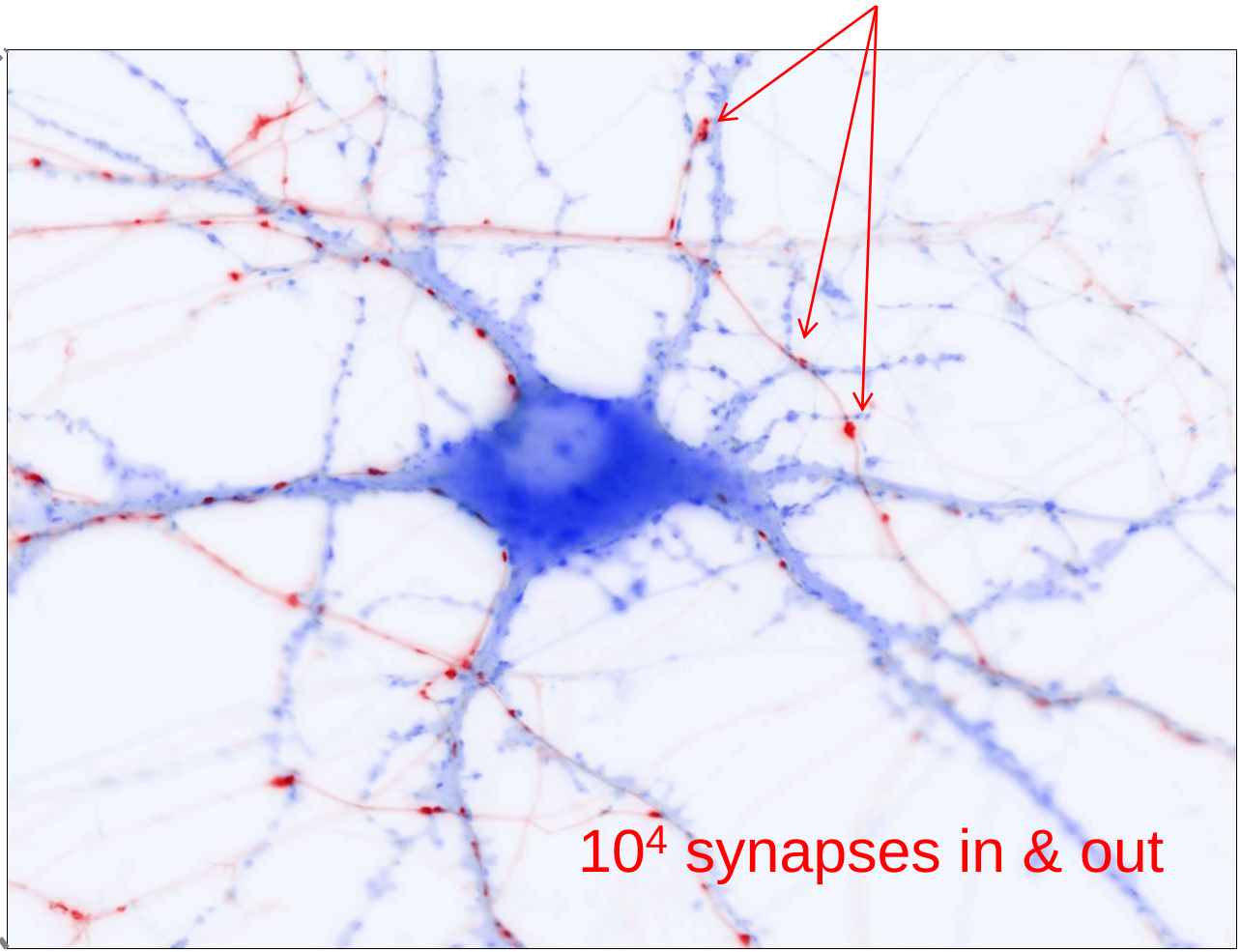
# Perceptron and the rule δ
# (Perceptrón a pravidlo δ)

*Lubica Benuskova*

Reading: AIMA 3rd ed., Chap. 18.6.3 – 18.6.4

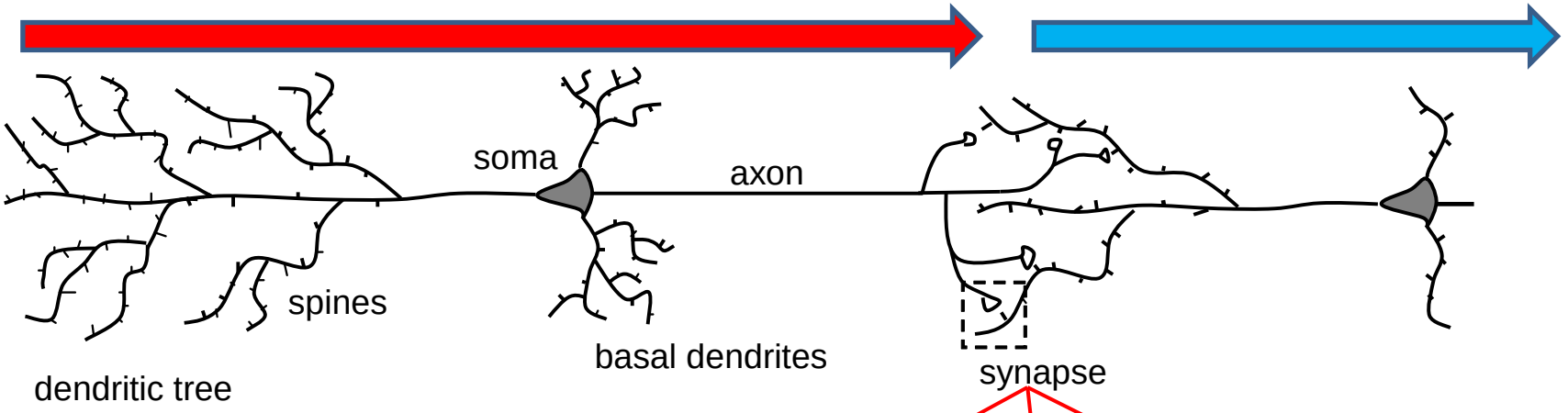
# Brain is comprised of networks of neurons connected and communicating via synapses

$86x10^9$ neurons

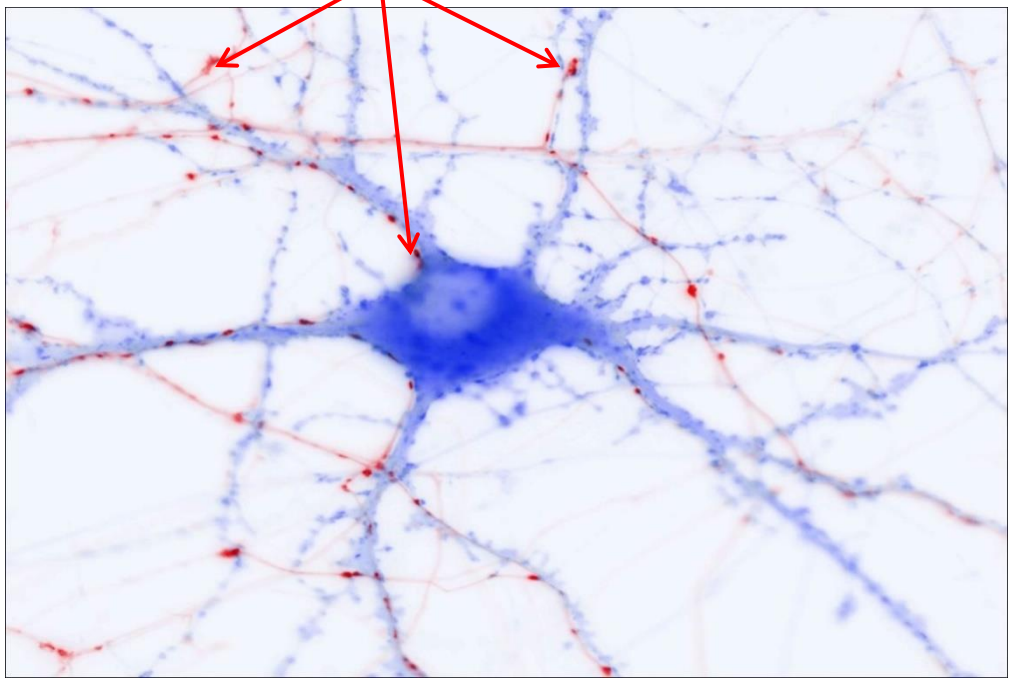


$10^4$ synapses in & out

# Presynaptic and postsynaptic: flow of information

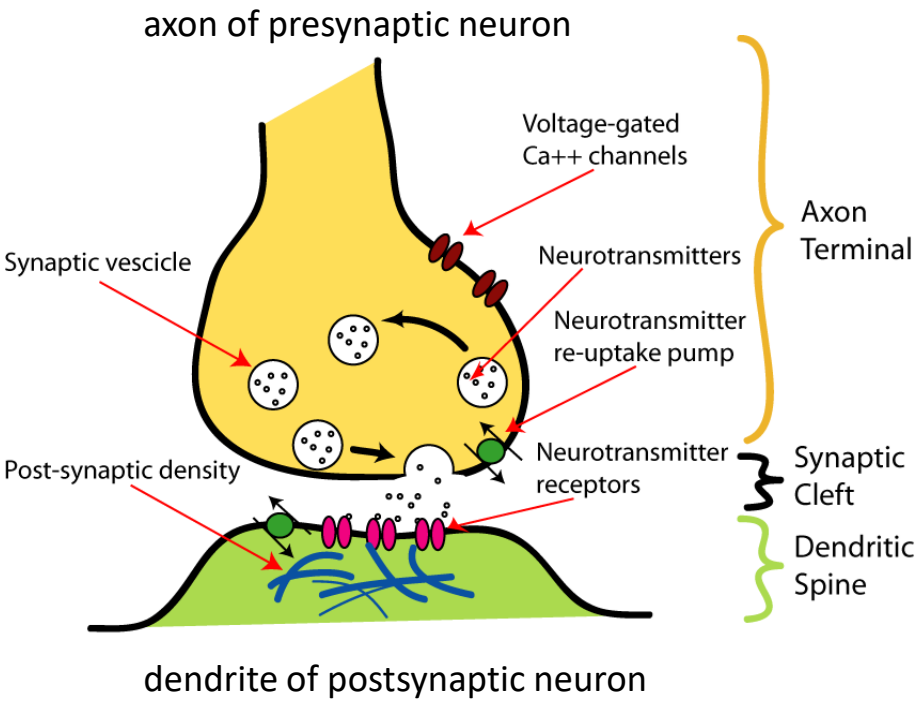


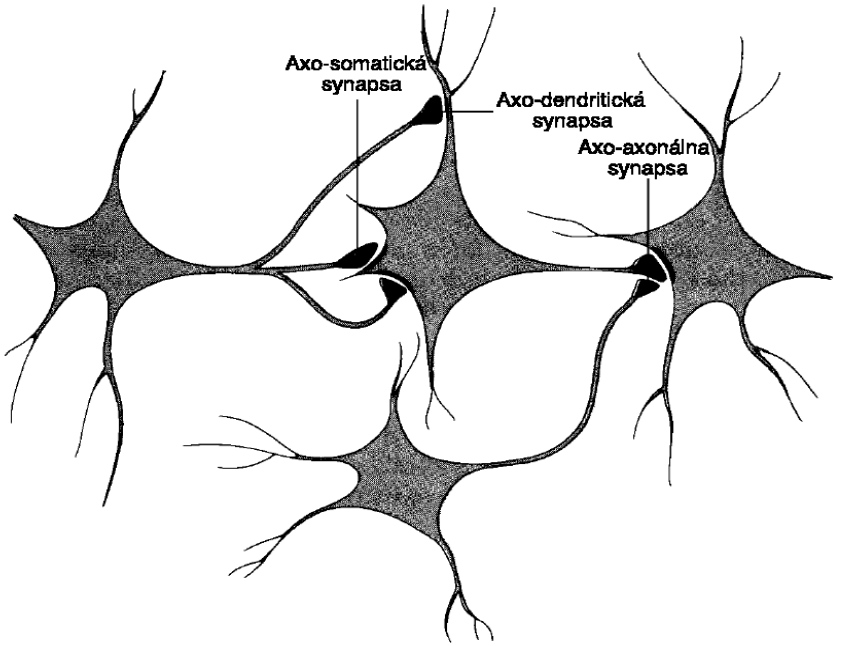

Axons of presynaptic neurons make synapses (little blobs) on the body and dendrites of the postsynaptic neuron



# Neurons in the networks communicate via synapses

- **Presynaptic electric** pulse (i.e. **spike**) arrives to terminal. This causes release of neurotransmitter molecules from vesicles.
- Neurotransmitter binds to receptors in the dendritic membrane.
- Ion channels associated with receptors open and allow influx of charged ions into the spine – evoking a **postsynaptic potential** at the synapse.

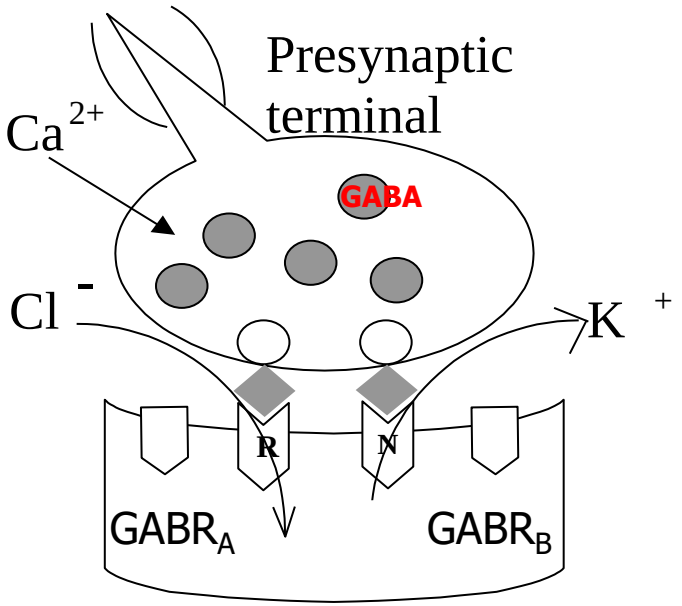


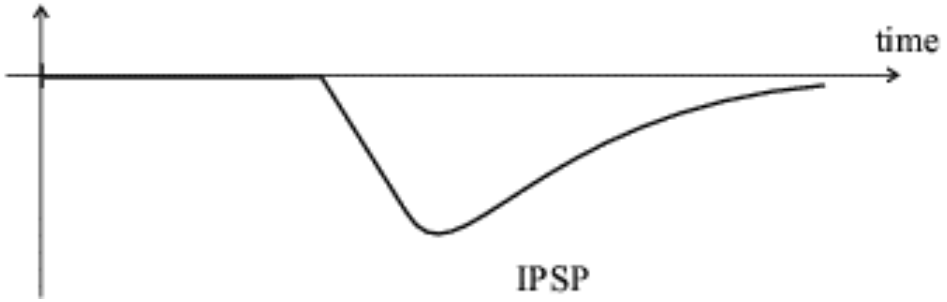



# Inhibitory synapses

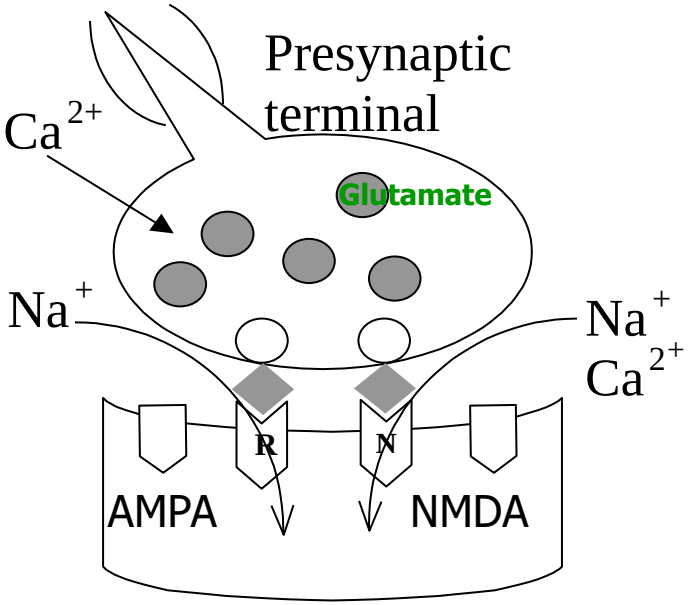
- Neutrotransmitter GABA, 2 types of postsynapstic receptors GABRA and GABRB, ion channels for $Cl^-$ and $K^+$, respectively.

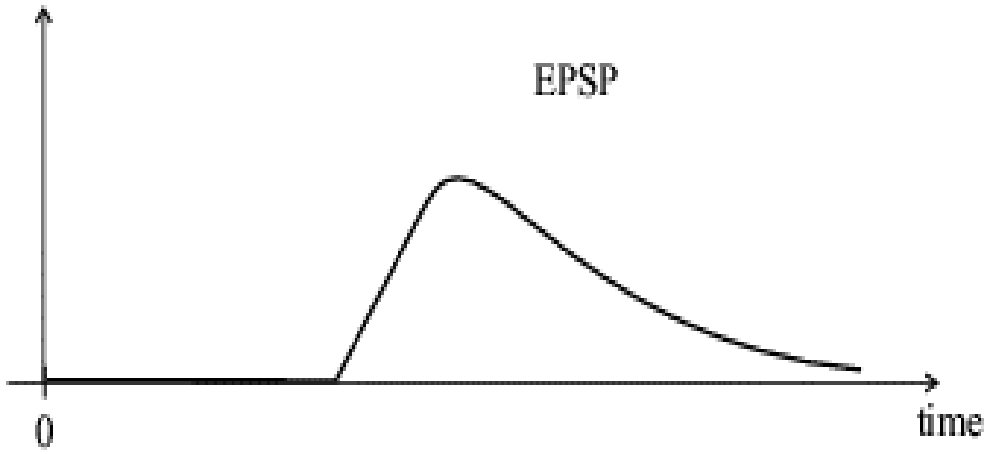
- When we measure an electric potential at the postsynaptic site, we see something like this: a negative deviation from the resting potential, which is called **inhibitory postsynaptic potential (IPSP).**



# Excitatory synapses



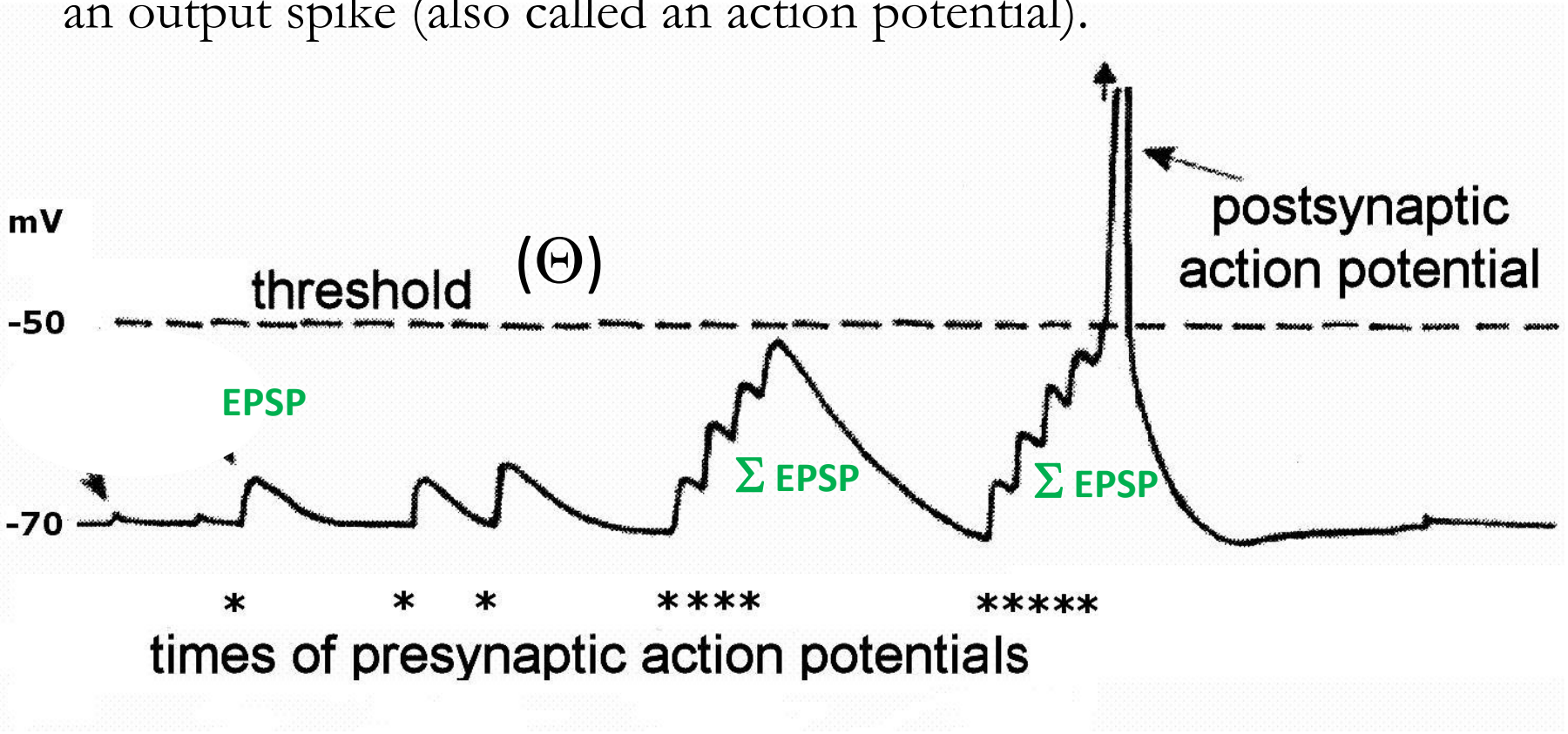




- Neutrotransmitter glutamate, 2 types of postsynaptic receptors AMPA and NMDA, ion channels for $Na^+$ and $Na^+$ and $Ca^{2+}$, respectively.

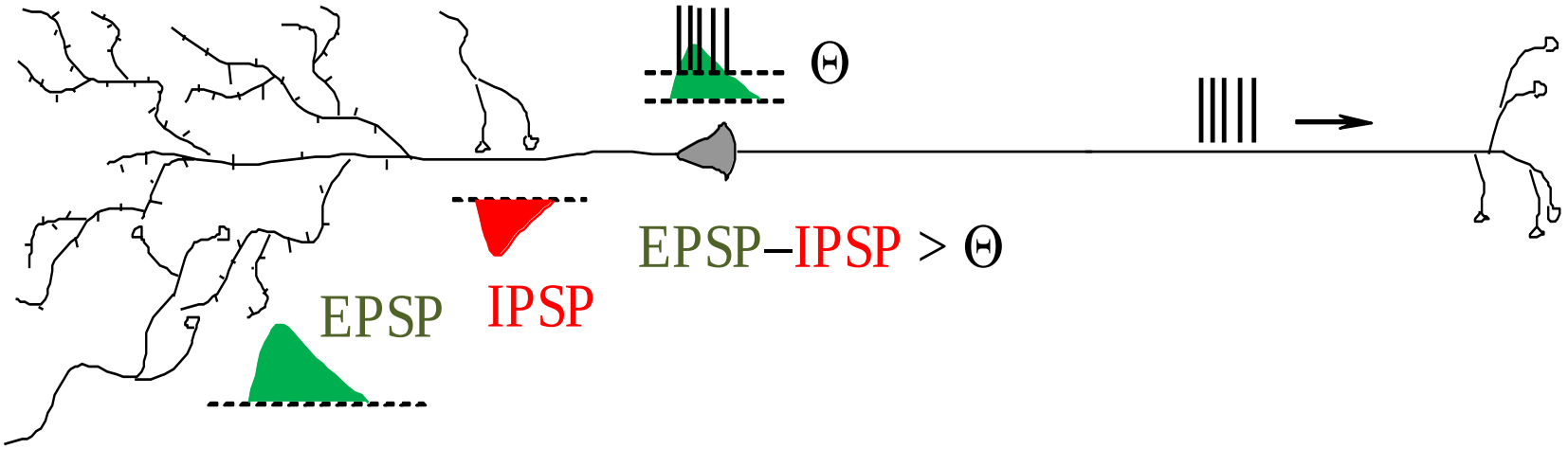
- When we measure an electric potential at the postsynaptic site, we see something like this: a positive deviation from the resting potential, which is called **excitatory postsynaptic potential (EPSP).**

# Summation of individual PSPs

- Each presynaptic electric pulse (spike) causes a single EPSP bump at the postsynaptic site. On arrival of many presynaptic spikes in close succession, these EPSPs sum on top of each other and if the cumulative sum crosses the firing threshold Θ, the neuron generates an output spike (also called an action potential).

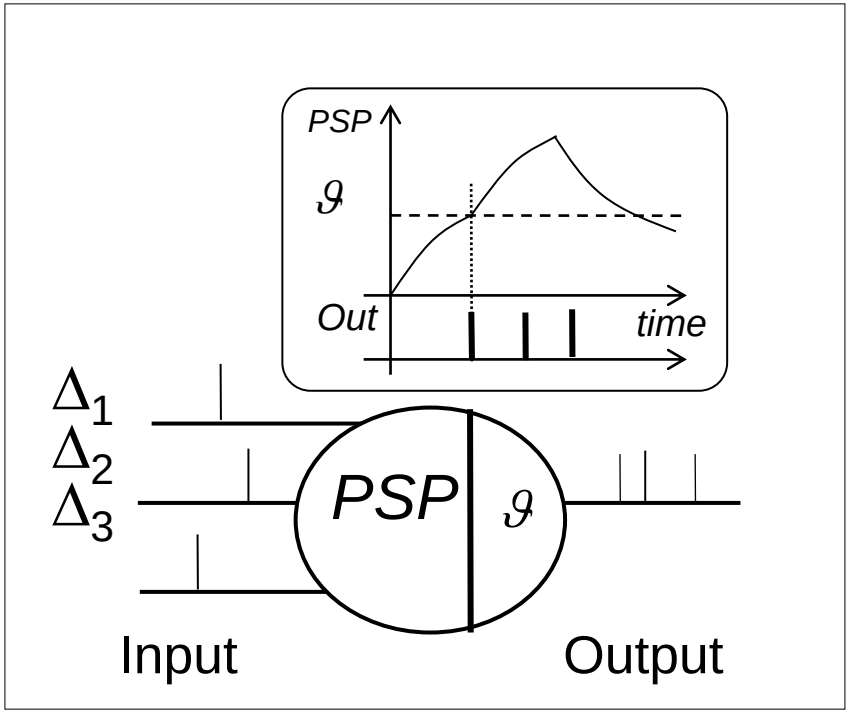


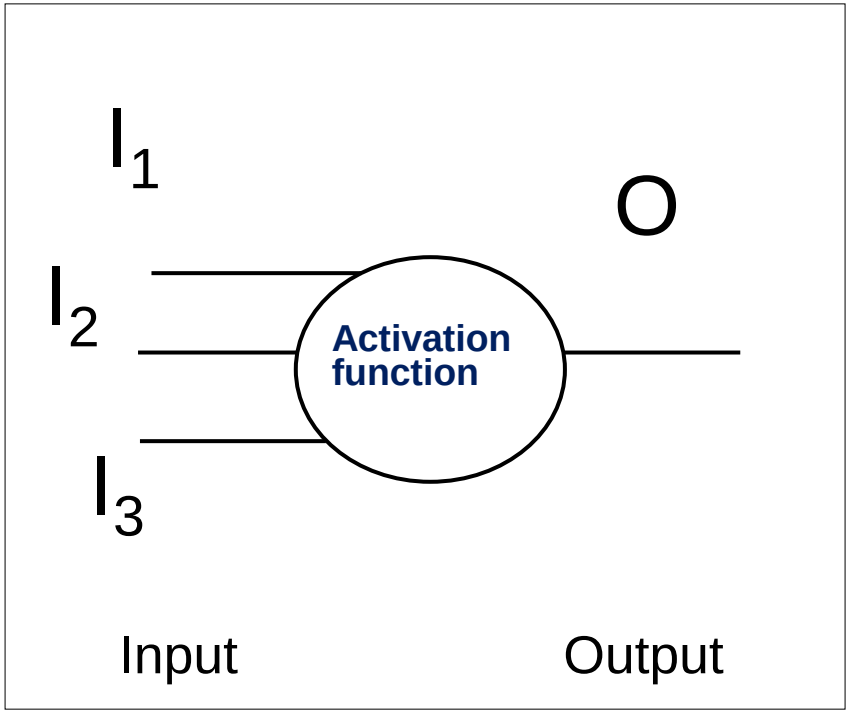
# Neuron: an analog-digital converter




- Since the inhibitory synapses can be activated as well, the total PSP= ΣEPSP–ΣIPSP. If PSP > Θ, neuron generates an output **spike train**.
- The output spike train has a certain **frequency.**
- Spike train propagates towards synapses, where it causes release of neurotransmitter.

# Spiking versus rate neuron models

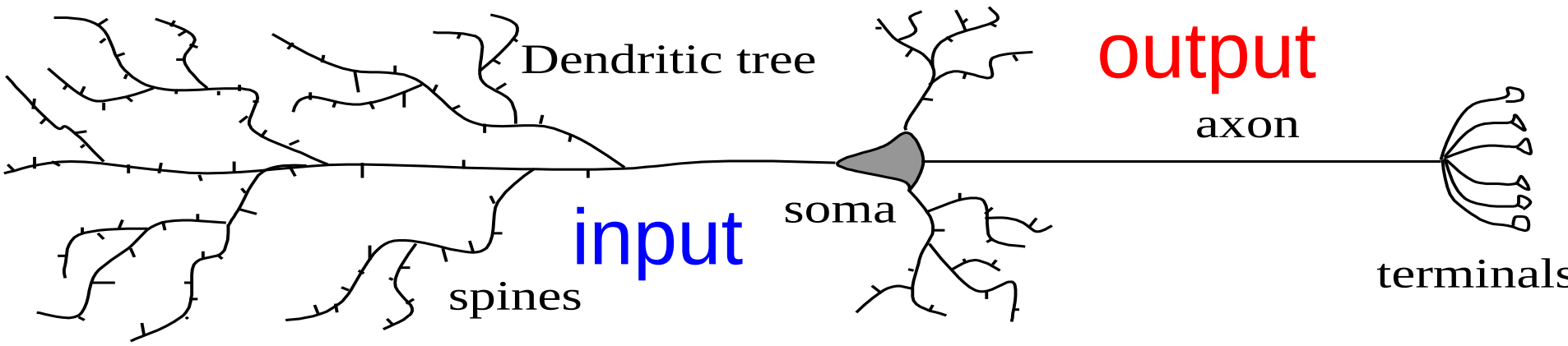


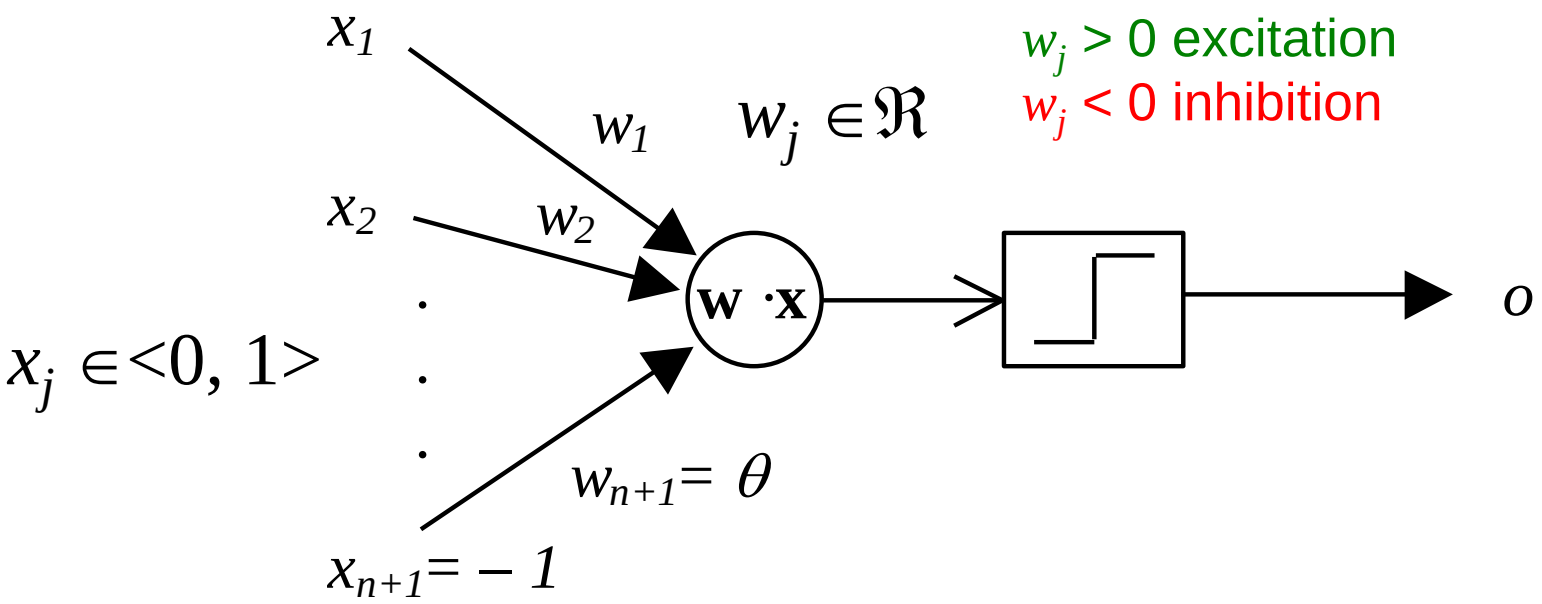




Spiking model: output depends on timing of input spikes

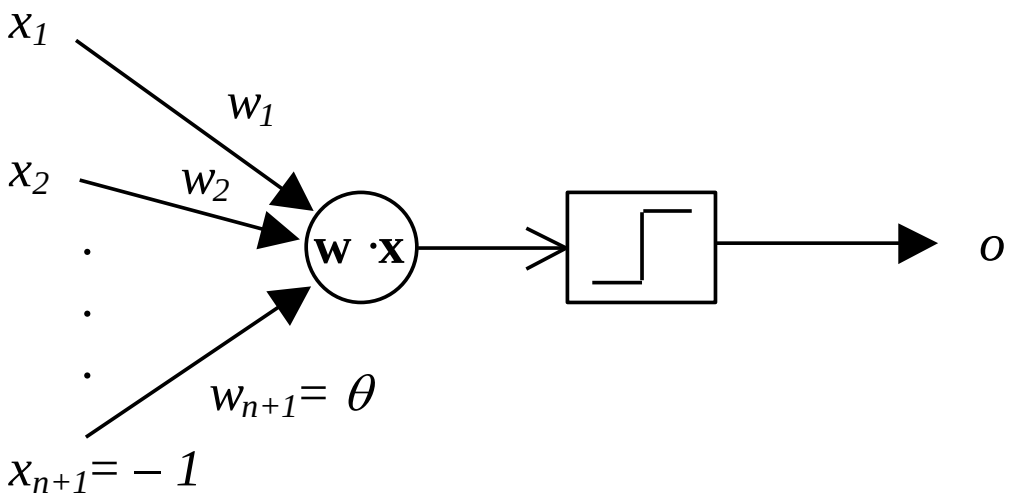
Rate model: output depends on the sum of input rates (frequencies)

# Perceptron - rate model of neuron



# Rosenblatt's binary perceptron



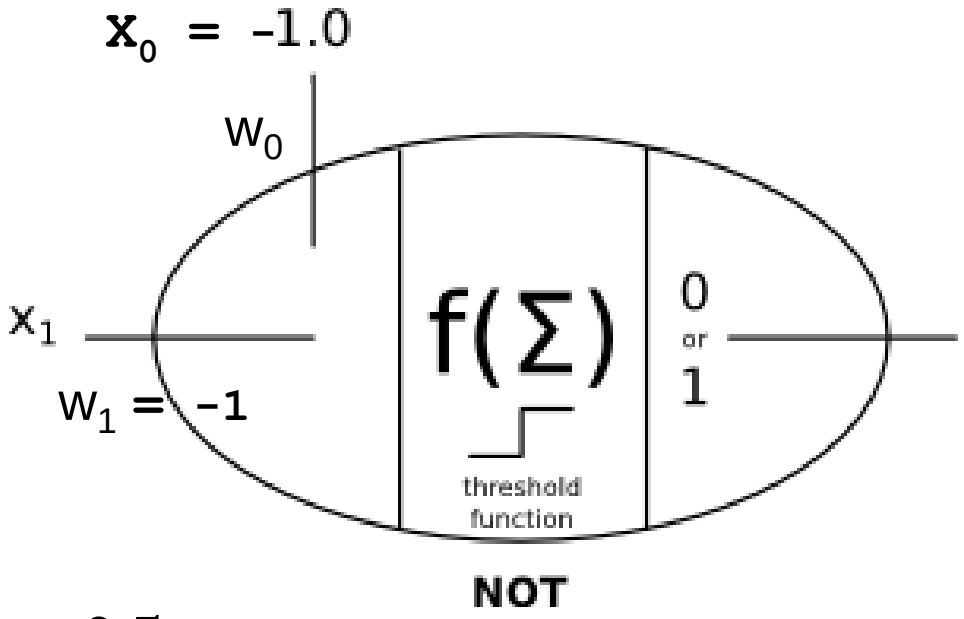

$$o = f(net) = f(\mathbf{w} \cdot \mathbf{x}) = f\left(\sum_{j=1}^{n+1} w_j x_j\right) = f\left(\sum_{j=1}^{n} w_j x_j - \theta\right)$$

$$f(net) = \begin{cases} +1 & if\ net \geq 0 \ \Leftrightarrow \ \sum_{j=1}^{n} w_j x_j \geq \theta \\ 0 & if\ net < 0 \ \Leftrightarrow \ \sum_{j=1}^{n} w_j x_j < \theta \end{cases}$$

# Binary perceptron and Boolean functions

- NOT (negation of input)

$$x_1 = \begin{cases} 0 \\ 1 \end{cases}$$



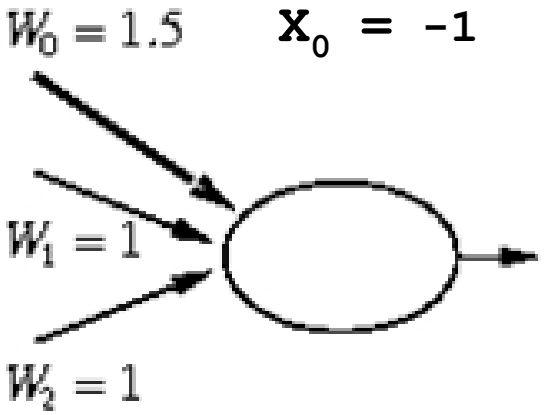

$$net = w_1 x_1 + w_0 x_0 = (-1)x_1 + 0.5$$

$$net = \begin{cases} (-1)(0) + 0.5 = +0.5 > 0 \quad \rightarrow \quad o = +1 \\ \\ (-1)(+1) + 0.5 = -0.5 < 0 \quad \rightarrow \quad o = 0 \end{cases}$$

# Binary perceptron – AND

- AND Boolean function

$$x_i = \begin{cases} 0 \\ 1 \end{cases}$$



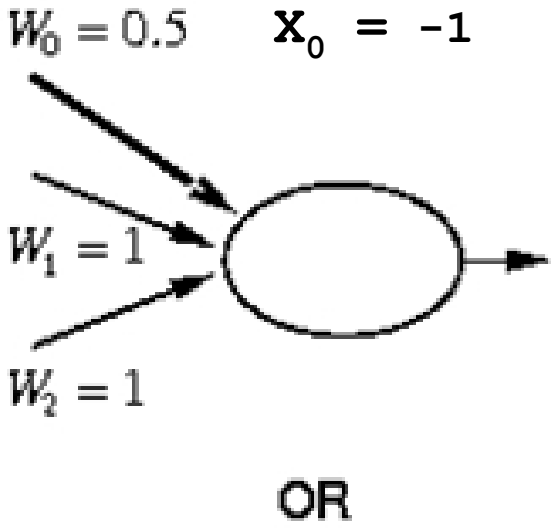

$$net = w_2 x_2 + w_1 x_1 + w_0 x_0$$

$$net = \begin{cases} +1 + 1 - 1.5 = +0.5 > 0 \rightarrow o = 1 \\ +1 + 0 - 1.5 = -0.5 < 0 \rightarrow o = 0 \\ +0 + 1 - 1.5 = -0.5 < 0 \rightarrow o = 0 \\ +0 + 0 - 1.5 = -1.5 < 0 \rightarrow o = 0 \end{cases}$$

# Binary perceptron – OR

- OR Boolean function

$$x_i = \begin{cases} 0 \\ 1 \end{cases}$$




$$net = w_2 x_2 + w_1 x_1 + w_0 x_0$$

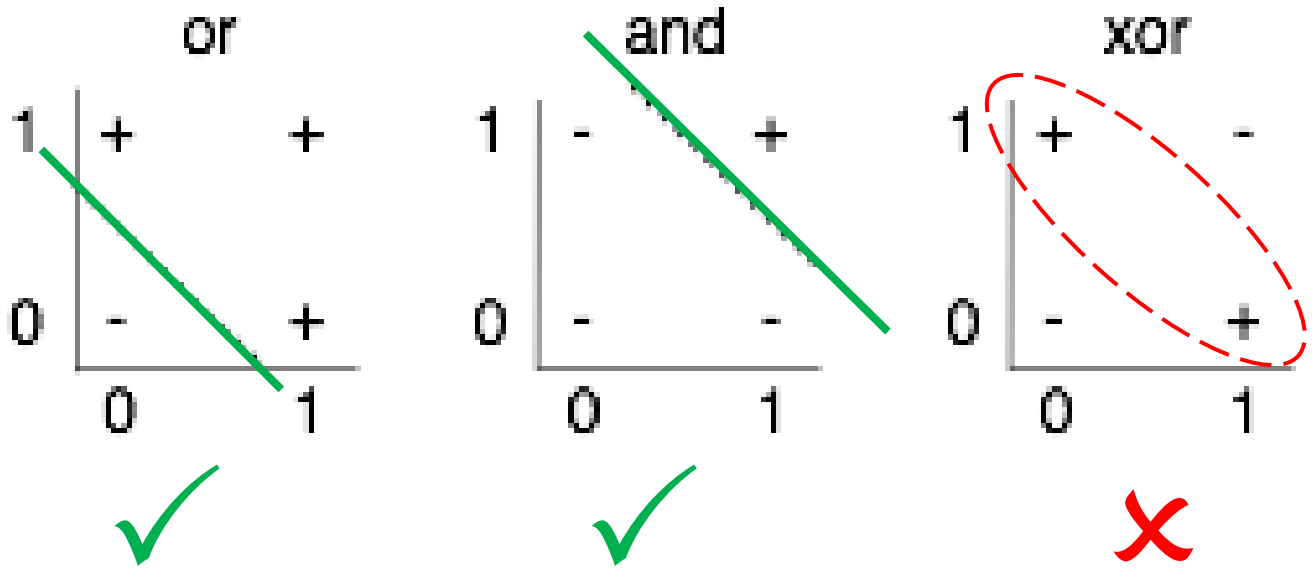
$$net = \begin{cases} +1 + 1 - 0.5 = 1.5 > 0 & \rightarrow o = +1 \\ +1 + 0 - 0.5 = 0.5 > 0 & \rightarrow o = +1 \\ +0 + 1 - 0.5 = 0.5 > 0 & \rightarrow o = +1 \\ +0 + 0 - 0.5 = -0.5 < 0 & \rightarrow o = 0 \end{cases}$$

# Linear classification boundary

- Perceptron output is the equation of *hyperplane* in $n$-dimensional space with axes $x_i$ and coefficients $w_i$

$$\sum_{i=1}^{n} w_i x_i + w_0 = 0$$

- Binary perceptron can solve only the so-called **linearly separable problems** with linear separating boundaries:



# Learning and synaptic plasticity

- Organism's ability to store, retain, and subsequently recall information is called a **memory**.
- The process of acquisition of memories is called ***learning***.
- It is widely accepted that the brain mechanism of learning are changes of synaptic weights / strengths in the relevant brain areas (i.e. cortex, hippocampus, cerebellum, etc.).
- Ability to change the strength of a synapse is called synaptic plasticity.
- **Synaptic plasticity** underlies the brain plasticity, which is a lifelong ability of the brain to reorganize neural circuits based on new experience.

# Perceptron learning

- The goal of a learning algorithm is to automatically find the values of **weights** to fit any training set of examples.

- The task can be any linear problem, not just Boolean functions. In fact Boolean functions are just one particular example of tasks for perceptron.

- For a single perceptron *initialized with small random* weights, upon presentation of each example a new weight array $\mathbf{w} = \{w_i\}$, is calculated to move the output of perceptron closer to the desired (target) output.

# Training set and error

- Let the training set be

$$A_{train} = \{(\mathbf{x}^1, d^1)(\mathbf{x}^2, d^2)...(\mathbf{x}^p, d^p)....(\mathbf{x}^P, d^P)\}$$

  - $\mathbf{x}^p$ is the input array or vector (also called a pattern)
  - $d^p$ is the target or desired output, being +1 for one class of inputs and 0 for the other class of inputs

- Error function = sum of squared errors:
  - Where

$$E = \frac{1}{2}\sum_{p=0}^{P}(d^p - o^p)^2$$

$$o = f(\mathbf{wx}) = \begin{cases} +1 & if \quad \mathbf{wx} \geq 0 \\ 0 & otherwise \end{cases}$$

# The $\delta$ (delta) rule

- The weights are updated due to each input pattern $\mathbf{x}^p$ as

$$w_j \leftarrow w_j + \alpha\ \delta\ x_j^p$$

  - **Delta is the error signal**, i.e. $\delta = (d^p - o^p)$
  - Constant $\alpha > 0$ is the learning speed

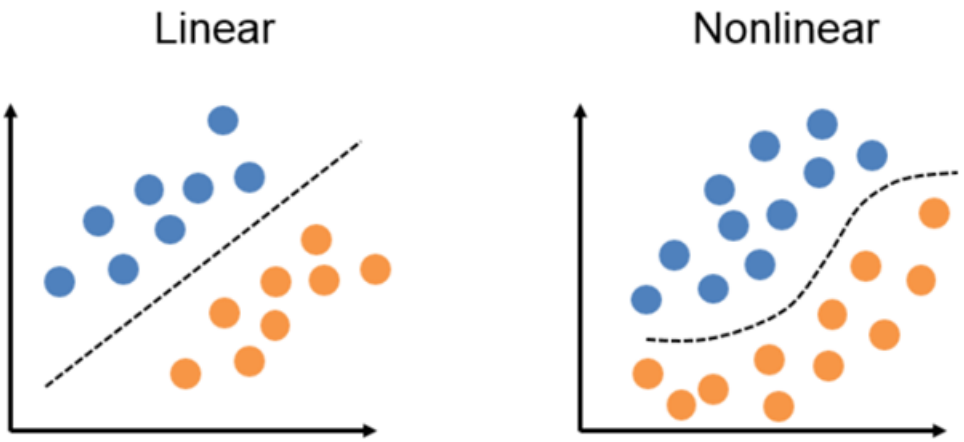
- If $x_i > 0$ and the error signal $\delta > 0$, then the weight is increased.

- If $x_j > 0$ & $\delta < 0$, then the weight is decreased.

```
Start with random initial weights in [-.5,.5]) and alpha = 0.5
do
{
    epoch++ // number of loops through the training set
    CORRECT = 0 // number of correctly classified inputs
    for p = 1 to p = P   // loop through all training samples
    {
        net = 0
        for j = 0 to j = N
        {
            net = net + w_j * x_j
        }
        if (net > 0) out = 1 else out  = 0
        if (out = = desired) CORECT++
        else
        {
          for j = 0 to N
         {
            w_j = w_j + alpha * (desired – out) * x_j
         }
        }
    }
while (CORRECT < P) //training stops when all training samples
are correctly classified
```

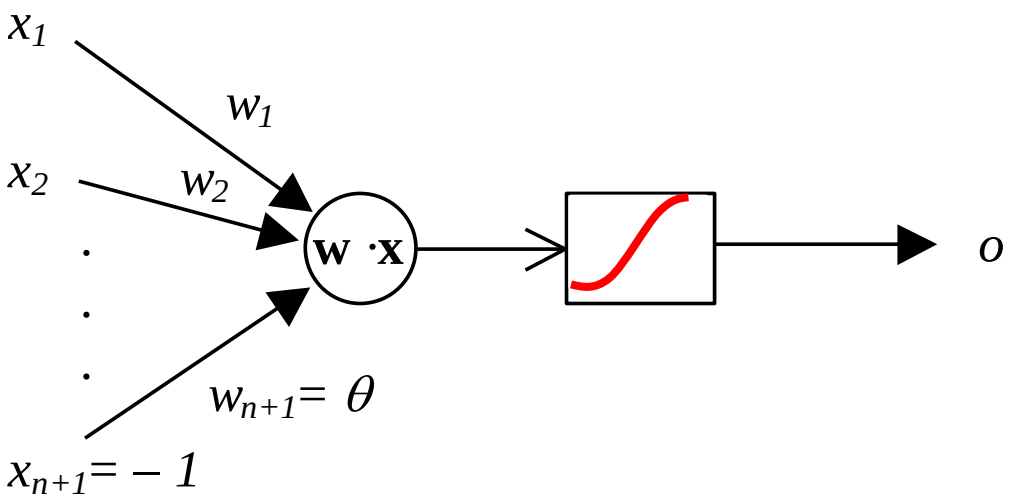
# Nonlinear classification boundary

- The dividing boundary between two (or more classes) of objects (inputs) can be nonlinear.



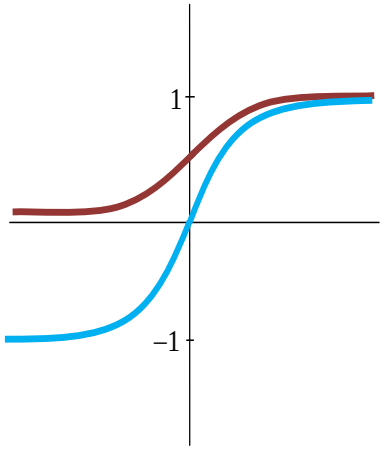

- For a particular problem, we can obtain the values of coefficients $w_i$ of the nonlinear function either analytically (using regression), or algorithmically using the method of gradient descent or some method of stochastic optimisation (e.g. simulated annealing or genetic algorithm).

# Continuous perceptron (sigmoid, tanh)




$$o = f(net) = f(\mathbf{w} \cdot \mathbf{x}) = f\left(\sum_{j=1}^{n+1} w_j x_j\right) = f\left(\sum_{j=1}^{n} w_j x_j - \theta\right)$$

$$f(net) = \frac{1}{1 + \exp(-net)}$$




$$f(net) = \tanh(net)$$

# Training set and error

- Let the training set be

$$A_{train} = \{(\mathbf{x}^1, d^1)(\mathbf{x}^2, d^2)...(\mathbf{x}^p, d^p)....(\mathbf{x}^P, d^P)\}$$

  - $\mathbf{x}^p$ is the input array or vector (also called a pattern)
  - $d^p$ is the target or desired output, being +1 for one class of inputs and 0 for the other class of inputs

- Error function = sum of squared errors:

$$E = \frac{1}{2}\sum_{p=0}^{P}(d^p - o^p)^2$$

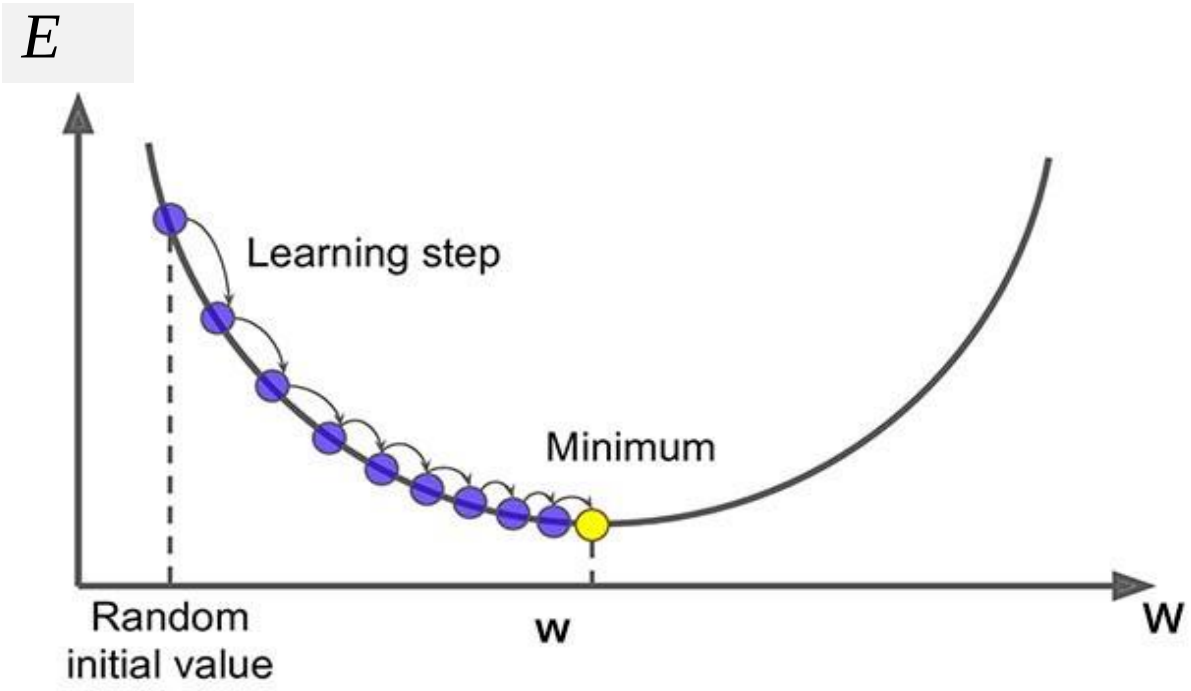
  - Where

$$o = f(\mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w}\cdot\mathbf{x})}}$$

# Gradient descent

- We will adjust perceptron's weights after each input pattern to minimize the error *E* step by step in order to reach its minimum.

- This algorithm of step-like error minimisation is called ***gradient descent.***



# Gradient of a function

- The gradient of a scalar function is a vector, which points in the direction of the greatest rate of increase of the scalar function and whose magnitude is the greatest rate of change.

- The gradient of an error function $E(\mathbf{w})$ with respect to the vector of weights $\mathbf{w} = (w_1, ..., w_n)$ is defined as a vector, the components of which are partial derivatives of $E$ according to individual weights, such that

$$\text{grad}(E) = \nabla E = \left( \frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_n} \right)$$

# Generalized δ rule

- Let the error per pattern $p$ be:

$$E^p = \tfrac{1}{2}(d^p - o^p)^2 = \tfrac{1}{2}\delta^2$$

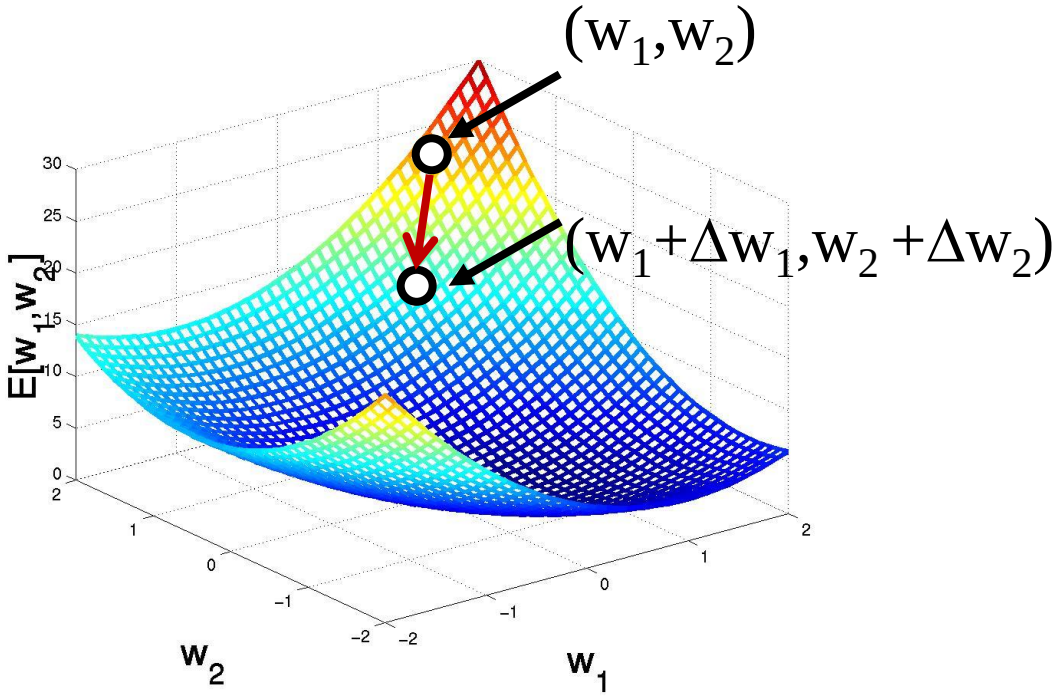
- Let's calculate partial derivative of $E^p$ according to weights, i.e.

$$\frac{\partial E^p}{\partial w_j} = \frac{\partial E^p}{\partial \delta}\frac{\partial \delta}{\partial w_j} = \delta \frac{\partial}{\partial w_j}\left(d^p - f\left(\sum_{j=1}^{n} w_j x_j^p\right)\right) = -\delta\, f'(net)\, x_j^p$$

$$w_j \leftarrow w_j + \alpha\ \delta\ f'\ x_j^p$$

# Weights optimization by gradient descent

- Minimization of the error function $E$ – moving **against** the gradient of $E$




$$w_j \leftarrow w_j - \alpha \frac{\partial E}{\partial w_j}$$

$$w_j \leftarrow w_j + \alpha \ \delta \ f' \ x_j^p$$

- The second term, partial derivative of $E$ according to the weight, is the so-called **generalized error signal**

```
Start with random initial weights in [-.5,.5]) and alpha = 0.5
do
{
    epoch++ // number of loops through the training set
    CORRECT = 0 // number of correctly classified inputs
    for p = 1 to p = P   // loop through all training samples
    {
        net = 0
        for j = 0 to j = N
        {
            net = net + w_j * x_j
        }
        if (net > 0) out = 1 else out  = 0
        if (out = = desired) CORECT++
        else
        {
          for j = 0 to N
         {
            w_j = w_j + alpha * (desired – out) * x_j  * f_der
         }
        }
    }
while (E(w*) < ε) //training stops when error is minimal
```

# Derivative of sigmoid

- The weights are updated due to each example $\mathbf{x}^p$ as

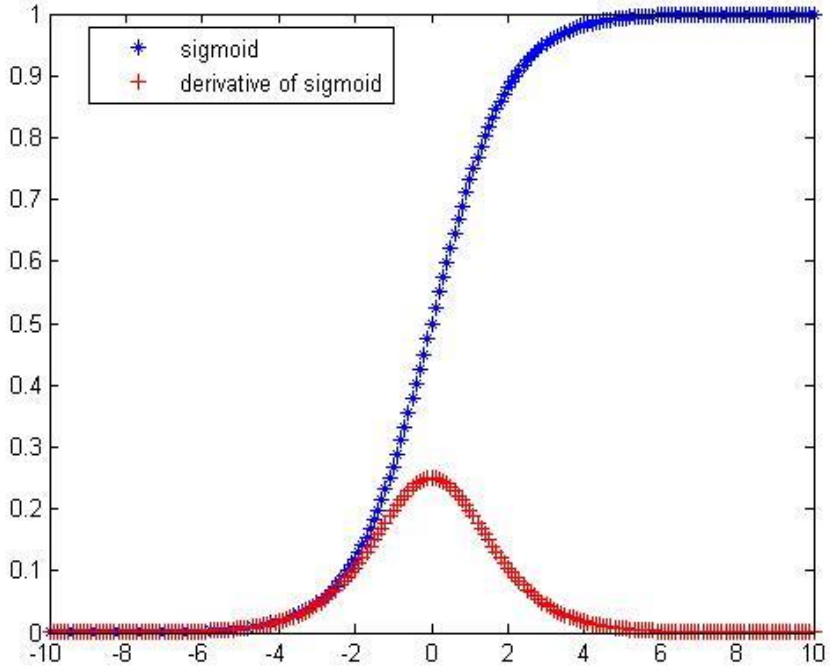
$$w_j \leftarrow w_j + \alpha\ \delta\ f'\ x_j^p$$

  - Constant $\alpha > 0$ is the learning speed
  - The error signal

$$\delta = (d^p - o^p)$$

  - The derivative of sigmoid function

$$f' = f(1-f)$$

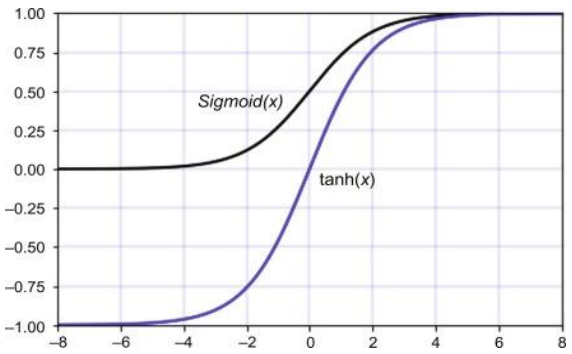


# Intermediate summary

- Binary perceptron performs a linear regression – can find a linear boundary between classes.
  - For inputs $x_i \in [-1, 1]$, we can use the activation function called signum, $\text{sign}(z) = 1$ if $z \geq 0$ and $\text{sign}(z) = -1$ if $z < 0$.

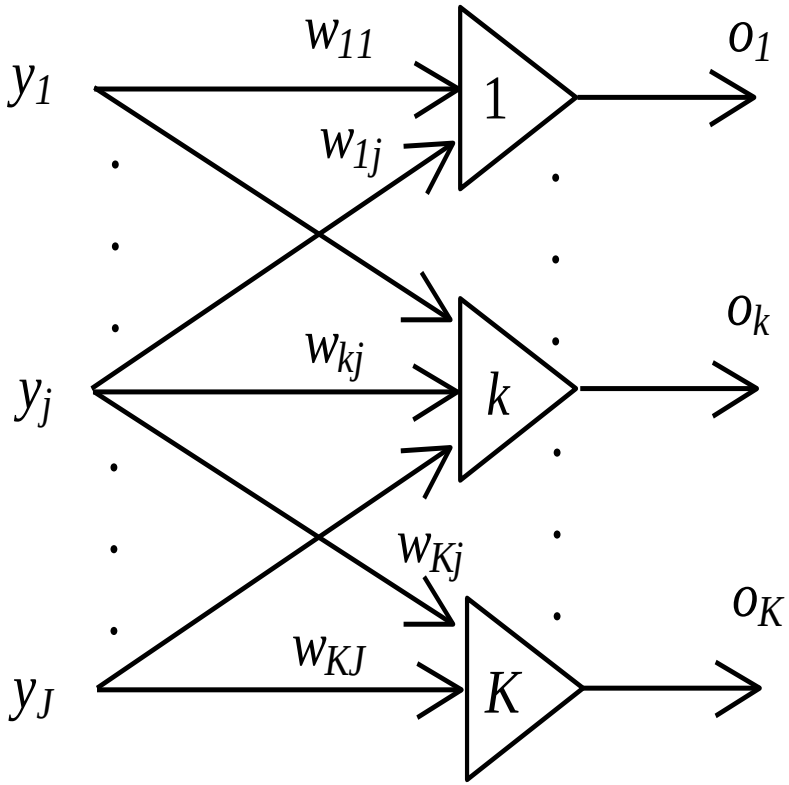
- Nonlinear perceptron performs nonlinear regression.

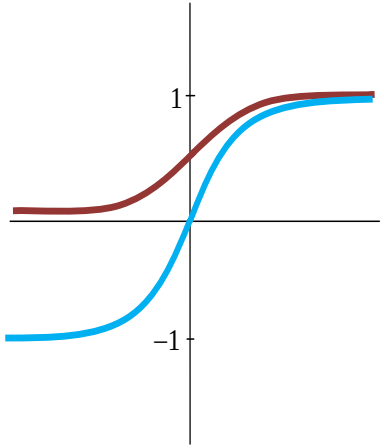
- Nonlinear activation function should have properties like boundedness, monotonicity and differentiability.
  - Function **`tanh(x)`** is often used, too.




- Several perceptrons arranged in a single layer are needed for classification of input patterns into more than two classes.

# Feed-forward perceptron with single (output) layer




$$o_k = f_k(net_k)$$

$$net_k = \sum_{j=1}^{J} w_{kj} y_j$$

$$y_J = -1 \text{ and } w_{kJ} = \theta_k$$

# Towards δ rule for a single output layer

- Let the training set be

$$A_{train} = \{(\mathbf{y}^1, \mathbf{d}^1)(\mathbf{y}^2, \mathbf{d}^2)...(\mathbf{y}^p, \mathbf{d}^p)....(\mathbf{y}^P, \mathbf{d}^P)\}$$

  - $\mathbf{y}^p$ is the input vector (pattern)
  - $\mathbf{d} = (d_1, d_2, ...,d_k, ...,d_K)$ is the desired output vector

- Error function per pattern = sum of squared errors over $k$

$$E^p = \frac{1}{2}\sum_{k=1}^{K}(d_k^p - o_k^p)^2 = \frac{1}{2}\sum_{k=1}^{K}(d_k^p - f_k(net_k))^2$$

  - Where $o_{pk}$ is the actual output of unit $k$ for pattern $p$

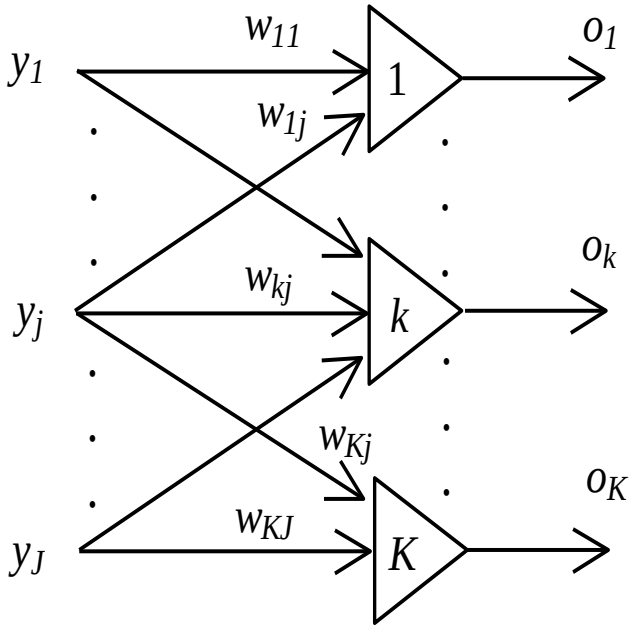
# Generalised δ rule for a single output layer

- After presentation of each pattern we adjust the weights of all output neurons to minimize $E^p$

$$\Delta w_{kj} = -\alpha \frac{\partial E^p}{\partial w_{kj}} = \alpha \, (d_k^p - o_k^p) \; f_k' \; y_j$$

- Generalised error signal produced by the (output) neuron $k$

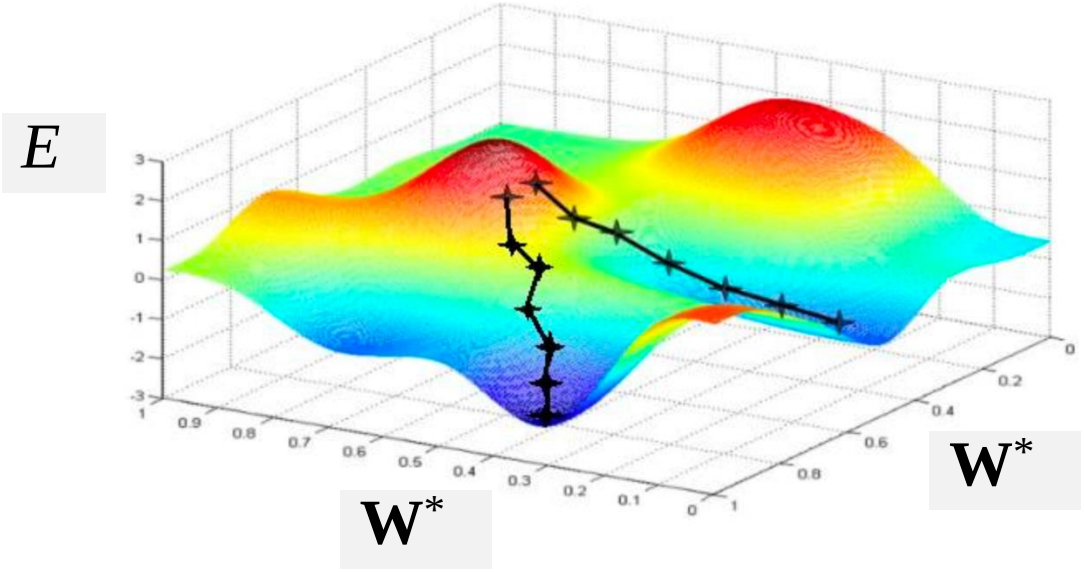
$$\delta_{ok} = (d_k^p - o_k^p) \; f_k'$$



# Conclusion

- The goal of a learning algorithm is to automatically find the values of **weights** to fit any training set of examples.

- Perceptron is *initialized with small random* weights. This affects the final solution. Many solutions to a given problem are possible.




- The nonlinear boundary can be very complex, thus several layers of perceptrons are needed – multilayer perceptron.