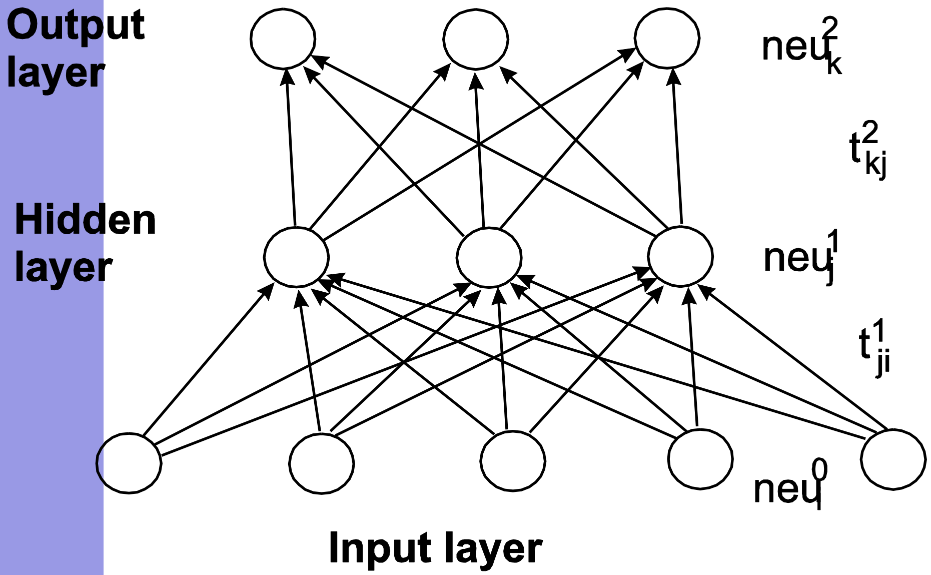
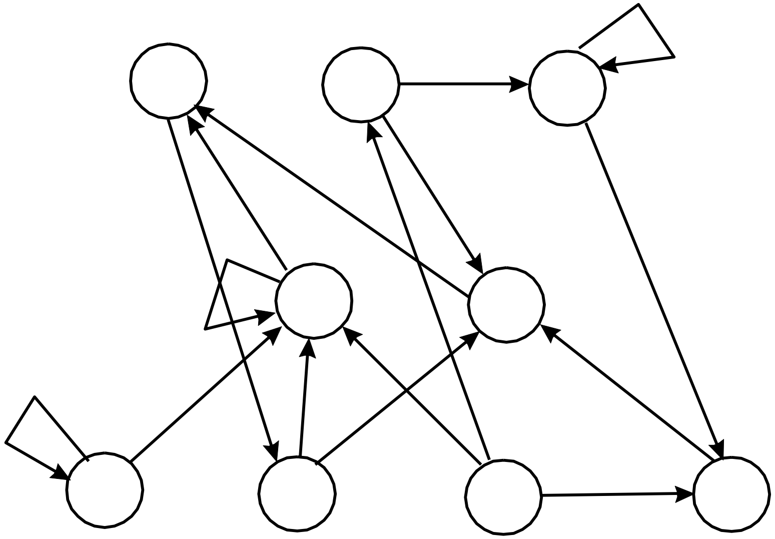
Feed-forward vs. Recurrent neural networks

Feed-forward network



Recurrent network



structure of connections + weights

define what a NN will do.

Learning in NNs

The main advantages of using NNs:

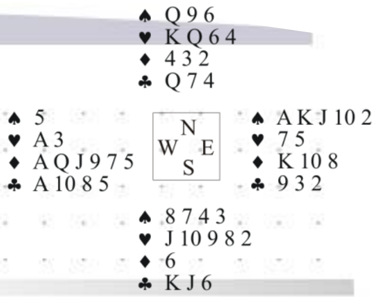
- ability to learn,
- natural ability to parallel processing.

Learning = changing of weights

Three basic learning techniques:

- *supervised learning*
- *unsupervised learning*
- *reinforcement learning*

Supervised learning



Training samples are of the form: $(x^i, d^i) \in R^{m+k}, i = 1,\dots, n$

i.e. *m*-dimensional learning vectors and *k*-dimensional desired outputs.

The task consists in minimization of the recognition error in the output layer (vectors d^i) in case vectors x^i are presented as the inputs.

Error function: $E = \sum_{i=1}^{n} \| y^i - d^i \|$ i.e. $E = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{k}\left(y_j^i - d_j^i\right)^2$

Training patterns are input to the network in a pre-defined order.

Learning consists in the gradual (iterative) change of weights so as to minimize the above error formula.

Simple gradient method

- Algorithm

  1. Choose the starting point x_0

  2. $\mathbf{x_{k+1}} = \mathbf{x_k} - \alpha_k \nabla f(\mathbf{x_k})$

  3. Check stopping criterion, if fulfilled then STOP

  4. If $f(\mathbf{x_{k+1}}) \geqslant f(\mathbf{x_k})$ then decrease α_k and repeat point 2 for the *k*-th step

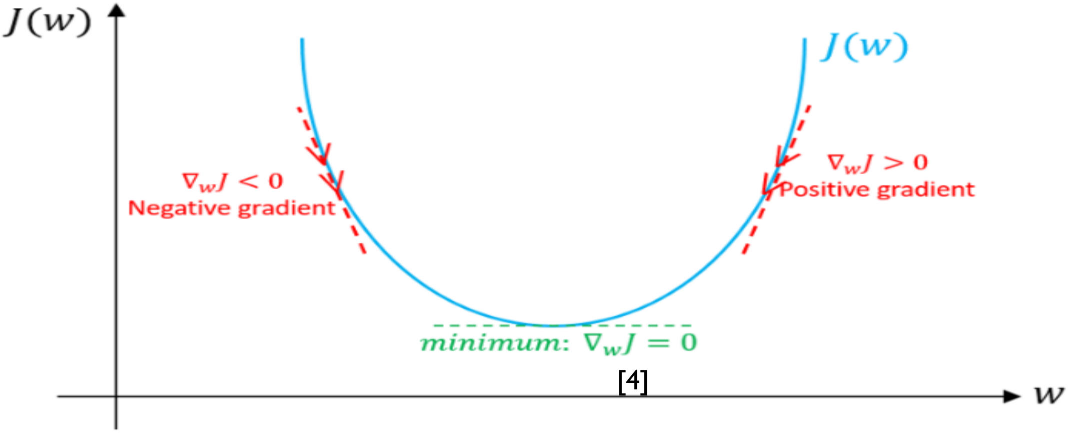
  5. Repeat point 2 for the next step (*k+1*)

- Possible stopping criteri

  ➢

  ➢ $\|\nabla f(\mathbf{x_k})\| \leqslant \epsilon$

  $\|\mathbf{x_{k+1}} - \mathbf{x_k}\| \leqslant \epsilon$



[4]

Backpropagation learning

Iterative weights change: $w(t+1) = w(t) + \eta(t)\, p\big(w(t)\big)$

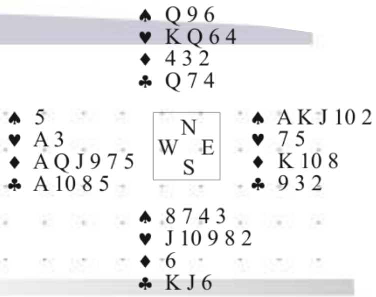
Gradient-based updating method: $p\big(w(t)\big) = -\nabla E\big(w(t)\big)$

Gradient $\nabla E(w)$ can be calculated directly only in the output layer. Its calculation in the previous layers is matematically a bit more complex.

Weights can be updated

- after each presentation of the learning pair (*on-line backpropagation*)
- after presentation of the whole training set (*off-line, batch backprop.*)

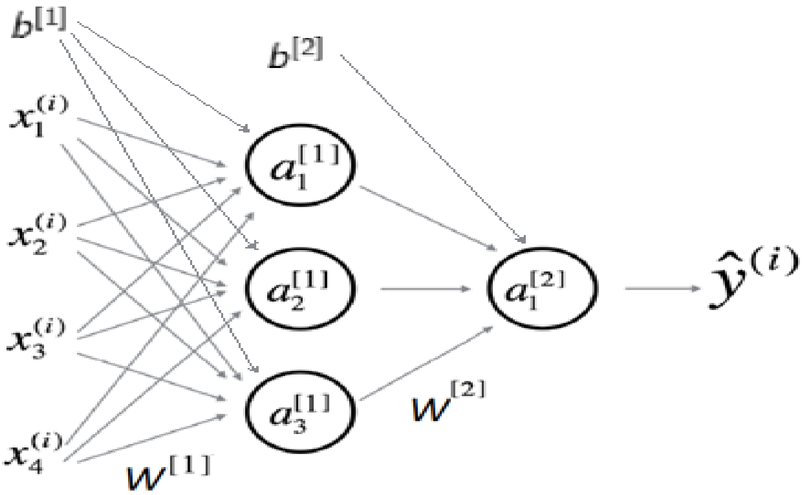
Backpropagation learning: propagation of the error back towards the input layer. The update takes place only in the case of non-zero error (momentum, QuickProp, RProp, …)



Backpropagation

- Notation
  - *m* – the size of training data set
  - Upper index *(i)* (*i* in parenthesis) refers to the *i*-th training sample
  - Upper index *[i]* (*i* in square brackets) refers to the *i*-th network layer
  - Lower index *i* refers to the *i*-th neuron in a layer
  - $W^{[i]}$ – weight matrix between neurons in the *(i-1)*-th and *i*-th leyers
  - $b^{[i]}$ –bias vector in the *i*-th layer
  - $z^{[i]}$ – vector od sums which define inputs to the *i*-th layer neurons
  - $g^{[i]}$ – vector of activation functions in the *i*-th layer
  - $a^{[i]}$ – vector of outputs in the *i*-th layer

Backpropagation



[1]

- Notation
  - $\hat{y}$ – the actual network output
  - $L(\hat{y} ,, y)$ – loss function
  - L – the number of layers

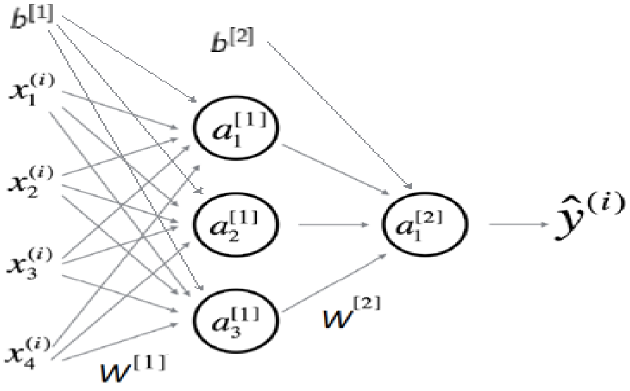
- Observations
  - $\hat{y}$ may also be denoted by $a^{[L]}$
  - $a_j^{[l]} = g^{[l]}(\sum_k w_{jk}^{[l]} a_k^{[l-1]} + b_j^{[l]}) = g^{[l]}(z_j^{[l]})$

- We assume that $g^{[L]}$ is sigmoidal

Backpropagation

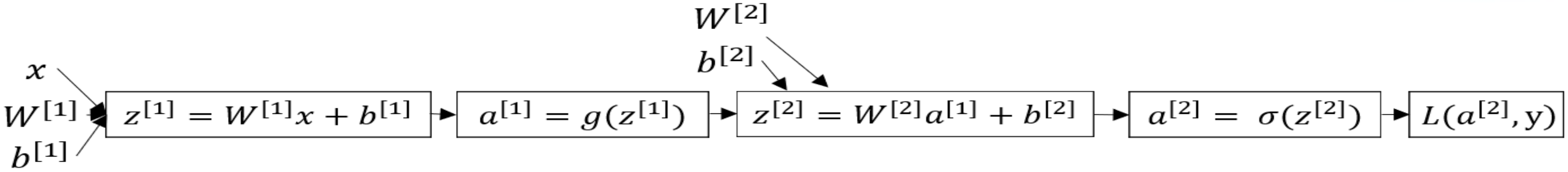
- Algorithm

```
$Repeat\{$
    $//forward\ propagation$
    $z^{[1]} = W^{[1]}x + b^{[1]}$
    $a^{[1]} = g^{[1]}(z^{[1]})$
    $z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$
    $a^{[2]} = g^{[2]}(z^{[2]}) = \sigma(z^{[2]}) = \hat{y}$
    $//backpropagation$
    $W^{[2]} = W^{[2]} - \alpha\frac{\partial L}{\partial W^{[2]}}$
    $b^{[2]} = b^{[2]} - \alpha\frac{\partial L}{\partial b^{[2]}}$
    $W^{[1]} = W^{[1]} - \alpha\frac{\partial L}{\partial W^{[1]}}$
    $b^{[1]} = b^{[1]} - \alpha\frac{\partial L}{\partial b^{[1]}}$
$\}$
```



[1]

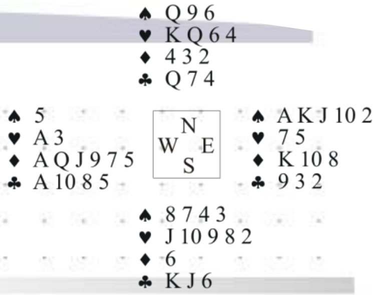
Backpropagation


x
$W^{[1]}$
$b^{[1]}$
$z^{[1]} = W^{[1]}x + b^{[1]}$
$a^{[1]} = g(z^{[1]})$
$W^{[2]}$
$b^{[2]}$
$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$L(a^{[2]}, y)$

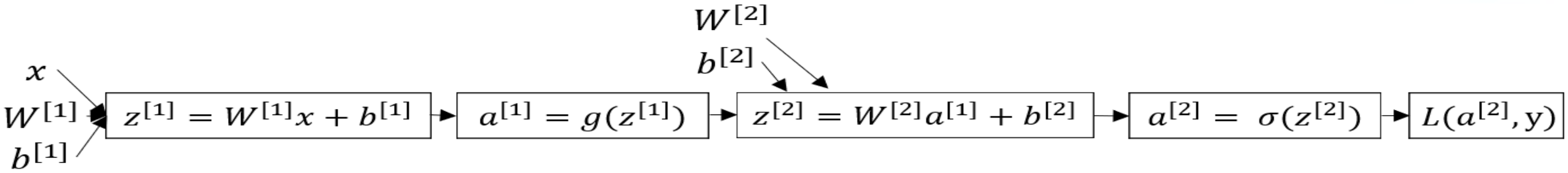
$$L(a^{[2]}, y) = L(\hat{y}, y) = -ylog\,\hat{y} - (1-y)log(1-\hat{y}) =- ylog(a^{[2]}) - (1-y)log(1-a^{[2]})$$

$$\frac{\partial L(a^{[2]}, y)}{\partial a^{[2]}} = \frac{-y}{a^{[2]}} + \frac{1-y}{1-a^{[2]}}$$

$$\frac{\partial L(a^{[2]}, y)}{\partial z^{[2]}} = \frac{\partial L(a^{[2]}, y)}{\partial a^{[2]}}\frac{\partial a^{[2]}}{\partial z^{[2]}} = (\frac{-y}{a^{[2]}} + \frac{1-y}{1-a^{[2]}})\frac{\partial \frac{1}{1+e^{-z^{[2]}}}}{\partial z^{[2]}} = (\frac{-y}{a^{[2]}} + \frac{1-y}{1-a^{[2]}})\frac{e^{-z^{[2]}}}{(1+e^{-z^{[2]}})^2} =$$
$$(\frac{-y}{a^{[2]}} + \frac{1-y}{1-a^{[2]}})\frac{1+e^{-z^{[2]}}-1}{(1+e^{-z^{[2]}})^2} = (\frac{-y}{a^{[2]}} + \frac{1-y}{1-a^{[2]}})(\frac{1}{1+e^{-z^{[2]}}} - \frac{1}{(1+e^{-z^{[2]}})^2}) =$$
$$(\frac{-y}{a^{[2]}} + \frac{1-y}{1-a^{[2]}})(a^{[2]} - (a^{[2]})^2)) = (\frac{-y}{a^{[2]}} + \frac{1-y}{1-a^{[2]}})a^{[2]}(1-a^{[2]}) = -y + ya^{[2]} + a^{[2]} - ya^{[2]} = a^{[2]} - y$$



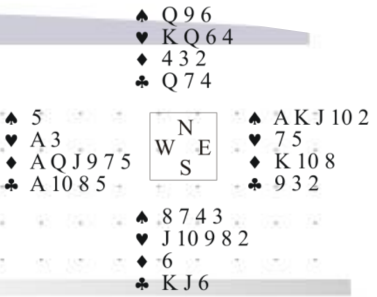
Backpropagation


x
$W^{[1]}$
$b^{[1]}$
$z^{[1]} = W^{[1]}x + b^{[1]}$
$a^{[1]} = g(z^{[1]})$
$W^{[2]}$
$b^{[2]}$
$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$L(a^{[2]}, y)$

$$\frac{\partial L(a^{[2]}, y)}{\partial W^{[2]}} = \frac{\partial L(a^{[2]}, y)}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial W^{[2]}} = \frac{\partial L(a^{[2]}, y)}{\partial z^{[2]}} \frac{\partial (W^{[2]} \cdot {a^{[1]}}^{T} + b^{[2]})}{\partial W^{[2]}} = (a^{[2]} - y) \cdot {a^{[1]}}^{T}$$

$$\frac{\partial L(a^{[2]}, y)}{\partial b^{[2]}} = \frac{\partial L(a^{[2]}, y)}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial b^{[2]}} = \frac{\partial L(a^{[2]}, y)}{\partial z^{[2]}} \frac{\partial (W^{[2]} \cdot {a^{[1]}}^{T} + b^{[2]})}{\partial b^{[2]}} = a^{[2]} - y$$

$$\frac{\partial L(a^{[2]}, y)}{\partial a^{[1]}} = \frac{\partial L(a^{[2]}, y)}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}} = \frac{\partial L(a^{[2]}, y)}{\partial z^{[2]}} \frac{\partial (W^{[2]} \cdot {a^{[1]}}^{T} + b^{[2]})}{\partial a^{[1]}} = (a^{[2]} - y) \cdot {W^{[2]}}^{T}$$


♠ Q 9 6
♥ K Q 6 4
♦ 4 3 2
♣ Q 7 4

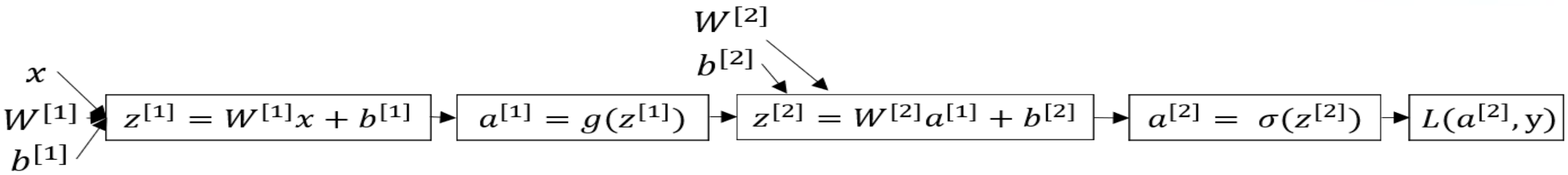
♠ 5
♥ A 3
♦ A Q J 9 7 5
♣ A 10 8 5

N
W E
S

♠ A K J 10 2
♥ 7 5
♦ K 10 8
♣ 9 3 2

♠ 8 7 4 3
♥ J 10 9 8 2
♦ 6
♣ K J 6

Backpropagation



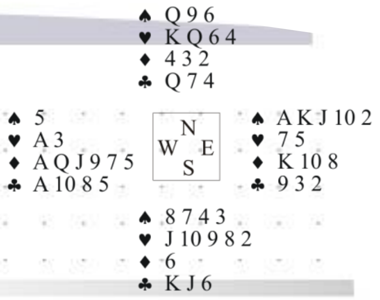
x
$W^{[1]}$
$b^{[1]}$
$z^{[1]} = W^{[1]}x + b^{[1]}$
$a^{[1]} = g(z^{[1]})$
$W^{[2]}$
$b^{[2]}$
$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$L(a^{[2]}, y)$

$$\frac{\partial L(a^{[2]}, y)}{\partial z^{[1]}} = \frac{\partial L(a^{[2]}, y)}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial z^{[1]}} = (a^{[2]} - y) \cdot W^{[2]^T} \cdot g^{[1]\prime}(z^{[1]})$$

$$\frac{\partial L(a^{[2]}, y)}{\partial W^{[1]}} = \frac{\partial L(a^{[2]}, y)}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial W^{[1]}} = (a^{[2]} - y) \cdot W^{[2]^T} \cdot g^{[1]\prime}(z^{[1]}) \cdot \frac{\partial (W^{[1]} \cdot x^T + b^{[1]})}{\partial W^{[1]}} = (a^{[2]} - y) \cdot W^{[2]^T} \cdot g^{[1]\prime}(z^{[1]}) \cdot x^T$$

$$\frac{\partial L(a^{[2]}, y)}{\partial b^{[1]}} = \frac{\partial L(a^{[2]}, y)}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial b^{[1]}} = (a^{[2]} - y) \cdot W^{[2]^T} \cdot g^{[1]\prime}(z^{[1]}) \cdot \frac{\partial (W^{[1]} \cdot x^T + b^{[1]})}{\partial b^{[1]}} = (a^{[2]} - y) \cdot W^{[2]^T} \cdot g^{[1]\prime}(z^{[1]})$$

Backpropagation – pros and cons



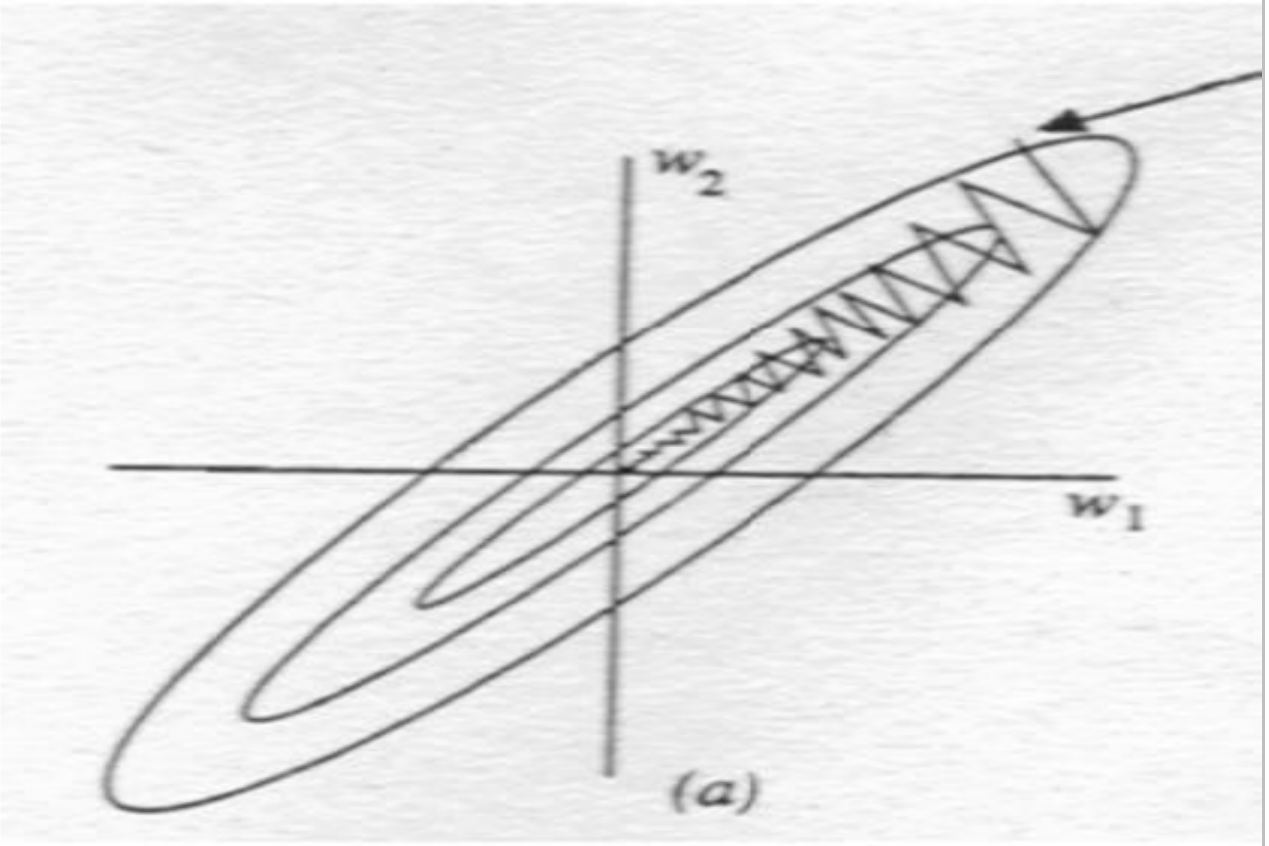
Basic advantage:

- efficiency and universality

Problems:

- local minima
- computational load
- slow convergence, espacially
  - in flat regions of E
  - close to local minima
- possibility of oscillation
- catastrophic interference problem

Oscillation - example



Oscillation – ***momentum*** term

Alleviates the oscillation problem (introduces inertia):

$$\Delta w(t) = -\eta \nabla E\big(w(t)\big) + \alpha \Delta w(t-1)$$

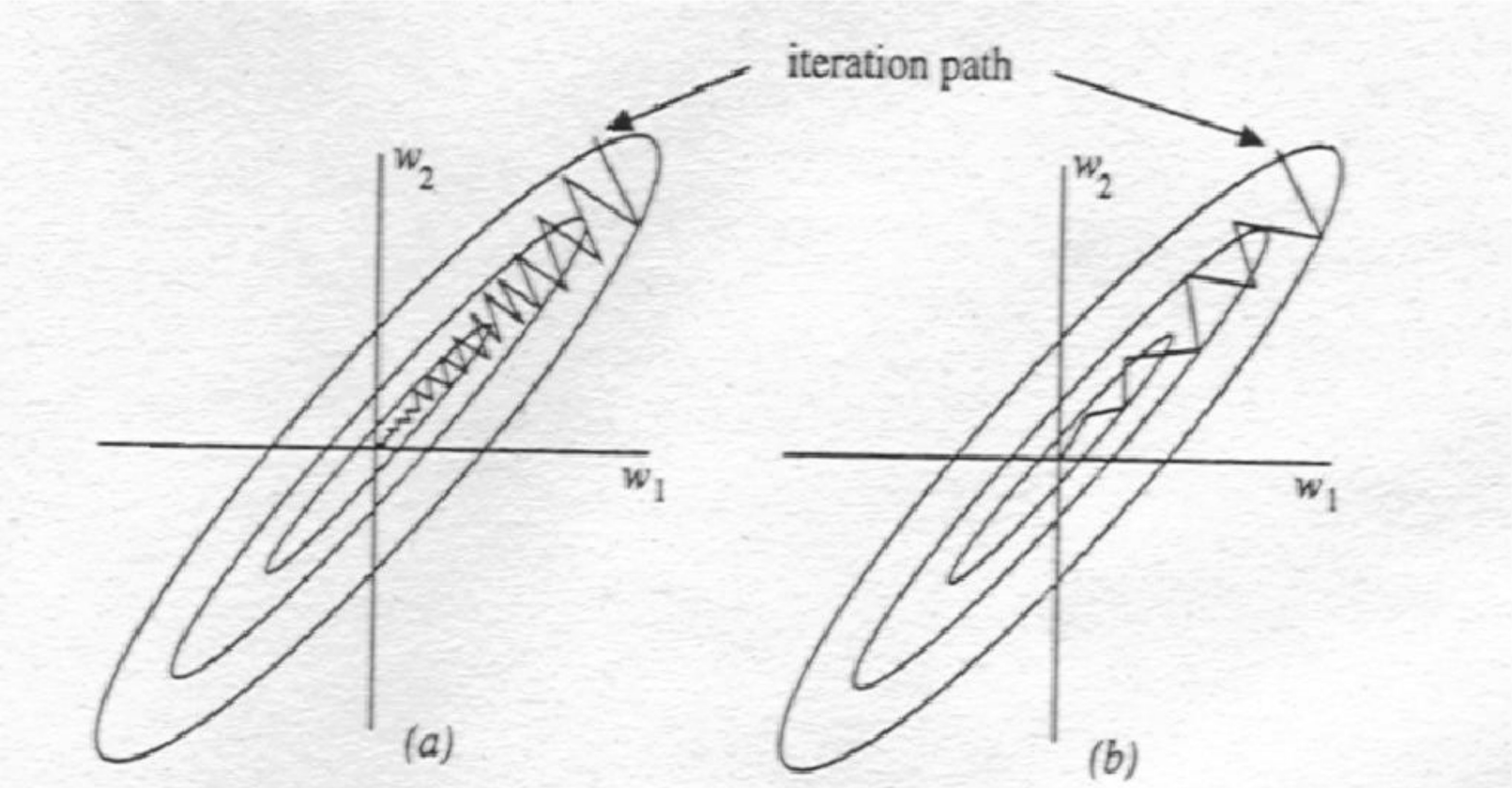
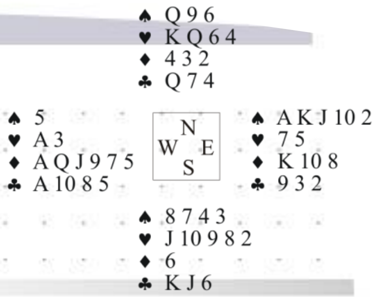


Fig. 8.1. Backpropagation without (a) or with (b) momentum term

Introduction of ***momentum*** term



High advantage in flat regions of E. If there is no momentum term used:

$$\Delta w(t) \approx \Delta w(t-1)$$

$$\Rightarrow \Delta w(t) = \frac{\eta}{1-\alpha}\left(-\nabla E\left(w(t)\right)\right)$$

E.g. for $\alpha = 0.9$ there is a 10-times speed-up.

Selection of momentum value

Coefficient α should be as high as possible, but below a threshold value α_{opt} (generally unknown).

In the case of α too high …

In the case of α too low …

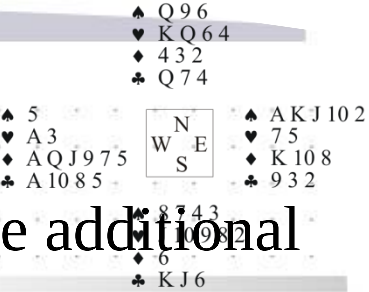
Selection of ***momentum*** – cont.

Usually one controls on-line changes of *E(w(t))*, e.g. as follows:

- if E(t+1) < 1.05 E(t), the change of weights is accepted
- otherwise $\Delta w(t) = 0$ is set and then ...

in the next iteration only gradient term (without momentum) is taken into account

Selection of ***learning rate***



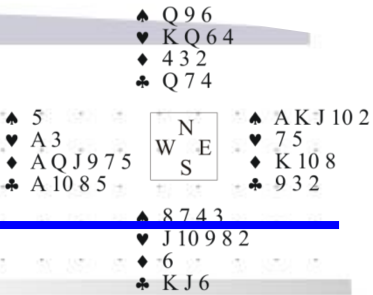
- usually η is selected based on experience, unless some additional information is available (e.g. correlation matrix (X^TX))
- too high value of momentum coefficient leads to oscillations and instability of the learning process
- conservative approach: ...

  Start with a small coefficient; if stuck in a local minimum, increase η and start again, etc.

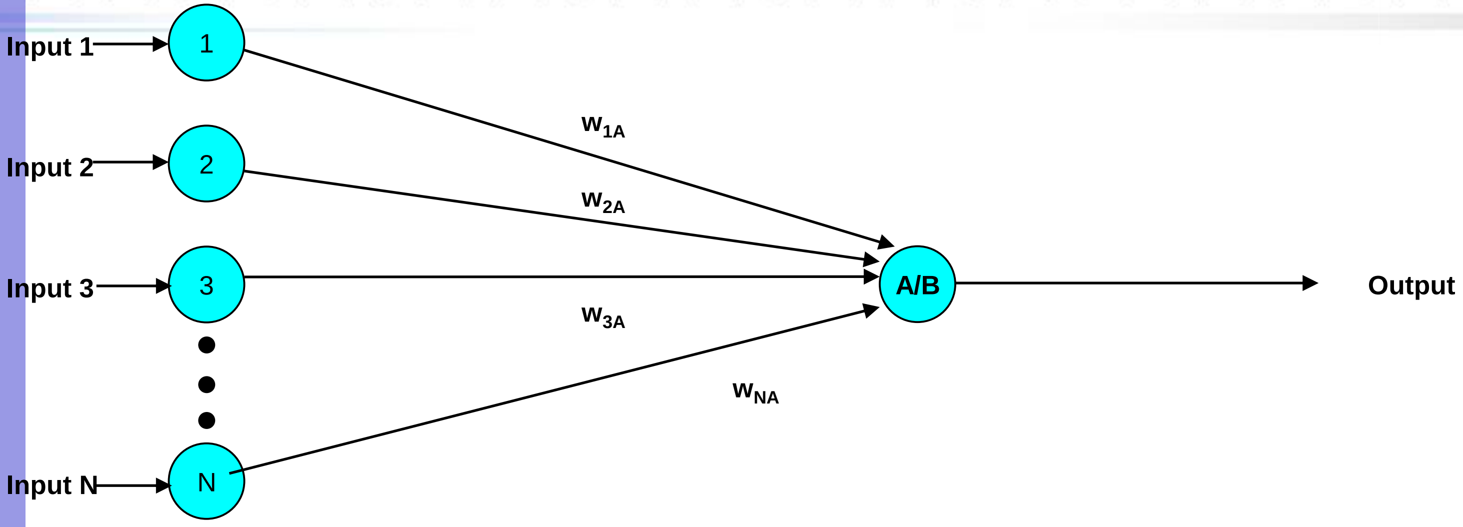
- on-line approach

  Adaptive selection of η - based on the current status of the learning process

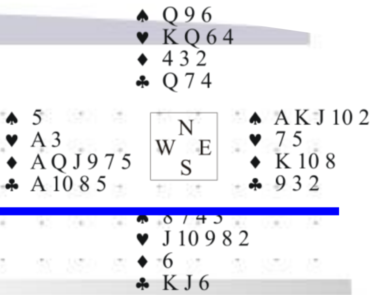
Statistical data pre-processing: decorrelation, dimensionality decreasing, etc.



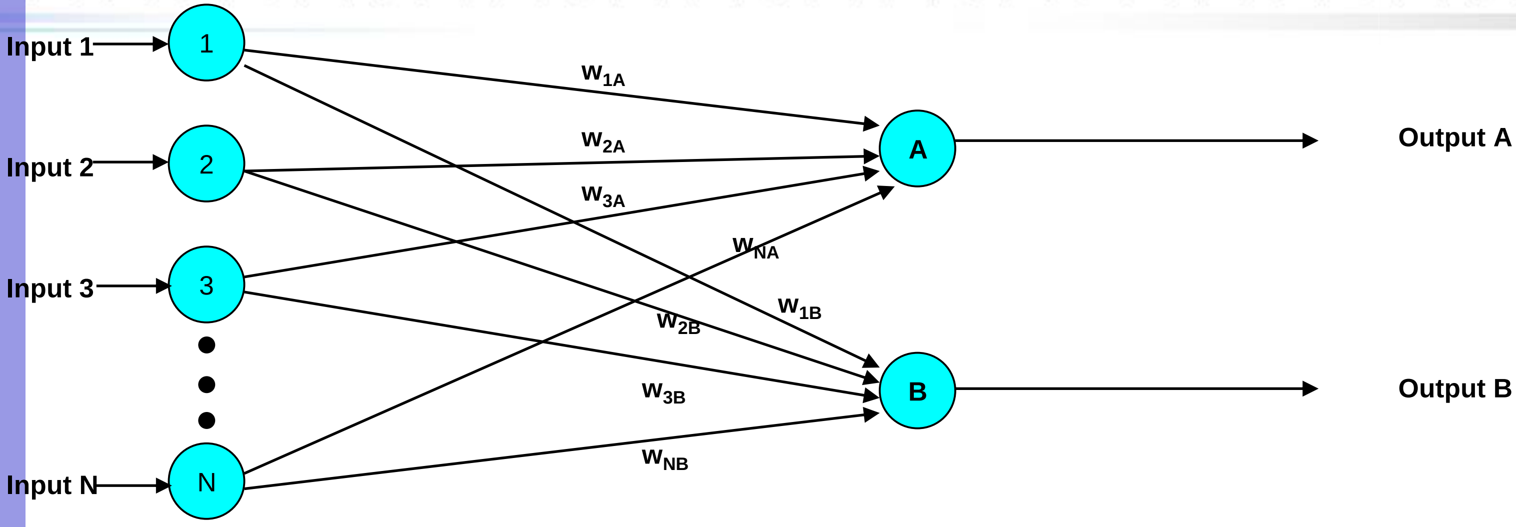
Classification problems



Input / Object	Output
Class A	1
Class B	0

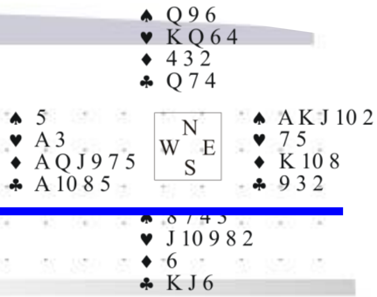


Classification problems



Output Object	Output A	Output B
Class A	1	0
Class B	0	1

Simple perceptron learning



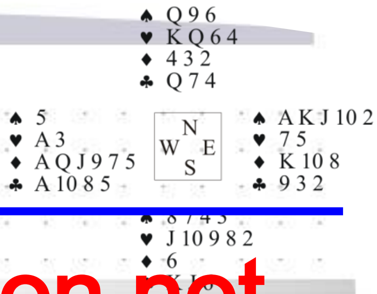
It can be proved that:

„ ... given it is possible to classify a series of inputs, ... then a perceptron network will find this classification”.

another words

„a perceptron will learn the solution, if there is a solution to be found”

Simple perceptron learning



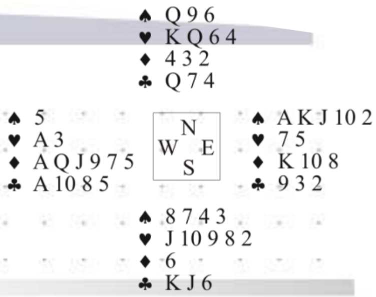
Unfortunately, such the solution not always exists !!!

EXAMPLE (?)

Linear ……

What shall we do (?)

Learning capacity



Kolmogorov (Cybenko) Theorem

One hidden layer perceptron with high enough number of hidden nodes using continuously increasing nonlinearities can compute any continuous function of n variables.

Standard multilayer feed-forward network with a single hidden layer that contains enough (but finite) number of hidden neurons with arbitrary (increasing) activation function are universal approximators on a compact subset of $\mathbb{R}^n$.