Topological Experience Replay for Fast Q-Learning

Zhang-Wei Hong [1] **Tao Chen** [1] **Yen-Chen Lin** [1] **Joni Pajarinen** [2] **Pulkit Agrawal** [1]

Abstract

State-of-the-art deep Q-learning methods update Q-values using state transition tuples sampled from the experience replay buffer. Often this strategy is to randomly sample or prioritize data sampling based on measures such as the temporal difference (TD) error. Such sampling strategies are agnostic to the structure of the Markov decision process (MDP) and can therefore be data inefficient at propagating reward signals from goal states to the initial state. To accelerate reward propagation, we make use of the MDP structure by organizing the agent's experience into a graph. Each edge in the graph represents a transition between two connected states. We perform value backups via a breadth-first search that expands vertices in the graph starting from the set of terminal states successively moving backward. We empirically show that our method is substantially more data-efficient than several baselines on sparse reward tasks.

1. Introduction

A significant challenge in reinforcement learning (RL) is to overcome the need for large amounts of data. A typical RL agent updates its policy using the interaction data it collects to maximize the expected sum of rewards. On-policy RL methods (Sutton & Barto, 2018; Schulman et al., 2017; Mnih et al., 2016) discard past interaction data after every policy update. In contrast, off-policy algorithms such as Q-learning (Watkins & Dayan, 1992) leverage experience replay (Lin, 1992) to achieve greater data efficiency by making use of all the past interactions. This approach has also been scaled to Q-learning from high-dimensional state spaces using deep neural networks (Mnih et al., 2015). In Q-learning, the Q-function is trained to predict the expected sum of the intermediate reward and the succeeding state's Q-value. This process of using Q-values of successor states to update Q-values of preceding states is referred to as *bootstrapping*.

Because bootstrapping is used to update the Q-values, the ordering of states used for the updating the Q-value can have a substantial influence on the convergence speed of Q-value. A suboptimal update order can exacerbate issues such as the overestimation bias that plaque Q-learning (Kumar et al., 2020; Van Hasselt et al., 2016). As a motivating example, consider the Markov decision process (MDP) shown in Figure 1. Let the agent receive a reward when it reaches the goal state (labeled as G), but not at any other state. Starting from state C, the agent can obtain the reward by visiting states in the sequence $C \rightarrow D \rightarrow G$. Now if the Q-value $Q(D, a)$ of taking action a at state D is inaccurate, then using it to bootstrap $Q(C, a)$ will not reduce the error in the estimate of $Q(C, a)$. It is therefore natural to start at the terminal state G and first bootstrap Q-values of preceding states (D, E), and then for states (A, B, C). Backing up Q-values in the reverse order (i.e., starting at a terminal state and ending at an initial state) ensures that only meaningful updates to the Q-value are made. In fact, it has been shown that for acyclic MDPs such a *reverse sweep* is the optimal order for performing backups (Bertsekas et al., 2000). Several extensions for finding an optimal backup ordering in cyclical MDPs have been proposed (Dai et al., 2011; 2009; Dai & Hansen, 2007). However, these prior methods assume access to the environment model for both cyclic and acyclic MDPs.

State-of-the-art Q-learning methods (Mnih et al., 2015; Lillicrap et al., 2015; Haarnoja et al., 2018; Fujimoto et al., 2018) sample data from the replay buffer using strategies such as uniform random or prioritized sampling. While these strategies are guaranteed to result in convergence of the Q-function (Watkins & Dayan, 1992), they can be suboptimal in their speed of convergence. For instance, a popular prioritized sampling strategy orders states by the bootstrapping error (also called as temporal difference or the TD-error) (Schaul et al., 2015). Since during training the TD-error depends on (possibly) erroneous estimate of the Q-values, ordering states based on TD-error can lead to suboptimal convergence speed. The speed of convergence matters because in Q-learning, data collection is interleaved

[1]Department of Computer Science, Massachusetts Technology of Institute, United States [2]Department of Computer Science, Aalto University, Finland. Correspondence to: Zhang-Wei Hong <zwhong@mit.edu>.

Reinforcement Learning for Real Life (RL4RealLife) Workshop in the 38th International Conference on Machine Learning, 2021. Copyright 2021 by the author(s).

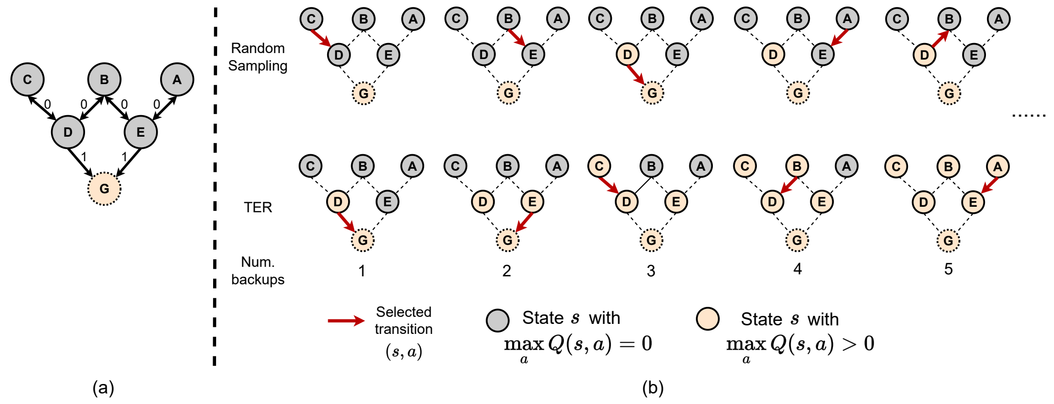
(a)
(b)
Random Sampling
TER
Num. backups
1
2
3
4
5
Selected transition (s,a)
State s with $\max_a Q(s,a) = 0$
State s with $\max_a Q(s,a) > 0$

Figure 1. The ordering of states used for updating Q-values directly effects the convergence speed. **(a)** Consider the graphical representation of the MDP with the goal state (G). Each node in the graph is a state. The arrows denote possible transitions (s, a, r, s') and the numbers on the arrows are the rewards r associated with the transition. **(b)** Random sampling wastes many backups at states with zero values (gray) while each backup of TER propagates values (orange) one step back.

with Q-value updates. If the Q-values are incorrect, the current policy may take actions that result in low rewards. Such data may not even be useful for updating Q-values along the optimal trajectory. Therefore, the slower convergence of Q-values is directly linked to the data inefficiency. One way to reduce dependence on data is to artificially increase the speed of convergence by increasing the ratio between the Q-learning update steps and the data collection steps. However, when function approximators are used to estimate the Q-value, excessive updates on a fixed set of interaction data can lead to over-fitting and overestimation of the Q-values, resulting in worse overall performance.

Despite the knowledge that sampling based methods for learning from a replay buffer are suboptimal, state-of-the-art algorithms still employ them. The reason is that using the optimal strategy of reverse sweep requires knowledge of the environment model which is generally unknown in high-dimensional state spaces. Several recent works have attempted to tackle this challenge: *Episodic Backward Update (EBU)* (Lee et al., 2019) replays state-action trajectories in the reverse order of state visitations in every episode. However, such an order is not equivalent to the optimal backup order as parts of states could be updated repeatedly in a loop. On the other hand, *DisCor* (Kumar et al., 2020) re-weights the error of state transitions to approximate the optimal backup order. However, the conservative weighting scheme of DisCor impedes policy improvement.

We propose a method that overcomes the limitations of prior works and speeds up convergence of Q-values estimated directly from high-dimensional states. The core of our method is an algorithm for building a graph of state-transitions directly from visual observations. Our method works for episodic MDPs with a discrete action space. Each state is a node in the graph and two states are connected with an edge if the agent transitions between them during exploration. Graph building and exploration proceed iteratively. The state-transition graph aids efficient exploration by enabling fast convergence of Q-values. Exploration in turn provides data to extend the graph. Q-values are updated by performing a reverse sweep on the graph using the breadth-first search (BFS) algorithm (Cormen et al., 2009). BFS is initiated from a set of terminal states because no backups are required to determine the correct Q-value for these states. We call this Q-learning method *Topological Experience Replay* (TER) because the update order of Q-values is based on the topology of the state space.

2. Preliminaries

We consider reinforcement learning (RL) (Sutton & Barto, 2018) in an episodic discrete-time Markov decision process (MDP). The objective of RL is to find the optimal policy π^* that maximizes expected return via interaction with the environment. A MDP can be represented by $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \mathcal{E}, T)$ where S denotes the state space, A the action space, R : $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ the reward function, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$ the state transition function, $\mathcal{E} : \mathcal{S} \mapsto [0, 1]$ defines the termination condition of an episode, and T the maximum length of each episode. Starting from $t = 0$ in an episode, an agent takes an action $a_t = \pi(s_t), a_t \in \mathcal{A}, s_t \in \mathcal{S}$, transitions to the next state $s_{t+1} = \mathcal{P}(s_t, a_t)$, and receives a reward $r_t = \mathcal{R}(s_t, a_t, s_{t+1})$. The episode terminal condition $\mathcal{E}(s_{t+1}) = 1$ when the task is complete, otherwise 0. Any s satisfying $\mathcal{E}(s) = 1$ is a terminal state. The optimal policy is $\pi^* = \arg\max_\pi \mathbb{E}\big[\sum_{\tau=t}^{T-1} \gamma^{\tau-t} r_\tau | s_t = s\big]\ \forall s \in \mathcal{S}$, where γ is a discount factor (Sutton & Barto, 2018).

Q-learning is a popular algorithm for reward maximization. Q-learning approximates the Q function $Q(s, a) =$

$\mathbb{E}\big[\sum_{\tau=t}^{T-1}\gamma^{\tau-t}r_\tau|s_t = s, a_t = a\big]$ using iterative backup operations: $Q(s,a) \leftarrow \mathcal{R}(s,a,s') + \gamma\max_{a'} Q(s',a')$ where s' is the state reached after the agent executes action a in the state s. The policy $\pi(s)$ can be easily derived as: $\pi(s) := \arg\max_a Q(s,a)$. When $\mathcal{S}$ is high-dimensional, the Q function is usually represented by a deep neural network (i.e., deep Q-network (DQN) (Mnih et al., 2015)). The interaction data collected by the agent is stored in an experience replay buffer (Mnih et al., 2015) in the form of state transitions (s_t, a_t, r_t, s_{t+1}). The Q function is updated using stochastic gradient descent on batches of data randomly sampled from the replay buffer.

3. Method

Our method is motivated from the backward value iteration (BVI) algorithm (Dai & Hansen, 2007). BVI exploits the state topology to perform the value backups backward from the goal states. Dai & Hansen (2007) show that BVI is more efficient at value propagation than random sampling (Bertsekas et al., 2000) and prioritized value backup (Moore & Atkeson, 1993) methods. However, BVI requires the knowledge of state transition function $\mathcal{P}$ for recursively updating the Q-values. $\mathcal{P}$ is hard to obtain in high-dimensional state space such as images. Our proposed method, Topological Experience Replay, overcomes these challenges and scales to high-dimensional state spaces. In Section 3.1 we describe the procedure for building the state transition graph from high-dimensional states. Next, in Section 3.2 we describe how the graph is used to determine the backup order for Q-learning.

3.1. Topological Experience Replay

The replay buffer stores a collection of trajectories and can be thought of as a partial non-parametric transition function (van Hasselt et al., 2019). We use the data from the replay buffer to build an unweighted graph of state transitions. Because the same state can appear in multiple trajectories, constructing a graph involves connecting individual trajectories at these common states. Let unweighted graph be $\mathcal{G} = \{V, E\}$, where V, E denote the set of vertices and edges, respectively. Let $v \in V, e \in E$ denote the individual vertices and edges, respectively. A straightforward way to build the graph of state transitions is to process trajectories sequentially and compare each state with existing vertices in the graph. If the state is already in the graph, then no new vertices are created. Otherwise, a new vertex is added and connected with the preceding vertices in the graph with a directed edge. Each edge between vertices, v, v', written as $e(v, v')$ therefore corresponds to a state transition $\{(s', a, r, s')\}$. We update the graph in an online fashion as more data is added to the replay buffer.

In discrete and finite state spaces, each vertex corresponds to an individual state of the MDP. However, when operating with high-dimensional observations the state space grows rapidly and it is not possible to store raw observations due to memory limits. Secondly, the state comparison required to add edges in the graph can become computationally expensive. These issues can be overcome by computing a low-dimensional representation of states that also preserve the local topology of the state-space. To this end, we make use of random projection (Bingham & Mannila, 2001) which approximately preserves the distance between states in the transformed low-dimensional spaces according to the Johnson-Lindenstrauss lemma (Dasgupta & Gupta, 1999). Prior work has empirically found that random projections form a useful low-dimensional representation of visual observations (Burda et al., 2019). We implement random projection as a function $\phi(s) := \mathbf{M}s$ where $s \in \mathcal{S}$, $\mathbf{M}$ is a matrix with elements randomly sampled from a normal distribution $\texttt{Normal}(0, 1/Z)$, where Z denotes the dimensionality of projected vectors[1]. Note that random projections are only used for graph-building. The Q function is learned directly from the original high-dimensional state representation.

3.2. Topological Reverse Sweep

Once the graph has been constructed, the next step is to determine the ordering of states for updating the Q-values. For this we maintain a record of vertices that correspond to terminal states and this set is denoted by $V_\mathcal{E}$. Each terminal vertex acts as the root node. We perform reverse breadth-first search (BFS) on the graph starting from every terminal vertex separately and in parallel. Given a vertex v', we first enumerate all its predecessors v and use the state-transition tuple (s, a, r, s') for updating $Q(s, a)$. BFS only expands each vertex once, and thus infinite loop on cyclic (i.e., bi-directional) edges will not happen. When there is no vertex to expand, reverse BFS is restarted from the root vertices again. We perform BFS iteratively and repeatedly until Q-values have converged. Note that even though terminal states might be states that the agent has to avoid entering, starting value backups from these states do not impede Q-learning. For instance, propagating values from a terminal state that gives a negative reward can make the agent learn to avoid entering this state again, though it may not help the agent learn the optimal policy directly.

3.3. Batch Mixing

One potential drawback of starting the reverse search from only terminal states is that the states that are unreachable from the terminal states will not be updated. Consequently, the Q-values of such states are not updated, which in turn

[1]If s is an image, we flatten s into an one dimensional vector before transformation

impedes the convergence of Q-learning. Prior work (Li & Littman, 2008) has shown that interleaving value updates at randomly sampled transitions and that at transitions selected by any prioritization mechanism (e.g., BVI) ensure the convergence. Thus, we mix experience from TER and PER to form training batches using a mixing ratio $\eta \in [0, 1]$. For each batch, η fraction of the data is from PER, and $1 - \eta$ is from TER.

4. Experiments

4.1. Setup

Environment We evaluate TER on *Sokoban* (Schrader, 2018) where typical RL approaches, such as DQN, struggle due to sparse rewards (Racanière et al., 2017). The agent perceives a 84×84 RGB top-down view image of the game map, as shown in Figure 2. The task is complete once all the boxes (yellow) are pushed into the target positions (red). The agent can only push box on the walk-able (black).

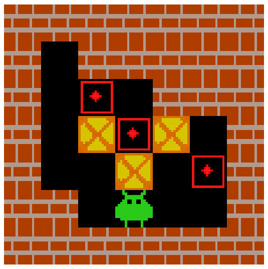

Figure 2. The illustration of *Sokoban* environment.

Baselines. We compare TER with uniform experience replay (UER) (Mnih et al., 2015), prioritized experience replay (PER) (Schaul et al., 2016), episodic backward update (EBU) (Lee et al., 2019), and DisCor (Kumar et al., 2020). UER uniformly samples state-transition tuples from the replay buffer. PER prioritizes experience with high temporal difference (TD) (Sutton & Barto, 2018) errors defined as $|r_t + \max_{a'} Q(s', a') - Q(s, a)|$. EBU uniformly samples an episode from the replay buffer, and replays state-action pairs in the reverse order of state visitations in every episode. For example, given an episode $[s_1, a_1, \ldots s_{T-1}, a_{T-1}, r_{T-1}, s_T]$, EBU updates in a sequence $[Q(s_{T-1}, a_{T-1}), Q(s_{T-2}, a_{T-2}), \ldots, Q(s_1, a_1)]$. DisCor re-weights each Q-function update inversely proportional to the estimated bootstrapping error $|Q^*(s', a') - Q(s', a')|$ of a state-action pair (s, a), where Q^* denotes the optimal Q-function.

Implementation. We implement double deep Q-learning (DQN) (Van Hasselt et al., 2016) with the same architecture used by (Mnih et al., 2015). The Q function is updated once per four environment steps unless specified otherwise. We use ϵ-greedy method for exploration. For TER, we set the the dimension of the projected states $|Z| = 3$ (see Section 3.1).

Evaluation Metric. We ran each experiment with 5 different random seeds and report the mean (solid or dashed line) and 95%-confidence interval estimated by the bootstrapping method (shaded part) (DiCiccio et al., 1996) on the learning curves. The learning curves indicate the average normalized return over 100 testing episodes in a varying number of environment steps.

4.2. Results

We test TER on *Sokoban* environments with map sizes and the number of boxes, where we denote a configuration in this format `Sokoban-<map width>x<map height>-<num. boxes>`. As shown in Figure 3, TER learns significantly faster and achieves higher final performance than all the baselines in all tasks. Such improvement is even more significant when the environment map is bigger and there are more boxes. Surprisingly, we can see from Figure 3 that EBU and DisCor perform worse than UER. We find that the average Q value of EBU is much bigger than the maximum possible return in the environment, suggesting that EBU suffers from the over-estimation problem (Hasselt, 2010) that hinders the Q learning. On the other hand, we find that the exploding value error estimates in DisCor impede the learning progress, where the updates are down-weighted exceedingly due to huge value error estimated values.

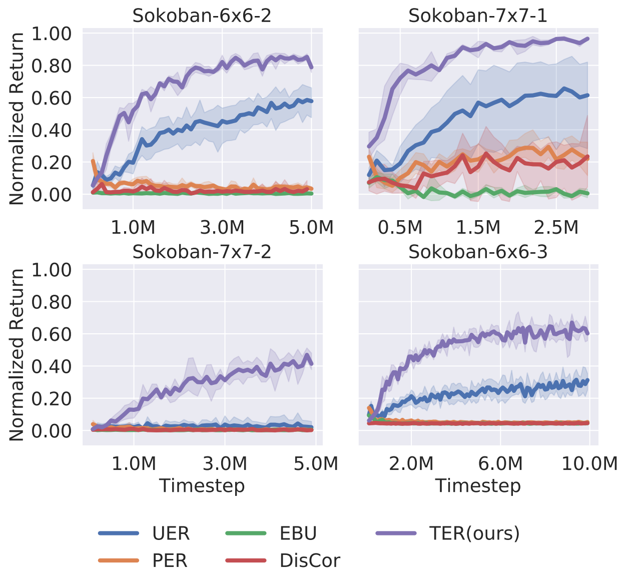

Figure 3. The plots are ordered by task difficulty from top to bottom. The performance gain of TER increases with task difficulty, which shows that TER better exploits the sparse rewards than the baselines.

References

Bertsekas, D. P. et al. *Dynamic programming and optimal control: Vol. 1*. Athena scientific Belmont, 2000.

Bingham, E. and Mannila, H. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 245–250, 2001.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1lJJnR5Ym.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to algorithms*. MIT press, 2009.

Dai, P. and Hansen, E. A. Prioritizing bellman backups without a priority queue. In *ICAPS*, pp. 113–119, 2007.

Dai, P., Weld, D. S., et al. Focused topological value iteration. In *Nineteenth International Conference on Automated Planning and Scheduling*, 2009.

Dai, P., Weld, D. S., Goldsmith, J., et al. Topological value iteration algorithms. *Journal of Artificial Intelligence Research*, 42:181–209, 2011.

Dasgupta, S. and Gupta, A. An elementary proof of the johnson-lindenstrauss lemma. *International Computer Science Institute, Technical Report*, 22(1):1–5, 1999.

DiCiccio, T. J., Efron, B., et al. Bootstrap confidence intervals. *Statistical science*, 11(3):189–228, 1996.

Fujimoto, S., Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pp. 1587–1596. PMLR, 2018.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pp. 1861–1870. PMLR, 2018.

Hasselt, H. Double q-learning. *Advances in neural information processing systems*, 23:2613–2621, 2010.

Kumar, A., Gupta, A., and Levine, S. Discor: Corrective feedback in reinforcement learning via distribution correction. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 18560–18572. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/d7f426ccbc6db7e235c57958c21c5dfa-Paper.pdf.

Lee, S. Y., Sungik, C., and Chung, S.-Y. Sample-efficient deep reinforcement learning via episodic backward update. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/e6d8545daa42d5ced125a4bf747b3688-Paper.pdf.

Li, L. and Littman, M. Prioritized sweeping converges to the optimal value function. 2008.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Lin, L.-J. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PMLR, 2016.

Moore, A. W. and Atkeson, C. G. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning*, 13(1):103–130, 1993.

Racanière, S., Weber, T., Reichert, D. P., Buesing, L., Guez, A., Rezende, D., Badia, A. P., Vinyals, O., Heess, N., Li, Y., et al. Imagination-augmented agents for deep reinforcement learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 5694–5705, 2017.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. In *ICLR (Poster)*, 2016. URL http://arxiv.org/abs/1511.05952.

Schrader, M.-P. B. gym-sokoban. https://github.com/mpSchrader/gym-sokoban, 2018.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

van Hasselt, H. P., Hessel, M., and Aslanides, J. When to use parametric models in reinforcement learning? In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/1b742ae215adf18b75449c6e272fd92d-Paper.pdf.

Watkins, C. J. and Dayan, P. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.