Winter 2020

Experimenting with a Biologically Plausible Neural Network

Dmitri Murphy
Portland State University

Experimenting with a Biologically Plausible Neural Network

Honors Undergraduate Thesis

Dmitri Murphy, Dr. Dan Hammerstrom (advisor)

Abstract

We present research on an implementation of a biologically inspired Bayesian Confidence Propagation Neural Network (BCPNN). Based on previous work by Christopher Johansson and Anders Lansner, our implementation seeks to test and understand the various properties of this model. The floating-point implementation we built uses discrete time and bit-vectors as input/output. We found that the column based BCPNN model is able to memorize a decent number of input vectors and is able to restore noisy versions of these vectors with relatively high accuracy. We examine the model's capacity, noise recovery ability and cross-column connection influence, among other attributes. The clearest trends that we found were in the capacity and noise recovery properties of the model, while the influence of cross-column connections was less clear. Further research and development of this model implementation is needed to increase speed, capacity and error correction capabilities.

1. Background

At present, there has been much research done on neural network designs that have been highly abstracted from how biological brains function, such as research on deep and convolutional neural networks. These designs function very well for certain tasks and are important to develop, but they fail to do some significant things that biological brains achieve. For example, one established way that biological brains learn is a principle called Hebbian learning, proposed by Donald Hebb in 1949 (Hebb, 1949). This learning theory states that when neurons are activated or fire at the same time, the connection between them is strengthened. However, many modern machine-learning algorithms instead focus on non-biological learning methods such as back-propagation. Even though these learning methods have been shown to be equivalent in some ways (Xie et. al, 2003), back-propagation may not be the best way to achieve advanced artificial intelligence that mimics human learning. Biological brains are much better suited to adaptive learning and pattern recognition, which makes studying and emulating them important for advancing the field long term.

One of the important differences between standard deep neural networks and the human brain is the training and learning cycle. The standard type of deep neural network is trained on potentially thousands of labeled examples for multiple epochs, as it builds up an internal model of a singular pattern or item (Santoro et al., 2016). This contrasts directly with the way that biological brains learn, which some researchers call on-line or one-shot learning. Humans, for example, don't have to be shown tens of thousands of cats to be able to identify a cat. This discrepancy is where some researchers think a return to biologically inspired computational models may be in order for significant progress in the field. There is research that has already been done on these

quicker kinds of learning and has been demonstrated to work in Spiking Neural Networks (Bugmann, 2012), as well as for extrapolating information from a limited set of training data (Kimura, 2018). However, some of this research doesn't take a biological approach, which is what we aim to do in this paper.

The second element of our research concerns the connection pattern of nodes in neural networks. The standard fully connected neural networks, with full layers of neurons, have significant memory and computational requirements (Lu et. al, 2018). With the end of Moore's Law looming, top people in the industry such as Naveen Rao (currently the vice president of Intel's AI division) have expressed concern that neural networks are quickly getting too large for what current hardware can support (Woodie, 2019). Hence, as networks and models become bigger and more complex, it will be important to understand where and how sparse connectivity can be used to improve them. Sparse connectivity patterns are also important as they are observed in the way that biological brains function and transfer information (Roy, 2019).

Finally, the third part of this research concerns how neural networks are structured (e.g. with cortical columns). Like sparse connectivity, cortical columns are a biological concept that comes from the cortical structures in biological brains (Figure 1). The idea of more biologically plausible networks having architectures that use cortical columns with more sparse connectivity between them has been raised by Christopher Johansson and Anders Lansner (Johansson and Lansner, 2007). More specifically, they discuss minicolumns and hypercolumns as the two types of cortical columns present in biological brains. In the minicolumns, neuron density is higher (especially near the center of the column). The hypercolumns are then made up of many minicoulumns. These types of architectures more closely follow the biological structure of mammal brains (Johansson and Lansner, 2007). In addition to possibly improving computational speed and power, implementing cortical structures found in the brain may help improve other properties of artificial neural networks. For example, the human brain has built-in redundancy, where other areas of the brain can take over tasks that an injured section used to do (Johansson and Lansner, 2007). In this way, modeling networks after cortical structures could improve redundancy for certain tasks, as well as build networks that can be more adaptive to new information.

Speaking more generally, many biological concepts can be used in further developing neural networks. For the cortical columns mentioned previously, Johansson and Lansner (2007) note that, for humans, these kinds of columns in our visual cortex can function as a winner-take-all construct for the neurons they contain. In other words, the winning (or most active) neuron sets the output value for the entire column of neurons. Similarly, patterns of inhibitory and excitatory connections between columns can be created to mimic the kinds input that these columns and neurons receive in biological brains. Figure 1 provides a good visualization of two columns, their internal structure, and the types of connections between them.

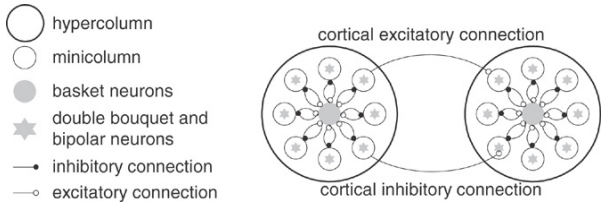

Figure 1: Two Hypercolumns (Johansson and Lansner, 2007)

2. Model

We will now describe the BCPNN model that we adapted from "Towards Cortex Sized Artificial Neural Systems" by Christopher Johansson and Anders Lansner (2007). The model uses several of the techniques that we have discussed previously, including neural columns, sparse connectivity between columns, and winner takes all (WTA) functionality. These are the techniques described by Johansson and Lansner (2007). However, we adapt their implementation to function in a more relaxed computing environment – allowing the use of floating-point numbers and floating-point computations in the model's calculations.

The following training algorithm was used:

1. Present an input vector to the model.
2. Model splits the input vector into equal sections and presents each section to a network column.
3. The network columns compute their initial activation and probability values given the input.
4. The network then computes the final activation values for each node, taking into account the cross-column connections and node bias values.
5. The WTA approach is applied to the output of each column, setting the node with the highest activation value as the winning node of the column.

2.1 Model Equations

The equations for our model are based on the ones described by Johansson and Lansner (2007). Some slight changes have been made for readability, such as the equations assuming that they are being performed at a timestep *t*. A glossary of terms is below, followed by the model equations.

Z	*Units per column*
P_i	*Current probability of node i*
V	*Input vector to the network*
V_k	*Bit k from the input vector to the network*
τ_p	*Memory plasticity control (constant)*
β	*Bias of a node*
W_{ij}	*Weight between nodes i and j*
O_k	*Activation value of node k*
C_{ij}	*Cross-column connection weight between nodes i and j*

Initial Values

$$P_i = \frac{1}{Z}$$

$$P_{ij} = \frac{1}{Z^2}$$

Node Probability Equation

$$\frac{\partial P_i}{\partial t} = \frac{V_i - P_i}{\tau_p}$$

All of the values above, such as P_i, are taken to be the values in the network at the current timestep/cycle. In this case, $\frac{\partial P_i}{\partial t}$ is the change in the probability, which is applied before the next cycle.

Node Co-Activation Probability Equation

$$\frac{\partial P_{ij}}{\partial t} = \frac{V_i V_j - P_{ij}}{\tau_p}$$

Node Bias Equation

$$\beta_i = log_{10}(P_i)$$

Weight Update Equation

$$w_{ij} = log_{10}(\frac{P_{ij}}{P_i P_j})$$

Node Activation Value Equation

$$O_j = \beta_j + \sum_i^N w_{ij} \cdot o_i$$

Cross-Column Connection Equation

Cross column connections were handled specially, with the connection weights (C_{ij}) being tuned based on if both of the connected nodes won their respective columns at the same timestep. The equations for these updates were:

$$C_{ji} = \begin{cases} 1.05C_{ji} & if\ O_j\ won\ and\ O_i\ won \\ 0.95C_{ji} & otherwise \end{cases}$$

2.2 Discrete vs. Continuous Time

One important change that we made in this model, when compared with the one described by Johansson and Lansner (2007), is the use of discrete instead of continuous time. This decision primarily served to reduce model complexity. However, to preserve the functionality of the model, we still simulate continuous time in the implementation. This is done using a set of "cycles" in each epoch for the input. Each input vector is run through the model for *n* cycles (per epoch) in order to simulate it being presented to a continuous model for some amount of time.

Given this adjustment, we have not needed to make large changes to the core model equations. However, we did make minor ones to approximate the derivative equations, as we were no longer in a continuous environment. To approximate the first order derivatives, we used finite difference equations. We do not think that this approximation impacts the model in a substantial way, as the model still converges. Further research could investigate whether using a better derivative approximation method would improve this model's performance.

2.3 Model Design and Layout

In the structural layout of this model, we arrange neurons in a set number of neural columns. Since the columns function as a winner-take-all environment designed to work as a unit, all of the nodes within these neural columns are fully connected. To provide some information transfer between columns, a set number of connections between columns are also randomly assigned. These connections connect various neurons in different columns, providing sparse cross-column connectivity.

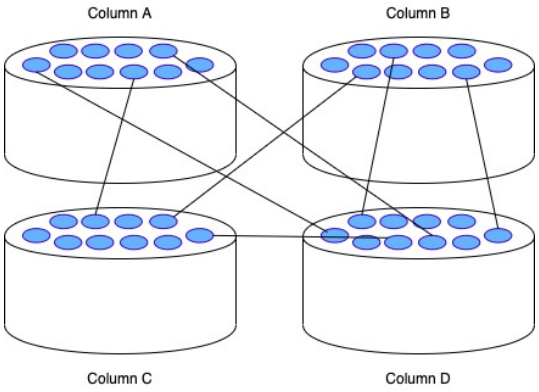

Figure 2: Cross-Column Connectivity

As the above visualization demonstrates, this network is essentially a slightly simplified version of the BCPNN that is presented by Johansson and Lansner (2007). In our model, the cross-column connections between the neurons are implemented differently than described by Johansson and Lansner, but still serve the same purpose of allowing information transfer between the different columns of the network.

3. Model Performance

Our implementation of this model had 100 columns with 10 neurons per column, and was created to work with bit vectors ($Vector\ V_i \in [0,1]$). This allows us to simplify some of the model properties, while still evaluating other important ones such as capacity and error correction. Vectors of 1000 bits were used for the tests described in this section. Each of these vectors had 100 sequences of 10 bits where a single "active" or 1 bit was set, while the rest of the bits were set to 0. These 10-bit sequences in the input were created to match the columnar layout of the network. This type of input, with a single active bit, was also reflective of the model's the winner take all approach in each column. An analysis of the Hamming distance of the training vectors showed most were relatively distinct from each other, with a majority having approximately 10 active bits in common.

To evaluate this model, we will examine some of its properties, including how many bit vectors it can fully retain, how well it can restore noisy versions of these vectors, and how the number of cross-column connections affects the model. The test vectors for this model were created by taking each of the training vectors and then flipping a certain number of bits randomly to add noise.

3.1 Network Capacity

We will start with our examination of how many bit-vectors the model could retain. For this test, all of the network properties were held constant, other than the number of vectors the network was trained on. The constant properties included: using 5 epochs per model run and 5 cycles per epoch, each column having 10 cross-column connections to other columns, and the test vectors having 5 bits of noise randomly added to them.

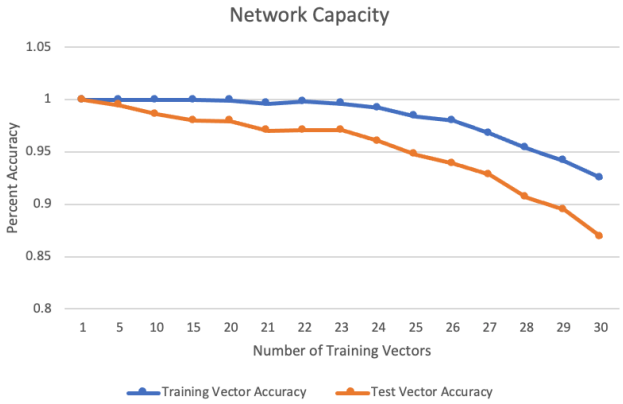

Figure 3: Network Capacity – holding cross-column connections, noise and epochs/cycles constant.

From the results of this test, we can see that the network does very well up until approximately 15 training vectors. Given previous research on Hopfield networks, which showed that their maximum capacity was 15%, this result makes sense in our columnar model (Takasaki, 2007). We have 100 columns in the model, indicating that this result is the 15% threshold that has been established previously. It is important to note that it therefore appears the number of columns, not the total number of neurons, determine network capacity.

Examining the results after 15 vectors, we see that performance degrades slightly up to 23 vectors and falls off more sharply afterwards. In terms of the test vector accuracy, the model maintained a test vector restoration accuracy level above 95% up until 24 vectors. We note that this number is an average, and that the network was able to fully restore some training vectors in testing, but struggled more with others.

The full capacity of the model is dependent on various factors, but given the results of this test, it could fully memorize a maximum of approximately 23-bit vectors (1000 bits in length), while retaining an average test

accuracy greater than 95%. This network capacity could be increased, but at the cost of worse noise recovery for the bit vectors. This makes sense from a theoretical standpoint, as adding more vectors to the network requires the network to store and recall additional information.

3.2 Noise Recovery

We will next examine the noise recovery properties of the network. In this context, noise was added to the test vectors by inverting a number of randomly selected bits in the vector. The amount of noise added refers to this number of randomly selected bits. In most cases, this meant additional active bits were set in the test vectors. However, there were also some cases where the active bit in a column flipped from being active to being inactive. For this test, the model was trained with 6 epochs and 5 cycles per epoch, each column had 10 cross-column connections to other columns, and 10 training input vectors. The amount of noise in the test vectors was varied for this test. As is common with BCPNNs, this model was relatively robust in terms of filtering out the additional noise being added to the bit vectors.

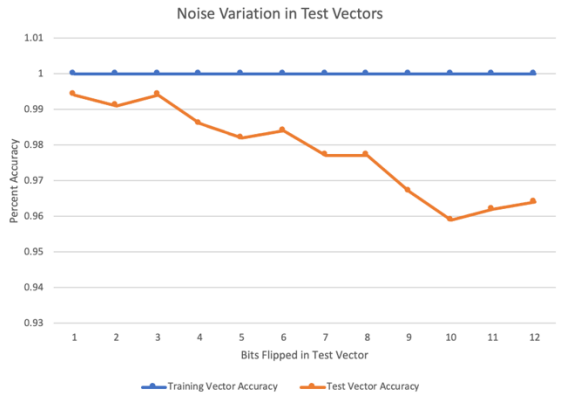

Figure 4: Test Vector Noise – holding cross-column connections, training vector number and epochs/cycles constant.

As could be expected, we can see a general downward trend in the ability of the model to recover the original vector as more noise is added. The model has a relatively good recovery rate, which stays above 96% with less than 10 vector bits flipped. However, we can see that overall there were several factors that impacted the ability of the network to recover the original vector when given a noisy one. From the capacity graph (Figure 3) and the noise graph (Figure 4), we can see that both of these factors had some impact on the ability of the network to recover the original vector. The number of training vectors especially seemed important. Given that, as mentioned previously, when the network is required to remember more vectors, it is harder for it to clearly restore an original vector from a noisy version.

3.3 Cross Column Connections

For examining the cross-column connections in this model, we followed the same approach as with the other tests. The model was trained for 5 epochs and 5 cycles per epoch, with 5 bits of noise were added to generate the test vectors, and 10 total training input vectors. The number of cross-column connections was varied for this test. The results of this examination are below.

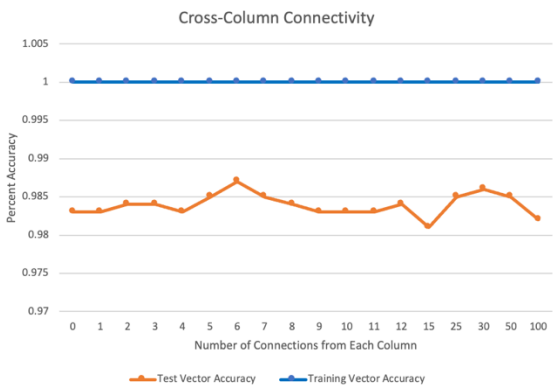

Figure 5: Cross-Column Connections – holding test vector number, noise and epochs/cycles constant.

From the graph of the results, we cannot see a clear pattern in how the number of cross column connections affects the network. We can see that the network test accuracy increases from 0 to 5 connections per column (500 total) and peaks at 6 connections per column (600 total). The network accuracy then drops down slightly and has minimal variation for the rest of the test. This is an unexpected result, as we would expect that the more connected the network is, the better its test accuracy would be.

3.4 Number of Epochs

Lastly, we examined how varying the number of epochs affected the model. The model was trained with 10 vectors, 5 bits of noise in each test vector, and 10 total training input vectors. The number of cycles per epoch was set to 5 cycles.

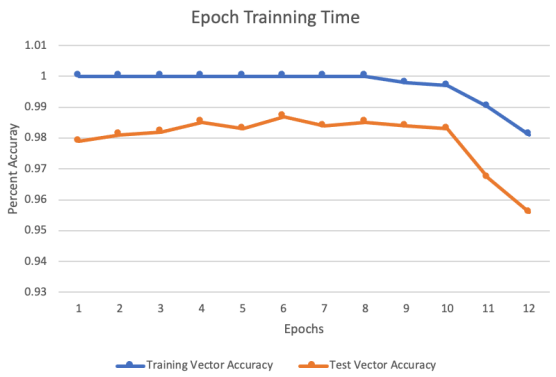

Figure 6: Epoch Impact on Training – all attributes other than epoch number were held constant.

As can be seen in the graph above, this network was able to achieve a preliminary training accuracy very quickly. In fact, it appears to do so almost instantly, achieving 100% accuracy on the 10 different bit-vectors used for training in the first epoch. However, the test vector accuracy takes several epochs to peak – achieving a maximum value around 6 epochs. Past 10 epochs, the network also experiences a rapid decline in both training and test vector accuracy. This is an unexplained effect and might indicate a problem with the model. More research is needed to determine the root cause of this issue and correct it.

3.5 Hyper-Parameter Tuning

In addition to testing the other parameters of the network, we also ran tests to verify the best hyper-parameters to use with the bit-vector tests that we were conducting. These included the multiplier for the bias value of the nodes and the various τ_p time constants.

Bias Results

From all of the various bias weights that we tested in this model, it appears that not having any bias multipliers is the best option. A multiplier of 1 faired the best in our testing. Small multipliers (below 1) decreased test accuracy performance but didn't have a major impact. Larger multipliers greater than 2 had a much larger impact on performance, degrading the number of vectors the network was able to fully retain.

Tau Constants

Examining the tau constants, we wanted to find values that allowed the network to be able to train at a reasonable pace. These adjustments were mainly done manually, so we do not have a relationship graph. However, we found that the following time constants worked the best.

$$Calculation\ Tau: 80$$
$$Cross-Column\ Tau: 250$$
$$Weight\ Update\ Tau: 1250$$

4. Summary

We have provided a general overview of a floating point, discrete time, bit-vector implementation of a Bayesian Confidence Propagation Neural Network based on previous work by Christopher Johansson and Anders Lansner. In general, we found the biologically inspired BCPNN design to be able to learn up to 24 bit-vectors and recover them with a 96% accuracy from a moderate amount of noise (5 bits).

We found clear trends in our examination of the capacity and noise recovery of the model but failed to determine good explanations for the results when varying the number of cross column connections and when running the model at higher epoch values. Future work may focus on addressing the unexpected behavior of the model when varying the number of cross-column connections and increasing the number of epochs.

It may also be beneficial for future work to make other improvements to the model or to integrate this model into another system for more practical applications. The applications of the model presented here mostly include being used as a component of a larger system, given its unsupervised style of Hebbian learning. This may involve developing this model into a spiking model or changing it to be heteroassociative. There are many avenues that are possible using this research as a starting point.

Code Repository

https://github.com/Dmitri-2/BCPNN-Sim-Python

References

Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, Anthony Maida. (2019). “Deep learning in spiking neural networks.” Neural Networks, Volume 111, Pages 47-63, ISSN 0893-6080.

Bugmann, Guido. (2012). One-shot pattern recognition learning in a deep multilayer neural network of biological inspiration. Perception. 41. 1273-1274.

Bullinaria, John. (2015). “Hebbian Learning and Gradient Descent Learning.” Neural Computation : Lecture 5, The University of Birmingham, https://www.cs.bham.ac.uk/~pxt/NC/l5_JB.pdf.

Fei Fei, Li. (2006). Knowledge Transfer in Learning to Recognize Visual Object Classes. University of Illinois Urbana-Champaign, http://vision.stanford.edu/documents/Fei-Fei_ICDL2006.pdf.

Hammerstrom, Dan. (2019). “The Cortical Graph (CorGraph) Project – An Investigation Into the Representation of Complex Contextual Relationships Based on the Structure of Cerebral Cortex – Year 2.” AFRL (Grant) Proposal. Portland State University.

Hebb, Donald Olding. (1949). *The Organization of Behavior: A Neuropsychological Theory*. John Wiley & Sons.

Hsu, Jeremy. (2015). “Estimate: Human Brain 30 Times Faster than Best Supercomputers.” *IEEE Spectrum: Technology, Engineering, and Science News*, IEEE, https://spectrum.ieee.org/tech-talk/computing/networks/estimate-human-brain-30-times-faster-than-best-supercomputers.

Johansson, Christopher and Lansner, Anders. (2007). Towards cortex sized artificial neural systems. Neural Networks. 20, 1 (January 2007), 48-61. DOI=http://dx.doi.org/10.1016/j.neunet.2006.05.029

Kimura, Akisato et al. (2018). “Few-shot learning of neural networks from scratch by pseudo example optimization.” *BMVC*.

Koch, G., Zemel, R. & Salakhutdinov, R. (2015). “Siamese Neural Networks for One-shot Image Recognition.” Department of Computer Science, University of Toronto.

Krotov, Dmitry. (2019). “Biologically Inspired Neural Networks: A Biologically Plausible Learning Algorithm for Neural Networks” MIT/IBM Watson AI Lab Guest Lecture. https://youtu.be/4lY-oAY0aQU

Kurzweil, Ray. (2000). *The Age of Spiritual Machines: When Computers Exceed Human Intelligence*. Penguin.

Lansner, Anders. (2009). "Associative memory models: from the cell-assembly theory to biophysically detailed cortex simulations." *Trends in Neurosciences* 32: 178-186.

Lecun, Yann & Bottou, Leon & Bengio, Y. & Haffner, Patrick. (1998). Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE. 86. 2278 - 2324. 10.1109/5.726791.

Lu, Y., Wang, C., Gong, L. et al. Int J Parallel Prog (2018) 46: 648. https://doi.org/10.1007/s10766-017-0528-8

Palm, Günther & Schwenker, Friedhelm & Sommer, Friedrich & Strey, Alfred. (1997). Neural Associative Memories. Department of Neural Information Processing. University of Ulm, Germany.

Pedro Domingos. (2012). A few useful things to know about machine learning. Commun. ACM 55, 10 (October 2012), 78-87. DOI: https://doi.org/10.1145/2347736.2347755

Raschka, Sebastian. (2015). *Single-Layer Neural Networks and Gradient Descent*. sebastianraschka.com/Articles/2015_singlelayer_neurons.html.

Rumelhart, D.E., Hinton,G. E.,Williams, R. J. (1986). Learning Representations By Back−Propagating Errors. Nature. Pgs. 323,533.

Roy, K., Jaiswal, A. & Panda, P. (2019). Towards spike-based machine intelligence with neuromorphic computing. Nature 575, 607–617 doi:10.1038/s41586-019-1677-2

Schuman, Catherine & Potok, Thomas & Patton, Robert & Birdwell, J. & Dean, Mark & Rose, Garrett & Plank, James. (2017). A Survey of Neuromorphic Computing and Neural Networks in Hardware. arXiv:1705.06963 [cs.NE].

Sremath Tirumala, Sreenivas & Ali, Shahid & Ramesh, Phani. (2016). Evolving Deep Neural Networks : A New Prospect. 10.1109/FSKD.2016.7603153

Sheridan, Patrick et al. (2017). "Sparse coding with memristor networks." *Nature nanotechnology* 12 8: 784-789.

Santoro, Adam, et al. (2016). "One-Shot Learning with Memory-Augmented Neural Networks." *ArXiv:1605.06065 [Cs]*.

Takasaki, Kevin. (2007). "Critical Capacity of Hopfield Network." Department of Physics, MIT.

Woodie, Alex. (2019). "Deep Learning Has Hit a Wall, Intel's Rao Says." *Datanami*, Accessed 13 Nov. 2019, https://www.datanami.com/2019/11/13/deep-learning-has-hit-a-wall-intels-rao-says/

Xie, Xiaohui, Seung, Hyunjune. (2003). Equivalence of Backpropagation and Contrastive Hebbian Learning in a Layered Network. Neural computation. 15. 441-54. 10.1162/089976603762552988.

Zhu, Shaojuan. (2008). "Associative Memory as a Bayesian Building Block," PhD Dissertation, OGI School of Science & Engineering at Oregon Health & Science University.