Cybernetics and Artificial Intelligence

4. Clustering and neural networks

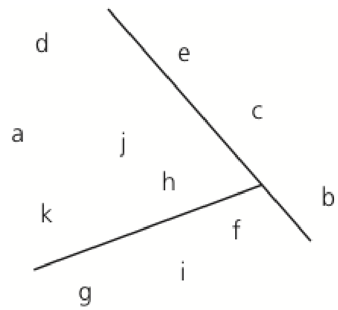
Gerstner laboratory
Dept. of Cybernetics
Czech Technical University in Prague

Summary of the last lecture

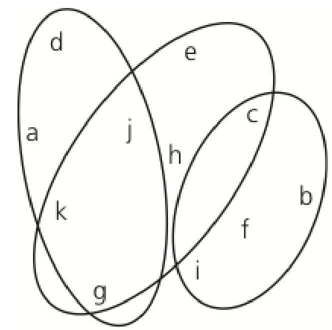
- We know pdf and its parameters. We can use Bayes theorem applying maximum a posteriori approach, $\arg\max_s p(s|x, \mu_s, \sigma_s)$
- In case of Gaussian distribution we get quadratic discrimination function, in case of equal covariance we obtain linear function
- Linear discrimination fce (without pdf knowledge), parameters estimation leads to percepton algorithm
- Zero error classification only for linear separable data
- Non-linear separable problem
  - Transformation of features to higher dimension, e.g. using quadratic transformation
  - Using more sophisticated classifiers, e.g. neural nets
  - Decision trees: discrimination boundaries, construction $\rightarrow$, entropy meassures

Clustering

- No training data
- Natural clusters
- (a) k-means, (b) fuzzy clustering (c) probability using probability mixture , (d) hierarchical clustering (dendogram)



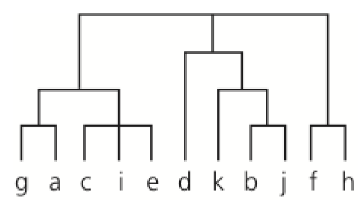
(a)



(b)

	1	2	3
a	0.4	0.1	0.5
b	0.1	0.8	0.1
c	0.3	0.3	0.4
d	0.1	0.1	0.8
e	0.4	0.2	0.4
f	0.1	0.4	0.5
g	0.7	0.2	0.1
h	0.5	0.4	0.1

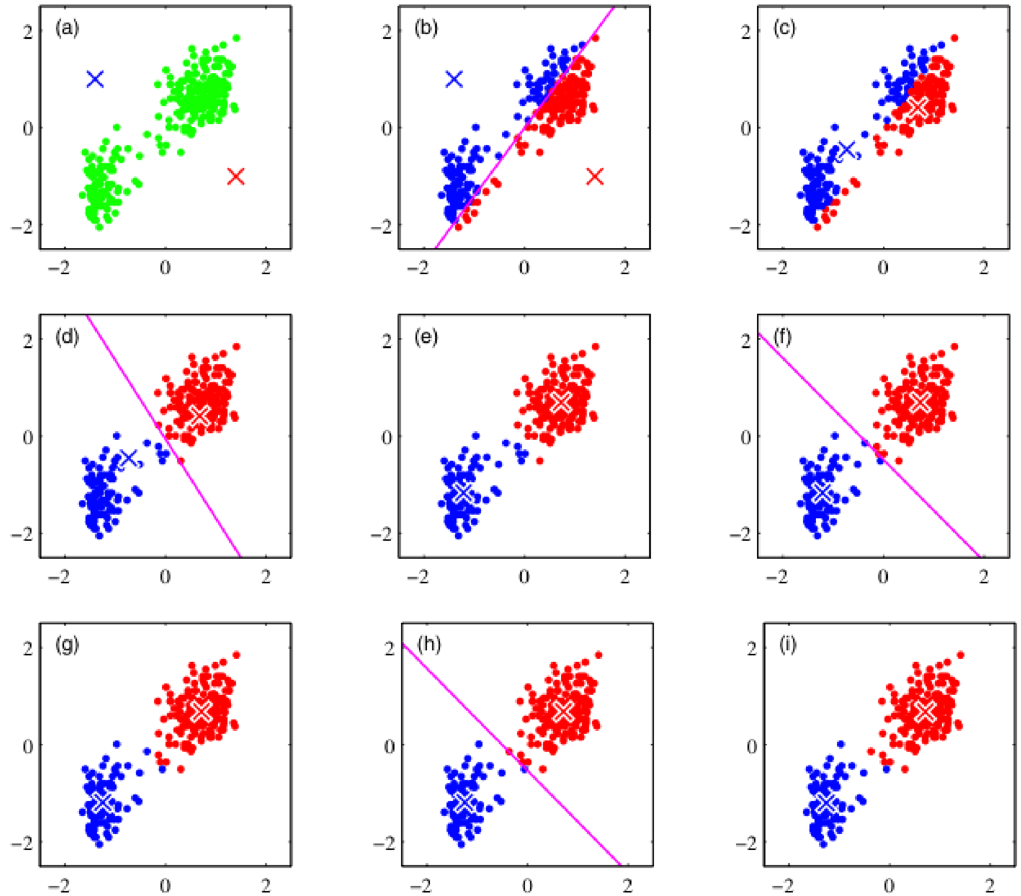
(c)



(d)

K-means

1. beginInicialize $k, \mu_1, \mu_2, \dots, \mu_k$
2. do classify sample according to nearest μ_i
3. update μ_i
4. until no change μ_i
5. return $\mu_1, \mu_2, \dots, \mu_k$
6. end



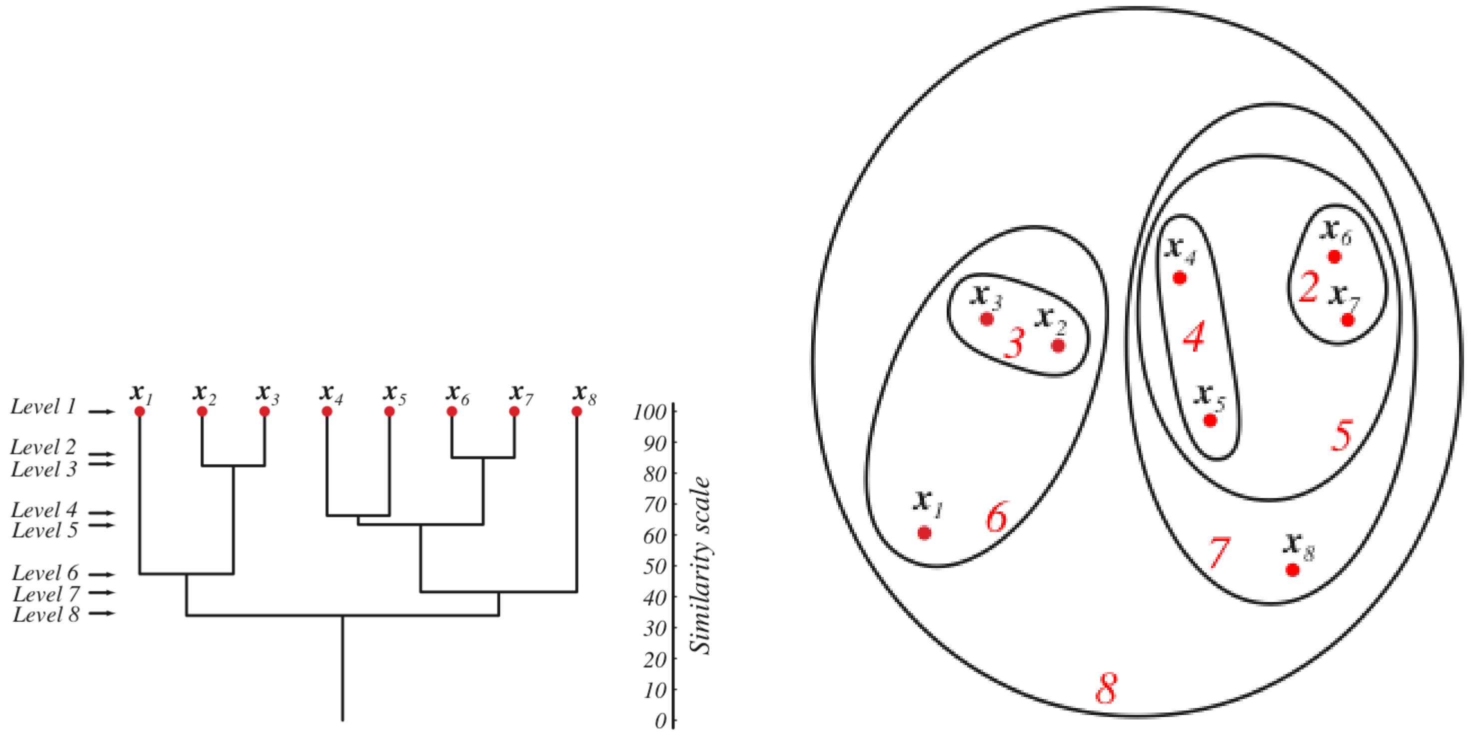
Hierarchical clustering

- agglomerative: bottom-up $\rightarrow$ merging
- divisive: top-down $\rightarrow$ splitting

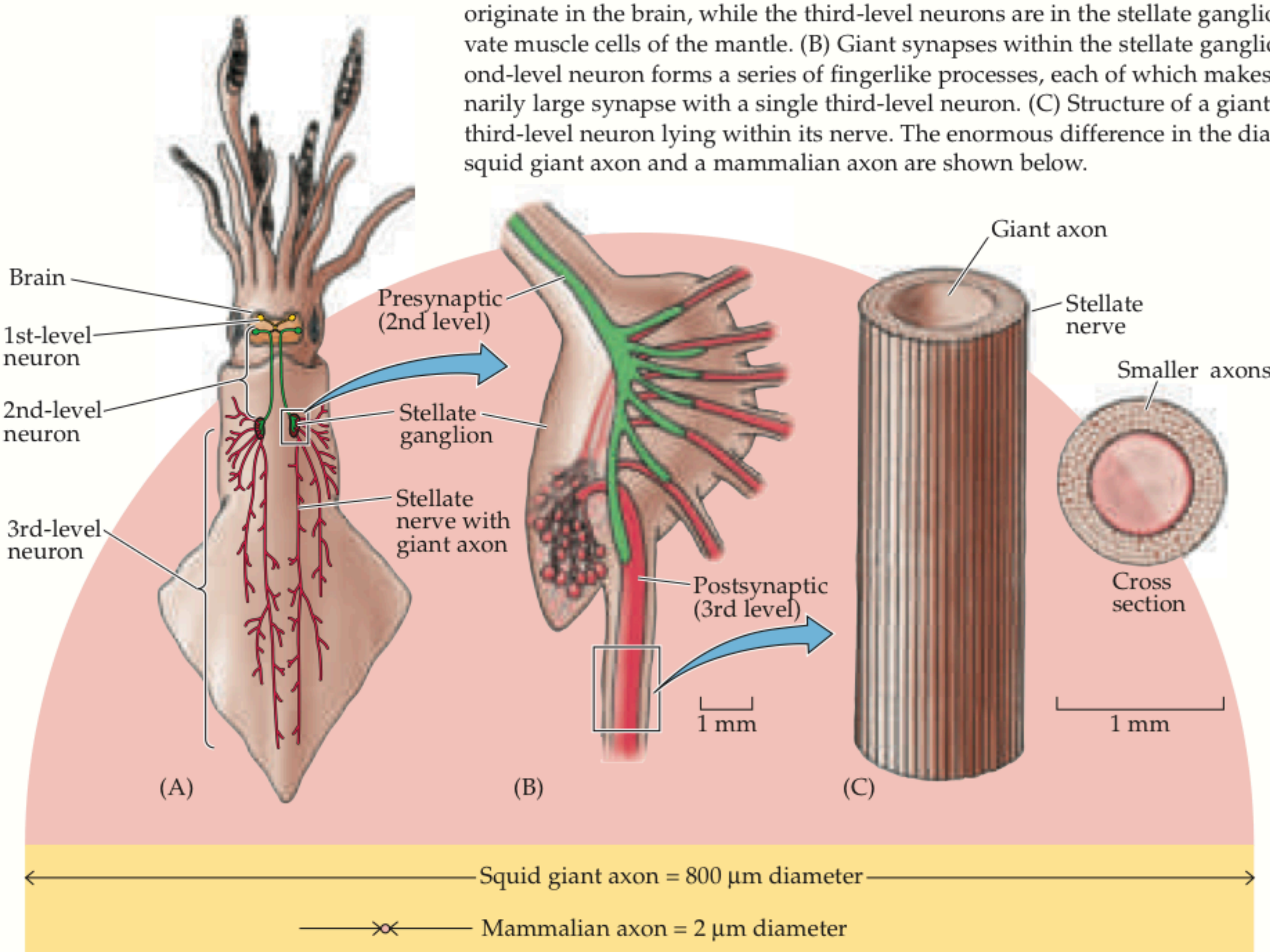
1. begin Initialize$k, \hat{k} \leftarrow n, \mathcal{D}_i \leftarrow \{X_i\}, i = 1, \ldots, n$
2. do $\hat{k} = \hat{k} - 1$
3. find nearest clusters. $\mathcal{D}_i$ a $\mathcal{D}_j$
4. until $k = \hat{k}$
5. return k clusters
6. end

- $d_{min}(x, x^{'}) = \min \|x - x^{'}\|$, $x \in \mathcal{D}_i$, $x^{'} \in \mathcal{D}_i$

Hierarchical clustering - example

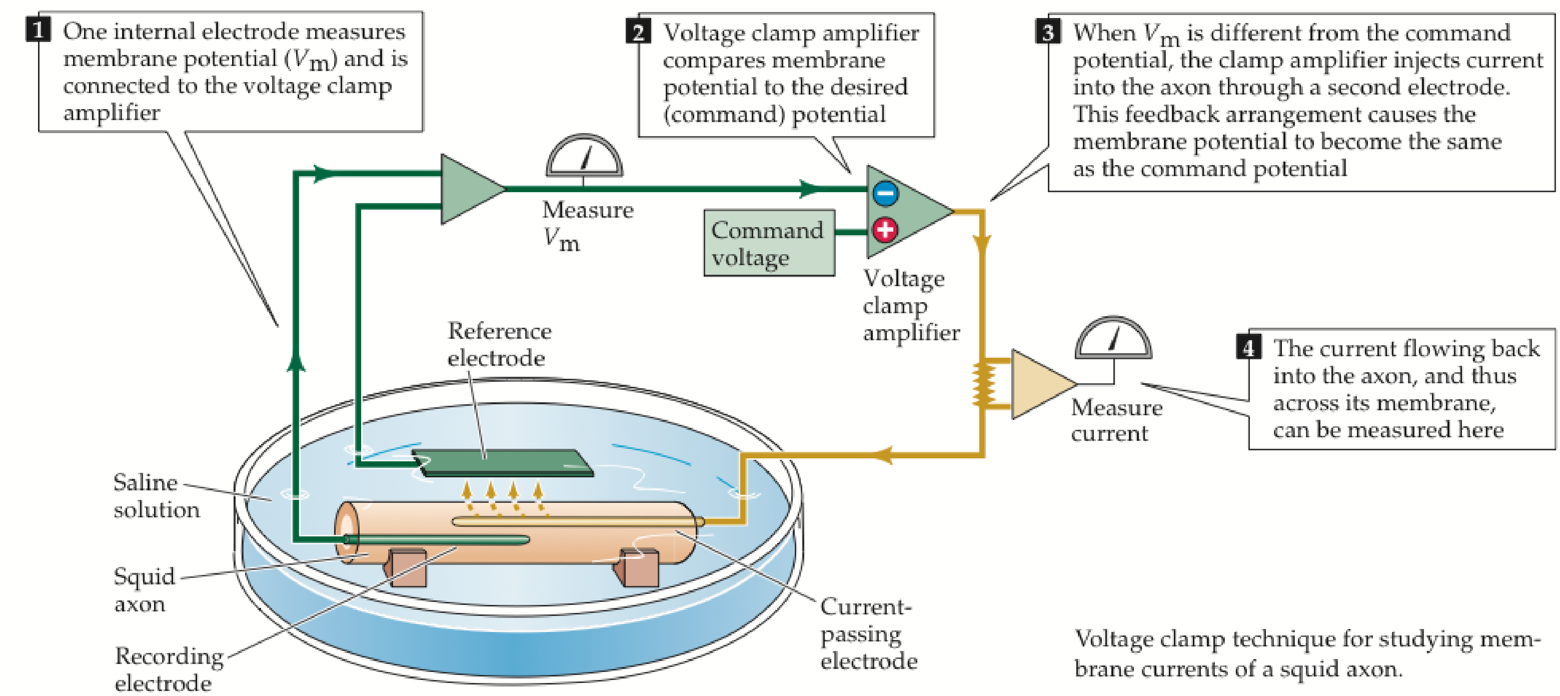


[Giant Nerve Cells of Squid]



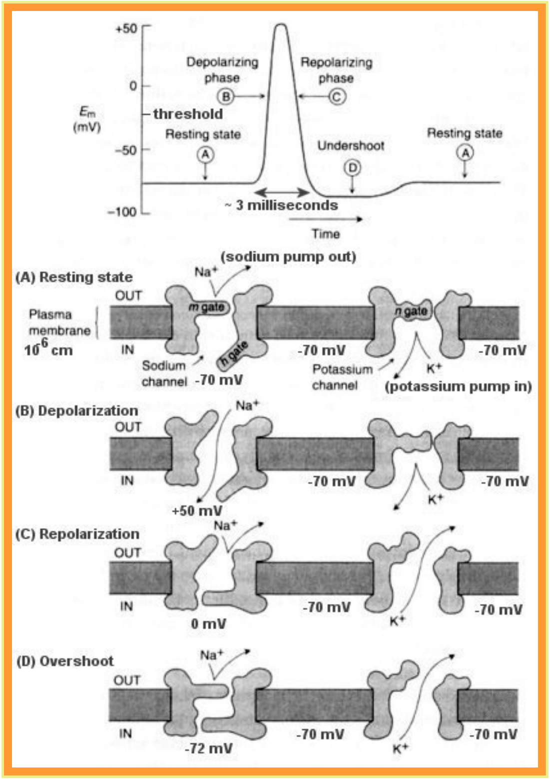
(A) Diagram of a squid, showing the location of its giant nerve cells. Different colors indicate the neuronal components of the escape circuitry. The first- and second-level neurons originate in the brain, while the third-level neurons are in the stellate ganglion and innervate muscle cells of the mantle. (B) Giant synapses within the stellate ganglion. The second-level neuron forms a series of fingerlike processes, each of which makes an extraordinarily large synapse with a single third-level neuron. (C) Structure of a giant axon of a third-level neuron lying within its nerve. The enormous difference in the diameters of a squid giant axon and a mammalian axon are shown below.

[Voltage Clamp Method]



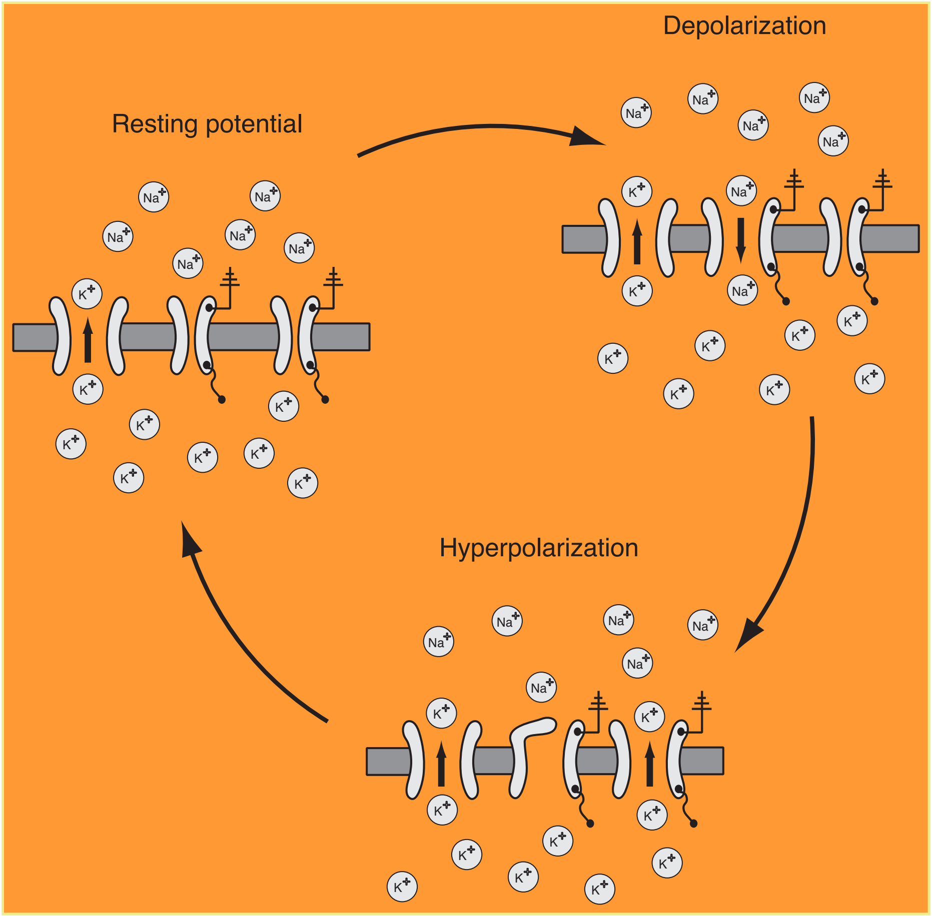
Voltage clamp technique for studying membrane currents of a squid axon.

[Hodgkin–Huxley model]

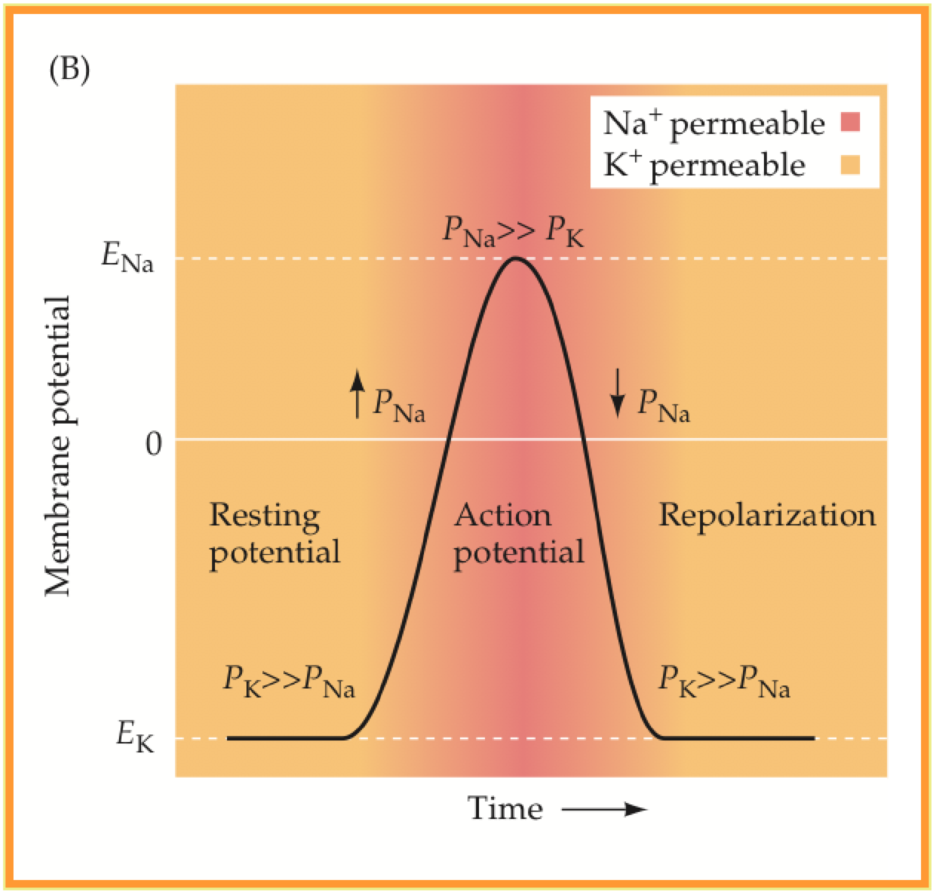


Obrázek 1: Typical form of an action potential; redrawn from an oscilloscope picture from Hodgkin and Huxley (1939).

[The minimal mechanisms]



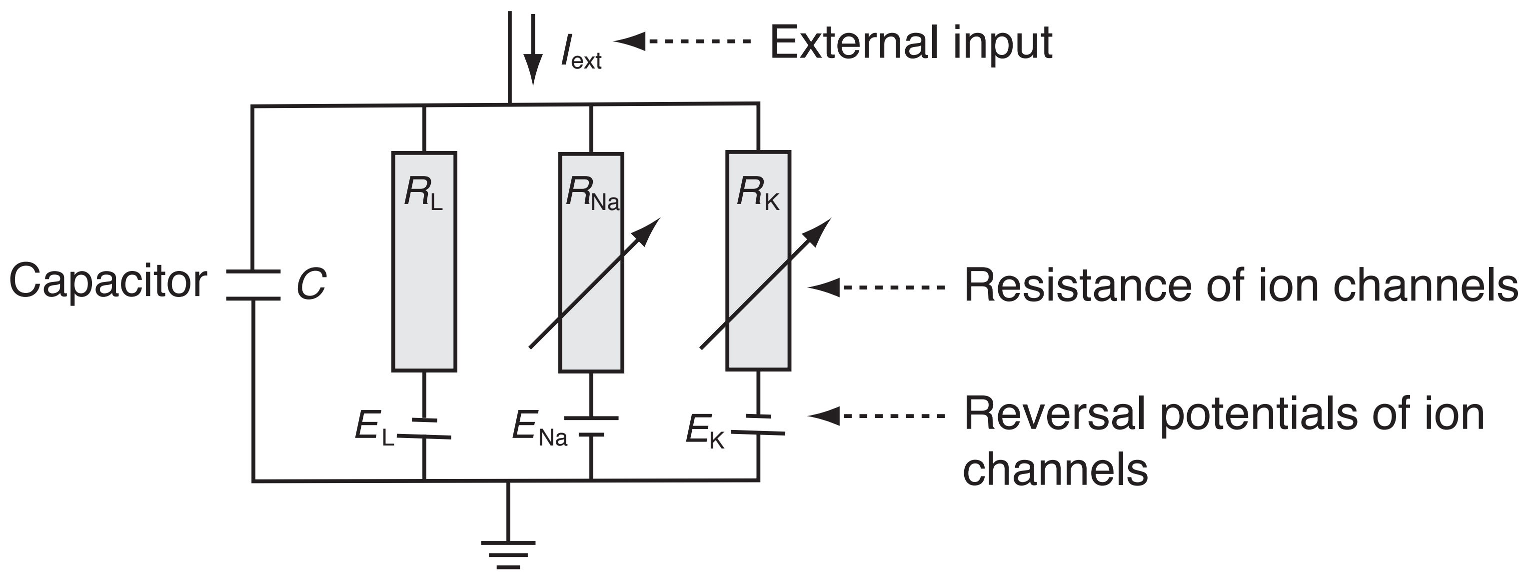
[Concentration of Na,K]



[HH stucture]

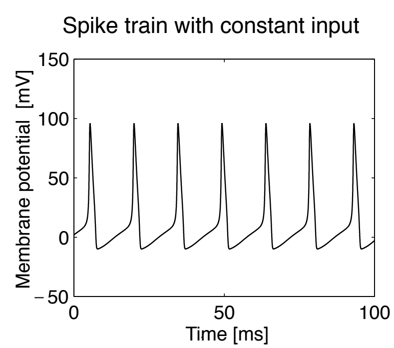
- $I_{ion} = \hat{g_{ion}}(V - E_{ion})$
- voltage and time dependent variables $n(V,t)$,$m(V,t)$,$h(V,t)$

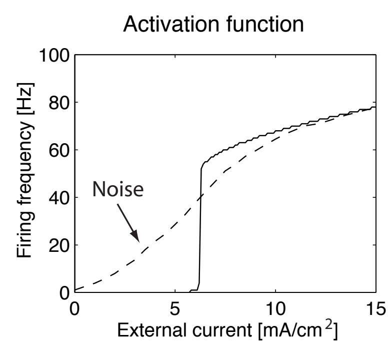
$$\hat{g_K}(V,t) = g_K n^4$$
$$\hat{g_{Na}}(V,t) = g_{Na} m^3 h$$



[Hodgkin–Huxley equations and simulation]

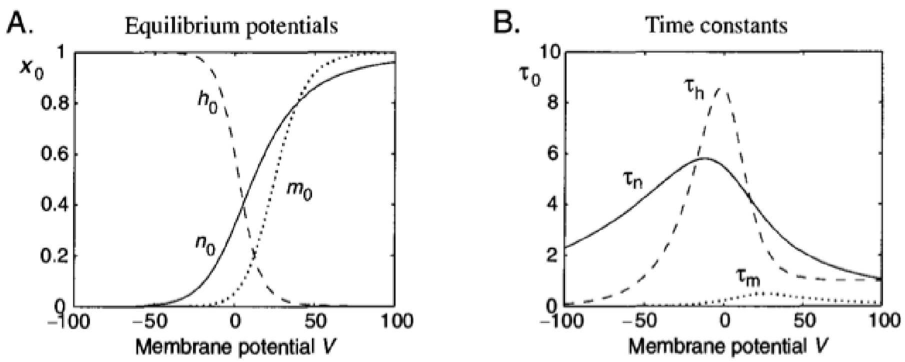
$$
\begin{aligned}
C\frac{\mathrm{d}V}{\mathrm{d}t} &= -g_{\mathrm{K}}n^4(V-E_{\mathrm{K}}) - g_{\mathrm{Na}}m^3h(V-E_{\mathrm{Na}}) - g_{\mathrm{L}}(V-E_{\mathrm{L}}) + I_{ext}(t) \\
\tau_{\mathrm{n}}(V)\frac{\mathrm{d}n}{\mathrm{d}t} &= -[n - n_0(V)] \\
\tau_{\mathrm{m}}(V)\frac{\mathrm{d}m}{\mathrm{d}t} &= -[m - m_0(V)] \\
\tau_{\mathrm{h}}(V)\frac{\mathrm{d}h}{\mathrm{d}t} &= -[h - h_0(V)] \\
\frac{dx}{dt} &= -\frac{1}{\tau_x(V)}[x - x_0(V)] \rightarrow x(t+\Delta t) = (1-\frac{\Delta t}{\tau_x})x(t) + \frac{\Delta t}{\tau_x}x_0
\end{aligned}
$$





[Ion channels resistance]

$$
\begin{aligned}
x(0) &= \frac{\alpha}{\alpha+\beta}, t_x = \alpha\beta, x \in= \{n, m, h\} \\
\alpha_n &= \frac{10-V}{100(e^{\frac{10-V}{10}}-1)}, \beta_n = 0.125e^{-\frac{V}{80}} \\
\alpha_m &= \frac{25-V}{10(e^{\frac{25-V}{10}}-1)}, \beta_m = 4e^{-\frac{V}{18}} \\
\alpha_h &= 0.07e^{\frac{V}{20}}, \beta_h = \frac{1}{e^{\frac{30-V}{10}}+1}
\end{aligned}
$$

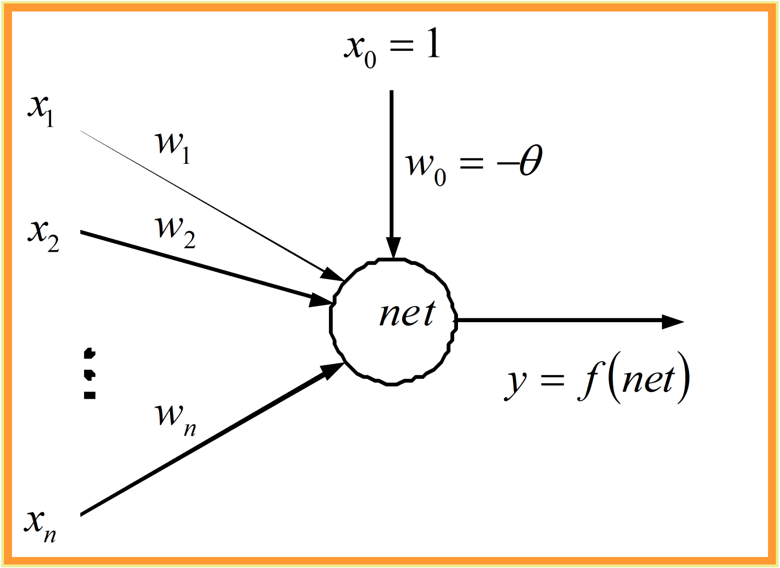


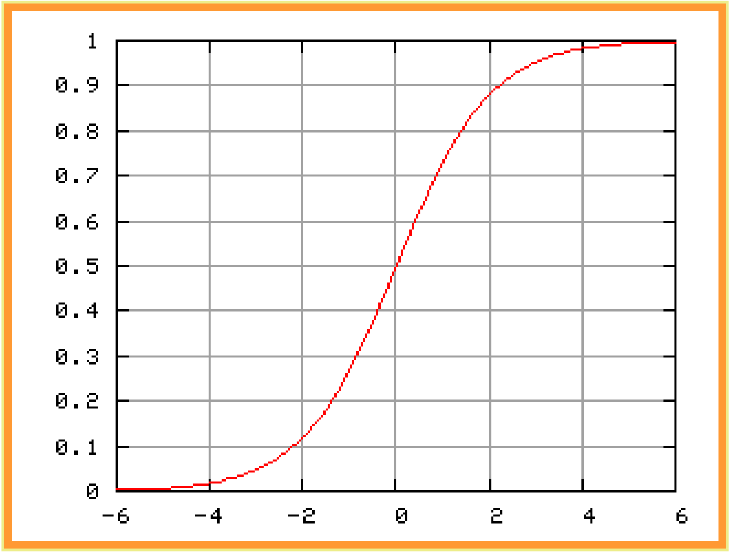
[Matlab implementation]

```
%% Integration of Hodgkin--Huxley equations with Euler method
 clear; figure;%clf;
%% Setting parameters
 % Maximal conductances (in units of mS/cm^2); 1=K, 2=Na, 3=R
 g(1)=36; g(2)=120; g(3)=0.3;
 % Battery voltage ( in mV); 1=n, 2=m, 3=h
 E(1)=-12; E(2)=115; E(3)=10.613;
 % Initialization of some variables
 I_ext=0; V=-10; x=zeros(1,3); x(3)=1; t_rec=0;
 % Time step for integration
   dt=0.01;
%% Integration with Euler method
 for t=-30:dt:500
    if t==10; I_ext=6; end  % turns external current on at t=10
    if t==400; I_ext=0; end   % turns external current off at t=40
  % alpha functions used by Hodgkin-and Huxley
    Alpha(1)=(10-V)/(100*(exp((10-V)/10)-1));
    Alpha(2)=(25-V)/(10*(exp((25-V)/10)-1));
    Alpha(3)=0.07*exp(-V/20);
  % beta functions used by Hodgkin-and Huxley
    Beta(1)=0.125*exp(-V/80);
    Beta(2)=4*exp(-V/18);
    Beta(3)=1/(exp((30-V)/10)+1);
  % tau_x and x_0 (x=1,2,3) are defined with alpha and beta
    tau=1./(Alpha+Beta);
    x_0=Alpha.*tau;
  % leaky integration with Euler method
    x=(1-dt./tau).*x+dt./tau.*x_0;         % x is m,n,h
  % calculate actual conductances g with given n, m, h
    gnmh(1)=g(1)*x(1)^4;
    gnmh(2)=g(2)*x(2)^3*x(3);
    gnmh(3)=g(3);
  % Ohm's law
    I=gnmh.*(V-E);
  % update voltage of membrane
    V=V+dt*(I_ext-sum(I));
  % record some variables for plotting after equilibration
    if t>=0;
         t_rec=t_rec+1;
         x_plot(t_rec)=t;
         y_plot(t_rec)=V;
    end
```

Neuron definition

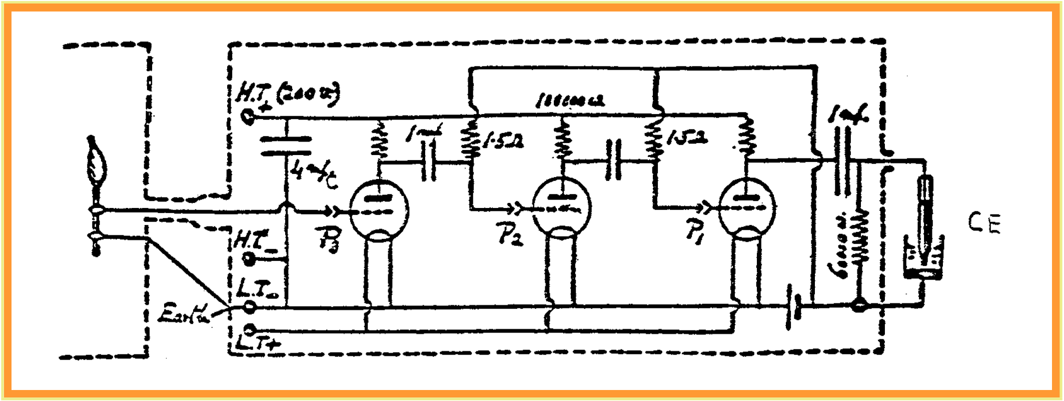
- Neuron is basic computational unit
- Inputs x_i are weighted by ω_i
- $net = \sum_{i=1}^{n} x_i \omega_i + w_0 = \sum_{i=0}^{n} \vec{w}^t \vec{x}$
- We indtoduce non-linearity to nrural nets $y = f(net)$, e.g. sigmoid fce: $\tanh$ or logistic fce $y = f(net) = \frac{1}{1+\exp^{-\lambda * net}}$

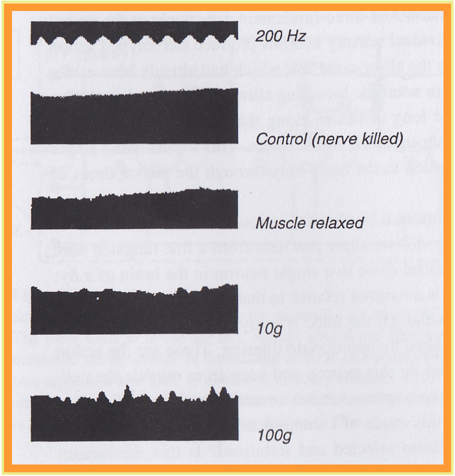


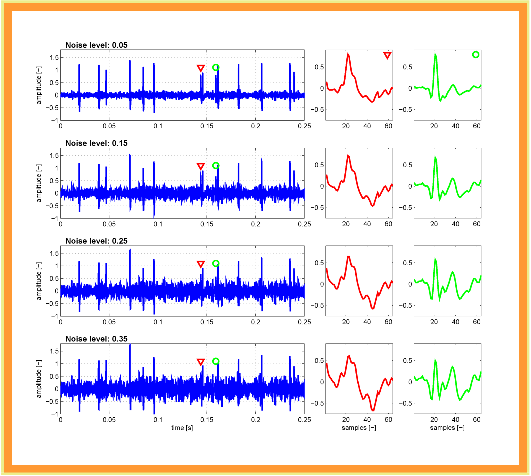


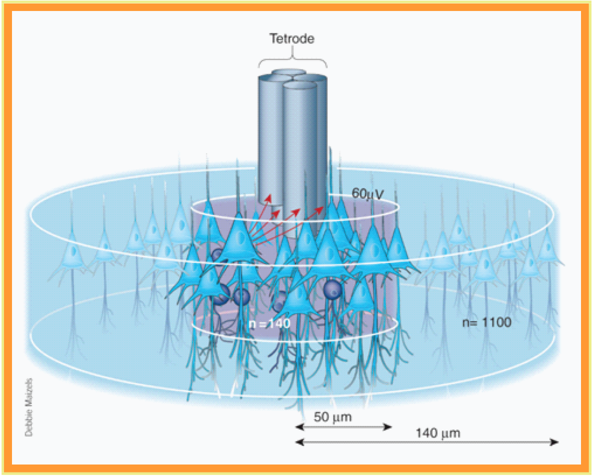
Physiology

- Nobel prize for medicine - year 1932)
  http://nobelprize.org/nobel_prizes/medicine/laureates/1932/adrian-bio.html#



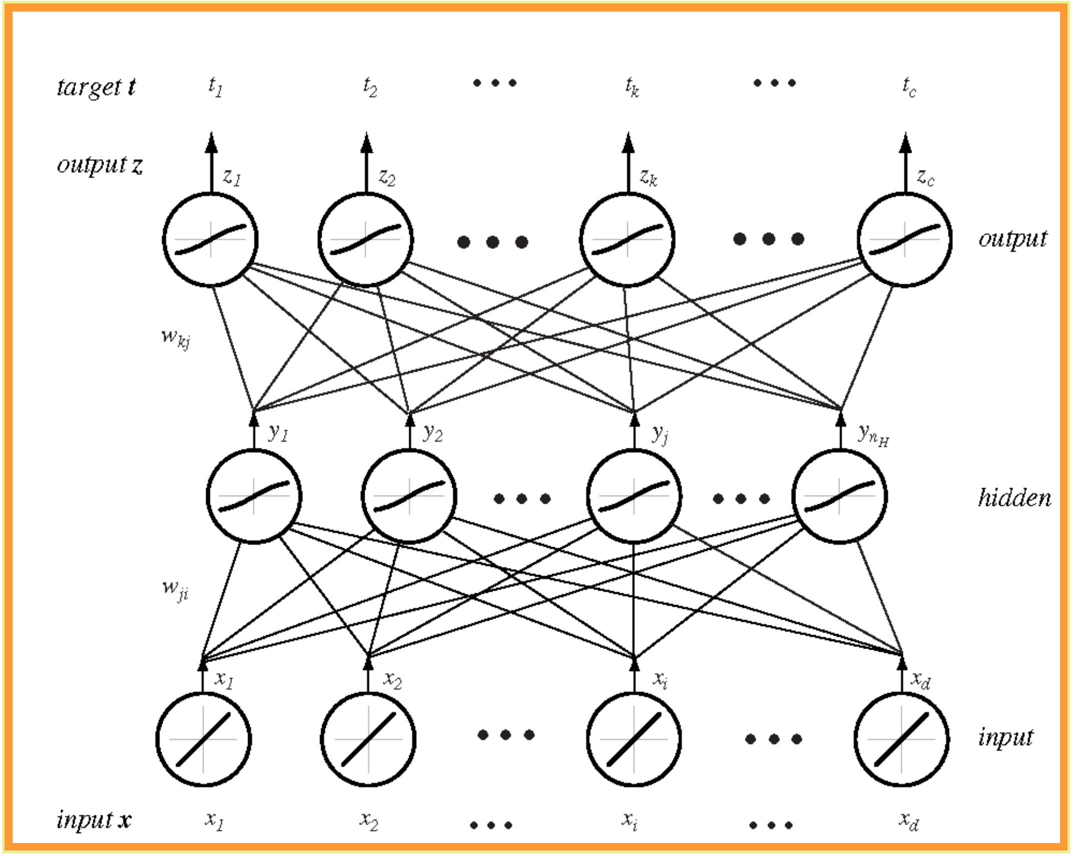






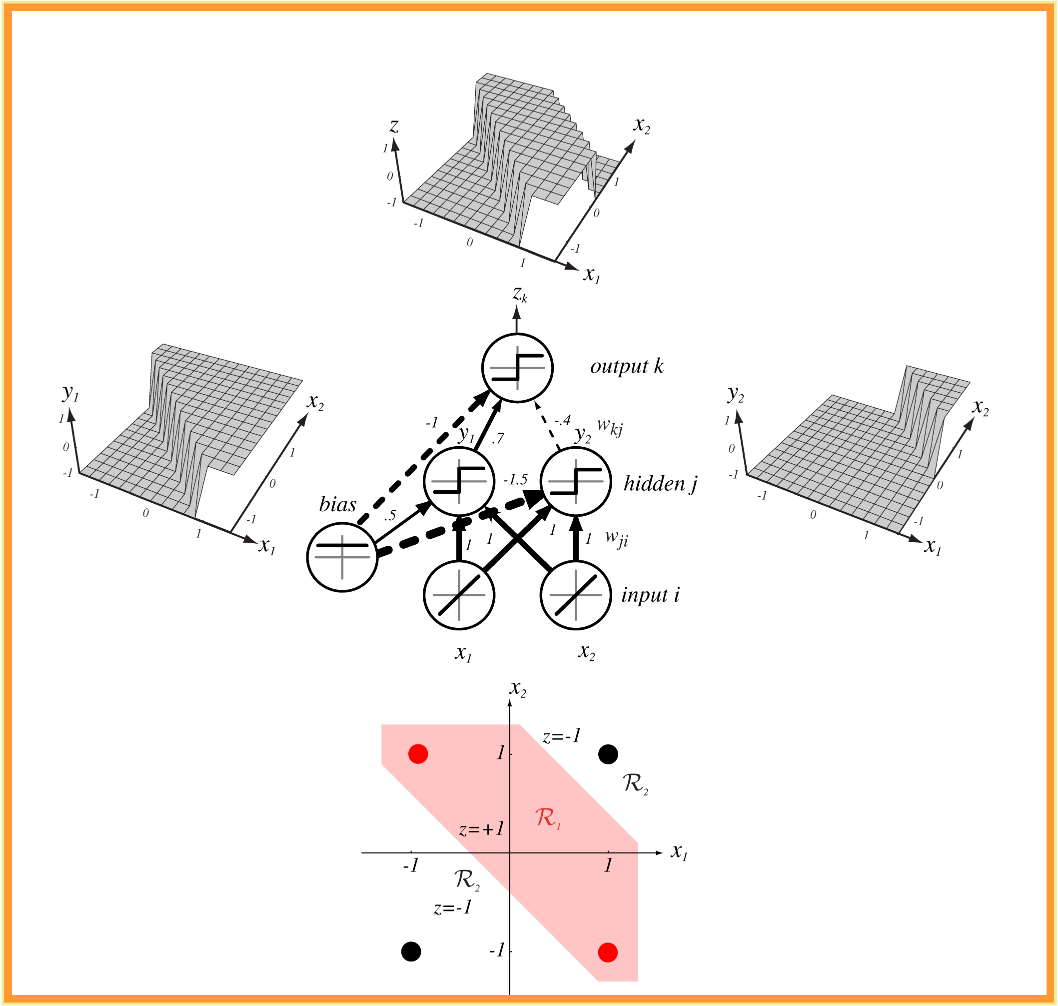
3-layers neural net($d - n_H - c$)

- The first layer is input layer, activation fce is linear, number of neurons equal to dimension of input vector $1 \dots d$
- The second layer is hidden layer, arbitrary number of neurons, $1 \dots n_H$
- The third layer is output layer, number of neurons equal to number of classes, $1 \dots c$



Example 3-layer neural net - XOR problem

- $0 \bigoplus 0 = 0$, $1 \bigoplus 1 = 0$, $1 \bigoplus 0 = 1$, $0 \bigoplus 1 = 1$
- $-1 \bigoplus -1 = -1$, $1 \bigoplus 1 = -1$, $1 \bigoplus -1 = 1$, $-1 \bigoplus 1 = 1$



XOR problem solution

- Hidden neuron decision boundary y_1

$$x_1 + x_2 + 0,5 = 0 \begin{cases} \geq 0 & \text{if } y_1 = +1 \\ < 0 & \text{if } y_1 = -1 \end{cases}$$

- Hidden neuron decision boundary y_2

$$x_1 + x_2 - 1.5 = 0 \begin{cases} \geq 0 & \text{if } y_2 = +1 \\ < 0 & \text{if } y_2 = -1 \end{cases}$$

- Neuron in output layer z

$$0.7y_1 - 0.4y_2 - 1 = 0 \begin{cases} \geq 0 & \text{if } z = +1 \\ < 0 & \text{if } z = -1 \end{cases}$$

Neuron activation

- Activation net_j of neuron in the hidden layer $net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} = \vec{w_j}^t \vec{x}$
- i indexes output layer, j hidden layer, w_{ji} is neuron weight in j hidden layer, which is connected to input neuron i (synapse).
- Neuron output in hidden layer $y_j = f(net_j)$
- XOR problem

$$f(net) = sgn(net) \begin{cases} 1 & \text{net } \geq 0 \\ -1 & \text{net } < 0 \end{cases}$$

- fce $f(.)$ is call activation fce.
- Similarly, activation fce net_k of neuron in output layer is $net_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} = \vec{w_k}^t \vec{y}$
- k index neuron in output layer, n_H is number of hidden neurons
- Neuron output in output layer $z_k = f(net_k)$
- In case of c classes, net computes c discrimination fces $z_k = g_k(\vec{x})$ and classifies input $\vec{x}$ according to biggest discrimination fce $g_k(\vec{x}) \qquad \forall k = 1, \ldots c$

Forward operation

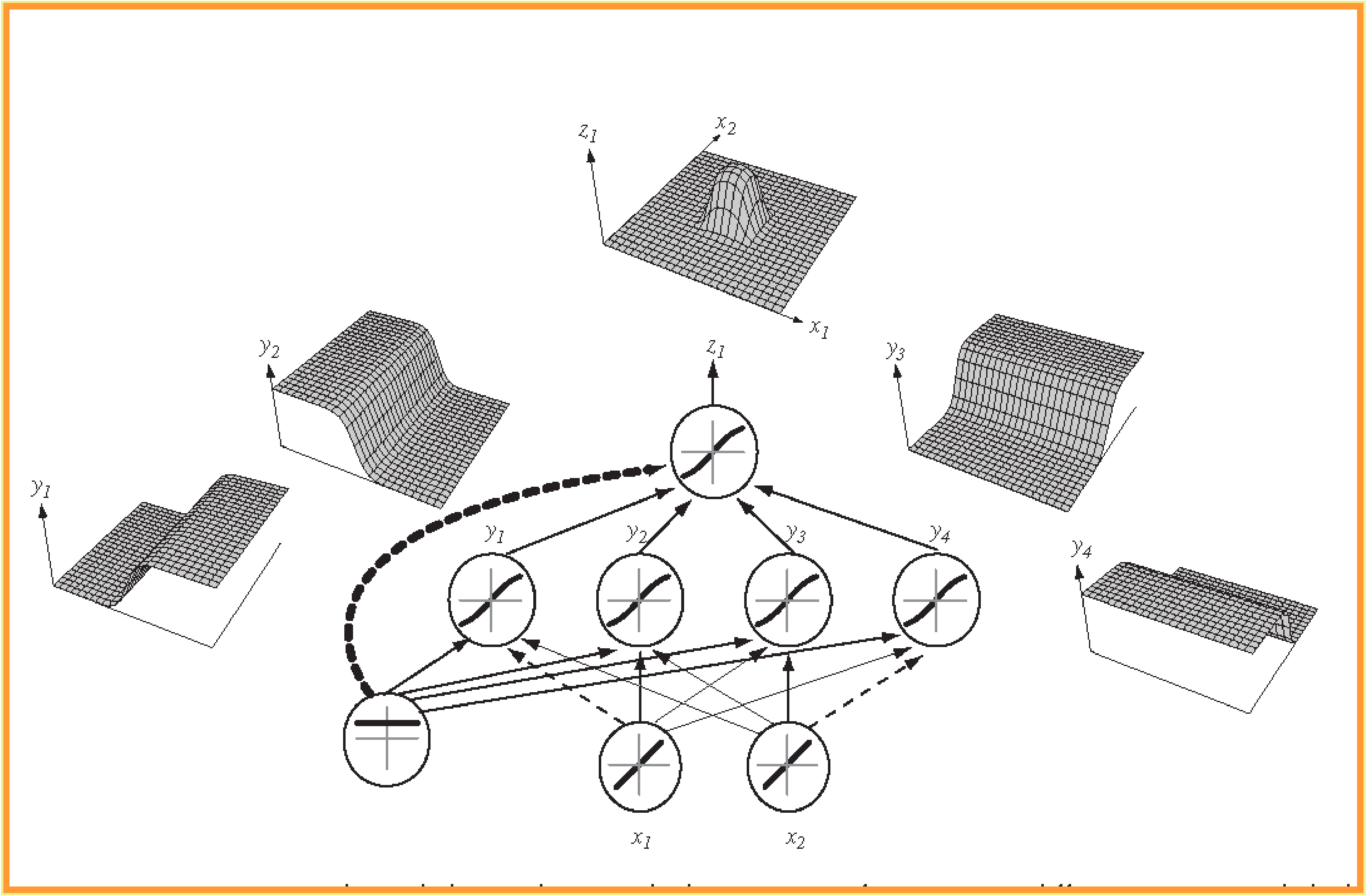
- Net output

$$g_k(\vec{x}) = z_k = f(\sum_{j=1}^{n_H} w_{kj} f(\sum_{i=1}^{d} w_{ji} x_i + w_{j0}) + w_{k0}) \qquad \forall k = 1 \dots c$$

- Hidden layer enables realization of complicated non-linear fces
- Each neuron can have its own activation fce
- We suppose that we have only ONE type of activation fce
- **QUESTION: Can 3-forward layer approximate any non-linear function?**
- **ANSWER: YES- thanks to A.Kolmogorov**
  Any continuous fce can be implemented by 3-layes net under assumption of sufficient number of n_H hidden neurons,suitable non-linearities and weights w.

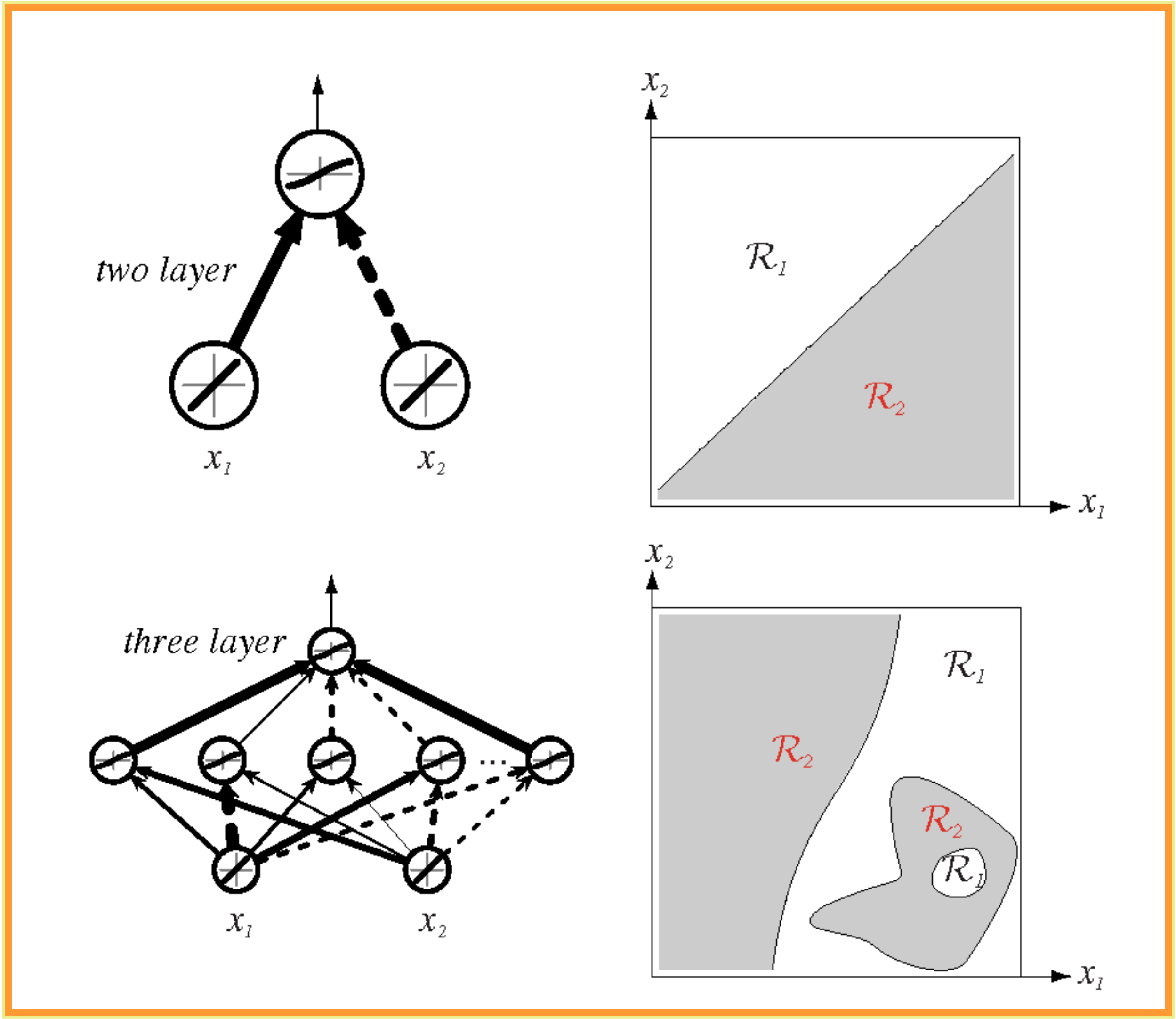
Non-linear fce approximation

- Fourier transform ANALOGY



Example of decision surface

- Comparision of 2-layer and 3-layer net



Andrej Kolmogorov

- He constructed "perpetuum mobile"in high school, his teacher could not discover the trick
- First he studied history in Moscow university
- He published the first scientific work on realities in Novgorod area during 15. a 16. centurary
- The biggest contribution in probability field

How can we learn the net ????

- Our goal is to set weights based on training data and desired output t_k
- We devise the method for error back propagation
- Le's t_k is k real output and z_k is output calculated , where $k = 1, \ldots, c$. We define error as

$$J(\vec{w}) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2} \parallel \vec{t} - \vec{z} \parallel$$

- Back-propagation algorithm is based on gradient approach (see percepton). Weight are initialized and changed according to steepest direction of error reduction

$$\triangle\vec{w} = -\eta\frac{\partial J}{\partial \vec{w}}$$

- η is learning parameter controlling relative weight change

$$\vec{w}(m+1) = \vec{w}(m) + \triangle\vec{w}$$

- where m s m-th template $(\vec{x}_m, t_m)$

Deduction

- Weight error (hidden-output)

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = \delta_k \frac{\partial net_k}{\partial w_{kj}}$$

- Hence $net_k = \sum_{j=0}^{n_H} y_j w_{kj} = \vec{w_k}^t \vec{y}$, thus $\frac{\partial net_k}{\partial w_{kj}} = y_j$
- where sensitivity of k-th neuron is defined as $\delta_k = -\frac{\partial J}{\partial net_k}$ and describes, how the total error changes with with activation fce net_k, $\frac{\partial z_k}{\partial net_k} = f^{'}(net_k)$

$$\delta_k = -\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k} = (t_k - z_k) f^{'}(net_k)$$

- Weights (hidden-output) are updated as

$$\boxed{\triangle w_{kj} = \eta \delta_k y_j = \eta (t_k - z_k) f^{'}(net_k) y_j}$$

- Weight error (input-hidden)

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}$$

- Hence

$$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right] = -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j} =$$

$$-\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j} = -\sum_{k=1}^{c}(t_k - z_k)f'(net_k)w_{kj}$$

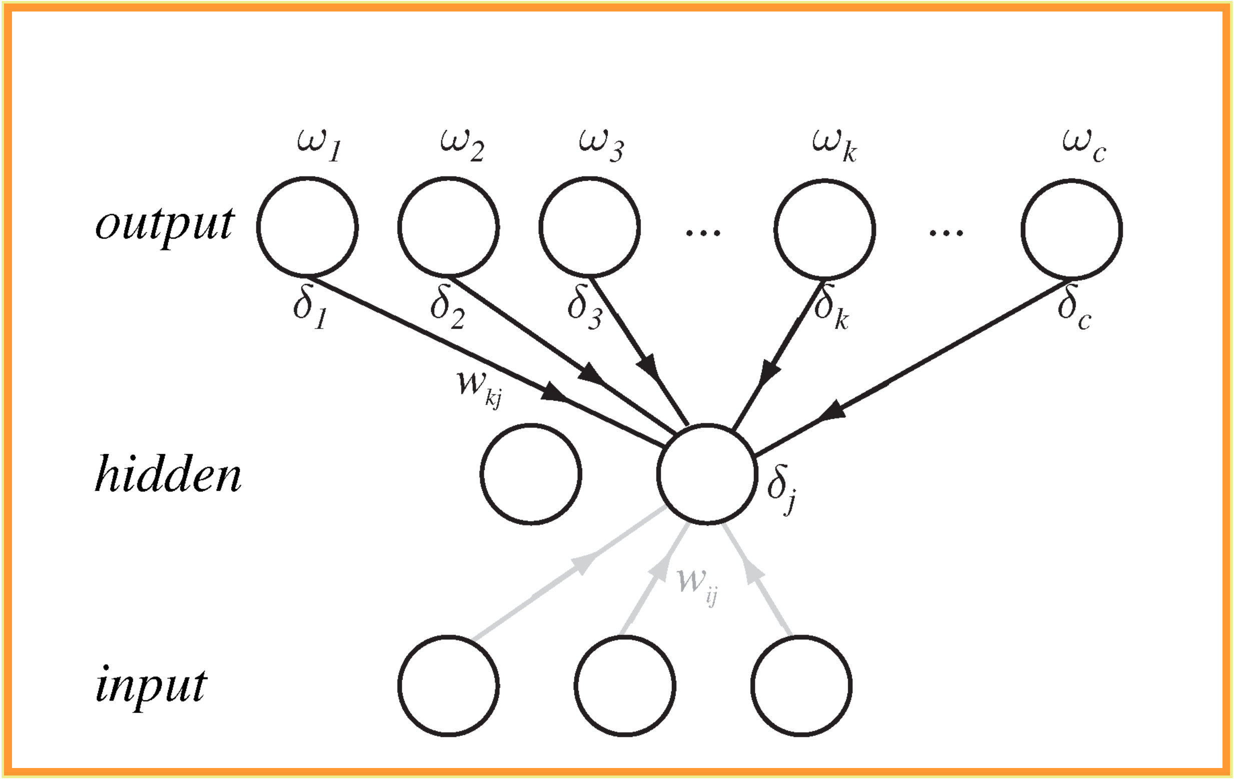
- So $\frac{\partial net_k}{\partial y_j} = w_{kj}$, because $net_k = \sum_{j=0}^{n_H} y_j w_{kj} = \vec{w_k}^t\vec{y}$
- Thus $\frac{\partial J}{\partial y_j} = -\sum_{k=1}^{c}\delta_k w_{kj}$, because $\delta_k = (t_k - z_k)f'(net_k)$
- Let's define $\frac{\partial y_j}{\partial net_j} = f'(net_j)$
- Let's $\frac{\partial net_j}{\partial w_{ji}} = x_i$,because $net_j = \sum_{i=0}^{d} x_i w_{ji} = \vec{w_j}^t\vec{x}$
- let's define sensitivity for hidden unit. Sensitivity is weighted sum of output sensitivities, multiplyed by activation fce of hidden neuron

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = \underbrace{-\sum_{k=1}^{c}\delta_k w_{kj} f'(net_j)}_{\delta_j} x_i$$

Why back-propagation ?

- The rule for weight update (input-hidden) is

$$\boxed{\triangle w_{ji} = \eta x_i \delta_j = \eta \sum (w_{kj}\delta_k) f^{'}(net_j) x_i}$$



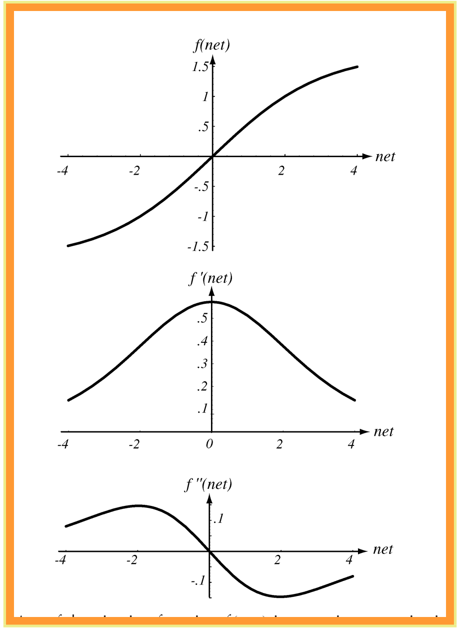
Pseudocode

- incremental learning - stochastic

begin initialize network topology (# hidden units), $\mathbf{w}$, criterion $\theta, \eta, m \leftarrow 0$
  do $m \leftarrow m + 1$
    $\mathbf{x}^m \leftarrow$ randomly chosen pattern
    $w_{ij} \leftarrow w_{ij} + \eta \delta_j x_i; \quad w_{jk} \leftarrow w_{jk} + \eta \delta_k y_j$
  until $\nabla J(\mathbf{w}) < \theta$
return $\mathbf{w}$
end

– Matlab implementation: *Backpropagation_Stochastic.m*. $\tanh$, $a = 1.716$, $b = \frac{2}{3}$, so $f'(0) \simeq 1$.

$$f(net) = a \tanh(b \star net) = \frac{2a}{1 + \exp^{b \star net}} - a$$



Batch learning

- We have n inputs $\vec{x_i}$, we express total error as

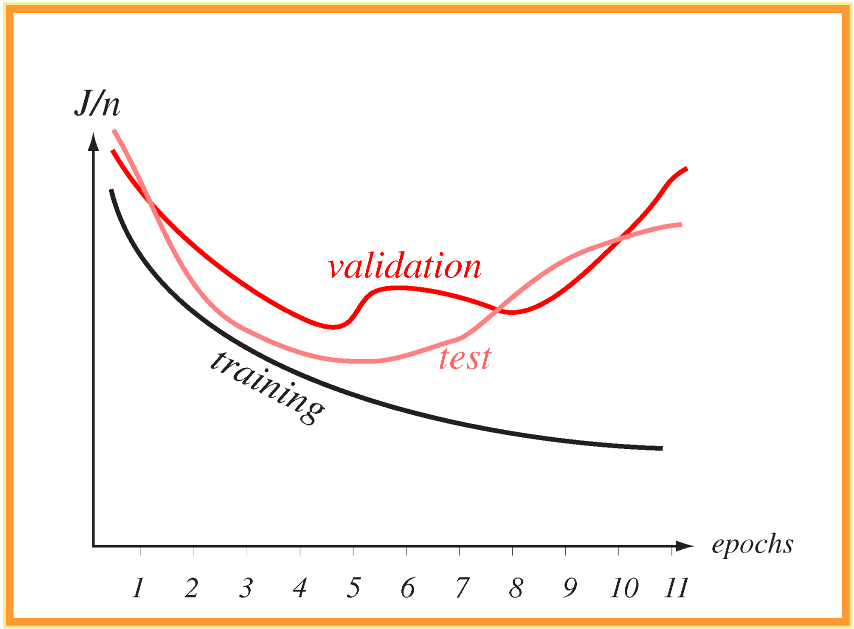
$$J = \sum_{p=1}^{n} J_p$$

- It is not necessary to select inputs one by one
- Epoch is one representation of all inputs , step 2: $r = r + 1$

1 **begin initialize** network topology (# hidden units), $\mathbf{w}$, criterion $\theta, \eta, r \leftarrow 0$
2   **do** $r \leftarrow r + 1$ (increment epoch)
3     $m \leftarrow 0;\ \Delta w_{ij} \leftarrow 0;\ \Delta w_{jk} \leftarrow 0$
4   **do** $m \leftarrow m + 1$
5       $\mathbf{x}^m \leftarrow$ select pattern
6       $\Delta w_{ij} \leftarrow \Delta w_{ij} + \eta \delta_j x_i;\quad \Delta w_{jk} \leftarrow \Delta w_{jk} + \eta \delta_k y_j$
7   **until** $m = n$
8     $w_{ij} \leftarrow w_{ij} + \Delta w_{ij};\quad w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$
9   **until** $\nabla J(\mathbf{w}) < \theta$
10 **return** $\mathbf{w}$
11 **end**

Validation

- Error of training sert in monotonic-decreasing fce because of gradient algorithm optimization
- we devide data to training and validation set We use validation as stopping criteria (e.g. the first minimum)



- DEMO - Neural Network Toolbox v Matlabu
  `http://www.mathworks.com/products/neuralnet/`
- Data are from UCI Machine Learning Repository
  `http://mlearn.ics.uci.edu/MLRepository.html`