Procedia Computer Science

Volume 71, 2015, Pages 68–75

2015 Annual International Conference on Biologically Inspired Cognitive Architectures

Comparison of Different Learning Algorithms for Pattern Recognition with Hopfield's Neural Network

Tomasz Szandała
Wroclaw University of Technology
tomasz.szandala@gmail.com

Abstract

Hopfield neural networks can be used for compression, approximation, steering. But they are most commonly used for pattern recognition thanks to their associative memory trait. In order to fulfill this task, the network has to be trained with one of algorithms. In this paper I will try to implement three of the most popular ones and compare their effectiveness by trying to recognize various patterns consisting of binary input arrays. The tests will use Hebbian learning, Oja's Hebbian modification and pseudo-inverse, which proves to be most promising training algorithm.

Keywords: hopfield, machine learning, pattern recognition, hebbian, oja, pseudoinverse

1 Introduction

In the recent years so called "artificial intelligence" is more and more common in our daily life. Unfortunately nowadays it still suffers because of not sufficient interaction with human's aspects like: voice, picture or hand writing recognition. In 1982 John Hopfield proposed a neural network, that can be used for pattern recognition, which was using Hebb's rule to learn patterns and search for similarities in received inputs. Since that time, there were invented many new learning algorithms, some of them could better fulfill the role of pattern recognition training solution, but, like for example pseudo-inverse, require much, much more computing power in order to be used.

Today, when it is possible to implement and run those learning methods, it should be mandatory to test and pick up best solution for this purpose. In this paper I will implement small Hopfield network consisting of 56 neurons (which thanks to Python can be easily increased), which will be trained with 7 patterns by different algorithms and I will decide which of the solution is the most promising one.

Selection and peer-review under responsibility of the Scientific Programme Committee of BICA 2015

doi:10.1016/j.procs.2015.12.205

2 Hopfield network

A Hopfield network is a form of recurrent artificial neural network invented by John Hopfield in 1982[1]. Hopfield nets serve as content-addressable memory systems with binary threshold nodes. They are guaranteed to converge to a local minimum, but convergence to a false pattern (wrong local minimum) rather than the stored pattern (expected local minimum) can occur[7].

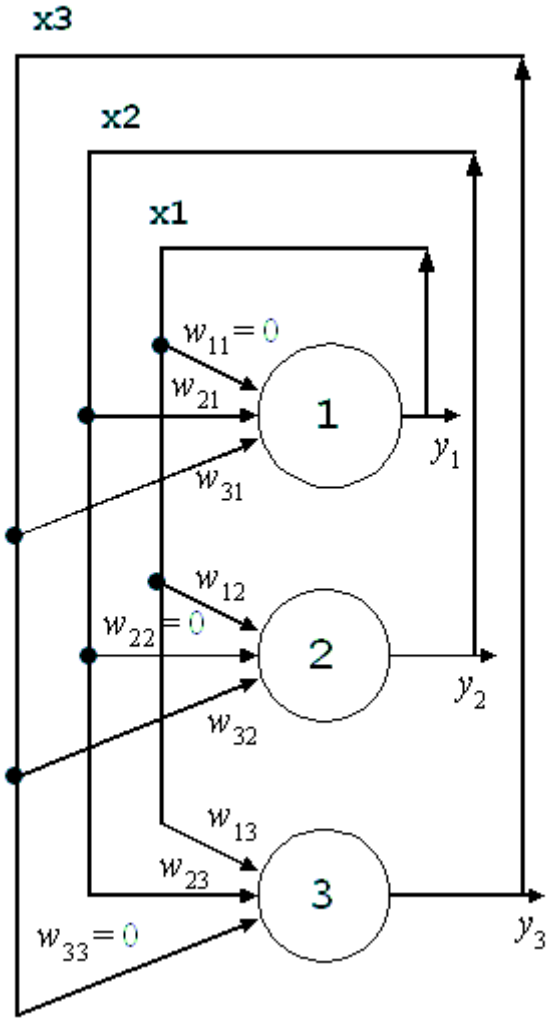


Fig 1. Sample diagram Hopfield neural network

Each Hopfield's network consists of *n* neurons, in my realization: each neuron is responsible for one pixel in the pattern. Output of each neuron is the value of activation function from sum of factors weight and previous value for each neuron. This can be written with equation below(1).

$$y_i(t+1) = f\left(\sum_{j=1}^{n} w_{ij} * y_j(t)\right) \quad (1)$$

Fig 2. Counting output for i'th neuron

In above equitation: y's are the outputs of j'th neuron and *w* is the weight of this connection to i'th neuron. The *f* function is so called activation function, which in standard Hopfield's network is(2):

$$f(x) = \begin{cases} 1 & for\ x \geq 0 \\ -1 & for\ x < 0 \end{cases} \quad (2)$$

Fig 3. Hopfield's activation function

Like all neural networks: working with Hopfield's one consist of two parts: training where using the learning algorithms we set the values of weights for each neuron and using, when we give a pattern to network, and wait for the output.

The pattern reproduction is multistep. First we set testing image as output from the network, subsequently we use above equitation to calculate the output. If the output differs from the previous one, we repeat this process. Reproduction ends when output does not differ from previous one or there were 30 cycles done.

Moreover the Hopfield network is limited by its memory capacity, which is equal (3) to only 13,8%[6] of its total neurons number.

$$cap. = \frac{n}{2 * \ln n} \quad (3)$$

Fig 4. Equation for computing standard Hopfield's network capacity

We have the above equation, taken from reference [11], where n is the number of neurons and cap. is the maximum recoverable number of patterns. This leads to the conclusion: if we have 56 neurons, we can hold only 7 patterns in it. All number above this one will suffer with lower recognition probability. Of course there are methods to increase the capacity [6], but this is not matter of this paper.

Furthermore the memory capacity is vital for correct pattern recognition. In similar paper[9] author did not take into consideration this factor, which resulted in highly corrupted results.

.

3 Learning algorithms

For this paper have been chosen three training algorithms:

- Hebb's learning rule,
- Oja's modification of Hebb's rule,
- pseudo-inverse pattern transformation rule.

3.1 Hebb's learning rule

Hebbian learning is one of the oldest learning algorithms, and is based in large part on the dynamics of biological systems. It was introduced by Donald Hebb in his 1949 book The *Organization of Behavior*. A synapse between two neurons is strengthened when the neurons on either side of the synapse (input and output) have highly correlated outputs. In essence, when an input neuron fires, if it frequently leads to the firing of the output neuron, the synapse is strengthened. Following the analogy to an artificial system, the tap weight is increased with high correlation between two sequential neurons[4].

This learning sets weights as follows:

$$w_{ij} = \frac{1}{n}\sum_{k=1}^{m} x_i^k * x_j^k \quad (4)$$

Fig 5. Hebbian's learning weights equation

Where *n* is the number of neurons and *m* is the number of training patterns.

This solution has problem while working with big neural networks. Some neurons can be exaggerate weights values, which can and in shading other synapses. Nevertheless this simple method worked quite well, until new, more advanced solutions were found.

3.2 Oja's learning method

Oja's learning rule, or simply Oja's rule, first proposed by Finnish computer scientist Erkki Oja in 1982 in his work *Simplified neuron model as a principal component analyzer [5]*, is a model of how neurons in the brain or in artificial neural networks change connection strength, or learn, over time. It is a modification of the standard Hebb's Rule that, through multiplicative normalization, solves all stability problems and generates an algorithm for principal components analysis. This is a computational form of an effect which is believed to happen in biological neurons.

The main idea for this method is to keep weights of each neuron normalized to 1.

$$|w_i| = 1 \quad (5)$$

Fig 6. Main idea behind Oja's learning rule

In order to achieve this we have to add one more element to weights computing, the *V* factor, which is defined:

$$V_{ij} = \sum_{k=1}^{m} w_{ij} * x_j^k \quad (6)$$

Fig 7. The V factor for Oja's method

Now we can set the weights vector:

$$w_{ij}^{k+1} = w_{ij}^k + u * V * (x_j^{k+1} - V * w_{ij}^k) \quad (7)$$

Fig 8. Weights computing equation

The *u* parameter is learning speed rate.

The only problem encountered here is the floating point variable limits. Sometimes, mostly in the last iterations really small numbers are found, which leads to *NaN* values and therefore: totally wrong recognition result.

3.3 Pseudo-inverse method

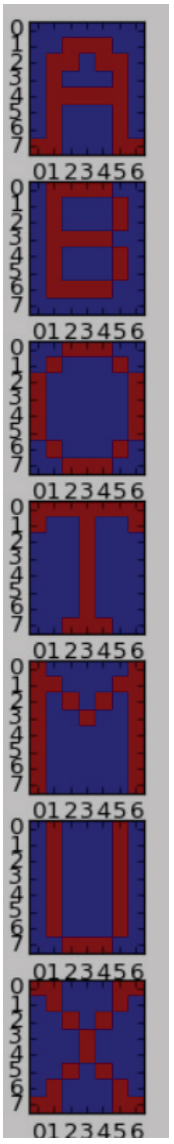
This odd sounding solution has quite easy bases. We assume that each pattern on input, gives exactly himself on output. If we have matrix of weights(W) and we multiply the received pattern(X) by it we are getting the pattern itself[7]. Simple is that!

In order to prove this there is need for some matrix computations:

$$W * X = X$$
$$W = X * X^{+} \quad (8)$$
$$W = X * (X^T * X^{-1}) * X^T$$

Fig 9. Pseudo-inverse equations

The trick here is between second and third line. + near *X* means pseudo-inverse of matrix, but we have linear independent matrix *X*, we can safely write as follows[8].



Those computations seems hard, but for modern computers they are matter of seconds to compute.

4 Experiment preparation

As it has been already stated: testing network will be learnt with seven patterns of randomly chosen pictures resembling letters: A, B, O, T, M, U, X. In program they are represented by bits of 1's and -1's, but for the purpose of readability they will be presented using *pyplot* diagrams.

For testing patterns set there were prepared 49 bloated patterns. Each subsequent column consists of more randomly reversed bits (pixels). The mutation stopped, when there was no clear letter visible from human's point of view.

All patterns are visible as pictures of 8x7 pixels, but in the program they are represented as vectors of 56 cells of +/- 1s values. Each pattern was correctly recognized, by network trained with each of the algorithms.

Afterwards training, the weights were "frozen" and each pattern was bloated with randomly modified (multiplied by -1) pixel or pixels, depending on wave. Each subsequent wave was more and more noised, as we can on figure 11. In the end there were seven waves of deformation, because next wave made patters unrecognizable by the human.

Fig 10. Original learning patterns of 8x7 letters

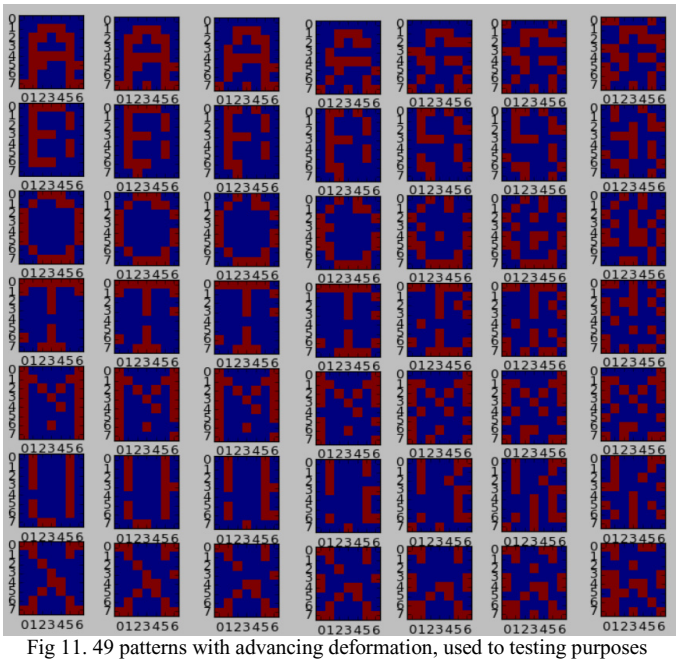


Fig 11. 49 patterns with advancing deformation, used to testing purposes

5 Tests results

Each pattern column has been used on the network taught by each of the three training methods. Results has rated in binary way: correct ☑ or incorrect ☒. Incorrect recognition was set, when the output was somehow bloated or wrong pattern (comparing to human recognition) was

chosen or no pattern was chosen. Results of the test have been inserted into table, for better readability.

Table 1. Results of conducted tests. Tick means correct recognition, cross means incorrect recognition or issue when network stuck in local-minima

	Pattern	Hebb's	Oja's	Ps-inv.
Column I	**A**	☑	☑	☑
	B	☒	☑	☑
	O	☑	☑	☑
	T	☑	☑	☑
	M	☑	☑	☑
	U	☑	☑	☑
	X	☑	☑	☑
Column II	**A**	☒	☑	☑
	B	☒	☒	☑
	O	☑	☑	☑
	T	☒	☑	☑
	M	☑	☑	☑
	U	☑	☑	☑
	X	☑	☑	☑
Column III	**A**	☑	☑	☑
	B	☒	☒	☑
	O	☑	☑	☑
	T	☒	☒	☑
	M	☑	☑	☑
	U	☑	☑	☑
	X	☒	☑	☑
Column IV	**A**	☑	☑	☑
	B	☒	☒	☑
	O	☑	☑	☑
	T	☒	☑	☑
	M	☑	☑	☑
	U	☑	☑	☑
	X	☑	☑	☑
Column V	**A**	☒	☑	☑
	B	☒	☒	☑
	O	☑	☑	☑
	T	☑	☑	☑
	M	☑	☑	☑

	U	☒	☒	☑
	X	☑	☑	☑
Column VI	**A**	☒	☒	☑
	B	☒	☒	☑
	O	☑	☑	☑
	T	☑	☑	☑
	M	☑	☑	☑
	U	☒	☒	☑
	X	☑	☑	☑
Column VII	**A**	☒	☒	☒
	B	☒	☒	☑
	O	☑	☑	☑
	T	☒	☒	☑
	M	☑	☒	☑
	U	☒	☒	☒
	X	☒	☑	☑

By analyzing the results we can find that Hebbian's method failed in over 40% patterns, which is not very satisfying result. Results are disappointing mostly, because used patterns are simple, consisting only from 56 bits, rarely met in nowadays real environment. Noticeably better reliability is shown using Oja's training method: these tests were failed in 14 cases, which leads to 28% wrong classification. Still not perfect, but significantly better. And the last method: the pseudo-inverse, seemed perfect, until last column, where it failed on 2 occasions. Nevertheless its 4% wrong recognitions classifies this method as the best of all three. It is also worth to mention, that all simple classifiers, with accuracy above 85% are usually satisfying[12] and tested-pseudo-inverse achieved 96% accuracy in the field of pattern recognition.

It is also worth to mention, that bloated pattern "B", in seventh wave, was recognized as "O", but since the man-made recognition implies it can resemble "O", it was classified as correct recognition.

Other issue of the learning algorithms is the time needed for chosen method. Pseudo-inverse takes longer to be carried out, but its accuracy is worth its time. Nevertheless the time aspect of those methods is not the subject of this paper, so it was neglected for now, perhaps it will be the matter of another experiment.

6 Conclusions

This paper is just a brief look on comparison of Hopfield's network learning algorithms. From the conducted tests we can see that the very first learning method - Hebbian's rule can be wrong even on slightly noised patterns. Oja's method proves to be a bit better, but according to reference [10] it shows bigger advance over Hebb's when we use it in bigger networks, consisting of hundreds of neurons. In this example it recognized few of the patterns which Hebbian could not. Of course we need to remember, that Oja's method is only a modification of the previous one.

The last method, the pseudo-inverse, acts really good. Even highly bloated patterns are being recognized without many mistakes. Nevertheless, even if for those few patterns it works pretty

well, it is still biased with memory capacity restrictions, so it may lack this accuracy for more advanced pattern recognitions.

To sum up: this essay is just a simple neural networks learning methods comparison which is only a tip of the iceberg in filling the gap between human and computer communication.

7 References

[1] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities", *Proceedings of the National Academy of Sciences of the USA*, vol. 79 no. 8 pp. 2554–2558, April 1982.

[2] NumPy Community, "NumPy Reference - Release 1.10.0", November 2014.

[3] John Hunter, Darren Dale, Eric Firing, Michael Droettboom, "Matplotlib Release 1.4.2", October 2014.

[4] NeuralNetworkSolutions, "Unsupervised Learning in Neural Networks - Hebbian Principle", http://www.neuralnetworksolutions.com/nn/ unsupervised2.php, access: January 2015

[5] Erkkin Oja, "Simplified neuron model as a principal component analyzer", *Journal of Mathematical Biology,* November 1982.

[6] Arnos Storkey, "Increasing the capacity of a Hopfield netowrk without sacrificing functionality", 1997.

[7] Bronisław Langner, "Systemy wspomagania decyzji Sieci wielowarstwowe Sieci rekurencyjne (Hopfielda, Hamminga)", 2014.

[8] James, M., "The generalised inverse", *Mathematical Gazette* 62, June 1978, 109–114.

[9] Radosław Matusik, "Metody uczenia sieci nauronowej Hopfielda", 2009.

[10] Delmas JP., Cardos JF., "Asymptotic distributions associated to Oja's learning equation for neural networks", 1998.

[11] Robert J. McEliee, Edward C. Posner, "The Capacity of the Hopfield Associative Memory", 1987.

[12] Andrew R. Webb, "Statistical Pattern Recognition", 2002.