Learning Deep Neural Network Policies with Continuous Memory States

Marvin Zhang, Zoe McCarthy, Chelsea Finn, Sergey Levine, Pieter Abbeel
Department of Electrical Engineering and Computer Science, UC Berkeley
{zhangmarvin,sergey.levine,zmccarthy,cbfinn,pabbeel}@berkeley.edu

Abstract

Policy learning for partially observed control tasks requires policies that can remember salient information from past observations. In this paper, we present a method for learning policies with internal memory for high-dimensional, continuous systems, such as robotic manipulators. Our approach consists of augmenting the state and action space of the system with continuous-valued memory states that the policy can read from and write to. Learning general-purpose policies with this type of memory representation directly is difficult, because the policy must automatically figure out the most salient information to memorize at each time step. We show that, by decomposing this policy search problem into a trajectory optimization phase and a supervised learning phase through a method called guided policy search, we can acquire policies with effective memorization and recall strategies.

1 Introduction

In this paper, we investigate a simple approach for endowing policies with memory, by augmenting the state space to include memory states that can be written to by the policy. Naïvely using such a state representation with standard policy search algorithms is quite challenging, because the algorithm must simultaneously figure out how to use the memory states to choose the action and how to store the right information into these states so that it can be recalled later. The computations needed to make such decisions about memory states require a complex, nonlinear policy structure. Such policies are difficult to efficiently learn with model-free methods, while model-based methods also require a model of the system dynamics, which can be difficult to obtain [1]. We show how the guided policy search algorithm can be adapted to the task of training policies with internal memory.

In guided policy search, the policy is optimized using supervised learning [2, 3]. The supervision is provided by using a simple trajectory-centric reinforcement learning algorithm to individually solve the task from a collection of fixed initial states. Since each trajectory-centric "teacher" only needs to solve the task from a single initial state, it is faced with a much easier problem. The final policy is trained with supervised learning, which allows us to use a nonlinear, high-dimensional representation for this final policy, such as a multilayer neural network, in order to learn complex behaviors with good generalization.

To incorporate memory states into this method, we add the memory states to both the trajectory-centric teacher and the final neural network policy. Since the trajectories are adapted to the neural network policy, the teacher selects memory states that will cause the neural network to take the right action, essentially telling the network which information should be memorized to achieve good performance. Because of this, the neural network only needs to learn how to reproduce the memory states chosen by the teacher. The teacher effectively shows the neural network which information needs to be written into the memory, and the network need only figure out how to obtain this information from the observations.

2 Memory States and Guided Policy Search

The aim of our method is to control a partially observed system in order to minimize the expectation of a cost function over the entire execution of a policy $\pi_\theta(\mathbf{u}_t|\mathbf{o}_1, \ldots, \mathbf{o}_t)$, given by $E_{\pi_\theta}[\sum_{t=1}^{T} \ell(\mathbf{x}_t, \mathbf{u}_t)]$ in the finite-horizon episodic setting. Here, $\mathbf{x}_t$ denotes the true state of the system, $\mathbf{u}_t$ denotes the action, $\mathbf{o}_t$ denotes the observation, and $\ell(\mathbf{x}_t, \mathbf{u}_t)$ is the cost function that specifies the task.

In our approach, the memory states $\mathbf{h}_t$ are directly concatenated to the physical state of the system $\mathbf{x}_t$ and the observation $\mathbf{o}_t$, to produce an augmented state $\tilde{\mathbf{x}}_t$ and augmented observation $\tilde{\mathbf{o}}_t$, and the action is also augmented to include memory writing actions $\mathbf{a}_t$ to produce an augmented action $\tilde{\mathbf{u}}_t$:

$$\tilde{\mathbf{x}}_t = \begin{bmatrix} \mathbf{x}_t \\ \mathbf{h}_t \end{bmatrix} \qquad \tilde{\mathbf{o}}_t = \begin{bmatrix} \mathbf{o}_t \\ \mathbf{h}_t \end{bmatrix} \qquad \tilde{\mathbf{u}}_t = \begin{bmatrix} \mathbf{u}_t \\ \mathbf{a}_t \end{bmatrix}.$$

The dynamics then factorize according to

$$p(\tilde{\mathbf{x}}_{t+1}|\tilde{\mathbf{x}}_t, \tilde{\mathbf{u}}_t) = p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)p(\mathbf{h}_{t+1}|\mathbf{h}_t, \mathbf{a}_t),$$

where there are various ways to choose $p(\mathbf{h}_{t+1}|\mathbf{h}_t, \mathbf{a}_t)$ depending on the semantics of $\mathbf{a}_t$. In this work, we define $p(\mathbf{h}_{t+1}|\mathbf{h}_t, \mathbf{a}_t) = \mathcal{N}(\mathbf{h}_t + \mathbf{a}_t, \sigma^2\mathbf{I})$, where σ^2 is chosen to be a small constant (10^{-6} in our implementation) to ensure that the trajectory distributions remain well-conditioned.

Training a policy $\pi_\theta(\tilde{\mathbf{u}}_t|\tilde{\mathbf{o}}_t)$ on this augmented dynamical system produces a policy that, in principle, can use the memory actions $\mathbf{a}_t$ to write to the memory states $\mathbf{h}_t$. In fact, prior work has proposed using discrete memory with storage actions [4]. In practice, the particular choice of policy optimization algorithm makes a significant difference in how well the policy can utilize the memory states, since there is no guidance on how they should be used, aside from overall task performance. For high-dimensional, continuous tasks, the logic required to choose which information to store and recall and when can become quite complex. This necessitates the use of powerful, expressive function approximators with hundreds or even thousands of parameters to represent $\pi_\theta(\tilde{\mathbf{u}}_t|\tilde{\mathbf{x}}_t)$, which are generally very difficult to train with standard policy search techniques [1]. However, guided policy search has previously been shown to be effective at learning these types of policies.

In guided policy search, the policy search task is transformed into a supervised learning problem, where supervision is provided by a set of simple trajectory-centric "teachers" that train linear-Gaussian controllers, denoted $p(\mathbf{u}_t|\mathbf{x}_t)$, that are each optimized independently on separate instances of the task, typically corresponding to different initial states. A key component in guided policy search is adaptation between the trajectories produced by the teacher and the final policy. This adaptation ensures that, at convergence, the teacher does not take actions that the final policy cannot reproduce. This is realized by an alternating optimization procedure, which iteratively optimizes the policy to match each teacher, while the teachers adapt to gradually match the behavior of the final policy. This alternating optimization is reminiscent of expectation maximization (EM), and previous work has formalized guided policy search as EM [5]. In our work, the trajectory-centric teachers use a reinforcement learning method that fits local, time-varying dynamics models, which gives us the added advantage of not requiring a known model of the system [2]. The partially observed variant of guided policy search, which we build off of, takes these ideas further, by also providing a different input to the trajectory-centric "teachers" compared to the policy. In this method, the linear-Gaussian controllers are trained under full state observation, while the policy is trained to mimic these controllers using only the observations $\mathbf{o}_t$ as input. This forces the policy to handle partial observation, while keeping the task easy for the controllers. This type of instrumented setup is natural for many robotic tasks, where training is done in a known laboratory setting, while the final policy must succeed under a variety of uncontrolled conditions. However, this method does not itself provide a way of handling internal memory, which is why we introduce memory states.

Since the memory states and memory writing actions are simply appended to the observation and action vectors, the supervised learning procedure for the policy remains identical, and the policy is automatically trained to use the memory actions to mimic the pattern of memory activations optimized by the trajectory-centric "teacher" algorithms. The trajectory-centric teacher optimizes linear-Gaussian controllers $p(\tilde{\mathbf{u}}_t|\tilde{\mathbf{x}}_t)$ that control both the physical and memory states, essentially choosing the memory that the policy needs to have in order to take the right action. This happens automatically, because guided policy search adds a term to the cost function that penalizes deviation from the policy $\pi_\theta(\tilde{\mathbf{u}}_t|\tilde{\mathbf{x}}_t)$ in terms of KL-divergence.

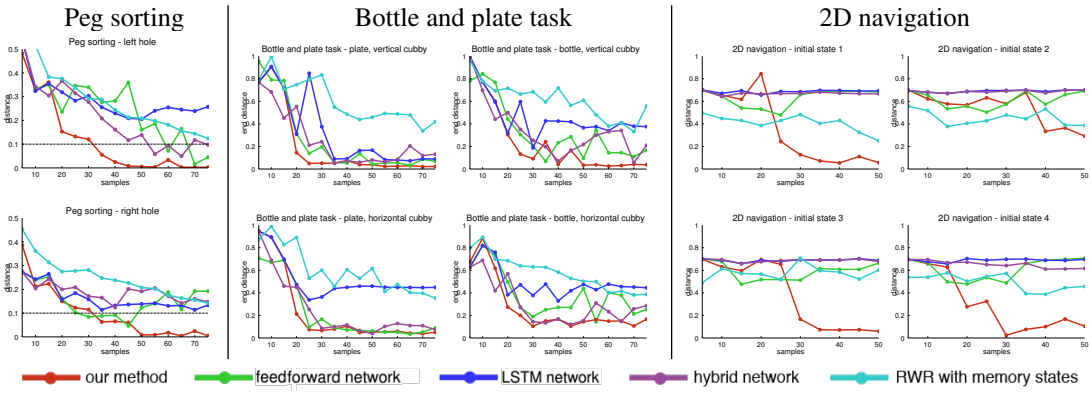

Figure 1: Plots of the distance to the target in terms of the number of samples. For each method, the different lines show the distance for a different target position under the same policy. Note that a successful policy must succeed on *all* of the conditions for the task. Our policy with memory states is able to successfully solve each of the tasks, while the alternative architectures and methods fail on at least one condition for each of the tasks.

3 Experimental Results

The aim of these experiments was to answer the following questions. First, can guided policy search with memory states solve complex tasks that require memory? Second, how do memory states compare with more standard RNN policies? Lastly, does guided policy search make it easier to train policies with memory states, compared to alternative policy search methods?

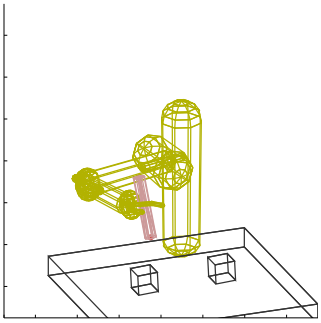

Figure 2: Starting configuration for the peg sorting task, where the robot is required to insert the pink peg into one of the two holes, which is specified on the first time step.

Our three experimental tasks were: (1) a simple navigation task that required the agent to travel to a designated position and, after the first half of the episode, return to the starting location, (2) a peg sorting task that required a robot was required to sort a peg into one of two holes, based on a target that it must remember, and (3) a bottle and plate task that required a robot to insert plates and bottles through a horizontal or vertical slot ("cubby"), and position them in the desired pose. All of our tasks involved some partially observed component, where the observation $\mathbf{o}_t$ received by the policy did not contain all of the information necessary to accomplish the task.

In addition to guided policy search with memory states, we evaluated three alternative policy representations, all trained with guided policy search, as well as an alternative optimization algorithm. The three alternate representations were: (1) a feedforward neural network, without memory, (2) a recurrent network with LSTM units, which have previously been shown to achieve good results for long-term memorization tasks [6], and (3) a hybrid network that consisted of both a feedforward branch and a recurrent LSTM branch at the first layer. Besides guided policy search, we also evaluated the memory states approach with the reward-weighted regression (RWR) algorithm [7, 8].

The results for each method on each of the tasks are presented in Figure 1. Each of the graphs shows the distance to the target for each task in terms of the number of samples used for training. For each task, we show separate plots for each condition. For the peg sorting task, there are two conditions corresponding to the two targets, and the dotted line shows the depth of the hole. Policies with minimum distances above this line fail to insert the peg into the hole. For the cubby task, the conditions correspond to the orientation of the cubby and whether the robot is holding a plate or bottle. For the navigation task, the conditions correspond to different starting states.

Good performance on each task requires the policy to succeed for all of the conditions. Our method was able to successfully complete all of the tasks under all conditions. The feedforward policy, with no memory capabilities, could not complete any of the tasks under all conditions, but could

successfully complete the manipulation tasks under some conditions. For example, for the peg sorting task, the feedforward policy simply picked the same target under both conditions. In general, we found the standard LSTM network to be more difficult to train than our method, which required substantially less tuning.[1] The hybrid network outperformed both the feedforward and pure LSTM policy, but still did not achieve the same performance as our memory states method. For both manipulation tasks, RWR was unable to discover an effective policy, either with a neural network parameterization or with the linear parameterization shown in the plots, though the linear variant achieved slightly lower cost. Due to the substantially lower dimensionality of the navigation task, RWR was in fact able to discover a policy that succeeded on one of the four conditions, but could not learn to effectively utilize the memory states to succeed from all four initial states.

The project website contains supplementary videos that illustrate the behavior of these policies.[2]

4 Discussion and Future Work

We presented a method for training policies for continuous control tasks that require memory. We augment the state space of the system with memory states, which the policy can choose to read from and write to as needed to accomplish the task. In order to make it tractable for the policy to learn effective memorization and recall strategies, we use guided policy search, which employs a simple trajectory-centric reinforcement learning algorithm to optimize over the memory state activations. This trajectory optimization procedure effectively tells the policy which information needs to be stored in the memory states, and the policy only needs to figure out how to reproduce the memory state activations. This tremendously simplifies the problem of searching over memorization strategies in comparison to model-free policy search methods and, unlike standard model-based methods for recurrent policies, it also avoids the need to backpropagate the gradient through time. However, when viewed together with the memory states, the policy is endowed with memory, and can be regarded as a recurrent neural network.

An interesting direction for follow-up work is to apply our approach for training recurrent networks for general supervised learning tasks, rather than just robotic control. In this case, the memory state comprises the entire state of the system, and the cost function is simply the supervised learning loss. Since the hidden memory state activations are optimized separately from the network weights, such an approach could in principle be more effective at training networks to perform complex reasoning over temporally extended intervals. Furthermore, since our method trains stochastic policies, it would also be able to train stochastic recurrent neural networks, where the transition dynamics are non-deterministic. These types of networks are typically quite challenging to train, and exploring this further is an exciting direction for future work.

References

[1] M. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," *Foundations and Trends in Robotics*, vol. 2, no. 1-2, pp. 1–142, 2013.

[2] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[3] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *arXiv preprint arXiv:1504.00702*, 2015.

[4] L. Peshkin, N. Meuleau, and L. Kaelbling, "Learning policies with external memory," *arXiv preprint cs/0103003*, 2001.

[5] S. Levine and V. Koltun, "Variational Policy Search via Trajectory Optimization," in *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] J. Peters and S. Schaal, "Applying the episodic natural actor-critic architecture to motor primitive learning," in *European Symposium on Artificial Neural Networks (ESANN)*, 2007.

[8] J. Kober and J. Peters, "Learning motor primitives for robotics," in *International Conference on Robotics and Automation (ICRA)*, 2009.

[1] We tested a variety of hyperparameters for the LSTM baseline and chose the best-performing policy.

[2] `http://rll.berkeley.edu/gpsrnn/`