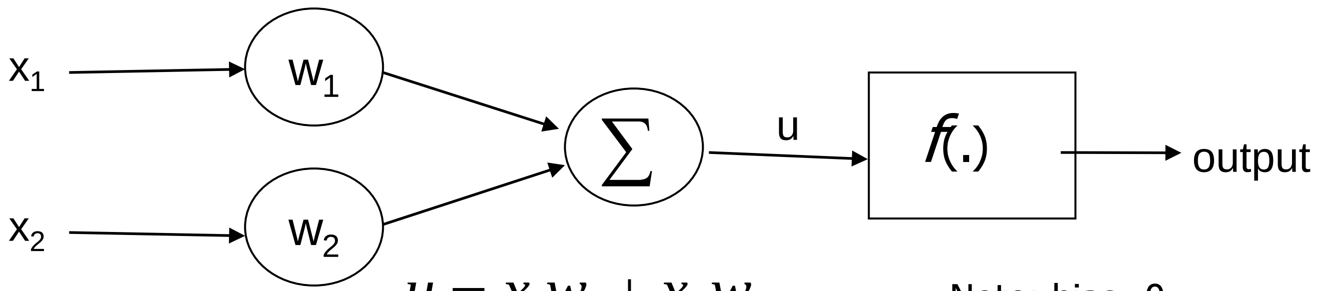
Chapter 2
Multilayer Perceptrons

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

XOR Problem

- Some classifications are impossible
- 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

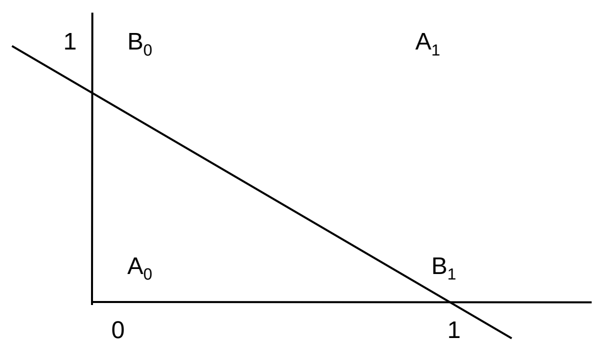


$$u = x_1 w_1 + x_2 w_2$$

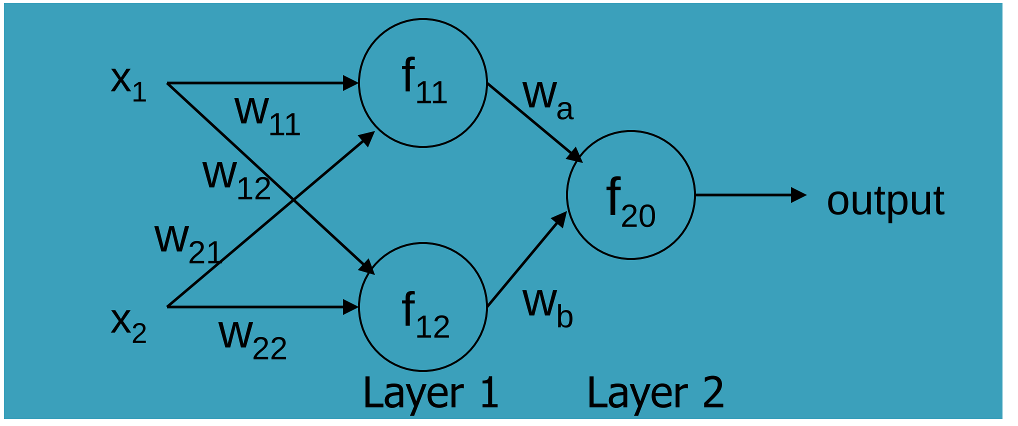
Note: bias=0

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

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

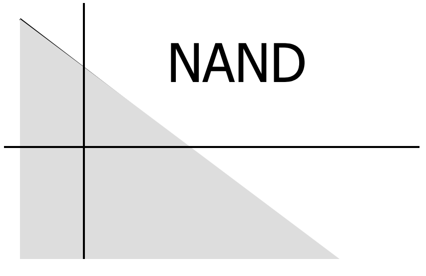


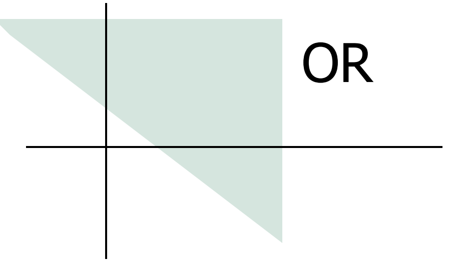
Note: bias=0

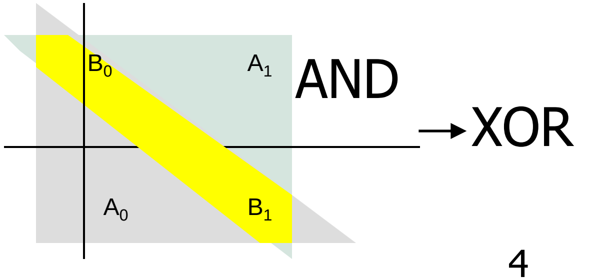
W_{11}=0.375 W_{12}=0.75 W_a=0.375

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

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







Why Perceptron

- Why (single-layer) Perceptron is limited
  - **Step or linear function**
- Difference between single layer and multilayer Perceptrons
  - Activation function
  - Learning algorithm
- Multilayer Perceptron 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

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

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

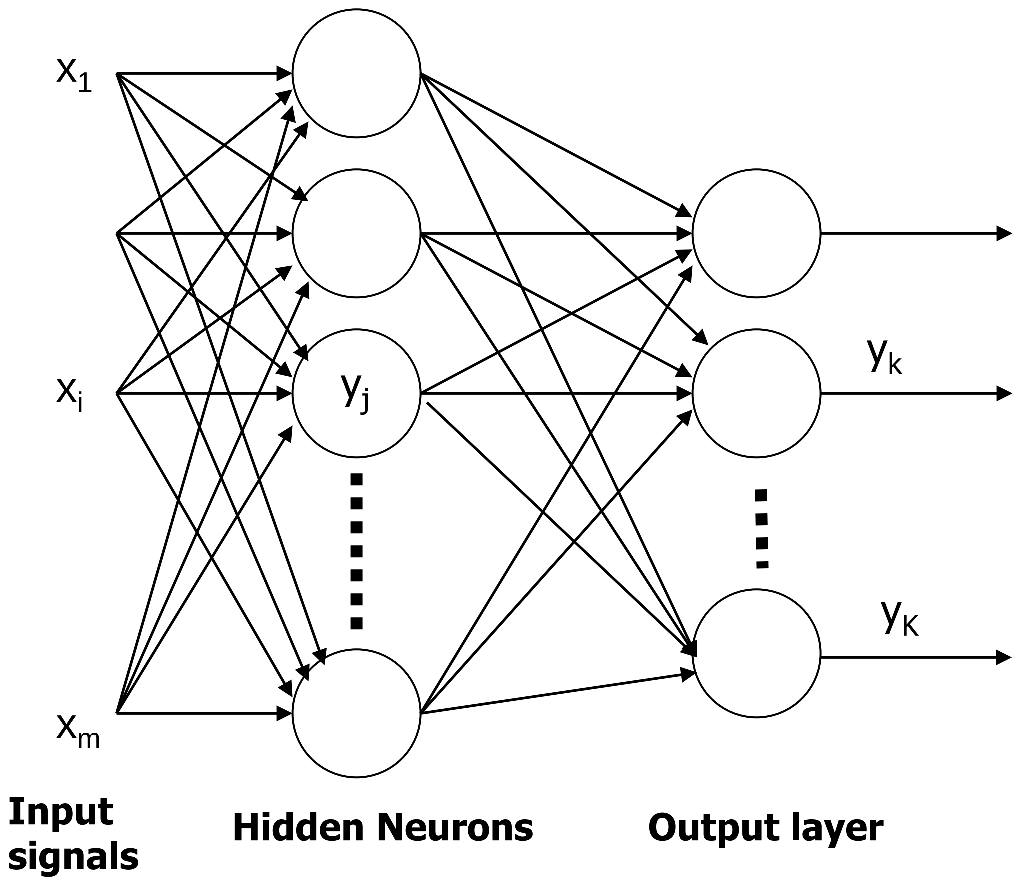
$$f'(x) = f(x)(1 - f(x))$$

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

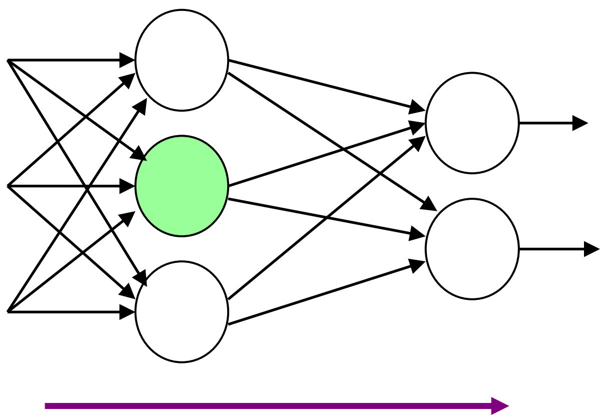
Structure

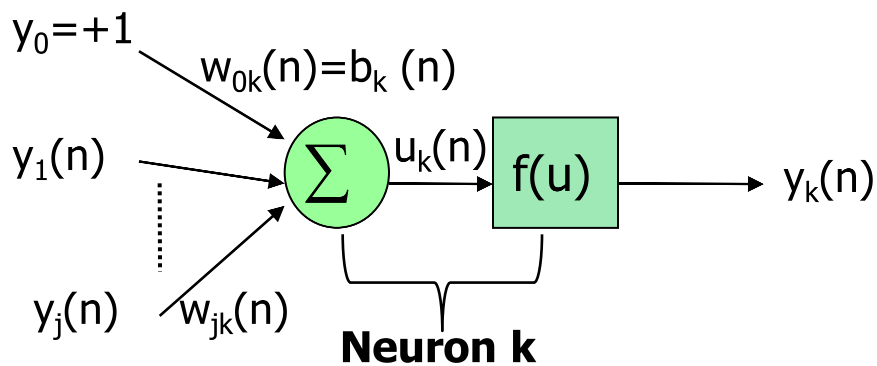
- Each layer has perceptrons which can extract (nonlinear) features
- Subsequent layers combine features
- Given enough hidden neurons, multilayer perceptron (MLP) can approximate any continuous function arbitrarily well
- Activation propagates from inputs to outputs

Network Structure



Forward Computing





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

Sigmoid

$$y_k(n) = sigmoid\left(u_k(n)\right)$$

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

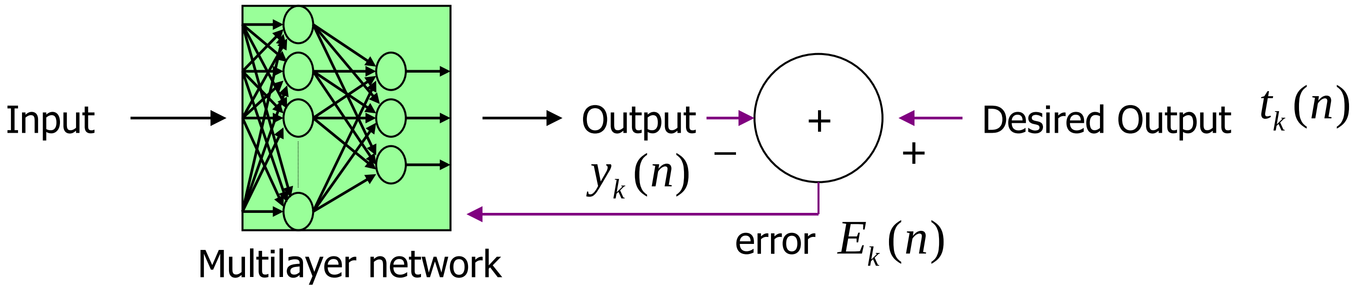
$$\mathrm{E}_k(n) = \frac{1}{2}(t_k(n) - y_k(n))^2$$

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

$$E_k(n) = -t_k \log y_k(n)$$

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

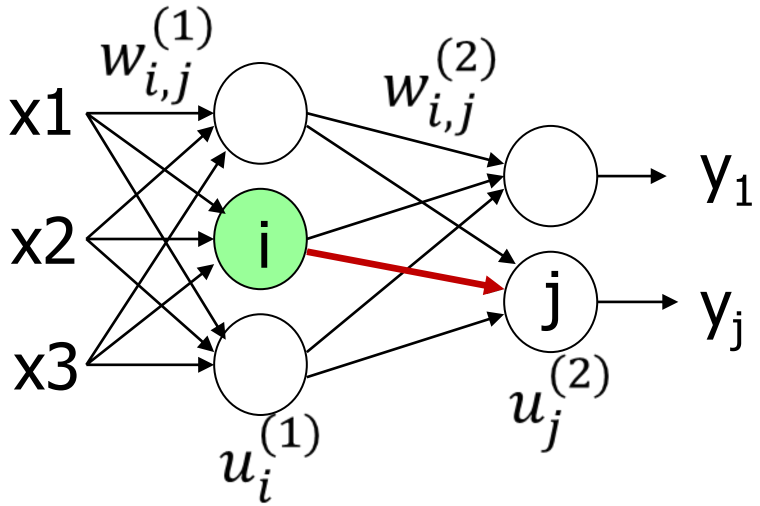
Error Propagation

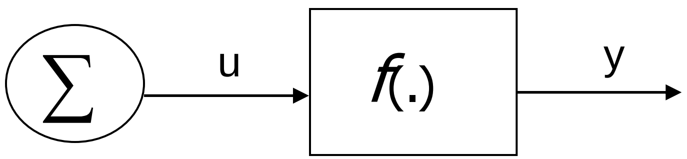


Synaptic adjustment $$\Delta w_{kj}(n) = -\eta \frac{\partial E(n)}{\partial w_{kj}}$$

Updated synaptic weight $$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n)$$

Starting from Simple Example





Loss function

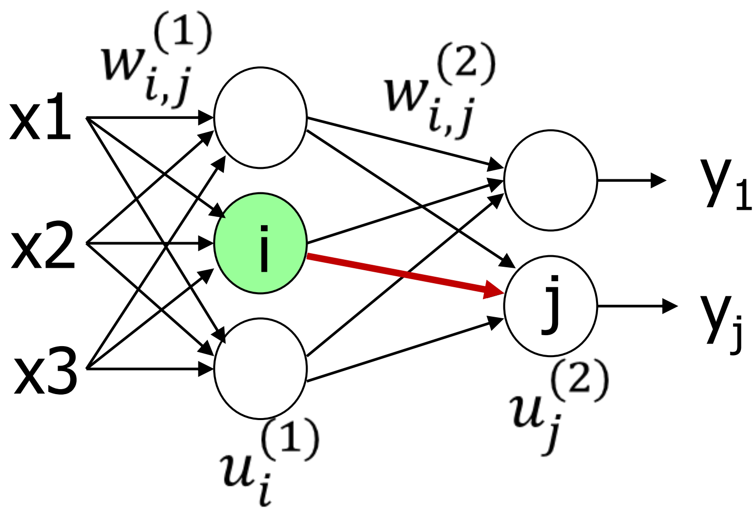
$$E_k(n) = \frac{1}{2}(t_k(n) - y_k(n))^2$$

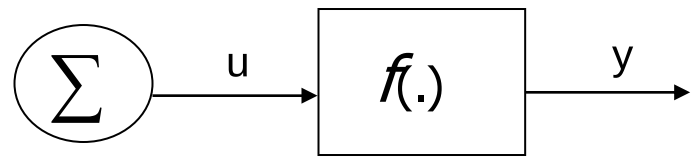
$$E(n) = \sum_k E_k(n)$$

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

$y_k(n)$-$t_k(n)$ $\quad f'(u_j^{(2)}(\mathrm{n}))$ $\quad y_i^{(1)}(n)$

Starting from Simple Example





Loss function

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

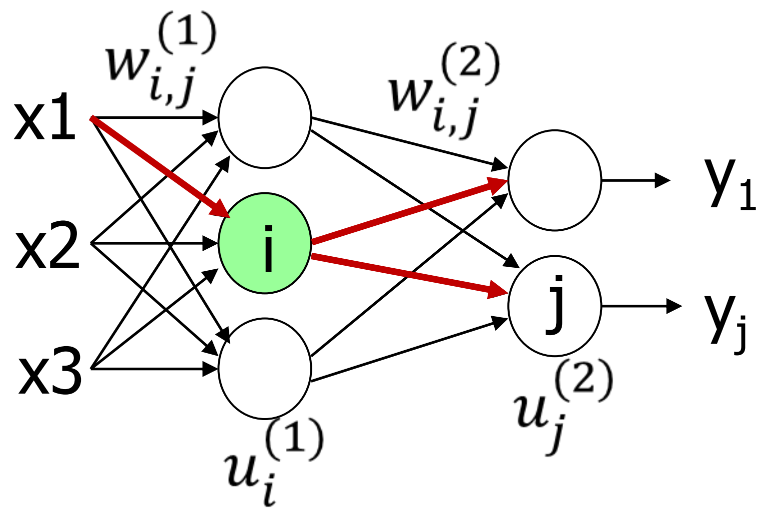
$$E(n) = \sum_k E_k(n)$$

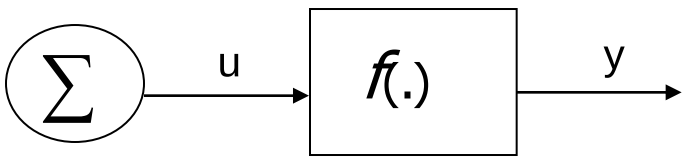
$$\frac{\partial E(n)}{\partial w_{ij}^{(2)}} = \frac{\partial E(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial u_j^{(2)}(n)} \frac{\partial u_j^{(2)}(n)}{\partial w_{ij}^{(2)}}$$

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

Local gradient

Starting from Simple Example





Loss function

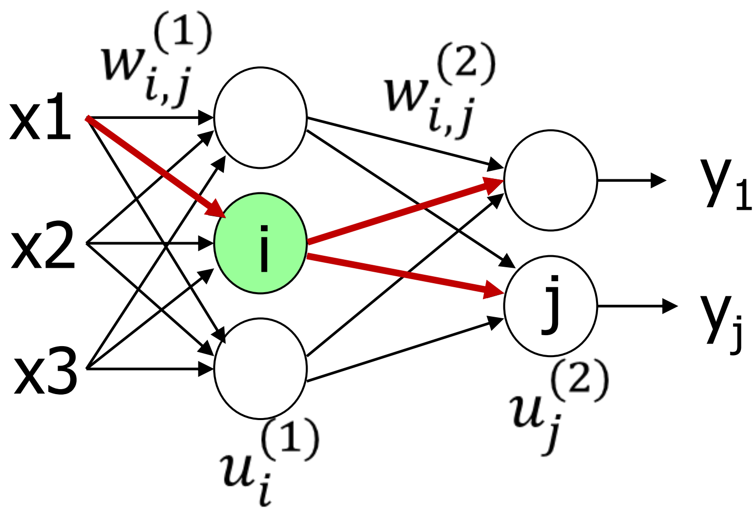
$$E_k(n) = \frac{1}{2}(t_k(n) - y_k(n))^2$$

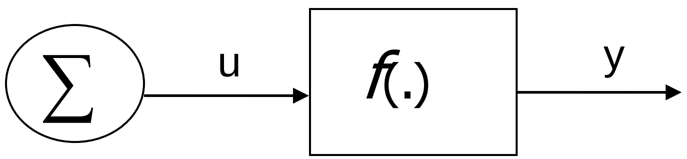
$$E(n) = \sum_k E_k(n)$$

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

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

Starting from Simple Example





Loss function

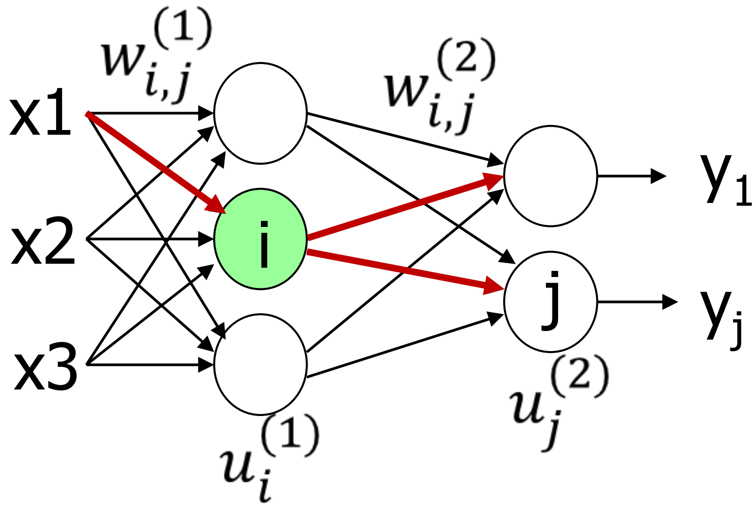
$$E_k(n) = \frac{1}{2}(t_k(n) - y_k(n))^2$$

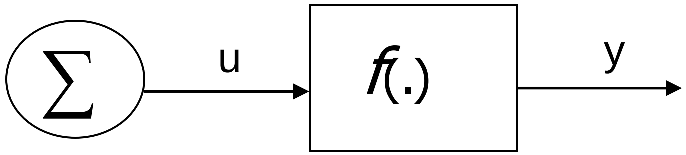
$$E(n) = \sum_k E_k(n)$$

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

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

Starting from Simple Example





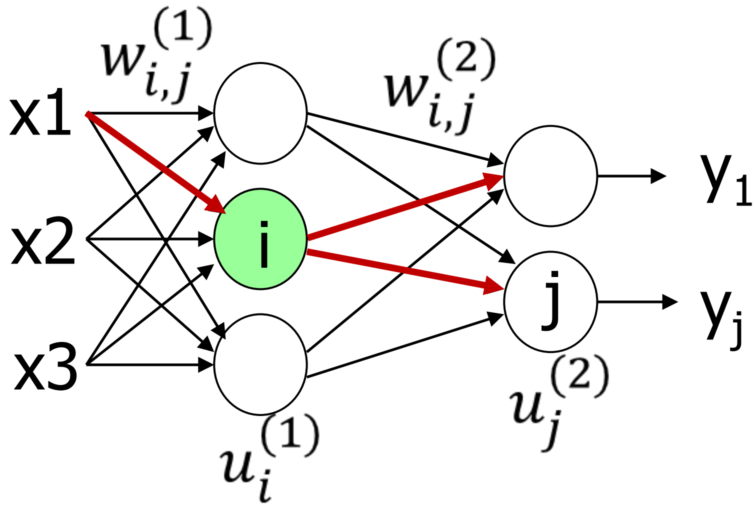
Loss function

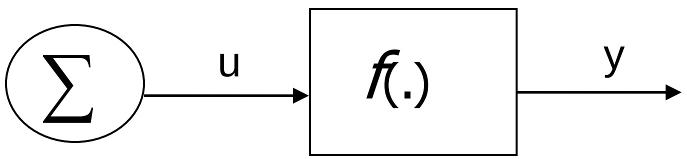
$$\mathrm{E}_k(n) = \frac{1}{2}(t_k(n) - y_k(n))^2$$

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

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

Starting from Simple Example





Loss function

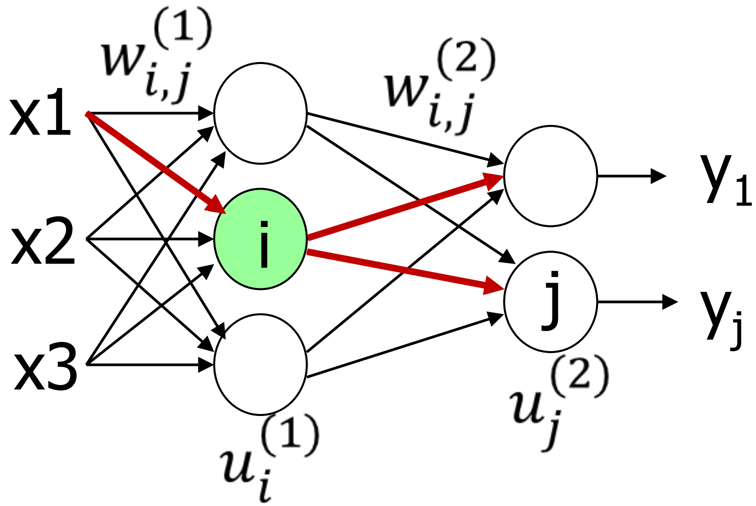
$$E_k(n) = \frac{1}{2}(t_k(n) - y_k(n))^2$$

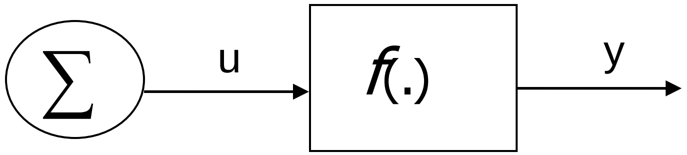
$$E(n) = \sum_k E_k(n)$$

$$\frac{\partial E(n)}{\partial w_{ki}^{(1)}} = \delta_1^{(2)}(n) w_{i1}^{(2)} * \frac{\partial y_i^{(1)}(n)}{\partial w_{ki}^{(1)}}$$

$$+\delta_2^{(2)}(n) w_{i2}^{(2)} * \frac{\partial y_i^{(1)}(n)}{\partial w_{ki}^{(1)}}$$

Starting from Simple Example





Loss function

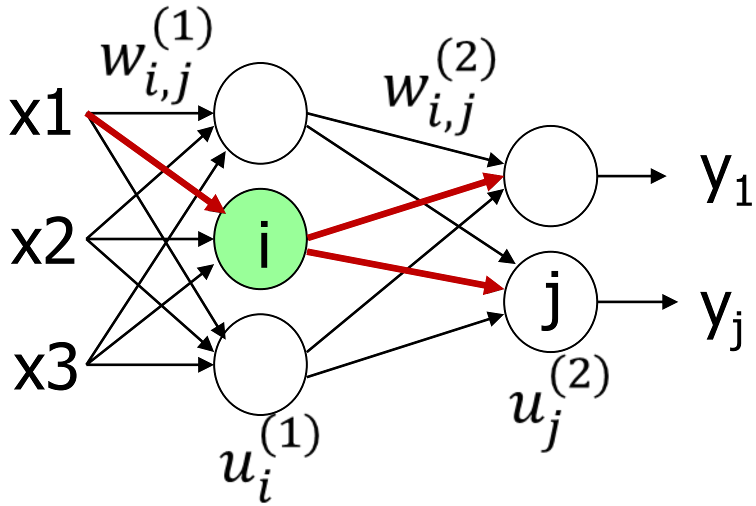
$$E_k(n) = \frac{1}{2}(t_k(n) - y_k(n))^2$$

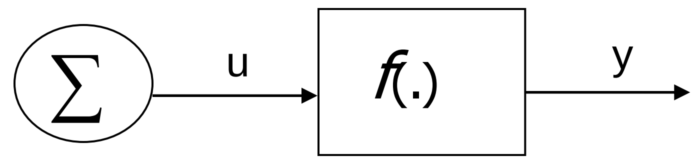
$$E(n) = \sum_k E_k(n)$$

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

Local gradient **Local gradient**

Starting from Simple Example





Loss function

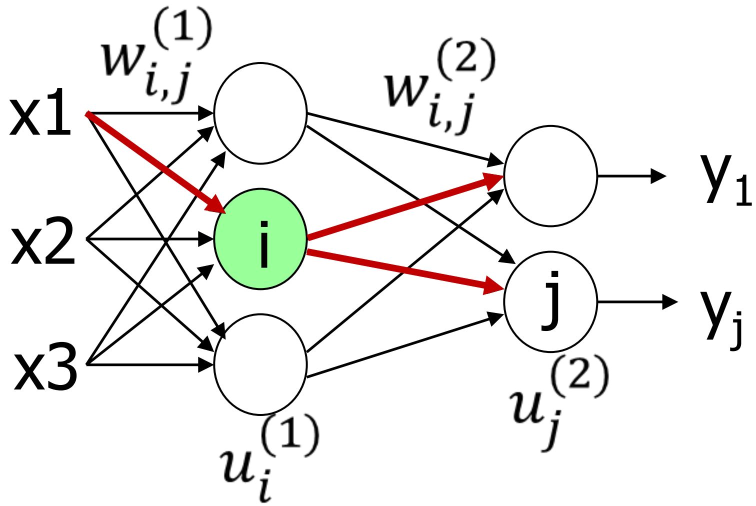
$$\mathrm{E}_k(n) = \frac{1}{2}(t_k(n) - y_k(n))^2$$

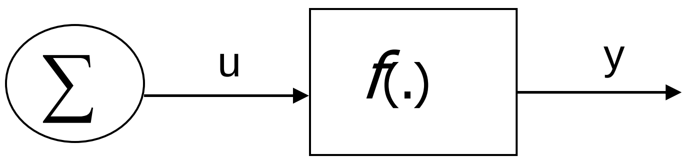
$$E(n) = \sum_k E_k(n)$$

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

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

Starting from Simple Example





Loss function

$$\mathrm{E}_k(n) = \frac{1}{2}(t_k(n) - y_k(n))^2$$

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

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

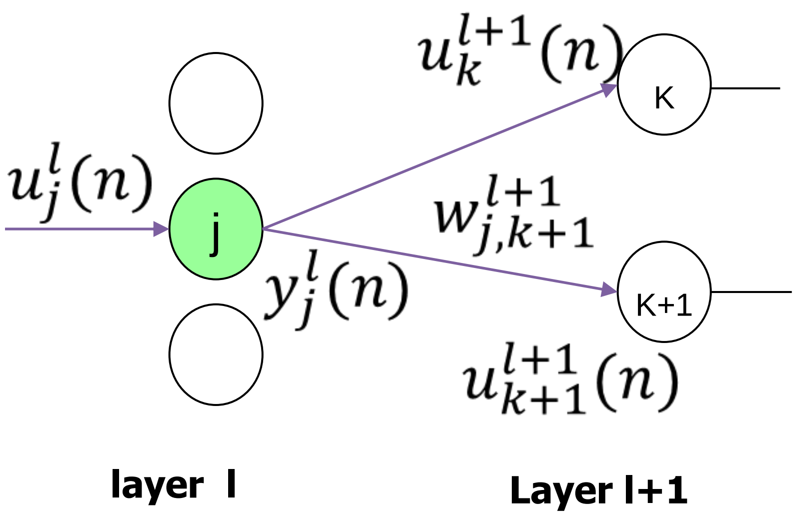
Local gradient **Local gradient** **Local gradient**

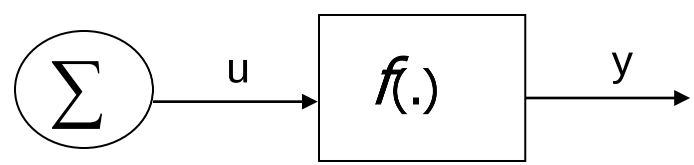
Error Back-propagation

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

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

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



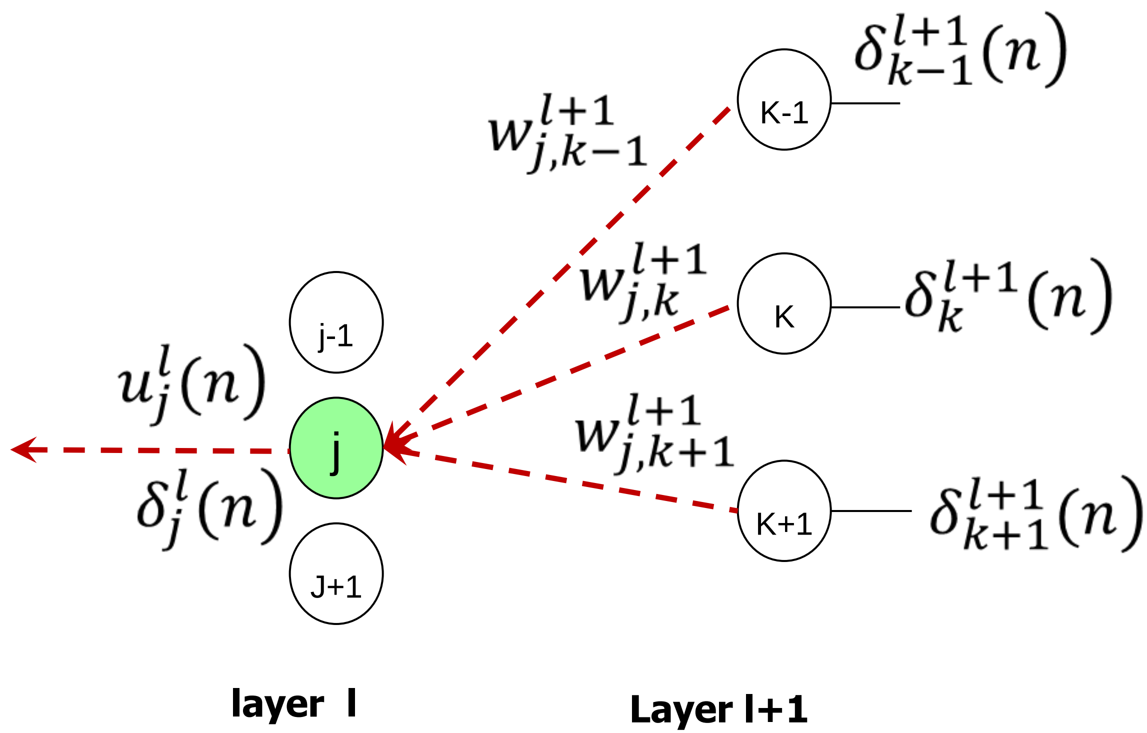


$$y_j^l(n) = f(u_j^l(n))$$

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

Error Back-propagation (cont.)

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



Different Cost Functions

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

Mean Square Error

$$\mathrm{E}_k(n) = \frac{1}{2}(t_k(n) - y_k(n))^2$$

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

Cross Entropy

$$\mathrm{E}_k(n) = -\sum_k t_k(n) \log p_k(n)$$

$$p_k(n) = \frac{\exp(y_{\mathrm{k}})}{\sum_m \exp(\mathrm{y}_m)}$$

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

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

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

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

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) - \eta \delta_j(n) y_i(n)$$

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?

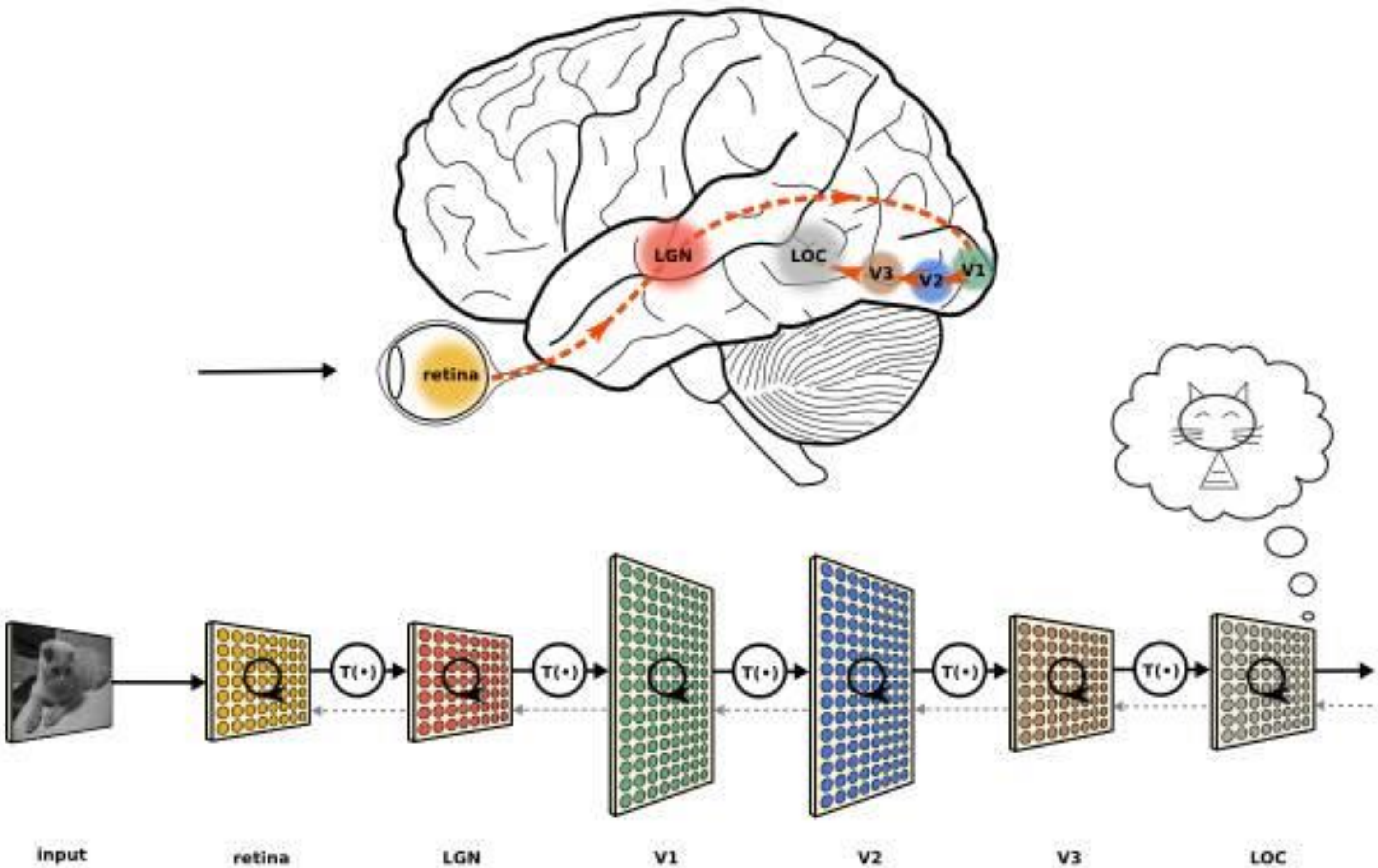
he is
"deeply suspicious" of back-propagation"
and
*"**My view is throw it all away and start again**,"*
The worry is that neural networks don't seem to learn like we do:
*"**I don't think it's how the brain works. We clearly don't need all the labeled data.**"*

o Start

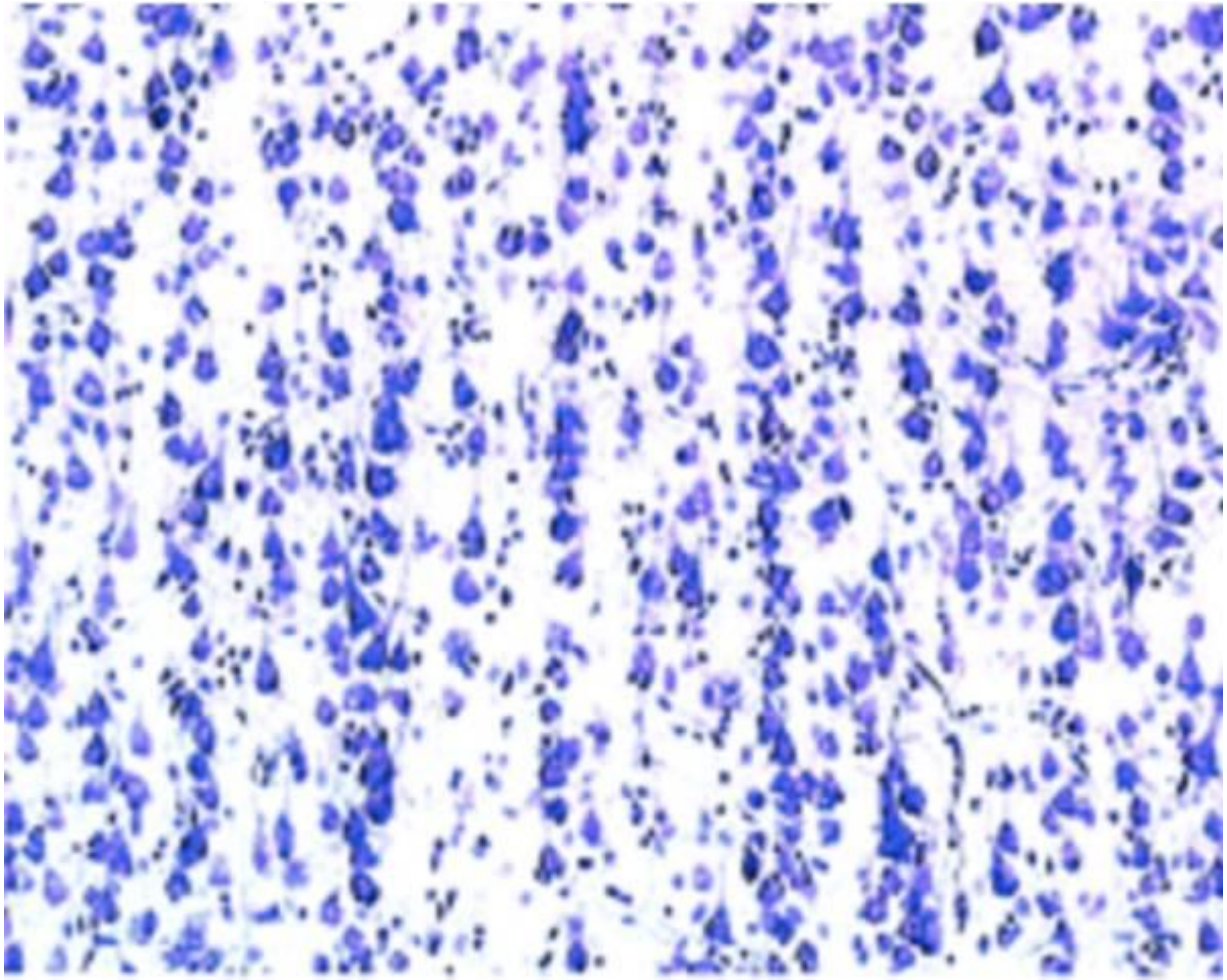
r of the current
it back
go when
ff interview, he
hat AI should

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

Human Visual System



Mini-column = Capsule



End