CS57300
PURDUE UNIVERSITY

FEBRUARY 26, 2019

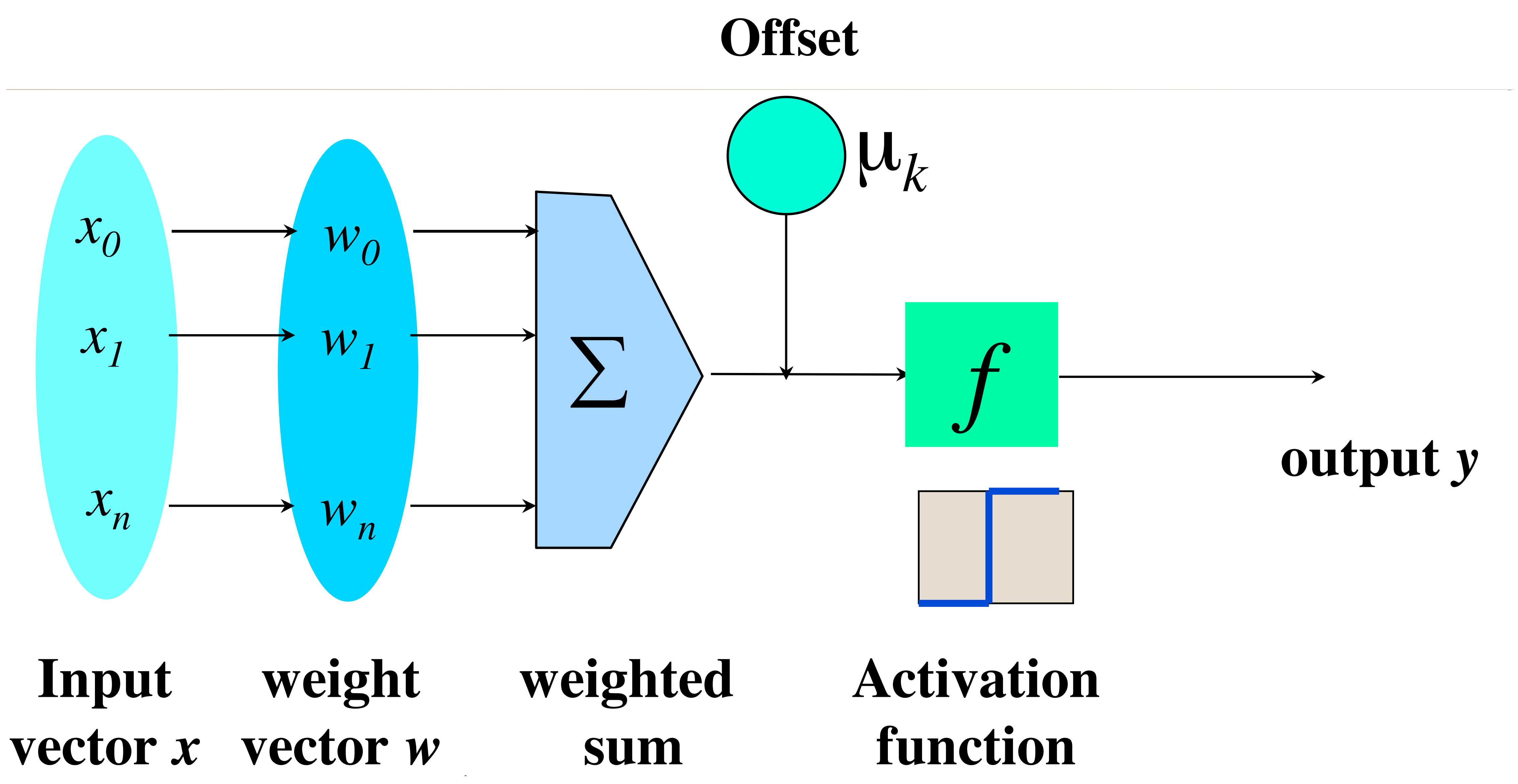
DATA MINING

ANNOUNCEMENTS

- Final project guideline is out
- Final project proposal
  - Due date: March 17 (11:59pm)
  - A two-page maximum document
- Final project pitch presentation
  - Final project pitch: In class (March 26), slides due on March 24 (11:59pm)
- No extension days for any project-related due dates

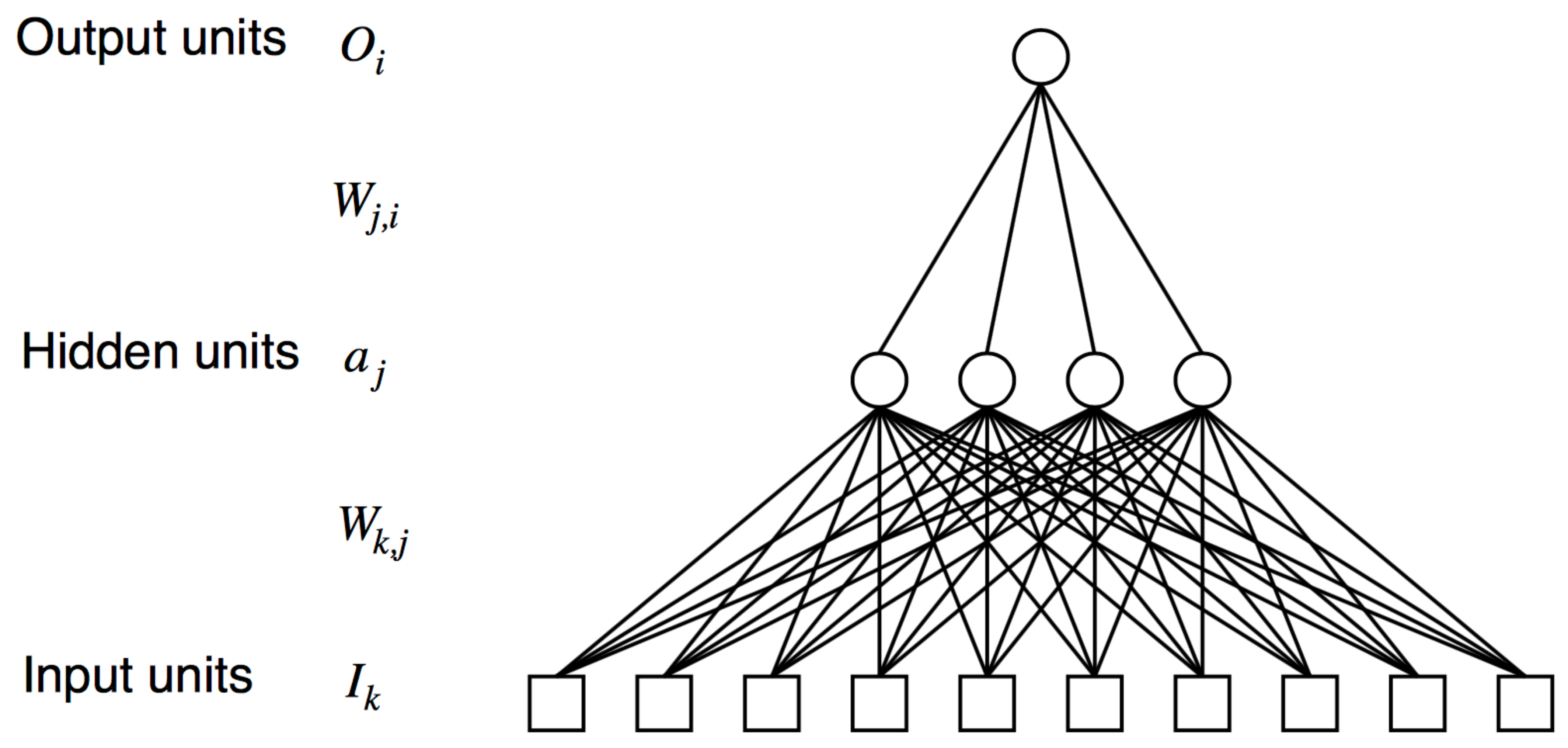
NEURAL NETWORK

NEURON


Offset
μ_k
x_0
x_1
x_n
w_0
w_1
w_n
Σ
f
output y
Input vector x
weight vector w
weighted sum
Activation function

MULTI-LAYER NEURAL NETWORK

- Increase expressive power by combining multiple perceptrons into ensemble
- Two-layer neural network: each perceptron output is a hidden unit, which are then aggregated into a final output



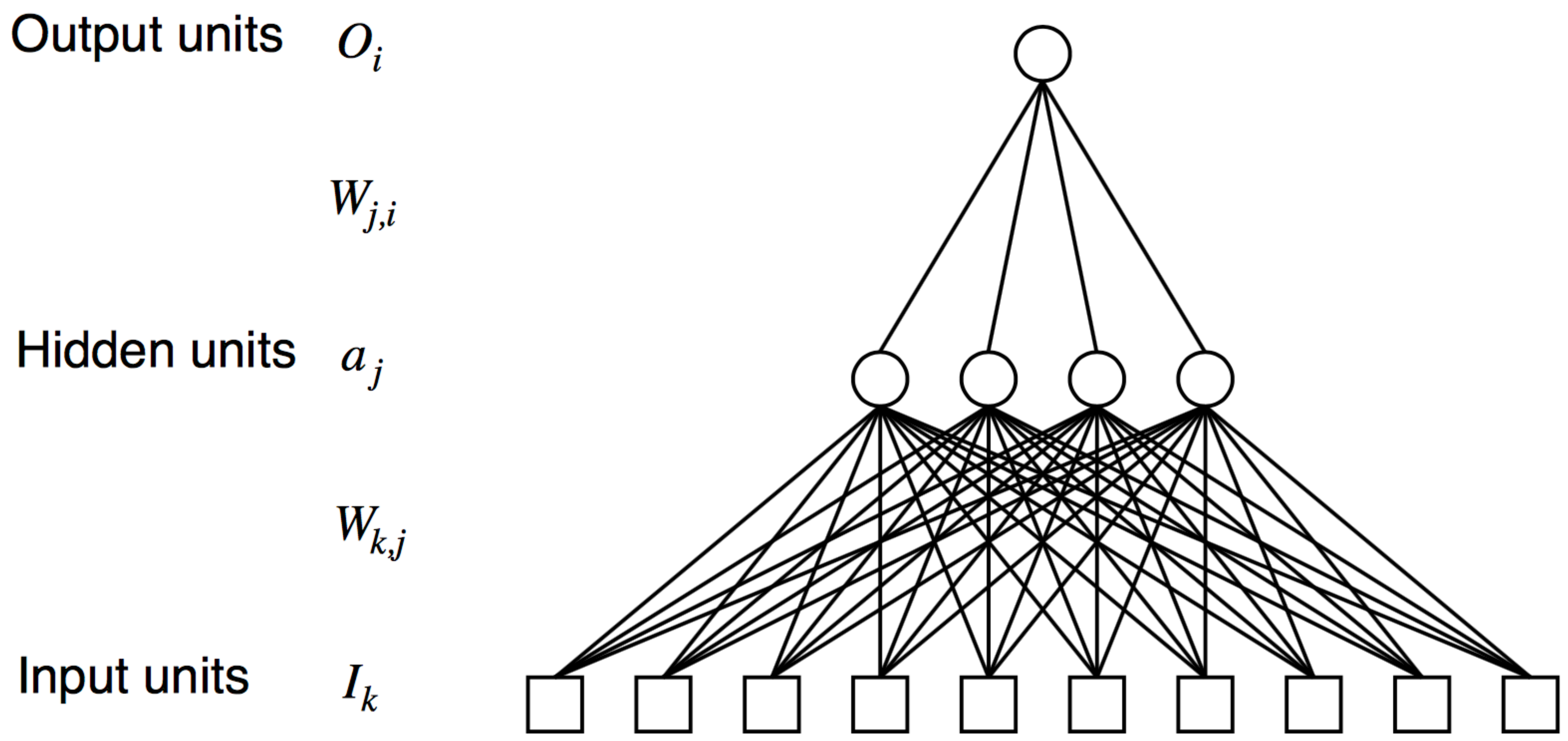
Output $$O_i = g(\sum_j W_{j,i} a_j)$$

Hidden units $$a_j = g(\sum_k W_{k,j} I_k)$$

Figure: M. Velosa

LEARNING MULTI-LAYER NEURAL NETWORKS

- Does the algorithm used for learning perceptron still work?



- Randomly set an initial set of weight
- Compute the outputs for each hidden unit and output unit
- Compare outputs from output unit and true labels, and update $W_{j,i}$
- Wait…what about weights associated with hidden units, $W_{k,j}$?

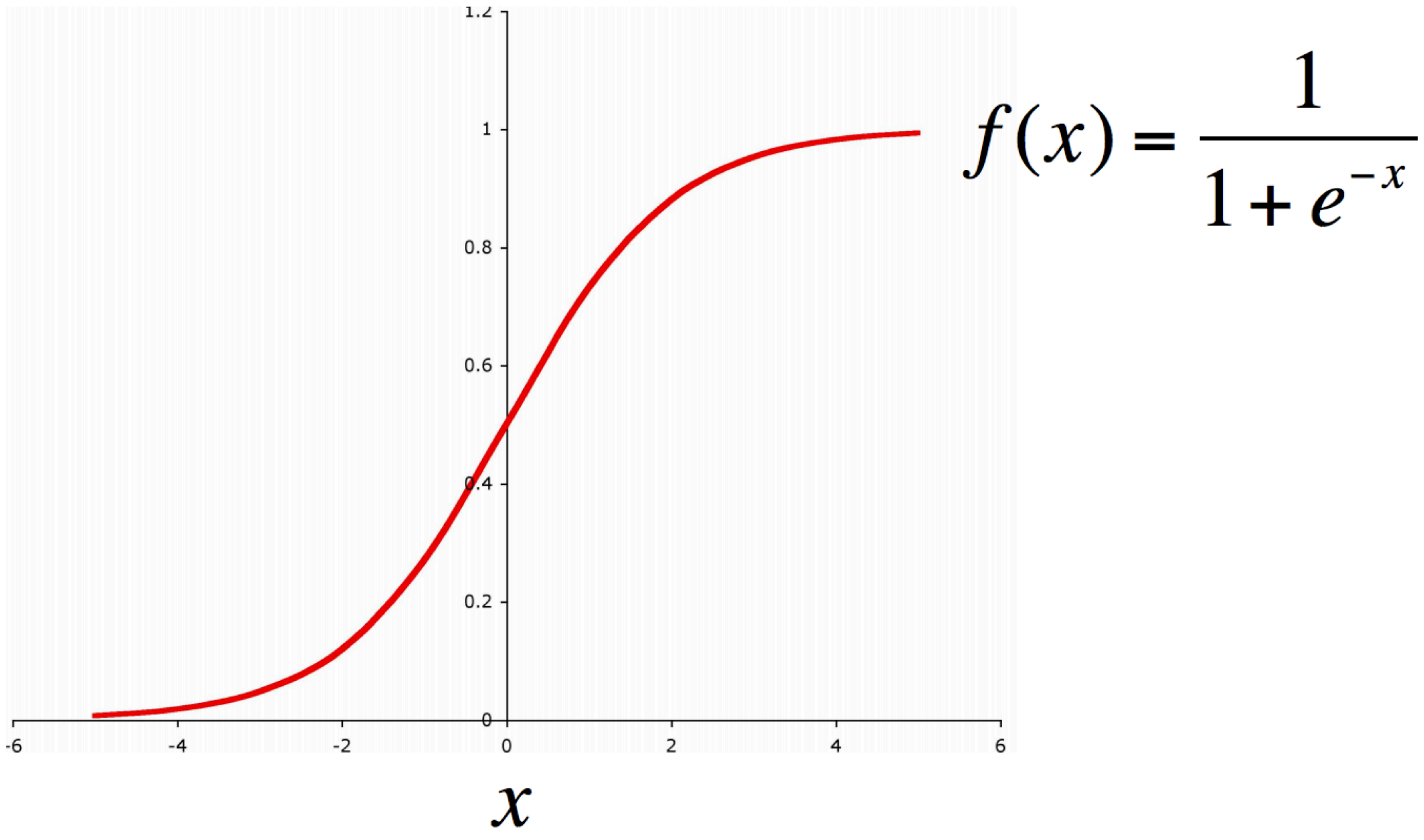
DIFFERENTIABLE SCORING FUNCTIONS AND ACTIVATION FUNCTIONS

- The scoring function *S* will take as inputs **x** (attributes), y (true label), $W_{k,j}$ (weights associated with hidden units), $W_{j,i}$ (weights associated with output units)
- If S is a differentiable function, we can use gradient-based optimization techniques to update weights!
- Differentiable scoring function: $E(\mathbf{w}) = \frac{1}{2}\sum_{d=1}^{N}(y^{(d)} - o^{(d)})^2$ instead of 0-1 loss
- Differentiable activation function: replacing step functions with something differentiable…

SIGMOID FUNCTION

- The output of a hidden unit (or an output unit) associated with weight ***w*** and input ***x*** will generate an output of:

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



HIGH-LEVEL GRADIENT-BASED LEARNING FRAMEWORK

Given a training dataset with N data points: $D = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$

Initialize the weights: $\mathbf{w}=\mathbf{w}_0$

Repeat

for each $(\mathbf{x}^{(d)}, y^{(d)})$ in D:

$o^{(d)} = f(\mathbf{w}, \mathbf{x}^{(d)})$, f is given by the neural network's structure

$$E(\mathbf{w}) = \frac{1}{2}\sum_{d=1}^{N}(y^{(d)} - o^{(d)})^2$$

Compute the error gradient for the entire set of training data

Compute the gradient: $\nabla E(\mathbf{w})$

Update: $\mathbf{w} = \mathbf{w} - \eta \nabla E(\mathbf{w})$

Until stopping criteria is met

BATCH LEARNING

STOCHASTIC GRADIENT-BASED LEARNING FRAMEWORK

Given a training dataset with N data points: $D = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$

Initialize the weights: $\mathbf{w}=\mathbf{w}_0$

Repeat

Randomly sample $n \in \{1,2,\ldots,N\}$:

$o^{(n)} = f(\mathbf{w}, \mathbf{x}^{(n)})$, f is given by the neural network's structure

$$E(\mathbf{w}) = \frac{1}{2}(y^{(n)} - o^{(n)})^2$$

Stochastic gradient descent

Compute the gradient: $\nabla E(\mathbf{w})$

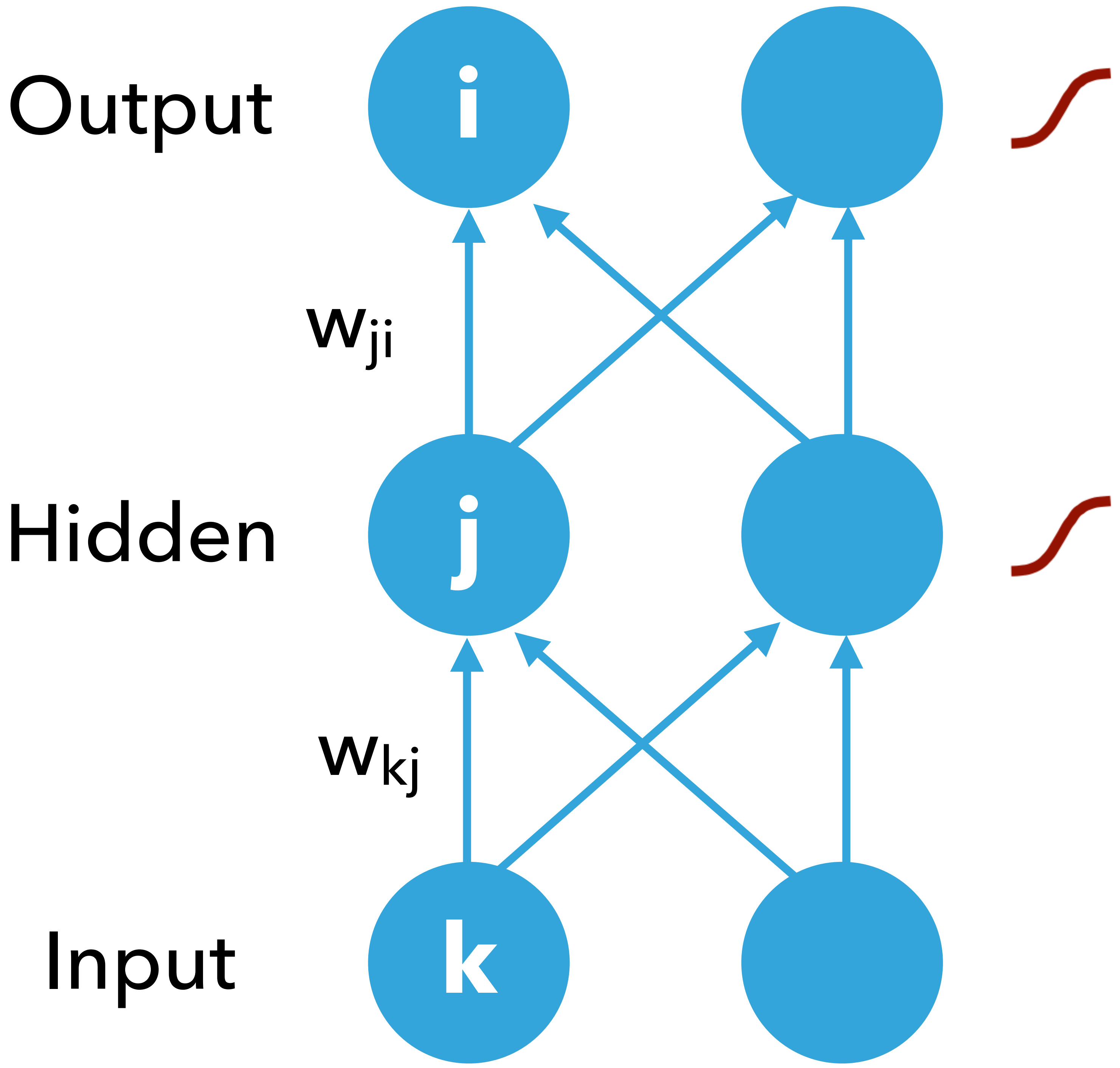
Update: $\mathbf{w} = \mathbf{w} - \eta \nabla E(\mathbf{w})$

Until stopping criteria is met

ONLINE LEARNING

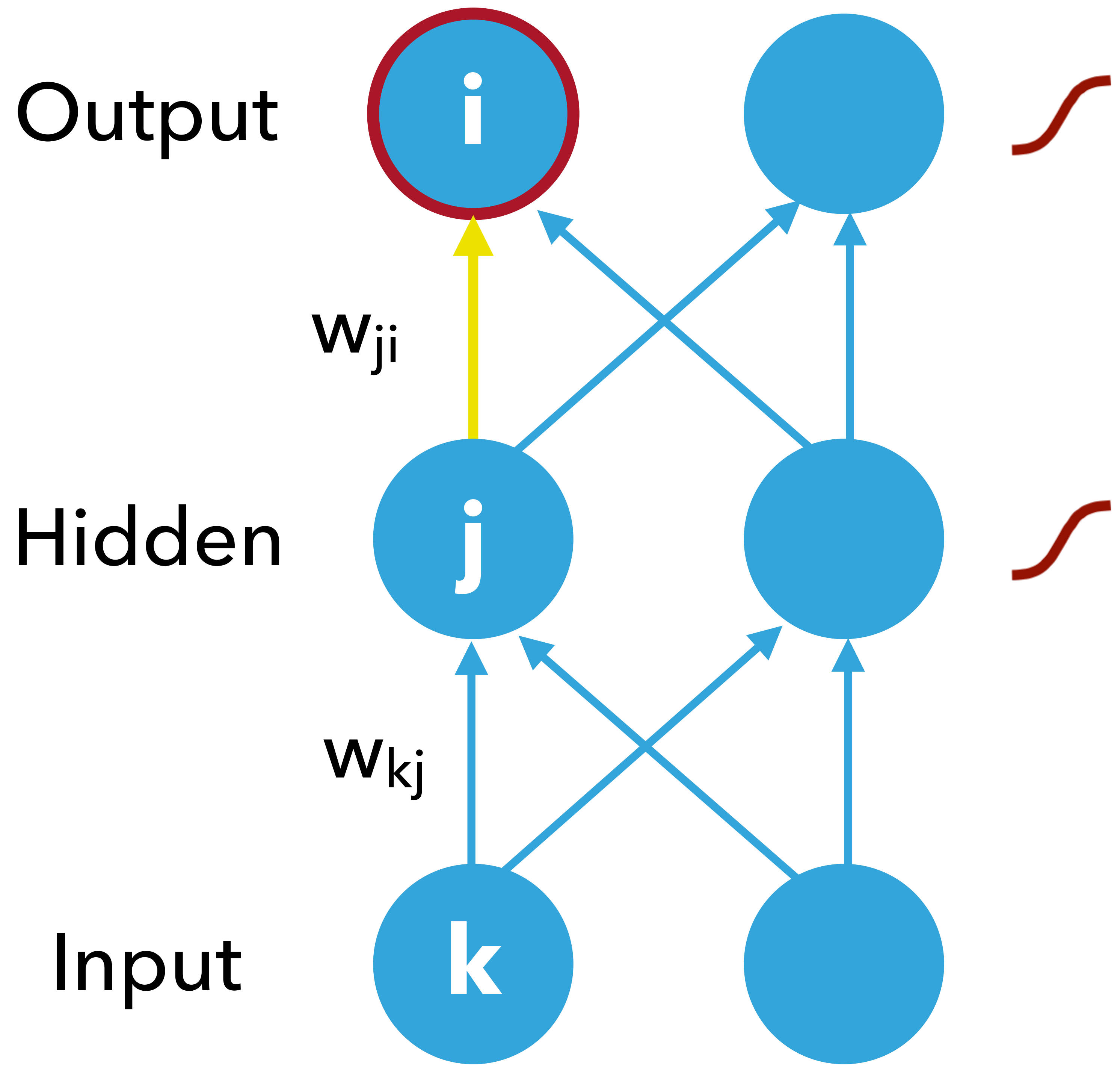
LEARNING NEURAL NETWORKS: SETUP

- Consider one training data (**x**, **y**), where the output **y** has M units
- Activation function for both hidden and output units are sigmoid functions
  - Suppose the output of node *z* is o_z, the input of node *z* is i_z (i_z=**x** if z is a hidden node, i_z are outputs of hidden nodes in the previous layer if z is an output node)
  - $o_z = \frac{1}{1+e^{-w^T i_z}}$, w is the weights associated with i_z
  - Denote $net_z = w^T i_z$, then $o_z = \frac{1}{1+e^{-net_z}}$



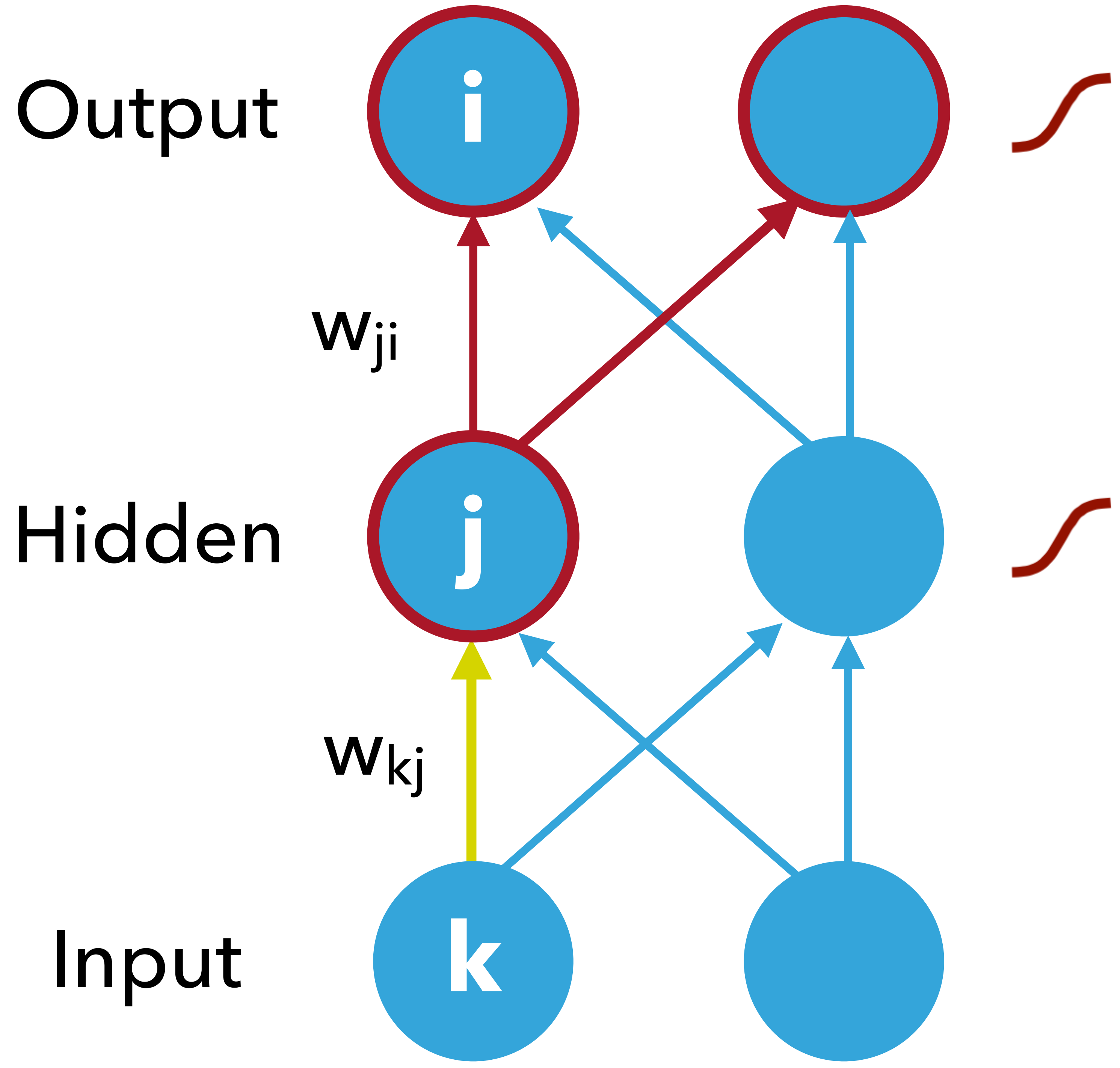
BACKPROPAGATION: LEARNING OUTPUT UNITS WEIGHTS

- Scoring function: $E(w) = \frac{1}{2}\sum_{m=1}^{M}(y_m - o_m)^2$
- Weights of output units w_{ji} will only affect E(w) through o_i
- $$\frac{\partial E(w)}{\partial w_{ji}} = \frac{\partial E(w)}{\partial o_i}\frac{\partial o_i}{\partial net_i}\frac{\partial net_i}{\partial w_{ji}}$$
  $$= -(y_i - o_i)\frac{\partial o_i}{\partial net_i}\frac{\partial net_i}{\partial w_{ji}}$$
  $$= -(y_i - o_i)o_i(1 - o_i)\frac{\partial net_i}{\partial w_{ji}}$$
  $$= -(y_i - o_i)o_i(1 - o_i)o_j$$



BACKPROPAGATION: LEARNING HIDDEN UNITS WEIGHTS

- Weights of hidden units w_{kj} will only affect E(w) through o_j
- Denote downstream(j) as the set of output units that take o_j as inputs
- $$\frac{\partial E(w)}{\partial w_{kj}} = \sum_{i \in downstream(j)} \frac{\partial E(w)}{\partial o_i} \frac{\partial o_i}{\partial net_i} \frac{\partial net_i}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{kj}}$$
  $$= \sum_{i \in downstream(j)} -(y_i - o_i) o_i (1 - o_i) \frac{\partial net_i}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{kj}}$$
  $$= \sum_{i \in downstream(j)} -(y_i - o_i) o_i (1 - o_i) w_{ji} o_j (1 - o_j) x_k$$



PUTTING TOGETHER: BACKPROPAGATION FOR LEARNING NEURAL NETWORK

Given a training data set with *N* data points: $D = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$

Initialize the weights: $\mathbf{w}=\mathbf{w}_0$

Repeat

Randomly sample $n \in \{1,2,...,N\}$:

Compute the outputs $o_z^{(n)}$ for each hidden/output node *z* given the current weight **w** and data $\mathbf{x}^{(n)}$

For output units weights: $\nabla w_{ji} = -(y_i^{(n)} - o_i^{(n)})o_i^{(n)}(1 - o_i^{(n)})o_j^{(n)}$

For hidden units weights: $\nabla w_{kj} = -\sum_{i \in downstream(j)} (y_i^{(n)} - o_i^{(n)})o_i^{(n)}(1 - o_i^{(n)})w_{ji}o_j^{(n)}(1 - o_j^{(n)})x_k^{(n)}$

Update:

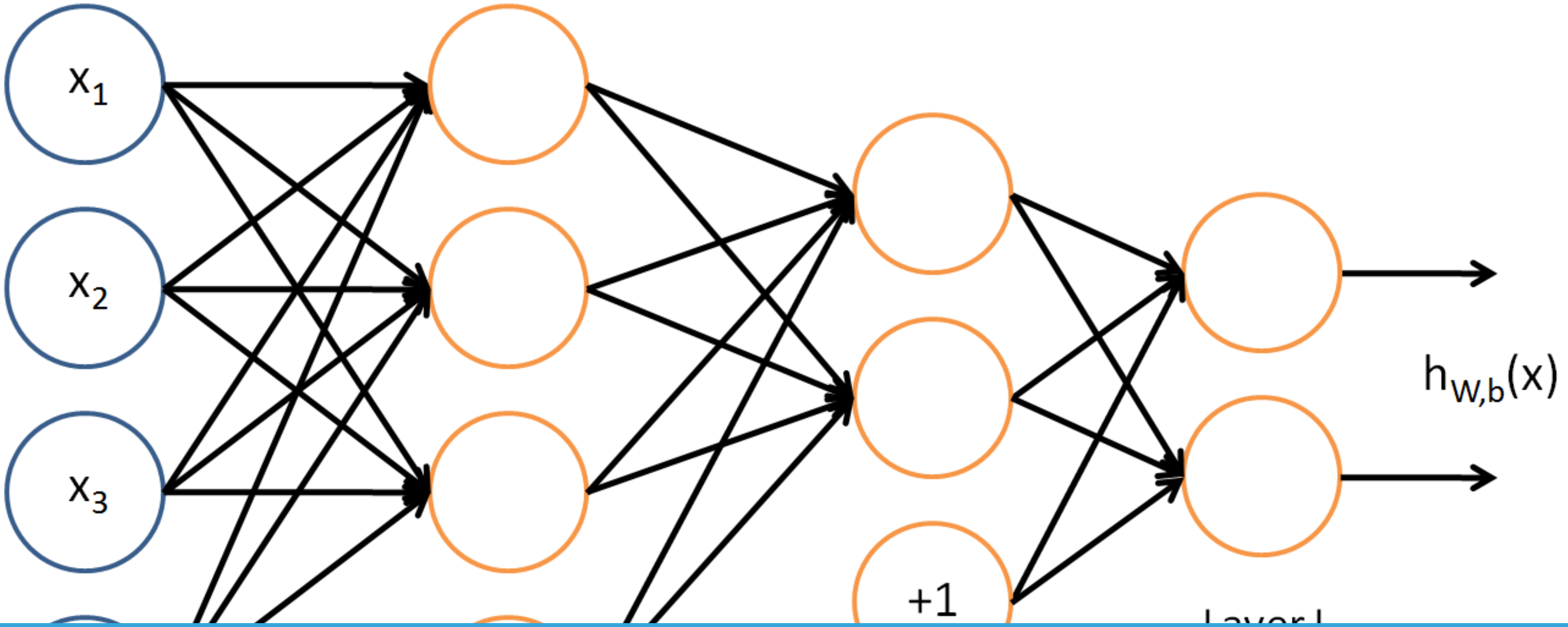
$$w_{ji} = w_{ji} - \eta \nabla w_{ji}; w_{kj} = w_{kj} - \eta \nabla w_{kj}$$

Until stopping criteria is met

NEURAL NETWORK COMPONENTS

- **Model space**
  - Set of weights **w** and b's (can combine them into a new weight vector)
- **Search algorithm**
  - Iterative refinement of weights, using backpropagation
- **Score function**
  - Minimize error (typically squared error)

FROM NEURAL NETWORKS TO DEEP LEARNING



ADDING LAYERS IN NEURAL NETWORKS GIVES THE MODEL MORE FLEXIBILITY —TRIED IN 1980S BUT DID NOT IMPROVE PERFORMANCE SUBSTANTIALLY BECAUSE BACK PROP ESTIMATION WOULD GET STUCK IN (SUBPAR) LOCAL MAXIMA

DEEP LEARNING

- Breakthrough in learning parameters for neural networks with a large number of hidden layers
- Guest lecture: Professor Yexiang Xue (March 21)