Neural-Symbolic Learning and Reasoning
AAAI Technical Report WS-12-11

Towards Bridging the Gap Between Pattern Recognition and Symbolic Representation Within Neural Networks

Tsvi Achler
Los Alamos National Labs
achler@gmail.com

Abstract

Underlying symbolic representations are opaque within neural networks that perform pattern recognition. Neural network weights are sub-symbolic, they commonly do not have a direct symbolic correlates. This work shows that by implementing network dynamics differently, during the testing phase instead of the training phase, pattern recognition can be performed using symbolically relevant weights. This advancement is an important step towards the merging of neural-symbolic representation, memory, and reasoning with pattern recognition.

Introduction

Sensory recognition is an essential foundation upon which cognition and intelligence is based. Without recognition the brain cannot interact with the world e.g.: form internal understanding, memory, logic, display creativity, or reason. Rich symbolic information is present in learned recognition representations. However it remains difficult to implement high-level human cognitive processes (e.g., rule-based inference) using low-level, brain-like architectures that can perform simple recognition (e.g., neural networks). A significant part of the difficulty arises because weights favorable for recognition relate poorly to symbolic representations (e.g. Sun 2002). Thus cognitive models using symbolic representations often assume that sensory recognition is previously processed and symbolically-coded representations are available. This often limits the literature to less-satisfying examples disconnected from the foundations of recognition models. A neural form of information with better access to underlying sub-symbolic relations will allow closer integration of symbolic systems with neural network structures. This work presents an artificial neural network structure that can utilize symbolic-type weights.

In order to understand how symbolic-type weights are attained, it is important to understand neural network structures. Neural recognition methods are best described based on their testing configuration (their recognition-phase structure). Most supervised recognition algorithms (e.g. neural networks) perform recognition using feedforward weights during testing, thus described as feedforward models. These methods may learn the feedforward weights during training through dynamic, gradient-descent methods: using feedforward and feedback connections iteratively to adjust weights. However, once the weights are learned, testing uses feedforward weights (e.g. Rosenblatt 1958; Rumelhart & McClelland 1986).

The key innovation presented here is a supervised recognition algorithm that uses a feedforward-feedback configuration to solve recognition where the dynamic-phase, based on gradient-descent, is implemented during testing. This allows learning to be much simpler and the weights to represent symbolic-like expectations.

It is shown that equivalent recognition can be implemented using either feedforward or feedforward-feedback methods. However, the feedforward-feedback configuration is especially advantageous to the neural-symbolic community because it learns and uses symbolically-relevant weights. These weights can in turn be better integrated with cognitive systems.

Structure/Method	Dynamics	Weights	Relation
Feedforward	During Learning	**W**	Sub-Symbolic
Feedforward-feedback	During Testing	**M**	Symbolic

Table 1: comparison of weights and dynamics between feedforward and feedforward-feedback supervised classifiers

The following sections review the recognition literature and derive a neural network that solves recognition with symbolic-type weights. Subsequent sections provide examples to demonstrate: symbolic weights, how learning changes, how dynamics differentiates between algorithms, and recognition systems using multiple patterns. Finally, the current limitations of this method and future work are discussed.

Background

Cognitive Symbolic Systems

Brain-motivated cognitive models that utilize symbolic representations are designed to explain how the brain can perform cognitive tasks and/or to understand the origins of its observed limits. A small sample of such models include: SHRUTI, LIDA, SOAR, ACT-R, CLARION, EPIC, ICARUS (e.g. Franklin & Patterson 2006; Laird 2008; Meyer & Kieras 1997; Shastri 2000; Sun 2002). Whether symbolic systems are more biologically-motivated or rule-based, most assume that sensory recognition is previously processed and symbolically coded representations are available. The processed representations are often hand coded. From this starting point, the cognitive-symbolic processing is implemented. This confines many symbolic systems to less-satisfying examples with synthetic starting points. An advancement where traditionally sub-symbolic information present in neural networks becomes available to symbolic systems, would be beneficial.

Recognition Algorithms

Although many recognition algorithms have been developed over the past half century, supervised recognition systems share a commonality that they predominantly utilize a feedforward architecture during testing. Based on feedforward weights **W** they solve the recognition relationship:

$$\mathbf{Y}=\mathbf{W}\mathbf{X} \quad \text{or} \quad \mathbf{Y}=f(\mathbf{W},\mathbf{X}) \qquad (1)$$

Y represents the activity of a set of labeled nodes that may be called output neurons, or classes in different literatures and individually written as $\mathbf{Y}=(Y_1, Y_2, Y_3,\ldots Y_H)^T$. In supervised classifiers the nodes are labeled, for example: Y_1 is associated with *"dog"*, Y_2 is associated with *"cat"*, and so on. Vector **X** represents sensory nodes that sample the environment, or input space to be recognized, and are composed of individual features $\mathbf{X}=(X_1, X_2, X_3,\ldots X_N)^T$. The input features can be sensors that detect edges, lines, frequencies, and so on. **W** represents a matrix of weights or parameters that associates inputs and outputs. The relationship **WX** calculates the output using the feedforward weights and inputs. Thus the direction of information flow *during recognition* is feedforward: one-way from inputs to the outputs.

Learning **W** weights may require feedback, where the outputs are compared to supervised labels and the error signals from the outputs are projected to the inputs using methods such as the delta rule (e.g. Rosenblatt 1958). However once **W** is defined, recognition during testing is feedforward.

This configuration can be found within the majority of recognition algorithms, for example: Single-layer Perceptrons (Rosenblatt 1958), multilayer Neural Networks with nonlinearities introduced into calculation of **Y** (Rumelhart & McClelland 1986), and machine learning methods such as Support Vector Machines (SVM) with nonlinearities introduced into the inputs through the *kernel trick* (Vapnik 1995). Although these algorithms vary in specifics such as nonlinearities determining the function *f*, they share the commonality in that recognition involves a feedforward transformation using **W** during recognition.

Some feedforward algorithms include lateral connections for competition between output nodes **Y**. One variant, Adaptive Resonance Theory (Carpenter & Grossberg 1987) measures a goodness-of-fit after a winner-take-all competition. However such competition methods still rely on initially calculating **Y** node activities based on the feedforward transformation **W**. Thus they fall into the feedforward category.

Recurrent networks are feedforward networks in a hierarchy where a limited number of outputs are also used as inputs. These networks can be unfolded into a recursive feedforward network e.g. (Schmidhuber 1992; Williams & Zipser 1994; Boden 2006). Thus they also fall into a feedforward category.

Networks based on feedforward-feedback structures are auto-associative networks and generative models. Typically in auto-associative networks, the feedforward and feedback connections have the same weights; however there are exceptions (McFadden et al 1992; Bogacz et al 1998). In generative models the feedback weight is the opposite sign of the feedforward weight. Generally auto-associative networks function during testing while supervised generative models are used to aid learning.

Two famous auto-associative networks are Anderson's "brain-state-in-a-box" and Hopfield networks (Anderson et al 1977; Hopfield 1982). In these networks when part of a learned input pattern is given, the network can complete the whole pattern through a dynamic process. However, the auto-associative networks do not perform supervised classification and solve equation 1 above.

Generative models are used to generate patterns and subtract them from the inputs. This is a strategy used to optimize learning of Bolzman (binary activation) networks e.g. (Hinton & Salakhutdinov 2006) and unsupervised networks where outputs are unlabeled e.g. (Olshausen & Field 1996). However these methods do not use the feedforward-feedback component during testing of supervised networks. For example it common to see an unsupervised generative network followed by a supervised feedforward classifier e.g. (Zieler et al 2010).

Given the context of the current literature, the recognition method proposed here uses feedforward-feedback connections to perform supervised recognition with symbolic-like weights and solve equation 1 dynamically during recognition. Let's derive how it works.

Mathematical Derivations

Equation 2 of a linear perceptron can be rewritten using an inverse of equation 1:

$$W^{-1}\vec{Y} = \vec{X} \qquad (2)$$

Lets define matrix **M** as the inverse or pseudoinverse of matrix **W**. The relation becomes:

$$M\vec{Y} - \vec{X} = 0 \qquad (3)$$

Models founded on this equation are referred to as generative models because they "generate" a reconstruction. The term **MY** is an internal prototype of pattern(s) constructed using learned information **M**, that best matches patterns present in **X**.

Information flows from **Y** to **X** using **M** and recognition is determined in the input or **X** domain. This flow of information describes top-down feedback, the opposite direction of feedforward. The fixed-points or solutions of equations 3 and 1 are identical, so the same **Y**'s match the feedforward and feedback equations. This duality suggests that analogous network connectivity can be described using both feedforward and feedback perspectives. The feedback perspective is more symbolically relevant.

However equation 3 does not provide a way to project input information to the outputs. To get around this, we use dynamic networks or equations that converge to equation 3. One method that can be used is based on least-squares to minimize the energy function: **E=||X - MY||^2**. Taking the derivative relative to **Y** and solving the equation becomes:

$$\frac{d\vec{Y}}{dt} = M^T\left(M\vec{Y} - \vec{X}\right) \qquad (4)$$

This equation can be iterated until steady state, dy/dt=0 (with no weights adjusted) resulting in the fixed point solution that is equivalent to **Y=WX**. Both feedforward and feedback weights are designated by **M**. **MY** transforms **Y** information into the **X** domain, thus a feedback process. Transposed weights **M^T** transform **X** information into the **Y** domain, thus a feedforward process. Another way to converge to equation 3 is to use Regulatory Feedback e.g. (Achler 2011; Achler & Bettencourt 2011). The equation can be written as:

$$\frac{d\vec{Y}}{dt} = \vec{Y}\left(\frac{1}{V}M^T\left(\frac{\vec{X}}{M\vec{Y}}\right) - 1\right) \text{ where } V = \sum_{j=1}^{N} M_{ji} \qquad (5)$$

In alternative notation, this can be written as:

$$\frac{dY_i}{dt} = \frac{Y_i}{\left(\sum_{j=1}^{N} M_{ji}\right)} \sum_{k=1}^{N} M_{ki}\left(\frac{X_k}{\sum_{h=1}^{H} M_{kh}Y_h}\right) - Y_i \qquad (6)$$

where M_{NxH} are the dimensions of **M**. Both generative-type models have the identical fixed points (Achler & Bettencourt 2011).

In summary, **Y** can be found using **M** in the testing domain. The derivations establish that the same solutions to equation 1 can be solved by the generative method in equations: 4, 5, or 6. The end point solutions or purpose of networks are indicated by the weights in **M**. This forms the basis of symbolic association where the role of weights can be understood from **M**.

Demonstrations

Several examples are given here. The first is a simple example showing how feedforward-feedback weights **M** are symbolic as opposed to feedforward weights **W**. The second shows that generalized learning can be achieved intuitively using **M**. The third refers to multiclass classification scenarios.

A Simple Example Using Symbolic Weights

Let's define a matrix that represents expectations and let's pose a simplified recognition problem. Suppose we want to discriminate between drawings of a moving bicycle or unicycle using features of wheels horizontal lines and vertical lines. We can describe the expectation based of those features in the expectation matrix **Ex** below.

$$\text{Exp=}\begin{array}{cccccc} X_1 & X_2 & X_3 & X_4\ldots & & \\ \left[2\right. & 1 & 1 & \left.1\right] & Y_1 & \text{Moving Bicycle} \\ \left[1\right. & 0 & 1 & \left.1\right] & Y_2 & \text{Moving Unicycle} \end{array} \qquad (7)$$

Expectation matrix **Exp** indicates the differences between bicycles and unicycles based on features X_1= circles (wheels), X_2: horizontal lines, X_3: vertical lines, X_4 : seat features. Although binary values are given they can be any real number. For example if 50% of bicycles have seats then the entry can be 0.5.

Two wheels are expected in a bicycle. Horizontal and vertical frame features are expected in a bicycle. One wheel is expected and no vertical frame features are expected in a moving unicycle.

A matrix such as this is easy to learn since it only requires the expectation of the features relative to the label to be stored. This can be determined by a simple averaging function or co-occurrence of features with labels. It can also be determined by symbolic expressions.

In the above example, given a supervised classifier if **X**= $[1,0,1,1]^T$, a unicycle, then the system should respond with **Y**=$[0, 1]^T$, indicating a unicycle label.

Optimal Feedforward Weights are Sub-Symbolic

Even though supervised weights **W** may store image-label associations as well, it is not trivial to incorporate symbolic expectations into **W**. Let's assume feedforward weights represent symbolic expectations and set **W0 = Exp** to demonstrate the difficulty. Let's set the input to represent a moving unicycle: $\mathbf{X_{test}} = [1,0,1,1]^T$. How does recognition fair? Solving $\mathbf{Y=W0\ X_{test}}$ we get $\mathbf{Y}=[4\ 3]^T$. This is not the expected solution: $\mathbf{Y}=[0,\ 1]^T$. **W** should be trained using the symbolic expectation matrix as a training set.

Solving recognition directly using M

Using the proposed method one should be able to: take the expectation matrix and make it equal to **M** (**M1=Exp**), then insert **M1** and $\mathbf{X_{test}}$ into either equations 5 or 6, and wait until the dynamics go to zero: $d\mathbf{Y}/dt\rightarrow 0$. The solution for $\mathbf{X_{test}} = [1,0,1,1]^T$ is $\mathbf{Y}=[0,1]^T$. $\mathbf{X_{test}}$ is correctly recognized as a unicycle. Now if $\mathbf{X_{test}}=[2,1,1,1]^T$ representing a bicycle, then $\mathbf{Y}=[1,0]^T$. Thus both patterns are correctly recognized. This demonstrates how recognition can be achieved using the expectation matrix.

M vs. W

To demonstrate the relation of **M** to **W** and feedforward recognition, let's go back to equation 2 and calculate **W1** from **M1** using the pseudoinverse. Since these matrixes may not be square, the standard pseudoinverse method is used where $\mathbf{W}=(\mathbf{M}^T\mathbf{M})^{-1}\mathbf{M}^T$. The transpose **W1** is shown.

$$\mathrm{W1}= \begin{bmatrix} 0.4 & 0.6 & -0.2 & -0.2 \\ -0.2 & -0.8 & 0.6 & 0.6 \end{bmatrix} \qquad (8)$$

W represents the feedforward weights. **W1** is more complex, has negative values, and the values are sub-symbolic. To demonstrate that **W1** denotes the correct feedforward weights lets calculate $\mathbf{Y=W1\ X_{test}}$. The answers are again $\mathbf{Y}=[1,0]^T$ for $\mathbf{X_{test}}=[2,1,1,1]^T$ and $\mathbf{Y}=[0,1]^T$ for $\mathbf{X_{test}} = [1,0,1,1]^T$. The same answers were obtained from equation 5 using expectation values from matrix 7, as the feedforward method in equation 1 using **W** values from matrix 8. The disadvantage of the feedforward method is that **W** is sub-symbolic.

Retrieving Symbolic Information

Suppose now we want to ask symbolic questions: do bicycles have wheels? How many? Using **M1** we can look up label for bicycle Y_1 and feature for wheel X_1 and read the value: 2. If we want to do the same thing for unicycle we can look up label for unicycle Y_2 and feature for wheel X_1 and read the value: 1. If we attempt this with **W1** we do not retrieve symbolically useful information. Thus **M** has symbolic properties while also representing a neuraly-plausible recognition model.

Learning Example Using Symbolic Weights

We have shown the relationship between **M** and **W**. The purpose of this example is to show that learning through **M** can achieve the same results as learning through **W**, but learning through **M** can be simpler, faster, and more intuitive. This example demonstrates the training and classifying of linearly separable data. 400 training data points are generated and separated into two labels along a linear separation (see figure 1). 100 separate points are generated as test points.

Both a single-layer linear preceptron and the feedforward-feedback method are trained on the same learning data and tested on the same testing data. The performance, number of cycles, and computation time are calculated.

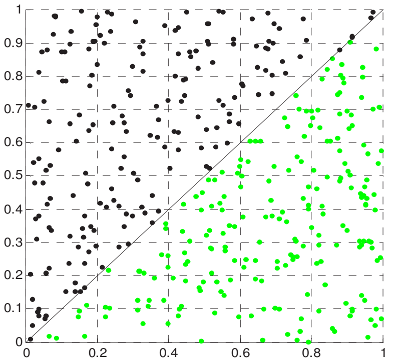

Figure 1 training and testing data points separated by a linear boundary.

Testing the Linear Perceptron

The perceptron network is trained using the 400 samples until **W** is found where the number of errors on the training sample is 0. The learning rate is 0.5. The number of iterations required and the computer time on a PC running matlab are recorded. Then the perceptron is tested on the 100 tests. The number correct and the computer time required to perform the tests are also recorded. The exact numbers are not as important as comparing when each method takes more time. The number of required iterations for training the perceptron varied depending on the random sets and initial conditions. A typical data set required about 6000 iterations and about 18 seconds.

Testing the perceptron on the 100 samples was very fast and required about 0.001 of a second. The perceptron did not have any testing errors as long as the testing points did not fall on the linear separator.

Testing the Feedforward-Feedback Algorithm

The feedforward-feedback algorithm is trained and tested on the same data. Performance, number of iterations and time are also recorded and compared.

To determine the expectation matrix **M** from the training data, the points corresponding to each label are averaged. All of the black points above the line and all of the green points below the line are averaged together. The resulting values from one run are shown in equation 9. Calculating the mean is not iterative and took only 0.02 seconds.

$$M = \left[\qquad \qquad \right]$$

The regulatory feedback equations (5, 6) were used for testing. The time required was about 0.2 seconds for the 100 tests total. The average number of iterations per test was 21. The time and number of iterations were sensitive to the threshold value of d**Y**/dt used to stop the simulation. When all nodes of **Y** changed less than 0.0001 then the simulation was stopped. The identification of **Y** was determined by the node within **Y** with the highest value.

The performance was analogous to the perception and did not have any testing errors as long as the testing points did not fall on the linear separator.

Comparing Methods

The feedforward-feedback method was much faster in learning, about 900 times faster. However the percetron was much faster in testing, over 20 times faster. This is because the dynamic aspect of the perceptron algorithm is during learning and the dynamic aspect of the feedforward-feedback algorithm is during testing. The dynamics take the most time. It is important to note that the dynamics of the feedforward-feedback testing was faster than those of the perceptron in part because the perceptron learning requires dynamics on all of the training data, while the feedforward-feedback dynamics were only required for individual test data points. Thus dynamics during testing is more efficient because it does not have to iterate using all data points, only for each test point individually as needed. In the benchmarks, the time reported is combined for all of the 100 tests. Beyond the test and training times, both methods performed similarly and both are governed by the same limitations. These limitations are discussed further in the discussion section.

Multiclass Classification

We showed differences in symbolic-like relations between **M** & **W** followed by differences in learning. Here we point the reader to comparisons in multiclass classification. A random pattern set is generated with arbitrary supervised input-label patterns. We tested up to 200 patterns. The most important issue in both the linear perceptron and the feedforward-feedback network is to make sure that there are more input features than patterns. Otherwise, the patterns will not be linearly separable. As long as the patterns are linearly separable, when each pattern is presented, the networks correctly identify the patterns.

For further analysis, the reader is encouraged to look at previous multiclass classification work e.g. (Achler, Omar, Amir 2008, Achler, Vural, Amir, 2009; Achler 2011). In that work, the input patterns are further manipulated and presented as mixtures to demonstrate mixture processing.

$$\begin{array}{cccl} X_1 & X_2 & & \\ 0.36 & 0.69 & Y_1 & \text{Black Label} \\ 0.66 & 0.33 & Y_2 & \text{Green Label} \end{array} \qquad (9)$$

Discussion

In feedforward algorithms the fixed points (stable solutions) are opaque and cannot be directly observed through matrix **W**. For equations 1-6 including the feedforward equations, **M** describes the fixed points or solutions. This is why determining network behavior using **M** is easier. The solutions of the network mirror the values of **M**. Thus performing recognition based on **M** has two advantages: 1) learning is easier because the desired network can be set by simple, representative entries in **M**, 2) the solutions or purpose of the network and its components can be read from **M**. These properties form the basis of the neural-symbolic associations shown here. Thus the advantage of using **M** and feedforward-feedback architectures for the symbolic community is a new ability to model high-level human cognitive processes (e.g. rule-based inference) directly from brain-like architectures that perform recognition.

M can be determined by the expectation of the features relative to the label to be stored. Expectation can be determined by a simple averaging function, co-occurrence, or symbolic-like expressions between features with labels. This does not require feedback dynamics or error propagation during learning. Subsequently, **M** can be learned using simple Hebbian-type learning, or one can calculate the likelihood a feature is present when a label is present, or one can average the number of times a feature is present when a label is present as was done in the learning example presented here. There are other possible methods to optimally determine expectation or label-to-feature correlation. Such methods may use hierarchy, focus on reducing condition numbers, or avoid ill-formed matrices.

Ill formed matrices are defined as an ambiguous situation when **M** has linearly dependent rows. This means the same pattern has two labels. This violates the linear dependence limitation imposed on both the perceptron and the feedforward-feedback methods.

This limitation is important because even after an ideal learning, degraded information or illusions may cause an ill-posed state equivalent to an ambiguous scenario. For example if common information to two nodes is present in **X**, but no distinctive information between the two nodes is available, the two nodes can behave as if they are linearly dependent. This can also happen for example if a data point falls on the linear separator line. Two or more labeled nodes will be equidistant from that point.

With an ill-formed **M**, the network does not have enough information to choose from dependent patterns. In such case, the initial conditions will determine which of the linear dependant patterns will predominate. This is not catastrophic for the rest of the network, because the dependent nodes interact with the rest of the network as one node. Subsequently, other nodes in the network can still be recognized correctly (Achler 2011). Thus the network obtains a reasonable result even for linearly dependent, ill-formed matrices.

Similar difficulties can occur within the learning dynamics of feedforward methods. Thus it is also fair to ask similar questions about the dynamics of learning **W**. In some cases learning may not converge in scenarios where data is not separable. Failure during feedforward learning may be more problematic because all of the representations are learned together. Poor convergence affects learning of all representations and it may not be as clear which of the representations are problematic. Moreover, the larger the data set the more this is likely (Bottou & Le Cun 2004).

The symbolic properties of **M** can help answer questions such as do bicycles have wheels. It may also be possible to manipulate **M** for more complex recognition-logic questions. For example, a neuro-symbolic classifier may be extended to answer more complex questions such as do cows have opposable thumbs. Suppose cow and thumb have been learned. It is possible based on the label cow to generate the patterns composed of features associated with cows. Subsequently those patterns can be used as inputs. A node of interest (e.g. thumb) can be evaluated for its response to the generated patterns. Such possibilities are referred to as recognition logic and planned for future work.

The most important finding presented here is that the form of information stored in **M** represents fixed points, and forms a bridge between weights favorable for recognition and weights favorable for neural-symbolic representations. Without the weights representing fixed points, it is not possible to manipulate the representations and perform symbolic analysis on the stored patterns.

It is important to state, as any work on recognition and classification work should, that all of the methods proposed suffer from difficulties of recognition in the real world involving: shading, lighting, size or rotation invariance, size invariance, and so on. So there is still much work to be able to perform recognition like a human.

In summary, this paper presents a method towards bridging the gap between pattern recognition and symbolic representation within neural networks. With the advent of recognition with more symbolic-like weights, a better merging of connectionist and symbolic systems is possible.

References

Achler T., 2011 Non-Oscillatory Dynamics to Disambiguate Pattern Mixtures, Chapter 4 in Relevance of the Time Domain to Neural Network Models, Eds: Rao R, Cecchi G A, Springer

Achler T., Bettencourt L., 2011 Evaluating the Contribution of Top-Down Feedback and Post-Learning Reconstruction, Biologically Inspired Cognitive Architectures AAAI Proceedings

Achler, T., Amir, E., 2008 Hybrid Classification and Symbolic-Like Manipulation Using Self-Regulatory Feedback Networks, NESY, Ceur-ws.

Achler, T., Vural D., Amir E., "Counting Objects with Biologically Inspired Regulatory-Feedback Networks." 2009 IEEE Proceedings on Neural Networks (IJCNN'09), 2009.

Anderson J., Silverstein J., Ritz S., Jones R. (1977). Distinctive features, categorical perception, and probability learning: Some applications of a neural model. Psych. Rev., 84, 413-451

Anderson J. R., Bothell D., Byrne M. D., Douglass S., Lebiere C., Qin Y. (2004) An integrated theory of the mind. Psychological Review, 1036–1060

Boden, M. (2006) A guide to recurrent neural networks & back-propagation www.itee.uq.edu.au/~mikael/papers/rn_dallas.pdf

Bogacz R., Giraud-Carrier C., 1998 BRAINN: A Connectionist Approach to Symbolic Reasoning. Proc ICSC/IFAC Symposium on Neural Computation (NC'98).

Bottou L, LeCun Y: Large Scale Online Learning, Advances in Neural Information Processing Systems (NIPS 2003)

Carpenter, G. A., Grossberg S. (1987) A Massively Parallel Architecture for a Self-Organizing Neural Pattern-Recognition Machine. Computer Vision Graphics and Image Processing 37(1): 54-115.

Franklin, S., Patterson, F. G. J. (2006). The LIDA Architecture: Adding New Modes of Learning to an Intelligent, Autonomous, Software Agent IDPT-2006

Hinton G. E., Salakhutdinov R. R. (2006) Reducing the dimensionality of data with neural networks. Science 313:5786 504-7

Hopfield, J. J. (1982) Neural networks and physical systems with emergent collective computational abilities PNAS v79.8 p2554–8

Hyvärinen A, J Karhunen, E. Oja (2001) Independent Component Analysis, New York: Wiley, ISBN 978-0-471-40540-5

Laird, J. E. 2008. Extending the Soar Cognitive Architecture. Artificial General Intelligence Conference, Memphis, TN.

McFadden F.E., Peng Y., Reggia J.A., Local Conditions for Phase-Transitions in Neural Networks with Variable Connection Strengths. Neural Networks, 1993. 6(5): p. 667-676.

Meyer D.E., Kieras D.E., (1997) A computational theory of executive cognitive processes and multiple-task performance: Part 1. Basic mechanisms. Psychological Review, 104(1), 3-65.

Olshausen, B A, Field D J. 1996. "Emergence of simple-cell receptive field properties by learning a sparse code for natural images." Nature 381: 607-609.

Rosenblatt, F. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological Review 65 6:386-408.

Rumelhart D. E, McClelland J. L. 1986. Parallel distributed processing: explorations in the microstructure of cognition. V1

Schmidhuber. J. (1992) Learning complex, extended sequences using the principle of history compression. Neural Computation, 4(2):234-242.

Shastri L. 2000 Types and Quantifiers in SHRUTI: A Connectionist Model of Rapid Reasoning and Relational Processing. In Wermter & Sun (eds.) Hybrid Neural Symbolic Integration Lecture Notes in Artificial Intelligence, pp. 28–45.

Sun, R. (2002). Duality of the Mind: A Bottom-up Approach Toward Cognition. Mahwah, NJ: Lawrence Erlbaum Associates.

Vapnik, V. N. 1995 The Nature of Statistical Learning Theory. New York: Springer Verlag

Williams R. J. Zipser D. (1994) Gradient-based learning algorithms for recurrent networks and their computational complexity. In Back-propagation:. Hillsdale, NJ: Erlbaum.

Zeiler M, Krishnan D, Taylor G, Fergus R, 2010 Deconvolutional Networks for Feature Learning CVPR