Neural Network Fundamentals

Qing-Yuan Jiang

The school of Computer Science and Engineering, NJUST

2026.10.22

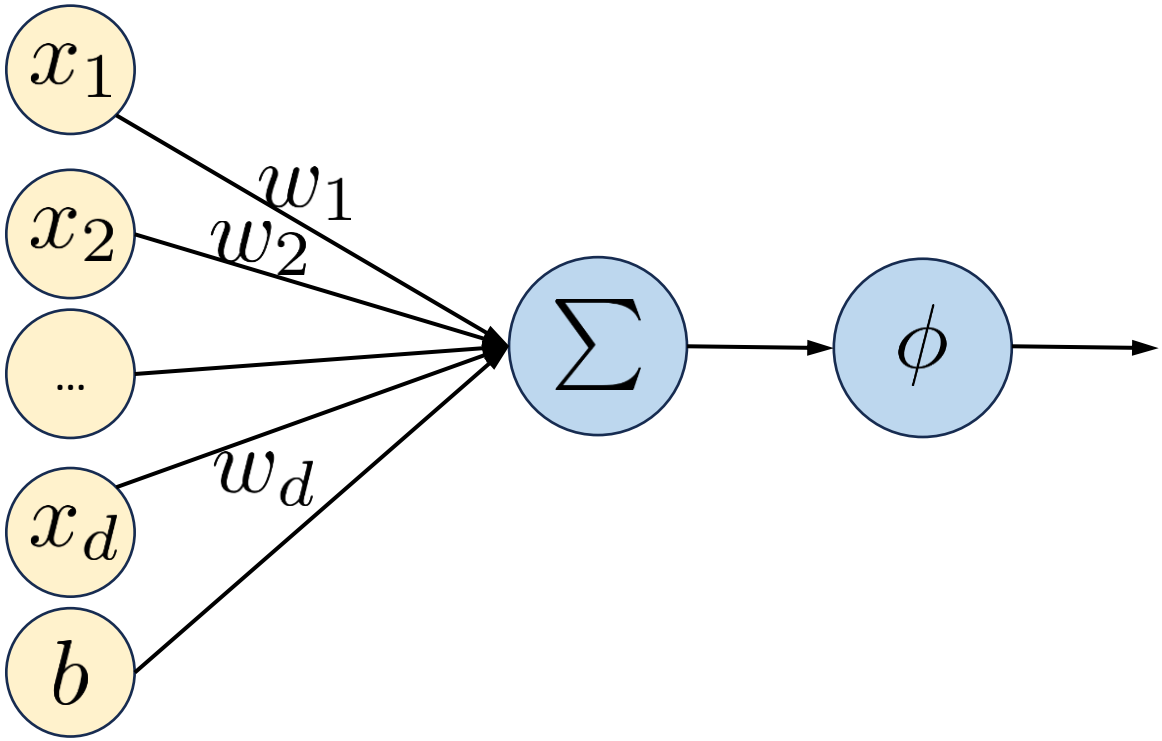
Review of Previous Class

- What is a margin, and how do functional and geometric margins differ?
- Why do we maximize the margin when choosing a decision boundary?
- How do we formulate a hard-margin support vector machine?
- How are the primal and dual formulations of SVM connected?
- Why do we need soft-margin SVM, and what do slack variables represent?
- How does hinge loss connect soft-margin SVM to regularized risk minimization?
- How can feature mapping help us handle nonlinear classification problems?
- How does the kernel trick avoid explicitly computing feature mappings?

From Neurons to Networks

The Artificial Neuron

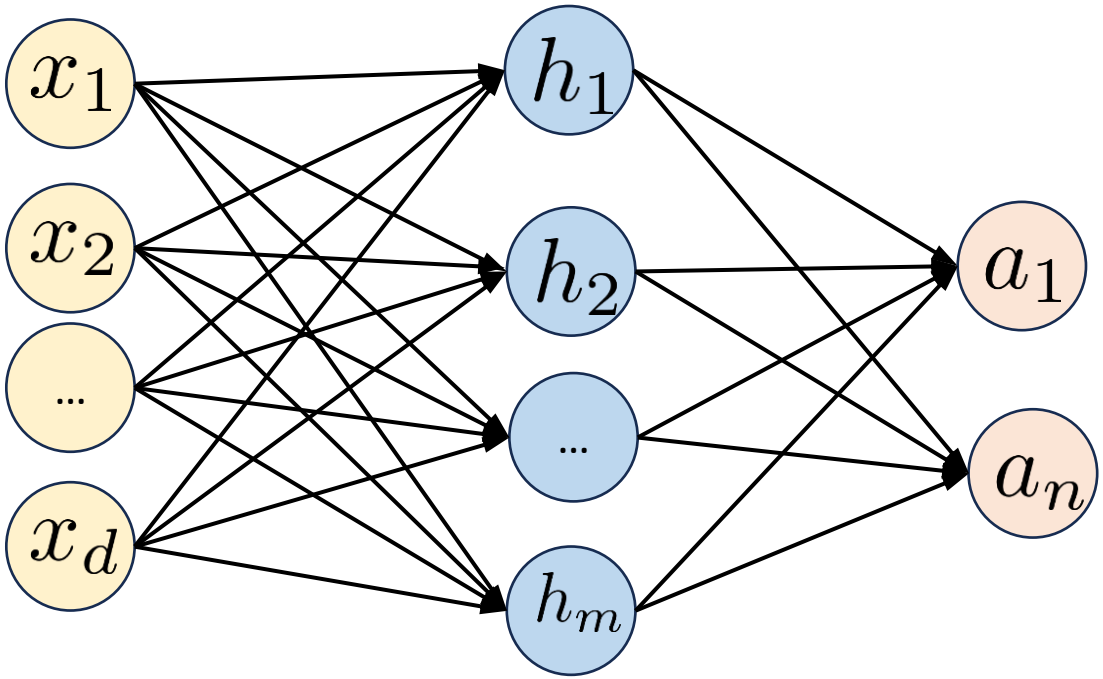
- Inputs $x_1, x_2, \cdots, x_d$ carry features or outputs from a previous layer.
- Weights determine how strongly each input influences the unit.
- A bias shifts the activation threshold, and a nonlinear function produces the output.



A neuron is an affine transformation followed by an activation function.

From a Neuron to a Layer

- A layer evaluates many neurons in parallel.
- Each row of W stores the incoming weights for one neuron.
- The activation is applied elementwise unless stated otherwise.



$$z = Wx + b, \quad a = \phi(z)$$

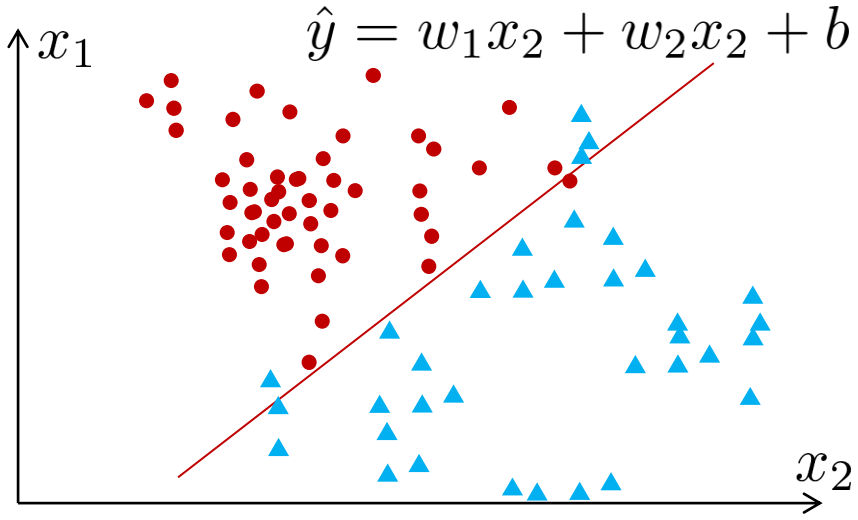
Vectorization turns a collection of neurons into one matrix operation.

Affine Transformation

A hyperplane

$$Wx + b = 0$$

- One side predicts +1
- The other side predicts -1
- Only one straight boundary is available in the original feature space.



Linear separability is a property of the representation as well as the classifier.

Activation Functions

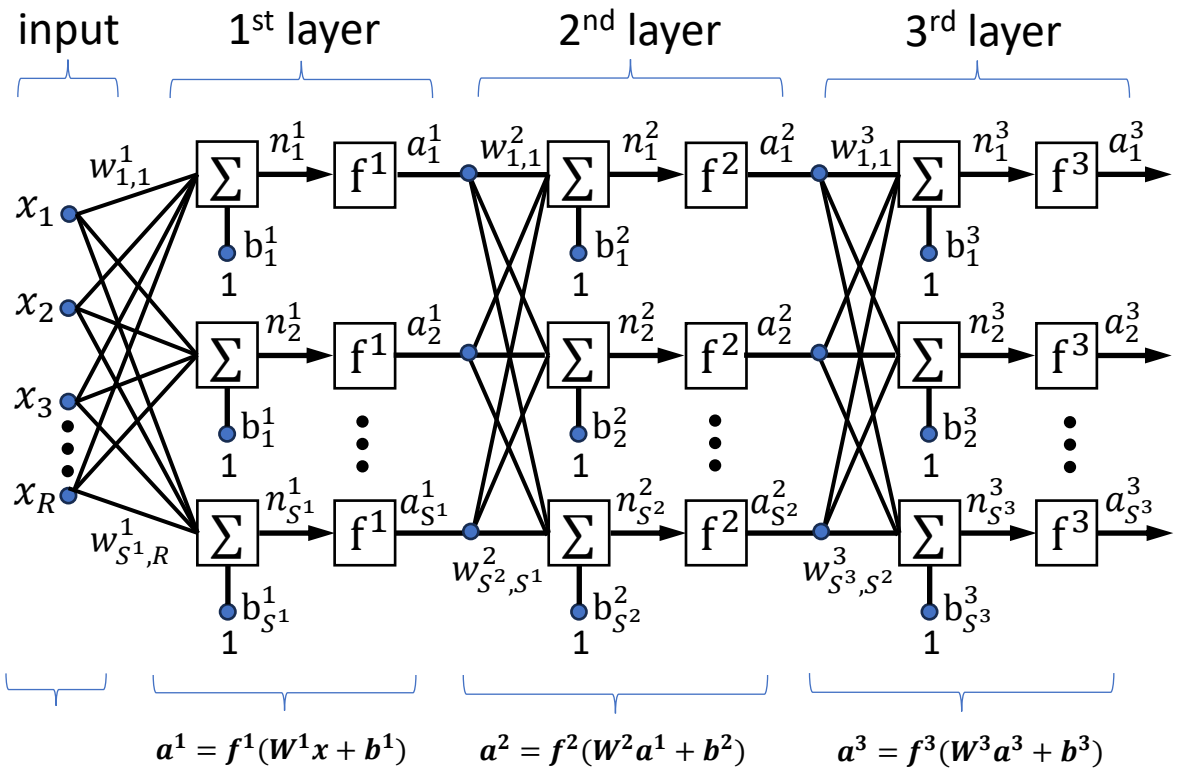
- Introduce nonlinear property into models

Name	Formula	Description
Linear	$\phi(x) = x$	Useful at regression outputs. Stacking linear layers remains linear.
Sigmoid	$\phi(x) = \frac{1}{1+e^{-x}}$	Maps to (0,1). Useful for binary probabilities, but saturates.
tanh	$\phi(x) = \tanh(x)$	Zero-centered output. It can also saturate for large |z|.
ReLU	$\phi(x) = \max(0, x)$	Simple gradients on z>0. Inactive units have zero gradient.

Nonlinearity gives a multilayer network more expressive power than one linear map.

Multiple Layers Neural Network

- A multilayer neural network maps inputs to outputs through successive affine transformations and nonlinear activations



$$a^{(l)} = f^{(l)}\left(W^{(l)} a^{(l-1)} + b^{(l)}\right), \quad a^{(0)} = x$$

Why Multiple Layers?

- More layers introduce more trainable parameters.
  - Each additional fully connected layer adds weights and biases, increasing the network's modeling capacity.
- Greater depth enables more complex representations.
  - Successive nonlinear transformations allow the network to capture increasingly intricate patterns and relationships.
- Greater modeling capacity supports more challenging tasks.
  - Deeper networks can learn complex mappings for tasks

- For a fully connected network, the total number of parameters is

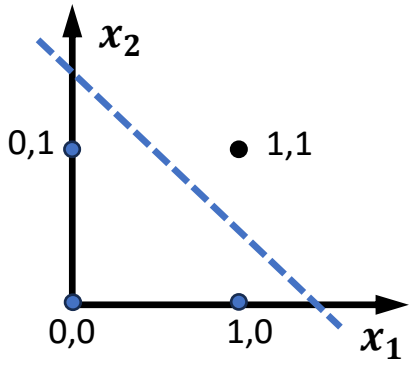
$$P = \sum_{l=1}^{L} \left(n_{l-1} n_l + n_l \right),$$

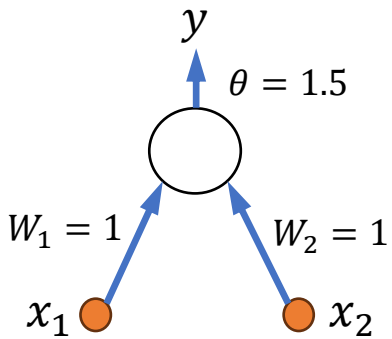
Example: AND

- A perceptron implements a logic unit:
  - Implements the logical AND operation.
  - Using the four input–output pairs in the truth table, train the corresponding weights W_1, W_2 and threshold θ to obtain a perceptron that implements AND.

$$y^k = F(S^k) = \mathrm{sgn}(S^k) = \begin{cases} 1, & S^k > \theta, \\ -1\ (0), & S^k \le \theta. \end{cases}$$

k	x_1	x_2	T
1	0	0	-1(0)
2	0	1	-1(0)
3	1	0	-1(0)
4	1	1	1



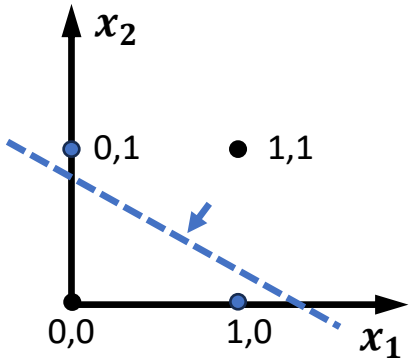


Neural Network

Example: XOR

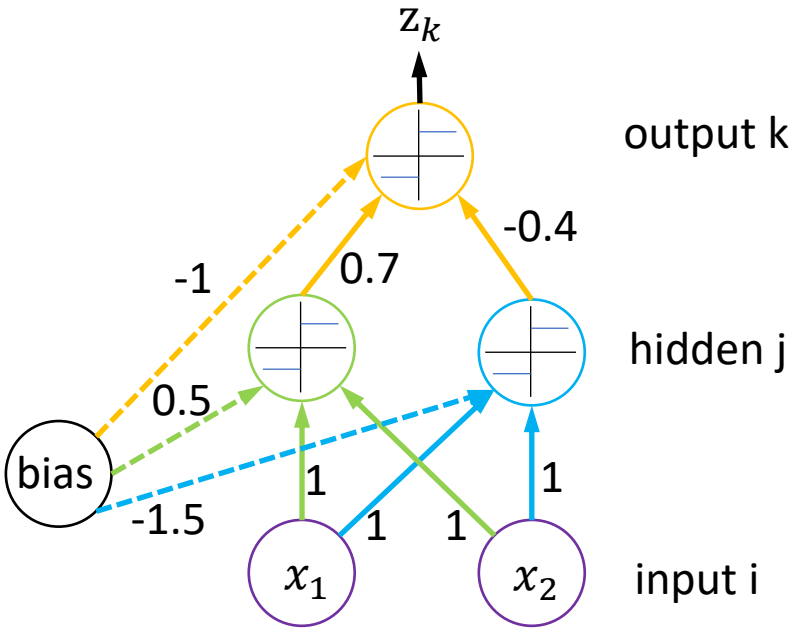
- The XOR Problem
  - A perceptron has limited representational power. Some functions, such as **exclusive OR (XOR)**, cannot be represented, regardless of how the weights and threshold are adjusted.
  - With two inputs forming a two-dimensional input space, a perceptron cannot implement the logical XOR function.

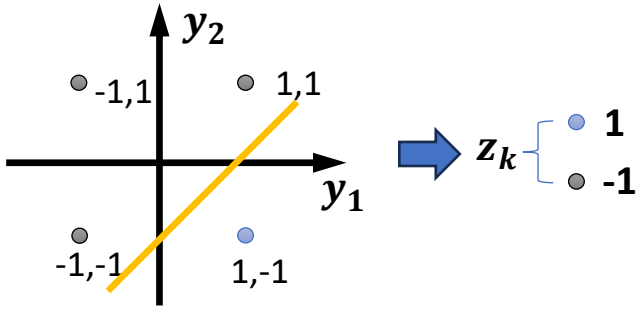
k	x_1	x_2	T
1	0	0	-1(0)
2	0	1	1
3	1	0	1
4	1	1	-1(0)

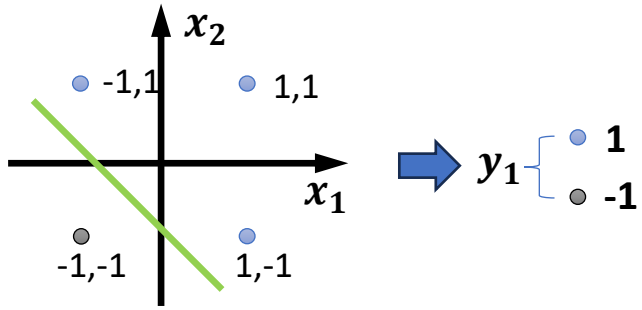


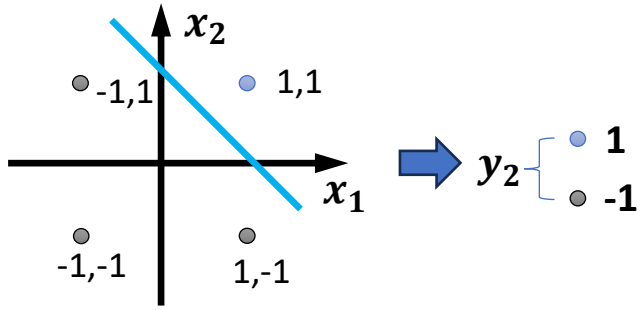
Example: XOR

- A two-layer perceptron solves the XOR problem
  - Neuron symbol: Sign function



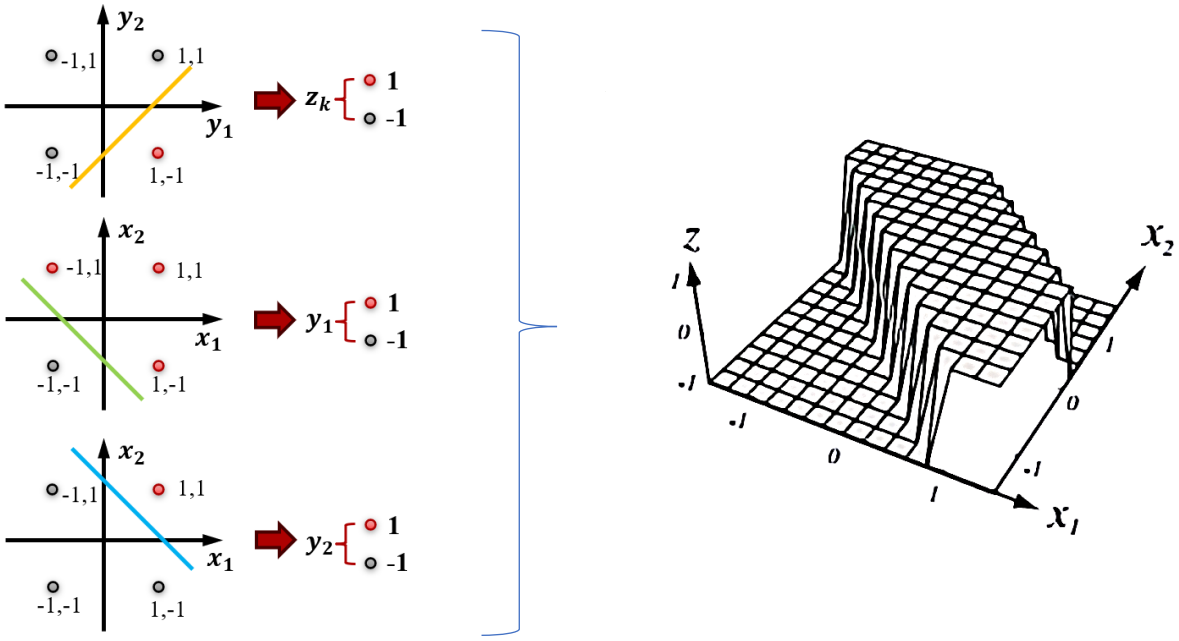






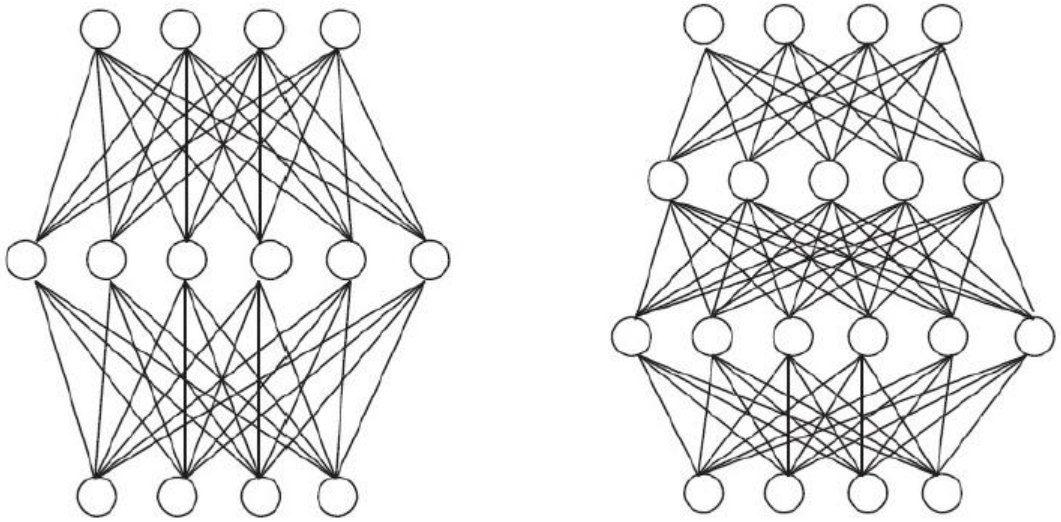
Example: XOR

- From the three plots on the previous slide, we can derive the output z as a function of x_1 and x_2. The resulting surface shows that the network successfully implements XOR.



Multilayer Feedforward Neural Networks

- **Definition:** Neurons in adjacent layers are fully connected, with no connections within a layer or connections that skip layers.
- **Feedforward:** The network receives external input signals, which are processed by neurons in the hidden and output layers to produce outputs.
- **Learning:** Connection weights and neuron thresholds are adjusted based on training data.
- **Multilayer network:** A network containing one or more hidden layers.



Backpropagation Algorithm

Backpropagation Algorithm

- Training challenge:
  - The hidden layers of a multilayer perceptron are not directly connected to the external environment, so their errors cannot be calculated directly.
- Backpropagation algorithm:
  - Propagates output-layer errors backward, layer by layer, to indirectly compute hidden-layer errors. It consists of two phases:
    - **Forward pass**: Compute each neuron's output sequentially, from the input layer through the hidden layers to the output layer.
    - **Backward pass**: Propagate output-layer errors backward to compute each hidden neuron's error, then use these errors to update the corresponding connection weights.

Backpropagation Algorithm

Backpropagation is an iterative method for finding an optimal solution.

- **Objective:**
  - Make the network's actual input–output mapping match the desired mapping.
- **Basic ideas:**
  - Define a cost function as the objective for iterative learning.
  - Use the partial derivatives of the cost function with respect to each parameter to determine parameter updates.
  - Exploit the network's layered structure to compute derivatives layer by layer, simplifying the calculation.

Backpropagation Algorithm

Basic Principle:

- **The Chain Rule:**
  - If a function is expressed as a composition of functions, its derivative can be calculated as the product of the derivatives of its constituent functions. For example:

$$z = (x+y)^2 \quad \Longrightarrow \quad \begin{cases} z = t^2, \\ t = x + y. \end{cases}$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x}$$

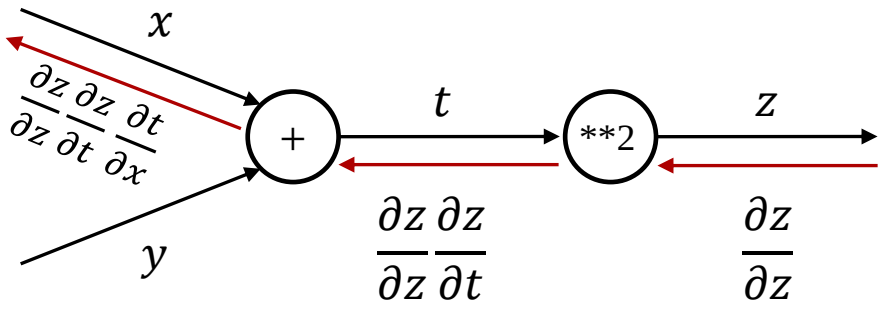
Backpropagation Algorithm

Basic Principle:

$$z = (x+y)^2 \quad \Longrightarrow \quad \begin{cases} z = t^2, \\ t = x + y. \end{cases}$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x}$$

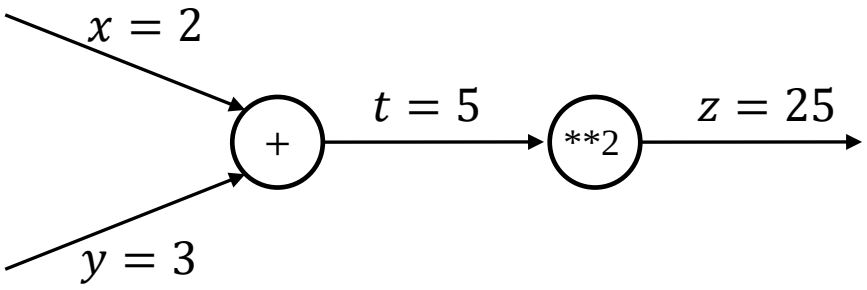
- **Computational Graph:**



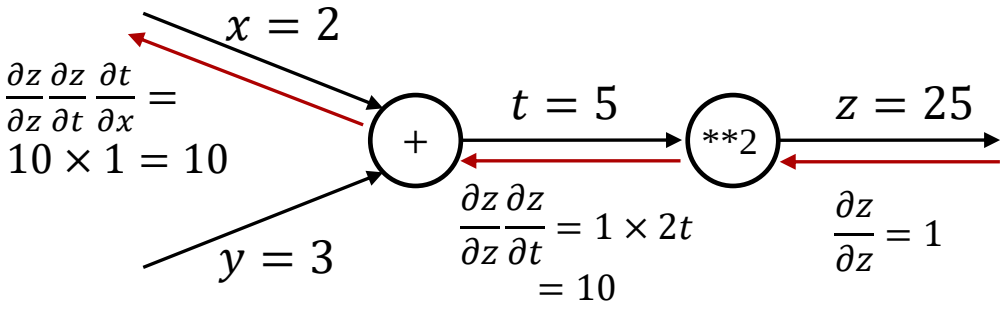
Backpropagation Algorithm

Basic Principle:

- **Forward Computational Graph:**



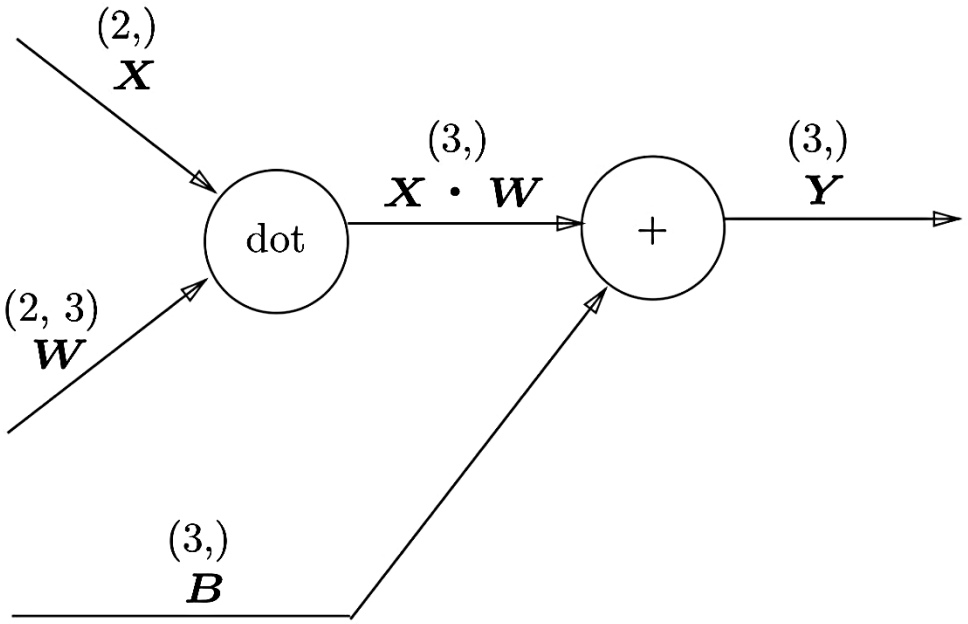
- **Backward Computational Graph:**



Backpropagation Algorithm

Basic Principle:

- **Forward pass using matrix operations:**



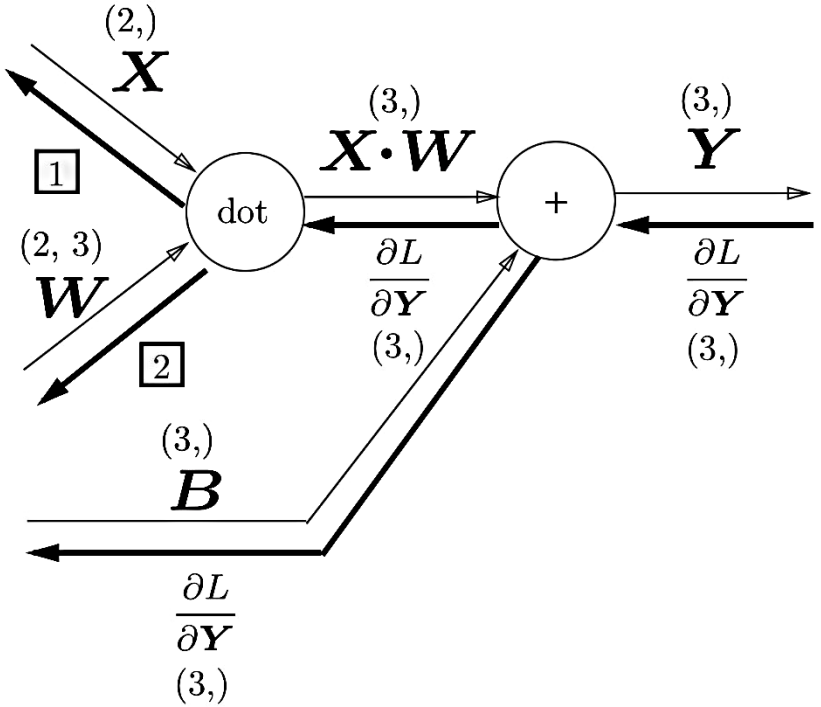
Backpropagation Algorithm

Basic Principle:

- **Backward pass using matrix operations:**

$$\boxed{1} \quad \underset{(2,)}{\frac{\partial L}{\partial \boldsymbol{X}}} = \underset{(3,)}{\frac{\partial L}{\partial \boldsymbol{Y}}} \cdot \underset{(3,\ 2)}{\boldsymbol{W}^{\mathrm{T}}}$$

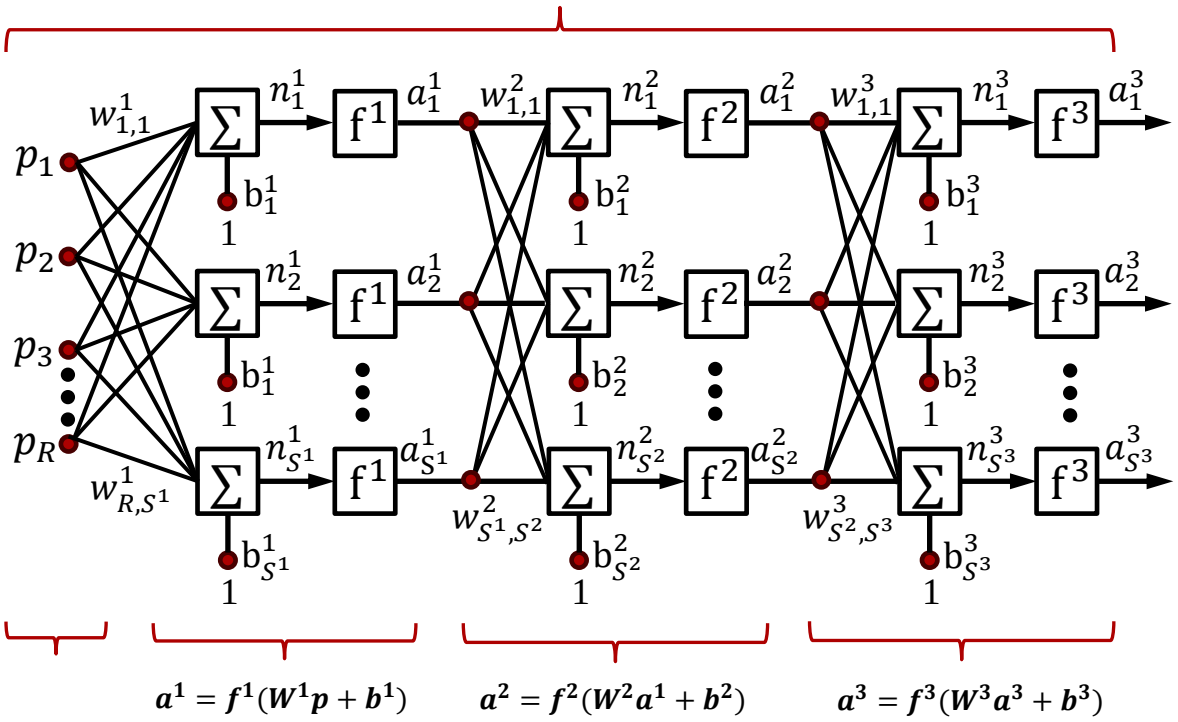
$$\boxed{2} \quad \underset{(2,\ 3)}{\frac{\partial L}{\partial \boldsymbol{W}}} = \underset{(2,\ 1)}{\boldsymbol{X}^{\mathrm{T}}} \cdot \underset{(1,\ 3)}{\frac{\partial L}{\partial \boldsymbol{Y}}}$$



Backpropagation Algorithm

- **Forward pass:**

$$a_j^{(k)} = f\left(n_j^{(k)}\right), \qquad n_j^{(k)} = \boldsymbol{w}_j^{(k)} \cdot a^{(k-1)} = \sum_i w_{ij}^{(k)} a_i^{(k-1)}$$

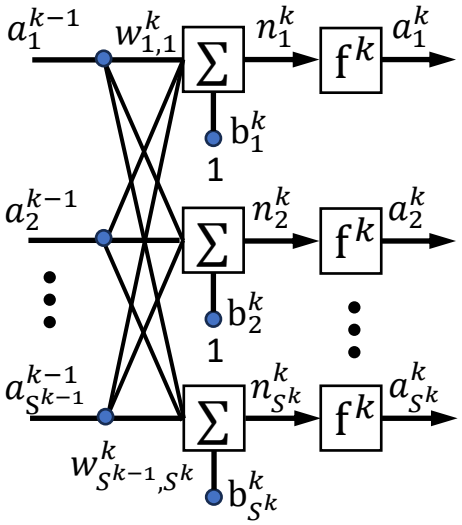


Backpropagation Algorithm

- **Objective Function:**

$$L_{\text{SSE}} = \frac{1}{2S}\sum_{j}^{S}(t_j - a_j)^2$$

  - $T = (t_1, t_2, \ldots, t_S)$: ground-truth labels
  - $A = (a_1, a_2, \ldots, a_S)$: output-layer outputs
  - $N = (n_1, n_2, \ldots, n_S)$: results of matrix multiplication before applying the activation function



- **Weight Update: Gradient Descent**
  - Weight update for layer k

$$\Delta \boldsymbol{w}_j = -\eta \frac{\partial L_{SSE}}{\partial \boldsymbol{w}_j} = -\eta \frac{\partial L_{SSE}}{\partial n_j} \cdot \frac{\partial n_j}{\partial \boldsymbol{w}_j} = -\eta \cdot \frac{\partial L_{SSE}}{\partial n_j} \cdot \boldsymbol{a^{k-1}}$$

Backpropagation Algorithm

- **Algorithm Steps** (assuming sigmoid activation functions in all layers):
  - Initialize the weights $W(0)$.
  - Perform forward propagation to compute the outputs $a_j(t)$ of all neurons.
  - Compute the output-layer error signals:
$$\delta_j = (t_j - a_j)a_j(1 - a_j).$$
  - Propagate backward to compute the hidden-layer error signals:
$$\delta_j = a_j(1 - a_j)\sum_i w_{ij}\delta_i.$$
  - Compute and store the weight updates:
$$\Delta w_{ij} = -\eta\,\delta_i a_j.$$
  - Update the weights:
$$w_{ij}(t + 1) = w_{ij}(t) + \Delta w_{ij}.$$
  - Check for convergence. If converged, stop; otherwise, return to Step 2.

Skills

Beyond Basic Backpropagation

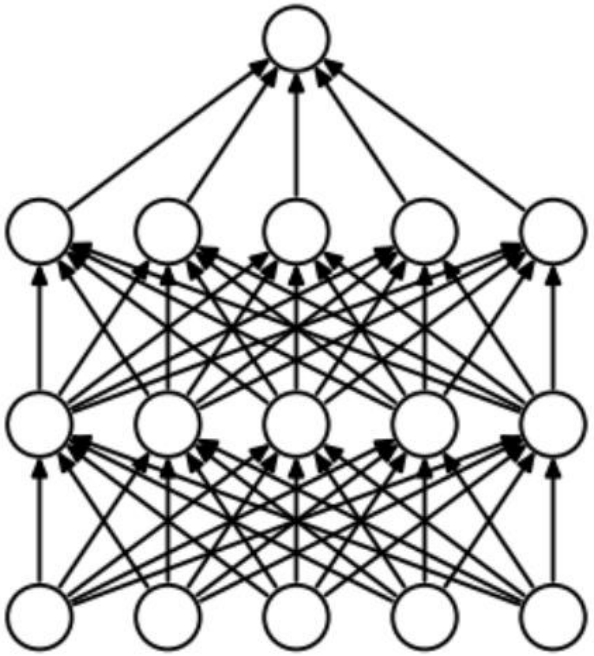
- BP applies the chain rule to a computational graph. We can change a layer if we can compute its local gradients.

What changes?	How BP handles it
Dropout mask	Multiply the gradient by the same mask
Normalization	Differentiate through mean and variance
Residual addition	Add the gradients from both paths

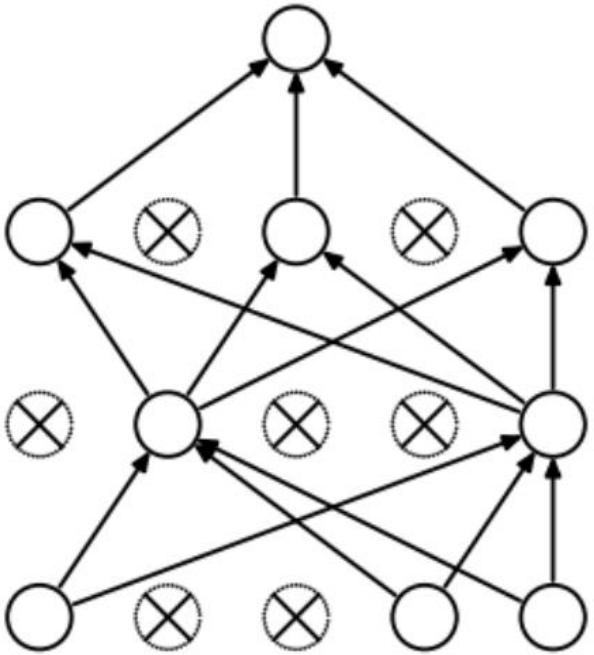
BP computes gradients. An optimizer uses them to update parameters.

Dropout

- Randomly mask activations during training to reduce overfitting and reliance on particular units.



(a) Standard Neural Net



(b) After applying dropout.

Batch Normalization

- Normalize each feature across a mini-batch. This often makes training more stable and less sensitive to scale.

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}, \quad y_i = \gamma \hat{x}_i + \beta$$

  - μ and σ are the batch mean and variance.

- Backward pass:
  - Gradients pass through the batch mean and variance, as well as the trainable scale γ and shift β.

Training	Inference
Use current mini-batch statistics	Use running estimates of mean and variance
Update γ and β through BP	Keep learned γ and β

Weight Initialization

- Poor initial scales can make activations or gradients vanish or explode before learning gets started.

Initialization	Typical use	Weight variance
Xavier	Linear or tanh layers	$W \sim \mathcal{N}(0, \sqrt{\frac{2}{n_{in}+n_{out}}})$
He	ReLU layers	$W \sim \mathcal{N}(0, \sqrt{\frac{2}{n_{in}}})$

- **Connection to BP:**
  - Random weights break symmetry. Their variance controls the initial scale of forward signals and backward gradients.

L2 Regularization

- Penalize large weights to control model complexity. Choose the penalty strength λ using validation data.

$$J(W) = L(W) + \frac{\lambda}{2}||W||_2^2$$

$$\frac{\partial J(W)}{\partial W} = \frac{\partial L(W)}{\partial W} + \lambda W$$

- The penalty adds λW to the gradient and shrinks weights at every SGD step.

Thanks