CS 1675 Machine Learning
Lecture 14

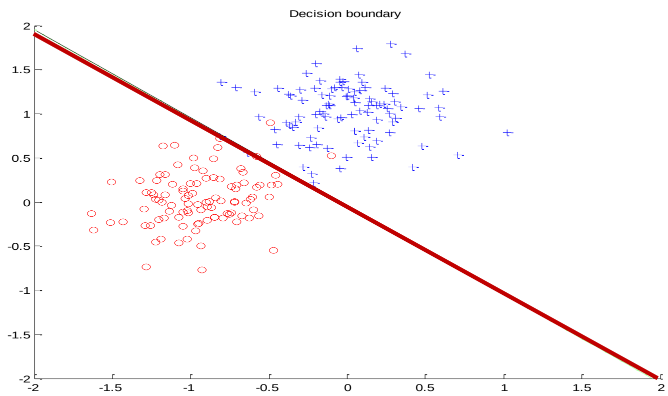
Multilayer neural networks

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Classification with the linear model

The majority of the models covered so far are linear

Example: 2 classes (blue and red points)



Modeling nonlinearities

- **Feature (basis) functions** to model **nonlinearities**

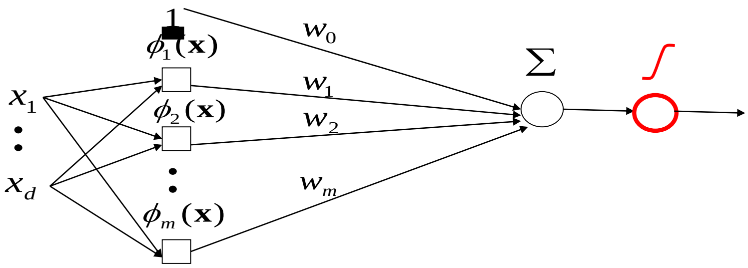
Linear regression

$$f(\mathbf{x}) = w_0 + \sum_{j=1}^{m} w_j \phi_j(\mathbf{x})$$

Logistic regression

$$f(\mathbf{x}) = g(w_0 + \sum_{j=1}^{m} w_j \phi_j(\mathbf{x}))$$

$\phi_j(\mathbf{x})$ - an arbitrary function of **x**



Modeling nonlinearities

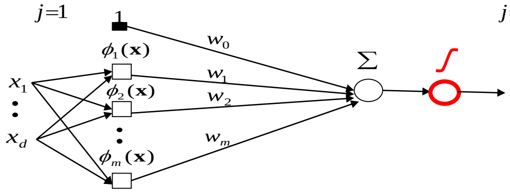
Feature (basis) functions model **nonlinearities**

Linear regression

$$f(\mathbf{x}) = w_0 + \sum_{j=1}^{m} w_j \phi_j(\mathbf{x})$$

Logistic regression

$$f(\mathbf{x}) = g(w_0 + \sum_{j=1}^{m} w_j \phi_j(\mathbf{x}))$$



Advantage:

- The same problem as learning of the weights of linear units

Limitations/problems:

- How to define the right set of basis functions
- Many basis functions → many weights to learn

Modeling nonlinearities

Support vector machines model nonlinearities via:

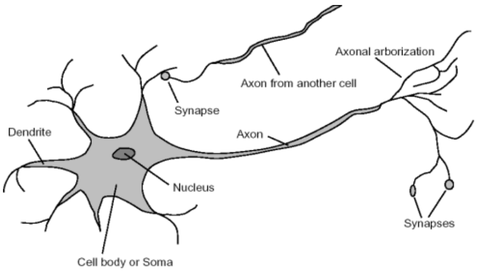
- **feature expansion**
- **Folded in efficient kernels**

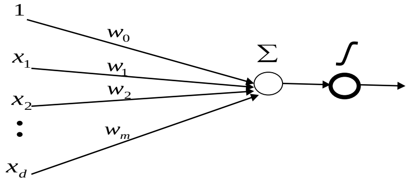
Advantage:

- The learning problem is similar to the problem of learning weights of a linear model
- Efficient kernels reduce the computational complexity
- **Problem:**
- How to define the right the kernels

Multi-layered neural networks

- An alternative way to model **nonlinearities for regression /classification problems**
- **Idea:** Cascade several simple nonlinear models (e.g. logistic units) **to approximate nonlinear functions** for regression /classification. Learn/adapt these simple models.
- **Motivation:** neuron connections



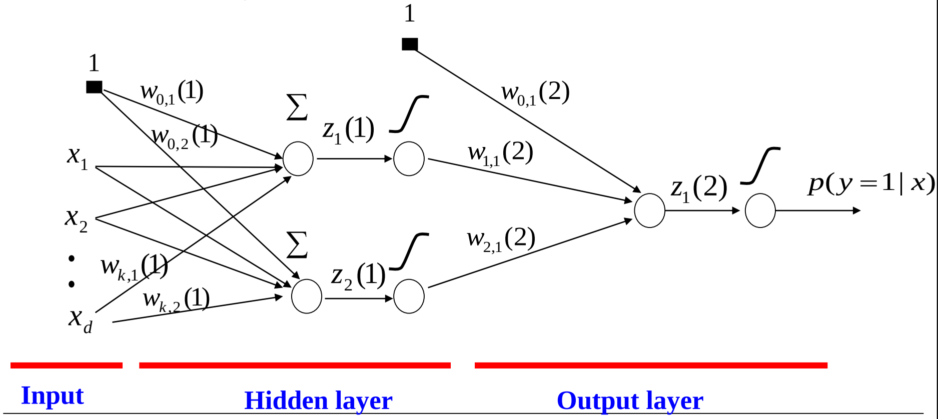


Multilayer neural network

Also called a **multilayer perceptron (MLP)**

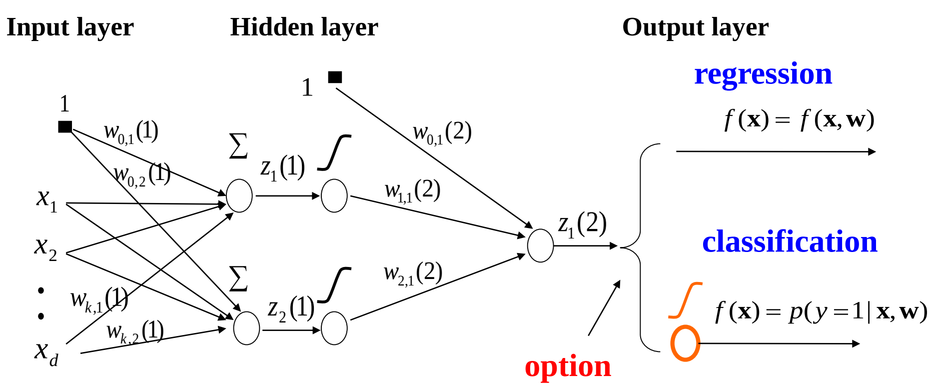
Cascades multiple **non-linear (e.g. logistic regression)** units

Example: (2 layer) classifier with non-linear decision boundaries



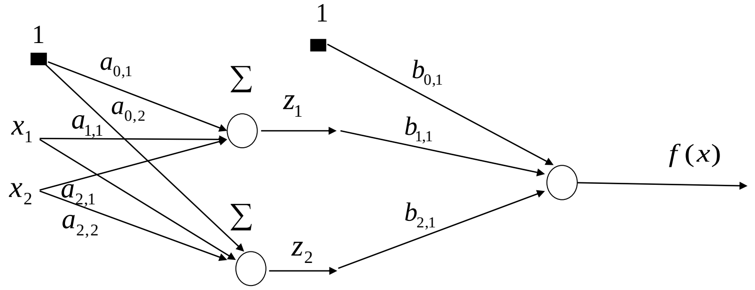
Multilayer neural network

- Models **non-linearity through nonlinear switching units**
- Can be applied to both **regression and binary classification problems**



Why we need nonlinearities? Why not multiple linear units

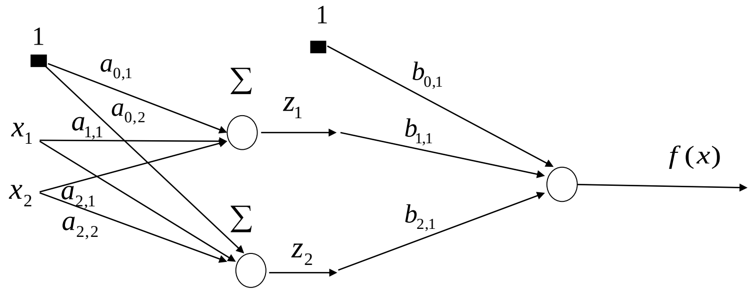
Cascading of multiple linear units is equivalent to one linear unit



$$f(\mathbf{x}) = b_{0,1} + b_{1,1}z_1 + b_{2,1}z_2$$

Why we need nonlinearities? Why not multiple linear units

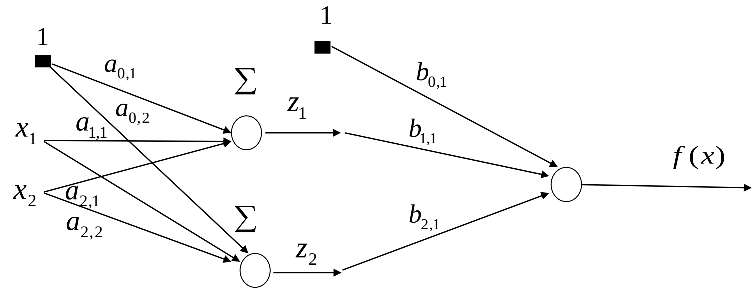
Cascading of multiple linear units is equivalent to one linear unit



$$f(\mathbf{x}) = b_{0,1} + b_{1,1}z_1 + b_{2,1}z_2$$
$$= b_{0,1} + b_{1,1}(a_{0,1} + a_{1,1}x_1 + a_{2,1}x_2) + b_{2,1}(a_{0,2} + a_{2,1}x_1 + a_{2,2}x_2)$$

Why we need nonlinearities? Why not multiple linear units

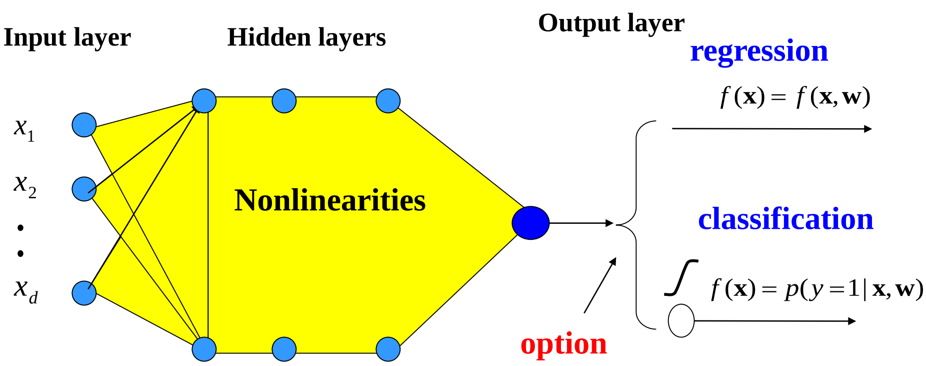
Cascading of multiple linear units is equivalent to one linear unit



$$\begin{aligned} f(\mathbf{x}) &= b_{0,1} + b_{1,1} z_1 + b_{2,1} z_2 \\ &= b_{0,1} + b_{1,1}(a_{0,1} + a_{1,1} x_1 + a_{2,1} x_2) + b_{2,1}(a_{0,2} + a_{2,1} x_1 + a_{2,2} x_2) \\ &= b_{0,1} + b_{1,1} a_{0,1} + b_{1,1} a_{1,1} x_1 + b_{1,1} a_{2,1} x_2 + b_{2,1} a_{0,2} + b_{2,1} a_{2,1} x_1 + b_{2,1} a_{2,2} x_2 \\ &= b_{0,1} + b_{1,1} a_{0,1} + b_{2,1} a_{0,2} + (b_{1,1} a_{1,1} + b_{2,1} a_{2,1}) x_1 + (b_{1,1} a_{2,1} + b_{2,1} a_{2,2}) x_2 \\ &= c + d_1 x_1 + d_2 x_2 \end{aligned}$$

Multilayer neural network

- **Non-linearities are modeled using multiple hidden nonlinear units (organized in layers)**
- The output layer determines whether it is a **regression or a binary classification problem**

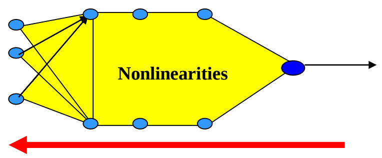


Learning with MLP

- How to learn the parameters of the neural network?
- **Gradient descent algorithm**
  - Weight updates based on the error: $J(D, \mathbf{w})$

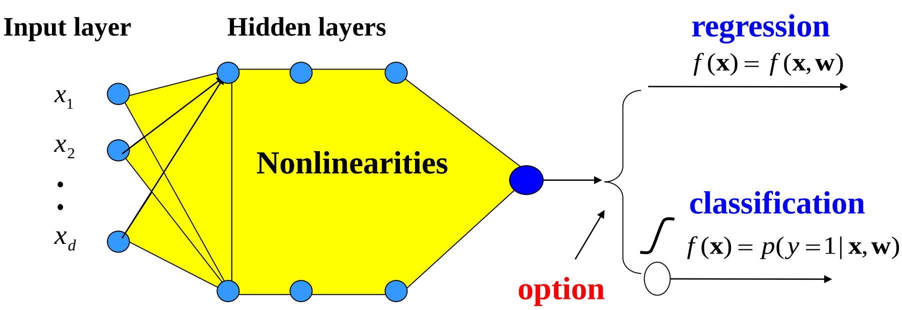
$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} J(D, \mathbf{w})$$

- We need to **compute gradients for weights in all units**
- **Can be computed in one backward sweep through the net !!!**



- The process is called **back-propagation**

Backpropagation: error function

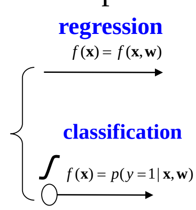


- **Error function**: $J(D, \mathbf{w})$ (online) error where D is a data point
  - **Regression**

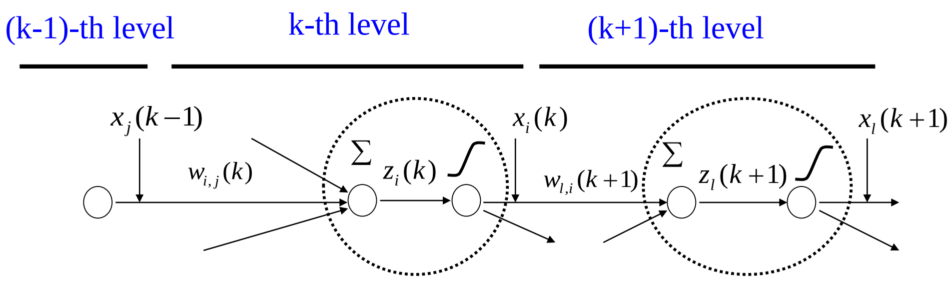
$$J(D, \mathbf{w}) = (y_u - f(\mathbf{x}_u))^2$$

  - **Classification**

$$J(D, \mathbf{w}) = -\log p(y_u \mid f(\mathbf{x}_u))$$



Backpropagation



$x_i(k)$ - output of the unit i on level k

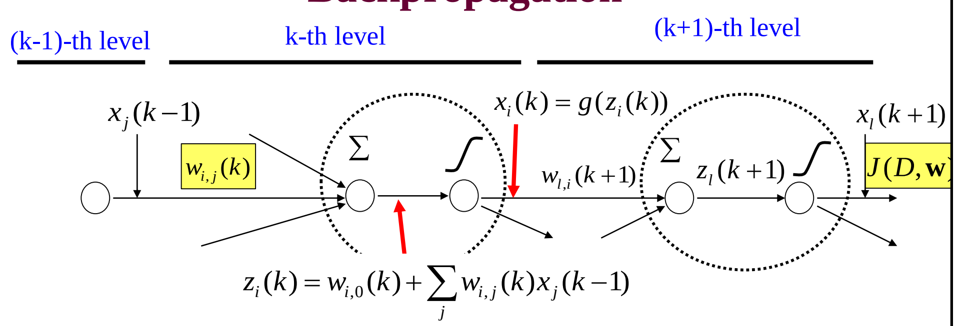
$x_i(k) = g(z_i(k))$

$z_i(k)$ - input to the sigmoid function on level k

$z_i(k) = w_{i,0}(k) + \sum_j w_{i,j}(k) x_j(k-1)$

$w_{i,j}(k)$ - weight between units j and i on levels (k-1) and k

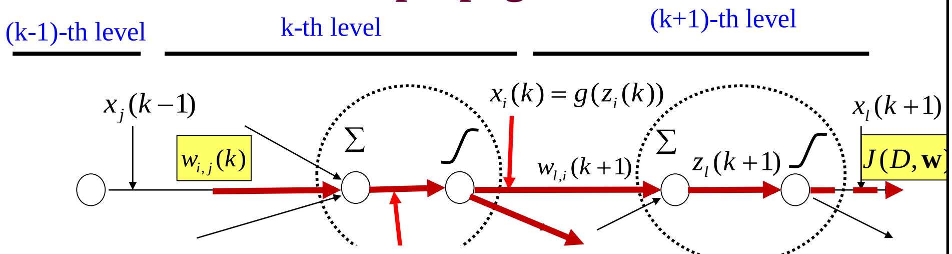
Backpropagation



- **Gradient descent:** $w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \frac{\partial}{\partial w_{i,j}(k)} J(D,\mathbf{w})$

$$\frac{\partial}{\partial w_{i,j}(k)} J(D,\mathbf{w}) = ?$$

Backpropagation



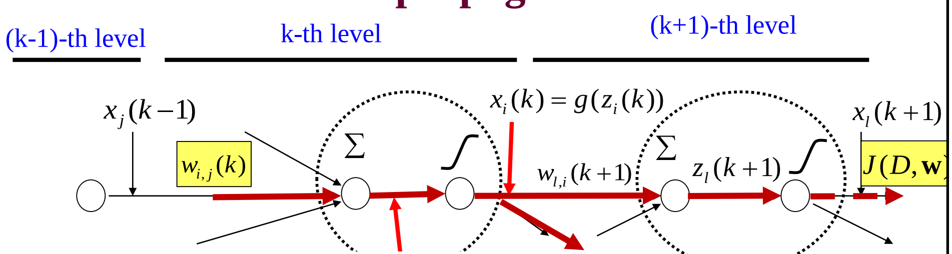
$$z_i(k) = w_{i,0}(k) + \sum_j w_{i,j}(k) x_j(k-1)$$

- **Gradient descent:** $w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \frac{\partial}{\partial w_{i,j}(k)} J(D,\mathbf{w})$

$$\frac{\partial}{\partial w_{i,j}(k)} J(D,\mathbf{w}) = ?$$

$$\frac{\partial f(g(u))}{\partial u} = \frac{\partial f(g(u))}{\partial g(u)} \frac{\partial g(u)}{\partial u}$$

Backpropagation



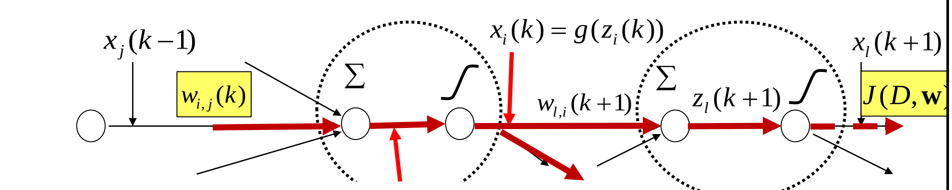
$$z_i(k) = w_{i,0}(k) + \sum_j w_{i,j}(k) x_j(k-1)$$

- **Gradient descent:** $w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \frac{\partial}{\partial w_{i,j}(k)} J(D,\mathbf{w})$

$$\frac{\partial}{\partial w_{i,j}(k)} J(D,\mathbf{w}) = \frac{\partial J(D,\mathbf{w})}{\partial z_i(k)} \frac{\partial z_i(k)}{\partial w_{i,j}(k)}$$

$$\frac{\partial f(g(u))}{\partial u} = \frac{\partial f(g(u))}{\partial g(u)} \frac{\partial g(u)}{\partial u}$$

Backpropagation



$$z_i(k) = w_{i,0}(k) + \sum_j w_{i,j}(k) x_j(k-1)$$

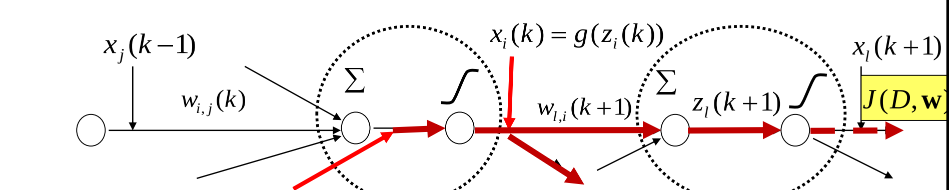
- **Gradient descent:** $w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \frac{\partial}{\partial w_{i,j}(k)} J(D,\mathbf{w})$

$$\frac{\partial}{\partial w_{i,j}(k)} J(D,\mathbf{w}) = \frac{\partial J(D,\mathbf{w})}{\partial z_i(k)} \frac{\partial z_i(k)}{\partial w_{i,j}(k)} = \delta_i(k) x_j(k-1)$$

$$\delta_i(k) = \frac{\partial}{\partial z_i(k)} J(D,\mathbf{w}) \qquad x_j(k-1)$$

$$\frac{\partial f(g(u))}{\partial u} = \frac{\partial f(g(u))}{\partial g(u)} \frac{\partial g(u)}{\partial u}$$

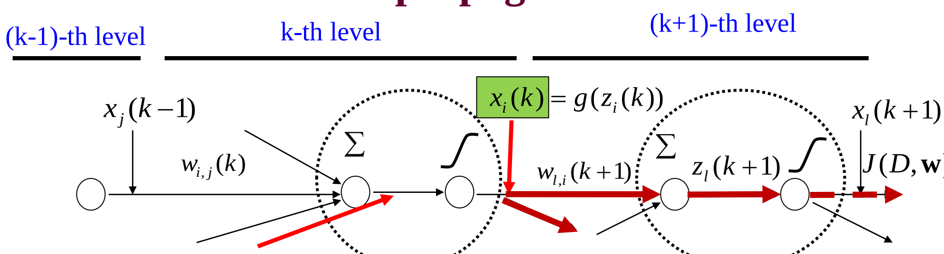
Backpropagation



$$z_i(k) = w_{i,0}(k) + \sum_j w_{i,j}(k) x_j(k-1)$$

- **Derivation:** $\delta_i(k) = \frac{\partial}{\partial z_i(k)} J(D,\mathbf{w}) = \frac{\partial}{\partial x_i(k)} J(D,\mathbf{w}) * \frac{\partial x_i(k)}{\partial z_i(k)}$

Backpropagation

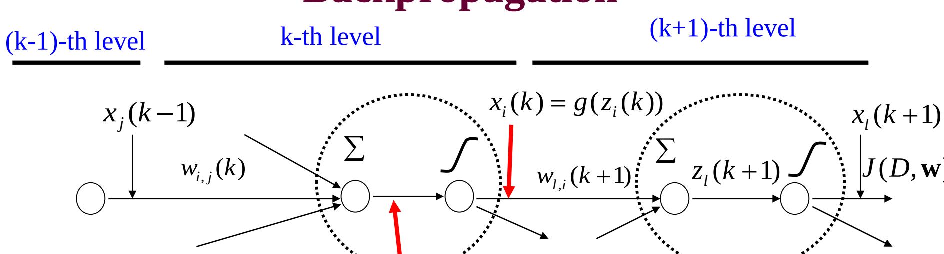


$$z_i(k) = w_{i,0}(k) + \sum_j w_{i,j}(k) x_j(k-1)$$

- **Derivation:** $\delta_i(k) = \frac{\partial}{\partial z_i(k)} J(D,\mathbf{w}) = \frac{\partial}{\partial x_i(k)} J(D,\mathbf{w}) * \frac{\partial x_i(k)}{\partial z_i(k)}$

$$\frac{\partial}{\partial x_i(k)} J(D,\mathbf{w}) = \sum_l \underbrace{\frac{\partial}{\partial z_l(k+1)} J(D,\mathbf{w})}_{\delta_l(k+1)} * \underbrace{\frac{\partial z_l(k+1)}{\partial x_i(k)}}_{w_{l,i}(k+1)}$$

Backpropagation



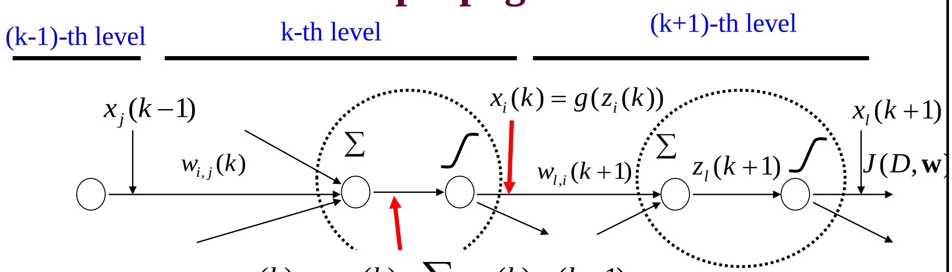
$$z_i(k) = w_{i,0}(k) + \sum_j w_{i,j}(k) x_j(k-1)$$

- **Derivation:** $\delta_i(k) = \frac{\partial}{\partial z_i(k)} J(D,\mathbf{w}) = \frac{\partial}{\partial x_i(k)} J(D,\mathbf{w}) * \frac{\partial x_i(k)}{\partial z_i(k)}$

$$\frac{\partial}{\partial x_i(k)} J(D,\mathbf{w}) = \sum_l \underbrace{\frac{\partial}{\partial z_l(k+1)} J(D,\mathbf{w})}_{\delta_l(k+1)} * \underbrace{\frac{\partial z_l(k+1)}{\partial x_i(k)}}_{w_{l,i}(k+1)}$$

$$\frac{\partial x_i(k)}{\partial z_i(k)} = x_i(k)(1 - x_i(k))$$

Backpropagation



$$z_i(k) = w_{i,0}(k) + \sum_j w_{i,j}(k) x_j(k-1)$$

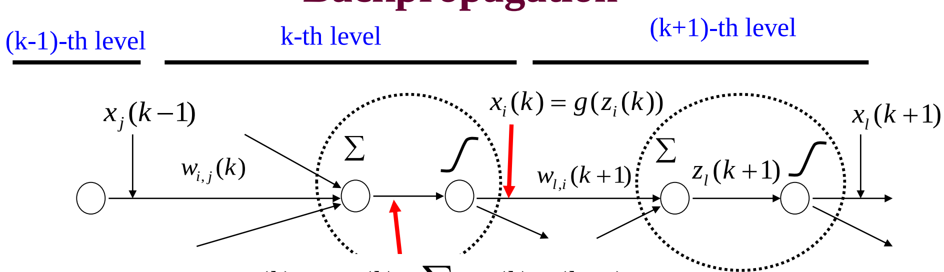
- **Derivation:**

$$\delta_i(k) = \frac{\partial}{\partial z_i(k)} J(D,\mathbf{w}) = \frac{\partial}{\partial x_i(k)} J(D,\mathbf{w}) * \frac{\partial x_i(k)}{\partial z_i(k)}$$

$$\frac{\partial}{\partial x_i(k)} J(D,\mathbf{w}) = \sum_l \underbrace{\frac{\partial}{\partial z_l(k+1)} J(D,\mathbf{w})}_{\delta_l(k+1)} * \underbrace{\frac{\partial z_l(k+1)}{\partial x_i(k)}}_{w_{l,i}(k+1)} \qquad \frac{\partial x_i(k)}{\partial z_i(k)} = x_i(k)(1 - x_i(k))$$

$$\delta_i(k) = \left[\sum_l \delta_l(k+1) w_{l,i}(k+1) \right] x_i(k)(1 - x_i(k))$$

Backpropagation



$$z_i(k) = w_{i,0}(k) + \sum_j w_{i,j}(k) x_j(k-1)$$

- **Gradient:**

$$w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \left[\delta_i(k) x_j(k-1) \right]$$

$$\delta_i(k) = \left[\sum_l \delta_l(k+1) w_{l,i}(k+1) \right] x_i(k)(1 - x_i(k))$$

- **Last unit** (is the same as for the regular linear units),

  E.g. for regression:

$$\delta_i(K) = -(y_u - f(\mathbf{x}_u, \mathbf{w}))$$

Backpropagation

Update weight $w_{i,j}(k)$ using data D $\quad D = \{<\mathbf{x}, y>\}$

$$w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \frac{\partial}{\partial w_{i,j}(k)} J(D,\mathbf{w})$$

Let $\quad \delta_i(k) = \frac{\partial}{\partial z_i(k)} J(D,\mathbf{w})$

Then: $\quad \frac{\partial}{\partial w_{i,j}(k)} J(D,\mathbf{w}) = \frac{\partial J(D,\mathbf{w})}{\partial z_i(k)} \frac{\partial z_i(k)}{\partial w_{i,j}(k)} = \delta_i(k) x_j(k-1)$

S.t. $\delta_i(k)$ is computed from $x_i(k)$ and the next layer $\delta_l(k+1)$

$$\delta_i(k) = \left[\sum_l \delta_l(k+1) w_{l,i}(k+1)\right] x_i(k)(1 - x_i(k))$$

Last unit (is the same as for the regular linear units):

$$\delta_i(K) = -(y_u - f(\mathbf{x}_u, \mathbf{w}))$$

It is the same for the classification with the log-likelihood measure of fit and linear regression with least-squares error!!!

Learning with MLP

- **Online gradient descent algorithm**
  - Weight update:

$$w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \frac{\partial}{\partial w_{i,j}(k)} J_{\text{online}}(D_u, \mathbf{w})$$

$$\frac{\partial}{\partial w_{i,j}(k)} J_{online}(D_u, \mathbf{w}) = \frac{\partial J_{online}(D_u, \mathbf{w})}{\partial z_i(k)} \frac{\partial z_i(k)}{\partial w_{i,j}(k)} = \delta_i(k) x_j(k-1)$$

$$\boxed{w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \delta_i(k) x_j(k-1)}$$

$x_j(k-1)$ - j-th output of the (k-1) layer

$\delta_i(k)$ - derivative computed via backpropagation

α - a learning rate

Online gradient descent algorithm for MLP

Online-gradient-descent (*D*, *number of iterations*)

Initialize all weights $w_{i,j}(k)$

for *i*=1:1: *number of iterations*

do **select** a data point D_u=<$\mathbf{x}$,*y*> from *D*

set learning rate α

compute outputs $x_j(k)$ for each unit

compute derivatives $\delta_i(k)$ via **backpropagation**

update all weights (in parallel)

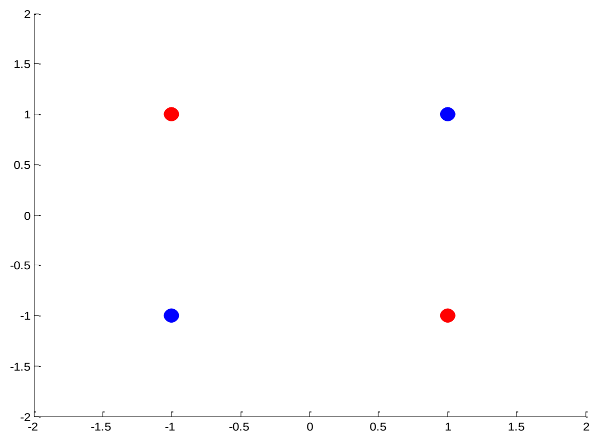
$$w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \delta_i(k) x_j(k-1)$$

end for

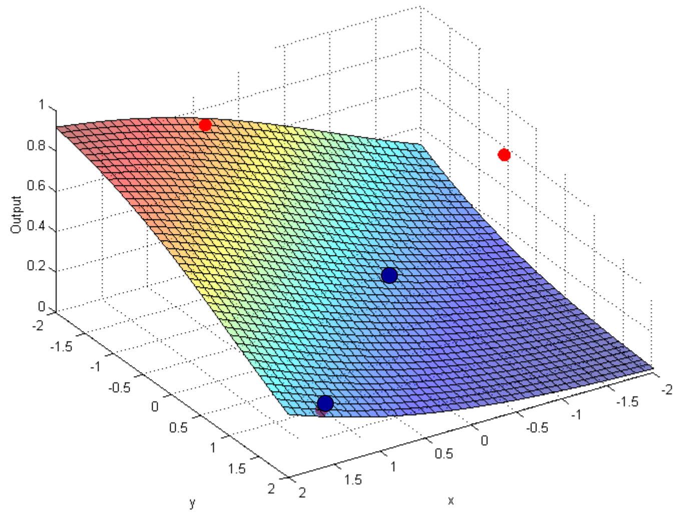
return weights **w**

Xor Example.

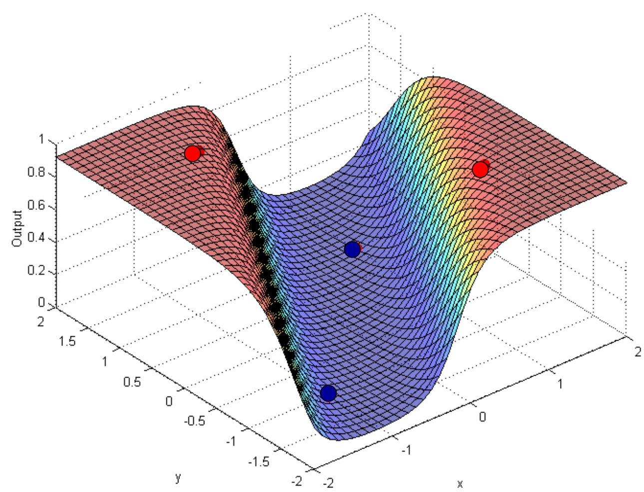
• linear decision boundary does not exist



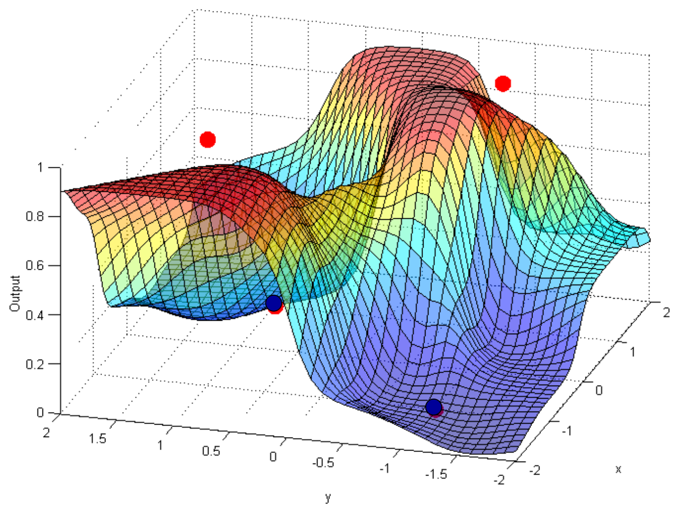
Xor example. Linear unit



Xor example.
Neural network with 2 hidden units



Xor example. Neural network with 10 hidden units



Neural networks

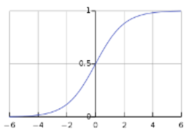
Activation (transfer) functions

- **Determine how inputs are transformed to output**

Possible choices of nonlinear transfer functions:

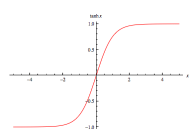
- **Logistic function**

$$f(z) = \frac{1}{1+e^{-z}} \qquad f(z)' = f(z)(1 - f(z))$$



- **Hyperbolic tangent**

$$f(z) = \tanh(z) = \frac{2}{1+e^{-2z}} - 1 \qquad f(z)' = 1 - f(z)^2$$



- **Rectified linear function (Relu)**

$$f(z) = \begin{matrix} 0 & z < 0 \\ z & z \geq 0 \end{matrix}$$