Lecture 1.3

Perceptrons and multi-layer perceptrons

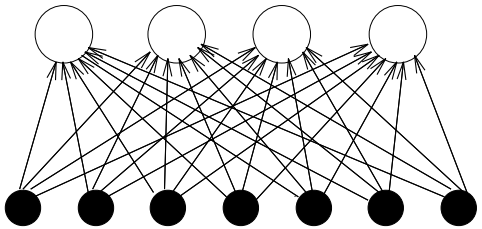
Jonathan Shapiro

Department of Computer Science, University of Manchester

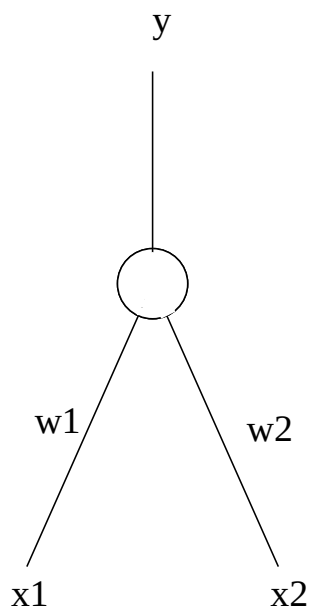
February 2, 2003

Simple Perceptrons

A layer of hard-limiting neurons.



We will consider a single neuron (since each neuron in a layer acts independently).

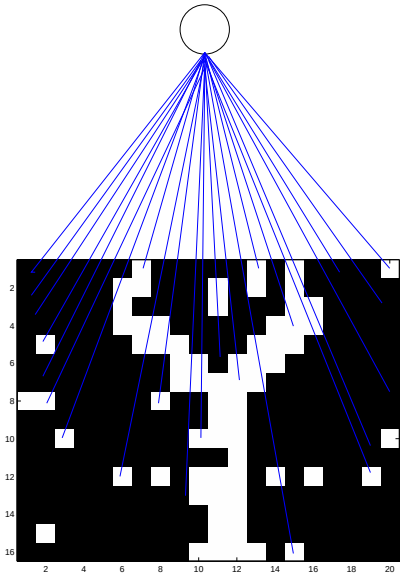


- Divides input patterns into two classes:
  - those which produce output 1;
  - those which produce output 0.
- Weights w_i and bias b control the division into the two classes.

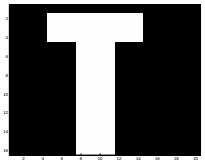
What can a perceptron compute?

Two views:

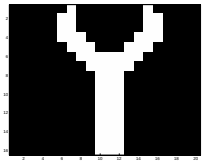
- Weighted sum of evidence from *individual* input components;



Is it:

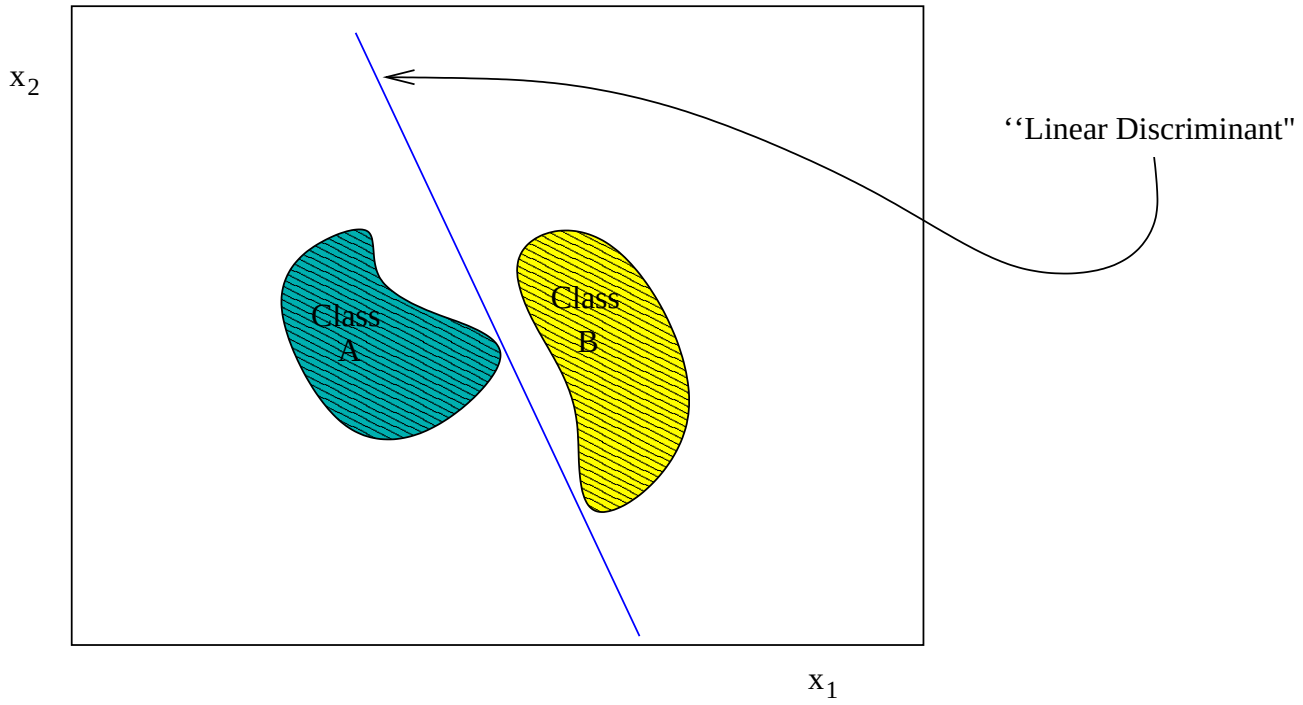


This?



or this?

- Separates the two classes with a line (plane, hyperplane, etc.).



Problems 1

problem 1.1 *Consider using a perceptron as a simple gate. It takes two inputs, each input can be* 0 *(meaning false) or* 1 *(meaning true). Using both the weighted-sum-of-evidence view and the linear discriminant view, discuss whether the perceptron can do these tasks: AND, XOR.*

problem 1.2 *Think of examples of image processing tasks which can and cannot be classified by summing evidence from individual components.*

A simple learning algorithm

Perceptron Learning Algorithm (PLA)

repeat

get input x^p and *target class* t^p (pattern p).

if the network correctly classifies the input pattern make no change

if the network classifies incorrectly

if target class is $t^p = 1$ (class 1)

$w_i = w_i + x_i^p$ for all i

$b = b + 1$

if target class is $t^p = 0$ (class 0)

$w_i = w_i - x_i^p$ for all i

$b = b - 1$

until all patterns correctly classified

A Hebb-like algorithm, where

- Desired output is ± 1;
- Only learns after a mistake;
- bias is like a weight to an input which is always 1.

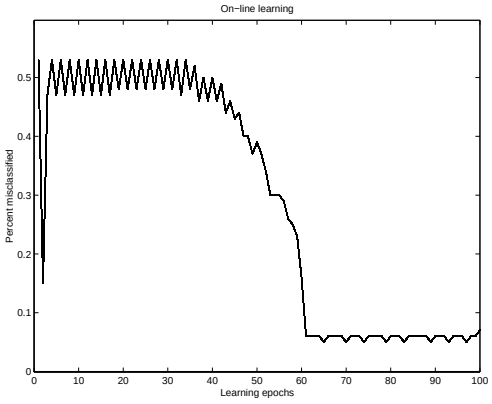
Results

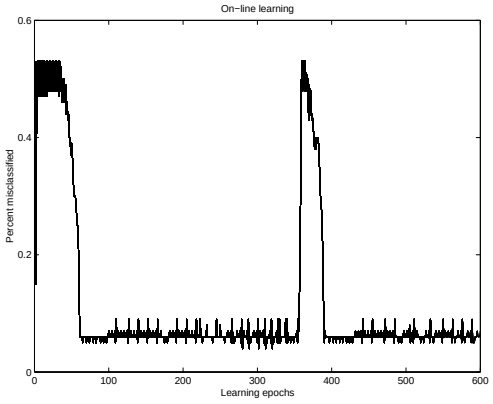
A simple system which learns from examples.

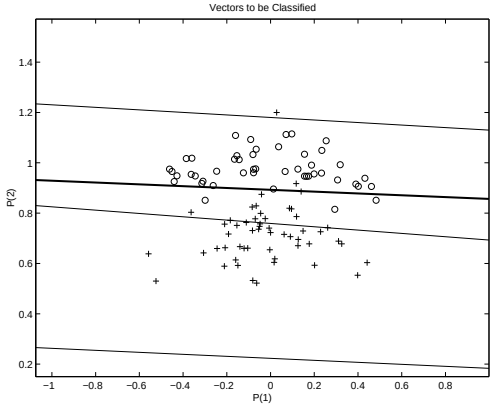
Realizable case: If there exists a set of weights which give correct classification for *all* training data, this algorithm will terminate.

All training patterns will be correctly classified.

Unrealizable case: If there are no weights which give perfect performance:



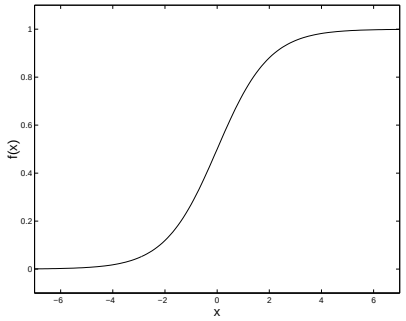




Gradient Descent Learning

Gradient descent cannot be applied to perceptrons, because the network output is not a continuous function of the weights.

Solution: replace hard-limiting function with a "sigmoid" function, a continuous function $f(x) \in [0,1]$.



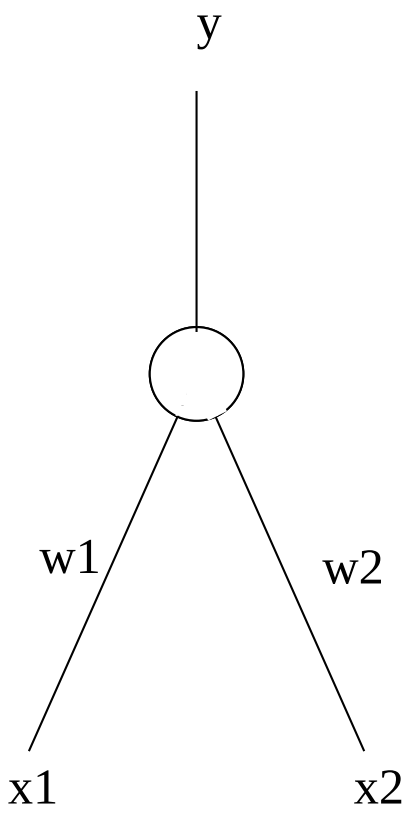
Gradient: For the perceptron,

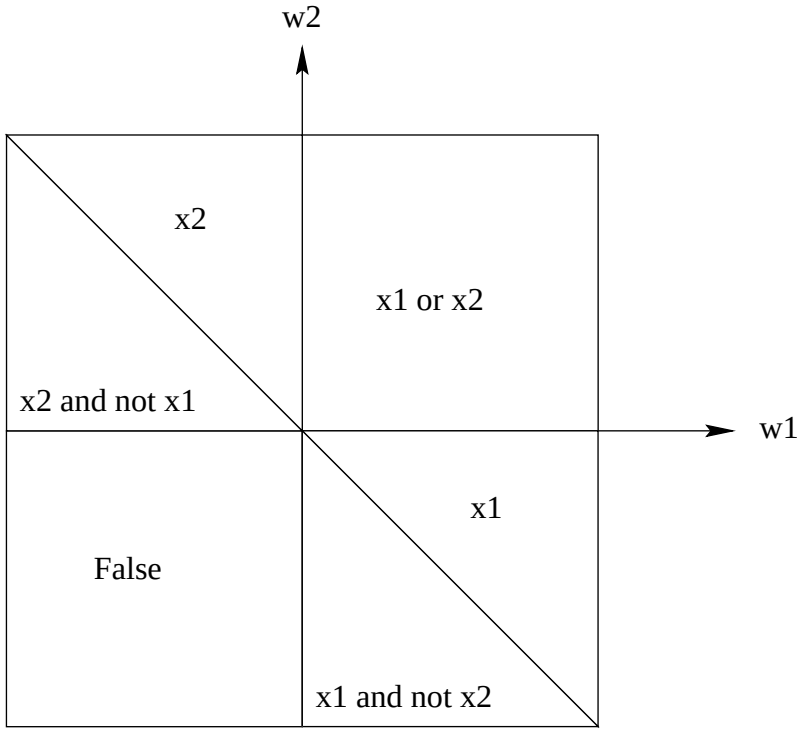
$$\frac{\partial E}{\partial w_i} = (y^p - t^p)\, f' \left(\sum_j w_j x_j + b \right) x_i$$

where $f'()$ is the derivative of the sigmoid function.

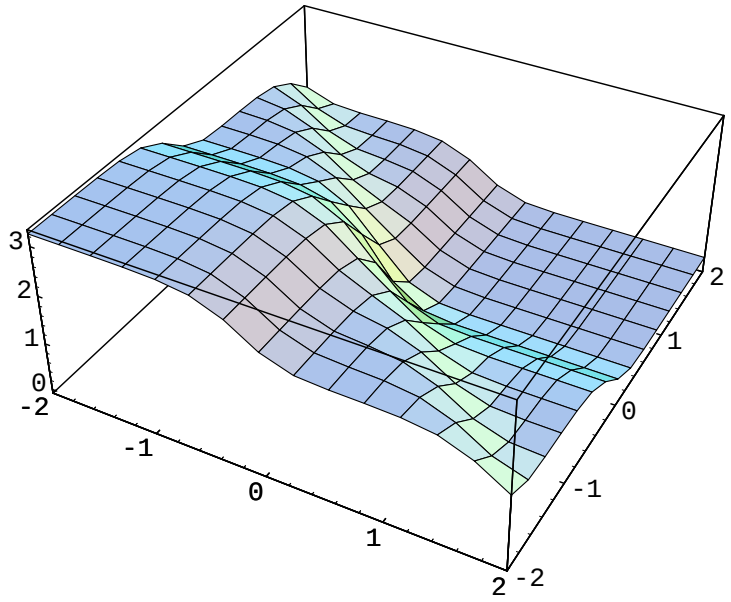
Example

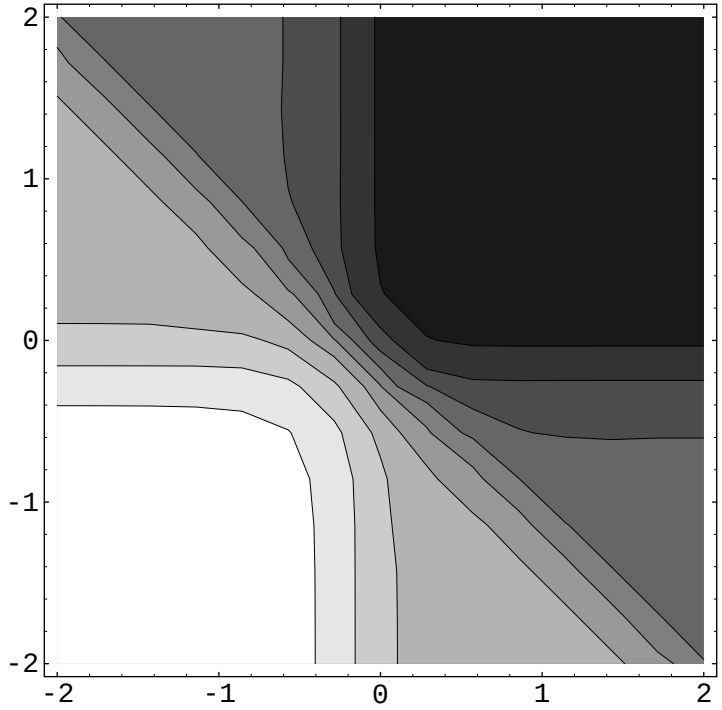
Train a two input perceptron *with no bias* to perform OR of its inputs.





Example (Continued)





Gradient descent in single layer networks

To use sigmoid network for classification, either

- threshold the output. E.g. $y > 0.5$ is for one class; $y <= 0.5$ is for the other
- Interprete the output as the probability of the class.

Sigmoid neurons can also be used for regression (continuous output) problems.

Another cost function

For classification problems, it is useful to interpret the network output interpreted as the probability of the class. Then, get the network to produce outputs which make the training data as likely as possible.

probability of a training pattern: if t^p is the assigned class $(0/1)$, and y^p is the probability of this pattern being in class 1, the probability of the assignment can be written

$$\left(y^p\right)^{t^p}\left(1-y^p\right)^{1-t^p} \tag{1}$$

log probability of the training data: Thus, the log probability of the training data under these assumptions is

$$\sum_p [t^p \log y^p + (1 - t^p) \log (1 - y^p)] \tag{2}$$

One can use gradient descent to *maximize* the log probability of the training data.

This error measure is called cross-entropy error.

Problems 2

problem 2.3 *Derive equations 1 and 2.*

problem 2.4 *Show that if all patterns are identical, but have been put in different classes, the network output which maximises the probability of the training data is the fraction of patterns classified as class* 1*.*

problem 2.5 *Find the gradient descent learning rule for a single node with sigmoidal output with the cross-entropy error measure.*

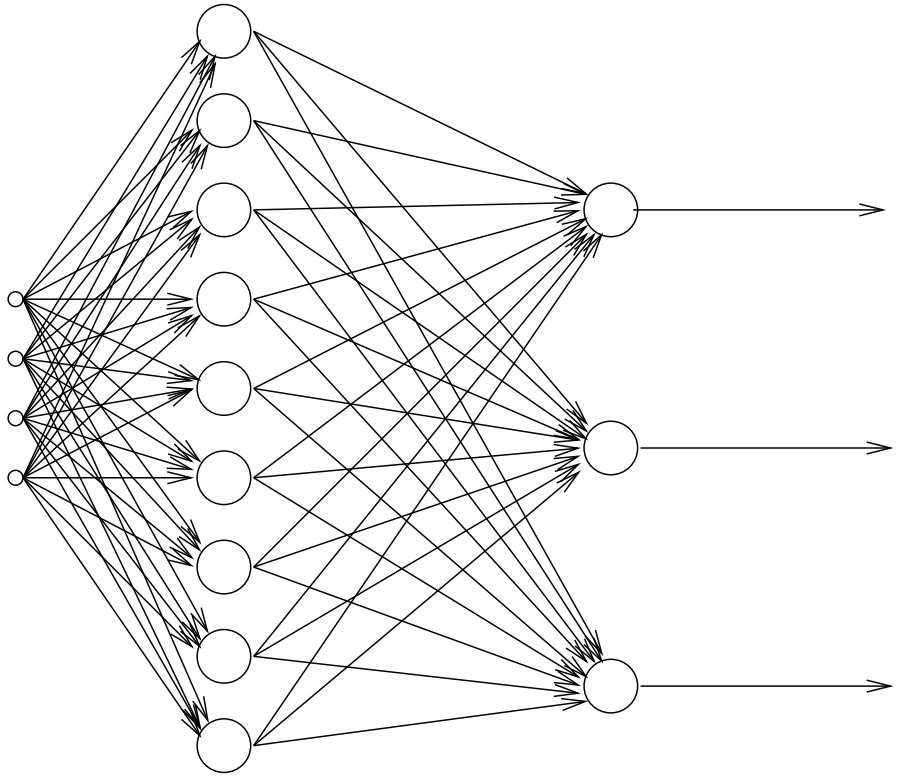
Summary of Perceptrons

- A simple, non-linear layer of hard-limiting or sigmoidal neurons.
- Can classify using a linear discriminant or weighted sum of evidence from *independent* components.
- Perceptron Learning Algorithm will learn any task which the hardlimiting perceptron can compute perfectly.
- Gradient descent learning can be used on sigmoid units.

What about tasks which perceptrons cannot compute?

Multi-layer perceptrons

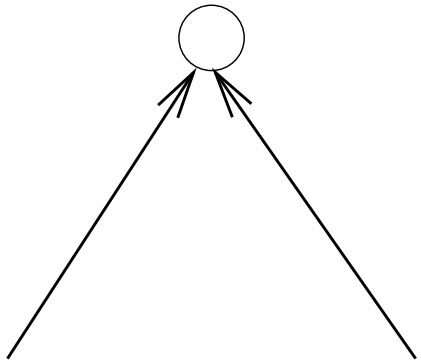
MLPs: Feed-forward neural network with multiple layers of processing neurons.

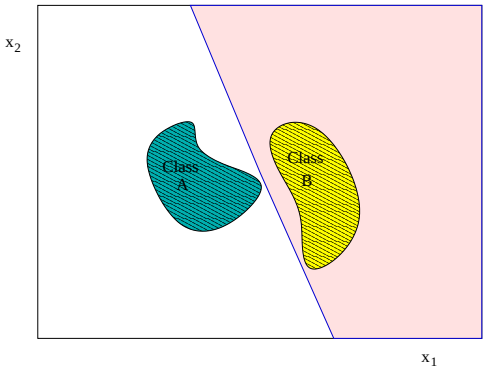


What can MLPs compute? Can be used to form arbitrarily complex relationships between input and output (given a sufficiently complex network architecture).

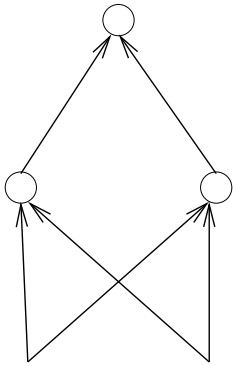
- For any classification problem, there is an network with no more than two layers of hidden units which can represent it.
- For any smooth function between input and output spaces there is a MLP with no more than two layers of hidden units which can represent it.

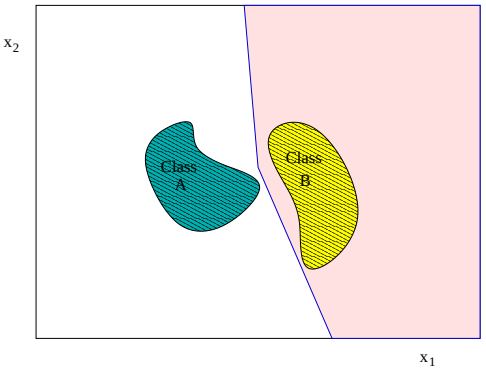
Single Layer



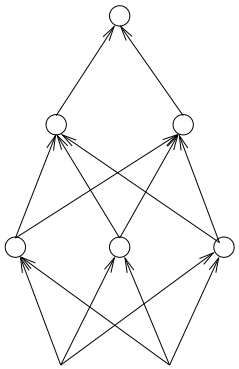


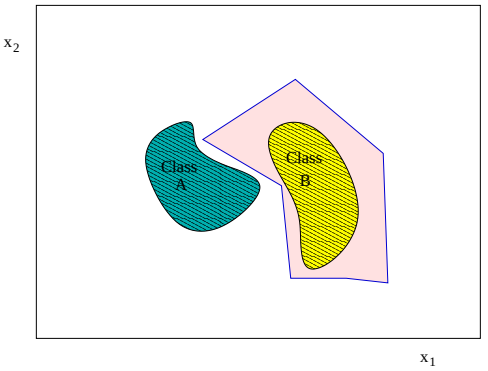
Two Layer





Three Layer





How to train MLPs?

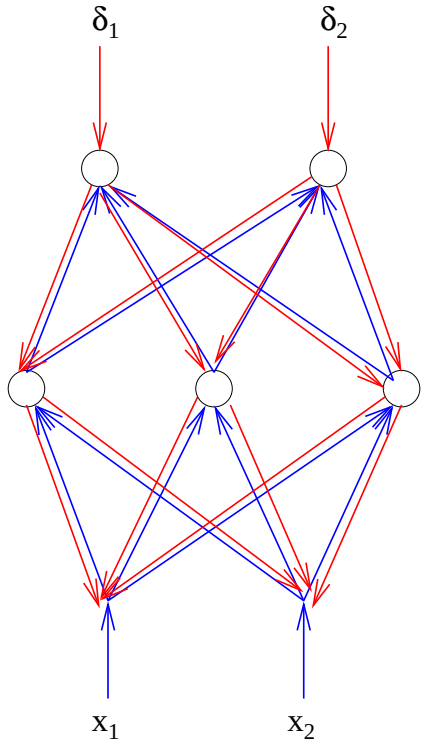
- No local Hebbian rule known.
- Inputs affect outputs through all of the hidden nodes — a complex problem to set the weights.

Backpropagation

A efficient gradient descent algorithm for MLPs.

Forward pass: to compute network output

Backward pass: propagate error back through the network to recursively compute weight changes (via chain rule differentiation).



More details of the algorithm in the notes.

Forward pass: compute all neuron outputs y_i^k for the i node in the mth layer

Backward pass: compute the weight changes according to "generalized" δ-rule,

$$w_{ij}^m \leftarrow w_{ij}^m + s\delta_i^m y_j^{m-1}$$

where δ's are computed recursively, starting at the output layer,

$$\delta_i^M = f'(h_i^M)\left[t_i^k - y_i^M\right];$$

and working back

$$\delta_i^{m-1} = f'(h_i^{m-1})\sum_j w_{ji}^m \delta_i^m.$$

Here h_i^m is the weighted sum of inputs at the ith node in the mth layer.

An efficient computation of the gradient

- Let W be the number of weights.
- The computation of the error takes $O(W)$. There are W components to the derivative to compute, so you expect the computation of the gradient to be $O(W^2)$.
- Backprop is $O(W)$, just like the forward pass. A consequence of the chain rule of differentiation.

Result – a learning algorithm for a powerful learning model

There is no guarantee that backpropagation will find the best weights.

- Local minima and flat regions can exist,
- Stepsize problem can be important.
  - Use algorithm which adapts the stepsize
  - Use algorithm which uses second order information (better).
    Examples: conjugate gradient descent, LM algorithm,
    quasi-Newton algorithms.
- Learning can be slow and computationally expensive.