w_i} \Rightarrow \Delta w_i = -\eta \sum_{d \in D} \frac{\partial \mathrm{C}(t_d, o_d)}{\partial w_i}$$

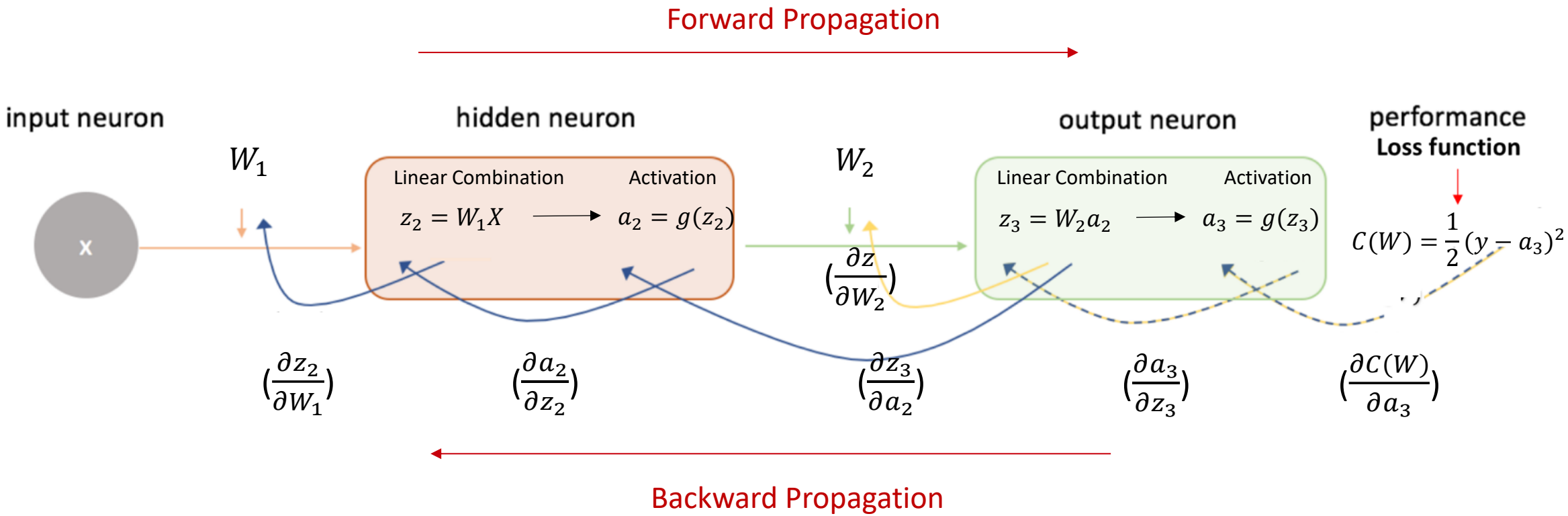
- ONLINE-GRADIENT-DESCENT
  - Initialize each $w_i$ to a random value with a smaller absolute value.
  - Before the termination condition is met, do as follows:
  - Randomly select <X, t> in the training set:
    - Input X to this unit and compute the output o.
    - For each $w_i$ in this unit: $w_i$ += -η$\sum_{d \in D} \frac{\partial \mathrm{C}(t_d, o_d)}{\partial w_i}$

# Mini-batch Gradient Descent (MBGD) Algorithm

- To address the defects of the previous two gradient descent algorithms, the MBGD algorithm was proposed and has been most widely used. It uses a small batch of samples with a fixed batch size to compute $\Delta w_i$ and update weights.
- BATCH-GRADIENT-DESCENT
  - Initialize each $w_i$ to a random value with a smaller absolute value.
  - Before the termination condition is met, do as follows:
    - Initialize each $\Delta w_i$ to zero.
    - For each $< X, t >$ in the samples obtained from the training set, do as follows:
      - Input $X$ to this unit and compute the output $o$.
      - For each $w_i$ in this unit: $\Delta w_i$ += $-\eta \frac{1}{n} \sum_{\mathrm{x}} \sum_{d \in D} \frac{\partial \mathrm{C}(t_d, o_d)}{\partial w_i}$
    - For each $w_i$ in this unit: $w_i$ += $\Delta w_i$
    - If it is the last batch, shuffle the training samples.

# Network Training Process

- Forward propagation: $C(W) = \frac{1}{2}(y - a_3)^2 = \frac{1}{2}(y - (g(W_2 g(W_1 X))))^2$.
- If the parameter $W_1$ needs to be updated, according to $W_{t+1} = W_t - \eta \frac{dC}{dW}$, compute: $\frac{dC}{dW_1} = \frac{\partial C(W)}{\partial a_3}\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial W_1}$.



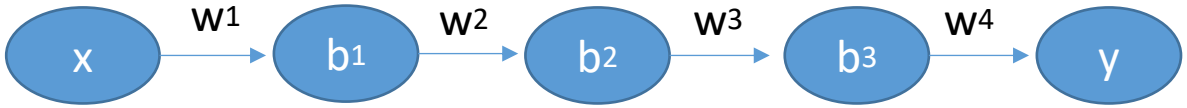
# Backward Propagation

- Error backward propagation is an important algorithm in neural networks. It uses the **chain rule** to send the errors of the output layer back to the network, so that a gradient with respect to the weights of the neural network can be easily computed. The steps are as follows:
  - Backward propagate the loss function values to each compute unit.
  - Each compute unit updates the weight according to the obtained errors.

# Vanishing and Exploding Gradients (1)

- Vanishing gradient: As network layers increase, the derivative value of backward propagation decreases and the gradient vanishes.
- Exploding gradient: As network layers increase, the derivative value of backward propagation increases and the gradient explodes.
- Cause:

$$y_i = \sigma_{(z_i)} = \sigma(w_i x_i + b_i),$$ where $\sigma$ is a sigmoid function.

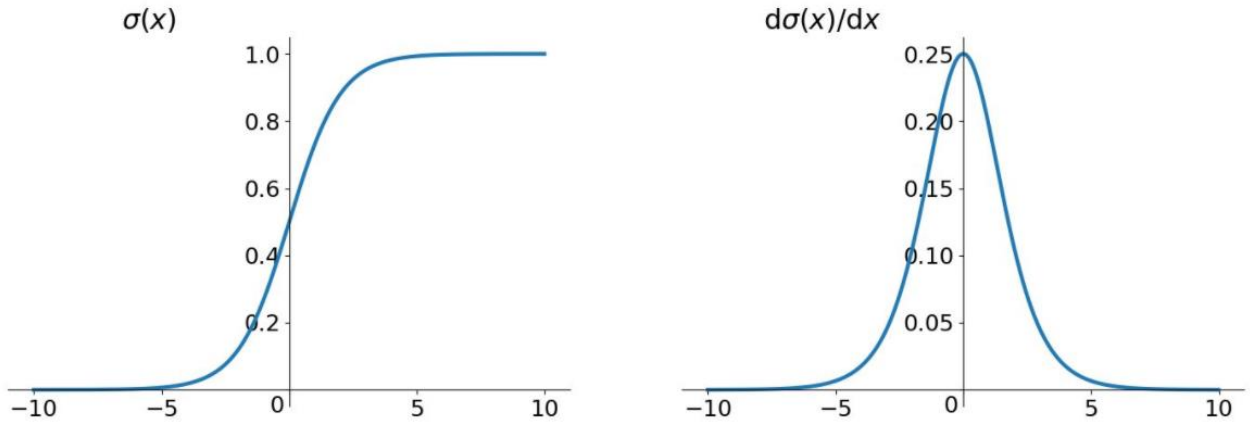



- Backward propagation can be deduced as follows:

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial y_4}\frac{\partial y_4}{\partial z_4}\frac{\partial z_4}{\partial x_4}\frac{\partial x_4}{\partial z_3}\frac{\partial z_3}{\partial x_3}\frac{\partial x_3}{\partial z_2}\frac{\partial z_2}{\partial x_2}\frac{\partial x_2}{\partial z_1}\frac{\partial z_1}{\partial b_1}$$
$$= \frac{\partial C}{\partial y_4}\sigma'(z_4)w_4\sigma'(z_3)w_3\sigma'(z_2)w_2\sigma'(z_1)x$$

# Vanishing and Exploding Gradients (2)

- The maximum value of $\sigma'(x)$ is $\frac{1}{4}$:

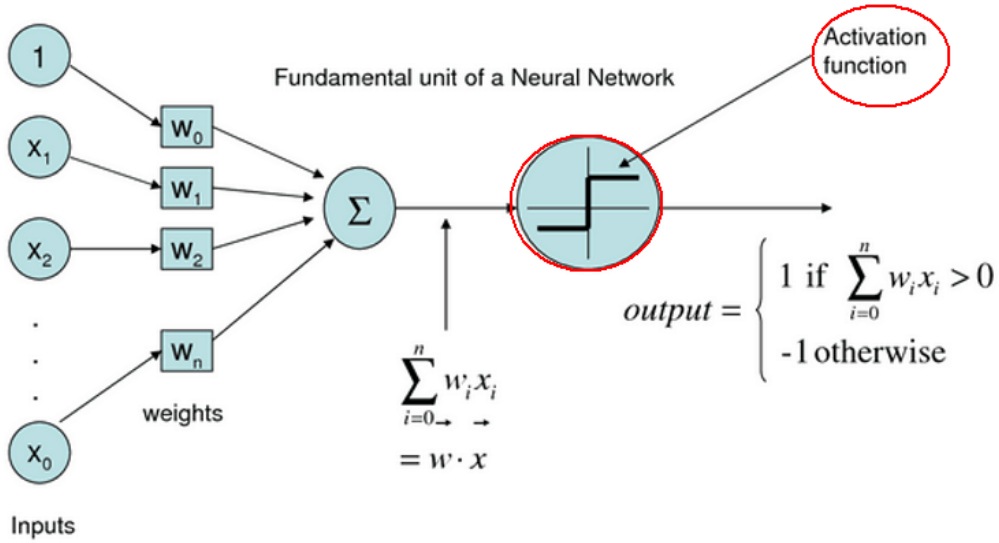



- The network weight $|w|$ is usually less than 1. Therefore, $|\sigma'(z)w| \leq \frac{1}{4}$. When using the chain rule, the value of $\frac{\partial C}{\partial b_1}$ becomes smaller when the number of layers increases, resulting in the vanishing gradient problem.
- When the network weight $|w|$ is large, that is $|\sigma'(z)w| > 1$, the exploding gradient problem occurs.
- Solutions: The ReLU activation function and LSTM neural network are used to solve the vanishing gradient problem, and gradient clipping is used to solve the exploding gradient problem.

# Contents

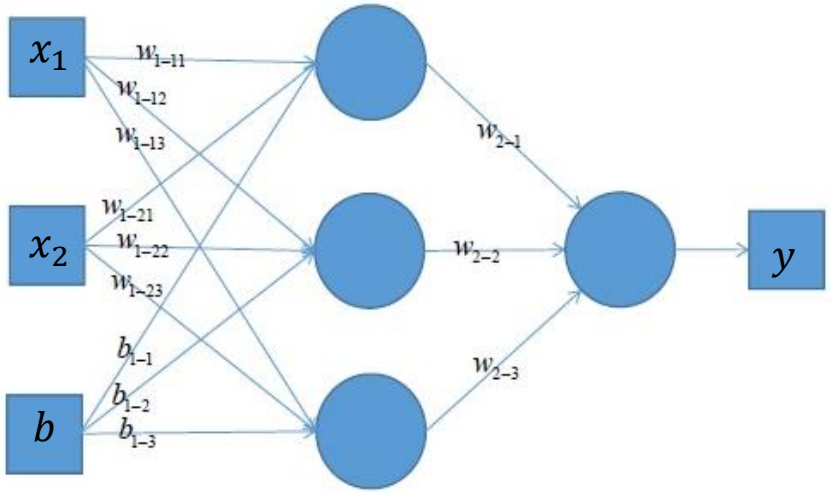
# Concept of Activation Functions

- In a neural network, each neuron receives the outputs of neurons at the previous layer as its inputs and then transmits the inputs to the next layer. Neurons at the input layer directly transmit input attribute values to the next layer (a hidden or output layer). In a multi-layer neural network, there is a function between outputs of nodes at the previous layer and inputs of nodes at the next layer. Such function is referred to as an **activation function**.



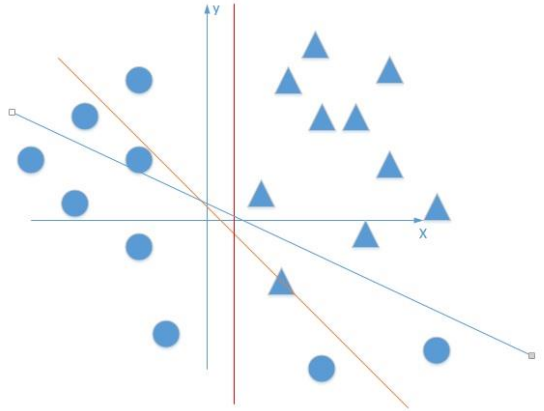

Single-layer Perceptron

# Purpose of Activation Functions (1)




$$y = w_{2\text{-}1}(w_{1\text{-}11}x_1 + w_{1-21}x_2 + b_{1\text{-}1}) + w_{2\text{-}2}(w_{1\text{-}12}x_1 + w_{1-22}x_2 + b_{1\text{-}2}) + w_{2\text{-}3}(w_{1\text{-}13}x_1 + w_{1-23}x_2 + b_{1\text{-}3})$$

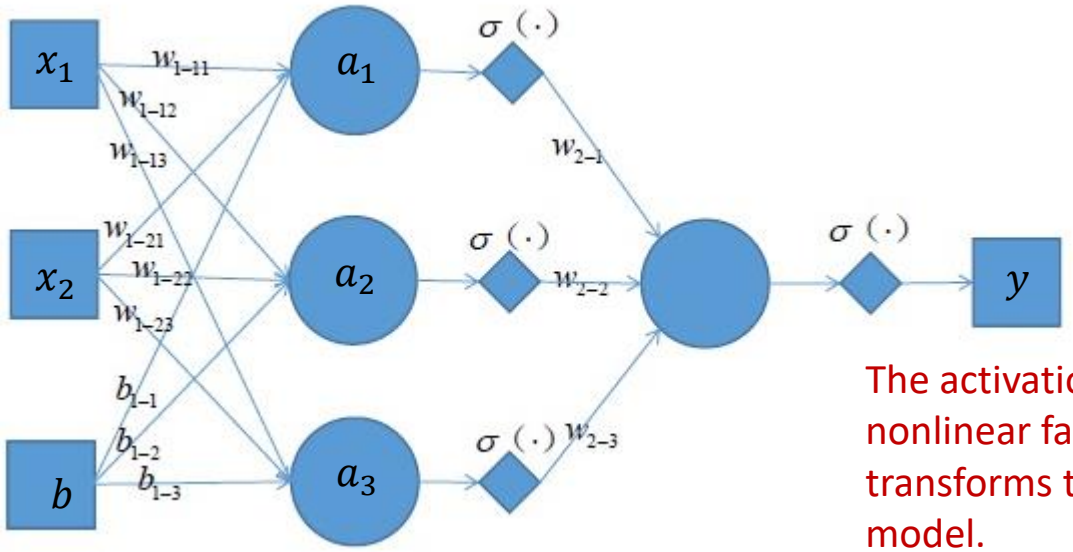
Multiple perceptrons without activation functions are equivalent to linear functions.






HUAWEI

# Purpose of Activation Functions (2)



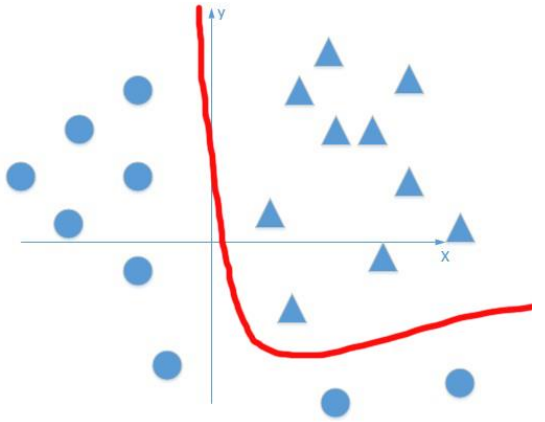

$$a1 = w_{1\text{-}11}x_1 + w_{1-21}x_2 + b_{1\text{-}1}$$

$$a2 = w_{1\text{-}12}x_1 + w_{1-22}x_2 + b_{1\text{-}2}$$

$$a3 = w_{1\text{-}13}x_1 + w_{1-23}x_2 + b_{1\text{-}3}$$

$$y = \sigma(w_{2-1}\sigma\ (a1) + w_{2-2}\sigma\ (a2) + w_{2-3}\sigma\ (a3))$$

The activation function is equivalent to adding nonlinear factors to a neural network, which transforms the neural network into a nonlinear model.

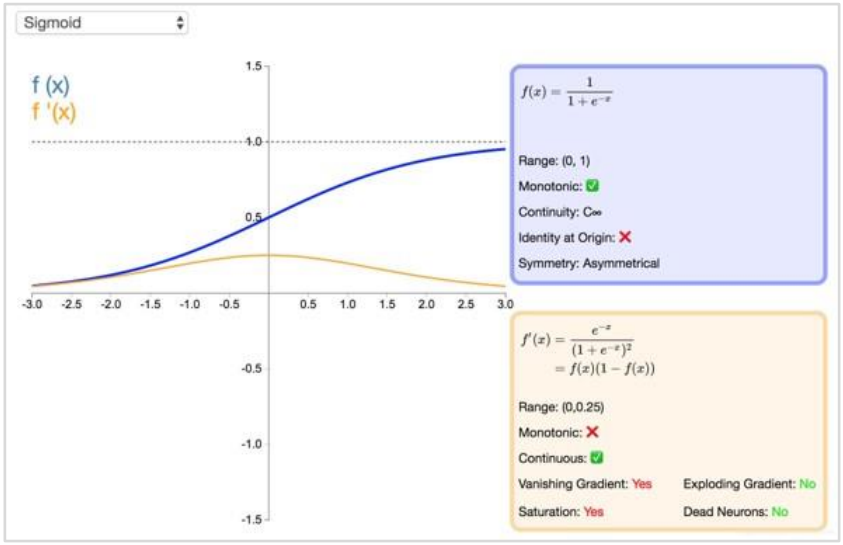





HUAWEI

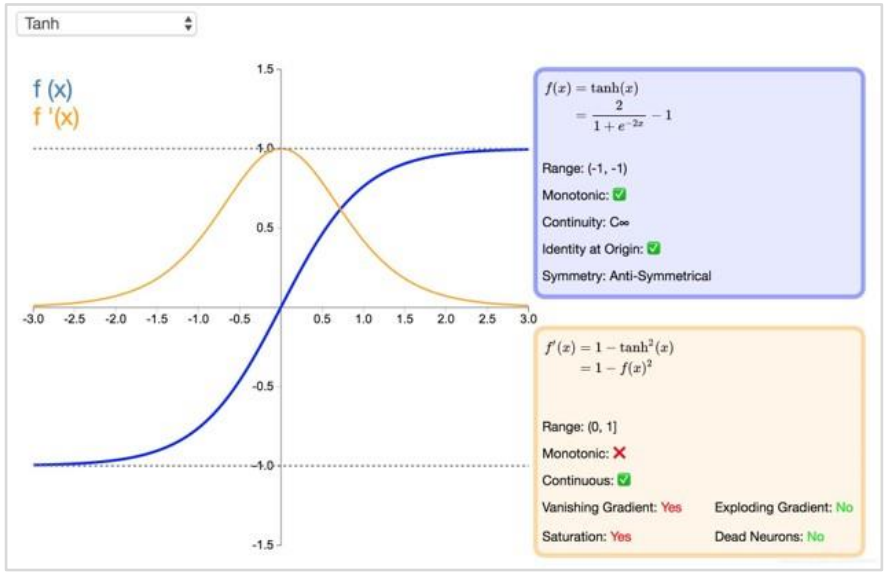
# Sigmoid Function

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



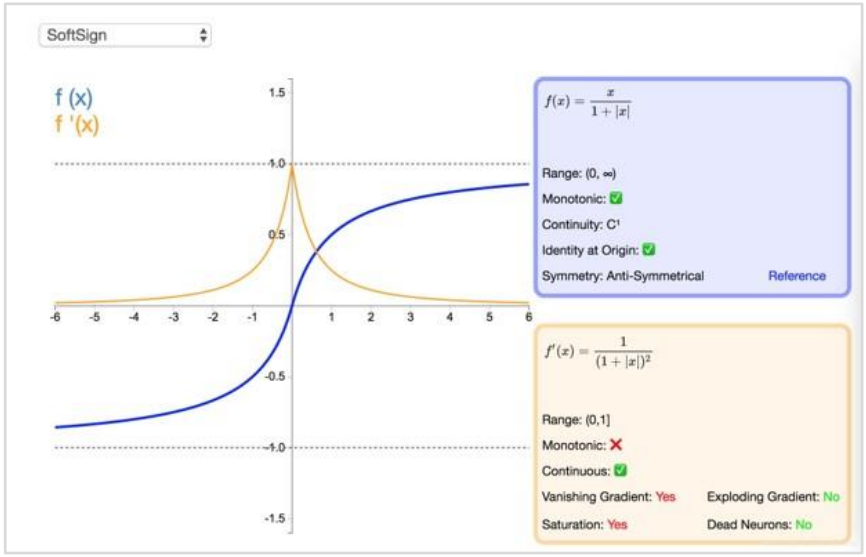
# Tanh Function

$$\tanh x = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



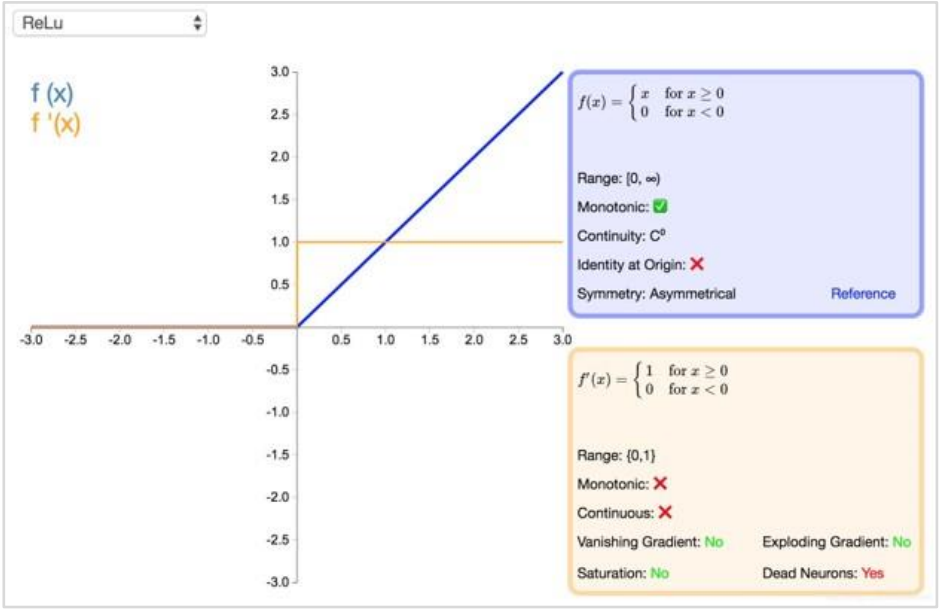
# Softsign Function

$$f(x) = \frac{x}{|x| + 1}$$



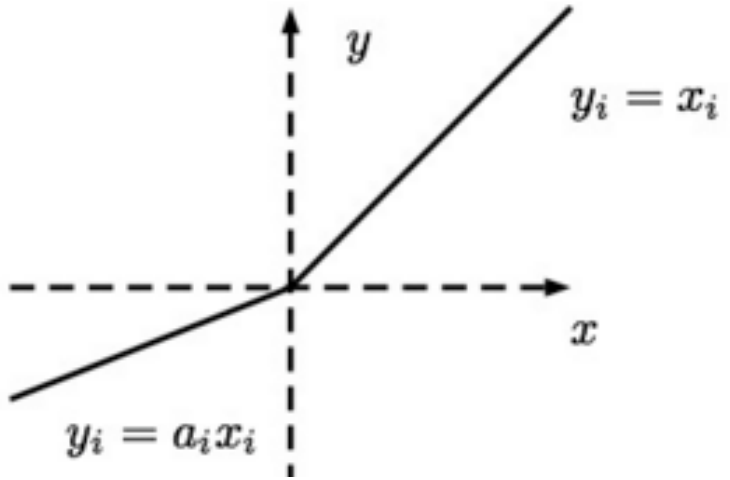
# Rectified Linear Unit (ReLU) Function

$$y = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$



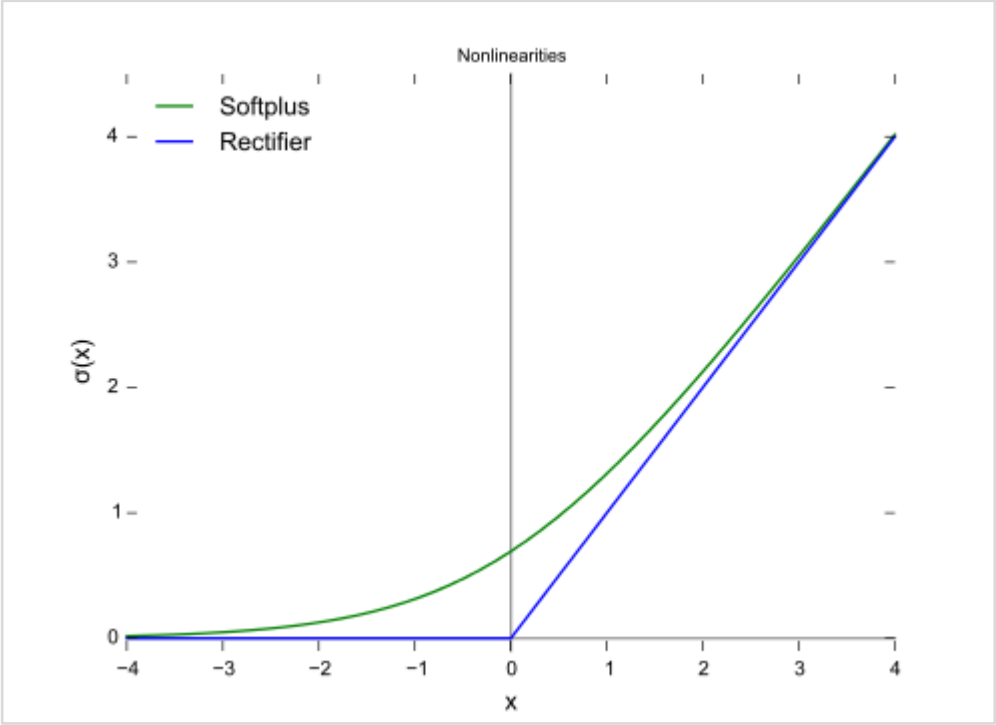
# LeakyRelu Function

$$y_i = \begin{cases} x_i & \text{if } x_i \geq 0 \\ \frac{x_i}{a_i} & \text{if } x_i < 0, \end{cases}$$



# Softplus Function

$$f(x) = \ln(e^x + 1)$$

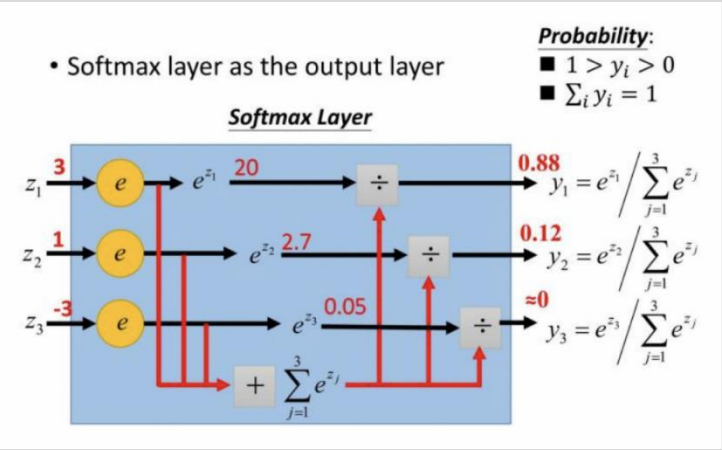


# Softmax Function

- Softmax function body:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

- The Softmax function is used to map a K-dimensional vector of arbitrary real values to another K-dimensional vector of real values, where each vector element is in the interval (0, 1). All the elements add up to 1.
- The softmax function is often used as the output layer of a multiclass classification task.



# Contents

# Overfitting

- Problem description: The model performs well in the training set, but poorly in the test set.
- Root cause: There are too many feature dimensions, model assumptions, and parameters, too much noise, but very less training data. As a result, the fitting function almost perfectly predicts the training set, whereas the prediction result of the test set of new data is poor. Training data is overfitted without considering the generalization capability of the model.

# Normalization

- Normalization is a very important and effective technology to reduce generalization errors in machine learning. It is especially useful for deep learning models which tend to have overfitting due to diverse parameters. Therefore, researchers have also proposed many effective technologies to prevent overfitting, including:
  - Adding constraints to parameters, such as $L_1$ and $L_2$ norms
  - Expanding the training set, such as adding noise and transforming data
  - Dropout
  - Early stopping

# Parameter Penalty

- Many normalization methods limit the learning ability of the model by adding a parameter penalty $\Omega(\theta)$ to the objective function $J$. Assume that the normalized objective function is $\tilde{J}$.

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta),$$

where $\alpha\epsilon[0, \infty)$ is a hyperparameter that weighs the relative contribution of the norm penalty term $\Omega$ and the standard objective function $J(X; \theta)$. If $\alpha$ is set to 0, no normalization is performed. A larger value of $\alpha$ indicates a larger normalization penalty.

# $L_1$ Normalization

- Add $L_1$ norm constraints to model parameters, that is:

$$\tilde{J}(w; X, y) = J(w; X, y) + \alpha \|w\|_1,$$

- If a gradient method is used to resolve the value, the parameter gradient is:

$$\nabla \tilde{J}(w) = \alpha sign(w) + \nabla J(w).$$

- The parameter optimization method is:

$$w_{t+1} = (w - \varepsilon \alpha sign(w)) - \varepsilon \nabla J(w)$$

# $L_2$ Normalization

- The $L_2$ norm penalty term is added to the parameter constraint. This technique is used to prevent overfitting.

$$\tilde{J}(w; X, y) = J(w; X, y) + \frac{1}{2}\alpha\|w\|_2^2,$$

A parameter optimization method can be inferred using an optimization technology (such as a gradient method):

$$w = (1 - \varepsilon\alpha)w - \varepsilon\nabla J(w),$$

where $\varepsilon$ is the learning rate. Compared with the normal gradient optimization formula, the parameter is multiplied by a reduction factor.

# $L_2$ vs. $L_1$

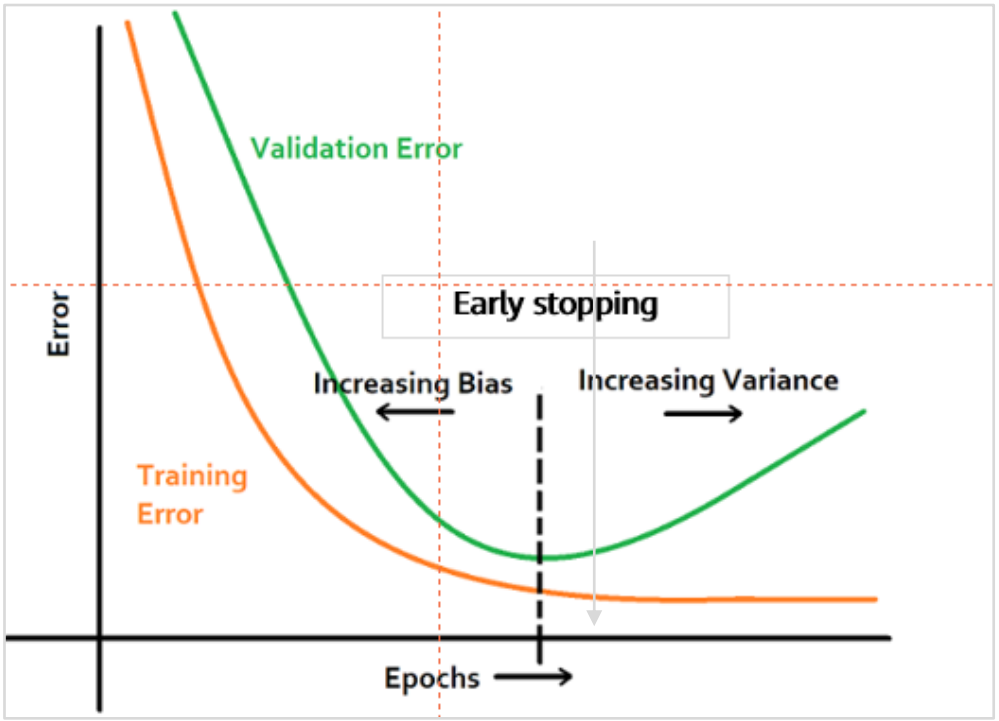
- The differences between $L_2$ and $L_1$ are as follows:
  - According to the preceding analysis, $L_1$ can generate a sparser model than $L_2$. When the value of parameter $w$ is small, $L_1$ normalization can directly reduce the parameter value to 0, which can be used for feature selection.
  - From the perspective of probability, many norm constraints are equivalent to adding prior probability distribution to parameters. In $L_2$ normalization, the parameter value complies with the Gaussian distribution rule. In $L_1$ normalization, the parameter value complies with the Laplace distribution rule.

# Data Augmentation

- The most effective way to prevent overfitting is to add a training set. A larger training set has a smaller overfitting probability. Data augmentation is time-saving and effective but not applicable in certain fields.
  - A common method in the object recognition field is to rotate or scale images. (The prerequisite to image transformation is that the type of the image should not be changed through transformation. For example, for handwriting digit recognition, classes 6 and 9 can be easily changed after rotation).
  - Random noise is added to the input data in speech recognition.
  - A common practice of natural language processing (NLP) is replacing words with their synonyms.

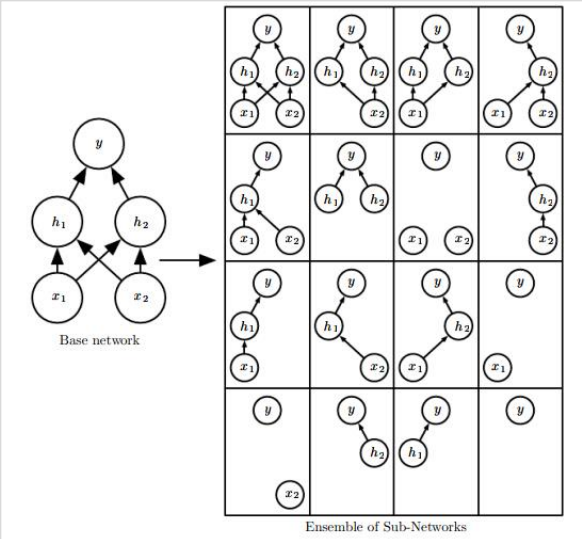
# Early Stopping of Training

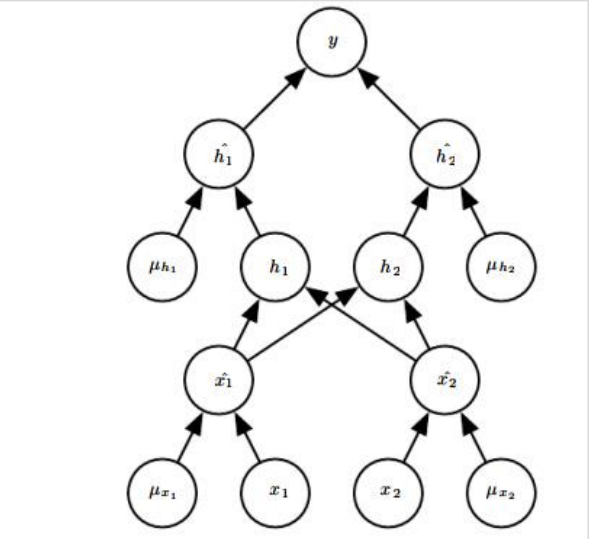
- A test on data of the validation set can be inserted during the training. When the data loss of the validation set increases, the training is stopped in advance.



# Dropout

- Dropout is a common and simple normalization method, which has been widely used since 2014. Simply put, dropout randomly discards some inputs during the training process. In this case, the parameters corresponding to the discarded inputs are not updated. Dropout is an integration method. It combines all sub-network results and obtains sub-networks by randomly dropping inputs. For example:



# Contents

# Optimizers

- There are various improved versions of gradient descent algorithms. In object-oriented language implementation, different gradient descent algorithms are often encapsulated into objects called **optimizers**.
- The **purposes** of the algorithm optimization include but are not limited to:
  - Accelerating algorithm convergence.
  - Preventing or jumping out of local extreme values.
  - Simplifying manual parameter setting, especially the learning rate (LR).
- Common optimizers: common GD optimizer, **momentum optimizer**, Nesterov, **AdaGrad**, AdaDelta, **RMSProp**, **Adam**, AdaMax, and Nadam.

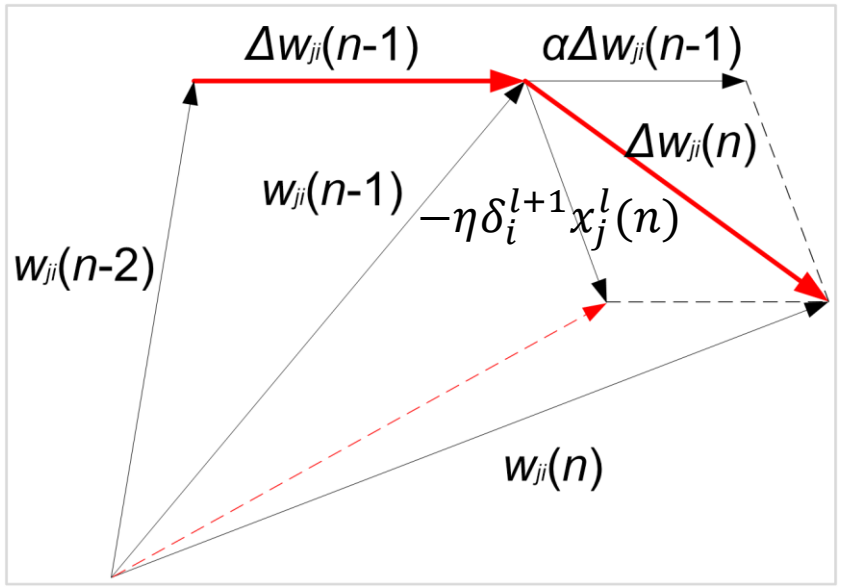
# Momentum Optimizer

- A most basic improvement is to add momentum terms for $\Delta w_{ji}$. Assume that the weight correction of the $n$-th iteration is $\Delta w_{ji}(n)$ . The weight correction rule is:

$$\Delta w_{ji}^{l}(n) = -\eta \delta_i^{l+1} x_j^l(n) + \alpha \Delta w_{ji}^{l}(n-1)$$

Where α is a constant (0 ≤ α < 1) called momentum coefficient and $\alpha \Delta w_{ji}^{l}(n-1)$ is a momentum term.

- Imagine a small ball rolls down from a random point on the error surface. The introduction of the momentum term is equivalent to giving the small ball inertia.



# Advantages and Disadvantages of Momentum Optimizer

- Advantages:
  - Enhances the stability of the gradient correction direction and reduces mutations.
  - In areas where the gradient direction is stable, the ball rolls faster and faster (there is a speed upper limit because $\alpha < 1$), which helps the ball quickly overshoot the flat area, and accelerates convergence.
  - A small ball with inertia is more likely to roll over some narrow local extrema.
- Disadvantages:
  - The learning rate $\eta$ and momentum $\alpha$ need to be manually set, which often requires more experiments to determine the appropriate value.

# AdaGrad Optimizer (1)

- The common feature of the stochastic gradient descent (SGD) algorithm, small-batch gradient descent algorithm (MBGD), and momentum optimizer is that each parameter is updated with the **same LR**.
- According to the approach of AdaGrad, different learning rates need to be set for different parameters.

$$g_t = \frac{\partial C(t,o)}{\partial w_t}$$ Computing gradient

$$r_t = r_{t-1} + g_t^2$$ Square gradient accumulation

$$\Delta w_t = -\frac{\eta}{\varepsilon + \sqrt{r_t}} g_t$$ Computation update

$$w_{t+1} = w_t + \Delta w_t$$ Application update

- $g_t$ indicates the $t$th gradient, $r_t$ is a gradient accumulation variable, and the initial value of $r$ is 0, which **increases continuously**. $\eta$ indicates the global LR, which needs to be set manually. $\varepsilon$ is a small constant, and is set to about $10^{-7}$ for numerical stability.

# AdaGrad Optimizer (2)

- It can be seen from the AdaGrad optimization algorithm that, as the algorithm is continuously iterated, the $r$ becomes larger and the overall learning rate becomes smaller. This is because we hope the **LR becomes slower** as the number of updates increases. In the initial learning phase, we are far away from the optimal solution to the loss function. As the number of updates increases, we are closer and closer to the optimal solution, and therefore LR can decrease.
- Advantages:
  - The learning rate is automatically updated. As the number of updates increases, the learning rate decreases.
- Disadvantages:
  - The denominator keeps accumulating. As a result, the learning rate will eventually become very small and the algorithm will become ineffective.

# RMSProp Optimizer

- The RMSProp optimizer is an improved AdaGrad optimizer. It introduces an attenuation coefficient to ensure **a certain attenuation ratio** for $r$ in each round.
- The RMSProp optimizer solves the problem of the AdaGrad optimizer ending the optimization process too early. It is suitable for non-stable target handling and has good effects on RNNs.

$$g_t = \frac{\partial C(t,o)}{\partial w_t}$$ Computing gradient

$$r_t = \beta r_{t-1} + (1-\beta) g_t^2$$ Square gradient accumulation

$$\Delta w_t = -\frac{\eta}{\varepsilon + \sqrt{r_t}} g_t$$ Computation update

$$w_{t+1} = w_t + \Delta w_t$$ Application update

- $g_t$ indicates the $t$th gradient, $r_t$ is a gradient accumulation variable, and the initial value of $r$ is 0, **which may not increase and may need to be adjusted using a parameter.** $\beta$ indicates the decay factor, and $\eta$ indicates the global LR, which needs to be set manually. $\varepsilon$ is a small constant, and is set to about $10^{-7}$ for numerical stability.

# Adam Optimizer (1)

- Adaptive moment estimation (Adam): Developed based on AdaGrad and AdaDelta, Adam maintains two additional variables $m_t$ and $v_t$ for each variable to be trained:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

where $t$ represents the $t$-th iteration and $g_t$ is the computed gradient. $m_t$ and $v_t$ are moving averages of the gradient and square gradient. From the statistical perspective, $m_t$ and $v_t$ are estimates of the first moment (the average value) and the second moment (the uncentered variance) of the gradients respectively, hence the name of the method.

# Adam Optimizer (2)

- If $m_1$ and $v_1$ are initialized using the zero vector, $m_t$ and $v_t$ are close to 0 during the initial iterations, especially when $\beta_1$ and $\beta_2$ are close to 1. To solve this problem, we use $\hat{m}_t$ and $\hat{v}_t$:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

- The weight update rule of Adam is as follows:

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

- Although the rule involves manual setting of $\eta$, $\beta_1$, and $\beta_2$, the setting is much simpler. According to experiments, the default settings are $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, and $\eta$ = 0.001. In practice, Adam will converge quickly. When convergence saturation is reached, xx can be reduced. After several times of reduction, a satisfying local extrema will be obtained. Other parameters do not need to be adjusted.

# Contents

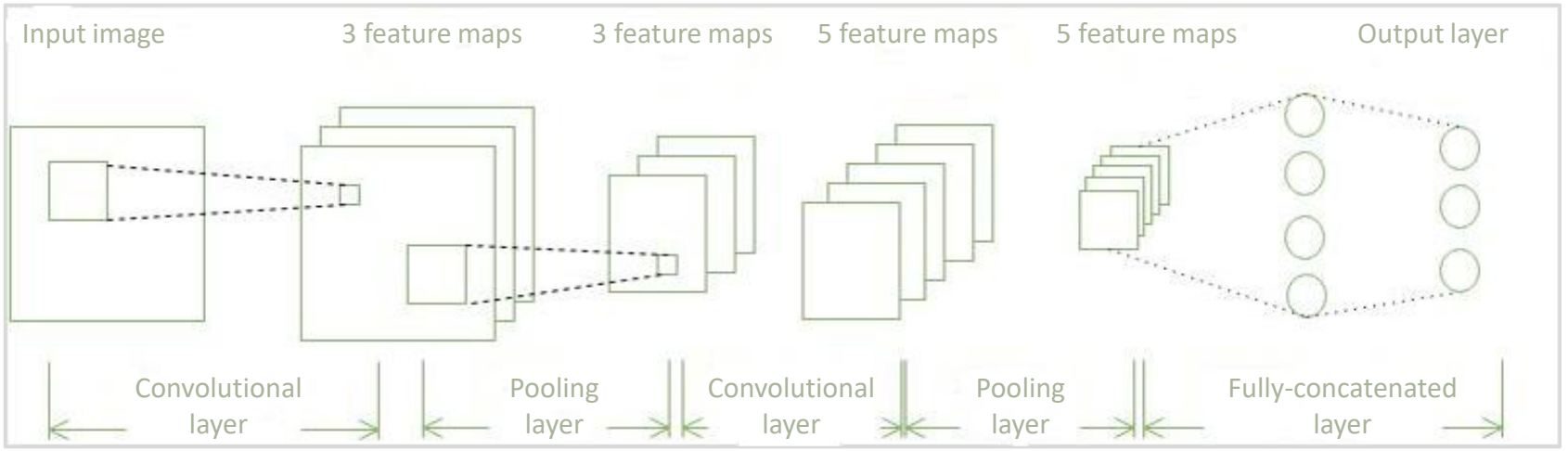
# Convolutional Neural Network (CNN)

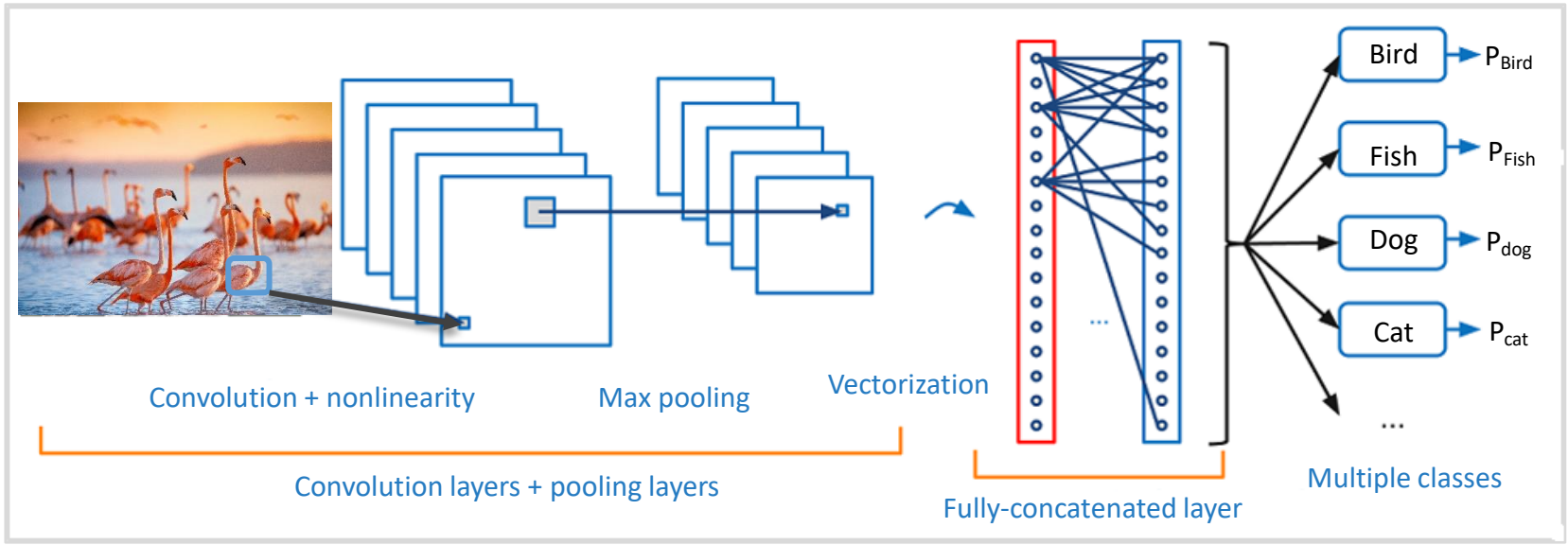
- A CNN is a feedforward neural network. Its artificial neurons may respond to **units within the coverage range**. CNN excels at image processing. It includes a **convolutional layer**, a **pooling layer**, and a **fully-connected layer**.
- In the 1960s, Hubel and Wiesel studied cats' cortex neurons used for local sensitivity and direction selection and found that their unique network structure could simplify feedback neural networks. They then proposed the CNN.
- Now, CNN has become one of the research hotspots in many scientific fields, especially in the pattern classification field. The network is widely used because it can avoid complex pre-processing of images and directly input original images.

# Main Concepts of CNN

- **Local receptive field:** It is generally considered that human perception of the outside world is from local to global. **Spatial correlations among local pixels of an image are closer than those among pixels that are far away**. Therefore, each neuron does not need to know the global image. It only needs to know the local image and the local information is then combined at a higher level to generate global information.
- **Parameter sharing:** One or more convolution cores may be used to scan input images. Parameters carried by the convolution cores are weights. In a layer scanned by convolution cores, each core uses the same parameters during weighted computation. Weight sharing means that when each convolution core scans an entire image, **the parameters of the convolution core are fixed.**

# CNN Architecture



# Computing a Convolution Kernel (1)

- Convolution computation

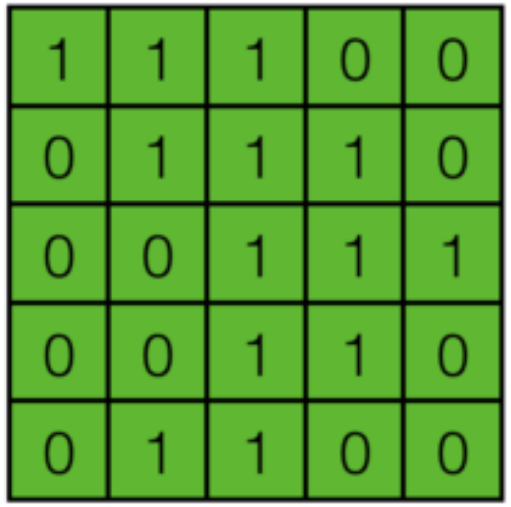
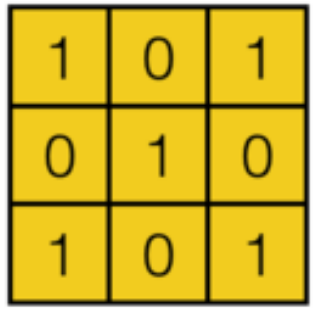
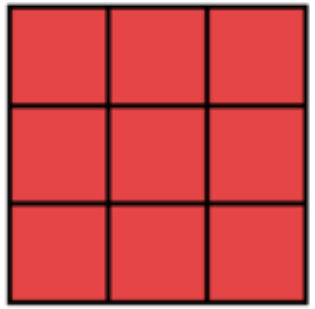



image 5*5




bias=0

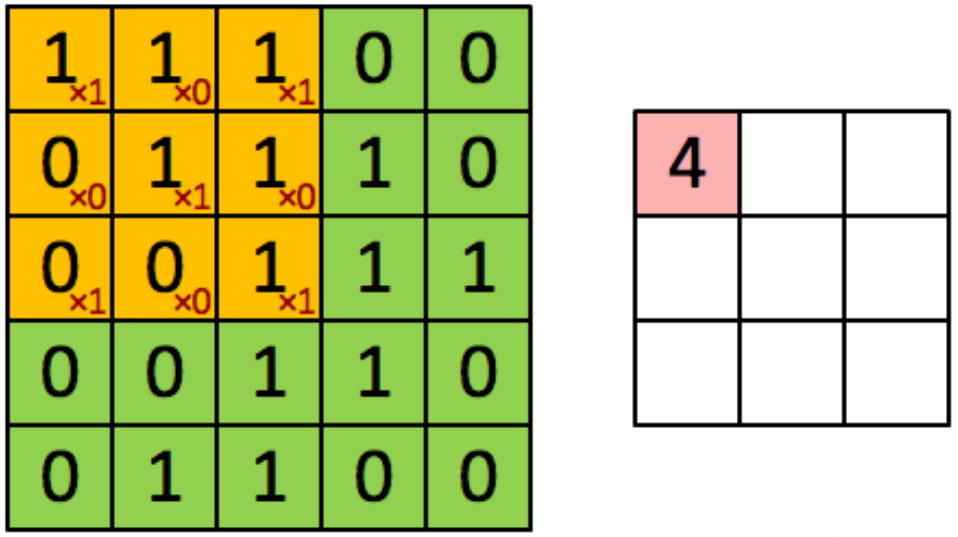
filter 3*3



feature map 3*3

# Computing a Convolution Kernel (2)

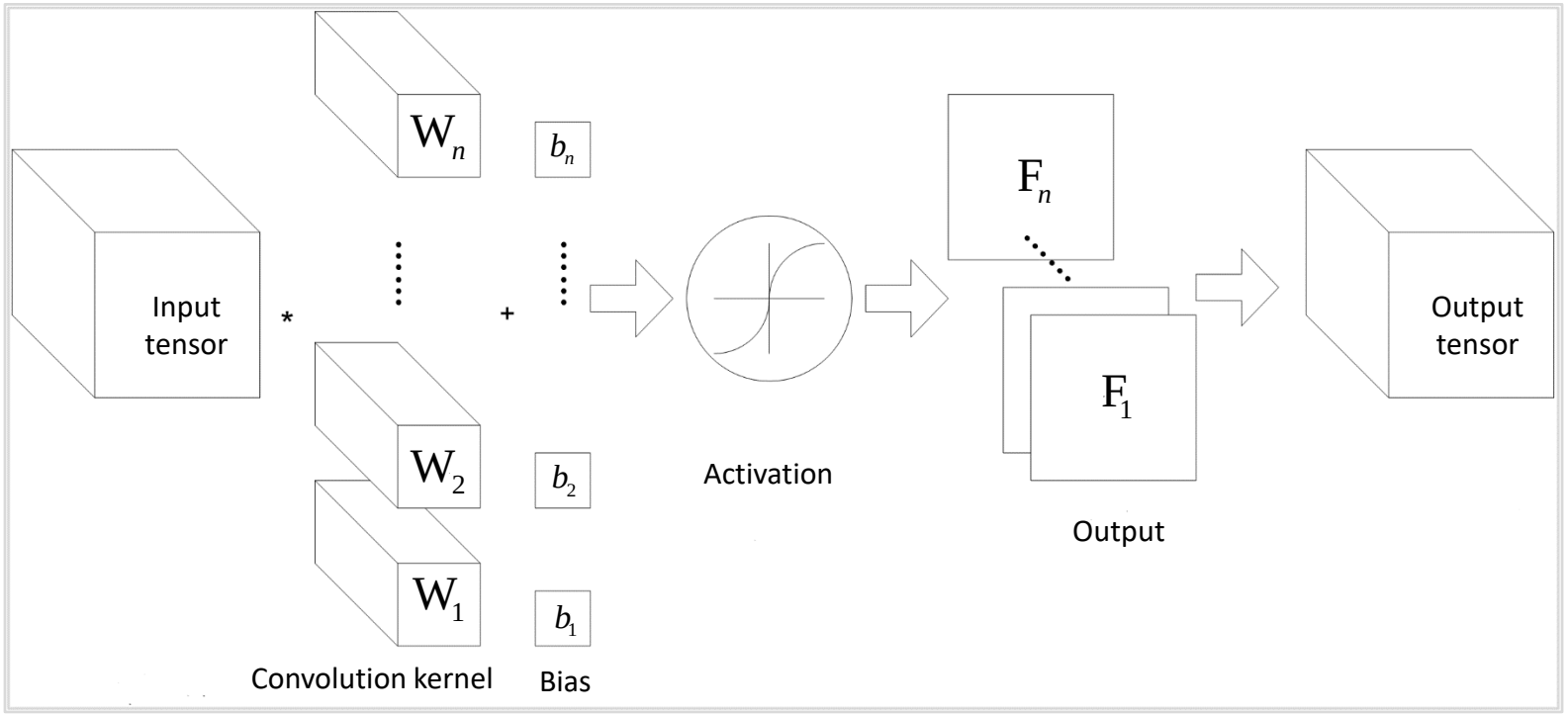
- Convolution computation result




hanbingtao, 2017, CNN

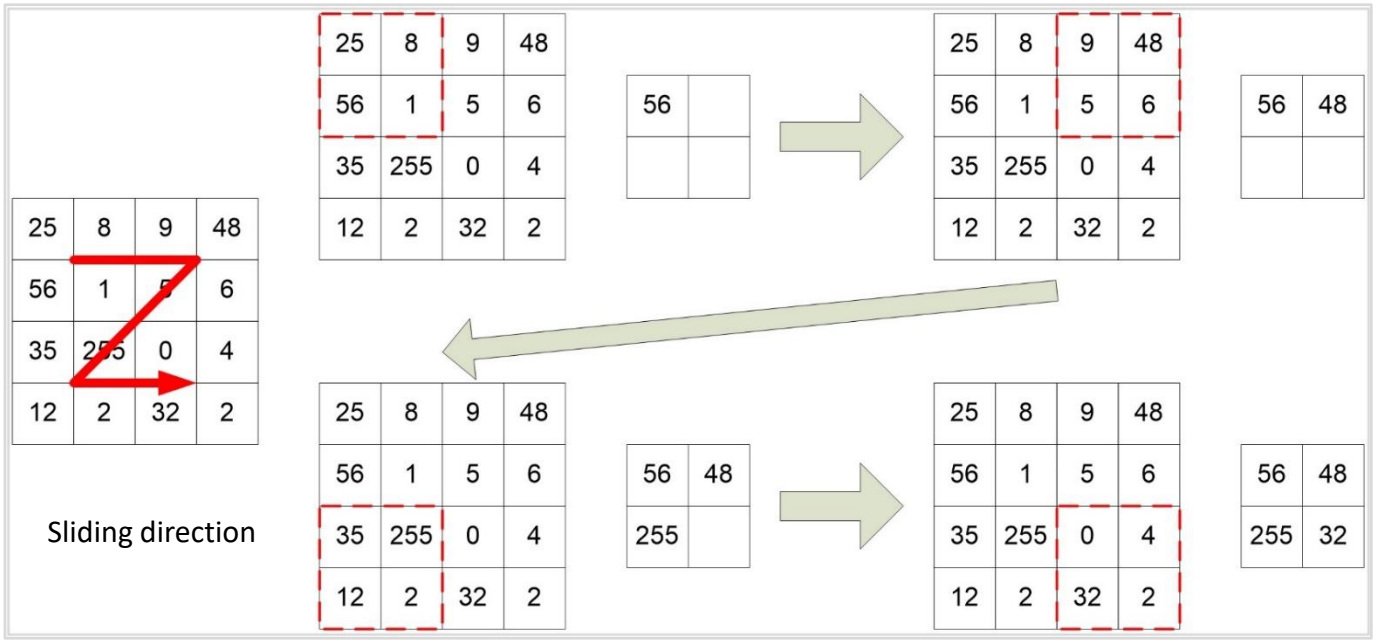
# Convolutional Layer

- The basic architecture of a CNN is a multi-channel convolution consisting of multiple single convolutions. The output of the previous layer (or the original image of the first layer) is used as the input of the current layer. It is then convolved with the convolution core of the layer and serves as the output of this layer. The convolution kernel of each layer is the weight to be learned. Similar to the fully-connected layer, after the convolution is complete, the result is biased and activated through activation functions before being input to the next layer.



# Pooling Layer

- Pooling combines nearby units to reduce the size of the input on the next layer, **reducing dimensions**. Common pooling includes max pooling and average pooling. When max pooling is used, the maximum value in a small square area is selected as the representative of this area, whereas the mean value is selected as the representative when average pooling is used. The side of this small area is the pool window size. The following figure shows the max pooling operation whose pooling window size is 2.



# Fully-concatenated Layer

- The fully connected layer is essentially a classifier. The features extracted on the convolutional layer and pooling layer are straightened and placed at the fully connected layer to output and classify results.
- Generally, the softmax function is used as the activation function of the final fully connected output layer to combine all local features into global features and compute the score of each type.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

# Recurrent Neural Network (RNN)

- RNN is a neural network that **captures dynamic information in sequential data** through periodical connections of hidden layer nodes. It can classify sequential data.
- Unlike FNNs, the RNN can keep a context state and even store, learn, and express related information in context windows of any length. Different from traditional neural networks, it is not limited to the space boundary **but it supports time sequences.** In other words, there is a side between the hidden layer of the current moment and the hidden layer of the next moment.
- The RNN is widely used in scenarios related to sequences, such as videos consisting of image frames, audio consisting of clips, and sentences consisting of words.

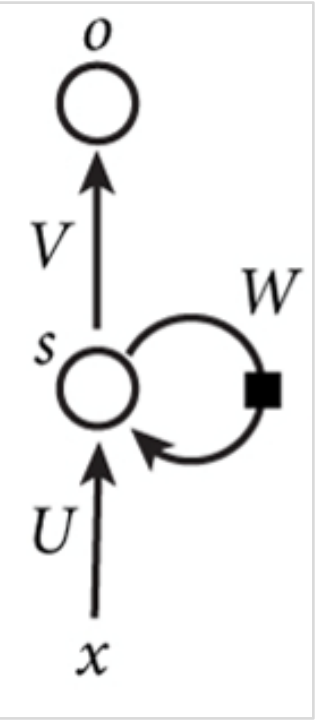
# RNN Architecture (1)

- $X_t$ is the input of the input sequence at time **t**.
- $S_t$ is the memory cell of the sequence at time **t** and caches previous information.

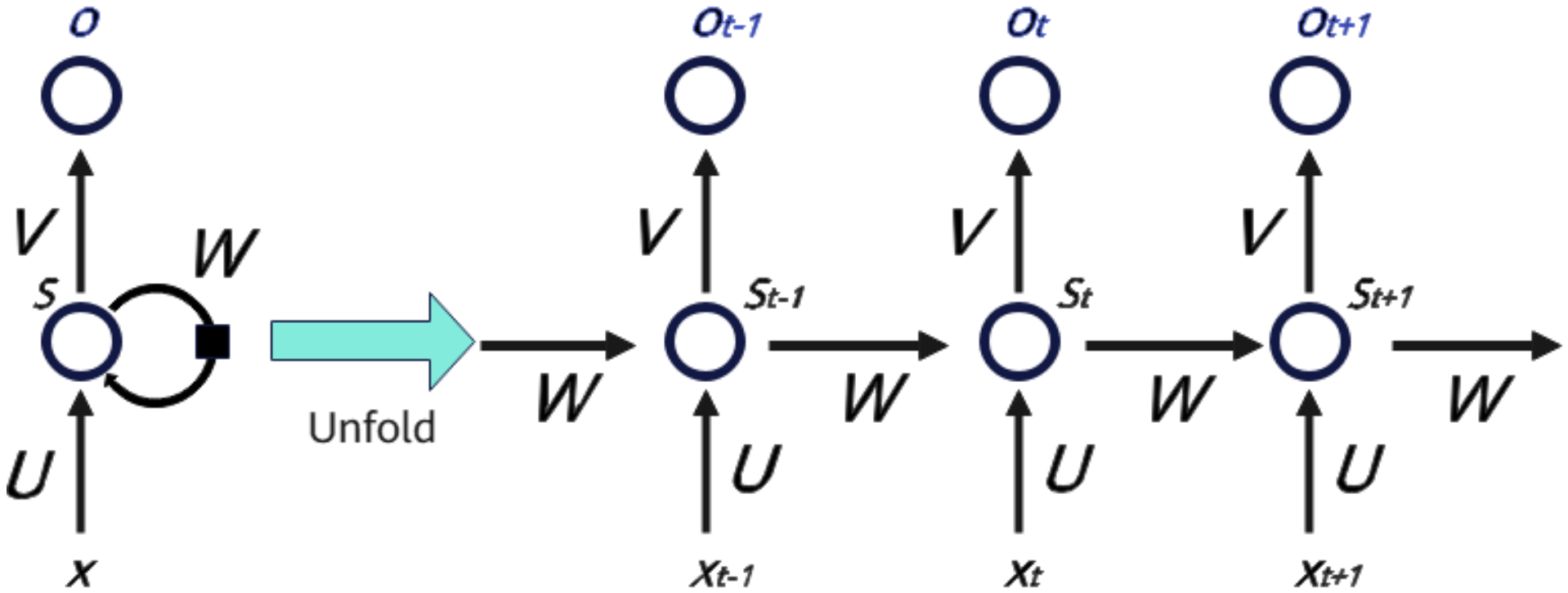
$$S_t = \sigma(UX_t + WS_{t-1}).$$

- $S_t$ passes through multiple hidden layers, and then passes through the fully-connected layer **V** to obtain the final output $O_t$ at time **t**.

$$O_t = softmax(S_tV).$$



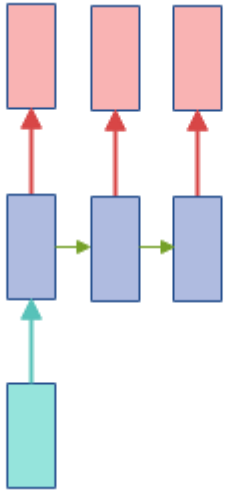
# RNN Architecture (2)



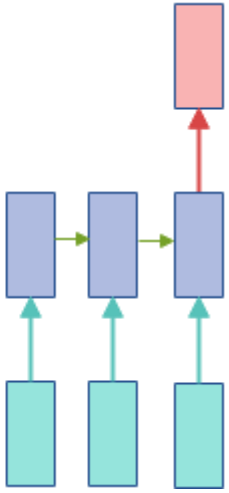
# RNN Types

One to one

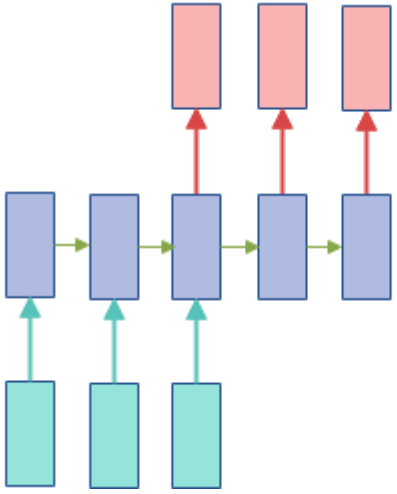
One to many



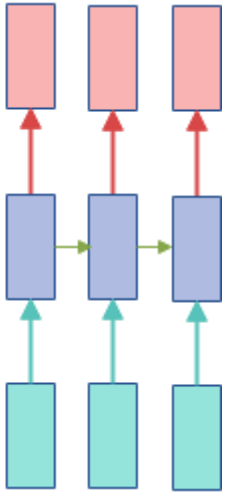
Many to one



Many to many



Many to many



# Backward Propagation Through Time (BPTT)

- BPTT:
  - It is an extension of traditional backward propagation based on time sequences.
  - There are two errors in a memory cell at time sequence **t**: 1. Partial derivative of the error $C_t$ at time sequence **t** to the memory cell. 2. Partial derivative of the error at the next time sequence **t+1** of the memory cell to the memory cell at time sequence **t**. The two errors need to be added.
  - If the time sequence is longer, it is more likely that the loss of the last time sequence to the gradient of weight w in the first time sequence will cause the vanishing gradient or exploding gradient problem.
  - The total gradient of weight w is the accumulation of the gradient of the weights in all time sequences.
- Three steps of BPTT:
  - Compute the output of each neuron (forward).
  - Compute the error $\delta_j$ of each neuron (backward).
  - Compute the gradient of each weight.
- Updating weights using the SGD algorithm

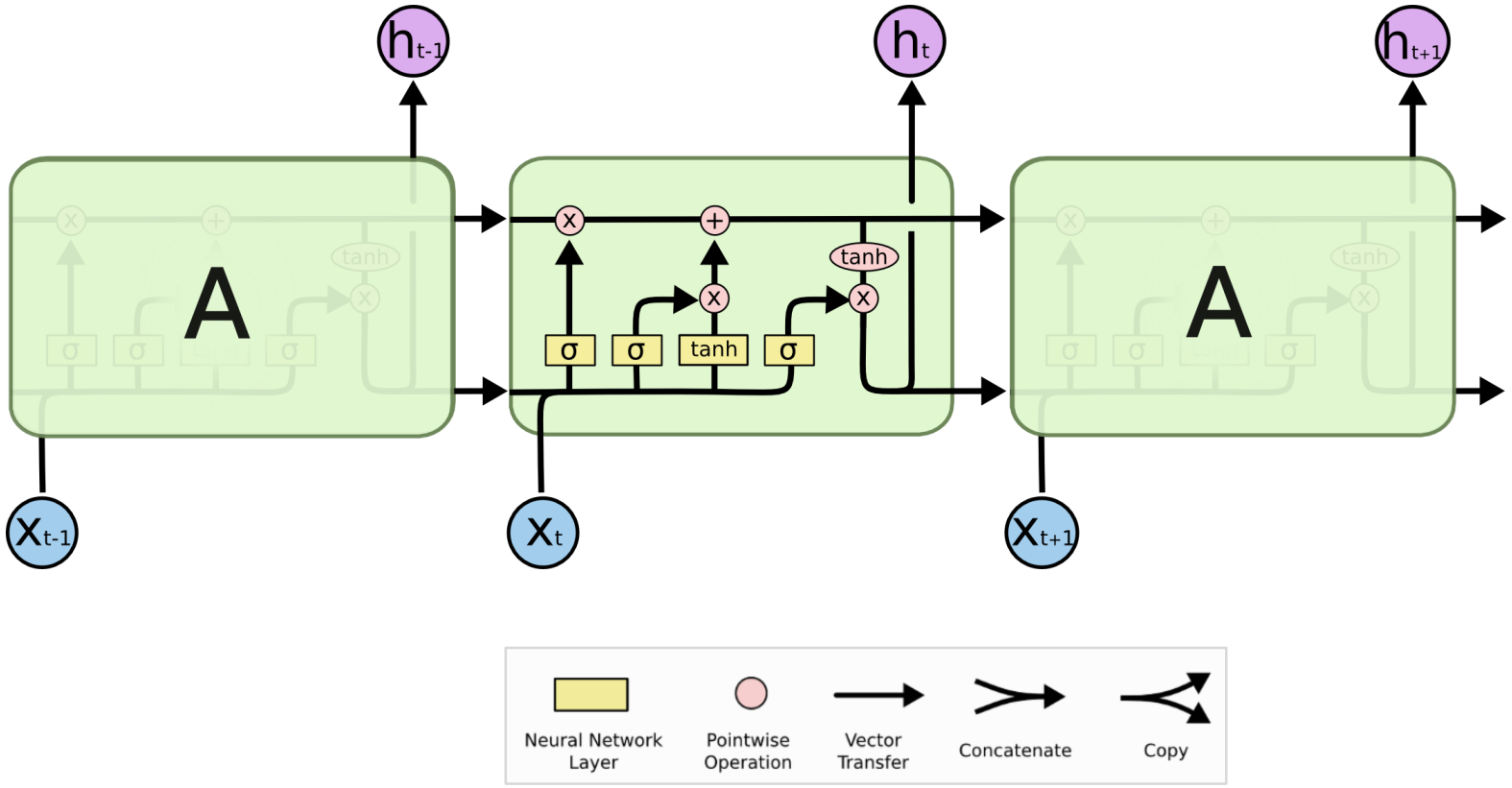
# RNN Problems

- $S_t = \sigma(UX_t + WS_{t-1})$ is extended based on time sequences.

- $S_t = \sigma\left(UX_t + W\left(\sigma\left(UX_{t-1} + W\left(\sigma\left(UX_{t-2} + W(\dots)\right)\right)\right)\right)\right)$

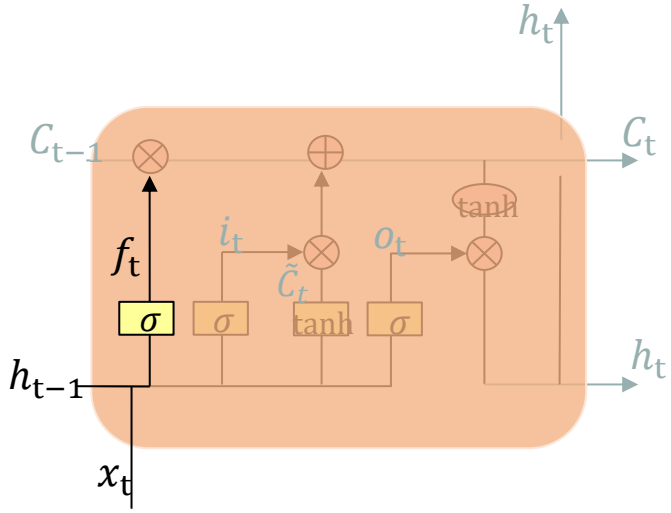
- Despite that the standard RNN structure solves the problem of information memory, **the information attenuates in the case of long-term memory**.
- Information needs to be saved for a long time in many tasks. For example, a hint at the beginning of a speculative fiction may not be answered until the end.
- RNN may not be able to save information for long due to the limited memory cell capacity.
- We expect that memory cells can remember key information.

# Long Short-Term Memory (LSTM) Network



# Forget Gate in LSTM

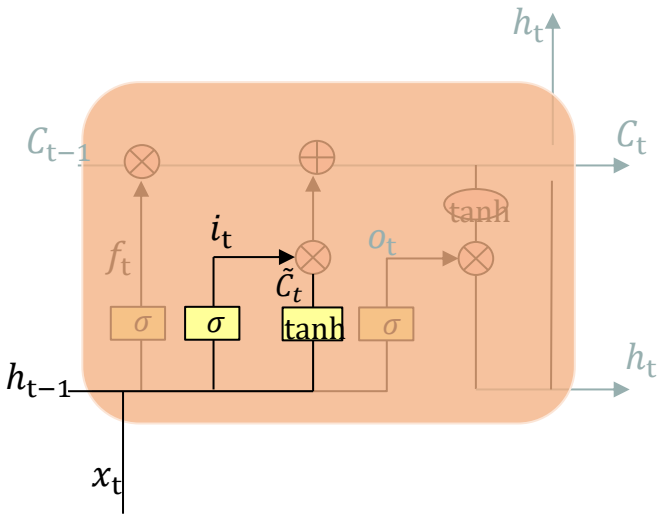
- The first step in LSTM is to decide what information to discard from the cell state.
- The decision is made through the forget gate. This gate reads $h_{t-1}$ and $x_t$ and outputs a numeric value in the range from 0 to 1 for each digit in the cell state $C_{t-1}$. The value **1** indicates that the information is completely retained while the value **0** indicates that the information is completely discarded.




$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

# Input Gate in LSTM

- This step is to determine what information is stored in the cell state.
- It consists of two parts:
  - The sigmoid layer is called the input gate layer, which determines the value to be updated.
  - A candidate value vector is created at the tanh layer and is added to the state.

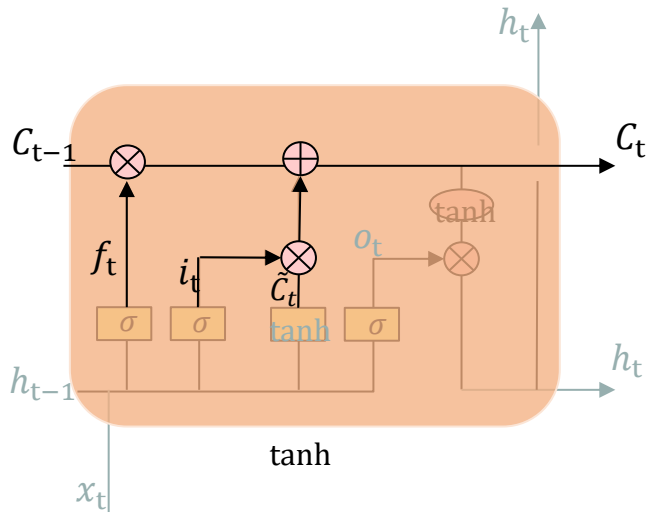



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \sigma(W_C \cdot [h_{t-1}, x_t] + b_C)$$

# Update in LSTM

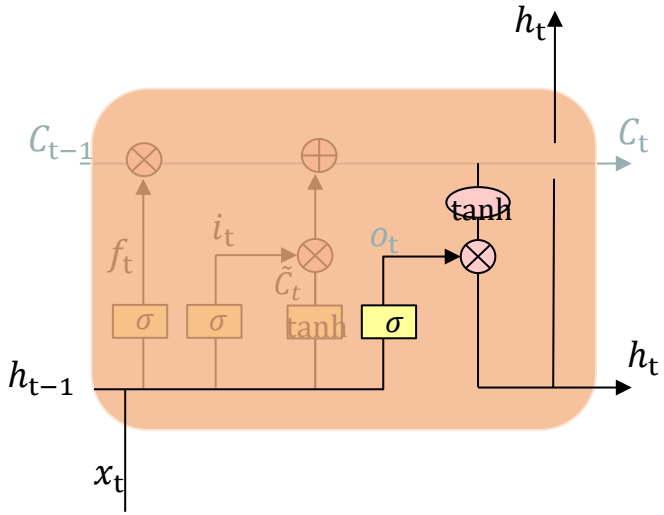
- In this step, it is time to update the state of the old cell. $C_{t-1}$ is updated as $C_t$.
- We multiply the old state by $f_t$ and discard information we decide to discard. Then add $i_t * C_t$. This is a new candidate value, scaled by how much we decided to update each state value.




$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

# Output Gate in LSTM

- We run a sigmoid layer to determine which part of the cell state will be output.
- Then we process the cell state through tanh (a value between –1 and 1 is obtained) and multiply the value by the output of the sigmoid gate. In the end, we output only the part we determine to output.




$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * tanh(C_t)$$

# Quiz

1. (Single-answer question) Which of the following is not a deep learning neural network? ( )

   A. CNN

   B. RNN

   C. LSTM

   D. Logistic

# Quiz

2. (Multiple-answer question) Which of the following are CNN "components"? ( )

A. Activation function

B. Convolutional kernel

C. Pooling

D. Fully-connected layer

# Quiz

3. (True or false) Compared with RNN, CNN is more suitable for image recognition. ( )

A. True

B. False

# Summary

- This chapter mainly introduces the definition and development of neural networks, perceptrons and their training rules, and common types of neural networks.

# Recommendations

- Huawei Talent
  - https://e.huawei.com/en/talent/#/home
- Huawei Support Knowledge Base
  - https://support.huawei.com/enterprise/en/knowledge?lang=en

# Thank you.

Bring digital to every person, home, and organization for a fully connected, intelligent world.



HUAWEI