Q-Networks for Binary Vector Actions*

Naoto Yoshida
Tohoku University
Aramaki Aza Aoba 6-6-01
Sendai 980-8579, Miyagi, Japan
`naotoyoshida@pfsl.mech.tohoku.ac.jp`

Abstract

In this paper reinforcement learning with binary vector actions was investigated. We suggest an effective architecture of the neural networks for approximating an action-value function with binary vector actions. The proposed architecture approximates the action-value function by a linear function with respect to the action vector, but is still non-linear with respect to the state input. We show that this approximation method enables the efficient calculation of greedy action selection and softmax action selection. Using this architecture, we suggest an online algorithm based on Q-learning. The empirical results in the grid world and the blocker task suggest that our approximation architecture would be effective for the RL problems with large discrete action sets.

1 Introduction

One of the big challenges in reinforcement learning (RL) is learning in high dimensional state-action spaces. Recent advances in deep learning technologies have enabled us to treat RL problems with the high-dimensional state space, and it achieved an impressive result in general game playing tasks (*e.g.* ATARI game plays) [1].

Even though several approaches are suggested for RL with continuous actions [2][3], RL with a large action space is still problematic, especially when we treat binary vectors as representations of the actions. The difficulty is that the number of actions exponentially grows as the length of the binary vector grows. Recently, several approaches have been used to tackle this problem. Sallans & Hinton suggested an energy-based approach in which restricted Boltzmann machines [4] were adopted in the algorithm and their free energy was used as the function approximator [5]. Heess *et al.* followed their energy-based approach and investigated natural actor-critic algorithms with energy-based policies by RBMs [6]. Although energy-based approaches are known to be effective in large discrete domains, exact action sampling is intractable due to the nonlinearity of the approximation architecture. Hence, an energy-based approach samples actions by Gibbs sampling.

However, the Gibbs sampling-based action selection is computationally expensive and requires careful tuning of the parameters. Also, because of the intractability of the exact sampling of greedy actions, no Q-learning-based online off-policy RL algorithm has so far been proposed for the large discrete action domain. From this background, we treat this issue and suggest novel architecture for the off-policy RL with the a large discrete action set.

---

*This paper was accepted at Deep Reinforcement Learning Workshop, NIPS 2015

2 Preliminaries

2.1 Markov Decision Process and Reinforcement Learning

The value-based reinforcement learning algorithms utilize the Markov decision process (MDP) assumption. The MDP is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, P, R \rangle$. $\mathcal{S}$ is the state set, $\mathcal{A}$ is the action set, P is the transition probability $P(s'|s, a)$, where $s' \in \mathcal{S}$ is the next state given a state-action pair (s, a). Finally R is the average reward function $R(s, a) = \mathbb{E}[r|s, a]$ and r is the reward sample.

In the value-based RL, the action-value function $Q^\pi(s, a)$ is defined by

$$Q^\pi(s, a) = \mathrm{E}_\pi\Big[\sum_{t=0}^{\infty} \gamma^t r_t \Big| s_0 = s, a_0 = a\Big], \tag{1}$$

here, $0 \leq \gamma < 1$ is the discount factor. In value-based RL, we look for the optimal policy π^* that maximizes the action-values for every state-action pair. Q-learning is an algorithm for finding the optimal policy in MDP [7], and the advantage of Q-learning is its off-policy property: the agent can directly approximate the action-value of an optimal policy π^* while following the other policy π.

Although Q-learning is guaranteed to approximate optimal action-values when we use the tabular functions in a discrete state-action environment [8], tabular function-based approaches become quickly inefficient for RL with large state-action spaces. Then, function approximations become necessary in such domains.

2.2 Q-learning with Function Approximation

In the Q-learning algorithm with function approximations, we approximate the optimal value function by the function $Q_\theta(s, a)$ where θ is the parameter of the function.

The gradient-based update of the function $Q_\theta(s, a)$ calculates the gradient of the error function

$$L = \frac{1}{2}(T - Q_\theta(s, a))^2, \tag{2}$$

where T is the target signal. Then, the gradient of the error is obtained by

$$\frac{\partial L}{\partial \theta} = -(T - Q_\theta(s, a))\frac{\partial Q_\theta(s, a)}{\partial \theta}. \tag{3}$$

The target signal in the Q-learning is $T = r + \gamma \max_{\hat{a}} Q_\theta(s', \hat{a})$ given a transition sample $\{s, a, r, s'\}$. Then the direction of the parameter update $\Delta\theta$ is given by

$$\Delta\theta \quad = \quad -\frac{\partial L}{\partial \theta} \tag{4}$$

$$= \quad \Big(r + \gamma \max_{\hat{a}} Q_\theta(s', \hat{a}) - Q_\theta(s, a)\Big)\frac{\partial Q_\theta(s, a)}{\partial \theta}. \tag{5}$$

The first term of the product in the second equality is called the TD error. Using this gradient, the stochastic gradient descent or more sophisticated gradient-based algorithms are used for approximating the optimal action-value function [9][10].

The Q-learning-based gradient requires the max operation of $Q_\theta(s, a)$ given a state. In the previous research with small discrete action sets, this max operation were tractable. However, if the actions are composed of binary vectors or factored representation [5][11], the number of total actions exponentially grows and quickly become intractable.

3 Proposed Method

In this study, we assume that the function approximation is done by the multi-layer perceptrons (MLPs) parameterized by θ. To efficiently calculate the max operations in Q-learning with a large
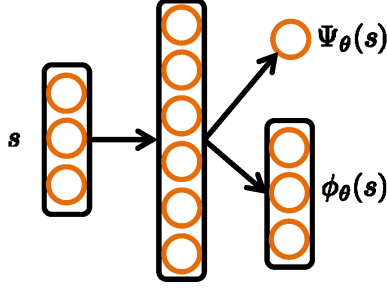
Figure 1: An example of the network architecture of the proposed method.

discrete action space, we propose the network architecture of MLPs shown in Figure 1. In this architecture, the outputs of the network are composed of a continuous scalar variable $\Psi_\theta(s)$ and continuous vector variable $\phi_\theta(s)$. In this study, we approximate the action-value function by the linear function with respect to the action vector:

$$Q_\theta(s, a) = \Psi_\theta(s) + \sum_{i=1}^{K} a_i \phi_\theta^i(s) \tag{6}$$

$$= \Psi_\theta(s) + a^\top \phi_\theta(s) \tag{7}$$

Here, a is the action represented by the binary vector, and a_i is the i-th component of the action.

The gradient of the function $Q_\theta(s, a)$ is given by

$$\frac{\partial Q_\theta(s, a)}{\partial \theta} = \frac{\partial \Psi_\theta(s)}{\partial \theta} + \sum_{i=1}^{K} a_i \frac{\partial \phi_\theta^i(s)}{\partial \theta}, \tag{8}$$

and this is efficiently obtained by the back propagation algorithm.

3.1 Sampling of the Actions

The proposed approximation architecture provides an efficient calculation of the greedy action. For actions with the one-hot representation, the greedy policy is obvious. This is

$$\pi_{\text{greedy}}(s) = \underset{a \in \{1,\dots K\}}{\arg \max} \ Q_\theta(s, a) \tag{9}$$

$$= \underset{i \in \{1,\dots K\}}{\arg \max} \ \phi_\theta^i(s), \tag{10}$$

where $\phi_\theta^i(s)$ is the i-th element of the outputs $\phi_\theta(s)$.

For the K-bits binary vector actions, sampling of the greedy actions with respect to the function 7 is still tractable. The i-th element of the greedy action vector is given by

$$\pi_{\text{greedy}}^i(s) = \begin{cases} 0 & \phi_\theta^i(s) < 0 \\ 1 & \text{otherwise.} \end{cases} \tag{11}$$

Because we can efficiently sample the greedy action, the ϵ-greedy action selection is tractable in our case. In the experiment section, we tested some variants of the ϵ-greedy action selection.

The exact sampling from the softmax action selection for binary vector actions is also tractable. Substituting the equation 7 into the conventional softmax policy with the inverse temperature $\beta > 0$

gives the equality

$$\pi(a|s) \quad = \quad \frac{e^{\beta Q_\phi(s,a)}}{\sum_{a'\in\mathcal{A}} e^{\beta Q_\phi(s,a')}} \tag{12}$$

$$= \quad \frac{e^{\beta \sum_{i=1}^{K} a_i \phi_\theta^i(s)}}{\sum_{a'\in\mathcal{A}} e^{\beta \sum_{i=1}^{K} a_i' \phi_\theta^i(s)}} \tag{13}$$

$$= \quad \prod_{i=1}^{K} \frac{e^{\beta a_i \phi_\theta^i(s)}}{\sum_{a_i'\in\{0,1\}} e^{\beta a_i' \phi_\theta^i(s)}} \tag{14}$$

$$= \quad \prod_{i=1}^{K} \pi_i(a_i|s), \tag{15}$$

where $\pi_i(a_i|s)$ is the bernoulli distribution for the i-th element of the action. The firing probability of the i-th bit of the action is given by the logistic function

$$\pi_i(a_i = 1|s) \quad = \quad \frac{1}{1 + e^{-\beta\phi_\theta^i(s)}}. \tag{16}$$

When the environment is represented by the factored MDP [5][11], the action may be represented by the binary vector, which is composed of a concatenation of one-hot representation vectors (for example, the agent may have to decide one of 2 options and one of 3 options simultaneously. In this case, if the agent takes the first option and third option, an action is represented as a 5-bit vector $(1, 0| 0, 0, 1)^\top$). The greedy action for the factored environment is given by

$$\pi_{\text{greedy}}^j(s) \quad = \quad \underset{i\in\{1,\ldots K_j\}}{\arg\max} \; \phi_\theta^{ij}(s), \tag{17}$$

where j is the index of the factored action sets, and K_j is the size of the j-th action set. Following a similar transformation of the equation 15, the softmax policy for the factored action is given as

$$\pi(a|s) = \prod_{j=1} \pi_j(a^j|s), \tag{18}$$

and $\pi_j(a^j|s)$ is the softmax function with respect to the j-th factored action set

$$\pi_j(a_i^j = 1|s) \quad = \quad \frac{e^{\beta\phi_\theta^{ij}(s)}}{\sum_{i=1}^{K_j} e^{\beta\phi_\theta^{ij}(s)}}. \tag{19}$$

4 Experiment

In the experiment, we tested our proposed architecture in several domains. In all of the experiments, we used the three-layer MLPs described in Figure 1. We also set the activation function of the hidden units using the rectifier linear units (ReLU). All weights connected with output units are sampled from the uniform distribution over $[-0.01, 0.01]$, and all weights between input units and hidden units are sampled from the uniform distribution over $[-\sqrt{6}/\sqrt{N_{\text{hidden}} + N_{\text{input}}}, \sqrt{6}/\sqrt{N_{\text{hidden}} + N_{\text{input}}}]$, where N_{hidden} and N_{input} are the number of units in the layers. The update of the parameter was done by the stochastic gradient descent with a constant step size $\alpha = 0.01$. The discount rate of the objective function in RL is also same in the all of the experiments, so we used $\gamma = 0.95$.

4.1 Grid World with One-hot Representation

First, we tested our algorithm in the conventional grid world with a one-hot representation. This task is the shortest-path problem in the grid world, as suggested by Sutton & Barto [12]. The state
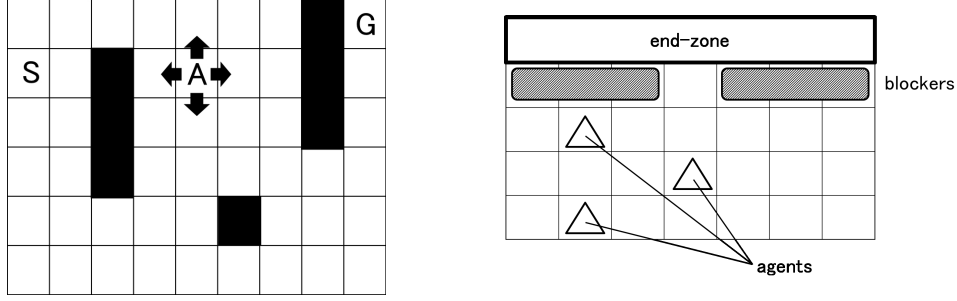
Figure 2: Environments: Sutton's grid world (left) and Blocker (right)

space is composed of 47 discrete states and they are given by the one-hot representation. The agent has 4 discrete actions that correspond to the 4 direction moves (North, South, East, West). The action in this experiment is represented by a one-hot representation (for example, the "North" action corresponds to the vector $(1, 0, 0, 0)^{\top}$). The agent receives a zero reward when the agent reaches the goal, but otherwise receives a -1 reward. The agent was trained in an episodic manner, a single episode was terminated when the agent reached the goal or passed 800 time steps in the episode. The agents were implemented by MLPs with 50 hidden units. The ϵ-greedy policy was used as the behavior policy. In this task, we used $\epsilon = 0.1$.

The left panel of Figure 3 is the result of the experiment. The horizontal axis represents the number of episodes, the vertical axis is the step size in the episode. The black line is the mean performance of 10 runs and the bars are standard deviations. The broken line is the optimal step size. As expected, the agent successfully obtained the optimal policy.

4.2 Grid World with 4-bit Binary Vector Actions

Table 1: Binary Vector Actions

Action	Binary Vector
North	1,1,0,0
South	0,0,1,1
East	1,0,1,0
West	0,1,0,1
Stay	otherwise

In this environment, the task is also the shortest-path problem in the same grid world. The state is given as a one-hot representation, as well. The agent receives a zero reward when it reaches the goal, but otherwise receives a -1 reward. The training is episodic and the termination rule of a single episode is the same as in the previous experiment. In this experiment, actions are represented by 4-bit binary vectors as shown in Table 1. Only 4 of $2^4 = 16$ patterns move the agent to the corresponding direction, and the agent stays at the same state if the other action patterns are selected. The agents were again implemented by MLPs with 50 hidden units. The ϵ-greedy policy was used as the behavior policy. In this task, we used $\epsilon = 0.2$.

Right panel of Figure 3 The horizontal axis represents the number of episodes, the vertical axis is the step size in the episode. The black line is the mean performance of 10 runs and the bars are standard deviations. The broken line is the optimal step size. Again, the agent successfully obtained the optimal policy through the experiment even in the binary vector action domain. This result shows that the proposed method successfully improved the behavior of the agent without any Monte-Calro based samplings of the actions, even when the representation of actions is not a one-hot representation.

4.3 Grid World with Population Coding

Again, the task is the shortest path problem in the same grid world. The state representation, the reward function and termination rules of a single episode are the same as in the previous experiments. In this experiment, the action is represented by a 40-bit binary vector. And the moves of the agent
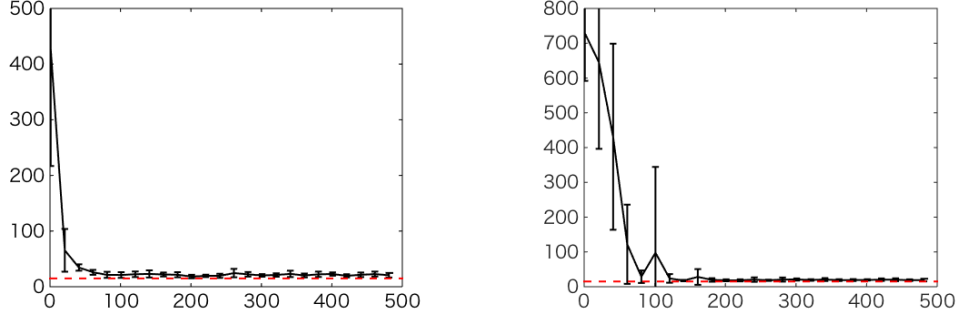
Figure 3: Results in Sutton's grid world. The broken line represents the minimum time step. Left: grid world; Right: grid world with 4-bit binary vector actions.

are driven according to the type of population coding. Concretely, when the environment receives a 40-bit vector, one of the four-direction moves (1: North; 2: South; 3: East; 4: West) or the stay behavior (5: Stay) occurs according to the probability

$$P_j = \frac{E_j}{\sum_{k=1}^{5} E_k} \quad j = 1, 2, 3, 4, 5, \tag{20}$$

where E_j are give by the action $a_i \in \{0, 1\}$, $i = 1, \dots, 40$ following the equations

$$E_1 = \sum_{i=1}^{10} a_i, \ E_2 = \sum_{i=11}^{20} a_i, \ E_3 = \sum_{i=21}^{30} a_i, \ E_4 = \sum_{i=31}^{40} a_i \tag{21}$$

and

$$E_5 = \max\Big(10 - \sum_{k=1}^{4} E_k, \ 0\Big). \tag{22}$$

In this experiment, because the discrete action space exponentially grows according to the length of the binary action vector, the size of the corresponding action space is huge $|\mathcal{A}| = 2^{40} > 10^{12}$. Therefore, efficient sampling of the action is also required in this domain.

In this experiment, we used MLPs with 50 hidden units. We tested three types of behavior policies. The first policy is the conventional ϵ-greedy policy. We used $\epsilon = 0.3$ in the task. the second policy is the bit-wise ϵ-greedy policy. In this policy, each bit of the action element undertakes ϵ_{bit}-greedy exploration. More concretely, the i-th element of the action vector takes the random action ($a_i = 1$ with probability 0.5) with probability ϵ_{bit}. Because we can sample the greedy actions with ease, we can explicitly take this behavior policy. We used $\epsilon_{\text{bit}} = 0.05$ in this experiment. The third policy is the sofmax policy that was explained in section 3.1. We used $\beta = 20$ in this experiment.

Figure 4 shows the result of the experiment. The horizontal axis represents the number of episodes, the vertical axis is the step size in the episode. The solid lines are the mean performance of 10 runs and the bars are standard deviations. The broken lines are the optimal step size. The results show that all three behavior policies successfully improved the performance of the agent in the high-dimensional action space. From these results, the bit-wise ϵ-greedy policy (center) and the softmax policy (right) shows better performance than that of the conventional ϵ-greedy policy (left). This would be because of the large exploration rate in the ϵ-greedy policy ($\epsilon = 0.3$), but running with a smaller exploration rate ($\epsilon \leq 0.2$) sometimes resulted in divergence of the parameters during the learning.
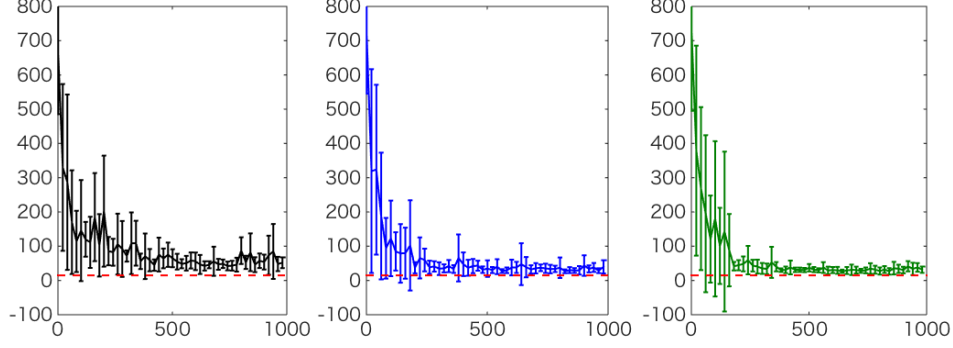
6

Figure 4: The result of the grid world with population coding task. The left most panel shows the result of the ϵ-greedy action selection with $\epsilon = 0.3$, the center panel is that of the bit-wise ϵ-greedy with $\epsilon_{\text{bit}} = 0.05$. The right most panel is the result of the softmax action selection with the inverse temperature $\beta = 20$.

4.4 Blocker

The blocker is the multi-agent task suggested by Sallans and Hinton [5][11]. This environment consists of a 4×7 grid, three agents, and two pre-programmed blockers. Agents and blockers never overlap each other in the grid. To obtain a positive reward, agents need to cooperate in this environment. The "team" of agents obtain a $+1$ reward when any one of the three agents enters the end-zone, otherwise the team receives a -1 reward. The state vector is given as a 141 binary vector, composed of the positions (grid cells) of all the agents (28 bits $\times$ 3 agents), the east most positions of each blocker (28 bits $\times$ 2 blockers) and a bias bit that is always one (1 bit). Each agent can move to any of the four directions. Hence the size of the action space is $4^3 = 64$. In this environment, the representation of the action is given as a 12-bit binary vector in which the three one-hot representation is concatenated (for example, (North, North, North) actions corresponding to the vector $(1, 0, 0, 0|1, 0, 0, 0|1, 0, 0, 0)^\top$). In each episode, the agents start at a random position in the bottom row of the grid. When one of the agents enters the end-zone or 40 time steps have passed, the episode terminates and the next episode starts after the initialization of the environment.

In this task, we used MLPs with 100 hidden units. The ϵ-greedy policy was used as the behavior policy. In this task, we used $\epsilon = 0.3$. Also, we tested the agent-wise ϵ-greedy policy as the behavior policy. This policy is a modified version of the ϵ-greedy policy for actions with factored representation, and each agent follows the ϵ-greedy policy independently. In the case of the agent-wise-ϵ-greedy policy, we used $\epsilon = 0.1$ for each agent.

Figure 5 shows the results of the experiment. The horizontal axis represents the time steps, and the vertical axis represents the average reward during the last 1000 steps. The left panel is the result of the conventional ϵ-greedy action selection, the right panel is that of the agent-wise ϵ-greedy action selection. Both results are competitive, but in this experiment, agent-wise ϵ-greedy agents tend to escape from the local optima.

5 Discussion

In the environment with one-hot representation actions, the linear function approximation of the action-value corresponds to the bilinear function with respect to the action vector and the state vector
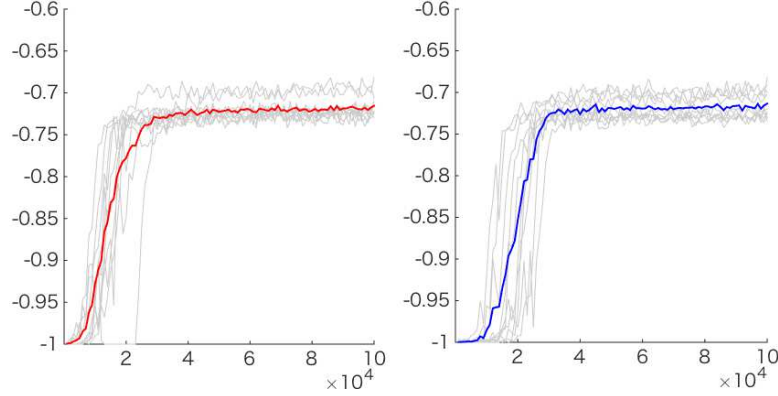
$$Q_\theta(s, a) = a^\top \theta s. \tag{23}$$

7

Figure 5: The result of the blocker task. The left panel is the result of the conventional ϵ-greedy action selection with $\epsilon = 0.3$. The right panel is that of the agent-wise ϵ-greedy action selection with $\epsilon_{\text{agent}} = 0.1$. The gray lines are the results of 10 individual runs, and thick lines (red and blue) are the average performance.

In this case, the parameter θ is give as a matrix. If the state is given by a one-hot representation, this approximation is identical with the table representation. As suggested in our method, the linear architecture with respect to the action enables efficient sampling of the greedy action. More recently, Mnih *et al.* proposed a DQN architecture [10]. In this case, we evaluate the action-values corresponding to all the discrete actions by a single forward propagation. And then the training of the approximator is done only on the output, which corresponds to the selected action. This architecture can be interpreted as a linear function approximation with respect to the actions

$$Q_\theta(s, a) = a^\top \phi_\theta(s). \tag{24}$$

If we construct $\phi_\theta(s)$ by some nonlinear function with high representational power such as deep neural networks, this approximation is sufficient for approximating the Q-values when actions are given by one-hot representation vectors.

The goal of our architecture (equation 7) is to adapt these ideas to the RL with binary vector actions. Although our function approximator is strongly restricted by the linear architecture with respect to the action, our function approximator is sufficient to represent an arbitrary deterministic policy $\pi(s)$ by $\arg\max_a Q_\theta(s, a)$ even when we treat the binary vector actions, as long as we represent $\phi_\theta(s)$ by a universal function approximator.

6 Conclusion

In this paper, we suggest a novel architecture of multilayer perceptrons for RL with a large discrete action set. In our architecture, the action-value function is approximated by a linear function with respect to the vector actions. This approximation method enables us to efficiently sample from the greedy policy and the softmax policy. The Q-learning-based off-policy algorithm is therefore tractable in our architecture without any Monte-Carlo approximations. We empirically tested our method in several discrete action domains, and the results supported its effectiveness. Based on these promising results, we expect to extend our approach using deep architectures in a future work.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[2] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[3] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. Reinforcement learning of motor skills in high dimensions: A path integral approach. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2397–2403. IEEE, 2010.

[4] P Smolensky. Information processing in dynamical systems: foundations of harmony theory. In *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1*, pages 194–281. MIT Press, 1986.

[5] Brian Sallans and Geoffrey E Hinton. Using free energies to represent q-values in a multiagent reinforcement learning task. In *NIPS*, pages 1075–1081, 2000.

[6] Nicolas Heess, David Silver, and Yee Whye Teh. Actor-critic reinforcement learning with energy-based policies. In *EWRL*, pages 43–58. Citeseer, 2012.

[7] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards.* PhD thesis, University of Cambridge, 1989.

[8] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[9] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[11] Brian Sallans and Geoffrey E Hinton. Reinforcement learning with factored states and actions. *The Journal of Machine Learning Research*, 5:1063–1088, 2004.

[12] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

This figure "grid.png" is available in "png" format from:

http://arxiv.org/ps/1512.01332v1

This figure "grid_v4.png" is available in "png" format from:

http://arxiv.org/ps/1512.01332v1

This figure "network.png" is available in "png" format from:

http://arxiv.org/ps/1512.01332v1