Modeling, Identification and Control, Vol. 43, No. 1, 2022, pp. 1–8, ISSN 1890–1328

Model-free Control of Partially Observable Underactuated Systems by pairing Reinforcement Learning with Delay Embeddings

M. Knudsen [1] S. Hendseth [1] G. Tufte [2] A. Sandvig [3]

[1] *Department of Engineering Cybernetics, Norwegian University of Science and Technology, N-7491 Trondheim, Norway. E-mail: {Martinius.Knudsen,Sverre.Hendseth}@ntnu.no*

[2] *Department of Computer Science, Norwegian University of Science and Technology, N-7491 Trondheim, Norway. E-mail: {Gunnar.Tufte}@ntnu.no*

[3] *Department of Neuromedicine and Movement Science, Norwegian University of Science and Technology, N-7491 Trondheim, Norway. E-mail: {Axel.Sandvig}@ntnu.no*

Abstract

Partial observability is a problem in control design where the measured states are insufficient in describing the systems trajectory. Interesting real-world systems often exhibit nonlinear behavior and noisy, continuous-valued states that are poorly described by first principles, and which are only partially observable. If partial observability can be overcome, these conditions suggest the use of reinforcement learning (RL). In this paper we tackle the problem of controlling highly nonlinear underactuated dynamical systems, without a model, and with insufficient observations to infer the systems internal states. We approach the problem by creating a time-delay embedding from a subset of the observed state and apply RL on this embedding rather than the original state manifold. We find that delay embeddings work well with learning based methods, as such methods do not require a precise description of the systems state. Instead, RL learns to map any observation to appropriate action (determined by a reward function), even if these observations do not lie on the original geometric state manifold.

Keywords: Control, Delay embeddings, Reinforcement learning, Partial observability, Underactuated systems

1 Introduction

In order to efficiently design feedback controllers for dynamical systems, a key prerequisite is the ability to measure or infer the systems internal state. When the measurements are insufficient to describe such states, we have what is known as a partially observable system Åström (1965); Kaelbling et al. (1998). Such systems are problematic in terms of control design but not uncommon. Any system with an insufficient number of sensors or inadequate sensor placements may render partially observable. One means to control such a system is by constructing a state estimator/observer Åström (1965); Kaelbling et al. (1998); Khalil (2002) that derives the hidden system states, if possible, as to acquire sufficient measurements for the design of a controller. Two requirements must be fulfilled for such an approach; (1) that the system is *observable*, i.e. the measurements are sufficient to fully reconstruct the system state and (2) that the system can be modelled by first principles. If a system model however cannot be developed, one must approach the problem differ-

doi:10.4173/mic.2022.1.1

ently. Reinforcement learning (RL) is a great candidate for this. By applying RL directly on the measurements (or observations as they are commonly called in RL literature), these algorithms learn to associate states with value. Whether these states lie on the original manifold or a one-to-one embedding need not matter. Such an approach is not possible using traditional first-principles control methods.

Even RL however, requires a sufficient observation vector in order to learn a task. When dealing with systems with unknown models, it is not trivial to know how many sensors are required, nor where to place them in order to provide enough coverage for a learning-based agent to acquire a good control policy. The notion of insufficient observations may be associated with a partially observable Markov decision process (POMDP), which in RL literature is often described in the context of multi-agent games. In the POMDP case we also do not have full state information and apply probability of expectations to derive our policy. On the other hand, the modeling of a deterministic dynamical system relies on the concept of a phase space, the collection of possible system states. The system state at time t consists of all information needed to uniquely determine the future system states for times $\geq$ t; e.g., in many cases, positions and velocities. For a system that can be modeled mathematically, the phase space is known from the equations of motion. For experimentally observed dynamical systems, the phase space and a mathematical description of the system are often unknown. Whether one whishes to apply first-principles based control or learning-based control, observability generally is a requirement. An example would be to observe the position of a cart but not the velocity. Simply knowing the position does not allow us to predict the next state and thereby learn a control policy.

Connecting the dots between dynamical systems, Takens Embedding Theorem (introduced in 1.1) and RL, we were interested in seeing if time-delay embeddings (TDEs) could be used to learn control policies for partially observable feedback (POF) systems. We present here a method for overcoming partial observability in unknown systems using Takens Embedding Theorem to reconstruct the systems manifold. We experiment with learning on TDEs of nonlinear dynamical systems such as the underactuated classical control problems Cartpole, DoublePendulum and Acrobot. The goal of this paper is to assess the application of TDEs on POF systems rather than developing a new RL algorithm, therefor we perform our investigations using established well tested RL methods. Through our investigations we find for what control tasks delay embeddings are appropriate, and for which they are less successful.

1.1 Delay embeddings

In the study of dynamical systems, a delay embedding theorem gives the conditions under which a chaotic dynamical system can be reconstructed from a sequence of observations of the systems states. Sauer (2006) I.e. a delay embedding has the ability to reconstruct the phase space using only a subspace of the actual systems states. The reconstruction preserves the properties of the dynamical system that do not change under smooth coordinate changes, but does not preserve the geometric shape of structures in phase space. The time series of these observation subspaces are thereby used to build a proxy of the full system manifold.

Takens Embedding Theorem is the 1981 delay embedding theorem of Floris Takens Takens (1981). It shows that a single time-delayed measured quantity $[y(t), y(t-\tau), y(t-2\tau), ..., y(t-n\tau)]$ is sufficient to embed an n-dimensional manifold. The theorem particularly provides the conditions under which a smooth attractor can be reconstructed from time-delay coordinates. An example of such an embedding is shown in Figure 1 as presented by Sugihara et al. Sugihara et al. (2012). Here, the Lorenz attractor Lorenz (1976) is reconstructed from time-delay embeddings of each of the three system coordinates. Similar to the chaotic Lorenz system, two of the systems we present in section 2.1.1 also display chaotic behavior when passive.

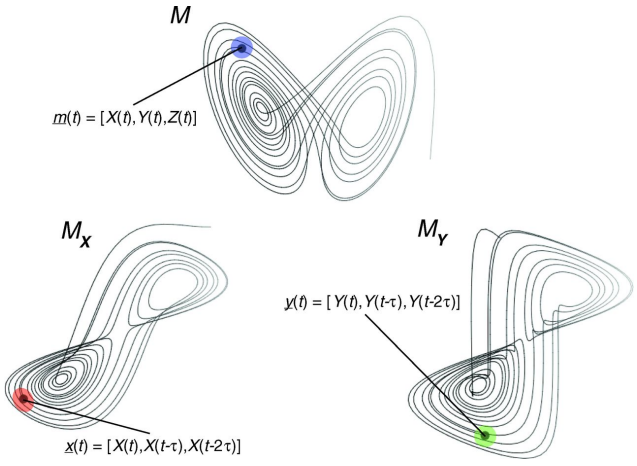


Figure 1: Reconstruction of 3D Lorenz attractor Lorenz (1976) as presented in Sugihara et al. (2012). On top we see the original manifold, while below we see time-delay embedding for the x and y coordinates. These delay embeddings are one-to-one with the original manifold.

1.2 A possible connection between delay-embeddings and neural recurrency

Partial observability is very normal in our complex and dynamic world. Yet we, as biological agents, have a great ability to forecast, manipulate and control such dynamical systems all around us. Given the highly recurrent nature of our brains wiring, we hypothesize that part of the reasoning for this recurrency is to enable the ability to create delay-embeddings in which we can learn on rather than having to rely on full state feedback from all the systems we encounter. After all, internal states are rarely observable to us, let alone the full state. Take for example the systems presented in this paper; their full state includes velocity states. Velocity can only be inferred in the presence of time, and the brain does not receive velocity values directly when observing a system, only the systems geometric coordinates. Although this is somewhat speculative, positive findings in this area make way to support this hypothesis.

2 Method

The overall setting here is that we are to train an artificial neural network (ANN) model (the agent) to solve 3 different tasks in 3 different environments. This is to be accomplished with the agent only being given POF from the environment. The agent constructs a TDE using a built-in memory which it will try to learn the task on rather than on the environments original 4-state manifold.

2.1 Environments and tasks

2.1.1 Environments

We have tested the performance of using a TDE on 3 systems: CartPole, DoublePendulum and Acrobot. These pendulum systems were chosen due to their extensive history as benchmark problems in underactuated control Tedrake (2020). While generally regarded as solved using traditional first principles methods, these problems are still very challenging for methods that rely on learning. Particularly, when the task comes down to stabilization such as in balancing an upright pendulum.

The CartPole and Acrobot systems are implementations within the OpenAI Gym Brockman et al. (2016) framework; a framework specifically tailored for RL benchmarking. The DoublePendulum environment is a custom OpenAI Gym compatible environment we have created. Within these environments we have trained the RL algorithm on both standard OpenAI Gym tasks as well as our own custom tasks. The tasks are described in section 2.1.2. The environments are described here. The terms pole, pendulum and link are used interchangeably:

The **CartPole** system (Figure 2), often referred to as the cart-pendulum system, is comprised of a single link mounted on a cart by a freely rotating joint. Only the cart is actuated, thus the link angle can only be manipulated by moving the cart left or right.

The **DoublePendulum** system (Figure 3) is comprised of two joints and two links, where the one joint at the end is actuated while the middle joint is not. The 2-link pendulum is an especially interesting system in the context of Takens embedding theorem, as the system without any actuation is known to behave chaotically Shinbrot et al. (1992) like the Lorenz attractor described earlier.

The **Acrobot** system (Figure 3) is very similar to the DoublePendulum as it is also comprised of two links and two joints. However, in this system, actuation is now only applied to the *center* joint.

2.1.2 Tasks

We attempt to solve 3 different tasks in all three environments: swing-up, balance and swing-up + balance.

Task 1: Swing-up: starting with the link(s) in a hanging down state, we attempt to swing the link(s) into the upright position. When there are two links, the goal state is achieved when the total height of the chained links is close to their maximum upright height. The episode is terminated upon reaching the goal height or if the episodes maximum length has been reached.

Task 2: Balance: starting in a random almost fully upright state, the goal is to maintain this upright position by balancing the link(s) for as long as possible. The task is terminated when reaching a given maximum episode length or when the total height of the link(s) falls below a given height close to maximum.

Task 3: Swing-up and balance: if the previous two tasks are successful, we in this task attempt to combine the two. Even if the preceding tasks were successful, this is a significantly more challenging task as the agent must now swing up the link(s) with just the right momentum as to not overshoot when reaching the top. The episode is terminated after a set episode length.

The environment-task configurations such as the reward functions and maximum episode lengths are shown in Table 2.

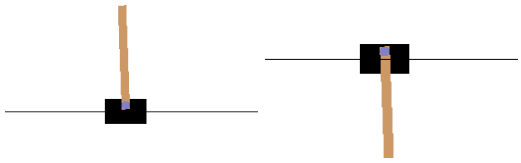


Figure 2: CartPole: A single link attached to an unactuated rotary joint on a cart. The left figure is a screenshot of the pendulum being balanced, while the in the right the pendulum is hanging downwards and being prepared to be swung up.

Figure 3: DoublePendulum and Acrobot: 2 link underactuated pendulum systems fixed to a freely rotating point at the end joint and actuated at either the end joint or the middle joint. The figure is a screenshot of the system in its upright state.

2.2 Implementing a time-delay embedding

Each of the three environments we are studying have four internal states as shown in Table 1. The observations given to the agent are however, not necessarily these internal states. Table 1 further shows the observations given to the agent. Our goal is to solve the previously described environments-tasks using only POF, which by itself is not sufficient feedback information for the agent to solve the task. As such, the agent is in a situation where it only has access to a subset of the full original observation space. In order to create the embedding we need N time-delayed coordinates, where N is preferably equal to the number of dimensions of the full state manifold. Determining the dimensionality of an unknown system can be a challenge in identification and nonlinear dynamic analysis, and is outside the scope of this paper. Section 2.3 in Bush and Pineau (2009) discusses some approaches to this problem. As we know the dimensionality of our environments *a prioi*, we construct a delay embedding vector of the same dimension.

In the case of the CartPole, we observe only the cart position and link angle, while in the DoublePendulum and Acrobot systems, we only observe the angle of the first link. We require two observations for the CartPole system, as we found that the cart position cannot be inferred by a delay embedded angle and vice-versa. For the two latter systems, this however is possible for the angles of the second link. There is no particular reason that we have chosen the angle of the first link rather than the second as our observation. According to Takens theorem, any of the time-delayed states should be able to recreate a one-to-one embedding of the original manifold. Takens (1981)

The TDE was implemented using an agent memory buffer. For each simulation step, we would push the current observed state into the agents memory. When it was time for the agent to receive the time-delay embedded observation, we simply retrieved the last 4-6 (depending on the environment) observations and passed to the agent. The agent then learns on this delay embedding rather than the original state manifold. The implementation of the memory buffer in Python is shown in Listing 1.

```
class Memory:
  def __init__(self, stateSize, lag=1):
    self.nstates = stateSize
    self.size = stateSize*lag
    self.lag = lag
    self.memory = np.zeros(self.size)
    self.out = np.zeros(stateSize)

  def reset(self,obs):
    for i in range(self.size):
      self.memory[i] = obs[2]

  def push(self,obs):
    self.memory[1:] = self.memory[:-1]
    self.memory[0] = obs

  def get(self):
    for i in range(0,len(self.out)):
      self.out[i] = self.memory[self.lag*i]
    return self.out
```

Listing 1: Python code for the agents memory buffer

2.3 RL algorithm: Stable-Baseline3 PPO

Stable Baselines3 (SB3) is a set of reliable implementations of reinforcement learning algorithms in PyTorch Raffin et al. (2019), created specifically to be run in OpenAI Gym environments. These RL baselines have been externally tested and their performance verified. SB3 provides several implementations of well known RL algorithms such as A2C, DDPG, DQN, HER, PPO, SAC and TD3. On their *RL Algorithms* page Stable baselines3 (2017) one can find a comparison of these algorithms for use with OpenAI Gym. Among these algorithms, A2C, DQN and PPO are compatible with our environments actions and observations space configurations (Box and Discrete respectively). We tested the performance of these three meth-

Environment	Original	Partial time-delayed
States		
CartPole	$[x, \dot{x}, \theta, \dot{\theta}]$	$[x_t, \theta_t, \theta_{t+1}, \theta_{t+2}]$
DoublePendulum	$[\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]$	$[\theta_{1_t}, \theta_{1_{t+1}}, \theta_{1_{t+2}}, \theta_{1_{t+3}}]$
Acrobot	$[\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]$	$[\theta_{1_t}, \theta_{1_{t+1}}, \theta_{1_{t+2}}, \theta_{1_{t+3}}]$
Observations		
CartPole	$[x, \dot{x}, \theta, \dot{\theta}]$	$[x_t, \theta_t, \theta_{t+1}, \theta_{t+2}]$
DoublePendulum	$[cos(\theta_1), sin(\theta_1), cos(\theta_2), sin(\theta_2), \dot{\theta}_1, \dot{\theta}_2]$	$[cos(\theta_{1_t}), sin(\theta_{1_{t+1}}), cos(\theta_{1_{t+2}}), sin(\theta_{1_{t+3}}), \theta_{1_{t+4}}, \theta_{1_{t+5}}]$
Acrobot	$[cos(\theta_1), sin(\theta_1), cos(\theta_2), sin(\theta_2), \dot{\theta}_1, \dot{\theta}_2]$	$[cos(\theta_{1_t}), sin(\theta_{1_{t+1}}), cos(\theta_{1_{t+2}}), sin(\theta_{1_{t+3}}), \theta_{1_{t+4}}, \theta_{1_{t+5}}]$

Table 1: Original states and observations are as defined in OpenAI Gym. $cos(\theta)$ and $sin(\theta)$ put bounds on the state-space which is necessary for RL. The θ_1 internal state is pushed to the embedding memory, and we create an embedding of a similar form as the vector of the original observation.

ods using full observation feedback (FOF) and found PPO to perform the best.

PPO, which stands for Proximal Policy Optimization Schulman et al. (2017), combines ideas from Asynchronous Actor Critic (A2C) Mnih et al. (2016) by having multiple workers, and Trust Region Policy Optimization (TRPO) Schulman et al. (2015) which uses a trust region to improve the actors network weights. The main idea is that after an update, the new policy should not be too far from the old policy Raffin et al. (2019). PPO was developed by OpenAI and quickly became their default RL algorithm in 2017 due to its ease of use and good performance. As such, we were confident that it would suffice to be used in our experiments.

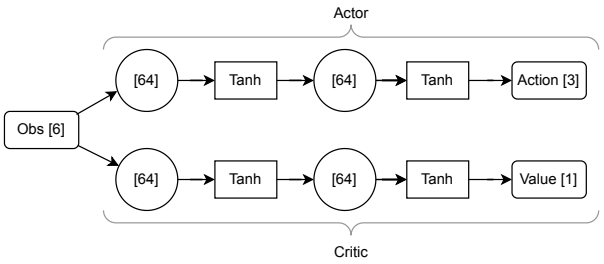


Figure 4: ANN graph of the Actor-Critic PPO network. This specific example shows the network for the DoublePendulum and Acrobot environments, with 6 input observations, 3 discrete output actions and 1 value output.

2.4 ANN model

As mentioned above, PPO borrows implementation from A2C which is configured in an Actor-Critic setup with separate action and value networks. The actor maps observations to actions, while the critic evaluates the value of the observation given the current policy. In our experiments, both the actor and the critic have 2 hidden layers of [64,64] neurons as can be seen in Figure 4. Larger network architectures were also tested such as [64,128,64] and [64,128,64,32,16] without any significant difference in performance. [64,64] is the default architecture of SB3's PPO implementation. Similarly, tanh activation functions are also part of the default configuration.

3 Results

Table 2 shows the setup of each experiment-task. The 3 columns to the left show the results of the training. The *Steps* column is the number of training steps until the task was solved. The columns FOF and POF are the *full observation feedback* and *partial observation feedback* experiment results respectively. Green indicates that the agent solved the task, yellow that it got close to solve the task and red indicates that the agent was unable to solve the task. Grey tasks were not attempted as their simpler subtasks (balance alone) were not successful by POF. In addition to these configurations we also attempted training with different interval time delays for the POF case. The solver used is Runga-Kutta RK4 Dormand and Prince (1980). Figure 3 shows the plots from the training progress of each experiment-task. Blue lines are using FOF and orange using POF. We used a delay of 1 simulation time step for all experiment-tasks, as we found that shorter delays resulted in better performance. Likely, this was due to the controller being able to respond to system changes faster. A simulation step corresponds to 0.02s for CartPole and 0.2s for the DoublePendulum and Acrobot environments. A description of each graph is included in the figure. The graphs were plotted using Tensorboard Abadi et al. (2015).

From the results we can see that the RL agent quickly learns to solve tasks that involve movement such as swing-up. Stabilizing/balancing the link(s) upright is more challenging, however not impossible.

Environment	Task	Reward per step	Done condition	Ep. len	Steps	FOF	POF
CartPole	swing-up	• beyond x threshold: -3000 • episode end: 0 • $cos(\theta) > 0.85$: 2 • else: -1	• $\cos(\theta) > 0.85$ • beyond x threshold	1000	150000		
CartPole	balance	• each step: 1	• $\theta < 0.21$ • beyond x threshold	500	150000		
CartPole	swing-up balance	• same as swing-up	• beyond x threshold	1000	750000		
DoublePendulum	swing-up	• each step: -1	• $-\cos(\theta_1) - \cos(\theta_1 + \theta_2) > 1$	1000	40000		
DoublePendulum	balance	• each step: 1	• $-\cos(\theta_1) - \cos(\theta_1 + \theta_2) < 1.5$	400	300000		
DoublePendulum	swing-up balance	-	-	-	-		
Acrobot	swing-up	• above given height: 2 • height > 1.8: 1 • height > 1.5: 0 • else: -1	• $-\cos(\theta_1) - \cos(\theta_1 + \theta_2) > 1.9$	500	150000		
Acrobot	balance	• each step: 1	• $-\cos(\theta_1) - \cos(\theta_1 + \theta_2) < 1.8$	500	750000		
Acrobot	swing-up balance	-	-	-	-		

Table 2: Showing the configuration of each experiment-task and the results from training in the last 3 columns. *Steps* is the training steps until the task was solved. FOF is the full feedback run, while POF is the partial feedback run using the time-delay embedding. Green indicates that a task was solved, yellow that the task was almost solved and red that the agent was unable to solve the task. Experiments marked grey where not performed as the simpler task was not successful.

The agent actually learned the swing-up tasks almost equally fast on the embedding as it did on the original manifold. While we were hoping that the agent would also learn the swing-up + balance tasks of the DoublePendulum and Acrobot environments, the PPO algorithm simply converged to a low score and did not learn the task.

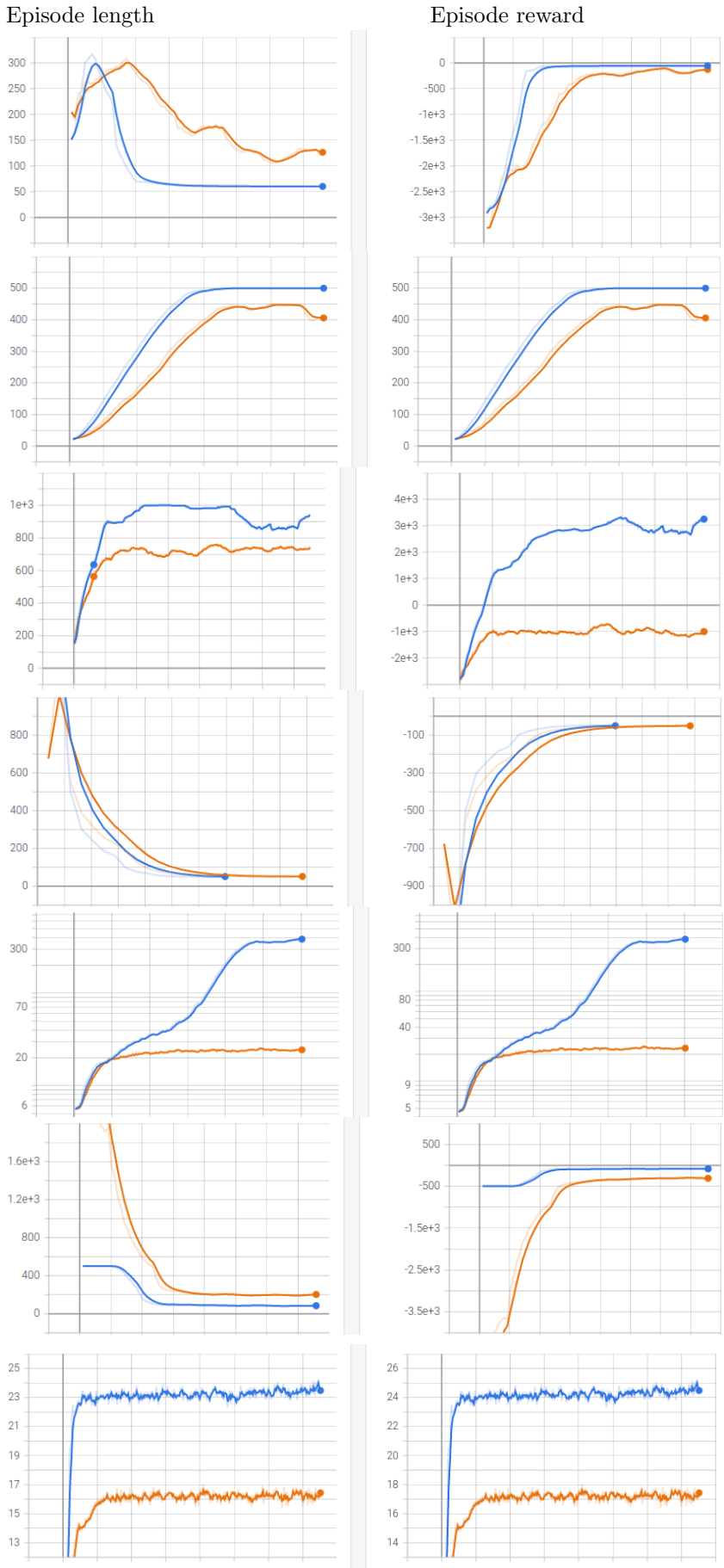
4 Conclusion

We have in this paper presented the results of applying RL to time-delayed embeddings as a method to solve the problem of partial observability of unknown environments. The results show that the RL agent quickly learns to solve tasks that involve movement, such as swing-up, while steady state tasks such as stabilizing/balancing are more challenging, but still achievable. The agent actually learned the swing-up tasks almost equally fast on the embedding as it did on the original manifold, and we conclude that tasks that are efficiently solved on the original manifold are also efficiently solved on a time-delay embedding. Interestingly, we also found that shorter delay intervals yielded better performance. All in all, we have found the application of applying RL on delay embeddings to be a promising method to overcome partial observability.

We also hypothesized that the brains widely recurrent neural wiring may be utilizing delay embeddings as a means to operate in a highly complex, dynamic and partially observable world. While this hypothesis requires much more rigorous investigation going forward, we find the results presented here to be supportive to this claim by showing how partial observability indeed can be addressed by the use of delay embeddings.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke,



(a) CartPole - swing-up: This problem was efficiently solved with both methods of feedback. Since the episode terminates when completing the task, lower episode lengths are better. Training took no more than 5 minutes in both cases.

(b) CartPole - balance: This problem was efficiently solved with both methods of feedback. Since a reward of 1 is given for each step the agent stays alive (i.e. balances) episode length and reward are equivalent.

(c) CartPole - swing-up and balance: Solved using FOF, but PPO struggled using POF.

(d) DoublePendulum - swing-up: This problem was efficiently solved with both methods of feedback. Since the episode terminates when completing the task, lower episode lengths are better.

(d) DoublePendulum - balance: Solved using FOF, but PPO struggled using POF. We found that using θ_2 as the time lagged coordinate was better then using θ_1.

(d) Acrobot - swing-up: This problem was efficiently solved with both methods of feedback.

(e) Acrobot - balance: This was a challenging task for both feedback methods. Using FOF, the agent was able to balance for some time before falling down, while POF struggles to balance more than 16 steps. We attempted training for $>$ 3 million steps but the training reward simply flattened out without solving the task.

Table 3: Results with configurations (among them the x-axis time steps) according to Table 2. Full observation feedback is colored blue, while partial observation feedback is orange.

M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015.

Åström, K. J. Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 1965. 10(1):174–205. doi:10.1016/0022-247x(69)90163-2.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. OpenAI Gym. *arXiv:1606.01540 [cs]*, 2016.

Bush, K. and Pineau, J. Manifold Embeddings for Model-Based Reinforcement Learning under Partial Observability. *Advances in Neural Information Processing Systems*, 2009. 22.

Dormand, J. R. and Prince, P. J. A family of embedded Runge-Kutta formulae. *Journal of computational and applied mathematics*, 1980. 6(1):19–26. doi:10.1016/0771-050x(80)90013-3.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 1998. 101(1-2):99–134. doi:10.1016/s0004-3702(98)00023-x.

Khalil, H. *Nonlinear Systems*. Cambridge University Press, 2002. doi:10.1017/CBO9781139172455.

Lorenz, E. N. Nondeterministic theories of climatic change. *Quaternary Research*, 1976. 6(4):495–506. doi:10.1016/0033-5894(76)90022-3.

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*. PMLR, pages 1928–1937, 2016.

Raffin, A., Hill, A., Ernestus, M., Gleave, A., Kanervisto, A., and Dormann, N. Stable baselines3. GitHub, 2019.

Sauer, T. D. Attractor reconstruction. *Scholarpedia*, 2006. 1(10):1727. doi:10.4249/scholarpedia.1727.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International Conference on Machine Learning*. PMLR, pages 1889–1897, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]*, 2017.

Shinbrot, T., Grebogi, C., Wisdom, J., and Yorke, J. A. Chaos in a double pendulum. *American Journal of Physics*, 1992. 60(6):491–499. doi:10.1119/1.16860.

Stable baselines3. Proximal Polociy Optimization (PPO) algorithm. https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html, 2017.

Sugihara, G., May, R., Ye, H., Hsieh, C.-h., Deyle, E., Fogarty, M., and Munch, S. Detecting Causality in Complex Ecosystems. *Science*, 2012. 338(6106):496–500. doi:10.1126/science.1227079.

Takens, F. Detecting strange attractors in turbulence. In D. Rand and L.-S. Young, editors, *Dynamical Systems and Turbulence, Warwick 1980*, Lecture Notes in Mathematics. Springer, Berlin, Heidelberg, pages 366–381, 1981. doi:10.1007/BFb0091924.

Tedrake, R. Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation (Course Notes for MIT 6.832). http://underactuated.mit.edu/, 2020.