High Performance Associative Neural Networks: Overview and Library

Oleksiy K. Dekhtyarenko[1⋆] and Dmitry O. Gorodnichy[2]

[1] Dept. of Computing Science, University of York, Heslington, York, UK, Y010 5DD
[2] Institute for Information Technology, National Research Council of Canada, M-50 Montreal Rd, Ottawa, Ontario, K1A 0R6, Canada
`http://synapse.vit.iit.nrc.ca`

Abstract. Associative neural networks are regaining the popularity due to their recent successful application to the problem of real-time memorization and recognition in video. This paper presents a comparative overview of several most popular models of these networks, such as those learnt by the Projective Learning rules and having Sparse architecture, and introduces an Open Source Associative Neural Network Library which allows one to implement these models.
Keywords: Associative memory, projection learning, sparse neural network.

1 Introduction

The Associative Neural Network (AsNN) is a dynamical nonlinear system capable of processing information via the evolution of its state in high dimensional state-space. It evolved from the network of simple interconnected nonlinear elements, introduced by Hebb in 1949 as formal neurons, and received most research attention after the work of Amari [2] in 1972 and Hopfield [19] in 1982, which showed that this network possesses associative properties.

Having many parallels with biological memory, often considered as an alternative to von-Neumann processing, AsNNs offer a number of advantages over other neural network models. They provide distributed storage of information, within which every neuron stores fragments of information needed to retrieve any stored data record. The failure of several neurons does not lead to the failure of the entire network.Like most artificial neural networks, AsNNs are characterized by the parallel way of operation, which enables efficient hardware implementation [18]. At the same time, unlike many other neural paradigms (Feed-forward, RBF networks, Kohonen's SOM, etc.) AsNNs can be trained using non-iterative learning rules, which results in fast training of these networks. Among non-iterative learning rules, Projection Learning rules are known to be the most efficient, on the basis of their high capacity and best error-correction rates [27,21,15,26].

Besides being fast and deterministic, non-iterative training has another advantage. It allows one to incrementally learn new patterns as well as to delete the ones already

⋆ Address for correspondences: Institute of Mathematical Machines and Systems, Dept. of Neurotechnologies, 42 Glushkov Ave., Kiev, 03187, Ukraine. Email: olexii@mail.ru.

stored in memory [25,22]. The computational complexity of adding/erasing few images to/from memory is considerably lower than that of the complete network retraining with the modified dataset.

These properties of AsNNs make them very suitable for a variety of applications. Having high generalization and error-correction capabilities, it can be used as an efficient noise-removing filter or error-tolerant associative memory. Associative lookup is used for DB operations [5,7]. The energy minimization property allows solving of combinatory [24] and nonlinear [30,20] optimization problems (with particular application to computer networking [1,32]). The generalization abilities of AsNNs are used in many classification tasks, such as image segmentation [6] and chemical substances recognition [29]. Finally, AsNNs have been found recently very useful for video-based recognition, where they are used to associate visual stimuli to a person's name, with both memorization and recognition done in real-time [17].

When designing an associative network one has to consider the following factors contributing to the network performance:

1. Error correction capability, which shows how much noise can be tolerated by the network;
2. Capacity, which show how many patterns can be stored, given a certain error-correction rate;
3. Training complexity, which describes the amount of computations needed to memorize a pattern;
4. Memory requirements, which defines the amount of computer memory required to operate the network, which is usually a function of synaptic weights used in storing the patterns by the network; and
5. Execution time, which shows how many computations are needed in pattern retrieval.

While some of these are factors are determined by the training procedure applied to the network, the others depend on the network architecture. Recently, it has been shown [8,11] that Sparsely connected models of Associative Neural Networks (SAsNN), which use only a subset of all possible inter-neuron connections, can significantly reduce the computational and memory cost of the network, while not deteriorating much its associative quality.

This paper summarizes the state of the art in the area (Section 2) and presents a comparative performance analysis among several most popular models of SAsNNs (Section 3). It then introduces an Open Source library developed by authors, which is made publicly available to enable other researchers to develop their own models AsNNs of different configurations and architectures (Section 4).

2 Sparse network model

A general model of a sparse associative network consists of N binary discrete-time neurons connecting each other according to the following connectivity rule. Neuron j is connected to neuron i if and only if

$$j \in \{N_i\} \tag{1}$$

where $\{N_i\} \in \{1, ..., N\}$ is a subset of unique indices. The network architecture is defined by the connectivity pattern – a set of all existing inter-neuron connections:

$$\{N_i\}, i = 1, ..., N. \tag{2}$$

The connectivity pattern is characterized by the density of connections (connectivity degree):

$$\rho = \sum_{i=1}^{N} |N_i| \Big/ N^2 \tag{3}$$

and the total connection length:

$$l = \sum_{i=1}^{N} \sum_{j=1}^{|N_i|} |i - N_i[j]| \tag{4}$$

The input, or the postsynaptic potential, of the i-th neuron, is calculated as a weighted sum of the network outputs:

$$S_i = \sum_{j \in N_i} \big(1 - \delta_{ij}(1 - D)\big) W_{ij} Y_j \tag{5}$$

where W is a $N \times N$ synaptic matrix of inter-neuron connections and D is the desaturation coefficient specifying the degree of the neuron self-connections reduction [14,16].

Recognition of a pattern is performed as a result of network evolution governed by the following update rule. The output, or the potential, of the i-th neuron at the next time step is obtained after applying the sign function to the neuron input at the current time step:

$$Y_i(t+1) = \mathrm{sign}\big(S_i(t)\big). \tag{6}$$

Neuron states can be updated synchronously (all at time) or asynchronously (one at time). This paper considers the synchronous update mode only, which favors parallel processing in a hardware implementation and provides better associative properties for the network.

The energy functions of the network, of which two are defined:

$$E_H(t) = \boldsymbol{Y}(t)\boldsymbol{S}(t+1) \tag{7}$$

$$E_R(t) = \boldsymbol{Y}(t)\boldsymbol{S}(t) \tag{8}$$

guarantee that, as a result of the evolution, the network converges to a stable state, called attractor, which is given the stability condition:

$$\boldsymbol{Y}(t^*) = \boldsymbol{Y}(t) = \boldsymbol{Y}(t+1), \tag{9}$$

where $\boldsymbol{Y}$ and $\boldsymbol{S}$ are the vectors made of is neuron inputs and outputs.

It is these attractors that represents the memory content of the network. The error-correction exhibited by the network is a result of its convergence to an attractor is described by the attraction radius (AR) of the network, which is defined as the maximal

Hemming distance from which the network is guaranteed to converge to a desired attractor.

It can be seen that the size of the attraction radius, is the function entirely of the synaptic weights: the way they connect the neurons (i.e. network architecture) and the way they are computed (i.e. the learning rule).

2.1 Network architectures

Associative networks can be designed using one of the following architectures.
Fully-connected architecture. This architecture, often referred to as Hopfield-like, is studied the best in literature. It provides a fast close-form solution for the network training and allows one to store (with non-zero error-correction) up to for $M = 70\%N$ prototypes in the case of the limited size data, and up to $M = 20\%N$ prototypes when storing data from a continuous stream. Real-time nature of learning for this architecture made it very suitable for such tasks as on-line memorization of objects in video[17,13]. This architecture however requires significant amount of space to store the network weights. In particular, the amount of memory used by the network of N neurons is $N(N+1)/2 * bytes\_per\_weight$. This makes, for example, the network of size $N = 1739$ used in [17,13] occupy 3.5Mb on memory.

Random architecture. This architecture uses only a certain fraction of randomly located synaptic connections. It is easy to construct and is shown to offer a good performance for the network. It however involves many long-range interactions, which could be a problem for hardware implementation.

Local or cellular architecture. In this architecture, only the neurons satisfying the neighborhood relation (usually the location proximity) are connected. It is ideal for hardware implementation, but results in significant deterioration of associative performance, due to slow propagation of information and a tendency to form localized groups of neurons with erroneous response.

Small-World architecture. It is a combination of local and random architectures [33,3,8] and consists mostly of local connections with a few added long-range ones. It is good for hardware implementation and is found to perform almost as good as the random architecture. Many real-world networks fall into this category.

In **Scale-Free architecture**, each neuron has power-low distribution of number of connections, that is most of the neurons would have few random connections, but some of neurons would act as ”highly connected hubs” being connected to many other neurons [**?**]. It’s associative performance is very close to that of the random architecture.

Adaptive architecture. This, the most advanced, architecture is constructed according to the dataset to be stored. It offers the best associative properties (with the highest capacity per synapse) among all sparse architecture models.

2.2 Learning algorithms

During the learning stage, the AsNN is designed in such a way, that every pattern from the dataset

$$\{\boldsymbol{V}^m\}_{m=1,\ldots,M}, \boldsymbol{V}^m \in \{-1,+1\}^N \tag{10}$$

to be memorized becomes an attractor the network, defined by the stability condition of Eq. 9.

Let the network architecture Eq.2 and the training dataset to be stored Eq.10 be given. The Learning Rule (LR) is a way of finding the weight matrix W such that satisfies the connectivity constraints and makes each training data vector an attractor state with a sufficiently large radius of attraction.

Projective LR The Projective LR [23,27] is usually used for training fully-connected AsNNs. It results in the weight matrix being equal to the projective matrix for the subspace spanned on the training vectors V^m:

$$W = proj\Big(\{\boldsymbol{V}^m\}_{m=1,\ldots,M}\Big) = \mathbf{V}\mathbf{V}^+ \tag{11}$$

where $\mathbf{V}$ is a matrix made of prototype vectors as its columns.

This non-iterative LR can be implemented either by using Gram-Schmidt orthogonalization, or by using the Pseudo-Inverse operation, which allows one to computed it incrementally:

$$W_{ij}^0 = 0 \tag{12}$$

$$W_{ij}^m = W_{ij}^{m-1} + dW_{ij}^m \tag{13}$$

$$dW_{ij}^m = (V_i^m - S_i^m)(V_j^m - S_j^m)\Big/||\boldsymbol{V}^m - \mathbf{W}\boldsymbol{V}^m||^2 \tag{14}$$

Having the fully-connected AsNN trained for a particular dataset, the sparse network is obtained by simply pruning the weight matrix in accordance with the connectivity constraints (Eq. 2).

Hebbian LR The use of Hebbian learning principle for Sparse AsNN was proposed in [12]. It is often called the Local Projection LR and is computed as follows.

For some threshold value $T > 0$ and initial $W = 0$ repeat:
1. For next training vector $\boldsymbol{V}^m$, the postsynaptic potential $\boldsymbol{S}^m$ is calculated;
2. Weight coefficients of every i-th neuron with $S_i^m V_i^m < T$ are updated:

$$j \in N_i : W_{ij} = W_{ij} + \frac{V_i^m V_j^m}{N_i} \tag{15}$$

3. Until $S_i^m V_i^m > T$ for all neurons and all data vectors, or a certain number of steps is executed.

Delta LR Iterative Delta LR is an adaptation of the Widrow-Hoff Delta learning rule [34] which is used for perceptron-like models. It searches for W as a solution of (10) using the first order gradient-descent optimization:

$$W = \arg_W \min E(W) = \arg_W \| \min W\boldsymbol{V} - \boldsymbol{V}\|^2 \tag{16}$$

For some learning rate value $\alpha > 0$, the desired error value $\epsilon > 0$ and initial $W = 0$ repeat:
1. For next training vector $\boldsymbol{V}^m$, the postsynaptic potential $\boldsymbol{S}^m$ is calculated;
2. Weight coefficients of each i-th neuron are updated:

$$j \in N_i : W_{ij} = W_{ij} + \alpha(S_i^m - V_i^m)V_j^m \tag{17}$$

3. Until $E(W) < \epsilon$, or a certain number of steps is executed.

Pseudo-Inverse LR Non-iterative Pseudo-Inverse LR was originally proposed for the calculation of symmetrical weight matrices in Cellular AsNN [4]. Here we describe the simplified version of this algorithm. This version produces a nonsymmetrical weight matrix, which voids the condition for the guaranteed absolute stability of the network, but yet results in better associative performance.

To allow for structural contraints imposed by the sparse architecture, a selection operator Φ^i that sparsifies the columns of a matrix is introduced:

$$\Phi^i : (l \times n) \rightarrow (l \times N_i) \tag{18}$$

This operator retains only those columns of its matrix argument that correspond to neuron indices contained in N_i.

Denoting the i-th row of the training data matrix as $\mathbf{V}_i$, the weights of the i-th neuron are calculated as a solution of the following ”fixed point” equation:

$$\Phi^i(W^i) \cdot \Phi^i(\mathbf{V^T})^\mathbf{T} = \mathbf{V_i} \tag{19}$$

The solution to this equation can be found using the matrix pseudo-inversion operator:

$$\Phi^i(W^i) = \mathbf{V}_i \cdot \left(\Phi^i(\mathbf{V^T})^\mathbf{T}\right)^+ \tag{20}$$

3 Comparative performance analysis

To compare the associative performance of the networks trained with different architectures and learning algorithms, we provide experimental results obtained using random data vectors with independent non-biased components $Y_I \in \{-1, +1\}$.

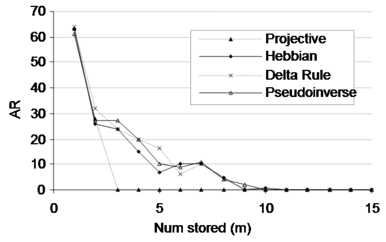
3.1 Influence of learning rule

Figure 1a shows the error-correction rates, measured by the average Attraction Radius of the network, as a function of the learning rule. The network of size $N = 256$ neurons with cellular architecture defined by the following connectivity criterion is used:

$$j \in N_i \iff (i - j) \bmod N \leq 2r \tag{21}$$

where r is the Connection Radius of the network with the value $r = 12$, which corresponds to the network with connectivity degree $\rho = (2r + 1)n/n^2 \approx 0.1$.

The threshold parameter for the Hebbian LR is set to $T = 10$, error value for Delta LR $\epsilon = 0.0001$. For each new data vector added to the memory, Attraction Radius is calculated, as maximum amount of noise which is guaranteed to be completely removed by the network.

a)


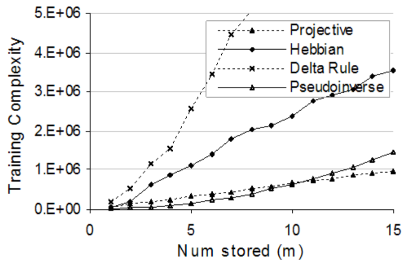
b)


Fig. 1. Associative performance and training complexity as function of learning rule.

Figure 1b shows the computational complexity of the training algorithms, which is calculated as follows. For the iterative learning rules, it is taken equal to $2rnlm$ where l is the number of training epochs. For the Projective LR, the training complexity is taken as the complexity of training the fully-connected network, which is $n2m$. For the Pseudo-Inverse LR, it equals $n2rm2$, where $2rm2$ is the complexity of the solution Eq. 20 for a single neuron.

3.2 Influence of network architecture

It is not only the training algorithm that determines the associative properties of the network, but also the number and the location of the synaptic connections.

The empirical study from [31] shows that trained fully connected Hopfield network still performs well even after the removal of 80% of neuron connections that have the least absolute values. The location of the remaining connections becomes of great importance for the associative properties of the network and reflects the hidden interrelationships of stored data. This selection rule can be used to set the architecture of the sparse network for further training.

Work [10] studies the sparse AsNN learnt with Pseudo-Inverse LR , the architecture of which is set according to above decribed weight selection rule and shows that the performance of the networks with such an adaptive architecture is superior to the networks with other types of connectivity. This is further illustrated in Figure 2, which shows the attraction radius as a function of the connectivity degree for the several sparse associative networks: with Adaptive (PI WS), random, cellular 21 and anti-Adaptive (PI WS Reverse) architectures. The last one is based on the least absolute values computed with the fully connected Hopfield network. Attraction radius for the network of $N = 200$ neurons which stores $M = 20$ patterns is computed.

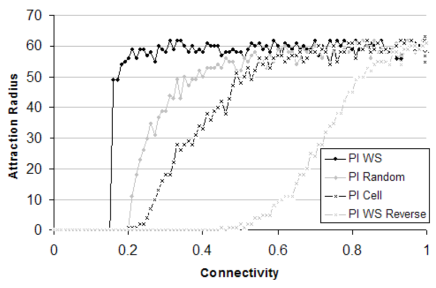


Fig. 2. Associative performance of the network a function of the network architecture.

It can be seen that the network with adaptive architecture approaches the associative performance of its fully-connected counterpart using less than 20% of available connections. It also be noticed that the performance of the network changes drastically, in phase transition manner: from no associative recall at all to a very good associative recall, when only a few extra neurons are added. As shown in [11], the network connectivity degree at which such a phase transition happens depends linearly on normalized memory loading.

It can also be mentioned that the idea of Adaptive Architecture selection can also be used to implement the efficient rewiring of the networks with Small-World connectivity architecture [9], where the architecture aims to minimize the total connection length of Eq. 4.

4 Library

The associative neural network models described in this paper are implemented as an Open Source *Associative Neural Network Library*, written in C++, the description of which follows.

4.1 Classes

The relationship between the main library classes is shown in Fig. 3. Solid lines mark the inheritance and dashed ones mark the ”uses” relationship.

The base class of all associative networks is `AssociativeUnit`. It defines the interface of two main functions `train` and `recall` to store and retrieve data pairs $\{X, Y\}$ (stored in `IOData` class) in hetero-associative memory fashion.

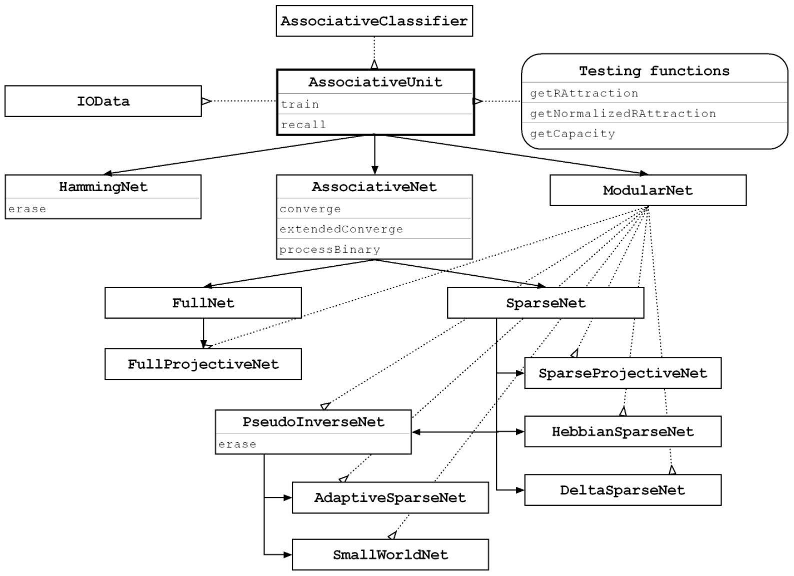


Fig. 3. Hierarchy of main classes in Open Source *Associative Neural Network Library*.

The derived class `AssociativeNet` is the base class for autoassociative neural networks. The most important function of this class is `converge` function which implements the dynamical behavior of the network, i.e. the convergence process used to retrieve data from the memory. Extended version of this function `extendedConverge` is used to analyze the behavior of the convergence process itself, as it keeps track of all points visited by the network during its evolution in the state-space and signalizes if a cycle occurs.

The memory desaturation technique (Eq.5) is implemented in the same class by using the `desatCoeff` parameter. For sparse neural networks with rapidly growing diagonal elements the near-optimal value of desaturation coefficient can be set by calling `adjustDesaturationCoefficient`

$$D = 0.15 \frac{M}{trace(W)} \tag{22}$$

Usually only few neurons change their state between two consecutive convergence steps. Therefore the new postsynaptic potentials can be calculated incrementally by updating only those of them that have changed since the last iteration This incremental procedure, called Update Flow Technique [16], increases significantly the speed of convergence process, compared to the direct calculation of the postsynaptic potential by Eq.5. The interface for optimized postsynapse calculation is specified by `processBinary` function.

Two classes `FullNet` and `SparseNet` are derived from `AssociativeNet` and provide the weights storage for fully connected and sparse network models. Both classes implement the processBinary function which supports the memory desaturation technique. Weights in `SparseNet` are stored in two arrays: as array of weight values and as array of weight indices. Network architecture can be set to 1D local (Eq.16), 2D local, random by calling one of `set*Architecture` functions.

Classes derived from `FullNet` or `SparseNet` implement the learning algorithms. Fully-connected AsNN with projective LR is implemented by `FullProjectiveNet` class. Its sparse analogue with the same LR is implemented by the class `SparseProjectiveNet`.

Classes `HebbianSparseNet` and `DeltaSparseNet` correspond to sparse networks with iterative Hebbian and Delta iterative LRs. Non-iterative Pseudo-Inverse LR is implemented by `PseudoInverseNet` class with an extra `erase` operation that enables incremental deletion of patterns from the memory. Two classes, `AdaptiveSparseNet` and `SmallWorldNet` are derived from `PseudoInverseNet` and provide the functionality for the construction on networks with more efficient architectures that can exploit the correlations in the stored data.

`ModularNet` represents the associative memory model that uses a set of AsNN (modules) organized in a growing tree structure. It has the ability to store amounts of data exceeding by far its dimension, thus overcoming the memory limitation of a single AsNN [28]. Any AsNN class can be used as the building block of `ModularNet`.

4.2 Testing functions

The main testing functions `getRAttraction` and `getNormalizedRAttraction` are used to estimate absolute (Eq.8) and normalized AR [21] of the network. Network capacity, equal to the maximum number of vectors that can be stored in network subject to certain value of AR, is calculated by `getCapacity` function. Testing function can be applied to any subclass of `AssociativeUnit`.

4.3 Associative Classifier

Though the classification task represents a hereroassociative mapping, it can also be solved using the AsNNs. The idea is to store both the input features and the class label as a single state-vector. At the classification stage, the part of the AsNN input corresponding to input features is set, while the other part, representing class label, remains unspecified (or set to some random values). The missing class information is then restored during the convergence process due to the pattern-completion capabilities of the AsNN. This functionality is implemented by the `AssociativeClassifier` class.

Using the AsNN as a hereroassociative classifier was found very useful for real-time applications, the example of which is face memorization/recognition from video done in [17], where the name of the person in video is recovered as a result of the network evolution from the state defined by the visual stimulus presented to the network.

5 Conclusion

This paper provided a motivation and the necessary background needed for implementing associative neural networks of different architectures. It has also presented experimental results comparing the performance of different sparse associative neural networks and introduced the *High Performance Associative Neural Network* library which allows one to implement these networks.

This library is made publicly available at http://synapse.vit.iit.nrc/memory/pinn, mirrored at http://perceptual-vision.com/memory/pinn. Its development was motivated by several factors, the most important of which are the associative properties of these networks which make them very suitable for many applications. Recent results in applying these networks to one of the most challenging computer vision tasks — real-time memorization and recognition of faces from video, is a clear demonstration of this.

Acknowledgements

The sparse associative neural network library described in this paper is implemented by the first author of the paper, as part of his PhD project at the Department of Neurotechnologies of the Institute of Mathematical Machines and Systems of the Ukrainian Academy of Sciences.

References

1. C. Ahn, R. Ramakrishna, C. Kang, and I. Choi. Shortest path routing algorithm using hopfield neural network. *Electronics Letters*, 37(19)(19):1176–1178, 13 September 2001.
2. S. I. Amari. Learning patterns and pattern sequences by self-organizing nets. *IEEE Transactions on Computers*, 21(11):1197–1206, November 1972.
3. J. W. Bohland and A. A. Minai. Efficient associative memory using small-world architecture. *Neurocomputing*, 38:489–496, 2001. Elsevier.
4. M. Brucoli, L. Carnimeo, and G. Grassi. Discrete-time cellular neural networks for associative memories with learning and forgetting capabilities. *IEEE Transactions on Circuits and Systems*, 42:396399, July 1995.
5. C.-H. Chen and V. Honavar. A neural network architecture for high-speed database query processing. *Microcomputer Applications*, 15:7–13, 1996.
6. K.-S. Cheng, J.-S. Lin, and C.-W. Mao. The application of competitive hopfield neural network to medical image segmentation. *IEEE Transactions on Medical Imaging*, 15(4)(4):560–567, August 1996.
7. Y.-M. Chung, W. M. Pottenger, and B. R. Schatz. Automatic subject indexing using an associative neural network. In *3rd ACM International Conference on Digital Libraries*, pages 59–68, Pittsburgh, Pennsylvania, United States, June 23-26 1998.
8. N. Davey, B. Christianson, and R. Adams. High capacity associative memories and small world networks. In *In Proc. of IEEE IJCNN*, Budapest, Hungary, 25-29 July 2004.
9. O. Dekhtyarenko. Systematic rewiring in associative neural networks with small-world architecture. In *International Joint Conference on Neural Networks (IJCNN'05)*, pages 1178–1181, Montreal, Quebec, Canada, July 31 - August 4 2005.
10. O. Dekhtyarenko, A. Reznik, and A. Sitchov. Associative cellular neural networks with adaptive architecture. In *The 8th IEEE International Biannual Workshop on Cellular Neural Networks and their Application (CNNA'04)*, pages 219–224, Budapest, Hungary, July 22-24 2004.
11. O. Dekhtyarenko, V. Tereshko, and C. Fyfe. Phase transition in sparse associative neural networks. In *European Symposium on Artificial Neural Networks (ESANN'05)*, Bruges, Belgium, April 27-29 2005.

12. S. Diederich and M. Opper. Learning of correlated patterns in spin-glass networks by local learning rules. *Physical Review Letters*, 58(9):949–952, March 2 1987.
13. D. Gorodnichy. Projection learning vs correlation learning: from pavlov dogs to face recognition. In *AI'05 Workshop on Correlation learning, Victoria, BC, Canada, NRC 48209*, 2005.
14. D. Gorodnichy and A. Reznik. Increasing attraction of pseudo-inverse autoassociative networks. In Neural Processing Letters, *volume 5, issue 2, pp. 123-127*, 1997.
15. D. O. Gorodnichy. The optimal value of self-connection or how to attain the best performance with limited size memory. In *Proc. of IJCNN'99, Washington, DC, USA*, 1999.
16. D. O. Gorodnichy. The influence of self-connection on the performance of pseudo-inverse autoassociative networks. In *"Radio Electronics. Computer Science. Control" journal, Vol. 2, No. 2, pp. 49-57, online at http://zstu.edu.ua/RIC/pdf/01_2_2.pdf*, 2001.
17. D. O. Gorodnichy. Associative neural networks as means for low-resolution video-based recognition. In *International Joint Conference on Neural Networks (IJCNN'05), Montreal, Quebec, Canada, NRC 48217*, 2005.
18. A. Heittmann and U. Ruckert. Mixed mode vlsi implementation of a neural associative memory. *Analog Integrated Circuits and Signal Processing*, 30(2):159–172, February 2002.
19. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc Natl Acad Sci USA*, 79(8):2554–2558, 1982.
20. A. Jagota. Approximating maximum clique with hopfield networks. In *IEEE Transactions on Neural networks, Vol.6, No.3, pp.724-735*, 1994.
21. I. Kanter and H. Sompolinsky. Associative recall of memory without errors. *Physical Review A*, 35:380392, 1987.
22. N. Kirichenko, A. Reznik, and S. Shchetenyuk. Matrix pseudoinversion in the problem of design of associative memory. *Cybernetics and Systems Analysis*, 37(3):308–316, May 2001.
23. T. Kohonen. Correlation matrix memories. *IEEE Transactions on Computers*, 21:353–359, 1972.
24. Y. Liang. Combinatorial optimization by hopfield networks using adjusting neurons. *Information Sciences: an International Journal*, 94(1-4):261–276, October 1996.
25. S. Mohideen and V. Cherkassky. On recursive calculation of the generalized inverse of a matrix. *ACM Transactions on Mathematical Software (TOMS)*, 17(1):130–147, March 1991.
26. R. A. Neil Davey and S. Hunt. High capacity recurrent associative memories. In *Neurocomputing*, 2004.
27. L. Personnaz, I. Guyon, and G. Dreyfus. Collective computational properties of neural networks: New learning mechanisms. *Physical Review A*, 34(5):42174228, November 1986.
28. A. Reznik and O. Dekhtyarenko. Modular neural associative memory capable of storage of large amounts of data. In *International Joint Conference on Neural Networks (IJCNN'03)*, Portland, Oregon, US, July 20-24 2003.
29. A. Reznik, A. Galinskaya, O. Dekhtyarenko, and D. Nowicki. Preprocessing of matrix qcm sensors data for the classification by means of neural network. *Sensors and Actuators B*, 106:158–163, 2005.
30. I. Silva, L. Arruda, and W. Amaral. Nonlinear optimization using a modified hopfield model. In *IEEE World Congress on Computational Intelligence*, pages 1629–1633, Anchorage, AK, USA, May 4-9 1998.
31. A. Sitchov. Weights selection in neural networks (in russian). *Mathematical Machines and Systems*, 2:25–30, 1998.
32. V.-M.-N. Vo and O. Cherkaoui. Traffic switching optimization in optical routing using hopfield networks. In *Recherche Informatique Vietnam & Francophonie (RIVF04)*, pages 125–130, Hanoi, Vietnam, February 2-5 2004.
33. D. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, Jun 1998.
34. B. Widrow and J. M.E. Hoff. Adaptive switching circuits. *IRE Western Electric Show and Convention Record*, 4:96–104, August 23 1960.