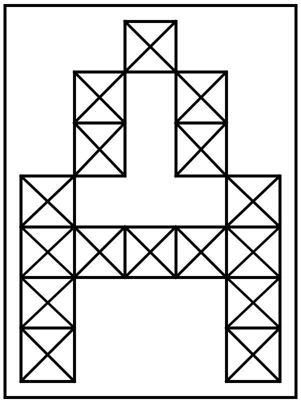
Appcr1: Character Recognition

It is often useful to have a machine perform pattern recognition. In particular, machines that can read symbols are very cost effective. A machine that reads banking checks can process many more checks than a human being in the same time. This kind of application saves time and money, and eliminates the requirement that a human perform such a repetitive task. The script `appcr1` demonstrates how character recognition can be done with a backpropagation network.

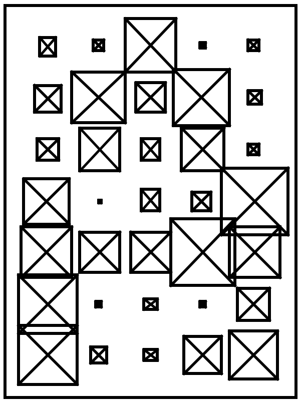
Problem Statement

A network is to be designed and trained to recognize the 26 letters of the alphabet. An imaging system that digitizes each letter centered in the system's field of vision is available. The result is that each letter is represented as a 5 by 7 grid of boolean values.

For example, here is the letter A.



However, the imaging system is not perfect and the letters may suffer from noise.



Perfect classification of ideal input vectors is required, and reasonably accurate classification of noisy vectors.

The twenty-six 35-element input vectors are defined in the function `prprob` as a matrix of input vectors called `alphabet`. The target vectors are also defined in this file with a variable called `targets`. Each target vector is a 26-element vector with a 1 in the position of the letter it represents, and 0's everywhere else. For example, the letter A is to be represented by a 1 in the first element (as A is the first letter of the alphabet), and 0's in elements two through twenty-six.

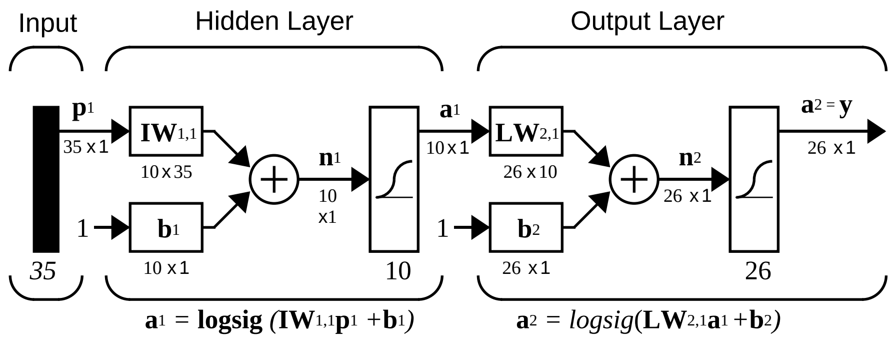
Neural Network

The network receives the 35 Boolean values as a 35-element input vector. It is then required to identify the letter by responding with a 26-element output vector. The 26 elements of the output vector each represent a letter. To operate correctly, the network should respond with a 1 in the position of the letter being presented to the network. All other values in the output vector should be 0.

In addition, the network should be able to handle noise. In practice, the network does not receive a perfect Boolean vector as input. Specifically, the network should make as few mistakes as possible when classifying vectors with noise of mean 0 and standard deviation of 0.2 or less.

Architecture

The neural network needs 35 inputs and 26 neurons in its output layer to identify the letters. The network is a two-layer `log-sigmoid/log-sigmoid` network. The log-sigmoid transfer function was picked because its output range (0 to 1) is perfect for learning to output boolean values.



The hidden (first) layer has 10 neurons. This number was picked by guesswork and experience. If the network has trouble learning, then neurons can be added to this layer.

The network is trained to output a 1 in the correct position of the output vector and to fill the rest of the output vector with 0's. However, noisy input vectors may result in the network not creating perfect 1's and 0's. After the network is trained the output is passed through the competitive transfer function `compet`. This makes sure that the output corresponding to the letter most like the noisy input vector takes on a value of 1, and all others have a value of 0. The result of this post-processing is the output that is actually used.

Initialization

The two-layer network is created with `newff`.

```
S1 = 10;
[R,Q] = size(alphabet);
[S2,Q] = size(targets);
P = alphabet;
net = newff(minmax(P),[S1 S2],{'logsig' 'logsig'},'traingdx');
```

Training

To create a network that can handle noisy input vectors it is best to train the network on both ideal and noisy vectors. To do this, the network is first trained on ideal vectors until it has a low sum-squared error.

Then, the network is trained on 10 sets of ideal and noisy vectors. The network is trained on two copies of the noise-free alphabet at the same time as it is trained on noisy vectors. The two copies of the noise-free alphabet are used to maintain the network's ability to classify ideal input vectors.

Unfortunately, after the training described above the network may have learned to classify some difficult noisy vectors at the expense of properly classifying a noise-free vector. Therefore, the network is again trained on just ideal vectors. This ensures that the network responds perfectly when presented with an ideal letter.

All training is done using backpropagation with both adaptive learning rate and momentum with the function `trainbpx`.

Training Without Noise

The network is initially trained without noise for a maximum of 5000 epochs or until the network sum-squared error falls beneath 0.1.

```
P = alphabet;
T = targets;
net.performFcn = 'sse';
net.trainParam.goal = 0.1;
net.trainParam.show = 20;
net.trainParam.epochs = 5000;
net.trainParam.mc = 0.95;
[net,tr] = train(net,P,T);
```

Training with Noise

To obtain a network not sensitive to noise, we trained with two ideal copies and two noisy copies of the vectors in `alphabet`. The target vectors consist of four copies of the vectors in `target`. The noisy vectors have noise of mean 0.1 and 0.2 added to them. This forces the neuron to learn how to properly identify noisy letters, while requiring that it can still respond well to ideal vectors.

To train with noise, the maximum number of epochs is reduced to 300 and the error goal is increased to 0.6, reflecting that higher error is expected because more vectors (including some with noise), are being presented.

```
netn = net;
netn.trainParam.goal = 0.6;
netn.trainParam.epochs = 300;
```

```
T = [targets targets targets targets];
for pass = 1:10
P = [alphabet, alphabet, ...
      (alphabet + randn(R,Q)*0.1), ...
      (alphabet + randn(R,Q)*0.2)];
[netn,tr] = train(netn,P,T);
end
```

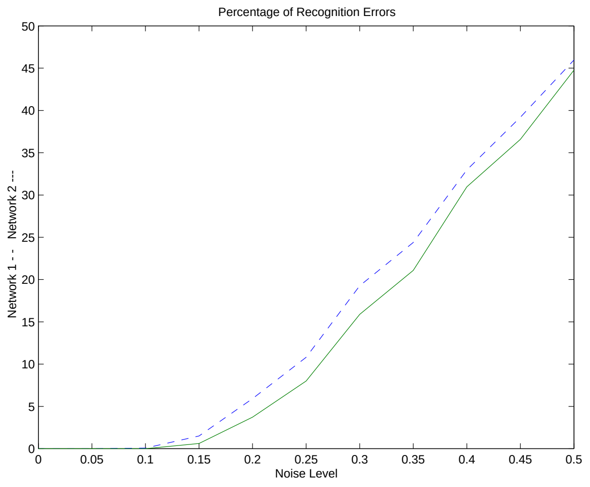
Training Without Noise Again

Once the network is trained with noise, it makes sense to train it without noise once more to ensure that ideal input vectors are always classified correctly. Therefore, the network is again trained with code identical to the "Training Without Noise" section.

System Performance

The reliability of the neural network pattern recognition system is measured by testing the network with hundreds of input vectors with varying quantities of noise. The script file `appcr1` tests the network at various noise levels, and then graphs the percentage of network errors versus noise. Noise with a mean of 0 and a standard deviation from 0 to 0.5 is added to input vectors. At each noise level, 100 presentations of different noisy versions of each letter are made and the network's output is calculated. The output is then passed through the competitive transfer function so that only one of the 26 outputs (representing the letters of the alphabet), has a value of 1.

The number of erroneous classifications is then added and percentages are obtained.



The solid line on the graph shows the reliability for the network trained with and without noise. The reliability of the same network when it had only been trained without noise is shown with a dashed line. Thus, training the network on noisy input vectors greatly reduces its errors when it has to classify noisy vectors.

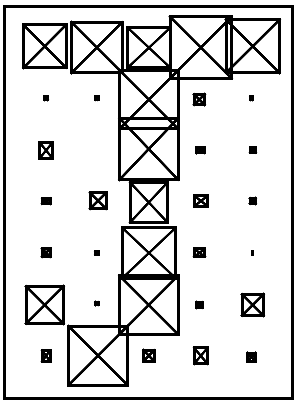
The network did not make any errors for vectors with noise of mean 0.00 or 0.05. When noise of mean 0.2 was added to the vectors both networks began making errors.

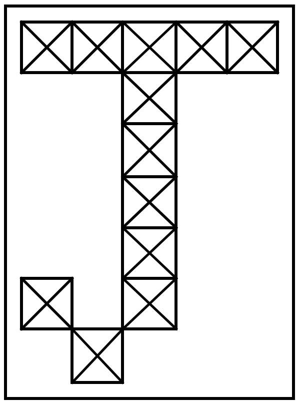
If a higher accuracy is needed, the network can be trained for a longer time or retrained with more neurons in its hidden layer. Also, the resolution of the input vectors can be increased to a 10-by-14 grid. Finally, the network could be trained on input vectors with greater amounts of noise if greater reliability were needed for higher levels of noise.

To test the system, a letter with noise can be created and presented to the network.

```
noisyJ = alphabet(:,10)+randn(35,1) * 0.2;
plotchar(noisyJ);
A2 = sim(net,noisyJ);
A2 = compet(A2);
answer = find(compet(A2) == 1);
plotchar(alphabet(:,answer));
```

Here is the noisy letter and the letter the network picked (correctly).





Summary

This problem demonstrates how a simple pattern recognition system can be designed. Note that the training process did not consist of a single call to a training function. Instead, the network was trained several times on various input vectors.

In this case, training a network on different sets of noisy vectors forced the network to learn how to deal with noise, a common problem in the real world.