Introduction to inverse and ill-posed problems: Classification algorithms. Neural networks for classification. Lecture 6

Introduction

- Machine learning is a field of artificial intelligence which gives computer systems the ability to “learn” using available data.
- We will study linear and polynomial classifiers, more precisely, least squares and perceptron learning algorithms for classification problems.
- Introduce artificial neural networks and study multilayer perceptron algorithm. We will discover convergence for all these algorithms and compare their performance with respect to applicability, reliability, accuracy, and efficiency. Programs written in Matlab will demonstrate performance for every algorithm.

Reference literature: Miroslav Kurbat, *An Introduction to Machine Learning*, Springer, 2017.

Christopher M. Bishop, *Pattern recognition and machine learning*, Springer, 2009.

L. Beilina, E. Karchevskii, M. Karchevskii, *Numerical Linear Algebra: Theory and Applications*, Springer, 2017 – see link to GitHub with Matlab code

Classification problem

- Suppose that we have data points $(x_i, y_i), i = 1, ..., m$. These points are separated into two classes A and B. Assume that these classes are linearly separable.

  Definition

  Let A and B are two data sets of points in an n-dimensional Euclidean space. Then A and B are linearly separable if there exist $n + 1$ real numbers $\omega_1, ..., \omega_n, l$ such that every point $x \in A$ satisfies $\sum_{i=1}^{n} \omega_i x_i > l$ and every point $x \in B$ satisfies $\sum_{i=1}^{n} \omega_i x_i < -l$.

- Our goal is to find the decision line which will separate these two classes. This line will also predict in which class will the new point fall.

Least squares and classification

Least squares can be used for classification problems appearing in machine learning algorithms.

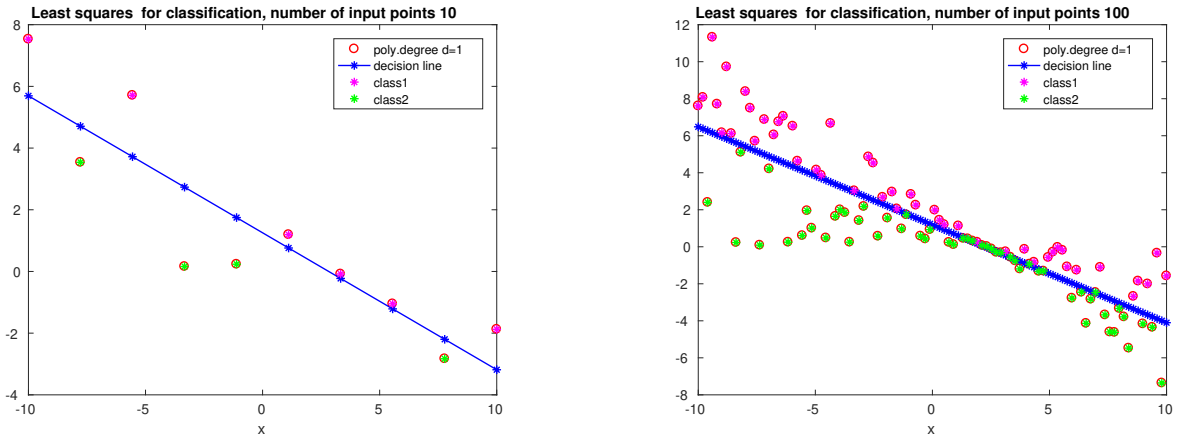


Figure: Examples of working least squares for classification.

Least squares minimization $\min_x ||Ax - b||_2^2$ for classification is working fine when we know that two classes are linearly separable.

Least squares and classification

Least squares can be used for classification problems appearing in machine learning algorithms.

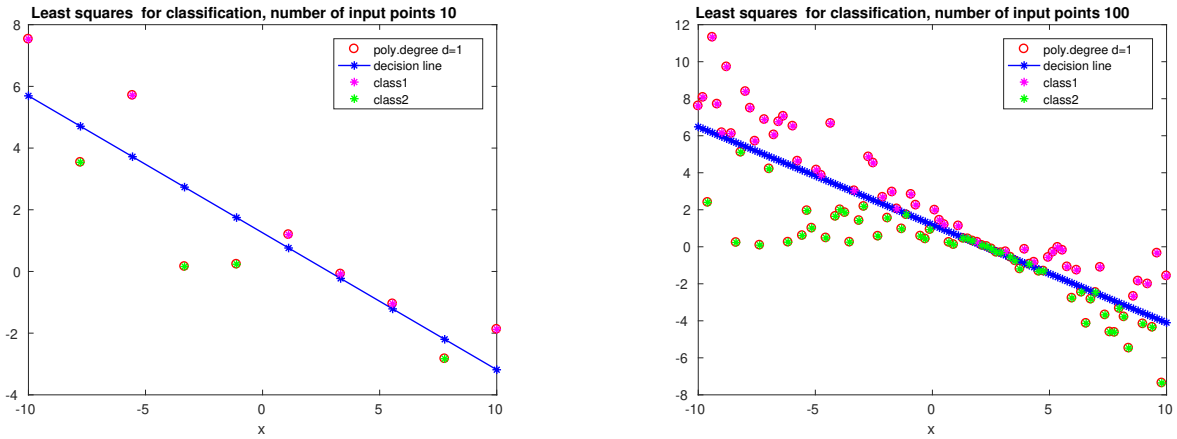


Figure: Examples of working least squares for classification.

Least squares minimization $\min_x ||Ax - b||_2^2$ for classification is working fine when we know that two classes are linearly separable.

Least squares and classification

Least squares can be used for this classification problem. Let us consider two-class model. Let the first class consisting of l points with coordinates $(x_i, y_i), i = 1, ..., l$ is described by it's linear model

$$f_1(x, c) = c_{1,1}\phi_1(x) + c_{2,1}\phi_2(x) + ... + c_{n,1}\phi_n(x). \quad (1)$$

Let the second class consisting of k points with coordinates $(x_i, y_i), i = 1, ..., k$ is also described by the same linear model

$$f_2(x, c) = c_{1,2}\phi_1(x) + c_{2,2}\phi_2(x) + ... + c_{n,2}\phi_n(x). \quad (2)$$

Here, functions $\phi_j(x), j = 1, ..., n$ are called basis functions. Our goal is to find the vector of parameters $c = c_{i,1} = c_{i,2}, i = 1, ..., n$ of the size n which will fit best to the data $y_i, i = 1, ..., m, m = k + l$ of both model functions, $f_1(x_i, c), i = 1, ..., l$ and $f_2(x_i, c), i = 1, ..., k$ with $f(x, c) = [f_1(x_i, c), f_2(x_i, c)]$ such that

$$\min_c \sum_{i=1}^{m} (y_i - f(x_i, c))^2 \quad (3)$$

with $m = k + l$.

Least squares and classification

If the function $f(x, c)$ in

$$\min_c \sum_{i=1}^{m} (y_i - f(x_i, c))^2 \tag{4}$$

is linear then we can solve the problem (4) using least squares method. Let now the matrix A of the size $m \times n$, $m = k + l$ in

$$Ac = b$$

will have entries $a_{ij} = \phi_j(x_i), i = 1, ..., m; j = 1, ..., n$, and vector b will be such that $b_i = y_i, i = 1, ..., m$.

Least squares and classification

Then a linear data fitting problem takes the form

$$Ac = b \tag{5}$$

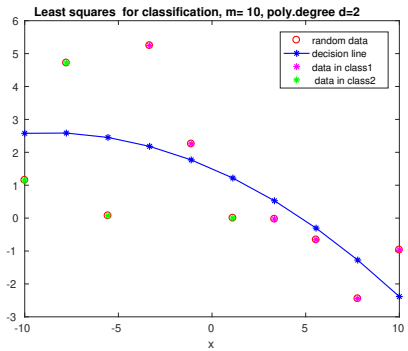
Elements of the matrix A are created by basis functions $\phi_j(x), j = 1, ..., n$. Solution of (5) can be found by the method of normal equations:

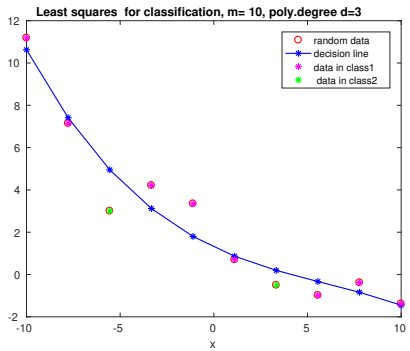
$$c = (A^T A)^{-1} A^T b = A^+ b \tag{6}$$

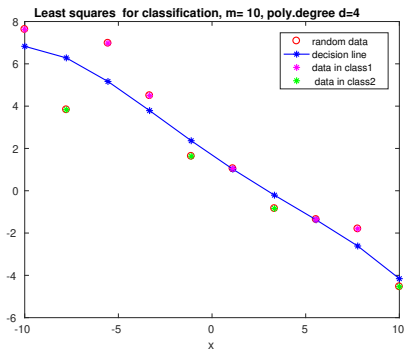
Different basis functions can be chosen. We have considered $\phi_j(x) = x^{j-1}, j = 1, ..., n$ in the problem of fitting to a polynomial. The matrix A constructed by these basis functions is a Vandermonde matrix. Linear splines (or hat functions) and bellsplines also can be used as basis functions.

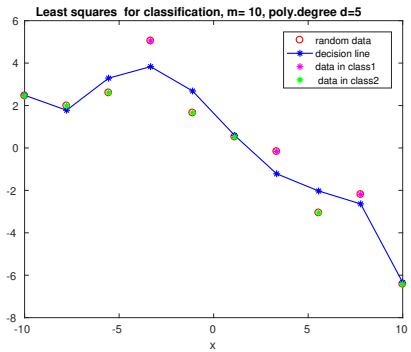
Least squares and classification: example

Least squares minimization for classification is working fine when we know that two classes are linearly separable. Higher degree of polynomial separates two classes better. However, since Vandermonde's matrix can be ill-conditioned for high degrees of polynomial, we should carefully choose appropriate polynomial to fit data.





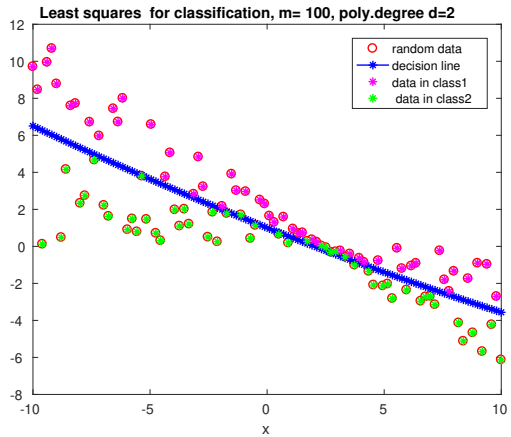


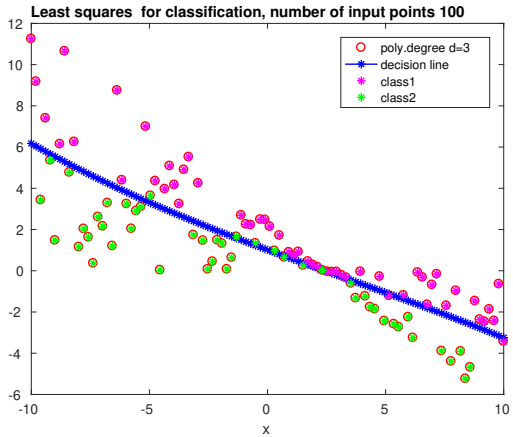


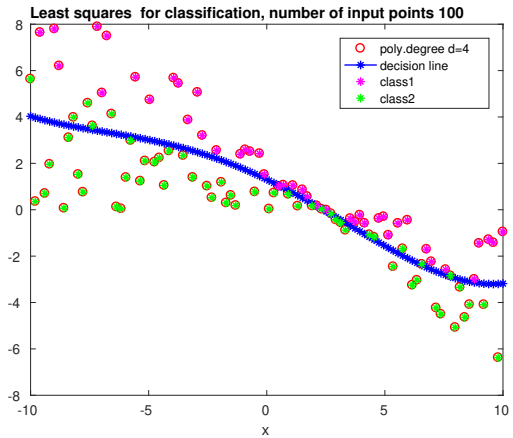
Examples of working least squares for separation of two classes for different degrees of polynomial and number of input

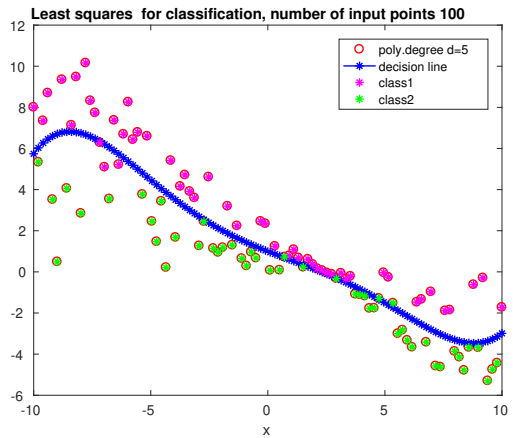
Least squares and classification

Examples below present computation of decision line for separation of two classes with $m = 100$ using basis functions $\phi_j(x) = x^{j-1}, j = 1, ..., d$, where d is degree of the polynomial.









Machine learning algorithms: linear and polynomial classifiers

Let us consider boolean domains where each attribute is true or false. and we will represent *true* by 1 and *false* by 0.
Below we present a table where is presented a boolean domain with two classes and two boolean attributes (here, true is 1 and false is 0).

x	y	Class
1	1	positive
1	0	negative
0	1	negative
0	0	negative

We observe that these two classes can be separated by linear equation

$$\omega_1 + \omega_2 x + \omega_3 y = 0. \tag{7}$$

Our goal is to find weights $\omega_1, \omega_2, \omega_3$ in order to determine the decision line $y(x)$. The decision line which will separate two classes will have the form $y(x) = (-\omega_1 - \omega_2 x)/\omega_3$.

Linear and polynomial classifiers

On the figure below, two classes should be separated:one example we labeled as positive class, another one as negative. In this case, two classes can be separated by linear equation

$$\omega_1 + \omega_2 x + \omega_3 y = 0 \tag{8}$$

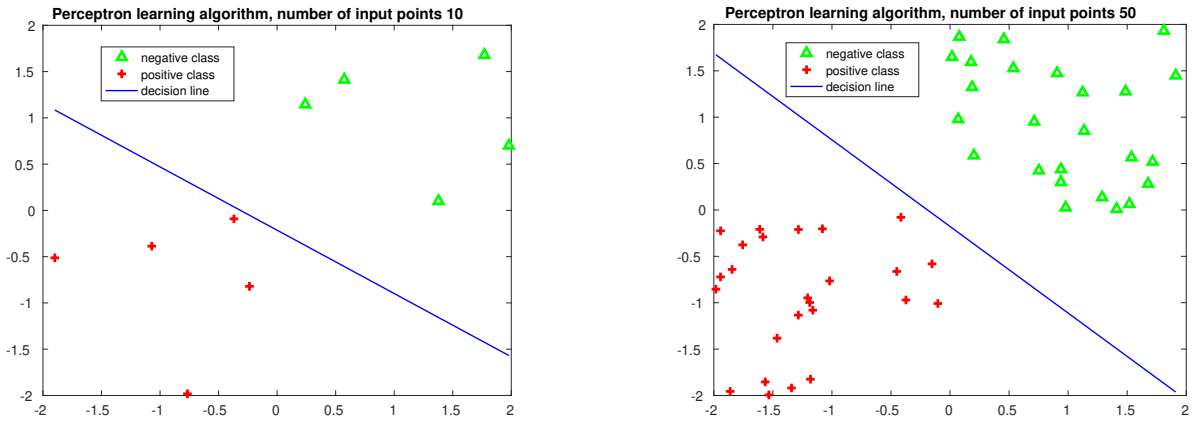


Figure: Examples of working perceptron learning algorithm which computes weights for decision line to separate two classes.

Linear and polynomial classifiers

In common case, two classes can be separated by the general equation

$$\omega_0 + \omega_1 x_1 + \omega_2 x_2 + ... + \omega_n x_n = 0 \tag{9}$$

which also can be written as

$$\sum_{i=0}^{n} \omega_i x_i = 0 \tag{10}$$

with $x_0 = 1$. If $n = 2$ then the above equation defines a line, if $n = 3$ - plane, if $n > 3$ - hyperplane. Our problem is to determine weights ω_i and the task of machine learning is to determine their appropriate values. Weights $\omega_i, i = 1, ..., n$ determine the angle of the hyperplane, ω_0 is called bias and determines the offset, or the hyperplanes distance from the origin of the system of coordinates.

Machine learning: Perceptron learning algorithm

- Let us assume that every training example $\mathbf{x} = (x_1, ..., x_n)$ is described by n attributes with values $x_i = 0$ or $x_i = 1$.
- We will label positive examples with $c(\mathbf{x}) = 1$ and negative with $c(\mathbf{x}) = 0$.
- Let us denote by $h(\mathbf{x})$ the classifier's hypothesis which also will have binary values $h(\mathbf{x}) = 1$ or $h(\mathbf{x}) = 0$.
- We will also assume that all examples where $c(\mathbf{x}) = 1$ are linearly separable from examples where $c(\mathbf{x}) = 0$.

Machine learning: Perceptron learning algorithm

- Step 0. Initialize weights ω_i to small random numbers.
- Step 1. If $\sum_{i=0}^{n} \omega_i x_i > 0$ we will say that the example is positive and $h(x) = 1$.
- Step 2. If $\sum_{i=0}^{n} \omega_i x_i < 0$ we will say the the example is negative and $h(x) = 0$.
- Step 3. Update every weight using the formula

$$\omega_i = \omega_i + \eta \cdot [c(\mathbf{x}) - h(\mathbf{x})] \cdot x_i.$$

- Step 4. If $c(\mathbf{x}) = h(\mathbf{x})$ for all learning examples - stop. Otherwise return to step 1.

Here, $\eta \in (0, 1]$ is called the learning rate.

Example of Linear Perceptron learning algorithm in Matlab

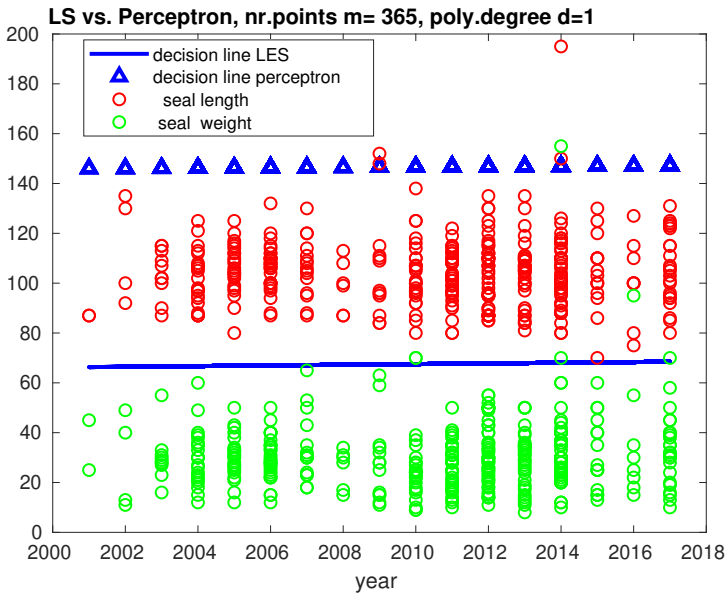
Below we present program for classification of grey seals into 2 classes (you can download files with data of grey seals on waves24.com/download). In the first class is data of seal length depending on their weight, in the second class is data of seal thickness depending on their weight.

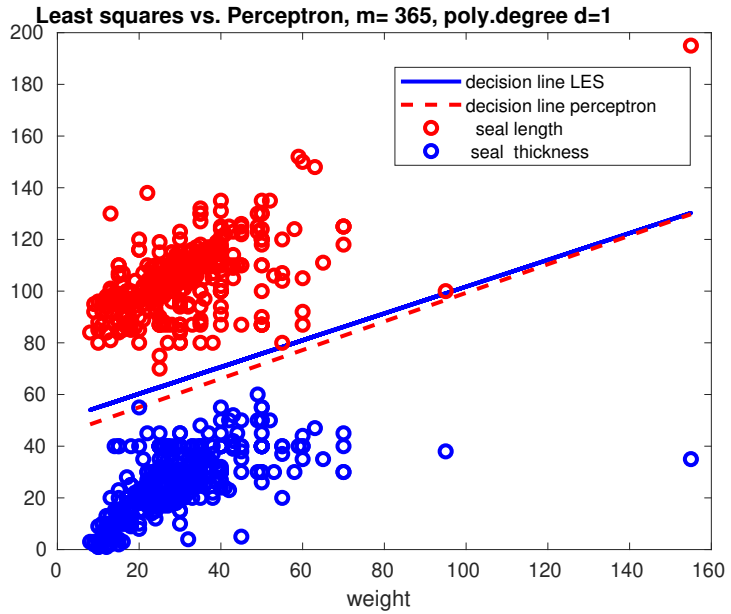
```
// files with data
load seallength.m; load weight.m; load thikness.m; m=size(weight,1);
//number of discretization points or rows in the matrix A
x=zeros(1,2*m); y=zeros(1,2*m);
class = zeros(1,2*m); hyp = zeros(1,2*m);
//classify points into 2 classes
for i=1:1:m
sch1 = sch1 +1;
// class1, class2 are for visualization of points
class1(sch1)= seallength(i); // it is x-axis x1(sch1) = weight(i);
sch2 = sch2 +1;
class2(sch2)= thikness(i); x2(sch2) = weight(i);
y(i) = seallength(i); // it is x-axis
x(i) = weight(i);
class(sch1) = 1;// class 1 in perceptron learning algorithm
hyp(sch1)= 0; end
```

Example of Linear Perceptron learning algorithm in Matlab

```
i=0; sch3=0;
for i=m+1:1:2*m sch3 = sch3 +1;
y(i) = thikness(sch3); x(i) = weight(sch3);
class(i) = 0;// class 2 in perceptron learning algorithm
hyp(i)= 1;  end
weight= rand(3,1); // init weights
eta = 0.5; // init learning rate
while sum(class |= (not equal) hyp)
count = 0;
for i=1:1:m
if weight(1) + weight(2)*x(i) + weight(3)*y(i) > 0
hyp(i)=1;
else
hyp(i)=0;
end
//update weights
weight(1) = weight(1) + eta*(class(i) - hyp(i)); weight(2) = weight(2) + eta*(class(i) - hyp(i))*x(i);
weight(3) = weight(3) + eta*(class(i) - hyp(i))*y(i);
count = count+1;
if count > 10000 // to avoid infinite loop
count break end end end
```

Linear classification: LS versus Perceptron





Polynomial of the second order

Coefficients of polynomials of the second order can be obtained by the same technique as coefficients for linear classifiers.
The second order polynomial function is:

$$\omega_0 + \omega_1 \underbrace{x_1}_{z_1} + \omega_2 \underbrace{x_2}_{z_2} + \omega_3 \underbrace{x_1^2}_{z_3} + \omega_4 \underbrace{x_1 x_2}_{z_4} + \omega_5 \underbrace{x_2^2}_{z_5} = 0 \qquad (11)$$

This polynomial can be converted to the linear classifier if we introduce notations:

$$z_1 = x_1, z_2 = x_2, z_3 = x_1^2, z_4 = x_1 x_2, z_5 = x_2^2.$$

Then equation (11) can be written in new variables as

$$\omega_0 + \omega_1 z_1 + \omega_2 z_2 + \omega_3 z_3 + \omega_4 z_4 + \omega_5 z_5 = 0 \qquad (12)$$

which is already linear function. Thus, the Perceptron learning algorithm can be used to determine weights $\omega_0, ..., \omega_5$ in (12).

Polynomial of the second order

Suppose that you have determined weights $\omega_0, ..., \omega_5$ in (12). To present the decision line you need to solve the quadratic equation for x_2:

$$\omega_0 + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_1^2 + \omega_4 x_1 x_2 + \omega_5 x_2^2 = 0 \tag{13}$$

with known weights $\omega_0, ..., \omega_5$ and known x_1. We can rewrite (13) as

$$\underbrace{\omega_5}_{a} x_2^2 + x_2 \underbrace{(\omega_2 + \omega_4 x_1)}_{b} + \underbrace{\omega_0 + \omega_1 x_1}_{c} = 0 \tag{14}$$

or as

$$a x_2^2 + b x_2 + c = 0. \tag{15}$$

Solutions of (15) will be

$$x_2 = \frac{-b \pm \sqrt{D}}{2a},$$
$$D = b^2 - 4ac.$$

Example of Quadratic Perceptron learning algorithm in Matlab

Below we present program for classification of grey seals into 2 classes (you can download files with data of grey seals on waves24.com/download) using quadratic perceptron learning algorithm. In the first class is data of seal length depending on their weight, in the second class is data of seal thickness depending on their weight. These classes are the same as in the example for linear perceptron learning algorithm.

Example of Quadratic Perceptron learning algorithm in Matlab

```
weight= rand(6,1); // init weights eta = 0.5; // init learning rate count = 0;
while sum(class |= (not equal) hyp)
for i=1:1:2*m
if weight(1) + weight(2)*x(i) + weight(3)*y(i) + weight(4)*x(i)*x(i)
+ weight(5)*x(i)*y(i) + weight(6)*y(i)*y(i) > 0
hyp(i)=1;
else
hyp(i)=0;
end
//update weights
weight(1) = weight(1) + eta*(class(i) - hyp(i));
weight(2) = weight(2) + eta*(class(i) - hyp(i))*x(i);
weight(3) = weight(3) + eta*(class(i) - hyp(i))*y(i);
weight(4) = weight(4) + eta*(class(i) - hyp(i))*x(i)*x(i);
weight(5) = weight(5) + eta*(class(i) - hyp(i))*x(i)*y(i);
weight(6) = weight(6) + eta*(class(i) - hyp(i))*y(i)*y(i);
end
count = count+1;
if count > 10000 // to avoid infinite loop count break end end
```

Perceptron learning algorithm for polynomial of the second order: example

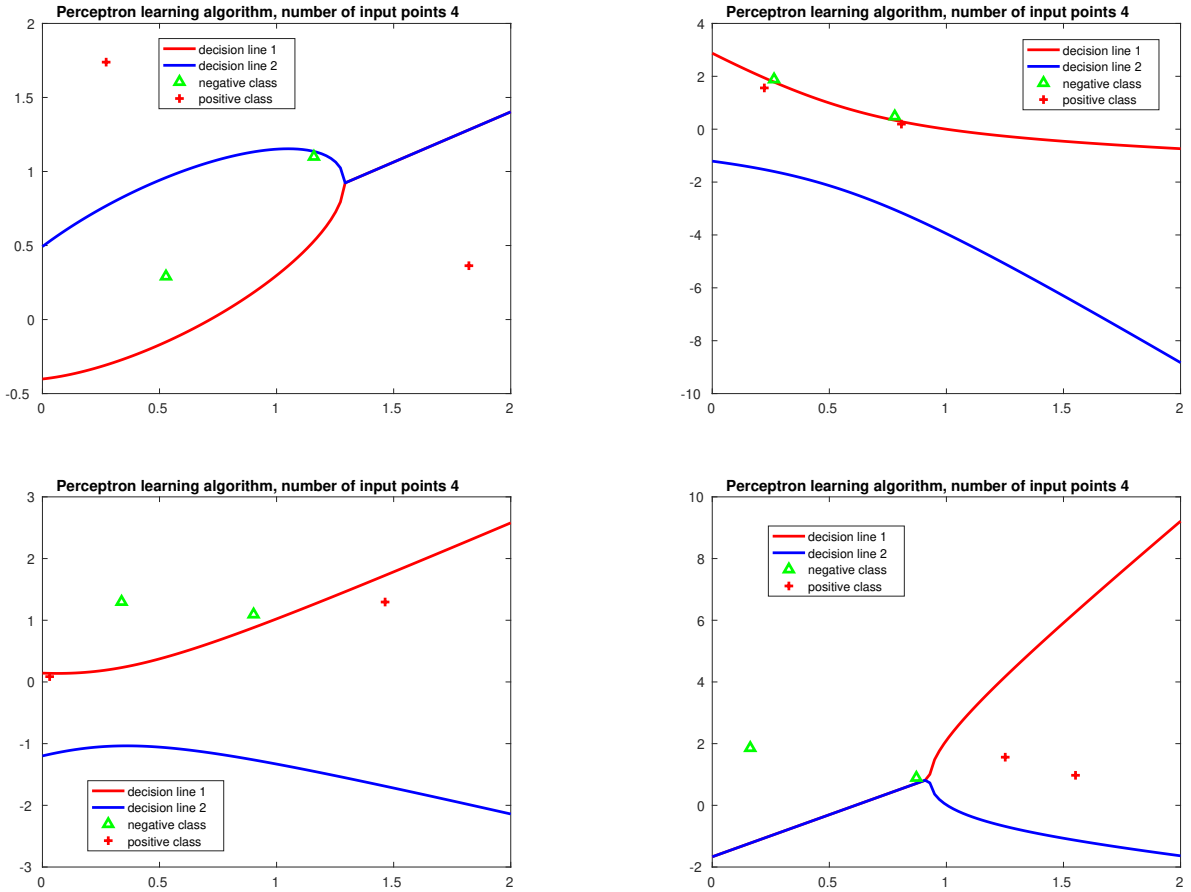


Figure: Separation of two classes by polynomials of the second order for 4 points.

Perceptron learning algorithm for polynomial of the second order: example

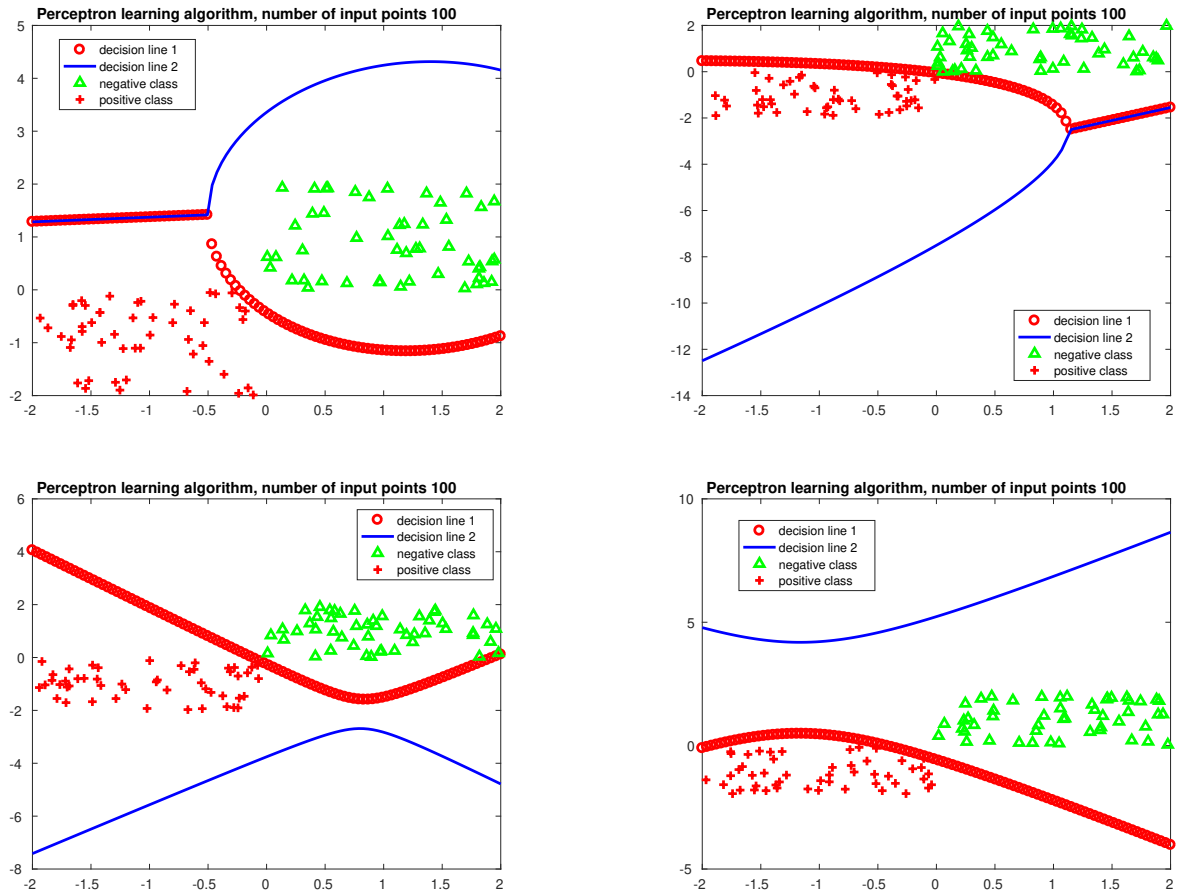


Figure: Separation of two linearly separated classes by polynomials of the second order for 100 points.

Perceptron learning algorithm for polynomial of the second order: example (continuation)

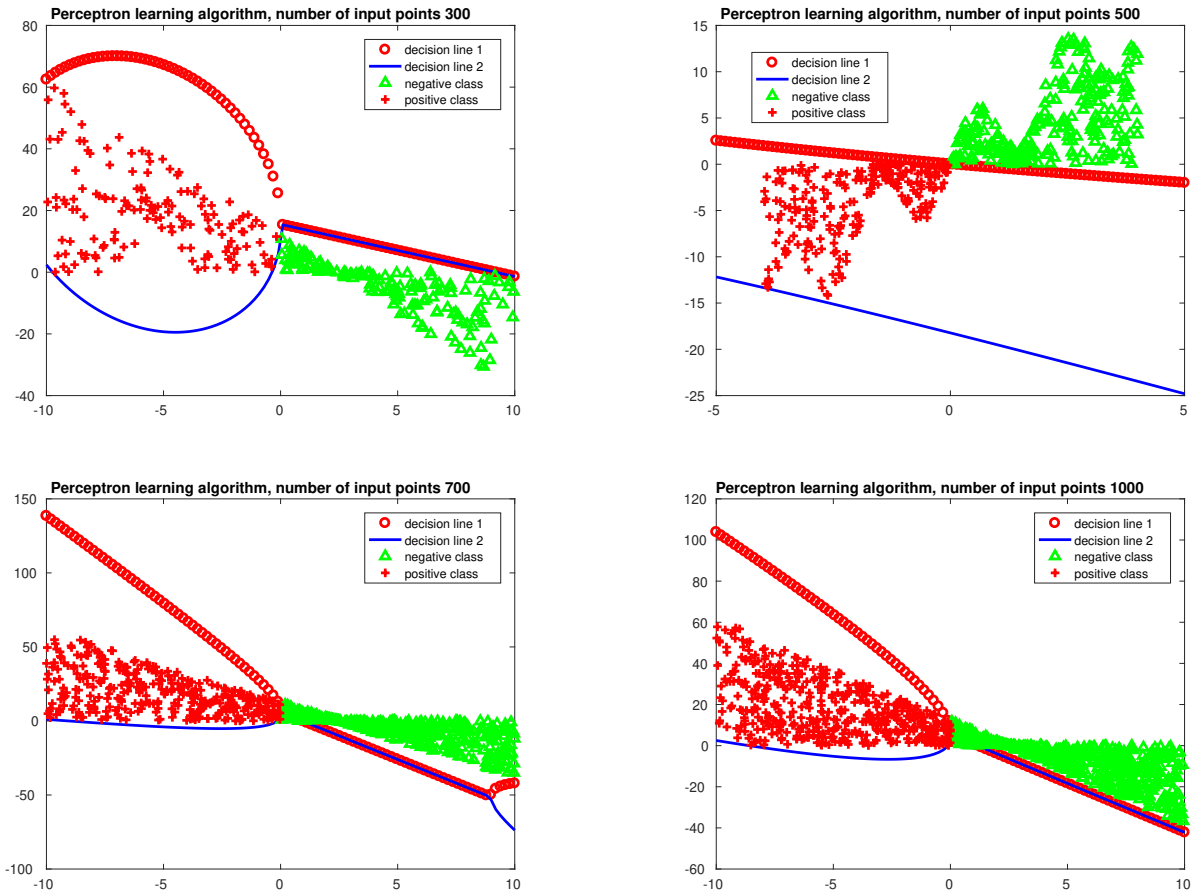


Figure: Separation of two linearly separated classes by polynomials of the second order.

Perceptron learning algorithm for polynomial of the second order: example

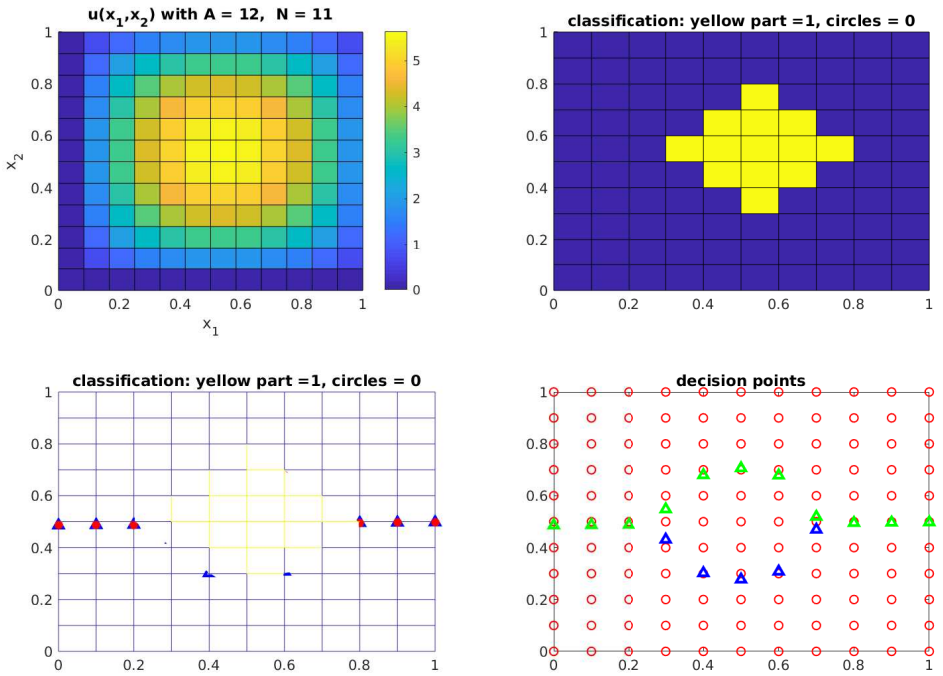


Figure: Separation of the solution of Poisson equation in 2D on the square (see example 8.1.3 in the course book) by polynomials of the second order.

Perceptron learning algorithm for polynomial of the second order: example

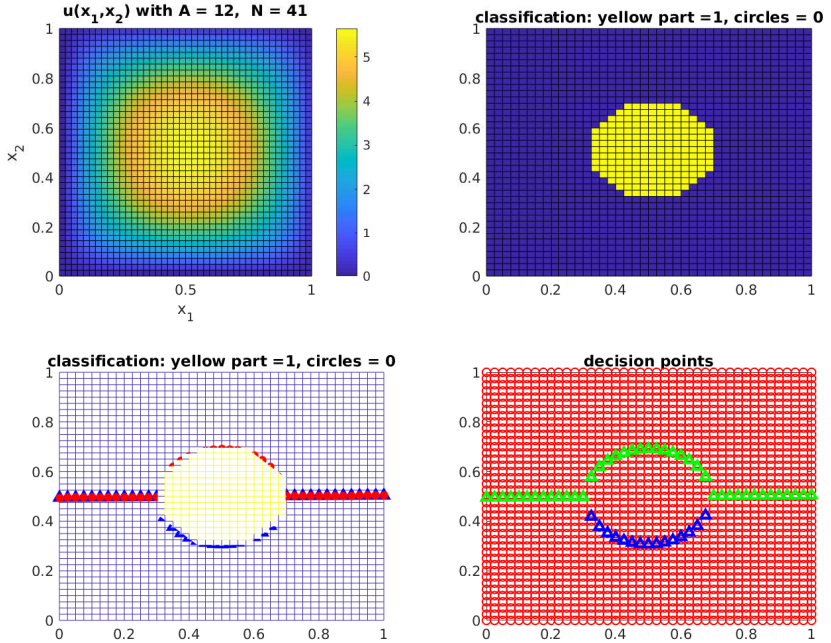


Figure: Separation of the solution of Poisson equation in 2D on the square (see example 8.1.3 in the course book) by polynomials of the second order.

Machine learning: WINNOW learning algorithm

Perceptron learning algorithm used additive rule, while WINNOW algorithm uses multiplicative rule: weights are multiplied in this rule.
We will again assume that all examples where $c(\mathbf{x}) = 1$ are linearly separable from examples where $c(\mathbf{x}) = 0$. Main steps in the WINNOW learning algorithm are:
Step 0. Initialize weights $\omega_i = 1$. Choose parameter $\alpha > 1$, usually $\alpha = 2$.
Step 1. If $\sum_{i=0}^{n} \omega_i x_i > 0$ we will say that the example is positive and $h(x) = 1$.
Step 2. If $\sum_{i=0}^{n} \omega_i x_i < 0$ we will say the the example is negative and $h(x) = 0$.
Step 3. Update every weight using the formula

$$\omega_i = \omega_i \cdot \alpha^{(c(\mathbf{x}) - h(\mathbf{x})) \cdot x_i}.$$

Step 4. If $c(\mathbf{x}) = h(\mathbf{x})$ for all learning examples - stop. Otherwise return to step 1.

Perceptron learning algorithm vs. WINNOW: example

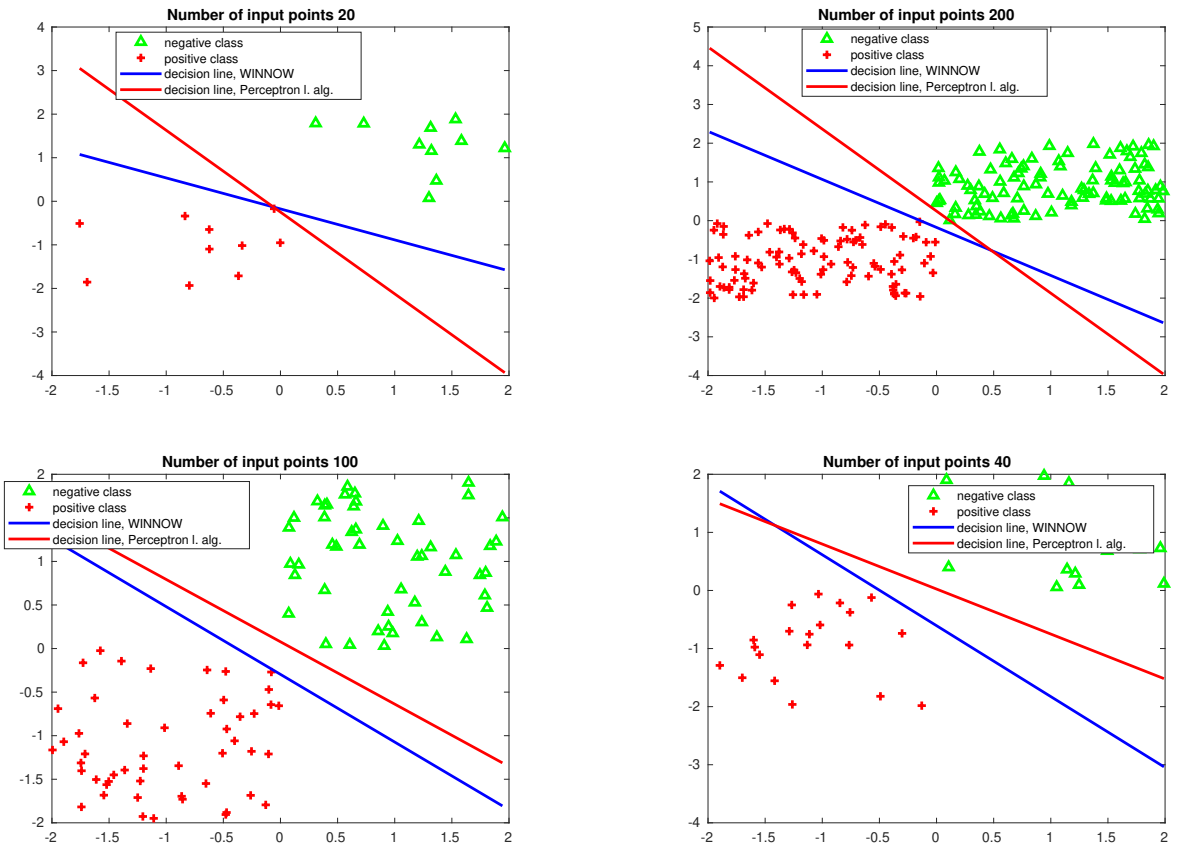


Figure: Comparison of two classification algorithms for separation of two classes: Perceptron learning algorithm (red line) and WINNOW (blue line).

Artificial neural networks

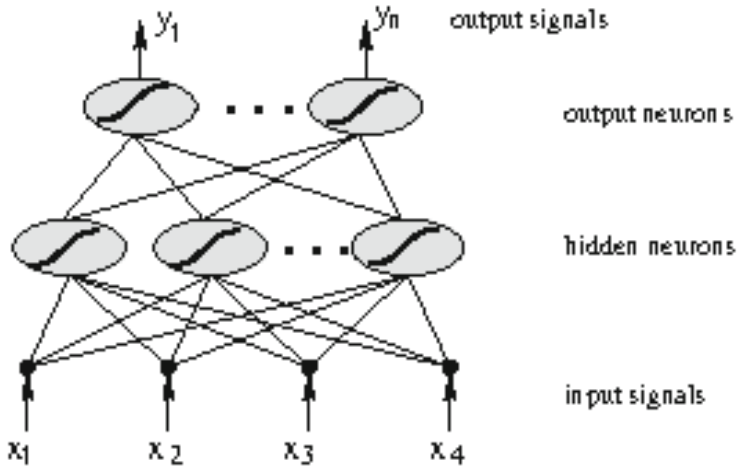


Figure: Example of neural network which contains two interconnected layers (M. Kurbat, *An Introduction to machine learning*, Springer, 2017.)

- In an artificial neural network simple units - neurons- are interconnected by weighted links into structures of high performance.
- Multilayer perceptrons and radial basis function networks will be discussed.

Neurons

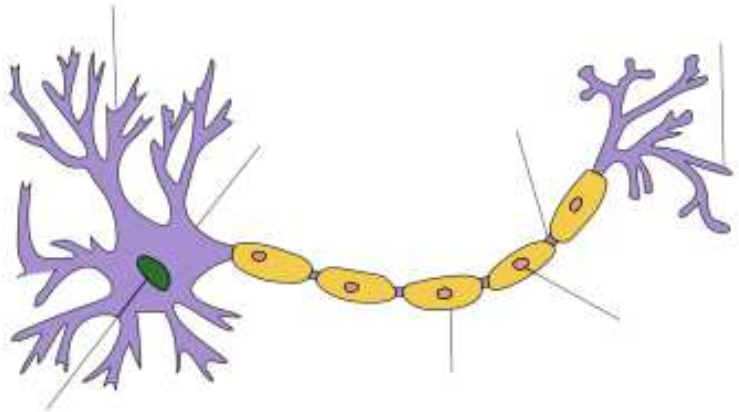


Figure: Structure of a typical neuron (Wikipedia).

- A neuron, also known as a nerve cell, is an electrically excitable cell that receives, processes, and transmits information through electrical and chemical signals. These signals between neurons occur via specialized connections called synapses.
- An artificial neuron is a mathematical function which presents a model of biological neurons, resulting in a neural network.

Artificial neurons

- Artificial neurons are elementary units in an artificial neural network. The artificial neuron receives one or more inputs and sums them to produce an output (or activation, representing a neuron's action potential which is transmitted along its axon).
- Each input is separately weighted by weights ω_{kj}, and the sum $\sum_k \omega_{kj} x_k$ is passed as an argument $\Sigma = \sum_k \omega_{kj} x_k$ through a non-linear function $f(\Sigma)$ which is called the activation function or transfer function.
- Assume that attributes x_k are normalized and belong to the interval $[-1, 1]$.

Artificial neurons

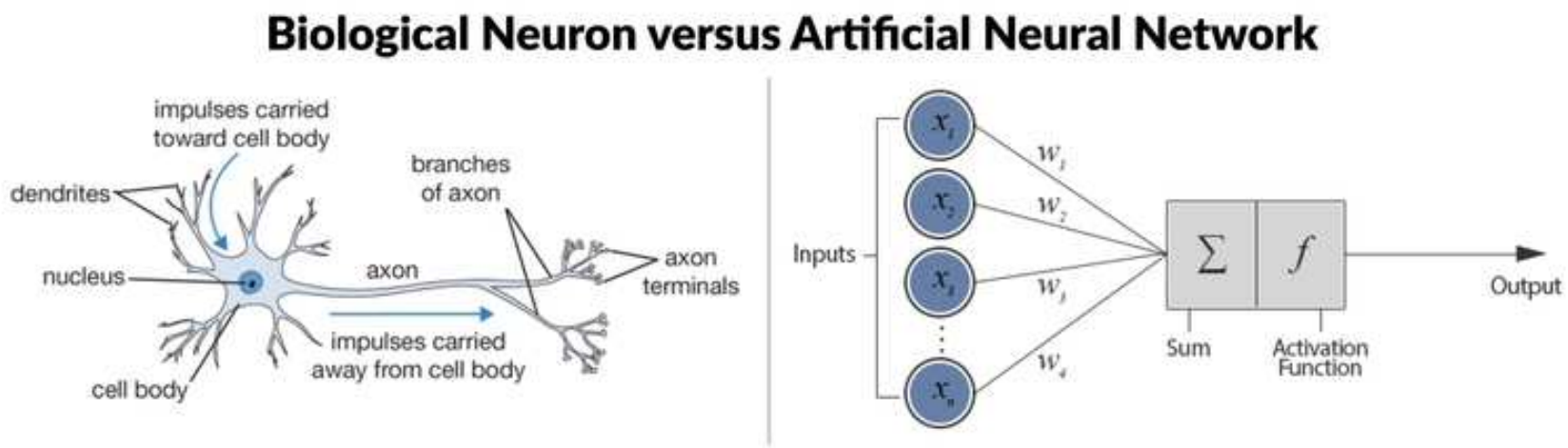


Figure: Perceptron neural network consisting of one neuron (source: DataCamp(datacamp.com)).

Each input is separately weighted by weights ω_{kj}, and the sum $\sum_k \omega_{kj} x_k$ is passed as an argument $\Sigma = \sum_k \omega_{kj} x_k$ through a non-linear function $f(\Sigma)$ which is called the activation function or transfer function.

Artificial neurons: transfer functions

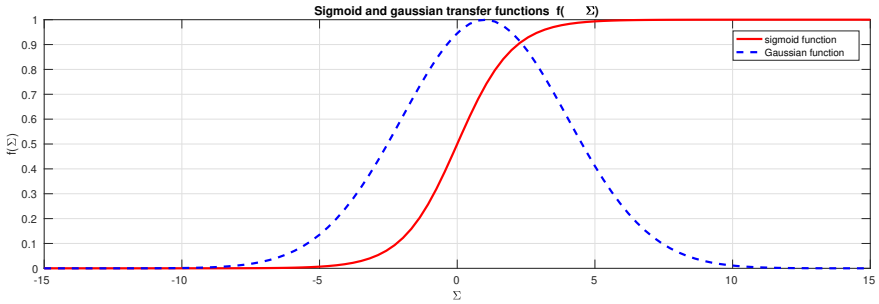


Figure: Sigmoid and Gaussian (for $b = 1, \sigma = 3$ in (17)) transfer functions.

- Different transfer (or activation) functions $f(\Sigma)$ with $\Sigma = \sum_k \omega_{kj} x_k$ are used. We will study sigmoid and gaussian functions.
- Sigmoid function:

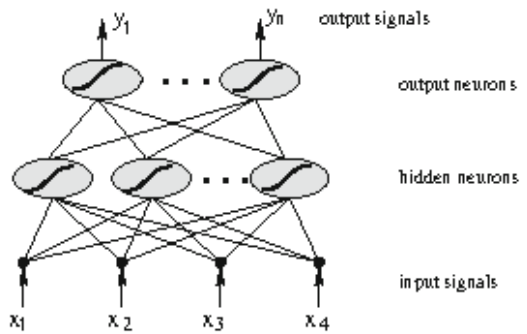
$$f(\Sigma) = \frac{1}{1 + e^{-\Sigma}} \tag{16}$$

- Gaussian function centered at b for a given variance σ^2

$$f(\Sigma) = \frac{e^{-(\Sigma - b)^2}}{2\sigma^2} \tag{17}$$

Forward propagation

Example of neural network called multilayer perceptron (one hidden layer of neurons and one output layer). (M. Kurbat, *An Introduction to machine learning*, Springer, 2017.)



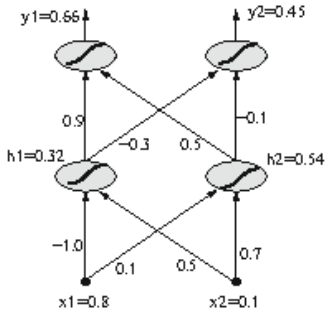
- Neurons in adjacent layer are fully interconnected.
- Forward propagation is implemented as

$$y_i = f(\Sigma_j \omega_{ji}^{(1)} x_j) = f(\Sigma_j \omega_{ji}^{(1)} \underbrace{f(\Sigma_k \omega_{kj}^{(2)} x_k)}_{x_j}), \qquad (18)$$

where $\omega_{ji}^{(1)}$ and $\omega_{kj}^{(2)}$ are weights of the output and the hidden neurons, respectively, f is the transfer function.

Example of forward propagation through the network

Source: M. Kurbat, *An Introduction to machine learning*, Springer, 2017.



- Using inputs x_1, x_2 compute inputs of hidden-layer neurons:

$$x_1^{(2)} = 0.8 * (-1.0) + 0.1 * 0.5 = -0.75, \; x_2^{(2)} = 0.8 * 0.1 + 0.1 * 0.7 = 0.15$$

- Compute transfer function (sigmoid $f(\Sigma) = \frac{1}{1+e^{-\Sigma}}$ in our case):

$$h_1 = f(x_1^{(2)}) = 0.32, \; h_2 = f(x_2^{(2)}) = 0.54.$$

- Compute input of output-layer neurons

$$x_1^{(1)} = 0.32 * 0.9 + 0.54 * 0.5 = 0.56, \; x_2^{(1)} = 0.32 * (-0.3) + 0.54 * (-0.1) = -0.15.$$

- Compute outputs of output-layer neurons using transfer function (sigmoid in our case):

$$y_1 = f(x_1^{(1)}) = 0.66, \; y_2 = f(x_2^{(1)}) = 0.45.$$

Backpropagation of error through the network

Our goal is to find optimal weights $\omega_{ji}^{(1)}$ and $\omega_{kj}^{(2)}$ in forward propagation

$$y_i = f(\Sigma_j \omega_{ji}^{(1)} x_j) = f(\Sigma_j \omega_{ji}^{(1)} \underbrace{f(\Sigma_k \omega_{kj}^{(2)} x_k)}_{x_j}). \tag{19}$$

To do this we introduce functional

$$F(\omega_{ji}^{(1)}, \omega_{kj}^{(2)}) = \frac{1}{2}\|t_i - y_i\|^2 = \frac{1}{2}\sum_{i=1}^{m}(t_i - y_i)^2. \tag{20}$$

Here, $t = t(x)$ is the target vector which depends on the concrete example x. In the domain with m classes the target vector $t = (t_1(x), ..., t_m(x))$ consists of m binary numbers such that

$$t_i(x) = \begin{cases} 1, & \text{example } x \text{ belongs to } i\text{-th class,} \\ 0, & \text{otherwise.} \end{cases} \tag{21}$$

Examples of target vector and mean square error

Let there exist three different classes c_1, c_2, c_3 and x belongs to the class c_2. Then the target vector is $t = (t_1, t_2, t_3) = (0, 1, 0)$.
The mean square error is defined as

$$E = \frac{1}{m}\|t_i - y_i\|^2 = \frac{1}{m}\sum_{i=1}^{m}(t_i - y_i)^2. \tag{22}$$

Let us assume that we have two different networks to choose from, every network with 3 output neurons corresponding to classes c_1, c_2, c_3. Let $t = (t_1, t_2, t_3) = (0, 1, 0)$ and for the example x the first network output is $y_1 = (0.5, 0.2, 0.9)$ and the second network output is $y_2 = (0.6, 0.6, 0.7)$.

$$E_1 = \frac{1}{3}\sum_{i=1}^{3}(t_i - y_i)^2 = \frac{1}{3}((0-0.5)^2 + (1-0.2)^2 + (0-0.9)^2)) = 0.57,$$

$$E_2 = \frac{1}{3}\sum_{i=1}^{3}(t_i - y_i)^2 = \frac{1}{3}((0-0.6)^2 + (1-0.6)^2 + (0-0.7)^2)) = 0.34.$$

Since $E_2 < E_1$ then the second network is less wrong on the example x than the first network.

Backpropagation of error through the network

To find minimum of the functional (41) $F(\omega)$ with $\omega = (\omega_{ji}^{(1)}, \omega_{kj}^{(2)})$, recall it below:

$$F(\omega) = F(\omega_{ji}^{(1)}, \omega_{kj}^{(2)}) = \frac{1}{2}\|t_i - y_i\|^2 = \frac{1}{2}\sum_{i=1}^{m}(t_i - y_i)^2, \tag{23}$$

we need to solve the minimization problem

$$\min_{\omega} F(\omega) \tag{24}$$

and find a stationary point of (23) with respect to ω such that

$$F'(\omega)(\bar{\omega}) = 0, \tag{25}$$

where $F'(\omega)$ is the Fréchet derivative such that

$$F'(\omega)(\bar{\omega}) = F'_{\omega_{ji}^{(1)}}(\omega)(\bar{\omega}_{ji}^{(1)}) + F'_{\omega_{kj}^{(2)}}(\omega)(\bar{\omega}_{kj}^{(2)}). \tag{26}$$

Backpropagation of error through the network

Recall now that y_i in the functional (23) is defined as

$$y_i = f(\sum_j \omega_{ji}^{(1)} x_j) = f(\sum_j \omega_{ji}^{(1)} \underbrace{f(\sum_k \omega_{kj}^{(2)} x_k)}_{x_j}). \quad (27)$$

Thus, if the transfer function f in (27) is sigmoid, then

$$\begin{aligned} F'_{\omega_{ji}^{(1)}}(\omega)(\bar{\omega}_{ji}^{(1)}) &= (t_i - y_i) \cdot y'_{i\,(\omega_{ji}^{(1)})}(\bar{\omega}_{ji}^{(1)}) \\ &= (t_i - y_i) \cdot x_j \cdot f(\sum_j \omega_{ji}^{(1)} x_j)(1 - f(\sum_j \omega_{ji}^{(1)} x_j))(\bar{\omega}_{ji}^{(1)}) \quad (28) \\ &= (t_i - y_i) \cdot x_j \cdot y_i(1 - y_i))(\bar{\omega}_{ji}^{(1)}), \end{aligned}$$

Backpropagation of error through the network

Here we have used that for the sigmoid function $f'(\Sigma) = f(\Sigma)(1 - f(\Sigma))$ since

$$
\begin{aligned}
f'(\Sigma) &= \frac{e^{-\Sigma}}{(1+e^{-\Sigma})^2} = \frac{1+e^{-\Sigma}-1}{(1+e^{-\Sigma})^2} \\
&= f(\Sigma)\left[\frac{(1+e^{-\Sigma})-1}{1+e^{-\Sigma}}\right] = f(\Sigma)\left[\frac{(1+e^{-\Sigma})}{1+e^{-\Sigma}} - \frac{1}{1+e^{-\Sigma}}\right] \\
&= f(\Sigma)(1 - f(\Sigma)).
\end{aligned} \tag{29}
$$

Backpropagation of error through the network

Again, since

$$y_i = f(\sum_j \omega_{ji}^{(1)} x_j) = f(\sum_j \omega_{ji}^{(1)} \underbrace{f(\sum_k \omega_{kj}^{(2)} x_k)}_{x_j}). \tag{30}$$

for the sigmoid transfer function f we also get

$$\begin{aligned} F'_{\omega_{kj}^{(2)}}(\omega)(\bar{\omega}_{kj}^{(2)}) &= (t_i - y_i) \cdot y'_{i\,(\omega_{kj}^{(2)})}(\bar{\omega}_{kj}^{(2)}) \\ &= \left[\underbrace{h_j(1-h_j)}_{f'(h_j)} \cdot \left[\sum_i \underbrace{y_i(1-y_i)}_{f'(y_i)}(t_i - y_i)\omega_{ji}^{(1)}\right] \cdot x_k\right](\bar{\omega}_{kj}^{(2)}), \end{aligned} \tag{31}$$

since for the sigmoid function f we have:
$f'(h_j) = f(h_j)(1 - f(h_j)), f'(y_i) = f(y_i)(1 - f(y_i))$ (prove this). Hint:
$h_j = f(\sum_k \omega_{kj}^{(2)} x_k), y_i = f(\sum_j \omega_{ji}^{(1)} x_j)$.

Backpropagation of error through the network

Usually, $F'_{\omega_{ji}^{(1)}}(\omega)/x_j$, $F'_{\omega_{kj}^{(2)}}(\omega)/x_k$ in (28), (31) are called responsibilities of output layer neurons and hidden-layer neurons $\delta_i^{(1)}, \delta_i^{(2)}$, respectively, and they are defined as

$$\begin{aligned} \delta_i^{(1)} &= (t_i - y_i)y_i(1 - y_i), \\ \delta_j^{(2)} &= h_j(1 - h_j) \cdot \sum_i \delta_i^{(1)} \omega_{ji}^{(1)}. \end{aligned} \tag{32}$$

By knowing responsibilities (32), weights can be updates using usual gradient update formulas:

$$\begin{aligned} \omega_{ji}^{(1)} &= \omega_{ji}^{(1)} + \eta \delta_i^{(1)} x_j, \\ \omega_{kj}^{(2)} &= \omega_{kj}^{(2)} + \eta \delta_j^{(2)} x_k. \end{aligned} \tag{33}$$

Here, η is the step size in the gradient update of weights and we use value of learning rate for it such that $\eta \in (0, 1)$.

Algorithm A1: backpropagation of error through the network with one hidden layer

- Step 0. Initialize weights.
- Step 1. Take example x in the input layer and perform forward propagation.
- Step 2. Let $y = (y_1, ..., y_m)$ be the output layer and let $t = (t_1, ..., t_m)$ be the target vector.
- Step 3. For every output neuron $y_i, i = 1, ..., m$ calculate its responsibility δ_i^1 as

$$\delta_i^{(1)} = (t_i - y_i) y_i (1 - y_i). \tag{34}$$

- Step 4. For every hidden neuron compute responsibility $\delta_j^{(2)}$ for the network's error as

$$\delta_j^{(2)} = h_j(1 - h_j) \cdot \sum_i \delta_i^{(1)} (\omega_{ji})^1, \tag{35}$$

where $\delta_i^{(1)}$ are computed using (37).

- Step 5. Update weights with learning rate $\eta \in (0, 1)$ as

$$\begin{aligned} \omega_{ji}^{(1)} &= \omega_{ji}^{(1)} + \eta(\delta_i^{(1)}) x_j, \\ \omega_{kj}^{(2)} &= \omega_{kj}^{(2)} + \eta(\delta_j^{(2)}) x_k. \end{aligned} \tag{36}$$

Algorithm A2: backpropagation of error through the network with l hidden layers

- Step 0. Initialize weights and take $l = 1$.
- Step 1. Take example x^l in the input layer and perform forward propagation.
- Step 2. Let $y^l = (y_1^l, ..., y_m^l)$ be the output layer and let $t^l = (t_1^l, ..., t_m^l)$ be the target vector.
- Step 3. For every output neuron $y_i^l, i = 1, ..., m$ calculate its responsibility $(\delta_i^{(1)})^l$ as

$$(\delta_i^{(1)})^l = (t_i^l - y_i^l) y_i^l (1 - y_i^l). \tag{37}$$

- Step 4. For every hidden neuron compute responsibility $(\delta_j^{(2)})^l$ for the network's error as

$$(\delta_j^{(2)})^l = h_j^l (1 - h_j^l) \cdot \sum_i (\delta_i^{(1)})^l (\omega_{ji}^{(1)})^l, \tag{38}$$

where $(\delta_i^{(1)})^l$ are computed using (37).

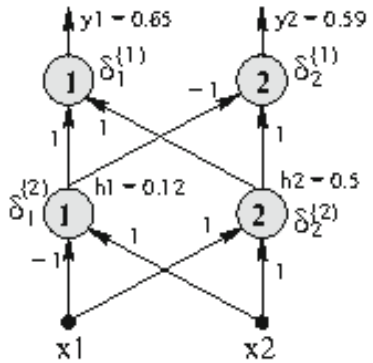
- Step 5. Update weights with learning rate $\eta^l \in (0, 1)$ as

$$\begin{aligned} (\omega_{ji}^{(1)})^{l+1} &= (\omega_{ji}^{(1)})^l + \eta^l (\delta_i^{(1)})^l x_j^l, \\ (\omega_{kj}^{(2)})^{l+1} &= (\omega_{kj}^{(2)})^l + \eta^l (\delta_j^{(2)})^l x_k^l. \end{aligned} \tag{39}$$

- Step 6. If the mean square error less than tolerance, or $\|(\omega_{ji}^{(1)})^{l+1} - (\omega_{ji}^{(1)})^l\| < \epsilon_1$ and $\|(\omega_{kj}^{(2)})^{l+1} - (\omega_{kj}^{(2)})^l\| < \epsilon_2$ stop, otherwise go to the next layer $l = l + 1$, assign $x^l = x^{l+1}$ and return to the step 1. Here, ϵ_1, ϵ_2 are tolerances chosen by the user.

Example of backpropagation of error through the network

Source: M. Kurbat, *An Introduction to machine learning*, Springer, 2017.



- Assume that after forward propagation with sigmoid transfer function we have

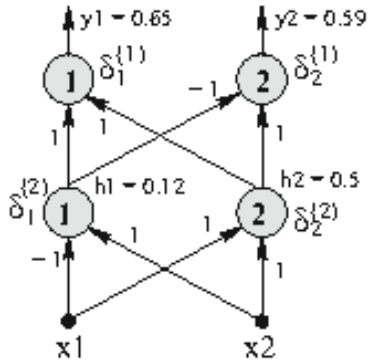
$$h_1 = f(x_1^{(2)}) = 0.12,\ h_2 = f(x_2^{(2)}) = 0.5,$$
$$y_1 = f(x_1^{(1)}) = 0.65,\ y_2 = f(x_2^{(1)}) = 0.59.$$

- Let the target vector be $t(x) = (1, 0)$ for the output vector $y = (0.65, 0.59)$.
- Compute responsibility for the output neurons:

$$\sigma_1^{(1)} = y_1 * (1 - y_1)(t_1 - y_1) = 0.65(1 - 0.65)(1 - 0.65) = 0.0796,$$
$$\sigma_2^{(1)} = y_2 * (1 - y_2)(t_2 - y_2) = 0.59(1 - 0.59)(0 - 0.59) = -0.1427$$

Example of backpropagation of error through the network

Source: M. Kurbat, *An Introduction to machine learning*, Springer, 2017.



- Compute the weighted sum for every hidden neuron

$$\delta_1 = \sigma_1^{(1)} w_{11}^{(1)} + \sigma_2^{(1)} w_{12}^{(1)} = 0.0796 * 1 + (-0.1427) * (-1) = 0.2223,$$
$$\delta_2 = \sigma_1^{(1)} w_{21}^{(1)} + \sigma_2^{(1)} w_{22}^{(1)} = 0.0796 * 1 + (-0.1427) * 1 = -0.0631.$$

- Compute responsibility for the hidden neurons for above computed δ_1, δ_2:

$$\sigma_1^{(2)} = h_1(1-h_1)\delta_1 = -0.0235,\ \sigma_2^{(2)} = h_2(1-h_2)\delta_2 = 0.0158.$$

Example of backpropagation of error through the network

- Compute new weights $\omega_{ji}^{(1)}$ for output layer with learning rate $\eta = 0.1$ as:

$$\omega_{11}^{(1)} = \omega_{11}^{(1)} + \eta\sigma_1^{(1)} h_1 = 1 + 0.1 * 0.0796 * 0.12 = 1.00096,$$
$$\omega_{21}^{(1)} = \omega_{21}^{(1)} + \eta\sigma_1^{(1)} h_2 = 1 + 0.1 * 0.0796 * 0.5 = 1.00398,$$
$$\omega_{12}^{(1)} = \omega_{12}^{(1)} + \eta\sigma_2^{(1)} h_1 = -1 + 0.1 * (-0.1427) * 0.12 = -1.0017,$$
$$\omega_{22}^{(1)} = \omega_{22}^{(1)} + \eta\sigma_2^{(1)} h_2 = 1 + 0.1 * (-0.1427) * 0.5 = 0.9929.$$

- Compute new weights $\omega_{kj}^{(2)}$ for hidden layer with learning rate $\eta = 0.1$ as:

$$\omega_{11}^{(2)} = \omega_{11}^{(2)} + \eta\sigma_1^{(2)} x_1 = -1 + 0.1 * (-0.0235) * 1 = -1.0024,$$
$$\omega_{21}^{(2)} = \omega_{21}^{(2)} + \eta\sigma_1^{(2)} x_2 = 1 + 0.1 * (-0.0235) * 1 = 1.0024,$$
$$\omega_{12}^{(2)} = \omega_{12}^{(2)} + \eta\sigma_2^{(2)} x_1 = 1 + 0.1 * 0.0158 * 1 = 1.0016,$$
$$\omega_{22}^{(2)} = \omega_{22}^{(2)} + \eta\sigma_2^{(2)} x_2 = 1 + 0.1 * 0.0158 * (-1) = 0.9984.$$

- Using computed weights for hidden and output layers, one can test a neural network for a new example.

- Non-regularized neural network

$$F(w) = \frac{1}{2}\|t_i - y_i(w)\|^2 = \frac{1}{2}\sum_{i=1}^{m}(t_i - y_i(w))^2. \quad (40)$$

- Regularized neural network

$$F(w) = \frac{1}{2}\|t_i - y_i(w)\|^2 + \frac{1}{2}\gamma\|w\|^2 = \frac{1}{2}\sum_{i=1}^{m}(t_i - y_i(w))^2 + \frac{1}{2}\gamma\sum_{j=1}^{M}|w_j|^2 \quad (41)$$

Here, γ is reg.parameter, $\|w\|^2 = w^T w = w_1^2 + ... + w_M^2$, M is number of weights.

Regularized neural network for perceptron

In regularized perceptron neural network we want to minimize

$$F(\omega) = \frac{1}{2}\|t - y(x,\omega)\|_2^2 + \frac{1}{2}\gamma\|w\|_2^2 = \frac{1}{2}\sum_{i=1}^{m}(t_i - y_i(x,\omega))^2 + \frac{1}{2}\gamma\sum_{i=1}^{m} w_i^2. \quad (42)$$

Here, t_i is the target function, or class c in the algorithm, which takes values 0 or 1.
The transfer function in perceptron is the following:

$$y(x,\omega) = \begin{cases} 1 & \text{if } \sum \omega_i x_i > 0, \\ 0 & \text{otherwise.} \end{cases} \quad (43)$$

Backpropagation of error through the network:perceptron

To find minimum of the functional (42) $F(\omega)$, recall it below:

$$F(\omega) = \frac{1}{2}\|t - y\|_2^2 + \frac{1}{2}\gamma\|w\|_2^2 = \frac{1}{2}\sum_{i=1}^{m}(t_i - y_i)^2 + \frac{1}{2}\gamma\sum_{i=1}^{m} w_i^2, \tag{44}$$

we need to solve the minimization problem

$$\min_{\omega} F(\omega) \tag{45}$$

and find a stationary point of (44) with respect to ω such that

$$F'(\omega)(\bar{\omega}) = 0, \tag{46}$$

where $F'(\omega)$ is the Fréchet derivative.

Backpropagation of error through the network: perceptron

Recall now that the transfer function in perceptron is defined as:

$$y(x,\omega) = \begin{cases} 1 & \text{if } \sum \omega_i x_i > 0, \\ 0 & \text{otherwise} \end{cases} \tag{47}$$

Then, for the functional (44) we have

$$F'_{\omega_i}(\omega)(\bar{\omega}_i) = (t_i - y_i) \cdot y'_{\omega_i}(\bar{\omega}_i) + \gamma\omega_i = (t_i - y_i) \cdot x_i + \gamma\omega_i. \tag{48}$$

Thus, we obtain perceptron algorithm with $c = t$ and $y = h$.

Perceptron regularized neural network

- Step 0. Initialize weights ω_i to small random numbers.
- Step 1. If $\sum_{i=0}^{n} \omega_i x_i > 0$ we will say that the example is positive and $h(x) = 1$.
- Step 2. If $\sum_{i=0}^{n} \omega_i x_i < 0$ we will say the the example is negative and $h(x) = 0$.
- Step 3. Update every weight ω_i using the algorithm of backpropagation of error through the network, or

$$\omega_i = \omega_i + \eta \cdot [[c(\mathbf{x}) - h(\mathbf{x})] \cdot x_i + \gamma \cdot \omega_i].$$

- Step 4. If $c(\mathbf{x}) = h(\mathbf{x})$ for all learning examples - stop. Otherwise return to step 1.

Here, $\eta \in (0, 1]$ is called the learning rate.