**Identifying numbers with Hopfield neural networks**

Balazs Magyar

C56QME

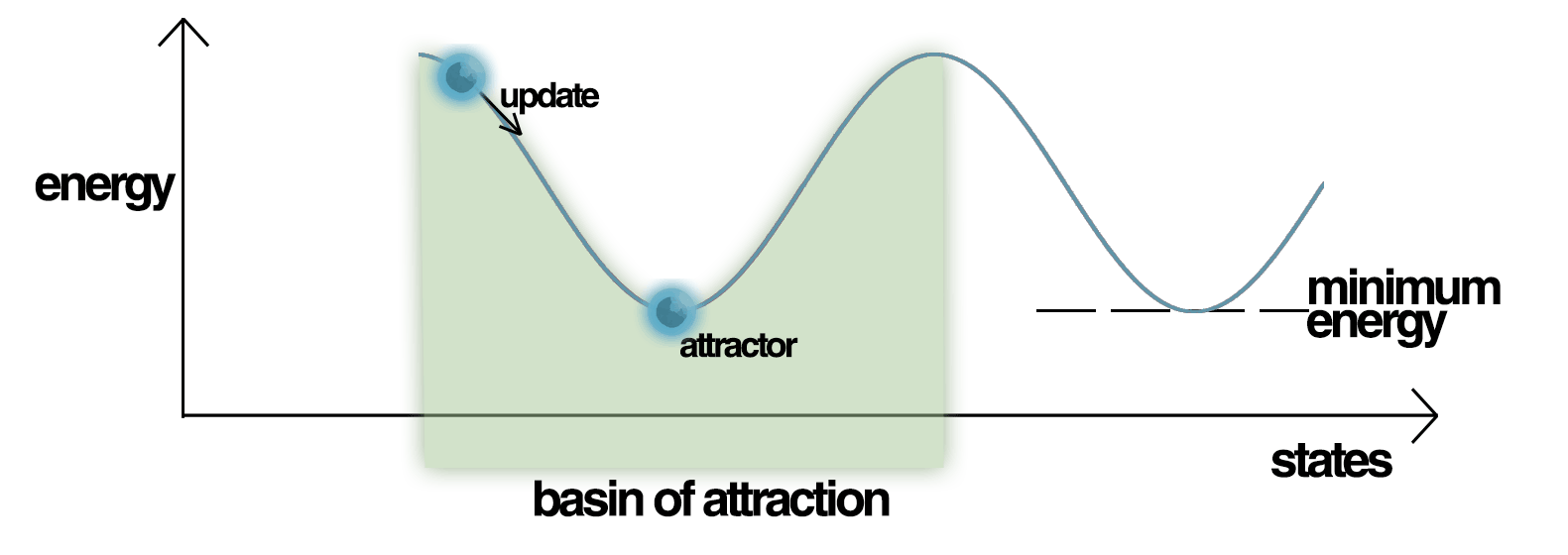
Abstract

My task was to implement a Hopfield neural network, which is capable of identifying numbers on 8 x 8 pixel black and white images. This solution is faster than finding the closest vector by comparing Hamming-distances. The implementation is a proof-of-concept windows application, with timers to measure the performance of the neural approach. It proved, that this method is faster, and this difference can be measured even at this scale: 10 pieces of 64 bit vectors.

**1. Introduction**

Identifying hand-written numbers basically means finding the closest stored pattern to the input pattern, regarding its Hamming distance. The traditional method would be comparing all the patterns with the input, one-by-one, pixel-by-pixel. The theoretical algorithmic complexity of this approach is O(n). With the use of a Hopfield neural network, we can reduce this complexity to O(log n), assuming that the W matrix is built in advance.

The energy of the neural network decreases (or stays the same) on every iteration, to reach a local minimum, which corresponds to a stored pattern. This makes it possible for the identification to run much faster.



1. Figure: The energy of a neural network

[1]

**2. The model**

**2.1. Teaching the network**

“Teaching” a neural network means generating a W matrix. [2] This is done by the following formula:

Where

*N* is the dimension of the vectors, in this case 64.

*M* is the number of patterns to be stored.

The formula is altered, because zero-based indexing is a lot easier to implement.

Example:

If there are two input vectors, , then

The W matrix size will be N\*N, in this case 4096.

**2.2. Identifying**

When the matrix is generated, we only need 256 iterations of the following formula, to get the network to a steady state.

Where

*y(0)* is the input vector

*k* is the current state of the network

.

This way, the network will reach a local minimum of energy, so we are done…

**2.3. Spurious Steady States**

… if that state was one of our training patterns. Because “sometimes the network will converge to spurious patterns (different from the training patterns). The energy in these spurious patterns is also a local minimum. For each stored pattern x, the negation -x is also a spurious pattern.” [1]

To solve this problem, we have to make the algorithm a bit more complicated.

“The network can be made more stable (less spurious states) if we set for all i.” [3]

“There are stable mixture states, linear combinations of an odd number

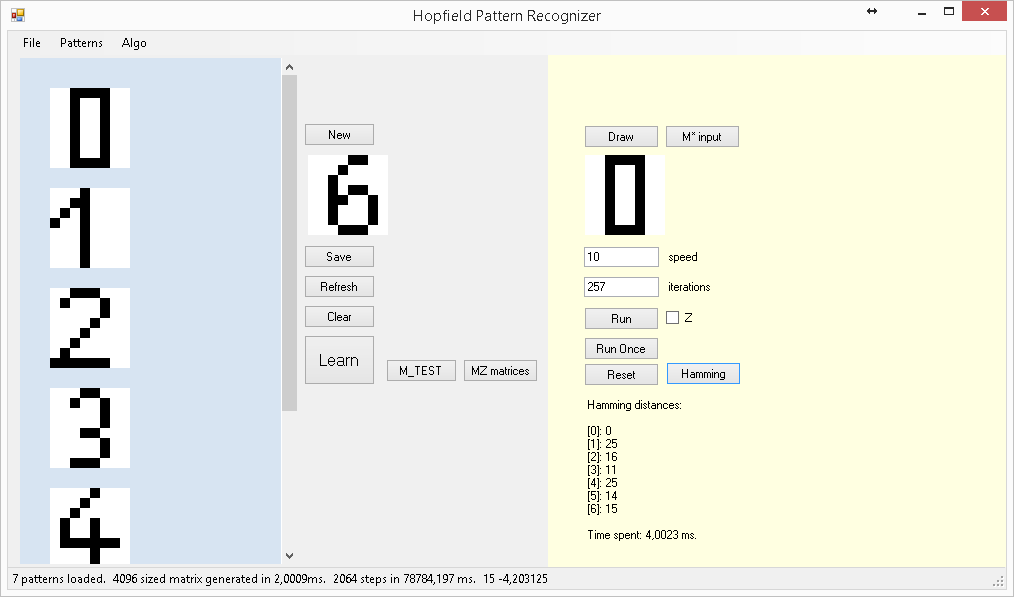
of patterns.” [3]

There is a method, which can be used to avoid Spurious Steady States, but the implementation is not yet fully working. A neural network is way harder to debug, than a traditional approach. The steps would be the following:

1. Generate a fully random vector to every S. The Z vector paired with
2. Generate an M matrix, using the following formula:
3. Generate a W matrix, using the following formula:
4. When recalling from the network, first multiply the input vector by the M matrix. Let the output of this be the U vector.
5. Now run the Hopfield network using the U as input and W as the W matrix.
6. The output should be a Z vector.
7. Find the pair of it.
8. That is the solution S vector.

**3. Solution**

To implement a solution, I translated these mathematical formulas to the C# language. An easy-to-use user interface was created using Windows Forms. This way, the user can teach patterns to the network, and later use it to recognize them.



2. Figure: The User Interface

The source code is available at https://github.com/magyarb/adahf

# **4. List of references**

|  |  |
| --- | --- |
| [1] | Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Hopfield\_network. |
| [2] | R. Wang. [Online]. Available: http://fourier.eng.hmc.edu/e161/lectures/NeuralNetworks/node5.html. |
| [3] | L. A. Belanche, "Hopfield networks," [Online]. Available: http://www.cs.upc.edu/~belanche/Docencia/apren/2009-10/Teoria/tema9.pdf. |