From perceptron to back-propagation

Functional Programming and Intelligent Algorithms

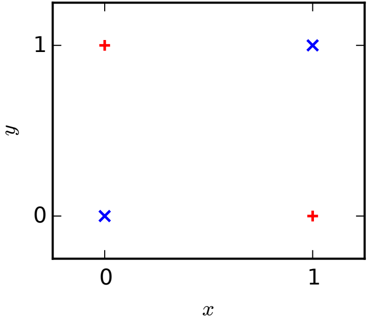
Prof Hans Georg Schaathun

Høgskolen i Ålesund

20th March 2017

The XOR problem

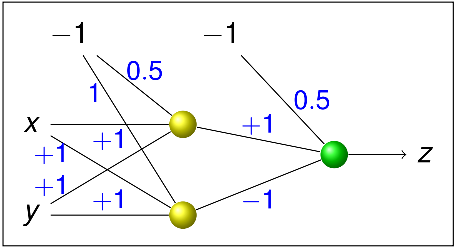
Features	Class
x, y	$x \oplus y$
0,0	0
0,1	1
1,0	1
1,1	0



Linear versus non-linear classifiers

1. Linear classifiers draw a hyperplane to separate classes.
    1.1 this may or may not succeed.
2. Other hyper-surfaces can be drawn instead
    2.1 quadratic, cubic, or polynomial in general
    2.2 many different classes of non-polynomial surfaces
3. Add neurons in hidden layer(s)
    3.1 no direct connection to input or output
    3.2 gives non-linear classifiers

Multi-layer perceptrons



Optimisation problem

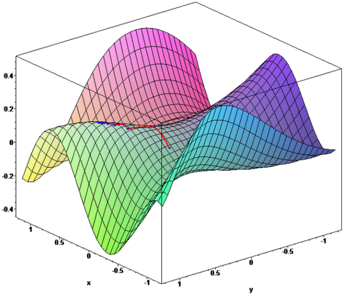
$$\min_{\mathbf{w}} |\mathbf{y} - \mathbf{t}|$$

1. We are allowed to choose the weights **w**
2. We aim reduce the error $\mathbf{y} - \mathbf{t}$

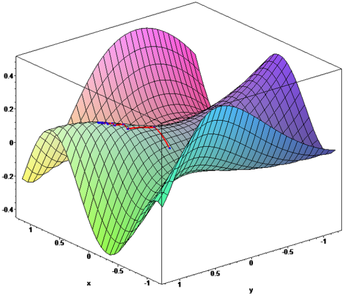
Error function

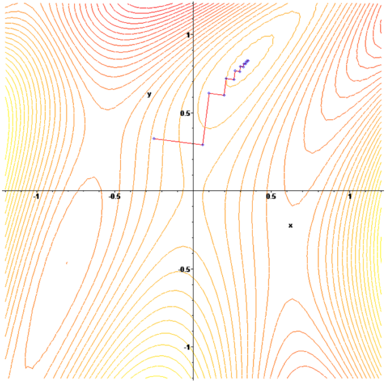
1. Different error functions are possible
    1.1 $E(\mathbf{y} - \mathbf{t}) = |\mathbf{y} - \mathbf{t}|$
    1.2 $E(\mathbf{y} - \mathbf{t}) = (\mathbf{y} - \mathbf{t})^2$
2. Similar optimisation problem
    - $\min_{\mathbf{w}} E(\mathbf{y} - \mathbf{t})$

Gradient descent



Gradient descent





Gradient descent with the perceptron

Recall $$y = g(\sum w_i x_i)$$

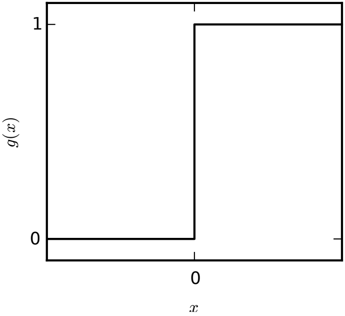
Error $$E(y,t) = (y-t)^2$$

Activation $$g(h) = \begin{cases} 0 & \text{when } h < 0 \\ 1 & \text{otherwise} \end{cases}$$

Exercise

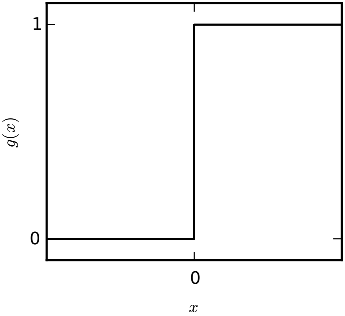
Differentiate $\frac{\partial E}{\partial w_i}$

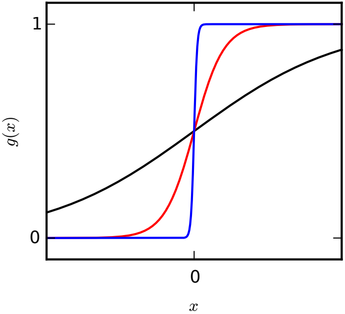
The activation function



$$g(h) = \begin{cases} 0 & \text{when } h < 0 \\ 1 & \text{otherwise} \end{cases}$$

The activation function





$$g(h) = \begin{cases} 0 & \text{when } h < 0 \\ 1 & \text{otherwise} \end{cases}$$

$$g(h) = \frac{1}{1 + \exp(-\beta h)}$$

Gradient descent with the sigmoid

Recall	$y = g(\sum w_i x_i)$
Error	$E(y,t) = (y-t)^2$
Activation	$g(n) = \dfrac{1}{1+\exp(-\beta h)}$
Derivative	$g'(n) = g(h)\big(1-g(h)\big)$

Exercise

Differentiate $\frac{\partial E}{\partial w_i}$

Gradient descent with the sigmoid

Recall $$y = g\big(\sum w_i x_i\big)$$

Error $$E(y,t) = \frac{1}{2}(y-t)^2$$

Activation $$g(n) = \frac{1}{1+\exp(-\beta h)}$$

Derivative $$g'(n) = g(h)\big(1-g(h)\big)$$

$$\frac{\partial E}{\partial w_i} = (y-t)y(1-y)x_i, \tag{1}$$

(2)

Gradient descent with the sigmoid

Recall $$y = g\left(\sum w_i x_i\right)$$

Error $$E(y,t) = \frac{1}{2}(y-t)^2$$

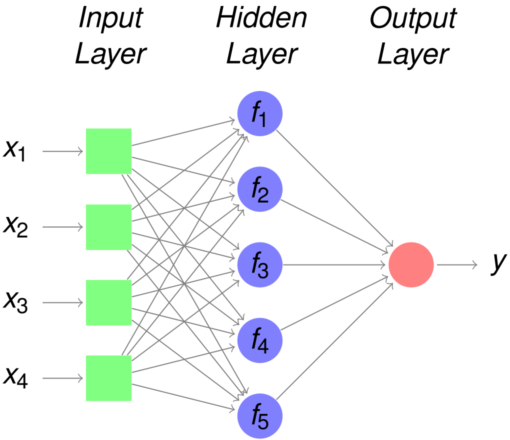
Activation $$g(n) = \frac{1}{1+\exp(-\beta h)}$$

Derivative $$g'(n) = g(h)\big(1-g(h)\big)$$

$$\frac{\partial E}{\partial w_i} = (y-t)y(1-y)x_i, \tag{1}$$

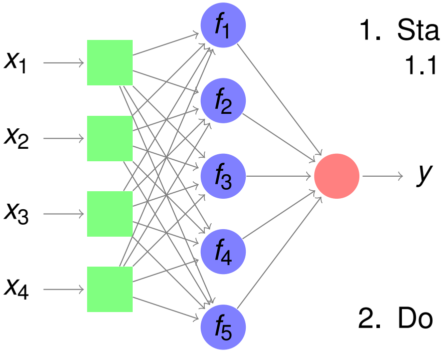
$$w_i := w_i - \eta(y-t)x_i. \tag{2}$$

Initialisation



All the weights are initialised with small random numbers.

Recall



1. Start with the hidden layer
    1.1 Do perceptron recall for each neuron in turn
        1.1.1 read the inputs
        1.1.2 calculate the weighted sum of inputs
        1.1.3 evaluate the activation function
        1.1.4 output 0 or 1, making input to the next layer
2. Do the same for the output layer

Training

- Calculate the gradient $\frac{\partial E(\mathbf{x},\mathbf{t})}{\partial w_i}$ for each output weight w_i
  - $\frac{\partial E}{\partial w_i} = \delta x'_i$
  - where $\delta = (y - t)y(1 - y)$
  - where x'_i is the output from the hidden layer
- Update the weights as before
  - $w_i := w_i - \eta \delta x'_i$

Gradients for the hidden layer

Recall	$y = g\big(\sum w_i x'_i\big)$	Output layer
Recall	$x'_i = g\big(\sum_j v_{i,j} x_j\big)$	Hidden layer
Error	$E(y,t) = \frac{1}{2}(y-t)^2$	
Activation	$g(n) = \frac{1}{1+\exp(-\beta h)}$	
Derivative	$g'(n) = g(h)\big(1-g(h)\big)$	

Exercise

Differentiate $\frac{\partial E}{\partial v_{i,j}}$

The back-propagation algorithm

- Update each output weight w_i, using
  - $w_i := w_i - \eta \delta x'_i$
  - where $\delta = (y - t)y(1 - y)$
- Calculate the gradient $\frac{\partial E(\mathbf{x},\mathbf{t})}{\partial w_{i,j}}$ for each hidden weight $v_{i,j}$
  - $\frac{\partial E}{\partial v_{i,j}} = \delta'_i x_j$
  - where $\delta'_i = x'_i(1 - x'_i)\delta w_i$
- Update the weights as before
  - $v_{i,j} := v_{i,j} - \eta \delta'_i x_j$

Implementation

1. Building on your implementation in Haskell
    1.1 implement the back-propagation algorithm
    1.2 test your implemenentation
    1.3 refactor the code so that it is easy to
        — initialise a network with one hidden layer
        — choose the number of nodes in each layer
        — train the network
        — test the network
2. Test your implementation with one hidden layer
    2.1 test different numbers of hidden neurons
    2.2 test on different data sets
        2.2.1 breast cancer data
        2.2.2 iris data
        2.2.3 optionally, find more data sets