Neural Networks 84 (2016) 17–27

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

From free energy to expected energy: Improving energy-based value function approximation in reinforcement learning

Stefan Elfwing [a,b,*], Eiji Uchibe [a,b], Kenji Doya [b]

[a] Department of Brain Robot Interface, ATR Computational Neuroscience Laboratories, 2-2-2 Hikaridai, Seikacho, Soraku-gun, Kyoto 619-0288, Japan

[b] Okinawa Institute of Science and Technology Graduate University, 1919-1 Tancha, Onna-son, Okinawa 904-0495, Japan

ARTICLE INFO

Article history:
Received 17 July 2015
Received in revised form 22 April 2016
Accepted 28 July 2016
Available online 26 August 2016

Keywords:
Reinforcement learning
Restricted Boltzmann machine
Expected energy
Function approximation
SZ-Tetris

ABSTRACT

Free-energy based reinforcement learning (FERL) was proposed for learning in high-dimensional state and action spaces. However, the FERL method does only really work well with binary, or close to binary, state input, where the number of active states is fewer than the number of non-active states. In the FERL method, the value function is approximated by the negative free energy of a restricted Boltzmann machine (RBM). In our earlier study, we demonstrated that the performance and the robustness of the FERL method can be improved by scaling the free energy by a constant that is related to the size of network. In this study, we propose that RBM function approximation can be further improved by approximating the value function by the negative expected energy (EERL), instead of the negative free energy, as well as being able to handle continuous state input. We validate our proposed method by demonstrating that EERL: (1) outperforms FERL, as well as standard neural network and linear function approximation, for three versions of a gridworld task with high-dimensional image state input; (2) achieves new state-of-the-art results in stochastic SZ-Tetris in both model-free and model-based learning settings; and (3) significantly outperforms FERL and standard neural network function approximation for a robot navigation task with raw and noisy RGB images as state input and a large number of actions.

1. Introduction

Sallans and Hinton (2004) proposed free-energy based reinforcement learning (hereafter, *FERL*) to handle high-dimensional state and action spaces. In the FERL method, the value function is approximated by the negative free energy, F, of a restricted Boltzmann machine (RBM) (Freund & Haussler, 1992; Hinton, 2002; Smolensky, 1986): $Q = -F = -\langle E \rangle + H$ for action-value based learning, where $\langle E \rangle$ is the expected energy and H is the entropy of the network. A considerable limitation of the FERL method is that it only works well with binary, or close to binary, state input. Furthermore, it is known that RBMs, traditionally, are not invariant to different state representations and require that the number of active states (values close to one) is much fewer than the number of non-active states (values close to zero) to work well.

We have earlier demonstrated (Elfwing, Uchibe, & Doya, 2013) that the robustness and the learning performance of FERL can be improved by scaling the free energy by a constant scaling factor, Z (i.e., $Q = -F/Z$), that is related to the size of the network. The purpose of this study is to show that expected energy based RBM function approximation (hereafter, *EERL*: $Q = -\langle E \rangle$) can achieve competitive learning performance, not only in tasks with binary state input and fewer active than non-active states, but also in tasks with continuous state input and in tasks with more active than non-active states. In the latter cases, we introduce a simple normalization by removing the mean of a state vector from each of its elements to improve the learning performance even further.

To validate our proposed method, we first use three versions of a gridworld task where the state input consists of (1) grayscale images of handwritten digits from the MNIST data set (LeCun, Bottou, Bengio, & Haffner, 1998); (2) inverted MNIST images; and (3) RGB images of the different objects from the CIFAR-10 data set (Krizhevsky, 2009). The purpose of the first version of the task is to test the learning performance of our proposed method for a task setting that is traditionally considered well-suited for RBMs: i.e., close to binary state input with much fewer active than non-active states. The purpose of the other two versions of the task is the opposite, i.e., a task with more active than non-active states, and a task with continuous state input. We then use the stochastic SZ-Tetris benchmark (Szita & Szepesvári, 2010) to validate the performance of EERL, in both model-free (Sarsa(λ)) and model-based

* Corresponding author at: Okinawa Institute of Science and Technology Graduate University, 1919-1 Tancha, Onna-son, Okinawa 904-0495, Japan.
E-mail addresses: elfwing@atr.jp (S. Elfwing), uchibe@atr.jp (E. Uchibe), doya@oist.jp (K. Doya).

http://dx.doi.org/10.1016/j.neunet.2016.07.013

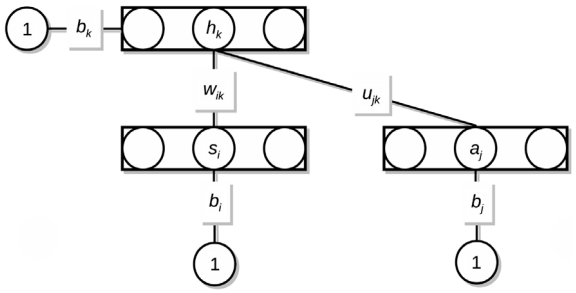

Fig. 1. RBM architecture for action-value based reinforcement learning. In the case of state-value based reinforcement learning, the action nodes, a_j, action biases, b_j, and the action weights, u_{jk}, are not included.

(TD(λ)) learning settings, for a task that is, in general, considered difficult for reinforcement learning algorithms. Finally, we use a robot visual navigation task with raw and noisy RGB camera images as state input. The goal of the task is to navigate to one of two goal areas. The correct goal can be inferred from the color of the upper part of four landmarks, which is randomly changed in each episode. In the robot navigation task, we investigate how well EERL can handle a large number of actions (pairs of velocities of the left and right wheels) by testing settings with 9, 25, and 100 possible actions.

Apart from the pioneering work by Sallans and Hinton (2004) and our earlier studies (Elfwing, Otsuka, Uchibe, & Doya, 2010; Elfwing et al., 2013), there have been few studies using RBMs as function approximation in reinforcement learning. Heess, Silver, and Teh (2012) proposed two energy-based actor-critic policy gradient algorithms and demonstrated that they were more robust and more effective than standard FERL in several high dimensional tasks. Otsuka, Yoshimoto, and Doya (2010) extended the FERL method to handle partially observable Markov decision processes, by incorporating a recurrent neural network that learns a memory representation that is sufficient for predicting future observations and rewards. In our recent work (Elfwing, Uchibe, & Doya, 2015), we demonstrated in the classification domain that the expected energy based RBM method significantly outperforms the free energy based RBM method.

2. Method

2.1. TD(λ) and Sarsa(λ)

The reinforcement learning (Sutton & Barto, 1998) methods that we propose in this study are based on the state-value function learning algorithm TD(λ) (Sutton, 1988) and the action-value function learning algorithm Sarsa(λ) (Rummery & Niranjan, 1994; Sutton, 1996). TD(λ) learns an estimate of the state-value function, V^π, and Sarsa(λ) learns an estimate of the action-value function, Q^π, while the agent follows policy π. If the approximated value functions, $V_t \approx V^\pi$ and $Q_t \approx Q^\pi$, are parameterized by the parameter vector $\boldsymbol{\theta}_t$, then the gradient-descent update of the parameters is computed by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\delta_t \boldsymbol{e}_t, \tag{1}$$

where TD-error, δ_t, is

$$\delta_t = r_t + \gamma V_t(s_{t+1}) - V_t(s_t), \tag{2}$$

for TD(λ) and

$$\delta_t = r_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t), \tag{3}$$

for Sarsa(λ). The eligibility trace vector, $\boldsymbol{e}_t$, is

$$\boldsymbol{e}_t = \gamma\lambda e_{t-1} + \nabla_{\boldsymbol{\theta}_t} V_t(s_t), \qquad \boldsymbol{e}_0 = \boldsymbol{0}, \tag{4}$$

for TD(λ) and

$$\boldsymbol{e}_t = \gamma\lambda e_{t-1} + \nabla_{\boldsymbol{\theta}_t} Q_t(s_t, a_t), \qquad \boldsymbol{e}_0 = \boldsymbol{0} \tag{5}$$

for Sarsa(λ). Here, s_t is the state at time t, a_t is the action selected at time t, r_t is the reward for taking action a_t in state s_t, α is the learning rate, γ is the discount factor of future rewards, λ is the trace-decay rate, and $\nabla_{\boldsymbol{\theta}_t} V_t$ and $\nabla_{\boldsymbol{\theta}_t} Q_t$ are the vectors of partial derivatives of the function approximators with respect to each component of $\boldsymbol{\theta}_t$.

2.2. Free energy value function approximation

The use of a RBM as a function approximator for reinforcement learning was proposed by Sallans and Hinton (2004). A RBM is a bi-directional neural network (see Fig. 1) which in the FERL method consists of binary state nodes, $\boldsymbol{s}$, binary hidden nodes, $\boldsymbol{h}$, and, in the case of action-value function learning, binary action nodes $\boldsymbol{a}$. The ith state node, s_i, is connected to hidden node h_k by the weight w_{ik}, and the jth action node, a_j, is connected to hidden node h_k by the weight u_{jk}. In addition, the state nodes, the hidden nodes and the action nodes are all connected to a constant bias input with a value of 1, with connection weights b_i, b_k, and b_j, respectively. The action vector $\boldsymbol{a}$ has an "one-out-of-J" representation and functions as a fixed input to the network for each action. Let $\boldsymbol{a}_j$ denote the vector for action j, where a_j is equal to one and the rest of the action nodes are equal to zero.

For state-value function learning, the energy, E, of the RBM for state vector $\boldsymbol{s}$ is given by

$$E(\boldsymbol{s}, \boldsymbol{h}) = -\sum_{k=1}^{K}\sum_{i=1}^{I} s_i w_{ik} h_k - \sum_{i=1}^{I} b_i s_i - \sum_{k=1}^{K} b_k h_k, \tag{6}$$

and for action-value function learning, the energy, E, of the RBM for state vector $\boldsymbol{s}$ and action vector $\boldsymbol{a}_j$ is given by

$$\begin{aligned} E(\boldsymbol{s}, \boldsymbol{a}_j, \boldsymbol{h}) &= -\sum_{k=1}^{K} h_k \left[\sum_{i=1}^{I} s_i w_{ik} + \sum_{j^*=1}^{J} a_{j^*} u_{j^*k}\right] \\ &\quad - \sum_{i=1}^{I} b_i s_i - \sum_{j^*=1}^{J} b_{j^*} a_{j^*} - \sum_{k=1}^{K} b_k h_k \\ &= -\sum_{k=1}^{K} h_k \left[\sum_{i=1}^{I} s_i w_{ik} + u_{jk}\right] \\ &\quad - \sum_{i=1}^{I} b_i s_i - b_j - \sum_{k=1}^{K} b_k h_k. \end{aligned} \tag{7}$$

Here, I is the number of state nodes, K is the number of hidden nodes, and J is the number of actions. The free energy, F, can be computed as the sum of the expected energy, $\langle E\rangle$, and the negative entropy, H, where the expectations are taken with respect to the posterior distribution of the hidden values ($P(\boldsymbol{h}|\boldsymbol{s})$ and $P(\boldsymbol{h}|\boldsymbol{s}, \boldsymbol{a}_j)$). The expected hidden activation (i.e., the probability that the hidden value is equal to one) of hidden node k is given by

$$\langle h_k\rangle = P(h_k = 1|\boldsymbol{s}) = \sigma\left(\sum_{i=1}^{I} s_i w_{ik} + b_k\right) = \sigma\left(x_k\right) \tag{8}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{9}$$

for state-value function learning and

$$\begin{aligned} \langle h_{jk}\rangle &= P(h_k = 1|\boldsymbol{s}, \boldsymbol{a}_j) = \sigma\left(\sum_{i=1}^{I} s_i w_{ik} + u_{jk} + b_k\right) \\ &= \sigma\left(x_{jk}\right), \end{aligned} \tag{10}$$

for action-value function learning. Here, x_k and x_{jk} are the inputs to the sigmoid activation function of hidden node k. The free energy can then be computed by

$$F(\boldsymbol{s}) = \langle E(\boldsymbol{s},\boldsymbol{h})\rangle + \overbrace{\langle \log P(\boldsymbol{h}|\boldsymbol{s})\rangle}^{-H(\boldsymbol{s})} \tag{11}$$

$$= -\sum_{k=1}^{K}\sum_{i=1}^{I} s_i w_{ik}\langle h_k\rangle - \sum_{i=1}^{I} b_i s_i - \sum_{k=1}^{K} b_k\langle h_k\rangle + \sum_{k=1}^{K}\langle h_k\rangle \log\langle h_k\rangle + (1-\langle h_k\rangle)\log(1-\langle h_k\rangle), \tag{12}$$

$$= -\sum_{k=1}^{K} x_k\langle h_k\rangle - \sum_{i=1}^{I} b_i s_i + \sum_{k=1}^{K}\langle h_k\rangle \log\langle h_k\rangle + (1-\langle h_k\rangle)\log(1-\langle h_k\rangle), \tag{13}$$

and

$$F(\boldsymbol{s},\boldsymbol{a}_j) = -\sum_{k=1}^{K} x_{jk}\langle h_{jk}\rangle - \sum_{i=1}^{I} b_i s_i - b_j + \sum_{k=1}^{K}\langle h_{jk}\rangle \log\langle h_{jk}\rangle \tag{14}$$

$$+\sum_{k=1}^{K}(1-\langle h_{jk}\rangle)\log(1-\langle h_{jk}\rangle). \tag{15}$$

In Sallans and Hinton's original proposal (Sallans & Hinton, 2004), the value functions are approximated by the negative free energy (i.e., $V(\boldsymbol{s}) = -F(\boldsymbol{s})$ and $Q(\boldsymbol{s},\boldsymbol{a}_j) = -F(\boldsymbol{s},\boldsymbol{a}_j)$). For Sarsa, the derivatives of the Q-function with respect to the function approximator parameters (w_{ik}, u_{jk}, b_i, b_j, and b_k), used in the learning updates (5), can be computed by

$$\nabla_{w_{ik}} Q(\boldsymbol{s},\boldsymbol{a}_j) = s_i\langle h_{jk}\rangle, \tag{16}$$

$$\nabla_{u_{jk}} Q(\boldsymbol{s},\boldsymbol{a}_j) = a_j\langle h_{jk}\rangle, \tag{17}$$

$$\nabla_{b_k} Q(\boldsymbol{s},\boldsymbol{a}_j) = \langle h_{jk}\rangle, \tag{18}$$

$$\nabla_{b_i} Q(\boldsymbol{s},\boldsymbol{a}_j) = s_i, \tag{19}$$

$$\nabla_{b_j} Q(\boldsymbol{s},\boldsymbol{a}_j) = a_j. \tag{20}$$

2.3. Expected energy value function approximation

The purpose of this study is to demonstrate that RBM based function approximation can be significantly improved by approximating the Q-function by the negative expected energy, and that EERL is an attractive general method for learning in high-dimensional state and action spaces. In EERL, the state-value and action-value functions are computed by

$$V(\boldsymbol{s}) = -\langle E(\boldsymbol{s},\boldsymbol{h})\rangle = \sum_{k=1}^{K} x_k\langle h_k\rangle + \sum_{i=1}^{I} b_i s_i, \tag{21}$$

$$Q(\boldsymbol{s},\boldsymbol{a}_j) = -\langle E(\boldsymbol{s},\boldsymbol{a}_j,\boldsymbol{h})\rangle = \sum_{k=1}^{K} x_{jk}\langle h_{jk}\rangle + \sum_{i=1}^{I} b_i s_i + b_j, \tag{22}$$

and an additional term:

$$\langle h_{jk}\rangle(1-\langle h_{jk}\rangle)x_{jk}, \tag{23}$$

is added to the derivative expressions with respect to the network parameters w_{ik}, u_{jk}, and b_k. For example, the derivatives of the Q-function with respect to w_{ik} (16) are changed to

$$\nabla_{w_{ik}} Q(\boldsymbol{s},\boldsymbol{a}_j) = s_i\left(\langle h_{jk}\rangle + \langle h_{jk}\rangle(1-\langle h_{jk}\rangle)x_{jk}\right). \tag{24}$$

Fig. 2 visualizes the differences between free energy and expected energy function approximation by showing the contributions to the state-value function from one hidden node k for FERL ($-F_k$) and EERL ($-\langle E\rangle_k$) as functions of the input to the hidden node, x_k:

$$-F_k = x_k\langle h_k\rangle - [\langle h_k\rangle \log\langle h_k\rangle + (1-\langle h_k\rangle)\log(1-\langle h_k\rangle)], \tag{25}$$

$$-\langle E\rangle_k = x_k\langle h_k\rangle. \tag{26}$$

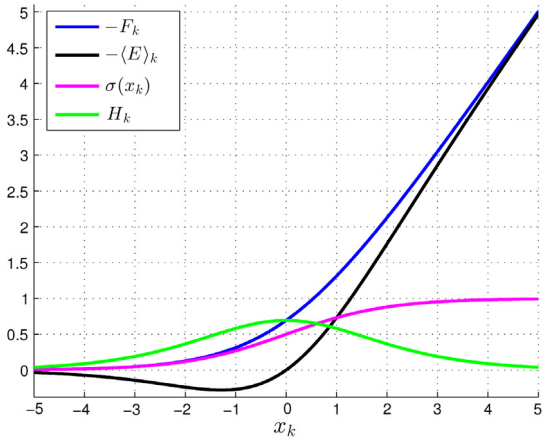

Fig. 2. The contributions to the state-value function from one hidden node k for FERL ($-F_k$) and EERL ($-\langle E\rangle_k$) as functions of the input to the hidden node, x_k, as well as the sigmoid hidden activation function and the entropy of the hidden node ($H_k = -[\langle h_k\rangle \log\langle h_k\rangle + (1-\langle h_k\rangle)\log(1-\langle h_k\rangle)]$).

For FERL, the $-F_k$-function is a monotonically increasing non-negative function that is approximately equal to the sigmoid function for x_k-values smaller than approximately -2 and approximately equal to x_k for large positive x_k-values. Since the contribution to the value function from the hidden nodes is always non-negative, the network has to counterbalance active hidden nodes with negative bias values for the state nodes (b_i) and, in the case of action value learning, the action nodes (b_j). Interestingly, for EERL, the $-\langle E\rangle_k$-function is not monotonically increasing and not non-negative. Instead, it has a global minimum value of approximately -0.28 for $x_k \approx -1.28$. For a more detailed analysis of the differences between free energy and expected energy function approximation, see Elfwing et al. (2015).

2.4. Action selection

In this study, we use softmax action selection with a Boltzmann distribution. For Sarsa, the probability to select action a in state s is defined as

$$\pi(a|s) = \frac{\exp(Q(s,a)/\tau)}{\sum_b \exp(Q(s,b)/\tau)}. \tag{27}$$

For TD(λ), we assume a model-based learning setting where the state transitions from the current state s to the next state s' are known and deterministic, for all possible next states. The probability to select an action a that leads to the next state s' is defined as

$$\pi(a|s) = \frac{\exp(V(f(s,a))/\tau)}{\sum_b \exp(V(f(s,b))/\tau)}. \tag{28}$$

Here, $f(s,a)$ returns the next state s' according to the deterministic state transition dynamics and τ is the temperature that controls the trade-off between exploration and exploitation. In this study, we used hyperbolic discounting of the temperature and the temperature was decreased in every episode i:

$$\tau(i) = \frac{\tau_0}{1+\tau_k i}. \tag{29}$$

Here, τ_0 is the initial temperature and τ_k controls the rate of discounting.

Our approach to action selection differs from the approach in the original proposal by Sallans and Hinton (2004), where they used Gibbs sampling to generate the Boltzmann distribution for softmax action selection.

2.5. Initialization and normalization

In our experience, to achieve robust and efficient learning, the amplitude of the random initialization of the weights between the action nodes and the hidden nodes (u_{jk}) has to be several magnitudes larger than the amplitude of the random initialization of the other weights. In this study, u_{jk} was initialized using a uniform distribution with values between -1 and 1. All other weights were initialized using a uniform distribution with values between -0.001 and 0.001. This means that the initial Q-function for both FERL and EERL will grow with the number of hidden nodes, with a faster rate for FERL. To ensure that the Q-values were initialized within an appropriate range, we used the scaling technique we proposed in Elfwing et al. (2013), by setting Z to approximately twice the initial Q-values without scaling (assuming a maximum reward of 1), i.e., $Q = -F/Z$ for FERL and $Q = -\langle E \rangle / Z$ for EERL.

Tang and Sutskever (2011) proposed a simple solution to improve standard RBM learning for classification problems with more active than non-active states. They normalized the training set by removing the mean of each input before learning. This normalization technique is not feasible in a reinforcement learning context where the state vectors are not known in advance. In this study, we propose an alternative normalizing method to improve the performance of RBM function approximation in tasks where the state input consists of more active than non-active states, the number active states varies greatly for different state vectors, and the state vector consists of continuous values. For each state vector, we remove the mean of the state vector from each of its elements:

$$z_i = s_i - \frac{\sum_{j=1}^{I} s_j}{I}. \tag{30}$$

Normalization by replacing s_i with z_i does not change any other parts of the FERL and EERL learning algorithms (hereafter, denoted *FERL-ZM* and *EERL-ZM*).

3. Experiments

To evaluate the proposed EERL method, we compare the performance with FERL, linear function approximation (hereafter, *linRL*), and function approximation using a two-layered feedforward neural network (hereafter, *NNRL*). For linRL, the approximated Q-values are computed by the weighted sum of the state vector elements, $Q(\boldsymbol{s}, a) = \sum_i s_i w_{ia}$, with derivatives with respect to the weight parameters computed as $\nabla_{w_{ia}} Q(\boldsymbol{s}, a) = s_i$. For NNRL, The state nodes s_i of the neural network are connected to K hidden nodes by weights w_{ik}. The hidden nodes have sigmoid activation functions, $\delta_k = \sigma(\sum_i w_{ik} s_i)$. The hidden nodes are connected to Q-value output nodes with linear activation by weights w_{ka}. The approximated Q-values are computed as the weighted sum of the hidden activations ($Q(\boldsymbol{s}, a) = \sum_k w_{ka} \delta_k$), with derivatives with respect to the weight parameters computed as

$$\nabla_{w_{ik}} (Q(\boldsymbol{s}, a)) = \delta_k (1 - \delta_k) w_{ka} s_i, \tag{31}$$

$$\nabla_{w_{ka}} (Q(\boldsymbol{s}, a)) = \delta_k. \tag{32}$$

We used a grid-like search for each method in each task to determine the appropriate values of the learning rate α and the temperature decay rate τ_k. The initial temperature τ_0 was set to 0.5 in all experiments.

3.1. Gridworld tasks

Fig. 3 shows the three versions of the gridworld task: MNIST (left panel), inverted MNIST (middle panel), and CIFAR10 (right panel). The agent started each episode at state '1' (airplane in the CIFAR10 task) and the goal of the task was to reach state '5' (deer) by moving counterclockwise along a path through states '2', '3', '6', '9', '8', '7', and '4' (car, bird, dog, ship, horse, frog, and cat). The agent received a small negative reward (-0.01) for premature state transitions to the absorbing goal state '5' (red lines in the left panel) and a positive reward ($+1$) for successful completion of the task, i.e., state transition from state '4' to state '5' (green line in the left panel). The rewards for all other state transitions were set to zero. There were four actions that moved the agent one step in the directions North, East, South, and West. If the agent moved into a wall (purple lines in the left panel), then the agent remained in the current state. In the MNIST task, each state consisted of a handwritten digit from the MNIST data set (LeCun et al., 1998). The 28×28 pixels (the dimension of the state vector was 784) grayscale images were scaled to the range [0; 1] by dividing the pixel values by 255. For each state, we used 20 different digit images that were randomly selected from the MNIST training data set. At the start of each episode, the image for each state was randomly selected among the 20 possible images. An episode ended either when the agent moved to the absorbing state (state '5') or after a maximum number of steps (set to 1000). The inverted MNIST task was identical to the MNIST task, except that the state images were inverted, i.e., $1 - pv$, where pv was the scaled pixel values in the MNIST task. In the CIFAR10 task, the states consisted of 32×32 pixels RGB images from the CIFAR10 data set (Krizhevsky, 2009) (the dimension of the state vector was $32 \times 32 \times 3 = 3072$). As in the MNIST tasks, we randomly selected 20 images of each class from the training data set and in the beginning of each episode, the image for each state was randomly selected among the 20 possible images. To validate our proposed normalization technique, we performed experiments for the inverted MNIST task and the CIFAR10 task where the mean of each state vector was subtracted from each of its elements, see (30). For the CIFAR10 task, the normalization was done separately for the red, green, and blue channels in the RGB images. The number of hidden nodes was set to 20, γ was set to 0.96, and λ was set to 0.8 for all methods in the three tasks. The determined values of α and τ_k are shown in Table 1.

The performance of EERL was compared with FERL, NNRL, and linRL, except for the CIFAR10 task, which linRL could not handle. Fig. 4 shows the average success rate, i.e., the agent reached the absorbing goal state using the correct path and received a positive reward of $+1$ (top row), and the average number of steps to the goal (bottom row), computed over 10 simulation runs and 100 episodes. The results are summarized in Table 2 as the average number of episodes to reach a success rate of 95%, the average number of episodes to reach a success rate of exactly 100%, and the average number of steps to the goal in the final 100 episodes ($\pm$ standard deviation) of the learning processes.

EERL (with the proposed normalization, EERL-ZM, for the inverted MNIST and the CIFAR10 tasks) outperformed the other methods. EERL had the fastest convergence to both 95% and 100% success rates, as well as significantly ($p < 0.001$) fewer steps to the goal in the final 100 episodes of learning, in all three tasks. Compared with FERL, the learning speed of EERL (measured by the

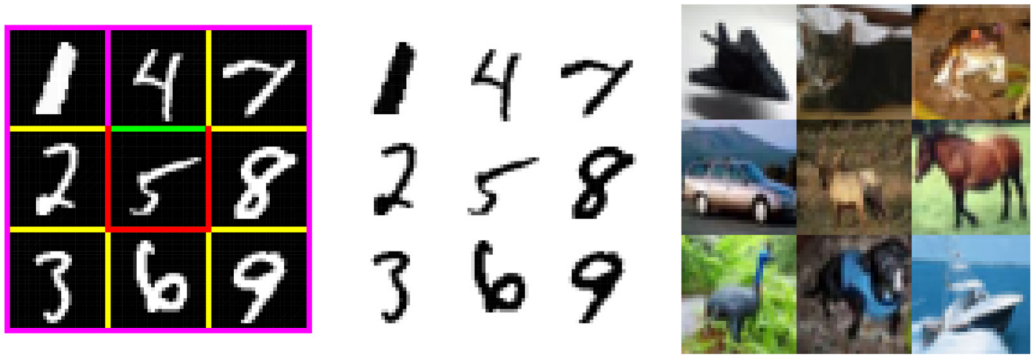

Fig. 3. The three versions of the gridworld task: MNIST (left panel), inverted MNIST (middle panel), and CIFAR10 (right panel). At the beginning of each episode the image for each state was randomly selected among 20 possible images.

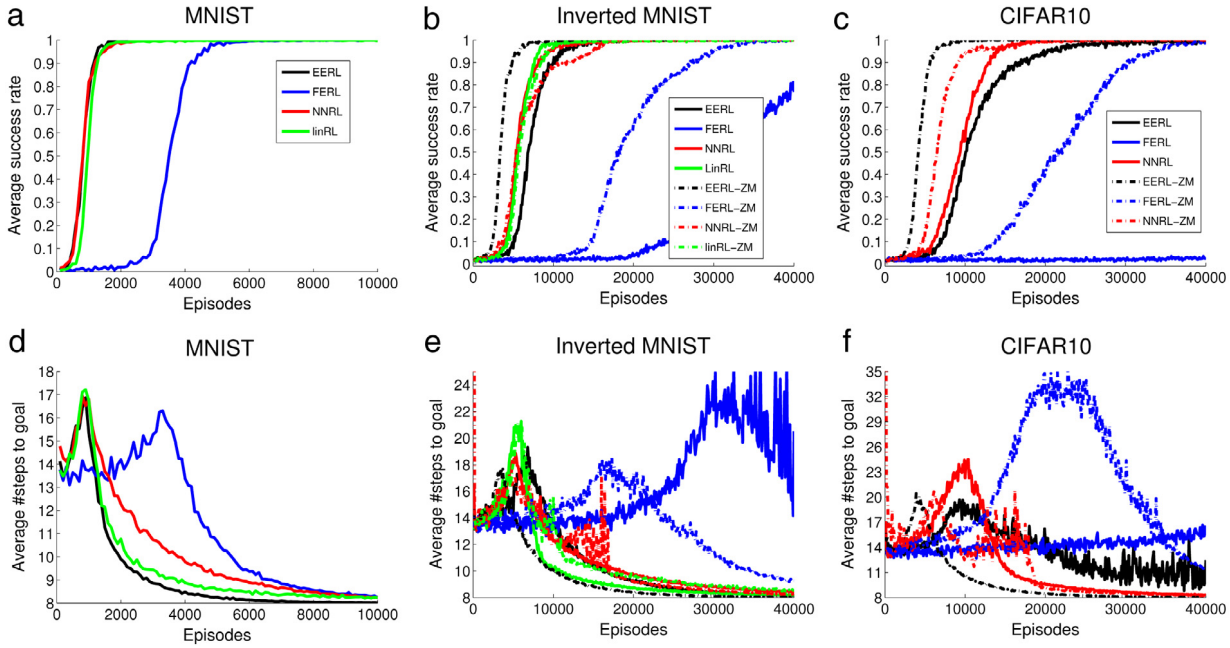

Fig. 4. The average success rate (top row), i.e., the agent reached the absorbing goal state using the correct path and received a positive reward of +1, and the average number of steps to the goal (bottom row), computed over 10 simulation runs and 100 episodes in the three version of the gridworld task. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

time to reach an average success rate of 95%) was more than three times faster in the MNIST task and, compared with FERL-ZM, the learning speed of EERL-ZM was more than five times faster in the inverted MNIST task and the CIFAR10 task.

The proposed normalization technique greatly improved the learning performance of both RBM function approximation methods in the inverted MNIST task and the CIFAR10 task. Without normalization, EERL performed similarly to NNRL and linRL in the inverted MNIST task and significantly worse than NNRL in the CIFAR10 task. In the CIFAR10 task, the learning became unstable, with large variance in the number of steps to the goal in later stage of the learning process (see the solid black line in Fig. 4(f)). For FERL without normalization, there was almost no learning progress at all in the CIFAR10 task and the average success rate at the end of the 40 000 episodes of learning was only approximately 80% in the inverted MNIST task. In contrast, the proposed normalization had little, and mostly negative, effect on the learning for NNRL and linRL. For NNRL, the normalization made the learning more unstable during the first half of the learning process (see the dashed red lines in Fig. 4(e) and (f)). For linRL, normalization made the learning slower and the final learning performance was significantly worse (see dashed green line in Fig. 4(e)).

3.2. SZ-Tetris

Stochastic SZ-Tetris (Burgiel, 1997) was proposed as a benchmark for reinforcement learning by Szita and Szepesvári (Szita & Szepesvári, 2010). It is played on a board of standard Tetris size with a width of 10 and a height of 20. In each step, either an S-shaped tetromino or a Z-shaped tetromino appears with equal probability. The agent can select a rotation (lying or standing) and a horizontal position within the board. In total, there are 17 possible actions for each tetromino (9 standing and 8 lying horizontal positions). After the action selection, the tetromino is dropped down the board, stopping when it hits another tetromino or the bottom of the board. If a row is completed, then it disappears. The agent gets a score of +1 point for one completed row and a score of +2 points for two completed rows. The game ends when a tetromino does not fit within the board.

SZ-Tetris preserves the core challenges of regular Tetris but allows the evaluation of different strategies within a feasible time frame. Several factors contribute to make Tetris a difficult problem for reinforcement learning algorithms, such as the relatively large number of action that can be selected in each state, the stochasticity in the state transitions, and that improved

Table 1
The determined values α and τ_k for each method in the three gridworld tasks.

Method	α	τ_k
MNIST task		
EERL	$0.01 \times Z$	0.01
FERL	$0.01 \times Z$	0.01
NNRL	0.01	0.005
linRL	0.001	0.01
Inverted MNIST task		
EERL	$0.001 \times Z$	0.002
EERL-ZM	$0.001 \times Z$	0.002
FERL	$0.001 \times Z$	0.002
FERL-ZM	$0.001 \times Z$	0.002
NNRL	0.001	0.001
NNRL-ZM	0.001	0.001
linRL	0.0001	0.002
linRL-ZM	0.0001	0.002
CIFAR10 task		
EERL	$0.001 \times Z$	0.001
EERL-ZM	$0.001 \times Z$	0.001
FERL	$0.001 \times Z$	0.001
FERL-ZM	$0.001 \times Z$	0.001
NNRL	0.001	0.001
NNRL-ZM	0.001	0.001

Table 2
The average number of episodes to reach a success rate of 95% (second column) and exactly 100% (third column), and the average number of steps to the goal in the final 100 episodes ($\pm$ standard deviation; fourth column) for each method in each of the three gridworld tasks.

Method	Success rate		Steps to goal final 100 epi
	95%	100%	
MNIST task			
EERL	**1 300**	**3 000**	**8.02 ± 0.12**
FERL	4 600	–	8.26 ± 0.63
NNRL	1 500	5 300	8.24 ± 0.51
linRL	1 400	–	8.25 ± 0.56
Inverted MNIST task			
EERL	11 200	35 700	8.17 ± 0.45
EERL-ZM	**5 800**	**17 900**	**8.03 ± 0.20**
FERL	–	–	20.5 ± 6.11
FERL-ZM	30 300	–	9.28 ± 1.86
NNRL	9 400	39 600	8.40 ± 0.74
NNRL-ZM	14 800	36 800	8.27 ± 0.57
linRL	8 300	36 000	8.18 ± 0.49
linRL-ZM	9 600	–	8.53 ± 1.22
CIFAR10 task			
EERL	22 000	–	12.5 ± 51.3
EERL-ZM	**6 300**	**17 500**	**8.03 ± 0.17**
FERL	–	–	15.4 ± 13.4
FERL-ZM	33 500	–	11.7 ± 4.18
NNRL	14 200	39 800	8.33 ± 0.66
NNRL-ZM	12 600	36 500	8.25 ± 0.56

performance increases the episode length. For an alternating sequence of S-shaped and Z-shaped tetrominos, the upper bound of the episode length is 69 600 (Burgiel, 1997) (corresponding to a score of 27 840 points), but the maximum episode length is probably much shorter, maybe a few thousands (Szita & Szepesvári, 2010). That means that to evaluate a good strategy, SZ-Tetris requires at least 100,000 times less computation than regular Tetris.

The standard learning approach for Tetris is to use a model-based learning setting and define the evaluation function or state-value function as the linear combination of hand-coded features. Using this approach, value-based reinforcement learning algorithms have a lousy track record in the Tetris domain. In regular Tetris, their reported performance levels are many magnitudes lower than black-box methods such as the cross-entropy method (CEM) and evolutionary approaches. For stochastic SZ-Tetris, the reported scores for a wide variety of reinforcement learning algorithms are either approximately zero (Szita & Szepesvári, 2010) or in the single digits.[1] Faußer and Schwenker (2013) used TD(λ) and a two layer neural network with 5 hidden nodes (i.e., NNRL) as an alternative approach to linear function approximation. They achieved a score of about 130 points for a single network, which is slightly worse than the reported performance of 133–138 points for CEM. Using an ensemble of 10 networks and average decisions, they achieved the current state-of-art performance of about 150 points for stochastic SZ-Tetris.

In this study, we compared the performance of EERL, FERL, and NNRL in three learning settings:

1. **TD(λ) with state features**. Model-based learning setting with 20 state features similar to the original Bertsekas and Ioffe features (Bertsekas & Ioffe, 1996): 10 column heights (10×21 binary states), 9 relative column height differences that were capped at ± 5 (9×11 binary states), and the number of holes (151 binary states, i.e., the number was capped at 150 holes).
2. **Sarsa(λ) with state features**. Model-free learning setting with the same state features as for TD(λ), with the addition of the current tetromino (2 binary states). There was 34 possible actions (i.e., action nodes in EERL and FERL, and Q-value nodes in NNRL), 17 for each tetromino. In the model-free learning setting, an episode ended when the selected position and rotation of a tetromino did not fit within the board, i.e., the possible actions were not limited to the actions that fit within the board as in the model-based setting.
3. **TD(λ) with board states**. Model-based learning setting where the state vector was equal to the board state, i.e., a state node was set to 1 if the corresponding board cell was occupied by a tetromino and set to 0 if the corresponding board cell was empty. To be able to handle that the number active state nodes varied dramatically between different states (from zero active state nodes when the board was empty to a majority of active state nodes at the end of the episodes), we used our proposed normalization technique in this learning setting for the three methods.

In all three learning settings, we used the same reward function as in Faußer and Schwenker (2013):

$$r(\boldsymbol{s}) = e^{-(\text{number of holes in } \boldsymbol{s})/33}. \tag{33}$$

The end of an episode was an absorbing state, in which the agent received a 0 reward. The number of hidden nodes was set to 50, γ was set to 0.99, and λ was set to 0.8 for EERL, FERL, and NNRL in all three learning settings.

Fig. 5 shows the average score computed over every 1000 learning episodes and 5 simulation runs. Table 3 summarizes the final average performance over the last 1000 episodes for the three methods in the three learning settings, as well as the determined values of α and τ_k.

In the model-based learning setting with state features (see Fig. 5(a)), the three methods reached very similar final average performance levels after 200 000 episodes, scores of slightly more than 200 points. The learning speed of NNRL was faster, reaching the final performance level after about only 50 000 episodes. EERL and FERL needed about 120 000 episodes to reach close to the final performance level. The scores of above 200 points are large improvements over the previous state-of-the-art learning result of about 150 points achieved by an ensemble of 10

[1] http://barbados2011.rl-community.org/program/SzitaTalk.pdf.

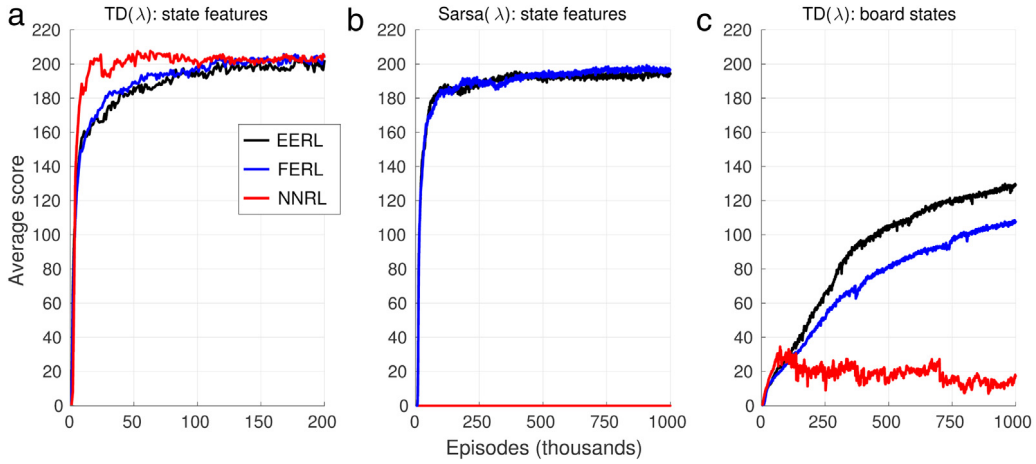

Fig. 5. The average score computed over every 1000 episodes and 5 simulation runs in the three learning settings for SZ-Tetris.

Table 3
Average scores (± standard deviation) computed over the final 1000 episodes and 5 simulations runs for SZ-Tetris (fourth column), as well as the determined values of α (second column) and τ_k (third column).

Method	α	τ_k	Final mean score
TD(λ) with state features			
EERL	$0.01 \times Z$	0.00025	202.2 ± 66.7
FERL	$0.01 \times Z$	0.00025	$\mathbf{204.5 \pm 61.5}$
NNRL	0.01	0.00025	203.7 ± 67.0
Sarsa(λ) with state features			
EERL	$0.01 \times Z$	0.00025	194.8 ± 52.0
FERL	$0.01 \times Z$	0.00025	$\mathbf{196.0 \pm 53.3}$
NNRL	0.01	0.00025	0
TD(λ) with board states			
EERL-ZM	$0.001 \times Z$	0.00025	$\mathbf{129.6 \pm 50.9}$
FERL-ZM	$0.001 \times Z$	0.00025	108.1 ± 44.5
NNRL-ZM	0.001	0.00025	17.7 ± 16.6

neural networks. The results were also achieved in much shorter learning time. The neural networks in the previous study reached their final performance levels after 5 million episodes. The large improvement in performance and learning speed in our NNRL implementation, compared with the implementation in Faußer and Schwenker (2013), can probably be explained by that we used 10 times as many nodes in the hidden layer and more efficient exploration by softmax action selection (compared with ϵ-greedy used in the their study). The scores of above 200 points are also higher than the score of 182 points reported for the hand-coded policy proposed by Szita and Szepesvári (2010) as a baseline result for the learning of good strategies in stochastic SZ-Tetris:

> *It divides the board to 5 two-blocks-wide columns; it puts only S-pieces in column 1 and 2, only Z-pieces in column 4 and 5, and tries to preserve the type of column 3, changing only if max. height is above 15.*

In the two more challenging learning settings (see Fig. 5(b) and (c)), EERL and FERL clearly outperformed NNRL. NNRL could not handle the model-free learning setting and achieved an average score of 0 points. In the model-based setting with board states, NNRL-ZM learning was unstable and it could only achieve a final performance of 17.7 points after 1 million episodes. The performances of EERL and FERL were particularly impressive in the model-free learning setting. The scores of 195 and 196 points achieved after 1 million episodes are more than 10 points higher than the score for the hand-coded policy described above, and only less than 10 points lower than the results in the model-based setting. In contrast to the settings with hand-coded state feature where the learning curves for EERL and FERL were almost identical, EERL-ZM clearly outperformed FERL-ZM in the model-based learning setting with board states. The score of about 130 points achieved by EERL-ZM is the same as the score of 130 points achieved by Faußer and Schwenker (2013) for a single network with state features.

The SZ-Tetris experiments show that FERL can perform as well as EERL, but only in specific task settings where the state vector has the following three characteristics: (1) binary values; (2) the number of active nodes is much fewer than the number of non-active nodes; and (3) the number of active nodes does not change between states. The SZ-Tetris experiments with state features fulfilled these characteristics since, in the model-based setting, the state vectors consisted of exactly 20 ones and 440 zeros, and, in the model-free setting, the state vectors consisted of exactly 21 ones and 441 zeros.

To investigate the learned policies, we used an alternating sequence of S- and Z-tetrominos. Fig. 6 shows the learned policy for the best EERL solutions found in the model-free (final mean score of 206 points) and model-based (222 points) learning settings with state features, as well as the corresponding expected hidden activations in the model-based learning setting. In the model-free setting (Fig. 6(a)), the learned policy was very similar to the hand-coded policy described above. The board was divided into 5 two-blocks-wide columns. Standing S-tetrominos were placed in columns 5:6 and 7:8, and standing Z-tetrominos were placed in columns 1:2 and 9:10. The type of tetromino that was placed in column 3:4 was switched approximately every 100 trials, creating two holes for every switch.

The learned model-based policy was more complex and interesting. In this case, the board was also divided into 5 two-blocks-wide columns. However, no column was dedicated to a single tetromino during the whole episode. Instead, it tried to switch the type of tetromino that used three columns without creating any holes. A switch was often initialized by placing a standing tetromino across two of the five two-blocks-wide columns and accomplished over several trials. Fig. 6(b) and (c) show a clear correlation between switches in policy (indicated by the horizontal lines) and changes in the hidden node activation pattern. The hidden node activation pattern was sparse and approximately binary. In most cases, a hidden node became active (changed its activation from approximately zero to approximately one) after a switch in policy and remained active until a later switch in policy.

Fig. 7 visualizes the switch that occurred between trial 107 and 113. The switch was accomplished by, first, placing a Z-tetromino in column 8:9 (trial 107), second, reducing the height of column 7:8

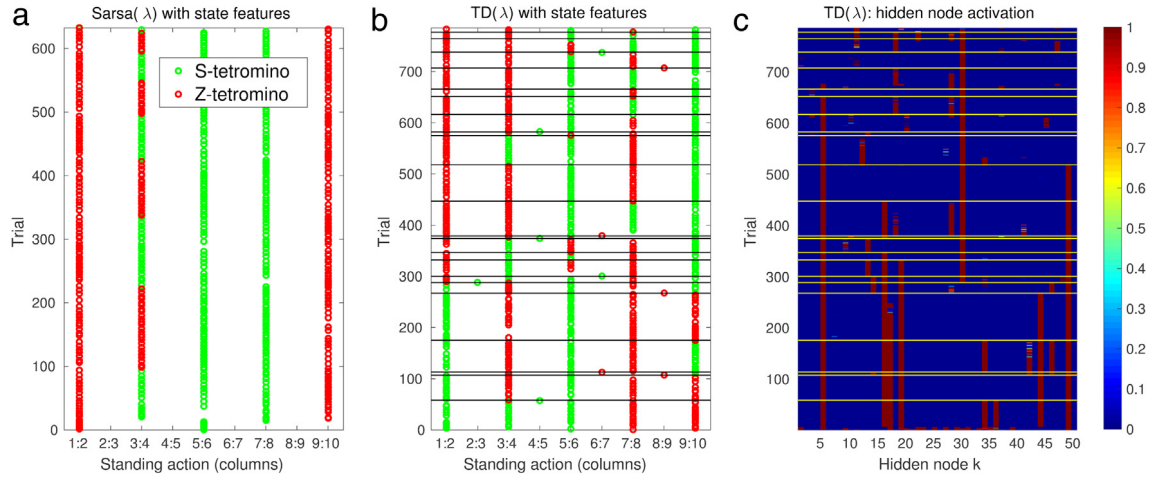

Fig. 6. Learned policy for an alternating sequence of S- and Z-tetrominos for the best EERL solutions in the model-free (a) and model-based (b) learning settings with state features, as well as the expected hidden node activations in the model-based learning setting (c). Lying actions were only used just before the end of episodes and are not shown in figures. The horizontal lines in figures (b) and (c) indicate switches in policy in the model-based learning setting.

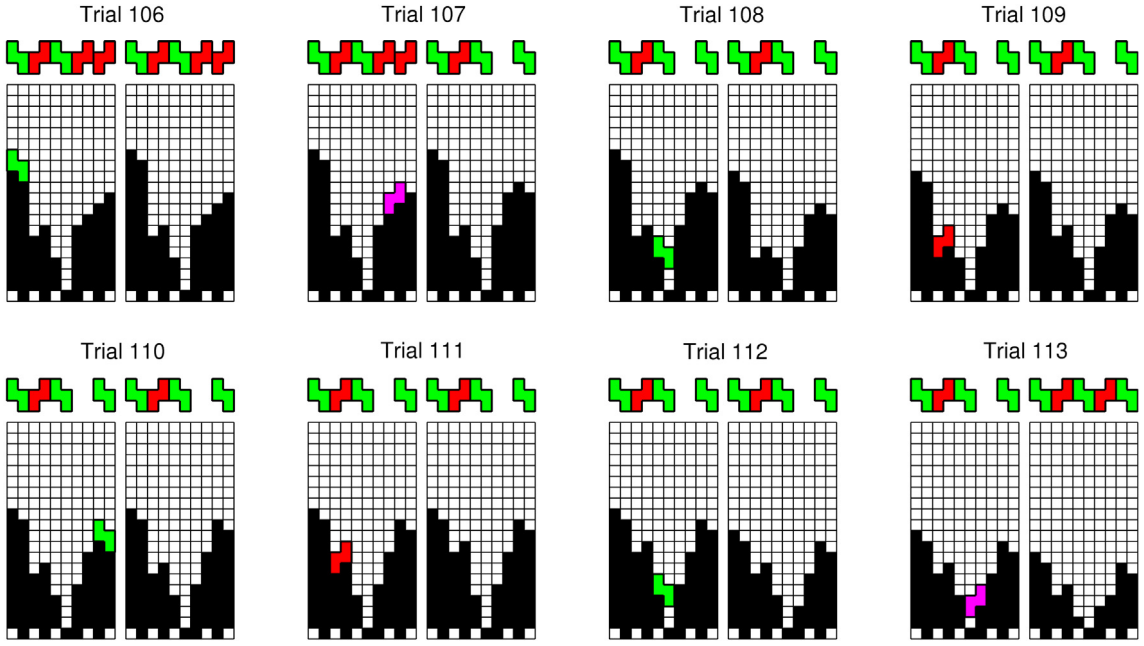

Fig. 7. Visualization of a switch of the type of tetromino that used three two-blocks wide columns learned by the best EERL network in the model-based setting with state features (trials 107–113). For each trial, the left panel shows the landing position of the current tetromino and the right panel shows the board state after the action. The tetrominos shown above the board indicate the current type for each column. S-tetrominos are shown in green and Z-tetrominos are shown in red, except for the Z-tetrominos that were placed across two of the five two-blocks-wide columns in the beginning (trial 107) and the end (trial 113) of the switch, which are shown in purple. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

by 4 by placing S-tetrominos twice in column 5:6 (trials 108 and 112), and, third, placing a Z-tetromino in column 6:7 (trial 113). This switch changed the type that used three columns from the Z-tetromino to the S-tetromino, while, in the process, removing 6 lines and reducing the maximum column height for the tetromino that used two columns from 14 to 5.

3.3. Robot navigation

For the robot navigation task, we used a simulation environment that was developed to mimic the properties of the Cyber Rodent robot (Doya & Uchibe, 2005). The Cyber Rodent is a small mobile robot, 22 cm in length and 1.75 kg in weight. The robot has a variety of sensors, including an omnidirectional C-MOS camera, an infrared range sensor, seven infrared proximity sensors, gyros, and accelerometers. It has two wheels and a maximum speed of 1.3 ms^{-1}.

Fig. 8(a) shows the robot visual navigation task that we introduced in Elfwing et al. (2013). The goal of the task is to navigate to one of the two goal areas (in the southwest and the northeast corners, see dashed quarter circles in Fig. 8(a)) of the 2.5×2.5 m experimental area, by learning to infer the correct goal area by the color of the upper part of the four landmarks. If the color of upper part of the landmarks is green (as shown in Fig. 8), then the correct goal area is in the southwest corner, and if the color is blue, then the correct goal area is in the northeast corner. At the start of each episode, the correct goal area is randomly changed, and the robot is randomly placed in one of the four starting areas (dotted rectangles in Fig. 8(a)). The initial position within the starting area and the robot's initial heading angle are also randomly selected.

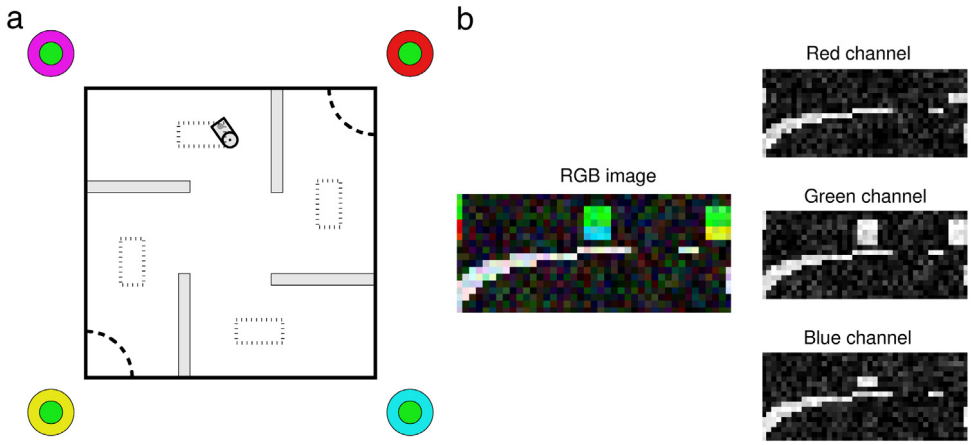

Fig. 8. (a) Overview of the experimental area for the navigation tasks. The dashed quarter circles at the southwest and northeast corners in the left panel indicate the two goal areas and the dotted rectangles indicate the four starting areas. The circles outside the experimental area indicate the four landmarks. The color of the lower part of each landmark (the larger circles) was unique and non-changing. The color of the upper part of all landmarks (the smaller circles) corresponded to the correct goal area and was randomly changed at the start of each episode. Note that the difference in radius between the lower and the upper part of the landmarks is only for illustrative purposes. In the experiments, both parts of the landmarks had the same radius. (b) an example of a noisy RGB camera image taken at the robot's position in (a), with corresponding red, green and blue channels. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 4
The RGB-values for the different objects in the binary RGB setting and the mean values in the noisy RGB setting. In the noisy setting, Gaussian noise with a standard deviation of 0.1 was added to the red, green, and blue channels for each pixel.

Object	RGB-values	
	Binary	Noisy (mean)
Upper landmarks (SW goal)	(0; 1; 0)	(0.1; 0.9; 0.1)
Upper landmarks (NE goal)	(0; 0; 1)	(0.1, 0.1, 0.9)
Lower SW landmark	(1; 1; 0)	(0.9; 0.9; 0.1)
Lower NW landmark	(1; 0; 1)	(0.9; 0.1; 0.9)
Lower NE landmark	(1; 0; 0)	(0.9; 0.1; 0.1)
Lower SE landmark	(0; 1; 1)	(0.1; 0.9; 0.9)
Obstacles	(1; 1; 1)	(0.9; 0.9; 0.9)
Background	(0; 0; 0)	(0.1; 0.1; 0.1)

Table 5
The average number of steps to goal (± standard deviation) computed over the final 200 episodes for the different learning settings in the robot navigation task.

Method	Final performance		
	9 Actions	25 Actions	100 Actions
Binary RGB-values			
EERL	**23.9 ± 9.1**	**30.4 ± 14.5**	**30.4 ± 13.3**
FERL	25.3 ± 10.1	127 ± 351	569 ± 821
NNRL	30.1 ± 12.3	33.9 ± 17.7	459 ± 721
Noisy RGB-values			
EERL	29.6 ± 15.0	**37.4 ± 44.0**	**35.5 ± 20.4**
FERL	**28.9 ± 15.5**	319 ± 634	810 ± 867
NNRL	42.8 ± 28.3	288 ± 615	1504 ± 782

The color of the lower part of each landmark is unique and non-changing, and can therefore be used for localization.

In our earlier study, we assumed that the robot was equipped with a perfect color blob detection system. The state vector was constructed by extracting a binary camera for each predefined color that the robot could detect. In this study, we used a more challenging and interesting task setting, where the state vector consisted of raw and noisy RGB camera images. To investigate how well EERL, FERL, and NNRL could handle noisy state input, we compared a binary setting (the RGB-values were set to either zero or one) with a setting where the RGB-values were sampled from a normal distribution with a standard deviation of 0.1 and a mean of either 0.9 or 0.1 (see Table 4). Fig. 8(b) shows an example of a noisy camera image and the three color channels corresponding to the robot's position in Fig. 8(a). The robot's simulated camera had a resolution of 738 (41×18) pixels covering a horizontal field of view of $\pm 75^\circ$, with a 3.75° distance between the pixels. Within the field of view, the landmarks were visible from all distances and the obstacles were visible up to 2 m. The size of an object in the camera image increased with the inverse of the distance to the object. In addition, the state vector consisted of three normalized real-valued distance measures from the robot's front proximity sensors, located at -30°, 0°, and $+30^\circ$ in relation to the robot's heading direction. The distance information was normalized to the range [0; 1] and higher values corresponded to shorter distances. The total length of the state vector was 2217 ($41 \times 18 \times 3 + 3$).

In our earlier study, we used, as is common in action-value based reinforcement learning, a small number of actions (velocities of the right and the left wheels) that were selected in a rather ad-hoc manner. In this study, we instead defined a list of possible wheel velocities that was common for both wheels. An additional purpose of the robot navigation experiment was to investigate how well EERL, FERL, and NNRL could handle an increased number of actions. We performed 3 sets of experiments with 9 actions (possible wheel velocities of $[-25, 25, 45]$ cm/s), 25 actions ($[-45, -25, 5, 25, 45]$ cm/s), and 100 actions ([−45:10:45][2] cm/s). Gaussian noise was added to each wheel velocity, with zero mean and a standard deviation equal to 1% of the amplitude of the velocity.

An episode ended either when the robot moved its head inside the correct goal area or when the length of the episode exceeded a fixed threshold of 2000 time steps. The robot received a +1 reward if it reached the correct goal area, otherwise the reward was set to 0. The number of hidden nodes was set to 50, $\gamma = 0.98$, and $\lambda = 0.8$. The values of α and τ_k were determined in the noisy RGB setting with 9 actions: $0.005 \times Z$ and 0.01 for EERL, $0.01 \times Z$ and 0.005 for FERL, and 0.005 and 0.005 for NNRL.

Fig. 9 shows the average number of steps to goal computed over every 200 episodes and 5 simulation runs for EERL, FERL, and NNRL in the binary and the noisy RGB settings for robots with 9, 25, and 100 actions. The average performance computed over the final 200 episodes is summarized in Table 5.

EERL converged faster than FERL and NNRL in all six settings and the average final performance was significantly ($p < 0.001$)

[2] Matlab notation.

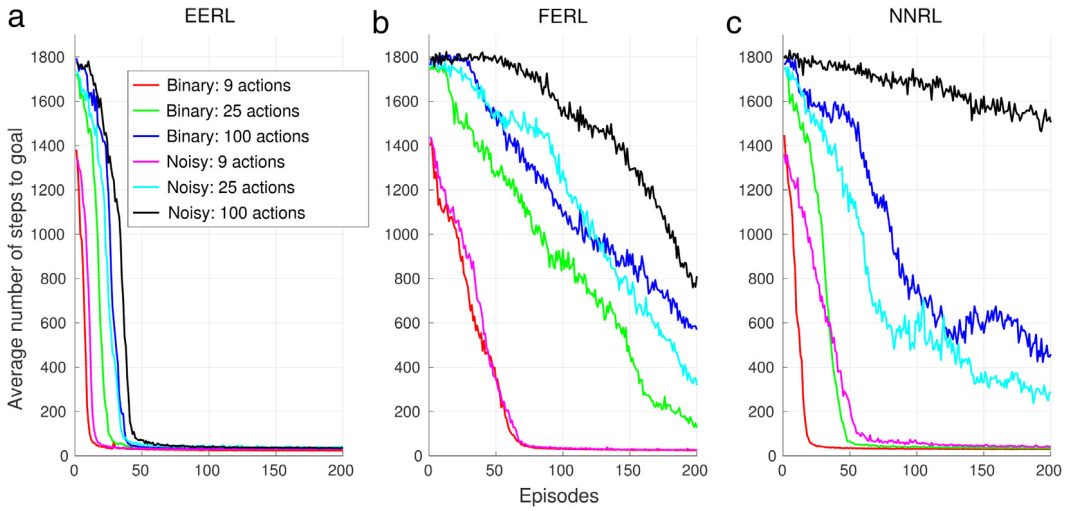

Fig. 9. The average number of steps to goal computed over every 200 episodes and 5 simulation runs in the robot navigation experiments in the binary and noisy RGB settings with 9, 25, and 100 actions for EERL (a), FERL (b), and NNRL (c).

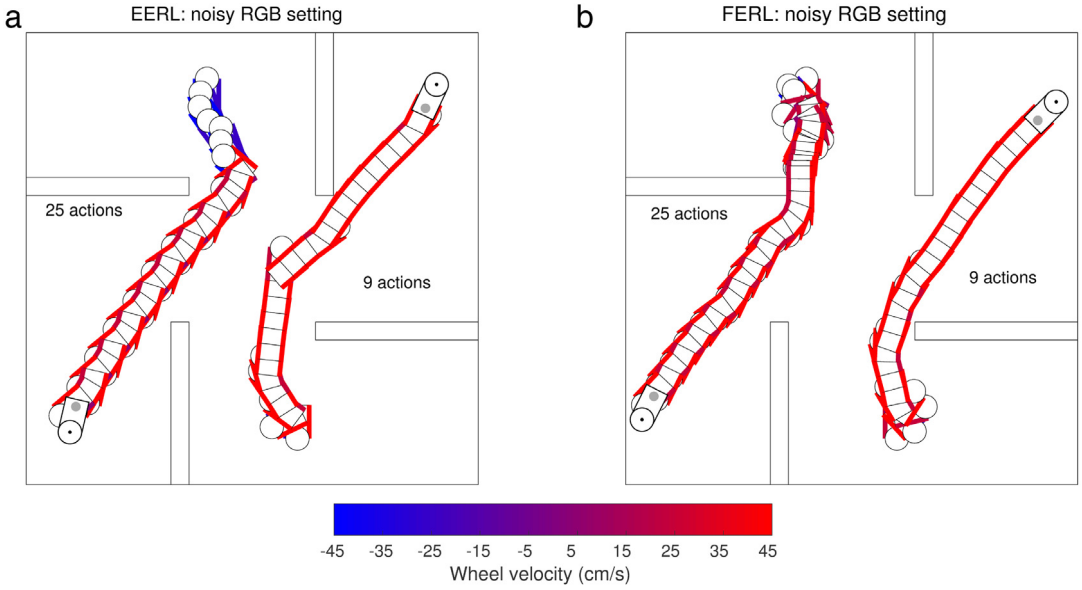

Fig. 10. Examples of learned trajectories for the best performing EERL (a) and FERL (b) agents in the noisy RGB setting with 9 (northeast goal) and 25 (southwest goal) actions. The color coding visualizes the velocities of the left and right wheels. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

better, except for in the noisy RGB setting with 9 actions where the average final performance achieved by FERL was slightly better than EERL, but not significantly so ($p = 0.31$). FERL and NNRL could only achieve a final average performance of less than 50 steps to goal in the settings with 9 actions, and, for NNRL, in the binary RGB setting with 25 actions. The performance of EERL was particularly impressive in the experiments with 100 actions. In both the binary and the noisy RGB settings, there were no significant difference in the average final performance between experiments with 25 and 100 actions.

Fig. 10 shows examples of learned trajectories by the best performing EERL (a) and FERL (b) agents in the noisy RGB setting with 9 (northeast goal) and 25 (southwest goal) actions. Fig. 11 shows the corresponding expected hidden activation patterns, which display clear differences between EERL and FERL. For FERL, the activation patterns were sparse and close to binary, with, in most states, a few active hidden nodes with activations close to 1 and a large majority of non-active hidden nodes with activations close to 0. In contrast, for EERL, the activation patterns were dense with large number of nodes with activations closer to 0.5 than 0 or 1. In the 9 actions setting, the learned behaviors of EERL and FERL were similar. In the 25 actions setting, there was one distinct difference in the learned initial behavior. For FERL, the agent learned, similar to the 9 actions setting, to immediately start to rotate to face the target. In contrast, for EERL, the agent learned to initially navigate backwards until it reached the opening to the center square and then it started to rotate to face the target. The attractiveness of initial backward navigation can be explained by that the possible number of pixels in the image corresponding to the closest landmark was larger (compared to the landmark closest to the target) and, therefore, provided a better guide for navigation to the opening to the center square. In the 9 actions setting, there was only one action with negative velocity (moving straight back with a velocity of -25 cm/s) and backward navigation was therefore not an option. Initial backward navigation was also observed for EERL in the 100 actions setting.

These results suggest that richer neural encoding is one explanation for the higher and more stable performance achieved

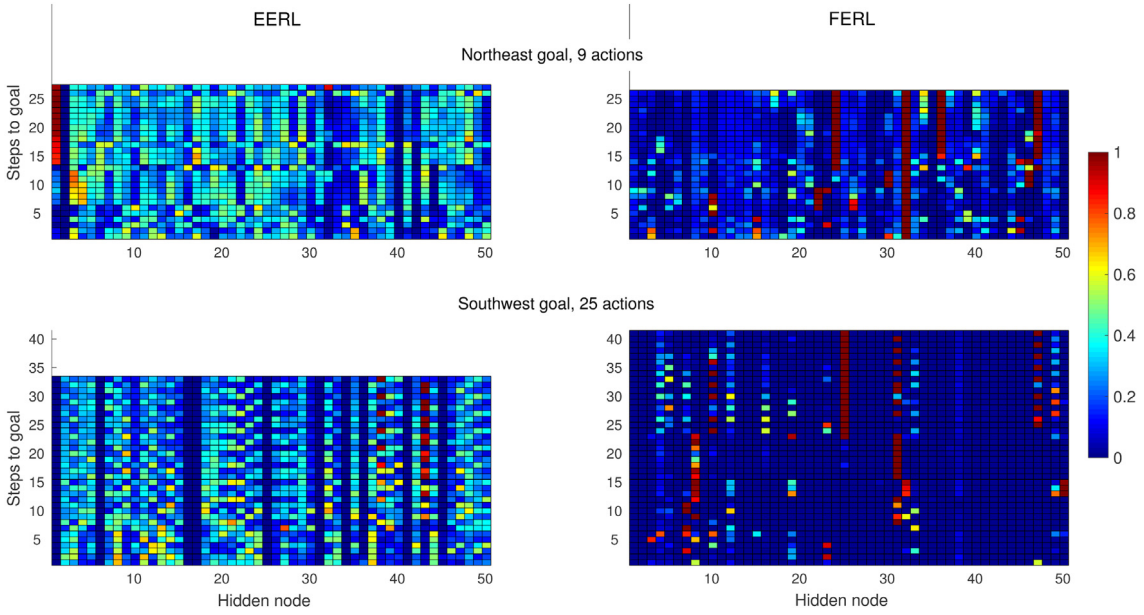

Fig. 11. Expected hidden activations along the learned trajectories shown in Fig. 10.

by EERL. For example, in the noisy RGB setting with 25 actions, the final mean performance for FERL varied from 47 to 639 steps to goal. In comparison, the final mean performance for EERL varied only from 30 to 46 steps to goal.

4. Conclusions

In this study, we proposed that RBM function approximation for reinforcement learning can be significantly improved by approximating the value function by the negative expected energy, instead of the negative free energy. We validated this approach by showing that EERL: (1) outperformed FERL, as well as NNRL and linear function approximation, for three versions of a gridworld tasks that tested learning in high-dimensional state spaces with more non-active than active states, more active than non-active states, and continuous state input; (2) achieved new state-of-the-art results in stochastic SZ-Tetris in both model-free (Sarsa(λ)) and model-based (TD(λ)) learning settings; and (3) significantly outperformed FERL and NNRL in a visual robot navigation task with raw and noisy RGB images as state input and a large number of actions, both in terms of learning speed and final performance.

Acknowledgments

This work was supported by project commissioned by the New Energy and Industrial Technology Development Organization (NEDO), KAKENHI Grant 23120007 from Ministry of Education, Culture, Sports, Science and Technology (MEXT), and Okinawa Institute of Science and Technology Graduate University research support to KD.

References

Bertsekas, D. P., & Ioffe, S. (1996). *Temporal differences based policy iteration and applications in neuro-dynamic programming. Technical Report LIDS-P-2349*. MIT.
Burgiel, H. (1997). How to lose at Tetris. *Mathematical Gazette*, *81*, 194–200.
Doya, K., & Uchibe, E. (2005). The cyber rodent project: Exploration of adaptive mechanisms for self-preservation and self-reproduction. *Adaptive Behavior*, *13*(2), 149–160.
Elfwing, S., Otsuka, M., Uchibe, E., & Doya, K. (2010). Free-energy based reinforcement learning for vision-based navigation with high-dimensional sensory inputs. In *Proceedings of the international conference on neural information processing, ICONIP2010* (pp. 215–222).
Elfwing, S., Uchibe, E., & Doya, K. (2013). Scaled free-energy based reinforcement learning for robust and efficient learning in high-dimensional state spaces. *Frontiers in Neurorobotics*, *7*(3).
Elfwing, S., Uchibe, E., & Doya, K. (2015). Expected energy-based restricted boltzmann machine for classification. *Neural Networks*, *64*(3), 29–38.
Faußer, S., & Schwenker, F. (2013). Neural network ensembles in reinforcement learning. *Neural Processing Letters*, 1–15.
Freund, Y., & Haussler, D. (1992). Unsupervised learning of distributions on binary vectors using two layer networks. In J. E. Moody, S. J. Hanson, & R. P. Lippmann (Eds.), *Advances in neural information processing systems, Vol. 4*. Morgan Kaufmann.
Heess, N., Silver, D., & Teh, Y.W. (2012). Actor-critic reinforcement learning with energy-based policies. In *JMLR workshop and conference proceedings, EWRL 2012, Vol. 24* (pp. 43–58).
Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, *12*(8), 1771–1800.
Krizhevsky, A. (2009). *Learning multiple layers of features from tiny images. Tech. Rep.*
LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.
Otsuka, M., Yoshimoto, J., & Doya, K. (2010). Free-energy-based reinforcement learning in a partially observable environments. In *Proceedings of the European symposium on artificial neural networks ESANN2010* (pp. 541–545).
Rummery, G. A., & Niranjan, M. (1994). *On-line Q-learning using connectionist systems. Tech. Rep. CUED/F-INFENG/TR 166*. Cambridge University Engineering Department.
Sallans, B., & Hinton, G. E. (2004). Reinforcement learning with factored states and actions. *Journal of Machine Learning Research*, *5*, 1063–1088.
Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart, & J. L. McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Volume 1: Foundations*. MIT Press.
Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine Learning*, *3*, 9–44.
Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In D. S. Touretzky, M. C. Mozer, & M. E. Hasselmo (Eds.), *Advances in neural information processing systems, Vol. 8* (pp. 1038–1044). MIT Press.
Sutton, R. S., & Barto, A. (1998). *Reinforcement Learning: An Introduction*. MIT Press.
Szita, I., & Szepesvári, C. (2010). SZ-Tetris as a benchmark for studying key problems of reinforcement learning. In *ICML 2010 workshop on machine learning and games*.
Tang, Y., & Sutskever, I. (2011). *Data normalization in the learning of restricted boltzmann machines. Technical Report UTML-TR-11–2*. Department of Computer Science, University of Toronto.