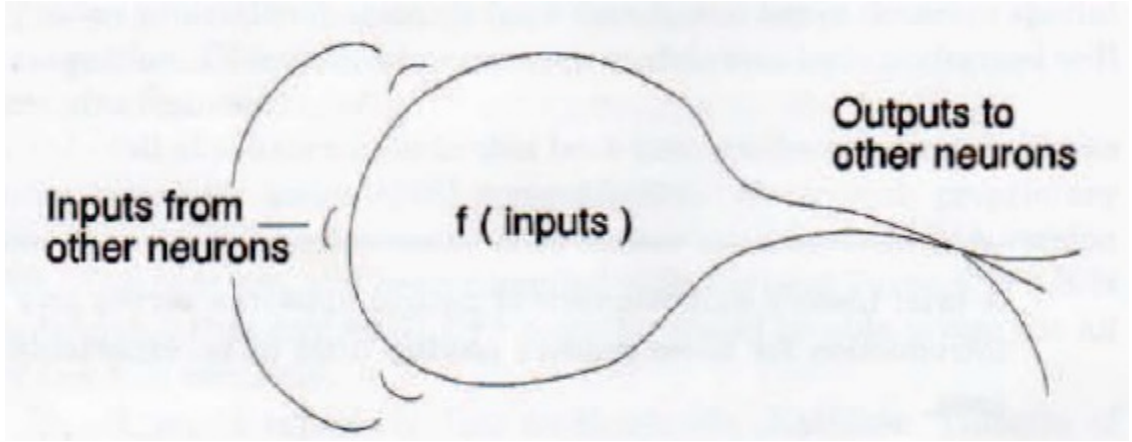
Neural Networks

Motivation

- From the time of the first primitive computing machines, their designers and users have been trying to push computers beyond the role of automatic calculators and into the realm of "thinking" machines.
- Methods for supposedly implementing human like thought processes with deterministic machine are likewise varied. Neural networks represent one of these approaches.

A biological neuron is a single cell capable of a sort of crude computation. It is stimulated by one or more inputs, and it generates an output that is sent to other neurons. The output is dependent on the strength of each of the inputs and on the nature of each input connection (**called a synapse**). Some synapses may be such that an input there will tend to excite the neuron (increase the output). Others may be inhibitory; an input to such synapses will tend to reduce the neurons output.



New Life for Old Techniques

A side effect of the surge in interest in neural networks is that some neglected statistical techniques are being rediscovered. When we say that something, is a «neural network», we imply that it **has a segmented structure**. A neural network is characterized by an architecture in which its operations are **distributed among many relatively simple processors**.

Perceptrons and Linear Separability

The *perceptron* [Rosenblatt, 1958] was the first artificial neural network. It was computationally feasible on the hardware of that time, was based on biological models, and was capable of learning. Unfortunately, it also suffered from a significant weakness: the inability to learn to perform an important family of classification tasks.

Training of the perceptron is accomplished by repeatedly presenting it with a sample input on the «retina», computing the outputs of the middle· layer neurons, using those values to compute the outputs of the response-layer neurons, and then updating the weights through which the middle layer connects to the response layer.

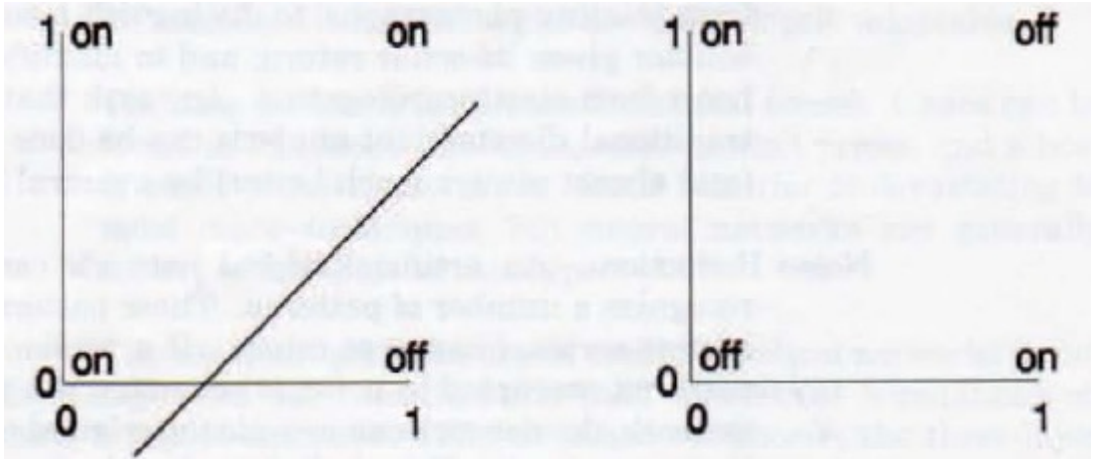
Unfortunately, the perceptron has a serious shortcoming. It is only capable of solving **classification problems** that are *linearly separable* at the output layer. For each output neuron, the training samples that activate it must be able to be separated from those that do not by means of a hyperplane.

Linear Separability

For example, suppose we have two inputs to a neuron, and the following responses are desired: This training data is shown in the left side of figure along with a line that can separate the set according to whether or not the output is to be activated. This training set can be learned by a perceptron. Compare this to the XOR (exclusive or) logical function in the right side of the figure. There is **no line** that can seperate this training set, so it cannot be learned by a perceptron.

in1	in2	out
0	0	1
1	0	0
0	1	1
1	1	1

in1	in2	out
0	0	0
1	0	1
0	1	1
1	1	0



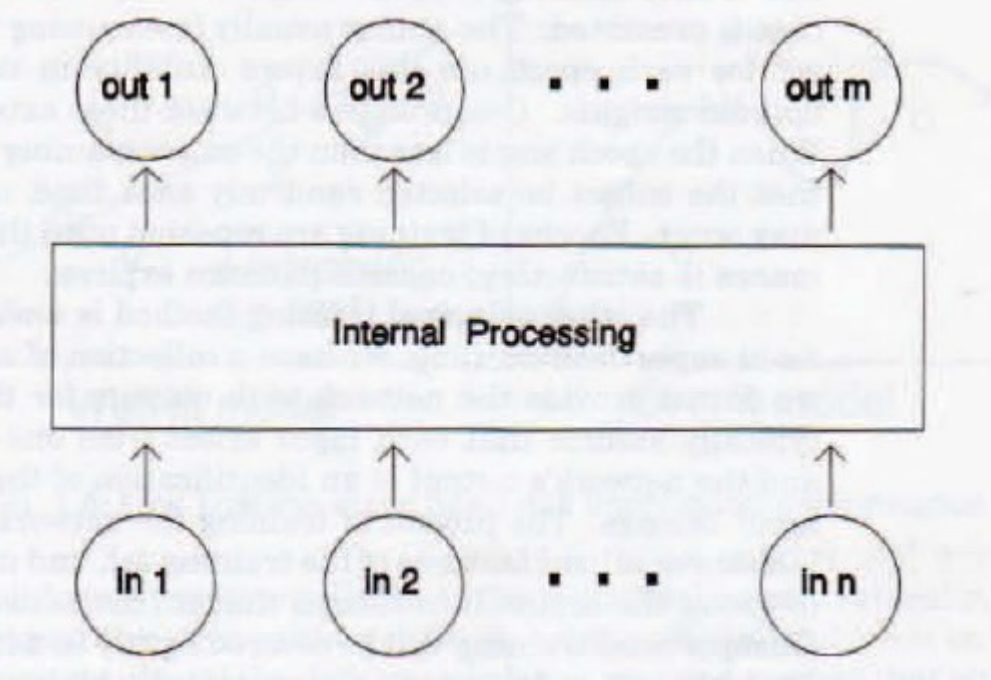
Neural Network Capabilities

Classification: Neural networks can be used to determine crop types from satellite photographs, to distinguish a submarine from a boulder given its sonar return, and to identify diseases of the heart from electrocardiograms. Any task that can be done by traditional discriminant analysis can be done at least as well (and almost always much better) by a neural network.

Noise Reduction: An artificial neural network can be trained to recognize a number of patterns. These patterns may be parts of time-series, images. et cetera. *If* a version of one of these patterns, corrupted by noise, is presented to a properly trained network, the network can provide the original pattern on which it was trained. This technique has been used with great success in some image restoration problems.

Prediction: A very common problem is that of predicting the value of a variable given historic values of itself (and perhaps other variables). Economic and meteorological models spring to mind. Neural networks have frequently been shown to outperform traditional techniques like ARIMA and frequency domain analysis.

Basic Structure of a Neural Network



Training

1. A neural network is (usually) trained in one of two ways. The most common is *supervised* training. We collect many samples to serve as exemplars. Each sample in this *training set* completely specifies all inputs, as well as the outputs that are desired when those inputs are presented.
2. Then we choose a subset of the training set and present the samples in that subset to the network one at a time. For each sample, we compare the outputs obtained by the network with the outputs we would like it to obtain.
3. After the entire subset of training samples has been processed, we update the weights that connect the neurons in the network. This **updating** is done in such a way that we (hopefully) reduce a measure of the error in the network's results.

Training

Epoch: One pass through the subset of training samples, along with an updating of the network's weights, is called an *epoch.*

Epoch Size: The number of samples in the subset is called the epoch size.

The other principal training method is *unsupervised training*. As in supervised training, we have a collection of sample inputs. But we do not provide the network with outputs for those samples. We typically assume that each input arises from one of several classes, and the network's output is an identification of the class to which its input belongs. The process of training the network consists of letting it discover significant features of the training set. and using these features to group the inputs into classes that it (the network) finds distinct.

There is a third, hybrid training method, which deserves brief mention. *Reinforcement learning* is unsupervised in that the exact outputs desired are not specified. At the same time, it is supervised in that when the network responds to a sample in the training set, it is told whether its response was good or bad.

Validation

The usual procedure is to separate the known cases into two disjoint sets. One is the training set, which is used to train the network. The other is the validation set, which is used to test the trained network.

There is a dangerous pitfall into which many practitioners tumble when in the validation phase of a project. It is imperative that the **validation set not be used as part of the training procedure** in any way whatsoever. Almost nobody "accidentally" lets part, or all the validation set slip into the training set. That is too obvious an **error**.

Why is the above process illegal? Remember that the purpose of the validation phase is to provide us with an unbiased indication of what can be expected of the network when it is used in the general population. By **including it in the training cycle, we have biased it**.

leave-k-out Method

There is a standard method for dealing with small training sets in traditional statistical techniques. It is called the ***leave-k-out*** method, also called ***jackknifing***. A small fraction of the known cases, often just one case, is held back for testing, and the rest of them are used for training. Then, the cases held back are classified. This is a fair, unbiased test of that particular discriminator's capability. Unfortunately. since just one or at most very few cases are tested, the random error inherent in the sampling procedure guarantees us a high probability of significant error in the performance estimate. So, we do it again, this time holding back a different small subset. By repeating this as many times as are needed to test every known case, we have made optimal use of our data, yet we still have a performance estimate that is not biased by ever using the same case for both training and testing.

This is OK for linear discriminant analysis but is not OK for neural networks. Therefore, we do not assert that a network trained with all of the known data is essentially identical to networks trained with subsets of the data.