MULTI-LAYER PERCEPTRON

INT301 Bio-computation, Week 5, 2021

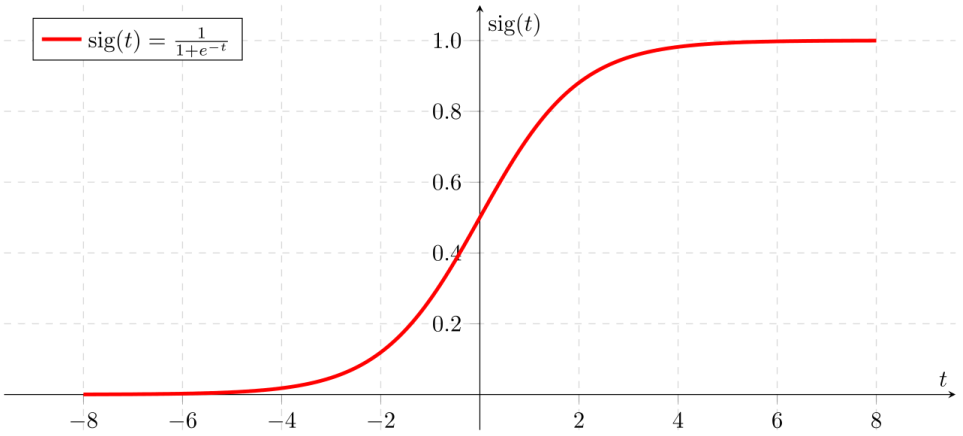
Multilayer Perceptrons

- The **multilayer perceptron** (**MLP**) is a hierarchical structure of several perceptrons, which overcomes the shortcomings of the single-layer networks.
- The MLP neural network is able to learn nonlinear function mappings.
  - learning a rich variety of nonlinear decision surfaces.
- Nonlinear functions can be represented by MLPs with units that use nonlinear activation functions.
  - Multiple layers of cascaded linear units still produce only linear mappings!

Differentiable Activation Functions

- Training algorithms for MLP require *differentiable, continuous nonlinear activation functions*.
- Such a function is the *sigmoid function*:

$$o = \sigma(s) = \frac{1}{1+e^{-s}}$$



where s is the sum:

$$s = \sum_{i=0}^{d} w_i x_i$$

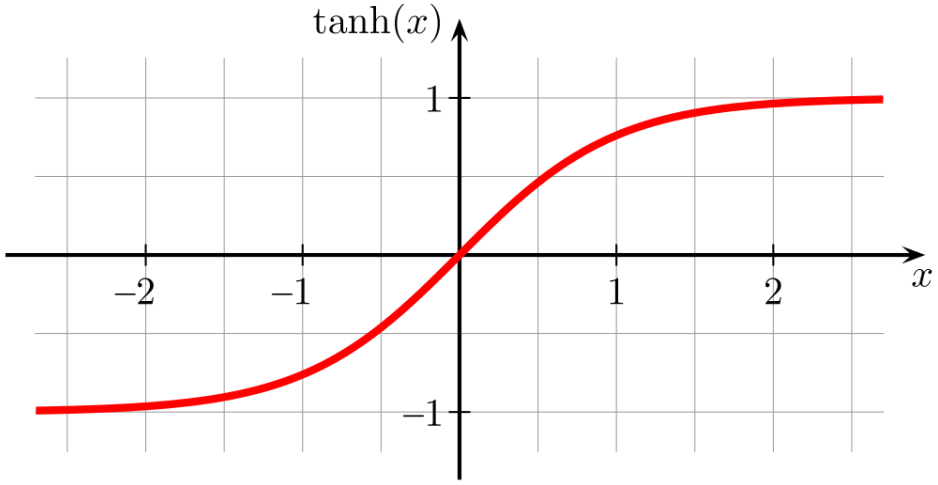
which is the products from the weights w_i and the inputs x_i.

You can use fplot('1/(1+exp(-x))',[-5 5])to plot sigmoid

Differentiable Activation Functions

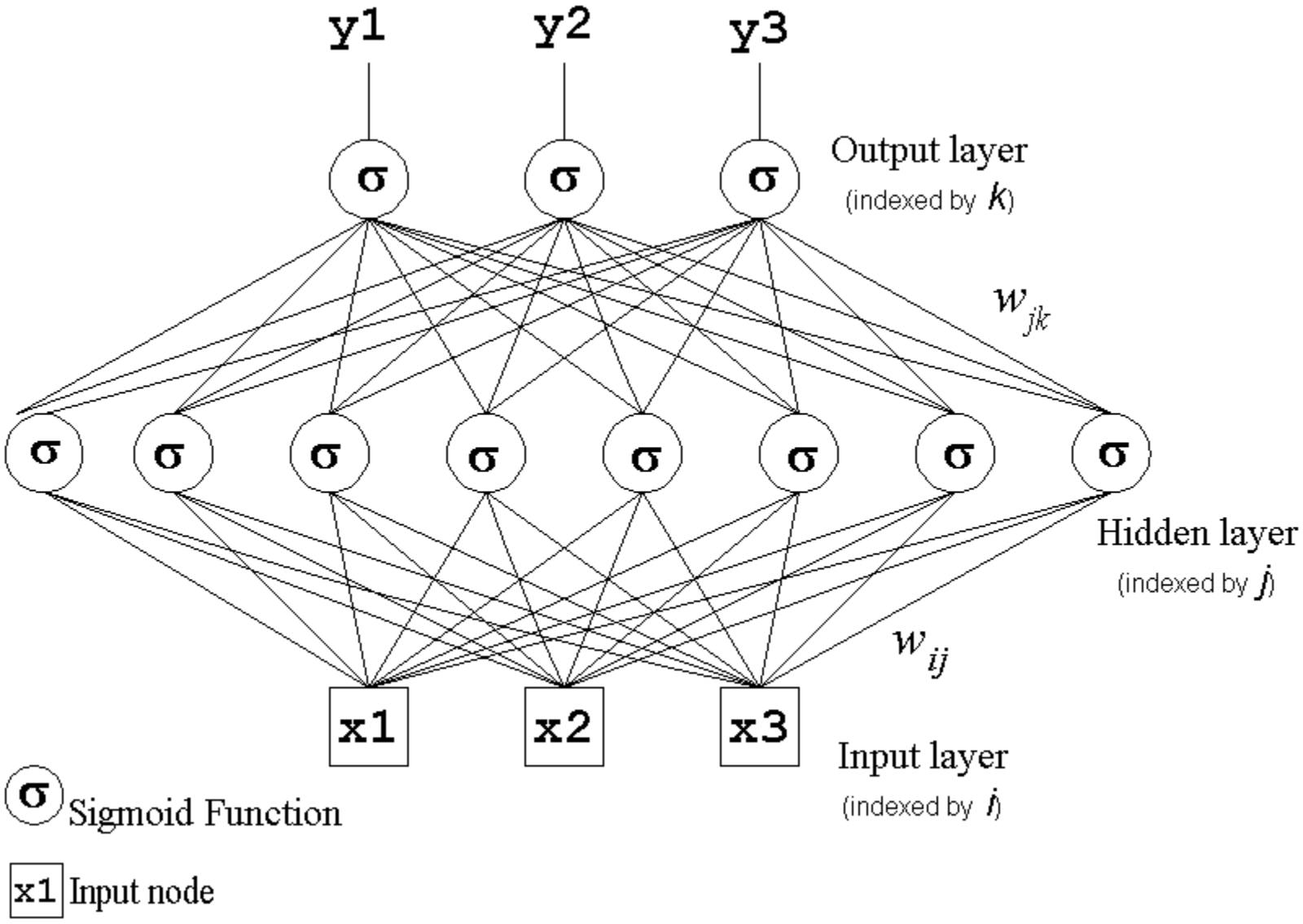
- Another nonlinear function often used in practice is the *hyperbolic tangent*:

$$o = \tanh(s) = \frac{e^{s} - e^{-s}}{e^{s} + e^{-s}}$$



- The mean of tanh is 0

Multilayer Network Structure



Multilayer Network Structure

- A two-layer neural network implements the function:

$$f(x) = \sigma\left(\sum_{j=1}^{J} w_{jk}\, \sigma\left(\sum_{i=1}^{I} w_{ij} x_i + w_{oj}\right) + w_{ok}\right)$$

Output from hidden layer

where: $\mathbf{x}$ is the input vector,

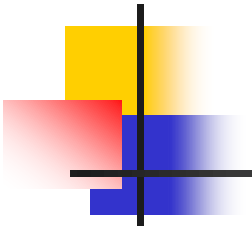
w_{0j} and w_{0k} are the bias terms,

w_{ij} are the weights connecting the input with the hidden nodes

w_{jk} are the weights connecting the hidden with output nodes

σ is the sigmoid activation function.

Multilayer Network Structure



- The hidden units enable the multilayer network to learn complex tasks by extracting ***progressively more meaningful information*** from the input examples.
- The MLP has a highly connected topology since every input is connected to all nodes in the first hidden layer, every unit in the hidden layers is connected to all nodes in the next layer, and so on.
- The input signals, initially these are the input examples, propagate through the neural network in a forward direction on a layer-by-layer basis, that is why they are often called *feedforward multilayer networks.*

Representation Power of MLP

- Properties concerning the representational power of MLP:
  - ***learning arbitrary functions***: any function can be learned with an arbitrary accuracy by a two-layer network;
  - ***learning continuous functions***: every bounded continuous function can be learned with a small error by a two-layer network (the number of hidden units depends on the function to be approximated);
  - ***learning Boolean functions***: every Boolean function can be learned exactly by a two-layer network although the number of hidden units grows exponentially with the input dimension.

Backpropagation Learning Algorithm

- MLP became applicable on practical tasks after the discovery of a supervised training algorithm, the ***error backpropagation learning algorithm***.

- The error backpropagation algorithm includes two passes through the network:
  - ***forward pass***, and
  - ***backward pass***.

Backpropagation Learning Algorithm

- During the backward pass the weights are adjusted in accordance with the ***error correction rule***. The actual network output is subtracted from the given output in the example and the weights are adjusted so as to make the network output close to the desired one.
- The backpropagation algorithm does gradient descent as it moves in direction opposite to the gradient of the error, that is in direction of the steepest decrease of the error.
- This is the direction of most rapid error decrease by varying all the weights simultaneously:

$$\nabla \mathrm{E}(\mathrm{w}) = [\frac{\partial \mathrm{E}}{\partial \mathrm{w}_0}, \frac{\partial \mathrm{E}}{\partial \mathrm{w}_1}, \ldots., \frac{\partial \mathrm{E}}{\partial \mathrm{w}_\mathrm{d}}]$$

Backpropagation Learning Algorithm

$$\nabla E(w) = [\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_d}]$$

- By gradient descent search, backpropagation training algorithm minimizes a cost function E (the mean square difference between the desired and actual net outputs).
- The network is trained initially selecting small random weights and then presenting all training data incrementally.
- Weights are adjusted after every trial using side information specifying the correct class until weights converge and the cost function is reduced to an acceptable value.

Backpropagation Training Algorithm

- ***Initialization***: Examples $\{(x_e, y_e)\}_{e=1}^{N}$, initial weights w_i set to small random values, learning rate η
- ***Repeat***
  - For each training example (x, y)

  Forward
  - *calculate the outputs* using the sigmoid function:

$$o_j = \sigma(s_j) = \frac{1}{1+e^{-s_j}}, s_j = \sum_{i=0}^{d} w_{ij} o_i$$

at the hidden units j

where $o_i = x_i$

$$o_k = \sigma(s_k) = \frac{1}{1+e^{-s_k}}, s_k = \sum_{i=0}^{d} w_{jk} o_j$$

at the output units k

Backpropagation Training Algorithm

Backward

- compute the benefit β_k at the node k in the output layer:

$$\beta_k = o_k(1-o_k)[y_k - o_k]$$

effects from the output nodes

- compute the changes for weights $j \rightarrow k$ on connections to nodes in the output layer:

$$\Delta w_{jk} = \eta \beta_k o_j$$

$$\Delta w_{0k} = \eta \beta_k$$

effects from the output of the neuron

- compute the benefit β_j for the hidden node j with the formula:

$$\beta_j = o_j(1-o_j)[\sum_k \beta_k w_{jk}]$$

effects from multiple nodes in the next layer

Backpropagation Training Algorithm

- compute the changes for the weights $i \rightarrow j$ on connections to nodes in the hidden layer:

$$\Delta w_{ij} = \eta \beta_j o_i$$

$$\Delta w_{0j} = \eta \beta_j$$

- *update the weights* by the computed changes:

$$w = w + \Delta w$$

until *termination condition is satisfied.*

On-line Training

Revision by example is called ***on-line (incremental) learning***.

- **Initialization:** Examples $\{(x_e, y_e)\}_{e=1}^{N}$, initial weights w_i set to small random values, learning rate η
- **Repeat**

  pick a training example (x, y)

  - forward propagate the example and calculate the outputs using the sigmoid function
  - backward propagate the error to calculate the benefits
  - update the weights by the computed changes:

    $$w = w + \Delta w$$

- **until** termination condition is satisfied.

Derivation of Backpropagation Algorithm

- The BP training algorithm for MLP is a generalized gradient descent rule, according to which with each training example every weight is updated as:

$$w = w + \Delta w$$

where: $\Delta w = -\eta \frac{\partial E_e}{\partial w}, \qquad E_e = \frac{1}{2}\sum_k (y_k - o_k)^2$

- The implementation of the generalized gradient descent rule requires to derive an expression for the computation of the derivatives $\partial E_e / \partial w$

$$\frac{\partial E_e}{\partial w} = \frac{\partial E_e}{\partial s} \cdot \frac{\partial s}{\partial w}$$

Derivation of Backpropagation Algorithm

- The first part $\partial E_e/\partial s$ reflects the change of the error as a function of the change in the network weighted input to the unit.
- The second part $\partial s/\partial w$ reflects the change in the network weighted input as a function of the change of particular weight w to that node.
- Since:

$$\frac{\partial s}{\partial w} = \frac{\partial(\sum_l w_l o_l)}{\partial w} = o$$

- The expression is reduced as follows:

$$\frac{\partial E_e}{\partial w} = \frac{\partial E_e}{\partial s} \cdot o$$

Derivation of Backpropagation Algorithm

- For weights $j \to k$ on connections to nodes in the output layer:

$$\frac{\partial E_e}{\partial w_{jk}} = \frac{\partial E_e}{\partial s_k} \cdot o_j$$

$$\frac{\partial E_e}{\partial s_k} = \frac{\partial E_e}{\partial o_k} \cdot \frac{\partial o_k}{\partial s_k}$$

$$\frac{\partial E_e}{\partial o_k} = \frac{\partial (\frac{1}{2} \sum_k (y_l - o_l)^2)}{\partial o_k} = \frac{\partial (\frac{1}{2} (y_k - o_k)^2)}{\partial o_k}$$

$$= \frac{1}{2} \cdot 2 \cdot (y_k - o_k) \frac{\partial (y_k - o_k)}{\partial o_k}$$

$$= -(y_k - o_k)$$

$$\frac{\partial o_k}{\partial s_k} = \frac{\partial \sigma(s_k)}{\partial s_k} = o_k (1 - o_k)$$

Derivation of Backpropagation Algorithm

- Therefore:

$$\frac{\partial E_e}{\partial s_k} = -(y_k - o_k)o_k(1-o_k) \qquad \frac{\partial E_e}{\partial w_{jk}} = \frac{\partial E_e}{\partial s_k} \cdot o_j$$

- Then we substitute:

$$\Delta w_{jk} = -\frac{\partial E_e}{\partial w_{jk}} = \eta \beta_k o_j \qquad \beta_k = (y_k - o_k)o_k(1-o_k)$$

- The gradient descent rule in previous lecture:

$$\Delta w_i = \Delta w_i + \eta(y_e - o_e)\sigma(s)(1-\sigma(s))x_{ie}$$

Derivation of Backpropagation Algorithm

- For weights $i \to j$ on connections to nodes in the hidden layer

$$\frac{\partial E_e}{\partial w_{ij}} = \frac{\partial E_e}{\partial s_j} \cdot o_i$$

- In this case the error depends on the errors committed by all output units:

$$\frac{\partial E_e}{\partial s_j} = \sum_k \frac{\partial E_e}{\partial s_k} \cdot \frac{\partial s_k}{\partial s_j} = \sum_k -\beta_k \cdot \frac{\partial s_k}{\partial s_j}$$

$$= \sum_k -\beta_k \cdot \frac{\partial s_k}{\partial o_j} \cdot \frac{\partial o_j}{\partial s_j}$$

$$= \sum_k (-\beta_k) \cdot w_{jk} \cdot \frac{\partial o_j}{\partial s_j} = \sum_k (-\beta_k) \cdot w_{jk} \cdot o_j (1 - o_j)$$

$$\frac{\partial E_e}{\partial s_k} = -(y_k - o_k) o_k (1 - o_k)$$

$$o_k = \sigma(s_k) = \frac{1}{1 + e^{-s_k}}, s_k = \sum_{i=0}^{d} w_{jk} o_j$$

$$o_j = \sigma(s_j) = \frac{1}{1 + e^{-s_j}}, s_j = \sum_{i=0}^{d} w_{ij} o_i$$

Derivation of Backpropagation Algorithm

- For the hidden units:

$$\Delta w_{ij} = \eta \beta_j o_i$$

$$\Delta w_{0j} = \eta \beta_j$$

$$\beta_j = -\frac{\partial E_e}{\partial s_j} = o_j(1-o_j)[\sum_k \beta_k w_{jk}]$$

Note: This analysis was made for a single training pattern, but it can be generalized so that:

$$\frac{\partial E_{total}}{\partial w_{ij}} = \sum_e \frac{\partial E_e}{\partial w_{ij}}$$

Thus, we just need to sum out weight changes over the examples.

Batch Backpropagation Algorithms

Revision by Epoch

- From mathematical point of view the error derivatives should be computed after each epoch, *i.e.,* after all examples in the training set have been processed.
  - This means that the error derivative is taken to be the sum of the error derivatives for all examples.
  - While this revision by epoch may have stronger theoretical motivation, revision by a particular example may yield better results and is more commonly used.
- Revision by epoch is called ***Batch Learning***.

Batch version of the backpropagation algorithm

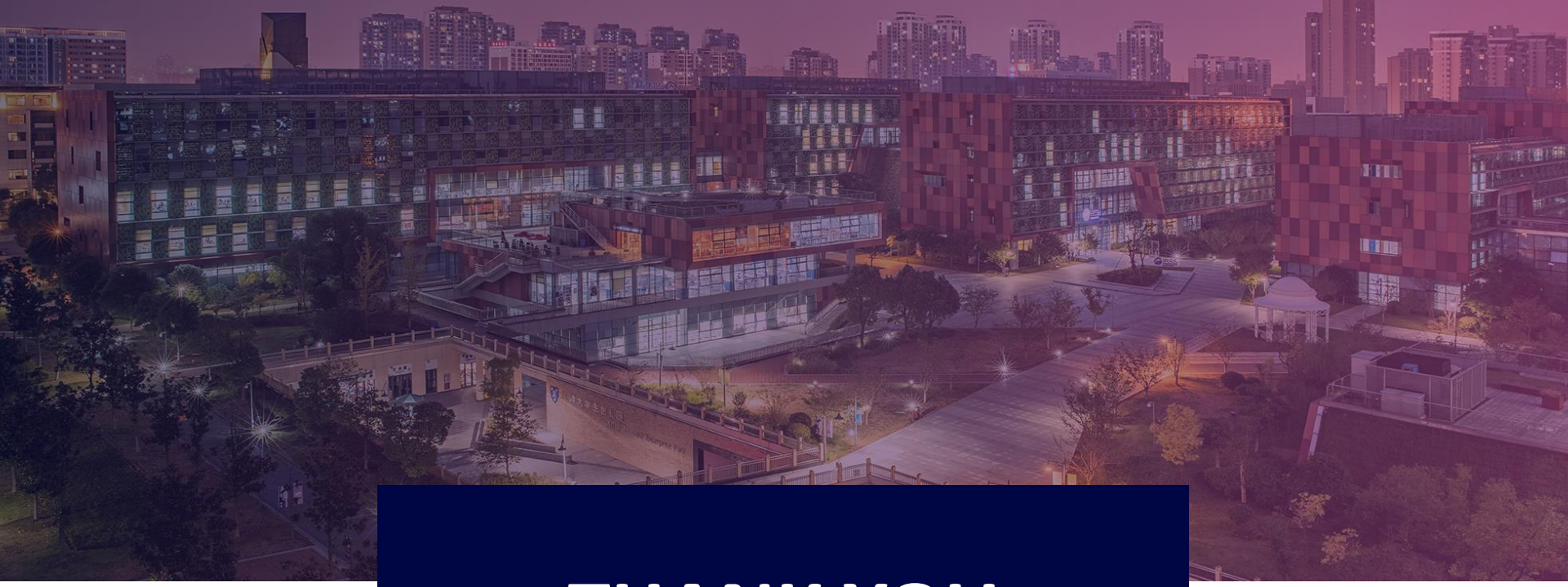
- **Initialization:**

  Examples $\{(x_e, y_e)\}_{e=1}^{N}$, *initial weights* w_i *set to small random values, learning rate* η

- **Repeat**
  - *for each training example (x, y)*
  - *forward propagate the example and calculate the outputs using the sigmoid function*
  - *backward propagate the error to calculate the benefits*
  - *after processing all examples update the weights by the computed changes:*

    $$w = w + \Delta w$$

- **until** *termination condition is satisfied.*



THANK YOU

VISIT US

WWW.XJTLU.EDU.CN

FOLLOW US

@XJTLU

Xi'an Jiaotong-Liverpool University
西交利物浦大學