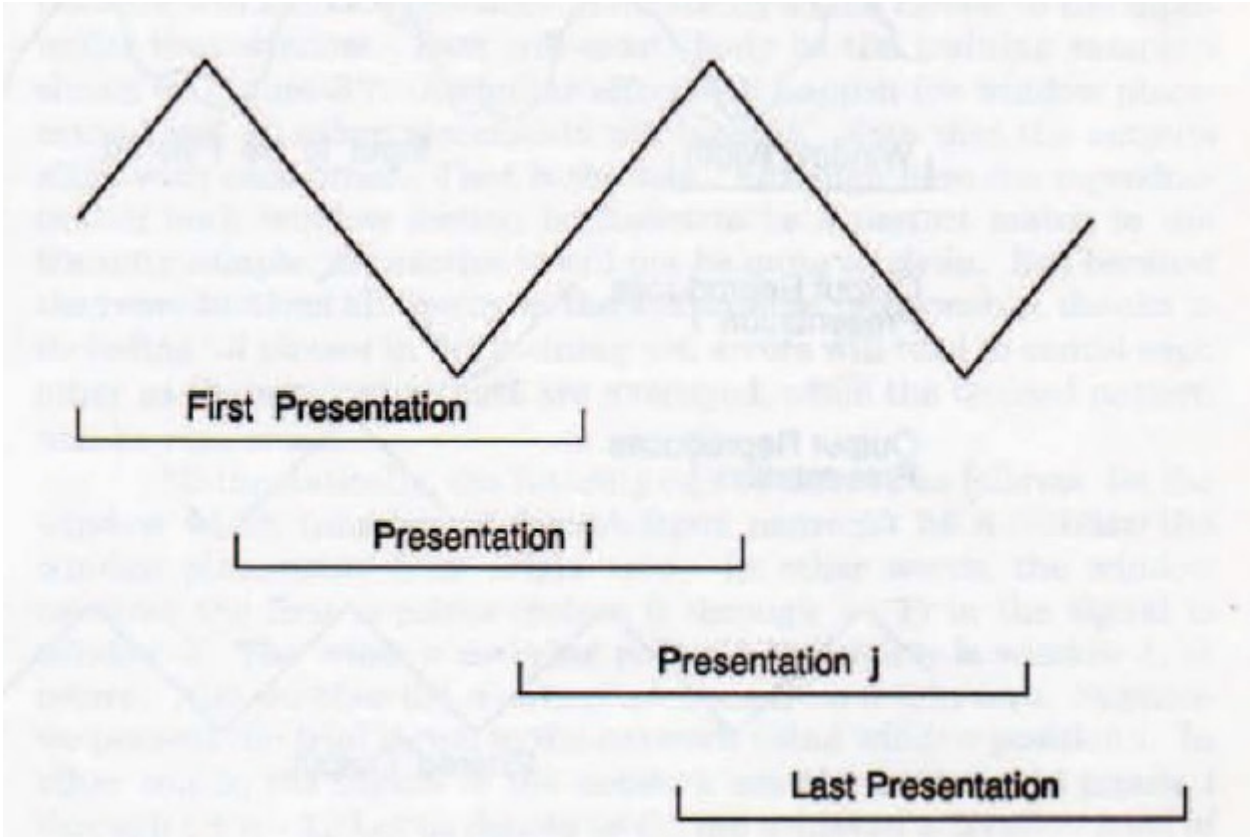
Autoassociation

Autoassociative Filtering

When a neural network has exactly as many output neurons as input neurons and is trained so that its outputs attempt to match its inputs for every member of a training set, it is said to be an *autoassociative* network.

The effect is that if a trained network is presented with an input pattern that resembles a pattern on which it was trained, its output will (hopefully) be close to the training pattern that most closely resembles the trial input. This means that if a trial input is nearly identical to one of the training patterns, except that it has been corrupted by noise or is partly missing, then the network will act as a noise filter or pattern completer.

Training

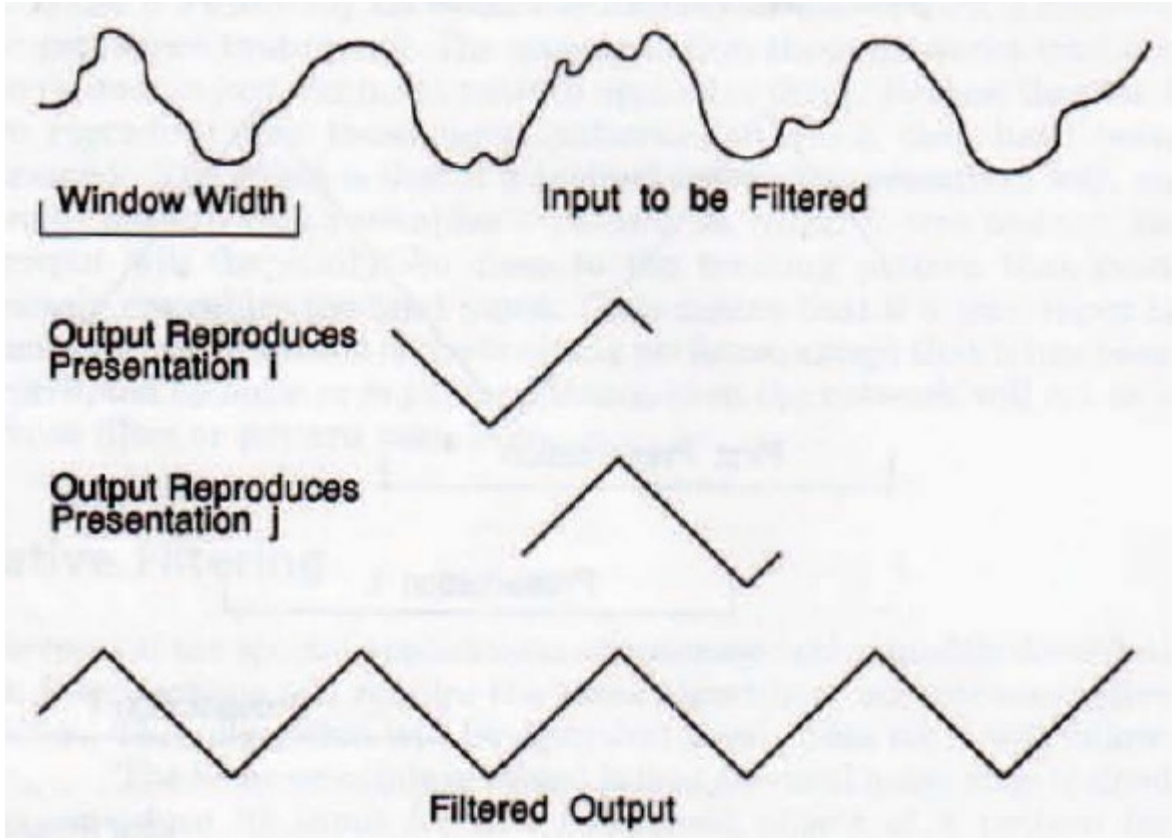

First Presentation
Presentation i
Presentation j
Last Presentation

Training

Each bracketed interval represents one training sample. It is generally best to **slide the window one point at a time**, so that maximum resolution is obtained. Two arbitrarily chosen sample windows are labeled i and j.

Note that the width of the sample window was set equal to ***the period of the pattern*** to be learned. This is the minimum necessary to do a reasonable job of filtering.

Filtering Process



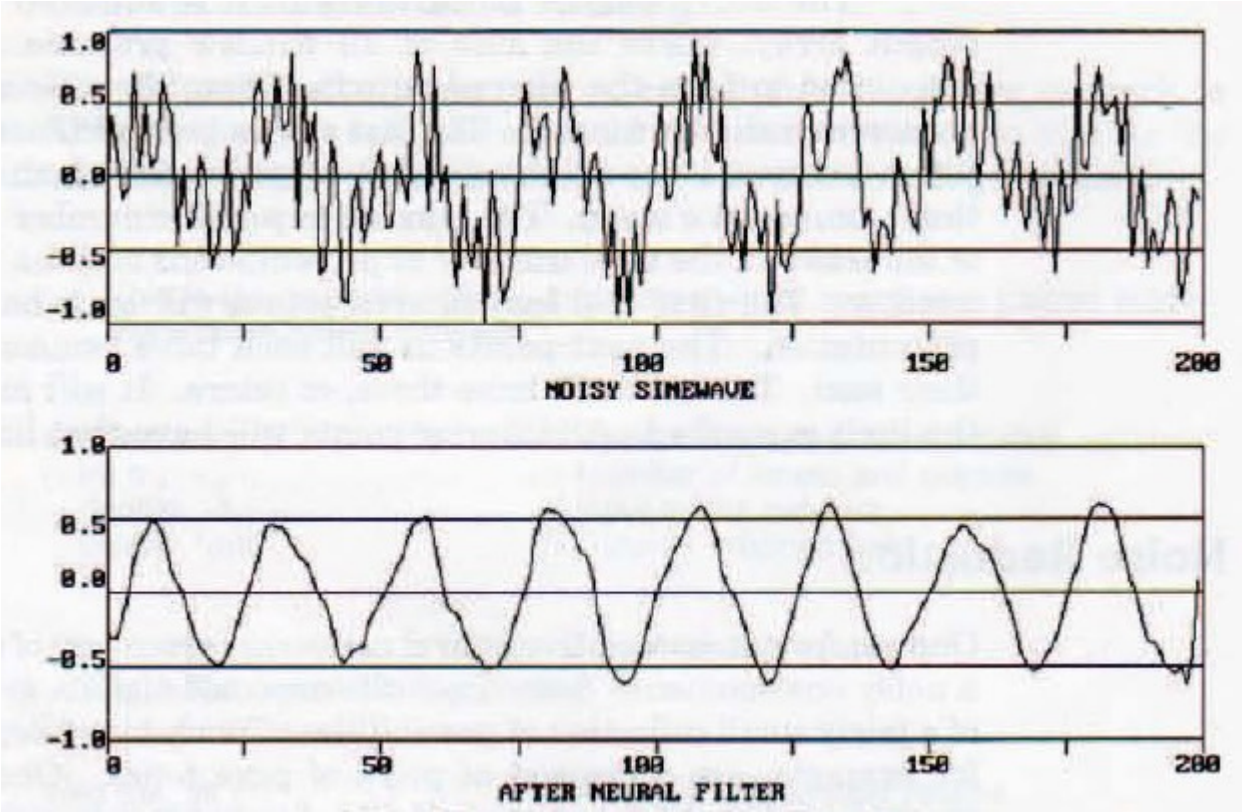
Filtering Process

Mathematically, the filtering can be defined as follows: let the window width (number of input/output neurons) be n. Index the window placements from origin zero. In other words. the window covering the first n points (points 0 through n-1) in the signal is window 0. The window covering points 1 through n is window 1, etcetera. Suppose we present the trial signal to the network using window position i. In other words, the inputs to the network are the trial signal inputs i through i + n - 1.

The sum of all window presentations will be cumulated to form the filtered output. Then, the presentations are done with a sliding window. **The last step is to divide each sum in the filtered output array by the number of presentations that went into that sum, to get a mean**. The maximum possible number in each sum is the lesser of the total number of presentations and the width of the window. The first and last filtered points will each have only one presentation. The next points in will each have two components in their sum. The next will have three, et cetera. It will increase until the limit is reached. *All interior points will have that limit*.

Noise Reduction

One use for autoassociative neural networks is recovery of signals from a noisy environment. Sometimes the expected signals are a member of a fairly small collection of possibilities. Touch-tone telephone codes, for example, are composed of pairs of pure tones. One can easily construct optimal filters for small sets of pure tones by using standard mathematical formulas. But such theoretically derived filters become complex when more than a few tones are involved. And they become nearly impossible to compute when the signals being sought are not pure tones.

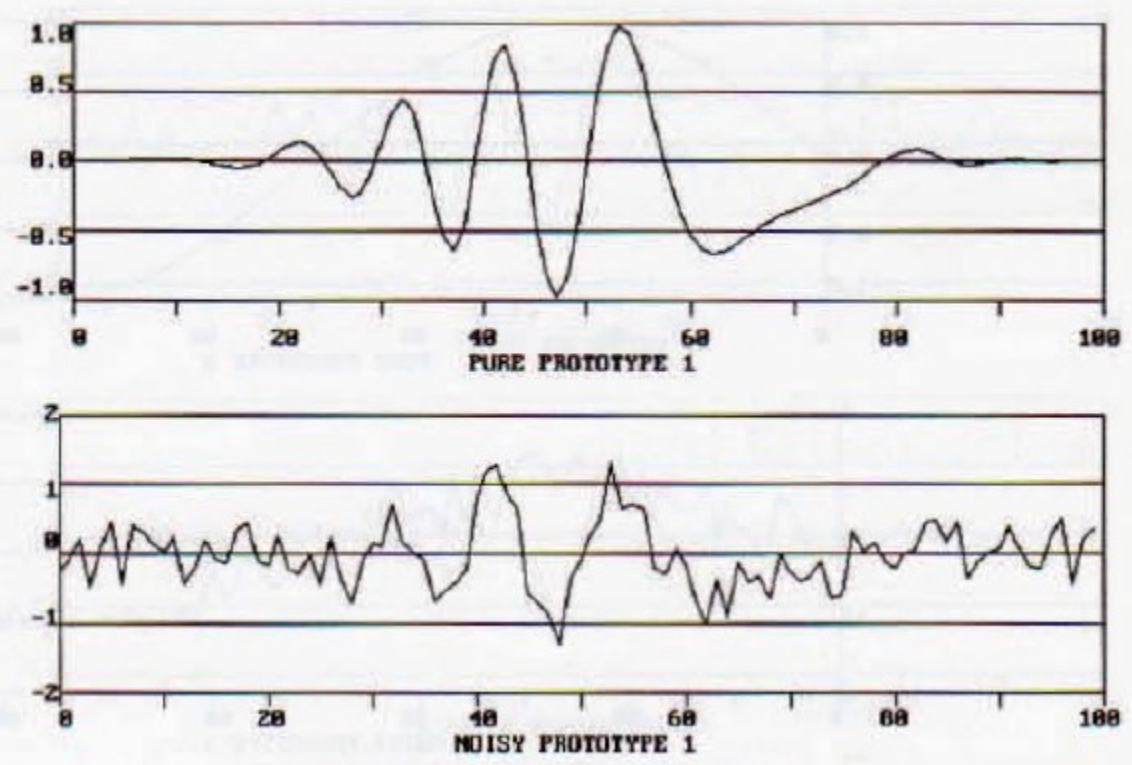


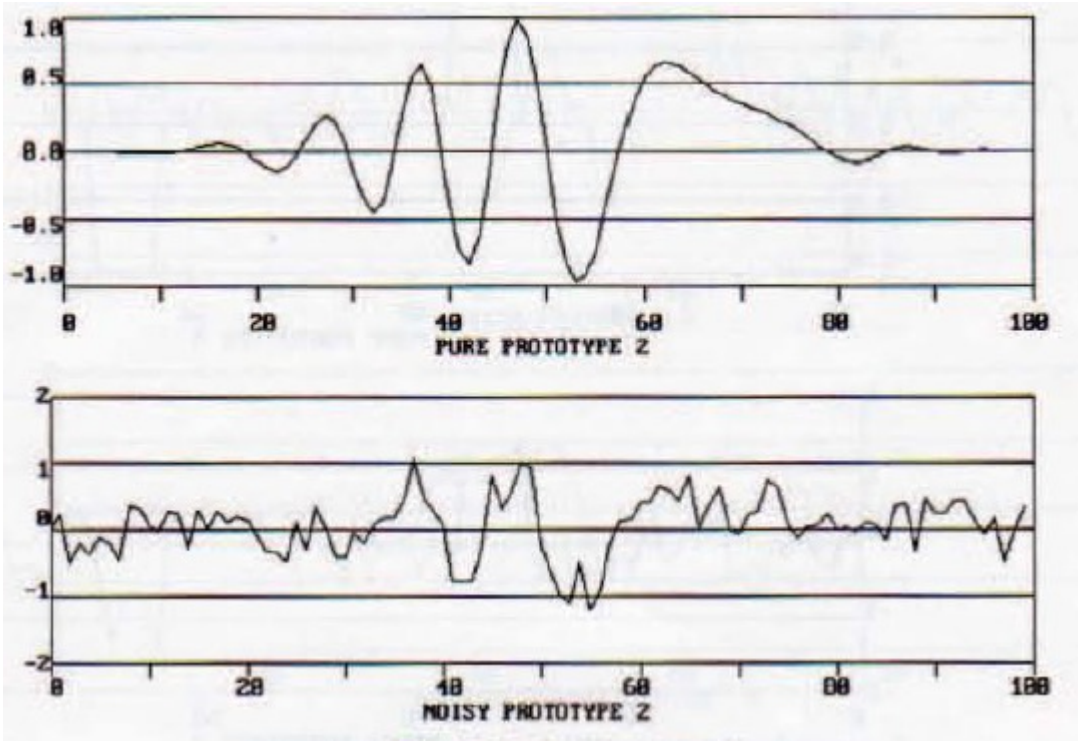
Noise Reduction

Two hundred points of a sine wave having a period of 25 were generated. Two hundred uniform random numbers on (-1 and 1) were also generated. A noisy sine wave was computed as the sum of 40 percent pure sine plus 60 percent random noise. The model chosen was a three-layer feedforward network having 25 input and output neurons and *10 hidden neurons*. The training set consisted of 25 samples of a pure sine wave, with the samples covering 25 equally spaced phase angles.

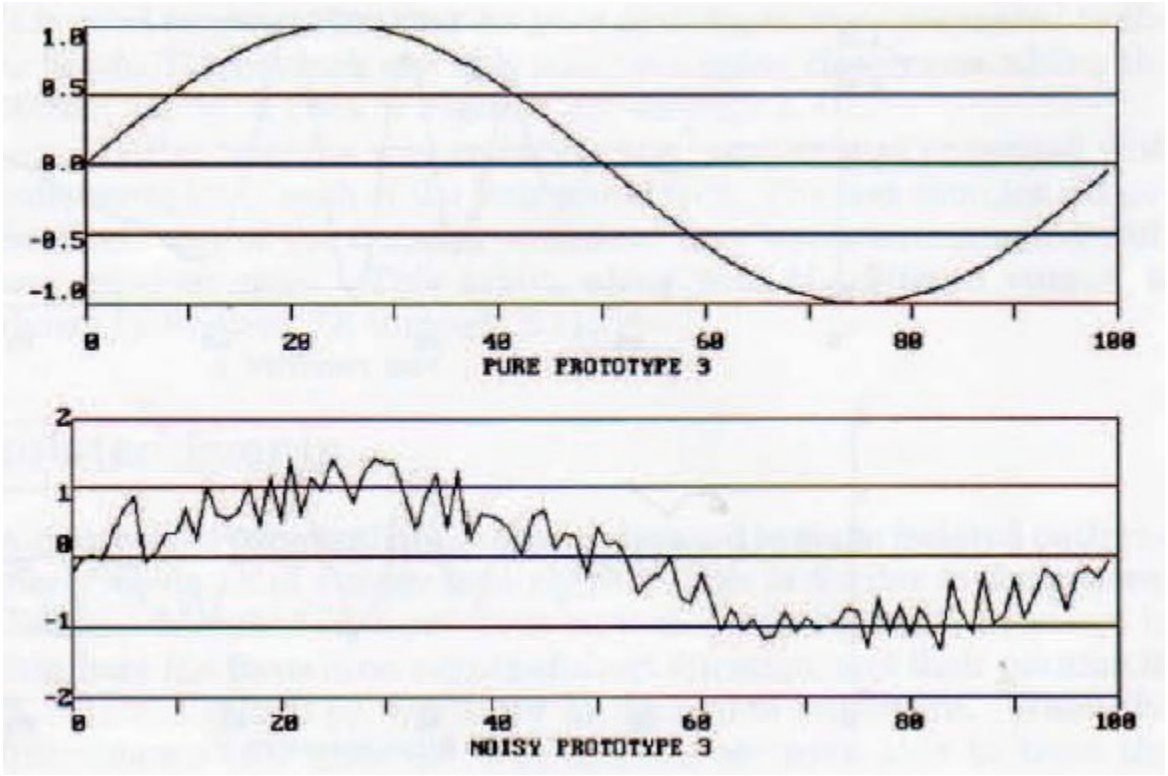
In order to filter the noisy signal. it was presented to the trained network in sets of 25 consecutive points. An output array 200 points was initialized to 0. For each of the 200 - 25 + 1 = 176 presentations, the network outputs were cumulated in the corresponding position in the output array. So first, points 0 - 24 of the input signal were applied to the network inputs, with the network outputs cumulating into points 0 - 24 of the output array. Then points 1 - 25, then 2 - 26. et cetera, were applied, until finally points 175 - 199 were done. **When all presentations were finished, each element in the output array was divided by the number of trials that went into the sum to get a mean**. This would be 25 for all interior points*, dwindling to just one point at the ends*.

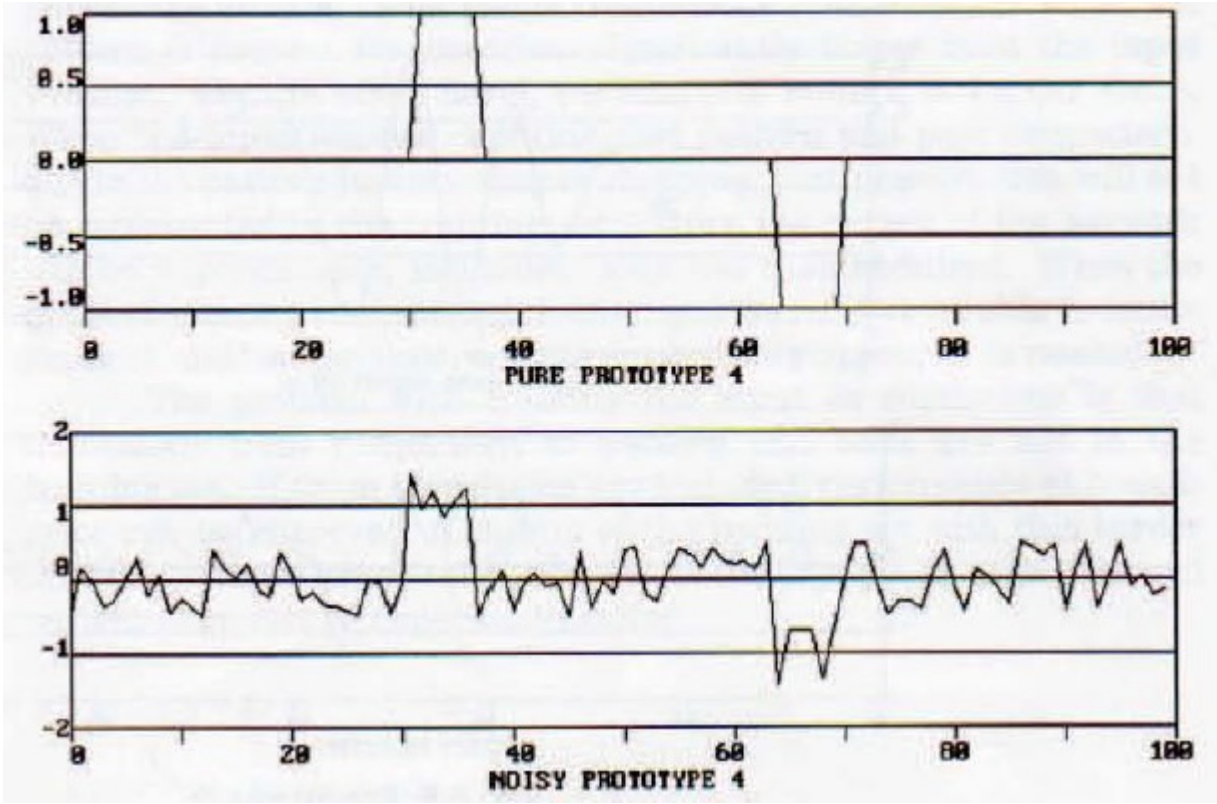
Learning a Prototype from Exemplars



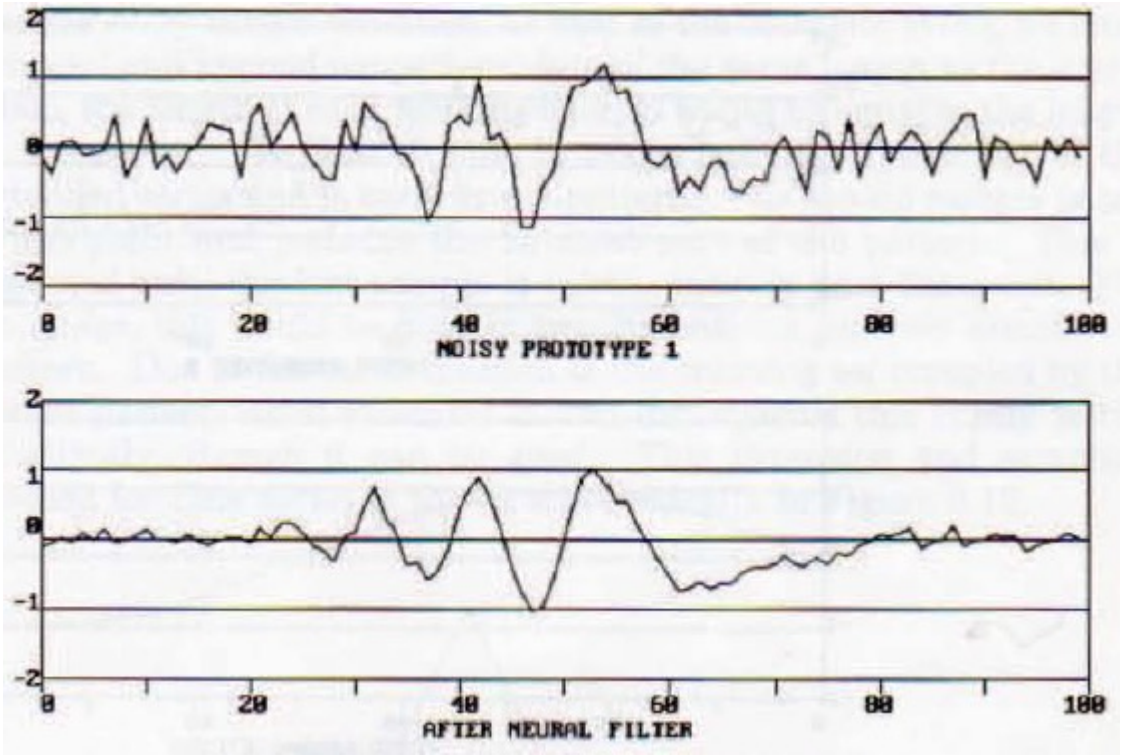


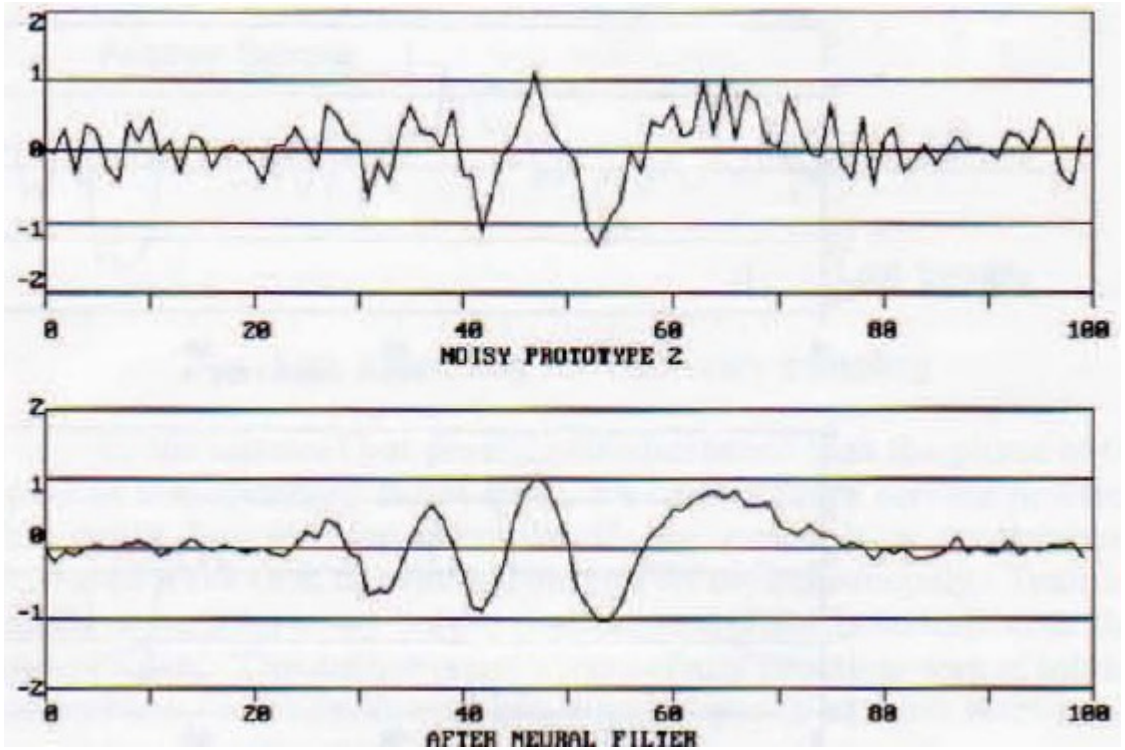
Learning a Prototype from Exemplars



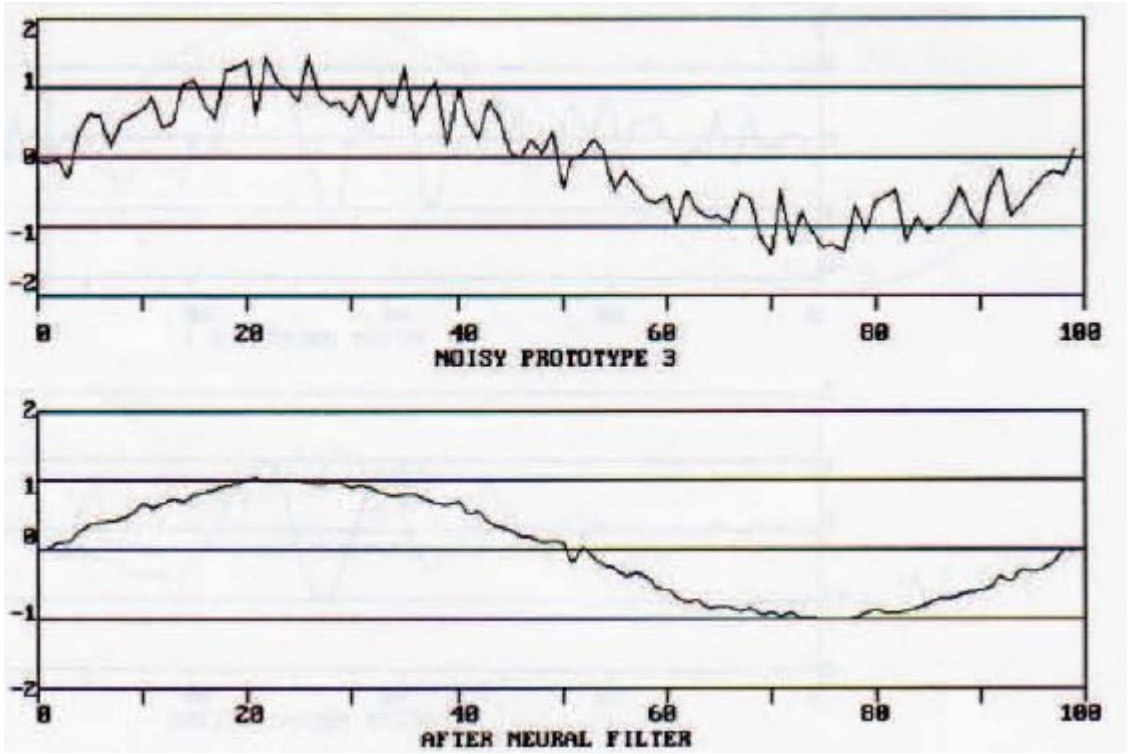


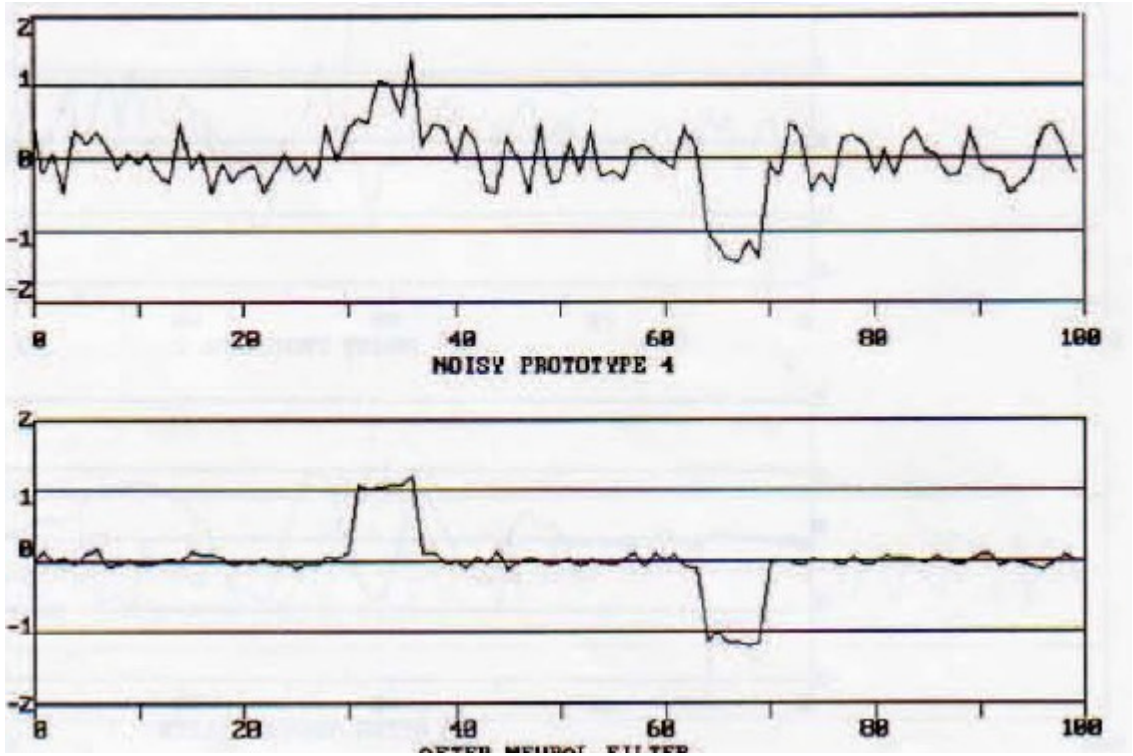
Neural Filter





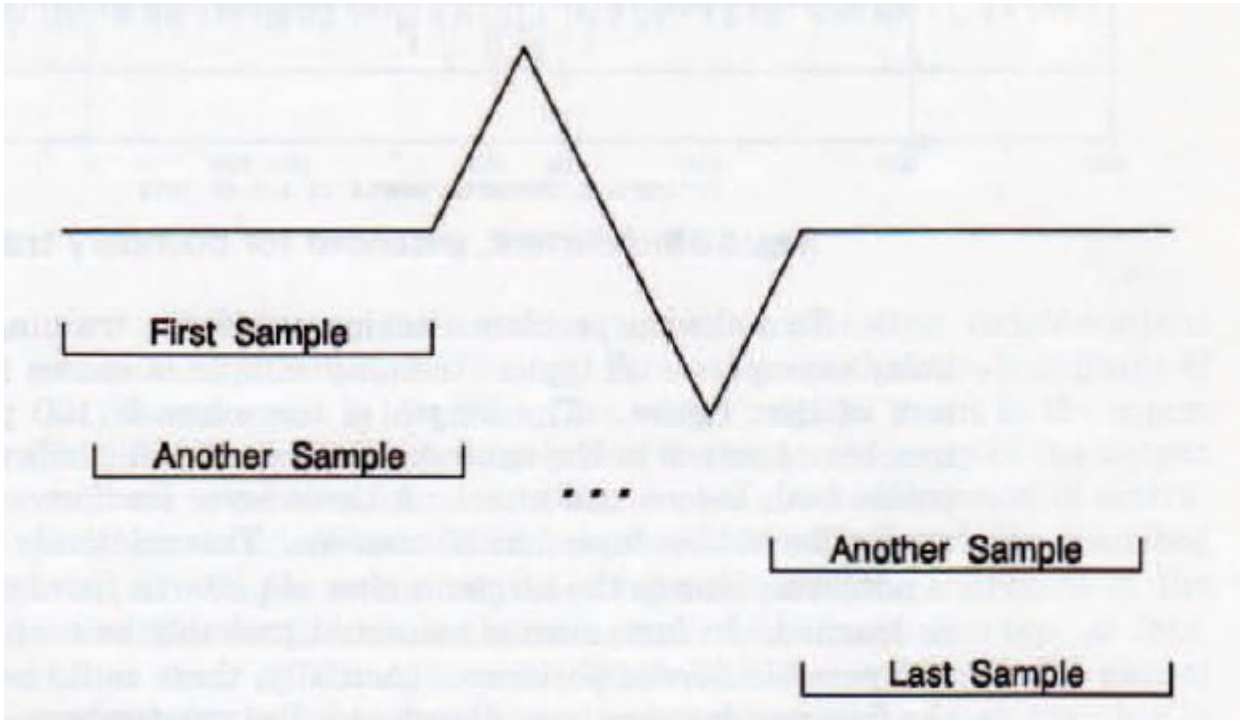
Neural Filter





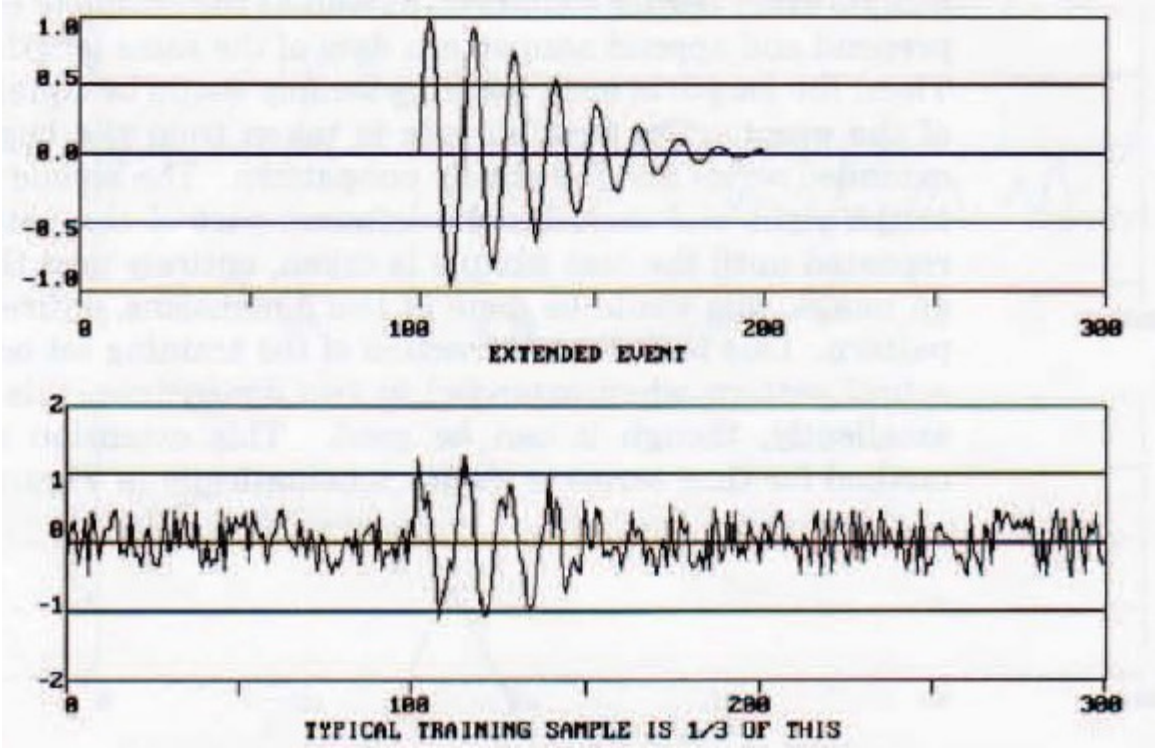
Exposing Isolated Events

Suppose that the event is well defined, both in shape and in duration. In order to sample every border condition, as well as the complete event, we must **prepend and append non-pattern data of the same length as the event.** Then, the length of each training sample would be equal to the length of the event. The first sample is taken from the beginning of the extended series and is entirely non-pattern. The second sample is one to the right and includes the leftmost part of the pattern. This is repeated until the last sample is taken, entirely past the event.



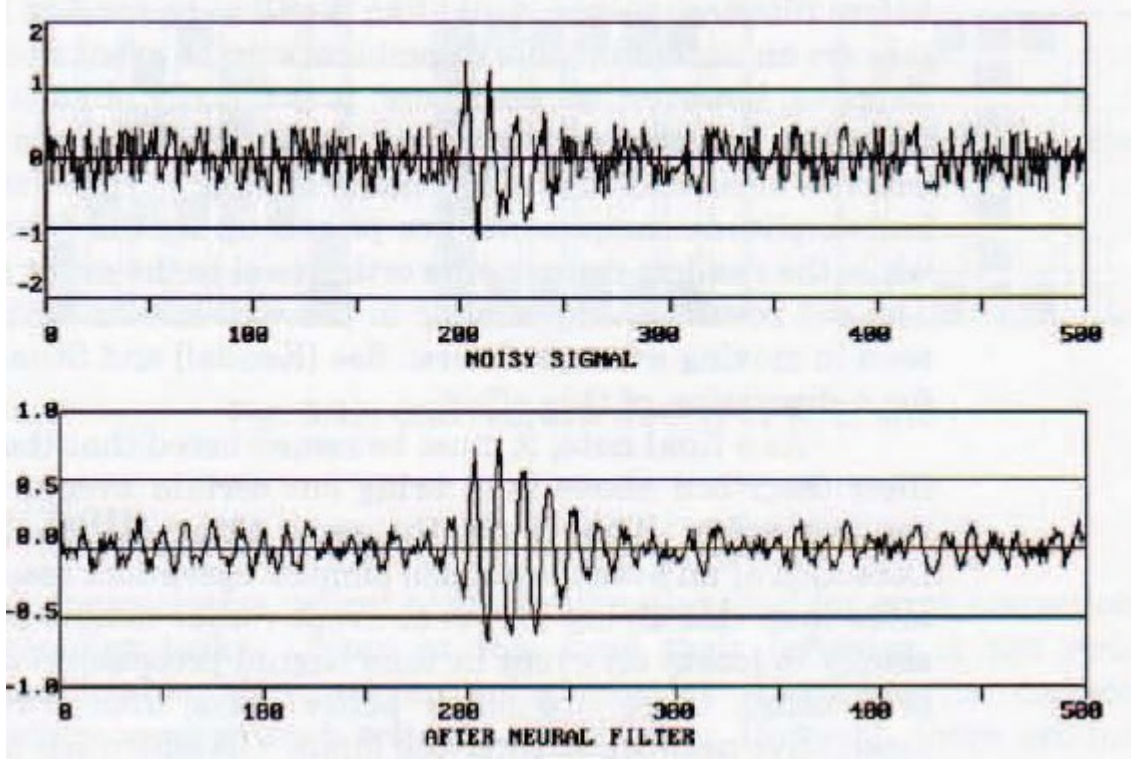
Extended for boundary training

A typical training sample is shown. The length of the event is 100 points, so the number of points in the extended pattern is 300 (the event, plus 100 points both before and after). A three-layer feedforward network is used. *The hidden layer has 30 neurons.* This relatively large number is necessary due to the large number of patterns (border positions) to be learned. In fact, even more would probably be useful. **There are 200 possible border positions.**



Filtered Event

Observe that the amplitude of the filtered event is less than that of the original event in the input. This is a common effect, largely due to the *nonlinearity of the output neuron's activation function*. When learning in feedforward networks, it will be seen that the compression at the extremes of the activation function *causes learning to be more focused on typical data, rather than on the extremes*.

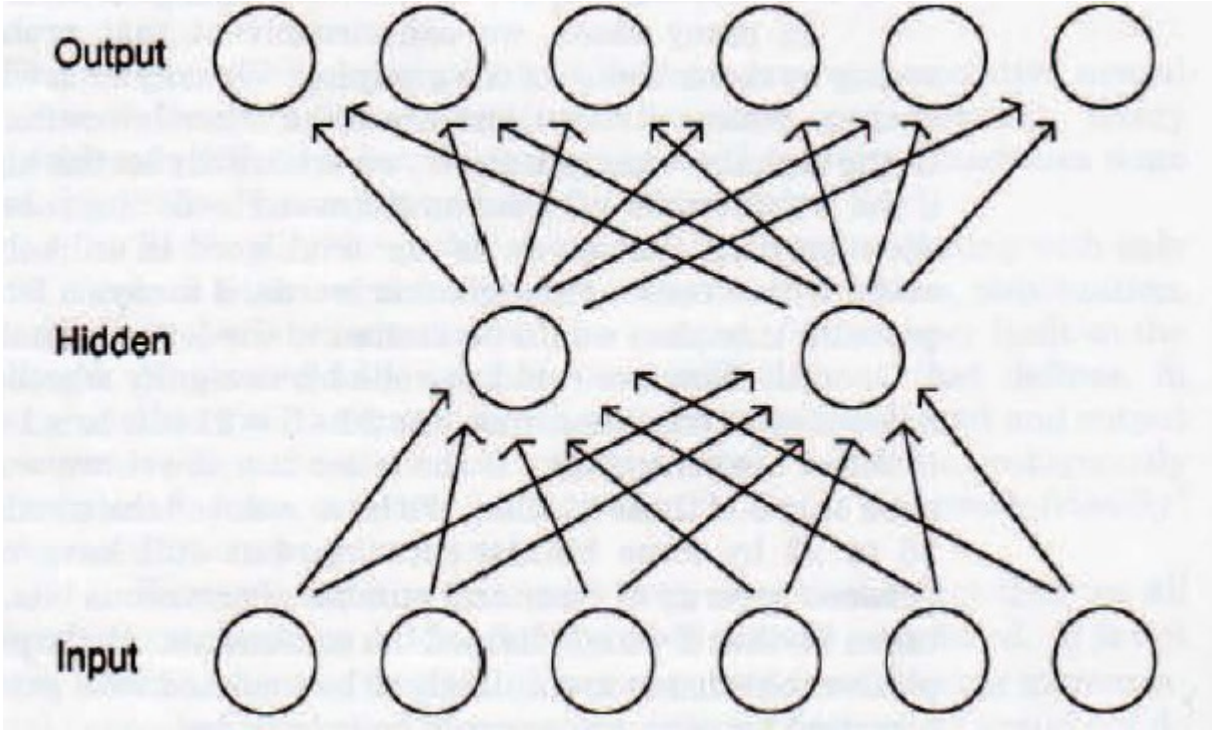


Pattern Completion

Autoassociative neural networks are **powerful pattern completers**. If a network is trained on examples of complete patterns (or, like the previous section, many examples of incomplete patterns). It will tend to reproduce the complete patterns when presented with partial patterns.

Data Compression

Three-layer feedforward networks trained for autoassociative recall are **sometimes used for data compression**. In general, there are far fewer neurons in the hidden layer than in the input and output layers. A three- layer feedforward network having six input and output neurons and two hidden-layer neurons is shown.



Data Compression

The principle of compression is as follows. The network is trained in autoassociative mode using blocks of typical samples of the data. **If we are compressing signals, we would use short, contiguous sections of many samples of the signal**. **If we are compressing images, we would use small, rectangular blocks of sample images.** The more our training samples resemble the actual data that will be compressed later, the better the quality of compression will be. After training is complete, we compress data by applying it to the input of the network. **For each data sample applied, compute the activations of the hidden neurons.** This comprises the compressed data. **To uncompress the data, use the hidden-neuron activations to compute the outputs**.

Data Compression

Obviously, there will be a tradeoff between the degree of compression and the quality of reproduction. **More hidden neurons will enable more accurate reproduction but will result in less compression**. Also, be aware that the quality of reproduction is directly related to the *completeness of the training*. As long as all possible data patterns are at least approximately represented, and training is sufficiently completed, *results will be excellent*. On the other hand, if the network is later called upon to compress data *that does not resemble anything in its training set*, it may not perform well.