Hamming Neural Network - classifier

Subject: **Applications of Artificial Intelligence**
Project: **Hamming Neural Network - classifier**
Authors: **Stanisław Kamiński**, **Marcin Samborski**
Translation: **Łukasz Adamowicz, Piotr Szopa**
Supervisor: **MSc. Adam Gołda**

Hamming's classifier – description

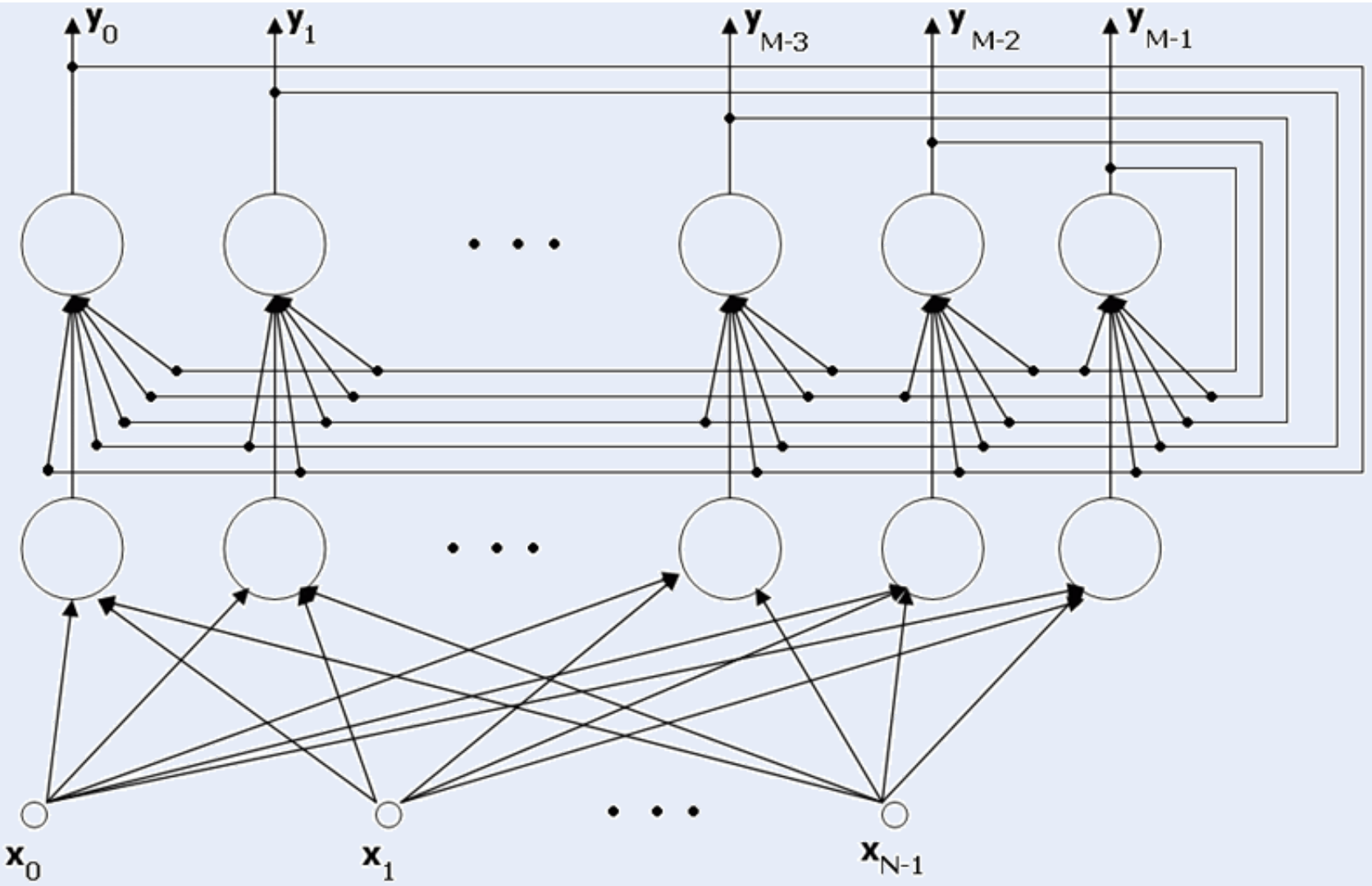
This project describes the properties, applications, and creation process of the Hamming Neural Network, working as the signals classifier.

You may notice that the model of network described in this project may be a little different than one described in a professional literature. That's because we created our network to implement it in the Matlab environment. Nevertheless, the network works correctly.

What kind of signals does the Hamming Network process?

Although the network works in an analog way, it processes binary signals – on the other hand, those signals can be "noisy", and have continuous values, not only zeros and ones.

What does the Hamming Network look like?



In the picture presented above we can see the Hamming Network. It can be divided into two basic sections:

- **input layer** – a layer built with neurons, all of those neurons are connected to all of the network inputs;
- **output layer** – which is called MaxNet layer; the output of each neuron of this layer is connected to input of each neuron of this layer, besides every neuron of this layer is connected to exactly one neuron of the input layer (as in the picture above).

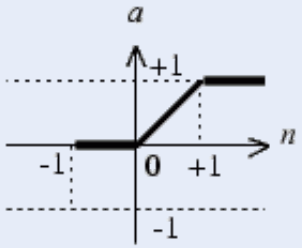
It’s easy to see, that both layers have the same number of neurons.

How does the Hamming Network work?

Input layer neurons are programmed to identify a fixed number of patterns; the number of neurons in this layer matches the number of those patterns (M neurons – M patterns). Outputs of these neurons realise the function, which "measures" the similarity of an input signal to a given pattern.

The output layer is responsible for choosing the pattern, which is the most similar to testing signal. In this layer, the neuron with the strongest response stops the other neurons responses (it usually happens after few calculating cycles, and there must be a "0" on x(i) inputs during these cycles).

There should be the "1 of M" code used on the output, which points at the network answer (1 of M patterns is recognized). To achieve this, proper transfer function has to be used – in our case the best function was



$a = satlin(n)$

How are the connections weights created?

Weights of the input layer neurons

Those weights are set to assure that the maximal responses of all neurons are equal, and that exactly one pattern causes specific neuron to response. There is no need to teach network to achieve that, it's sufficient to mathematically set weights.

$$w_{ij} = x_i^j \ ,$$

$$0 \le i \le N-1, \quad 0 \le j \le M-1$$

where:

- **w(i,j)** is the weight of the connection between neuron "j" and the input "i"
- **x(i,j)** is the value of signal "i" in pattern "j"

This equation becomes obvious, when we recall that the product of weights and impulses of a neuron can be interpreted as the cosinus of the angle between the weight vector and the impulse vector. When these vectors are equal, the neuron output will be "1", when vectors are different, the output value range will be from –1 to 1.

Input weights of the output layer neurons

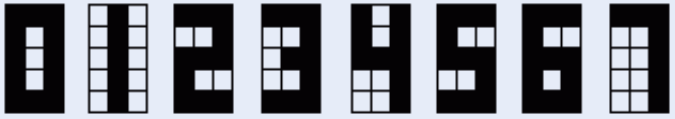
At first, we need to determine the number of the output layer neurons inputs.

Each of those neurons is connected:

- to itself – weight=1
- to all of neurons of the output layer – weight of each connection is -1/M
- to appropriate neuron of the input layer – weight=1

An example

To familiarize ourselves with the Hamming classifier, we made a network in the Matlab environment. Its task is to recognize 1 of 8 signs recorded in the network "memory". The picture below shows 8 used signs:



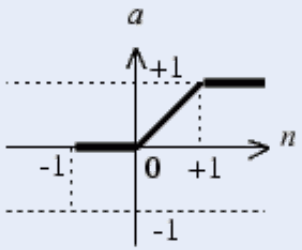
The matrices that represent these signs are two-dimensional (3x5 pixels), but it was more convenient to write them as a single row (15 positions) vectors. Black squares are ones, white squares are zeros. To make proper calculations, we normalised all of the row vectors.

We want to classify 8 signs, so our network will contain 16 neurons, divided into 2 groups, 8 neurons in each layer.

Weights of the input layer are predefined as transposed matrix of signs – previously normalised.

Weights of the output layer are defined according to the previously stated assumptions (see *'Input weights of the output layer neurons'* section), and the '–1/M' parameter in our case was set to '–1/8'. We cannot forget that it is necessary to normalise rows of that matrix.

Neuron activation function that we have chosen is:



$a = satlin(n)$

We assumed that 15 calculating cycles in the output layer will be sufficient to stabilize the network response; but we noticed that it can be not enough, when we are dealing with strong noise. It would be the best solution to check if there is a "1" on exactly one of the outputs each cycle, but it would be difficult to predict the amount of time needed to perform the classification then – and that is very important when working with DSP.

The last part of the experiment was to add some noise to our signs and then examine the network. The "analog" noise is generated with *rand()* function.

Summary

The network we created works correctly – it recognises given sign, even with noise, if only it is possible. It does not recognise properly when it is totally impossible to do it or when 2 neurons have strong responses, similar to each other – it would take more calculating cycles to recognise it, then.

Click here to download complete m-file.

Click here to download a whole project.

References

- prof. Ryszard Tadusiewicz, "Sieci neuronowe", Akademicka Oficyna Wydawnicza RM 1993
- prof. Andrzej Kos, "Przemysłowe zastosowania sztucznej inteligencji", Lecture

To the top