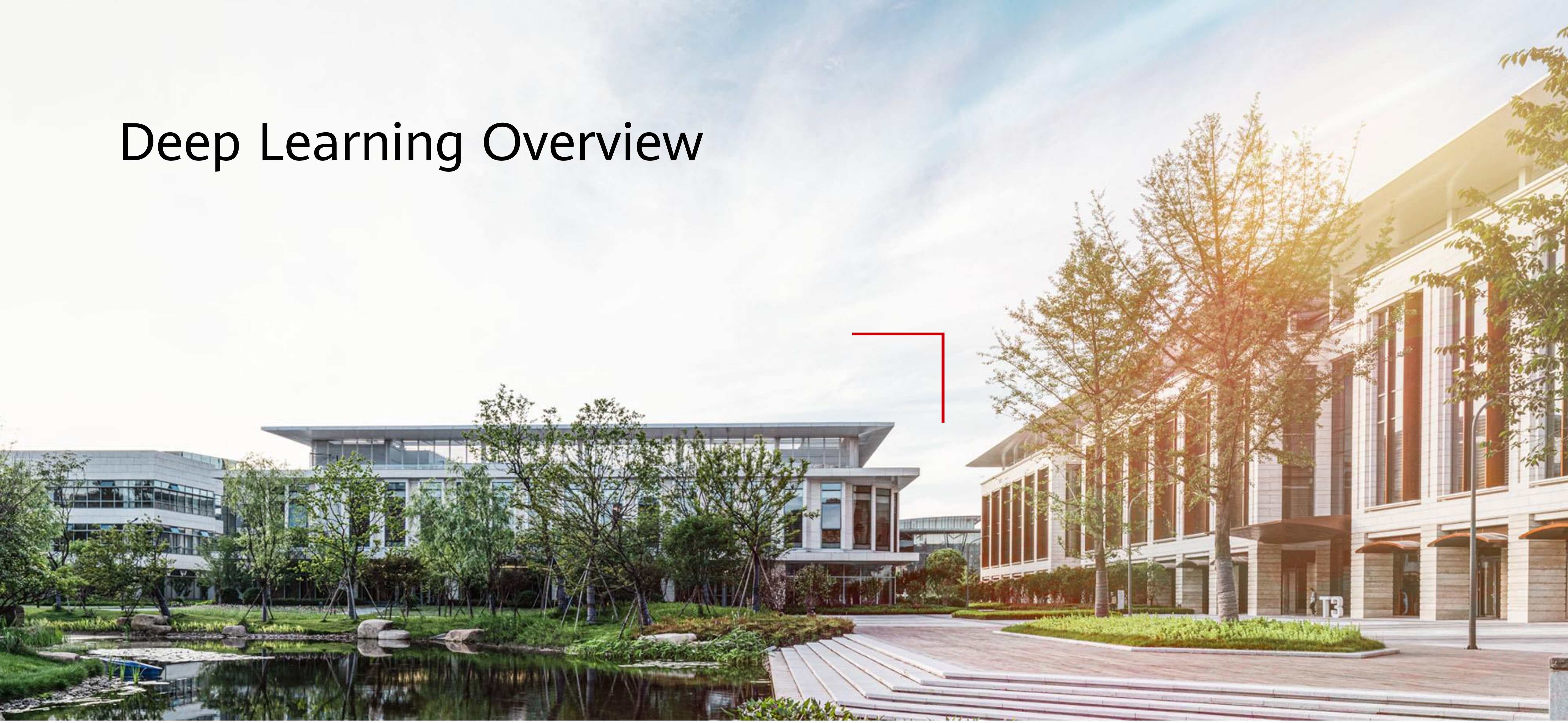
Deep Learning Overview



HUAWEI

Foreword

- The chapter describes the basic knowledge of deep learning, including the development history of deep learning, components and types of deep learning neural networks, and common problems in deep learning projects.

Objectives

On completion of this course, you will be able to:

- Describe the definition and development of neural networks.
- Learn about the important components of deep learning neural networks.
- Understand training and optimization of neural networks.
- Describe common problems in deep learning.

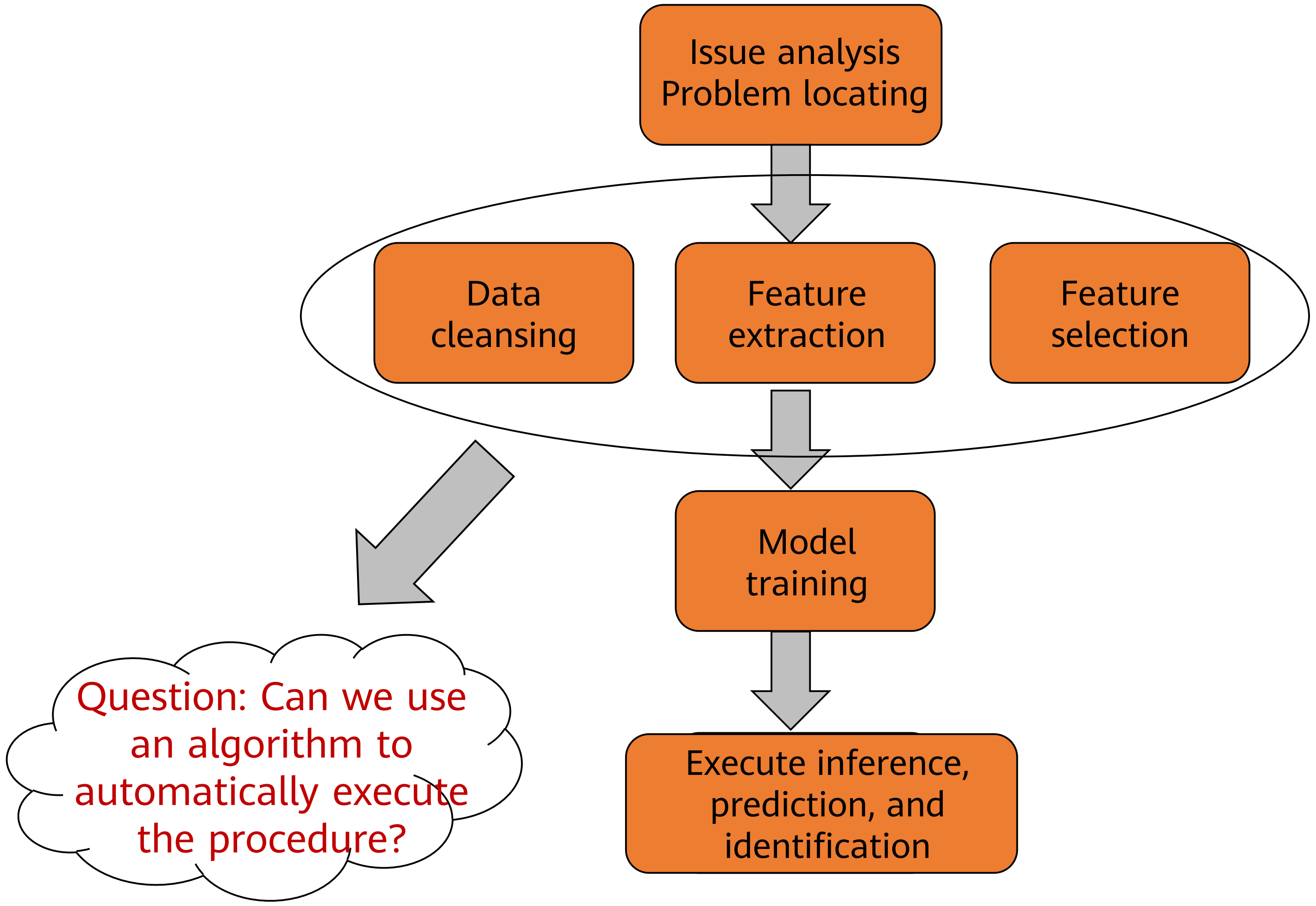
Contents

Traditional Machine Learning and Deep Learning

- As a model based on unsupervised feature learning and feature hierarchy learning, deep learning has great advantages in fields such as computer vision, speech recognition, and natural language processing.

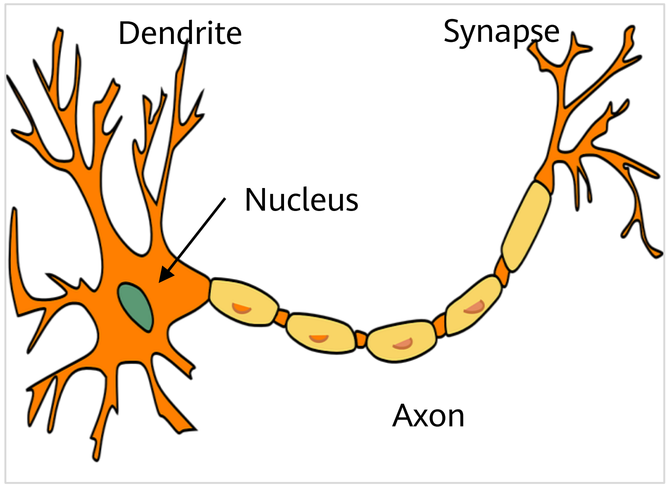
Traditional Machine Learning	Deep Learning
Low hardware requirements on the computer: Given the limited computing amount, the computer does not need a GPU for parallel computing generally.	Higher hardware requirements on the computer: To execute matrix operations on massive data, the computer needs a GPU to perform parallel computing.
Applicable to training under a small data amount and whose performance cannot be improved continuously as the data amount increases.	The performance can be high when high-dimensional weight parameters and massive training data are provided.
Level-by-level problem breakdown	E2E learning
Manual feature selection	Algorithm-based automatic feature extraction
Easy-to-explain features	Hard-to-explain features

Traditional Machine Learning

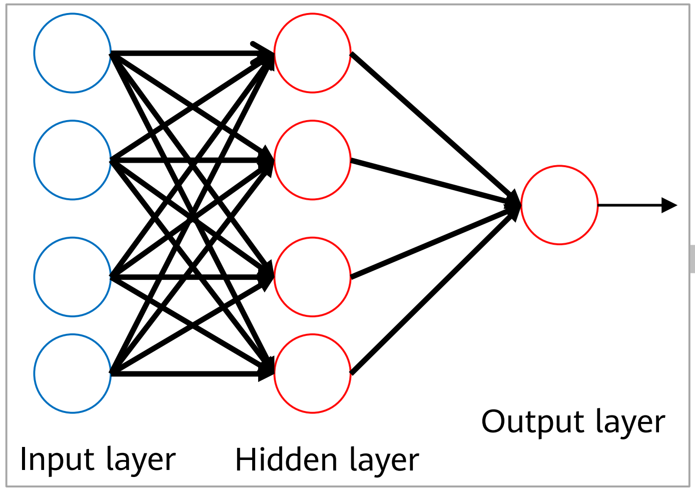

Issue analysis
Problem locating
Data cleansing
Feature extraction
Feature selection
Model training
Execute inference, prediction, and identification
Question: Can we use an algorithm to automatically execute the procedure?

Deep Learning

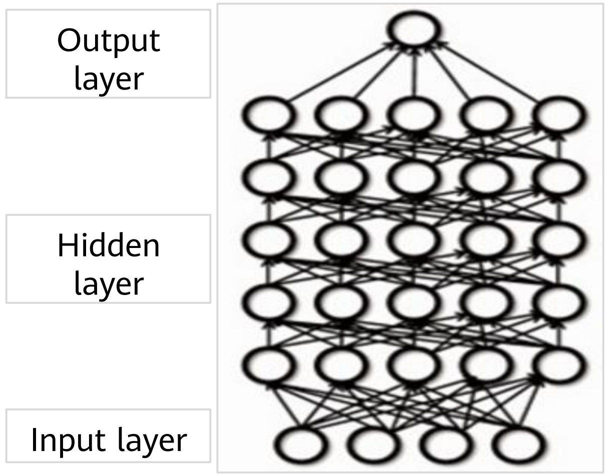
- Generally, the deep learning architecture is a deep neural network. "Deep" in "deep learning" refers to the number of layers of the neural network.



Human neural network



Perceptron

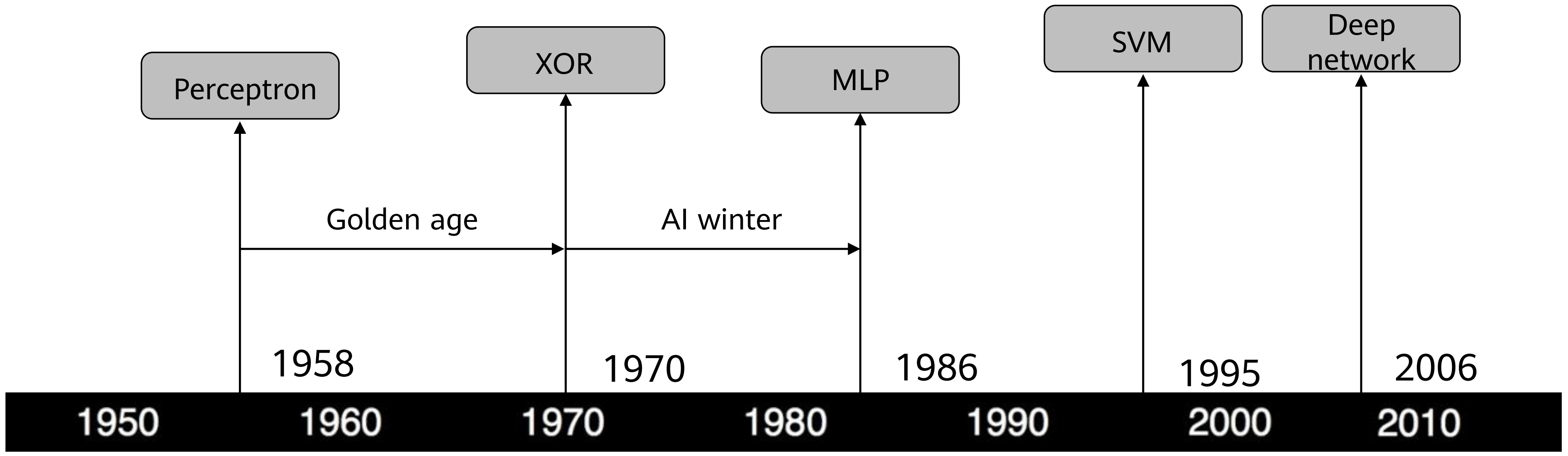


Deep neural network

Neural Network

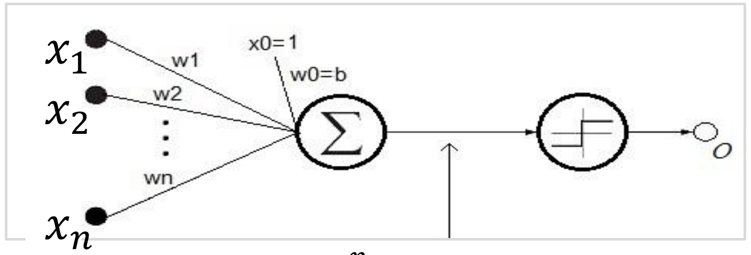
- Currently, the definition of the neural network has not been determined yet. Hecht Nielsen, a neural network researcher in the U.S., defines a neural network as a computer system composed of simple and highly interconnected processing elements, which process information by dynamic response to external inputs.
- A neural network can be simply expressed as an information processing system designed to imitate the human brain structure and functions based on its source, features, and explanations.
- Artificial neural network (neural network): Formed by artificial neurons connected to each other, the neural network extracts and simplifies the human brain's microstructure and functions. It is an important approach to simulate human intelligence and reflect several basic features of human brain functions, such as concurrent information processing, learning, association, model classification, and memory.

Development History of Neural Networks



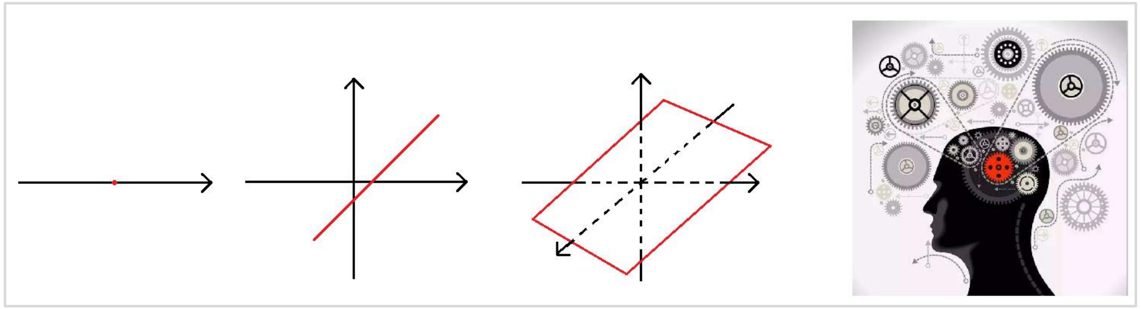
Single-Layer Perceptron

- Input vector: $X = [x_0, x_1, \dots, x_n]^T$
- Weight: $W = [\omega_0, \omega_1, \dots, \omega_n]^T$, in which ω_0 is the offset.
- Activation function: $O = sign(net) = \begin{cases} 1, net > 0, \\ -1, otherwise. \end{cases}$



$$net = \sum_{i=0}^{n} \omega_i x_i = \boldsymbol{W^T}\mathbf{X}$$

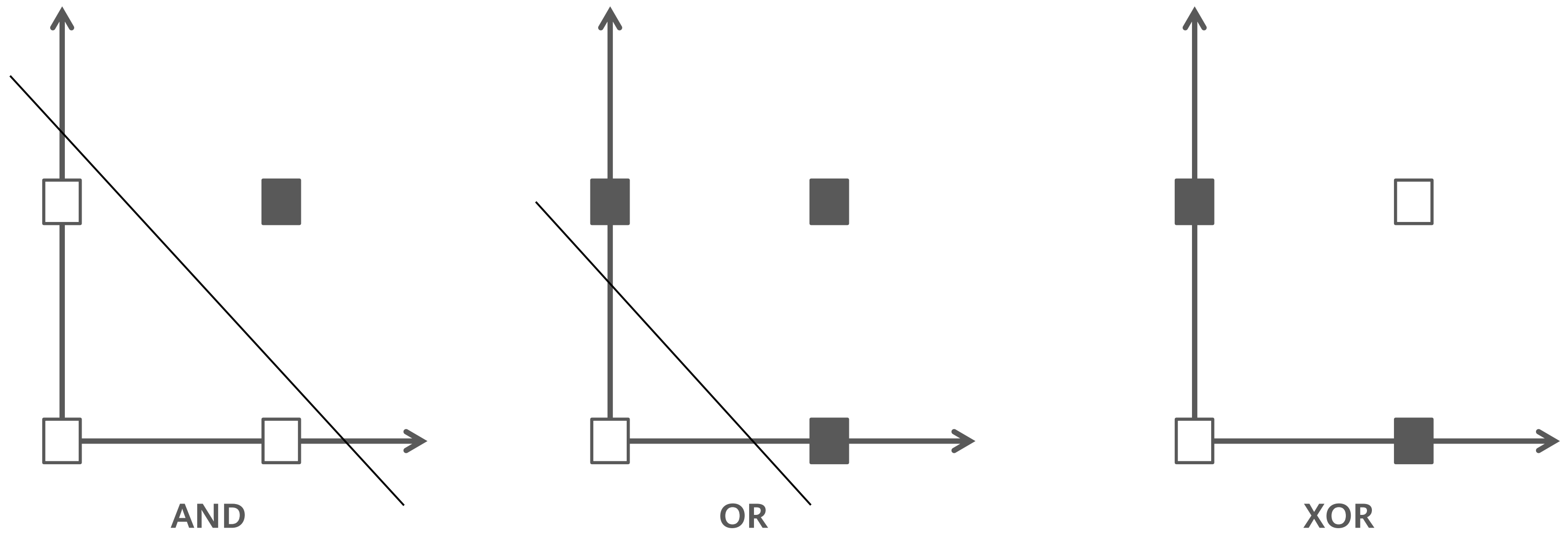
- The preceding perceptron is equivalent to a classifier. It uses the high-dimensional X vector as the input and performs binary classification on input samples in the high-dimensional space. When $\boldsymbol{W^T}\mathbf{X} > 0$, $\mathrm{O} = 1$. In this case, the samples are classified into a type. Otherwise, $\mathrm{O} = -1$. In this case, the samples are classified into the other type. The boundary of these two types is $\boldsymbol{W^T}\mathbf{X} = 0$, which is a high-dimensional hyperplane.



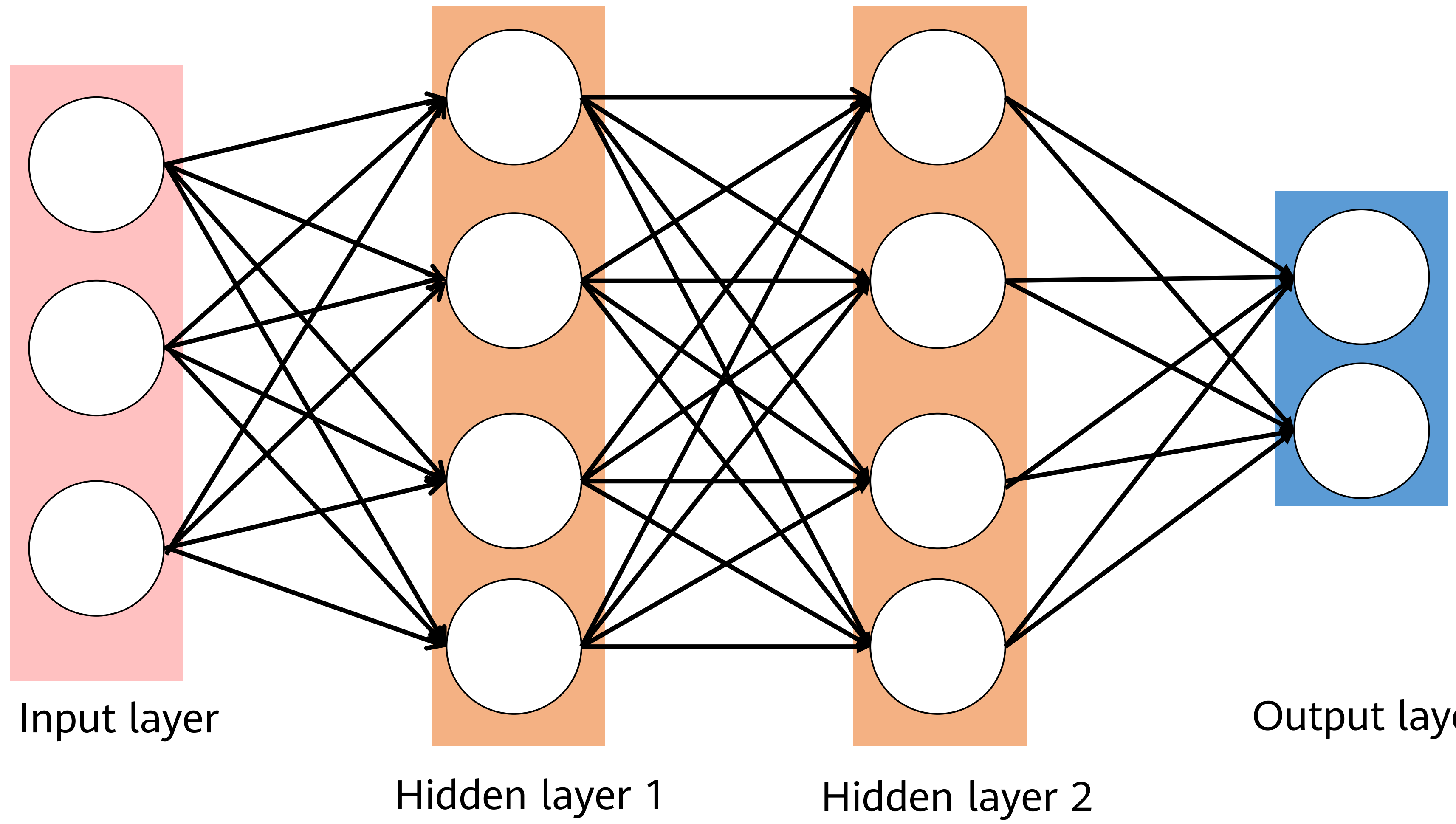
Classification point $Ax + B = 0$ Classification line $Ax + By + C = 0$ Classification plane $Ax + By + Cz + D = 0$ Classification hyperplane $W^T\mathrm{X} + b = 0$

XOR Problem

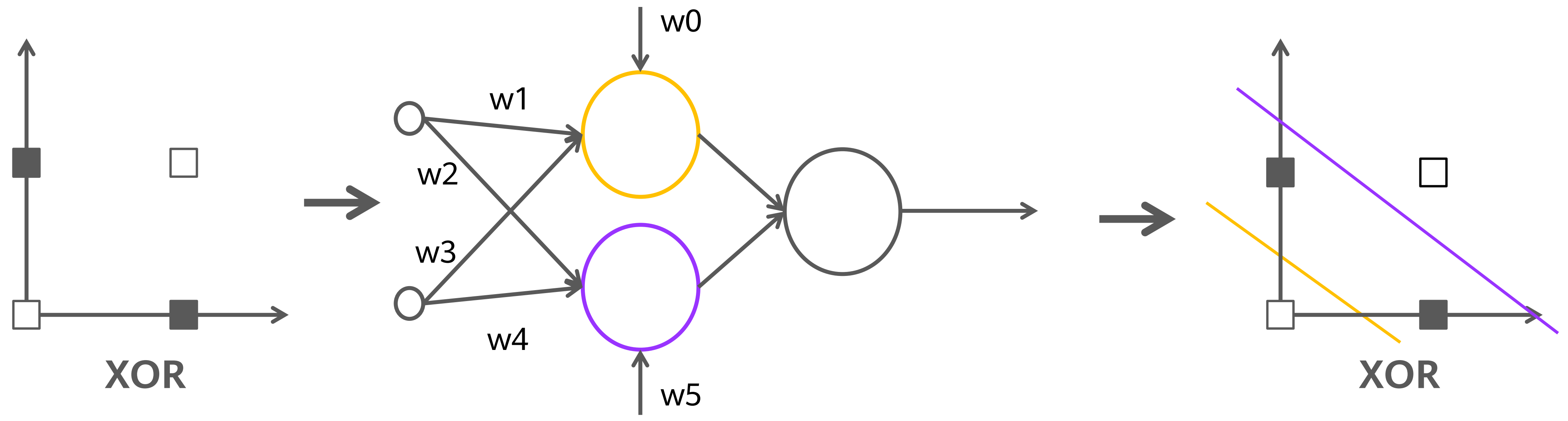
- In 1969, Minsky, an American mathematician and AI pioneer, proved that a perceptron is essentially a linear model that can only deal with linear classification problems, but cannot process non-linear data.



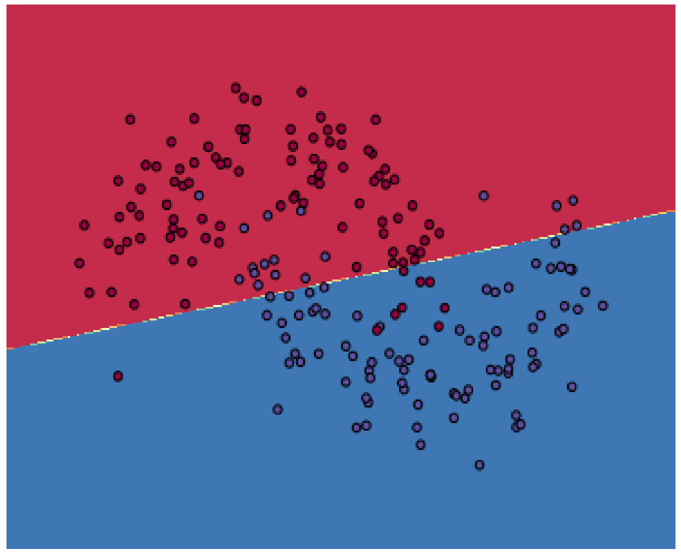
Feedforward Neural Network

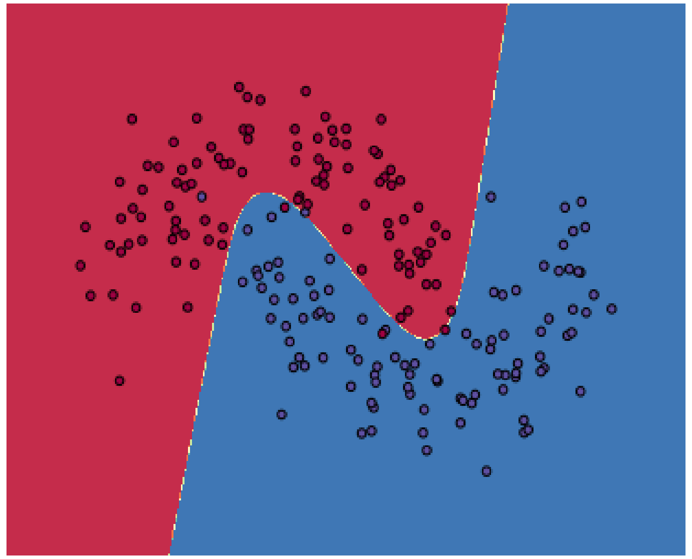


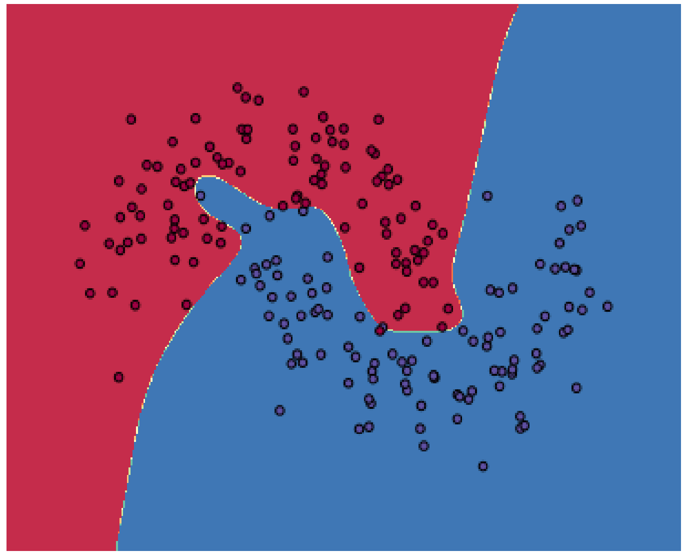
Solution of XOR



Impacts of Hidden Layers on A Neural Network


0 hidden layers


3 hidden layers


20 hidden layers

Contents

Gradient Descent and Loss Function

- The gradient of the multivariate function $o = f(x) = f(x_0, x_1, \ldots, x_n)$ at $X' = [x_0', x_1', \ldots, x_n']^T$ is shown as follows:

$$\nabla f(x_0', x_1', \ldots, x_n') = [\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}]^T |_{X=X'},$$

  The direction of the gradient vector is the fastest growing direction of the function. As a result, the direction of the negative gradient vector $-\nabla f$ is the fastest descent direction of the function.

- During the training of the deep learning network, target classification errors must be parameterized. **A loss function (error function)** is used, which reflects the error between the target output and actual output of the perceptron. For a single training sample x, the most common error function is the **Quadratic cost function**.

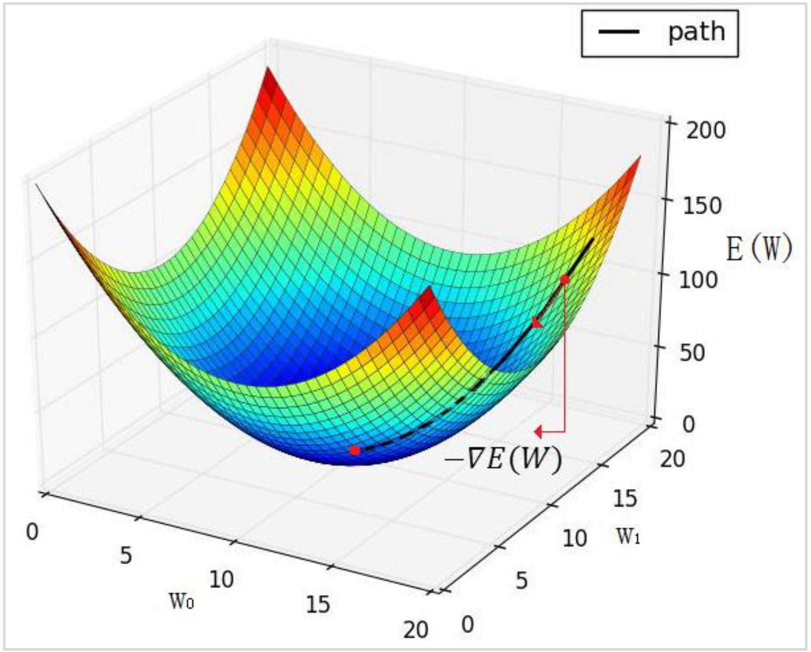
$$E(w) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2,$$

  In the preceding function, d is one neuron in the output layer, D is all the neurons in the output layer, t_d is the target output, and o_d is the actual output.

- The gradient descent method enables the loss function to search along the negative gradient direction and update the parameters iteratively, finally minimizing the loss function.

Extrema of the Loss Function

- Purpose: The loss function $E(W)$ is defined on the weight space. The objective is to search for the weight vector W that can minimize $E(W)$.
- Limitation: No effective method can solve the extremum in mathematics on the complex high-dimensional surface of $E(W) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$.



Example of gradient descent of binary paraboloid

Common Loss Functions in Deep Learning

- Quadratic cost function:

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

- Cross entropy error function:

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

- The cross entropy error function depicts the distance between two probability distributions, which is a widely used loss function for classification problems.
- Generally, the mean square error function is used to solve the regression problem, while the cross entropy error function is used to solve the classification problem.

Batch Gradient Descent Algorithm (BGD)

- In the training sample set D, each sample is recorded as $< X, t >$, in which X is the input vector, t the target output, o the actual output, and η the learning rate.
  - Initializes each w_i to a random value with a smaller absolute value.
  - Before the end condition is met:
    - Initializes each Δw_i to zero.
    - For each $< X, t >$ in D:
      - Input X to this unit and calculate the output o.
      - For each w_i in this unit: Δw_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+= Δw_i.
- The gradient descent algorithm of this version is not commonly used because:
  - The convergence process is very slow as all training samples need to be calculated every time the weight is updated.

Stochastic Gradient Descent Algorithm (SGD)

- To address the BGD algorithm defect, a common variant called Incremental Gradient Descent algorithm is used, which is also called the Stochastic Gradient Descent (SGD) algorithm. One implementation is called Online Learning, which updates the gradient based on each sample:

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

- ONLINE-GRADIENT-DESCENT
  - Initializes each w_i to a random value with a smaller absolute value.
  - Before the end condition is met:
  - Generates a random <X, t> from D and does the following calculation:
    - Input X to this unit and calculate the output o.
    - For each w_i in this unit: w_i += -η$\sum_{d \in D} \frac{\partial C(t_d, o_d)}{\partial w_i}$.

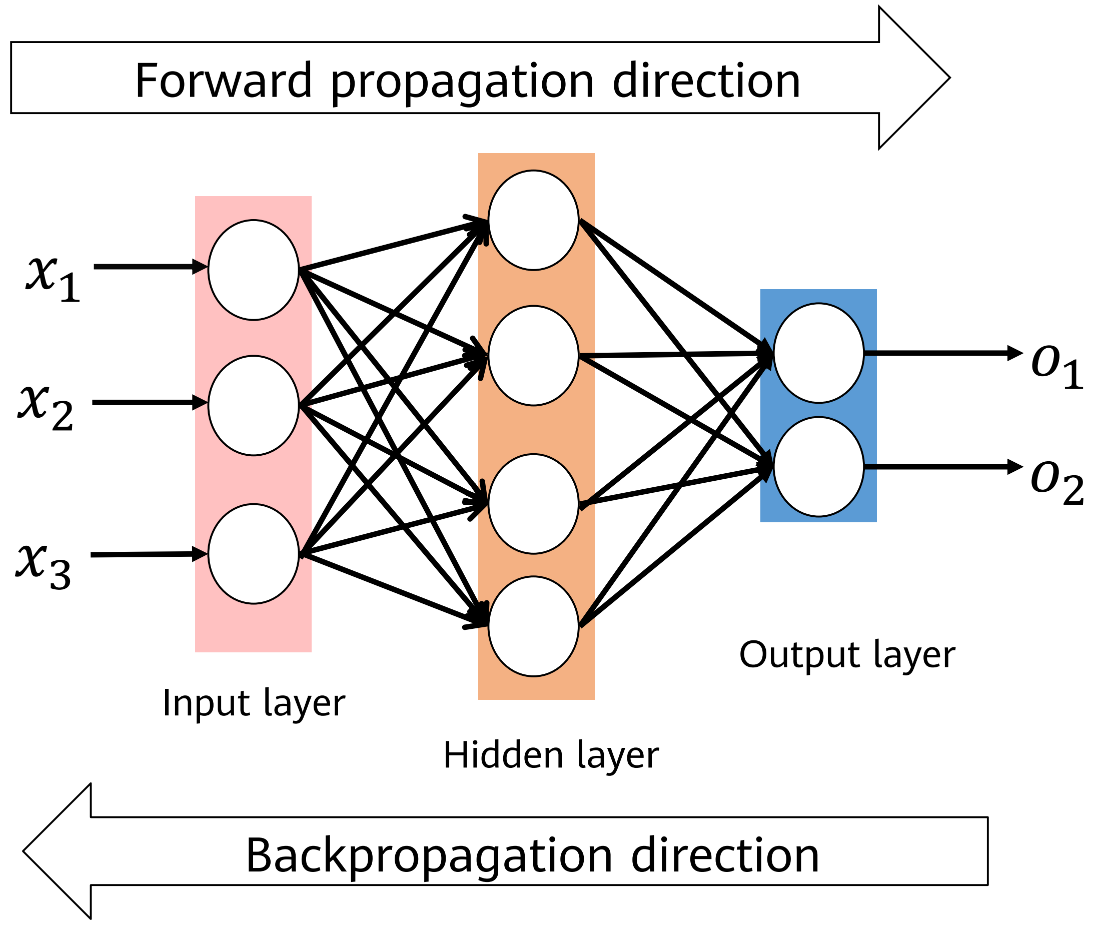
Mini-Batch Gradient Descent Algorithm (MBGD)

- To address the defects of the previous two gradient descent algorithms, the Mini-batch Gradient Descent Algorithm (MBGD) was proposed and has been most widely used. A small number of Batch Size (BS) samples are used at a time to calculate Δw_i, and then the weight is updated accordingly.
- Batch-gradient-descent
  - Initializes each w_i to a random value with a smaller absolute value.
  - Before the end condition is met:
    - Initializes each Δw_i to zero.
    - For each $< X, t >$ in the BS samples in the next batch in D:
      - Input X to this unit and calculate the output o.
      - For each w_i in this unit: Δw_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+= Δw_i
    - For the last batch, the training samples are mixed up in a random order.

Backpropagation Algorithm (1)

- Signals are propagated in forward direction, and errors are propagated in backward direction.
- In the training sample set D, each sample is recorded as <X, t>, in which X is the input vector, t the target output, o the actual output, and w the weight coefficient.
- Loss function:

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



Backpropagation Algorithm (2)

- According to the following formulas, errors in the input, hidden, and output layers are accumulated to generate the error in the loss function.
- wc is the weight coefficient between the hidden layer and the output layer, while wb is the weight coefficient between the input layer and the hidden layer. f is the activation function, D is the output layer set, and C and B are the hidden layer set and input layer set respectively. Assume that the loss function is a quadratic cost function:
  - Output layer error: $$E = \frac{1}{2}\sum_{(d\in D)}(t_d - o_d)^2$$
  - Expanded hidden layer error: $$E = \frac{1}{2}\sum_{(d\in D)}[t_d - f(net_d)]^2 = \frac{1}{2}\sum_{(d\in D)}\left[t_d - f(\sum_{(c\in C)} w_c y_c)\right]^2$$
  - Expanded input layer error: $$E = \frac{1}{2}\sum_{(d\in D)}\left[t_d - f\left(\sum_{(c\in C)} w_c f(net_c)\right)\right]^2 = \frac{1}{2}\sum_{(d\in D)}\left[t_d - f\left(\sum_{(c\in C)} w_c f\left(\sum_{b\in B} w_b x_b\right)\right)\right]^2$$

Backpropagation Algorithm (3)

- To minimize error E, the gradient descent iterative calculation can be used to solve wc and wb, that is, calculating wc and wb to minimize error E.
- Formula:

$$\Delta w_c = -\eta \frac{\partial E}{\partial w_c}, c \in C$$

$$\Delta w_b = -\eta \frac{\partial E}{\partial w_b}, b \in B$$

- If there are multiple hidden layers, chain rules are used to take a derivative for each layer to obtain the optimized parameters by iteration.

Backpropagation Algorithm (4)

- For a neural network with any number of layers, the arranged formula for training is as follows:

$$\Delta w_{jk}^{l} = -\eta \delta_{k}^{l+1} f_{j}(z_{j}^{l})$$

$$\delta_{j}^{l} = \begin{cases} f_{j}^{'}(z_{j}^{l})(t_{j} - f_{j}(z_{j}^{l})), l \in outputs, (1) \\ \sum_{k} \delta_{k}^{l+1} w_{jk}^{l} f_{j}^{'}(z_{j}^{l}), otherwise, (2) \end{cases}$$

- The BP algorithm is used to train the network as follows:
  - Takes out the next training sample <X, T>, inputs X to the network, and obtains the actual output o.
  - Calculates output layer δ according to the output layer error formula (1).
  - Calculates δ of each hidden layer from output to input by iteration according to the hidden layer error propagation formula (2).
  - According to the δ of each layer, the weight values of all the layer are updated.

Contents

Activation Function

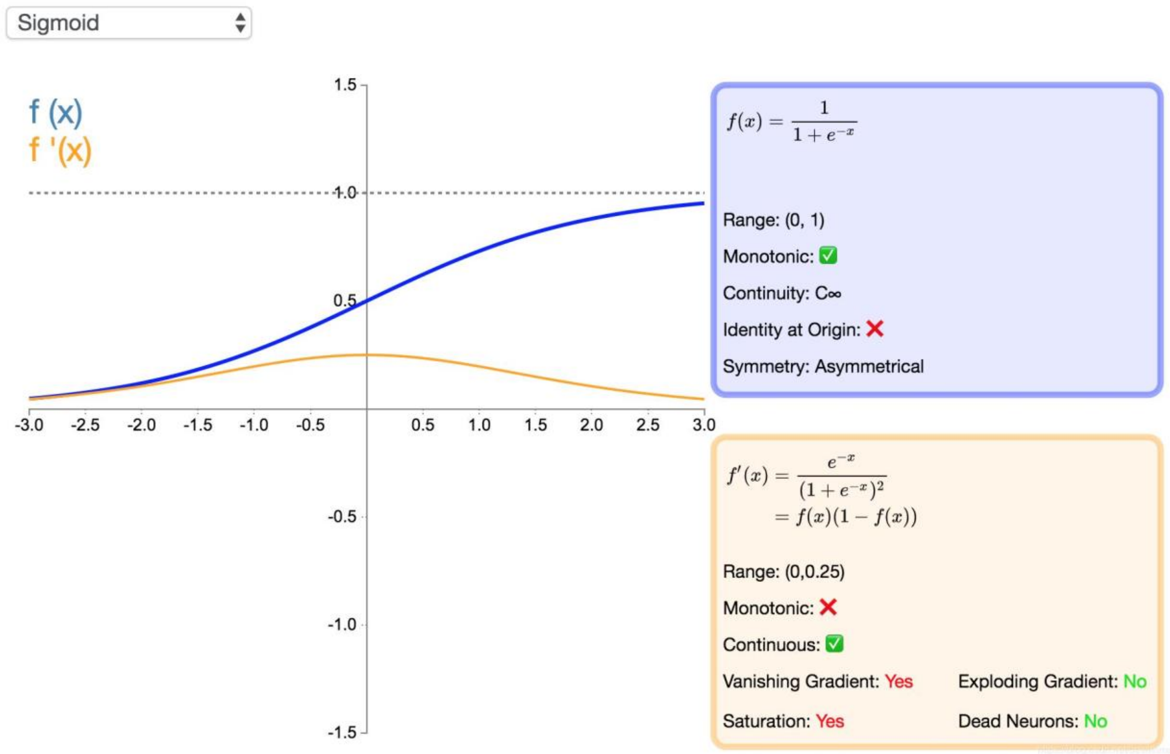
- Activation functions are important for the neural network model to learn and understand complex non-linear functions. They allow introduction of non-linear features to the network.
- Without activation functions, output signals are only simple linear functions. The complexity of linear functions is limited, and the capability of learning complex function mappings from data is low.

Activation Function

$$output = f(w_1x_1 + w_2x_2 + w_3x_3 \ldots) = f(W^t \bullet X)$$

Sigmoid

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



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

Range: (0, 1)

Monotonic: ✓

Continuity: C∞

Identity at Origin: ✗

Symmetry: Asymmetrical

$$f'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = f(x)(1 - f(x))$$

Range: (0,0.25)

Monotonic: ✗

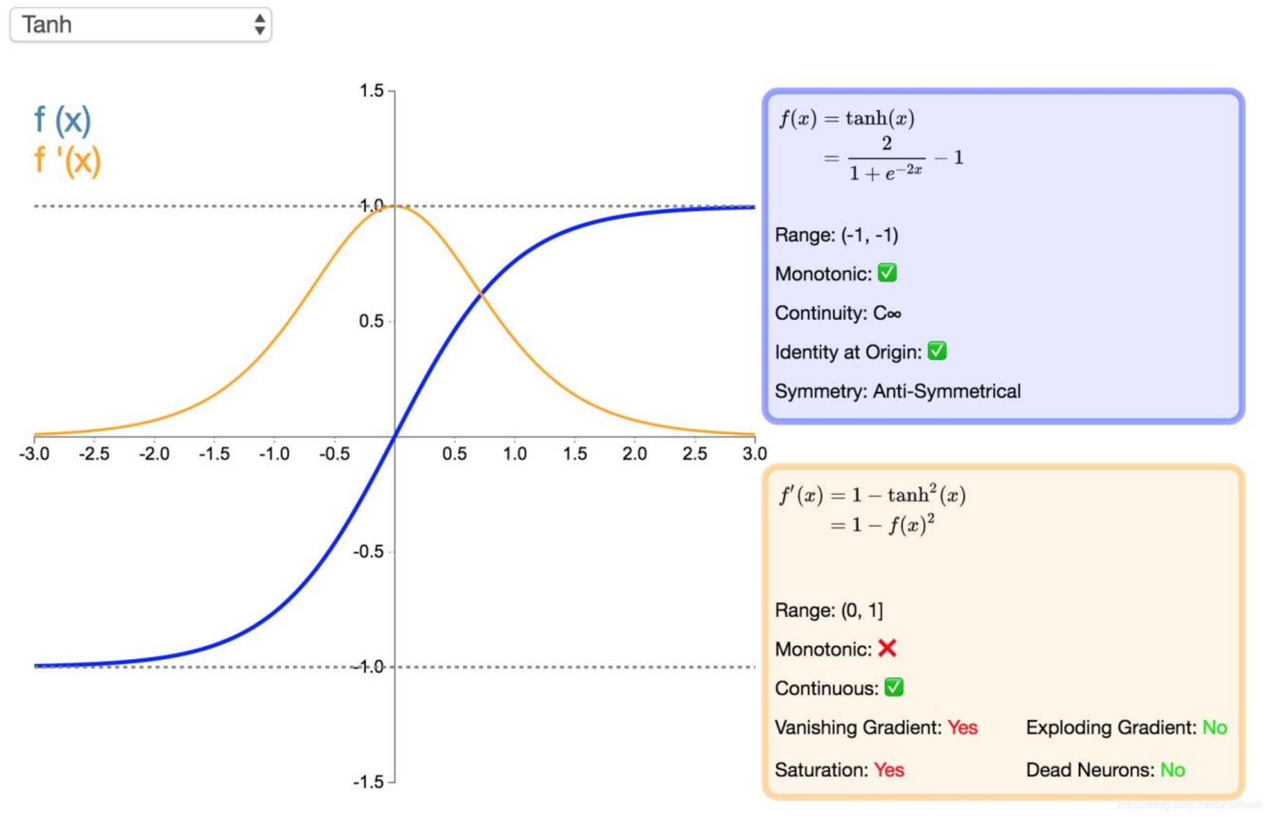
Continuous: ✓

Vanishing Gradient: Yes   Exploding Gradient: No

Saturation: Yes   Dead Neurons: No

Tanh

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



$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

Range: (-1, -1)

Monotonic: ✓

Continuity: C∞

Identity at Origin: ✓

Symmetry: Anti-Symmetrical

$$f'(x) = 1 - \tanh^2(x) = 1 - f(x)^2$$

Range: (0, 1]

Monotonic: ✗

Continuous: ✓

Vanishing Gradient: Yes

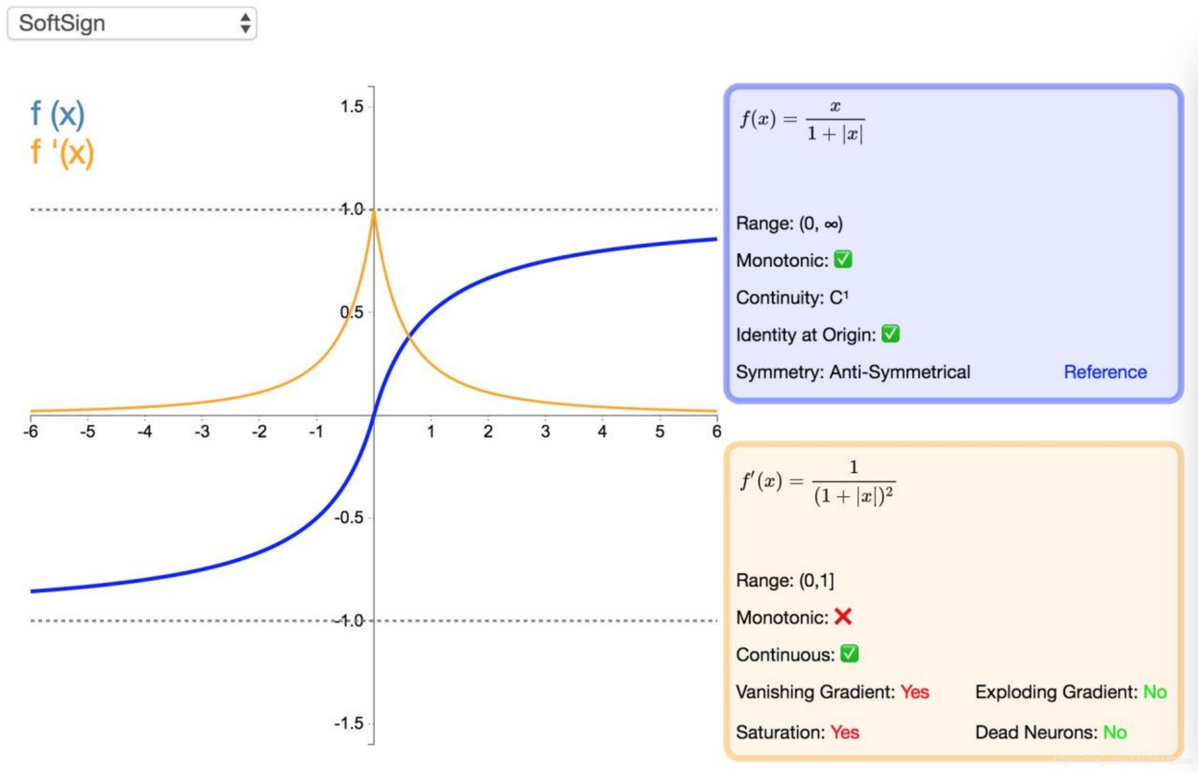
Exploding Gradient: No

Saturation: Yes

Dead Neurons: No

Softsign

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



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

Range: (0, ∞)

Monotonic: ✓

Continuity: C^1

Identity at Origin: ✓

Symmetry: Anti-Symmetrical

Reference

$$f'(x) = \frac{1}{(1 + |x|)^2}$$

Range: (0,1]

Monotonic: ✗

Continuous: ✓

Vanishing Gradient: Yes

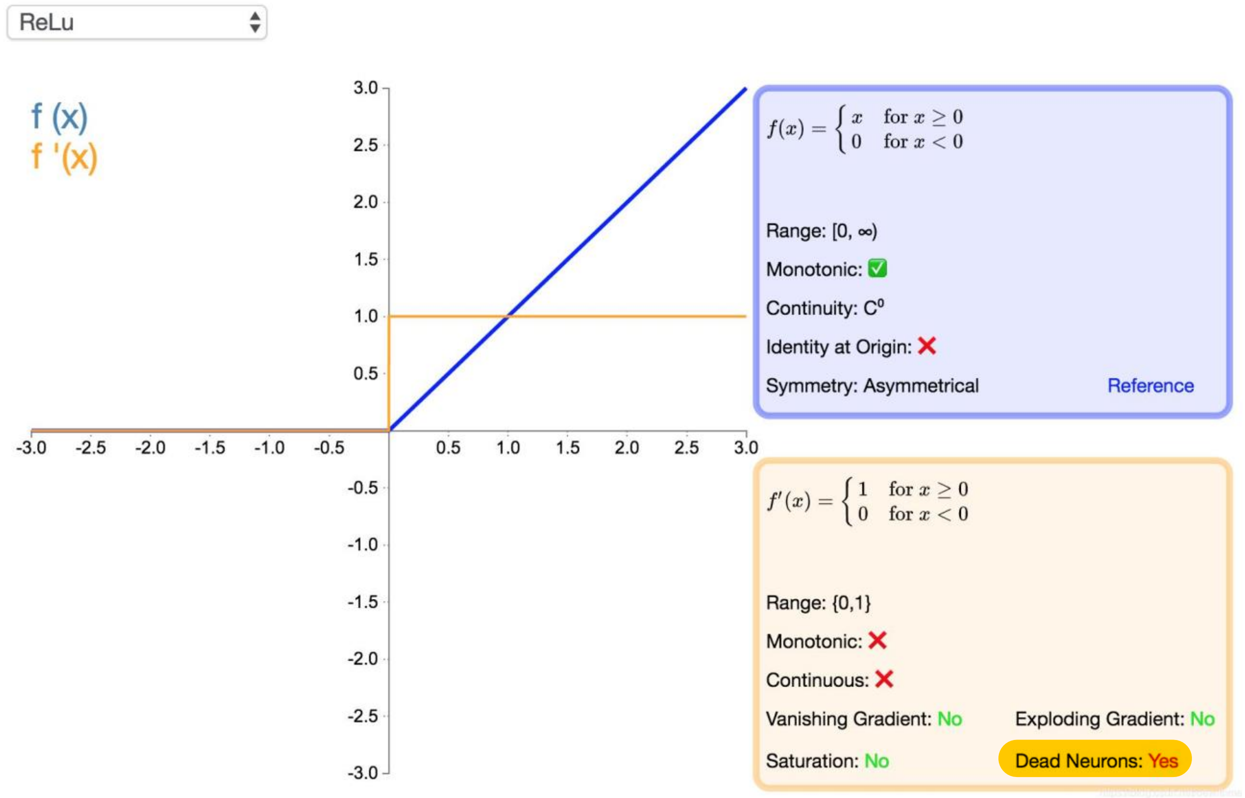
Exploding Gradient: No

Saturation: Yes

Dead Neurons: No

Rectified Linear Unit (ReLU)

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



$$f(x) = \begin{cases} x & \text{for } x \geq 0 \\ 0 & \text{for } x < 0 \end{cases}$$

Range: $[0, \infty)$

Monotonic: ✅

Continuity: C^0

Identity at Origin: ❌

Symmetry: Asymmetrical

Reference

$$f'(x) = \begin{cases} 1 & \text{for } x \geq 0 \\ 0 & \text{for } x < 0 \end{cases}$$

Range: {0,1}

Monotonic: ❌

Continuous: ❌

Vanishing Gradient: No

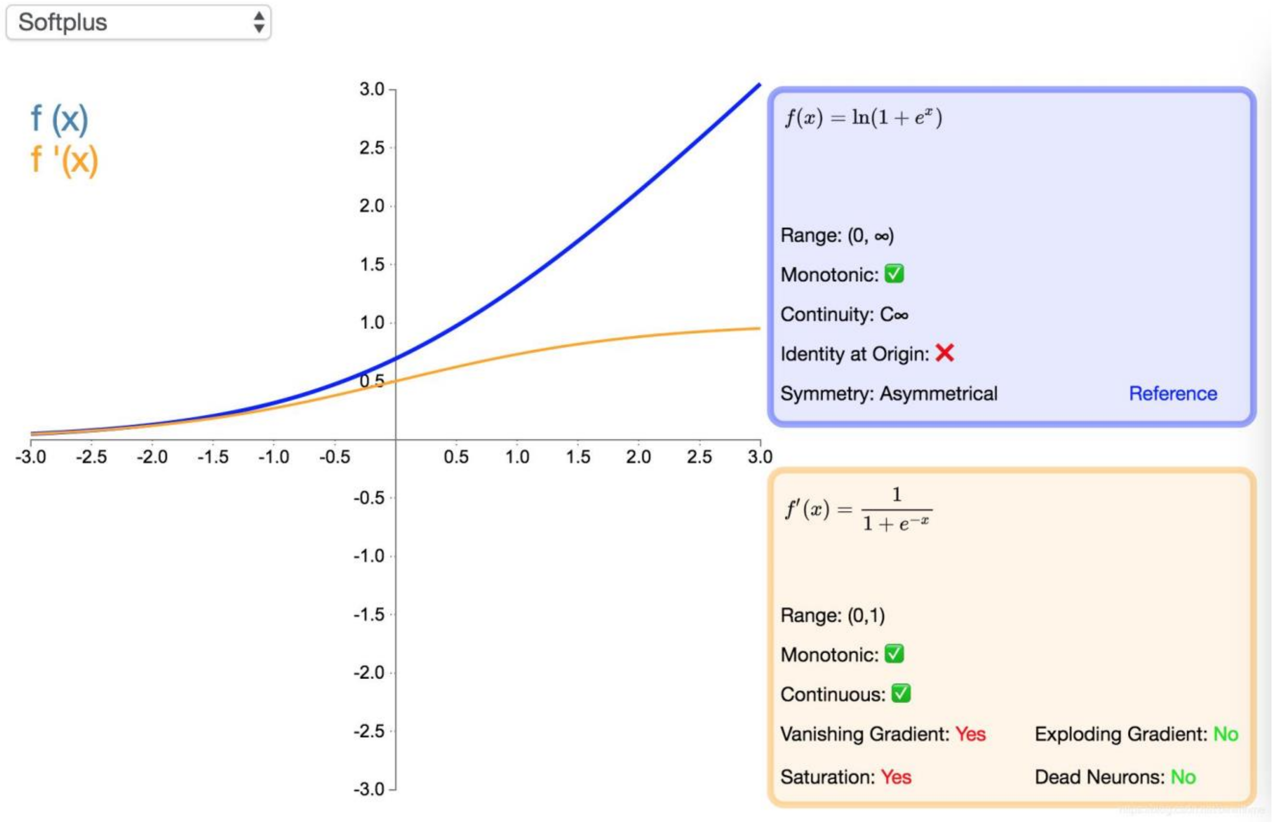
Exploding Gradient: No

Saturation: No

Dead Neurons: Yes

Softplus

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



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

Range: (0, ∞)

Monotonic: ✓

Continuity: C∞

Identity at Origin: ✗

Symmetry: Asymmetrical

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

Range: (0,1)

Monotonic: ✓

Continuous: ✓

Vanishing Gradient: Yes

Exploding Gradient: No

Saturation: Yes

Dead Neurons: No

Softmax

- Softmax function:

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

- 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

Normalizer

- Regularization is an important and effective technology to reduce generalization errors in machine learning. It is especially useful for deep learning models that tend to be over-fit due to a large number of parameters. Therefore, researchers have proposed many effective technologies to prevent over-fitting, 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

Penalty Parameters

- Many regularization methods restrict the learning capability of models by adding a penalty parameter $\Omega(\theta)$ to the objective function J. Assume that the target function after regularization is $\tilde{J}$.

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

- Where $\alpha \epsilon [0, \infty)$ is a hyperparameter that weights the relative contribution of the norm penalty term Ω and the standard objective function $J(X; \theta)$. If α is set to 0, no regularization is performed. The penalty in regularization increases with α.

L_1 Regularization

- Add L_1 norm constraint 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

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

L_2 Regularization

- Add norm penalty term L_2 to prevent overfitting.

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

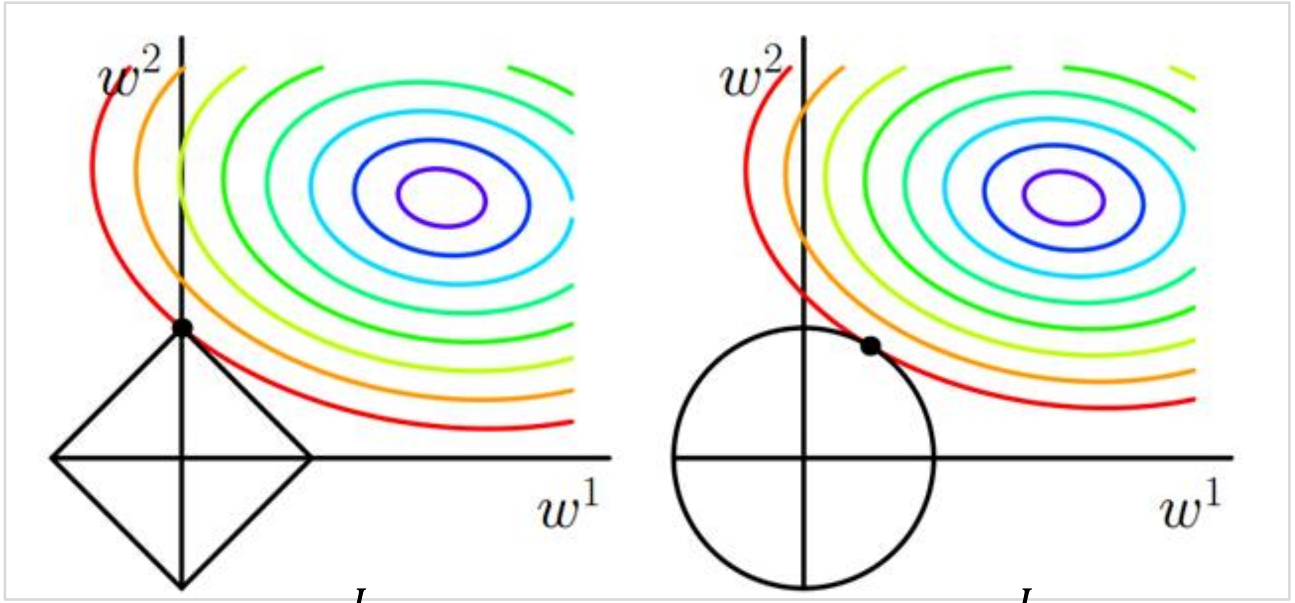
- A parameter optimization method can be inferred using an optimization technology (such as a gradient method):

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

- where ε is the learning rate. Compared with a common gradient optimization formula, this formula multiplies the parameter by a reduction factor.

L_1 v.s. L_2

- The major differences between L_2 and L_1:
  - According to the preceding analysis, L_1 can generate a more sparse model than L_2. When the value of parameter w is small, L_1 regularization 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 regularization, the parameter value complies with the Gaussian distribution rule. In L_1 regularization, the parameter value complies with the Laplace distribution rule.



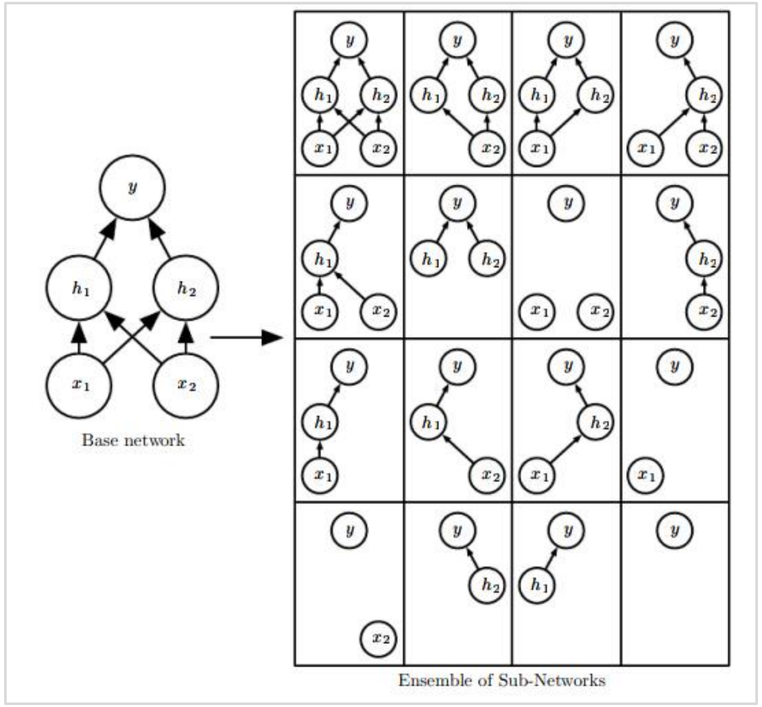
L_1 L_2

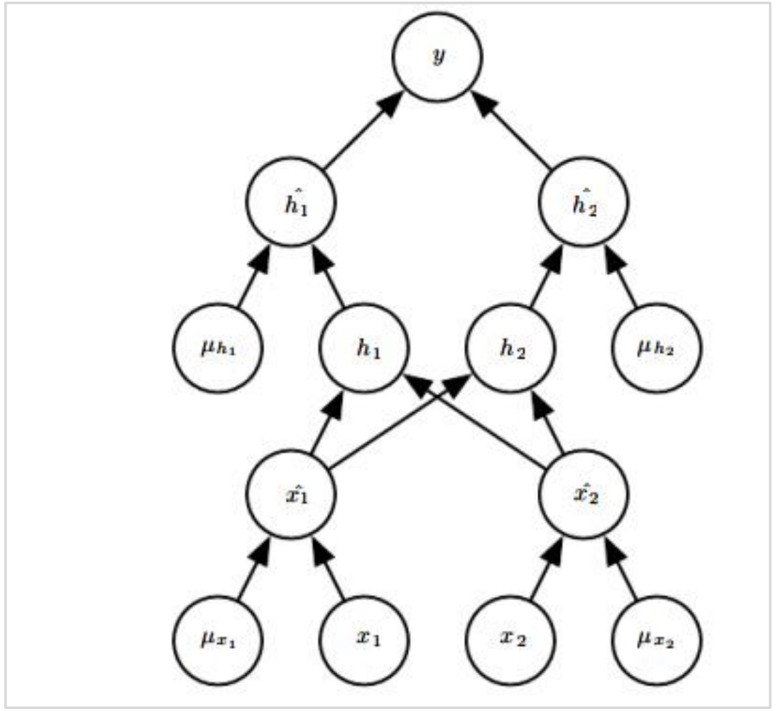
Dataset Expansion

- The most effective way to prevent over-fitting is to add a training set. A larger training set has a smaller over-fitting probability. Dataset expansion is a time-saving method, but it varies in different 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 cannot be changed through transformation. For example, for handwriting digit recognition, categories 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.
  - Noise injection can add noise to the input or to the hidden layer or output layer. For example, for Softmax classification, noise can be added using the label smoothing technology. If noise is added to categories 0 and 1, the corresponding probabilities are changed to $\frac{\varepsilon}{k}$ and $1-\frac{k-1}{k}\varepsilon$ respectively.

Dropout

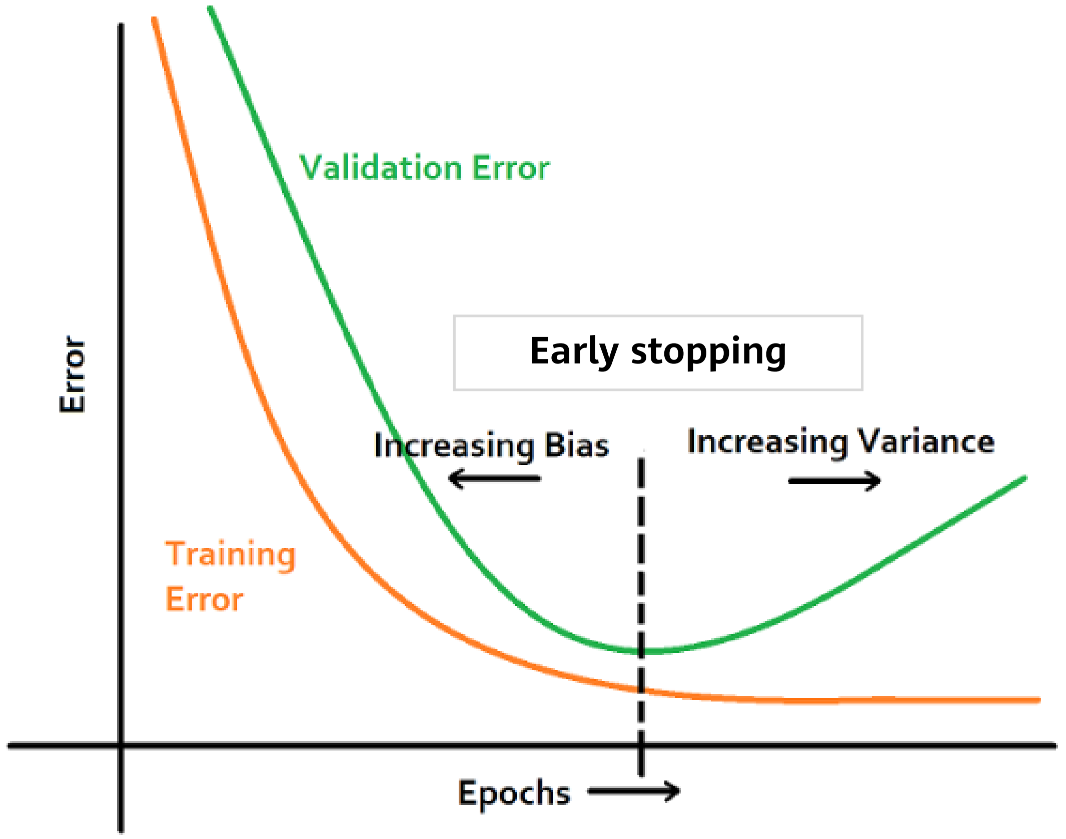
- Dropout is a common and simple regularization method, which has been widely used since 2014. Simply put, Dropout randomly discards some inputs during the training process. In this case, the parameters corresponding to the discarded inputs are not updated. As an integration method, Dropout combines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





Early Stopping

- A test on data of the validation set can be inserted during the training. When the data loss of the verification set increases, perform early stopping.



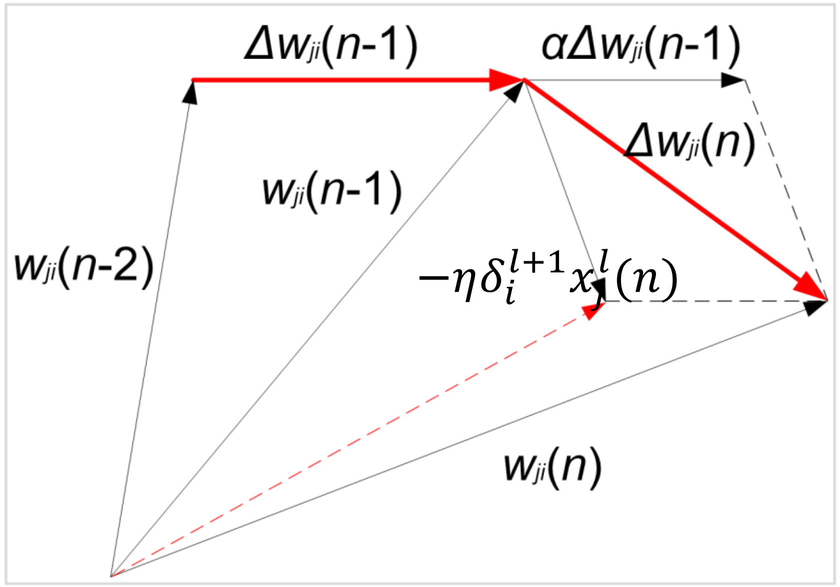
Contents

Optimizer

- There are various optimized versions of gradient descent algorithms. In object-oriented language implementation, different gradient descent algorithms are often encapsulated into objects called optimizers.
- 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.

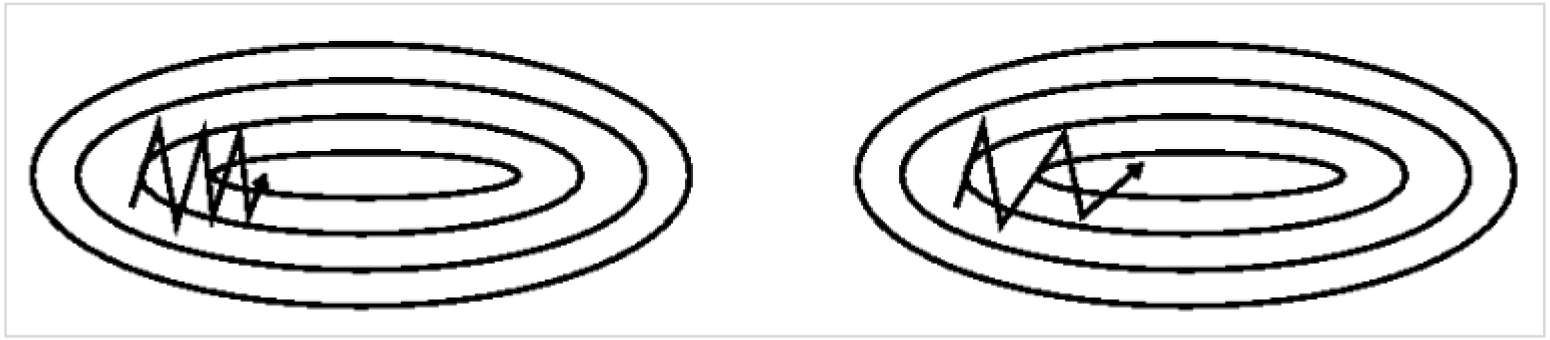
Momentum Optimizer

- A most basic improvement is to add momentum terms for Δw_{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 ($0 \leq \alpha < 1$) called Momentum Coefficient and $\alpha \Delta w_{ji}(n-1)$ 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 η and momentum α need to be manually set, which often requires more experiments to determine the appropriate value.



AdaGrad Optimizer (1)

- The common feature of the random gradient descent algorithm (SGD), 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.

$g_t = \dfrac{\partial C(t,o)}{\partial w_t}$	Gradient calculation
$r_t = r_{t-1} + g_t^2$	Square gradient accumulation
$\Delta w_t = -\dfrac{\eta}{\varepsilon + \sqrt{r_t}} g_t$	Computing update
$w_{t+1} = w_t + \Delta w_t$	Application update

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

AdaGrad Optimizer (2)

- The AdaGrad optimization algorithm shows that the r continues increasing while the overall learning rate keeps decreasing as the algorithm iterates. This is because we hope LR to decrease 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 to the optimal solution, and therefore LR can decrease.
- Pros:
  - The learning rate is automatically updated. As the number of updates increases, the learning rate decreases.
- Cons:
  - The denominator keeps accumulating so that 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 that the AdaGrad optimizer ends the optimization process too early. It is suitable for non-stable target handling and has good effects on the RNN.

$$g_t = \frac{\partial C(t,o)}{\partial w_t}$$ Gradient calculation

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

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

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

- g_t indicates the t-th gradient, r is a gradient accumulation variable, and the initial value of r is 0, which **may not increase and needs to be adjusted using a parameter**. β is the attenuation factor, η indicates the global LR, which needs to be set manually. ε 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 calculated 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, which also explains why the method is so named.

Adam Optimizer (2)

- If m_t and v_t are initialized using the zero vector, m_t and v_t are close to 0 during the initial iterations, especially when β_1 and β_2 are close to 1. To solve this problem, we use $\hat{m}_t$ and $\hat{v}_t$:

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

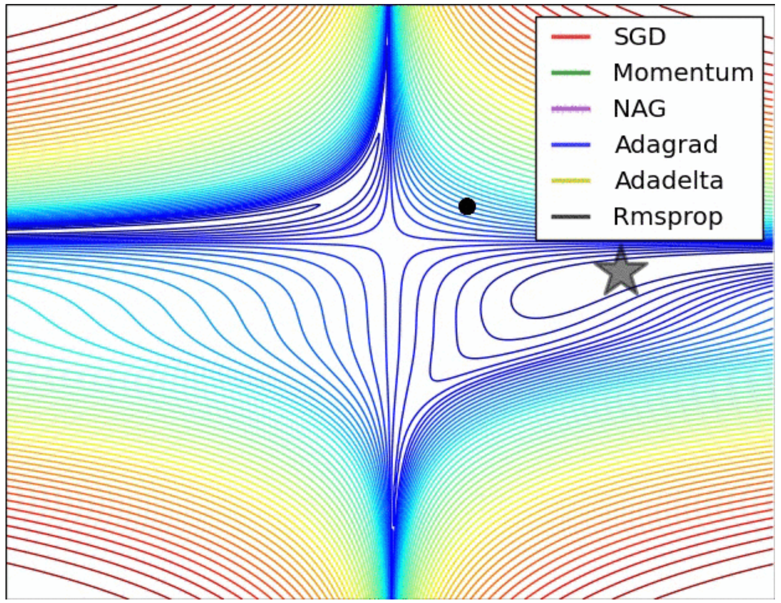
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

- The weight update rule of Adam is as follows:

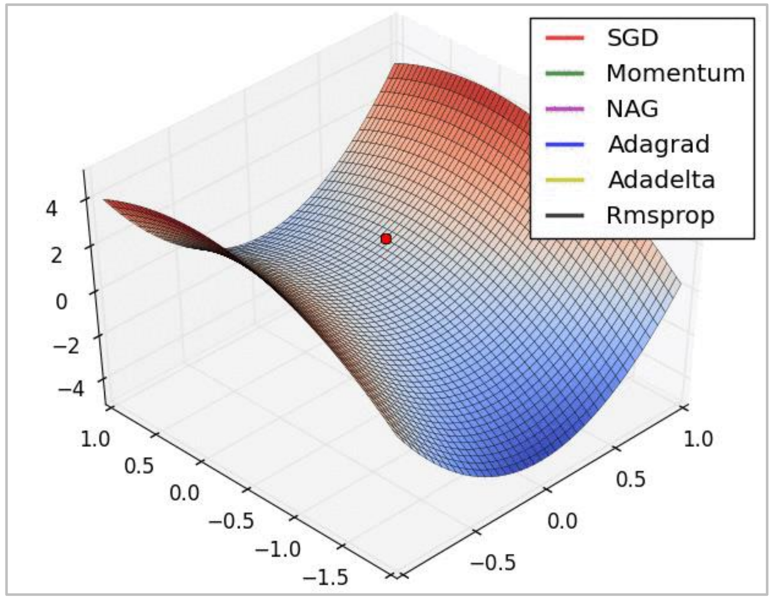
$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t$$

- Although the rule involves manual setting of η, β_1, and β_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 extremum will be obtained. Other parameters do not need to be adjusted.

Optimizer Performance Comparison



Comparison of optimization algorithms in contour maps of loss functions



Comparison of optimization algorithms at the saddle point

Contents

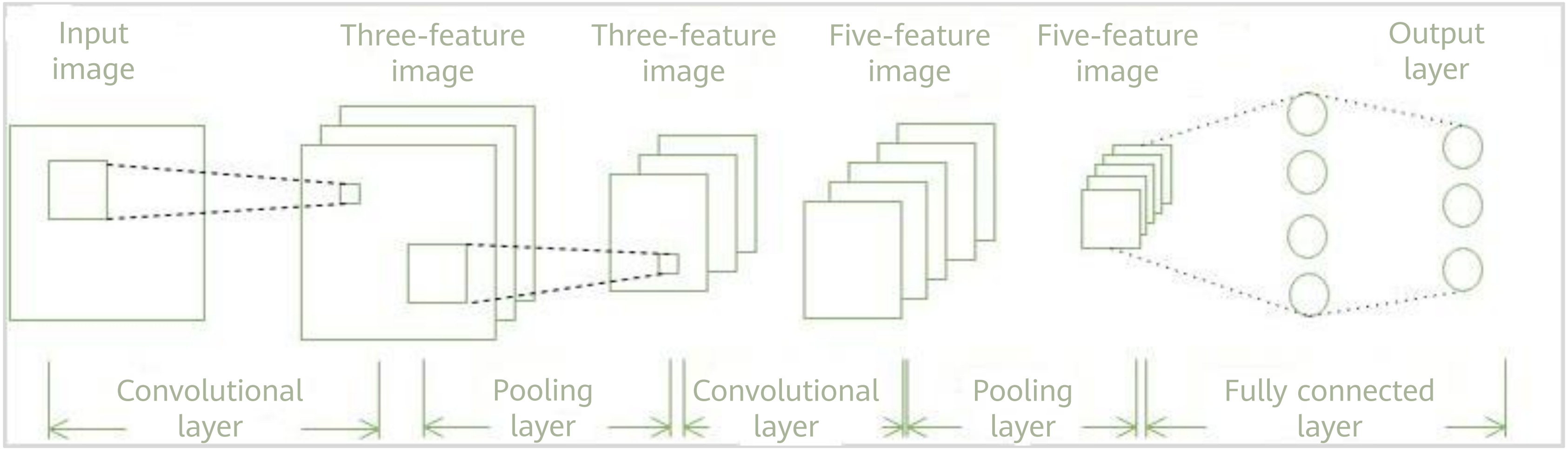
Convolutional Neural Network

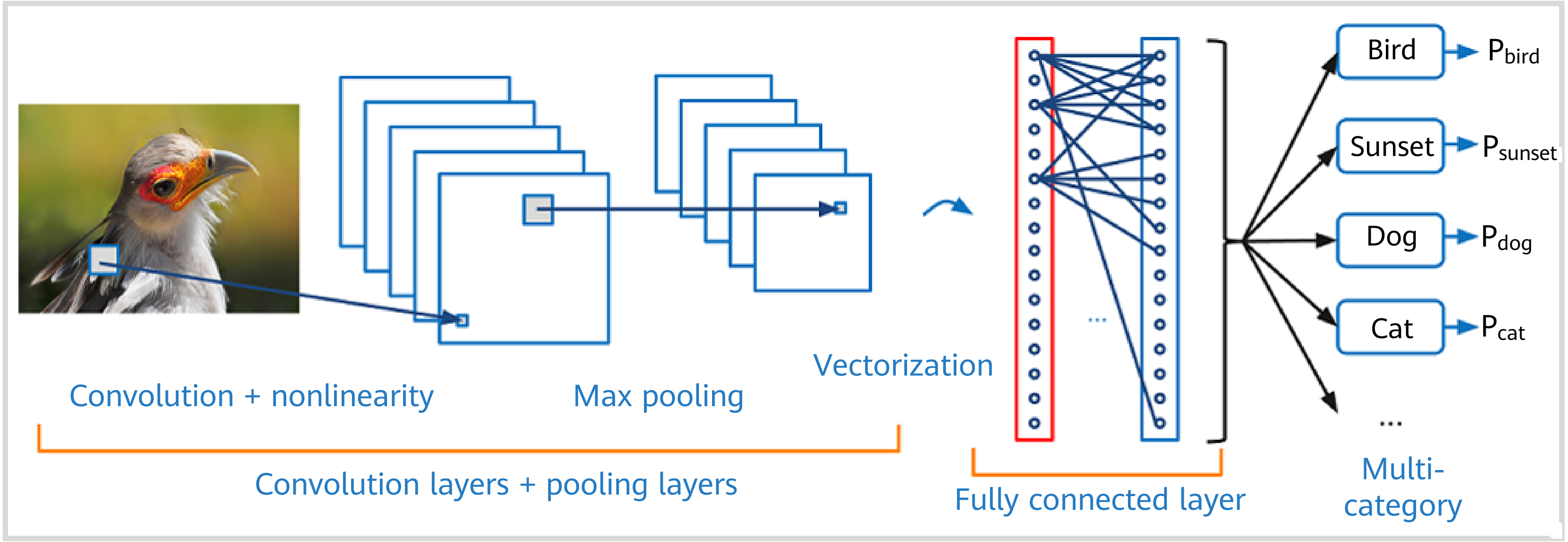
- A convolutional neural network (CNN) is a feedforward neural network. Its artificial neurons may respond to **surrounding units within the coverage range**. CNN excels at image processing. It includes a **convolutional layer**, a **pooling layer**, and a **fully connected layer**.
- In the 1960s, Hubel 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 distant pixels**. Therefore, each neuron does not need to know the global image. It only needs to know the local image. The local information is combined at a higher level to generate global information.
- **Parameter sharing:** One or more filters/kernels may be used to scan input images. Parameters carried by the filters are weights. In a layer scanned by filters, each filter uses the same parameters during weighted computation. Weight sharing means that when each filter scans an entire image, parameters of the filter are fixed.

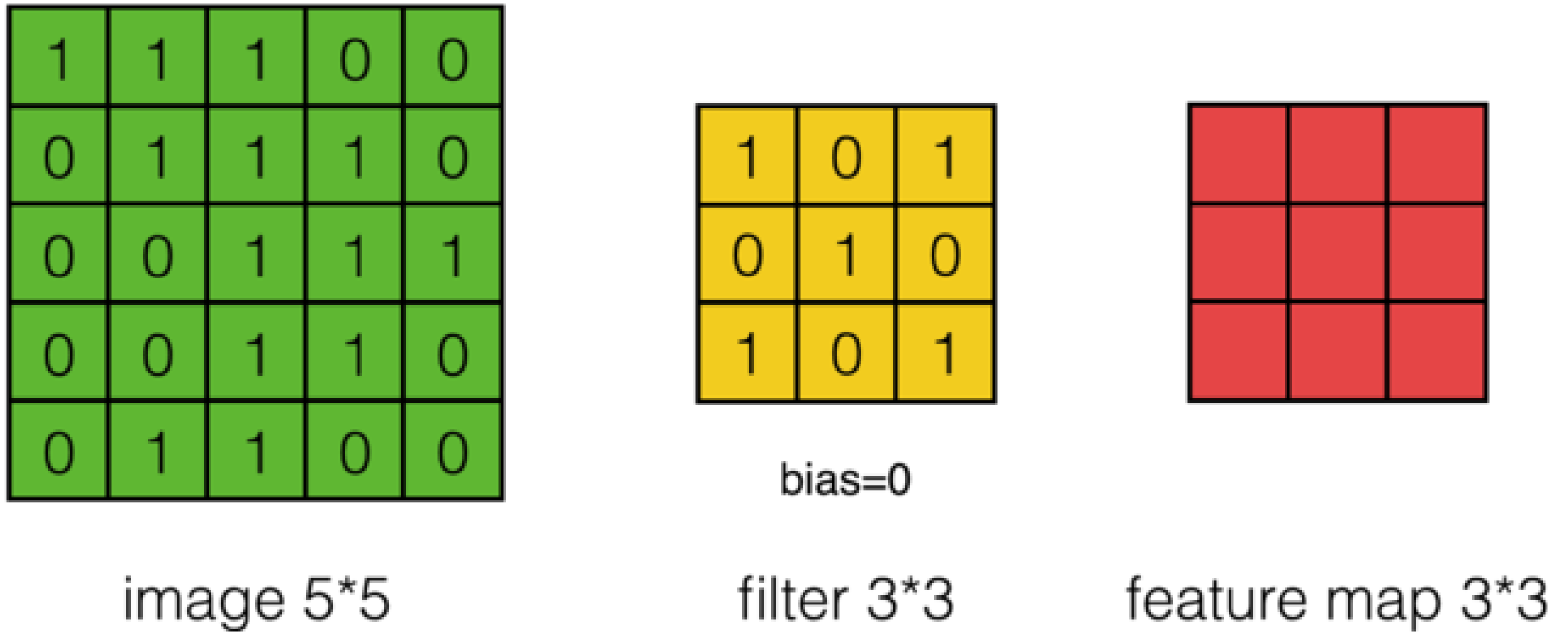
Architecture of Convolutional Neural Network





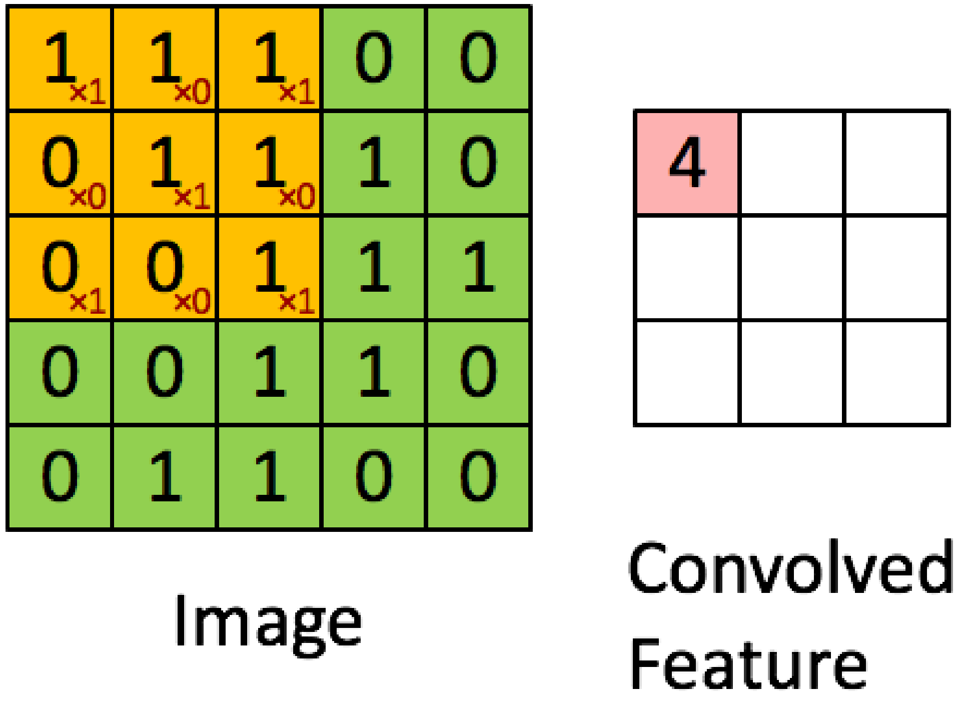
Single-Filter Calculation (1)

- Description of convolution calculation



Single-Filter Calculation (2)

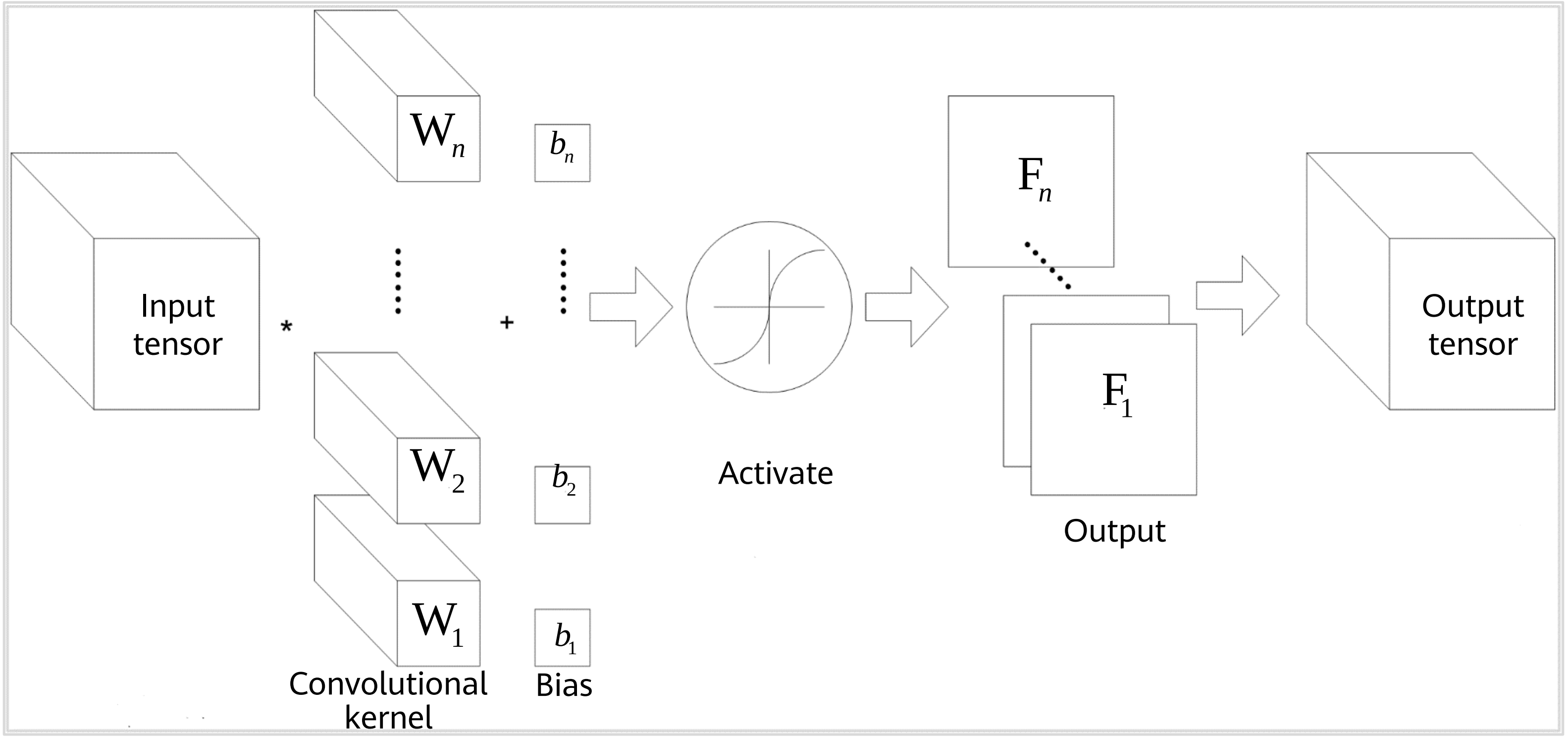
- Demonstration of the convolution calculation



Han Bingtao, 2017, Convolutional Neural Network

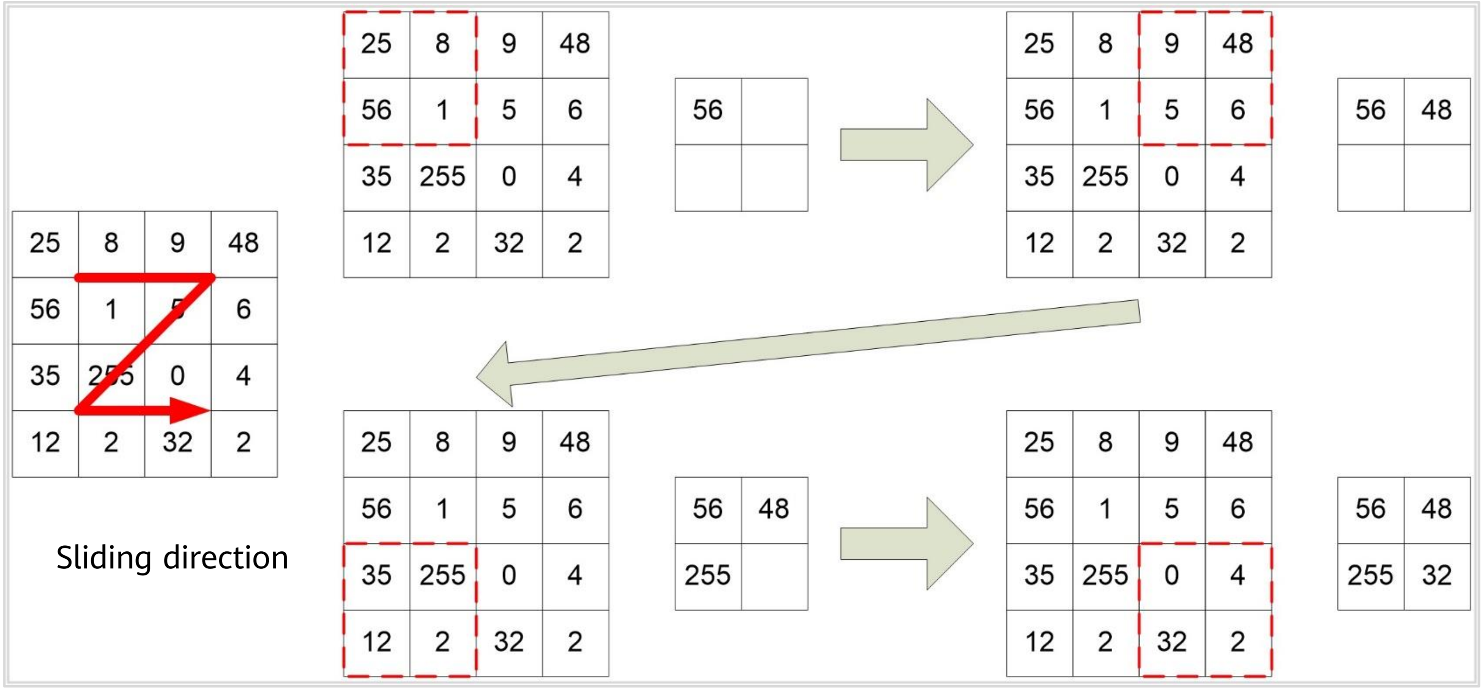
Convolutional Layer

- The basic architecture of a CNN is 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 filter in the layer and serves as the output of this layer. The convolution kernel of each layer is the weight to be learned. Similar to FCN, after the convolution is complete, the result should be 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, while 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 Connected 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 calculate the score of each type.

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

Recurrent Neural Network

- The recurrent neural network (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 other forward neural networks, 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 also 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.

Recurrent Neural Network Architecture (1)

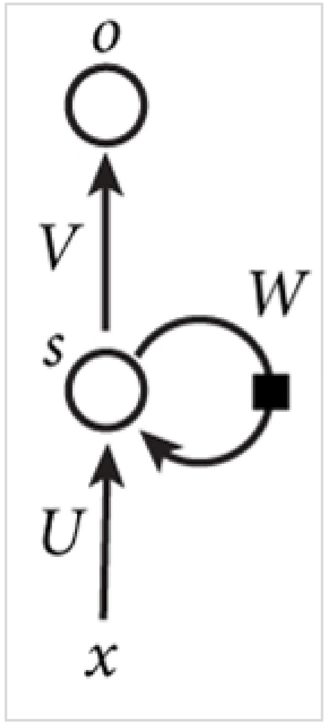
- X_t is the input of the input sequence at time t.
- S_t is the memory unit of the sequence at time t and caches previous information.

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

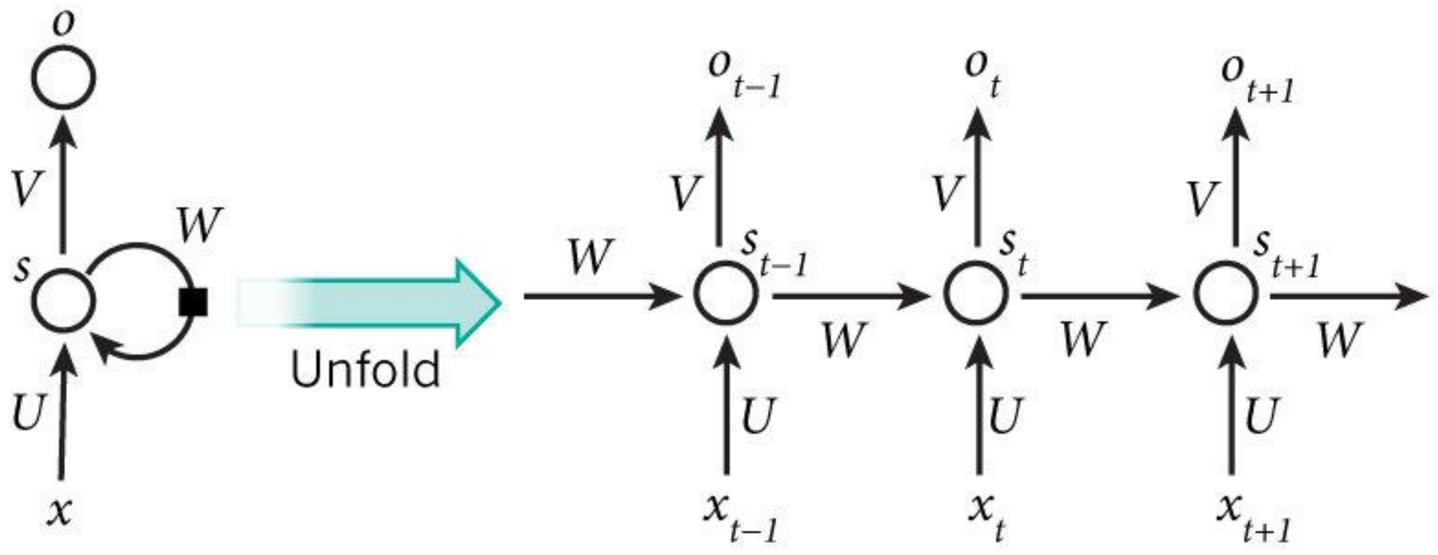
- O_t is the output of the hidden layer of the sequence at time t.

$$O_t = tanh(VS_t)$$

- O_t after through multiple hidden layers, it can get the final output of the sequence at time t.

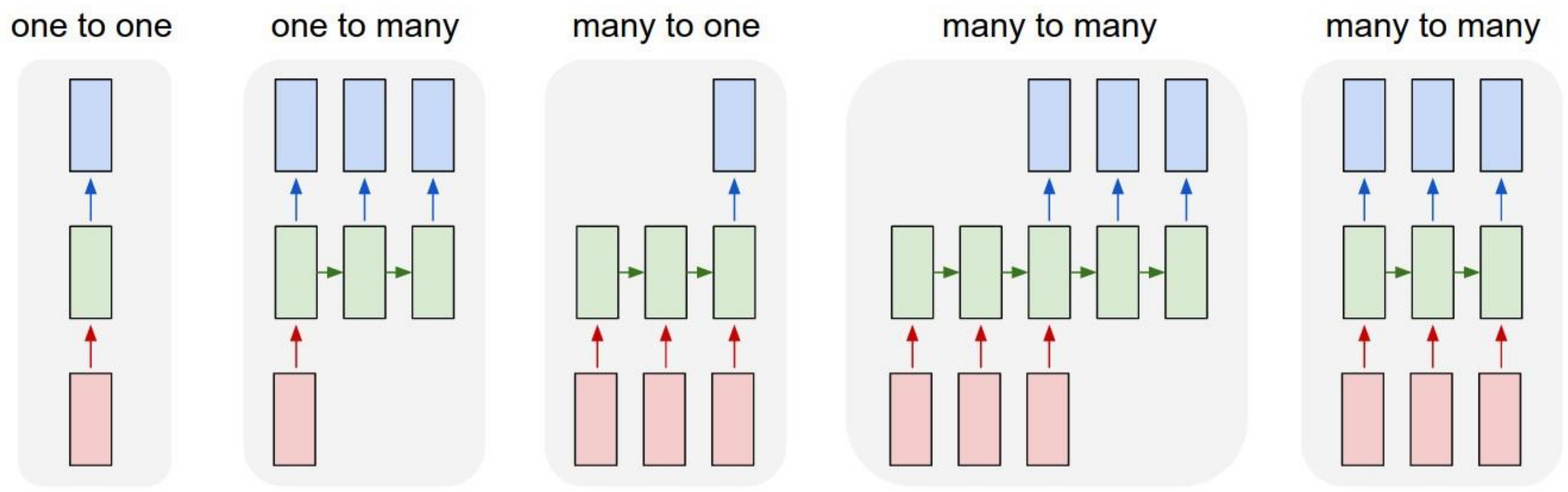


Recurrent Neural Network Architecture (2)



LeCun, Bengio, and G. Hinton, 2015, A Recurrent Neural Network and the Unfolding in Time of the Computation Involved in Its Forward Computation

Types of Recurrent Neural Networks



Andrej Karpathy, 2015, The Unreasonable Effectiveness of Recurrent Neural Networks

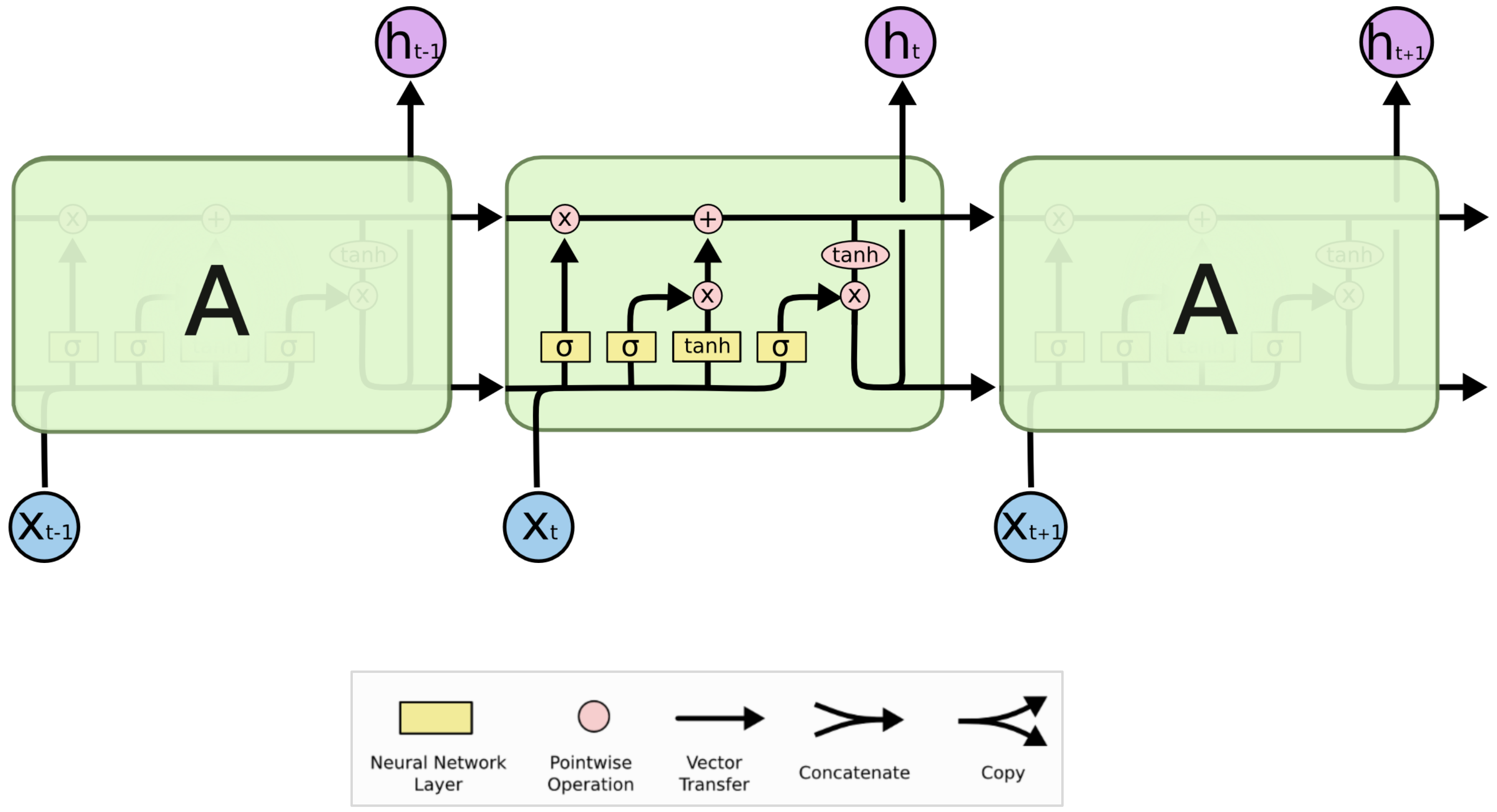
Backpropagation Through Time (BPTT)

- BPTT:
  - Traditional backpropagation is the extension on the time sequence.
  - There are two sources of errors in the sequence at time of memory unit: first is from the hidden layer output error at t time sequence; the second is the error from the memory cell at the next time sequence t + 1.
  - The longer the time sequence, the more likely the loss of the last time sequence to the gradient of w in the first time sequence causes the vanishing gradient or exploding gradient problem.
  - The total gradient of weight w is the accumulation of the gradient of the weight at all time sequence.
- Three steps of BPTT:
  - Computing the output value of each neuron through forward propagation.
  - Computing the error value of each neuron through backpropagation δ_j.
  - Computing the gradient of each weight.
- Updating weights using the SGD algorithm.

Recurrent Neural Network Problem

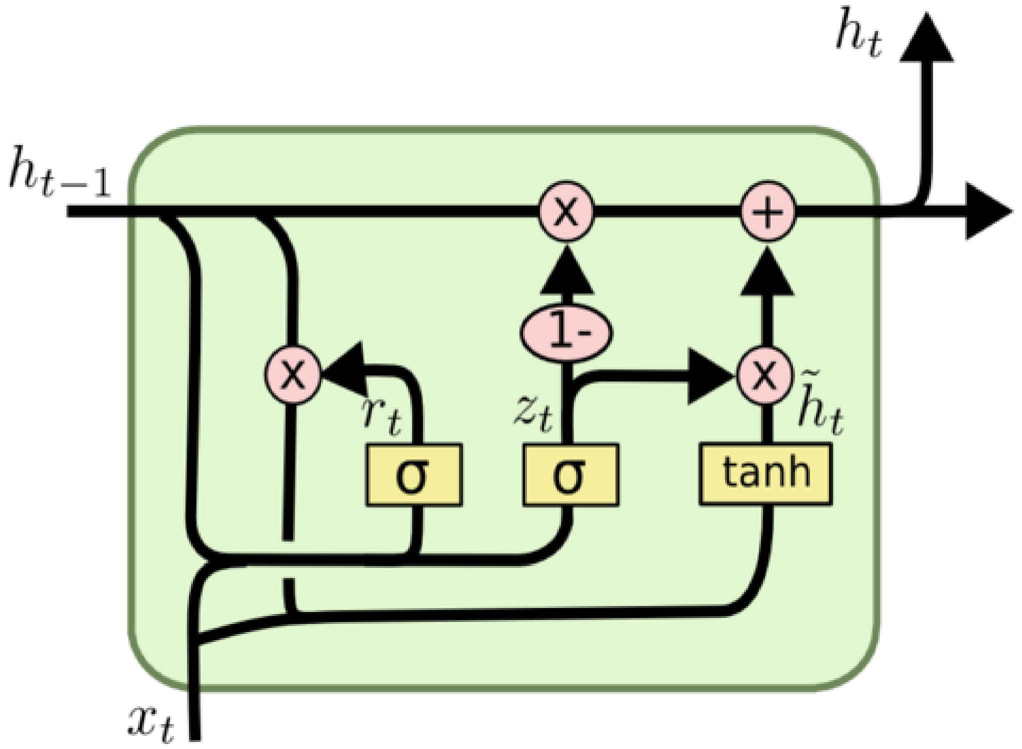
- $S_t = \sigma(UX_t + WS_{t-1})$ is extended on the time sequence.
- $S_t = \sigma\left(UX_t + W\left(\sigma\left(UX_{t-1} + W\left(\sigma(UX_{t-2} + W(\dots))\right)\right)\right)\right)$
- Despite that the standard RNN structure solves the problem of information memory, the information attenuates during long-term memory.
- Information needs to be saved long time in many tasks. For example, a hint at the beginning of a speculative fiction may not be answered until the end.
- The RNN may not be able to save information for long due to the limited memory unit capacity.
- We expect that memory units can remember key information.

Long Short-term Memory Network



Colah, 2015, Understanding LSTMs Networks

Gated Recurrent Unit (GRU)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

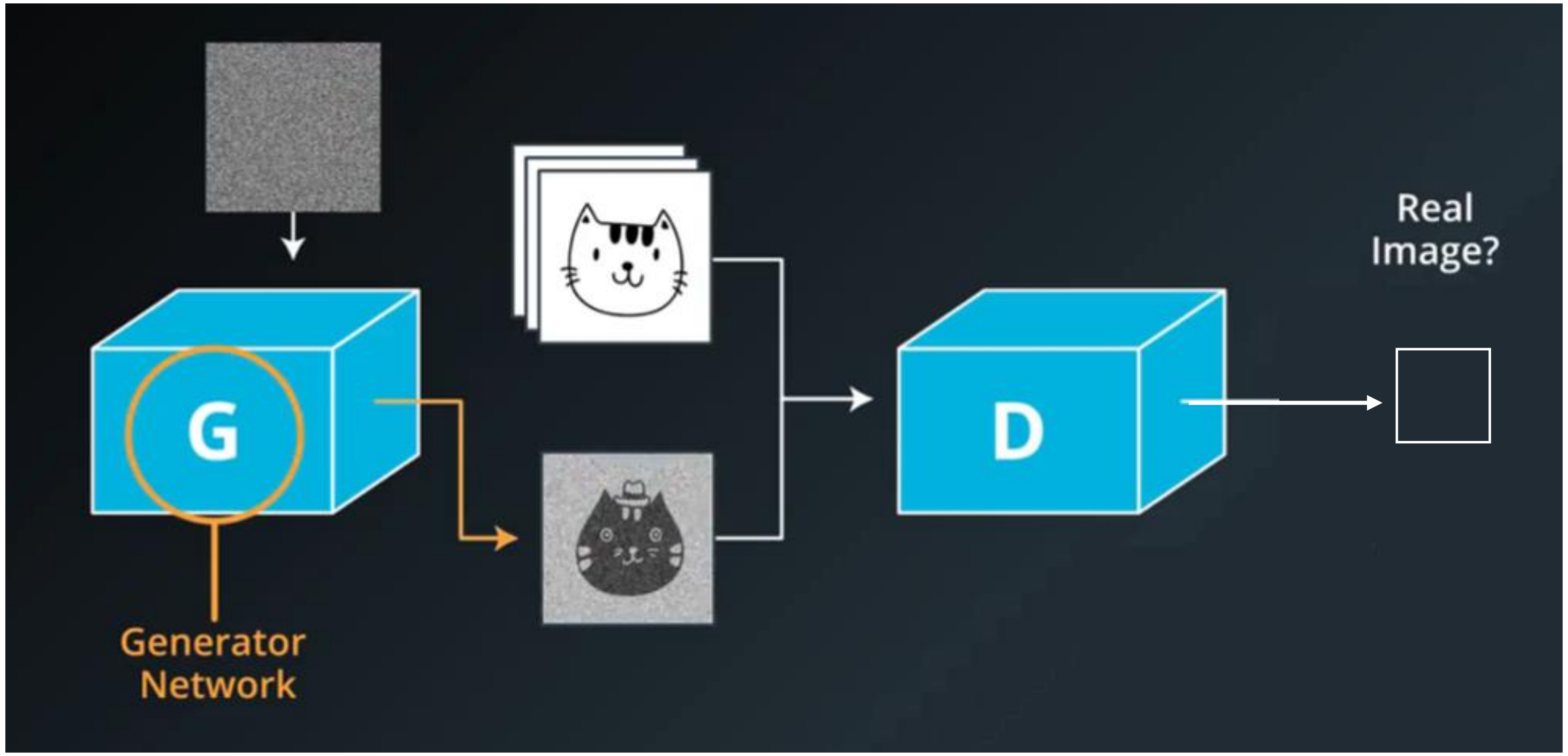
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Generative Adversarial Network (GAN)

- Generative Adversarial Network is a framework that trains generator G and discriminator D through the adversarial process. Through the adversarial process, the discriminator can tell whether the sample from the generator is fake or real. GAN adopts a mature BP algorithm.
- (1) Generator G: The input is noise z, which complies with manually selected prior probability distribution, such as even distribution and Gaussian distribution. The generator adopts the network structure of the multilayer perceptron (MLP), uses maximum likelihood estimation (MLE) parameters to represent the derivable mapping G(z), and maps the input space to the sample space.
- (2) Discriminator D: The input is the real sample x and the fake sample G(z), which are tagged as real and fake respectively. The network of the discriminator can use the MLP carrying parameters. The output is the probability D(G(z)) that determines whether the sample is a real or fake sample.
- GAN can be applied to scenarios such as image generation, text generation, speech enhancement, image super-resolution.

GAN Architecture

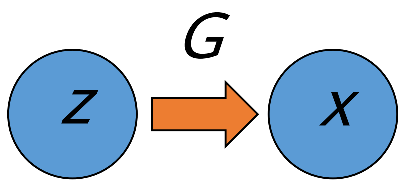
- Generator/Discriminator



Generative Model and Discriminative Model

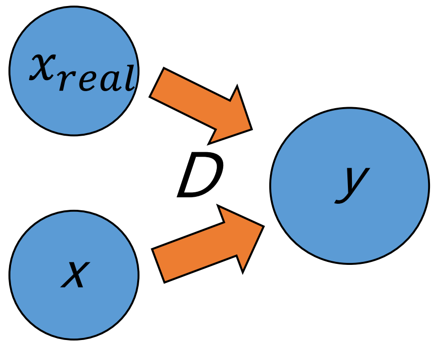
- Generative network
  - Generates sample data
    - Input: Gaussian white noise vector z
    - Output: sample data vector x

$$x = G(z;\theta^G)$$



- Discriminator network
  - Determines whether sample data is real
    - Input: real sample data x_{real} and generated sample data $x = G(z)$
    - Output: probability that determines whether the sample is real

$$y = D(x;\theta^D)$$



Training Rules of GAN

- Optimization objective:
  - Value function

$$\min_{G} \max_{D} V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_{z(z)}}[\log(1-D(G(z)))]$$

  - In the early training stage, when the outcome of G is very poor, D determines that the generated sample is fake with high confidence, because the sample is obviously different from training data. In this case, log(1-D(G(z))) is saturated (where the gradient is 0, and iteration cannot be performed). Therefore, we choose to train G only by minimizing [-log(D(G(z))].

Contents

Data Imbalance (1)

- Problem description: In the dataset consisting of various task categories, the number of samples varies greatly from one category to another. One or more categories in the predicted categories contain very few samples.
- For example, in an image recognition experiment, more than 2,000 categories among a total of 4251 training images contain just one image each. Some of the others have 2-5 images.
- Impacts:
  - Due to the unbalanced number of samples, we cannot get the optimal real-time result because model/algorithm never examines categories with very few samples adequately.
  - Since few observation objects may not be representative for a class, we may fail to obtain adequate samples for verification and test.

Data Imbalance (2)

Random undersampling

- Deleting redundant samples in a category

Random oversampling

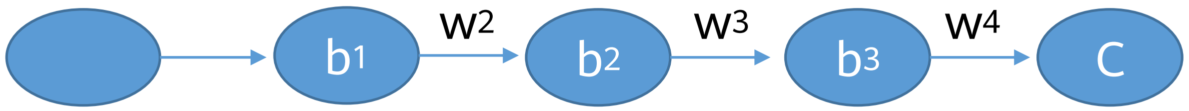
- Copying samples

Synthetic Minority Oversampling Technique

- Sampling
- Merging samples

Vanishing Gradient and Exploding Gradient Problem (1)

- Vanishing gradient: As network layers increase, the derivative value of backpropagation decreases, which causes a vanishing gradient problem.
- Exploding gradient: As network layers increase, the derivative value of backpropagation increases, which causes an exploding gradient problem.
- Cause: $y_i = \sigma_{(z_i)} = \sigma(w_i x_i + b_i)$ Where σ is sigmoid function.

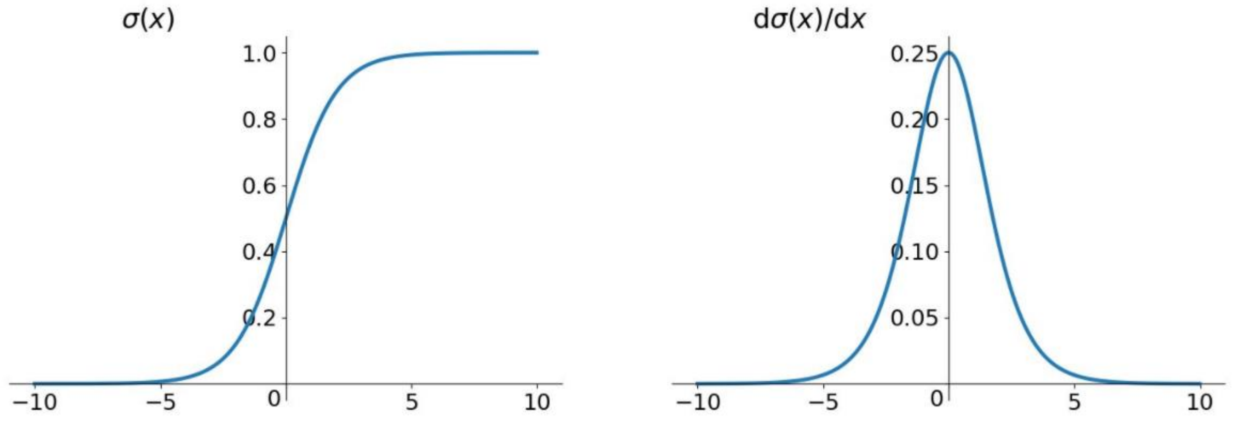


- Backpropagation can be deduced as follows:

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

Vanishing Gradient and Exploding Gradient Problem (2)

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



- However, the network weight $|w|$ is usually smaller than 1. Therefore, $|\sigma'(z)w| \leq \frac{1}{4}$. According to the chain rule, as layers increase, the derivation result $\frac{\partial C}{\partial b_1}$ decreases, resulting in the vanishing gradient problem.
- When the network weight $|w|$ is large, resulting in $|\sigma'(z)w| > 1$, the exploding gradient problem occurs.
- Solution: For example, gradient clipping is used to alleviate the exploding gradient problem, ReLU activation function and LSTM are used to alleviate the vanishing gradient problem.

Overfitting

- Problem description: The model performs well in the training set, but badly in the test set.
- Root cause: There are too many feature dimensions, model assumptions, and parameters, too much noise, but very few training data. As a result, the fitting function perfectly predicts the training set, while the prediction result of the test set of new data is poor. Training data is over-fitted without considering generalization capabilities.
- Solution: For example, data augmentation, regularization, early stopping, and dropout

Summary

- This chapter describes the definition and development of neural networks, perceptrons and their training rules, common types of neural networks (CNN, RNN, and GAN), and the Common Problems of neural networks in AI engineering and solutions.