Multilayer Perceptrons (MLP)

多层感知器（MLP） 是几个感知器的分层结构，它克服了单层网络的缺点。

MLP 神经网络能够学习非线性函数映射。

- 学习各种非线性决策表面（nonlinear decision surfaces）

Nonlinear functions can be represented by MLPs with
units that use nonlinear activation functions. （多层级 联系的 linear unit 仍然只生成线性映射）

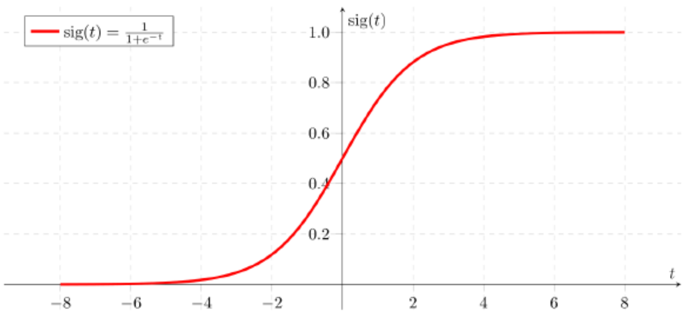
Differentiable Activation Functions

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

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

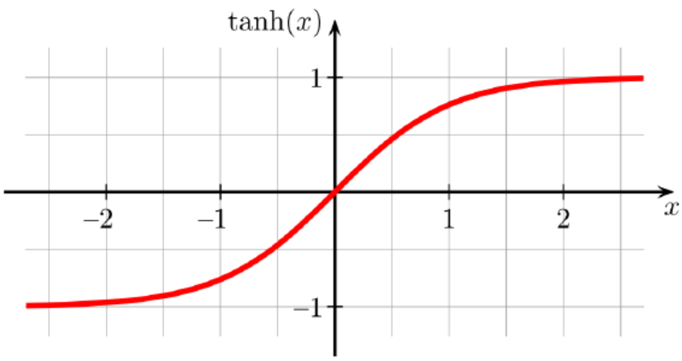
where *s* is the sum:

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



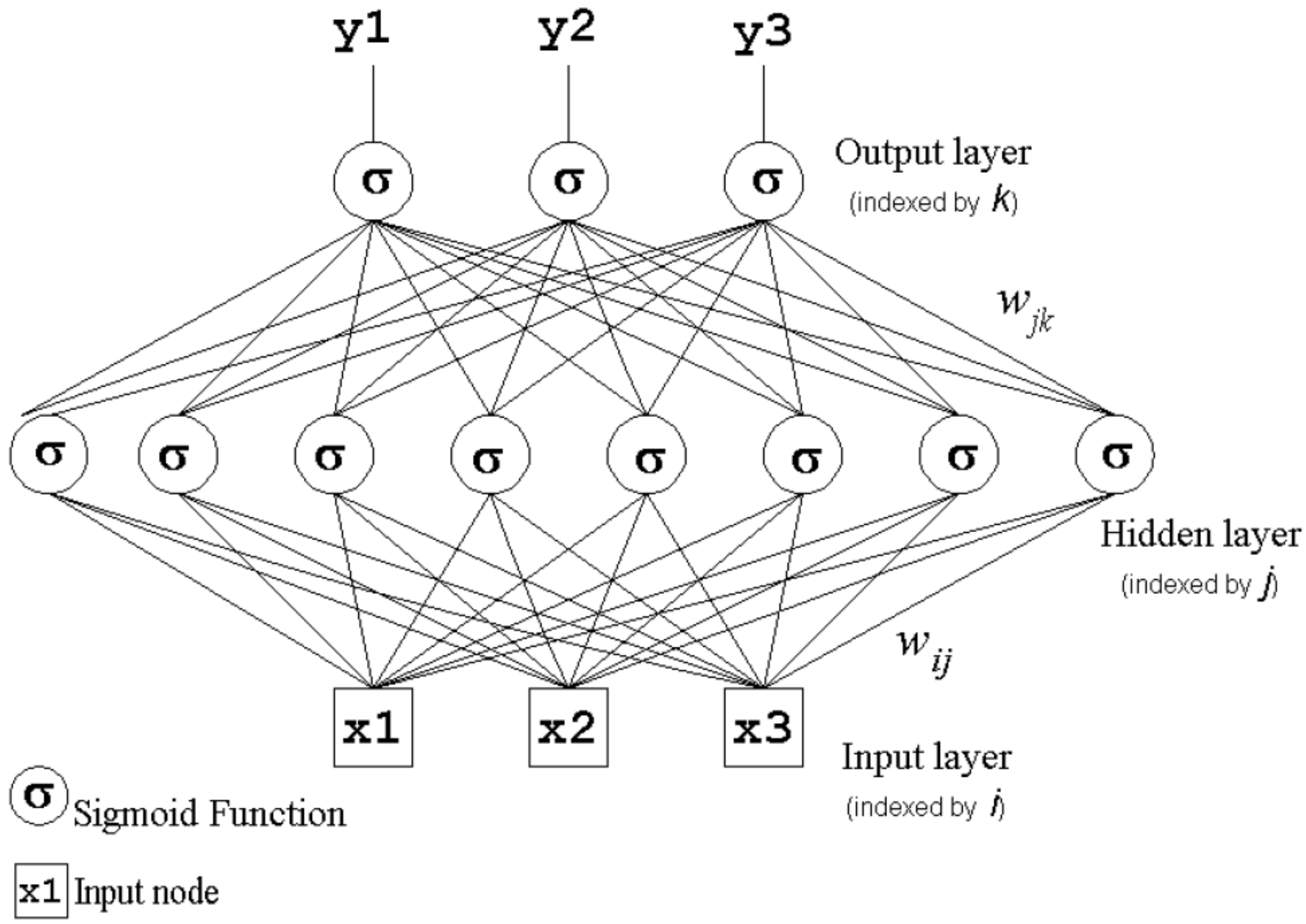
- Another nonlinear function often used in practice is the *hyperbolic tangent*:

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



- The mean of tanh is 0

Multilayer Network Structure



A two-layer neural network implements the function:

$$f(x) = \sigma\left(\sum_{j=1}^{J} w_{jk} \underbrace{\boxed{\sigma\left(\sum_{i=1}^{I} w_{ij} x_i + w_{oj}\right)}}_{\text{Output from hidden layer}} + w_{ok}\right)$$

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.

input signals（输入信号），最初是 input example，在逐层向前通过神经网络传播，这就是为什么它们经常被称为 **feedforward multilayer network**。

MLP 能力的属性：

- learning arbitrary functions
- learning continuous functions
- learning Boolean functions

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**

- ***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

注：o_i 代表第 i 个 input unit 的值（即输入）。

w_{ij} 代表从 input unit i 到 hidden unit j 的权重，w_{jk} 代表从 hidden unit j 到 output unit k 的权重。

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

13

注：y_k 指 output unit k 的 label。

- 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 (incremental) learning

根据 example 进行修正 (revision) 叫 on-line leaning。

- **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.

注：这个和 Incremental Gradient Descent 类似，都是用单个 example 来更新权重。

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

注：E_e 是模型的 loss 函数。

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

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

注：这里 s 和 w 不用角标，因为可以对 w_{jk} 和 w_{ij} 两个权重求梯度。

对 w_{jk}：$\frac{\partial E_e}{\partial w_{jk}} = \frac{\partial E_e}{\partial o_k} \cdot \frac{\partial o_k}{\partial s_k} \cdot \frac{\partial s_k}{\partial w_{jk}}$。

对 w_{ij}：$\frac{\partial E_e}{\partial w_{ij}} = \frac{\partial E_e}{\partial o_k} \cdot \frac{\partial o_k}{\partial s_k} \cdot \frac{\partial s_k}{\partial o_j} \cdot \frac{\partial o_j}{\partial s_j} \cdot \frac{\partial s_j}{\partial w_{ij}}$。

- 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:

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

- For weights $j \rightarrow k$ on connections to nodes in the output layer: $$\frac{\partial E_e}{\partial w_{jk}} = \frac{\partial E_e}{\partial s_k} \cdot o_j$$

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

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

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

$$= -(y_k - o_k)$$

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

- 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:

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

注：上面这段是对 w_{jk} 的。

- 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:

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

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

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

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

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

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

- For the hidden units:

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

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

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

在每个 epoch 后计算 loss，这种叫做 batch learning。

- **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.*