Using RNNs for the internal state representation when learning POMDPs

Yunshu Ouyang
ouyangy@student.ethz.ch

Abstract

Partially Observable Markov Decision Process (POMDP) is the standard model that captures the partial-information structure in Reinforcement Learning, which is often found in settings where agents learn to act even though the complete state of the underlying system is not given. It is well known that learning POMDPs is difficult in theory as it is both statistically and computationally intractable. There has been some attempts in extending the theory of Markov Decision Processes (MDPs) into the partially observed setting. However, most results rely on very restrictive assumptions about the environment. In this work, we investigate ways to improve upon the known algorithms for learning POMDPs by the use of Recurrent Neural Networks (RNNs) for the internal state representation. We theoretically quantify the performance gap between RNNs and Finite State Controllers (FSCs) in certain hand-crafted environments. Empirically, we compare the two algorithms in different modified MDP environments.

1 Introduction

Many real-world sequential decision making problems can be cast as Reinforcement Learning (RL) problems under partial observability, in which agents learn to make a sequence of decisions despite lacking complete information about the underlying state of the system. Partially Observable Markov Decision Process (POMDPs) is the standard model in RL that captures the partial-information structure. Learning POMDPs is notoriously difficult in theory due to its statistical and computational intractability in general.

As the RL theory gradually gained more attention in the fully observable setting, there has been various attempts in also extending the theory to the partially observed setting. Efroni et al. [1] gives theoretical sample-efficiency results based on strategic exploration for a sub-class of POMDPs. Azizzadenesheli et al. [2], Jin et al. [3] and Xiong et al. [4] show polynomial sample complexity for the under-complete and tabular POMDP setting by the use of spectral methods. Nevertheless, most theoretical results rely on strong assumptions about the partially observed setting and various aspects of POMDPs remain untouched.

The goal of this project is to investigate ways to improve upon the known algorithms for learning POMDPs. In particular, we focus on a theoretical analysis of the improvement on the performance of the agent when using recurrent neural networks for the internal state representation compared to using a fixed transition kernel.

2 Related works

Theoretical results for solving POMDPs: It is well known that the problem of computing the optimal policy is computationally intractable for POMDPs [5], as the optimal policy depends on the complete history of the system. Furthermore, Jin et al. [3] showed learning an unknown POMDP incurs a

Semester Project FS2022, Department of Computer Science, ETH Zürich.

sample complexity that scales exponentially with the horizon. To overcome this computational issue, different model-based and model-free RL approaches have been proposed in the literature that incorporate finite memory into the controller. Even though various algorithms such as the actor-critic framework has shown great promise in learning POMDPs in practice, theoretical analysis of these solvers seem largely absent. Most theoretical results for POMDPs restrict themselves to a subclass of POMDPs only. Indeed, Efroni et al. [1] identified a sub-class of POMDPs called the m-step decodable POMDPs that admits theoretical sample-efficiency based on strategic exploration. This sub-class of POMDPs assumes the state of the POMDP can be completely determined by the m past observations and actions. Azizzadenesheli et al. [2], Jin et al. [3] and Xiong et al. [4] obtain polynomial sample complexity results for the undercomplete tabular POMDP setting. This is another standard assumption in the literature of POMDPs. The undercomplete setting covers all tabular POMDPs in which the number of observations is larger than the number of latent states. Du et al. [6] and Misra et al. [7] provide algorithms that are provably efficient in the rich observation setting, where observation space is extremely large or possibly infinite, together with some structural conditions that permit sample efficiency.

Theoretical results for Recurrent Neural Networks: Recurrent Neural Networks (RNNs) have been widely used as an effective tool for a range of applications that store and process temporal sequences. However, it still remains one of the least theoretically understood neural networks. Khrulkov et al. [8] shows a certain class of RNNs are exponentially efficient than a feedforward neural network. Allen-Zhu et al. [9] proves the convergence of (stochastic) gradient descent for training RNNs. The Long Short Term Memory (LSTM) [10] architecture was proposed as a way to resolve the exploding and vanishing gradient problem, a problem which constrains RNNs to have a short-term memory.

Using Recurrent Neural Networks for POMDPs: In order to incorporate finite memory into the controller as to overcome the computational challenge, a plethora of approaches have been proposed which make use of RNNs. Hausknecht and Stone [11] modify DQN (DRQN) to handle POMDPs by replacing the first post-convolutional fully-connected layer with a recurrent LSTM. Wierstra et al. [12] presents Recurrent Policy Gradients. It involves approximating a policy gradient for a RNN by backpropagating return-weighted characteristic eligibilities through time. Bakker [13] presents a model-free algorithm for POMDPs which uses LSTM, advantage learning and directed exploration. The LSTM network learns Q-values and adapts the Q-learning exploration by training a non-recurrent NN to predict the variance of the Q-values. Heess et al. [14] extended Deterministic Policy Gradient to Recurrent DPG by incorporating LSTM. Meng et al. [15] improves upon that by proposing LSTM-TD3 which handles continuous control tasks with large observation and action spaces. Even though these methods have shown great performance in practice, theoretical analysis of the effect of using RNNs for learning POMDPs is largely absent in the literature. In this project, we investigate this fundamental problem.

3 POMDPs and Finite-State Controllers

This section gives a formal description of POMDPs, Finite-State Controllers and Recurrent Neural Networks which will be the focus of this work.

The POMDPs analyzed in this project can be modeled as a discrete-time dynamical system with a finite state space $\mathcal{X}$, and finite control space $\mathcal{U}$. The system state $\{x_k : k \geq 0\}$ is a time-homogeneous controlled Markov chain, which evolves according to $x_{k+1} \sim \mathcal{P}(\cdot|x_l, u_k)$, $k = 0, 1, \dots$, where $\mathcal{P}$ is the transition kernel, and $u_k \in \mathcal{U}$ is the control at time k. We assume the intial state x_0 is fixed. The system state is available to the controller only through a (noisy) discrete memoryless observation channel $y_k \sim \Phi(\cdot|x_k)$, $k = 0, 1, \dots$, where $y_k \in \mathcal{Y}$ is the observation and $\Phi(\cdot|x) \in \rho(\mathcal{Y})$ is the observation channel for any $x \in \mathcal{X}$. Applying control $u \in \mathcal{U}$ at state $x \in \mathcal{X}$ yields a reward $r(x, u) \in [0, r_{\max})$.

An admissible policy $\pi = (\mu_0, \mu_1, \dots)$ is a sequence of mappings $\mu_k : \mathcal{H} \times \mathcal{Y}^k \times \mathcal{U}^k \to \mathcal{U}$ for all $k \geq 0$. This project only considers deterministic policy. Let $\mathcal{Z}$ be a finite set, and consider a stochastic process $\{z_k : k \geq 0\}$ over $\mathcal{Z}$, that evolves according to the following transition:

$$z_{k+1} \sim \phi(\cdot|z_k, y_k, u_k),$$

where ϕ is a transition kernel.

For a given admissible policy $\pi \in \Pi_{F,\mathcal{Z},\phi}$, the value function is defined as the γ-discounted reward given the initial state x_0:

$$V^{\pi}(x_0) = \mathbb{E}\Big[\sum_{k=0}^{\infty} \gamma^k r(x_k, u_k)\Big| x_0\Big] \tag{1}$$

The ultimate objective is to find the optimal policy over the class of admissible policies that maximizes the discounted reward, namely,

$$\max_{\pi \in \Pi_{F,\mathcal{Z},\phi}} V^{\pi}(x_0)$$

where the optimization is over all transition kernels ϕ and policies $\pi \in \Pi_{F,\mathcal{Z},\phi}$.

3.1 Bayes Filtering and Belief State Formulation

It is well known that the POMDP objective can be reformulated as a perfect state information problem (MDP) by enlarging the state space (Kumar and Varaiya [16], Bertsekas [17], Krishnamurthy [18]). Let

$$b_k(x, h) = \mathbb{P}(x_k = x | h_k = h)$$

be the belief state at time $k \geq 0$. The belief can be computed in a recursive way by the following filtering transformation:

$$b_k(x, h) = \frac{\sum_{x'} b_{k-1}(x', h_{k-1}) \mathcal{P}(x|x', u_{k-1}) \mathbb{I}[\Phi(x) = y_k]}{\sum_{x',x''} b_{k-1}(x', h_{k-1}) \mathcal{P}(x''|x', u_{k-1}) \mathbb{I}[\Phi(x'') = y_k]}$$

which follows from the Bayes theorem.

For any $u \in \mathcal{U}$, let

$$\tilde{r}(b_k, u) = \sum_{x \in \mathcal{X}} b_k(x) r(x, u).$$

Then, the problem reduces to a fully observable MDP where (b_k, u_k) forms a controlled Markov chain, and action u_k at belief state b_k yields a reward $\tilde{r}(b_k, u_k)$.

The main drawback of this representation of POMDPs is the fact that the belief states are continuous-valued and of infinite size. Therefore, even though it can theoretically by used to solve the partially observable setting, it is in reality a highly intractable policy search problem.

3.2 RNNs vs FSCs for POMDPs

In this project, we compare the performance of using RNNs for the internal state representation against an important class of finite-state controllers called sliding-block controllers (SBC) which was shown to achieve good practical performance.

The finite-state controllers model a subclass of admissible policies $\pi = (\mu_0, \mu_1, \dots)$ such that the policy at time k bases its decision on the latest observation y_k and the internal state of the controller z_k. We denote this policy subclass by $\Pi_{F,\mathcal{Z},\phi}$. z_k is called the internal state of the controller. The knowledge of the controller about the system at time k is $(y_k, z_k) \in \mathcal{Y} \times \mathcal{Z}$, hence $\mathcal{H} = \mathcal{Y} \times \mathcal{Z}$.

For a given block-length $m > 0$, the internal state of a sliding-block controller is defined as follows:

$$z_k = (y_{k-m}^{k-1}, u_{k-m}^{k-1}) \in \mathcal{Y}^m \times \mathcal{U}^m$$

In this case, $\mathcal{H} = \mathcal{Y}^{m+1} \times \mathcal{U}^m$.

A RNN is a special type of an artificial neural network which is designed to work for sequential data such as time series. They have the concept of 'memory' which are hidden states that store relevant information of previous inputs to generate the next output of the sequence. We give a formal definition below. Let $y_t \in \mathbb{R}^k$ be the output of the network at time step t, $h_t \in \mathbb{R}^m$ be the hidden state vector at time t, $x_t \in \mathbb{R}^n$ be the input at time t. W_x, W_h, W_y the corresponding weight matrices and b_h, b_y the corresponding bias vectors associated with the neural network. Then, given an activation function ϕ, the hidden state is computed as

$$h_{t+1} = \phi(W_x x_t + W_h h_t + b_h)$$

and the output as

$$y_t = \phi(W_y h_t + b_y)$$

4 Theoretical results

In this section, we analyze the performance of sliding-block controllers against recurrent neural networks for different POMDPs. The aim is to show that the performance of sliding-block controllers can always be upper bounded by the one of recurrent neural networks. In some cases, an infinite block-length for the SBCs is required in order to achieve the same performance as a simple RNN.

4.1 Action defined POMDP

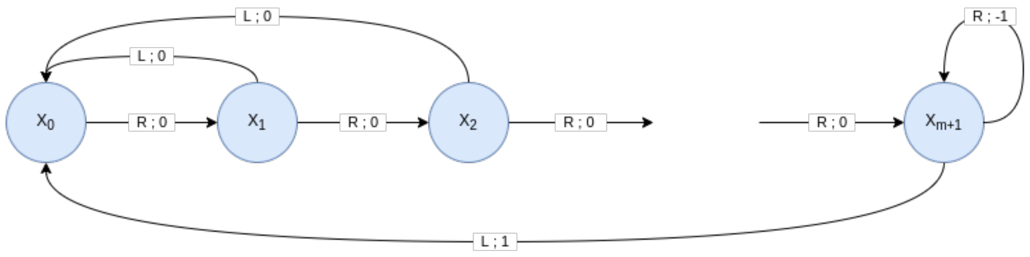


Figure 1: State transition of the MDP. The first component on the arrow describes the action and the second component the reward in taking the corresponding action at the relevant state.

We first present the main result of this section.

Theorem 4.1. *Given any SBC with block-length $m > 0$, there exist a RNN and a POMDP for which the expected return of the SBC is upper bounded by the expected return of the RNN.*

The proof of Theorem 4.1 is constructive. The following definition describes the corresponding POMDP.

Definition 4.2. *(Description of the POMDP) Consider a POMDP with state space $\mathcal{X} = \{x_0, \dots, x_{m+1}\}$, observation space $\mathcal{Y} = \{y_0, \dots, y_{m+2}\}$ and control space $\mathcal{U} = \{L, R\}$. The initial state is x_0. Figure 1 shows the corresponding MDP. The discrete memoryless observation channel is an erasure channel, meaning that the relevant information gets erased with some probability $p \in [0, 1]$.*

$$\Phi(y|x_i) = \begin{cases} 1-p & \text{if } y = y_i \\ p & \text{if } y = y_{m+2} \end{cases}$$

For the POMDP described in Definition 4.2, we have the following observations.

1. The number of states $m + 3$ of the POMDP depends on the block-length of the SBC.
2. The state of the POMDP is uniquely defined by the control. This suggests the optimal policy does not need any state information. Therefore, this POMDP is essentially a MDP.

Before comparing the performance of SBC and RNN on this POMDP, it is worth analyzing the optimal policy.

Lemma 4.3 (Expected return of the optimal policy)**.** *The optimal policy for the POMDP described in Definition 4.2 consists of always taking m steps R followed by one step L. The expected return achieved is*

$$\frac{\gamma^{m+1}}{1-\gamma^{m+2}}$$

Proof. The optimal policy is self-explained by looking at Figure 1. We still need to calculate the expected return. Using Equation (1), we have

$$\begin{aligned}
\mathbb{E}[\text{return}] &= V^{\pi*}(x_0) \\
&= \sum_{i=0}^{\infty} \gamma^i \mathbb{I}[i \equiv m+1 \mod (m+2)] \\
&= \sum_{i=0}^{\infty} \gamma^{i(m+2)+(m+1)} \\
&= \frac{\gamma^{m+1}}{1-\gamma^{m+2}}
\end{aligned}$$

□

In the following, we show that RNN only needs a hidden state of size one to model the optimal policy.

Lemma 4.4. *(Optimality of the RNN) A simple RNN is able to model the optimal policy and thus achieve optimal expected return.*

Proof. We do a constructive proof in which we show the RNN models the optimal policy. The RNN uses the following notation: L, resp. R, is encoded as 1, resp. 0.

The history information $h_k \in \mathbb{R}$ and control u_k for step k are defined recursively as follows:

$$h_k \longleftarrow (1 \quad -(m+2))\, \phi\left(\begin{pmatrix} 1 \\ 1 \end{pmatrix} (h_{k-1}) + \begin{pmatrix} 1 \\ -m \end{pmatrix} \right) \qquad u_k \longleftarrow \phi(h_k - m)$$

where ϕ is the ReLU activation function.

With $h_0 = 0$ as initial value, one can easily check that the RNN models the optimal policy and h corresponds to the current state of the system, that is

$$h_{t+1} = \begin{cases} h_t + 1 & \text{if } t \leq m \\ 0 & \text{otherwise} \end{cases} = x_{t+1} \qquad u_t = \begin{cases} 0 & \text{if } h \leq m \\ 1 & \text{otherwise} \end{cases}$$

Therefore, this RNN indeed models the optimal policy and thus achieves optimal expected return. □

The last step consists in giving an upper bound on the expected return of a SBC with block-length m.

Lemma 4.5 (Expected return of a SBC with block-length m)**.** *The expected return of a SBC with block-length m is upper bounded by*

$$\frac{\gamma^{m+1}(1-p^{m+1})}{1-\gamma^{m+1}p^{m+1}-\gamma^{m+2}(1-p^{m+1}),}$$

Proof. First of all, suppose the controller is able to choose the optimal control if any state information is contained in the history block or if the current state is given. In the other case, notice that the controller in unable to distinguish between states x_m and x_{m+1} since in both cases, the last m controls are all R. Therefore, in this case, the best choice for the controller would be to perform control L in order to avoid the potential negative reward in state x_{m+1}.

Notice that apart from states x_m and x_{m+1}, the controller is able to infer the current state of the system by looking directly at the history block and identifying the last control L that was performed.

For the current controller, the value function v_i of state x_i with $i \in \{0, \ldots, m+1\}$ can be summarized using the following system of equations:

$$\begin{aligned}
v_i &= \gamma v_{i+1} \quad \text{for } 0 \leq i < m \\
v_m &= p^{m+1}\gamma v_0 + (1-p^{m+1})\gamma v_{m+1} \\
v_{m+1} &= 1 + \gamma v_0
\end{aligned}$$

The equation for v_m follows from the fact that the transition from x_m to x_0 happens if and only if all previous $m+1$ state information were not given (including the current state). This happens with probability at most $1-p^{m+1}$ at any step $k \geq 0$.

The system can be simplified into a single equation for v_0:

$$\begin{aligned}
v_0 &= \gamma^m v_m \\
&= \gamma^{m+1}p^{m+1}v_0 + \gamma^{m+1}(1-p^{m+1})v_{m+1} \\
&= \gamma^{m+1}p^{m+1}v_0 + \gamma^{m+1}(1-p^{m+1})(1+\gamma v_0) \\
&= (\gamma^{m+1}p^{m+1} + \gamma^{m+2}(1-p^{m+1}))v_0 + \gamma^{m+1}(1-p^{m+1})
\end{aligned}$$

Rewriting gives

$$v_0 = \frac{\gamma^{m+1}(1-p^{m+1})}{1-\gamma^{m+1}p^{m+1}-\gamma^{m+2}(1-p^{m+1})},$$

which concludes the proof. □

The previous results allows us the prove Theorem 4.1.

Proof of Theorem 4.1. Combining Lemma 4.5 and Lemma 4.4, it suffices to prove for any $m > 0$,

$$\frac{\gamma^{m+1}}{1-\gamma^{m+2}} > \frac{\gamma^{m+1}(1-p^{m+1})}{1-\gamma^{m+1}p^{m+1}-\gamma^{m+2}(1-p^{m+1})}.$$

The above inequality can be rewritten as

$$\begin{aligned}
&1-\gamma^{m+1}p^{m+1}-\gamma^{m+2}(1-p^{m+1}) > (1-\gamma^{m+2})(1-p^{m+1}) \\
&\Leftrightarrow 1-\gamma^{m+1}p^{m+1}-\gamma^{m+2}+\gamma^{m+2}p^{m+1} > 1-\gamma^{m+2}-p^{m+1}+\gamma^{m+2}p^{m+1} \\
&\Leftrightarrow p^{m+1} > \gamma^{m+1}p^{m+1}
\end{aligned}$$

where the last inequality is straightforward. □

4.2 A 2-state POMDP

One downside of the previous result from Theorem 4.1 is that the POMDP actually depends on the block-length of the SBC. Furthermore, the construction proof makes use of a POMDP whose control completely determine the state hence one can say it is technically a MDP. In this section, we prove a more general claim for a true POMDP whose state is not completely specified by the control and investigate the performance change for a SBC when increasing its block-length.

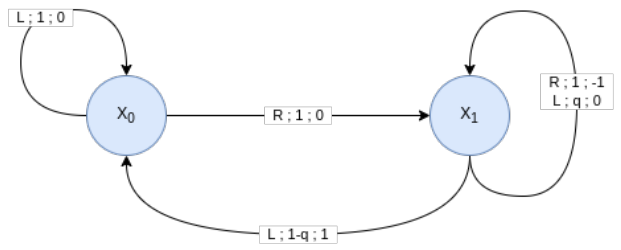


Figure 2: State transition of the MDP. The first component on the arrow describes the action, the second component the probability of this transition given the current state-action pair and the last component the corresponding reward.

The following theorem summarizes the main result of this section.

Theorem 4.6. *The exist a "true" POMDP for which the the performance of the SBC improves when increasing the block-length and reaches the same optimal performance as the RNN when the block-length is large enough.*

The proof of the above theorem follows a similar procedure as in Section 4.1. As the proof is constructive, we first define a POMDP.

Definition 4.7 (Description of the POMDP)**.** *Consider a POMDP with state space $\mathcal{X} = \{x_0, x_1\}$, observation space $\mathcal{Y} = \{y_0, y_1, y_2\}$ and control space $\mathcal{U} = \{L, R\}$. The initial state is x_0. Figure 2 shows the corresponding MDP with parameter $q \in [0, 1]$. The discrete memoryless observation channel is an erasure channel, hence,*

$$\Phi(y|x_i) = \begin{cases} 1 - p & \text{if } y = y_i \\ p & \text{if } y = y_2 \end{cases}$$

for $i \in \{0, 1\}$.

The next step consists in analyzing the optimal policy for this POMDP. For this analysis, we will make use of the belief-state description of the POMDP described in Section 3.1.

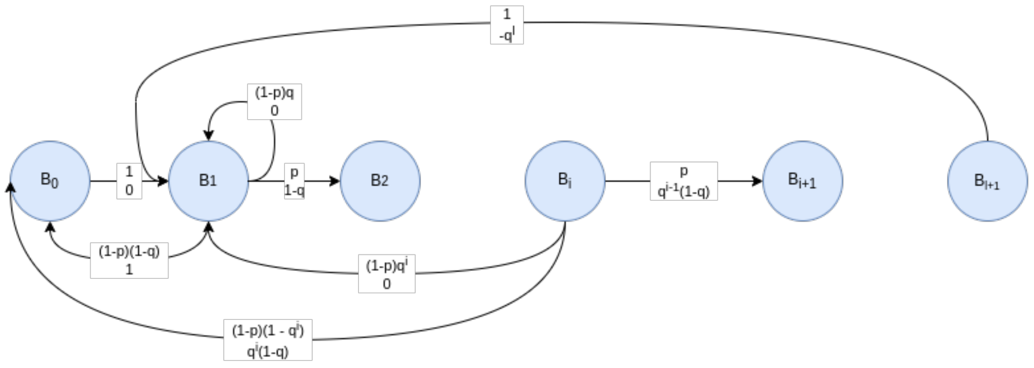


Figure 3: Transitions of the policy in the belief-state POMDP. The first component describes the probability of the corresponding transition and the second component describes its expected return.

Lemma 4.8 (Expected return of the optimal policy)**.** *For an optimal policy, its expected return is*

$$\gamma \cdot \max\left(\max_{l \in \mathbb{N}} \frac{-(\gamma p q)^l + (1-q)\frac{1-(\gamma p q)^l}{1-\gamma p q} - (1-p)(1-q)q\frac{1-(\gamma p q^2)^l}{1-\gamma p q^2}}{1 - (\gamma p)^l \gamma - (1-\gamma)\gamma(1-p)q\frac{1-(\gamma p q)^l}{1-\gamma p q} - \gamma^2(1-p)\frac{1-(\gamma p)^l}{1-\gamma p}}; \right. \tag{2}$$

$$\left. \frac{(1-q)\frac{1}{1-\gamma p q} - (1-p)(1-q)q\frac{1}{1-\gamma p q^2}}{1 - (1-\gamma)\gamma(1-p)q\frac{1}{1-\gamma p q} - \gamma^2(1-p)\frac{1}{1-\gamma p}} \right). \tag{3}$$

Proof. We first need to understand the structure of the optimal policy for the POMDP described in Definition 4.7. First notice that the control in the case the state is known is straightforward: control R at state x_0 and control L at state x_1. Note also that whenever control R is chosen, the end state is always x_1. The freedom for the optimal policy thus lies in the number of consecutive control L to perform following an action R given that no state information is given throughout the process. Let this number be $l \in \mathbb{N} \cup \{\infty\}$. The problem therefore reduces to optimizing the parameter l.

Figure 3 shows the belief-state reached by the optimal policy together with the transition graph. The number of states depends on l and is infinite in case $l = \infty$. The belief-states $b_i \in \mathbb{R}^2$ are defined as follows:

$$b_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{and} \quad b_i = \begin{pmatrix} 1 - q^{i-1} \\ q^{i-1} \end{pmatrix} \text{ for } i \in \{1, \dots, l+1\}.$$

The j-th coordinate of a belief-state describes the probability of being at state x_{i-1}.

Next, we will calculate the value function of the different belief-states. b_0 corresponds to the initial state x_1. Therefore, the expected return of the optimal policy is the value function of state b_0. Let v_i be the value function of the optimal policy for belief-state b_i. To calculate the value function of the policy, we get the following system of equations:

$$\begin{cases} v_0 = \gamma \cdot v_1 \\ v_i = (1-p)\Big(q^i \gamma v_1 + (1-q^i)(q^{i-1}(1-q) + \gamma v_0)\Big) + p\Big(q^{i-1}(1-q) + \gamma \cdot v_{i+1}\Big) \quad i \in [l] \\ v_{l+1} = -q^l + \gamma \cdot v_1 \end{cases}$$

To simplify the notations, consider

$$\begin{aligned} a &:= \gamma p \\ b_i &:= q^{i-1}(1-q) - (1-p)q^{2i-1}(1-q) \\ c_i &:= (1-p)(1-q^i)\gamma^2 + (1-p)q^i\gamma \end{aligned}$$

Then, rewriting gives for $i \in \{1, \dots, l\}$,

$$v_i = av_{i+1} + b_i + c_i v_1$$

and

$$\begin{aligned} v_1 &= \sum_{i=1}^{l-1} a^{i-1}(b_i + c_i v_1) + a^l v_{l+1} \\ &= \sum_{i=1}^{l-1} a^{i-1}(b_i + c_i v_1) - a^l q^l + a^l \gamma \cdot v_1 \end{aligned}$$

Rewriting gives

$$\begin{aligned} v_1 &= \frac{-a^l q^l \sum_{i=1}^{l-1} a^{i-1} b_i}{1 - a^l\gamma - \sum_{i=1}^{l-1} a^{i-1} c_i} \\ &= \frac{-(\gamma pq)^l + \sum_{i=1}^{l}\Big((1-q)(\gamma pq)^{i-1} - (1-p)(1-q)q(\gamma pq^2)^{i-1}\Big)}{1 - (\gamma p)^l\gamma + \sum_{i=1}^{l}\Big(\gamma^2(1-p)q(\gamma pq)^{i-1} - \gamma^2(1-p)(\gamma p)^{i-1} - (1-p)\gamma q(\gamma pq)^{i-1}\Big)} \\ &= \frac{-(\gamma pq)^l + (1-q)\frac{1-(\gamma pq)^l}{1-\gamma pq} - (1-p)(1-q)q\frac{1-(\gamma pq^2)^l}{1-\gamma pq^2}}{1 - (\gamma p)^l\gamma - (1-\gamma)\gamma(1-p)q\frac{1-(\gamma pq)^l}{1-\gamma pq} - \gamma^2(1-p)\frac{1-(\gamma p)^l}{1-\gamma p}} \end{aligned}$$

From here, one can see that the optimal policy is the one that chooses l such that v_1 is maximized. In the case where $l = \infty$, which means the agent never performs the control R without observing the current state, the value function in this case would then be:

$$v_1 = \sum_{i=1}^{\infty} a^{i-1}(b_i + c_i v_1)$$

which gives

$$v_1 = \frac{(1-q)\frac{1}{1-\gamma pq} - (1-p)(1-q)q\frac{1}{1-\gamma pq^2}}{1 - (1-\gamma)\gamma(1-p)q\frac{1}{1-\gamma pq} - \gamma^2(1-p)\frac{1}{1-\gamma p}}$$

Putting everything together, the expected return of the optimal policy is thus

$$\gamma \cdot \max\left(\max_{l\in\mathbb{N}} \frac{-(\gamma pq)^l + (1-q)\frac{1-(\gamma pq)^l}{1-\gamma pq} - (1-p)(1-q)q\frac{1-(\gamma pq^2)^l}{1-\gamma pq^2}}{1 - (\gamma p)^l\gamma - (1-\gamma)\gamma(1-p)q\frac{1-(\gamma pq)^l}{1-\gamma pq} - \gamma^2(1-p)\frac{1-(\gamma p)^l}{1-\gamma p}} ; \right.$$
$$\left. \frac{(1-q)\frac{1}{1-\gamma pq} - (1-p)(1-q)q\frac{1}{1-\gamma pq^2}}{1 - (1-\gamma)\gamma(1-p)q\frac{1}{1-\gamma pq} - \gamma^2(1-p)\frac{1}{1-\gamma p}} \right).$$

□

In the following, we will analyze an instance of the POMDP with the following parameters: $p = 0.98$, $q = 0.89$ and $\gamma = 0.9$.

Lemma 4.9. *For the POMDP described in Definition 4.7 with parameters* $p = 0.98$*,* $q = 0.89$ *and* $\gamma = 0.9$*, the expected return of the optimal policy is approximately* 0.97*.*

Proof. Plugging these values into Equation (2), the resulting function with respect to l is analytically intractable. In order to find its maximum, ... shows plots of the function. One can see that when $l = \infty$, the value function for x_1 is approximately 0.865. The integer l which maximizes v_1 is $l = 15$ and achieves $v_1 = 1.02$. The expected return in this case is $v_0 = \gamma \cdot v_1 \approx 0.97$. □

Similar to the previous section, we now prove the existence of a RNN which models the optimal policy.

Lemma 4.10 (Optimal expected return of the RNN)**.** *There exists a RNN with hidden state dimension* 2 *and depth* 2 *(number of hidden layers) which achieves optimal expected return.*

Proof. This proof is constructive. Let the control u and the $h \in \mathbb{R}$ be recursively defined as follows:

$$\begin{aligned} h &\longleftarrow \begin{pmatrix} 1 & 1 \end{pmatrix} \phi\left(\begin{pmatrix} 1 & -l & -l \\ 0 & 0 & 1 \end{pmatrix} \phi\left(\begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} h \\ x \end{pmatrix} + \begin{pmatrix} 1 \\ -(l-2) \\ 0 \end{pmatrix} \right) + \begin{pmatrix} 0 \\ -1 \end{pmatrix} \right) \\ u &\longleftarrow 1 - \phi(-h + 1) \end{aligned}$$

h keeps track of the current belief state where b_0 and b_l are both summarized into $h = 0$. One can check that, given $x = 0$ meaning non state information is given, h is incremented by one in case $h < l$ and becomes 0 otherwise. If $x \neq 0$, then h takes up 0 or 1 since b_0, resp. b_1 corresponds to S_1, resp. S_2. For the control u, it is 0, meaning taking action R, if and only if $h = 0$. This RNN perfectly models the optimal policy and uses two hidden layers as opposed to the SBC which needs a block-length of 15. □

The last part of the proof is to analyze the expected return of the SBC.

Lemma 4.11. *A SBC requires a block-length of* $m = 15$ *is order to model the optimal policy. For* $m < 15$*, the expected return is strictly smaller than the one of the RNN.*

Proof. Now consider a SBC with block-length m. If $m \geq 15$ then, the SBC is able to model the optimal policy and will therefore achieve highest expected return.

Suppose now that $m < l$. The optimal policy in that case would be to perform the optimal control whenever the current state information is given. When encountering no information states, there are two choices. Either the policy decides to take action R after performing k control L following state S_1, with the restriction that $k \leq m$, or the policy decides to always perform L.

That is, the SBC has the choice between the optimal policy with $k \in \{1, \ldots, m\} \cup \{\infty\}$. Table 1 shows the different value functions v_1 depending on k. □

$k :=$ Number of consecutive controls L	1	3	5	7	∞	9	11	13	15
Function value v_1	-7.45	-1.64	-0.0209	0.614	0.865	0.892	1.02	1.06	**1.08**

Table 1: For the SBCs, with a block-length $m \leq 7$, it is best to follow the policy $l = \infty$, never perform the action R without seeing the current state in order to avoid negative rewards. With a block-length m between 7 and 13, performing the action R once m control L have been performed is beneficial. When the block-length reaches 15, the SBC is able to model the optimal policy.

The proof of Theorem 4.6 directly follows from Lemma 4.11

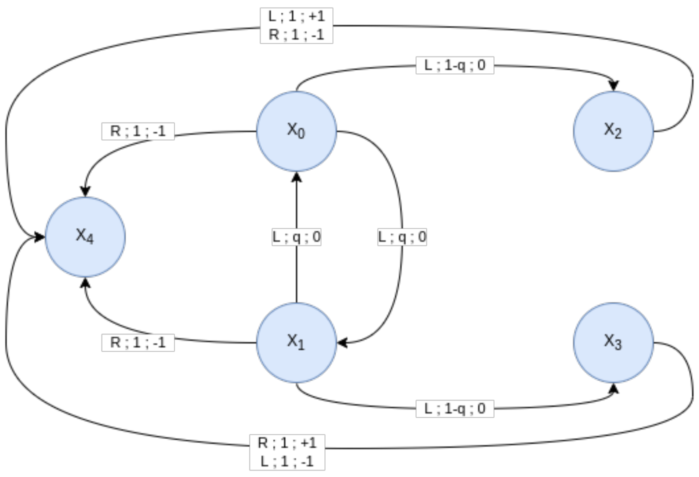


Figure 4: Transition graph of the MDP. x_0 is the initial state and x_4 the final state. For each transition, the first coordinate corresponds to the control, the second coordinate the probability of the transition and the last coordinate the corresponding reward.

4.3 Infinite-block length SBC required for modeling optimal policy

In this section, we show the existence of POMDPs in which a RNN is able to model the optimal policy whereas a SBC requires infinite block length in order to do so.

Definition 4.12 (Description of the POMDP)**.** *Consider a POMDP with state space* $\mathcal{X} = \{0, 1, 2, 3, 4\}$*, observation space* $\mathcal{Y} = \{0, 1, 2\}$ *and control space* $\mathcal{U} = \{0, 1\}$*. Figure 4 shows the corresponding MDP and the discrete memoryless observation channel is defined as follows:*

$$y = \begin{cases} 0 & \textit{if } x \in \{0, 1\} \\ 1 & \textit{if } x \in \{2, 3\} \\ 2 & \textit{otherwise} \end{cases}$$

The initial state of the MDP is always x_1*.*

For the POMDP described in Definition 4.12, we have the following observations.

1. This POMDP essentially requires the agent to keep track of the parity of the number of control 0 performed since the beginning.
2. The state x_4 is final and for each run, only a single reward of either $+1$ or -1 is given upon reaching the final state.
3. Unlike the previous two POMDPs, this one is not an erasure channel anymore. Instead each observation summarizes multiple states of the underlying MDP.

Before showing optimality of the RNN, the following lemma calculates the expected return of the optimal policy.

Lemma 4.13 (Expected return of the optimal policy)**.** *The optimal policy for this POMDP achieves an expected reward of* 1.

Proof. The optimal policy consists of choosing the correct control whenever state x_2 or x_3 is reached. Therefore, for each run of the environment, the agent incurs a reward of $+1$ at some step i. Hence, the expected return is

$$\mathbb{E}[r] = \sum_{i=0}^{\infty} \mathbb{P}[\text{ trajectory receiving reward in step } i+2](+1) = \sum_{i=0}^{\infty} q^i(1-q) = 1$$

$\square$

Lemma 4.14 (Optimality of the RNN)**.** *An one-layer RNN with a history of size* 2 *is able to model the optimal policy and thus achieve optimal expected return of* 1.

Proof. Consider the following RNN with history $h \in \{0,1\}^2$, observation y and control u:

$$\begin{aligned} h_{i+1} &\longleftarrow \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} h_i \\ y \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \text{with } h_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ u_i &\longleftarrow \phi\Big(\begin{pmatrix} 1 & 1 \end{pmatrix} h_i - 1 \Big) \end{aligned}$$

It is clear that the first coordinate of h_i summarizes the parity of the number of controls u_0 performed by the agent since of beginning of the run. Control u_1 is performed if and only if $y = 1$ and $(h)_1 = 1$ which suggests the agent is in state x_3. Therefore, this RNN is choosing the correct control when reaching states x_2 or x_3. So the RNN indeed models the optimal policy and will therefore achieve optimal expected return. □

The last part of this section consists in giving an upper bound of the expected return of a SBC.

Lemma 4.15. *The expected return achieved by a SBC with block-length* m *is at most*

$$1 - q^{m+1} \frac{2}{1+q}.$$

Proof. Notice that in order to achieve a positive return, any SBC should perform control u_0 when seeing observation y_0 and choose the correct control when reaching observation y_1.

To analyze the performance of the SBC, we distinguish two cases. We first consider the case where the observation y_1 is seen within the first m steps. Then, the SBC is able to keep track of the whole history since the beginning of the episode and is therefore able to infer the parity of the number of controls u_0 performed and therefore play the optimal control.

In the second case, notice that the history of the SBC will be $\Big((y_0)_{i=1}^m, (u_0)_{i=1}^m \Big)$. Given this history, the probability of observing y_1 when being in state x_2 vs x_3 is either

$$\sum_{i=0}^{\infty} q^{2i}(1-q) = \frac{1}{1+q}$$

or

$$\sum_{i=0}^{\infty} q^{2i+1}(1-q) = \frac{q}{1+q}$$

Which state is more probable depends on the parity of m, namely, if m is even, then x_2 is more probable than x_3. The control chosen by the SBC should be the one which optimizes the reward of the most probable state.

We can now upper bound the expected return of a SBC with

$$\begin{aligned} \sum_{i=0}^{m-1} q^i(1-q) + \sum_{i=m}^{\infty} (-1)^{i-m} q^i (1-q) &= 1 - q^m + q^m(1-q)\left(\sum_{i=0}^{\infty} q^{2i} - \sum_{i=0}^{\infty} q^{2i+1} \right) \\ &= 1 - q^m + q^m (1-q) \frac{1-q}{(1+q)(1-q)} \\ &= 1 - q^m \Big(1 - \frac{1-q}{1+q} \Big) \\ &= 1 - q^{m+1} \frac{2}{1+q} \end{aligned}$$

□

The following theorem combines the previous results and summarizes the main finding of this section.

Theorem 4.16 (Sub-optimality of SBCs compared to RNNs)**.** *For certain POMDPs, the expected return of SBCs is upper-bounded by the one of RNNs. SBCs is only able to achieve the same performance as RNNs when its block-length m tends to infinity.*

Proof. For the POMDP described in Definition 4.12, the expected return of the RNN is 1 (see Lemma 4.14) whereas the one for a SBC of block-length m is

$$1 - q^{m+1}\frac{2}{1+q}$$

(see Lemma 4.15).

It is clear that

$$1 - q^{m+1}\frac{2}{1+q} < 1$$

and

$$\lim_{m\to\infty} 1 - q^{m+1}\frac{2}{1+q} = 1$$

$\square$

5 Experiments

In this section we present our experimental results. We conducted experiments on different POMDPs to compare the performance of SBCs against RNNs.

The POMDP used are modifications of MDPs taken from the Gym [19] implementation. We chose to test the algorithms on three environments: CartPole, HalfCheetah and Ant. For each MDP environment chosen, we performed the following three types of modifications:

- Velocity removed: the state observations related to the velocity of the agent is removed.
- Flickering added: for each original observation, POMDP provides no state information with some probability p, meaning that the state observation returned is all zeros. This is equivalent to erasure channel case. We tried three different values for $p \in \{0.1, 0.2, 0.5\}$.
- Random noise added: some Gaussian noise with mean 0 and variance σ^2 is added to the state observations. For our experiments, $\sigma^2 \in \{0.1, 0.5\}$.

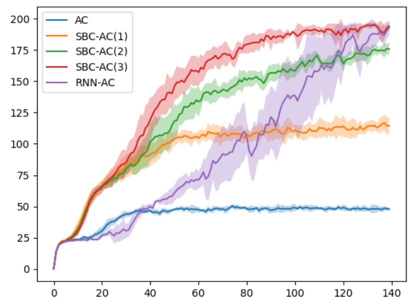
We used variants of the Actor-Critic (`AC`) algorithm for the CartPole environment where the action space is finite and variants of the Twin Delayed DDPG (`TD3`) algorithm for the other two environments with continuous state space. The variants differ in the input of the network as well as its structure. Even though the structure of models is mostly the same, linear layers were replaced by RNN layers in order to test for the performance of using RNNs for the internal state representation. For the FSCs, the input given for training as well as testing is a block of past observations and actions. More details about the experimental setup can be found in Appendix (A.1). We also provide an open-source implementation of our method available at `https://github.com/yooyoo9/RnnFscForLearningPomdps`.

The results in the next section show the return of different algorithms over a single episode in function of the training time. All results were averaged over five random executions.

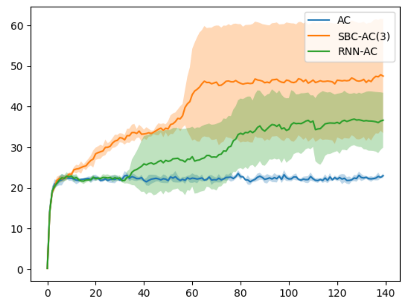
5.1 Actor-critic algorithms on modified Cart Pole environments

The CartPole environment is relatively simpler than the other two since its action space is finite. For this environment, we tried three different variants of the Actor-Critic (`AC`) algorithm. The two new variants are compared against the `AC` algorithm:

- `RNN-AC`: modified `AC` where one linear layer replaced with with one RNN layer. The input is the current observation and action.
- `SBC-AC`: original `AC` with past history of observations and actions given as input. The length of the past history is a hyper-parameter. The following history lengths where tried: $\{1, 2, 3\}$. The whole history of observations and actions are concatenated into a single vector.



(a) **Results on CartPole-vel.** `SBC-AC` with history length of 1, 2, and 3 is compared against `AC` and `RNN-AC`. As expected, `AC` is unable to achieve high return due to the lack of state information. `SBC-AC` with a history length of 3 is able to achieve same performance as `RNN-AC`. From the shape of the curve for `RNN-AC`, we see the training process of `RNN-AC` is much slower than `SBC-AC` as the latter is able to achieve high reward very early one.



(b) **Results for CartPole-fprob**0.5**.** This time, `SBC-AC` with history length 3 outperforms `RNN-AC`. One reason for this might be that `RNN-AC` needs more time to train in order to achieve good performance so it might outperform `SBC-AC` given enough training steps. It is also to notice that the return of all three algorithms is much less than in CartPole-vel.

Figure 5: **Performance comparison on different CartPole environments.** `AC`, `SBC-AC` with different history length and `RNN-AC` are compared for each environment. The plot shows a moving average of the return of a single episode against the training step. The return is averaged over five random executions.

Results show for this simple environment, `SBC-AC` with a block-length of 3 already performs as good as `RNN-AC` or even better. One can also notice that `RNN-AC` takes much longer time to train compared to `SBC-AC` which already achieves very high reward in the first few training steps.

5.2 TD3 algorithms on modified Half-Cheetah and Ant environments

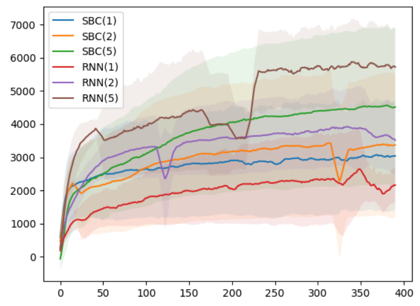
Since both HalfCheetah and Ant environments have continuous state and action spaces, we decided to implement variants of the Twin Delayed Deep Deterministic Policy Gradient (TD3) [20] algorithm for both environments.

The two variants are `RNN-TD3` and `SBC-TD3` and they both share a very similar structure: a history part with two layers which takes the past history as input, a feature part with two linear layers which takes the current observation and action as input and finally a combined part with two linear layers with the concatenation of the previous parts as input. The only difference between `RNN-TD3` and `SBC-TD3` lies in the history part and in the input. As of the structure of the network, one linear layer in `SBC-TD3` is replaced by a RNN layer in `RNN-TD3`. For the input, in the testing phase, `RNN-TD3` receives current and past observations whereas `RNN-TD3` only receives the current observation and hidden state.

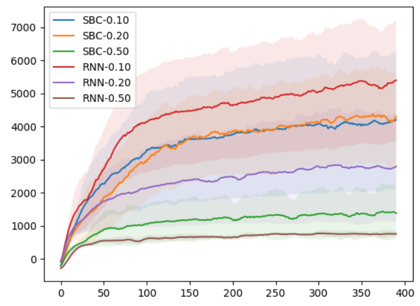
Results of the experiments are summarized in Figure 6. Even though `RNN-TD3` seems to outperform `SBC-TD3` despite its instability that causes the return to drop at certain times, its results on the Ant environment are not very satisfactory. `SBC-TD3` clearly outperforms `RNN-TD3` when Gaussian noise is incorporated into the environment. Furthermore, unlike stated in the theoretical results, `RNN-TD3` is not able to learn anything when the flickering or noise becomes too large. This is easily seen in both environments where a flickering of 0.5 or a noise with variance 0.5 is introduced. This might be due to the vanishing gradient problem of RNNs which constrains them to have a short-term memory.

6 Conclusion

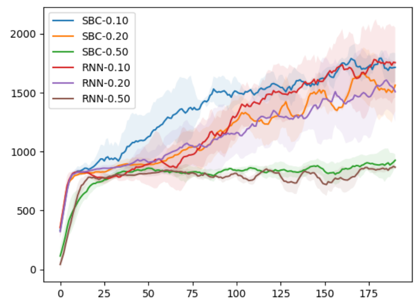
In this work, we compared the theoretical and empirical performance of using RNNs aginst FSCs. Even though theoretically RNNs proved to have a greater modeling power, FSCs seems to still have the advantage in practice. We make several hypothesis that explain this discrepancy.



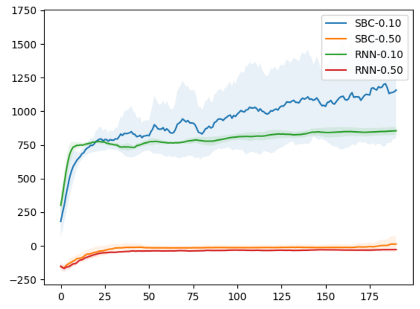
(a) **Comparison of `SBC-TD3` and `RNN-TD3` with different block-length on HalfCheetah-vel.** When comparing the two algorithms with the same bock length, `RNN-TD3` seems to have a slight advantage over `SBC-TD3`, except for the block-length of 1. However, one can see `RNN-TD3` is less stable in training compared to `SBC-TD3`.



(b) **Performance comparison for different HalfCheetah-fprob.** The experiments were carried out with $p \in \{0.1, 0.2, 0.5\}$. `RNN-TD3` only performs better than `SBC-TD3` when $p = 0.1$. This suggests `RNN-TD3` is unable to learn the optimal policy when the flickering probability becomes too large. The performance is however consistent as the larger the probability, the worse the performance of both algorithms.



(c) **Results on various Ant-fprob.** Again, the experiments were carried out with $p \in \{0.1, 0.2, 0.5\}$. The performance of `SBC-TD3` and `RNN-TD3` are very similar for all three values of p. Notice that similar to HalfCheetah-fprob with probability 0.5, both algorithm are unable to achieve a good return.



(d) **Results on Ant-rnoise.** Two values for the variance of the Gaussian noise were tried: 0.1 and 0.5. With $\sigma^2 = 0.5$, both algorithms do not learn anything useful. With $\sigma^2 = 0.1$, the performance of `RNN-TD3` seems to be better than the one of `SBC-TD3`. This is especially due to a large decrease in return of the `SBC-TD3` during training.

Figure 6: **Results for different HalfCheetah and Ant environments.** The history length used for training both models is listed in the legend for Figure (a) and equals 5 for all three other plots. All plots shows the test return over one episode in function of the training time.

- Training RNNs is notoriously harder than SBCs due to their inherent structure.
- Even though RNNs is theoretically able to keep track of all relevant information in the past history, RNNs are knwon to suffer from short-term memory due to a vanishing gradient problem that emerges when working with longer data sequences.
- The experiments were conducted on relatively simple environments which allow FSCs to easily perform well using only a small history block-length.

Several future research direction seem to extend from the current work. Having a better understanding of the learning of RNNs and the content of their hidden states might help to improve their performance in practice. One could also try determine a subclass of POMDPs that would benefit from RNNs instead of FSCs. Another possible way to go would be to define new models that outperform both, maybe by having a hybrid model using the RNN for the actor and a FSC for the critic. This requires having a deeper look into the role of both agents in addition to their requirements.

A Appendix

A.1 Experimental setup details

Implementation and Resources: All the experiments were conducted using PyTorch [21]. They were run on an Intel(R) Xeon(R) CPU E5-2697 v4 2.30GHz machine and an NVIDIA GeForce GTX 1080 Ti GPU.

Environments: Three different environments were taken from the Gym implementation [19]: CartPole [22], HalfCheetah [23] and Ant [24]. For each environment, the observations were modified in three different ways:

1. All state information related to velocity was removed.
2. With some probability $p_f \in \{0.1, 0.2, 0.5\}$, the observation given to the agent consisted of all zeros.
3. Random Gaussian noise was added to each component of the observation with mean 0 and variance $\sigma^2 \in \{0.1, 0.5\}$

Models: Variants of the `AC` algorithm were used for the CartPole environment. The structure of `AC` is as follows. Both the critic and the actor consists of two hidden layers of size $[128, 256]$ with ReLU activation. The difference between `AC` and `SBC-AC` only lies in the input to both networks. While `AC` only received the current observation, `SBC-AC` receives a block of past observations and actions as well as the current observation. Its input size is thus larger but the internal structure of both networks remain the same. In comparison, `RNN-AC` input is the same as `AC` but the two hidden layers are replaced by two RNN layers of the same size.

For the HalfCheetah and Ant environments, we used variants of the `TD3` algorithm. The critic and actor of `SBC-TD3` both consist of three parts: the history part, the current feature part and the combined part. The history part takes the past observations and actions as input to a two layer network of size 128 with ReLU activation. The current feature part received the current observation as input to a two layer network of size 128 with ReLU activation. The two outputs is concatenated and fed into the combined part which consists of one linear layer of size 128. The difference between the critic and actor is that the output of the critic is the output of the linear layer of the combined part whereas the actor has a Tanh activation on top of it in order to constrain the action to be between -1 and 1.

The structure of `RNN-TD3` is the same as for `SBC-TD3` apart from the fact that one linear layer in the history part is replaced by a RNN layer of the same size. The input is also different since `RNN-TD3` only received the current observation as input when testing whereas `SBC-TD3` receives both current and past observations and actions.

Hyper-Parameters:

1. ADAM [25]
2. Discount factor γ: 0.99
3. Training epochs: 15000 for CartPole, 400 for HalfCheetah and 200 for Ant
4. Learning rates: 0.00005 or 0.001 or
5. History lengths of input: 1, 2, 3, 5
6. Replay buffer size: 1000000
7. Interpolation factor in polyak averaging for target networks: 0.995
8. Batch size:
9. Number of steps for uniform-random action selection before acting according to policy: 10000
10. Number of environment interactions before starting to do gradient descent updates: 1000
11. Number of environment interactions between two consecutive gradient updates: 50
12. Standard deviation for Gaussian exploration noise added to the policy when training: 0.1
13. Standard deviation for smoothing noise added to the target policy: 0.2
14. Noise clip for target policy smoothing noise: 0.5

15. Number of delay steps for policy update: 2
16. Maximum number of test episodes after each epoch: 10
17. Maximum length of a single episode: 1000
18. Random seeds: 1003, 727, 527, 1225, 714

Evaluation: We evaluated the algorithms by analyzing the return over different episodes and the rewards within a single episode.

References

[1] Yonathan Efroni, Chi Jin, Akshay Krishnamurthy, and Sobhan Miryoosefi. Provable reinforcement learning with a short-term memory. *arXiv preprint arXiv:2202.03983*, 2022.

[2] Kamyar Azizzadenesheli, Alessandro Lazaric, and Animashree Anandkumar. Reinforcement learning of pomdps using spectral methods. In *Conference on Learning Theory*, pages 193–256. PMLR, 2016.

[3] Chi Jin, Sham Kakade, Akshay Krishnamurthy, and Qinghua Liu. Sample-efficient reinforcement learning of undercomplete pomdps. *Advances in Neural Information Processing Systems*, 33:18530–18539, 2020.

[4] Yi Xiong, Ningyuan Chen, Xuefeng Gao, and Xiang Zhou. Sublinear regret for learning pomdps. *arXiv preprint arXiv:2107.03635*, 2021.

[5] Christos H Papadimitriou and John N Tsitsiklis. The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

[6] Simon Du, Akshay Krishnamurthy, Nan Jiang, Alekh Agarwal, Miroslav Dudik, and John Langford. Provably efficient rl with rich observations via latent state decoding. In *International Conference on Machine Learning*, pages 1665–1674. PMLR, 2019.

[7] Dipendra Misra, Mikael Henaff, Akshay Krishnamurthy, and John Langford. Kinematic state abstraction and provably efficient rich-observation reinforcement learning. In *International conference on machine learning*, pages 6961–6971. PMLR, 2020.

[8] Valentin Khrulkov, Alexander Novikov, and Ivan Oseledets. Expressive power of recurrent neural networks. *arXiv preprint arXiv:1711.00811*, 2017.

[9] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. On the convergence rate of training recurrent neural networks. *Advances in neural information processing systems*, 32, 2019.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[11] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*, 2015.

[12] Daan Wierstra, Alexander Foerster, Jan Peters, and Juergen Schmidhuber. Solving deep memory pomdps with recurrent policy gradients. In *International conference on artificial neural networks*, pages 697–706. Springer, 2007.

[13] Bram Bakker. Reinforcement learning with long short-term memory. *Advances in neural information processing systems*, 14, 2001.

[14] Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.

[15] Lingheng Meng, Rob Gorbet, and Dana Kulić. Memory-based deep reinforcement learning for pomdps. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5619–5626. IEEE, 2021.

[16] Panqanamala Ramana Kumar and Pravin Varaiya. *Stochastic systems: Estimation, identification, and adaptive control*. SIAM, 2015.

[17] Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*, volume 1. Athena scientific, 2012.

[18] Vikram Krishnamurthy. *Partially observed Markov decision processes*. Cambridge university press, 2016.

[19] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[20] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[21] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Luca Antiga Alban Desmaison, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS 2017 Workshop Autodiff Submission*, 2017.

[22] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

[23] Paweł Wawrzyński. A cat-like robot real-time learning to run. In *International Conference on Adaptive and Natural Computing Algorithms*, pages 380–390. Springer, 2009.

[24] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.