



مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY
BY EVERYONE'S SMART UNIVERSITY

HCIA-AI Huawei

Lecture 7

Deep Learning Overview

Prepared by
Eng. Eman Elgallad

www.midocean.ae

Lecture

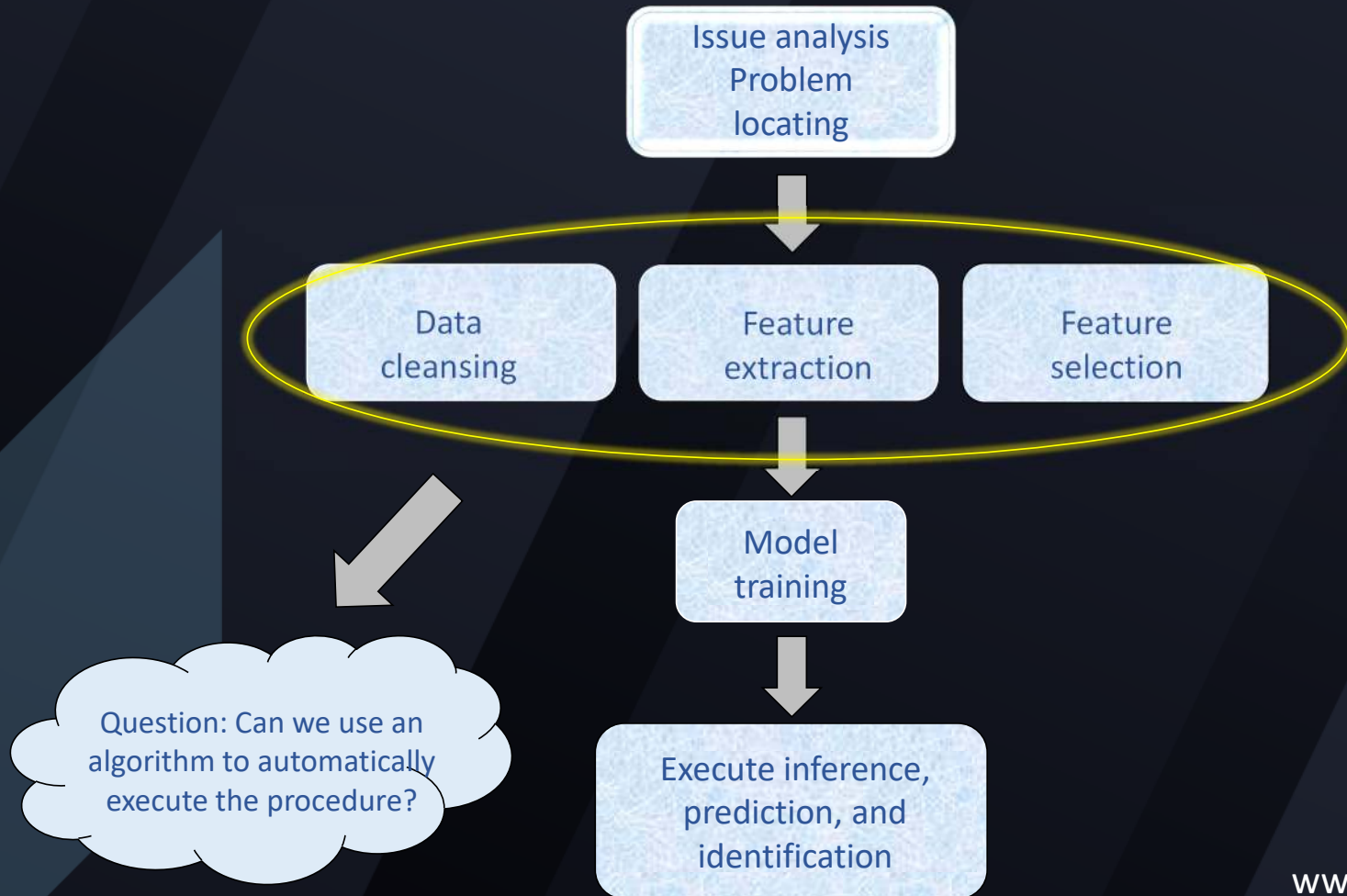
07

Deep Learning

Deep learning is a subfield of machine learning, and it has gained significant importance and popularity in recent years due to its ability to tackle complex tasks that were challenging for traditional machine learning algorithms.

Lecture
07

Traditional Machine Learning



Traditional Machine Learning and Deep Learning

Traditional Machine Learning

Low hardware requirements on the computer:
Given the limited computing amount, the computer does not need a GPU for parallel computing generally.

Applicable to training under a small data amount and whose performance cannot be improved continuously as the data amount increases.

Level-by-level problem breakdown

Manual feature selection

Easy-to-explain features

Deep Learning

Higher hardware requirements on the computer: To execute matrix operations on massive data, the computer needs a GPU to perform parallel computing.

The performance can be high when high-dimensional weight parameters and massive training data are provided.

E2E learning

Algorithm-based automatic feature extraction

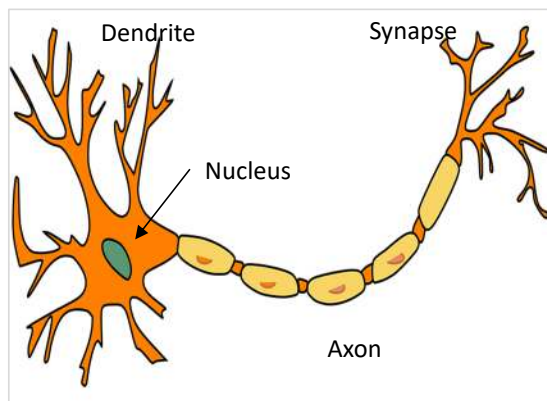
Hard-to-explain features

Why We Need Deep Learning:

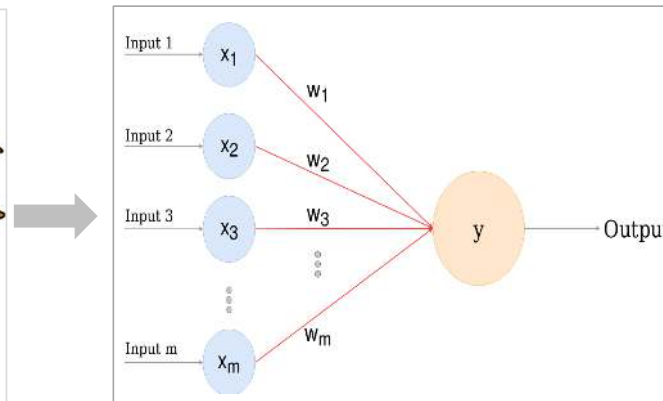
1. **Handling Complex Data:** Deep learning excels at processing and understanding complex data types such as images, audio, text, and sequences. Which required in various applications like image recognition, speech recognition, natural language processing, and even autonomous driving.
2. **Feature Extraction:** In traditional machine learning, feature engineering is a manual and time-consuming process. Deep learning, on the other hand, can automatically learn relevant features from raw data, reducing the need for feature engineering.
3. **Scalability:** Deep learning models can scale with the amount of data and computational resources available. This scalability makes them suitable for handling large datasets and complex problems.
4. **Hierarchical Representation:** Deep neural networks learn hierarchical representations of data. Lower layers capture simple patterns, while higher layers combine these patterns to learn more abstract and complex features. This makes deep learning models capable of capturing intricate relationships in data.
5. **State-of-the-Art Performance:** Deep learning has achieved state-of-the-art performance in various tasks, including image classification, object detection, machine translation, and game playing. This has led to its adoption in industry and academia.

Deep Learning Is Deep?

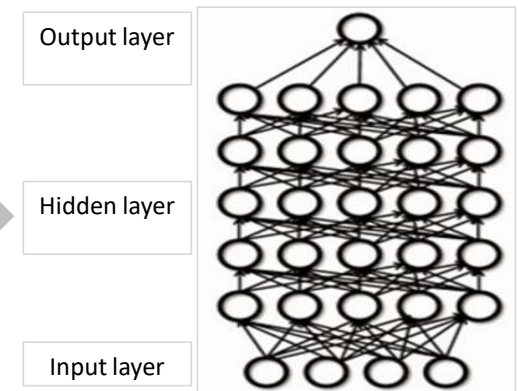
- Generally, the deep learning architecture is a deep neural network. "Deep" in "deep learning" refers to the number of layers of the neural network.



Human neural network



Perceptron

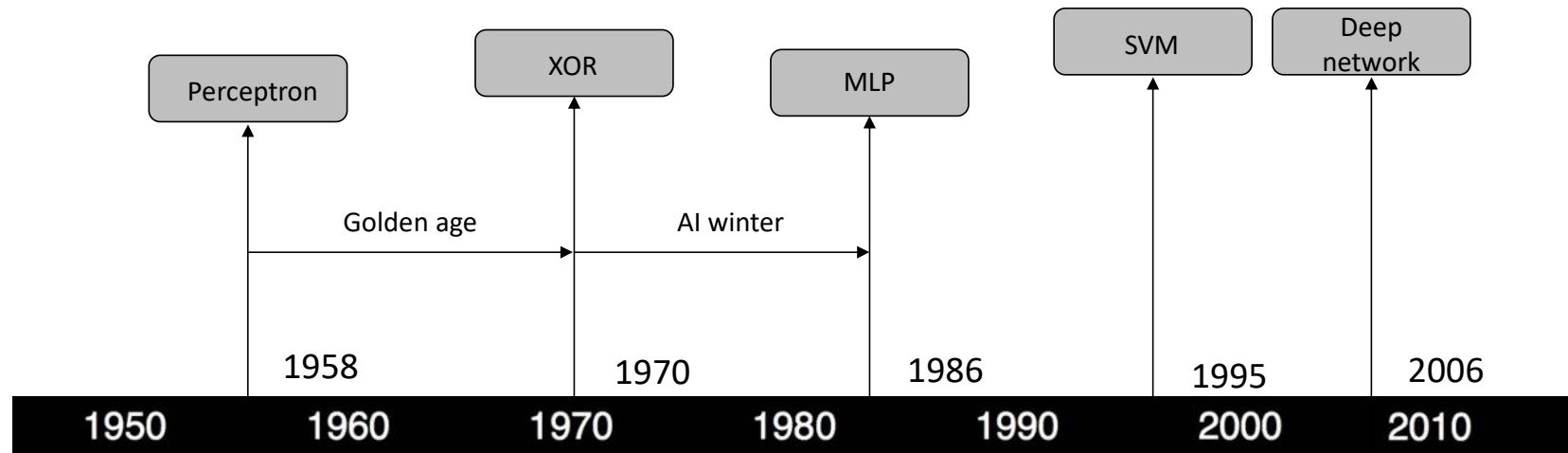


Deep neural network

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



Perceptron

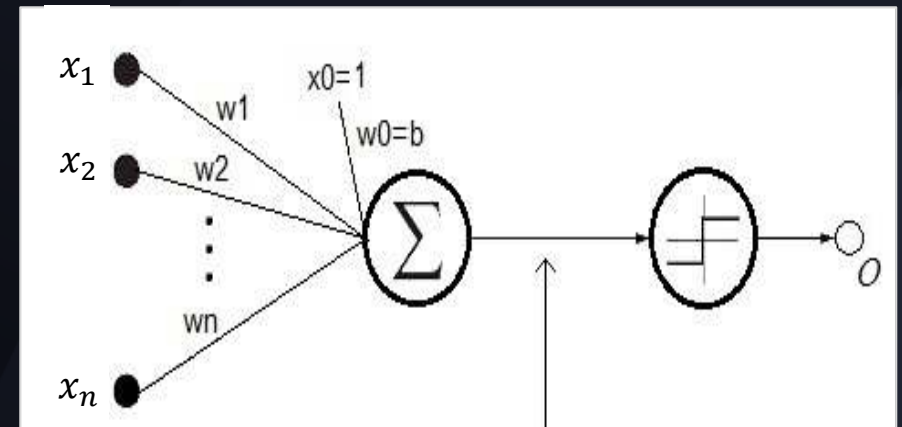
Lecture

07

First, let's take a look at the simplest neural network — single-layer perceptron.

In 1958, Frank Rosenblatt invented the perceptron algorithm. Such an algorithm coexisted with machine learning for more than a decade.

.



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

Single-Layer Perceptron

A single perceptron is a linear binary classifier. It takes a weighted sum of its inputs and applies an activation function (usually a step function or threshold) to produce an output.

- 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: $O = \text{sign}(\text{net}) = \begin{cases} 1, & \text{net} > 0, \\ -1, & \text{otherwise.} \end{cases}$

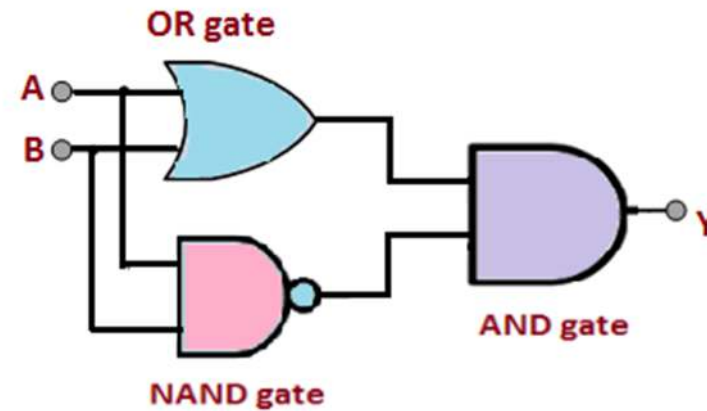
Mathematically, the output of a perceptron (y) can be represented as:

$$y = \text{activation}(w_1 * x_1 + w_2 * x_2 + \dots + w_n * x_n + \text{bias})$$

XOR

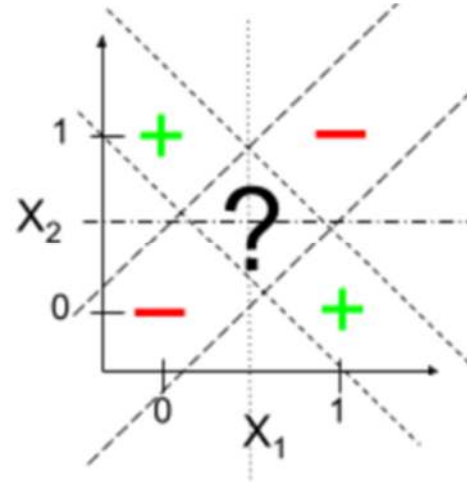
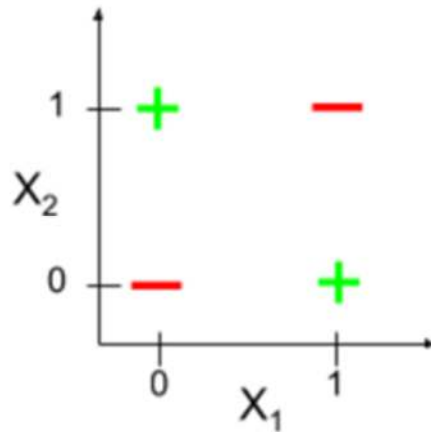
Input A | Input B | Output (XOR)

0		0		0
0		1		1
1		0		1
1		1		0



In 1969, Marvin Minsky, pioneer of artificial intelligence in the US, questioned that the perceptron could handle only linear classification problems and failed to classify even the simplest XOR problems. The research on the perceptron was doomed to failure

MLP



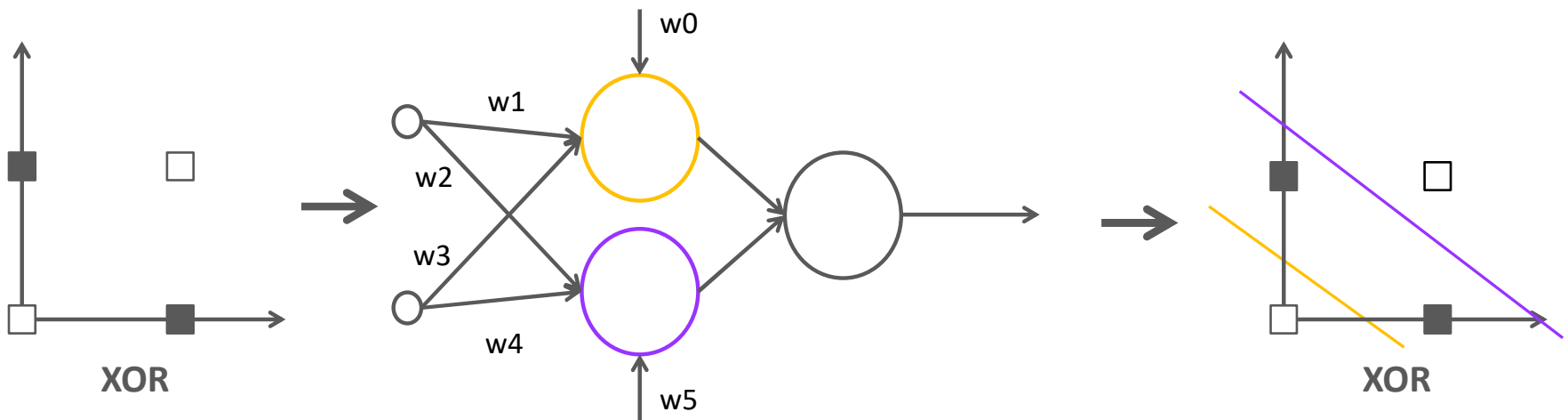
Solution: Multi-Layer Perceptron (MLPs):

To solve the XOR problem and other non-linearly separable problems, we need to use multi-layer perceptron (MLPs), also known as feedforward neural networks with hidden layers.

Hidden layers introduce non-linearity into the model by applying activation functions to their weighted sums. This allows the network to learn complex patterns and decision boundaries that single perceptron cannot capture.

By using one or more hidden layers with appropriate activation functions (such as sigmoid or ReLU), an MLP can learn to approximate the XOR function effectively, making it capable of solving this and a wide range of other non-linear problems.

Solution of XOR



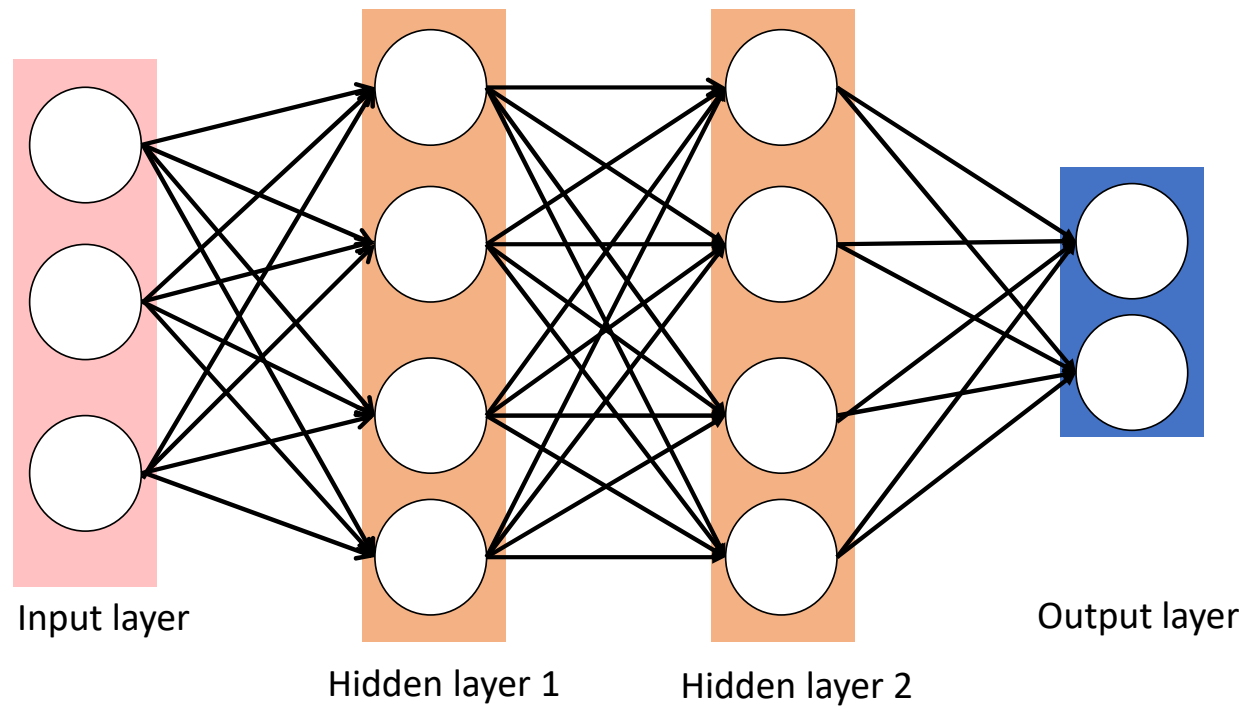
Steps Involved in Applying Deep Learning:

1. **Problem Definition:** Clearly define the problem you want to solve. Identify the type of data you're working with (e.g., images, text, time series) and the desired outcome (e.g., classification, regression, generation).
2. **Data Collection:** Gather a high-quality dataset that is relevant to your problem. Ensure the data is properly labeled and preprocessed. This might involve data cleaning, normalization, and splitting into training, validation, and test sets.
3. **Model Selection:** Choose an appropriate deep learning architecture (e.g., convolutional neural networks for images, recurrent neural networks for sequences, transformer models for NLP). Consider factors like model size, complexity, and available computational resources.
4. **Model Design:** Create the neural network architecture by defining the layers, activation functions, and connections. Design the input and output layers based on your problem's requirements.

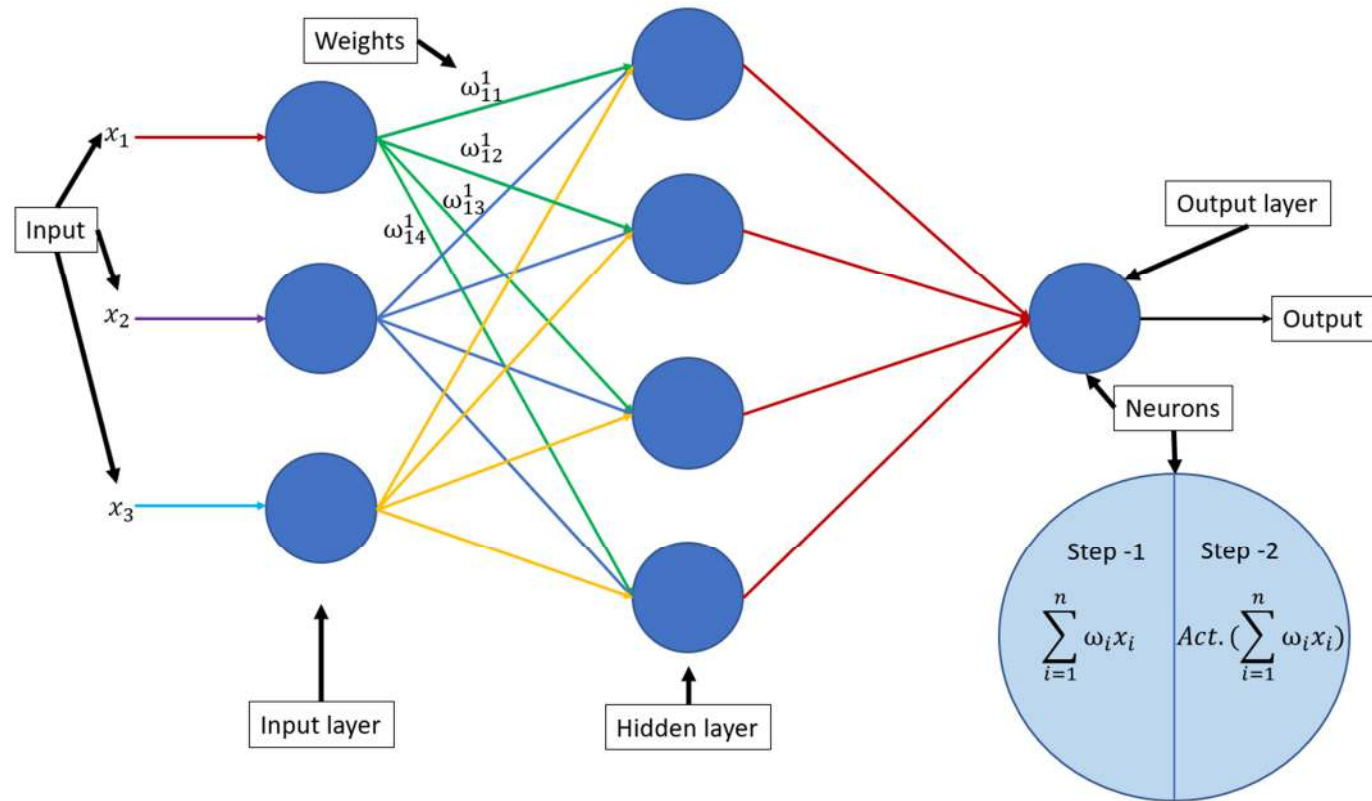
Steps Involved in Applying Deep Learning cont.:

5. **Training:** Train the model using the training dataset. This involves forward and backward passes, gradient descent optimization, and updating the model's weights to minimize a loss function. Monitor the model's performance on the validation set to avoid overfitting.
6. **Hyperparameter Tuning:** Fine-tune hyperparameters such as learning rate, batch size, and regularization techniques to optimize the model's performance.
7. **Evaluation:** Assess the model's performance on the test dataset to ensure it generalizes well to unseen data. Use appropriate evaluation metrics for your problem (e.g., accuracy, F1-score, mean squared error).
8. **Deployment:** Once you're satisfied with the model's performance, deploy it in a production environment. This may involve converting the model to a suitable format (e.g., TensorFlow Saved Model, ONNX) and integrating it into your application.
9. **Monitoring and Maintenance:** Continuously monitor the model's performance in the real-world environment and update it as needed to adapt to changing data patterns or requirements.

Feed forward NN:



Feed forward NN:



Feed forward NN:

- The multilayer perceptron is a feedforward neural network.
- It is the simplest neural network with neurons arranged in layers. It is more widely used and develops faster than other artificial neural networks.
- Input nodes do not have the calculation capability and are only used to represent element values of the input vector.
- The neuron having the computing capability at each layer is referred to as the computing unit. Each neuron is connected only to the neurons of the previous layer.
- A unidirectional multi-layer structure is used to receive the output of the previous layer and send the output to the next layer. Each layer includes several neurons, and the neurons at the same layer are not connected to each other. Only unidirectional inter-layer information transmission is supported.

Gradient Descent and Loss Function

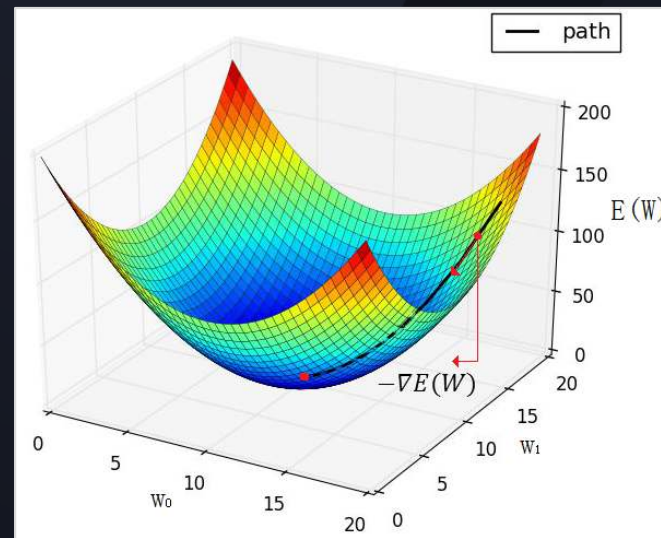
- During the training of the deep learning network, target classification errors must be parameterized. A **loss function (error function)** is used, which reflects the error between the target output and actual output of the perceptron.

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



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:

$$E(W) = - \sum_{d \in D} t_d \ln o_d$$

- 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.



Optimizer

BGD, MBGD and SGD

Gradient Descent: Most optimizers are based on the concept of gradient descent. The gradient of the loss function with respect to the model parameters indicates the direction and magnitude of the steepest increase in the loss. Optimizers use this gradient information to update the parameters in the opposite direction, effectively "descending" the loss function to reach a minimum.

- **Batch Gradient Descent (BGD):** Computes the gradient using the entire training dataset in each iteration.
- **Mini-Batch Gradient Descent:** Computes the gradient using a randomly selected subset (mini-batch) of the training data in each iteration.
- **Stochastic Gradient Descent (SGD):** Computes the gradient using only one training example at a time in each iteration.

Batch Gradient Descent Algorithm (BGD)

- **Update Frequency:** In BGD, the model's weights are updated only once per epoch (a complete pass through the entire training dataset).
- **Computational Efficiency:** It can be slower for large datasets since it processes the entire dataset before updating the model.
- **Convergence:** BGD often converges to the minimum of the cost function more accurately because it computes gradients using the entire dataset.
- **Stability:** It tends to have more stable and smoother convergence due to its use of a consistent gradient estimate.
- **Memory Usage:** BGD requires memory to store the entire dataset and gradients, which can be a drawback for large datasets.

Stochastic Gradient Descent Algorithm (SGD)

- **Update Frequency:** In SGD, the model's weights are updated after each training example (one randomly chosen instance) or a small subset of examples.
- **Computational Efficiency:** It is computationally efficient and often faster per iteration than BGD and mini-batch GD since it processes one or a few examples at a time.
- **Convergence:** SGD can have a noisy and less stable convergence due to frequent updates based on individual examples. It may take longer to converge but can escape local minima better.
- **Stability:** The noise in the updates can make the cost function oscillate, which can be mitigated with learning rate schedules or techniques like momentum.
- **Memory Usage:** It requires minimal memory since it processes only a single example or a small subset of examples at a time.

Mini-Batch Gradient Descent Algorithm (MBGD)

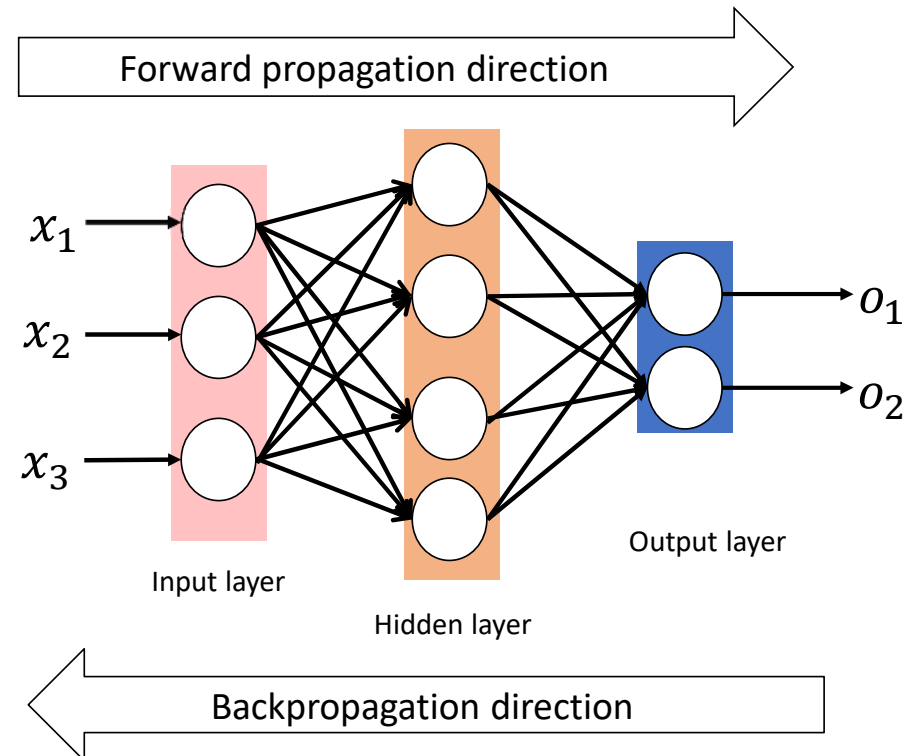
- **Update Frequency:** In mini-batch gradient descent, the training dataset is divided into smaller, equally-sized subsets called mini-batches. The model's weights are updated once per mini-batch.
- **Computational Efficiency:** It strikes a balance between BGD and SGD, making it more computationally efficient than BGD for large datasets while maintaining some of the benefits of stochasticity.
- **Convergence:** It converges faster than BGD since updates are more frequent, and it can escape local minima better due to the stochastic nature of mini-batch sampling.
- **Stability:** Mini-batch updates introduce some randomness, which can help escape poor local minima, but it may also lead to more oscillations in the cost function.
- **Memory Usage:** It requires memory to store mini-batches and gradients, which is less demanding than BGD.

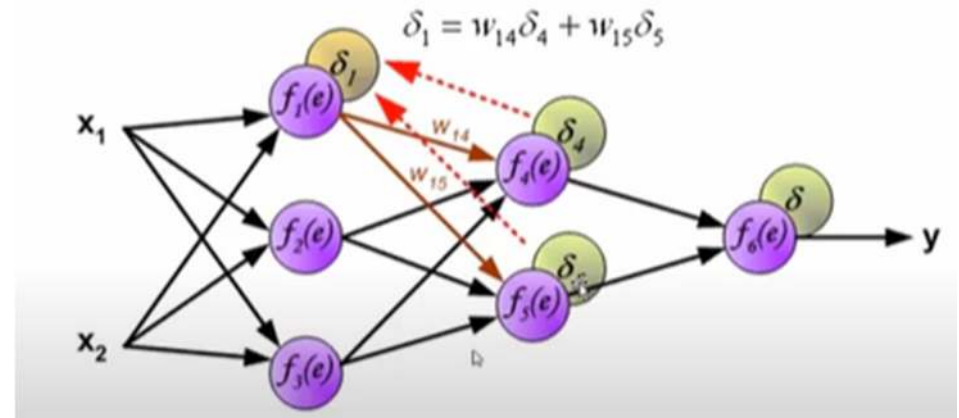
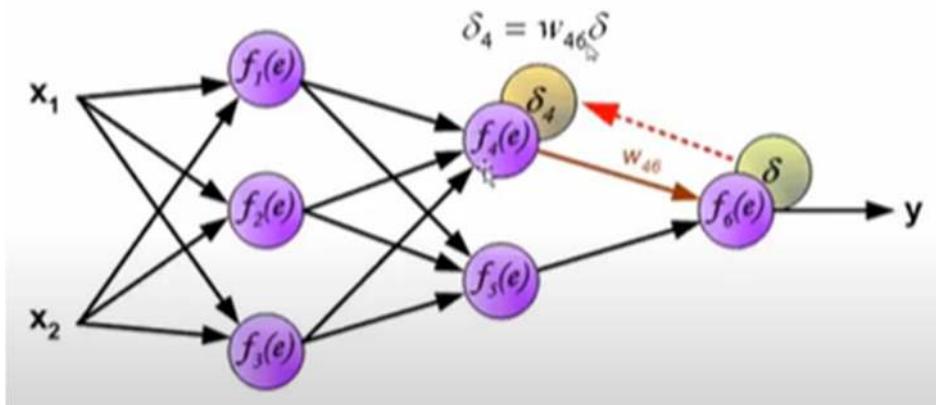
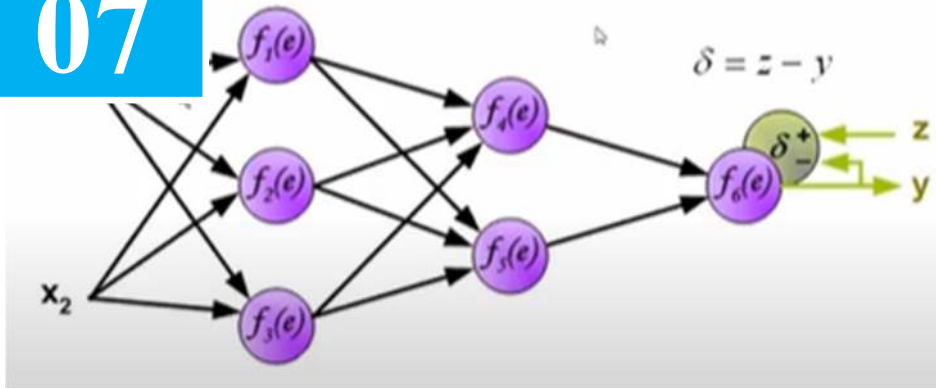
Training

Backpropagation Algorithm

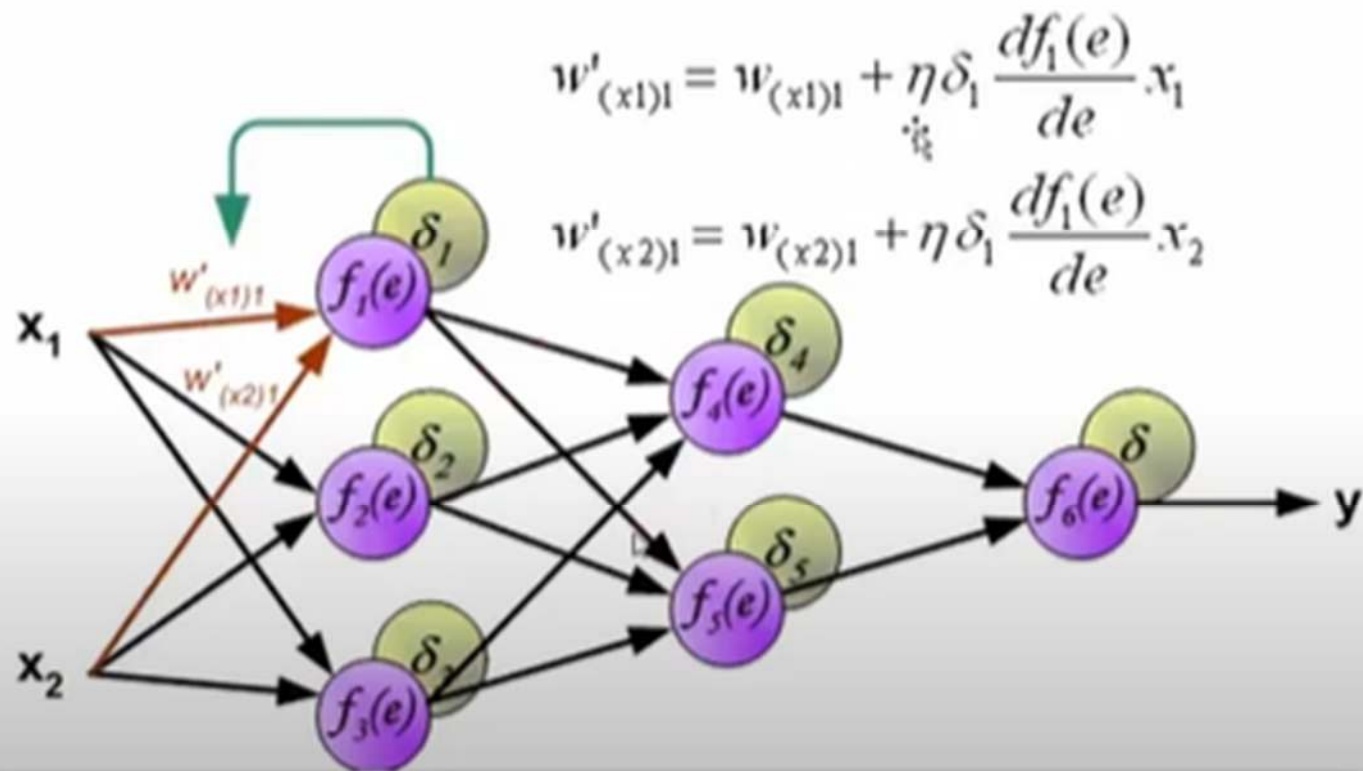
- Signals are propagated in forward direction, and errors are propagated in backward direction.
- In the training sample set D , each sample is recorded as $\langle X, t \rangle$, in which X is the input vector, t the target output, o the actual output, and w the weight coefficient.
- Loss function:

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





Update (W)



Backpropagation Algorithm

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

$$\Delta w_{jk}^l = -\eta \delta_k^{l+1} f_j'(z_j^l)$$

$$\delta_j^l = \begin{cases} f_j'(z_j^l)(t_j - f_j(z_j^l)), l \in \text{outputs}, (1) \\ \sum_k \delta_k^{l+1} w_{jk}^l f_j'(z_j^l), \text{otherwise}, (2) \end{cases}$$

- The BP algorithm is used to train the network as follows:
 - Takes out the next training sample $\langle X, T \rangle$, 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.

Activation Functions

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.

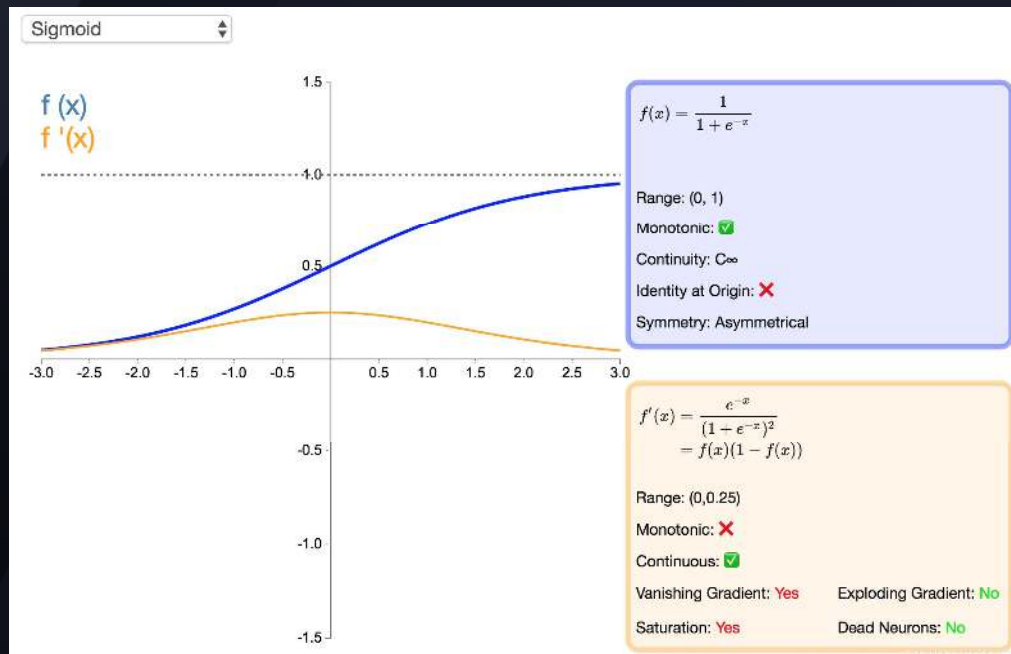
Activation Function

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

Sigmoid

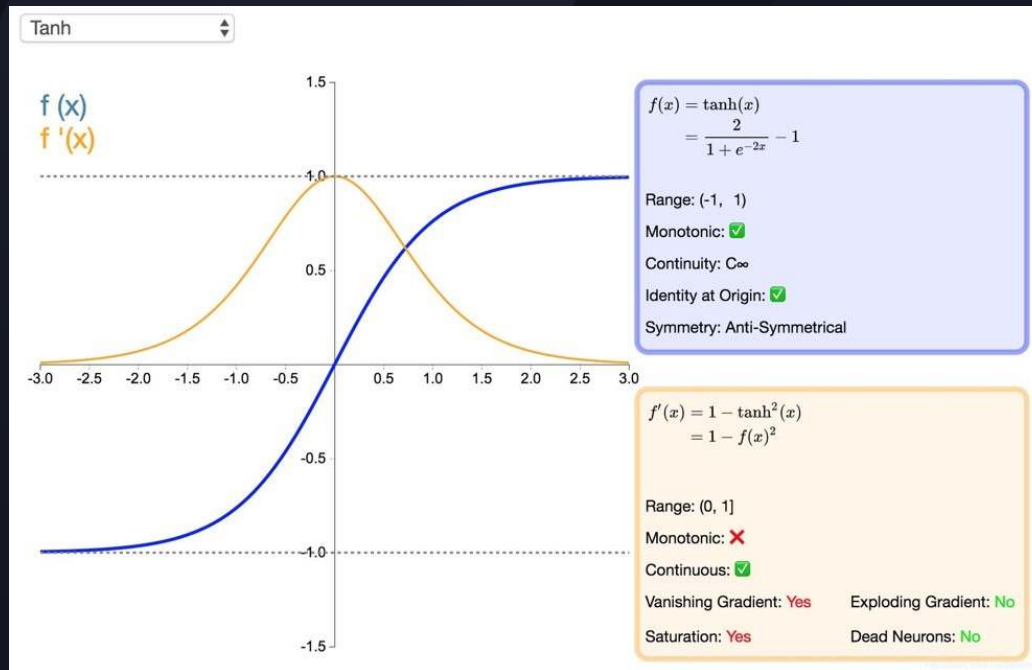
$$f(x) = \frac{1}{1 + e^{-x}}$$

- **Range:** (0, 1)
- **Usage:**
 - **Binary Classification:** The sigmoid function is commonly used in the output layer of binary classification models. It squashes the model's output to a probability between 0 and 1, representing the likelihood of the input belonging to the positive class.



Tanh

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



Tanh

Range: $(-1, 1)$

Usage:

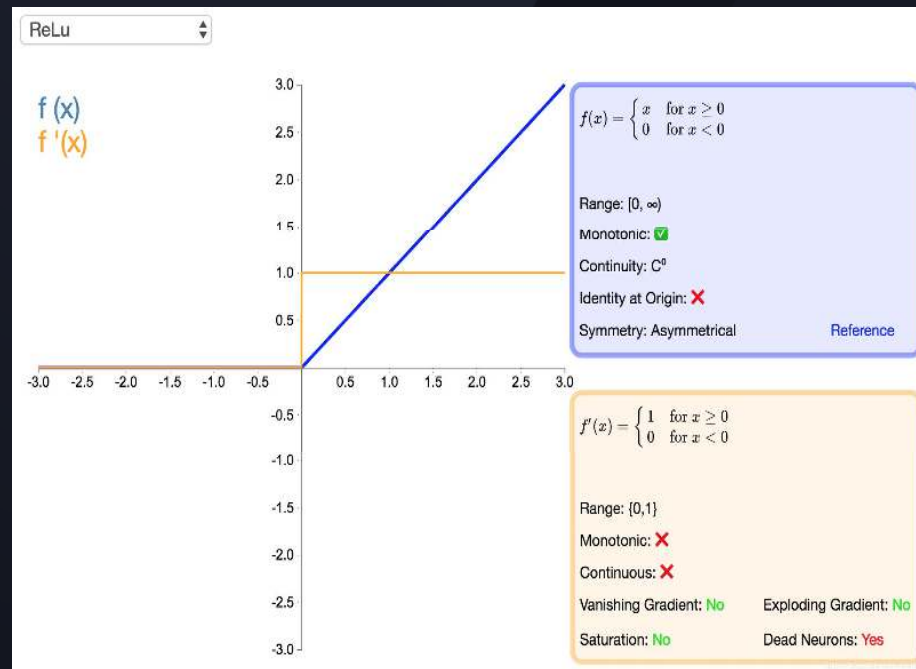
Hidden Layers: Tanh is often used as an activation function in hidden layers of neural networks. Like the sigmoid, it introduces non-linearity into the model. However, because it's zero-centered (unlike sigmoid), it helps mitigate some of the issues with vanishing gradients that can occur in deep networks.

Rescaling Data: Tanh can be used to rescale and center data before feeding it into a neural network. By mapping input features to the $(-1, 1)$ range, it can help improve the convergence and stability of the training process.

Rectified Linear Unit (ReLU)

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

- Range: $[0, \infty)$



Rectified Linear Unit (ReLU) Cont.

- **Usage:**

- **Hidden Layers:** ReLU is a popular activation function for hidden layers in deep neural networks. It introduces non-linearity and helps address the vanishing gradient problem, enabling deep networks to learn complex representations efficiently.
- **Sparse Activation:** ReLU neurons are either fully activated (output > 0) or completely inactive (output = 0), which can lead to sparse activations, making the network more computationally efficient.

Softmax

- Softmax function:

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

- **Range:** (0, 1) for each element, with the sum of all elements in the vector equal to 1.
- **Usage:**
- **Multi-Class Classification:** Softmax is used in the output layer of models for multi-class classification tasks. It converts the model's raw scores (logits) into a probability distribution over multiple classes, allowing you to choose the class with the highest probability as the model's prediction.
- **Cross-Entropy Loss:** Softmax is often paired with the cross-entropy loss function to train classification models effectively.

Regularization

Regularization

- 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 $\tilde{J}$.

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

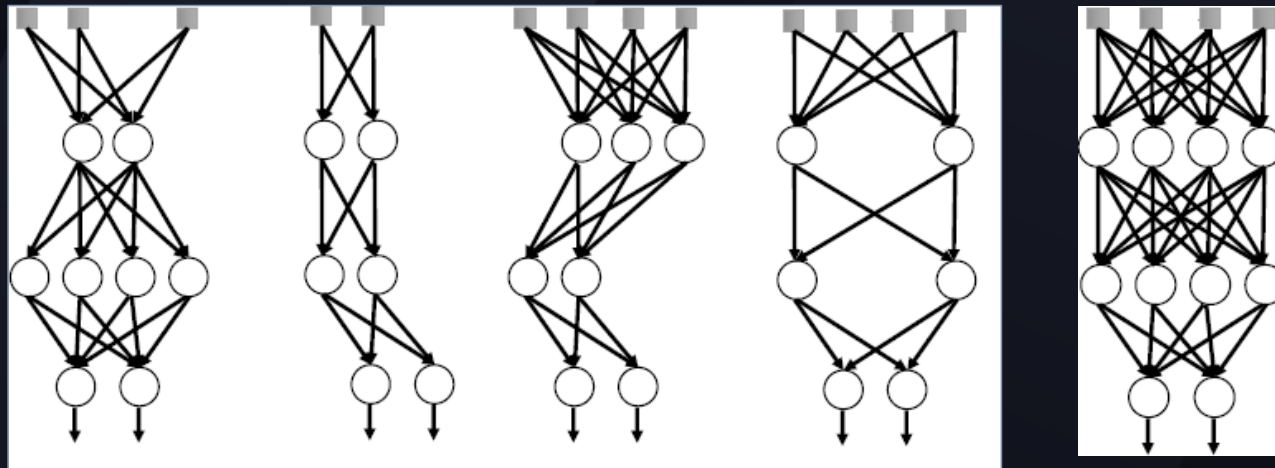
- Where $\alpha \in [0, \infty)$ is a hyperparameter that weights the relative contribution of the norm penalty term Ω and the standard objective function $J(X; \theta)$. If α is set to 0, no regularization is performed. The penalty in regularization increases with α .

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.

Dropout (Integration Method)

- 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:

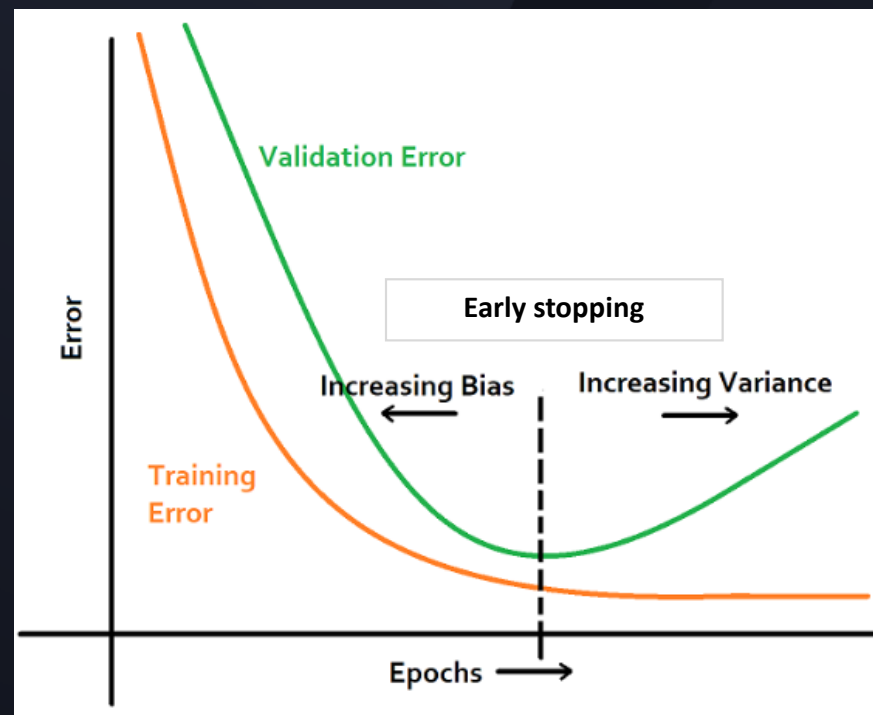


Dropout in training

Testing

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.





مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY
BY EVERYONE'S SMART UNIVERSITY



Thank You

www.midocean.ae