Journal of Machine Learning Research 5 (2004) 1063–1088 Submitted 3/02; Revised 1/04; Published 8/04

Reinforcement Learning with Factored States and Actions

Brian Sallans SALLANS@OEFAI.AT
Austrian Research Institute for Artificial Intelligence
Freyung 6/6
A-1010 Vienna, Austria

Geoffrey E. Hinton HINTON@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
Toronto, Canada

Editor: Sridhar Mahadevan

Abstract

A novel approximation method is presented for approximating the value function and selecting good actions for Markov decision processes with large state and action spaces. The method approximates state-action values as negative free energies in an undirected graphical model called a product of experts. The model parameters can be learned efficiently because values and derivatives can be efficiently computed for a product of experts. Actions can be found even in large factored action spaces by the use of Markov chain Monte Carlo sampling. Simulation results show that the product of experts approximation can be used to solve large problems. In one simulation it is used to find actions in action spaces of size 2^{40}.

Keywords: product of experts, Boltzmann machine, reinforcement learning, factored actions

1. Introduction

An agent must be able to deal with high-dimensional and uncertain states and actions in order to operate in a complex environment. In this paper we focus on two related problems: estimating the value of a state-action pair in large state and action spaces; and selecting good actions given these estimates. Our approach is to borrow techniques from the graphical modeling literature and apply them to the problems of value estimation and action selection.

Inferring the state of the agent's environment from noisy observations has been a popular subject of study in the engineering, artificial intelligence and machine learning communities. One formalism is the graphical model (Cowell et al., 1999). A graphical model represents the distribution of observed data with a probabilistic model. The graphical representation of the model indicates which variables can be assumed to be conditionally independent. Given observations of some variables, inferring the distribution over unobserved (or hidden) variables is of paramount importance in using and learning the parameters of these models. Exact and approximate inference algorithms have been and still are intensely studied in the graphical models and engineering literatures (Kalman, 1960; Neal, 1993; Jordan et al., 1999; Cowell et al., 1999).

Acting on certain or uncertain information has been studied in a different body of literature. Reinforcement learning involves learning how to act so as to maximize a reward signal given samples

©2004 Brian Sallans and Geoffrey E. Hinton.

of sequences of state-action pairs and rewards from the environment (Sutton and Barto, 1998). It has been closely linked to Markov decision processes and stochastic dynamic programming (see for example Sutton, 1988; Bertsekas and Tsitsiklis, 1996). There has been work on reinforcement learning with large state spaces, state uncertainty and partial observability (see for example Bertsekas and Tsitsiklis, 1996; Jaakkola et al., 1995). In particular there are exact dynamic programming methods for solving fully and partially observable Markov decision processes (see Lovejoy, 1991, for an overview). There are also approximate methods for dealing with real-valued state and action variables (see for example Baird and Klopf, 1993; Sutton, 1996; Santamaria et al., 1998).

Recently, techniques from the graphical models literature have started to come together with those from the planning and reinforcement-learning community. The result has been new algorithms and methods for learning about decision processes and making decisions under uncertainty in complex and noisy environments (see for example Boutilier and Poole, 1996; McAllester and Singh, 1999; Thrun, 2000; Sallans, 2000; Rodriguez et al., 2000; Poupart and Boutilier, 2001).

In this article, we propose to make use of techniques from graphical models and approximate inference to approximate the values of and select actions for large Markov decision processes. The value function approximator is based on an undirected graphical model called a product of experts (PoE). The value of a state-action pair is modeled as the negative free energy of the state-action under the product model.

Computing the free energy is tractable for a product of experts model. However, computing the resulting distribution over actions is not. Given a value function expressed as a product of experts, actions can be found by Markov chain Monte Carlo (MCMC) sampling. As with any sampling scheme, it is possible that action sampling will perform poorly, especially as the action space becomes large. There are no theoretical guarantees as to the effectiveness of sampling for short periods of time.

The advantage of using MCMC sampling for action selection is that there has been a concerted effort put into making sampling methods work well in large state (or in our case, action) spaces. It is also possible to use this technique to approximate value functions over real-valued state and action spaces, or mixtures of discrete and real-valued variables, and to make use of MCMC sampling methods designed for continuous spaces to do action selection. It is an empirical question whether or not action sampling works well for specific problems.

Our technique uses methods from reinforcement learning and from products of experts modeling. We therefore include a short review of the Markov decision process formalism, reinforcement learning, and products of experts. For clarity, we will focus on one particular kind of product of experts model: the restricted Boltzmann machine.

We then describe the method, which uses a product of experts network as a novel value function approximator. We demonstrate the properties of the PoE approximation on two tasks, including an action-selection task with a 40-bit action space. We conclude with some discussion of the approximation method and possible future research.

2. Markov Decision Processes

An agent interacting with the environment can be modeled as a Markov decision process (MDP) (Bellman, 1957b). The task of learning which action to perform based on reward is formalized by reinforcement learning. Reinforcement learning in MDPs is a much studied problem. See for example Sutton and Barto (1998).

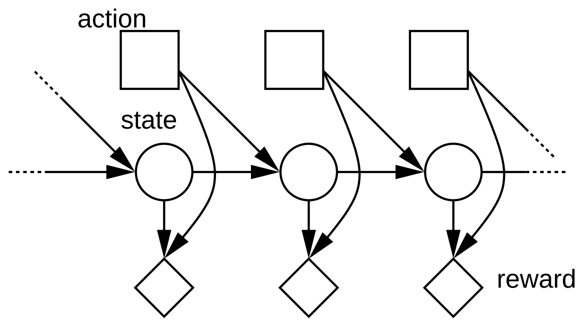

Figure 1: A Markov decision process. Circles indicate visible variables, and squares indicate actions. The state is dependent on the previous state and action, and the reward depends on the current state and action.

If the sets of states and actions are finite, then the problem is called a finite MDP. Much of the theoretical work done in reinforcement learning has focused on the finite case, and we focus on finite MDPs in this article.

Formally, an MDP consists of

- A set of states $\mathcal{S}$, and actions $\mathcal{A}$,
- An initial state s^0 or distribution $P(s^0)$,
- A transition distribution $P(s^{t+1}|s^t, a^t)$, $s^t, s^{t+1} \in \mathcal{S}$, $a^t \in \mathcal{A}$, and
- A reward distribution $P(r^t|s^t, a^t)$, $s^t \in \mathcal{S}$, $r^t \in \mathbb{R}$, $a^t \in \mathcal{A}$.

In the above, t indexes the time step, which ranges over a discrete set of points in time. We will denote the transition probability $Pr(s^{t+1} = j\,|s^t = i, a^t = a)$ by $\mathbf{P}_{ij}(a)$. We will also denote the expected immediate reward received by executing action a in state i by

$$r_i(a) = \left\langle r^t \,\middle|\, s^t = i, a^t = a \right\rangle_{P(r^t|s^t,a^t)},$$

where $\langle\cdot\rangle_P$ denotes expectation with respect to distribution P. Bold-face text denotes vectors or matrices.

The goal of solving an MDP is to find a *policy* which maximizes the total expected reward received over the course of the task. A policy tells the learning agent what action to take for each possible state. It is a mapping π from states to actions or distributions over actions. We will focus on *stationary* policies, in which the same mapping is used at every point in time.

The expected discounted *return* for a policy π is defined as the sum of discounted rewards that is expected when following policy π:

$$\left\langle R^t \right\rangle_\pi \;=\; \left\langle r^t + \gamma r^{t+1} + \gamma^2 r^{t+2} + \ldots \right\rangle_\pi \;=\; \left\langle \sum_{k=0}^{\infty} \gamma^k r^{t+k} \right\rangle_\pi,$$

where t is the current time, and $\pi(s,a)$ is the probability of selecting action a in state s. Note that the discounting is required to ensure that the sum of infinite (discounted) rewards is finite, so that the quantity to be optimized is well-defined; the discount factor $\gamma \in [0,1)$. The expectation is taken with respect to the policy π, the initial state distribution $P(s^0)$, the transition distribution $\mathbf{P}_{ij}(a)$, and the reward distribution $P(r|s,a)$.

To solve an MDP, we must find a policy that produces the greatest expected return. With knowledge of transition probabilities $\mathbf{P}_{ij}(a)$ and expected immediate rewards $r_i(a)$, and given a stochastic policy π, we can calculate the expected discounted return after taking the action a from the current state s and following policy π thereafter:

$$\begin{aligned} Q^\pi(s,a) &= \left\langle \sum_{k=0}^{\infty} \gamma^k r^{t+k} \, \middle| s^t = s, a^t = a \right\rangle_\pi \\ &= \left\langle r^t + \gamma \sum_{k=0}^{\infty} \gamma^k r^{t+k+1} \, \middle| s^t = s, a^t = a \right\rangle_\pi \\ &= \sum_j \mathbf{P}_{sj}(a) \left[r_s(a) + \sum_b \pi(j,b) \gamma Q^\pi(j,b) \right]. \end{aligned} \tag{1}$$

Here t denotes the current time. The function Q^π is called the *action-value function* for policy π. The action-value function tells the agent the expected return that can be achieved by starting from any state, executing an action, and then following policy π.

Equation 1 is often called the Bellman equations for Q^π. It is a set of $|\mathcal{S}| \times |\mathcal{A}|$ linear equations (one for each state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$). The set of coupled equations can be solved for the unknown values $Q^\pi(s,a)$. In particular, given some arbitrary initialization Q^π_0, we can use the following iterative update:

$$Q^\pi_{k+1}(s,a) = \sum_j \mathbf{P}_{sj}(a) \left[r_s(a) + \sum_b \pi(j,b) \gamma Q^\pi_k(j,b) \right] \tag{2}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$. The iteration converges to the unique fixed-point Q^π as $k \to \infty$. This technique is called iterative policy evaluation.

The class of MDPs is a restricted but important class of problems. By assuming that a problem is Markov, we can ignore the history of the process, and thereby prevent an exponential increase in the size of the domain of the policy (Howard, 1960). The Markov assumption underlies a large proportion of control theory, machine learning and signal processing including Kalman filters (Kalman, 1960), hidden Markov models (HMMs) (Rabiner and Juang, 1986), two-time-slice dynamic Bayesian networks (Dean and Kanazawa, 1989), dynamic programming (DP) (Bellman, 1957a) and temporal difference learning (TD) (Sutton, 1988).

When the states or actions are composed of sets of variables, we will refer to them as “factored” states or actions. It is common for problems with large state or action spaces to have states and actions expressed in a “factored” form. There has been a lot of research and discussion about the problems of dealing with large state spaces represented in factored form (see for example Bertsekas and Tsitsiklis, 1996). There has been comparatively little on dealing with large action spaces or factored actions (Dean et al., 1998; Meuleau et al., 1998; Peshkin et al., 1999; Guestrin et al., 2002). In practice action spaces tend to be very large. For example consider the activation of large numbers of muscles, or the simultaneous actions of all of the players on a football field. The state or action space could also be continuous (Baird and Klopf, 1993; Santamaria et al., 1998).

3. Temporal Difference Learning

Temporal difference (TD) learning (Sutton, 1988) addresses the problem of predicting time-delayed rewards. We can view TD as performing approximate dynamic programming to compute future reward. Because they use a value function as an estimate of future performance instead of sampled rewards, TD algorithms trade off bias against variance in estimates of future reward. The nature of the approximation, this tradeoff, and the convergence of TD algorithms have been the subjects of much study.

Temporal difference algorithms can be used to estimate the value of states and actions. TD techniques update the value estimate for states and actions as they are visited and executed. Backing up the values of states only as they are visited gives rise to a number of TD update rules. The SARSA algorithm computes the action-value function of the current policy (Rummery and Niranjan, 1994; Sutton, 1996):

$$Q(s^t,a^t) \leftarrow (1-\kappa)Q(s^t,a^t) + \kappa\left[r^t + \gamma Q(s^{t+1},a^{t+1})\right], \tag{3}$$

where κ is a learning rate.

This update can be viewed as a Monte Carlo approximation to the update from Eq.(2). The name derives from the fact that the update depends on the set of values $\{s^t,\ a^t,\ r^t,\ s^{t+1},\ a^{t+1}\}$. SARSA computes the expected return conditioned on a state-action pair. The update is designed to move the estimated value function closer to a "bootstrapped" Monte Carlo estimate. If κ is reduced over time in the appropriate manner, and all states continue to be visited an infinite number of times, then this algorithm will converge to the value function of the current policy (Singh et al., 2000).

Given the state-action value function, a greedy policy with respect to the value function can be found by maximizing over possible actions in each state:

$$\pi(s) = \operatorname*{argmax}_a Q^\pi(s,a).$$

Note that this involves an explicit maximization over actions. When the action space is large or continuous, this maximization will become difficult.

The optimal value function and policy can be found using SARSA, by combining value function estimation (Eq.3) with policies which become greedy with respect to the value function, in the limit of infinite time (i.e. an infinite number of value function updates, with all states and actions continuing to be visited/executed). See Singh et al. (2000) for a proof of convergence and conditions on the policies.

4. Function Approximation

In many problems there are too many states and actions to represent the action-value function as a state-action table. One alternative is to use function approximation. If the approximation is differentiable with respect to its parameters, the parameters can be learned by trying to minimize the TD error. The TD error is defined as

$$E_{\mathrm{TD}}(\mathbf{s}^t,\mathbf{a}^t) = \left[r^t + \gamma Q(\mathbf{s}^{t+1},\mathbf{a}^{t+1})\right] - Q(\mathbf{s}^t,\mathbf{a}^t). \tag{4}$$

Consider an approximate value function $Q(\mathbf{s},\mathbf{a};\theta)$ parameterized by parameter θ. The update rule for the parameter θ is given by

$$\Delta\theta = \lambda E_{\mathrm{TD}}(\mathbf{s},\mathbf{a})\nabla_\theta Q(\mathbf{s},\mathbf{a};\theta), \tag{5}$$

where λ is a learning rate. This is the approach taken by Bertsekas and Tsitsiklis (1996) among others. Although this approach can work in some cases, there are in general no guarantees of convergence to a specific approximation, or guarantees of the accuracy of the approximation if it does converge.

5. Boltzmann Machines and Products of Experts

We will use a product of experts model to approximate the values of states and actions in a Markov decision process (Hinton, 1999, 2002). Products of experts are probabilistic models that combine simpler models by multiplying their distributions together.

In this section we will focus on a particular kind of product of experts, called a restricted Boltzmann machine (Smolensky, 1986; Freund and Haussler, 1992; Hinton, 2002). This case is interesting because inference and learning in this model has been intensely studied (Ackley et al., 1985; Hinton and Sejnowski, 1986; Smolensky, 1986; Freund and Haussler, 1992; Hinton, 2002), and the binary values are relatively easy to interpret.

Boltzmann machines are undirected models. That means that the model specifies joint probabilities, rather than conditional probabilities. Directed graphical models, for example Bayesian networks, have also been used in conjunction with value function approximation, where the Bayesian network encodes a compact model of the environment (see for example Boutilier and Poole, 1996; Boutilier et al., 2000; Rodriguez et al., 2000; Sallans, 2000; Thrun, 2000). They have also been used to directly encode utilities (Boutilier et al., 1999, 2001).

The free energy of a directed model could also be used to encode an approximate value function. However, unlike with product models, the computation of the free energy and its derivatives is generally intractable for directed models. This is because inference in the model is not tractable. Research in approximate inference techniques for directed models is an important area of current research.

In a directed model, it is also intractable to compute the conditional distribution over actions given states, as with the product of experts models. It would be possible to combine an approximate inference technique with a directed model to approximate the value, and also use the approximate inference technique to sample actions from the network. The result would have the flavor of an actor-critic network, where the free energy of the directed model plays the role of the critic, and the approximate inference technique plays the role of the actor. The advantage would be that the distribution over actions could be evaluated, rather than just sampled from as with the product of experts. The disadvantage is that we would use an approximation not just for action selection, but also to compute the free energy and its derivatives.

We begin the next section with a discussion of the general Boltzmann machine (Ackley et al., 1985) and review some necessary concepts such as energy and free energy, the Boltzmann distribution, and Markov chain Monte Carlo sampling. We then discuss the restricted Boltzmann machine. For completeness, we include some derivations, but defer them to Appendix A. The reader is directed to Hertz et al. (1991), chapter 7, for a more in-depth introduction to the Boltzmann machine. Finally, we discuss more general products of experts.

5.1 Boltzmann Machines

A Boltzmann machine is an undirected graphical model. The nodes represent binary random variables that have values of 1 or 0 and the weighted edges represent pairwise symmetric interactions between the variables.[1] The nodes are usually divided into two disjoint subsets, the "visible" variables, $\mathbf{V}$ and the "hidden" variables, $\mathbf{H}$. An assignment of binary values to the visible or hidden variables will be denoted by $\mathbf{v}$ or $\mathbf{h}$ and the binary value of an individual visible or hidden variable, V_i or H_k, will be denoted by v_i or h_k. The symmetric weight between node i and node k is w_{ik}. In a general Boltzmann machine, weights can occur between any pair of nodes.

The weights determine the "energy" of every possible joint configuration of the visible and hidden variables:

$$E(\mathbf{v},\mathbf{h}) = -\sum_{i,k} w_{ik} v_i h_k - \sum_{i<j} w_{ij} v_i v_j - \sum_{k<m} w_{km} h_k h_m,$$

where i and j are indices over visible variables and k and m are indices over hidden variables. The energies of the joint configurations determine their equilibrium probabilities via the Boltzmann distribution:

$$P(\mathbf{v},\mathbf{h}) = \frac{\exp(-E(\mathbf{v},\mathbf{h}))}{\sum_{\widehat{\mathbf{v}},\widehat{\mathbf{h}}} \exp(-E(\widehat{\mathbf{v}},\widehat{\mathbf{h}}))},$$

where $\widehat{\mathbf{v}},\widehat{\mathbf{h}}$ indexes all joint configurations of the visible and hidden variables.

The probability distribution over the visible variables can be obtained by summing over all possible configurations of the hidden variables:

$$\exp(-F(\mathbf{v})) = \sum_{\mathbf{h}} \exp(-E(\mathbf{v},\mathbf{h})), \tag{6}$$

$$P(\mathbf{v}) = \frac{\exp(-F(\mathbf{v}))}{\sum_{\widehat{\mathbf{v}}} \exp(-F(\widehat{\mathbf{v}}))}.$$

$F(\mathbf{v})$ is called the "equilibrium free energy" of $\mathbf{v}$. It is the minimum of the "variational free energy" of $\mathbf{v}$ which can be expressed as an expected energy minus an entropy:

$$F_q(\mathbf{v}) = \sum_{\mathbf{h}} q(\mathbf{h}) E(\mathbf{v},\mathbf{h}) + \sum_{\mathbf{h}} q(\mathbf{h}) \log q(\mathbf{h}), \tag{7}$$

where q is any distribution over all possible configurations of the hidden units. To make the first term in Eq.(7) low, the distribution q should put a lot of mass on hidden configurations for which $E(\mathbf{v},\mathbf{h})$ is low, but to make the second term low, the q distribution should have high entropy. The optimal trade-off between these two terms is the Boltzmann distribution in which $P(\mathbf{h}|\mathbf{v}) \propto \exp(-E(\mathbf{v},\mathbf{h}))$:

$$P(\mathbf{h}|\mathbf{v}) = \frac{\exp(-E(\mathbf{v},\mathbf{h}))}{\sum_{\widehat{\mathbf{h}}} \exp(-E(\mathbf{v},\widehat{\mathbf{h}}))}. \tag{8}$$

This is the posterior distribution over the hidden variables, given the visible variables. Using this distribution, the variational free energy defined by Eq.(7) is equal to the equilibrium free energy in Eq.(6):

$$F(\mathbf{v}) = \sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v}) E(\mathbf{v},\mathbf{h}) + \sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v}) \log P(\mathbf{h}|\mathbf{v}). \tag{9}$$

1. We will assume that the variables can take on values of 1 or 0. Alternatively, the Boltzmann machine can be formulated with values of ± 1. We also omit biases on variables, which can be incorporated into the weights by adding a visible variable which always has a value of one. Weights from this "always on" variable act as biases to the other variables.

In Appendix A, we show that the equilibrium free energy can be written either as in Eq.(6) or as in Eq.(9) (see Appendix A, Eq.11).

One important property of Boltzmann machines is that, with enough hidden variables, a Boltzmann machine with finite weights is capable of representing any "soft" distribution over the visible variables, where "soft" means that the distribution does not contain any probabilities of 1 or 0. Another important property is that there is a simple, linear relationship between $F(\mathbf{v})$ and each of the weights in the network (Ackley et al., 1985). For a weight between a visible and a hidden unit this relationship is

$$\frac{\partial F(\mathbf{v})}{\partial w_{ik}} = -v_i \langle h_k \rangle_{P(h_k|\mathbf{v})},$$

where the angle brackets denote the expectation of h_k under the distribution $P(h_k|\mathbf{v})$. At first sight, it is surprising that this derivative is so simple because changing w_{ik} will clearly change the equilibrium distribution over the hidden configurations. However, to first order, the changes in this equilibrium distribution have no effect on $F(\mathbf{v})$ because the equilibrium distribution $P(\mathbf{h}|\mathbf{v})$ is the distribution for which $F(\mathbf{v})$ is minimal, so the derivatives of $F(\mathbf{v})$ *w.r.t.* the probabilities in the distribution are all zero (see Appendix A).

For a general Boltzmann machine, it is computationally intractable to compute the equilibrium distribution over the hidden variables given a particular configuration, $\mathbf{v}$, of the visible variables. (Cooper, 1990). However, values can be sampled from the equilibrium distribution by using a Markov chain Monte Carlo method. Once the Markov chain reaches equilibrium (in other words, all information about the initialization has been lost), hidden configurations are sampled according to the Boltzmann distribution (Eq.8).

One sampling method is called the Gibbs sampler (Geman and Geman, 1984). Given a fixed configuration $\mathbf{v}$ of the visible variables, the Gibbs sampler proceeds as follows.

1. Initialize all the hidden variables with arbitrary binary values:
   $h_1^0, \ldots h_k^0, \ldots, h_K^0$.

2. Repeat the following until convergence:
   In each iteration $t = 1, 2, \ldots$, and for each random variable h_k:

   (a) Compute the energy $E_1 = E(\mathbf{v}, h_k = 1, \{h_m = h_m^{t-1} : m \neq k\})$.

   (b) Compute the energy $E_0 = E(\mathbf{v}, h_k = 0, \{h_m = h_m^{t-1} : m \neq k\})$.

   (c) Set $h_k = 1$ with probability $\exp(-E_1)/(\exp(-E_0) + \exp(-E_1))$
   and set $h_k = 0$ with probability $\exp(-E_0)/(\exp(-E_0) + \exp(-E_1))$.

This procedure should be repeated until the Markov chain converges to its stationary distribution which is given by Eq.(8). Assessing whether or not convergence has occurred is not trivial, and will not be discussed here.

The Gibbs sampler is only one possible sampling technique. There are many others, including the Metropolis-Hastings algorithm (Metropolis et al., 1953), and hybrid Monte Carlo algorithms (Duane et al., 1987). The reader is directed to Neal (1993) for a review of Markov chain Monte Carlo methods.

The difficulty of computing the posterior over the hidden variables in a general Boltzmann machine makes it unsuitable for our purposes, because it means that the free energy of a visible vector can not be easily evaluated. However, a restricted class of Boltzmann machines (Smolensky, 1986; Freund and Haussler, 1992; Hinton, 1999, 2002) is more useful to us. In a restricted Boltzmann machine there are no hidden-hidden or visible-visible connections but any hidden-visible connection is allowed. The connectivity of a restricted Boltzmann machine therefore forms a bipartite graph.

In a restricted Boltzmann machine the posterior distribution over the hidden variables factors into the product of the posteriors over each of the individual hidden units (Freund and Haussler, 1992) (see Appendix A for a derivation):

$$P(\mathbf{h}|\mathbf{v}) = \prod_k P(h_k|\mathbf{v}).$$

The posterior over hidden variables can therefore be computed efficiently, because each individual hidden-unit posterior is tractable:

$$P(h_k = 1|\mathbf{v}) = \frac{1}{1+\exp(-\sum_i v_i w_{ik})}.$$

This is crucial, because it allows for efficient computation of the equilibrium free energy $F(\mathbf{v})$ and of its derivatives with respect to the weights.

After a Boltzmann machine has learned to model a distribution over the visible variables, it can be used to complete a visible vector that is only partially specified. If, for example, one half of the visible vector represents a state and the other half represents an action, the Boltzmann machine defines a probability distribution over actions for each given state. Moreover, Gibbs sampling in the space of actions can be used to pick actions according to this distribution (see below).

In summary, restricted Boltzmann machines have the properties that we require for using negative free energies to approximate Q-values: The free energy and its derivatives can be efficiently computed; and, given a state, Gibbs sampling can be used to sample actions according a Boltzmann distribution in the free energy.

5.2 Products of Experts

Restricted Boltzmann machines are only one example of a class of models called products of experts (Hinton, 2002). Products of experts combine simple probabilistic models by multiplying their probability distributions. In the case of restricted Boltzmann machines, the individual "experts" are stochastic binary hidden variables. Products of experts share the useful properties discussed above for restricted Boltzmann machines. A product of experts model defines a free energy whose value and its derivative can be efficiently computed; and instantiations of the random variables can be sampled according to the Boltzmann distribution. Products of experts can include more complex individual experts than binary variables. Examples of product models include products of hidden Markov models (Brown and Hinton, 2001) and products of Gaussian mixtures (Hinton, 2002). Notice that in each case the individual experts (hidden Markov models and Gaussian mixtures) are themselves tractable, meaning that the posterior distribution over an individual expert's hidden variables, and derivatives of the free energy with respect to the parameters, can be computed tractably. This is all that is required for the entire product model to be tractable. Although we focus on restricted Boltzmann machines in this work, other products of experts models could also be used, for example, to model real-valued variables.

6. The Product of Experts as Function Approximator

Consider a product of experts model, where the visible variables are state and action variables. The free energy allows a PoE to act as a function approximator, in the following sense. For any input (instantiation of the visible variables), the output of the function approximator is taken to be the free energy. With no hidden variables, the output is simply the energy. For a Boltzmann machine, this is similar to a linear neural network with no hidden units. With hidden variables, the Boltzmann machine is similar to a neural network with hidden units. However, unlike traditional neural networks, having probabilistic semantics attached to the model allows us to (at least approximately) sample variables according to the Boltzmann distribution. This is ideal for value function approximation, because we can sample actions according to a Boltzmann exploration policy, conditioned on settings of the state variables, even in large action spaces for which actually computing the Boltzmann distribution would be intractable. To do this, we have to create a correspondence between the value of a state-action pair, and its negative free energy under the Boltzmann machine model.

We create this correspondence using the parameter update rule for reinforcement learning with function approximation (Eq.5). The parameters of the PoE model are updated to try to reduce the temporal-difference error (Eq.4). By reducing the temporal-difference error, we make the value approximated by the product of experts closer to the correct value.

Once the negative free energy under the PoE model approximates the value, we use MCMC sampling to select actions. After training, the probability of sampling an action from the product of experts while holding the state fixed is given by the Boltzmann distribution:

$$P(\mathbf{a}|\mathbf{s}) = \frac{e^{-F(\mathbf{s},\mathbf{a})/T}}{Z} \approx \frac{e^{Q(\mathbf{s},\mathbf{a})/T}}{Z},$$

where Z is a normalizing constant, and T is the exploration temperature. Samples can be selected at a particular exploration temperature by dividing the free energy by this temperature.

Intuitively, good actions will become more probable under the model, and bad actions will become less probable under the model. Although finding optimal actions would still be difficult for large problems, selecting an action with a probability that is approximately the probability under the Boltzmann distribution can normally be done with a small number of iterations of MCMC sampling (and could include simulated annealing). In principle, if we let the MCMC sampling converge to the equilibrium distribution, we could draw unbiased samples from the Boltzmann exploration policy at a given temperature. In particular we can select actions according to a Boltzmann exploration policy that may be intractable to compute explicitly, because normalization would require summing over an exponential number of actions. In practice, we only sample for a short period of time. It should be noted that this "brief" sampling comes with no performance guarantees, and may be problematic in large action spaces. However, we can easily incorporate improvements in sampling techniques to improve performance in large discrete and real-valued action spaces.

6.1 Restricted Boltzmann Machines

Here we detail the approximation architecture for the specific example of a restricted Boltzmann machine. We approximate the value function of an MDP with the negative free energy of the restricted Boltzmann machine (Eq.6). The state and action variables will be assumed to be discrete, and will be represented by the visible binary variables of the restricted Boltzmann machine.

In the following section, the number of binary state variables will be denoted by N; the number of binary action variables by M; and the number of hidden variables by K. We will represent a discrete multinomial state or action variable of arity J by using a "one-of-J" set of binary variables which are constrained so that exactly one of them is unity, and the rest are zero.

We will use Gibbs sampling to select actions. To take the multinomial restriction into account, the sampling method must be modified. Specifically, instead of sampling each variable in sequence, we will sample simultaneously over a group of J variables that represents a multinomial variable of arity J. This is done by first computing the energy of each instantiation where one of the group takes on a value of unity, and the others are zero. Let F_i be the free energy of the instantiation where $s_i = 1$. This instantiation is selected as the new sample with probability $e^{-F_i}/[\sum_j e^{-F_j}]$.

The restricted Boltzmann machine is shown in Figure 2(a). We use s_i to denote the i^{th} state variable and a_j to denote the j^{th} action variable. We will denote the binary hidden variables by h_k. Weights between hidden and state variables will be denoted w_{ik}, and weights between hidden and action variables will be denoted u_{jk} (Figure 2 (b)).

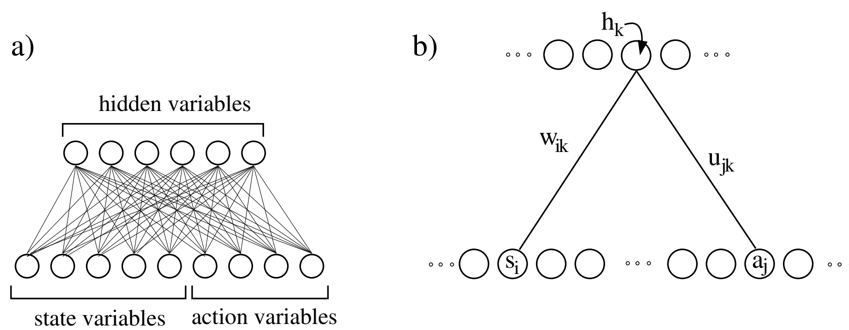

Figure 2: a) The restricted Boltzmann machine. The estimated action-value of a setting of the state and action variables is found by holding these variables fixed and computing the negative free energy of the model. Actions are selected by holding the state variables fixed and sampling from the action variables.
b) The state variables are denoted s_i, the actions a_j and the hidden variables h_k. A hidden-state weight is denoted by w_{ik} and a hidden-action weight by u_{jk}.

In the following, keep in mind that state variables are always held fixed, and the actions are always sampled such that any one-of-J multinomial restrictions are respected. Given these restrictions, we can ignore the fact that the binary vector may represent the values of a set of multinomial variables. The representation of the free energy is the same as in the binary case.[2]

For a state $\mathbf{s} = \{s_i : i \in \{1,...,N\}\}$ and an action $\mathbf{a} = \{a_j : j \in \{1,...,M\}\}$, the free energy is given by Eq.(6), restated here in terms of state, action, and hidden variables:

$$\begin{aligned} F(\mathbf{s},\mathbf{a}) \quad = \quad & -\sum_{k=1}^{K}\left(\sum_{i=1}^{N}(w_{ik}s_i\langle h_k\rangle)+\sum_{j=1}^{M}(u_{jk}a_j\langle h_k\rangle)\right) \\ & +\sum_{k=1}^{K}\langle h_k\rangle\log\langle h_k\rangle+(1-\langle h_k\rangle)\log(1-\langle h_k\rangle). \end{aligned} \tag{10}$$

2. This is equivalent to the Potts multinomial model formulation (Potts, 1952).

The expected value of the hidden variable $\langle h_k \rangle$ is given by

$$\langle h_k \rangle = \sigma\left(\sum_{i=1}^{N} w_{ik} s_i + \sum_{j=1}^{M} u_{jk} a_j\right),$$

where $\sigma(x) = 1/(1+e^{-x})$ denotes the logistic function. The first line of Eq.(10) correspond to an expected energy, and the second to the negative entropy of the distribution over the hidden variables given the data. The value of a state-action pair is approximated by the negative free energy

$$\widehat{Q}(\mathbf{s},\mathbf{a}) = -F(\mathbf{s},\mathbf{a}).$$

6.2 Learning Model Parameters

The model parameters are adjusted so that the negative free energy of a state-action pair under the product model approximates its action-value. We will use the temporal difference update rule SARSA (Eq.3). The temporal-difference error quantifies the inconsistency between the value of a state-action pair and the discounted value of the next state-action pair, taking the immediate reinforcement into account.

The SARSA update is a parameter update rule where the target for input $(\mathbf{s}^t,\mathbf{a}^t)$ is $r^t + \gamma\widehat{Q}(\mathbf{s}^{t+1},\mathbf{a}^{t+1})$. The update for w_{ik} is given by

$$\Delta\, w_{ik} \;\propto\; \left(r^t + \gamma\widehat{Q}(\mathbf{s}^{t+1},\mathbf{a}^{t+1}) - \widehat{Q}(\mathbf{s}^t,\mathbf{a}^t)\right) s_i^t \left\langle h_k^t \right\rangle.$$

The other weights are updated similarly:

$$\Delta\, u_{jk} \;\propto\; \left(r^t + \gamma\widehat{Q}(\mathbf{s}^{t+1},\mathbf{a}^{t+1}) - \widehat{Q}(\mathbf{s}^t,\mathbf{a}^t)\right) a_j^t \left\langle h_k^t \right\rangle.$$

This is found by plugging the derivative of the free energy with respect to a parameter into the update rule (Eq.5). Although there is no proof of convergence in general for this learning rule, it can work well in practice even though it ignores the effect of changes in parameters on $\widehat{Q}(\mathbf{s}^{t+1},\mathbf{a}^{t+1})$. It is possible to derive update rules that use the actual gradient. See for example Baird and Moore (1999).

6.3 Exploration

Given that we can select actions according to their value, we still have to decide on an exploration strategy. One common action selection scheme is Boltzmann exploration. The probability of selecting an action is proportional to $e^{\widehat{Q}(s,a)/T}$. It can move from exploration to exploitation by adjusting the "temperature" parameter T. This is ideal for our product of experts representation, because it is natural to sample actions according to this distribution.

Another possible selection scheme is ε-greedy, where the optimal action is selected with probability $(1-\varepsilon)$ and a random action is selected with probability ε. The exploration probability ε can be reduced over time, to move the learner from exploration to exploitation.

If the SARSA update rule is used with Boltzmann exploration then samples from the Boltzmann distribution at the current temperature are sufficient. This is what we do in our experimental section. If ε-greedy is used we must also evaluate $\max_{\mathbf{a}} \widehat{Q}(\mathbf{s},\mathbf{a})$. This can be approximated by sampling at a low temperature. To improve the approximation, the temperature can be initialized to a high value and lowered during the course of sampling.

7. Simulation Results

To test the approximation we introduce two tasks: the large-action task and the blockers task. The former involves no delayed rewards, and is designed to test action sampling in a large action space. The latter is smaller, but tests learning with delayed reward. We compare performance against two competing algorithms: a direct policy algorithm and a feed-forward neural network with simulated annealing action optimization.

First, we implemented the direct policy algorithm of Peshkin et al. (2000). This algorithm is designed to learn policies for MDPs on factored state and action spaces. To parameterize the policy, we used a feed-forward neural network with one hidden layer. The number of hidden units was chosen to match the number of hidden variables in the competing restricted Boltzmann machine. The output layer of the neural network consisted of a softmax unit for each action variable, which gave the probability of executing each value for that action variable. For example, if an action variable has four possible values, then there are separate inputs (weights and activations) entering the output unit for each of the four possible values. The output unit produces four normalized probabilities by first exponentiating the values, and then normalizing by the sum of the four exponentiated values.

The parameterized policy is therefore factored. In other words, each action variable in the collective action is selected independently, given the probabilities expressed by the softmax output units. However, the hidden layer of the network allows the policy to ”co-ordinate” these probabilities conditioned on the state.

Second, we implemented an action-value function approximation using a feed-forward neural network with one hidden layer (Bertsekas and Tsitsiklis, 1996). The output of this network was a linear unit which gave the estimated value for the state and action presented on the input units. We used the SARSA algorithm to modify the parameters of the network (Eq.5). The gradient was computed using error backpropagation (Rumelhart et al., 1986). We used a greedy-epsilon exploration strategy, where the optimal action was approximated by simulated annealing. The number of iterations of simulated annealing was matched to the number of iterations of sampling used to select actions from the restricted Boltzmann machine, and the number of hidden units was matched to the number of hidden variables in the corresponding restricted Boltzmann machine.

7.1 The Large-Action Task

This task is designed to test value representation and action selection in a large action space, and is not designed to test temporal credit assignment. The large-action task has only immediate rewards. The state at each point in time is selected independent of previous states. The update rules were therefore used with the discount factor γ set to zero.

Consider a version of the task with an N-bit action. The task is generated as follows: Some small number of state-action pairs are randomly selected. We will call these “key” pairs. During execution, a state is chosen at random, and an action is selected by the learner. A reward is then generated by finding the key state closest to the current state (in Hamming distance). The reward received by the learner is equal to the number of bits that match between the key-action for this key state and the current action. So if the agent selects the key action it receives the maximum reward of N. The reward for any other action is found by subtracting the number of incorrect bits from N. The task (for $N = 5$) is illustrated in Figure 3.

A restricted Boltzmann machine with 13 hidden variables was trained on an instantiation of the large action task with an 12-bit state space and a 40-bit action space. Thirteen key states were

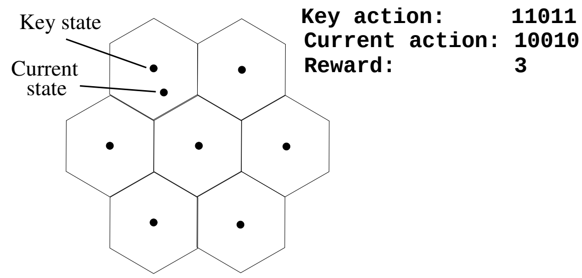

Figure 3: The large action task. The space of state bit vectors is divided into clusters of those which are nearest to each "key" state. Each key state is associated with a key action. The reward received by the learner is the number of bits shared by the selected action and the key action for the current state.

randomly selected. The network was run for 12 000 actions with a learning rate going from 0.1 to 0.01 and temperature going from 1.0 to 0.1 exponentially over the course of training. Each iteration was initialized with a random state. Each action selection consisted of 100 iterations of Gibbs sampling. The task was repeated 10 times for each method. The competing methods also had learning rates and (in the case of the backprop network) exploration schedules. The backprop network used a learning rate going from 0.005 to 0.004, and an ε-greedy exploration strategy going from 1 to 0 linearly over the course of the task. The direct policy method used a learning rate going from 0.1 to 0.01 over the course of the task. All learning parameters were selected by trial and error during preliminary experiments, with the best-performing parameters reported here.

Because the optimal action is known for each state we can compare the results to the optimal policy. We also compare to the two competing methods: the direct-policy method of Peshkin et al. (2000), and the feedforward neural network. The results are shown in Figure 4.

The learner must overcome two difficulties. First, it must find actions that receive rewards for a given state. Then, it must cluster the states which share commonly rewarded actions to infer the underlying key states. As the state space contains 2^{12} entries and the action space contains 2^{40} entries, this is not a trivial task. Yet the PoE achieves almost perfect performance after 12 000 actions. In comparison, the two other algorithms achieve suboptimal performance. The direct policy method seems to be particularly susceptible to local optima, yielding a large variance in solution quality. The backpropagation network may have continued to improve, given more training time.

7.2 The Blockers Task

The blockers task is a co-operative multi-agent task in which there are offensive players trying to reach an end zone, and defensive players trying to block them (see Figure 5).

The task is co-operative: As long as one agent reaches the end-zone, the "team" is rewarded. The team receives a reward of $+1$ when an agent reaches the end-zone, and a reward of -1 otherwise. The blockers are pre-programmed with a fixed blocking strategy. Each agent occupies one square on the grid, and each blocker occupies three horizontally adjacent squares. An agent cannot move into a square occupied by a blocker or another agent. The task has non-wrap-around edge conditions on the bottom, left and right sides of the field, and the blockers and agents can move up, down, left or right. Agents are ordered. If two agents want to move in to the same square, the first agent in

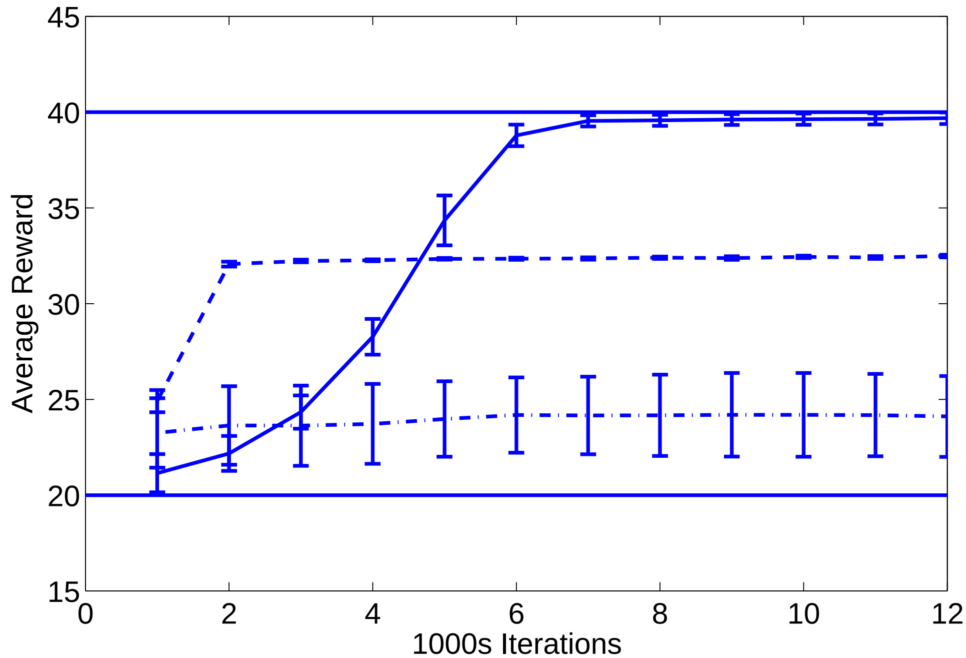

Figure 4: Results for the large action task. The graphs shows average reward versus iteration of training for three algorithms. The optimal policy gives an average reward of 40 (upper line). A random policy gives gives an average return of 20 (lower line). The solid line shows the PoE network, the dashed line shows the backprop network performance, and the dash-dotted line shows the direct policy method. Errorbars indicate 95% confidence intervals, computed across 10 repetitions of the task.

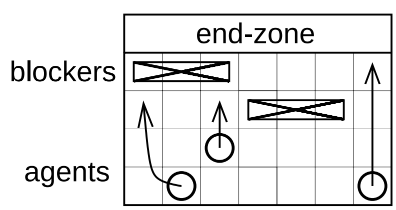

Figure 5: An example of the “blocker” task. Agents must get past the blockers to the end-zone. The blockers are pre-programmed with a strategy to stop them, but if the agents co-operate the blockers cannot stop them all simultaneously.

the ordering will succeed, and any others trying to move into that square will be unsuccessful. Note that later agents can not see the moves of earlier agents when making decisions. The ordering is just used to resolve collisions. If a move is unsuccessful, then the agent remains in its current square.

The blockers' moves are also ordered, but subsequent blockers do make decisions based on the moves of earlier blockers. The blockers operate a zone-based defense. Each blocker takes responsibility for a group of four columns. For example, blocker 1 is responsible for columns 1 through 4, blocker 2 is responsible for columns 4 through 7, and so on. If an agent moves into one of its columns and is in front of the end-zone, a blocker will move to block it. Because of the ordering, blockers will not move to stop agents that have already been stopped by other blockers.

A restricted Boltzmann machine with 4 hidden variables was trained using the SARSA learning rule on a 5×4 blocker task with two agents and one blocker. The collective state consisted of three position variables (two agents and one blocker) which could take on integer values $\{1,\ldots,20\}$. The collective action consisted of two action variables taking on values from $\{1,\ldots,4\}$. The PoE was compared to the backpropagation network and the direct policy method.

Each test was replicated 10 times. Each test run lasted for 300 000 collective actions, with a learning rate going from 0.1 to 0.01 linearly and temperature going from 1.0 to 0.01 exponentially over the course of training. Gibbs sampling and simulated annealing lasted for 10 iterations. The learning rates of the competing methods were the same as for the PoE network. The backpropagation network used an ε-greedy policy going linearly from 1 to 0 over the course of the task. The parameters for all of the methods were selected by trial and error using initial experiments, and the best performing values are reported here.

Each trial was terminated after either the end-zone was reached, or 20 collective actions were taken, whichever occurred first. Each trial was initialized with the blocker placed randomly in the top row and the agents placed randomly in the bottom row. The results are shown in Figure 6.

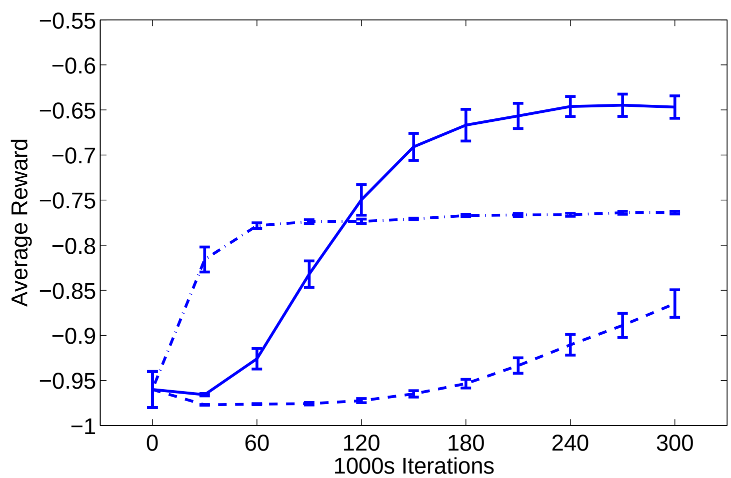

Figure 6: Results for the 2-agent blocker task. The graph shows average reward versus iteration of training for three algorithms. The solid line shows the PoE approximation; the dot-dashed line shows the direct policy method; and the dashed line shows the backprop network. The error bars show 95% confidence intervals.

Overall, the PoE network performs better than the two other algorithms. All three algorithms have the potential to find suboptimal local optima. Again, the direct policy algorithm seems to be particularly susceptible to this. The backprop network might have done better if it was allowed to continue training. The direct policy method finds a solution noticeably faster than the other two algorithms.

A restricted Boltzmann machine with 16 hidden variables was trained on a 4×7 blockers task with three agents and two blockers. Again, the input consisted of position variables for each blocker and agent, and action variables for each agent. The network was trained for 300 000 collective actions, with a learning rate going from 0.1 to 0.05 linearly and temperature from 1 to 0.06 exponentially over the course of the task. Each trial was terminated after either the end-zone was reached, or 40 steps were taken, whichever occurred first. Again, the two competitor algorithms were also used. Each competitor had 16 hidden units, and simulated annealing and Gibbs sampling lasted for 10 iterations. The competing methods used the same learning rates and exploration strategy as in the previous experiment. Again, the task was replicated 10 times for each algorithm. The results are shown in Figure 7.

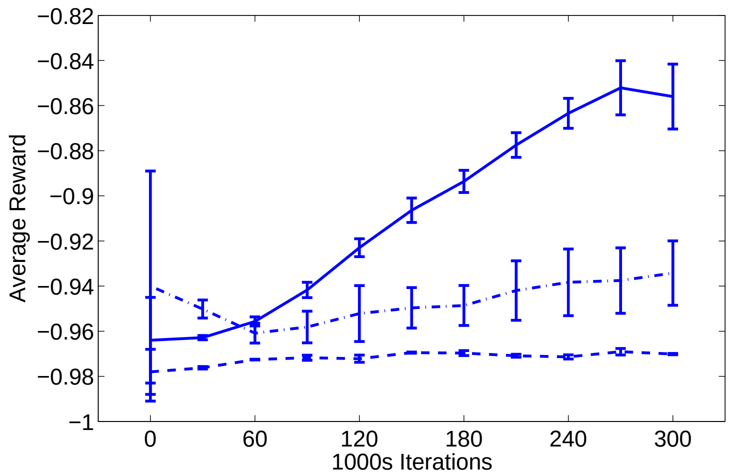

Figure 7: Results for the 3-agent blocker task. The graph shows average reward versus iteration of training for three algorithms. The solid line shows the PoE approximation; the dot-dashed line shows the direct policy method; and the dashed line shows the backprop network. The error bars show 95% confidence intervals.

In this larger version of the task, the backprop network does extremely poorly. The direct policy method does significantly worse than the PoE method. Of the three methods, the PoE method was able to find the best solution, although a suboptimal one. An example of a typical run for the 4×7 task is shown in Figure 8. The strategy discovered by the learner is to force the blockers apart with two agents, and move up the middle with the third. In the example, notice that Agent 1 seems to distract the “wrong” blocker given its earlier position. The agents in this example have learned a sub-optimal policy, where Agent 1 moves up as far as possible, and then left as far as possible, irrespective of its initial position.

Examples of features learned by the experts are shown in Figure 9. The hidden variables become active for a specific configuration in state space, and recommend a specific set of actions. Histograms below each feature indicate when that feature tends to be active during a trial. The histograms show that feature activity is localized in time. Features can be thought of as macro-actions or short-term policy segments. Each hidden variable becomes active during a particular “phase” of a trial, recommends the actions appropriate to that phase, and then ceases to be active.

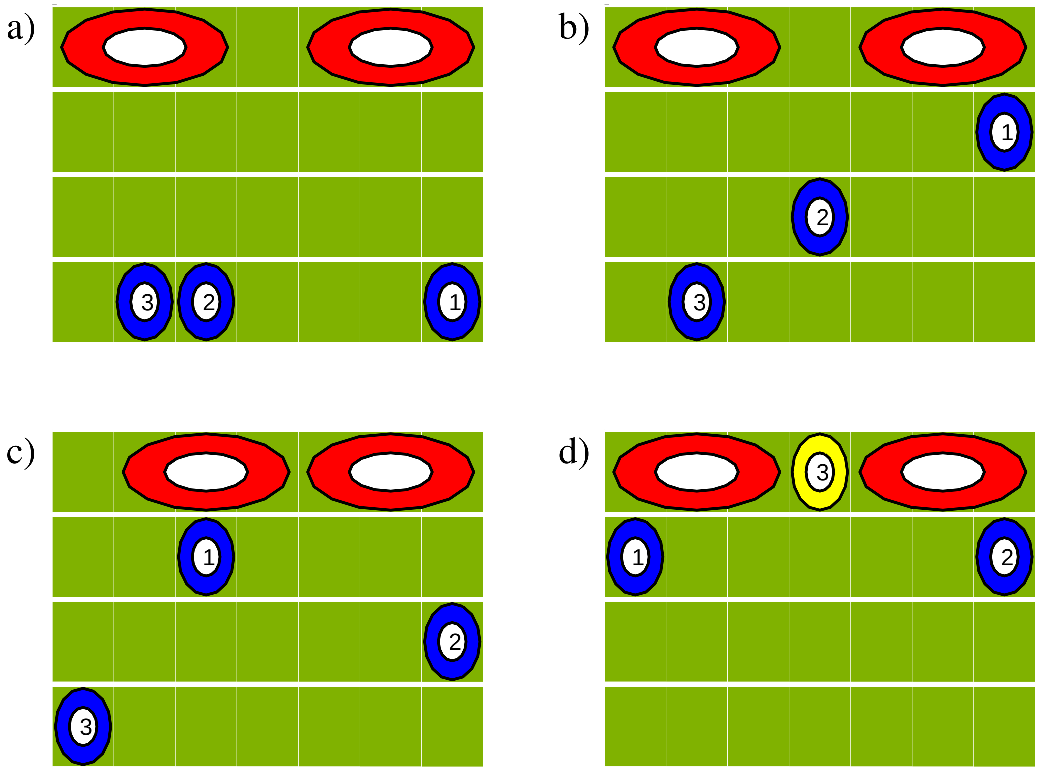

Figure 8: Example agent strategy after learning the 4×7 blocker task. a) The three agents are initialized to random locations along the bottom of the field. b) Two of the agents run to the top of the playing field. c) These two agents split and run to the sides. d) The third agent moves up the middle to the end-zone.

8. Discussion

The action sampling method is closely related to actor-critic methods (Sutton, 1984; Barto et al., 1983). An actor-critic method can be viewed as a biased scheme for selecting actions according to the value assigned to them by the critic. The selection is biased by the choice of actor parameterization. The sampling method of action selection is unbiased if the Markov chain is allowed to converge, but requires more computation. This is exactly the trade-off explored in the graphical models literature between the use of Monte Carlo inference (Neal, 1992) and variational approximations (Neal and Hinton, 1998; Jaakkola, 1997). Further, the resultant policy can potentially be more complicated than a typical parameterized actor would allow. This is because a parameterized distribution over actions has to be explicitly normalized. For example, an actor network might parameterize all policies in which the probability over each action variable is independent. This is the restriction implemented by Peshkin et al. (2000), and is also used for the direct policy method in our experimental section.

The sampling algorithm is also related to probability matching (Sabes and Jordan, 1996), in which good actions are made more probable under a model, and the temperature at which the prob-

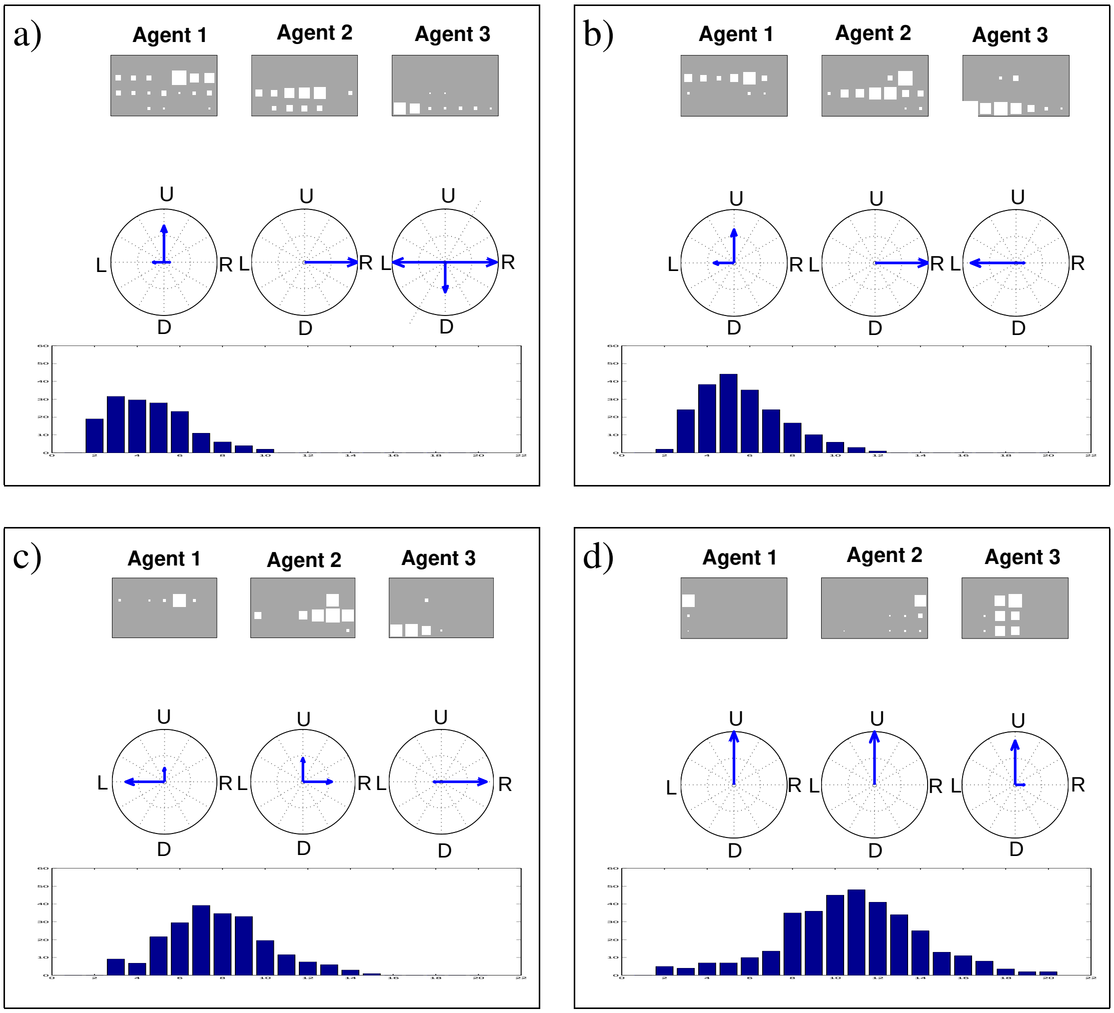

Figure 9: Features of the learned value function approximator for the 3-agent blocker task. The four features (a,b,c and d) correspond to the four stages shown in Figure 8. Each feature corresponds to a hidden variable in the RBM. The Hinton diagram shows where each of the three agents must be in order to "activate" the hidden variable (cause it to have a value of unity with high probability). The vector diagram indicates what actions are recommended by the hidden variable. The histogram is a plot of frequency of activation of the hidden variable versus time in a trial. It shows when during a run this feature tends to be active. The learned features are localized in state space and action space. Feature activity is localized in time.

ability is computed is slowly reduced over time in order to move from exploration to exploitation and avoid local minima. Unlike the sampling algorithm, the probability matching algorithm used a parameterized distribution which was maximized using gradient descent, and it did not address temporal credit assignment.

The PoE approximator could also be used in direct policy method. The network would directly encode the probabilities of selecting actions given states, rather than encoding the values of states and actions. Given a state, an action could be found using a sampling method. The sampling method would select actions approximately according to the policy encoded by the PoE.

Direct policy methods can be advantageous, because encoding the relative goodness of actions or a ranking of actions might be simpler than encoding actual values. That is, the optimal policy might be easier to learn than the value function. A PoE could be used with any direct policy method that only requires samples from the policy. This is because it is in general intractable to evaluate the probabilities of actions encoded in the PoE network, but possible to approximately sample actions using an MCMC method.

It is possible that the Gibbs sampling method which we use might not work well for some problems. In this case, other sampling methods could be used, which are better suited to avoiding local minima. While the need to sample actions using MCMC can be viewed as a disadvantage of our technique, an advantage is that improvements in sampling methods can be easily incorporated as they are developed.

8.1 Macro Actions

One way to interpret the individual experts in the product model is that they are learning “macro” or “basis” actions. As we have seen with the Blockers task, the hidden variables come to represent sets of actions that are spatially and temporally localized. We can think of the hidden variables as representing “basis” actions that can be combined to form a wide array of possible actions. The benefit of having basis actions is that it reduces the number of possible actions, thus making exploration more efficient. The drawback is that if the set of basis actions do not span the space of all possible actions, some actions become impossible to execute. By optimizing the set of basis actions during reinforcement learning, we find a set that can form useful actions, while excluding action combinations that are either not seen or not useful.

The “macro” actions learned by the PoE should not be confused with “temporally abstract actions”. The learning and use of temporally abstract actions is an important area of current research in reinforcement learning (Parr and Russell, 1998; Precup et al., 1998; McGovern, 1998; Dietterich, 2000). The “macro” actions learned by the PoE have some features in common with these temporally abstract actions. In particular, the PoE macro actions tend to remain active for temporally prolonged periods. However, that does not make them temporally abstract actions. They do not come with the formal machinery of most temporally abstract actions (such as termination conditions), and it would be difficult to fit them in to one of the existing frameworks for temporal abstraction. The PoE “basis” actions should be thought of as finding a smaller subset of “good” actions within a large space of possible actions.

This suggests one way to improve the performance of an existing PoE solution. If the solution is performing poorly, it could be because some of the action space is not spanned by basis actions. Adding and learning parameters for additional hidden variables, while holding the parameters of the pre-existing variables constant, would allow the policy to improve without having to re-learn the entire solution. Similarly, if some useful collective actions are known a priori, they can be “hard-coded” into the PoE by fixing the hidden-action weights, and allowing the PoE to learn when (in which collective states) to use them.

8.2 Conditional Completion

If the action allowed at a particular time step is constrained, it is natural to want to know what is the best action consistent with the constraints. For example, if one of the agents in the blocker task becomes unable to move in directions other than up, we would like to ask for actions for the other agents that are consistent with this restriction. Sampling allows us to do this easily by fixing a subset of action variables to their required values, and sampling the rest. The result is a set of good values for some action variables conditioned on the fixed values of the others.

Similarly, we can fix only some of the state variables, and sample others. No doubt this would be most useful for data completion: If a state variable is missing, it would be nice to fill it in with its most probable value, conditioned on the others. Unfortunately we can not do this with a single PoE model. Instead of filling in values according to how probable they are under the dynamics of the environment, it will fill in values that yield high expected returns. In other words, the values that will be filled in for state variables will be those that are most desirable, not most probable. This could be used for an optimistic form of state completion, giving an upper bound on what reward we expect to see given that we do not really know what values those state variables take on. This could also be used to identify valuable "target" states that should be achieved if possible.

9. Summary

In this article we have shown that a combination of probabilistic inference, model learning and value function approximation allows for the solution of large Markov decision processes with factored states and actions. We have shown that the sampling technique can select actions in large action spaces (40 bit actions). We have drawn links between approximate inference, state representation and action selection. Future research on hierarchical value functions and directly learning stochastic policies represented as PoE models might be particularly fruitful.

Acknowledgments

The authors would like to thank Radford Neal, Craig Boutilier, Zoubin Ghahramani, Peter Dayan, Mike Mozer, Yee-Whye Teh and Andy Brown for valuable discussions and suggestions, and the anonymous referees for many helpful comments which substantially improved the paper. The majority of this research was completed while the authors were at the University of Toronto and the Gatsby Computational Neuroscience Unit, University College London. This research was supported by the Natural Science and Engineering Research Council of Canada and the Gastby Charitable Foundation. The Austrian Research Institute for Artificial Intelligence is supported by the Austrian Federal Ministry of Education, Science and Culture and by the Austrian Federal Ministry for Transport, Innovation and Technology.

Appendix A.

In this appendix we give some derivations related to restricted Boltzmann machines. First, we show that, for a restricted Boltzmann machine, the posterior distribution over hidden variables given visible variables factors into the product of the posterior distributions over each individual hidden variable. Second, we show that the two expressions for the equilibrium free energy are equivalent, and compute the derivative of the equilibrium free energy of a restricted Boltzmann machine with respect to a parameter.

Given a set of binary random variables $\mathbf{V}$ with values $\mathbf{v}$, binary hidden variables $\mathbf{H}$ with values $\mathbf{h}$, and symmetric weighted edge w_{ik} connecting visible variable i to hidden variable k, the equilibrium free energy is given by

$$F(\mathbf{v}) = \langle E(\mathbf{v},\mathbf{h})\rangle_{P(\mathbf{h}|\mathbf{v})} + \langle \log P(\mathbf{h}|\mathbf{v})\rangle_{P(\mathbf{h}|\mathbf{v})}.$$

In the above, $E(\mathbf{v},\mathbf{h})$ denotes the energy, $P(\mathbf{h}|\mathbf{v})$ denotes the posterior distribution of the hidden variables given the visible variables, and $\langle\cdot\rangle_P$ denotes an expectation with respect to distribution P (see Section 5.1).

Consider the posterior distribution over the hidden variables. In the following, v_i denotes the value of visible variable i, and h_k denotes the value of hidden variable k. The notation $\sum_{\widehat{\mathbf{h}}}$ denotes a summation over all possible assignments to the binary variables in the set $\mathbf{H}$.

$$\begin{aligned}
P(\mathbf{h}|\mathbf{v}) &= \frac{\exp\{\sum_{i,k} w_{ik} v_i h_k\}}{\sum_{\widehat{\mathbf{h}}} \exp\{\sum_{i,k} w_{ik} v_i \widehat{h}_k\}} \\
&= \frac{\prod_k \exp\{\sum_i w_{ik} v_i h_k\}}{\sum_{\widehat{\mathbf{h}}} \prod_k \exp\{\sum_i w_{ik} v_i \widehat{h}_k\}} \\
&= \frac{\prod_k \exp\{\sum_i w_{ik} v_i h_k\}}{\prod_k \sum_{\widehat{h}_k=0}^{1} \exp\{\sum_i w_{ik} v_i \widehat{h}_k\}} \\
&= \prod_k \frac{\exp\{\sum_i w_{ik} v_i h_k\}}{\sum_{\widehat{h}_k=0}^{1} \exp\{\sum_i w_{ik} v_i \widehat{h}_k\}} \\
&= \prod_k P(h_k|\mathbf{v}).
\end{aligned}$$

The posterior factors into the product of the posterior distributions over each separate hidden variable, given the values of the visible variables. The posterior over hidden variables can be computed efficiently, because each individual hidden-unit posterior is tractable:

$$P(h_k = 1|\mathbf{v}) = \sigma\left(E(\mathbf{v}, h_k = 1)\right),$$

where $\sigma(\cdot)$ denotes the logistic function: $\sigma(x) = 1/(1+e^{-x})$.

We will now compute the derivative of the equilibrium free energy with respect to a weight. We follow the technique of (Hertz et al., 1991). First, we prove the correspondence between the negative equilibrium free energy and the log of the normalizing constant of the posterior distribution (see Eqs. 6 and 9) :

$$\begin{aligned}
F(\mathbf{v}) &= \langle E(\mathbf{v},\mathbf{h})\rangle_{P(\mathbf{h}|\mathbf{v})} + \langle \log P(\mathbf{h}|\mathbf{v})\rangle_{P(\mathbf{h}|\mathbf{v})} \\
&= \langle E(\mathbf{v},\mathbf{h})\rangle_{P(\mathbf{h}|\mathbf{v})} + \langle \log P(\mathbf{h}|\mathbf{v})\rangle_{P(\mathbf{h}|\mathbf{v})} \\
&\qquad + \log \sum_{\widehat{\mathbf{h}}} \exp\{-E(\mathbf{v},\widehat{\mathbf{h}})\} - \log \sum_{\widehat{\mathbf{h}}} \exp\{-E(\mathbf{v},\widehat{\mathbf{h}})\} \\
&= -\langle \log \exp\{-E(\mathbf{v},\mathbf{h})\}\rangle_{P(\mathbf{h}|\mathbf{v})} + \langle \log P(\mathbf{h}|\mathbf{v})\rangle_{P(\mathbf{h}|\mathbf{v})} \\
&\qquad + \log \sum_{\widehat{\mathbf{h}}} \exp\{-E(\mathbf{v},\widehat{\mathbf{h}})\} - \log \sum_{\widehat{\mathbf{h}}} \exp\{-E(\mathbf{v},\widehat{\mathbf{h}})\}
\end{aligned}$$

$$
\begin{aligned}
&= -\left\langle \log \frac{\exp\{-E(\mathbf{v},\mathbf{h})\}}{\sum_{\widehat{\mathbf{h}}} \exp\{-E(\mathbf{v},\widehat{\mathbf{h}})\}} \right\rangle_{P(\mathbf{h}|\mathbf{v})} \\
&\qquad + \langle \log P(\mathbf{h}|\mathbf{v}) \rangle_{P(\mathbf{h}|\mathbf{v})} - \log \sum_{\widehat{\mathbf{h}}} \exp\{-E(\mathbf{v},\widehat{\mathbf{h}})\} \\
&= -\langle \log P(\mathbf{h}|\mathbf{v}) \rangle_{P(\mathbf{h}|\mathbf{v})} + \langle \log P(\mathbf{h}|\mathbf{v}) \rangle_{P(\mathbf{h}|\mathbf{v})} - \log \sum_{\widehat{\mathbf{h}}} \exp\{-E(\mathbf{v},\widehat{\mathbf{h}})\} \\
&= -\log \sum_{\widehat{\mathbf{h}}} \exp\{-E(\mathbf{v},\widehat{\mathbf{h}})\} \\
&= -\log Z_h, \qquad (11)
\end{aligned}
$$

where Z_h denotes the normalizing constant of the posterior distribution over the hidden variables given the visible variables.

Next, we can take the derivative of $-\log Z_h$ with respect to a weight w_{ik}.

$$
\begin{aligned}
-\frac{\partial \log Z_h}{\partial w_{ik}} &= -\frac{1}{Z_h}\frac{\partial Z_h}{\partial w_{ik}} \\
&= \frac{-1}{Z_h}\frac{\partial}{\partial w_{ik}}\left[\sum_{\widehat{\mathbf{h}}} \exp\{-E(\mathbf{v},\widehat{\mathbf{h}})\}\right] \\
&= \frac{-1}{Z_h}\frac{\partial}{\partial w_{ik}}\left[\sum_{\widehat{\mathbf{h}}} \exp\{\sum_{i,k} w_{ik} v_i \widehat{h}_k\}\right] \\
&= \sum_{\widehat{\mathbf{h}}} \frac{-1}{Z_h}\left[v_i \widehat{h}_k \exp\{\sum_{i,k} w_{ik} v_i \widehat{h}_k\}\right] \\
&= -\sum_{\widehat{\mathbf{h}}} \frac{\exp\{\sum_{i,k} w_{ik} v_i \widehat{h}_k\}}{Z_h} v_i \widehat{h}_k \\
&= -\sum_{\widehat{\mathbf{h}}} P(\widehat{\mathbf{h}}|\mathbf{v}) v_i \widehat{h}_k \\
&= -v_i \langle h_k \rangle_{P(\mathbf{h}|\mathbf{v})}.
\end{aligned}
$$

Thus, the derivative of the equilibrium free energy with respect to a weight is simply the expected value of the hidden variable, times the value of the visible variable.

References

D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

L. Baird and H. Klopf. Reinforcement learning with high-dimensional continuous actions. Technical Report WL-TR-93-1147, Wright Laboratory, Wright-Patterson Air Force Base, 1993.

L. Baird and A. Moore. Gradient descent for general reinforcement learning. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems*, volume 11. The MIT Press, Cambridge, 1999.

A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, 13:835–846, 1983.

R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957a.

R. E. Bellman. A Markov decision process. *Journal of Mathematical Mechanics*, 6:679–684, 1957b.

D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

C. Boutilier, F. Bacchus, and R. I. Brafman. Ucp-networks: A directed graphical representation of conditional utilities. In *Proceedings of the Seventeenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-01)*, pages 56–64, 2001.

C. Boutilier, R. I. Brafman, H. H. Hoos, and D. Poole. Reasoning with conditional ceteris paribus preference statements. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning*, pages 71–80, 1999.

C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Articial Intelligence*, 121:49–107, 2000.

C. Boutilier and D. Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proc. AAAI-96*, 1996.

A. D. Brown and G. E. Hinton. Products of hidden Markov models. In T. S. Jaakkola and T. Richardson, editors, *Proceedings of Artificial Intelligence and Statistics 2001*, pages 3–11, 2001.

G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405, 1990.

R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag, 1999.

T. Dean, R. Givan, and K. Kim. Solving stochastic planning problems with large state and action spaces. In *Proc. Fourth International Conference on Artificial Intelligence Planning Systems*, 1998.

T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5, 1989.

T. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth. Hybrid monte carlo. *Physics Letters B*, 195: 216–222, 1987.

Y. Freund and D. Haussler. Unsupervised learning of distributions on binary vectors using two layer networks. In John E. Moody, Steven J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4. Morgan Kaufmann , San Mateo, 1992.

S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems*, volume 14. The MIT Press, Cambridge, 2002.

J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Reading, MA, 1991.

G. E. Hinton. Products of experts. In *ICANN-99*, volume 1, pages 1–6, 1999.

G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14 (8):1771–1800, 2002.

G. E. Hinton and T. J. Sejnowski. *Parallel Distributed Processing*, volume 1, chapter Learning and Relearning in Boltzman Machines, pages 282–317. The MIT Press, Cambridge, 1986.

R. A. Howard. *Dynamic Programming and Markov Processes*. The MIT Press, Cambridge, MA, 1960.

T. S. Jaakkola. *Variational Methods for Inference and Estimation in Graphical Models*. Department of Brain and Cognitive Sciences, MIT, Cambridge, MA, 1997. Ph.D. thesis.

T. S. Jaakkola, S. P. Singh, and M. I. Jordan. Reinforcement learning algorithm for partially observable Markov decision problems. In Gerald Tesauro, David S. Touretzky, and Todd K. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 345–352. The MIT Press, Cambridge, 1995.

M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37:183:233, 1999.

R. E. Kalman. A new approach to linear filtering and prediction problems. *Trans. ASME, Series D, Journal of Basis Engineering*, 82:35–45, March 1960.

W. S. Lovejoy. A survey of algorithmic methods for partially observable Markov decision processes. *Annals of Operations Research*, 28:47–66, 1991.

D. A. McAllester and S. P. Singh. Approximate planning for factored POMDPs using belief state simplification. In *Proc. UAI'99*, 1999.

A. McGovern. acQuire-macros: An algorithm for automatically learning macro-actions. In *NIPS98 Workshop on Abstraction and Hierarchy in Reinforcement Learning*, 1998.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

N. Meuleau, M. Hauskrecht, K-E Kim, L. Peshkin, L. P. Kaelbling, T. Dean, and C. Boutilier. Solving very large weakly coupled Markov decision processes. In *Proc. AAAI-98*, 1998.

R. M. Neal. Connectionist learning of belief networks. *Artificial Intelligence*, 56:71–113, 1992.

R. M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto, 1993.

R. M. Neal and G. E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 355–368. Kluwer Academic Publishers, 1998.

R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, Cambridge, 1998.

L. Peshkin, K-E Kim, N. Meuleau, and L. P. Kaelbling. Learning to cooperate via policy search. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, 2000.

L. Peshkin, N. Meuleau, and L. P. Kaelbling. Learning policies with external memory. In *Proc. 16th International Conference on Machine Learning*, pages 307–314, 1999.

R. B. Potts. Some generalized order-disorder transformations. *Proc. Camb. Phil. Soc.*, 48:106–109, 1952.

P. Poupart and C. Boutilier. Vector-space analysis of belief-state approximation for POMDPs. In *Proceedings of the Seventeenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-01)*, 2001.

D. Precup, R. Sutton, and S. Singh. Theoretical results on reinforcement learning with temporally abstract options. In *Proceedings of the Tenth European Conference on Machine Learning*, 1998.

L. R. Rabiner and B.-H. Juang. An introduction to hidden Markov models. *IEEE ASSAP Magazine*, 3:4–16, January 1986.

A. Rodriguez, R. Parr, and D. Koller. Reinforcement learning using approximate belief states. In S. A. Solla, T. K. Leen, and K-R Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. The MIT Press, Cambridge, 2000.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by back-propagating errors. *Nature*, 323:533–536, 1986.

G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Engineering Department, Cambridge University, 1994.

P. N. Sabes and M. I. Jordan. Reinforcement learning by probability matching. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1080–1086. The MIT Press, Cambridge, 1996.

B. Sallans. Learning factored representations for partially observable Markov decision processes. In S. A. Solla, T. K. Leen, and K-R Müller, editors, *Advances in Neural Information Processing Systems*, volume 12, pages 1050–1056. The MIT Press, Cambridge, 2000.

J. C. Santamaria, R. S. Sutton, and A. Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, 6(2), 1998.

S. Singh, T. Jaakkola, M. Littmann, and C. Szepasvfiri. Convergence results for single-step on-policy reinforcementlearning algorithms. *Machine Learning*, 38(3):287–308, 2000.

P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*. The MIT Press, Cambridge, MA, 1986.

R. S. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. University of Massachusetts, Amherst, 1984. Ph.D. thesis.

R. S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44, 1988.

R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1038–1044. The MIT Press, Cambridge, 1996.

R. S Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 1998.

S. Thrun. Monte Carlo POMDPs. In S. A. Solla, T. K. Leen, and K-R Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. The MIT Press, Cambridge, 2000.