Reinforcement Learning Based on a Bayesian Confidence Propagating Neural Network

Christopher Johansson, Peter Raicevic and Anders Lansner

Department of Numerical Analysis and Computing Science,
Royal Institute of Technology, 100 44 Stockholm, Sweden
{cjo, peterr, ala}@nada.kth.se

We present a system capable of reinforcement learning (RL) based on the Bayesian confidence propagating neural network (BCPNN). The system is called BCPNNRL and its architecture is somewhat motivated by parallels to biology. We analyze the systems properties and we benchmark it against a simple Monte Carlo (MC) based RL algorithm, pursuit RL methods, and the Associative Reward Penalty (A_{R-P}) algorithm. The system is used to solve the n-armed bandit problem, pattern association, and path finding in a maze.

Reinforcement learning (RL) is an interesting learning paradigm since it is quite general and can utilize information that is readily available. RL can be applied to many different types of problems and no tailored made examples are needed as in supervised learning. In a typical RL problem there is an *agent* that acts in an *environment*. In this environment the agent is subjected to *rewards* and *punishments* depending on its *actions*. In most problems there is a delay i.e. the agent takes multiple actions, before the reward or punishment is given. This gives rise to a temporal credit assignment problem amongst the actions taken.

A subject that has received little attention is how an RL system can be implemented in a biologically plausible architecture, e.g. by neural networks (NNs). NNs have been used together with RL but they have then been used to solve isolated tasks.

Our aim here is to build a general RL system based on neuromorphic building blocks where the neural system makes up for all functionality. Such an RL module could serve as the connection between perception and action components in a controller for an autonomous learning agent. There have been attempts to do this e.g. many have tried to build neural controllers for robots [1, 2] that uses RL. However, little general knowledge about how an RL system can be implemented with NNs has been extracted.

The RL system we have developed is based on a Bayesian confidence propagating neural network (BCPNN). Previously, the BCPNN architecture has been thoroughly evaluated in perceptual and parallel, distributed associative memory tasks and it has been shown to scale favorably [3, 4]. We expect that these properties will also be useful in mastering the computational complexity of real world RL, for instance, by a consequent use of distributed state and action representations. Further, a preliminary study indicates that BCPNN can be used to model classical conditioning [5].

Here we study the convergence properties and performance of BCPNNRL (BCPNN Reinforcement Learning) in a couple of basic RL tasks. We also show how somewhat more complex problems can be handled including tasks where the reward structure changes. We compare the performance to that of MC based RL [6], Pursuit methods [6] and the A_{R-P} algorithm [7].

The operation of the BCPNNRL system has similarities to those of nonlinear learning automata [6, 8] and pursuit RL methods [6].

Methods

Bayesian Confidence Propagating Neural Network

The Bayesian confidence propagating neural network (BCPNN) algorithm is derived from Bayesian statistics [9]. The weights are computed according to Hebb's principle [10] of strengthening co-activated units. A *population* is defined as a set of units and a *projection* is a connection between two populations. A projection has a direction and it is defined by its weights and biases. There is one bias-term for each unit on the receiving side of the projection. A unit has several bias-terms if it receives several projections. A very important feature of the BCPNN is the partitioning of units into hypercolumns. In this paper we will mostly use populations with a single hypercolumn. More information about hypercolumns is found in the references on the BCPNN [9, 11, 12]. In the following equations N is used to denote the total number of units in a population and H (which mostly equals 1) the number of hypercolumns in a population. In a population with k hypercolumns, each hypercolumn will have U_k units, which gives $N=\sum_k U_k$.

For a biological motivation of the learning rule see Wahlgren and Lansner or Johansson and Lansner [5, 13]. The equations of the BCPNN are now given briefly. The first two equations, eq. (1) and (2), represents the synaptic potential in the pre- and postsynaptic synapses. The activity of the pre- and postsynaptic units are represented by the variables S_i and S_j.

$$\frac{dZ_i(t)}{dt} = \frac{S_i(t) - Z_i(t)}{\tau_{Zpre}} \tag{1}$$

$$\frac{dZ_j(t)}{dt} = \frac{S_j(t) - Z_j(t)}{\tau_{Zpost}} \tag{2}$$

The second set of equations, eq. (3)-(5), represent the synaptic trace in a connection between two units. The function of these three equations is to delay the storage of correlations as an eligibility trace.

$$\frac{dE_i(t)}{dt} = \frac{Z_i(t) - E_i(t)}{\tau_E} \tag{3}$$

$$\frac{dE_j(t)}{dt} = \frac{Z_j(t) - E_j(t)}{\tau_E} \tag{4}$$

$$\frac{dE_{ij}(t)}{dt} = \frac{Z_i(t)Z_j(t) - E_{ij}(t)}{\tau_E} \tag{5}$$

In the last three update equations, eq. (6)-(8), the P-variables constitute the memory of a connection and are thus intended to correspond to the long-term potentiation (LTP) in a synaptic coupling [14]. When the *printnow-signal* is activated (i.e. κ is greater than zero) the information stored in the state of the synaptic coupling, E-variables, is transferred into the memory (P-variables). The printnow-signal is assumed to correspond to the release of neuromodulator substances e.g. dopamine [15]. The κ is in the range of $[0,\infty[$ and a large κ induces a large change in the memory.

$$\frac{dP_i(t)}{dt} = \kappa\frac{E_i(t) - P_i(t)}{\tau_P} \tag{6}$$

$$\frac{dP_j(t)}{dt} = \kappa\frac{E_j(t) - P_j(t)}{\tau_P} \tag{7}$$

$$\frac{dP_{ij}(t)}{dt} = \kappa\frac{E_{ij}(t) - P_{ij}(t)}{\tau_P} \tag{8}$$

After each update of the P-variables the weights and biases are computed. The constant λ_0 is used to avoid the logarithm of zero.

$$\beta_i(t) = \log(P_i(t) + \lambda_0) \tag{9}$$

$$w_{ij}(t) = \frac{P_{ij}(t) + \lambda_0^2}{(P_i(t) + \lambda_0)(P_j(t) + \lambda_0)} \tag{10}$$

The potential h_i is computed by integrating over the support given from the incoming connections by eq. (11). We define a set $C(k)$ that consists of all units, j, in a hypercolumn k. The new activity, o_i, is computed by normalizing the support in each hypercolumn.

$$\frac{dh_i(t)}{dt} = \left(\beta_i(t) + \sum_k^H \log\left(\sum_{j \in C(k)} w_{ij}(t) o_j(t) \right) - h_i(t) \right) / \tau_C \quad (11)$$

for each hypercolumn *k*:

$$o_i(t) = \frac{e^{Gh_i(t)}}{\sum_{j \in C(k)}^N e^{Gh_j(t)}} \quad (12)$$

The differential equations were solved with Euler's method and the time-step, Δt, was set to 1.

The characteristic of the BCPNN is determined by the values of τ_{Zpre}, τ_{Zpost}, τ_E, τ_P, τ_C and *G*. The values of the four τ parameters are in the range of $[\Delta t,\infty[$ and the *G* parameter is in the range of $[0,\infty[$. The gain, *G*, is used to control the shape of the Gibbs distribution used to compute the output activity from a hypercolumn. Through out the paper $\tau_{Zpre}=\tau_{Zpost}=\Delta t$, $\tau_C=5\Delta t$ and $\lambda_0=0.0001$.

Initially all trace and memory variables, *Z*, *E*, *P*, are set to their bias values which for unit *i* in hypercolumn *k* and unit *j* in hypercolumn *l* is

$$bias_i = 1/U_k \quad (13)$$

$$bias_j = 1/U_l \quad (14)$$

and for unit *i* connected with unit *j* is:

$$bias_{ij} = 1/U_k U_l \quad (15)$$

The BCPNNRL System

The BCPNNRL system is presented in two steps. First we present the most basic system possible with only one projection to the action-population and then we extend the system to incorporate two projections to the action-population and add a value-projection and value-population to the system.

The most basic BCPNNRL system is designed with two populations of units. One is used to represent the current state of the agent, the state-population. The other, the action-population, is used to represent the actions that the agent can perform. Between these two populations is a feed-forward projection from the state-population to the action-population. Figure 1 shows the layout of the described system. This basic system is used to solve the n-armed bandit problem.

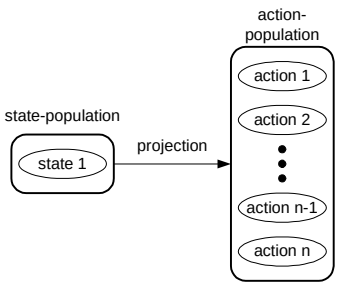

Figure 1 The most basic BCPNNRL system that is used in the n-Armed Bandit task. The projection between the two populations contains n connections.

Only positive rewards can be handled by the most basic BCPNNRL system. The strength of the κ is set equal to the reward. A positive κ induces a strengthening in the connection between the active state and action.

Both the state- and action-population are represented with only one hypercolumn in all experiments except those for the pattern association task. Each unit in the state-population corresponds to a single state and each unit in the action-population corresponds to a single, elementary, action. Many biologically inspired models concerned with navigation use a similar state representation [16] and this form of state representation is thought to exist e.g. in hippocampal place cells [14]. As mentioned earlier the activity in a single hypercolumn sum to one and in the state- and action-populations it is interpreted as a probability density functions (pdf) over the set of possible states or actions. One action is selected using this pdf.

The more elaborate BCPNNRL system handles both positive and negative rewards and it also has the ability to act on changes in the level of reward. The system incorporates two projections (Figure 2), one positive and one negative projection. When the system gets a positive reward the positive projection is updated and the negative projection is decayed and vice versa if the system gets a negative reward.

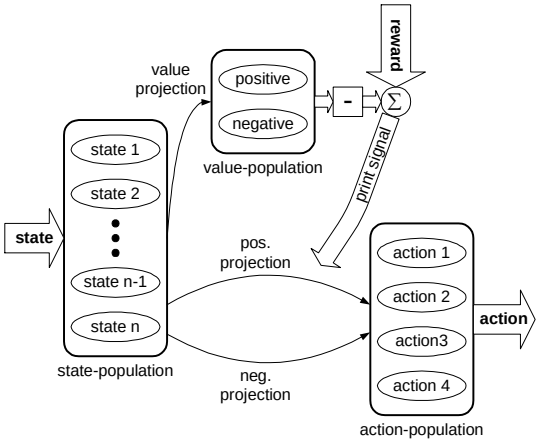

Figure 2 The complete BCPNNRL system as implemented to solve the grid world problem. There are two projections from the state-population to the action-population. If the system generates a positive κ the pos. projection is strengthened and vice versa for a negative κ. The value-population maintains the reward average that is used to compute the κ.

The two projections are combined in the action-population. The neg. projection is inverted simply by negating the support values, *h*. The result is that the input from the negated projection inhibits the units in the action-population.

A comparison-based [6, 17] κ is used in the second system; the κ is computed based on a comparison of the current reward and the average of previous rewards. The value-population is only activated when a reward is present and it does not form a value-function but rather a reward-function[1].

The BCPNNRL systems policy function is expressed in the activity of the action-population. The activity in the action-population is controlled by the softmax parameter *G*. The parameter *G* can regulate the exploitation-exploration balance and it may thus be thought of as a noradrenerigc neuromodulator [18, 19].

[1] Instead of assigning a value to each state, a reward-function only stores values in those states where an absolute reward have been received.

The parameter τ_P is used to regulate the learning rate. The learning rate and the balance between exploration-exploitation can in the BCPNNRL system be viewed as the same thing. Both *G* and τ_P affect the learning rate of the system and can be used to regulate it, but τ_P is preferably used.

The parameter τ_E controls the length of the eligibility trace. If τ_E=1 there is no trace and the system cannot handle delayed rewards i.e. go through a series of states before getting the reward.

Experiments and Results

We chose to benchmark the BCPNNRL system on three tasks; the n-armed bandit problem, pattern association and path finding in a maze. We compare the BCPNNRL system with several RL algorithms; Monte Carlo based RL (MC), Pursuit RL, and Associative Reward-Penalty (A_{R-P}). The MC uses full backups and both the MC and Pursuit RL uses a softmax function for action selection.

The n-armed bandit problem is the classical problem of selecting the action with the highest return. In the context of RL we can formulate this problem as having one state in which we can perform n different actions and the goal is to find the action that yields the highest return under the constraint that we want to maximize our total reward.

A 2-armed bandit is used in an experiment where the actions a_1 and a_2 receives reward r_1=1.0 and r_2=0.5 (Figure 3). We set τ_p = 100, a rather large value, to get a smoother curve in the figure. After 800 epochs the reward changes to r_1=0.5 and r_2=1.0. The BCPNN system first converged towards selecting action a_1 with probability 1.0. After the reward structure changes at epoch 800, the system converged towards action a_2.

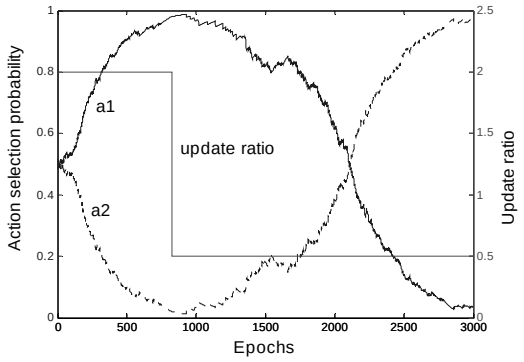

Figure 3 The most basic BCPNNRL system with one input unit and one output hypercolumn with two units is used to solve a 2-armed bandit task. The reward for action a_1 and a_2 is r_1=1.0 and r_2=0.5 and the system converged towards action a_1. At epoch 800 the rewards were changed to r_1=0.5 and r_2=1.0. The system responded to the new reward structure and converged towards action a_2. The ratio described in eq. (18) is experimentally calculated and displayed with its x-axis on the right side of the graph. Up to epoch 800 the ratio is close to 2 = r_1 / r_2, and then close to 0.5 = r_1 / r_2. BCPNNRL parameters: τ_E=1, τ_P=100, G=1.

We investigated the convergence properties of the most basic BCPNNRL on the 2-armed bandit problem using a system with one state and two output units. The actions a_1 and a_2 are selected with probability o_1 and o_2 according to eq. (12) and are generating positive reward r_1 and r_2 respectively. Since o_1+o_2=1, we looked only on changes in o_1. With only one input unit we have that P_{ij}=P_j giving the rate of change of probability o_1 as,

$$o'_{1,increase} = \frac{do_1}{dt} = r_1 f(P_{j=1}, P_{j=2})$$
$$o'_{1,decrease} = \frac{do_1}{dt} = -r_2 f(P_{j=1}, P_{j=2}) \quad (16)$$

where $o'_{1,\ increase}$ is the increase rate in o_1 from selecting action a_1 and $o'_{1,\ decrease}$ is the decrease rate in o_1 from selecting action a_2.

By multiplying eq. (16) with the probability to select the actions we get the expected rate of change of probability as,

$$E(o'_{1,increase}) = o'_{1,increase} o_1$$
$$E(o'_{1,decrease}) = o'_{1,decrease} o_2 \quad (17)$$

This means that a system selecting actions repeatedly, where each update is so small that the probabilities o_i remain the same, will receive a total change in probability o_1 proportional to $E(o'_{1,increase})$+ $E(o'_{1,decrease})$. Calculating the ratio between the expected increase and decrease for the action a_1 gives,

$$\left| \frac{E(o'_{1,increase})}{E(o'_{1,decrease})} \right| = \left| \frac{o'_{1,increase} o_1}{o'_{1,decrease} o_2} \right| = \frac{r_1}{r_2} \quad (18)$$

The result shows that with small time steps and with r_1>r_2 the probability of selecting action a_1 will increase on average leading to a convergence towards selecting action a_1 with probability 1.0 (and action a_2 with probability 0.0). If r_1<r_2 the system will converge towards selecting action a_1 with probability 0.0 (and action a_2 with probability 1.0).

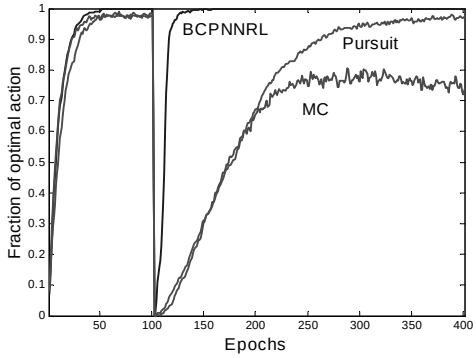

Figure 4 The MC alg. and BCPNNRL system solved the 10-armed bandit problem. One of the 10 arms (actions) gave a reward of 1 while the other actions gave a reward of 0.1. At 100 epochs the action that gave reward 1 was altered. The task was to find the arm that returned the highest reward. The results were averaged over 500 runs. MC parameters: γ=1, α=0.3, G=6; Pursuit parameters: α=0.3, G=6; BCPNNRL parameters: *value projection*: τ_E=1, τ_P=200, G=1.12, *action projection*: τ_E=1, τ_P=4, G=3.

The experiment with a 10-armed bandit tested the ability of the BCPNNRL, MC and Pursuit

algorithms to learn an optimal behavior and then relearn a new optimal behavior (Figure 4). At 100 epochs the action that gave the highest reward was altered.

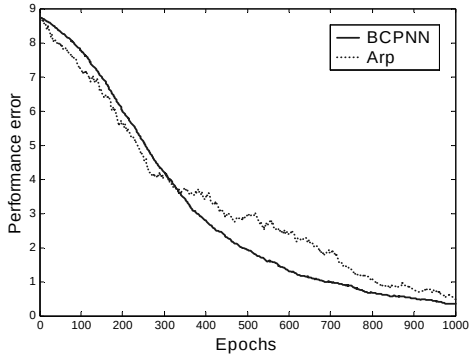

Figure 5 Performance of an extended version of the most basic BCPNNRL system on the pattern association task. Each of the five inputs are presented to the system 1000 times and the performance error is an average over 10 runs and is compared with the Associative Reward-Penalty algorithm. BCPNNRL parameters: τ_E=1, τ_p=200, G=1.

The pattern association task was to match a single state with a complex output pattern. The reward was increased, as the match was getting better.

In the pattern association task we extend the most basic BCPNNRL architecture to consist of one input hypercolumn with five units and seven output hypercolumns with two units each. The system is trained on a pattern association task where it has to learn a target pattern for each of the five inputs. The reward is calculated as one minus the Hamming distance between the output and the target pattern divided by the number of output units, resulting in a reward of 1.0 for a completely correct output and 0.0 for an output where all hypercolumns give the wrong answer. The performance error is calculated as the sum of the squared difference between the target pattern for unit *i* and the probability o_i, for all five target patterns. We compare the result for the BCPNN with τ_p = 200 with the $A_{R\text{-}P}$ algorithm using the best parameters found: ρ=0.15 and λ=0.01. The convergence speed for this task was similar for BCPNN and $A_{R\text{-}P}$ (Figure 5).

The Gridworld was a simple maze. Each grid in the maze corresponded to a state and in each state four different actions were available; go north, east, south or west. Each performed action was a step. We used two different tasks; the first task in the Gridworld was to find a single reward. When the reward was found a new epoch was initiated and the agent was returned to its starting state. The agent was allowed to make moves into a wall or into the out-of-bounds area around the maze. These moves did not affect the position of the agent but they could affect the reward. The second task was a T-maze where the left side reward was initially 1 and the right side reward – 1 or 0.01. After 50 epochs the positive and negative rewards switched side, after another 50 epochs they switched side again and so fourth. The agent had to follow the positive reward as it moved from side to side in the T-maze and this could only be done by the dual projection BCPNNRL system.

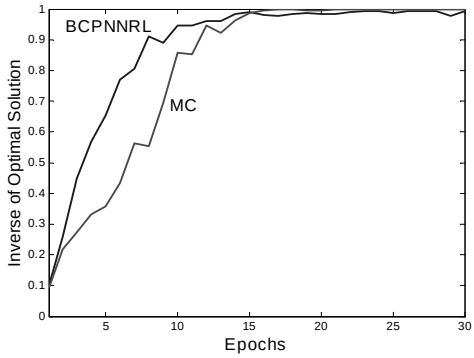

Figure 6 The path-finding problem in a 4x4 maze. This task requires that the agents can handle delayed rewards. The result was averaged over 100 runs. MC parameters: γ=0.99, α=0.3, G=20 BCPNNRL parameters: *value projection*: τ_E=1, τ_P=200, G=1.12, *action projection*: τ_E=2, τ_p=4, G=3.

The first experiment in the Gridworld was the task of finding a single reward in the maze (Figure 6). The BCPNNRL agent had the same parameters as in the 10-armed bandit task but one of the parameters of the MC algorithm was

altered; G=20. The effect of setting G=20 in the 10-armed bandit task would be that the MC algorithm would have difficulties to relearn.

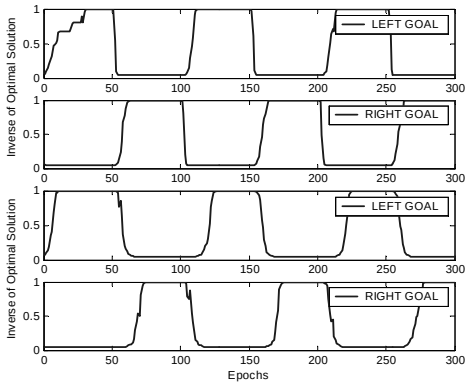

Figure 7 A T-maze task in the Gridworld. The upper two plots show the result when the reward shifted between 1,-1. The lower two plots show the result when the reward shifted from 1 to 0.01. The results were averaged over 100 runs. BCPNNRL parameters; *value projection*: τ_E=1, τ_P=10, G=1, *action projection*: τ_E=2, τ_P=2, G=3.

The second experiment in the Gridworld was done with a T-maze (Figure 7). Two setups were tried; in the first case the rewards were 1 and –1, in the second case the rewards were 1 and 0.01. The BCPNNRL was able to follow the highest reward as it changed from side-to-side in the T-maze.

Discussion

The BCPNN learning rule was not designed to handle RL tasks but we have shown that the BCPNNRL is indeed capable of doing this. An important feature of this system is that no external memory or computations are needed. All processing of information is done within the network of BCPNN units. To the extent that the BCPNN is considered to be biologically realistic the BCPNNRL is a biologically plausible structure.

We have evaluated the BCPNNRL systems on tasks such as the n-armed bandit problem, pattern association and path finding in a maze. Currently the capabilities of the BCPNNRL system in the general case is on par with those of a simple MC based RL or A_{R-P} algorithms.

When a value-population is added to the BCPNNRL system it becomes more sensitive to parameter settings but the system also becomes more powerful. With the value-population the BCPNNRL system can react on the lack of reward, i.e. be frustrated.

The BCPNNRL system is, as all types of sample return based RL systems, best suited for small problems. When the problem is large it is very advantageous to use RL methods that are based on temporal difference (TD) learning. The present BCPNNRL system cannot implement TD-learning. In order to implement this we need to be able to extract state-values and compute differences in the state-values between states. This problem needs to be addressed in future versions of the BCPNNRL system.

Instead of using only one hypercolumn it is possible to have several hypercolumns in either the state- or action-population. The coding then becomes distributed and this can increase the noise tolerance and improve scaling properties of the system. Distributed coding may also simplify the pre-processing of input data (used to represent the states) since the input can be mapped directly from the sensors into the BCPNN and the BCPNNRL system. Further, the populations can have one or several feedback or recurrent projections. This means that e.g. the states may be represented as attractor memories. It is also conceivable idea to have high order action hierarchies instead of only elementary actions. We probed these possibilities only slightly and this is an area for future studies.

Acknowledgement
Support for this work from the Swedish Research Council for Engineering Sciences (TFR dnr 2000-143) and the Swedish research council (VR dnr 621-2001-2548) is gratefully acknowledged.

References

1. Sporns, O. and W.H. Alexander, *Neuromodulation and plasticity in an autonomous robot.* Neural Networks, 2002. **15**(4-6): p. 761-774.
2. Gadanho, S.C. and J. Hallam, *Robot Learning Driven by Emotions.* Adaptive Behavior, 2001. **9**(1): p. 42-64.
3. Johansson, C., *A Capacity Study of a Bayesian Neural Network with Hypercolumns*. 2001, Nada, SANS: Stockholm.
4. Johansson, C., *A Parallel Implementation of a Bayesian Neural Network with Hypercolumns*. 2001, Nada, SANS: Stockholm.
5. Johansson, C. and A. Lansner, *An Associative Neural Network Model of Classical Conditioning*. 2002, SANS: Stockholm.
6. Sutton, R.S. and A.G. Barto, *Reinforcement Learning: An Introduction*. 1998: MIT Press.
7. Barto, A. and M. Jordan. *Gradient following without back-propagation in layered networks.* in *Proceedings of the IEEE First Annual Conference on Neural Networks*. 1987: SOS Printing.
8. Unsal, C., *Intelligent Navigation of Autonomous Vehicles in an Automated Highway System: Learning Methods and Interacting Vehicles Approach*, in *Electrical Engineering*. 1998, Virginia Polytechnic Institute and State University: Blacksburg, Virginia.
9. Holst, A., *The Use of a Bayesian Neural Network Model for Classification Tasks*, in *Dept. of Numerical Analysis and Computing Science*. 1997, Kungl. Tekniska Högskolan, Stockholm.
10. Hebb, D.O., *The Organization of Behavior*. 1949, New York: John Wiley Inc.
11. Johansson, C., A. Sandberg, and A. Lansner. *A Neural Network with Hypercolumns*. in *ICANN*. 2002. Spain, Madrid: Springer-Verlag Berlin.
12. Sandberg, A., et al., *A Bayesian attractor network with incremental learning*. Network: Computation in Neural Systems, 2002. **13**(2): p. 179-194.
13. Wahlgren, N. and A. Lansner, *Biological evaluation of a Hebbian-Bayesian learning rule.* Neurocomputing, 2001. **38-40**: p. 433-438.
14. Kandel, E.R., J.H. Schwartz, and T.M. Jessell, *The Anatomical Organization of the CNS, Coding of Sensory Information*, in *Principles of Neural Science*. 2000, McGraw-Hill Companies. p. 318-336, 411-429.
15. Reynolds, J.N.J. and J.R. Wickens, *Dopamine-dependent plasticity of corticostriatal synapses.* Neural Networks, 2002. **15**(4-6): p. 507-521.
16. Foster, D.J., R.G.M. Morris, and P. Dayan, *Models of Hippocampally Dependent Navigation, Using The Temporal Difference Learning Rule.* Hippocampus, 2000. **10**: p. 1-16.
17. Barto, A.G., R.S. Sutton, and C.W. Anderson, *Neuronlike adaptive elements that can solve difficult learning control problems.* IEEE Transactions on Systems, Man and Cybernetics, 1983. **SMC-13**: p. 834-846.
18. Usher, M., et al., *The Role of Locus Coeruleus in the Regulation of Cognitive Performance.* Science, 1999. **283**: p. 549-554.
19. Doya, K., *Metalearning and neuromodulation.* Neural Networks, 2002. **15**(4-6): p. 495-506.