Artificial Neural Network (ANN) Paradigms

Overview:

The Backpropagation Network (BPN)
Supervised Learning in the BPN
The Self-Organizing Map (SOM)
Unsupervised Learning in the SOM

The Backpropagation Network

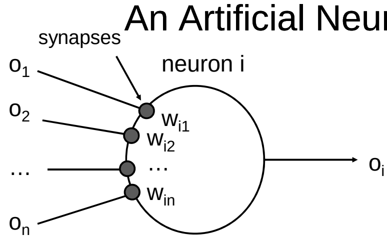
The backpropagation network (BPN) is the most popular type of ANN for applications such as classification or function approximation.

Like other networks using supervised learning, the BPN is not biologically plausible.

The structure of the network is as follows:

- Three layers of neurons,
- Only feedforward processing: input layer → hidden layer → output layer,
- Sigmoid activation functions

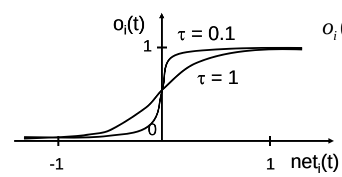
An Artificial Neuron



net input signal $$\text{net}_i(t) = \sum_{j=1}^{n} w_{ij}(t) o_j(t)$$

output signal $$o_i(t) = \frac{1}{1 + e^{-(\text{net}_i(t) - \theta)/\tau}}$$

Sigmoidal Neurons



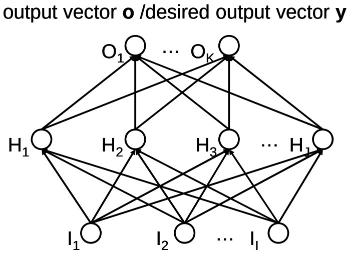
$$o_i(t) = \frac{1}{1 + e^{-(\text{net}_i(t) - \theta)/\tau}}$$

The output (spike frequency) of every neuron is simulated as a value between 0 (no spikes) and 1 (maximum frequency).

The Backpropagation Network

BPN structure:

output vector **o** /desired output vector **y**



input vector **x**

Supervised Learning in the BPN

In supervised learning, we train an ANN with a set of vector pairs, so-called **exemplars**.

Each pair (**x**, **y**) consists of an input vector **x** and a corresponding output vector **y**.

Whenever the network receives input **x**, we would like it to provide output **y**.

The exemplars thus describe the function that we want to "teach" our network.

Besides **learning** the exemplars, we would like our network to **generalize**, that is, give plausible output for inputs that the network had not been trained with.

Supervised Learning in the BPN

Before the learning process starts, all weights (synapses) in the network are **initialized** with pseudorandom numbers.

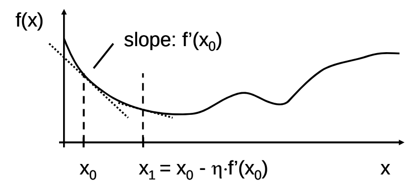
We also have to provide a set of **training patterns** (exemplars). They can be described as a set of ordered vector pairs $\{(x_1, y_1), (x_2, y_2), \ldots, (x_P, y_P)\}$.

Then we can start the backpropagation learning algorithm.

This algorithm iteratively minimizes the network's error by **finding the gradient** of the error surface in weight-space and **adjusting the weights** in the opposite direction (gradient-descent technique).

Supervised Learning in the BPN

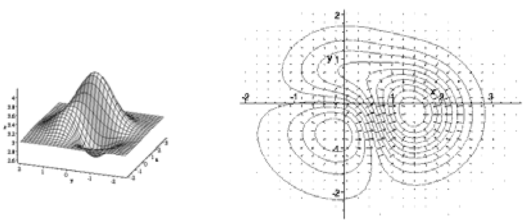
Gradient-descent example: Finding the absolute minimum of a one-dimensional error function f(x):



Repeat this iteratively until for some x_i, $f'(x_i)$ is sufficiently close to 0.

Supervised Learning in the BPN

Gradients of two-dimensional functions:



The two-dimensional function in the left diagram is represented by contour lines in the right diagram, where arrows indicate the gradient of the function at different locations. Obviously, the gradient is always pointing in the direction of the steepest increase of the function. In order to find the function's minimum, we should always move against the gradient.

Supervised Learning in the BPN

In the BPN, learning is performed as follows:

1. Randomly select a vector pair $(\mathbf{x_p}, \mathbf{y_p})$ from the training set and call it $(\mathbf{x}, \mathbf{y})$.
2. Use **x** as input to the BPN and successively compute the outputs of all neurons in the network (bottom-up) until you get the network output **o**.
3. Compute the error of the network, i.e., the difference between the desired output **y** and the actual output **o**.
4. Apply the backpropagation learning rule to update the weights in the network so that its output **o** for input **x** is closer to the desired output **y**.

Supervised Learning in the BPN

Repeat steps 1 to 4 for all vector pairs in the training set; this is called a training **epoch**.

Run as many epochs as required to reduce the network error E to fall below a **threshold** that you set beforehand.

Supervised Learning in the BPN

Now our BPN is ready to go!

If we choose the type and number of neurons in our network appropriately, after training the network should show the following behavior:

- If we input any of the training vectors, the network should yield the expected output vector (with some margin of error).
- If we input a vector that the network has never "seen" before, it should be able to generalize and yield a plausible output vector based on its knowledge about similar input vectors.