Neural Networks

doc. RNDr. Iveta Mrázová, CSc.

DEPARTMENT OF THEORETICAL COMPUTER SCIENCE AND MATHEMATICAL LOGIC

FACULTY OF MATHEMATICS AND PHYSICS, CHARLES UNIVERSITY IN PRAGUE

Neural Networks:

Multi-layered Neural Networks

doc. RNDr. Iveta Mrázová, CSc.

DEPARTMENT OF THEORETICAL COMPUTER SCIENCE AND MATHEMATICAL LOGIC

FACULTY OF MATHEMATICS AND PHYSICS, CHARLES UNIVERSITY IN PRAGUE

Neural Networks:

Contents:

- Introduction to the Field
- Perceptron and Linear Separability
- Multi-layered Neural Networks

Contents:

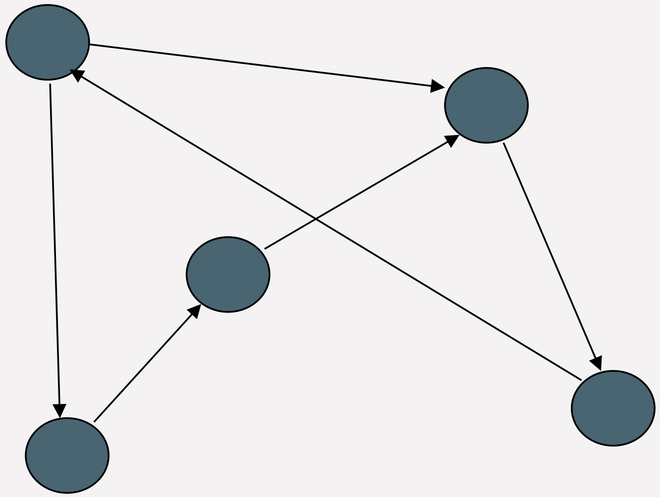
- Introduction to the Field
  - Motivation and a Brief History
  - Biological Background
  - Adaptation and Learning
  - Feature Selection and Ordering
  - Probability and Hypotheses Testing (Review)
- Perceptron and Linear Separability
  - A Formal Neuron
  - Perceptron and Linear Separability
  - Perceptron Learning Algorithm
  - Convergence of Perceptron Learning
  - The Pocket Algorithm

Multi-layered neural networks (1)

D **A neural network** is a 6-tuple $M = (N, C, I, O, w, t)$, where:

- N is a finite non-empty set of neurons,
- $C \subseteq N \times N$ is a non-empty set of oriented interconnections among neurons
- $I \subseteq N$ is a non-empty set of input neurons
- $O \subseteq N$ is a non-empty set of output neurons
- $w: C \to R$ is a weight function
- $t: N \to R$ is a threshold function

  (R is the set of all real numbers)
- (N, C) is called the inter-connection graph of M



Multi-layered neural networks (2)

D **A Back-Propagation network** (BP-network) B is a neural network with a directed acyclic inter-connection graph. Its set of neurons consists of a sequence of $l + 2$ pairwise disjunctive non-empty subsets called layers.

- The first layer called **the input layer** is the set of all input neurons of B, these neurons have no predecessors in the inter-connection graph; their input value x equals their output value.
- The last layer called **the output layer** is the set of all output neurons of B; these neurons are those having no successors in the inter-connection graph.
- All other neurons called hidden neurons are grouped in the remaining l **hidden layers**.

Back-propagation training algorithm (1)

The aim: find such a set of weights that ensure that for each input vector, the output vector produced by the network is the same as (or sufficiently close to) the desired output vector

The actual or desired output values of the hidden neurons are not specified by the task.

- For a fixed, finite training set, the objective function represents the total error between the desired and actual outputs of all the output neurons in the BP-network taken for all the training patterns.

Back-propagation training algorithm (2)

The Error Function

- corresponds to the difference between the actual and desired network output:

Classical MSE

desired output

$$E = \underline{\frac{1}{2}} \sum_{p} \sum_{j} \left(y_{j,p} - d_{j,p} \right)^2$$

patterns

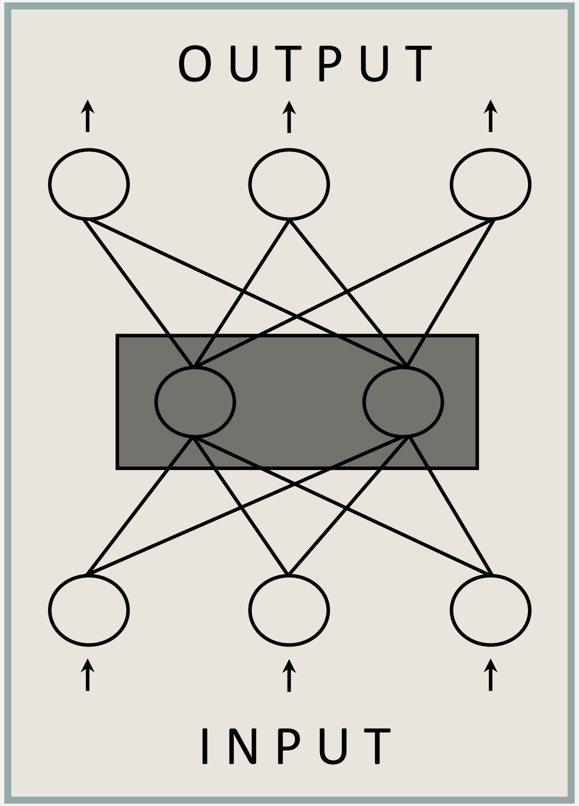
output neurons

actual output

> lépe se pak derivuje a minimalizaci nevadí

- during training, this difference should be minimized on the given training set $\Longrightarrow$ **the back-propagation training algorithm**

Multi-layered neural networks
(BP-networks)



- produce the actual output for the presented input pattern
- compare the actual and desired outputs
- adjust the weights and thresholds
  - against the gradient of the error function
  - from the output layer towards the input layer

BP-networks: adjustment rules (1)

Synaptic weights are adjusted against the gradient:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta_E w_{ij}(t)$$

$\Delta_E w_{ij}(t)$ .... the change of w_{ij} to minimize E

→ rozepíšu a vyjmu složky, které se nepodílí (podílí se pouze y_j, ξ_j, w_{ij}

error at network output

potential of the neuron j

$$\Delta_E w_{ij} = -\frac{\partial E}{\partial w_{ij}} = -\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial \xi_j}\frac{\partial \xi_j}{\partial w_{ij}}$$

actual output

connection weight

protože ostatní členy jsou 0 v té sumě.

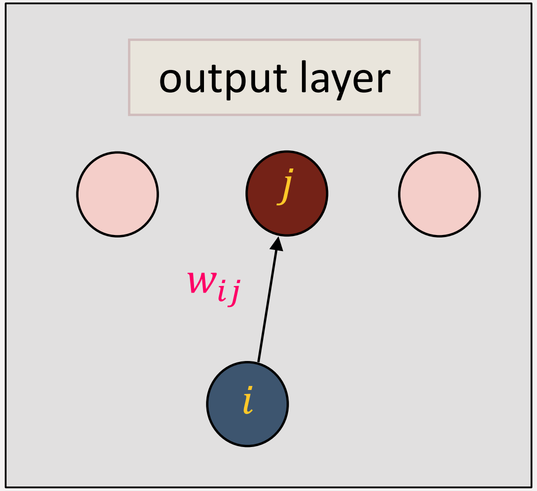
BP-networks: adjustment rules (2)

<u>Weight adjustment in the output layer:</u>

$$\Delta_E w_{ij} \cong -\frac{\partial E}{\partial w_{ij}} = -\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial \xi_j}\frac{\partial \xi_j}{\partial w_{ij}} = -\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial \xi_j}\overbrace{\frac{\partial}{\partial w_{ij}}\overbrace{\sum_{i'} w_{i'j} y_{i'}}^{\xi_j}}^{y_i} =$$

$$= -\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial \xi_j}\, y_i = -\frac{\partial E}{\partial y_j}\,\underbrace{f'(\xi_j)}_{\text{aktivační fce}}\, y_i =$$

$$= -(y_j - d_j)\, f'(\xi_j)\, y_i = \delta_j\, y_i$$



BP-networks: adjustment rules (3)

Weight adjustment in hidden layers:

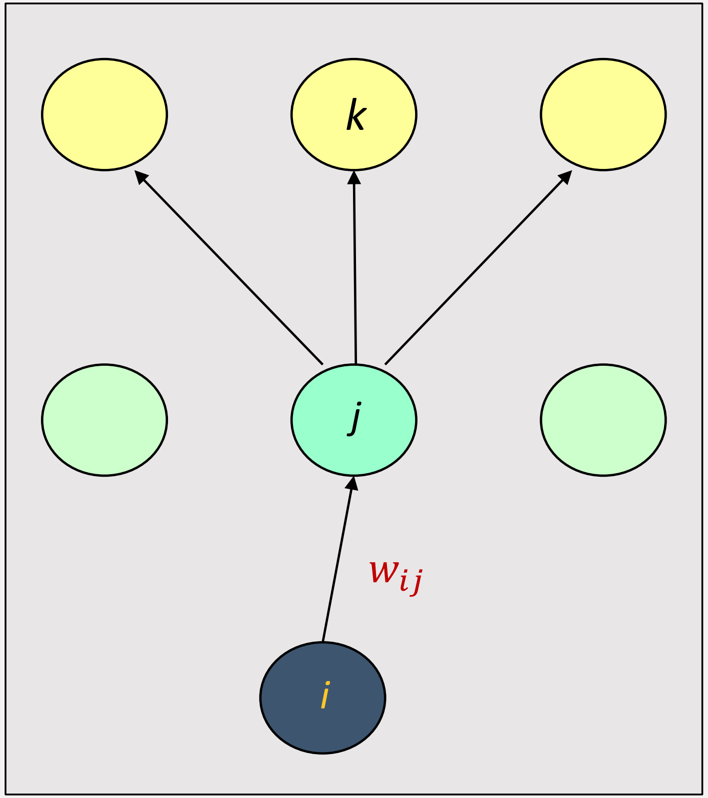
musím jít přes všechny neurony

$$\Delta_E w_{ij} \cong -\frac{\partial E}{\partial w_{ij}} = -\left(\sum_k \frac{\partial E}{\partial \xi_k}\frac{\partial \xi_k}{\partial y_j}\right)\frac{\partial y_j}{\partial \xi_j}\, y_i =$$

$$= -\left(\sum_k \frac{\partial E}{\partial \xi_k}\frac{\partial}{\partial y_j}\sum_{j'} w_{j'k} y_{j'}\right)\frac{\partial y_j}{\partial \xi_j}\, y_i =$$

$$= -\left(\sum_k \frac{\partial E}{\partial \xi_k} w_{jk}\right)\frac{\partial y_j}{\partial \xi_j}\, y_i =$$

$$= \left(\sum_k \delta_k w_{jk}\right) f'(\xi_j)\, y_i = \delta_j\, y_i$$



BP-networks: adjustment rules (4)

- The derivative of the sigmoidal transfer function is:

$$f'(\xi_j) = \lambda y_j (1 - y_j)$$

- Weight adjustment according to:

→ jak moc rychle budu zapomínat historii

$$w_{ij}(t+1) = w_{ij}(t) + \alpha \delta_j y_i + \alpha_m \left(w_{ij}(t) - w_{ij}(t-1) \right)$$

where:

$$\delta_j = \begin{cases} (d_j - y_j)\lambda y_j(1 - y_j) & \text{for an output neuron} \\ \left(\sum_k \delta_k w_{jk}\right)\lambda y_j(1 - y_j) & \text{for a hidden neuron} \end{cases}$$

Back-propagation training algorithm (1)

Step 1: Initialize the weights to small random values

Step 2: Present a new training pattern in the form of:

$$[\mathit{input}\ \vec{x},\ \mathit{desired\ output}\ \vec{d}]$$

Step 3: Calculate actual output in each layer, the activity of the neurons is given by:

$$y_j = f(\xi_j) = \frac{1}{1+e^{-\lambda \xi_j}}, \quad \text{where} \quad \xi_j = \sum_i y_i w_{ij}$$

The activities expressed in this way form the input of the following layer.

Back-propagation training algorithm (2)

Step 4: Weight adjustment starts at the output layer and proceeds back towards the input layer according to:

půjdu opačným směrem od výstupu

$$w_{ij}(t+1) = w_{ij}(t) + \alpha\delta_j y_i + \alpha_m\Big(w_{ij}(t) - w_{ij}(t-1)\Big)$$

$$\delta_j = \begin{cases} (d_j - y_j)\lambda y_j(1-y_j) & \text{for an output neuron} \\ \left(\sum_k \delta_k w_{jk}\right)\lambda y_j(1-y_j) & \text{for a hidden neuron} \end{cases}$$

$w_{ij}(t)$ ……….. weight from neuron $\boldsymbol{i}$ to neuron $\boldsymbol{j}$ in time $\boldsymbol{t}$
α , α_m ……….. learning rate, resp. moment ($\mathbf{0} \leq \boldsymbol{\alpha},\ \boldsymbol{\alpha_m} \leq \mathbf{1}$)
ξ_j, resp. δ_j …… potential, resp. local error on neuron $\boldsymbol{j}$
k …………….. index for the neurons from the layer above the neuron $\boldsymbol{j}$
$\boldsymbol{\lambda}$ …………….. slope of the transfer function

Step 5: Repeat by going to Step 2

An alternative example:

the sample multi-class labels are one hot binary vectors

The SOFTMAX transfer function is used for the output neurons (indexed by j'):

(all the desired output values are either 0 or 1; when using one-hot encoding, there is just one positive class (for the neuron j), all the other ones are negative)

$$y_j = \frac{e^{\xi_j}}{\sum_{j'} e^{\xi_{j'}}} \text{, then: } \frac{\partial y_j}{\partial \xi_j} = \frac{\partial}{\partial \xi_j}\left(\frac{e^{\xi_j}}{\sum_{j'} e^{\xi_{j'}}}\right) = \frac{\left(e^{\xi_j}\right)' \sum_{j'} e^{\xi_{j'}} - e^{\xi_j}\left(\sum_{j'} e^{\xi_{j'}}\right)'}{\left(\sum_{j'} e^{\xi_{j'}}\right)^2} =$$

$$= \frac{e^{\xi_j} \sum_{j'} e^{\xi_{j'}}}{\left(\sum_{j'} e^{\xi_{j'}}\right)^2} - \frac{e^{\xi_j} e^{\xi_j}}{\left(\sum_{j'} e^{\xi_{j'}}\right)^2} = y_j\,(1 - y_j) \text{ for the derivative according to } \xi_j$$

$$\text{and: } \frac{\partial y_j}{\partial \xi_k} = \frac{\partial}{\partial \xi_k}\left(\frac{e^{\xi_j}}{\sum_{j'} e^{\xi_{j'}}}\right) = \frac{\left(e^{\xi_j}\right)' \sum_{j'} e^{\xi_{j'}} - e^{\xi_j}\left(\sum_{j'} e^{\xi_{j'}}\right)'}{\left(\sum_{j'} e^{\xi_{j'}}\right)^2} = \frac{0 \,.\, \sum_{j'} e^{\xi_{j'}}}{\left(\sum_{j'} e^{\xi_{j'}}\right)^2} - \frac{e^{\xi_j} e^{\xi_k}}{\left(\sum_{j'} e^{\xi_{j'}}\right)^2} =$$

$$= 0 - \frac{e^{\xi_j} e^{\xi_k}}{\left(\sum_{j'} e^{\xi_{j'}}\right)^2} = -\,y_j\, y_k \text{ for the derivative according to } \xi_k \text{ with } k \neq j$$

→ ??

sigmoid je jen speciální případ softmax.

An alternative example:

the sample multi-class labels are one hot binary vectors

Cross entropy loss function (~ negative log-likelihood)

$$L = -\sum_{j'} d_{j'} \log y_{j'}\,, \quad \text{then}$$

$$\frac{\partial L}{\partial \xi_j} = \frac{\partial}{\partial \xi_j}\left(-\sum_{j'} d_{j'} \log y_{j'}\right) = -\sum_{j'} d_{j'} \frac{\partial \log y_{j'}}{\partial y_{j'}} \frac{\partial y_{j'}}{\partial \xi_j} =$$

$$= -d_j \frac{1}{y_j} y_j (1 - y_j) - \sum_{j' \neq j} d_{j'} \frac{1}{y_{j'}} (-y_{j'} y_j) =$$

$$= -d_j (1 - y_j) + \sum_{j' \neq j} d_{j'}\, y_j = -d_j + y_j \sum_{j'} d_{j'}$$

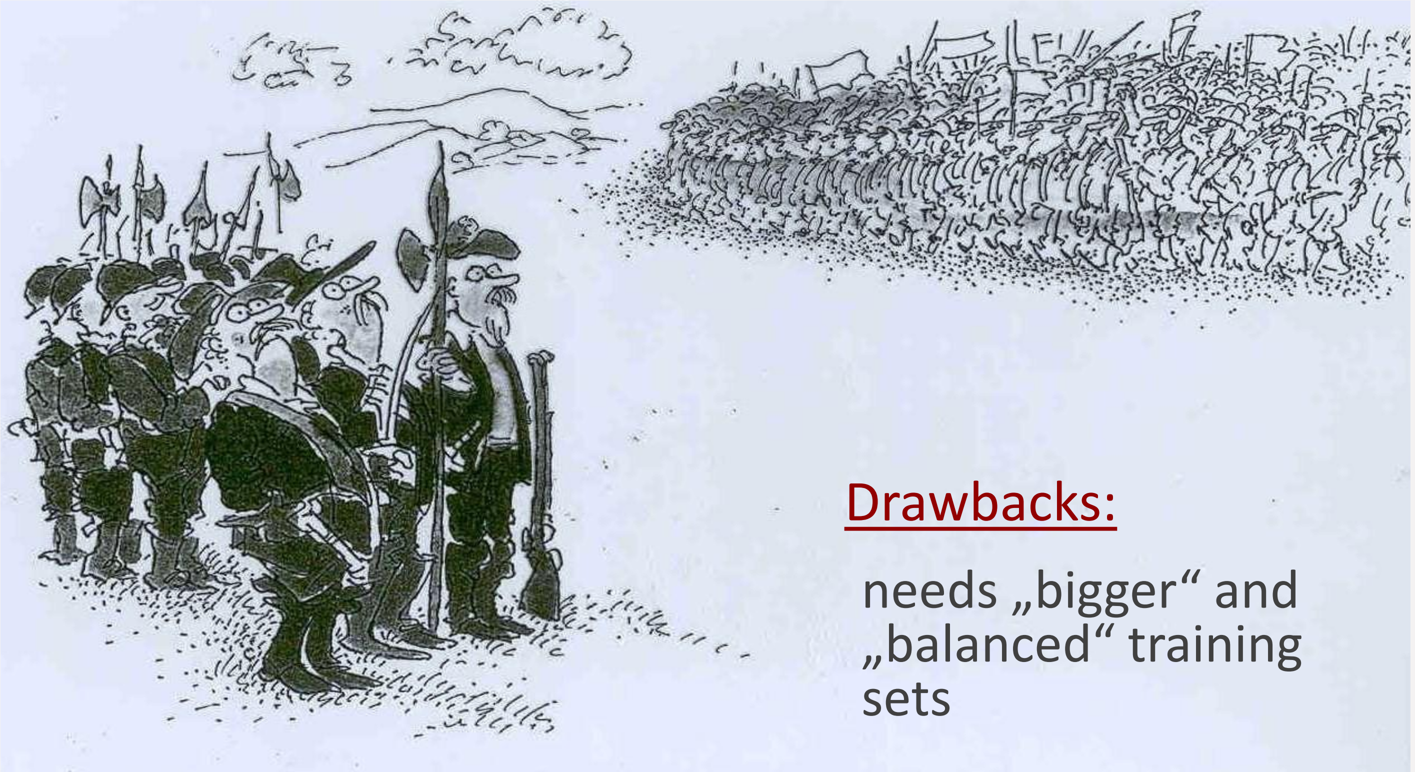
Altogether, we obtain: $\dfrac{\partial L}{\partial \xi_j} = y_j \sum_{j'} d_{j'} - d_j = y_j - d_j$

??

BP-networks: analysis of the model

- Simple training algorithm
- A very often used approach
- Relatively good results
- Drawbacks:
  - Internal knowledge representation – „black box“
  - the number of neurons and generalization capabilities
    - pruning and retraining
  - error function (knowledge of the desired outputs)
    - „bigger“ and „balanced“ training sets
    - assessment of network outputs during recall

BP-networks: analysis of the model



<u>Drawbacks:</u>

needs „bigger“ and „balanced“ training sets

Back-propagation training algorithm: speeding-up the training process (1)

- The standard back-propagation training algorithm is rather slow
  - → a malicious selection of network parameters can make it even slower
- For artificial neural networks, the learning problem is NP-complete in the worst case
  - → computational complexity grows exponentially with the number of the variables
  - → despite of that the standard back-propagation performs often better than many „fast learning algorithms“
    - especially when the task achieves a realistic level of complexity and the size of the training set goes beyond a critical threshold

Back-propagation training algorithm:
speeding-up the training process (2)

Algorithms speeding-up the training process:

- **Keeping a fixed network topology**
- **Modular networks**
  - considerable improvement of network approximation abilities
- **Adjustment of both the parameters** (weights, thresholds, etc.) **and the network topology**

Back-propagation training algorithm: initial weight selection (1)

- The weights should be uniformly distributed over the interval $[-a, +a]$
- Zero mean value
  - leads to an expected zero value of the total input to each node in the network (potential)
- The derivative of the sigmoidal transfer function is reached its maximum for zero (~ 0.25)
  - larger values of the backpropagated errors
  - more significant weight updates when training starts

Back-propagation training algorithm: initial weight selection (2)

Problem:

- **Too small weights paralyze learning**
  - The error backpropagated from the output layer to hidden layers is too small ⇝ and it might even disappear
- **Too large weights lead to saturation of neurons and slow learning** (in flat zones of the error function) ⇝ can't converge

→ **Learning then stops at a suboptimal local minimum**

× the right choice of initial weights can significantly reduce the risk of getting stuck in a local minimum

Back-propagation training algorithm: initial weight selection (3)

Reduce the danger of local minima:

~ initialize the weights with small random values

Motivation:

- **Small weight values**
  - Large weight values impact saturation of hidden neurons (too active or too passive for all training patterns) → such neurons are incapable of further training (the derivative of the transfer function – sigmoid – is almost zero)
- **Random weight values**
  - The goal is to „break the symmetry“ → hidden neurons should specialize in the recognition of different features

Back-propagation training algorithm:
initial weight selection (4)

IDEA:

- **The potential of a hidden neuron is given by:**

$$\xi = w_0 + w_1 x_1 + \cdots + w_n x_n$$

w_0 is the threshold

x_i … the activity of the i-th neuron from the preceding layer

w_i …the weight from the i-th neuron from the preceding layer

- **Expected value of the potential for hidden neurons:**

$$\mathrm{E}\{\xi_j\} = \mathrm{E}\left\{\sum_{i=0}^{n} w_{ij} x_i\right\} = \sum_{i=0}^{n} \mathrm{E}\{w_{ij}\}\ \mathrm{E}\{x_i\} = 0$$

22

  - the weights are independent of the input patterns
  - the weights are random variables with zero mean

Back-propagation training algorithm:
initial weight selection (5)

IDEA - continue:

- **The variance of the potential ξ is given by:**

$$\sigma_{\xi}^{2} = \mathrm{E}\left\{\left(\xi_j\right)^2\right\} - \underbrace{\mathrm{E}^2\{(\xi_j)\}}_{=0} = \mathrm{E}\left\{\left(\sum_{i=0}^{n} w_{ij}\ x_i\right)^2\right\} - 0 =$$

$$= \sum_{i,k=0}^{n} \mathrm{E}\{(w_{ij} w_{kj}\ x_i\ x_k)\} =$$

mutual independence for all j

$$= \sum_{i=0}^{n} \mathrm{E}\left\{\left(w_{ij}\right)^2\right\} \mathrm{E}\{(x_i)^2\}$$

Back-propagation training algorithm:
initial weight selection (6)

IDEA - continue:

- **Further, we assume** that the training patterns are normalized and from the interval $[0,1]$. Then:

$$\mathrm{E}\{\,(x_i)^2\} \;=\; \int_0^1 x_i^2\;\mathrm{d}x \;=\; \frac{x^3}{3}\Big|_0^1 = \frac{1}{3}$$

- Assumed that the weights of the hidden neurons are also random variables with a zero mean and uniformly distributed in the interval $\langle -a, a\rangle$, then:

$$\mathrm{E}\left\{\left(w_{ij}\right)^2\right\} = \int_{-a}^{a} w_{ij}^2 \cdot \frac{1}{2a}\,\mathrm{d}w_{ij} \;=\frac{w_{ij}^3}{6a}\Big|_{-a}^{a} = \frac{a^2}{3}$$

- N … number of weights leading to the considered neuron $(= n+1)$

Back-propagation training algorithm:
initial weight selection (7)

IDEA - continue:

- **Standard deviation will thus correspond to:**

$$A = \sigma_\xi = \sqrt{N}\frac{a}{3} \qquad \left(\rightarrow \quad a = A\frac{3}{\sqrt{N}}\right)$$

- **Neuron potential should be a random variable with the standard deviation** A (that is moreover independent of the number of weights leading to this neuron);
- **Select initial weights (roughly) from the interval:**

ideální nastavení vah na začátku.

$$\left[-\frac{3}{\sqrt{N}} \cdot A, \frac{3}{\sqrt{N}} \cdot A\right]$$

  - especially for $A = 1$ large gradient (i.e., quick learning)