ARTICLE **Communicated by Laurenz Wiskott**

Reward-Modulated Hebbian Learning of Decision Making

Michael Pfeiffer
pfeiffer@igi.tugraz.at
Bernhard Nessler
nessler@igi.tugraz.at
Institute for Theoretical Computer Science, Graz University of Technology, A-8010 Graz, Austria

Rodney J. Douglas
rjd@ini.phys.ethz.ch
Institute of Neuroinformatics, University of Zürich and ETH Zürich, CH-8057 Zürich, Switzerland

Wolfgang Maass
maass@igi.tugraz.at
Institute for Theoretical Computer Science, Graz University of Technology, A-8010 Graz, Austria

We introduce a framework for decision making in which the learning of decision making is reduced to its simplest and biologically most plausible form: Hebbian learning on a linear neuron. We cast our Bayesian-Hebb learning rule as reinforcement learning in which certain decisions are rewarded and prove that each synaptic weight will on average converge exponentially fast to the log-odd of receiving a reward when its pre- and postsynaptic neurons are active. In our simple architecture, a particular action is selected from the set of candidate actions by a winner-take-all operation. The global reward assigned to this action then modulates the update of each synapse. Apart from this global reward signal, our reward-modulated Bayesian Hebb rule is a pure Hebb update that depends only on the coactivation of the pre- and postsynaptic neurons, not on the weighted sum of all presynaptic inputs to the postsynaptic neuron as in the perceptron learning rule or the Rescorla-Wagner rule. This simple approach to action-selection learning requires that information about sensory inputs be presented to the Bayesian decision stage in a suitably preprocessed form resulting from other adaptive processes (acting on a larger timescale) that detect salient dependencies among input features. Hence our proposed framework for fast learning of decisions also provides interesting new hypotheses regarding neural nodes and computational goals of cortical areas that provide input to the final decision stage.

1 Introduction

A typical decision-making task of an organism requires the evaluation of multiple alternative actions, with the goal of maximizing the probability of obtaining positive reward. If input signals provide only uncertain cues and reward is obtained stochastically in response to actions, then Bayesian statistics provides a mathematical framework for the optimal integration of all available information. Bayes' theorem can be used to calculate the probability that an action yields a reward, given the current sensory input and the current internal state of an organism. The goal of this article is to present the simplest possible neural network model that can make such an evaluation, where simplicity is assessed in terms of both computational operations and the complexity of the learning method.

A large number of experimental results suggest that animals do indeed make decisions based on Bayesian integration of information about stimulus-action-reward contingencies. For example, Sugrue, Corrado, and Newsome (2004) have shown that monkeys use the matching behavior strategy, in which the frequency with which a particular action is chosen matches the expected reward for that action. Yang and Shadlen (2007) have shown that the previous experience of macaque monkeys in probabilistic decision tasks is represented by the firing rates of neurons in area LIP in the form of the log-likelihood ratio (or log-odd) of receiving a reward for a particular action a in response to a stimulus $\mathbf{x}$ (in an experiment where the monkey received in each trial either no reward, or a reward of unit size, depending on the choice of the monkey among two possible actions).

We show that an optimal action selection policy can be reduced to a winner-take-all (WTA) operation applied to linear gates, which receive suitably preprocessed inputs (see Figure 1). Furthermore, we show that the updating of the WTA circuit in the face of new evidence can be reduced to the application of a local reward-modulated Hebbian learning rule to each linear gate. We call this rule the Bayesian Hebb rule. Despite the simplicity of this model, one can prove that it enables fast learning of near-optimal decision making, which is remarkable because rigorous insight into convergence properties of Hebbian learning rules is often lacking.

WTA (see Yuille & Geiger, 2003, for a review) is a very simple computational operation that selects the largest among l values $L_1, \ldots, L_l$. This selection is usually encoded through l binary outputs $z_1, \ldots, z_l$, where $z_a = 1$ if L_a is selected as the largest input (ties can be broken arbitrarily), else $z_a = 0$ (see Figure 1). In an action selection framework, this output then triggers the selection of the ath among l possible actions. Each value L_a is just a weighted sum

$$L_a = \sum_{i=0}^{n} w_{a,i}\, y_i$$

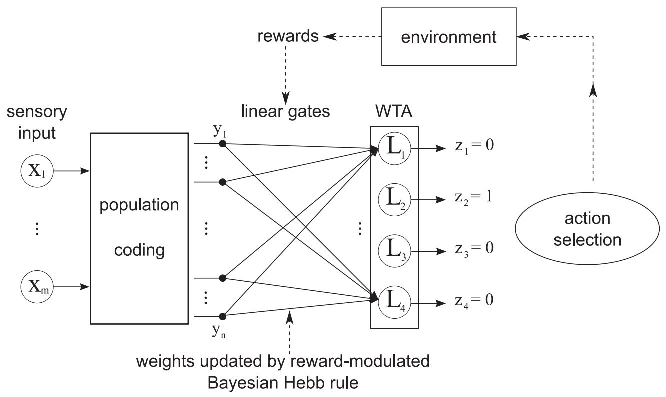

Figure 1: Winner-take-all (WTA) architecture for learning of decision making. First, the multinomial input variables $x_1, \ldots, x_m$ are preprocessed by a fixed circuit (which implements some type of population coding) to yield binary variables $y_1, \ldots, y_n$. For every possible action a, there is an associated linear neuron L_a that computes a weighted sum $\sum_{i=0}^{n} w_{a,i} y_i$ of the variables $y_1, \ldots, y_n$. The neuron L_a with the largest weighted sum "wins," that is, $z_a = 1$, and action a is selected.

of variables $y_1, \ldots, y_n$ (and a dummy variable $y_0 \equiv 1$ that allows using $w_{a,0}$ as a bias). Despite its simplicity, the resulting WTA circuit is computationally quite powerful (Maass, 2000).

The main contribution of this article is a novel learning algorithm for the weights $w^*_{a,i}$ of the linear gates L_a. We show that for a suitable fixed preprocessing (that transfers the original input variables x_k into binary variables y_i), the optimal value $w^*_{a,i}$ for the weight $w_{a,i}$ in Figure 1 is the log-likelihood ratio (or log-odd) of receiving a reward for a particular action a, provided that the binary feature y_i is activated by the preprocessing function:

$$w^*_{a,i} = \log \frac{p(r = 1 \mid y_i = 1, a)}{p(r = 0 \mid y_i = 1, a)}. \tag{1.1}$$

In the asymptotic case, where all weights $w_{a,i}$ have converged to their respective target values $w^*_{a_i}$, the policy of the WTA circuit in Figure 1 is optimal in the sense that for any input signal, the action with the highest chance to deliver reward is chosen. We also show that after finitely many training trial steps, the weights closely approximate the optimal weights that can be inferred from the previously observed data.

Our algorithm for reward-modulated learning of optimal weights uses only Hebbian learning, a form of learning for which there is strong experimental evidence (Abbott & Nelson, 2000; Frégnac, 2003; Caporale &

Dan, 2008). Hebb (1949) proposed (see Frégnac, 2003, for a recent review) that a synapse from neuron A to neuron B is strengthened if A and B often fire together. But several studies have shown that Hebbian synaptic plasticity requires a third signal (often in the form of neuromodulators) in order to consolidate weight changes (Bailey, Giustetto, Huang, Hawkins, & Kandel, 2000; Reynolds, Hyland, & Wickens, 2001; Farries & Fairhall, 2007; Legenstein, Pecevski, & Maass, 2008). It is often assumed that the third signal provides information about reward or reward expectations. Hence learning rules involving these signals are referred to as reward-modulated learning rules.

Hebbian learning, such as in the proposed Bayesian Hebb rule, should be contrasted with non-Hebbian learning rules such as the perceptron learning rule (also referred to as delta rule) or the Rescorla-Wagner rule (Rescorla & Wagner, 1972), which are harder to support on the basis of experimental data for synaptic plasticity. In these latter learning rules, the change Δw_i of a synaptic weight w_i at a single synapse depends not only on the current activation values of the pre- and postsynaptic neuron and the current value of w_i (and possibly a reward-related third signal), but also on the current values of the other weights and the activation values of all other neurons that provide synaptic input to the same postsynaptic neuron (more precisely: on the value of the weighted sum of all presynaptic inputs).

We present a mechanism for reward-modulated local learning of the weights $w_{a,i}$ that permits them to converge (on average) to the ideal value, equation 1.1. Learning from rewards is conceptually different from learning with a supervisor who informs the learner about the correct choice. In reward-based learning, the learner must explore different actions multiple times even if he assumes that other actions would be better in the given situation. This strategy is necessary to avoid premature convergence to suboptimal policies.

We want to make clear that in this article, we do not study the learning of sequences of actions as in general reinforcement learning (Sutton & Barto, 1998), but investigate scenarios like those in operant conditioning, where decisions have to be made based on learned immediate reward probabilities for single actions. We follow the terminology proposed, for example, in Dayan and Abbott (2001) and subsume the latter also under the term reinforcement learning.

We provide in this article a rigorous theoretical analysis of the convergence properties of the Bayesian Hebb rule. Because our learning rule makes online updates after every training trial, rather than performing a batch update after collecting a set of data, we are interested in the asymptotic behavior of the rule, as well as its online performance. Non-Hebbian learning rules usually perform gradient descent optimization along an error surface. If local minima exist on the error surface, this approach always carries the risk of becoming trapped in suboptimal solutions, from which it cannot escape. In contrast, the optimal values of the weights to be learned by the Bayesian Hebb rule act as global fixed-point attractors in weight-space with regard to expected weight updates of the Bayesian Hebb rule. Our analysis shows that the weights learned during training are very close to the optimal values that can be inferred from finitely many training trials, and they converge exponentially fast to the optimal values. We also demonstrate that an extremely simple linear approximation to the Bayesian Hebb rule performs almost equally well.

Bayesian decision making combines information from many variables and therefore must consider statistical dependencies among them. An influential paper by Roth (1999) noted that decision making can be reduced to the computation of weighted sums, provided that the input signals are properly preprocessed (see also Domingos & Pazzani, 1997). This observation motivates our use of the neural network model shown in Figure 1. Roth (1999) proved his results in the context of linear statistical queries for probabilistic classification. We now extend this approach to the case of policy learning by incorporating a WTA gate for action selection. Roth (1999) noted that the set of features produced by the preprocessing function must be related to independence assumptions among input variables. We show that these features correspond to the factors in a factor graph (Kschischang, Frey, & Loeliger, 2001) of the input and reward distribution.

One particularly simple case is naive Bayes, which assumes that all input variables are conditionally independent given one particular target variable, for example, the occurrence of reward. In this case, it is sufficient to know the reward-prediction probabilities for every input variable and every action separately, since then the reward probability given the complete input is the product of all individual predictors. We provide a simple preprocessing function for this case, which does not use any information about statistical dependencies of input variables but leads to satisfactory policies.

The general case, in which there are statistical dependencies among input variables, requires more complex algorithms for Bayesian inference. Graphical models like Bayesian networks (Bishop, 2006) and factor graphs (Kschischang et al., 2001) are used to model conditional dependencies among variables, and inference algorithms operate by passing messages along edges of the graphs. Factor graphs are particularly useful tools. They consider groups of dependent variables as factor nodes, in which functions of all connected variable nodes are computed. Inference in these models is performed using the sum-product algorithm (Bishop, 2006; Kschischang et al., 2001), which is conceptually simpler than the belief propagation algorithms used for inference in general Bayesian networks. Recent work (Steimer, Maass, & Douglas, 2009) has shown that these factor nodes can be implemented in networks of spiking neurons. In this article, we define an optimal generalized preprocessing function based on the factor graph representation of the reward distribution. This provides a concrete processing goal for multimodal integration in sensory areas and links the theory of factor graphs to experimentally observed neural population codes. These codes, like all other components of our framework, are easily implemented in neural networks and allow fast and robust learning with the Hebbian learning algorithms presented in this article.

We assume here that the graph structure of the underlying Bayesian network is known, but not the parameters of it (i.e., the probability distribution). We do not address the problem of structure learning, which is a very different task and thus requires different algorithms. Whereas the parameters that define decision strategies require very fast adaptation, statistical dependencies between inputs reflect invariances in the environment, which could be learned by separate learning processes on much longer timescales.

This article is organized as follows: We present the Bayesian Hebb rule for reinforcement learning tasks in section 2 and analyze its convergence behavior for learning reward log-odds. In section 3 we present a linear approximation to the Bayesian Hebb rule that is much simpler to implement but exhibits similar convergence behavior. In section 4, we show that after a suitable preprocessing of sensory variables $\mathbf{x}$, one arrives at a population code $\mathbf{y}$ for which optimal decisions can be represented by WTA applied to weighted sums of the variables y_i. The required weights can be learned quite fast with the Bayesian Hebb rule, even if there exist conditional dependencies among the input variables $\mathbf{x}$. Section 5 gives experimental results on the performance of the Bayesian Hebb rule in various action selection tasks. Section 5.2 addresses the case of nonstationary reward distributions. In section 6 the learning rule is generalized to handle tasks in environments with continuous input signals $\mathbf{x}$. We discuss in section 7 salient aspects of the presented results, an application of the Bayesian Hebb rule to model the experimental data of Yang and Shadlen (2007), related work, and open problems.

2 The Bayesian Hebb Rule

In this section we introduce a simple local learning rule, the reward-modulated Bayesian Hebb rule, which learns log-odds of reward probabilities conditioned on binary input variables. Analyzing the convergence behavior of the rule, one sees that the true reward log-odds are fixed-point attractors for expected weight changes under the reward-modulated Bayesian Hebb rule. The Bayesian Hebb rule also learns fast, since the online learned weights are close to what an optimal Bayesian learning approach, using (biologically unrealistic) counters and auxiliary variables, would achieve. It is further shown that an even simpler rule, which approximates the Bayesian Hebb rule, learns weights that are close to the optimum, and is sufficient for reliable decision making.

2.1 Action Selection Strategies and Goals for Learning.

We consider the standard operant conditioning scenario, where the learner receives at each trial an input $\mathbf{x} = \langle x_1, \ldots, x_m \rangle$ (e.g., a sensory stimulus or internal state signals of the organism) with multinomial variables x_j, chooses an action a out of a set of l possible actions $A = \{a_1, \ldots, a_l\}$, and receives a reward $r \in \{0, 1\}$ with probability $p(r \mid \mathbf{x}, a)$. The learner's goal is to learn (as fast as possible) a policy $\pi(\mathbf{x}, a) = p(a \mid \mathbf{x})$ (or $\pi(\mathbf{x})$ in the case of a deterministic policy) so that action selection according to this policy maximizes the average reward. A structural difference to supervised prediction problems is that it does not suffice that the learner passively observes the outcomes of trials, since the reward received for action a in response to stimulus $\mathbf{x}$ provides no information about the probability of rewards for alternative actions a' in response to the same stimulus $\mathbf{x}$. He therefore needs to try out different actions for the same input through an exploration process in order to learn the reward probabilities for all actions.

In this article, the goal of the learner is fast learning of a policy that approximates the optimal policy. The learner does not necessarily maximize the online performance during learning and does not specifically try to reduce uncertainty about the outcome of unexplored action. The strategies employed during learning are therefore not Bayes optimal in the sense of decision theory and sequential analysis (Dayan & Daw, 2008). Optimal solutions to the exploration problem for a restricted subclass of tasks can be computed (Gittins, 1979; Lai & Robbins, 1985; Auer, Cesa-Bianchi, & Fischer, 2002), but neural network implementations of these mechanisms are beyond the scope of this article. During learning we follow heuristic strategies that are commonly used in reinforcement learning (Sutton & Barto, 1998). The actions are chosen based on the currently learned weights, which approximate the Bayes optimal estimates for the reward log-odds. In order to maintain a rather high level of rewards during exploration, the agent might, for example, choose actions stochastically with $p(a \mid \mathbf{x}) = p(r{=}1 \mid \mathbf{x}, a)$. This corresponds to the matching behavior phenomenon observed in biology, where the fraction of choices for one action exactly matches the fraction of total rewards from that action (Sugrue et al., 2004). This policy was used during training in all our computer experiments.

If the goal of the agent is to accumulate as many rewards as possible and rewards are binary, the agent will choose the action with the highest probability $p(r = 1 \mid \mathbf{x}, a)$ to yield reward. Since the function that maps a probability p onto $\log \frac{p}{1-p}$ is strictly monotonically increasing, the agent can choose instead the action a that has the highest log-odd:

$$\log \frac{p(r = 1 \mid \mathbf{x}, a)}{p(r = 0 \mid \mathbf{x}, a)}. \tag{2.1}$$

Hence the optimal policy for maximizing the probability of reward can be written in the form

$$\pi(\mathbf{x}) = \arg\max_{a \in A} \; \log \frac{p(r = 1 \mid \mathbf{x}, a)}{p(r = 0 \mid \mathbf{x}, a)}. \tag{2.2}$$

We assume for now that the input $\mathbf{x} = \langle x_1, \ldots, x_m \rangle$ consists of m input variables that are arbitrary multinomial discrete random variables with unknown joint distribution (in section 6, we consider the case of continuous inputs $\mathbf{x}$). We assume that these m variables are represented through binary states (firing or nonfiring) $\mathbf{y} = \langle y_1, \ldots, y_n \rangle$ of n neurons in a population coding manner. We define the encoding scheme later in section 4 and show that different encodings allow different representations of statistical dependencies. For every possible action a, there exists in our simple model (see Figure 1) a linear neuron that receives as inputs the components $y_1, \ldots, y_n$ of $\mathbf{y}$. The activation L_a of this linear neuron is defined by the weighted sum

$$L_a = w_{a,0} + \sum_{i=1}^{n} w_{a,i}\; y_i. \tag{2.3}$$

Our approach aims at learning weights $w_{a,i}$ for every action a such that L_a corresponds to the reward log-odd (see equation 2.1), which indicates how desirable it is to execute action a in the current situation defined by $\mathbf{x}$ and its neural encoding $\mathbf{y}$. The action with the highest assumed probability of yielding reward is then selected by a winner-take-all (WTA) operation that is formally defined through the binary outputs $z_1, \ldots, z_l$ as follows:

$$z_a = \begin{cases} 1, & \text{if } L_a \geq L_b \quad \text{for } b \neq a \\ 0, & \text{else} \end{cases}. \tag{2.4}$$

This action selection strategy is commonly referred to as the greedy strategy.

If the goal is not only to exploit preceding experience in order to choose an action that maximizes the probability of reward for the current stimulus $\mathbf{x}$, but to simultaneously keep on learning and exploring reward probabilities for other actions, the matching behavior strategy (Sugrue et al., 2005) offers an attractive compromise. It can be implemented with the help of the learned parameters $w_{a,i}$ in the following way: the linear gate L_a in Figure 1 is replaced by a sigmoidal gate (i.e., the weighted sum L_a according to equation 2.3 is replaced by $\sigma(L_a) = \frac{1}{1+\exp(-L_a)}$, and the deterministic WTA gate is replaced by a stochastic soft WTA gate (which selects a as winner with probability $\frac{\sigma(L_a)}{\sum_b \sigma(L_b)}$).

2.2 A Local Rule for Learning Reward Log-Odds. We now present a learning rule and an appropriate input encoding for learning weights, which asymptotically approach target values such that the architecture in Figure 1 selects actions optimally. Consider first the case where for a single binary input y_i and action a, the reward log-odd $\log \frac{p(r=1|y_i=1,a)}{p(r=0|y_i=1,a)}$ should be learned in the weight $w_{a,i}$. A traditional frequentist's approach would use counter variables,

$$\alpha_{a,i} = \#[r = 1 \wedge y_i = 1 \wedge \text{action } a \text{ selected}],$$
$$\beta_{a,i} = \#[r = 0 \wedge y_i = 1 \wedge \text{action } a \text{ selected}],$$

to estimate the reward log-odds $w^*_{a,i}$ after finitely many steps by

$$\hat{w}_{a,i} = \log \frac{\alpha_{a,i}}{\beta_{a,i}} \quad \text{for } i = 1, \ldots, n.$$

In a rewarded trial (i.e., $r = 1$) where $y_i = 1$ and action a is selected, this leads to the update

$$\begin{aligned} \hat{w}^{new}_{a,i} &= \log \frac{\alpha_{a,i} + 1}{\beta_{a,i}} = \log \frac{\alpha_{a,i}}{\beta_{a,i}} \left(1 + \frac{1}{\alpha_{a,i}}\right) \\ &= \hat{w}_{a,i} + \log \left(1 + \frac{1}{N_{a,i}}(1 + e^{-\hat{w}_{a,i}})\right), \end{aligned} \tag{2.5}$$

where $N_{a,i} := \alpha_{a,i} + \beta_{a,i}$ is the total number of previous updates; thus, $\frac{1}{\alpha_{a,i}} = \frac{1}{N_{a,i}}(1 + \frac{\beta_{a,i}}{\alpha_{a,i}})$.

Analogously, an update after a new unrewarded trial ($r = 0$) gives rise to the update

$$\hat{w}^{new}_{a,i} = \hat{w}_{a,i} - \log \left(1 + \frac{1}{N_{a,i}}(1 + e^{\hat{w}_{a,i}})\right). \tag{2.6}$$

Using the approximation $\log(1 + x) \approx x$ and a constant learning rate η instead of the factor $\frac{1}{N_{a,i}}$, update rules 2.5 and 2.6 can be combined to yield a new local learning rule, which does not use any counters.[1] We call this rule the reward-modulated Bayesian Hebb rule. The update for weight $w_{a,i}$, whenever action a is selected and $y_i = 1$ is:

$$\Delta w_{a,i} = \begin{cases} \eta \cdot (1 + e^{-w_{a,i}}), & \text{if } r = 1 \\ -\eta \cdot (1 + e^{w_{a,i}}), & \text{if } r = 0 \end{cases}. \tag{2.7}$$

This rule increases the weight whenever reward is encountered and decreases the strength of the synapse otherwise. Learning rule 2.7 is purely local; it depends only on quantities that are available at the trained synapse, not on the activity of other presynaptic neurons.

[1]Using the approximation $\log(1 + x) \approx x$ did not visibly affect the performance of the learning rule in the computer simulations in section 5.

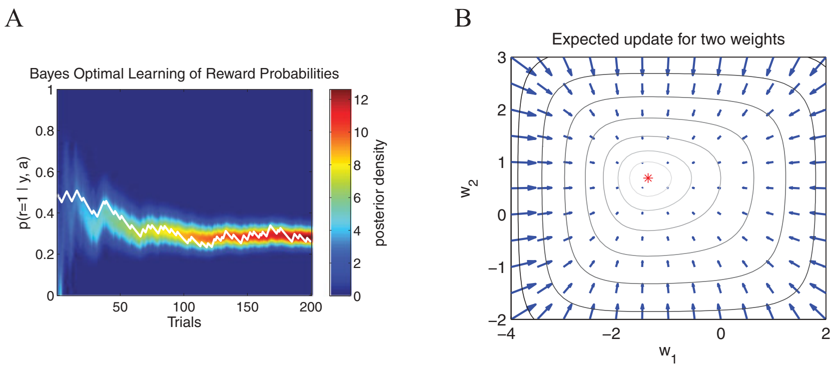

Figure 2: Convergence behavior of the Bayesian Hebb rule. (A) The weights learned by the Bayesian Hebb rule approximate Bayes-optimal learning. The posterior for the reward probability $q_{a,i} = p(r = 1 \mid y_i = 1, a)$ at every training trial was modeled by a beta$(\alpha_{a,i} + 1, \beta_{a,i} + 1)$ distribution, with counters $\alpha_{a,i}$ and $\beta_{a,i}$ for rewarded and unrewarded trials. The color shows the estimated posterior density function for $q_{a,i}$ at every training trial. The white curve shows the approximation learned by the Bayesian Hebb rule, 2.7 (with constant learning rate $\eta = 0.02$). The weight $w_{a,i}$ was transformed into an estimated reward probability by $\hat{q}_{a,i} = \frac{1}{1+\exp(-w_{a,i})}$. One can see that the approximation follows the optimal estimate closely. (B) Attractor property of the Bayesian Hebb rule, equation 2.7, plotted for two weights w_1 and w_2. The expected update (indicated by a blue arrow) is always in the direction of the optimal weights (marked by a red star). Gray curves connect points with the same amount of expected weight change.

The approximation of the reward-modulated Bayesian Hebb rule to the exact counting model, which computes for every parameter the Bayes-optimal estimate that can be inferred from a fixed finite set of data, is illustrated in Figure 2A. In order to estimate a single parameter $q_{a,i} = p(r = 1 \mid y_i = 1, a)$, a uniform prior on $[0, 1]$ was initially imposed on $q_{a,i}$. The counters $\alpha_{a,i}$ and $\beta_{a,i}$, as defined above, were incremented as training samples became available, and the posterior distribution for $q_{a,i}$ was given by the beta$(\alpha_{a,i} + 1, \beta_{a,i} + 1)$ distribution (Neapolitan, 2004). The same samples were simultaneously used to update the weight $w_{a,i}$ by rule 2.7. The weights $w_{a,i}$, which represent log-odds $\log \frac{p(r=1|y_i=1,a)}{p(r=0|y_i=1,a)}$, were transformed into probabilities by the transformation

$$\hat{q}_{a,i} = \frac{1}{1 + \exp(-w_{a,i})}.$$

Figure 2A shows the optimal posterior for a single $q_{a,i}$ after every update and the approximation obtained by equation 2.7. The probability estimated by the Bayesian Hebb rule is always close to the Bayes-optimal estimate.

2.3 Convergence Properties of the Bayesian Hebb Rule in Reinforcement Learning. The Bayesian Hebb rule is an online learning rule that has no prior knowledge of its target values. However, one can prove that the weights learned with equation 2.7 converge (in expectation) to their optimal values $w^*_{a,i} = \log \frac{p(r=1|y_i=1,a)}{p(r=0|y_i=1,a)}$, on the basis of just the statistics of pre- and postsynaptic values they encounter. This is in fact very easy to prove, since the equilibrium of the rule is reached when the expected update $E[\Delta w_{a,i}]$ under rule 2.7 vanishes, and this can be written as

$$\begin{aligned} E[\Delta w_{a,i}] = 0 \Leftrightarrow p(r = 1 \mid y_i = 1, a) \cdot \eta \cdot (1 + e^{-w_{a,i}}) - \\ -p(r = 0 \mid y_i = 1, a) \cdot \eta \cdot (1 + e^{w_{a,i}}) = 0. \end{aligned}$$

As we show in appendix A, the latter explicitly holds iff $w_{a,i}$ is at the target value $w^*_{a,i} = \log \frac{p(r=1|y_i=1,a)}{p(r=0|y_i=1,a)}$. If a vector of $n+1$ weights $\langle w_{a,0}, \ldots, w_{a,n}\rangle$ for an action a is learned simultaneously, the point $\langle w^*_{a,0}, \ldots, w^*_{a,n}\rangle$ is a global fixed-point attractor in the weight space $\mathbb{R}^{n+1}$ with regard to expected weight changes under the Bayesian Hebb rule (see Figure 2B).

Another unusual feature of the Bayesian Hebb rule is that one can prove (see appendix A) that it converges exponentially fast to $w^*_{a,i}$ (with regard to $E[\Delta w_{a,i}]$). In particular, weight updates move the weight in larger steps toward the attractor $w^*_{a,i}$ if they are farther off, without requiring any change of the learning rate, or knowledge of the ideal values $w^*_{a,i}$.

3 The Linear Bayesian Hebb Rule

The reward-modulated Bayesian Hebb rule, equation 2.7, includes exponential terms $\exp(-w_{a,i})$ and $\exp(w_{a,i})$. One may argue that an exact calculation of the exponential function is beyond the capabilities of a synaptic learning process. Therefore, we have also analyzed a linear approximation to the Bayesian Hebb rule. The exponential function is defined by the Taylor series:

$$\exp(x) = \sum_{i=0}^{\infty} \frac{x^i}{i!}. \tag{3.1}$$

Thus, the first-order approximations for $\exp(w_{a,i})$ and $\exp(-w_{a,i})$ are

$$\exp(w) \approx 1 + w \tag{3.2}$$

$$\exp(-w) \approx 1 - w. \tag{3.3}$$

By inserting the approximations 3.2 and 3.3 into 2.7, we obtain a computationally simpler learning rule, which we call the linear Bayesian Hebb rule.

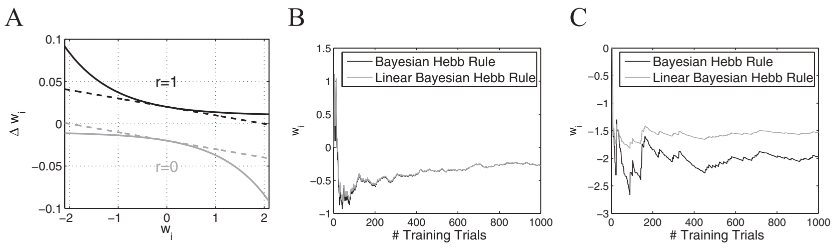

Figure 3: Linear approximation of the Bayesian Hebb rule. (A) Update Δw_i of the Bayesian Hebb rule, equation 2.7 (solid lines), and the linear Bayesian Hebb rule, equation 3.4 (dashed lines), plotted as a function of the current weight value w_i for training trials with $r = 1$ (black curves) and $r = 0$ (gray curves). (B) Example of the evolution of a single weight under the Bayesian Hebb rule, equation 2.7, and the linear Bayesian Hebb rule, equation 3.4. The target value is close to 0, where the approximation of the linear Bayesian Hebb rule is very good. (C) Another example of the weight evolution, in which the two rules converge to different weights. The target weight is close to -2, which is the border of the weight range that the linear Bayesian Hebb rule can cover. The approximation error is therefore large compared to B.

Whenever action a is selected and $y_i = 1$, it updates weight $w_{a,i}$ by

$$\Delta w_{a,i} = \begin{cases} \eta \cdot (2 - w_{a,i}), & \text{if } r = 1 \\ -\eta \cdot (2 + w_{a,i}), & \text{if } r = 0 \end{cases}. \tag{3.4}$$

This new rule resembles strongly the typical Hebb rule with a regularization term. The weights are increased by a constant if the pre- and postsynaptic neurons "fire together" (i.e., $y_i = 1$ and action a is selected), and decreased by a constant if they do not. The $\pm w_{a,i}$ term prevents the weights from growing too large or too small. Actually, for $\eta \leq 1$, it always keeps the weights within the range $[-2, 2]$. This shows immediately that the linear Bayesian Hebb rule cannot learn the true reward log-odds for arbitrary distributions, only an approximation. Figure 3A shows the updates by the linear Bayesian Hebb rule (dashed lines) in comparison to those of the exact rule, equation 2.7 (solid lines). One can see that the difference between the updates grows for larger values of the target weight $w^*_{a,i}$. However, our computer experiments in section 5 will demonstrate that the linear Bayesian Hebb rule performs remarkably well for many benchmark tasks.

3.1 Convergence of the Linear Bayesian Hebb Rule. We show in appendix B that the equilibrium value for the linear Bayesian Hebb rule, equation 3.4 (the weight value where $E[\Delta w_{a,i}] = 0$), is at

$$\begin{aligned} w^{+}_{a,i} &= -2 + 4 \cdot p(r = 1 \mid y_i = 1, a) \\ &= 2 \cdot (p(r = 1 \mid y_i = 1, a) - p(r = 0 \mid y_i = 1, a)). \end{aligned}$$

This equilibrium value is monotonically increasing with $w^{*}_{a,i}$, the equilibrium value of the exact Bayesian Hebb rule, equation 2.7. They are equal only when $p(r = 1 \mid y_i = 1, a) = p(r = 0 \mid y_i = 1, a)$: $w^{*}_{a,i} = w^{+}_{a,i} = 0$.

In Figures 3B and 3C the evolution of two weights during learning for a random distribution is shown. In Figure 3B, the target value is close to zero, where the target values for the exact rule, equation 2.7, and the linear Bayesian Hebb rule, equation 3.4, are very similar. Thus, no big difference in weight space is visible. In Figure 3C, however, the target value is close to the maximum value that the linear rule can represent; therefore, the two rules do not converge to the same value, indicating a larger approximation error for the linear rule. Hence, the linear Bayesian Hebb rule can be expected to perform well if the target values of the weights have small absolute values.

4 Population Codes for Hebbian Learning of Asymptotically Optimal Decisions

In this section, two preprocessing mechanisms are presented, which are based on different assumptions about statistical dependencies among input variables. Applied to these population encodings of the input, the WTA circuit in Figure 1 selects actions that maximize the probability of obtaining reward, according to the current statistical model represented by the input encoding and the reward log-odds learned with the Bayesian Hebb rule.

We have previously shown that the reward-modulated Bayesian Hebb rule, equation 2.7, has a unique equilibrium at the reward log-odd,

$$w^{*}_{a,i} = \log \frac{p(r = 1 \mid y_i = 1, a)}{p(r = 0 \mid y_i = 1, a)}. \tag{4.1}$$

In order to approximate the true reward probabilities for every action as weighted sums as in equation 2.3, every vector of input variables $\mathbf{x} = \langle x_1, \ldots, x_m \rangle$ needs to be suitably preprocessed into a population code vector $\mathbf{y} = \langle y_1, \ldots, y_n \rangle$. If the weights $w_{a,i}$ for every y_i and every action a are learned with the Bayesian Hebb rule, our previous analysis guarantees that the resulting policy will asymptotically approach the best policy that can be inferred for the given preprocessing function.

Let the input variables $x_1, \ldots, x_m$ be some arbitrary multinomial random variables with unknown joint distribution, where each variable x_k assumes m_k different values $v^k_1, \ldots, v^k_{m_k}$. For simplicity, we assume that $v^k_j = j$ for $j = 1, \ldots, m_k$ and $k = 1, \ldots, m$.

We first present a very simple population coding, which is sufficient to represent the optimal policy as a weighted sum if the naive Bayes assumption holds for the input variables; that is, the input variables x_k are conditionally independent of each other given the selected action a and the reward r:

$$\begin{aligned} &p(x_k \mid r, a, x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_m) \\ &\quad = p(x_k \mid r, a) \quad \text{for all } k \in \{1, \ldots, m\}. \end{aligned} \tag{4.2}$$

In this case, it holds that

$$\frac{p(r = 1 \mid \mathbf{x}, a)}{p(r = 0 \mid \mathbf{x}, a)} = \frac{p(r = 1 \mid a)}{p(r = 0 \mid a)} \prod_{k=1}^{m} \frac{p(x_k \mid r = 1, a)}{p(x_k \mid r = 0, a)}. \tag{4.3}$$

Every x_k is discrete and can take on only finitely many different values. Applying Bayes' theorem and using an indicator function I, which is defined as $I(\texttt{true}) = 1$ and $I(\texttt{false}) = 0$, one can rewrite equation 4.3 as (see appendix C for the full derivation)

$$\begin{aligned} \frac{p(r = 1 \mid \mathbf{x}, a)}{p(r = 0 \mid \mathbf{x}, a)} &= \frac{p(r = 1 \mid a)}{p(r = 0 \mid a)} \\ &\times \prod_{k=1}^{m} \left(\frac{p(r = 0 \mid a)}{p(r = 1 \mid a)} \prod_{j=1}^{m_k} \left(\frac{p(r = 1 \mid x_k = j, a)}{p(r = 0 \mid x_k = j, a)} \right)^{I(x_k = j)} \right). \end{aligned} \tag{4.4}$$

This suggests representing every x_k by a population code, which has $m_k + 1$ binary variables, one for every possible value of x_k, and one bias variable to account for the term $\frac{p(r=0|a)}{p(r=1|a)}$. Formally we define the simple preprocessing (SP) $\boldsymbol{\phi}(x_k)$ for a single variable x_k as

$$\boldsymbol{\phi}(x_k) = \left[-1, \varphi_1, \ldots, \varphi_{m_k}\right]^T, \quad \text{where } \varphi_j = \begin{cases} 1, & \text{if } x_k = j \\ 0, & \text{otherwise} \end{cases}. \tag{4.5}$$

As an example, we consider the simple reward distribution with two input variables $\mathbf{x} = \langle x_1, x_2 \rangle$, modeled by the Bayesian network in Figure 4A. Under the naive Bayes assumption, the dependency of x_2 on the input variable x_1 is neglected; that is, the arrow $x_1 \rightarrow x_2$ in the Bayesian network is ignored. For binary x_k, the population code under this assumption is illustrated in Figure 4C. Each input variable x_k is encoded separately by three variables y_i, where one is constantly -1, and only one other y_i is active, depending on the value of x_k.

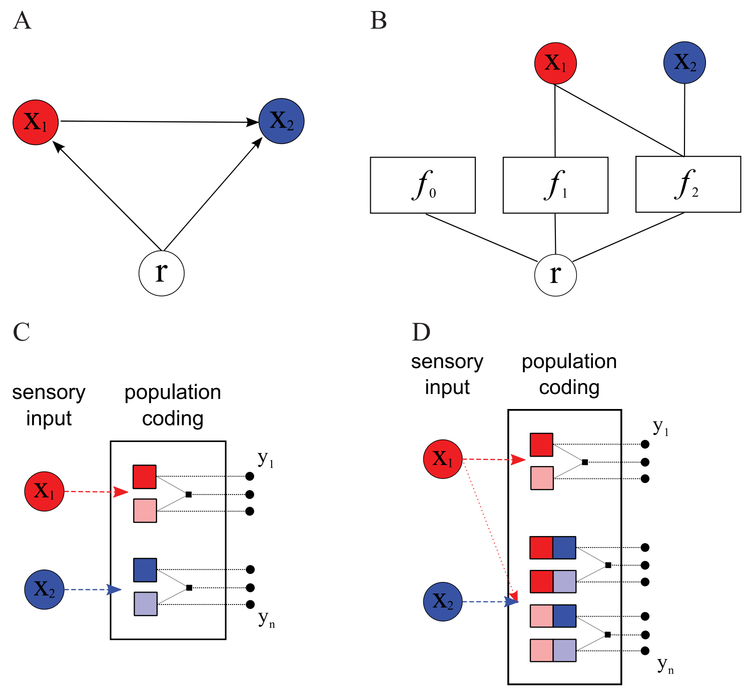

Figure 4: Preprocessing for tasks with arbitrary statistical dependencies. (A) An example Bayesian network for the joint distribution of sensory inputs $\mathbf{x} = \langle x_1, x_2 \rangle$ and reward r. (B) Factor graph representation for the prediction of r, according to the Bayesian network in *A*. Here, f_0 represents the prior $p(r)$, and the factors f_1 and f_2 represent the conditional probabilities $p(x_1 \mid r)$ and $p(x_2 \mid x_1, r)$, respectively. (C) Population coding under the naive Bayes assumption, which we refer to as simple preprocessing (SP). For every possible value of the variables x_k (here x_1, x_2 are binary), there is one variable y_i (indicated by a black circle) that outputs the value 1. Additionally there is one variable y_i for every x_k, which is constantly at -1 (black square). The constant bias term y_0 is not shown. (D) Population coding applied to the factors in the factor graph shown in *B*. For each combination of values of the variables $\{x_k, \mathbf{x}_{\mathrm{P}_k}\}$ of a factor, there is exactly one variable y_i (indicated by a black circle) associated with the factor that outputs the value 1. Other variables y_i represent ORs of these values (black squares) and yield either 0 or -1. The constant bias term y_0 is not shown. We refer to the resulting preprocessing circuit that maps sensory inputs $\mathbf{x}$ onto internal variables $\mathbf{y}$ that support Hebbian learning of optimal decisions as generalized preprocessing (GP).

The vectors $\boldsymbol{\phi}(x_k)$ for $k = 1, \ldots, m$ are concatenated into one population code vector $\mathbf{y}$ for the whole input. $\mathbf{y}$ has $n = 1 + m + \sum_{k=1}^{m} m_k$ entries, of which exactly $2 \cdot m + 1$ are nonzero, and the first entry $y_0 \equiv 1$ corresponds to the bias term $\frac{p(r=1|a)}{p(r=0|a)}$ in equation 4.4:

$$\mathbf{y} = \Phi(\mathbf{x}) = \begin{bmatrix} 1 \\ \boldsymbol{\phi}(x_1) \\ \boldsymbol{\phi}(x_2) \\ \vdots \\ \boldsymbol{\phi}(x_m) \end{bmatrix}. \tag{4.6}$$

Substituting the definition of $\mathbf{y}$ from equations 4.5 and 4.6 into 4.4 and taking the logarithm then yields the log-odd function

$$\log \frac{p(r=1 \mid \mathbf{y}, a)}{p(r=0 \mid \mathbf{y}, a)} = \log \frac{p(r=1|a)}{p(r=0|a)} + \sum_{i=1}^{n} y_i \log \frac{p(r=1 \mid y_i \neq 0, a)}{p(r=0 \mid y_i \neq 0, a)}. \tag{4.7}$$

If we use the population code 4.6 for $\mathbf{y}$, we can apply the reward-modulated Bayesian Hebb rule 2.7 for every y_i to learn reward log-odds conditioned on feature y_i being active.[2] For a y_i that is constantly active, such as y_0, the weight $w_{a,i}$ will converge to the prior reward probability $\log \frac{p(r=1|a)}{p(r=0|a)}$ for action a. Inserting the target values, equation 4.1, of the weights into equation 4.7, we can therefore write

$$\log \frac{p(r=1 \mid \mathbf{y}, a)}{p(r=0 \mid \mathbf{y}, a)} = \sum_{i=0}^{n} w^*_{a,i} y_i. \tag{4.8}$$

During learning, the current values of the weights $w_{a,0}, \ldots, w_{a,n}$ are used to approximate the true reward log-odd for every action a as the weighted sums in equation 2.3. Actions are selected by a heuristic method according to their predicted probability of yielding reward (e.g., greedy or matching behavior). If the naive Bayes assumption holds, the reward-modulated Bayesian Hebb rule, in combination with a simple population coding for every input variable x_k, is therefore sufficient to asymptotically learn the optimal action selection policy.

4.1 Learning Decisions for Arbitrary Discrete Distributions. We now address the more general case, where conditional independence of the input variables $x_1, \ldots, x_m$ cannot be assumed. We show that with a fixed preprocessing of the input that takes their dependencies into account, the

[2]We consider a feature y_i active if it is nonzero, that is, both $y_i = 1$ and $y_i = -1$ are active features.

Bayesian Hebb rule enables the resulting neural network to converge quite fast to the best performance that any action selection mechanism could possibly achieve. The dependency structure of the underlying input and reward distribution is given in terms of an arbitrary Bayesian Network BN for discrete variables (e.g., Figure 4A). BN can be represented, like every other Bayesian network, by a directed graph without directed cycles. We do not assume any further restrictions on the structure of the Bayesian network, so BN does not have to be a tree (as assumed in Deneve, 2008), and it is not required to have no undirected cycles (as necessary for guaranteed convergence of belief propagation algorithms; Bishop, 2006).

Without loss of generality, we choose a numbering scheme such that the direct children of the reward node r in BN are $x_1, \dots, x_{m'}$. The dependencies in BN can be described by $m + 1$ parent sets P_k, which are possibly empty, and explicitly exclude the reward node r. P_k is thus defined as

$$\mathrm{P}_k = \{i \mid \text{a directed edge } x_i \to x_k \text{ exists in BN and } x_i \neq r\}.$$

Additionally we define P_r as the set of all parents of the reward node r. The joint probability distribution on the variables $r, x_1, \dots, x_m$ in the Bayesian network for action a can then be factored, giving rise to a factor graph (Kschischang et al., 2001) as indicated in Figure 4B:

$$p(r, \mathbf{x} \mid a) = p(r \mid \mathbf{x}_{\mathrm{P}_r}, a) \prod_{k=1}^{m'} p(x_k \mid \mathbf{x}_{\mathrm{P}_k}, r, a) \prod_{k=m'+1}^{m} p(x_k \mid \mathbf{x}_{\mathrm{P}_k}, a). \tag{4.9}$$

When calculating the log-odd of obtaining reward or not, the last terms in equation 4.9 cancel out, and a simple application of Bayes' theorem leads to

$$\log \frac{p(r=1 \mid \mathbf{x}, a)}{p(r=0 \mid \mathbf{x}, a)} = \log \frac{p(r=1 \mid \mathbf{x}_{\mathrm{P}_r}, a)}{p(r=0 \mid \mathbf{x}_{\mathrm{P}_r}, a)} + \sum_{k=1}^{m'} \left(\log \frac{p(r=1 \mid x_k, \mathbf{x}_{\mathrm{P}_k}, a)}{p(r=0 \mid x_k, \mathbf{x}_{\mathrm{P}_k}, a)} - \log \frac{p(r=1 \mid \mathbf{x}_{\mathrm{P}_k}, a)}{p(r=0 \mid \mathbf{x}_{\mathrm{P}_k}, a)} \right). \tag{4.10}$$

This is a sum of conditional reward log-odds, which can all be learned with the reward-modulated Bayesian Hebb rule. We now develop a suitable sparse encoding of $x_1, \dots, x_m$ into binary variables $y_1, \dots, y_n$ (with $n \gg m$), such that the reward log-odd can be written as a weighted sum,

$$\log \frac{p(r=1 \mid \mathbf{y}, a)}{p(r=0 \mid \mathbf{y}, a)} = \sum_{i=1}^{n} w_{a,i}\, y_i,$$

and the weights $w_{a,i}$ correspond to conditional reward log-odds of y_i's. For the example Bayesian network in Figure 4A, the corresponding sparse code is illustrated in Figure 4D: one binary variable is created for every possible value assignment to a variable x_k and all its parents $\mathbf{x}_{\mathrm{P}_k}$, and additional binary variables are created for every possible value assignments to the parent nodes only. One should contrast this with the simple population code in Figure 4C, which assumes that the naive Bayes condition holds and therefore ignores that x_2 is dependent on x_1.

BN can also be viewed as a factor graph (see Figure 4B), in which there is for every variable x_k a factor f_k, which is connected to r, x_k, and $\mathbf{x}_{\mathrm{P}_k}$, the parents of x_k in BN. The preprocessing is then computed separately for every factor f_k. We define the fixed generalized preprocessing (GP) operation for f_k with $k \geq 1$ as

$$\boldsymbol{\Phi}(x_k, \mathbf{x}_{\mathrm{P}_k}) = \begin{bmatrix} \boldsymbol{\phi}(x_k, \mathbf{x}_{\mathrm{P}_k}) \\ -\boldsymbol{\phi}(\mathbf{x}_{\mathrm{P}_k}) \end{bmatrix}. \tag{4.11}$$

The summands of the sum on the right-hand side of equation 4.10 are split into two parts, and $\boldsymbol{\phi}(x_k, \mathbf{x}_{\mathrm{P}_k})$ defines the preprocessing for the first part, whereas $-\boldsymbol{\phi}(\mathbf{x}_{\mathrm{P}_k})$ defines the preprocessing for the latter part. The variables $\langle x_k, \mathbf{x}_{\mathrm{P}_k} \rangle$ are viewed as a single multinomial variable, and $\boldsymbol{\phi}(x_k, \mathbf{x}_{\mathrm{P}_k})$ is a representation of this multinomial variable through simple population coding. Thus, $\boldsymbol{\phi}(x_k, \mathbf{x}_{\mathrm{P}_k})$ has as many binary output variables $y_{k,i}$ as there are different assignments of values to all variables in $\langle x_k, \mathbf{x}_{\mathrm{P}_k} \rangle$, and exactly one variable $y_{k,i}$ has value 1 for each such assignment. Let $y_{k,i}$ be the binary output variable that corresponds to some assignment $x_k = j,\ \mathbf{x}_{\mathrm{P}_k} = \mathbf{u}$; then the corresponding weight $w_{a,k,i}$ for action a can be learned through the same reward-modulated Bayesian Hebb rule, 2.7, as in the naive Bayes case. The target value, to which $w_{a,k,i}$ will converge, is then

$$\begin{aligned} w^*_{a,k,i} &= \log \frac{p(r = 1 \mid y_{k,i} = 1, a)}{p(r = 0 \mid y_{k,i} = 1, a)} \\ &= \log \frac{p(r = 1 \mid x_k = j, \mathbf{x}_{\mathrm{P}_k} = \mathbf{u}, a)}{p(r = 0 \mid x_k = j, \mathbf{x}_{\mathrm{P}_k} = \mathbf{u}, a)}. \end{aligned} \tag{4.12}$$

Analogously, the application of the reward-modulated Bayesian Hebb rule, 2.7, for every component $y_{\mathrm{P}_k,i}$ of $-\boldsymbol{\phi}(\mathbf{x}_{\mathrm{P}_k})$ leads to the target weights,

$$w^*_{a,\mathrm{P}_k,i} = \log \frac{p(r = 1 \mid y_{\mathrm{P}_k,i} = -1, a)}{p(r = 0 \mid y_{\mathrm{P}_k,i} = -1, a)} = \log \frac{p(r = 1 \mid \mathbf{x}_{\mathrm{P}_k} = \mathbf{u}, a)}{p(r = 0 \mid \mathbf{x}_{\mathrm{P}_k} = \mathbf{u}, a)}, \tag{4.13}$$

with the only formal modification to the update rule, equation 2.7, being that updates are made not only when $y_i = 1$ but also when $y_i = -1$, which obviously does not change the behavior of the learning process. Formally, all preprocessed vectors $\boldsymbol{\Phi}(x_k, \mathbf{x}_{\mathrm{P}_k})$ are concatenated into one vector $\mathbf{y}$ with

$n = \sum_{k=1}^{m'} N_k + N_{\mathrm{P}_k}$ entries:

$$\mathbf{y} = \begin{bmatrix} \boldsymbol{\Phi}(\mathbf{x}_{\mathrm{P}_r}) \\ \boldsymbol{\Phi}(x_1, \mathbf{x}_{\mathrm{P}_1}) \\ \vdots \\ \boldsymbol{\Phi}(x_{m'}, \mathbf{x}_{\mathrm{P}_{m'}}) \end{bmatrix}.$$

This sparse, redundant input encoding provides a weighted sum representation of the reward log-odd,

$$\log \frac{p(r = 1 \mid \mathbf{y}, a)}{p(r = 0 \mid \mathbf{y}, a)} = \sum_{i=1}^{n} w_{a,i}\, y_i,$$

where the weights $w_{a,1}, \ldots, w_{a,n}$ can all be learned through the reward-modulated Bayesian Hebb rule, 2.7, as described above.

5 Results of Computer Simulations

We now evaluate the performance of the reward-modulated Bayesian Hebb rule and its linear approximation and compare it to the standard learning model for simple conditioning tasks, the non-Hebbian Rescorla-Wagner rule (Rescorla & Wagner, 1972).

The reward-modulated Bayesian Hebb rule, 2.7, was tested on a variety of action selection tasks with four possible actions. A Bayesian network with dependency structure as in Figure 4A was used to model the distribution $p(r, x_1, x_2 \mid a)$ for every action a, where r is the binary reward signal and x_1, x_2 are the two binary input signals. We assigned a constant reward prior $p(r \mid a) = 0.25$ to every action a and randomly generated the conditional probability tables for $p(x_1 \mid r, a)$ and $p(x_2 \mid x_1, r, a)$: for every action a, every x_k ($k \in \{1, 2\}$), and every possible value assignment to the parent nodes $\langle \mathbf{x}_{\mathrm{P}_k}, r \rangle$, a random sample $q \in [0, 1]$ was drawn from a beta distribution, and $p(x_k = 1 \mid \mathbf{x}_{\mathrm{P}_k}, r, a)$ was set to q.

The Bayesian networks that model the reward distribution were also used to create the samples of input vectors $\mathbf{x} = \langle x_1, x_2 \rangle$ for every training trial. First, one of the four Bayesian networks was chosen randomly with equal probability, so the distribution of input or test samples does not depend on the action selection during learning. Inputs $\mathbf{x}$ were drawn as random samples from the selected network. The agent then received the input $\mathbf{x}$ and chose its action a. The binary reward signal r was sampled from the distribution $p(r \mid \mathbf{x}, a)$ and thus depends on the chosen action. The agent used the tuple $\langle \mathbf{x}, a, r \rangle$ to update its weights $w_{a,i}$. Training consisted of 2000 trials, in which the matching behavior strategy (see section 2.1) was used for action selection during learning. The evaluation of the performance of the resulting policy after every trial used the greedy strategy, equation 2.2, choosing actions on 500 independent test trials and measuring the average reward. The experiment was averaged over 250 different tasks with different reward distributions.

The preprocessed binary vectors $\mathbf{y} = \mathbf{\Phi}(\mathbf{x}) \in \{0, 1\}^n$ were created either by simple population coding (see equation 4.6 and Figure 4C), which is suitable for the naive Bayes case, equation 4.2, or generalized preprocessing (see equation 4.11 and Figure 4D). The former mechanism is referred to as Bayesian Hebb SP in Figure 5 and the remainder of this article, whereas the generalized preprocessing mechanism is referred to as Bayesian Hebb GP. The Bayesian Hebb rule with these two kinds of preprocessing mechanisms was compared to the non-Hebbian Rescorla-Wagner rule (Rescorla & Wagner, 1972). This rule predicts the value of a (multidimensional) stimulus as a linear sum,

$$V(\mathbf{y}) = w_0 + \sum_{i=1}^{n} w_i y_i,$$

and minimizes the prediction error with a delta learning rule,

$$\Delta w_i = \eta y_i \left(r - w_0 - \sum_{i=1}^{n} w_i y_i \right). \tag{5.1}$$

It can be seen from equation 5.1 that for the update of a single weight, the complete prediction of value for the current state, which depends on all weights, is needed. In the experiments, the Rescorla-Wagner rule was used to learn weights for every action separately. The classical Rescorla-Wagner rule, equation 5.1, which we use for comparison, is directly applied to the inputs $\mathbf{x}$. We show in appendix E that the performance and learning speed of Rescorla-Wagner can also be improved if it is applied to the preprocessed vectors $\mathbf{y} = \mathbf{\Phi}(\mathbf{x})$, using the same SP and GP preprocessing mechanisms as for the Bayesian Hebb rule.

In addition, the reward-modulated Bayesian Hebb rule was compared to a Bayes-optimal weight learning rule. In this case, the conditional probabilities in the Bayesian network in Figure 4A were estimated using counter variables (see section 2.2), and exact inference was used to compute reward probabilities for every action.

Figure 5 shows that the reward-modulated Bayesian Hebb rule for both types of preprocessing learns faster than the non-Hebbian Rescorla-Wagner rule and converges to better policies. If generalized preprocessing is used, the learned policy after approximately 200 trials is almost indistinguishable from the policy of an optimal learner, and after approximately 1000 trials, the performance is very close to the optimal performance level.

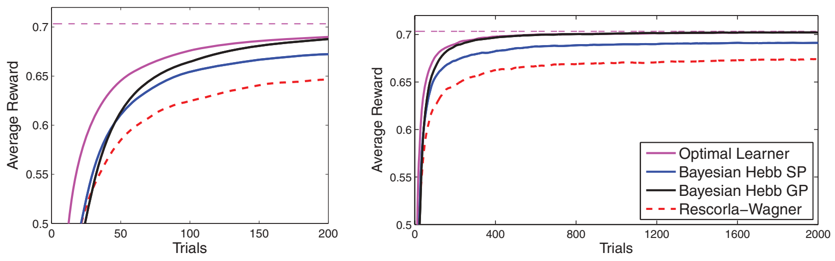

Figure 5: Performance of the reward-modulated Bayesian Hebb rule for action selection in a four-action task with stochastic rewards. Each learner was trained on 2000 trials, and after every trial, the performance was measured as the average reward of the greedy policy of each learner on 500 independent test trials (left: performance during the first 200 training trials). The results were averaged over 250 different problems, all having the statistical dependency structures as in Figure 4A, but random reward distributions (average learning and preprocessing time per problem on a dual-core 2.66 GHz, 16 GB RAM PC: 0.9 s for SP, and 4.1 s for GP). The horizontal dashed line reflects the best possible performance of an optimal policy. The Bayesian Hebb rules with simple population coding (Bayesian Hebb SP) and generalized preprocessing (Bayesian Hebb GP) were compared to action learning with the non-Hebbian Rescorla-Wagner rule. The learning rate was set to $1/N_{a,i}$, and stochastic action selection was used for exploration during training. The Bayesian Hebb rule for both preprocessing methods learned faster than the non-Hebbian Rescorla-Wagner rule and converged to better policies. With generalized preprocessing, the Bayesian Hebb rule converged to the optimal action selection policy, as predicted by the theoretical analysis. Error bars are in the range of 10^{-3} and are omitted for clarity.

5.1 Approximations to the Bayesian Hebb Rule. We showed in section 3 that the linear Bayesian Hebb rule, equation 3.4, can be derived as a first-order Taylor approximation of the reward-modulated Bayesian Hebb rule, 2.7. There are no theoretical guarantees that the linear Bayesian Hebb rule will asymptotically converge toward weight values that allow optimal decision making. We compared the two rules on the same random Bayesian network tasks for action selection empirically, using both the simple preprocessing (SP) for the naive Bayes case, and the generalized preprocessing (GP) for arbitrary reward distributions. Figure 6 shows that this even simpler rule found good policies as quick as the exact rule. The quality of the final policy was almost indistinguishable from the policies found by the exact Bayesian Hebb rule.

5.2 Adaptation to Changing Reward Distributions. In most realistic scenarios, an organism experiences during its lifetime changes in the environment in which it lives. It is therefore important that a learning rule can adapt quickly to a changing reward or input distribution. It is clear that a learning rate that decays with $\frac{1}{N_i}$ (where N_i is the number of updates for a weight w_i) is unsuitable for changing environments. We therefore used for this task the variance tracking mechanism for learning rate adaptation, which was first introduced by Nessler, Pfeiffer, and Maass (2009). This mechanism keeps track of the variance of each weight and adapts learning rates accordingly. Learning rates are reduced for weights with small fluctuations, whereas they are increased for weights with high variance, an indication that those weights have not yet settled at their equilibrium values.

The learning rate adaptation mechanism uses two auxiliary variables, which can be locally estimated for every weight w_i: a running average of the weight is computed in $\bar{w}_i$, and a running average of the squared weight in $\bar{q}_i$, using the following simple update rules:

$$\begin{aligned} \bar{w}_i^{new} &\leftarrow (1-\eta_i)\,\bar{w}_i + \eta_i\, w_i \\ \bar{q}_i^{new} &\leftarrow (1-\eta_i)\,\bar{q}_i + \eta_i\, w_i^2. \end{aligned} \tag{5.2}$$

With these values, the short-time variance of each weight can be estimated as $\bar{q}_i - \bar{w}_i^2$. Assuming that samples are drawn from stationary input distributions, Nessler et al. (2009) showed that the variance of a weight w_i can be related to the sample size N_i in the Bayes-optimal learning case (see also section 2.2), where exact counters for all combinations of inputs, actions, and rewards are used and conditional reward probabilities are modeled with beta distributions. According to this analysis, the new learning rate η_i^{new} can be set as

$$\eta_i^{new} \leftarrow \frac{\bar{q}_i - \bar{w}_i^2}{1+\cosh \bar{w}_i}. \tag{5.3}$$

In practice, this mechanism decays like $\frac{1}{N_i}$ under stationary conditions. It can also handle changing input distributions, because a new target value for w_i leads to larger updates Δw_i, thus increasing the short-time variance of the weight, and by equation 5.3, the learning rate η_i. Further details, and the theory behind this mechanism are described in Nessler et al. (2009).

The variance tracking mechanism is an analytically justified rule for setting learning rates. Biological implementations of qualitatively similar processes are plausible, since all auxiliary quantities can be observed locally at the synapse. What is required is essentially a process that locally modulates potentiation or depression of synapses and itself is dependent on the magnitude of recent local synaptic weight changes. This could in principle be achieved by a large variety of metaplasticity mechanisms that are known to modulate synaptic plasticity (see Abraham, 2008, for a recent review). Neuromodulators such as acetylcholine and norepinephrine could play a

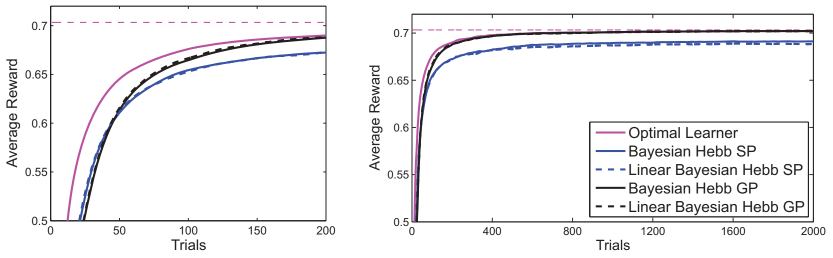

Figure 6: Performance of the linear approximations to the reward-modulated Bayesian Hebb rule in the same four-action tasks as in Figure 5 (left: performance during the first 200 training trials). Both for simple population coding (SP) and generalized preprocessing (GP), the linear approximation to the learning rule learned as well as the exact rule. Error bars are in the range of 10^{-3} and are omitted for clarity.

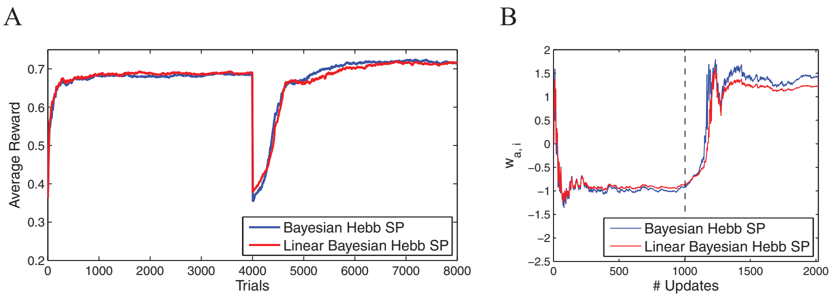

Figure 7: Behavior of the Bayesian Hebb rule when the reward distribution changes during training. (A) Performance of the agent if a new reward distribution is introduced after 4000 training trials. There is an immediate drop when the distribution changes, but good performance is recovered quickly by both rules. (B) Evolution of a single weight $w_{a,i}$ when the reward distribution changes. The weights are plotted at every trial where action a is selected, and an update for the plotted weight occurs. The weight first settles at the desired value for the first distribution and then quickly adapts to the new target value when the distribution changes (indicated by the black dashed line).

special role in the control of learning rates and the reduction of oscillations of weight updates (Doya, 2002; Yu & Dayan, 2003).

In the experiment shown in Figure 7, the weights were learned in 4000 training trials, after which the environment was changed and the learner was trained for another 4000 trials on the new input and reward distributions. Figure 7A shows that the performance of the learners initially improved, then dropped as soon as the distributions were switched, but quickly adapted to the new distribution, reaching almost the same

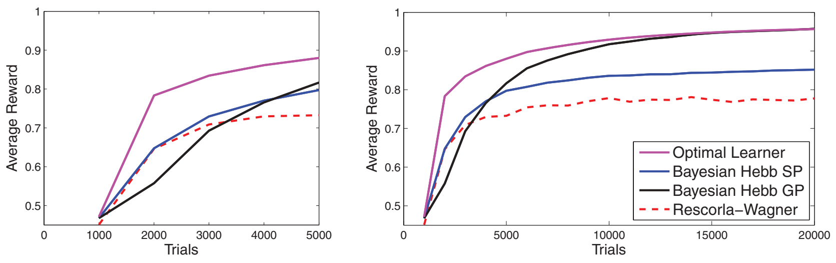

Figure 8: The Bayesian Hebb rule works well also for simulations with large input and action spaces. Each learner was trained on 20,000 trials of action selection problems with 10 actions, 100 binary input attributes, and stochastic rewards. Every 1000 trials, the performance was measured as the average reward of the greedy policy of each learner on 1000 independent test trials (left: performance during the first 5000 training trials). The results were averaged over 40 different problems with random statistical dependency structures and random reward distributions (average learning and preprocessing time per problem on a 2-core 2.66 GHz, 16 GB RAM PC: 27.8 s for SP, and 301.6 s for GP). The learning rates were set to $1/N_{a,i}$, and random action selection was used for exploration during training. With generalized preprocessing, the Bayesian Hebb rule approached the performance of an optimal learning mechanism. Error bars are in the range of 10^{-2} and are omitted for clarity.

performance. Figure 7B shows the evolution of a single weight in this scenario for all trials in which it was updated. It can be seen that the weight first settled around the equilibrium value of the first distribution and grew to reach the new target value after the switch.

5.3 Simulations for Large Input and Action Spaces. The Bayesian Hebb rule also works well for significantly larger problems. The same algorithms as in the previous sections were applied to problems with 100 binary input attributes and 10 possible actions. The structures of the Bayesian networks that define the reward distributions for every action were generated randomly, using the algorithm described in Ide and Cozman (2002). Every node in the network could have a maximum of five parent nodes. The protocol for the generation of training samples and rewards was the same as for the previous experiments (see beginning of section 5). During learning, actions were selected randomly, and the greedy policy was used for the evaluation on 1000 independent test trials (once every 1000 training trials).

Figure 8 shows that the Bayesian Hebb rule learns fast, both for simple population coding (SP) and generalized preprocessing (GP). The latter initially performs worse than SP because the number of weights to learn is very large (about 1000 weights for every action), and approximation errors sum up. Given more training data, the Bayesian Hebb rule with generalized preprocessing approaches the performance of an optimal learner. The linear approximations to the reward-modulated Bayesian Hebb rule perform equally well on this task for both types of preprocessing.

6 Decision Making with Continuous Inputs

The Bayesian Hebb rule can be generalized to action-selection problems defined on continuous input distributions. A rule very similar to equation 2.7 learns reward log-odds on a continuous input encoding, comparable to population codes with bell-shaped tuning curves that are observed in the brain.

The Bayesian Hebb rule has previously been defined only for discrete inputs x_k, which were mapped to binary variables y_i with various ways of preprocessing. We now present a learning rule to approximate distributions of a binary reward variable for continuous inputs. The preprocessing for this case is a population code that uses radial basis functions (RBFs)[3] to map continuous input variables x_k to new continuous features y_i, which may, for example, correspond to firing rates in a neural population code. Population codes with RBF- or bell-shaped tuning curves have been observed, for example, in area MT of the visual system for direction-sensitive cells (see Pouget & Latham, 2002, for a review), place cells in rat hippocampus (O'Keefe, Burgess, Donnett, Jeffery, & Maguire, 1998), or for the encoding of movement directions in primate motor cortex (Georgopoulos, Schwartz, & Kettner, 1986). Networks of RBF units are also commonly used for models of visual object recognition (Riesenhuber & Poggio, 1999).

Consider the input variables $\mathbf{x} = \langle x_1, \ldots, x_m \rangle \in X \subseteq \mathbb{R}^m$ and a binary reward variable $r \in \{0, 1\}$. The continuous input $\mathbf{x}$ is mapped to a new set of n continuous nonnegative features y_i. The activation of feature y_i is proportional to the activation of an RBF kernel $\phi_i(\mathbf{x})$:

$$\phi_i(\mathbf{x}) = \exp\left(-\sum_{k=1}^{m} \frac{|x_k - c_{i,k}|^2}{s_{i,k}^2}\right). \tag{6.1}$$

The centers of the RBF kernels are located at $\mathbf{c}_i = \langle c_{i,1}, \ldots, c_{i,m} \rangle$, and the widths of the kernels are given by $s_{i,k}$ (different widths may be used for different input dimensions). The preprocessed vector $\mathbf{y} = \langle y_1, \ldots, y_n \rangle$ is obtained by calculating the activations of all n different RBFs and normalizing the vector:

$$y_i(\mathbf{x}) = \frac{\phi_i(\mathbf{x})}{\sum_{j=1}^{n} \phi_j(\mathbf{x})}. \tag{6.2}$$

[3]Other mappings are also possible but are not presented in this article.

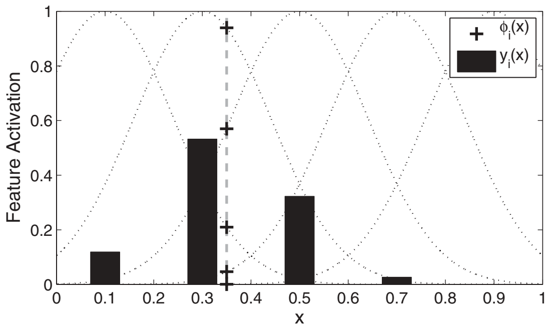

Figure 9: Example of a continuous population code with five equally spaced RBF kernels (width $s = 0.2$) for a one-dimensional input x. The activations of the RBF kernels $\phi_i(x)$ depend on the distance between x and the center c_i of the kernel. The normalized features $y_i(x)$ are obtained by dividing every $\phi_i(x)$ by the total sum of activations. The RBF kernel activations $\phi_i(x)$ (black crosses mark the intersection of the vertical line at $x = 0.35$ with the five RBF kernels indicated by dotted lines) and the normalized feature activations $y_i(x)$ (dark bars) are shown here for an example input at $x = 0.35$ (gray dashed line).

Notice that this kind of preprocessing can take combinations of variables into account, such as RBF kernels on $\mathbb{R}^m$, not only single variables. Figure 9 illustrates a simple continuous population code for five RBF kernels in one input dimension.

A rule for learning reward log-odds conditioned on a single feature $y_i = y_i(\mathbf{x})$ can be defined by generalizing the reward-modulated Bayesian Hebb rule, 2.7. Whenever action a is selected, every weight $w_{a,i}$ is updated by

$$\Delta w_{a,i} = \begin{cases} \eta \cdot y_i(\mathbf{x}) \cdot (1 + e^{-w_{a,i}}), & \text{if } r = 1 \\ -\eta \cdot y_i(\mathbf{x}) \cdot (1 + e^{w_{a,i}}), & \text{if } r = 0 \end{cases}. \tag{6.3}$$

This rule is a generalization of rule 2.7 in which the updates are weighted by the activation of feature y_i. For the previously described discrete population codes, where y_i is either 0 or 1, rule 6.3 is equivalent to 2.7.

For the analysis of the equilibrium of rule 6.3, we use an alternative population code of virtual binary features $\tilde{y}_1, \ldots, \tilde{y}_n$. We interpret $y_1(\mathbf{x}), \ldots, y_n(\mathbf{x})$ as (nonnormalized) probabilities for randomly selecting one $i \in \{1, \ldots, n\}$, for which one sets $\tilde{y}_i = 1$ (while setting $\tilde{y}_j = 0$ for $j \neq i$). This gives a new interpretation to the continuous population code features $y_i(\mathbf{x})$, because they are proportional to the probability that $\tilde{y}_i = 1$ (we then say that "feature $\tilde{y}_i$ is active").

To find the equilibrium of rule 6.3 for the weight $w_{a,i}$, we set the expected update $E[\Delta w_{a,i}]$ to zero and rewrite it as

$$E[\Delta w_{a,i}] = 0 \Leftrightarrow (1 + e^{-w_{a,i}}) \int_X y_i(\mathbf{x})\ p(r = 1, \mathbf{x} \mid a)\, d\mathbf{x}$$
$$-(1 + e^{w_{a,i}}) \int_X y_i(\mathbf{x})\ p(r = 0, \mathbf{x} \mid a)\, d\mathbf{x} = 0.$$

We show in appendix D that this condition is fulfilled if and only if $w_{a,i}$ is at the target value:

$$w^*_{a,i} = \log \frac{p(r = 1 \mid \tilde{y}_i = 1, a)}{p(r = 0 \mid \tilde{y}_i = 1, a)}.$$

If the active (virtual) feature $\tilde{y}_i$ was known, the corresponding weight $w_{a,i}$ would directly indicate the log-odd of obtaining reward with action a. In this scenario, however, only the continuous features $y_i(\mathbf{x}), i = 1, \ldots, n$ are known. Due to the normalization, the feature values sum up to 1, and one can therefore weight every $w_{a,i}$ by $y_i(\mathbf{x})$, yielding,

$$L_a(\mathbf{x}) = \sum_{i=1}^{n} w_{a,i}\ y_i(\mathbf{x}), \tag{6.4}$$

which is an interpolation between the reward log-odds $w_{a,i}$ for different features $\tilde{y}_i$. The interpolation weights are in this case the factors $y_i(\mathbf{x})$, that means that those features $\tilde{y}_i$ that are more likely to be active contribute more to the weighted sum, since $y_i(\mathbf{x})$ is proportional to $p(\tilde{y}_i = 1 \mid \mathbf{x})$. $L_a(\mathbf{x})$ thus approximates the reward log-odd $\log \frac{p(r=1|\mathbf{x},a)}{p(r=0|\mathbf{x},a)}$, and the reward probability $p(r = 1 \mid \mathbf{x}, a)$ can be approximated by

$$p(r = 1 \mid \mathbf{x}, a) \approx \sigma(L_a(\mathbf{x})) = \frac{1}{1 + e^{-L_a(\mathbf{x})}}, \tag{6.5}$$

where $\sigma(.)$ is the log-sigmoidal transfer function.

6.1 Computer Experiments with Continuous Input. For the following experiment, reward distributions were defined on single continuous input variables $x \in [0, 1]$. For every action, a different reward distribution was modeled, and the learner's task was to approximate the true reward distributions with the continuous Bayesian Hebb rule, 6.3, and to choose the action with the highest reward probability. Two thousand training trials with inputs drawn from a uniform distribution on $[0, 1]$ were used, and

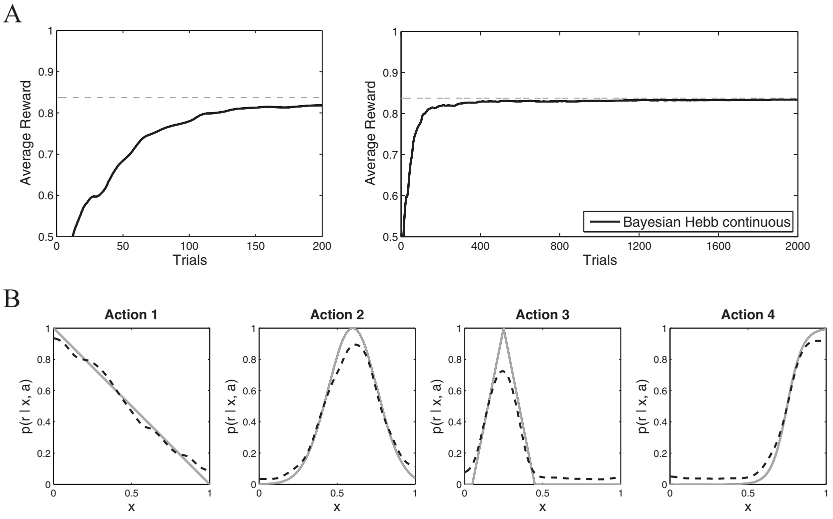

Figure 10: Performance of the Bayesian Hebb rule for continuous inputs. The input preprocessing consists of 20 RBF kernels that yield a population code **y** for the continuous inputs **x**. (A) Average reward of the learner obtained on 500 independent test trials during training on 2000 trials (left: performance during the first 200 training trials). The performance level rises quickly and in the end is close to the best possible performance of an optimal action selector (horizontal dashed line). Error bars are in the range of 10^{-3} and are omitted for clarity. Results are averaged over 32 runs. (B) Approximation of the reward probabilities learned by the continuous Bayesian Hebb rule after 2000 training trials. The learned approximation (dashed line) is very close to the true reward distribution (gray solid line).

the performance after every update was measured on 500 independent test trials. Twenty RBFs with constant widths $s = 0.05$ were used for the input preprocessing. The centers of the RBFs were equally distributed in the interval $[0, 1]$.

Figure 10 shows the performance at every training trial, and the approximations of the reward distributions that were obtained after 2000 training trials. The average reward obtained after training is close to the best possible performance, and the reward distributions are learned accurately.

7 Discussion

7.1 Summary and Open Problems. We have proposed in this article a simple neural network architecture for learning and decision making, which makes use of two learning processes that operate on two different timescales. We assume that generic dependencies among sensory input variables or features, or in other words, the factors of the underlying Bayesian network, are detected on a larger timescale, and that combinations of conditionally dependent input features are presented to the decision stage through sparse population coding. We have shown that on the basis of such preprocessing, the optimal policy can be represented as a WTA operation applied to weighted sums, and the corresponding weights can be learned very fast. In fact, we have shown that a very simple Hebbian learning rule (the reward-modulated Bayesian Hebb rule) can integrate information from experience in a close to optimal way. The models that we presented and analyzed are biologically plausible and arguably minimal with regard to their complexity, but nevertheless can be shown to asymptotically approximate theoretically optimal performance. All information from experience is stored in synaptic weights of simple linear neuron models and can therefore immediately be used for online decision making. In contrast to other learning rules that have previously been proposed for modeling animal learning—such as the Rescorla-Wagner rule (Rescorla & Wagner, 1972; Yuille, 2006), the perceptron learning rule, or learning rules based on the Kalman filter model (Sutton, 1992; Dayan & Kakade, 2001)—this new learning rule is a truly Hebbian learning rule. Its weight updates depend on the current pre- and postsynaptic activity, as well as on a third signal (Bailey et al., 2000) that contains information about the success or failure of the currently selected decision, but not on the current values of the other weights (or the resulting weighted sums of input variables). All information required for the weight update is therefore available locally at the synapse.

A major advantage of the local nature of purely Hebbian learning rules is that synapses can be removed or added to a neuron without changing the target weights of the other synapses. One can therefore view the reward-modulated Bayesian Hebb rule as a candidate for learning in self-organizing organisms with developing neural structure. Assume, for example, that an input variable x_{new} is added, and the population code is appropriately modified. Then all weights belonging to factors in the factor graph that are not connected to x_{new} are unaffected and can still be used for decision making. Removal or addition of single weights does, however, affect the decision-making process if the resulting population code does not match either the SP or GP encoding.

The Bayesian Hebb rule is one of very few online learning rules that admit a rigorous theoretical analysis of their convergence properties. We have shown that the theoretically optimal values of the weights are fixed-point attractors for expected weight changes (see Figure 2B). This implies, in particular, that learning cannot get stuck in local minima of some loss function. In fact, one can easily show that the expected weight updates give rise to an exponentially fast contracting dynamical system in weight space. Hence, this learning process falls into the theoretical framework of contracting systems, proposed by Lohmiller and Slotine (1998). According to this theory, this learning process can therefore be combined with other adaptive processes that also exhibit a contracting dynamics of adaptive parameters. Their theory guarantees that the resulting hybrid learning system will also converge.

We have also considered in section 3 a computationally simpler linear version of the Bayesian Hebb rule. Although this rule is only an approximation to the Bayesian Hebb rule and theoretical convergence results are weaker (see the discussion in section 3.1), we have shown that it performs almost equally well in a large number of complex decision-making tasks (see Figures 6, 7, and 11). The linear Bayesian Hebb rule is similar to well-known mathematical models for Hebbian learning and may therefore provide a new interpretation of these learning rules as approximations to more complex plasticity mechanisms

In this article, we have studied the scenario of online reward-based learning of decision making with multiple alternatives from stochastic rewards and input signals, which is important for fields like operant conditioning or reinforcement learning. In section 2.2 we have shown analytically and empirically (see Figure 2A) that the Bayesian Hebb rule achieves near-optimal learning in terms of learning speed and asymptotically approaches the optimal policy for the given preprocessing mechanism. We have supported this theoretical prediction through a variety of computer simulations of decision tasks (see Figures 5, 8, and 10). The resulting higher learning speed is particularly interesting in our context of reward-based learning, where most learning algorithms are too slow to be applicable to real-world problems. Hence the contribution of this article can be seen as another step in the program to speed up reinforcement learning by making near-optimal use of previous experience. We have shown in section 5.2 that this approach can also be applied to nonstationary distributions of inputs and rewards.

The question of how the brain forms decisions that involve more than two alternatives is one of the most important open research problems (Gold & Shadlen, 2007). For binary decisions, Wald's sequential probability ratio test (Wald & Wolfowitz, 1948) provides a theoretically optimal tool for learning and decision making from limited evidence. In this case, it is sufficient to update a single decision variable and compare it to a threshold value. For problems with more than two alternatives, it is unclear whether an optimal test exists, and tests that guarantee asymptotic optimality, such as the method developed by Dragalin, Tartakovsky, and Veeravalli (1999) become much more complex (see Gurney & Bogacz, 2006, for a possible neural implementation). In this article we have studied a simpler network model, which does not select actions optimally in the sense of sequential analysis. It converges asymptotically to an optimal policy and uses heuristic strategies for choosing actions during learning. We have analyzed a model that is based on the winner-take-all (WTA) operation and directly uses the learned weights for the evaluation of actions. We have shown that if WTA is applied to several linear neurons, each of which learns with the Bayesian Hebb rule to approximate the log-odd of receiving a reward for an associated action (see Figure 1), our simple model can handle the case of more than two decision alternatives without any extra effort (see sections 2 and 3 for the theoretical analysis and Figures 5, 6, 8, and 10 for empirical tasks).

WTA circuits are of interest in the context of neural network models for action selection, since it has been suggested that generic cortical microcircuits implement a soft version of WTA circuits (where $z_a > 0$ also for the runner-ups in the competition among the L_a; see Douglas & Martin, 2004). This view is supported by the anatomical observation that the output cells (pyramidal neurons) of cortical microcircuits are subject to lateral inhibition (each pyramidal neuron excites inhibitory interneurons that target other pyramidal neurons). It is also supported by the physiological observation that simultaneous activation of very large numbers of sensory neurons (for example in the retina) is transformed through cortical processing into sparse activity of neurons in higher sensory areas (e.g., area IT). Consequently, WTA circuits have become a primary target for the design of neurally inspired electronic hardware (Hahnloser, Sarpeshkar, Mahowald, Douglas, & Seung, 2000; Neftci, Chicca, Indiveri, Slotine, & Douglas, 2008).

The components of our neural network model (see Figure 1) have substantial experimental and theoretical support. Hebbian learning, and the use of weighted sums for decision making (Roth, 1999), is clearly feasible for biological neurons. The other essential ingredient of our model for reward-based learning of decision making is a suitable preprocessing of variables $\mathbf{x}$ (typically representing sensory inputs) that form the evidence on which a decision has to be based in a single trial. Our model requires a sparse population coding of the values of these variables (for both variables with discrete and with continuous values; see section 6). Sparse encodings (Olshausen & Field, 1996) or population codes are common models for coding strategies of the brain, and experimental evidence for the existence of such codes has been found in various brain areas of different species (see, e.g., Pouget & Latham, 2002; O'Keefe et al., 1998; Georgopoulos et al., 1986).

Furthermore, in the case of conditioned dependencies among variables, our model assumes that there exists a population coding for "complex features" (reminiscent of neural codes reported for example for visual areas V2 and IT), that is, for combinations of variables (see Figure 4 for an example). Hence, our simple neural network model for learning decision making entails concrete predictions for the computational strategies, neural codes, and learning mechanisms in those cortical areas that provide information about sensory inputs in a highly processed form to other cortical areas where decisions are made. It proposes that those subgroups of sensory variables (from the same or different sensory modalities) that have statistical dependencies, such as those represented by a factor graph (Kschischang et al., 2001), are brought together in some cortical microcircuits and that projection neurons from these cortical microcircuits each assume a high firing rate for a particular combination of values of these variables (thereby mimicking the output variables y_i of our general preprocessing; see section 4.1).

This link of factor graph theory and experimentally observed population codes provides a novel view on the potential role of sensory areas that provide input to higher decision-making stages in the brain. The proposed preprocessing has the advantage of relieving the subsequent decision stage from complex computations (such as belief propagation by message passing) and nonlinear learning devices. In fact, it enables the decision stage to use only linear operations in conjunction with WTA. It also enables the decision stage to accumulate evidence from history through the very simple and robust Hebbian learning processes discussed in this article.

In this article, we assume that the graph structure of the factor graph is known, which is a very common assumption for parameter learning algorithms in graphical models (see, e.g., Neapolitan, 2004; Jensen & Nielsen, 2007). The evolution of preprocessing circuits is obviously a complex process, and the design of learning algorithms that generate such preprocessing of sensory inputs is an interesting open problem. Testing variables for (conditional) dependence is perhaps a less formidable problem for a neural network than it may appear on first sight, provided one assumes that numerous autonomous learning processes try to predict each variable in terms of others. Dependencies among the variables exist and can in principle be found autonomously by this process, whenever such prediction learning turns out to be successful. As mentioned above, such relationships between input signals may be learned on much longer timescales than decision strategies, which require very fast adaptation.

Other obvious open problems that arise from our model are whether it can be implemented with spiking neurons and whether there exist relationships between the theoretically optimal reward-modulated Bayesian Hebb rule and concrete heterosynaptic learning mechanisms of biological synapses such as those discussed in Bailey et al. (2000). Another open problem concerns a possible extension of our model to rewards signals with more than two values, to third signals that represent predictions of rewards, and to reward-based learning in continuous time.

Altogether our simple neural network model for learning decision making has shown that this problem is in some aspects less difficult than it may appear on first sight. It remains to be explored whether biological neural systems have adopted related implementation strategies or have found even simpler solutions to this problem.

7.2 Related Work

7.2.1 Models for Decision Making. The study of decision making in biological systems dates back to the classical experiments by Pavlov, in which dogs learned associations between cues and rewards. On the other hand, operant or instrumental conditioning is concerned with associations between actions and rewards and how behavior is modified through reward and punishment. The goal is to learn a policy, that is, a way to select actions near-optimally in response to environmental stimuli. According to Sugrue, Corrado, and Newsome (2005), biological organisms first transform sensory input into decision-related variables, such as value representations in area LIP for visual discrimination tasks in monkeys (Yang & Shadlen, 2007). An unknown computational mechanism maps the values of these variables to the probability of reward for executing various actions, which then leads to a motor response. An actor-critic model is assumed, in which the actor and the critic are two modules that operate with a common reward currency. The critic adapts the value of every action to the perceived reward probabilities, thereby altering the decision transformation, which the actor uses to choose actions. An example for models of instrumental conditioning is the experiment of Montague, Dayan, Person, and Sejnowski (1995), in which the behavior of a foraging bee is simulated with a neural network model and a suitable learning rule (a variation of the Rescorla-Wagner rule). Wang (2002) has described a recurrent cortical network model, which uses feedback and winner-take-all mechanisms to integrate information in visual discrimination tasks with two possible outcomes. Gurney and Bogacz (2006) have presented a model for optimal decision making with multiple actions, which models the functionality of the basal ganglia. Further neural network models for decision making have been reviewed in Sakai, Okamoto, and Fukai (2006).

7.2.2 Learning Rules for Decision Making. The classical model for learning associations of stimuli, actions, and rewards is the Rescorla-Wagner rule (Rescorla & Wagner, 1972). It was the first mathematical model for learning that could explain most of the effects observed in animal behavior studies. In particular it was able to explain reactions based on combinations of stimuli. Reward associations for many conditioning paradigms, such as partial reinforcement, inhibitory conditioning, or extinction, can be learned by the Rescorla-Wagner rule (and also by the Bayesian Hebb rule). The associative model of the Rescorla-Wagner rule represents the predicted amount of reward as a weighted sum of stimuli, and weights are updated using the difference between the predicted and the actually received reward (see equation 5.1). The Rescorla-Wagner rule is therefore not a strictly Hebbian learning rule, because this error signal, rather than the activation of the postsynaptic neuron is required for the update. Studies by Schultz, Dayan, and Montague (1997) have, however, indicated that such an error signal may be available in the form of the neuromodulator dopamine.

Learning rules that minimize prediction errors were also useful to explain blocking phenomena in conditioning (Dayan & Abbott, 2001). However, some observed effects like backward blocking—an established reward association is unlearned, because another stimulus sufficiently explains the occurrence of rewards—cannot be sufficiently captured by the Rescorla-Wagner rule or the Bayesian Hebb rule. The reason is that weights in these models can only be reduced if unrewarded trials are observed (which is not the case in the backward-blocking paradigm). Algorithms that specifically address learning of reward associations in the backward-blocking scenario are based on Kalman filter models for conditioning (Sutton, 1992). Dayan and Yu (2003) argue that in addition to error correction, it is necessary to model the uncertainty in the parameter estimates during learning, and neuromodulators like acetylcholine or norepinephrine could signal such uncertainty in biological systems (Yu & Dayan, 2003). An artificial recurrent neural network model, which approximates the Kalman filter estimates of reward associations for backward blocking, was presented by Dayan and Kakade (2001). A different learning mechanism is suggested by Griffiths and Tenenbaum (2005), who argue that phenomena like backward blocking could also be modeled by learning changes in the causal structure of the problem rather than by learning new reward associations.

The mathematical problem of learning optimal action selection is also well studied in the field of reinforcement learning (RL) (Sutton & Barto, 1998). Typical RL algorithms learn value- or Q-functions, which estimate the expected reward resulting from the execution of action a in state $\mathbf{x}$. The goal of RL is to converge to optimal policies, which select for every state those actions that maximize the expected reward (typically a discounted long-term reward for sequential decision problems). Classical RL algorithms do not directly aim at maximizing the online performance, that is, the amount of reward obtained during learning, but typically employ some heuristics to tackle the exploration-exploitation dilemma. This dilemma concerns the trade-off of online performance (exploitation) and exploration of unseen parts of the state and action space in order to improve the final policy. More recently the problem of optimizing online performance has attracted more attention in the RL literature (e.g., Kearns & Singh, 1998; Audibert, Munos, & Szepesvari, 2007; Auer, Jaksch, & Ortner, 2009). Asymptotic convergence of RL algorithms to the optimal policy can be guaranteed for discrete environments only if action values are stored in look-up tables with one entry for every combination of state and action. Such tabular representations are biologically not realistic, and for computers, the memory requirements are too large for most real-world applications. Value functions are therefore approximated, but convergence results exist only for a limited number of approximation schemes (Bertsekas & Tsitsiklis, 1996).

Using Bayesian inference for action selection in uncertain environments was studied by Attias (2003) and Verma and Rao (2006). They consider the problem of planning action sequences of fixed length for partially observable Markov decision processes with one or more fixed goal states. The dynamics of the environment are initially unknown. The learning part uses frequency counters to update conditional probabilities for transition and reward models. Planning is reduced to Bayesian inference in graphical models based on the learned parameters, which is computed with standard algorithms, for example, belief propagation or the junction tree algorithm (Bishop, 2006). The posterior over actions, given that start and goal state are fixed, is computed, and the maximally likely sequence of actions (and intermediate states in Verma & Rao, 2006) is selected. This approach is conceptually quite different from our approach, since our approach does not learn sequences of actions and does not require a defined goal state. The learned parameters in our model (the weights $w_{a,i}$) are not auxiliary variables but are directly used in the decision-making process. Furthermore, our approach requires only very basic and apparently biologically feasible mechanisms like Hebbian learning, weighted summations, and winner-take-all. Implementing full Bayesian inference is a much more difficult process, for which it is not clear how the brain can achieve it efficiently, although some models have been proposed (e.g., Rao, 2007; Deneve, 2008). Lansner and Ekeberg (1998), Kononenko (1998), Lansner and Holst (1996), and Sandberg, Lansner, Petersson, and Ekeberg (2002) have studied various learning rules (although not in a reinforcement learning context) that approximate optimal Bayesian inference. The learning rules differ from the Bayesian Hebb rule that was introduced in this article primarily by the fact that they require auxiliary counters for storing evidence from past experience.

7.2.3 Analogies to Recent Experimental Studies of Decision Making in Primates. Recent experimental results by Yang and Shadlen (2007) have shown that the previous experience of macaque monkeys in probabilistic decision tasks is represented by the firing rates of neurons in area LIP in the form of the log-likelihood ratio of receiving a reward for a particular action a in response to a stimulus $\mathbf{x}$, as in equation 1.1 of our framework. In their experiment, a monkey had to choose at each trial between two possible actions. It could choose to move the eyes towards either a red target R ($a = R$) or a green target G ($a = G$). The probability that a reward was received at either choice depended on four visual input stimuli $\mathbf{x} = (x_1, x_2, x_3, x_4)$ that had been shown at the beginning of the trial. Every stimulus $x_k, k = 1, \ldots, 4$, was one shape s_j out of a set of 10 possibilities $\{s_1, \ldots, s_{10}\}$ and had an associated weight $\omega_k = \omega(s_j)$, which had been defined by the experimenter. The log-odd of obtaining a reward was equal to the sum of $\omega_1, \ldots, \omega_4$:

$$\log \frac{p(r = 1 \mid \mathbf{x}, a = R)}{p(r = 1 \mid \mathbf{x}, a = G)} = \sum_{k=1}^{4} \omega_k \quad . \tag{7.1}$$

The monkey thus had to combine the evidence from four visual stimuli to optimize its action selection behavior. It also had to find out that reward probabilities depended only on the presented shapes, not on the order or location in which they were presented. A reward was assigned before the trial to one of the targets according to the distribution 7.1.

One can easily model this task in our framework, using a simple population code $\mathbf{y} = \mathbf{\Phi}(\mathbf{x})$ as in equation 4.6, where the stimulus $\mathbf{x}$ was encoded by a 40-dimensional binary vector $\mathbf{y}$ with exactly $m = 4$ inputs being 1. The positions of the 1′s corresponded to the four visual shapes shown during a trial. The log-odd of obtaining reward with action $a = R$ can then be written as a weighted sum,

$$\log \frac{p(r = 1 \mid \mathbf{y}, a = R)}{p(r = 0 \mid \mathbf{y}, a = R)} = \sum_{i=1}^{40} w_i^* y_i, \tag{7.2}$$

with

$$w_i^* = \log \frac{p(r = 1 \mid y_i = 1, a = R)}{p(r = 0 \mid y_i = 1, a = R)}. \tag{7.3}$$

Due to the symmetry of the task (reward is either at R or G), the log-odds in equations 7.1 and 7.2 are equivalent. The weights w_i can be learned with an efficient version of the reward-modulated Bayesian Hebb rule, 2.7, which takes this symmetry into account. The equilibrium w_i^* of weight w_i under this slightly modified rule is then exactly at the desired value (see equation 7.3). We simulated this task using a learner with the reward-modulated Bayesian Hebb rule and a $1/N_i$ learning rate for every weight. Figure 11A shows that this task can be successfully learned both by the exact reward-modulated Bayesian Hebb rule 2.7 and the linear approximation, 3.4. The learning rules learn as fast as the non-Hebbian Rescorla-Wagner rule, 5.1, and their performance is close to the theoretical optimum after 1000 training trials. Furthermore Figures 11B and 11C show that the intermediate and final policies resemble the behavior that was reported for two monkeys in Yang and Shadlen (2007).

The experimental data of Yang and Shadlen (2007) are consistent with the assumption that monkeys apply a WTA operation to the log-likelihood ratios,

$$L_a = \log \frac{p(r = 1 \mid \mathbf{x}, a)}{p(r = 0 \mid \mathbf{x}, a)},$$

which are, according to their model, represented through firing rates of neurons in area LIP. It is not known which values are represented by the firing rates y_i of the presynaptic neurons of these neurons. In our simple model, we model the neurons within the WTA circuit as linear neurons and assume that their output L_a can be written as a linear sum $L_a = \sum_{i=0}^{n} w_{a,i}\, y_i$ of variables y_i that represent a population coding of the sensory input $\mathbf{x}$. As we have shown in section 4, if this population coding is chosen in a suitable way, the true reward log-odd $\log \frac{p(r=1|\mathbf{x},a)}{p(r=0|\mathbf{x},a)}$ can in fact be written

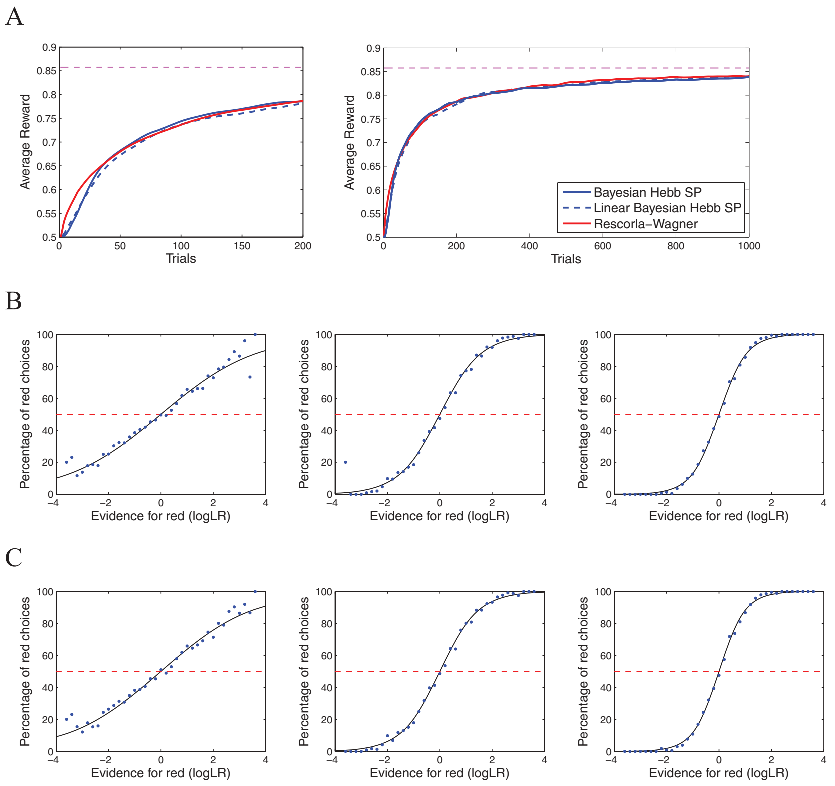

Figure 11: Performance of the reward-modulated Bayesian Hebb rule in the model for the conditioning task by Yang and Shadlen (2007). (A) The reward-modulated Bayesian Hebb rule learns as fast as the non-Hebbian Rescorla-Wagner rule (curves result from averaging over 32 repetitions of the experiment, where the average reward was measured on 500 independent test trials). The horizontal dashed line reflects the theoretically best possible performance. Error bars are in the range of 10^{-2} and are omitted for clarity. (B, C) Action selection policies (greedy policy according to equation 2.2) resulting from the model using the exact Bayesian Hebb rule, 2.7 (B) or the linear Bayesian Hebb rule, 3.4 (C) after 100 (left), 500 (middle), and 1000 (right) trials, fitted by sigmoidal curves (results are from 32 repetitions of the experiment, where the behavior was measured on 1000 independent test trials). The policies represented by the left and right panels are qualitatively similar to the policies adopted by monkeys H and J in the experiments by Yang and Shadlen (2007) after learning (see Figure 1b in Yang and Shadlen, 2007).

as such a weighted sum. Hence our theoretical framework makes concrete predictions about the nature of the transformation of raw sensory inputs $\mathbf{x}$ to inputs $\mathbf{y}$ for higher brain areas that select suitable responses. The required weights $w_{a,i}$ can be learned by the reward-modulated Bayesian Hebb rule, and a linear Poisson neuron whose weights are updated according to this rule will adapt for each trial a firing rate proportional to the log-likelihood ratio $\log \frac{p(a=R|\mathbf{x})}{p(a=G|\mathbf{x})}$. This response matches that of the neurons in area LIP shown in Figures 2c and 3b of Yang and Shadlen (2007).[4]

The Bayesian Hebb rule provides an arguably minimal model for the biological data of Yang and Shadlen (2007). One difference between their results and our model is that learning is much faster in our model. This could be explained by the fact that many aspects of the probabilistic decision task of Yang and Shadlen (2007)—for example, the fact that the reward policy was stationary, the fact that the reward probabilities did not depend on the order of appearance or the spatial location of the shown icons, and the fact that reward probabilities did not depend on any other aspects that the monkeys had perceived before or during a session—also had to be learned by the monkeys, whereas they were assumed as given in our model. Learning of these invariances and symmetries was actually quite hard in the setup of Yang and Shadlen (2007) since rewards were given stochastically rather than by deterministic laws (note that even many humans believe they can "learn" various misleading reward-predictors while gambling for a long time in the lottery or casinos). An interesting open question is whether reward-based learning of decision making by humans or animals can approach the learning speed of the Bayesian Hebb rule when such differences between the learning tasks of the living organisms and the mathematical model have been removed.

8 Conclusion

We have demonstrated the functionality of a simple neural network model for learning of asymptotically optimal action selection, which uses only biologically plausible mechanisms such as reward-modulated Hebbian learning, sparse population coding, and winner-take-all computations. Furthermore we have shown that on the basis of a suitable preprocessing that takes dependencies among salient variables into account, a very simple Hebbian learning rule can converge toward optimal policies extremely fast. Our approach offers concrete processing goals for brain areas that integrate multimodal sensory input in order to facilitate learning and decision making in higher brain areas. Empirical results have confirmed that the new reward-modulated Bayesian Hebb rule, and an even simpler linear

[4]Note that the optimal weights w_i^* are equal to the weights $\omega_k = \omega(s_j)$ that were assigned to the different visual shapes s_j.

approximation to it, compare favorably to well-known non-Hebbian learning rules for action-selection tasks. Our results suggest that learning and decision making under uncertainty can be implemented very efficiently in biological neural systems.

Appendix A: Convergence Proofs for the Bayesian Hebb Rule

We assume that $p(r \mid \mathbf{y}, a)$, the reward probability conditioned on the current input and action, is stationary, and $p(y_i = 1, a) > 0$ for all $a \in A$ and $i \in \{1, \ldots, n\}$. Apart from the latter assumption, the equilibrium is independent of the exploration policy $\pi(\mathbf{x}, a)$. The constraint on $p(y_i = 1, a)$ means that all values of all input variables must have a nonzero probability in the input distribution, and every action must have a nonzero probability of being tried out. If $p(y_i = 1, a) = 0$ for some y_i and a, then such trials are never encountered, and no meaningful weight $w_{a,i}$ can be learned.

Since updates of $w_{a,i}$ in equation 2.7 are made only when a is executed and $y_i = 1$, one can write

$$
\begin{aligned}
E[\Delta w_{a,i}] = 0 \Leftrightarrow\ & p(r = 1 \mid y_i = 1, a) \cdot \eta \cdot (1 + e^{-w_{a,i}}) - \\
& -p(r = 0 \mid y_i = 1, a) \cdot \eta \cdot (1 + e^{w_{a,i}}) = 0 \\
\Leftrightarrow\ & \frac{1 + e^{w_{a,i}}}{1 + e^{-w_{a,i}}} = \frac{p(r = 1 \mid y_i = 1, a)}{p(r = 0 \mid y_i = 1, a)} \\
\Leftrightarrow\ & e^{w_{a,i}} = \frac{p(r = 1 \mid y_i = 1, a)}{p(r = 0 \mid y_i = 1, a)} \\
\Leftrightarrow\ & w_{a,i} = \log \frac{p(r = 1 \mid y_i = 1, a)}{p(r = 0 \mid y_i = 1, a)}.
\end{aligned}
$$

The above is a chain of equivalence transformations; therefore $w^*_{a,i} = log \frac{p(r=1|y_i=1,a)}{p(r=0|y_i=1,a)}$ is the only equilibrium value of rule 2.7.

One can also show that the expected update of weights $w_{a,i}$ is always in the right direction:

$$
\begin{aligned}
E[\Delta w_{a,i}]|_{w^*_{a,i}+2\epsilon} &= E[\Delta w_{a,i}]|_{w^*_{a,i}+2\epsilon} - E[\Delta w_{a,i}]|_{w^*_{a,i}} \\
&\propto p(r{=}1|y_i = 1, a)e^{-w^*_{a,i}}(e^{-2\epsilon} - 1) - p(r{=}0|y_i = 1, a)e^{w^*_{a,i}}(e^{2\epsilon} - 1) \\
&= p(r{=}0|y_i = 1, a)(e^{-2\epsilon} - 1) - p(r{=}1|y_i = 1, a)(e^{2\epsilon} - 1) \\
&= \left[(p(r{=}0|y_i = 1, a)e^{-\epsilon} + p(r{=}1|y_i = 1, a)e^{\epsilon}\right](e^{-\epsilon} - e^{\epsilon}). \qquad \text{(A.1)}
\end{aligned}
$$

The first term in equation A.1 is always positive, and from the last term, one can, see that whenever $w_{a,i} > w^*_{a,i}$ (i.e., $\epsilon > 0$), the expected change of $w_{a,i}$ is negative, and it is positive if $\epsilon < 0$. The expected change of weights is therefore always in the direction of the optimal weight, and the initial weight values or perturbations of the weights decay exponentially fast. Furthermore, trajectories of weights that start at different initial values converge exponentially fast. Hence the resulting weight dynamics is contracting in the sense of Lohmiller and Slotine (1998).

Appendix B: Convergence Proof for the Linear Bayesian Hebb Rule

The expected update of the linear Bayesian Hebb rule, equation 3.4, vanishes when

$$\begin{aligned} E[\Delta w_{a,i}] = 0 \Leftrightarrow\ & p(r=1|y_i=1,a)\cdot\eta\cdot(2-w_{a,i}) \\ & -p(r=0|y_i=1,a)\cdot\eta\cdot(2+w_{a,i}) = 0 \\ \Leftrightarrow\ & 2(p(r=1|y_i=1,a)-p(r=0|y_i=1,a)) = \\ & = w_{a,i}\cdot(p(r=1|y_i=1,a)+p(r=0|y_i=1,a)) = w_{a,i} \\ \Leftrightarrow\ & w_{a,i} = 2(p(r=1|y_i=1,a)-1+p(r=1|y_i=1,a)) \\ \Leftrightarrow\ & w_{a,i} = -2+4\cdot p(r=1|y_i=1,a). \end{aligned}$$

We have used here that the reward is binary, and so

$$p(r=0|y_i=1,a)+p(r=1|y_i=1,a)=1.$$

The above is a chain of equivalence transformations, so $w^{+}_{a,i} = -2+4\cdot p(r=1|y_i=1,a)$ is the only equilibrium value of equation 3.4.

Appendix C: Derivation of the Population Code for the Naive Bayes Case

From the naive Bayes assumption, we know that

$$\frac{p(r=1|\mathbf{x},a)}{p(r=0|\mathbf{x},a)} = \frac{p(r=1|a)}{p(r=0|a)}\prod_{k=1}^{m}\frac{p(x_k|r=1,a)}{p(x_k|r=0,a)}\,. \tag{C.1}$$

Each discrete conditional distribution $p(x_k|r,a)$ for a fixed action a and a fixed value of r is fully described by m_k probability values, one for each possible value of x_k, and can be written in the form

$$\begin{aligned} p(x_k|r,a) = {} & p(x_k=1|r,a)^{I(x_k=1)} \\ & \cdot p(x_k=2|r,a)^{I(x_k=2)}\cdot\ldots\cdot p(x_k=m_k|r,a)^{I(x_k=m_k)}, \end{aligned}$$

where the indicator function I is defined as $I(\texttt{true}) = 1$ and $I(\texttt{false}) = 0$. With this notation, equation C.1 can be rewritten as

$$\frac{p(r=1|\mathbf{x},a)}{p(r=0|\mathbf{x},a)} = \frac{p(r=1|a)}{p(r=0|a)} \prod_{k=1}^{m}\prod_{j=1}^{m_k}\left(\frac{p(x_k=j|r=1,a)}{p(x_k=j|r=0,a)}\right)^{I(x_k=j)}$$
$$= \frac{p(r=1|a)}{p(r=0|a)} \prod_{k=1}^{m}\prod_{j=1}^{m_k}\left(\frac{p(r=1|x_k=j,a)}{p(r=0|x_k=j,a)}\cdot\frac{p(r=0|a)}{p(r=1|a)}\right)^{I(x_k=j)}$$
$$= \left(\frac{p(r=1|a)}{p(r=0|a)}\right)^{1-m} \prod_{k=1}^{m}\prod_{j=1}^{m_k}\left(\frac{p(r=1|x_k=j,a)}{p(r=0|x_k=j,a)}\right)^{I(x_k=j)}. \tag{C.2}$$

Appendix D: Convergence Proof for the Continuous Bayesian Hebb Rule

The equilibrium of the continuous Bayesian Hebb rule, equation 6.3, is reached when the expected update $E[\Delta w_{a,i}]$ vanishes:

$$E[\Delta w_{a,i}] = 0 \Leftrightarrow (1+e^{-w_{a,i}})\int_X y_i(\mathbf{x})\ p(r=1,\mathbf{x}|a)\,d\mathbf{x}$$
$$-(1+e^{w_{a,i}})\int_X y_i(\mathbf{x})\ p(r=0,\mathbf{x}|a)\,d\mathbf{x} = 0.$$

We now use the interpretation of $y_i(\mathbf{x})$ as $p(\tilde{y}_i = 1|\mathbf{x})$. Since the virtual population code feature $\tilde{y}_i$ depends only on $\mathbf{x}$, not on r, one can assume that r and $\tilde{y}_i$ are conditionally independent given $\mathbf{x}$:

$$p(r, \tilde{y}_i|\mathbf{x}, a) = p(r|\mathbf{x}, a)\cdot p(\tilde{y}_i|\mathbf{x}).$$

This assumption, and simple transformations using basic laws of probability lead to

$$E[\Delta w_{a,i}] = 0 \Leftrightarrow \frac{1+e^{w_{a,i}}}{1+e^{-w_{a,i}}} = \frac{\int_X p(\tilde{y}_i=1|\mathbf{x})\ p(r=1|\mathbf{x},a)\ p(\mathbf{x}|a)\,d\mathbf{x}}{\int_X p(\tilde{y}_i=1|\mathbf{x})\ p(r=0|\mathbf{x},a)\ p(\mathbf{x}|a)\,d\mathbf{x}}$$
$$\Leftrightarrow e^{w_{a,i}} = \frac{\int_X p(\tilde{y}_i=1, r=1|\mathbf{x},a)\ p(\mathbf{x}|a)\,d\mathbf{x}}{\int_X p(\tilde{y}_i=1, r=0|\mathbf{x},a)\ p(\mathbf{x}|a)\,d\mathbf{x}}$$
$$\Leftrightarrow e^{w_{a,i}} = \frac{\int_X p(\tilde{y}_i=1, r=1,\mathbf{x}|a)d\mathbf{x}}{\int_X p(\tilde{y}_i=1, r=0,\mathbf{x}|a)d\mathbf{x}}$$
$$\Leftrightarrow e^{w_{a,i}} = \frac{p(\tilde{y}_i=1, r=1|a)}{p(\tilde{y}_i=1, r=0|a)}$$

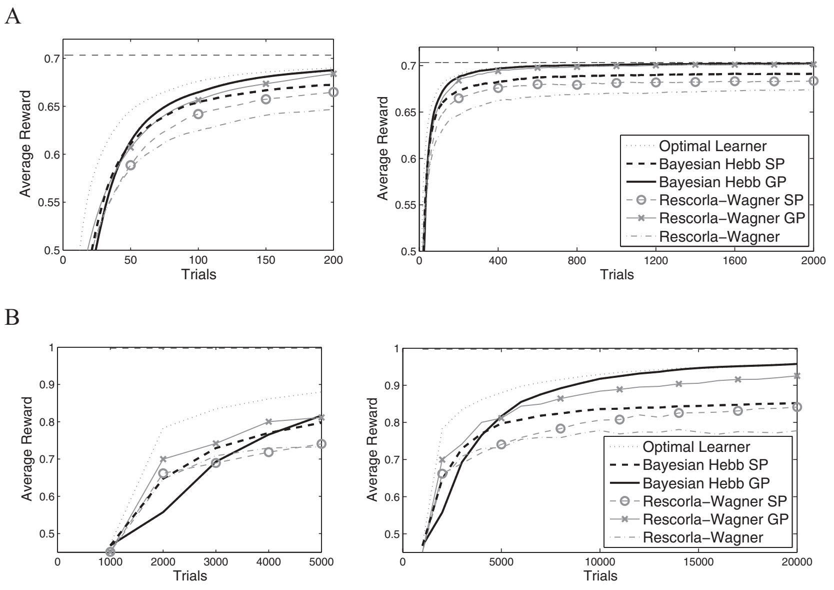

Figure 12: The performance of the Rescorla-Wagner rule can be improved by preprocessing input signals. The Rescorla-Wagner rule was applied to preprocessed inputs using simple population coding (Rescorla-Wagner SP), or generalized preprocessing (Rescorla-Wagner GP). The Rescorla-Wagner rule with preprocessing generally learned faster and converged to better policies than the classical Rescorla-Wagner rule. (A) Performance for the same 4-action tasks with 2 binary input variables as in Figure 5. (B) Performance in the same 10-action tasks with 100 binary input variables as in Figure 8.

$$\Leftrightarrow e^{w_{a,i}} = \frac{p(r=1|\tilde{y}_i=1,a)}{p(r=0|\tilde{y}_i=1,a)}$$

$$\Leftrightarrow w_{a,i} = \log\frac{p(r=1|\tilde{y}_i=1,a)}{p(r=0|\tilde{y}_i=1,a)} \quad .$$

Appendix E: Performance of the Rescorla-Wagner Rule with preprocessing

The performance of the Rescorla-Wagner rule, equation 5.1, can be improved by preprocessing input signals before the learning rule is applied. Figure 12 shows the average reward for the two tasks studied in Figure 5 (with 2 binary inputs and 4 actions), and Figure 8 (with 100 binary inputs and 10 actions). When the Rescorla-Wagner rule was applied to simple population coding (SP) or generalized preprocessing (GP), it learned faster and converged to better policies, although the performance of the Bayesian Hebb rule was mostly superior. These results suggest that the preprocessing methods presented in section 4 could also be beneficial for other learning mechanisms. The augmented Rescorla-Wagner rule (Yuille, 2006) uses a preprocessing mechanism similar to GP, but it did not perform better for the experiments in this article.

Acknowledgments

We thank Martin Bachler, Sophie Deneve, Konrad Körding, Rajesh Rao, and especially Dan Roth for inspiring discussions. This article was written under partial support by the Austrian Science Fund FWF project S9102-N13 and project FP6-015879 (FACETS), project FP7-216593 (SECO), project FP7-506778 (PASCAL2), and project FP7-231267 (ORGANIC) of the European Union.

References

Abbott, L. F., & Nelson, S. B. (2000). Synaptic plasticity: Taming the beast. *Nature Neuroscience*, *3*, 1178–1183.

Abraham, W. C. (2008). Metaplasiticity: Tuning synapses and networks for plasticity. *Nature Reviews Neuroscience*, *9*, 387–399.

Attias, H. (2003). Planning by probabilistic inference. In *Proc. of the 9th Int. Workshop on Artificial Intelligence and Statistics*. N.p.: Society for Artificial Intelligence and Statistics.

Audibert, J.-Y., Munos, R., & Szepesvari, C. (2007). Tuning bandit problems in stochastic environments. In *Proc. of the 18th International Conference on Algorithmic Learning Theory* (pp. 150–165).

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite time analysis of the multi-armed bandit problem. *Machine Learning*, *47*(2/3), 235–256.

Auer, P., Jaksch, T., & Ortner, R. (2009). Near-optimal regret bounds for reinforcement learning. In D. Koller, D. Schuurmans, Y. Bengio, & L. Bottou (Eds.), *Advances in neural information processing systems, 21* (pp. 89–96). Cambridge, MA: MIT Press.

Bailey, C. H., Giustetto, M., Huang, Y.-Y., Hawkins, R. D., & Kandel, E. R. (2000). Is heterosynaptic modulation essential for stabilizing Hebbian plasticity and memory? *Nature Reviews Neuroscience*, *1*, 11–20.

Bertsekas, D. P., & Tsitsiklis, J. (1996). *Neuro-dynamic programming*. Belmont, MA: Athena Scientific.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. New York: Springer.

Caporale, N., & Dan, Y. (2008). Spike timing-dependent plasticity: A Hebbian learning rule. *Annu Rev Neuroscience*, *31*, 25–46.

Dayan, P., & Abbott, L. F. (2001). *Theoretical neuroscience: Computational and mathematical modeling of neural systems*. Cambridge, MA: MIT Press.

Dayan, P., & Daw, N. (2008). Decision theory, reinforcement learning, and the brain. *Cognitive, Affective, and Behavioral Neuroscience*, *8*, 429–453.

Dayan, P., & Kakade, S. (2001). Explaining away in weight space. In T. K. Leen, T. G. Dietterich, & V. Tresp (Eds.), *Advances in neural information processing systems, 13* (pp. 451–457). Cambridge, MA: MIT Press.

Dayan, P., & Yu, A. (2003). Uncertainty and learning. *IETE Journal of Research*, *49*, 171–182.

Deneve, S. (2008). Bayesian spiking neurons I: Inference. *Neural Computation*, *20*(1), 91–117.

Domingos, P., & Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, *275*(29), 103–130.

Douglas, R. J., & Martin, K. A. (2004). Neuronal circuits of the neocortex. *Annu. Rev. Neurosci.*, *27*, 419–451.

Doya, K. (2002). Metalearning and neuromodulation. *Neural Networks*, *15*, 495–506.

Dragalin, V., Tartakovsky, A., & Veeravalli, V. (1999). Multihypothesis sequential probability ratio tests—Part I: Asymptotic optimality. *IEEE Transactions on Information Theory*, *45*(7), 2448–2461.

Farries, M. A., & Fairhall, A. L. (2007). Reinforcement learning with modulated spike timing-dependent synaptic plasticity. *Journal of Neurophysiology*, *98*, 3648–3665.

Frégnac, Y. (2003). Hebbian synaptic plasticity. In M. A. Arbib (Ed.), *The handbook of brain theory and neural networks* (pp. 515–522). Cambridge, MA: MIT Press.

Georgopoulos, A. P., Schwartz, A. P., & Kettner, R. E. (1986). Neuronal population coding of movement direction. *Science*, *233*, 1416–1419.

Gittins, J. (1979). Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society*, *41*, 148–177.

Gold, J. I., & Shadlen, M. N. (2007). The neural basis of decision making. *Annu. Rev. Neuroscience*, *30*, 535–574.

Griffiths, T., & Tenenbaum, J. (2005). Structure and strength in causal induction. *Cognitive Psychology*, *51*, 334–384.

Gurney, K., & Bogacz, R. (2006). The basal ganglia and cortex implement optimal decision making between alternative actions. *Neural Computation*, *19*, 442–477.

Hahnloser, R. H. R., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., & Seung, H. S. (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, *405*, 947–951.

Hebb, D. O. (1949). *The organization of behavior*. Hoboken, NJ: Wiley.

Ide, J. S., & Cozman, F. G. (2002). Random generation of Bayesian networks. In *Proc. of the 16th Brazilian Symposium on Artificial Intelligence: Advances in Artificial Intelligence* (pp. 366–375). Berlin: Springer-Verlag.

Jensen, F. V., & Nielsen, T. D. (2007). *Bayesian networks and decision graphs* (2nd ed.). Berlin: Springer.

Kearns, M., & Singh, S. (1998). Near-optimal performance for reinforcement learning in polynomial time. In *Proc. of the 15th International Conference on Machine Learning (ICML)* (pp. 260–268). San Francisco: Morgan Kaufmann.

Kononenko, I. (1998). Bayesian neural networks. *Biol. Cybernetics*, *61*, 361–370.

Kschischang, F. R., Frey, B. J., & Loeliger, H. A. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, *47*(2), 498–519.

Lai, T., & Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, *6*, 4–22.

Lansner, A., & Ekeberg, O. (1998). A one-layer feedback artificial neural network with a Bayesian learning rule. *International Journal of Neural Systems*, *1*, 77–87.

Lansner, A., & Holst, A. (1996). A higher order Bayesian neural network with spiking units. *International Journal of Neural Systems*, *7*(2), 115–128.

Legenstein, R., Pecevski, D., & Maass, W. (2008). A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS Computational Biology*, *4*(10), 1–27.

Lohmiller, W., & Slotine, J. J. (1998). Contraction analysis for nonlinear systems. *Automatica*, *34*(6), 683–696.

Maass, W. (2000). On the computational power of winner-take-all. *Neural Computation*, *12*(11), 2519–2536.

Montague, P., Dayan, P., Person, C., & Sejnowski, T. (1995). Bee foraging in uncertain environments using predictive Hebbian learning. *Nature*, *377*, 725–728.

Neapolitan, R. (2004). *Learning Bayesian networks*. Upper Saddle River, NJ: Prentice Hall.

Neftci, E., Chicca, E., Indiveri, G., Slotine, J., & Douglas, R. (2008). Contraction properties of VLSI cooperative competitive neural networks of spiking neurons. In J. C. Platt, D. Koller, Y. Singer, & S. Roweis (Eds.), *Advances in neural information processing systems, 20*. Cambridge, MA: MIT Press.

Nessler, B., Pfeiffer, M., & Maass, W. (2009). Hebbian learning of Bayes optimal decisions. In D. Koller, D. Schuurmans, Y. Bengio, & L. Bottou (Eds.), *Advances in neural information processing systems, 21*. Cambridge, MA: MIT Press.

O'Keefe, J., Burgess, N., Donnett, J., Jeffery, K., & Maguire, E. (1998). Place cells, navigational accuracy, and the human hippocampus. *Philosophical Transactions of the Royal Society of London*, *353*(1373), 1333–1340.

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.

Pouget, A., & Latham, P. (2002). Population codes. In M. A. Arbib (Ed.), *The handbook of brain theory and neural networks* (2nd ed., pp. 893–897). Cambridge, MA: MIT Press.

Rao, R. P. N. (2007). Neural models of Bayesian belief propagation. In K. Doya, S. Ishii, A. Pouget, & R. P. N. Rao (Eds.), *Bayesian brain* (pp. 239–267). Cambridge, MA: MIT Press.

Rescorla, R. A., & Wagner, A. R. (1972). Classical conditioning II. In A. H. Black & W. F., Prokasy (Eds.), *A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement* (pp. 64–99). New York: Appleton-Century-Crofts.

Reynolds, J. N., Hyland, B. I., & Wickens, J. R. (2001). A cellular mechanism of reward-related learning. *Nature*, *413*, 67–70.

Riesenhuber, M., & Poggio, T. (1999). Models of object recognition. *Nature Neuroscience*, *2*, 1019–1025.

Roth, D. (1999). Learning in natural language. In T. Dean (Ed.), *Proc. of the International Joint Conference on Artificial Intelligence* (pp. 898–904). San Francisco: Morgan Kaufmann.

Sakai, Y., Okamoto, H., & Fukai, T. (2006). Computational algorithms and neuronal network models underlying decision processes. *Neural Networks*, *19*(8), 1091–1105.

Sandberg, A., Lansner, A., Petersson, K. M., & Ekeberg, O. (2002). A Bayesian attractor network with incremental learning. *Network: Computation in Neural Systems*, *13*, 179–194.

Schultz, W., Dayan, P., & Montague, P. (1997). A neural substrate of prediction and reward. *Science*, *275*, 1593–1599.

Steimer, A., Maass, W., & Douglas, R. (2009). Belief-propagation in networks of spiking neurons. *Neural Computation*, *21*, 2502–2523.

Sugrue, L. P., Corrado, G. S., & Newsome, W. T. (2004). Matching behavior and the representation of value in the parietal cortex. *Science*, *304*, 1782–1787.

Sugrue, L. P., Corrado, G. S., & Newsome, W. T. (2005). Choosing the greater of two goods: Neural currencies for valuation and decision making. *Nature Reviews Neuroscience*, *6*(5), 363–375.

Sutton, R. S. (1992). Gain adaptation beats least squares. In *Proceedings of the 7th Yale Workshop on Adaptive and Learning Systems* (pp. 161–166). New Haven, CT: Yale University.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Verma, D., & Rao, R. (2006). Goal-based imitation as probabilistic inference over graphical models. In Y. Weiss, B. Schölkopf, & J. Platt (Eds.), *Advances in neural information processing systems, 18* (pp. 1393–1400). Cambridge, MA: MIT Press.

Wald, A., & Wolfowitz, J. (1948). Optimal character of the sequential probability ratio test. *Ann. Math. Statist.*, *19*, 326–339.

Wang, X. J. (2002). Probabilistic decision making by slow reverberation in cortical circuits. *Neuron*, *36*, 955–968.

Yang, T., & Shadlen, M. N. (2007). Probabilistic reasoning by neurons. *Nature*, *447*, 1075–1080.

Yu, A., & Dayan, P. (2003). Expected and unexpected uncertainty: ACh and NE in the neocortex. In S. Becker, S. Thrün, & K. Obermayer (Eds.), *Advances in neural information processing systems, 15* (pp. 157–164). Cambridge, MA: MIT Press.

Yuille, A. L. (2006). Augmented Rescorla-Wagner and maximum likelihood estimation. In Y. Weiss, B. Schölkopf, & J. Platt (Eds.), *Advances in neural information processing systems, 18* (pp. 1561–1568). Cambridge, MA: MIT Press.

Yuille, A. L., & Geiger, D. (2003). Winner-take-all networks. In M. A. Arbib (Ed.), *The handbook of brain theory and neural networks* (pp. 1228–1231). Cambridge, MA: MIT Press.

Received March 11, 2009; accepted September 10, 2009.