# Recurrent Open-loop Control in Offline Reinforcement Learning

**Alex Lewandowski, Vincent Zhang, Dale Schuurmans**
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada
lewandowski,vzhang,daes@ualberta.ca

## Abstract

Environments come preconfigured with hyperparameters, such as frame skips, which determines an agents window of temporal abstraction. However, the temporally abstract behavior induced by frame skipping is limited to repeating actions within the window. At the same time, learning temporally abstract behavior from logged experience is crucial for generalization in offline reinforcement learning. We propose to learn policies that dynamically interact with the environment within a fixed window of temporal abstraction. To achieve this, we use a recurrent neural network to construct a representation of future states, given only the first input state. The memory of the recurrent neural network enables temporal state aggregation, allowing action-values of future states to be estimated. Our proposed model can be trained using 1-step Q-learning but is further improved by using a variable form of $n$-step Q-learning, where $n$ is the remaining number of time steps in the window. Our experiments show that recurrent open-loop policies consistently outperform persistent policies that naively repeat actions.

## 1 Introduction

For complex environments such as the Arcade Learning Environment (Bellemare et al., 2012), it is common to artificially extend actions through time by repeating the last action. Similarly, classical control environments such as the MuJoCo physics simulator (Todorov et al., 2012) discretize a dynamical system by some prespecified value. These frame skip and discretization hyperparameters determine both the quality of the optimal policy and the ease of learning towards such a policy. A policy that interacts with the environment at a higher rate can simulate any slower policy. Hence, an optimal fast policy is better than an optimal slow policy. However, a fast discrete policy is capable of erratic and less smooth behavior which hinders learning. Learning a fast policy requires also more data, and algorithms like Q-learning do not work in the continuous limit (Tallec et al., 2019).

If experience is already collected and we use Offline RL (Levine et al., 2020), then there is another subtle benefit of learning slower policies. When actions are selected by their action-values, the approximation error for a particular action-value can exceed the action gap. As a result, the actions chosen according to the maximum approximated action-value will not be correlated with the optimal action. In the online setting, the approximation error shrinks as more experience is collected. Eventually the approximation error becomes smaller than the action gap, which enables informed action selection. An offline agent cannot collect more data to shrink the approximation error, but actions can be extended in time to increase the action gaps. The common approach to temporally extend actions is to naively repeat an action (Braylan et al., 2015). Repeating actions, or action persistence, restricts the expressible policy class and limits the aforementioned benefits of learning slower policies.

Offline Reinforcement Learning Workshop at Neural Information Processing Systems, 2020.

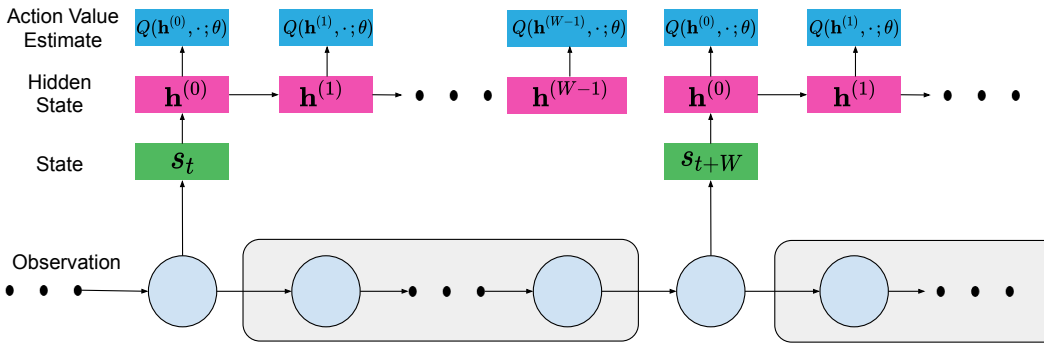


Figure 1: Open-loop control with recurrent neural networks. A hidden memory cell is initialized at $s_t$. The recurrent neural network outputs its estimate of the action-value and an action $a_t$ is selected. The next $W-1$ states are not fed into the recurrent neural network. Instead, the hidden memory cell at each time step serves as an approximate representation of the future states.

In this paper, we consider learning temporally abstract action-values in offline RL. We propose a novel method that learns the action-values using an open-loop recurrent neural network that takes as input the first state of a time window. At each time step, the action-values are updated against a target that bootstraps either the 1-step return or the n-step return, where n is the remaining steps in the temporal window. We show that the proposed recurrent open-loop policies consistently outperform persistent policies that naively repeat actions.

## 2 Background

We briefly describe the Markov Decision Process (MDP) formalism that underpins reinforcement learning (Lattimore and Szepesvári, 2020). An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, r, \mathcal{P}, \mu, \gamma)$, where $\mathcal{A}$ denotes the action space, $\mathcal{S}$ is the state space, $r : \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function that maps a state and an action to a reward, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition function, $\mu$ is the initial state distribution and $\gamma \in [0, 1]$ is the discount factor. Lastly, we say a policy $\pi$ is Markov if $\pi(a|s_t, s_{t-1}, \ldots, s_1) = \pi(a|s_t)$.

### 2.1 Offline Reinforcement Learning

The work presented here focuses on offline RL (Levine et al., 2020), also referred to as batch RL. The goal of control in offline reinforcement learning is to learn a policy $\pi_\theta$ parameterized by $\theta$ where the learning algorithm only has access to a dataset, and no further interaction with the environment (Agarwal et al., 2020). In particular, we consider a dataset consisting of tuples in the form $\{(s_t, a_t, r_{t+1}, s_{t+1})\}$ generated from some behavior policy $\pi_b$.

One representative algorithm for offline RL is Fitted Q-iteration (FQI, Riedmiller 2005). This algorithm is an offline version of Q-learning, where all experience is stored and sampled from a fixed replay buffer $\mathcal{D}$. For neural network approximators, FQI does not differ from implementation of DQN (Mnih et al., 2013). In particular, we still require a target network parameterized by $\theta'$, which is a lagged version of the parameters being learned $\theta$.

$$J(\theta) = \sum_{(s,a,r,s') \sim \mathcal{D}} \left( \underbrace{r + \gamma \cdot \max_{a'} Q(s', a'; \theta')}_{\text{target}} - Q(s, a; \theta) \right)^2$$

The algorithm that we compare against however is not FQI, but persistent FQI, which learns a policy that repeat actions (Metelli et al., 2020). Persistent FQI is similar to FQI in that it first applies a Q-learning update. However, the Q-learning greedy update is followed by a number of "persistent" updates that enforces action repetition. This way, the algorithm is trained to repeat actions and is optimal in the MDP that limits state transitions to those reachable by action repetition.

### 2.2 Recurrent Neural Networks

A recurrent neural network maintains a hidden state $\mathbf{h}$ that is updated at each time step. This allows recurrent architectures to process sequential data of varying length (Goodfellow et al., 2016). Recurrent neural networks accomplish this by sharing weight matrices across time steps. For example, the hidden-to-hidden connections $\mathbf{W}$, input-to-hidden connections $\mathbf{U}$ and hidden-to-output weights $\mathbf{V}$ are shared across time. In this paper, we focus on the one-to-many recurrent architecture that takes an input at a single time step and outputs a sequence. This is also referred to as open-loop execution, since no input is processed after the first state. The exact recurrence can be written as,

$$\begin{aligned}
\mathbf{Q}(s\,;\,\theta) &= \Big[Q(\mathbf{h}^{(0)}, \cdot\,; \theta), Q(\mathbf{h}^{(1)}, \cdot\,; \theta), \ldots, Q(\mathbf{h}^{(W-1)}, \cdot\,; \theta)\Big] \\
\mathbf{h}^{(0)} &= \tanh(\mathbf{U}s) \\
\mathbf{h}^{(w)} &= \tanh(\mathbf{W}\mathbf{h}^{(w-1)} + \mathbf{b}) \\
Q(\mathbf{h}^{(w)}, \cdot\,; \theta) &= \mathbf{V}\mathbf{h}^{(w)} + \mathbf{c}
\end{aligned}$$

where we denote the output $Q(\mathbf{h}^{(w)}, \cdot; \theta)$ as a function of the hidden state $\mathbf{h}^{(w)}$ and parameters $\theta = \{\mathbf{U}, \mathbf{W}, \mathbf{V}, \mathbf{b}, \mathbf{c}\}$ where $\mathbf{U}, \mathbf{W}, \mathbf{V}$ are weight matrices and $\mathbf{b}, \mathbf{c}$ are bias vectors.

## 3 Temporal Abstraction and Open-loop control

Temporal abstraction is often viewed interchangeably with hierarchical reinforcement learning (Barto and Mahadevan, 2003) and the options framework (Sutton et al., 1999; Precup, 2000). Options enforce hierarchical learning by changing the agent's primitive action set to an abstract option set. We describe an additional form of temporal abstraction, which we refer to as a bottom-up approach. The bottom-up approach to temporal abstraction maintains the primitive action-set, and directly learns action sequences from the primitive action-set and its action-values. Temporal abstraction is instead extracted from the way states are processed in the underlying discretization mechanism of the environment. For example, environments in the Arcade Learning Environment (Bellemare et al., 2012) use a frame to represent the temporal primitive of state. A bottom-up approach to temporal abstraction is determined by the way the agent processes a sequence of frames, such as stacking frames, skipping frames and using sticky actions (Machado et al., 2017). For environments that are discretized dynamical systems, the temporal abstraction is the rate at which we discretize the system. These temporal abstractions differ from the hierarchical (and options) perspective in that the actions available to the agent remain the same.

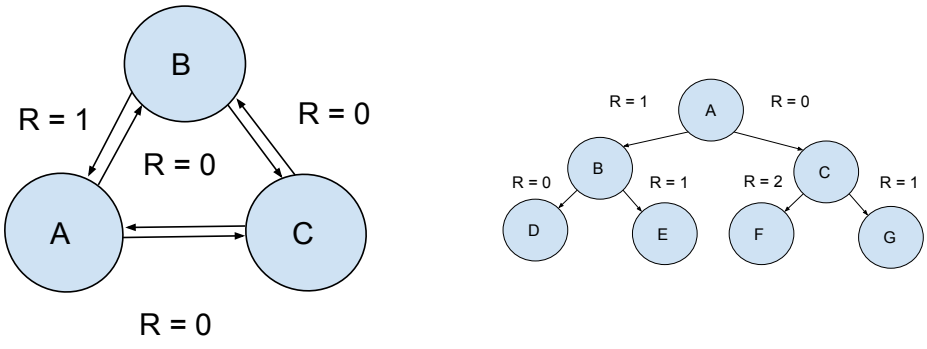


Figure 2: A motivating example to illustrate the differences between open-loop and closed-loop control. Left: A cyclic MDP where a reward is given for transitioning from state $B$ to state $A$. Right: A deterministic MDP, where persistent action selection would result in a pessimal return but an open-loop policy can learn the optimal policy.

To learn temporally abstract behavior in a bottom-up manner, we develop a new method that performs *open-loop* control where actions for $W - 1$ time steps into the future are determined at one particular state $s_t$. This corresponds to a temporal window of $W$, where states $s_{t+w}$ are ignored for

$w > 0$. This is in contrast to *closed-loop* control in which the policy always selects actions with knowledge of the state. Open-loop execution is fundamentally not Markovian, because the actions at intermediate states ($w > 0$) are determined by the state that executed the open-loop action sequence ($s_t$). To make this clear, we write the action-values with an additional time-window variable $w$. In the case of the MDP on the left in Figure 2, if the action-sequence is executed in state $A$, then we denote a closed-loop action-value for action $a = \text{left}$ as $Q(A, \text{left}, w = 0)$. After the first action in the sequence is executed in state $A$ and the agent transitions to $B$, we would write the resulting open-loop action-value for action $a = \text{right}$ as $Q(A, \text{right}, w = 1)$ (which approximates $Q(B, \text{right}, w = 0)$). When $w > 0$, the agent no longer has access to the state that it is currently in, only $w$ and the initializing state $s_t$, or some memory representation encoding $w$ and $s_t$.

To understand open-loop control in a tabular environment, consider the MDP on the right in Figure 2 with 3 states that involve the actions left and right. In tabular Q-learning, we store and learn the action-values stored in a matrix, $\mathbf{Q} \in \mathbb{R}^{3\times 2}$, for the 3 states with decision making and 2 actions respectively. In open-loop Q-learning, action-values of future time steps are learned. In particular, we learn to predict action-values at state $A$ and the action-values for the next time step, where the state at the next time step is either $B$ or $C$. The states $B$ and $C$ are temporally aggregated because they are both possible successors of state $A$. As a result, we only have to learn the 2 action-values from state $A$, reducing the dimension of the action-value matrix to $\mathbf{Q} \in \mathbb{R}^{2\times 2}$. In this example, both actions from $A$ are optimal because their optimal action-values are the same $Q^*(A, \text{left}) = Q^*(A, \text{right}) = 2$. However, open-loop Q-learning aliases the values for the intermediate states. Another form of open-loop control is action repetition. In this MDP, action persistence cannot represent the optimal policy that requires changing actions to reach the inner leaf node.

## 4 Recurrent Open-loop Control

Our central contribution is to learn temporally abstract action-values with a recurrent neural network that is initialized by the first state in the temporal window. The key idea is to not use the intermediate states as input, instead the agent must rely on the hidden-state of the RNN to predict the action-values of the interim states. These action-values can then induce a policy, such as the greedy policy with respect to action-values and the learned policy must learn temporally abstract behavior. Using a one-to-many recurrent network, denoted by $\mathbf{Q}(s_t\,;\,\theta)$ and parameterized by $\theta$, we take as input $s_t$ and predict not only the action values at state $s_t$ but also the action-values $W-1$ steps into the future. This corresponds to a total temporal window with length $W$. Only the first state is used as input, the action-value estimates for future time steps $t+w$ are calculated from the hidden state $\mathbf{h}^{(w)}$ which only depends on the first state $s_t$.

$$\mathbf{Q}(s_t\,;\,\theta) = \Big[Q(\mathbf{h}^{(0)}, \cdot\,;\theta), Q(\mathbf{h}^{(1)}, \cdot\,;\theta), \ldots, Q(\mathbf{h}^{(W-1)}, \cdot\,;\theta)\Big]$$

The recurrent network is used to estimate action-values, hence $Q(\mathbf{h}^{(w)}, a; \theta)$ represents the action-value estimate of $Q(s_{t+w}, a)$, using only the recurrent memory propagated from state $s_t$. Note that $Q(\mathbf{h}^{(0)}, a; \theta)$ is simply a feed-forward estimate of the action value at $Q(s_t, a)$ since there are no hidden-to-hidden transitions for the first time step in the temporal window. The subsequent entries of the vector are estimates of the action-values at the next time step irrespective of the state at that time step. Their value depends only on the hidden state of the recurrent network, which depends on the state $s_t$. To make inputs for our estimator clear, we will denote the output of each time step of the RNN as $Q(s_t, a\,;\,\theta, \mathbf{h}^{(w)}) = Q(\mathbf{h}^{(w)}, a\,;\theta)$, where the input is only the state $s_t$.

### 4.1 Learning With 1-Step and n-Step Bootstrapping

The first objective that we consider uses Q-learning with 1-step bootstrapped targets. We use the greedy target for each time step, where the target uses the underlying state and initialized hidden memory cell,

$$J(\theta) = \sum_{w=0}^{W-1} \Big(Q(s_{t+w}, a_{t+w}\,;\,\theta, \mathbf{h}^{(w)}) - r_{t+w+1} - \gamma \max_a Q(s_{t+w+1}, a\,;\,\theta', \mathbf{h}^{(0)}\Big)^2.$$

This is equivalent to persistent FQI if we restrict $\mathbf{h}^{(w)} = \mathbf{h}^{(0)}$ for all $w$. The 1-step update can be a naive approach for two reasons. First, the updates can be conflicting if the data is overlapping,

such as a sliding window over the stream of experience. In that case, we will have two successive temporal windows $\{s_t, s_{t+1}, \dots, s_{t+W-1}\}$, and $\{s_{t+1}, s_{t+2}, \dots, s_{t+W}\}$ where the update to $s_{t+1}$ is different in the first and second window. Although this can be alleviated by chunking the temporal windows to non-overlapping chunks, this would not be as data efficient as a sliding window. Leaving the issue of conflicting updates aside, 1-step updates are limited in that they do not make use of the temporal structure induced by the recurrent neural network. In the next section, we will discuss how to use n-step returns to leverage the structure in a temporal window of length $W$.

In order to train recurrent open-loop policies, we require sampling a sequence of length $W$ from the dataset. Once we have this temporal sequence, and unlike persistent FQI, we can use $n$-step returns for each time step in the RNN. In this case, there is only one target, and it is the greedy n-step target first target for $w = 0$ is the greedy n-step target. Then, we can write an objective analogous to the 1-step returns that considers the square difference,

$$J(\theta) = \Big(Q(s_t, a_t\,;\,\theta, \mathbf{h}^{(0)}) - \gamma^W \max_a Q(s_{t+W}, a\,;\,\theta', \mathbf{h}^{(0)}) - \sum_{s=0}^{W-1} r_{t+s+1}\Big)^2.$$

However, in doing so we are ignoring the predictions made by the network in the intermediate time steps. Instead, we can mix $n$-step targets for variable $n$ corresponding to the remaining number of steps in the sequence,

$$J(\theta) = \sum_{w=0}^{W-1} \Big(Q(s_{t+w}, a_{t+w}\,;\,\theta, \mathbf{h}^{(w)}) - \gamma^{W-w} \max_a Q(s_{t+W}, a\,;\,\theta', \mathbf{h}^{(0)}) - \sum_{s=w}^{W-1} r_{t+s+1}\Big)^2.$$

Note that unlike the targets in the 1-step update, all the n-step targets are greedy since the action-value function is with respect to the last time step and not at the intermediate time steps. This avoids the issue of conflicting updates because we are bootstrapping off the same state at the end of the sequence, instead of bootstrapping off of different intermediate states.

## 5 Related Work

There are three lines of work that are related to the recurrent open-loop policies that we propose in this paper. First, the open-loop policies discussed in Section 3 and 4 can be interpreted as a semi-Markov option (Sutton et al., 1999; Precup, 2000). While the method we propose can be cast in the options framework, there are subtle differences in both motivation and implementation. First, options are closed-loop policies that allow the agent to use high level actions for temporal abstraction and hierarchical control. However, our approach leverages the agent's ability to generalize from one time point to another, without any hierarchical controller. The implementation differs in that the RNN is executed in an open-loop, meaning that subsequent states are not used for action selection. Second, open-loop policies use action-values, and not option-values. Lastly, open-loop policies treat action-primitives as first-class objects. The last two points are tightly related by the fact that the action-values calculated by the open-loop policy are used to evaluate the policy, and since open-loop policies are semi-Markov options, it also evaluates the option itself.

Besides the vast options literature, there is a line of literature investigating temporally extended actions from both a theory and empirical perspective. Beginning with Bellemare et al. (2012) and continuing with Mnih et al. (2013), frame skips were introduced as one component of the success in training Atari agents. Empirical investigations treating the frame skip as a hyperparameter have also found that performance on many Atari games can heavily depend on a well-chosen frame skip (Lakshminarayanan et al., 2016; Braylan et al., 2015). Recent work by Metelli et al. (2020) proved that repeating actions at some persistence level induces a block MDP and leads to a better policy in practice, while the optimal persistent policy is within a bounded distance of the optimal policy of the base MDP. Lastly, there are other works exploring the use of recurrent networks for reinforcement learning. These works differ from ours in that they focus on solving the partial observability problem and non-Markovian tasks. For example, Bakker (2002) used an LSTM to learn an advantaged function to solve a maze problem with long-term dependencies, Wierstra et al. (2010) developed a recurrent policy gradient theorem, and (Hausknecht and Stone, 2015; Narasimhan et al., 2015) apply recurrent networks to Deep Q-Networks (DQN).

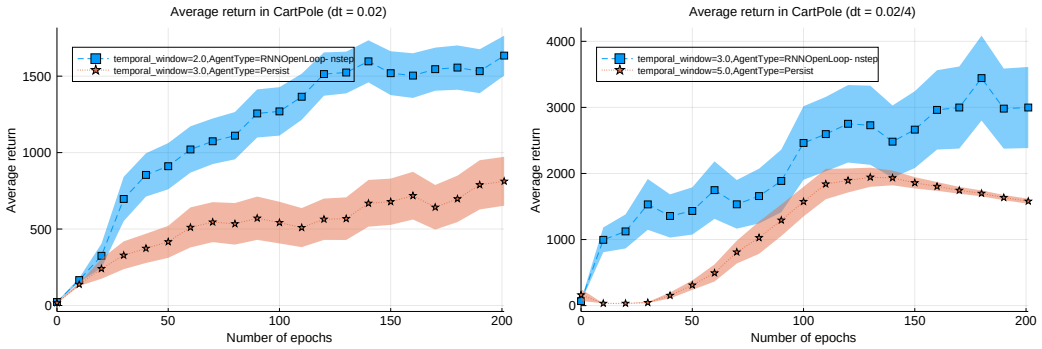


Figure 3: Average return over 100 runs for the best performer among the persistent agents and the proposed open-loop agents. The shaded region is a 95% confidence interval of the mean.

## 6 Experiments

To benchmark our proposed method, we choose to compare with the persistent fitted Q-iteration (Metelli et al., 2020) at the same temporal abstraction windows that we consider for recurrent open-loop policies. In particular, we compare with window sizes of $W = 1, 2, 3, 5$, which corresponds to making predictions $0, 1, 2, 4$ steps into the future. Note, for both our recurrent approach and persistent FQI, $W = 1$ reduces to vanilla FQI. We use the classic control environment Cartpole (Barto et al., 1983) with two different discretization rates, the original rate $dt = 0.02s$ and a quicker version $dt = (0.02/4)s$. A time limit is enforced for each discretization rate, 2000 time steps for the original environment and 8000 for the quicker environment. We train each agent offline with 100 episodes from a uniformly random behavior policy and plot the agent's performance at each optimization epoch. This environment setting was chosen because it provides a suitable learning signal for a uniformly random behavior policy.

The action-value network for the recurrent model consists of a feed-forward layer mapping the 4 dimensional input state to a 32 dimensional representation which initializes the recurrent memory cell, followed by a feed-forward layer mapping the memory cell to the 2 dimensional action-values. The action-value network for persistent FQI has the same architecture, except it does not have any recurrent connections and uses ReLU activations. Both models are trained with ADAM and the learning rate is searched over the set $\{0.1, 0.01, 0.001, 0.0001\}$. All experiments are averaged over 100 seeds and optimized for 200 epochs.

Our hypothesis is that recurrent open-loop controllers will match or outperform persistent action selection (Metelli et al., 2020). In particular, persistent action-selection will suffer when there is regularity in sequential states that requires switching actions. For example, state transitions in Cartpole are smooth, but repeating the same action will quickly lead the cart to falling over. On the other hand, a general open-loop policy can alternate between pushing the cart to the left and then to the right, irrespective of the state.

### 6.1 Results

The results for the Cartpole environment are given in Table 1. We see that a recurrent open-loop policy trained with n-step rewards can greatly improve over persistent action-selection. At larger window sizes however, recurrent open-loop policies trained with both 1-step and n-step returns perform similarly. For all window sizes, recurrent open-loop policies perform better than persistent policies. For the fast Cartpole environment ($dt = 0.02/4$), the results are given in Table 2. The best performing model is again recurrent open-loop with n-step returns. However, the difference between 1-step and n-step returns is not significant despite 100 independent runs. At a window size of 4, we do not see that recurrent open-loop policies are uniformly better. Despite averaging over 100 independent runs, the standard errors are still too high to discern performance differences at these larger window sizes.

| | Cartpole ($dt = 0.02$) | | | |
|---|---|---|---|---|
| | W=1 | W=2 | W=3 | W=5 |
| Persistent FQI | 196.9 (26.3) | 470.5 (126.8) | **541.8** (135.4) | 106.9 (5.0) |
| RNN (1-step) | 178.4 (9.8) | 906.2 (173.1) | 886.8 (168.0) | 140.4 (38.7) |
| RNN (n-step) | 244.4 (95.5) | **1596.8** (137.63) | 733.83 (155.7) | 177.2 (17.9) |

Table 1: Average return over 100 runs in the Cartpole environment with $dt = 0.02$ and for different temporal windows. The number in brackets is the standard error for the estimate of the mean. In bold is the best performing persistent and recurrent open-loop policy.

| | Fast Cartpole ($dt = 0.02/4$) | | | |
|---|---|---|---|---|
| | W=1 | W=2 | W=3 | W=5 |
| Persistent FQI | 1207.6 (405.7) | 1299.06 (404.68) | 1301.9 (71.6) | **1579.1** (37.6) |
| RNN (1-step) | 1021.2 (228.5) | 2365.8 (620.5) | 2575.6 (627.3) | 1656.1 (513.3) |
| RNN (n-step) | 1205.5 (419.6) | 2722.9 (624.3) | **2998.1** (613.6) | 2186.1 (425.5) |

Table 2: Average return over 100 runs in the Cartpole environment with $dt = 0.02/4$ and for different temporal windows. The number in brackets is the standard error for the estimate of the mean. In bold is the best performing persistent and recurrent open-loop policy.

## 7 Discussion

We have presented a method that uses a recurrent neural network for open-loop control in offline reinforcement learning. The proposed method is able to learn more flexible open-loop policies compared to naively repeat actions, and outperforms it on simple continuous control tasks at varying discretization rates. Some directions for future research remain, particularly in extending these results to the online setting. Despite being trained with back-propagation through time, the relatively small temporal windows used for frame skipping would bound the computation and diminish vanishing/exploding gradient issues. The online setting also lends itself to ideas from the options literature, such as interrupting the recurrent network as done in intra-option learning. This would allow for even more flexible execution of recurrent open-loop policies. Lastly, a theoretical characterization for the ease of learning a slower policy would provide further justification for hierarchical and open-loop RL.

## References


Agarwal, R., Schuurmans, D., and Norouzi, M. (2020). An optimistic perspective on offline reinforcement learning. In International Conference on Machine Learning.

Bakker, B. (2002). Reinforcement learning with long short-term memory. In Advances in neural information processing systems, pages 1475–1482.

Barto, A. G. and Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. Discrete event dynamic systems, 13(1-2):41–77.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. IEEE transactions on systems, man, and cybernetics, (5):834–846.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2012). The arcade learning environment: An evaluation platform for general agents. arXiv:1207.4708.

Braylan, A., Hollenbeck, M., Meyerson, E., and Miikkulainen, R. (2015). Frame skip is a powerful parameter for learning to play atari. In Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep Learning. MIT Press.

Hausknecht, M. and Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. arXiv:1507.06527.

Lakshminarayanan, A. S., Sharma, S., and Ravindran, B. (2016). Dynamic frame skip deep q network. arXiv:1605.05365.

Lattimore, T. and Szepesvári, C. (2020). Bandit Algorithms. Cambridge University Press.

Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. arXiv:2005.01643.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. (2017). Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. arXiv:1709.06009.

Metelli, A. M., Mazzolini, F., Bisi, L., Sabbioni, L., and Restelli, M. (2020). Control frequency adaptation via action persistence in batch reinforcement learning. arXiv:2002.06836.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv:1312.5602.

Narasimhan, K., Kulkarni, T., and Barzilay, R. (2015). Language understanding for text-based games using deep reinforcement learning. arXiv:1506.08941.

Precup, D. (2000). Temporal Abstraction in Reinforcement Learning. PhD thesis.

Riedmiller, M. (2005). Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In European Conference on Machine Learning, pages 317–328. Springer.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artificial intelligence, 112(1-2):181–211.

Tallec, C., Blier, L., and Ollivier, Y. (2019). Making deep q-learning methods robust to time discretization. arXiv:1901.09732.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 5026–5033. IEEE.

Wierstra, D., Förster, A., Peters, J., and Schmidhuber, J. (2010). Recurrent policy gradients. Logic Journal of the IGPL, 18(5):620–634.