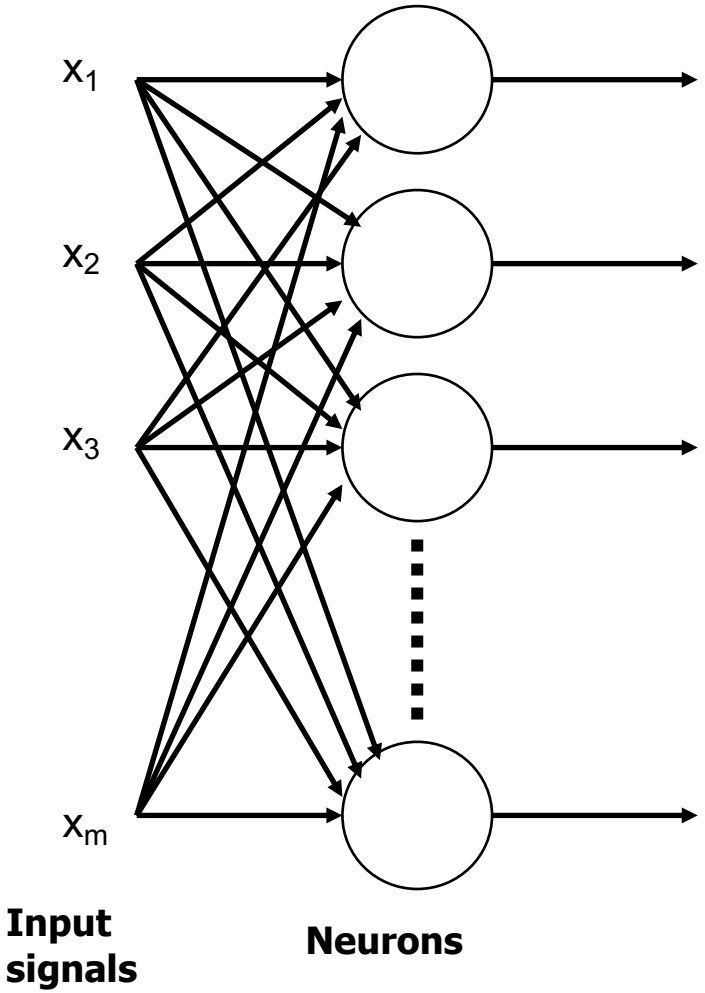
Perceptrons

Minlie Huang
aihuang@tsinghua.edu.cn
Dept. of Computer Science and Technology
Tsinghua University
http://coai.cs.tsinghua.edu.cn/hml/

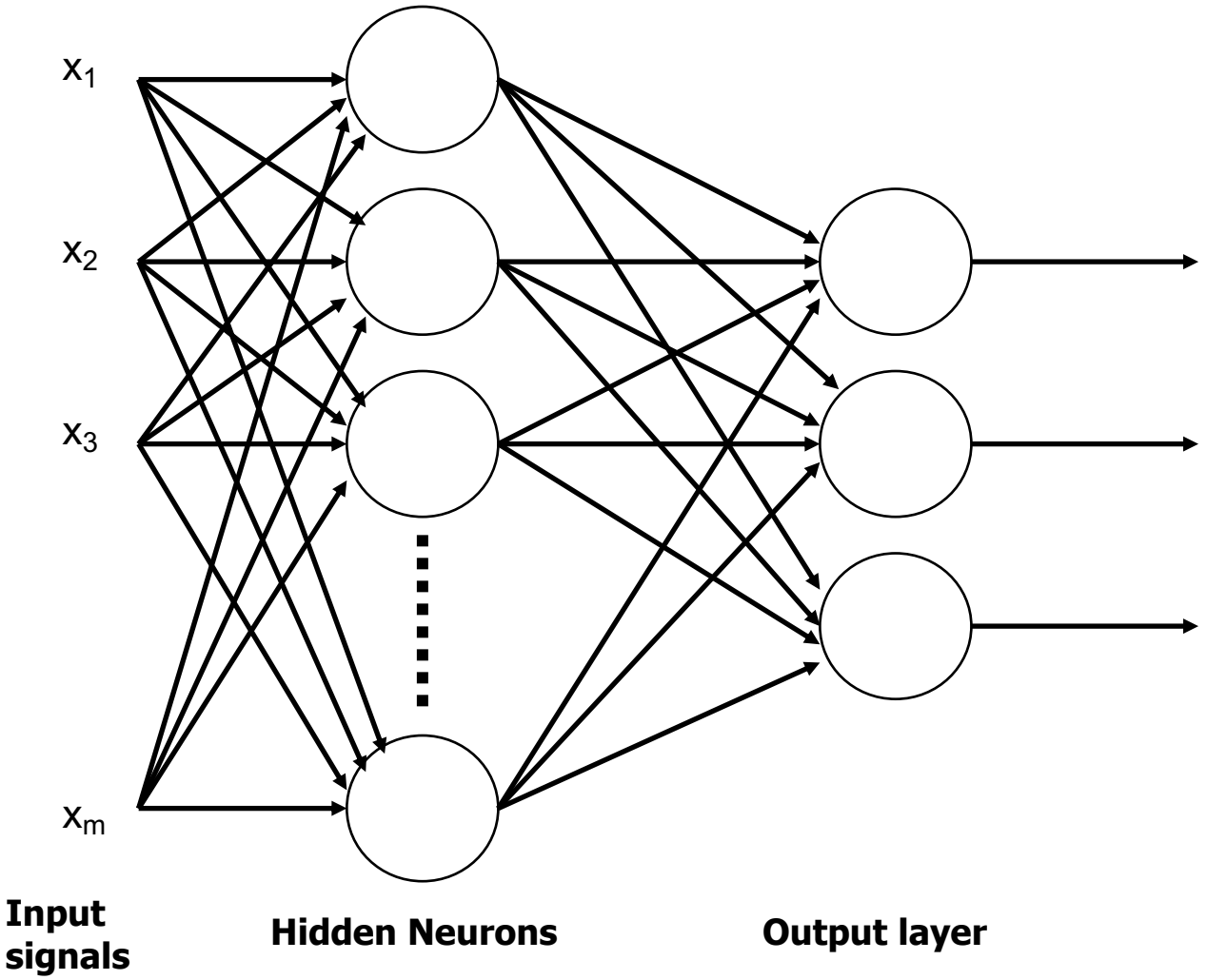
Typical Neural Structures

- Feed-forward network
- Recurrent neural network
- Convolutional neural network
- …

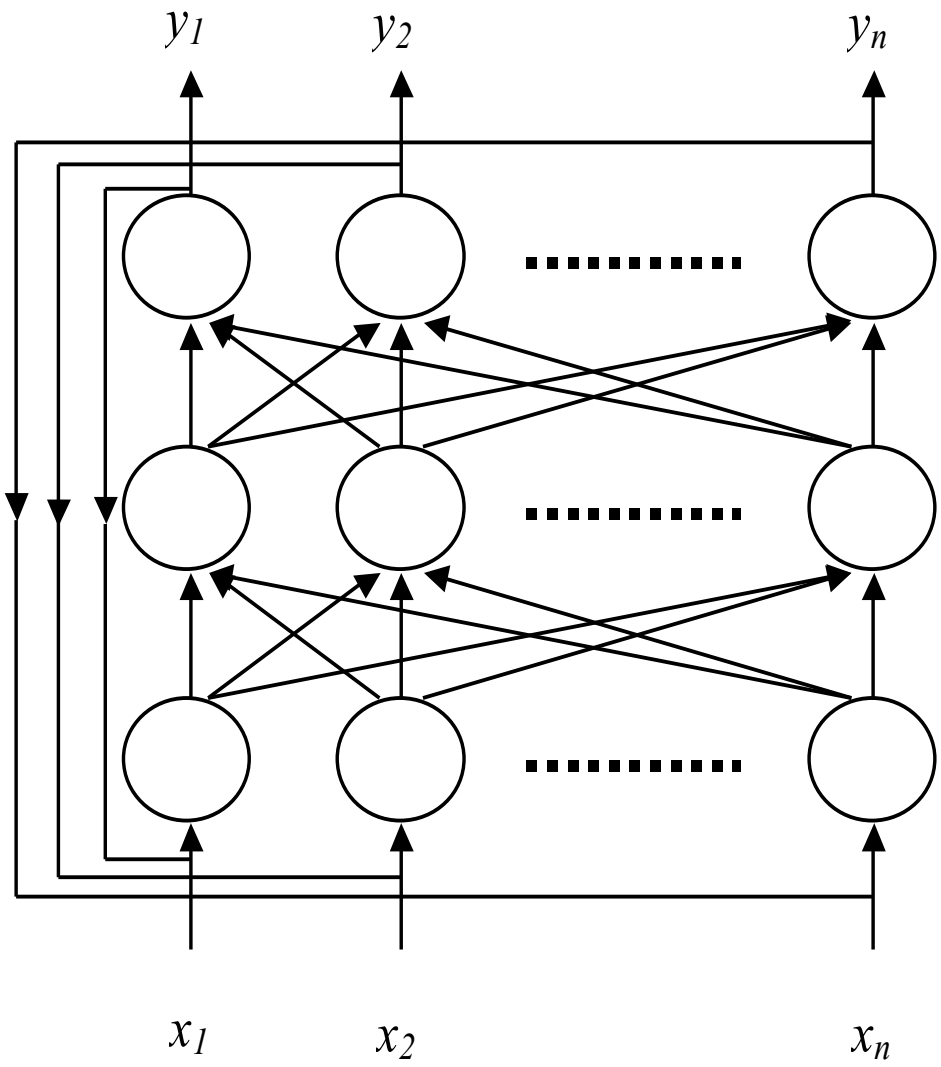
Single-layer Feed-forward Network



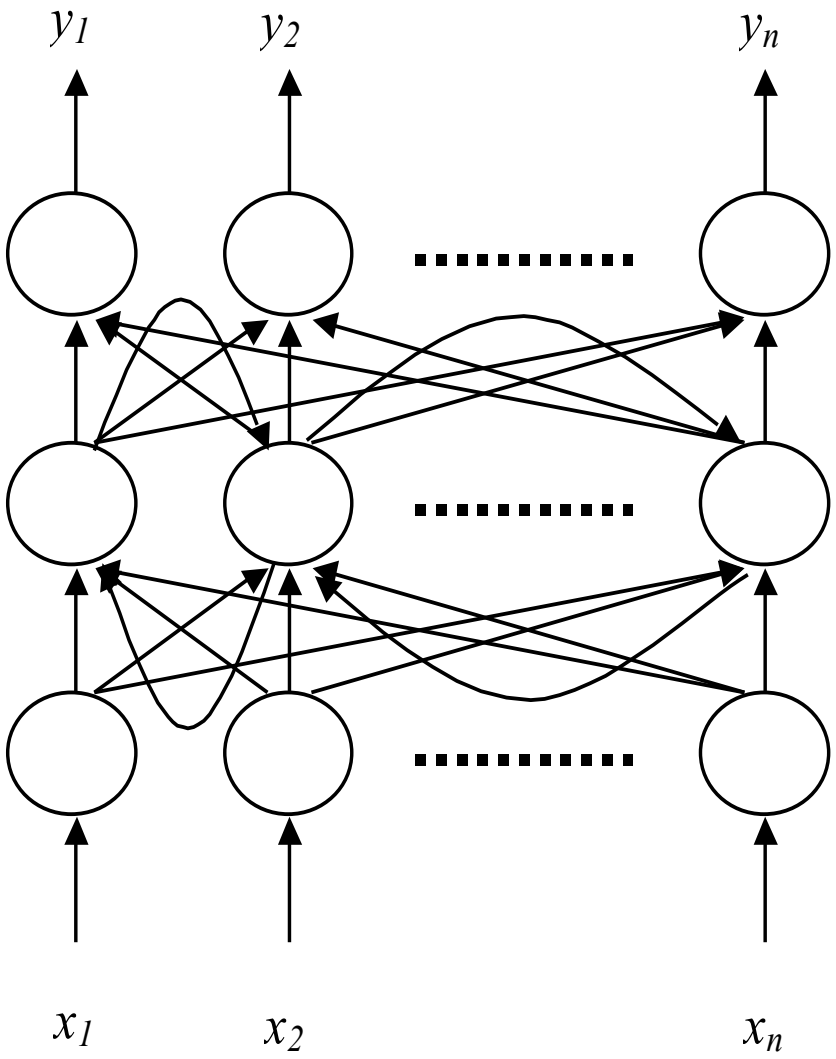
Multilayer Feed-forward Network



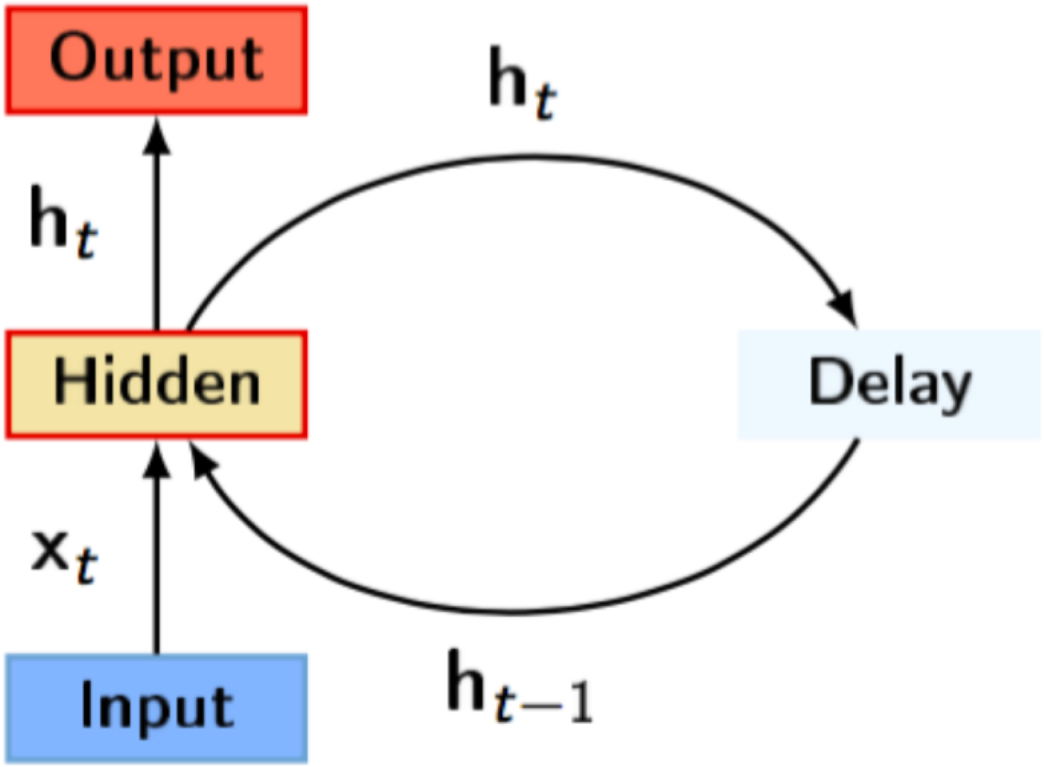
Feed-back Feed-forward Network



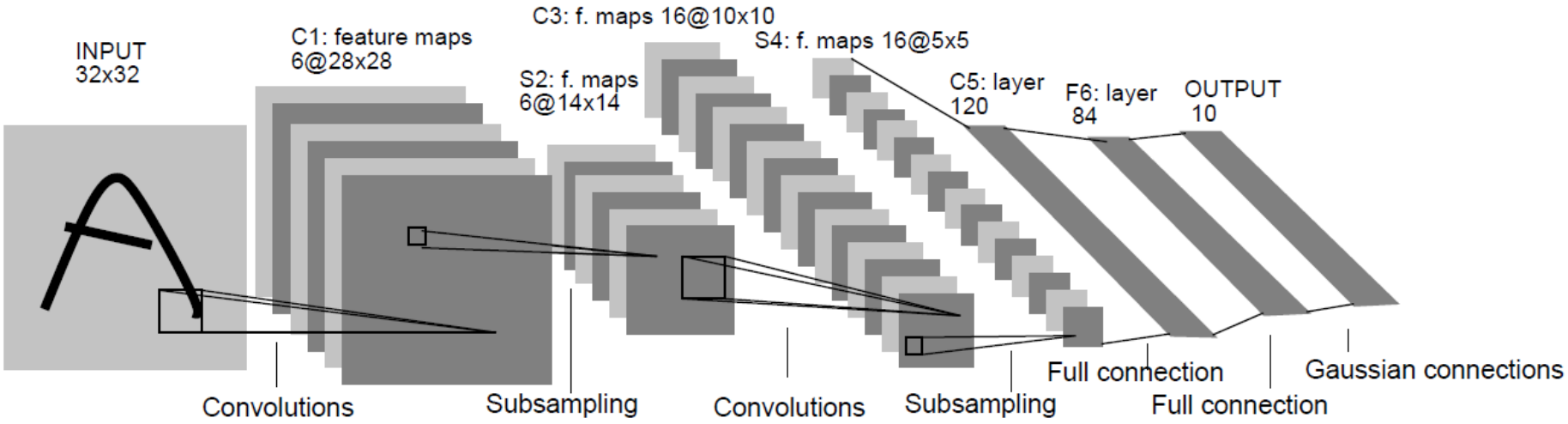
Inter-feedback Feed-forward Network



Recurrent Neural Networks



Convolutional Neural Network (CNN)



Typical Neural Structures

- Among all the structures, the most fundamental unit is **perceptron**

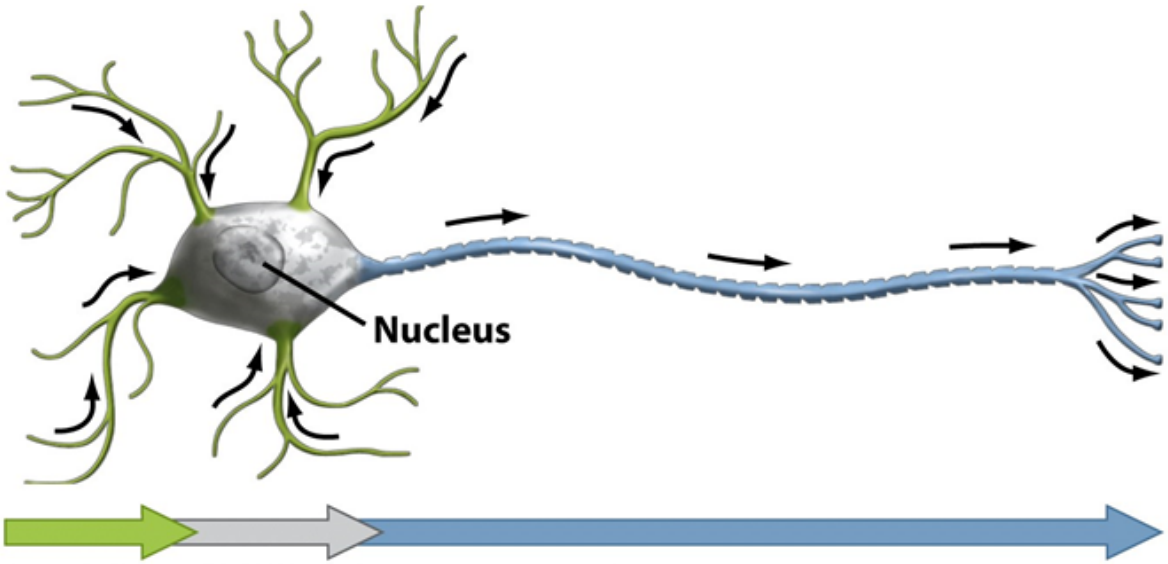
Overview

- Single Layer Perceptron
- Multilayer Perceptron
  - BP Learning Algorithm

Single Layer Perceptron

- Proposed by McCulloch and Pitts, and Hebb, Rosenblatt in 1957
- The simplest form of a neural network used for linearly separable problems
- ***Perceptron convergence theorem***
- One neuron for two-class problems, multiple neurons for multi-class problems (multi-layer perceptron)
- Theoretically, multilayer perceptron can be used to solve any classification and regression problem with BP learning algorithm

Biological Model



Dendrites
Collect electrical signals

Cell body
Integrates incoming signals and generates outgoing signal to axon

Axon
Passes electrical signals to dendrites of another cell or to an effector cell

Figure 45-2b Biological Science, 2/e

Weighted Input

- Synapses (receptors) of a neuron have weights. $W=(w_1, w_2, \ldots, w_n)$, which can have positive (excitatory) or negative (inhibitory) values. Each incoming signal is multiplied by the weight of the receiving synapse $w_i x_i$. Then all the "weighted" inputs are added together into a weighted sum v:

$$v = w_1x_1 + w_2x_2 + \ldots + w_nx_n = \langle w, x \rangle$$

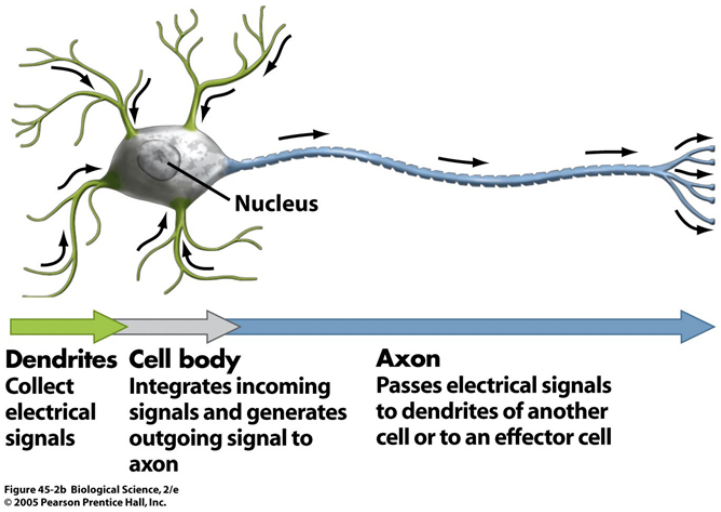
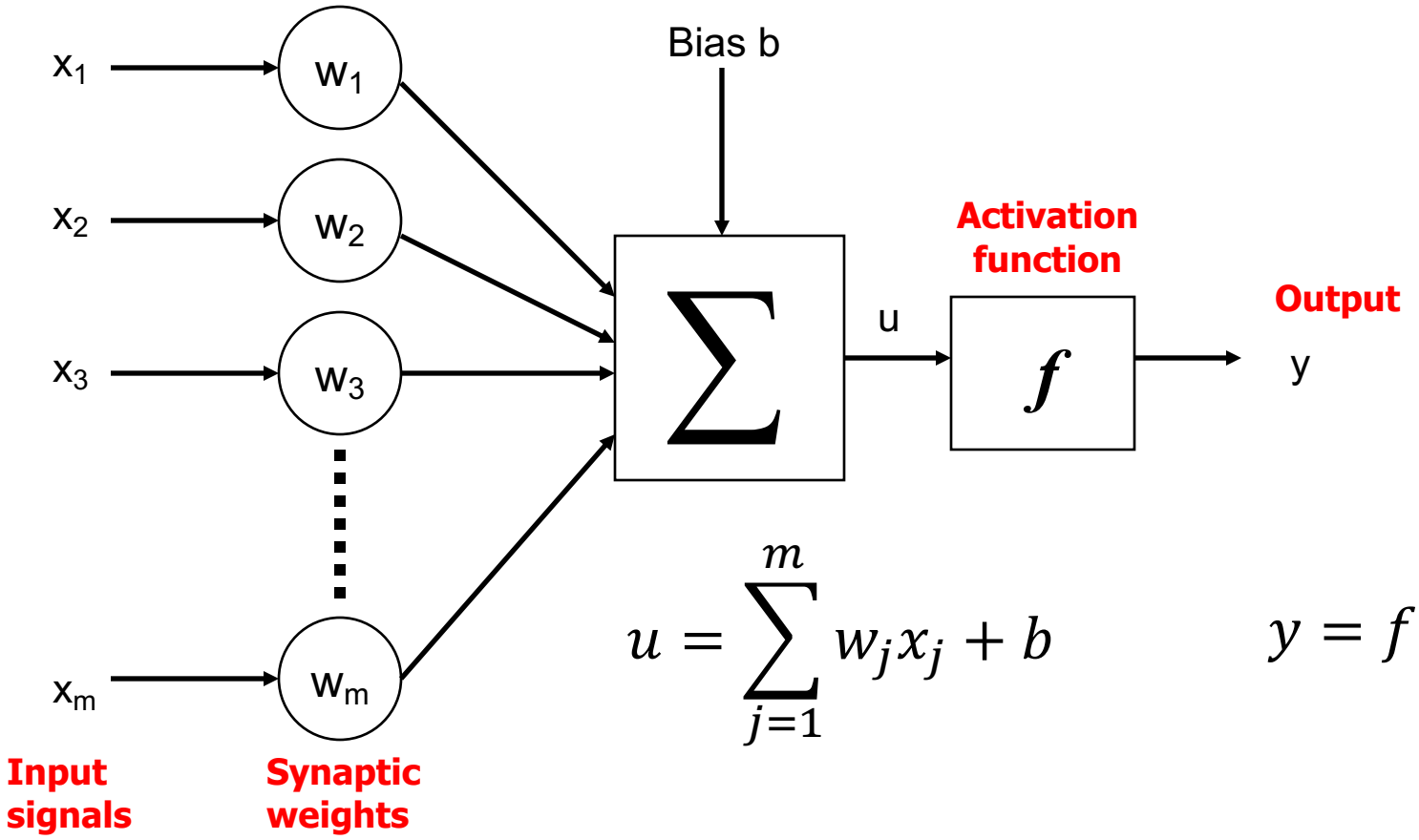


Figure 45-2b Biological Science, 2/e
© 2005 Pearson Prentice Hall, Inc.

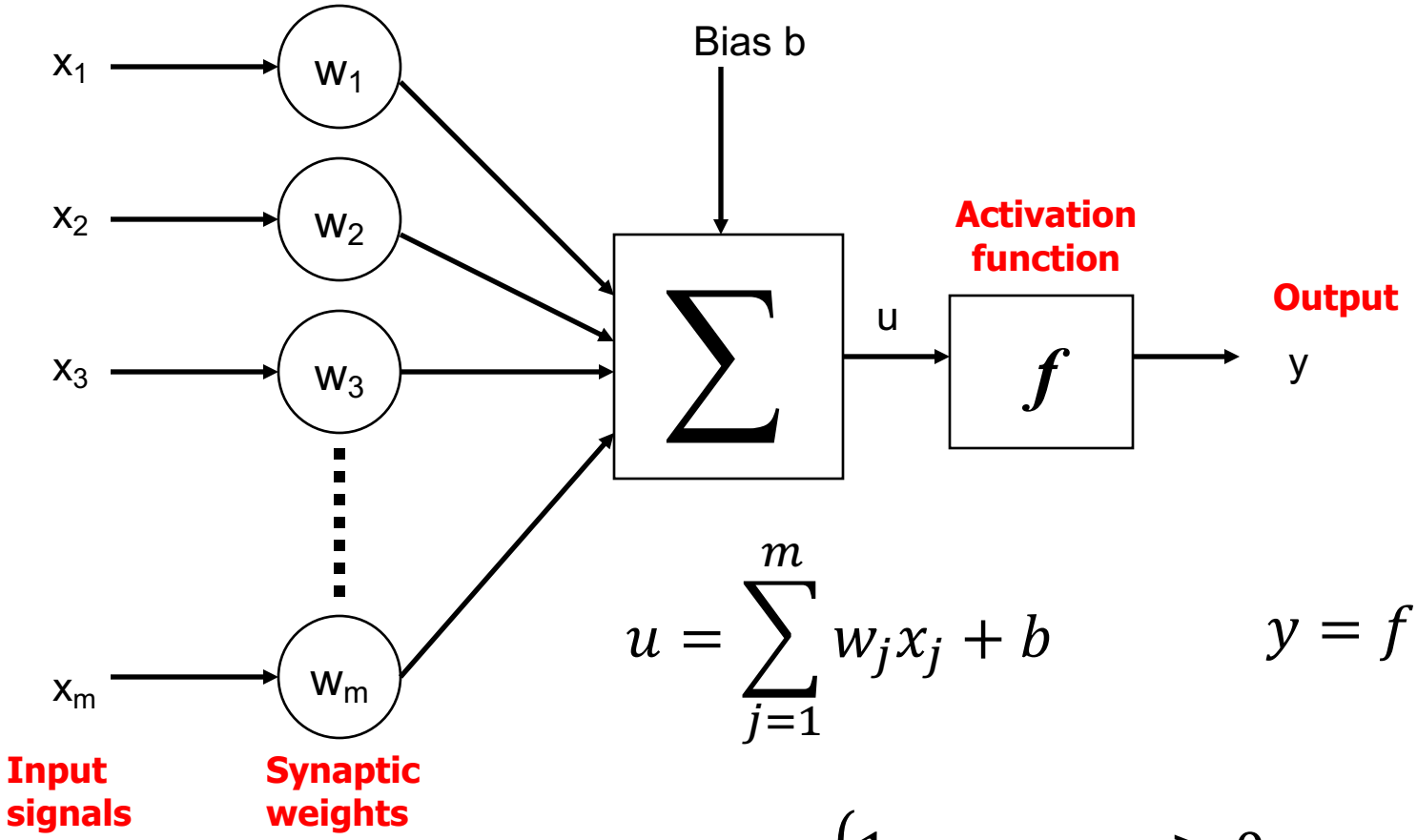
Neuron Model



$$u = \sum_{j=1}^{m} w_j x_j + b$$

$$y = f(u)$$

Computational Model



$$u = \sum_{j=1}^{m} w_j x_j + b \qquad y = f(u)$$

$$f(u) = \begin{cases} 1 & u \geq 0 \\ -1 & u < 0 \end{cases}$$

Supervised learning

- Teacher gives samples of inputs $x(n)$ and corresponding desired outputs $l(n)$
- Goal is to find weights which imitate the behavior of the teacher

Learning Rule

$$w_i \mathrel{+}= \eta\big(l(n) - y(n)\big)x_i(n)$$
$$i = 1,2,\dots,m$$

- If the n-th input $x(n)$ is correctly classified, i.e., $l(n)=y(n)$
  - Nothing happens

- Otherwise, $l(n)!=y(n)$
  - Update weights (two cases)

Adjust the Weights

- Adjust the weights
  - Start from randomly initialized weights
  - Update weights according to the rule
  - Stops when convergence or other condition is met

```
float w[n] = <n random floats>;
float w_0 = <a random float>;
bool errorDetected = true;
float η = <positive_number>;
while(errorDetected)  {
  errorDetected = false;
  for(int i = 1; i ≤ m; i++) {
     if (sgn(w^T x_i + w_0) ≠ ℓ_i) {
        errorDetected = true;
        w   += ηℓ_i x_i;
        w_0 += ηℓ_i;
     }
  }
}
return {w, w_0};
```

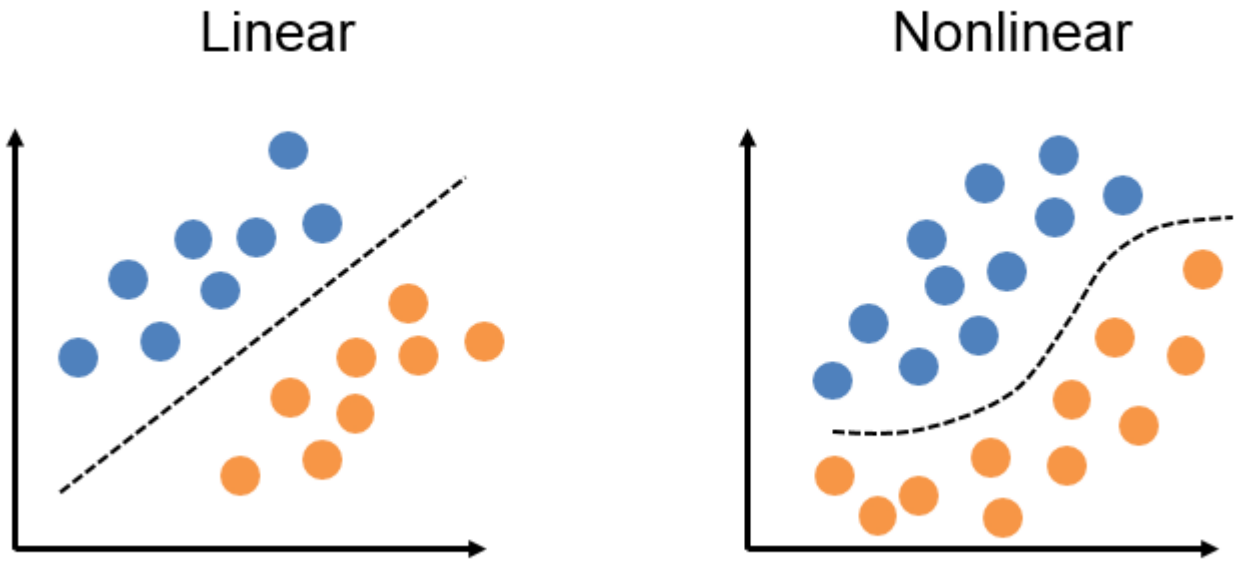
针对一组n-th数据的训练

Perceptron convergence theorem

- For linearly separable problems, the algorithm **converges at finite steps**
  - ***Perceptron convergence theorem***

- See proof (another pdf): http://coai.cs.tsinghua.edu.cn/hml/artificial_network_2019/

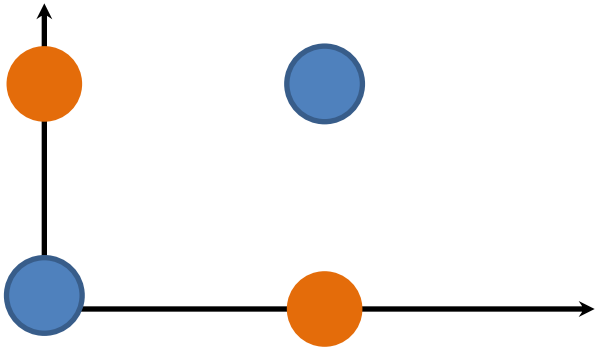
Linear and nonlinear data



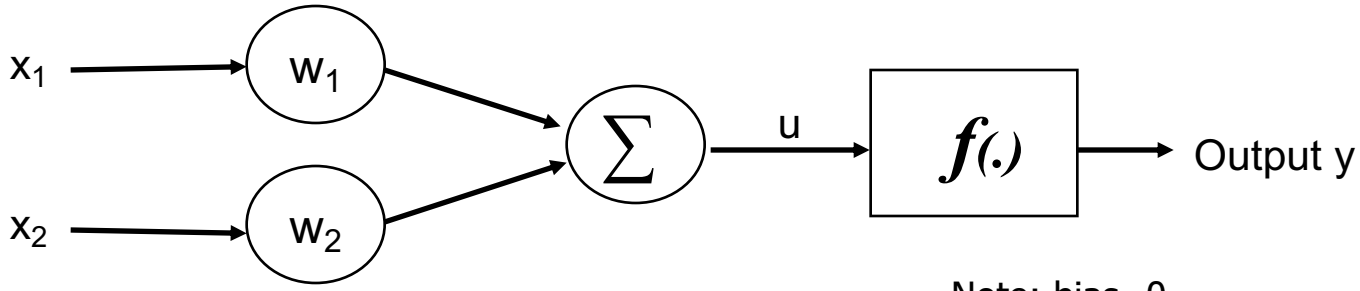
Linear function can separate the data without any error

XOR Problem

- A famous example: XOR problem
  - Class 1: (0, 0) and (1, 1)
  - Class 2: (1, 0) and (0, 1)
  - The classes are not linearly separable, i.e. there is no hyperplane (line in this case) separating the classes.



XOR Problem

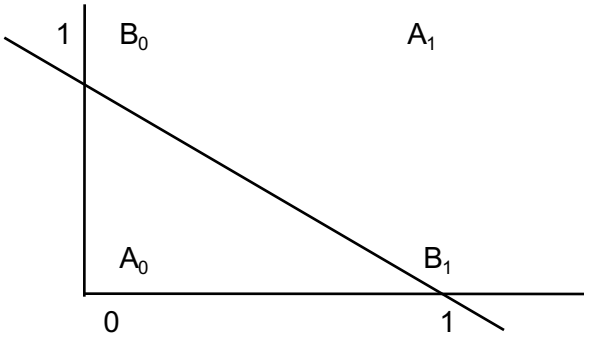


Note: bias=0

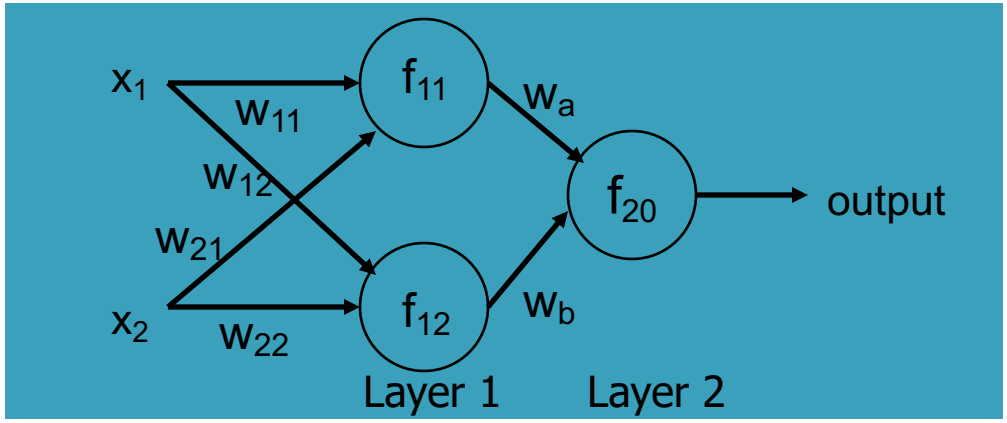
$$u = x_1 w_1 + x_2 w_2$$

$$f(u) = \begin{cases} 1 & u \geq 0.5 \\ 0 & u < 0.5 \end{cases}$$

Point	x1	x2	Desired Output
A_0	0	0	0
B_0	1	0	1
B_1	0	1	1
A_1	1	1	0



Two Layers Structure

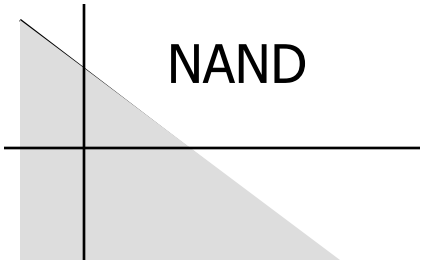


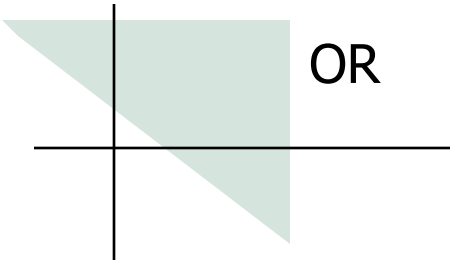
W_{11}=0.375 W_{12}=0.75 W_a=0.375

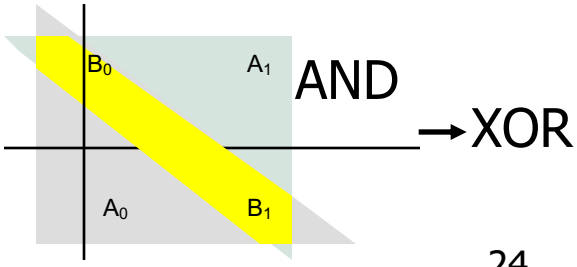
W_{21}=0.375 W_{22}=0.75 W_b=0.375

Note: bias=0

$$f_{11}(u)=\begin{cases}1 & u<0.5\\ 0 & u\geq 0.5\end{cases} \quad f_{12}(u)=\begin{cases}1 & u\geq 0.5\\ 0 & u<0.5\end{cases} \quad f_{20}(u)=\begin{cases}1 & u\geq 0.5\\ 0 & u<0.5\end{cases}$$







Why Perceptron

- Why (single-layer) Perceptron is limited
  - **Step or linear function**
- Difference between single layer and multilayer Perceptrons
  - Activation function
  - Learning algorithm
- Multilayer Perceptrons can approximate any continuous function arbitrarily well
  - Convergence problem

Sigmoid Activation Function

- Gradient of step-function is **either zero or does not exit**
- Linear function stays linear no matter how many layers
- Solution:
  - Sigmoid function
  - Linear Rectifier

$$f(x) = \frac{1}{1 + e^{-x}} \qquad f(x) = \max(0, x)$$

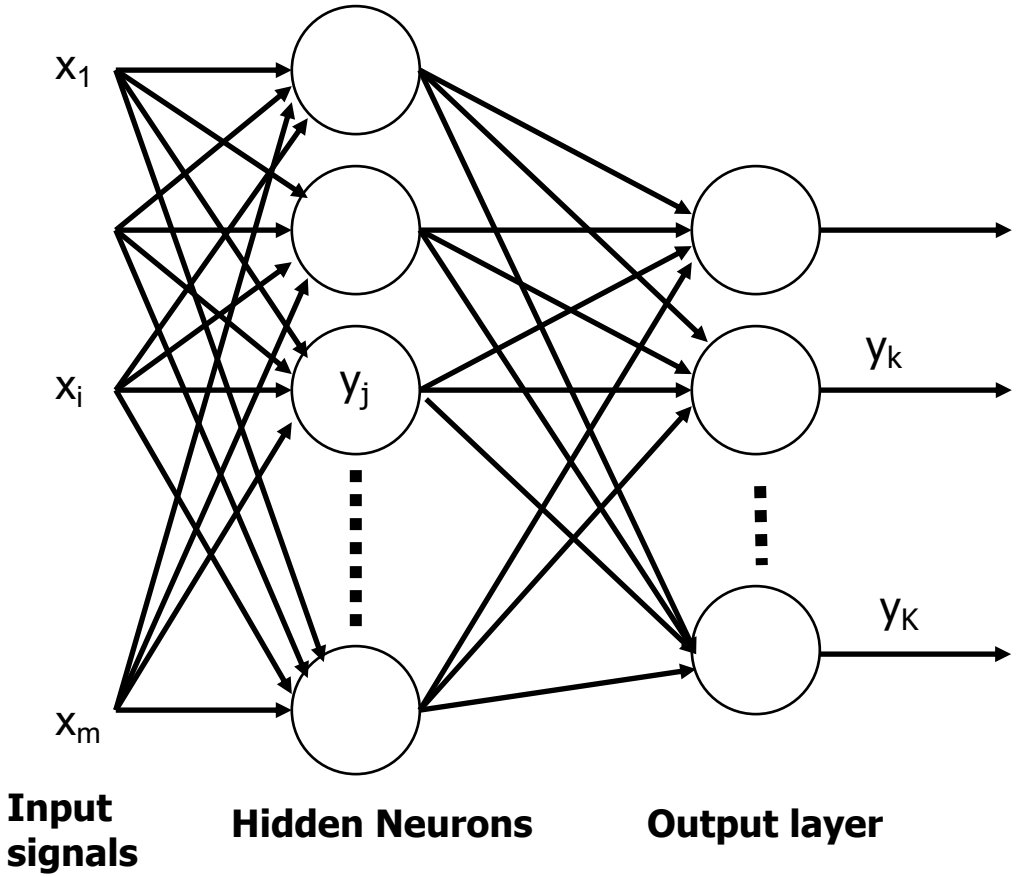
$$f'(x) = f(x)(1 - f(x)) \qquad f'(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases}$$

Structure

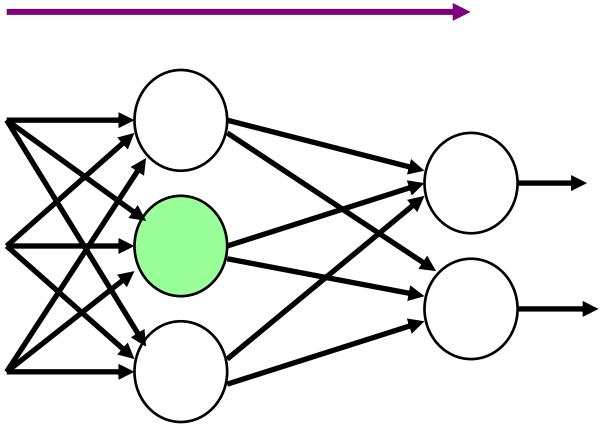
- Each layer has perceptrons which can extract (nonlinear) features
- Subsequent layers combine features
- Given enough hidden neurons, multilayer perceptron (MLP) can approximate any continuous function arbitrarily well
- Activation propagates from input to output

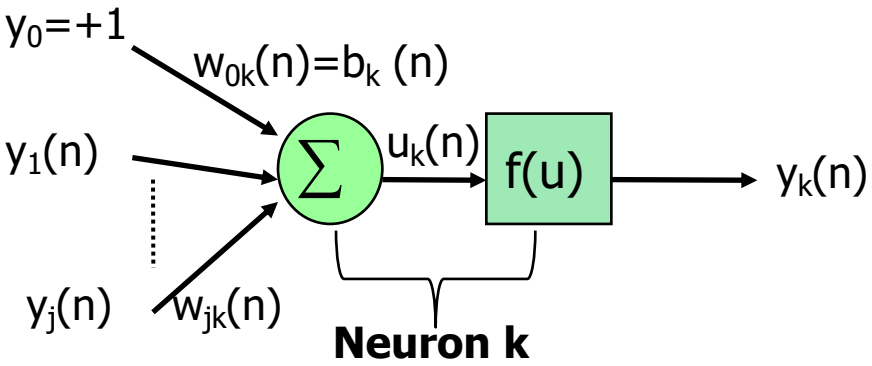
嫁接

Network Structure



Forward Computing





$$u_k(n) = \sum_{j=0} w_{jk}(n) y_j(n) \qquad y_k(n) = f(u_k(n))$$

Sigmoid

$$y_k(n) = sigmoid(u_k(n))$$

Softmax

$$y_k(n) = \frac{e^{u_k(n)}}{\sum_j e^{u_j(n)}}$$

Terminologies

- u_j--- weighted sum of the input to neuron j
- y_j--- output of neuron j
- W_{ij}---weight between neuron i (layer l) to neuron j (layer $l+1$)
- n--- the sample index

Error Propagation

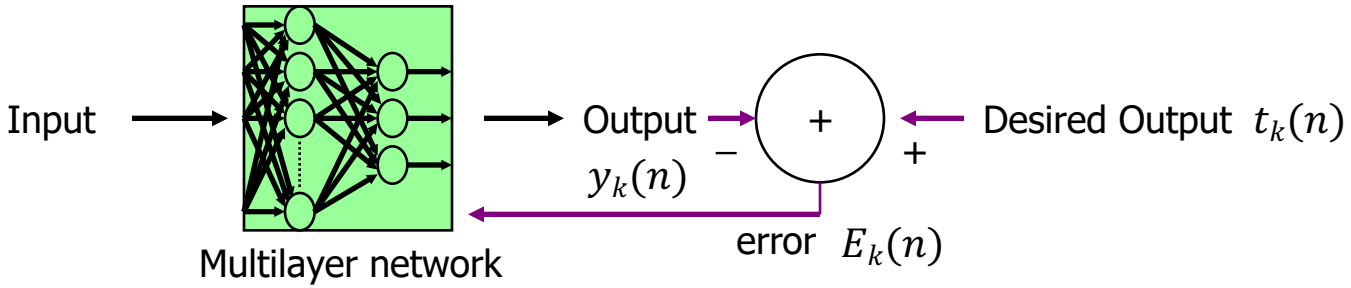
- An error is computed and the gradient of error propagates backward
  - Mean Square Error
  - Cross Entropy

y_k: output of the k-th neuron in the output layer

$$E_k(n) = \frac{1}{2}\left(t_k(n) - y_k(n)\right)^2 \qquad E_k(n) = -t_k log y_k(n)$$

$$E(n) = \sum_k E_k(n) \qquad E(n) = \sum_k E_k(n)$$

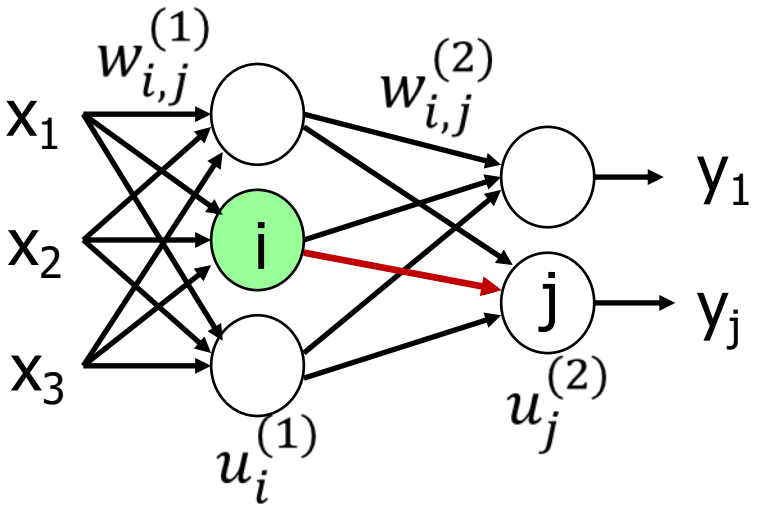
Error Propagation

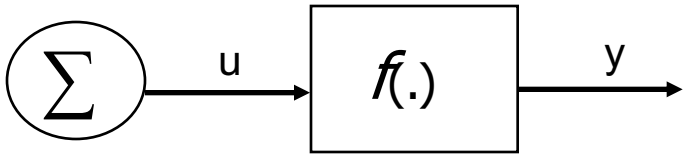


Synaptic adjustment $$\Delta w_{kj}(n) = -\eta \frac{\partial E(n)}{\partial w_{kj}}$$

Updated synaptic weight $$w_{kj}(new) = w_{kj}(old) + \Delta w_{kj}(n)$$

Starting from Simple Example





Loss function

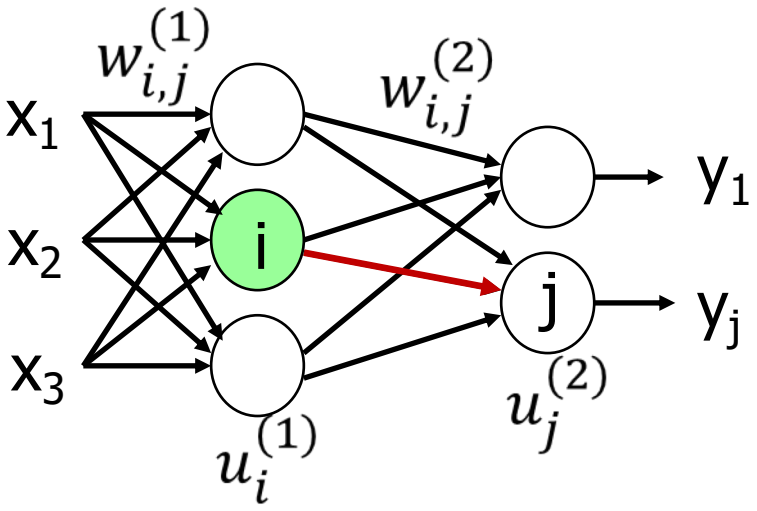
$$E_k(n) = \frac{1}{2}\left(t_k(n) - y_k(n)\right)^2$$

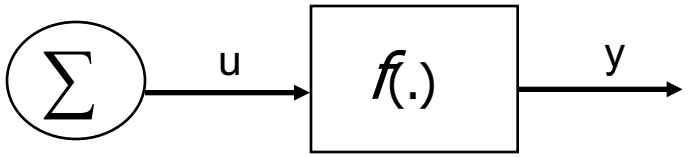
$$E(n) = \sum_k E_k(n)$$

$$\frac{\partial E(n)}{\partial w_{ij}^{(2)}} = \frac{\partial E(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial u_j^{(2)}(n)} \frac{\partial u_j^{(2)}(n)}{\partial w_{ij}^{(2)}}$$

$$y_j(n) - t_j(n) \quad f'(u_j^{(2)}(n)) \quad y_i^{(1)}(n)$$

Starting from Simple Example





Loss function

$$E_k(n) = \frac{1}{2}\left(t_k(n) - y_k(n)\right)^2$$

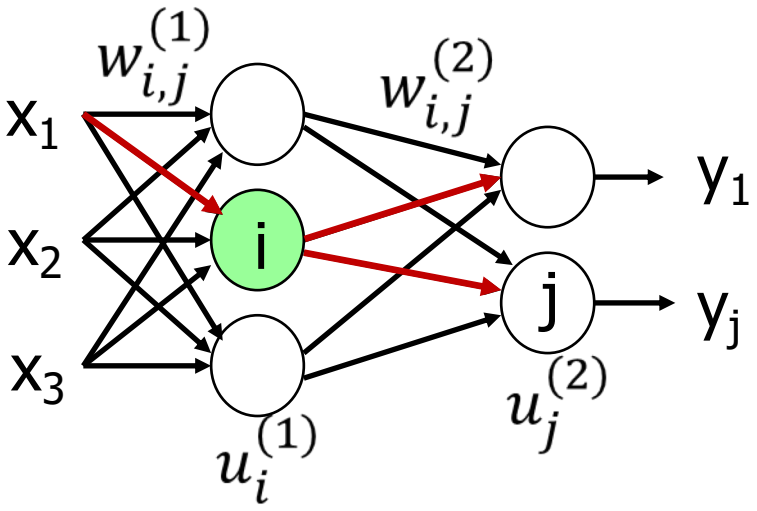
$$E(n) = \sum_k E_k(n)$$

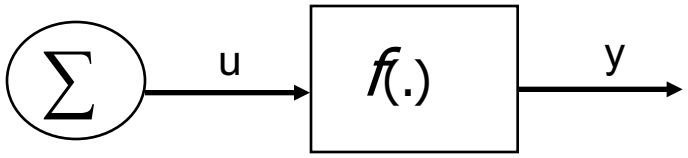
$$\frac{\partial E(n)}{\partial w_{ij}^{(2)}} = \frac{\partial E(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial u_j^{(2)}(n)} \frac{\partial u_j^{(2)}(n)}{\partial w_{ij}^{(2)}}$$

$$= \frac{\boldsymbol{\partial E(n)}}{\boldsymbol{\partial u_j^{(2)}(n)}} * y_i^{(1)}(n) = \boldsymbol{\delta_j^{(2)}(n)} y_i^{(1)}(n)$$

Local gradient

Starting from Simple Example





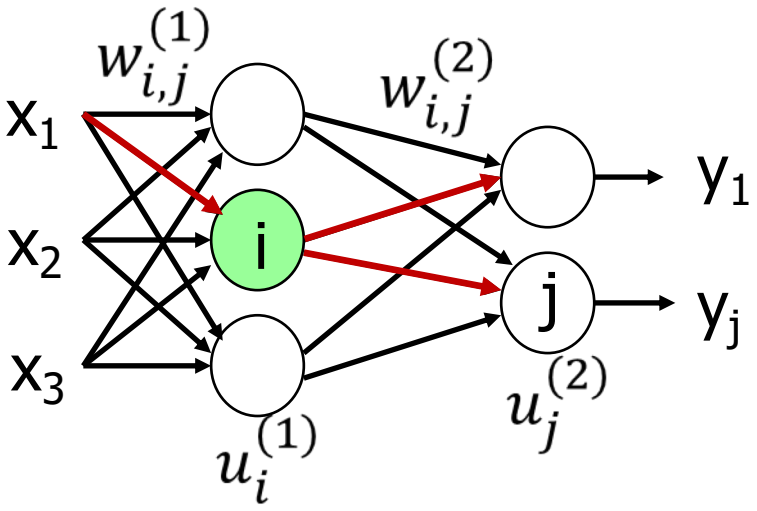
Loss function

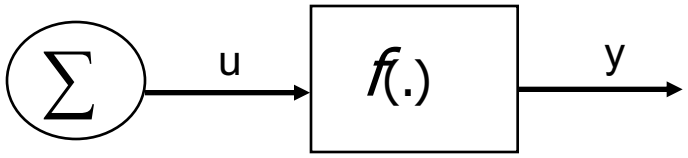
$$E_k(n) = \frac{1}{2}\left(t_k(n) - y_k(n)\right)^2$$

$$E(n) = \sum_k E_k(n)$$

$$\frac{\partial E(n)}{\partial w_{ki}^{(1)}} = \frac{\partial E(n)}{\partial y_1(n)} \frac{\partial y_1(n)}{\partial u_1^{(2)}(n)} \frac{\partial u_1^{(2)}(n)}{\partial y_i^{(1)}(n)} \frac{\partial y_i^{(1)}(n)}{\partial w_{ki}^{(1)}} + \frac{\partial E(n)}{\partial y_2(n)} \frac{\partial y_2(n)}{\partial u_2^{(2)}(n)} \frac{\partial u_2^{(2)}(n)}{\partial y_i^{(1)}(n)} \frac{\partial y_i^{(1)}(n)}{\partial w_{ki}^{(1)}}$$

Starting from Simple Example





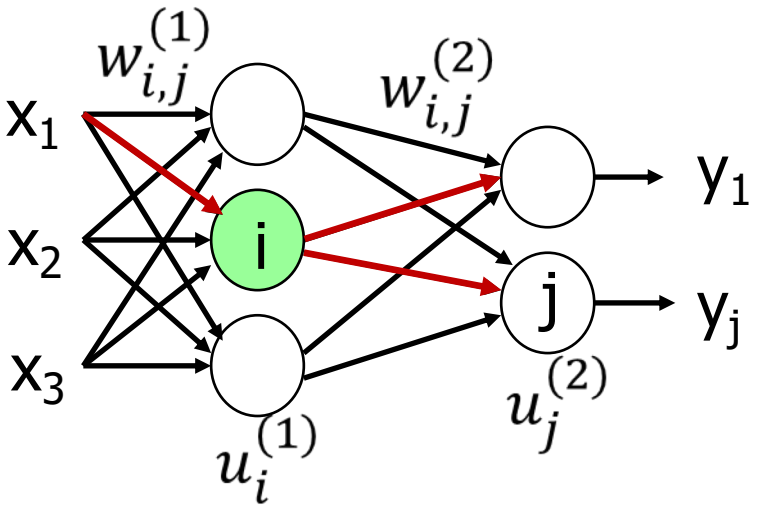
Loss function

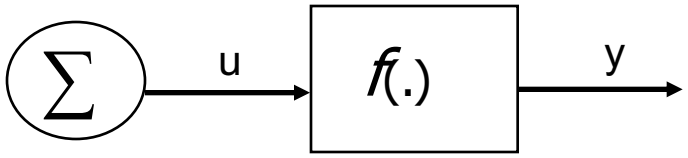
$$E_k(n) = \frac{1}{2}\left(t_k(n) - y_k(n)\right)^2$$

$$E(n) = \sum_k E_k(n)$$

$$\frac{\partial E(n)}{\partial w_{ki}^{(1)}} = \frac{\partial E(n)}{\partial y_1(n)} \frac{\partial y_1(n)}{\partial u_1^{(2)}(n)} \boldsymbol{w_{i1}^{(2)}} \frac{\partial y_i^{(1)}(n)}{\partial w_{ki}^{(1)}} + \frac{\partial E(n)}{\partial y_2(n)} \frac{\partial y_2(n)}{\partial u_2^{(2)}(n)} \boldsymbol{w_{i2}^{(2)}} \frac{\partial y_i^{(1)}(n)}{\partial w_{ki}^{(1)}}$$

Starting from Simple Example





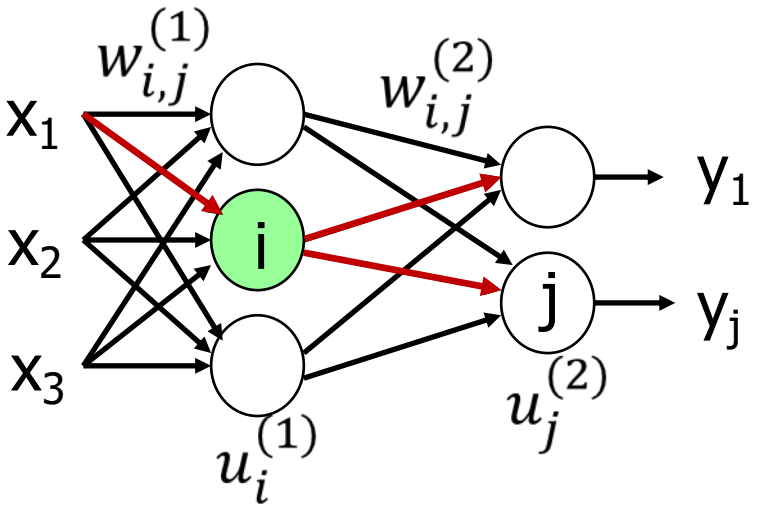
Loss function

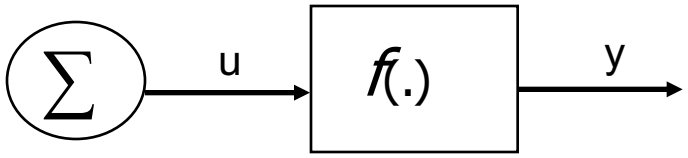
$$E_k(n) = \frac{1}{2}\left(t_k(n) - y_k(n)\right)^2$$

$$E(n) = \sum_k E_k(n)$$

$$\frac{\partial E(n)}{\partial w_{ki}^{(1)}} = \frac{\partial E(n)}{\partial u_1^{(2)}(n)} w_{i1}^{(2)} \frac{\partial y_i^{(1)}(n)}{\partial w_{ki}^{(1)}} + \frac{\partial E(n)}{\partial u_2^{(2)}(n)} w_{i2}^{(2)} \frac{\partial y_i^{(1)}(n)}{\partial w_{ki}^{(1)}}$$

Starting from Simple Example





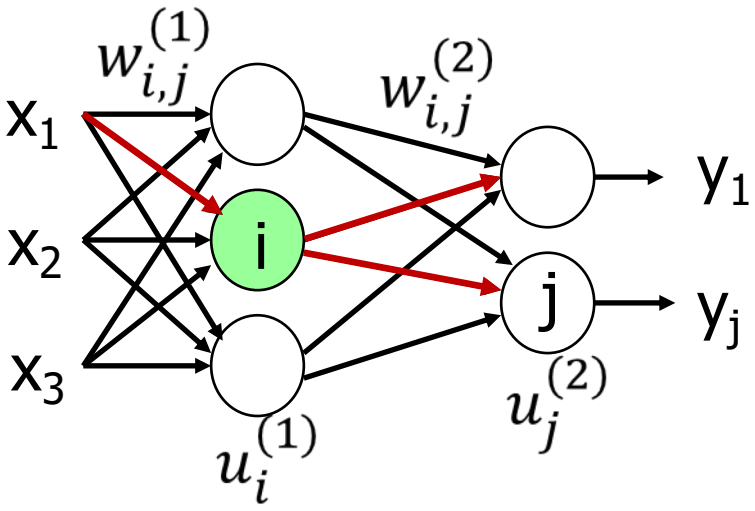
Loss function

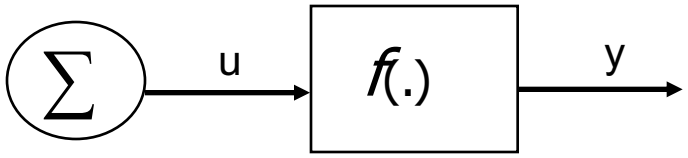
$$E_k(n) = \frac{1}{2}\left(t_k(n) - y_k(n)\right)^2$$

$$E(n) = \sum_k E_k(n)$$

$$\frac{\partial E(n)}{\partial w_{ki}^{(1)}} = \boldsymbol{\delta_1^{(2)}(n)} w_{i1}^{(2)} \frac{\partial y_i^{(1)}(n)}{\partial w_{ki}^{(1)}} + \boldsymbol{\delta_2^{(2)}(n)} w_{i2}^{(2)} \frac{\partial y_i^{(1)}(n)}{\partial w_{ki}^{(1)}}$$

Starting from Simple Example





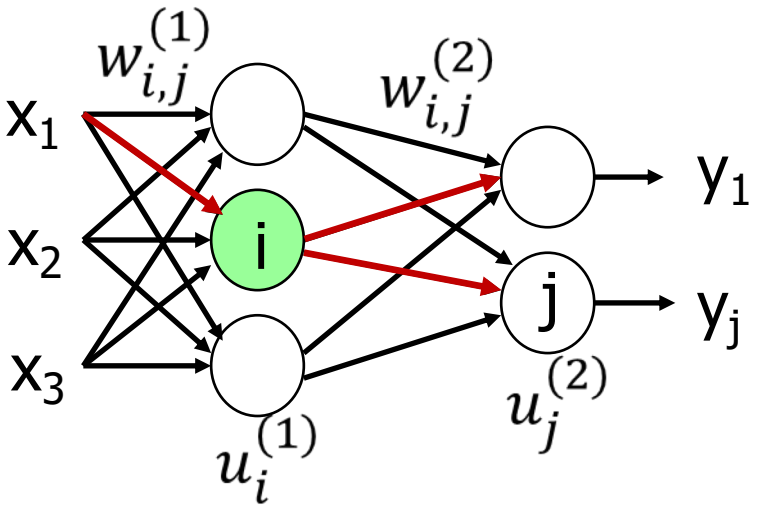
Loss function

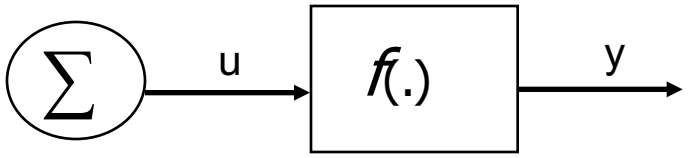
$$E_k(n) = \frac{1}{2}\left(t_k(n) - y_k(n)\right)^2 \qquad E(n) = \sum_k E_k(n)$$

$$\frac{\partial E(n)}{\partial w_{ki}^{(1)}} = \frac{\partial y_i^{(1)}(n)}{\partial w_{ki}^{(1)}}\left(\delta_1^{(2)}(n) w_{i1}^{(2)} + \delta_2^{(2)}(n) w_{i2}^{(2)}\right)$$

Local gradient Local gradient

Starting from Simple Example





Loss function

$$E_k(n) = \frac{1}{2}\left(t_k(n) - y_k(n)\right)^2$$

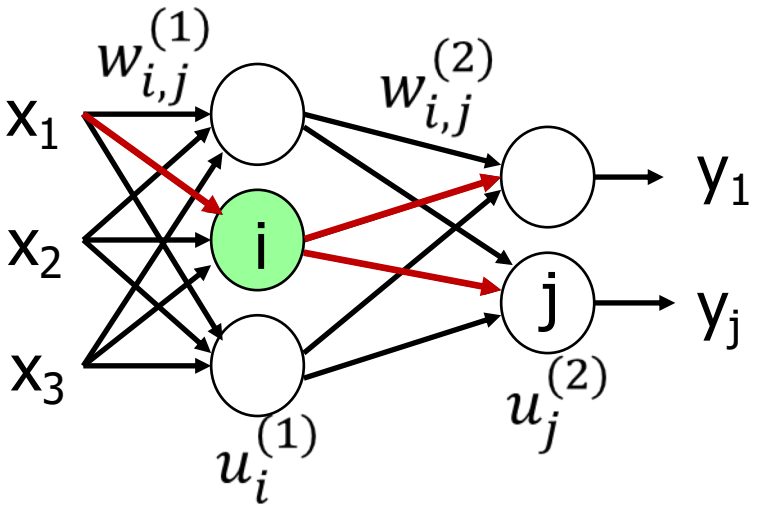
$$E(n) = \sum_k E_k(n)$$

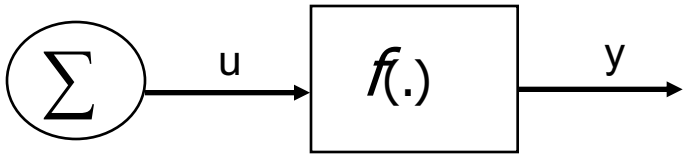
$$\frac{\partial E(n)}{\partial w_{ki}^{(1)}} = \frac{\partial E(n)}{\partial u_i^{(1)}(n)} \frac{\partial u_i^{(1)}(n)}{\partial w_{ki}^{(1)}}$$

$$= \frac{\partial y_i^{(1)}(n)}{\partial u_i^{(1)}(n)} \frac{\partial u_i^{(1)}(n)}{\partial w_{ki}^{(1)}} \left(\delta_1^{(2)}(n) w_{i1}^{(2)} + \delta_2^{(2)}(n) w_{i2}^{(2)}\right)$$

$$\frac{\partial E(n)}{\partial u_i^{(1)}(n)} = \frac{\partial y_i^{(1)}(n)}{\partial u_i^{(1)}(n)} \left(\delta_1^{(2)}(n) w_{i1}^{(2)} + \delta_2^{(2)}(n) w_{i2}^{(2)}\right)$$

Starting from Simple Example





Loss function

$$E_k(n) = \frac{1}{2}\left(t_k(n) - y_k(n)\right)^2$$

$$E(n) = \sum_k E_k(n)$$

$$\delta_i^{(1)}(n) = \frac{\partial y_i^{(1)}(n)}{\partial u_i^{(1)}(n)}\left(\delta_1^{(2)}(n) w_{i1}^{(2)} + \delta_2^{(2)}(n) w_{i2}^{(2)}\right)$$

Local gradient **Local gradient** **Local gradient**

Error Back-propagation

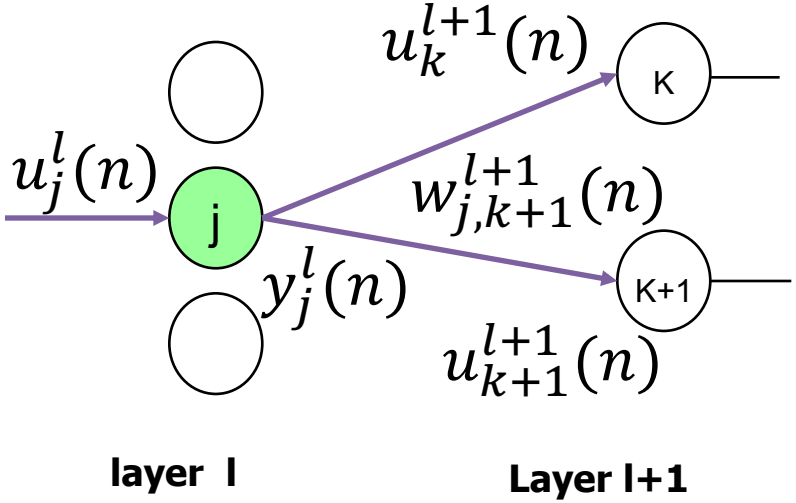
Local Gradient $\delta_j^l(n) = \frac{\partial E(n)}{\partial u_j^l(n)}$

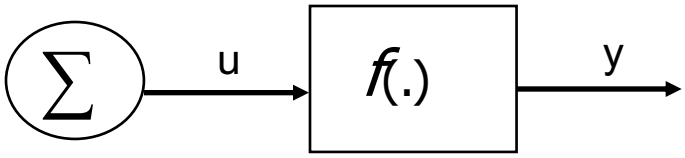
$$\frac{\partial E(n)}{\partial w_{kj}^l} = \frac{\partial E(n)}{\partial u_j^l(n)} \frac{\partial u_j^l(n)}{\partial w_{kj}^l} = \delta_j^l(n) y_k^{l-1}(n)$$

通过全连接层传递的error

激活函数求导

Layer propagation $\delta_j^l(n) = f'(u_j^l(n)) \sum_k \delta_k^{l+1}(n) w_{jk}^{l+1}$



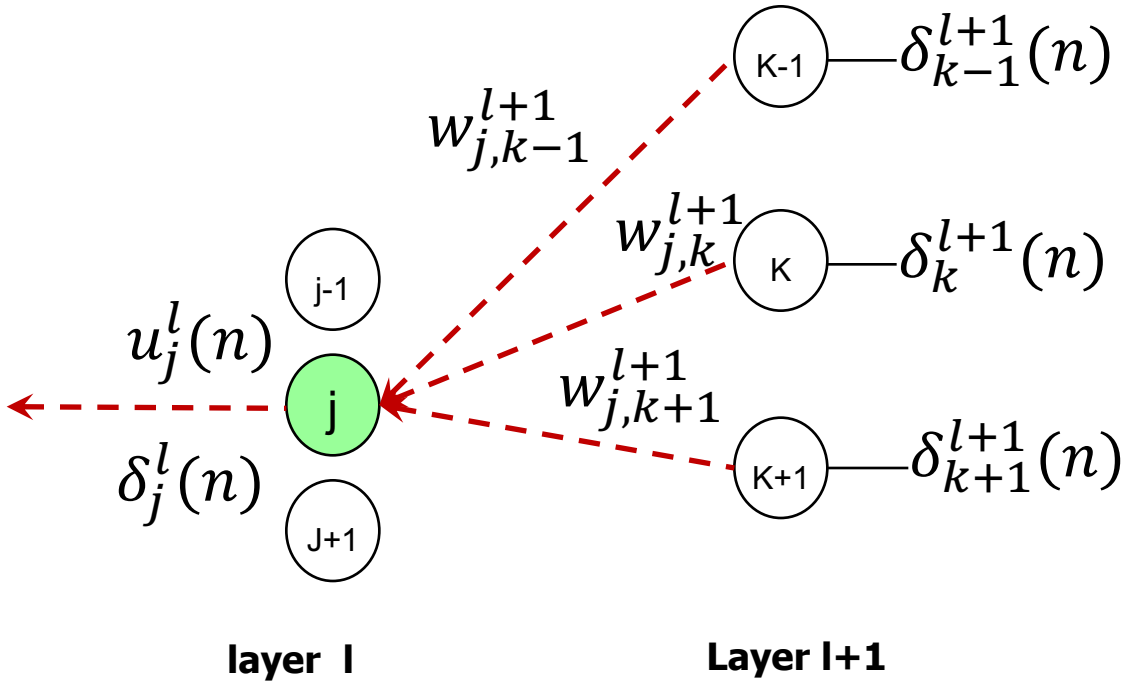


$$y_j^l(n) = f(u_j^l(n))$$

$$u_j^l(n) = \sum_k y_k^{l-1}(n) w_{kj}^l$$

Error Back-propagation (cont.)

$$\delta_j^l(n) = f'(u_j^l(n)) \sum_k \delta_k^{l+1}(n) w_{jk}^{l+1}$$



Vector Form?

Different Cost/Loss Functions

Depends on loss function

Activation of the output layer

Local gradient
Of the output layer

$$\delta_j^L = \frac{\partial E(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial u_j^{(L)}(n)}$$

实际上跑出来的垃圾

Mean Square Error

$$E_k(n) = \frac{1}{2}\left(t_k(n) - y_k(n)\right)^2$$

$$E(n) = \sum_k E_k(n)$$

监督下的目标输出（给定）

Cross Entropy

$$E_k(n) = -\sum_k t_{k(n)} log p_k(n)$$

$$p_k(n) = \frac{\exp(y_k)}{\sum_m \exp(y_m)}$$

Different Activation Functions

$$\delta_j^l(n) = f'(u_j^l(n)) \sum_k \delta_k^{l+1}(n) w_{jk}^{l+1}$$

Sigmoid function

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

$$f'(x) = f(x)(1-f(x))$$

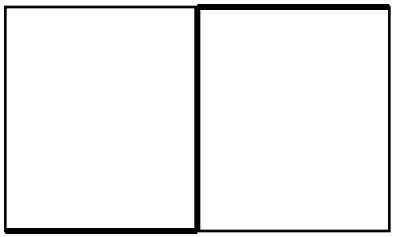
ReLU

$$f(x) = \max(0, x)$$

$$f'(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases}$$

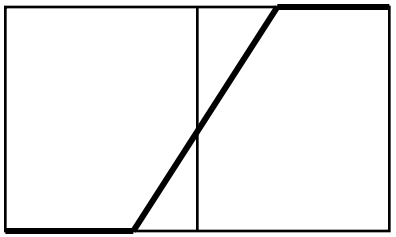
Activation Functions

- Threshold function



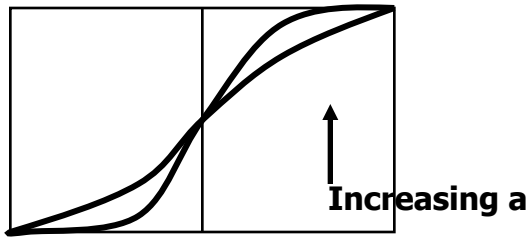
$$f(x) = \begin{cases} 1 \ if & v \geq 0 \\ 0 \ if & v < 0 \end{cases}$$

- Piecewise-linear function



$$f(v) = \begin{cases} 1 & v \geq \frac{1}{2} \\ v + \frac{1}{2} & \frac{1}{2} > v > -\frac{1}{2} \\ 0 & v \leq -\frac{1}{2} \end{cases}$$

- Sigmoid/Hyperbolic function



$$f(v) = \frac{1}{1 + \exp(-av)}$$

$$f(v) = \tanh(av)$$

Implementation

Computations

- **Forward pass** (start at the input layer)
  - Weights remain unchanged
  - Get activations of the neurons and final output
- **Backward pass** (start at the output layer)
  - Calculate δ for each neuron
  - Back-propagate δ from output to input
- **Weights update**
  - Weights change in accordance with delta rule

Learning Rate

- Smaller learning-rate parameter η, makes smaller changes to the weights
  - smoother trajectory in weight space (stable learning)

- Momentum term α

$$\Delta w_{ji}(\text{new}) = \alpha \Delta w_{ji}(\text{old}) - \eta \delta_i(n) y_j(n)$$

Note the subscripts

MLP Algorithm

W_1 = <M∗N ramdom floats>
W_2 = <K∗M random floats>
η = <positive number>
f_1 = <activation function for hidden layer>
f_2 = <activation function for output layer>
E = <error function>
X_i = <N dim inputs of sample i>
L_i = <K dim labels of sample i>

while not converge{
  for(int i=1;i<=m;i++){
    # forward pass
    $Y_i^{(1)} = f_1(W_1 \cdot X_i)$ (M dim)
    $Y_i^{(2)} = f_2(\mathrm{W}_2 \cdot Y_i^{(1)})$ (K dim)
    # backward pass
    $\delta_2 = \frac{\partial E\left(Y_i^{(2)}, L_i\right)}{\partial Y_i^{(2)}} \otimes f_2'(\mathrm{W}_2 \cdot Y_i^{(1)})$ (K dim)
    $\Delta W_2 = -\eta * \delta_2 \cdot Y_i^{(1)^T}$ (K∗M dim)
    $\delta_1 = (W_2^T \cdot \delta_2) \otimes f_1'(W_1 \cdot X_i)$ (M dim)
    $\Delta W_1 = -\eta * \delta_1 \cdot X_i^T$ (M∗N dim)
    $W_2 \mathrel{+}= \Delta W_2$
    $W_1 \mathrel{+}= \Delta W_1$
  }
}
return $\{W_1, W_2\}$

Training Model

- One complete presentation of the entire training set is called an ***epoch***.
  - Random order of training examples in each epoch
- Sequential model: (online/pattern/stochastic model)
  - Weight updating is performed after the presentation of each training example
- Batch model (**Gradient Descent**):
  - Weight updating is performed after the presentation of all the training examples.
- Mini-batch (**Stochastic Gradient Descent**):
  - Update gradients with a small number of examples

Training Stop Criteria

- The Euclidean norm of the gradient vector reaches a sufficiently small threshold
- The absolute rate of change in the loss per epoch is sufficiently small
- Early stopping: stop at a pre-determined epoch regardless of training results

Training Key Points

- **Weights** should be initialized randomly (but depends, sometimes sensitive)
- **Learning rate** should be suitable (too small vs. too large)
- Use a proper input and output **representations**: the way inputs and outputs represented can make a big difference
- Use training, test and validation sets to optimize hyper-parameters (**no overlap**)

Characteristics

Benefits

- Always give some answer even when the input information is not complete
- Neural network are good for **recognition** and **classification** problems (character recognition, analysis of time-series in financial data, etc)
- (MLP)Networks are easy to obtain

Limitations

- Not good for arithmetic and precise calculations (**multiplication**?)
- **Unexplainable** (e.g. air traffic control, medical diagnosis)
- Large neural networks are computationally expensive

Biological plausibility?

- Back-propagation is not in general biologically plausible
- There is no evidence of error propagating through several layers (or in most cases even a single layer)
- Back-propagation can be considered as a highly abstracted model of certain phenomena found in the brain
- Cerebellum does implement supervised learning for prediction but not quite like back-propagation

NO BP at all?

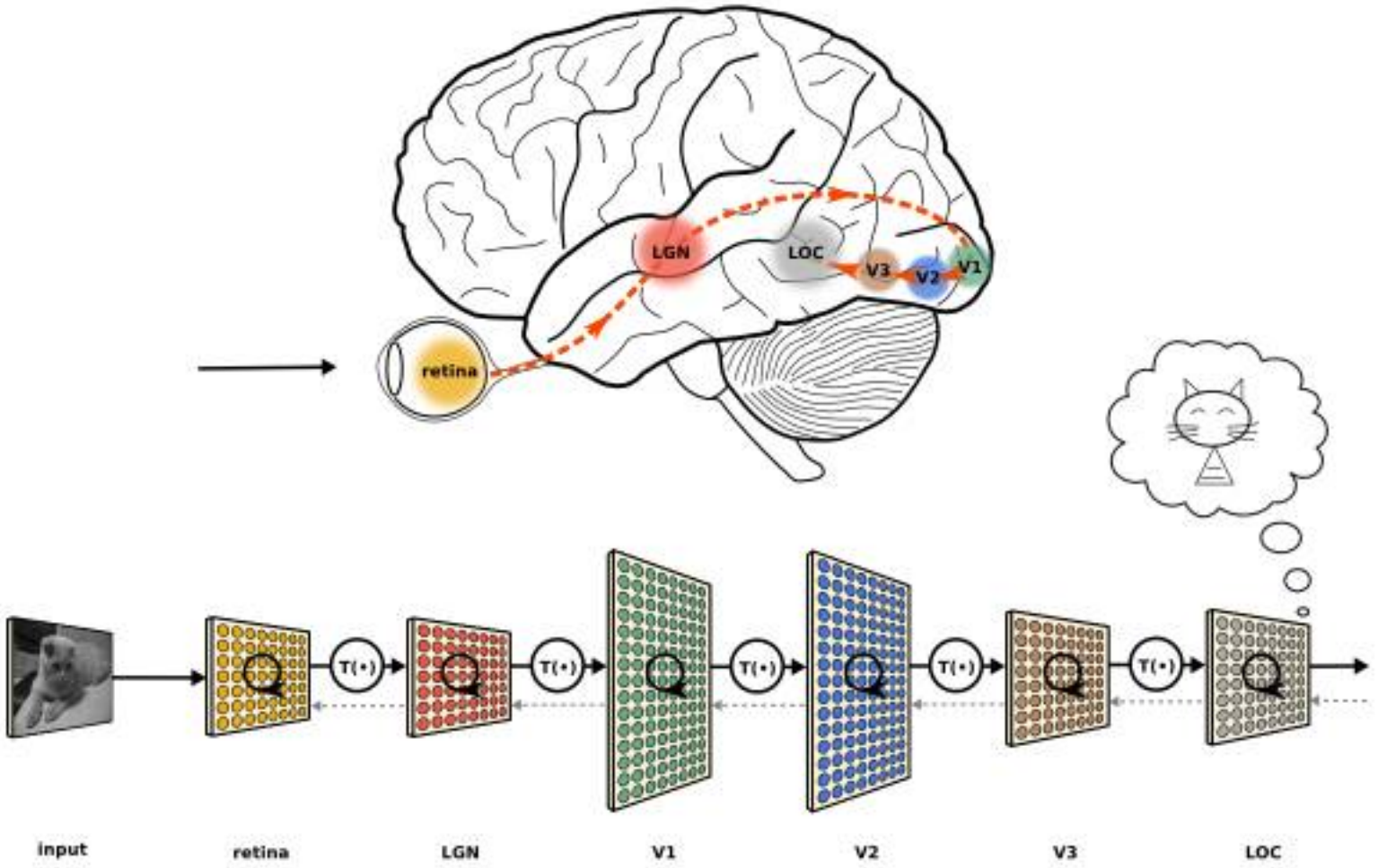
he is
"deeply suspicious" of back-propagation"
and
"My view is throw it all away and start again,"
The worry is that neural networks don't seem to learn like we do:
"I don't think it's how the brain works. We clearly don't need all the labeled data."

o Start

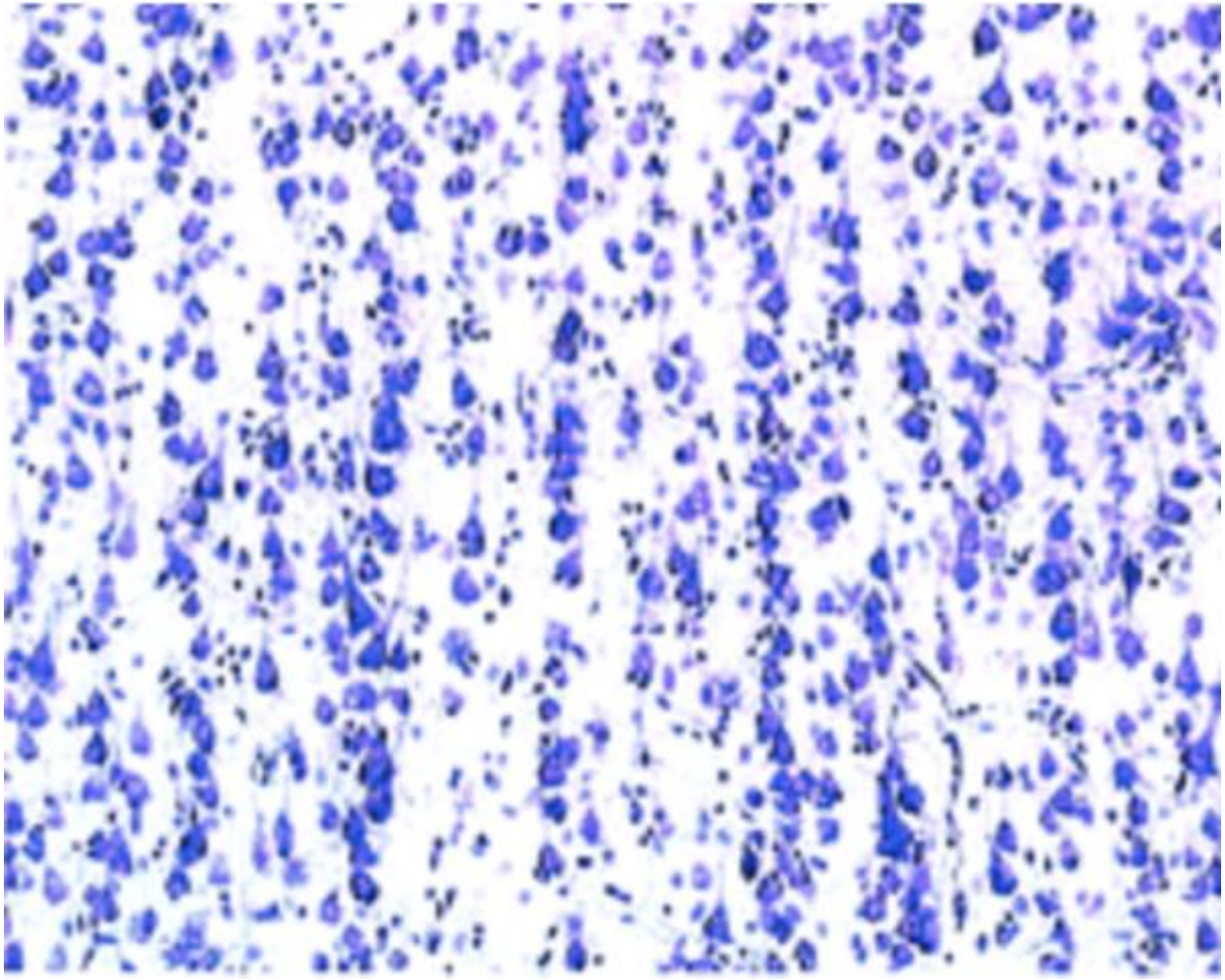
r of the current
t back
go when
ff interview, he
hat AI should

https://www.axios.com/ai-pioneer-advocates-starting-over-2485537027.html

Human Visual System



Mini-column = Capsule



Thanks for Your Attention