ESANN 2010 proceedings, European Symposium on Artificial Neural Networks - Computational Intelligence and Machine Learning. Bruges (Belgium), 28-30 April 2010, d-side publi., ISBN 2-930307-10-2.

Free-energy-based Reinforcement Learning in a Partially Observable Environment

Makoto Otsuka[1,2], Junichiro Yoshimoto[1,2] and Kenji Doya[1,2]

1- Initial Research Project, Okinawa Institute of Science and Technology
12-22 Suzaki, Uruma, Okinawa 904-2234, Japan

2 - Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0192, Japan

Abstract. Free-energy-based reinforcement learning (FERL) can handle Markov decision processes (MDPs) with high-dimensional state spaces by approximating the state-action value function with the negative equilibrium free energy of a restricted Boltzmann machine (RBM). In this study, we extend the FERL framework to handle partially observable MDPs (POMDPs) by incorporating a recurrent neural network that learns a memory representation sufficient for predicting future observations and rewards. We demonstrate that the proposed method successfully solves POMDPs with high-dimensional observations without any prior knowledge of the environmental hidden states and dynamics. After learning, task structures are implicitly represented in the distributed activation patterns of hidden nodes of the RBM.

1 Introduction

Partially observable Markov decision processes (POMDPs) are versatile enough to model sequential decision making in the real world. However, state-of-the-art algorithms for POMDPs [1, 2] assume prior knowledge of environments: in particular, a set of hidden states that makes the environment Markovian and the transition and observation probabilities for the states. They also have a difficulty in handling high-dimensional sensory inputs.

The use of an undirected counterpart of Bayesian networks has yielded a new algorithm to handle Markov decision processes (MDPs) with a large state space [3]. In this free-energy-based reinforcement learning (FERL), a restricted Boltzmann machine (RBM) is used to approximate the state-action value function as the negative free energy of the RBM.

In this study, we extend this FERL framework to handle POMDPs using Whitehead's recurrent-model architecture [4]. The proposed method can handle high-dimensional observations and solve POMDPs without any prior knowledge of the environmental state set and dynamics.

2 Free-energy-based reinforcement learning framework

We briefly review the FERL framework for MDPs [3]. In this framework, the agent is realized by a RBM (Fig. 1(a)). The visible layer $\boldsymbol{V}$ is composed of binary state nodes $\boldsymbol{S}$ and action nodes $\boldsymbol{A}$. The hidden layer is composed of

binary hidden nodes $\boldsymbol{H}$. A state node S_i is connected to a hidden node H_k by the connection weight w_{ik}, and an action node A_j is connected to a hidden node H_k by the connection weight u_{jk}. A hidden node h_k takes a binary value with the probability $Pr(h_k = 1) = \sigma(\sum_i w_{ik}s_i + \sum_j u_{jk}a_j)$ where $\sigma(z) \triangleq 1/(1+\exp(-z))$. The free energy of the system, which is the negative log-partition function of posterior probability over $\boldsymbol{h}$ given a configuration $(\boldsymbol{S} = \boldsymbol{s}, \boldsymbol{A} = \boldsymbol{a})$ in thermal equilibrium with the unit temperature, is given by

$$F(\boldsymbol{s},\boldsymbol{a}) = -\boldsymbol{s}^\top \boldsymbol{W}\hat{\boldsymbol{h}} - \boldsymbol{a}^\top \boldsymbol{U}\hat{\boldsymbol{h}} + \sum_{k=1}^{K} \hat{h}_k \log \hat{h}_k + \sum_{k=1}^{K} (1-\hat{h}_k)\log(1-\hat{h}_k)$$

where $\boldsymbol{W} \equiv [w_{ik}]$ and $\boldsymbol{U} \equiv [u_{ik}]$ are matrix notations of the connection weights. $\hat{h}_k \equiv \sigma([\boldsymbol{W}^\top \boldsymbol{s} + \boldsymbol{U}^\top \boldsymbol{a}]_k)$ is the conditional expectation of h_k given the configuration $(\boldsymbol{s}, \boldsymbol{a})$, where $[\cdot]_k$ denotes the k-th component of the vector enclosed within the brackets. The network is trained so that the negative free energy approximates the state-action value function, i.e., $-F(\boldsymbol{s}, \boldsymbol{a}) \simeq Q(\boldsymbol{s}, \boldsymbol{a}) \triangleq \mathbb{E}[r + \gamma Q(\boldsymbol{s}', \boldsymbol{a}')|\boldsymbol{s}, \boldsymbol{a}]$ where r, $\boldsymbol{s}'$, and $\boldsymbol{a}'$ are the reward, next state, and next action, and γ is the discount factor for future rewards. By applying the SARSA(0) algorithm with a function approximator [5], we obtain a simple update rule for the network parameters:

$$\Delta w_{ik} = \alpha \left(r_{t+1} - \gamma F(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) + F(\boldsymbol{s}_t, \boldsymbol{a}_t)\right) s_{i,t} \hat{h}_{k,t} \tag{1a}$$

$$\Delta u_{jk} = \alpha \left(r_{t+1} - \gamma F(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) + F(\boldsymbol{s}_t, \boldsymbol{a}_t)\right) a_{j,t} \hat{h}_{k,t} \tag{1b}$$

where the subscript t denotes the time and α denotes the learning rate. To select an action at a given state $\boldsymbol{s}$, we used the softmax action selection rule with inverse temperature β

$$\pi(\boldsymbol{s}, \boldsymbol{a}) = \Pr(\boldsymbol{a}|\boldsymbol{s}) \propto \exp\{-\beta F(\boldsymbol{s}, \boldsymbol{a})\} \tag{2}$$

by calculating the free energies for each action.

3 Model architecture

We incorporate Whitehead's recurrent-model architecture [4] into FERL framework for solving POMDPs, as shown in Fig. 1(b). The architecture consists of two modules: an Elman-type recurrent neural network (RNN) for one-step prediction (predictor) and a RBM for state-action value estimation (actor).

The predictor module predicts the upcoming observation $\boldsymbol{y}_t$ and reward $\boldsymbol{r}_t$[1] on the basis of the memory $\boldsymbol{m}_t$, which is supposed to summarize the history of all past events. At each time t, the memory is given by the sigmoid function $\sigma(\cdot)$ of a linear transformation of the previous observation, action, and memory $(\boldsymbol{y}_{t-1}, \boldsymbol{a}_{t-1}, \boldsymbol{m}_{t-1})$. Once the memory $\boldsymbol{m}_t$ is given, the network predicts $(\boldsymbol{y}_t,$

[1]The notation r denotes a scalar reward, and the vector notation $\boldsymbol{r}$ denotes a bit coding of the scalar reward with respect to all possible rewards.

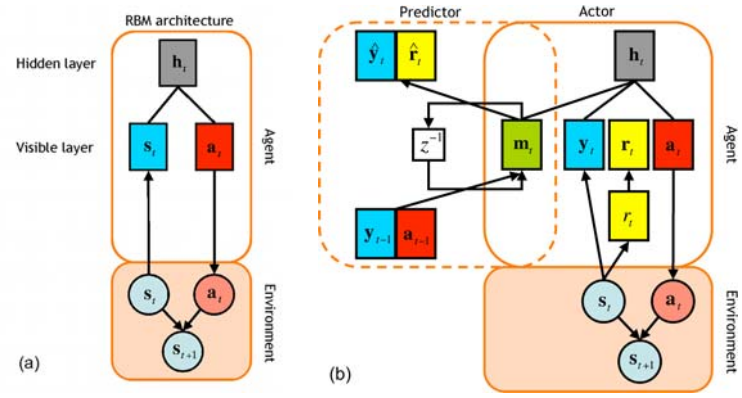


Fig. 1: Models for handling high-dimensional inputs. (a) An actor-only architecture for MDPs. (b) A predictor-actor architecture for POMDPs.

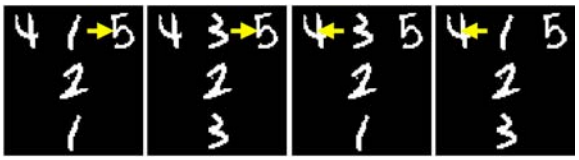


Fig. 2: Digit matching T-maze task. The optimal action at the T-junction is indicated by arrows.

$\boldsymbol{r}_t$) as the sigmoid function of a linear mapping of $\boldsymbol{m}_t$. All linear coefficients (weights and biases) of the network are trained by the backpropagation through time (BPTT) algorithm [6]. The actor module regards the combination of the current observation and predictor's memory $(\boldsymbol{y}_t, \boldsymbol{m}_t)$ as the state vector $\boldsymbol{s}$. The actor is trained by the SARSA(0) algorithm with Eq. (1).

4 Experiments

We designed a matching T-maze task in order to show the proposed model's ability to solve POMDPs without any prior knowledge of the environmental state set and dynamics. The matching T-maze task is an extension of the non-Markovian grid-based T-maze task [7] to investigate the coding and combinatorial usage of task-relevant information.

An agent can execute four possible actions: go one step North, West, East, or South. At each time step, an agent observes a binary vector depending on the position in the maze. In the first experiment, the observation is composed of five bits encoding the position: (1) the start position, (2) the middle of the corridor, (3) the T-junction, (4) the left goal, and (5) the right goal and two bits of signals specifying the rewarding goal position, observed at the start position and the

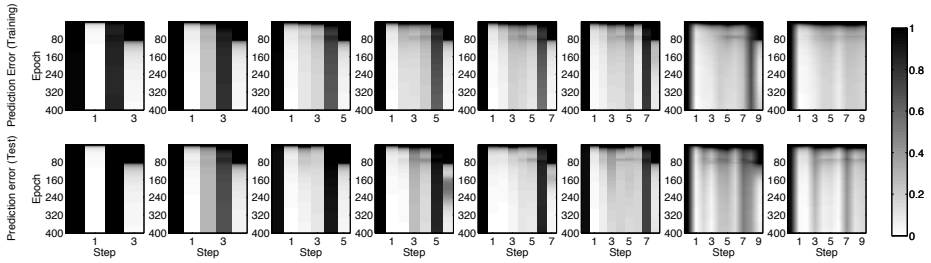


Fig. 3: Average weighted prediction errors of observations and rewards. The top and bottom rows show the error for the training and test dataset, respectively. The vertical and horizontal axes of each panel indicate the training epoch of RNNs and steps t in a episode, respectively.

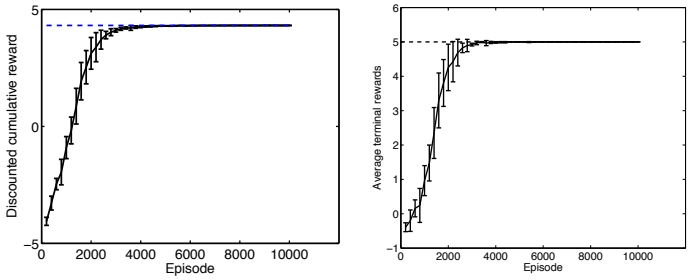
T-junction only. In the second experiment, observations are 784-dimensional binary hand-written digits (Fig. 2).

An episode ends either when the agent steps into the goal states or after the number of action selections exceeds 10 steps. If the two signals at the start position and the T-junction are the same, the agent receives a reward of $+5$ at the right goal and -5 at the left goal; if the two signals are not the same, the reward condition is reversed. When the agent hits a wall, the underlying environmental state does not change, and the agent receives a reward of -1. Otherwise, the agent receives a reward of -0.1. Upon each episode, the two signals are independently and randomly selected and fixed. We used 7 or 784 observation nodes $\boldsymbol{Y}$, 4 reward nodes $\boldsymbol{R}$, and 20 memory nodes $\boldsymbol{M}$ for the predictor module; we used 20 hidden nodes $\boldsymbol{H}$ and 4 action nodes $\boldsymbol{A}$ for the actor module.

4.1 Matching T-maze task with orthogonal bit codes

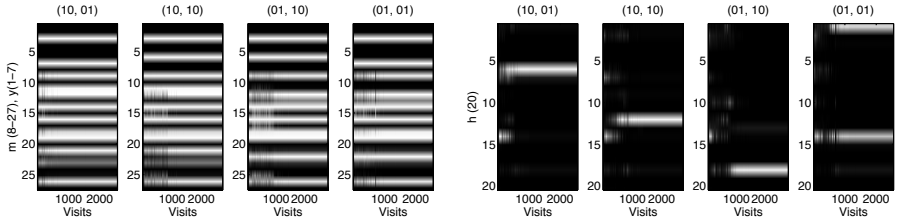
The predictor was first trained by 60 episodic training data with varying step lengths from 3 to 7, collected by the random action selection, repeatedly 400 epochs (Fig. 3). Using the pre-trained predictor, the proposed predictor-actor model successfully learned the optimal policy in this task (Figs. 4(a), (b)).

Figs. 5(a), (b) show the activations of the actor's state nodes $(\boldsymbol{M}, \boldsymbol{Y})$ and its hidden nodes $\boldsymbol{H}$ at the T-junction, respectively. The memory layer retained the signal at the start position (Fig. 5(a)). Principal component analysis (PCA) of the activation patterns in the actor's network revealed that the four signal conditions were well separated even before the actor's learning started (Fig. 6(a)). This clear separation in high-dimensional space was helpful for *state representation* in the actor module in that it allowed the agent to learn the optimal policy. In addition, the activations of the hidden nodes showed a gradual separation of firing patterns through the actor's learning process as though the activations were functionally differentiated. (Figs. 5(b) and 6(b)).



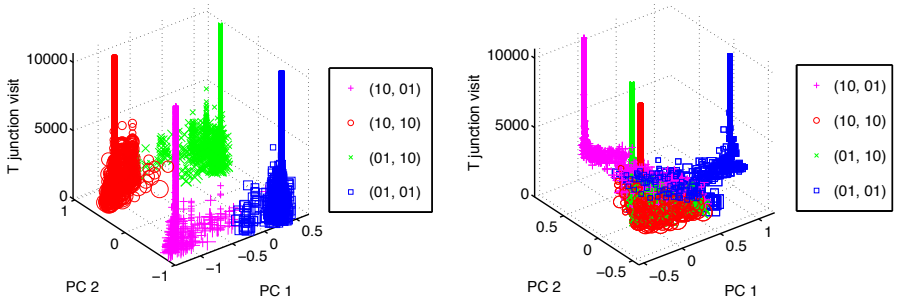
(a) Discounted return. (b) Terminal reward.

Fig. 4: Performance of the predictor-actor model in the matching T-maze task with low-dimensional bit-coded observations. The error bars show a standard deviation over 10 runs. The theoretically optimal performance is indicated by the dotted lines.



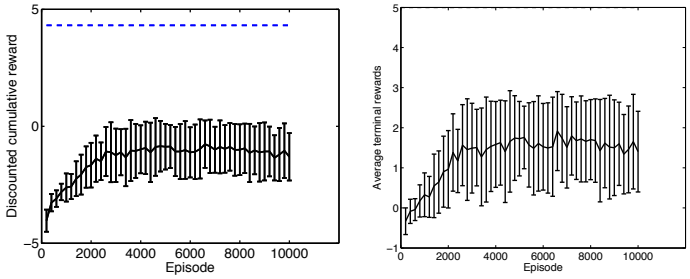
(a) Actor's state nodes. (b) Actor's hidden nodes.

Fig. 5: Activation patterns of the actor's nodes at T-junction. The bit patterns "10" and "01" enclosed within the parentheses show four conditions of the signals at two positions (start, T-junction). (a) Actor's state nodes composed of $\boldsymbol{M}$ and $\boldsymbol{Y}$. (b) Conditional activation of actor's hidden nodes $\boldsymbol{H}$.



(a) Actor's state nodes. (b) Actor's hidden nodes

Fig. 6: PCA analysis of actor's activation patterns in all T-junction visits. The size of the marker reflects the number of steps to the goals. The smallest marker indicates 3 steps, and the largest marker indicates 10 steps.



(a) Discounted return. (b) Terminal reward.

Fig. 7: Performance of the predictor-actor model in the matching T-maze task with high-dimensional pixel observations.

4.2 Matching T-maze task with high-dimensional observations

In the second task, pixel images of hand-written digits were used as observations. The performance of the agent remained suboptimal as shown in Fig. 7(a); however, the agent still showed the tendency to select the correct goal as shown in Fig. 7(b). It indicates that the information about the initial signal was at least retained in the predictor's hidden nodes.

5 Conclusion and future work

In this study, we extended the FERL framework to handle POMDPs. Here, neither the state transition probability nor the true set of the underlying Markovian state was given *a priori*. We used this approach to handle high-dimensional observations and obtained preliminary results.

In order to improve the performance of this architecture, the separation of a predictor for observations and rewards can be helpful. With this modification, several nuisance parameters are removed, and a scalar reward can be handled.

References

[1] J. Hoey and P. Poupart. Solving POMDPs with continuous or large discrete observation spaces. In *IJCAI*, volume 19, page 1332, 2005.

[2] M. Toussaint, L. Charlin, and P. Poupart. Hierarchical POMDP controller optimization by likelihood maximization. *UAI*, 2008.

[3] B. Sallans and G. E. Hinton. Reinforcement learning with factored states and actions. *Journal of Machine Learning Research*, 5:1063–1088, 2004.

[4] S. D. Whitehead and L. J. Lin. Reinforcement learning of non-Markov decision processes. *Artificial Intelligence*, 73:271–306, 1995.

[5] R. S. Sutton and A. G. Barto. *Reinforcement Learning*. MIT Press, 1998.

[6] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[7] B. Bakker. Reinforcement learning with long short-term memory. *NIPS*, 2:1475–1482, 2002.