Artificial neural networks

1943 – McCulloch, Pitts – first model of an artificial neuron

McCulloch, W., Pitts, W., *A Logical Calculus of the Ideas Immanent in Nervous Activity*, Bulettin of Mathematical Biophysics, vol. 5, 1943, pp. 115-133

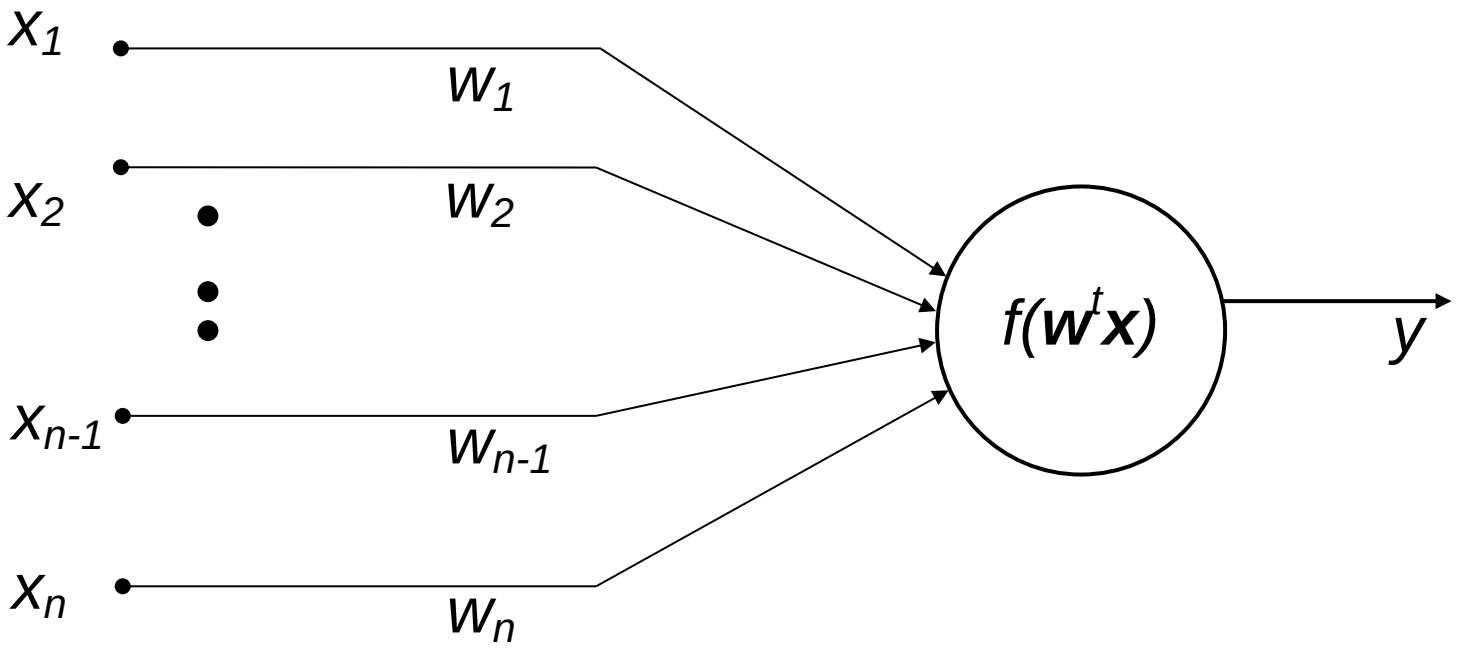
1958 – Rosenblatt prepares model of perceptron – architecture of multilayer network together with its learning method

Rosenblatt, F., *The Perceptron: a Probabilistic Model for Information Storage and Organization in the Brain*, Psychological Review, vol. 65, No. 3, 1958, pp. 386-408

1986 – designing of the delta learning rule and error back propagation algorithm

Rumelhart, D., Hinton G., Williams, R., *Learning Internal Representations by Error Propagation*, in: Parallel Distributed Processing, MIT Press, 1986, pp. 318–362

Neuron's model



Neuron's model

Activation function *f*:

$$y = f(\mathrm{w}^T \mathrm{x}) = f\left(\sum_{i=1}^{n} w_i x_i\right)$$

Weight vector ***w***:

$$w = [w_1, w_2, \ldots, w_n]^T$$

Input vector ***x***:

$$x = [x_1, x_2, \ldots, x_n]^T$$

Net (total) activation *net*:

$$net = \mathrm{w}^T \mathrm{x}$$

Activation functions

Bipolar – values from range [-1, 1]
Unipolar – values from range [0, 1]

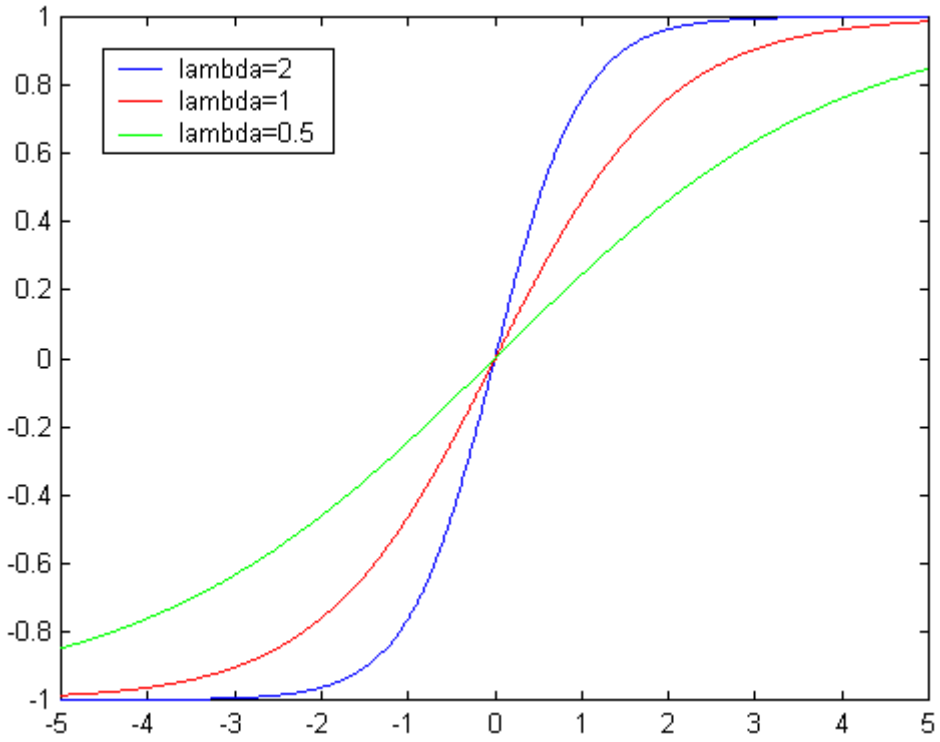
Threshold (discrete, binary) a.f.:

$$f(net) = \mathrm{sgn}(net) = \begin{cases} +1, & when\ net \geq 0 \\ -1, & when\ net < 0 \end{cases}$$

Continuous (sigmoidal) a.f.:

$$f(net) = \frac{2}{1 + \exp(-\lambda\, net)} - 1, \quad \lambda > 0$$

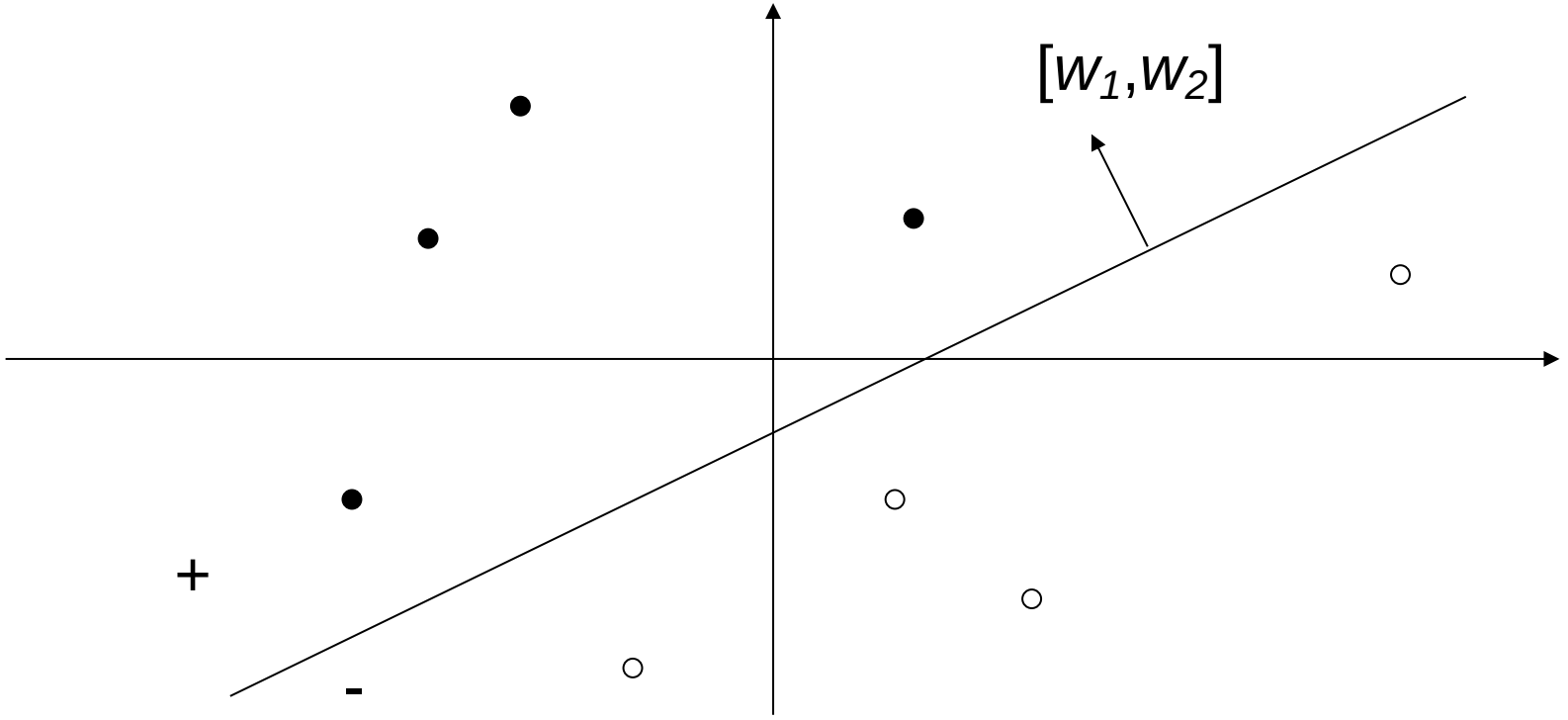
Sigmoidal activation function



Operation interpretation

Two-dimensional example:

$$y = \mathrm{sgn}\left(w_1 x_1 + w_2 x_2 + w_3 x_3\right), \quad x_3 = 1,\ w = \left[-0.5, 1, 1\right]$$



Neuron's learning

Batch learning – weights are updated after whole epoch

Incremental learning – weights are updated after every sample

Supervised and unsupervised learning (learning with critics)

General rule:
Weight vector $w_i = [w_{i1}, w_{i2}, \ldots, w_{in}]^T$ grows proportionally to the scalar product of input signal *x* and learning signal *r*.

Learning signal

$$r = r\left(w_i, x, d_i\right)$$

Change of the vector w_i in learning step:

$$\Delta w_i = c\, r\left(w_i, x, d_i\right) x$$

where c – learning rate (>0)

Corrected weight vector:

$$w_i^{k+1} = w_i^k + c\, r\left(w_i^k, x^k, d_i^k\right) x^k$$

Perceptron rule (Rosenblatt, 1958)

Used for supervised learning of discrete neurons.

$$r = d_i - y_i = d_i - \mathrm{sgn}\left(w_i^t x\right)$$

$$\Delta w_i = c\left[d_i - \mathrm{sgn}\left(w_i^t x\right)\right]x$$

Increase of single weight (when $d_i \neq y_i$):

$$\Delta w_{ij} = \pm 2cx_j, \quad i = 1,2,\ldots,K \quad j = 1,2,\ldots,J$$

Initial weight values: arbitrary.

Delta rule (1986)

Used for supervised learning of continuous neurons.

$$r = \delta = \left[d_i - f\left(w_i^T x\right)\right] f'\left(w_i^T x\right)$$

$$\Delta w_i = c\left[d_i - f\left(w_i^T x\right)\right] f'\left(w_i^T x\right) x, \;\; i = 1,2, \ldots, K$$

Increase of single weight:

$$\Delta w_{ij} = c\left[d_i - f\left(w_i^T x\right)\right] f'\left(w_i^T x\right) x_j, \;\; i = 1,2, \ldots, K \;\;\; j = 1,2, \ldots, J$$

Initial weight values: arbitrary.

One layer network

Desirable net output: $d = [d_1, d_2, \ldots, d_K]^T$

Approximation function error of sample y_l:

$$E_l = \frac{1}{2}\sum_{k=1}^{K}\left(d_{lk} - z_{lk}\right)^2$$

Modify weights in such way that error E_l is minimized (*for the whole training set)*

Gradient method:

$$w^{k+1} = w^k - \eta \nabla E\left(w^k\right)$$

Weight correction

Weight's change:

$$\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}}$$

E depends on w_{kj} through net activation, hence:

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}}, \text{where } \frac{\partial net_k}{\partial w_{kj}} = y_j$$

Delta error signal for k-th neuron:

$$\delta_{zk} = -\frac{\partial E}{\partial net_k}$$

Finally (almost):

$$\Delta w_{kj} = \eta \delta_{zk} y_j$$

Weight correction

Delta error:

$$\delta_{zk} = -\frac{\partial E}{\partial net_k} = -\frac{1}{2}\frac{\partial}{\partial net_k}(d_k - z_k)^2$$

$$= (d_k - z_k)\frac{\partial z_k}{\partial net_k} = (d_k - z_k) f'(net_k)$$

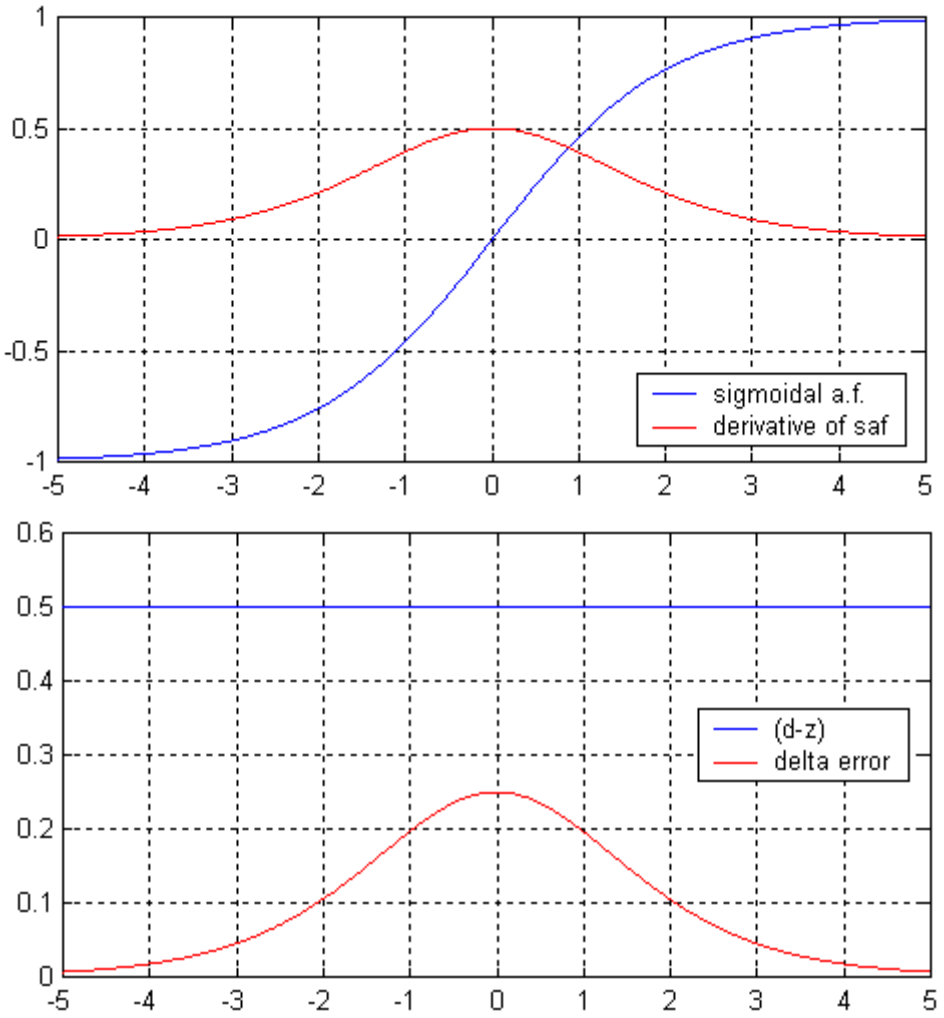
Derivative of the bipolar sigmoidal activation function:

$$f'(net_k) = \frac{2\exp(-net_k)}{[1 + \exp(-net_k)]^2} = \frac{1}{2}[1 - f^2(net_k)] = \frac{1}{2}\left(1 - z_k^2\right)$$

Delta error:

$$\delta_{zk} = (d_k - z_k)\frac{1}{2}\left(1 - z_k^2\right)$$

Delta error



Weight correction

Weight's change:

$$\Delta w_{kj} = \eta \delta_{zk} y_j = \eta \, (d_k - z_k) \frac{1}{2} \left(1 - z_k^2\right) y_j$$

Weights' change in vector form:

$$\mathbf{W}^{k+1} = \mathbf{W}^k + \eta \delta_z y^T$$

where

$$\delta_z = [\delta_{z1}, \delta_{z2}, \ldots, \delta_{zK}]^T$$
$$y^T = \left[y_1, y_2, \ldots, y_J\right]$$

Note that we have here product of column vector (delta) and row vector, so the result is matrix:

$$\begin{bmatrix} \delta_{z1} y_1 & \cdots & \delta_{z1} y_J \\ \vdots & \ddots & \vdots \\ \delta_{zK} y_1 & \cdots & \delta_{zK} y_J \end{bmatrix}$$

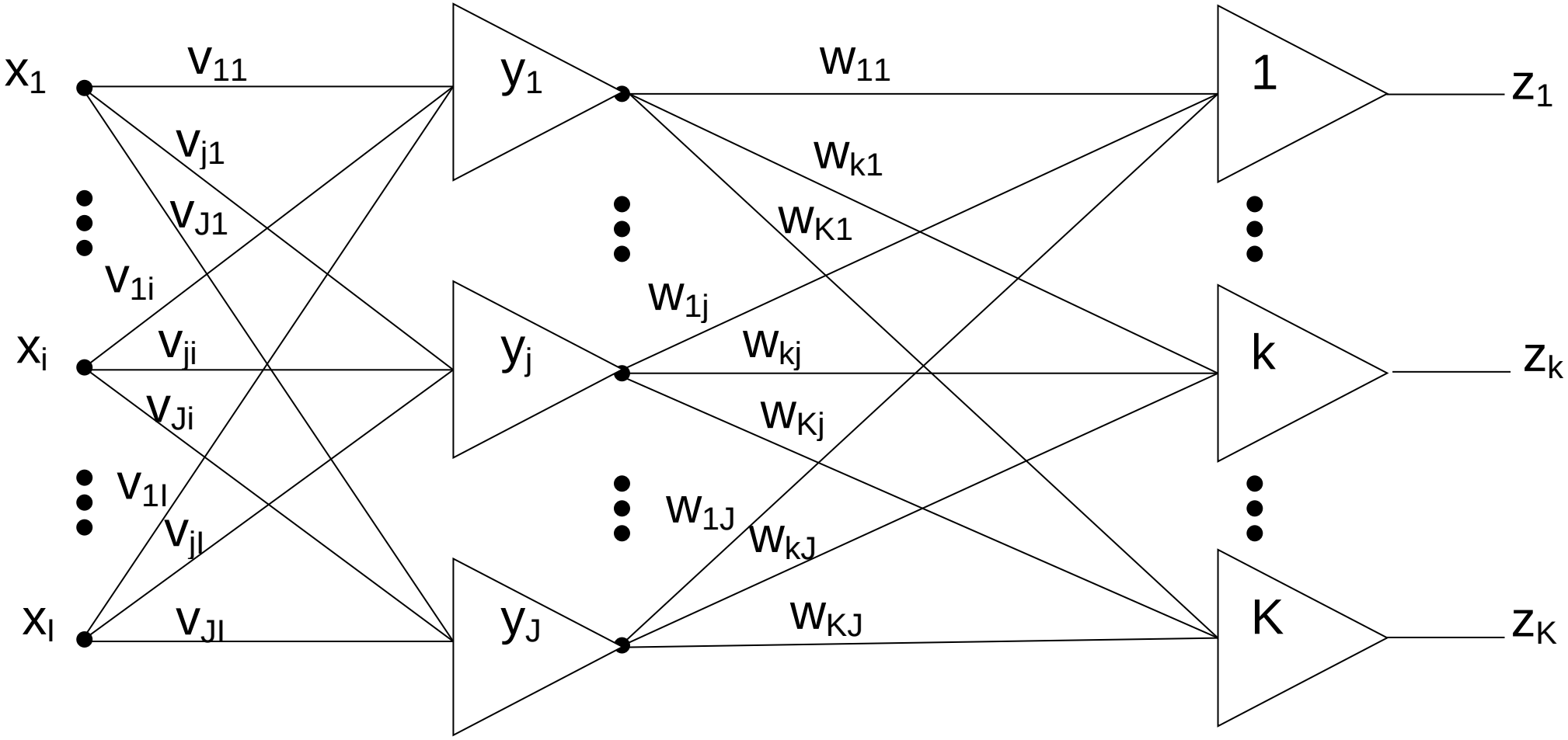
Learning algorithm

1. Select η and E_{max} (**$E_{validation}$=size of the validation set**)
2. Establish initial weights' values W
3. Reset total error $E_{sum}=0$
4. For each learning sample
   5. Compute network answer
   6. Increase total error
   7. Correct weights
8. ~~If $E_{sum} < E_{max}$ stop, in other cases return to 3.~~

Although stopping criterion of point 8 is simple and easy to compute, it can lead to overfitting. Instead we should compute classification error on the validation set and stop learning when this error increases.

8. $E_{validation\_ref} = E_{validation}$
9. Compute $E_{validation}$ – classification error on the validation set
10. If $E_{validation} > E_{validation\_ref}$ stop, ioc. return to 3.

Multilayer network



Weight correction

Weight's change: $\Delta v_{ij} = -\eta \frac{\partial E}{v_{ij}} = \eta \underbrace{\left(-\frac{\partial E}{\partial net_j} \right)}_{\partial_{yj}} \underbrace{\frac{\partial net_j}{\partial v_{ij}}}_{x_i}$

Delta error signal of *j-th* neuron in the hidden layer:

$$\delta_{yj} = -\frac{\partial E}{\partial net_j}$$

Almost finally: $\Delta v_{ji} = \eta \delta_{yj} x_i$

Weight correction

Delta error: $$\delta_{yj} = -\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial net_j} = -\frac{\partial E}{\partial y_j} f'\left(net_j\right)$$

$$\frac{\partial E}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{K}\left(d_k - z_k\right)^2\right] = -\sum_{k=1}^{K}\delta_{zk} w_{kj}$$

At last:

$$\Delta v_{ij} = \eta \underbrace{\left(\sum_{k=1}^{K}\delta_{zk} w_{kj}\right)}_{\substack{in\ the\ output\ layer\ it\ was \\ d_k - z_k}} f_j'\left(net_j\right) x_i$$

Backpropagation (of error) algorithm

1. Select η and E_{max} (**$E_{validation}$=size of the validation set**)
2. Establish initial weights' values W and V
3. Reset total error $E_{sum}=0$
4. For each learning sample
   5. Compute network answer
   6. Increase total error
   7. Compute error signal vectors
   8. Correct weights in output and hidden layers
9. ~~If $E_{sum} < E_{max}$ stop, in other cases return to 3.~~

9. $E_{validation\_ref} = E_{validation}$
10. Compute $E_{validation}$ – classification error on the validation set
11. If $E_{validation} > E_{validation\_ref}$ stop, ioc. return to 3.

Learning errors

Total error:

$$E = \frac{1}{2}\sum_{l=1}^{p}\sum_{k=1}^{K}\left(d_{lk} - z_{lk}\right)^2$$

Normalized mean square error:

$$E_{rms} = \frac{1}{pK}\sum_{l=1}^{p}\sum_{k=1}^{K}\left(d_{lk} - z_{lk}\right)^2$$

Decision error:

$$E_d = \frac{N_b}{pK}$$

Learning parameters

Incremental weight correction
Cumulative weight correction (after the whole epoch):

$$\Delta w = \sum_{l=1}^{p} \Delta w_l$$

Local minima:

- Adding random values to weights
- Adding random values to input vectors
- Random selection of vectors from the training set

Learning parameters

Initial weights

- Multiple learning with different initial values settings
- High weight values cause saturation of neurons (sigmoidal activation function)
- Zero weights are constant for discrete neurons
- General rule: select random weight values so that total activation net~1

Learning parameters

Learning rate η

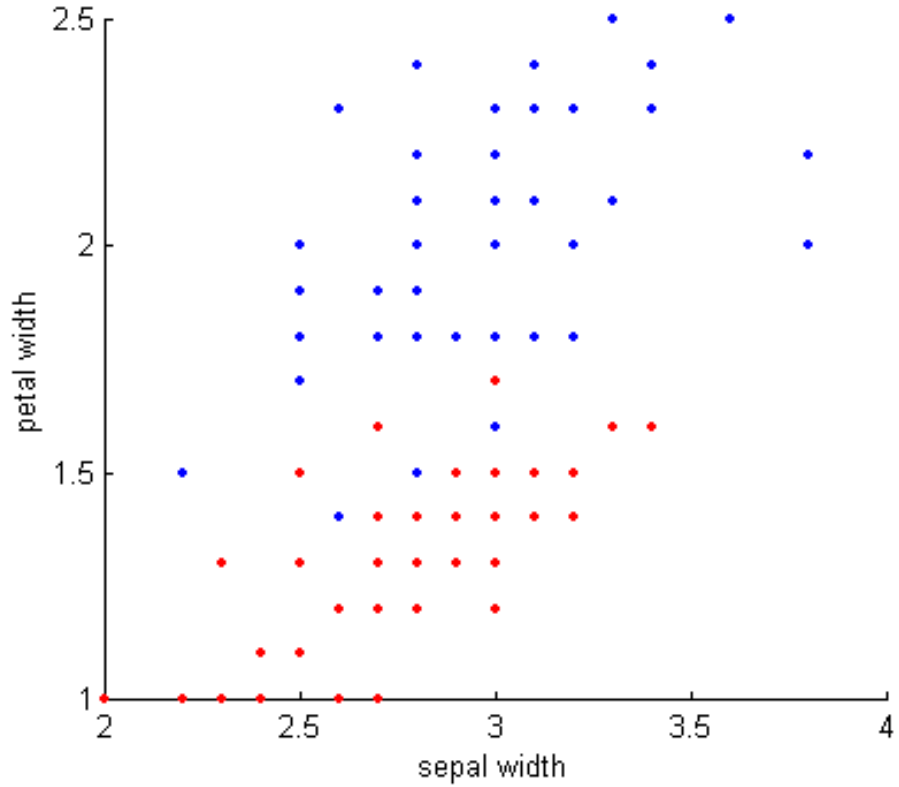
- Small – it is easy to be trapped in local minima
- High – oscillations may arise
- Variable:
  - Increased by constant value when total error systematically decreases
  - Decreased geometrically when total error increases

Adding momentum (aka inertia):

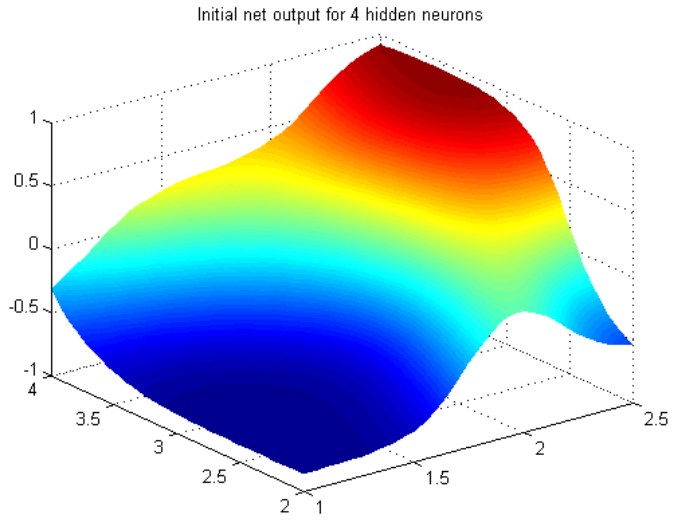
$$w^{k+1} = w^k - \eta \nabla E\left(w^k\right) + \mu \Delta w^k$$

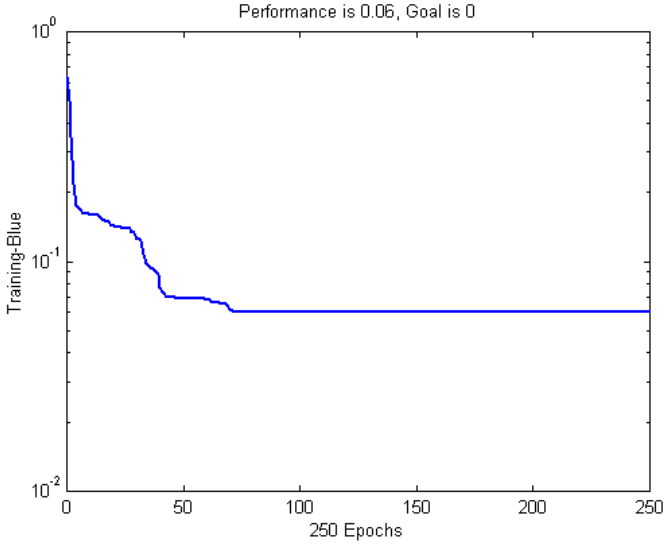
Example

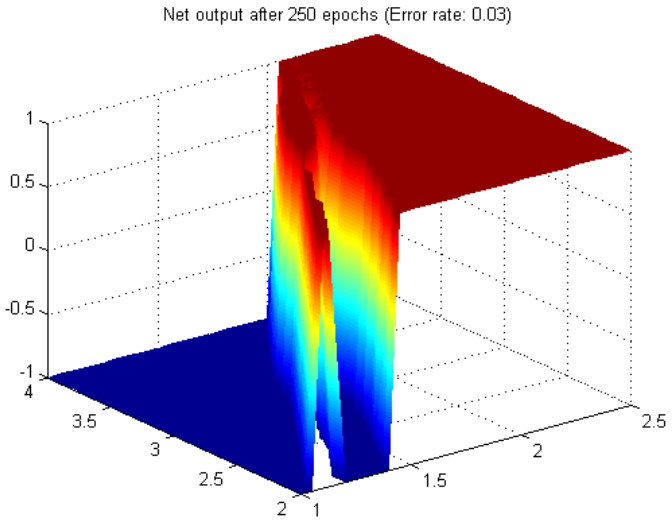
Two classes in 2D space (*Iris versicolor* and *Iris virginica*). Three-layer network 2-4-1 with sigmoidal activation functions. Whole work was done in MATLAB.



Example

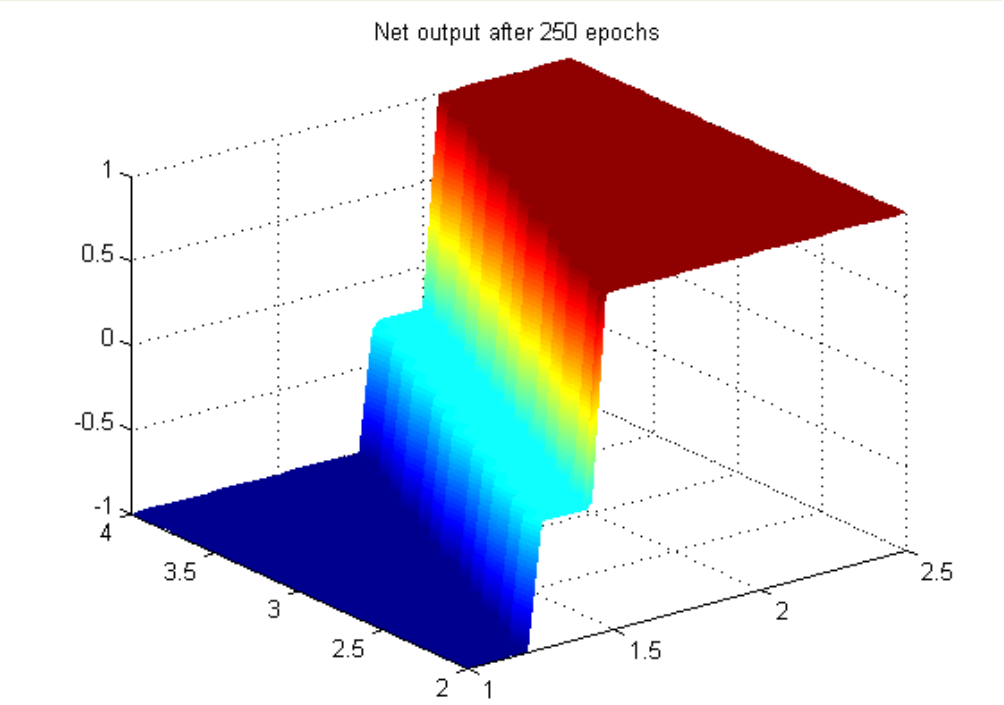

Initial net output for 4 hidden neurons
1
0.5
0
-0.5
-1
4
3.5
3
2.5
2
1
1.5
2
2.5


Performance is 0.06, Goal is 0
Training-Blue
250 Epochs
0
50
100
150
200
250


Net output after 250 epochs (Error rate: 0.03)
1
0.5
0
-0.5
-1
4
3.5
3
2.5
2
1
1.5
2
2.5

Example

It has happened once:



Will classifier presented on the previous page perform well for new values?