Artificial Neural Networks

Lecture 3: The Backpropagation Algorithm

Tudor Berariu
tudor.berariu@gmail.com

Faculty of Automatic Control and Computers
University Politehnica of Bucharest

Lecture : 21^{th} of October, 2015
Last Updated: 21^{th} of October, 2015

Today's Outline

1. Multi-Layer Perceptrons
2. Forward Computation
3. Error functions
4. Error Backpropagation
5. Learning the weights

Today's Outline

Let's add more layers

- Multi-Layer Perceptrons
  - bad name because perceptrons have discontinuous non-liniarities

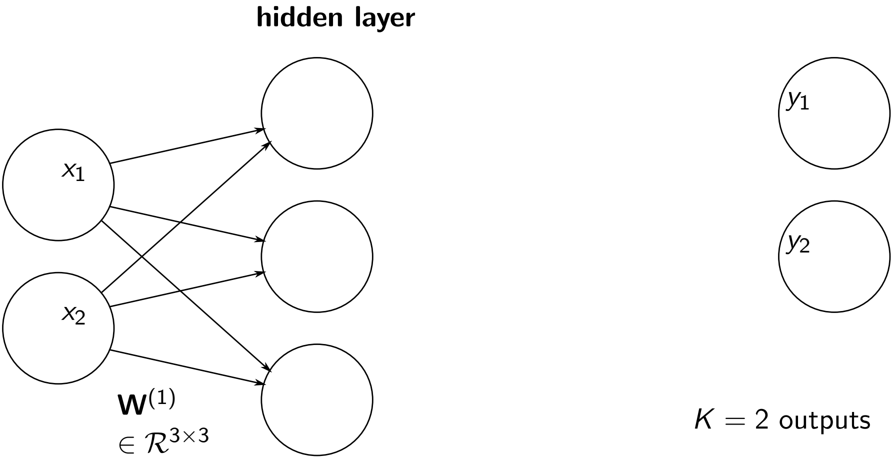
Network Elements

$D = 2$ inputs

$K = 2$ outputs

Network Elements

$D = 2$ inputs

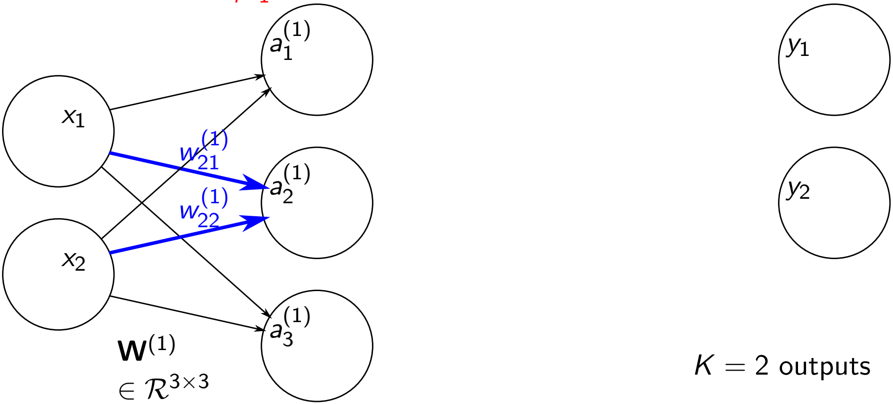


$K = 2$ outputs

Network Elements

$D = 2$ inputs

$$a_j^{(1)} = \sum_{i=1}^{D}(w_{ji}^{(1)} \cdot x_i) + w_{j0}^{(1)}$$

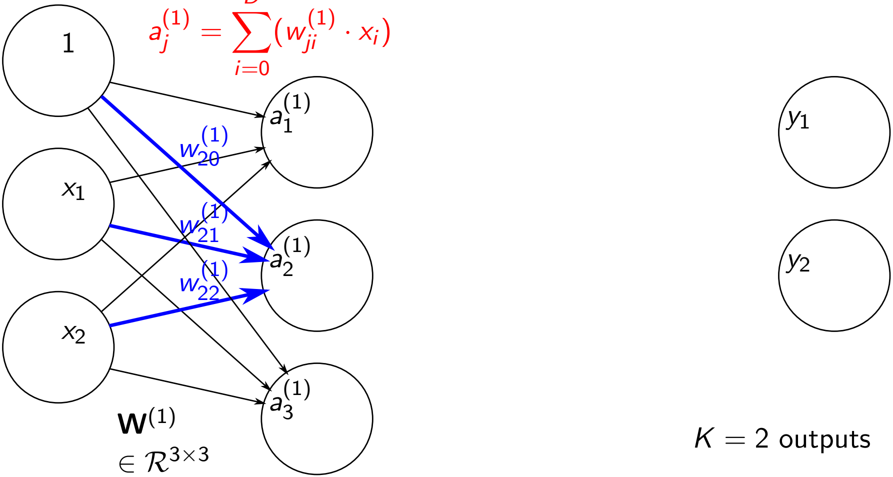


$K = 2$ outputs

Network Elements

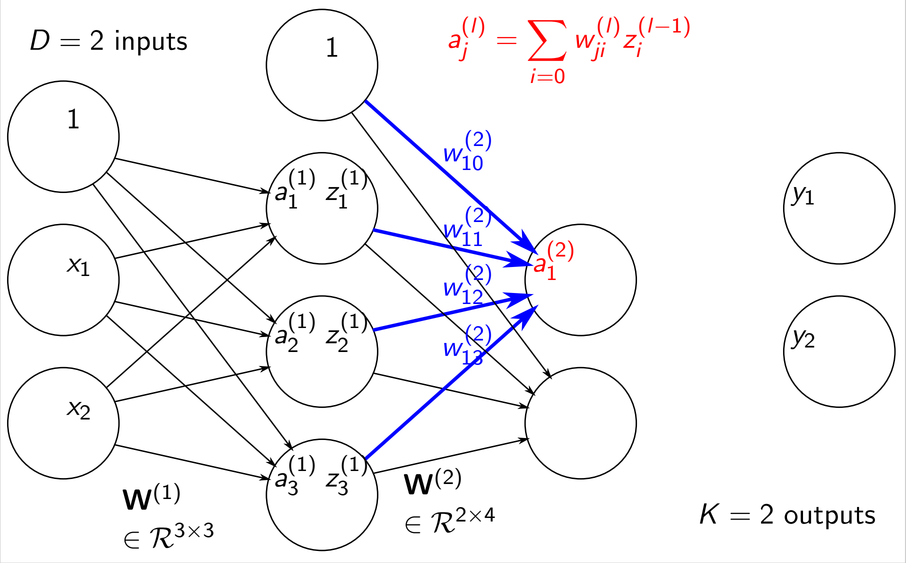
$D = 2$ inputs

$$a_j^{(1)} = \sum_{i=0}^{D} (w_{ji}^{(1)} \cdot x_i)$$

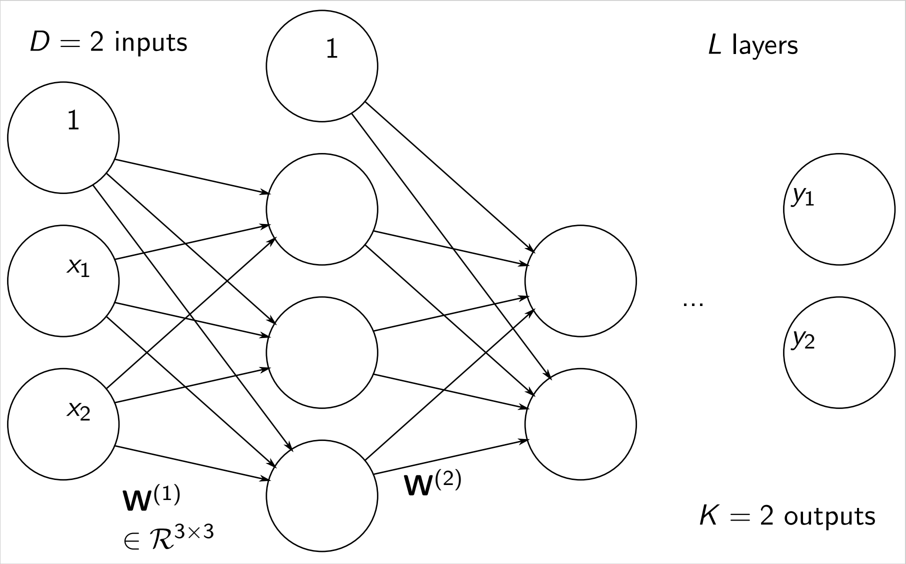


$K = 2$ outputs

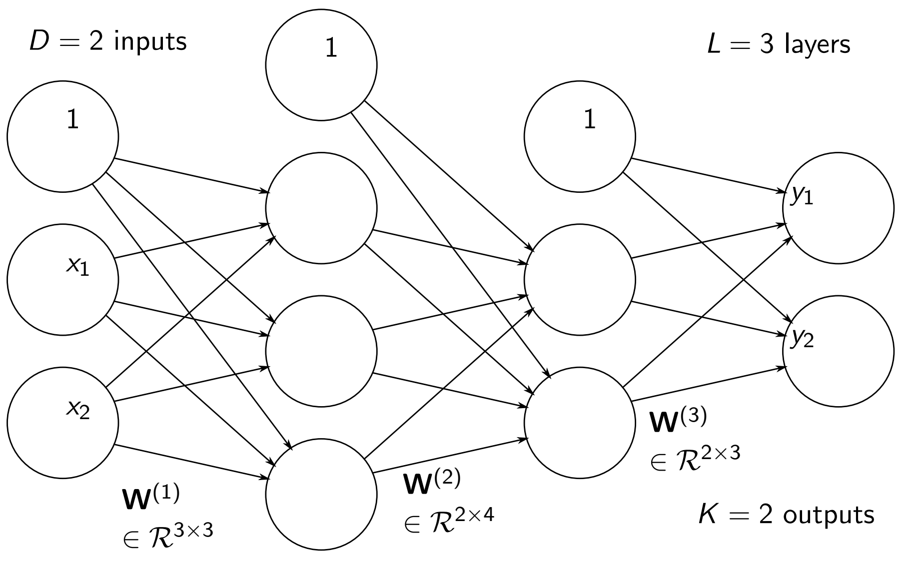
Network Elements



Network Elements



Network Elements



Notations

D - size of input space ($\mathbf{x}^{(n)} = (x_1^{(n)}, \ldots, x_D^{(n)})$;

K - size of output space ($\mathbf{t}^{(n)} = (t_1^{(n)}, \ldots, x_K^{(n)})$;

L number of layers (input layer is not included);

M_l number of units on layer l

- $M_0 = D$
- $M_L = K$

$\mathbf{W}^{(l)}$ weights that connect layer $l - 1$ and l

- $\mathbf{W}^{(l)} \in \mathcal{R}^{M_l \times (M_{l-1}+1)}$

The Network function

$$y_k\left(\mathbf{x}, \mathbf{W}\right) = f_L\left(\sum_{j=1}^{M_{L-1}} w_{kj}^{(L)} z_j^{(L-1)} + w_{k0}\right)$$

The Network function

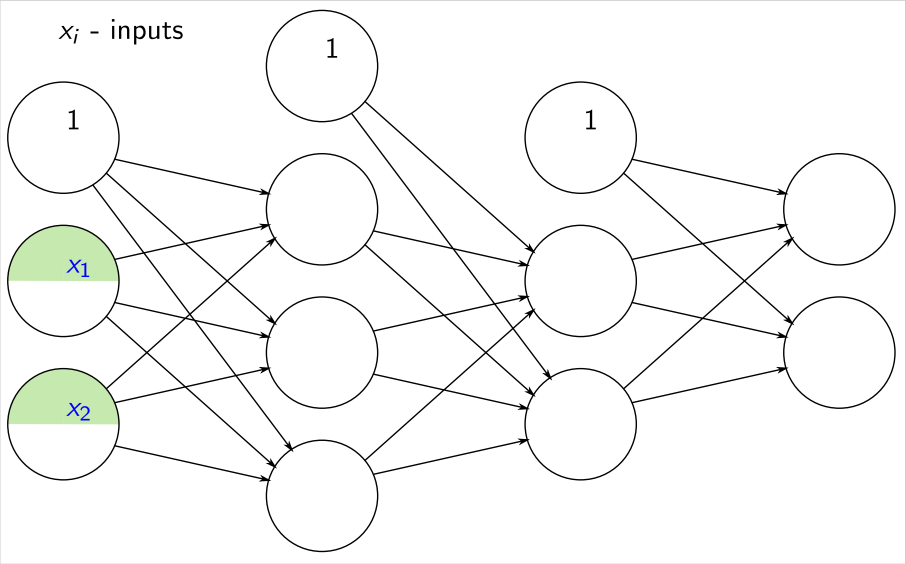
$$y_k(\mathbf{x}, \mathbf{W}) = f_L\left(\sum_{j=1}^{M_{L-1}} w_{kj}^{(L)} f_{L-1}\left(\sum_{i=1}^{M_{L-2}} w_{ji}^{(L-1)} z_i^{(L-2)} + w_{j0}\right) + w_{k0}\right)$$

The Network function

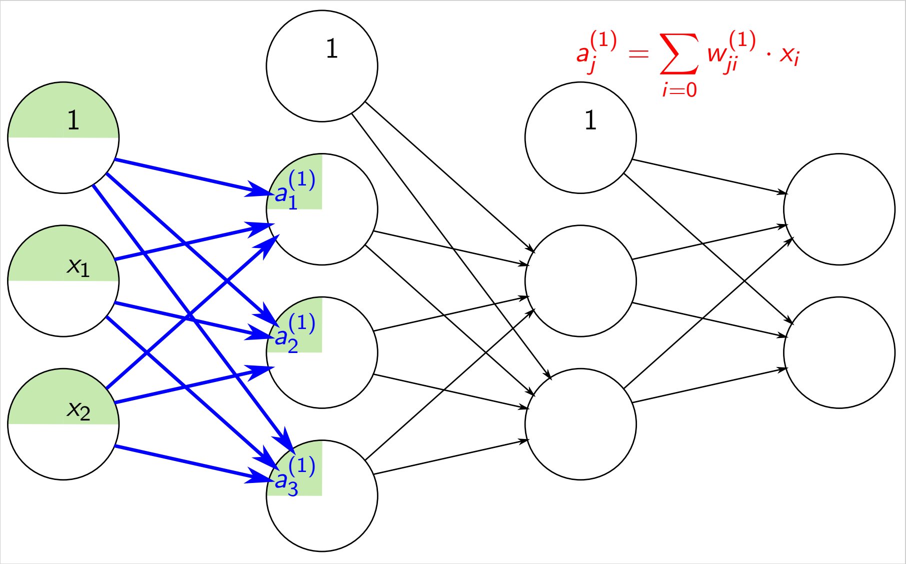
$$y_k(\mathbf{x}, \mathbf{W}) = f_L \left(\sum_{j=1}^{M_{L-1}} w_{kj}^{(L)} f_{L-1} \left(\sum_{i=1}^{M_{L-2}} w_{ji}^{(L-1)} f_{L-2} (\ldots \sum_{v=1}^{D+1} w_{uv}^{(1)} x_v + w_{u0}^{(1)} \ldots) + \right.\right.$$

Today's Outline

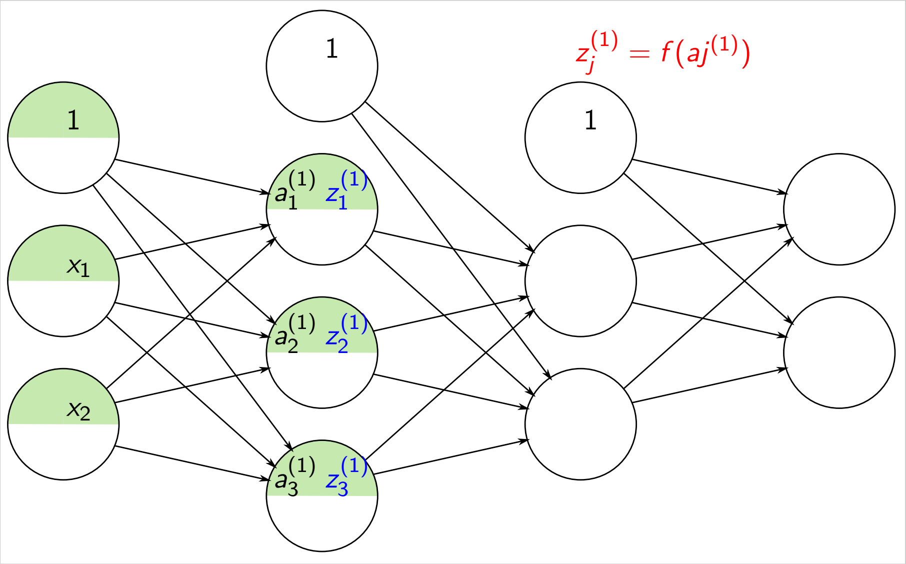
Feed-forward Computation



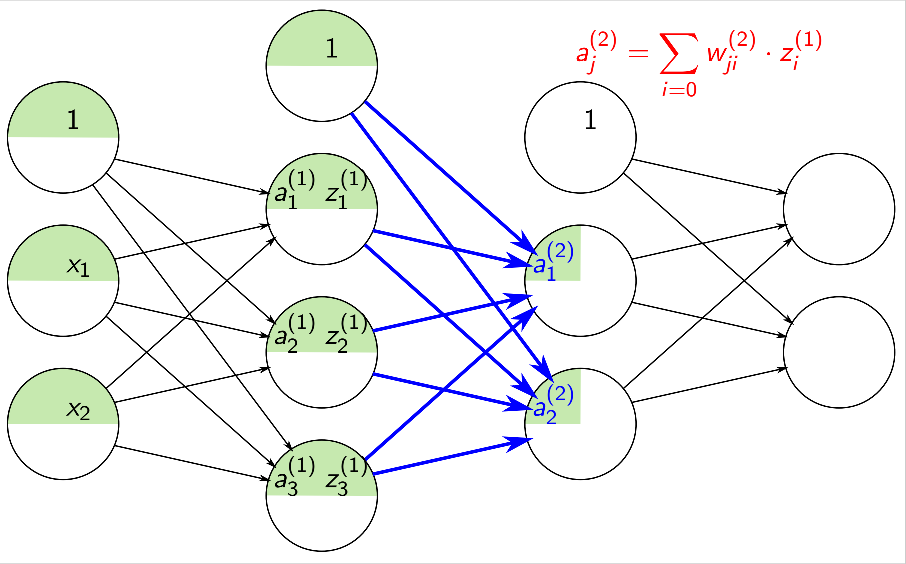
Feed-forward Computation



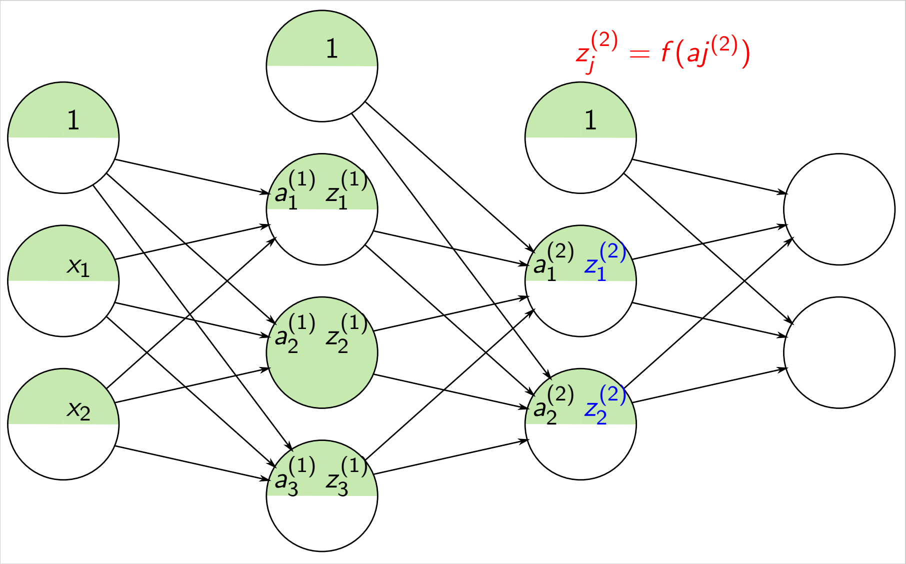
Feed-forward Computation



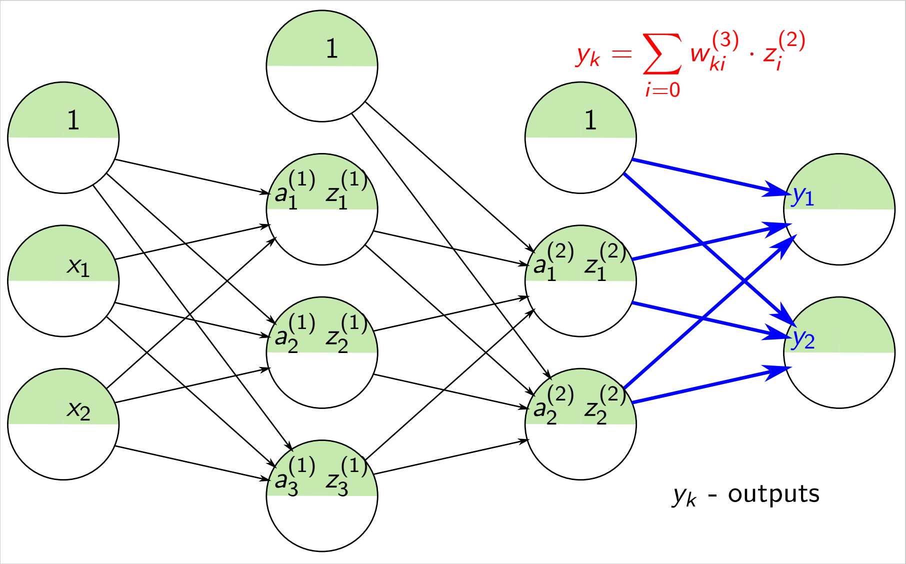
Feed-forward Computation



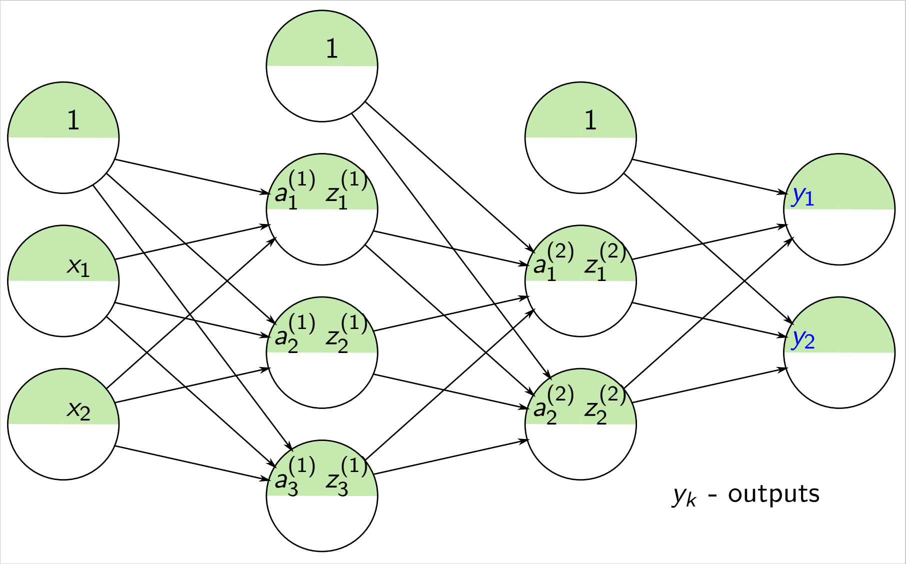
Feed-forward Computation



Feed-forward Computation



Feed-forward Computation



Today's Outline

Sum of Squares for Regression

- Sum of squares error:

$$
\begin{aligned}
E(\mathbf{W}) &= \frac{1}{2}\sum_{n}^{N}\left\| y(\mathbf{W},\mathbf{x}^{(n)}) - \mathbf{t}^{(n)} \right\|_2^2 \\
&= \frac{1}{2}\sum_{n}^{N}\sum_{k}^{K}(y_k(\mathbf{W},\mathbf{x}^{(n)}) - t_k^{(n)})^2
\end{aligned}
$$

Sum of Squares for Regression

- Sum of squares error:

$$\begin{aligned} E\left(\mathbf{W}\right) &= \frac{1}{2}\sum_{n}^{N}\left\|y(\mathbf{W},\mathbf{x}^{(n)})-\mathbf{t}^{(n)}\right\|_2^2 \\ &= \frac{1}{2}\sum_{n}^{N}\sum_{k}^{K}(y_k(\mathbf{W},\mathbf{x}^{(n)})-t_k^{(n)})^2 \end{aligned}$$

- its derivate w.r.t. y_k:

$$\frac{\partial E(\mathbf{W})}{\partial y_k} = y_k - t_k$$

Cross-Entropy for Classification (I)

- The network models the conditional distribution:

$$y_k(\mathbf{x}) = p(C_k|\mathbf{x})$$

- The likelihood of observing the data set (assuming i.i.d. data):

$$\begin{aligned} p(\mathbf{T}|\mathbf{X}) &= \prod_{n}^{N} p\left(\mathbf{t}^{(n)}|\mathbf{x}^{(n)}\right) \\ &= \prod_{n}^{N}\prod_{k}^{K}\left(y_k\left(\mathbf{x}^{(n)}\right)\right)^{t_k^{(n)}} \end{aligned}$$

Cross-Entropy for Classification (II)

- maximizing likelihood is the same with minimizing log-likelihood:

$$E = -\sum_{n}^{N} \sum_{k}^{K} t_k^{(n)} \log y_k(\mathbf{x}^{(n)})$$

- if softmax is used on the last layer:

$$y_k = a_k^{(L)} = \frac{e^{a_k^{(L)}}}{\sum_{k'}^{K} e^{a_{k'}^{(L)}}}$$

- outputs are interpreted as conditional probabilities $p(C_k|\mathbf{x})$

Cross-Entropy for Classification (III)

- the derivatives of the softmax error function

$$\frac{\partial E^{(n)}}{\partial a_k} = \sum_{k'} \frac{\partial E^{(n)}}{\partial y_{k'}} \frac{\partial y_{k'}}{\partial a_k}$$

where

$$\frac{\partial y_{k'}}{\partial a_k} = y_{k'} \delta_{kk'} - y_{k'} y_k$$

and

$$\frac{\partial E^{(n)}}{\partial y_{k'}} = -\frac{t_{k'}}{y_{k'}}$$

leads to:

$$\frac{\partial E^{(n)}}{a_k} = y_k - t_k$$

Today's Outline

The Training Process

- the training process for a feed-forward neural network:
  1. evaluation of the derivatives of the error function $E(\mathbf{w})$
  2. the derivatives are used to adjust the weights

The Training Process

- the training process for a feed-forward neural network:
  1. evaluation of the derivatives of the error function $E(\mathbf{w})$ - backpropagation
  2. the derivatives are used to adjust the weights - gradient descent

The Training Process

- the training process for a feed-forward neural network:
  1. evaluation of the derivatives of the error function $E(\mathbf{w})$ - backpropagation
     - not only for the sum-of-squares error
  2. the derivatives are used to adjust the weights - gradient descent
     - there are other optimization techniques beside gradient descent

The error function

Definition

$$E(\mathbf{w}) = \sum_{n}^{N} E(\mathbf{w}, \mathbf{x}^{(n)}) = \sum_{n}^{N} E_n \tag{1}$$

Assumption: i.i.d. data.

The error function

Definition

$$E(\mathbf{w}) = \sum_{n}^{N} E(\mathbf{w}, \mathbf{x}^{(n)}) = \sum_{n}^{N} E_n \tag{1}$$

Assumption: i.i.d. data.

- The goal: evaluate $\nabla E(\mathbf{w})$

The derivatives of the error

- Applying the chain rule for partial derivatives:

$$\frac{\partial E_n}{\partial w_{ji}^{(l)}} = \frac{\partial E_n}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial w_{ji}^{(l)}} \tag{2}$$

The derivatives of the error

- Applying the chain rule for partial derivatives:

$$\frac{\partial E_n}{\partial w_{ji}^{(l)}} = \frac{\partial E_n}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial w_{ji}^{(l)}} \tag{2}$$

- Remember that:

$$a_j^{(l)} = \sum_i w_{ji}^{(l)} \cdot z_i^{(l-1)}$$

$$\frac{\partial a_j^{(l)}}{\partial w_{ji}^{(l)}} = z_i^{(l-1)}$$

The derivatives of the error

- Applying the chain rule for partial derivatives:

$$\frac{\partial E_n}{\partial w_{ji}^{(l)}} = \frac{\partial E_n}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial w_{ji}^{(l)}} \tag{2}$$

- Remember that:

$$a_j^{(l)} = \sum_i w_{ji}^{(l)} \cdot z_i^{(l-1)}$$

$$\frac{\partial a_j^{(l)}}{\partial w_{ji}^{(l)}} = z_i^{(l-1)}$$

- Notation (δ - errors)

$$\delta_j^{(l)} = \frac{\partial E_n}{\partial a_j^{(l)}}$$

The derivatives of the error

- Applying the chain rule for partial derivatives:

$$\frac{\partial E_n}{\partial w_{ji}^{(l)}} = \frac{\partial E_n}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} \cdot z_i^{(l-1)} \tag{2}$$

- Remember that:

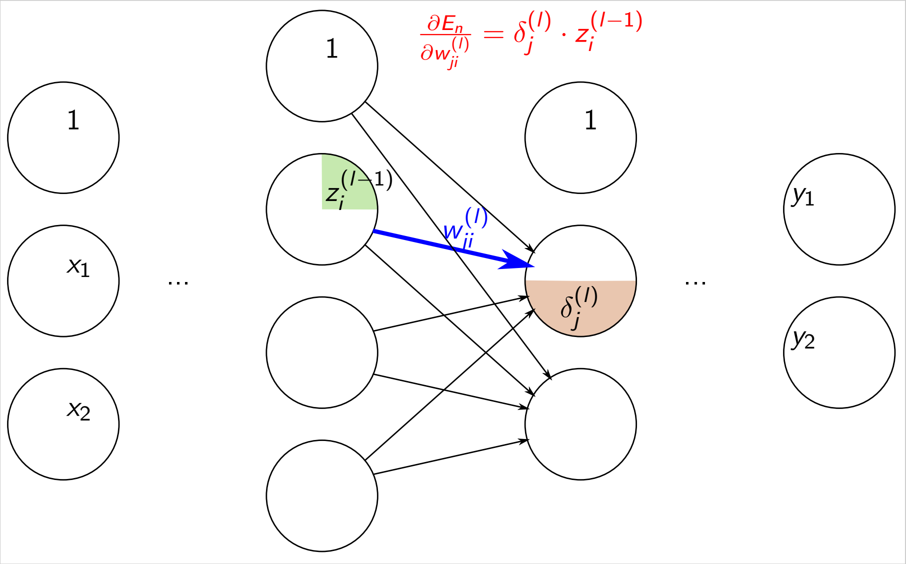
$$a_j^{(l)} = \sum_i w_{ji}^{(l)} \cdot z_i^{(l-1)}$$

$$\frac{\partial a_j^{(l)}}{\partial w_{ji}^{(l)}} = z_i^{(l-1)}$$

- Notation (δ - errors)

$$\delta_j^{(l)} = \frac{\partial E_n}{\partial a_j^{(l)}}$$

The derivatives of the error



Computing δ's

- On the last layer ($l = L, \forall k = 1 \dots K$):

$$\delta_k^{(L)} = \frac{\partial E_n}{\partial a_k^{(L)}} = \frac{\partial E_n}{\partial y_k} \tag{3}$$

Computing δ's

- On the last layer ($l = L, \forall k = 1 \dots K$):

$$\delta_k^{(L)} = \frac{\partial E_n}{\partial a_k^{(L)}} = \frac{\partial E_n}{\partial y_k} = y_k - t_k^{(n)} \tag{3}$$

Computing δ's

- On the last layer ($l = L, \forall k = 1 \dots K$):

$$\delta_k^{(L)} = \frac{\partial E_n}{\partial a_k^{(L)}} = \frac{\partial E_n}{\partial y_k} = y_k - t_k^{(n)} \tag{3}$$

- On the other layers ($l < L, \forall k = 1 \dots M_l$):

$$\delta_j^{(l)} = \frac{\partial E_n}{\partial a_j^{(l)}} = \sum_k \frac{\partial E_n}{\partial a_k^{(l+1)}} \cdot \frac{\partial a_k^{(l+1)}}{\partial a_j^{(l)}} \tag{4}$$

Computing δ's

- On the last layer ($l = L, \forall k = 1 \dots K$):

$$\delta_k^{(L)} = \frac{\partial E_n}{\partial a_k^{(L)}} = \frac{\partial E_n}{\partial y_k} = y_k - t_k^{(n)} \tag{3}$$

- On the other layers ($l < L, \forall k = 1 \dots M_l$):

$$\delta_j^{(l)} = \frac{\partial E_n}{\partial a_j^{(l)}} = \sum_k \frac{\partial E_n}{\partial a_k^{(l+1)}} \cdot \frac{\partial a_k^{(l+1)}}{\partial a_j^{(l)}} \tag{4}$$

- Remember that:
  - $a_k^{(l+1)} = \sum_j w_{kj}^{(l+1)} \cdot z_j^{(l)}$
  - $z_j^{(l)} = f(a_j^{(l)})$

Computing δ's

- On the last layer ($l = L, \forall k = 1 \dots K$):

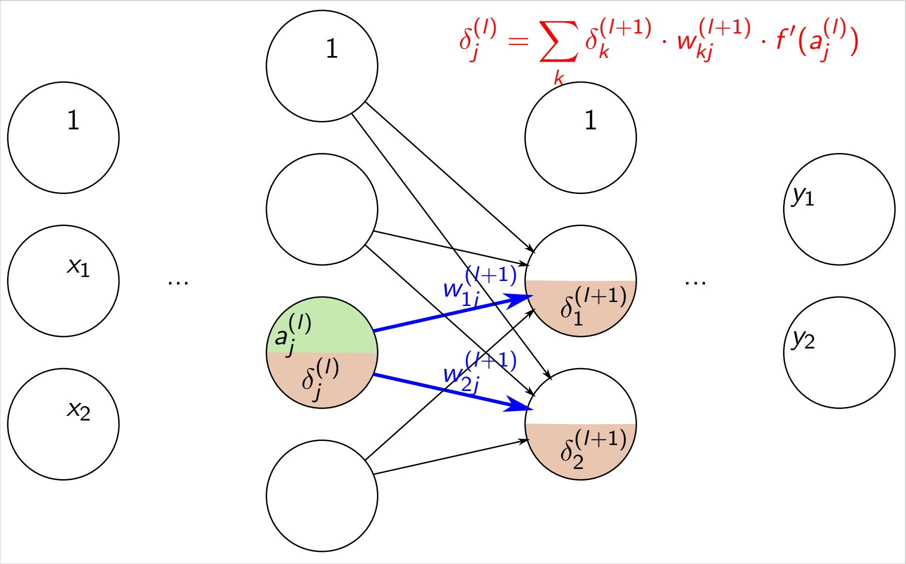
$$\delta_k^{(L)} = \frac{\partial E_n}{\partial a_k^{(L)}} = \frac{\partial E_n}{\partial y_k} = y_k - t_k^{(n)} \tag{3}$$

- On the other layers ($l < L, \forall k = 1 \dots M_l$):

$$\delta_j^{(l)} = \frac{\partial E_n}{\partial a_j^{(l)}} = \sum_k \frac{\partial E_n}{\partial a_k^{(l+1)}} \cdot \frac{\partial a_k^{(l+1)}}{\partial a_j^{(l)}} = \sum_k \delta_k^{(l+1)} \cdot w_{kj}^{(l+1)} \cdot f'(a_j^{(l)}) \tag{4}$$

- Remember that:
  - $a_k^{(l+1)} = \sum_j w_{kj}^{(l+1)} \cdot z_j^{(l)}$
  - $z_j^{(l)} = f(a_j^{(l)})$

The derivatives of the error



Error backpropagation algorithm

Algorithm 1 Error backpropagation

for $k = 1 \dots K$ **do**

  $\delta_k^{(L)} \longleftarrow y_k - t_k$

  $\frac{\partial E_n}{\partial w_{ki}^{(L)}} \longleftarrow \delta_k^{(L)} \cdot z_i^{(L-1)}$

for $l = (L-1) \dots 1$ **do**

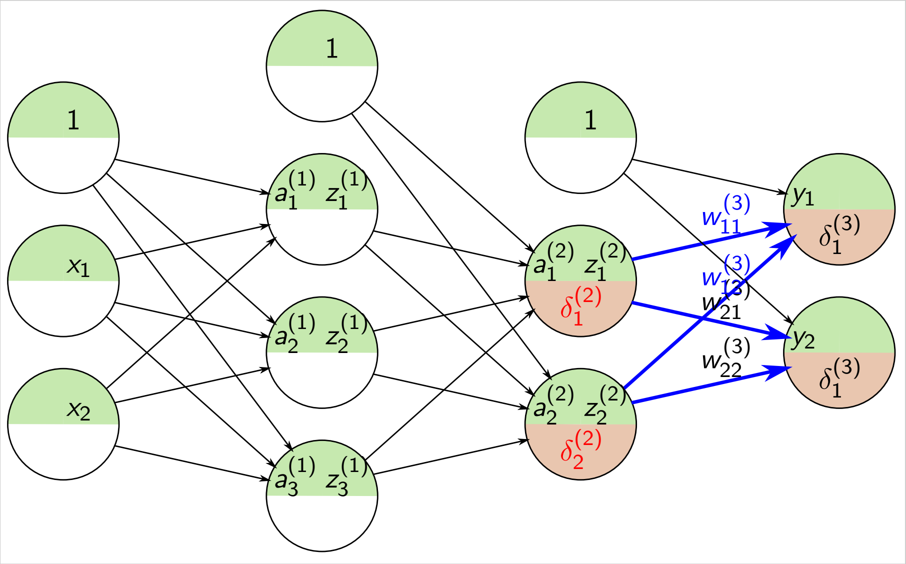
  for $j = 1 \dots M_l$ **do**

    $\delta_j^{(l)} \longleftarrow \sum_k \delta_k^{(l+1)} \cdot w_{kj}^{(l+1)} \cdot f'(a_j^{(l)})$

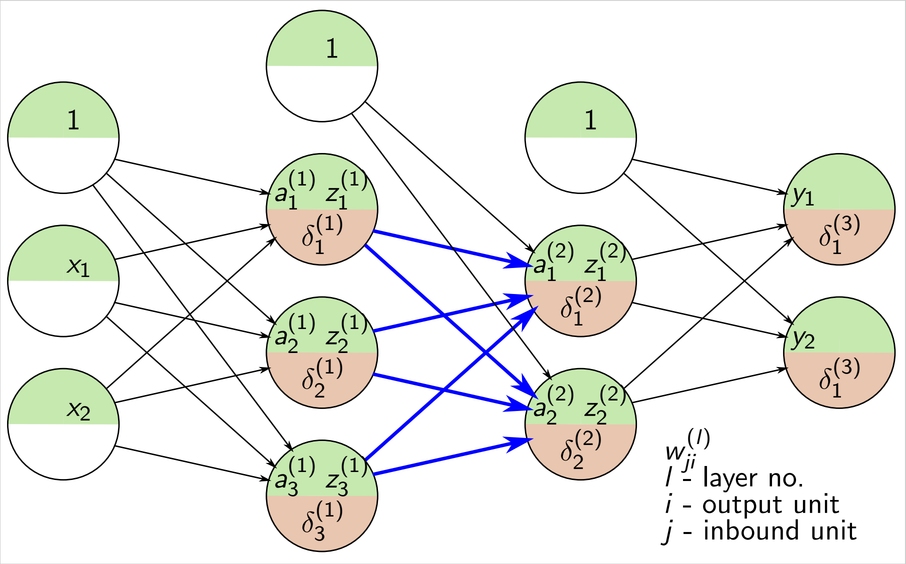
    $\frac{\partial E_n}{\partial w_{ji}^{(l)}} \longleftarrow \delta_j^{(l)} \cdot z_i^{(l-1)}$

Error Backpropagation Example

Error Backpropagation Example



Error Backpropagation Example



Today's Outline

Gradient descent

- today we consider the learning rate constant
- Gradient descent algorithm:
  - until *convergence*:
    1. Compute $\frac{\partial E_n}{\partial w_{ji}^{(l)}}$ using backpropagation
    2. $\mathbf{W}^{t+1} \longleftarrow \mathbf{W}^{(t)} - \eta \nabla E$

Questions

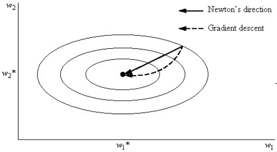
- How often do we change the weights?
- How do we prevent overfitting? Does the network generalize?

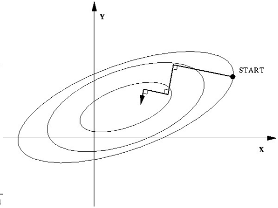
Stochastic vs. batch learning

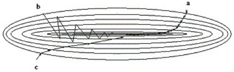
- Stochastic - update weights after each example
  - better for on-line learning
  - better for large redundant data sets
- Batch - update weights after computing error for all training examples
  - supports parallelization
  - better estimation of the error
- Mini-batch

Learning rate

- fixed or adaptive learning rate
- are there any better directions than the steepest descent?







Prevent overfitting

- use a validation set

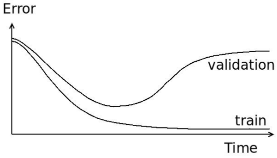


Figure: MSE for training and validation sets

- use regularization
  - optimize the risk: $\mathbf{R} = \alpha E(\mathbf{x}, \mathbf{W}) + (1 - \alpha) C(\mathbf{W})$
    - C - complexity penalty
    - $C = \sum_{w \in \mathbf{W}} ||w||$

Next time..

- strategies to adjust the learning rate
- regularization techniques
- better directions to move along the error surface

Summary

- Adjusting network parameters means:
  1. compute the derivatives of the error with respect to the parameters: $\frac{\partial E}{\partial w_{ij}^{(l)}}$
  2. use those derivatives to optimize weights

  discriminants.
- Error backpropagation gets the gradient ∇E
- Gradient descent optimizes the weights
  - It can be done stochastic, batch, or the mini-batch way

For the exam

For the exam you should know ...

- ... the backpropagation method to compute $\frac{\partial E}{\partial w_{ij}^{(l)}}$
- ... gradient descent optimizations

Read ...

- *Multilayer Perceptrons* [Hay09, Chapter 4]

Today's Outline

6 References

References I

Simon S. Haykin, *Neural networks and learning machines*, Prentice Hall, 2009.