Memory-based control with recurrent neural networks

Nicolas Heess* **Jonathan J Hunt*** **Timothy P Lillicrap** **David Silver**
Google Deepmind
* These authors contributed equally.
heess, jjhunt, countzero, davidsilver @ google.com

Abstract

Partially observed control problems are a challenging aspect of reinforcement learning. We extend two related, model-free algorithms for continuous control – deterministic policy gradient and stochastic value gradient – to solve partially observed domains using recurrent neural networks trained with backpropagation through time. We demonstrate that this approach, coupled with long-short term memory is able to solve a variety of physical control problems exhibiting an assortment of memory requirements. These include the short-term integration of information from noisy sensors and the identification of system parameters, as well as long-term memory problems that require preserving information over many time steps. We also demonstrate success on a combined exploration and memory problem in the form of a simplified version of the well-known Morris water maze task. Finally, we show that our approach can deal with high-dimensional observations by learning directly from pixels. We find that recurrent deterministic and stochastic policies are able to learn similarly good solutions to these tasks, including the water maze where the agent must learn effective search strategies.

1 Introduction

The use of neural networks for solving continuous control problems has a long tradition. Several recent papers successfully apply model-free, direct policy search methods to the problem of learning neural network control policies for challenging continuous domains with many degrees of freedoms [2, 6, 14, 21, 22, 12]. However, all of this work assumes fully observed state.

Many real world control problems are partially observed. Partial observability can arise from different sources including the need to remember information that is only temporarily available such as a way sign in a navigation task, sensor limitations or noise, unobserved variations of the plant under control (system identification), or state-aliasing due to function approximation. Partial observability also arises naturally in many tasks that involve control from vision: a static image of a dynamic scene provides no information about velocities, occlusions occur as a consequence of the three-dimensional nature of the world, and most vision sensors are bandwidth-limited and only have a restricted field-of-view.

Resolution of partial observability is non-trivial. Existing methods can roughly be divided into two broad classes:

On the one hand there are approaches that explicitly maintain a belief state that corresponds to the distribution over the world state given the observations so far. This approach has two major disadvantages: The first is the need for a model, and the second is the computational cost that is typically associated with the update of the belief state [8, 23].

On the other hand there are model free approaches that learn to form memories based on interactions with the world. This is challenging since it is a priori unknown which features of the observations will be relevant later, and associations may have to be formed over many steps. For this reason, most model free approaches tend to assume the fully-observed case. In practice, partial observability is often solved by hand-crafting a solution such as providing multiple-frames at each timestep to allow velocity estimation [16, 14].

In this work we investigate a natural extension of two recent, closely related policy gradient algorithms for learning continuous-action policies to handle partially observed problems. We primarily consider the Deterministic Policy Gradient algorithm (DPG) [24], which is an off-policy policy gradient algorithm that has recently produced promising results on a broad range of difficult, high-dimensional continuous control problems, including direct control from pixels [14]. DPG is an actor-critic algorithm that uses a learned approximation of the action-value (Q) function to obtain approximate action-value gradients. These are then used to update a deterministic policy via the chain-rule. We also consider DPG's stochastic counterpart, SVG(0) ([6]; SVG stands for "Stochastic Value Gradients") which similarly updates the policy via backpropagation of action-value gradients from an action-value critic but learns a stochastic policy.

We modify both algorithms to use recurrent networks trained with backpropagation through time. We demonstrate that the resulting algorithms, Recurrent DPG (RDPG) and Recurrent SVG(0) (RSVG(0)), can be applied to a number of partially observed physical control problems with diverse memory requirements. These problems include: short-term integration of sensor information to estimate the system state (pendulum and cartpole swing-up tasks without velocity information); system identification (cart pole swing-up with variable and unknown pole-length); long-term memory (a robot arm that needs to reach out and grab a payload to move it to the position the arm started from); as well as a simplified version of the water maze task which requires the agent to learn an exploration strategy to find a hidden platform and then remember the platform's position in order to return to it subsequently. We also demonstrate successful control directly from pixels.

Our results suggest that actor-critic algorithms that rely on bootstrapping for estimating the value function can be a viable option for learning control policies in partially observed domains. We further find that, at least in the setup considered here, there is little performance difference between stochastic and deterministic policies, despite the former being typically presumed to be preferable in partially observed domains.

2 Background

We model our environment as discrete-time, partially-observed Markov Decision process (POMDP). A POMDP is described a set of environment states $\mathcal{S}$ and a set of actions $\mathcal{A}$, an initial state distribution $p_0(s_0)$, a transition function $p(s_{t+1}|s_t, a_t)$ and reward function $r(s_t, a_t)$. This underlying MDP is partially observed when the agent is unable to observe the state s_t directly and instead receives observations from the set $\mathcal{O}$ which are conditioned on the underlying state $p(o_t|s_t)$.

The agent only indirectly observes the underlying state of the MDP through the observations. An optimal agent may, in principle, require access to the entire history $h_t = (o_1, a_1, o_2, a_2, ...a_{t-1}, o_t)$.

The goal of the agent is thus to learn a policy $\pi(h_t)$ which maps from the history to a distribution over actions $P(\mathcal{A})$ which maximizes the expected discounted reward (below we consider both stochastic and deterministic policies). For stochastic policies we want to maximise

$$J = \mathbb{E}_\tau \left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) \right], \tag{1}$$

where the trajectories $\tau = (s_1, o_1, a_1, s_2, \dots)$ are drawn from the trajectory distribution induced by the policy π: $p(s_1)p(o_1|s_1)\pi(a_1|h_1)p(s_2|s_1, a_1)p(o_2|s_2)\pi(a_2|h_2)\dots$ and where h_t is defined as above. For deterministic policies we replace π with a deterministic function μ which maps directly from states $\mathcal{S}$ to actions $\mathcal{A}$ and we replace $a_t \sim \pi(\cdot|h_t)$ with $a_t = \mu(h_t)$.

In the algorithms below we make use of the action-value function Q^π. For a fully observed MDP, when we have access to s, the action-value function is defined as the expected future discounted reward when in state s_t the agent takes action a_t and thereafter follows policy π. Since we are

interested in the partially observed case where the agent does not have access to s we instead define Q^π in terms of h:

$$Q^\pi(h_t, a_t) = \mathbb{E}_{s_t|h_t}\left[r_t(s_t, a_t)\right] + \mathbb{E}_{\tau_{>t}|h_t,a_t}\left[\sum_{i=1}^{\infty} \gamma^i r(s_{t+i}, a_{t+i})\right] \tag{2}$$

where $\tau_{>t} = (s_{t+1}, o_{t+1}, a_{t+1} \dots)$ is the future trajectory and the two expectations are taken with respect to the conditionals $p(s_t|h_t)$ and $p(\tau_{>t}|h_t, a_t)$ of the trajectory distribution associated with π. Note that this equivalent to defining Q^π in terms of the belief state since h is a sufficient statistic.

Obviously, for most POMDPs of interest, it is not tractable to condition on the entire sequence of observations. A central challenge is to learn how to summarize the past in a scalable way.

3 Algorithms

3.1 Recurrent DPG

We extend the Deterministic Policy Gradient (DPG) algorithm for MDPs introduced in [24] to deal with partially observed domains and pixels. The core idea of the DPG algorithm for the *fully observed* case is that for a deterministic policy μ^θ with parameters θ, and given access to the true action-value function associated with the current policy Q^μ, the policy can be updated by backpropagation:

$$\frac{\partial J(\theta)}{\partial \theta} = \mathbb{E}_{s\sim\rho^\mu}\left[\left.\frac{\partial Q^\mu(s,a)}{\partial a}\right|_{a=\mu^\theta(s)} \frac{\partial \mu^\theta(s)}{\partial \theta}\right], \tag{3}$$

where the expectation is taken with respect to the (discounted) state visitation distribution ρ^μ induced by the current policy μ^θ [24]. Similar ideas had previously been exploited in NFQCA [4] and in the ADP [13] community. In practice the exact action-value function Q^μ is replaced by an approximate (critic) Q^ω with parameters ω that is differentiable in a and which can be learned e.g. with Q-learning.

In order to ensure the applicability of our approach to large observation spaces (e.g. from pixels), we use neural networks for all function approximators. These networks, with convolutional layers have proven effective at many sensory processing tasks [11, 18], and been demonstrated to be effective for scaling reinforcement learning to large state spaces [14, 16]. [14] proposed modifications to DPG necessary in order to learn effectively with deep neural networks which we make use of here (cf. sections 3.1.1, 3.1.2).

Under partial observability the optimal policy and the associated action-value function are both functions of the entire preceding observation-action history h_t. The primary change we introduce is the use of recurrent neural networks, rather than feedforward networks, in order to allow the network to learn to preserve (limited) information about the past which is needed in order to solve the POMDP. Thus, writing $\mu(h)$ and $Q(h, a)$ rather than $\mu(s)$ and $Q(s, a)$ we obtain the following policy update:

$$\frac{\partial J(\theta)}{\partial \theta} = \mathbb{E}_\tau\left[\sum_t \gamma^{t-1} \left.\frac{\partial Q^\mu(h_t,a)}{\partial a}\right|_{a=\mu^\theta(h_t)} \frac{\partial \mu^\theta(h_t)}{\partial \theta}\right], \tag{4}$$

where we have written the expectation now explicitly over entire trajectories $\tau = (s_1, o_1, a_1, s_2, o_2, a_2, \dots)$ which are drawn from the trajectory distribution induced by the current policy and $h_t = (o_1, a_1, \dots, o_{t-1}, a_{t-1}, o_t)$ is the observation-action trajectory prefix at time step t, both as introduced above[1]. In practice, as in the fully observed case, we replace Q^μ by learned approximation Q^ω (which is also a recurrent network with parameters ω). Thus, rather than directly conditioning on the entire observation history, we effectively train recurrent neural networks to summarize this history in their recurrent state using backpropagation through time (BPTT). For

[1] A discount factor γ^t appears implicitly in the update which is absorbed in the discounted state-visitation distribution in eq. 3. In practice we ignore this term as is often done in policy gradient implementations in practice (e.g. [26]).

long episodes or continuing tasks it is possible to use truncated BPTT, although we do not use this here.

The full algorithm is given below (Algorithm 1).

RDPG is an algorithm for learning deterministic policies. As discussed in the literature [25, 20] it is possible to construct examples where deterministic policies perform poorly under partial observability. In RDPG the policy is conditioned on the entire history but since we are using function approximation state aliasing may still occur, especially early in learning. We therefore also investigate a recurrent version of the stochastic counterpart to DPG: SVG(0) [6] (DPG can be seen as the deterministic limit of SVG(0)). In addition to learning stochastic policies SVG(0) also admits on-policy learning whereas DPG is inherently off policy (see below).

Similar to DPG, SVG(0) updates the policy by backpropagation $\partial Q/\partial a$ from the action-value function, but does so for stochastic policies. This is enabled through a "re-parameterization" (e.g. [10, 19]) of the stochastic policy: The stochastic policy is represented in terms of a fixed, independent noise source and a parameterized deterministic function that transforms a draw from that noise source, i.e., in our case, $a = \pi^\theta(h, \nu)$ with $\nu \sim \beta(\cdot)$ where β is some fixed distribution. For instance, a Gaussian policy $\pi^\theta(a|h) = N(a|\mu^\theta(h), \sigma^2)$ can be re-parameterized as follows: $a = \pi^\theta(h, \nu) = \mu^\theta(h) + \sigma\nu$ where $\nu \sim N(\cdot|0, 1)$. See [6] for more details.

The stochastic policy is updated as follows:

$$\frac{\partial J(\theta)}{\partial \theta} = \mathbb{E}_{\tau,\nu}\left[\sum_t \gamma^{t-1} \left.\frac{\partial Q^{\pi^\theta}(h_t, a)}{\partial a}\right|_{a=\pi^\theta(h_t,\nu_t)} \frac{\partial \pi^\theta(h_t, \nu_t)}{\partial \theta}\right], \tag{5}$$

with τ drawn from the trajectory distribution which is conditioned on IID draws of ν_t from β at each time step. The full algorithm is provided in the supplementary (Algorithm 2).

3.1.1 Off-policy learning and experience replay

DPG is typically used in an off-policy setting due to the fact that the policy is deterministic but exploration is needed in order to learn the gradient of Q with respect to the actions. Furthermore, in practice, data efficiency and stability can also be greatly improved by using experience replay (e.g. [4, 5, 14, 16, 6]) and we use the same approach here (see Algorithms 1, 2). Thus, during learning we store experienced trajectories in a database and then replace the expectation in eq. (4) with trajectories sampled from the database.

One consequence of this is a bias in the state distribution in eqs. (3, 5) which no longer corresponds to the state distribution induced by the current policy . With function approximation this can lead to a bias in the learned policy, although this typically ignored in practice. RDPG and RSVG(0) may similarly be affected; in fact since policies (and Q) are not just a function of the state but of an entire action-observation history (eq. 4) the bias might be more severe.

One potential advantage of (R)SVG(0) in this context is that it allows on-policy learning although we do not explore this possibility here. We found that off-policy learning with experience replay remained effective in the partially observed case.

3.1.2 Target networks

A second algorithmic feature that has been found to greatly improve the stability of neural-network based reinforcement learning algorithms that rely on bootstrapping for learning value functions is the use of *target networks* [4, 14, 16, 6]: The algorithm maintains two copies of the value function Q and of the policy π each, with parameters θ and θ', and ω and ω' respectively. θ and ω are the parameters that are being updated by the algorithm; θ' and ω' track them with some delay and are used to compute the "targets values" for the Q function update. Different authors have explored different approaches to updating θ' and ω'. In this work we use "soft updates" as in [14] (see Algorithms 1 and 2 below).

Algorithm 1 RDPG algorithm

Initialize critic network $Q^{\omega}(a_t, h_t)$ and actor $\mu^{\theta}(h_t)$ with parameters ω and θ.
Initialize target networks $Q^{\omega'}$ and $\mu^{\theta'}$ with weights $\omega' \leftarrow \omega$, $\theta' \leftarrow \theta$.
Initialize replay buffer R.
for episodes = 1, M **do**
  initialize empty history h_0
  for t = 1, T **do**
    receive observation o_t
    $h_t \leftarrow h_{t-1}, a_{t-1}, o_t$ (append observation and previous action to history)
    select action $a_t = \mu^{\theta}(h_t) + \epsilon$ (with ϵ: exploration noise)
  end for
  Store the sequence $(o_1, a_1, r_1 ... o_T, a_T, r_T)$ in R
  Sample a minibatch of N episodes $(o_1^i, a_1^i, r_1^i, ... o_T^i, a_T^i, r_T^i)_{i=1,...,N}$ from R
  Construct histories $h_t^i = (o_1^i, a_1^i, \dots a_{t-1}^i, o_t^i)$
  Compute target values for each sample episode $(y_1^i, ... y_T^i)$ using the recurrent target networks

$$y_t^i = r_t^i + \gamma Q^{\omega'}(h_{t+1}^i, \mu^{\theta'}(h_{t+1}^i))$$

  Compute critic update (using BPTT)

$$\Delta\omega = \frac{1}{NT} \sum_i \sum_t \left(y_t^i - Q^{\omega}(h_t^i, a_t^i) \right) \frac{\partial Q^{\omega}(h_t^i, a_t^i)}{\partial \omega}$$

  Compute actor update (using BPTT)

$$\Delta\theta = \frac{1}{NT} \sum_i \sum_t \frac{\partial Q^{\omega}(h_t^i, \mu^{\theta}(h_t^i))}{\partial a} \frac{\partial \mu^{\theta}(h_t^i)}{\partial \theta}$$

  Update actor and critic using Adam [9]
  Update the target networks

$$\omega' \leftarrow \tau\omega + (1-\tau)\omega'$$
$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

end for

4 Results

We tested our algorithms on a variety of partial-observed environments, covering different types of memory problems. Videos of the learned policies for all the domains are included in our supplementary videos[2], we encourage viewing them as these may provide a better intuition for the environments. All physical control problems except the simulated water maze (section 4.3) were simulated in MuJoCo [28]. We tested both standard recurrent networks as well as LSTM networks.

4.1 Sensor integration and system identification

Physical control problems with noisy sensors are one of the paradigm examples of partially-observed environments. A large amount of research has focused on how to efficiently integrate noisy sensory information over multiple timesteps in order to derive accurate estimates of the system state, or to estimate derivatives of important properties of the system [27].

Here, we consider two simple, standard control problems often used in reinforcement learning, the under-actuated pendulum and cartpole swing up. We modify these standard benchmarks tasks such that in both cases the agent receives no direct information of the velocity of any of the components, i.e. for the pendulum swing-up task the observation comprises only the angle of the pendulum, and

[2]Video of all the learned policies is available at `https://youtu.be/V4_vb1D5NNQ`

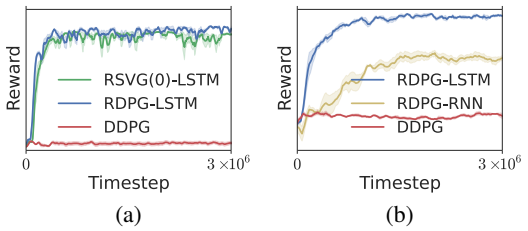


Figure 1: (a) The reward curve for the partially-observed pendulum task. Both RDPG and RSVG(0) are able to learn policies which bring the pendulum to an upright position. (b) The reward curve for the cartpole with no velocity and varying cartpole lengths. RDPG with LSTM, is able to reliably learn a good solution for this task; a purely feedforward agent (DDPG), which will not be able to estimate velocities nor to infer the pole length, is not able to solve the problem.

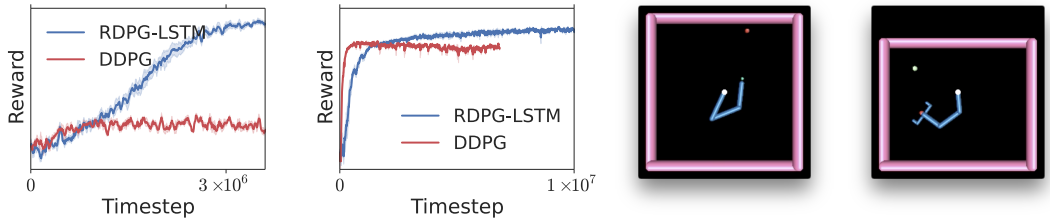


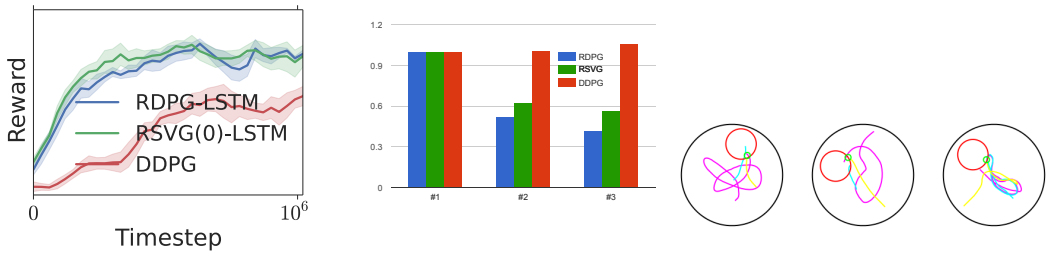
Figure 2: Reward curves for the (a) hidden target reacher task, and (b) return to start gripper task. In both cases the RDPG-agents with LSTMs are able to find good policies whereas the feedforward agents fail on the memory component. (In both cases the feedforward agents perform clearly better than random which is expected from the setup of the tasks: For instance, as can be seen in the video, the gripper without memory is still able to grab the payload and move it to a "default" position.) Example frames from the 3 joint reaching task (c) and the gripper task (d).

for cartpole swing-up it is limited to the angle of the pole and the position of the cart. Velocity is crucial for solving the task and thus it must be estimated from the history of the system. Figure 1a shows the learning curves for pendulum swing-up. Both RDPG and RSVG0 were tested on the pendulum task, and are able to learn good solutions which bring the pole to upright.

For the cartpole swing-up task, in addition to not providing the agent with velocity information, we also varied the length of the pole from episode to episode. The pole length is invisible to the agent and needs to be inferred from the response of the system. In this task the sensor integration problem is thus paired with the need for system identification. As can be seen in figure 1b, the RDPG agent with an LSTM network reliably solves this task every time while a simple feedforward agent (DDPG) fails entirely. RDPG with a simple RNN performs considerably less well than the LSTM agent, presumably due to relatively long episodes (T=350 steps) and the failure to backpropagate gradients effectively through the plain RNN. We found that a feedforward agent that does receive velocity information can solve the variable-length swing-up task partly but does so less reliably than the recurrent agent as it is unable to identify the relevant system parameters (not shown).

4.2 Memory tasks

Another type of partially-observed task, which has been less studied in the context of reinforcement learning, involves the need to remember explicit information over a number of steps. We constructed two tasks like this. One was a 3-joint reacher which must reach for a randomly positioned target, but the position of the target is only provided to the agent in the initial observation (the entire episode is 80 timesteps). As a harder variant of this task, we constructed a 5 joint gripper which must reach for a (fully-observed) payload from a randomized initial configuration and then return the payload to the initial position of its "hand" (T=100). Note that this is a challenging control problem even in the fully observed case. The results for both tasks are shown in figure 2, RDPG agents with LSTM networks solve both tasks reliably whereas purely feedforward agents fail on the memory components of the task as can be seen in the supplemental video.



(a) (b) (c) (d) (e)

Figure 3: (a) shows the reward curve for different agents performing the water maze task. Both recurrent algorithms are capable of learning good solutions to the problem, while the non-recurrent agent (DDPG) is not. It is particularly notable that despite learning a deterministic policy, RDPG is able find search strategies that allow it to locate the platform. (b) This shows the number of steps the agents take to reach the platform after a reset, normalized by the number of steps taken for the first attempt. Note that on the 2nd and 3rd attempts the recurrent agents are able to reach the platform much more quickly, indicating they are learning to remember and recall the position of the platform. Example trajectories for the (c) RDPG, (d) RSVG(0) and (e) DDPG agents. Trajectory of the first attempt is purple, second is blue and third is yellow.

4.3 Water maze

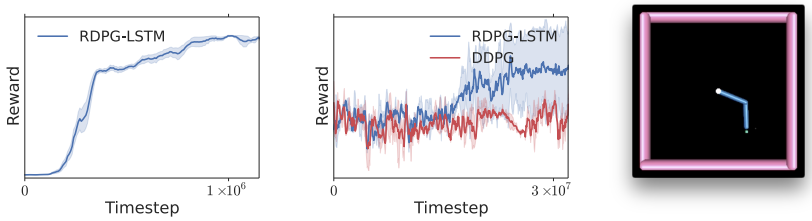
The Morris water maze has been used extensively in rodents for the study of memory [3]. We tested our algorithms on a simplified version of the task. The agent moves in a 2-dimensional circular space where a small region of the space is an invisible "platform" where the agent receives a positive reward. At the beginning of the episode the agent and platform are randomly positioned in the tank. The platform position is not visible to the agent but it "sees" when it is on platform. The agent needs to search for and stay on the platform to receive reward by controlling its acceleration. After 5 steps on the platform the agent is reset randomly to a new position in the tank but the platform stays in place for the rest of the episode (T=200). The agent needs to remember the position of the platform to return to it quickly.

It is sometimes presumed that a stochastic policy is required in order to solve problems like this, which require learning a search strategy. Although there is some variability in the results, we found that both RDPG and RSVG(0) were able to find similarly good solutions (figure 3a), indicating RDPG is able to learn reasonable, deterministic search strategies. Both solutions were able to make use of memory to return to the platform more quickly after discovering it during the initial search (figure 3b). A non-recurrent agent (DDPG) is able to learn a limited search strategy but fails to exploit memory to return the platform after having been reset to a random position in the tank.

4.4 High-dimensional observations

We also tested our agents, with convolutional networks, on solving tasks directly from high-dimensional pixel spaces. We tested on the pendulum task (but now the agent is given only a static rendering of the pendulum at each timestep), and a two-choice reaching task, where the target disappears after 5 frames (and the agent is not allowed to move during the first 5 frames to prevent it from encoding the target position in its initial trajectory).

We found that RDPG was able to learn effective policies from high-dimensional observations which integrate information from multiple timesteps to estimate velocity and remember the visually queued target for the full length of the episode (in the reacher task). Figure 4 shows the results.



(a) (b) (c)

Figure 4: RDPG was able to learn good policies directly from high-dimensional renderings for pendulum (a), and a two choice reaching task with a disappearing target (b). (c) Example frame from the reaching task.

5 Discussion

5.1 Variants

In the experiments presented here, the actor and critic networks are entirely disjoint. However, particularly when learning deep, convolutional networks the filters required in the early layers may be similar between the policy and the actor. Sharing these early layers could improve computational efficiency and learning speed. Similar arguments apply to the recurrent part of the network, which could be shared between the actor and the critic. Such sharing, however, can also result in instabilities as updates to one network may unknowingly damage or shift the other network. For this reason, we have not used any sharing here, although it is a potential topic for further investigation.

5.2 Related work

There is a large body of literature on solving partially observed control problems. We focus on the most closely related work that aims to solve such problems with learned memory.

Several groups [15, 1, 5] have studied the use of model-free algorithms with recurrent networks to solve POMDPs with discrete action spaces. [1] focused on relatively long-horizon ("deep") memory problems in small state-action spaces. In contrast, [5] modified the Atari DQN architecture [16] (i.e. they perform control from high-dimensional pixel inputs) and demonstrated that recurrent Q learning [15] can perform the required information integration to resolve short-term partial observability (e.g. to estimate velocities) that is achieved via stacks of frames in the original DQN architecture.

Continuous action problems with relatively low-dimensional observation spaces have been considered e.g. in [30, 31, 29, 32]. [30] trained LSTM-based stochastic policies using Reinforce; [31, 29, 32] used actor-critic architectures. The algorithm of [31] can be seen as a special case of DPG where the deterministic policy produces the parameters of an action distribution from which the actions are then sampled. This requires suitable exploration at the level of distribution parameters (e.g. exploring in terms of means and variances of a Gaussian distribution); in contrast, SVG(0) also learns stochastic policies but allows exploration at the action level only.

All works mentioned above, except for [32], consider the memory to be internal to the policy and learn the RNN parameters using BPTT, back-propagating either TD errors or policy gradients. [32] instead take the view of [17] and consider memory as extra state dimensions that can can be read and set by the policy. They optimize the policy using guided policy search [12] which performs explicit trajectory optimization along reference trajectories and, unlike our approach, requires a well defined full latent state and access to this latent state during training.

6 Conclusion

We have demonstrated that two related model-free approaches can be extended to learn effectively with recurrent neural networks on a variety of partially-observed problems, including directly from pixel observations. Since these algorithms learn using standard backpropagation through time, we

are able to benefit from innovations in supervised recurrent neural networks, such as long-short term memory networks [7], to solve challenging memory problems such as the Morris water maze.

References

[1] B. Bakker. Reinforcement learning with long short-term memory. In *NIPS*, 2002.

[2] D. Balduzzi and M. Ghifary. Compatible value gradients for reinforcement learning of continuous deep policies. *arXiv preprint arXiv:1509.03005*, 2015.

[3] R. DHooge and P. P. De Deyn. Applications of the morris water maze in the study of learning and memory. *Brain research reviews*, 36(1):60–90, 2001.

[4] R. Hafner and M. Riedmiller. Reinforcement learning in feedback control. *Machine learning*, 84(1-2):137–169, 2011.

[5] M. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*, 2015.

[6] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa. Learning continuous control policies by stochastic value gradients. In *NIPS*, 2015.

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

[8] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134, 1998.

[9] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

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

[13] F. L. Lewis and D. Vrabie. Reinforcement learning and adaptive dynamic programming for feedback control. *Circuits and Systems Magazine, IEEE*, 9(3):32–50, 2009.

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

[15] L.-J. Lin and T. M. Mitchell. Reinforcement learning with hidden states. In J.-A. Meyer, H. L. Roitblat, and S. W. Wilson, editors, *From animals to animats 2*, pages 271–280. MIT Press, Cambridge, MA, USA, 1993.

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

[17] L. Peshkin, N. Meuleau, and L. P. Kaelbling. Learning policies with external memory. In *ICML*, 1999.

[18] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pages 512–519. IEEE, 2014.

[19] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1278–1286, 2014.

[20] B. Sallans. *Reinforcement learning for factored markov decision processes*. PhD thesis, Citeseer, 2002.

[21] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz. Trust region policy optimization. In *ICML*, 2015.

[22] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015.

[23] G. Shani, J. Pineau, and R. Kaplow. A survey of point-based pomdp solvers. *Autonomous Agents and Multi-Agent Systems*, 27(1):1–51, 2013.

[24] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[25] S. P. Singh. Learning without state-estimation in partially observable markovian decision processes. In *ICML*, 1994.

[26] P. Thomas. Bias in natural actor-critic algorithms. In *Proceedings of The 31st International Conference on Machine Learning*, pages 441–448, 2014.

[27] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT press, 2005.

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

[29] H. Utsunomiya and K. Shibata. Contextual behaviors and internal representations acquired by reinforcement learning with a recurrent neural network in a continuous state and action space task. In M. Kppen, N. Kasabov, and G. Coghill, editors, *Advances in Neuro-Information Processing*, volume 5507 of *Lecture Notes in Computer Science*, pages 970–978. Springer Berlin Heidelberg, 2009.

[30] D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber. Solving deep memory pomdps with recurrent policy gradients. In *ICANN*, 2007.

[31] D. Wierstra and J. Schmidhuber. Policy gradient critics. In *ECML*, 2007.

[32] M. Zhang, S. Levine, Z. McCarthy, C. Finn, and P. Abbeel. Policy learning with continuous memory states for partially observed robotic control. *CoRR*, abs/1507.01273, 2015.

7 Supplementary

Algorithm 2 RSVG(0) algorithm

Initialize critic network $Q^{\omega}(a_t, h_t)$ and actor $\pi^{\theta}(h_t)$ with parameters ω and θ.
Initialize target networks $Q^{\omega'}$ and $\pi^{\theta'}$ with weights $\omega' \leftarrow \omega$, $\theta' \leftarrow \theta$.
Initialize replay buffer R.
for episodes = 1, M **do**
  initialize empty history h_0
  for t = 1, T **do**
    receive observation o_t
    $h_t \leftarrow h_{t-1}, a_{t-1}, o_t$ (append observation and previous action to history)
    select action $a_t = \pi^{\theta}(h_t, \nu)$ with $\nu \sim \beta$)
  end for
  Store the sequence $(o_1, a_1, r_1 ... o_T, a_T, r_T)$ in R
  Sample a minibatch of N episodes $(o_1^i, a_1^i, r_1^i, ... o_T^i, a_T^i, r_T^i)_{i=1,...,N}$ from R
  Construct histories $h_t^i = (o_1^i, a_1^i, \dots a_{t-1}^i, o_t^i)$
  Compute target values for each sample episode $(y_1^i, ... y_T^i)$ using the recurrent target networks

$$y_t^i = r_t^i + \gamma Q^{\omega'}(h_{t+1}^i, \pi^{\theta'}(h_{t+1}^i, \nu)) \quad \text{with } \nu \sim \beta$$

  Compute critic update (using BPTT)

$$\Delta\omega = \frac{1}{NT} \sum_i \sum_t (y_t^i - Q^{\omega}(h_t^i, a_t^i) \frac{\partial Q^{\omega}(h_t^i, a_t^i)}{\partial \omega}$$

  Compute actor update (using BPTT)

$$\Delta\theta = \frac{1}{NT} \sum_i \sum_t \frac{\partial Q^{\omega}(h_t^i, \pi^{\theta}(h_t^i, \nu))}{\partial a} \frac{\partial \pi^{\theta}(h_t^i, \nu)}{\partial \theta} \qquad \text{with } \nu \sim \beta$$

  Update actor and critic using Adam [9]
  Update the target networks

$$\omega' \leftarrow \tau\omega + (1-\tau)\omega'$$
$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

end for