Recurrent Predictive State Policy Networks

Ahmed Hefny [*1] **Zita Marinho** [*2 3] **Wen Sun** [2] **Siddhartha S. Srinivasa** [4] **Geoffrey Gordon** [1]

Abstract

We introduce Recurrent Predictive State Policy (RPSP) networks, a recurrent architecture that brings insights from predictive state representations to reinforcement learning in partially observable environments. Predictive state policy networks consist of a recursive filter, which keeps track of a belief about the state of the environment, and a reactive policy that directly maps beliefs to actions. The recursive filter leverages predictive state representations (PSRs) (Rosencrantz & Gordon, 2004; Sun et al., 2016) by modeling *predictive state* — a prediction of the distribution of future observations conditioned on history and future actions. This representation gives rise to a rich class of statistically consistent algorithms (Hefny et al., 2018) to initialize the recursive filter. Predictive state serves as an equivalent representation of a belief state. Therefore, the policy component of the RPSP-network can be purely reactive, simplifying training while still allowing optimal behaviour. We optimize our policy using a combination of policy gradient based on rewards (Williams, 1992) and gradient descent based on prediction error of the recursive filter. We show the efficacy of RPSP-networks under partial observability on a set of robotic control tasks from OpenAI Gym. We empirically show that RPSP-networks perform well compared with memory-preserving networks such as GRUs, as well as finite memory models, being the overall best performing method.

[*]Equal contribution [1]Machine Learning Department, Carnegie Mellon University, Pittsburgh, USA [2]Robotics Institute, Carnegie Mellon University, Pittsburgh, USA [3]ISR/IT, Instituto Superior Técnico, Lisbon, Portugal [4]Paul G. Allen School of Computer Science & Engineering, University of Washington, Seattle, USA. Correspondence to: Ahmed Hefny <ahefny@cs.cmu.edu>, Zita Marinho <zmarinho@cmu.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

1. Introduction

Recently, there has been significant progress in deep reinforcement learning (Bojarski et al., 2016; Schulman et al., 2015; Mnih et al., 2013; Silver et al., 2016). Deep reinforcement learning combines deep networks as a representation of the policy with reinforcement learning algorithms and enables end-to-end training.

While traditional applications of deep learning rely on standard architectures with sigmoid or ReLU activations, there is an emerging trend of using composite architectures that contain parts explicitly resembling other algorithms such as Kalman filtering (Haarnoja et al., 2016) and value iteration (Tamar et al., 2016). It has been shown that such architectures can outperform standard neural networks.

In this work, we focus on partially observable environments, where the agent does not have full access to the state of the environment, but only to partial observations thereof. The agent has to maintain instead a distribution over states, *i.e.*, a belief state, based on the entire history of observations and actions. The standard approach to this problem is to employ recurrent architectures such as Long-Short-Term-Memory (LSTM) (Hochreiter & Schmidhuber, 1997) and Gated Recurrent Units (GRU) (Cho et al., 2014). Despite their success (Hausknecht & Stone, 2015), these methods are difficult to train due to non-convexity, and their hidden states lack a predefined statistical meaning.

Models based on predictive state representations (Littman et al., 2001; Singh et al., 2004; Rosencrantz & Gordon, 2004; Boots et al., 2013) offer an alternative method to construct a surrogate for belief state in a partially observable environment. These models represent state as the expectation of sufficient statistics of future observations, conditioned on history and future actions. Predictive state models admit efficient learning algorithms with theoretical guarantees. Moreover, the successive application of the predictive state update procedure (i.e., filtering equations) results in a recursive computation graph that is differentiable with respect to model parameters. Therefore, we can treat predictive state models as recurrent networks and apply backpropagation through time (BPTT) (Hefny et al., 2018; Downey et al., 2017) to optimize model parameters. We use this insight to construct a *Recurrent Predictive State Policy* (RPSP) network, a special recurrent architecture that consists of (1) a

predictive state model acting as a recursive filter to keep track of a predictive state, and (2) a feed-forward neural network that directly maps predictive states to actions. This configuration results in a recurrent policy, where the recurrent part is implemented by a PSR instead of an LSTM or a GRU. As predictive states are a sufficient summary of the history of observations and actions, the reactive policy will have rich enough information to make its decisions, as if it had access to a true belief state. There are a number of motivations for this architecture:

- Using a PSR means we can benefit from methods in the spectral learning literature to provide an efficient and statistically consistent initialization of a core component of the policy.
- Predictive states have a well defined probabilistic interpretation as conditional distribution of observed quantities. This can be utilized for optimization.
- The recursive filter in RPSP-networks is fully differentiable, meaning that once a good initialization is obtained from spectral learning methods, we can refine RPSP-nets using gradient descent.

This network can be trained end-to-end, for example using policy gradients in a reinforcement learning setting (Sutton et al., 2001) or supervised learning in an imitation learning setting (Ross et al., 2011). In this work we focus on the former. We discuss the predictive state model component in §3. The control component is presented in §4 and the learning algorithm is presented in §5. In §6 we describe the experimental setup and results on control tasks: we evaluate the performance of reinforcement learning using predictive state policy networks in multiple partially observable environments with continuous observations and actions.

2. Background and Related Work

Throughout the rest of the paper, we will define vectors in bold notation $\mathbf{v}$, matrices in capital letters W. We will use $\otimes$ to denote vectorized outer product: $\mathbf{x} \otimes \mathbf{y}$ is $\mathbf{x}\mathbf{y}^\top$ reshaped into a vector.

We assume an agent is interacting with the environment in episodes, where each episode consists of T time steps in each of which the agent takes an action $\mathbf{a}_t \in \mathcal{A}$, and observes an observation $\mathbf{o}_t \in \mathcal{O}$ and a reward $r_t \in \mathbb{R}$. The agent chooses actions based on a stochastic policy $\pi_{\boldsymbol{\theta}}$ parameterized by a parameter vector θ: $\pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{o}_{1:t-1}, \mathbf{a}_{1:t-1}) \equiv p(\mathbf{a}_t \mid \mathbf{o}_{1:t-1}, \mathbf{a}_{1:t-1}, \boldsymbol{\theta})$. We would like to improve the policy rewards by optimizing $\boldsymbol{\theta}$ based on the agent's experience in order to maximize the expected long term reward $J(\pi_{\boldsymbol{\theta}}) = \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\left[\gamma^{t-1} r_t \mid \pi_{\boldsymbol{\theta}}\right]$, where $\gamma \in [0, 1]$ is a discount factor.

There are two major approaches for model-free reinforcement learning. The first is the value function-based approach, where we seek to learn a function (e.g., a deep network (Mnih et al., 2013)) to evaluate the value of each action at each state (a.k.a. Q-value) under the optimal policy (Sutton & Barto, 1998). Given the Q function the agent can act greedily based on estimated values. The second approach is direct policy optimization, where we learn a function to directly predict optimal actions (or optimal action distributions). This function is optimized to maximize $J(\theta)$ using policy gradient methods (Schulman et al., 2015; Duan et al., 2016) or derivative-free methods (Szita & Lrincz, 2006). We focus on the direct policy optimization approach as it is more robust to noisy continuous environments and modeling uncertainty (Sutton et al., 2001; Wierstra et al., 2010).

Our aim is to provide a new class of policy functions that combines recurrent reinforcement learning with recent advances in modeling partially observable environments using predictive state representations (PSRs). There have been previous attempts to combine predictive state models with policy learning. Boots et al. (2011) proposed a method for planning in partially observable environments. The method first learns a PSR from a set of trajectories collected using an explorative blind policy. The predictive states estimated by the PSR are then considered as states in a fully observable Markov Decision Process. A value function is learned on these states using least squares temporal difference (Boots & Gordon, 2010) or point-based value iteration (PBVI) (Boots et al., 2011). The main disadvantage of these approaches is that it assumes a one-time initialization of the PSR and does not propose a mechanism to update the model based on subsequent experience.

Hamilton et al. (2014) proposed an iterative method to simultaneously learn a PSR and use the predictive states to fit a Q-function. Azizzadenesheli et al. (2016) proposed a tensor decomposition method to estimate the parameters of a discrete partially observable Markov decision process (POMDP). One common limitation in the aforementioned methods is that they are restricted to discrete actions (some even assume discrete observations). Also, it has been shown that PSRs can benefit greatly from local optimization after a moment-based initialization (Downey et al., 2017; Hefny et al., 2018).

Venkatraman et al. (2017) proposed predictive state decoders, where an LSTM or a GRU network is trained on a mixed objective function in order to obtain high cumulative rewards while accurately predicting future observations. While it has shown improvement over using standard training objective functions, it does not solve the initialization issue of the recurrent network.

Our proposed RPSP networks alleviate the limitations of previous approaches: It supports continuous observations

and actions, it uses a recurrent state tracker with consistent initialization, and it supports end-to-end training after the initialization.

3. Predictive State Representations of Controlled Models

In this section, we give a brief introduction to predictive state representations, which constitute the state tracking (filtering) component of our model.[1] We provide more technical details in the appendix.

Given a history of observations and actions $\mathbf{a}_1, \mathbf{o}_1, \mathbf{a}_2, \mathbf{o}_2, \ldots, \mathbf{a}_{t-1}, \mathbf{o}_{t-1}$, a recursive filter computes a belief state $\mathbf{q}_t$ using a recursive update equation $\mathbf{q}_{t+1} = f(\mathbf{q}_t, \mathbf{a}_t, \mathbf{o}_t)$. Given the state $\mathbf{q}_t$, one can predict observations through a function $g(\mathbf{q}_t, \mathbf{a}_t) \equiv \mathbb{E}[\mathbf{o}_t \mid \mathbf{q}_t, \mathbf{a}_t]$. In a recurrent neural network (Figure 1 (a,b)), $\mathbf{q}$ is latent and the function g that connects states to the output is unknown and has to be learned. In this case, the output could be predicted from observations, when the RNN is used for prediction, see Figure 1 (a,b).

Predictive state models use a *predictive* representation of the state. That means the $\mathbf{q}_t$ is explicitly defined as the conditional distribution of future observations $\mathbf{o}_{t:t+k-1}$ conditioned on future actions $\mathbf{a}_{t:t+k-1}$.[2] (e.g., in the discrete case, $\mathbf{q}_t$ could be a vectorized conditional probability table).

Predictive states are thus defined entirely in terms of observable features with no latent variables involved. That means the mapping between the predictive state $\mathbf{q}_t$ and the prediction of $\mathbf{o}_t$ given $\mathbf{a}_t$ can be fully known or simple to learn consistently (Hefny et al., 2015b; Sun et al., 2016). This is in contrast to RNNs, where this mapping is unknown and requires non-convex optimization to be learned.

Similar to an RNN, a PSR employs a recursive state update that consists of the following two steps:[3]

- State extension: A linear map W_{ext} is applied to $\mathbf{q}_t$ to obtain an *extended state* $\mathbf{p}_t$. This state defines a conditional distribution over an extended window of $k+1$ observations and actions. W_{ext} is a parameter to be learned.

$$\mathbf{p}_t = W_{\text{ext}} \mathbf{q}_t \tag{1}$$

- Conditioning: Given $\mathbf{a}_t$ and $\mathbf{o}_t$, and a known conditioning function f_{cond}:

$$\mathbf{q}_{t+1} = f_{\text{cond}}(\mathbf{p}_t, \mathbf{a}_t, \mathbf{o}_t). \tag{2}$$

Figure 1 (c, d) depicts the PSR state update. The conditioning function f_{cond} depends on the representation of $\mathbf{q}_t$ and $\mathbf{p}_t$. For example, in a discrete system, $\mathbf{q}_t$ and $\mathbf{p}_t$ could represent conditional probability tables and f_{cond} amounts to applying Bayes rule. In continuous systems we can use Hilbert space embedding of distributions (Boots et al., 2013), where f_{cond} uses kernel Bayes rule (Fukumizu et al., 2013).

In this work, we use the RFFPSR model proposed in (Hefny et al., 2018). Observation and action features are based on random Fourier features (RFFs) of RBF kernel (Rahimi & Recht, 2008) projected into a lower dimensional subspace using randomized PCA (Halko et al., 2011). We use ϕ to denote this feature function. Conditioning function f_{cond} is kernel Bayes rule, and observation function g is a linear function of state $\mathbb{E}[\mathbf{o}_t \mid \mathbf{q}_t, \mathbf{a}_t] = W_{\text{pred}}(\mathbf{q}_t \otimes \phi(\mathbf{a}_t))$. See Section 9.1 in the appendix for more details.

3.1. Learning predictive states representations

Learning PSRs is carried out in two steps: an initialization procedure using method of moments and a local optimization procedure using gradient descent.

Initialization: The initialization procedure exploits the fact that $\mathbf{q}_t$ and $\mathbf{p}_t$ are represented in terms of observable quantities: since W_{ext} is linear and using (1), then $\mathbb{E}[\mathbf{p}_t \mid \mathbf{h}_t] = W_{\text{ext}} \mathbb{E}[\mathbf{q}_t \mid \mathbf{h}_t]$. Here $\mathbf{h}_t \equiv h(\mathbf{a}_{1:t-1}, \mathbf{o}_{1:t-1})$ denotes a set of features extracted from previous observations and actions (typically from a fixed length window ending at $t-1$). Because $\mathbf{q}_t$ and $\mathbf{p}_t$ are not hidden states, estimating these expectations on both sides can be done by solving a supervised regression subproblem. Given the predictions from this regression, solving for W_{ext} then becomes another linear regression problem. We follow this two-stage regression proposed by Hefny et al. (2018).[4] Once W_{ext} is computed, we can perform filtering to obtain the predictive states $\mathbf{q}_t$. We then use the estimated states to learn the mapping to predicted observations W_{pred}, which results in another regression subproblem, see Section 9.2 in the appendix for more details.

In RFFPSR, we use linear regression for all subproblems (which is a reasonable choice with kernel-based features). This ensures that the two-stage regression procedure is free of local optima.

Local Optimization: Although PSR initialization procedure is consistent, it is based on method of moments and hence is not necessarily statistically efficient. Therefore it

[1] We follow the predictive state controlled model formulation in Hefny et al. (2018). Alternative methods such as predictive state inference machines (Sun et al., 2016) could be contemplated.

[2] We condition on "intervention by actions" rather than "observing actions". That means $\mathbf{q}_t$ is independent of the policy that determines the actions. See (Pearl, 2009).

The length-k depends on the observability of the system. A system is k-observable if maintaining the predictive state is equivalent to maintaining the distribution of the system's latent state.

[3] See the appendix Section 9 for more details.

[4] We use the joint stage-1 regression variant for initialization.

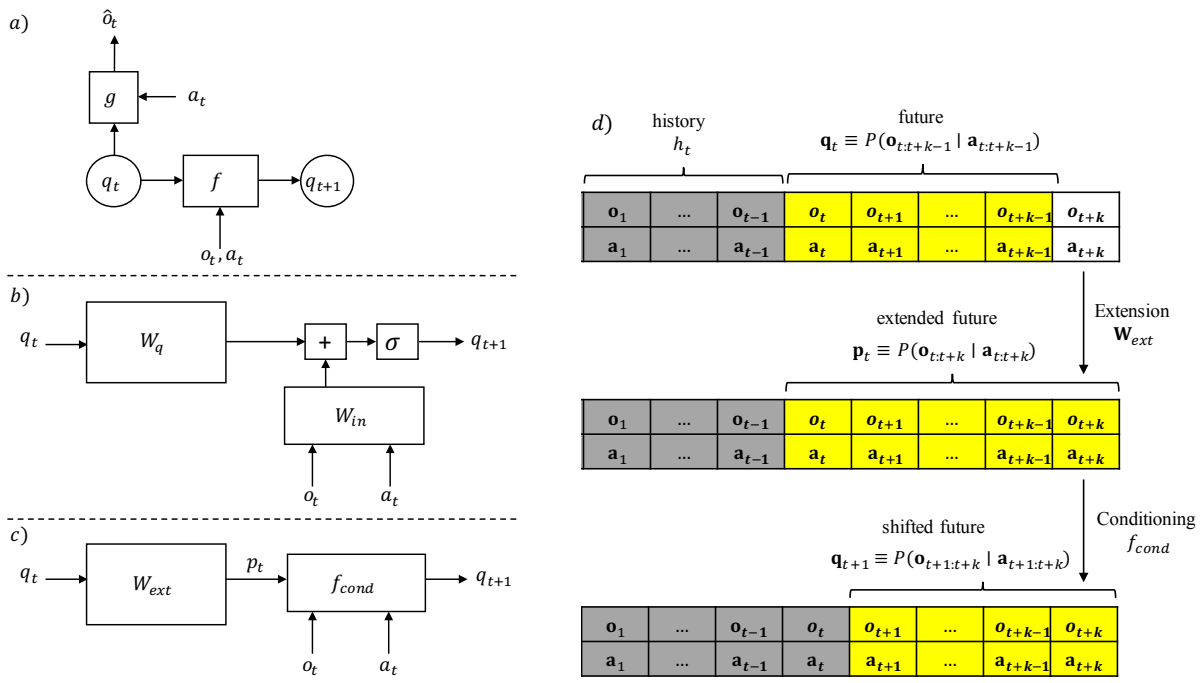


Figure 1: **Left:** a) Computational graph of RNN and PSR and b) the details of the state update function f for both a simple RNN and c) a PSR. Compared to RNN, the observation function g is easier to learn in a PSR (see §3.1). **Right:** Illustration of the PSR extension and conditioning steps.

can benefit from local optimization. Downey et al. (2017) and Hefny et al. (2018) note that a PSR defines a recursive computation graph similar to that of an RNN where we have

$$\begin{aligned} \mathbf{q}_{t+1} &= f_{\text{cond}}(W_{\text{ext}}(\mathbf{q}_t), \mathbf{a}_t, \mathbf{o}_t)) \\ \mathbb{E}[\mathbf{o}_t \mid \mathbf{q}_t, \mathbf{a}_t] &= W_{\text{pred}}(\mathbf{q}_t \otimes \phi(\mathbf{a}_t)), \end{aligned} \tag{3}$$

With a differentiable f_{cond}, the PSR can be trained using backpropagation through time to minimize prediction error.

In a nutshell, a PSR effectively constitutes a special type of a recurrent network where the state representation and update are chosen in a way that permits a consistent initialization, which is then followed by conventional backpropagation.

4. Recurrent Predictive State Policy (RPSP) Networks

We now introduce our proposed class of policies, Recurrent Predictive State Policies (RPSPs). We describe its components and in §5 we describe the learning algorithm .

RPSPs consist of two fundamental components: a state tracking component, which models the state of the system, and is able to predict future observations; and a reactive policy component, that maps states to actions, shown in Figure 2. The state tracking component is based on the PSR formulation described in §3. The reactive policy is a stochas-

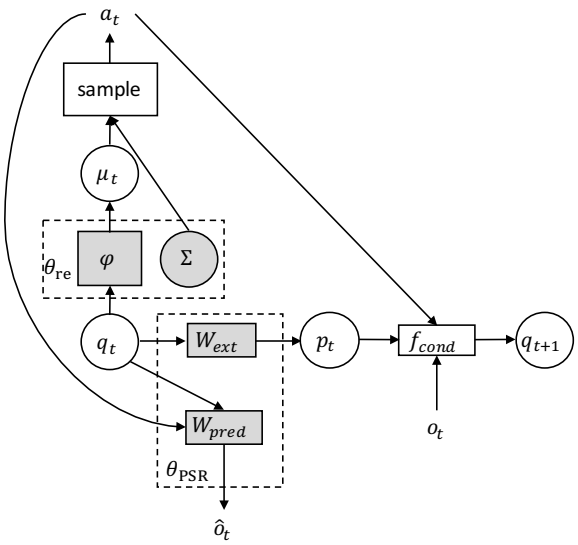


Figure 2: RPSP network: The predictive state is updated by a linear extension W_{ext} followed by a non-linear conditioning f_{cond}. A linear predictor W_{pred} is used to predict observations, which is used to regularize training loss (see §5). A feed-forward reactive policy maps the predictive states $\mathbf{q}_t$ to a distribution over actions.

tic non-linear policy $\pi_{\text{re}}(\mathbf{a}_t \mid \mathbf{q}_t) \equiv p(\mathbf{a}_t \mid \mathbf{q}_t; \boldsymbol{\theta}_{\text{re}})$ which maps a predictive state to a distribution over actions and is

Algorithm 1 Recurrent Predictive State Policy network Optimization (RPSPO)

Input: Learning rate η.
Sample initial trajectories: $\{(o_t^i, a_t^i)_t\}_{i=1}^M$ from π_{exp}.
Initialize PSR:
  $\boldsymbol{\theta}^0_{\text{PSR}} = \{\mathbf{q}_0, W_{\text{ext}}, W_{\text{pred}}\}$ via 2-stage regression in §3.
Initialize reactive policy $\boldsymbol{\theta}^0_{\text{re}}$ randomly.
for $n = 1 \dots N_{max}$ iterations **do**
  for $i = 1, \dots, M$ batch of M trajectories from π^{n-1}: **do**
    Reset episode: a_0^i.
    for $t = 0 \dots T$ *roll-in* in each trajectory: **do**
      Get observation o_t^i and reward r_t^i.
      Filter $\mathbf{q}_{t+1}^i = f_t(\mathbf{q}_t^i, \mathbf{a}_t^i, \mathbf{o}_t^i)$ in (Eq. 3).
      Execute $\mathbf{a}_{t+1}^i \sim \pi_{re}^{n-1}(\mathbf{q}_{t+1}^i)$.
    end for
  end for
  Update $\boldsymbol{\theta}$ using $\mathcal{D} = \{\{\mathbf{o}_t^i, \mathbf{a}_t^i, r_t^i, \mathbf{q}_t^i\}_{t=1}^T\}_{i=1}^M$:
    $\boldsymbol{\theta}^n \leftarrow \text{UPDATE}(\boldsymbol{\theta}^{n-1}, \mathcal{D}, \eta)$, as in §5.
end for
Output: Return $\boldsymbol{\theta} = (\boldsymbol{\theta}_{\text{PSR}}, \boldsymbol{\theta}_{\text{re}})$.

parametrized by $\boldsymbol{\theta}_{\text{re}}$. Similar to Schulman et al. (2015) we assume a Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_t, \Sigma)$, where

$$\boldsymbol{\mu} = \varphi(\mathbf{q}_t; \boldsymbol{\theta}_\mu); \qquad \Sigma = \text{diag}(\exp(\mathbf{r}))^2 \tag{4}$$

for a non-linear map φ parametrized by $\boldsymbol{\theta}_\mu$ (e.g. a feedforward network) , and a learnable vector $\mathbf{r}$. An RPSP is thus a stochastic recurrent policy with the recurrent part corresponding to a PSR. The parameters $\boldsymbol{\theta}$ consist of two parts: the PSR parameters $\boldsymbol{\theta}_{\text{PSR}} = \{\mathbf{q}_0, W_{\text{ext}}, W_{\text{pred}}\}$ and the reactive policy parameters $\boldsymbol{\theta}_{\text{re}} = \{\boldsymbol{\theta}_\mu, \mathbf{r}\}$. In the following section, we describe how these parameters are learned.

5. Learning RPSPs

As detailed in Algorithm 1, learning an RPSP is performed in two phases.[5] In the first phase, we execute an exploration policy to collect a dataset that is used to initialize the PSR as described in §3.1. It is worth noting that this initialization procedure depends on observations rather than rewards. This can be particularly useful in environments where informative reward signals are infrequent.

In the second phase, starting from the initial PSR and a random reactive policy, we iteratively collect trajectories using the current policy and use them to update the parameters of both the reactive policy $\boldsymbol{\theta}_{\text{re}} = \{\boldsymbol{\theta}_\mu, \mathbf{r}\}$ and the predictive model $\boldsymbol{\theta}_{\text{PSR}} = \{\mathbf{q}_0, W_{\text{ext}}, W_{\text{pred}}\}$, as depicted in Algorithm 1. Let $p(\tau \mid \boldsymbol{\theta})$ be the distribution over trajectories induced by the policy $\pi_{\boldsymbol{\theta}}$. By updating parameters, we seek to minimize the objective function in (5).

$$\mathcal{L}(\theta) = \alpha_1 \ell_1(\theta) + \alpha_2 \ell_2(\theta) \tag{5}$$
$$= -\alpha_1 J(\pi_\theta) + \alpha_2 \sum_{t=0}^{T} \mathbb{E}_{p(\tau|\boldsymbol{\theta})} \left[\|W_{pred}(\mathbf{q}_t \otimes \mathbf{a}_t) - \mathbf{o}_t\|^2 \right],$$

which combines negative expected returns with PSR prediction error.[6] Optimizing the PSR parameters to maintain low prediction error can be thought of as a regularization scheme. The hyper-parameters $\alpha_1, \alpha_2 \in \mathbb{R}$ determine the importance of the expected return and prediction error respectively. They are discussed in more detail in §5.3.

Noting that RPSP is a special type of a recurrent network policy, it is possible to adapt policy gradient methods (Williams, 1992) to the joint loss in (5). In the following subsections, we propose different update variants.

5.1. Joint Variance Reduced Policy Gradient (VRPG)

In this variant, we use REINFORCE method (Williams, 1992) to obtain a stochastic gradient of $J(\pi)$ from a batch of M trajectories.

Let $R(\tau) = \sum_{t=1}^{T} \gamma^{t-1} r_t$ be the cumulative discounted reward for trajectory τ given a discount factor $\gamma \in [0, 1]$. REINFORCE uses the likelihood ratio trick $\nabla_{\boldsymbol{\theta}} p(\tau|\boldsymbol{\theta}) = p(\tau|\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log p(\tau|\boldsymbol{\theta})$ to compute $\nabla_{\boldsymbol{\theta}} J(\pi)$ as

$$\nabla_{\boldsymbol{\theta}} J(\pi) = \mathbb{E}_{\tau \sim p(\tau|\boldsymbol{\theta})}[R(\tau) \sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{q}_t)],$$

In practice, we use a variance reducing variant of policy gradient (Greensmith et al., 2001) given by

$$\nabla_{\boldsymbol{\theta}} J(\pi) = \mathbb{E}_{\tau \sim p(\tau|\boldsymbol{\theta})} \sum_{t=0}^{T} [\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{q}_t)(R_t(\tau) - b_t)], \tag{6}$$

where we replace the cumulative trajectory reward $R(\tau)$ by a reward-to-go function $R_t(\tau) = \sum_{j=t}^{T} \gamma^{j-t} r_j$ computing the cumulative reward starting from t. To further reduce variance we use a *baseline* $b_t \equiv \mathbb{E}_{\boldsymbol{\theta}}[R_t(\tau) \mid \mathbf{a}_{1:t-1}, \mathbf{o}_{1:t}]$ which estimates the expected reward-to-go conditioned on the current policy. In our implementation, we assume $b_t = \mathbf{w}_b^\top \mathbf{q}_t$ for a parameter vector $\mathbf{w}_b$ that is estimated using linear regression. Given a batch of M trajectories, a stochastic gradient of $J(\pi)$ can be obtained by replacing the expectation in (6) with the empirical expectation over trajectories in the batch. A stochastic gradient of the prediction error can be obtained using backpropagation through time. With an estimate of both gradients, we can compute (5) and update the parameters trough gradient descent. For more details, see Algorithm 2 in the appendix.

[5] https://github.com/ahefnycmu/rpsp

[6] We minimize 1-step prediction error, as opposed to general k-future prediction error recommended by (Hefny et al., 2018), to avoid biased estimates induced by non causal statistical correlations (observations correlated with future actions) when performing on-policy updates when a non-blind policy is in use.

5.2. Alternating Optimization

In this section, we describe a method that utilizes the recently proposed Trust Region Policy Optimization (TRPO (Schulman et al., 2015)), an alternative to the vanilla policy gradient methods that has shown superior performance in practice.It uses a natural gradient update and enforces a constraint that encourages small changes in the policy in each TRPO step. This constraint results in smoother changes of policy parameters.

Each TRPO update is an approximate solution to the following constrained optimization problem in (7).

$$\boldsymbol{\theta}^{n+1} = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim p(\tau|\pi^n)} \sum_{t=0}^{T} \left[\frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{q}_t)}{\pi^n(\mathbf{a}_t|\mathbf{q}_t)} (R_t(\tau) - b_t) \right]$$

$$\text{s.t. } \mathbb{E}_{\tau\sim p(\tau|\pi^n)} \sum_{t=0}^{T} \left[D_{KL}\left(\pi^n(.|\mathbf{q}_t) \mid \pi_{\boldsymbol{\theta}}(.|\mathbf{q}_t)\right)\right] \leq \epsilon, \quad (7)$$

where π^n is the policy induced by $\boldsymbol{\theta}^n$, and R_t and b_t are the reward-to-go and baseline functions defined in §5.1.

While it is possible to extend TRPO to the joint loss in (5), we observed that TRPO tends to be computationally intensive with recurrent architectures. Instead, we resort to the following alternating optimization:[7] In each iteration, we use TRPO to update the reactive policy parameters $\boldsymbol{\theta}_{\text{re}}$, which involve only a feedforward network. Then, we use a gradient step on (5), as described in §5.1, to update the PSR parameters $\boldsymbol{\theta}_{\text{PSR}}$, see Algorithm 3 in the appendix.

5.3. Variance Normalization

It is difficult to make sense of the values of α_1, α_2, specially if the gradient magnitudes of their respective losses are not comparable. For this reason, we propose a more principled approach for finding the relative weights. We use $\alpha_1 = \tilde{\alpha}_1$ and $\alpha_2 = a_2\tilde{\alpha}_2$, where a_2 is a user-given value, and $\tilde{\alpha}_1$ and $\tilde{\alpha}_2$ are dynamically adjusted to maintain the property that the gradient of each loss weighted by $\tilde{\alpha}$ has unit (uncentered) variance, in (8). In doing so, we maintain the variance of the gradient of each loss through exponential averaging and use it to adjust the weights.

$$\mathbf{v}_i^{(n)} = (1-\beta)\mathbf{v}_i^{(n-1)} + \beta \sum_{\boldsymbol{\theta}_j \in \boldsymbol{\theta}} \|\nabla^{(n)}_{\boldsymbol{\theta}_j} \ell_i\|^2 \quad (8)$$

$$\tilde{\alpha}_i^{(n)} = \left(\sum_{\boldsymbol{\theta}_j \in \boldsymbol{\theta}} \mathbf{v}_{i,j}^{(n)} \right)^{-1/2},$$

6. Experiments

We evaluate the RPSP-network's performance on a collection of reinforcement learning tasks using OpenAI Gym Mujoco environments. [8] We consider partially observable environments: only the angles of the joints of the agent are visible to the network, without velocities.

Proposed Models: We consider an RPSP with a predictive component based on RFFPSR, as described in §3 and §4. For the RFFPSR, we use 1000 random Fourier features on observation and action sequences followed by a PCA dimensionality reduction step to d dimensions. We report the results for the best choice of $d \in \{10, 20, 30\}$.

We initialize the RPSP with two stage regression on a batch of M_i initial trajectories (100 for Hopper, Walker and CartPole, and 50 for Swimmer) (equivalent to 10 extra iterations, or 5 for Swimmer). We then experiment with both joint VRPG optimization (**RPSP-VRPG**) described in §5.1 and alternating optimization (**RPSP-Alt**) in §5.2. For RPSP-VRPG, we use the gradient normalization described in §5.3.

Additionally, we consider an extended variation (**+obs**) that concatenates the predictive state with a window w of previous observations as an extended form of predictive state $\tilde{\mathbf{q}}_t = [\mathbf{q}_t, \mathbf{o}_{t-w:t}]$. If PSR learning succeeded perfectly, this extra information would be unnecessary; however we observe in practice that including observations help the model learn faster and more stably. Later in the results section we report the RPSP variant that performs best. We provide a detailed comparison of all models in the appendix.

Competing Models: We compare our models to a finite memory model (**FM**) and gated recurrent units (**GRU**). The finite memory models are analogous to RPSP, but replace the predictive state with a window of past observations. We tried three variants, FM1, FM2 and FM5, with window size of 1, 2 and 5 respectively (FM1 ignores that the environment is partially observable). We compare to GRUs with 16, 32, 64 and 128-dimensional hidden states. We optimize network parameters using the *RLLab*[9] implementation of TRPO with two different learning rates ($\eta = 10^{-2}, 10^{-3}$).

In each model, we use a linear baseline for variance reduction where the state of the model (i.e. past observation window for FM, latent state for GRU and predictive state for RPSP) is used as the predictor variable.

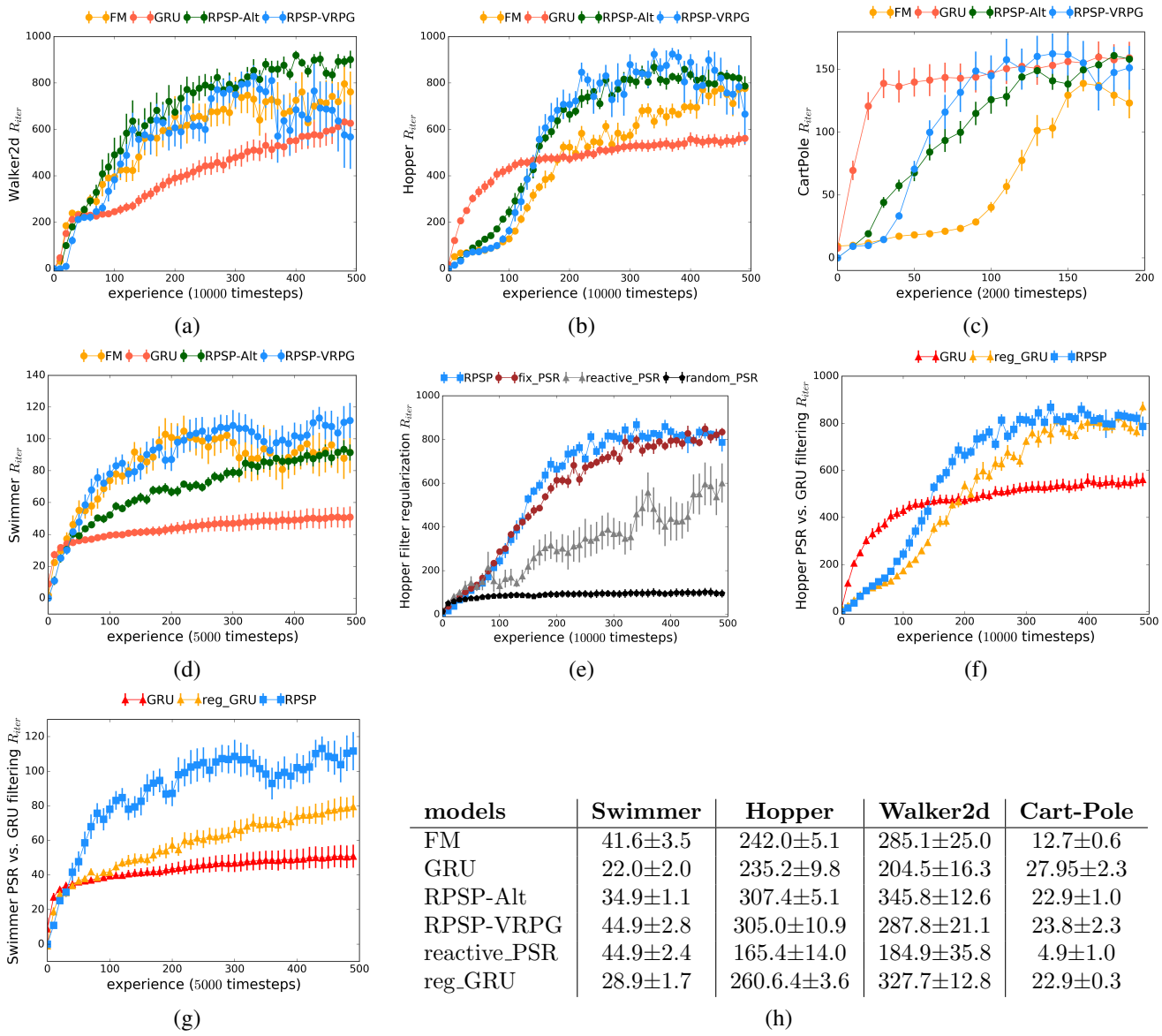
Evaluation Setup: We run each algorithm for a number of iterations based on the environment (see Figure 3). After each iteration, we compute the average return $R_{iter} = \frac{1}{M}\sum_{m=1}^{M}\sum_{j=1}^{T_m} r_m^j$ on a batch of M trajectories, where T_m is the length of the m^{th} trajectory. We repeat this process using 10 different random seeds and report the average and standard deviation of R_{iter} for each iteration.

For each environment, we set the number of samples in the batch to 10000 and the maximum length of each episode to

[7] We emphasize that both VRPG and alternating optimization models optimize the joint RL/prediction loss. They differ only on how to update the reactive policy parameters (which are independent of prediction error).

[8] https://gym.openai.com/envs#mujoco

[9] https://github.com/openai/rllab



models	Swimmer	Hopper	Walker2d	Cart-Pole
FM	41.6±3.5	242.0±5.1	285.1±25.0	12.7±0.6
GRU	22.0±2.0	235.2±9.8	204.5±16.3	27.95±2.3
RPSP-Alt	34.9±1.1	307.4±5.1	345.8±12.6	22.9±1.0
RPSP-VRPG	44.9±2.8	305.0±10.9	287.8±21.1	23.8±2.3
reactive_PSR	44.9±2.4	165.4±14.0	184.9±35.8	4.9±1.0
reg_GRU	28.9±1.7	260.6.4±3.6	327.7±12.8	22.9±0.3

(h)

Figure 3: Empirical average return over 10 epochs (bars indicate standard error). **(a-d):** Finite memory model $w = 2$ (FM), GRUs, best performing RPSP with joint optimization (RPSP-VRPG) and best performing RPSP with alternate optimization (RPSP-Alt) on four environments. **(e):** RPSP variations: fixed PSR parameters (fix_PSR), without prediction regularization (reactive_PSR), random initialization (random_PSR). **(f-g):** Comparison with GRU + prediction regularization (reg_GRU). RPSP graphs are shifted to the right to reflect initialization trajectories. **(h)**: Cumulative rewards (area under curve).

200, 500, 1000, 1000 for Cart-Pole, Swimmer, Hopper and Walker2d respectively.[10]

For RPSP, we found that a step size of 10^{-2} performs well for both VRPG and alternating optimization in all environments. The reactive policy contains one hidden layer of 16 nodes with ReLU activation. For all models, we report the results for the choice of hyper-parameters that resulted in the highest mean cumulative reward (area under curve).

[10]For example, for a 1000 length environment we use a batch of 10 trajectories resulting in 10000 samples in the batch.

7. Results and Discussion

Performance over iterations: Figure 3 shows the empirical average return vs. the amount of interaction with the environment (experience), measured in time steps. We observe that RPSP networks (especially RPSP-Alt) perform well in every environment, competing with or outperforming the top model in terms of the learning speed and the

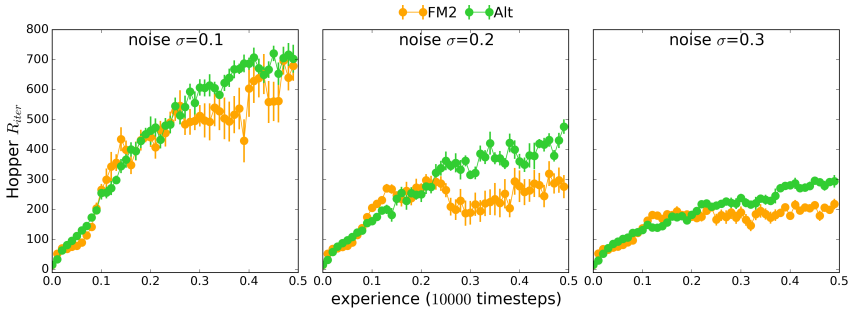


Figure 4: Empirical average return over 10 trials with a batch of $M = 10$ trajectories of $T = 1000$ time steps for Hopper. (Left to right) Robustness to observation Gaussian noise $\sigma = \{0.1,\ 0.2,\ 0.3\}$, best RPSP with alternate loss (Alt) and Finite Memory model (FM2).

final reward, with the exception of Cart-Pole where the gap to GRU is larger. We report the cumulative reward for all environments in Table 3(h). For all except Cart-Pole, come variant of RPSP is the best performing model. For Swimmer our best performing model is only statistically better than FM model (t-test, $p < 0.01$), while for Hopper our best RPSP model performs statistically better than FM and GRU models (t-test, $p < 0.01$) and for Walker2d RPSP outperforms only GRU baselines (t-test, $p < 0.01$). For Cart-Pole the top RPSP model performs better than the FM model (t-test, $p < 0.01$) and it is not statistically significantly different than the GRU model. We also note that RPSP-Alt provides similar performance to the joint optimization (**RPSP-VRPG**), but converges faster.

Effect of proposed contributions: Our RPSP model is based on a number of components: (1) State tracking using PSR (2) Consistent initialization using two-stage regression (3) End-to-end training of state tracker and policy (4) Using observation prediction loss to regularize training.

We conducted a set of experiments to verify the benefit of each component.[11] In the first experiment, we test three variants of RPSP: one where the PSR is randomly initialized (**random_PSR**), another one where the PSR is fixed at the initial value and only the reactive policy is further updated (**fix_PSR**), and a third one where we train the RPSP network with initialization and without prediction loss regularization (i.e. we set α_2 in (5)) to 0 (**reactive_PSR**). Figure 3(e) demonstrates that these variants are inferior to our model, showing the importance of two-stage initialization, end-to-end training and observation prediction loss respectively.

In the second experiment, we replace the PSR with a GRU that is initialized using BPTT applied on exploration data. This is analogous to the predictive state decoders proposed in (Venkatraman et al., 2017), where observation prediction loss is included when optimizing a GRU policy network (**reg_GRU**).[12] Figure 3(f-g) shows that a GRU model is inferior to a PSR model, where the initialization procedure is consistent and does not suffer from local optima.

Effect of observation noise: We also investigated the effect of observation noise on the RPSP model and the competitive FM baseline by applying Gaussian noise of increasing variance to observations. Figure 4 shows that while FM was very competitive with RPSP in the noiseless case, RPSP has a clear advantage over FM in the case of mild noise. The performance gap vanishes under excessive noise.

8. Conclusion

We propose RPSP-networks, combining ideas from predictive state representations and recurrent networks for reinforcement learning. We use PSR learning algorithms to provide a statistically consistent initialization of the state tracking component, and propose gradient-based methods to maximize expected return while reducing prediction error. We compare RPSP against different baselines and empirically show the efficacy of the proposed approach in terms of speed of convergence and overall expected return.

One direction to investigate is how to develop an online, consistent and statistically efficient method to update the RFFPSR as a predictor in continuous environments. There has been a body of work for online learning of predictive state representations (Venkatraman et al., 2016; Boots & Gordon, 2011; Azizzadenesheli et al., 2016; Hamilton et al., 2014). To our knowledge, none of them is able to deal with continuous actions and make use of local optimization. We are also interested in applying off-policy methods and more elaborate exploration strategies.

[11] Due to space limitation, we report results on Hopper environment. We report results for other environments in the appendix.

[12] We report results for partially observable setting which is different from RL experiments in (Venkatraman et al., 2017).

References

Azizzadenesheli, K., Lazaric, A., and Anandkumar, A. Reinforcement learning of pomdp's using spectral methods. *CoRR*, abs/1602.07764, 2016. URL http://arxiv.org/abs/1602.07764.

Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., and Zieba, K. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

Boots, B. and Gordon, G. An online spectral learning algorithm for partially observable nonlinear dynamical systems. In *Proceedings of the 25th National Conference on Artificial Intelligence (AAAI)*, 2011.

Boots, B. and Gordon, G. J. Predictive state temporal difference learning. In *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010.*, pp. 271–279, 2010.

Boots, B., Siddiqi, S., and Gordon, G. Closing the learning planning loop with predictive state representations. In *I. J. Robotic Research*, volume 30, pp. 954–956, 2011.

Boots, B., Gretton, A., and Gordon, G. J. Hilbert Space Embeddings of Predictive State Representations. In *Proc. 29th Intl. Conf. on Uncertainty in Artificial Intelligence (UAI)*, 2013.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Empirical Methods in Natural Language Processing, EMNLP*, pp. 1724–1734, 2014.

Downey, C., Hefny, A., Boots, B., Gordon, G. J., and Li, B. Predictive state recurrent neural networks. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 6055–6066. Curran Associates, Inc., 2017.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 1329–1338, 2016.

Fukumizu, K., Song, L., and Gretton, A. Kernel bayes' rule: Bayesian inference with positive definite kernels. *Journal of Machine Learning Research*, 14(1):3753–3783, 2013.

Greensmith, E., Bartlett, P. L., and Baxter, J. Variance reduction techniques for gradient estimates in reinforcement learning. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, pp. 1507–1514, Cambridge, MA, USA, 2001. MIT Press.

Haarnoja, T., Ajay, A., Levine, S., and Abbeel, P. Backprop kf: Learning discriminative deterministic state estimators. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 4376–4384. Curran Associates, Inc., 2016.

Halko, N., Martinsson, P. G., and Tropp, J. A. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 2011.

Hamilton, W., Fard, M. M., and Pineau, J. Efficient learning and planning with compressed predictive states. *J. Mach. Learn. Res.*, 15(1):3395–3439, January 2014.

Hausknecht, M. J. and Stone, P. Deep recurrent q-learning for partially observable mdps. *CoRR*, abs/1507.06527, 2015.

Hefny, A., Downey, C., and Gordon, G. J. Supervised learning for dynamical system learning. In *NIPS*. 2015a.

Hefny, A., Downey, C., and Gordon, G. J. Supervised learning for dynamical system learning. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, pp. 1963–1971. 2015b.

Hefny, A., Downey, C., and Gordon, G. J. An efficient, expressive and local minima-free method for learning controlled dynamical systems. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17089.

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

Littman, M. L., Sutton, R. S., and Singh, S. Predictive representations of state. In *In Advances In Neural Information Processing Systems 14*, pp. 1555–1561. MIT Press, 2001.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning, 2013. URL http://arxiv.org/abs/1312.5602. cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013.

Pearl, J. *Causality: Models, Reasoning and Inference*. Cambridge University Press, New York, NY, USA, 2nd edition, 2009. ISBN 052189560X, 9780521895606.

Rahimi, A. and Recht, B. Random features for large-scale kernel machines. In *NIPS*. 2008.

Rosencrantz, M. and Gordon, G. Learning low dimensional predictive representations. In *ICML*, pp. 695–702, 2004.

Ross, S., Gordon, G. J., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, pp. 627–635, 2011.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In Blei, D. and Bach, F. (eds.), *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1889–1897. JMLR Workshop and Conference Proceedings, 2015.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

Singh, S., James, M. R., and Rudary, M. R. Predictive state representations: A new theory for modeling dynamical systems. In *UAI*, 2004.

Sun, W., Venkatraman, A., Boots, B., and Bagnell, J. A. Learning to filter with predictive state inference machines. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 1197–1205, 2016.

Sutton, R., Mcallester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12 (Proceedings of the 1999 conference)*, pp. 1057–1063. MIT Press, 2001.

Sutton, R. S. and Barto, A. G. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

Szita, I. and Lrincz, A. Learning tetris using the noisy cross-entropy method. *Neural Computation*, 18(12):2936–2941, Dec 2006.

Tamar, A., Levine, S., Abbeel, P., and andonline Garrett Thomas, Y. W. Value iteration networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 2146–2154, 2016.

Venkatraman, A., Sun, W., Hebert, M., Bagnell, J. A., and Boots, B. Online instrumental variable regression with applications to online linear system identification. In *AAAI*, 2016.

Venkatraman, A., Rhinehart, N., Sun, W., Pinto, L., Boots, B., Kitani, K., and Bagnell., J. A. Predictive state decoders: Encoding the future into recurrent networks. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2017.

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

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, pp. 229–256, 1992.

Appendix

9. Predictive state representations and two stage regression

In this section we give more details on the filter component (RFFPSR) of the RPSP network. We explain the state update equation and the two-stage initialization algorithm. We encourage the reader to refer to (Hefny et al., 2018) for more details, derivations and theoretical analysis. Let $\mathbf{a}_{1:t-1} = \mathbf{a}_1, \dots, \mathbf{a}_{t-1} \in \mathcal{A}^{t-1}$ be the set of actions performed by an agent, followed by observations $\mathbf{o}_{1:t-1} = \mathbf{o}_1, \dots, \mathbf{o}_{t-1} \in \mathcal{O}^{t-1}$ received by the environment up to time t. Together they compose the entire *history* up to time t $\mathbf{h}_t^\infty \equiv \{\mathbf{a}_{1:t-1}, \mathbf{o}_{1:t-1}\}$. We define *future* observations at time t as the sequence of consecutive k observations $\mathbf{o}_{t:t+k-1} \in \mathcal{O}^k$ and *extended future* observations as the sequence of consecutive $k+1$ observations $\mathbf{o}_{t:t+k} \in \mathcal{O}^{k+1}$. Same definitions apply for actions.

We now define feature mappings shown in Figure 1, for immediate and future actions and observations:

- $\boldsymbol{\phi}^o(\mathbf{o}_t) : \mathcal{O} \mapsto \mathbb{R}^O$ of immediate observations,
- $\boldsymbol{\phi}^a(\mathbf{a}_t) : \mathcal{A} \mapsto \mathbb{R}^A$ of immediate actions,
- $\boldsymbol{\psi}^o(\mathbf{o}_{t:t+k-1}) : \mathcal{O}^k \mapsto \mathbb{R}^{F^O}$ of future observations,
- $\boldsymbol{\psi}^a(\mathbf{a}_{t:t+k-1}) : \mathcal{A}^k \mapsto \mathbb{R}^{F^A}$ of future actions
- $\boldsymbol{\xi}^o(\mathbf{o}_{t:t+k}) \equiv (\boldsymbol{\psi}^o_{t+1} \otimes \boldsymbol{\phi}^o_t, \boldsymbol{\phi}^o_t \otimes \boldsymbol{\phi}^o_t) : \mathcal{O}^{k+1} \mapsto \mathbb{R}^{F^O \times O} \times \mathbb{R}^{O \times O}$ of extended observations,
- $\boldsymbol{\xi}^a(\mathbf{a}_{t:t+k}) \equiv \boldsymbol{\psi}^a_{t+1} \otimes \boldsymbol{\phi}^a_t : \mathcal{A}^{k+1} \mapsto \mathbb{R}^{F^A \times A}$ of extended actions.

, where $\boldsymbol{\phi}^o_t$ is a shorthand for $\boldsymbol{\phi}^o(\mathbf{o}_t)$.

These feature maps basically compute Random Fourier features (Rahimi & Recht, 2008) projected on a lower dimensional space using PCA. For faster computation of the projection we use randomized PCA (Halko et al., 2011).

Given these feature functions we define the predictive state $\mathbf{q}_t$ to be a linear map from future action features $\boldsymbol{\psi}^t$ to expected future observation features $\mathbb{E}[\boldsymbol{\psi}^o_t \mid o_{1:t-1}, \mathbf{do}(a_{1:t+k-1})]$, where $\mathbf{do}(a_{1:t+k-1})$ means intervening by actions $a_{1:t+k-1}$ instead of observing actions $a_{1:t+k-1}$ (Pearl, 2009) (in the discrete case, think of a conditional probability table whose values are determined by observations and actions up to time $t-1$). Such a linear map is be represented by a matrix, which can be vectorized and projected using PCA.

We define the extended state $\mathbf{p}_t = (\mathbf{p}^\xi_t, \mathbf{p}^o_t)$ as a tuple consisting of two similar linear maps:

$$\mathbf{p}^\xi_t \equiv \boldsymbol{\xi}^a_t \rightarrow \mathbb{E}[\boldsymbol{\psi}^o_t \otimes \boldsymbol{\phi}^o_t \mid o_{1:t-1}, \mathbf{do}(a_{1:t+k})]$$
$$\mathbf{p}^o_t \equiv \boldsymbol{\xi}^o_t \rightarrow \mathbb{E}[\boldsymbol{\phi}^o_t \otimes \boldsymbol{\phi}^o_t \mid o_{1:t-1}, \mathbf{do}(a_{1:t})]$$

The key property in the extended state $\mathbf{p}_t$ is that, given o_t and a_t, we can compute $\mathbf{q}_{t+1}$ using kernel Bayes rule (Fukumizu et al., 2013) as we will demonstrate.

We assume there is a linear transformation $W_{\text{ext}} \equiv (W^\xi_{\text{ext}}, W^o_{\text{ext}})$ such that

$$\mathbf{p}_t = W_{\text{ext}} \mathbf{q}_t.$$

With this model formulation, it remains to show how to perform the state update and how to learn the parameter W_{ext}.

9.1. State update in RFFPSR

- Given a state $\mathbf{q}_t$ we compute the extended state $\mathbf{p}_t$ and undo the PCA projection.
- Given the action a_t and $\mathbf{p}^o_t$ we can compute

  $$C_t \equiv \mathbb{E}[\boldsymbol{\phi}^o_t \otimes \boldsymbol{\phi}^o_t \mid o_{1:t-1}, \mathbf{do}(a_{1:t})]$$

- We can think of the map $\mathbf{p}^\xi_t$ as a 4-mode tensors with modes corresponding to $\boldsymbol{\psi}^o_{t+1}$, $\boldsymbol{\phi}^o_t$, $\boldsymbol{\psi}^a_{t+1}$, and $\boldsymbol{\phi}^a_t$. Kernel Bayes rule tells us that, by multiplying $(C_t + \lambda I)^{-1}$ along the $\boldsymbol{\phi}^o_t$ mode we get a tensor for the map

  $$\boldsymbol{\psi}^a_{t+1}, \boldsymbol{\phi}^a_t, \boldsymbol{\phi}^o_t \rightarrow \mathbb{E}[\boldsymbol{\psi}^a_{t+1} \mid o_{1:t}, \mathbf{do}(a_{1:t+k})]$$

  Given a_t and o_t we compute the corresponding features and plug them in the previous map (by multiplying with the appropriate modes) to give the matrix for the map

  $$\boldsymbol{\psi}^a_{t+1} \rightarrow \mathbb{E}[\boldsymbol{\psi}^a_{t+1} \mid o_{1:t}, \mathbf{do}(a_{1:t+k})],$$

  which after vectorizing and projection is $\mathbf{q}_{t+1}$.

9.2. Learning RFFPSR Parameters

If we have access to examples of $\mathbf{q}_t$ and $\mathbf{p}_t$, we can learn W_{ext} using linear regression. However, obtaining these examples is as hard as learning the RFFPSR. Two-stage regression (Hefny et al., 2015a; 2018) is based on the observation that we can replace $\mathbf{q}_t$ and $\mathbf{p}_t$ with their expectations given history features

$$\bar{\mathbf{q}}_t \equiv \mathbb{E}[\mathbf{q}_t \mid \mathbf{h}_t]$$
$$\bar{\mathbf{p}}_t \equiv \mathbb{E}[\mathbf{p}_t \mid \mathbf{h}_t]$$

, where $\mathbf{h}_t \equiv h(\mathbf{a}_{1:t-1}, \mathbf{o}_{1:t-1})$ denotes a set of features extracted from previous observations and actions (typically from a fixed length window ending at $t-1$). Computing $\bar{\mathbf{q}}_t$ and $\bar{\mathbf{p}}_t$ is referred to as stage 1 (S1) regression. We use the joint S1 regression method described in (Hefny et al., 2018). We collect training data of tuples $(\mathbf{h}_t, \boldsymbol{\psi}^o_t, \boldsymbol{\psi}^a_t)$. We train a

Algorithm 2 UPDATE (VRPG)

Input: $\boldsymbol{\theta}^{n-1}$, trajectories $\mathcal{D}$=$\{\tau^i\}_{i=1}^M$, and learning rate η.
Estimate a linear baseline $b_t = \mathbf{w}_b^\top \mathbf{q}_t$, from the expected reward-to-go function for the batch $\mathcal{D}$:

$$\mathbf{w}_b = \arg\min_{\mathbf{w}} \left\| \frac{1}{TM} \sum_{i=1}^{M} \sum_{t=1}^{T_i} R_t(\tau_t^i) - \mathbf{w}^\top \mathbf{q}_t \right\|^2.$$

Compute the VRPG loss gradient w.r.t. $\boldsymbol{\theta}$, in (6):

$$\nabla_{\boldsymbol{\theta}} \ell_1 = \frac{1}{M} \sum_{i=1}^{M} \sum_{t=0}^{T_i} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t^i | \mathbf{q}_t^i)(R_t(\tau^i) - b_t).$$

Compute the prediction loss gradient:

$$\nabla_{\boldsymbol{\theta}} \ell_2 = \frac{1}{M} \sum_{i=1}^{M} \sum_{t=1}^{T_i} \nabla_{\boldsymbol{\theta}} \left\| W_{pred}(\mathbf{q}_t^i \otimes \mathbf{a}_t^i) - \mathbf{o}_t^i \right\|^2.$$

Normalize gradients $\nabla_{\boldsymbol{\theta}} \ell_j = \text{NORMALIZE}(\boldsymbol{\theta}, \ell_j)$, in (8).
Compute joint loss gradient as in (5):
$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \alpha_1 \nabla_{\boldsymbol{\theta}} \ell_1 + \alpha_2 \nabla_{\boldsymbol{\theta}} \ell_2$.
Update policy parameters: $\boldsymbol{\theta}^n = \text{ADAM}(\boldsymbol{\theta}^{n-1}, \nabla_{\boldsymbol{\theta}} \mathcal{L}, \eta)$
Output: Return $\boldsymbol{\theta}^n = (\boldsymbol{\theta}^n_{\text{PSR}}, \boldsymbol{\theta}^n_{\text{re}}, \eta)$.

kernel regression model (ridge regression on RFF features) to compute $A_t \equiv \mathbb{E}[\boldsymbol{\psi}_t^o \otimes \boldsymbol{\psi}_t^a \mid h_t]$ and another model to compute $B_t \equiv \mathbb{E}[\boldsymbol{\psi}_t^a \otimes \boldsymbol{\psi}_t^a \mid h_t]$. Then, for each time step we can compute

$$\bar{\mathbf{q}}_t = A_t (B_t + \lambda I)^{-1}$$

and then we can project these values using PCA. Computation of $\bar{\mathbf{p}}_t$ can be done in a similar fashion.

Then, we can learn W_{ext} through linear regression, which is referred to as stage 2 (S2) regression. Having learned the RFFPSR we can estimate the state $\mathbf{q}_t$ and set up a regression problem $o_t \approx W_{\text{pred}}(\mathbf{q}_t \otimes \phi_t^a)$ to learn prediction weights W_{pred}.

10. RPSP-network optimization algorithms

For clarity we provide the pseudo-code for the joint and alternating update steps defined in the $UPDATE$ step in Algorithm 1, in section §5. We show the joint VRPG update step in Algorithm 2, and the alternating (Alternating Optimization) update in Algorithm 3.

11. Comparison to RNNs with LSTMs/GRUs

RPSP-networks and RNNs both define recursive models that are able to retain information about previously observed inputs. BPTT for learning predictive states in PSRs bears many similarities with BPTT for training hidden states in LSTMs or GRUs. In both cases the state is updated via a series of alternate linear and non-linear transformations. For predictive states the linear transformation $\mathbf{p}_t = W_{\text{ext}}\ \mathbf{q}_t$ represents the system prediction: from expectations over futures $\mathbf{q}_t$ to expectations over extended features $\mathbf{p}_t$. The non-linear transformation, in place of the usual activation functions (tanh, ReLU), is replaced by f_{cond} that conditions on the current action and observation to update the expectation of the future statistics $\mathbf{q}_{t+1}$ in (2). It is worth noting

Algorithm 3 UPDATE (Alternating Optimization)

Input: $\boldsymbol{\theta}^{n-1}$, trajectories $\mathcal{D} = \{\tau^i\}_{i=1}^M$.
Estimate a linear baseline $b_t = \mathbf{w}_b^\top \mathbf{q}_t$, from the expected reward-to-go function for the batch $\mathcal{D}$:

$$\mathbf{w}_b = \arg\min_{\mathbf{w}} \left\| \frac{1}{TM} \sum_{i=1}^{M} \sum_{t=1}^{T_i} R_t(\tau_t^i) - \mathbf{w}^\top \mathbf{q}_t \right\|^2.$$

Update $\boldsymbol{\theta}_{\text{PSR}}$ using the joint VRPG loss gradient in (5):
$\boldsymbol{\theta}^n_{\text{PSR}} \leftarrow \text{UPDATE VRPG}(\boldsymbol{\theta}^{n-1}, \mathcal{D})$.
Compute descent direction for TRPO update of $\boldsymbol{\theta}_{\text{re}}$:
$v = H^{-1} g$, where

$$H = \nabla^2_{\boldsymbol{\theta}_{\text{re}}} \sum_{i=1}^{M} D_{KL}\left(\pi_{\boldsymbol{\theta}^{n-1}}(\mathbf{a}_t^i | \mathbf{q}_t^i) \mid \pi_{\boldsymbol{\theta}}(\mathbf{a}_t^i | \mathbf{q}_t^i)\right),$$

$$g = \nabla_{\boldsymbol{\theta}_{\text{re}}} \frac{1}{M} \sum_{i=1}^{M} \sum_{t=1}^{T_i} \frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}_t^i | \mathbf{q}_t^i)}{\pi_{\boldsymbol{\theta}^{n-1}}(\mathbf{a}_t^i | \mathbf{q}_t^i)} (R_t(\tau^i) - b_t).$$

Determine a step size η through a line search on v to maximize the objective in (7) while maintaining the constraint.
$\boldsymbol{\theta}^n_{\text{PSR}} \leftarrow \boldsymbol{\theta}^{n-1}_{\text{PSR}} + \eta v$
Output: Return $\boldsymbol{\theta}^n = (\boldsymbol{\theta}^n_{\text{PSR}}, \boldsymbol{\theta}^n_{\text{re}})$.

that these transformations represent non-linear state updates, as in RNNs, but where the form of the update is defined by the choice of representation of the state. For Hilbert Space embeddings it corresponds to conditioning using Kernel Bayes Rule. An additional source of linearity is the representation itself. When we consider linear transformations W_{pred} and W_{ext} we refer to transformations between kernel representations, between Hilbert Space Embeddings.

PSRs also define computation graphs, where the parameters are optimized by leveraging the states of the system. Predictive states can leverage history like LSTMs/GRUs, PSRs also have memory, since they learn to track in the Reproducing Kernel Hilbert Space (RKHS) space of distributions of future observations based on past histories. PSRs provide the additional benefit of being well-defined as conditional distributions of observed features and could be trained based on that definition. For this reason, RPSPs have a statistically driven form of initialization, that can be obtained using a moment matching technique, with good theoretical guarantees (Hefny et al., 2018).

12. Additional Experiments

In this section, we investigate the effect of using different variants of RPSP networks, we test against a random initialization of the predictive layer, and provide further experimental evidence for baselines.

RPSP optimizers: Next, we compare several RPSP variants for all environments. We test the two RPSP variants, joint and alternate loss with predictive states and with augmented predictive states (**+obs**). The first variant is the standard "vanilla" RPSP, while the second variant is an RPSP with augmented state representation where we concatenate the previous window of observations to the predictive state (+obs). We provide a complete comparison

of RPSP models using augmented states with observations for all environments in Figure 9. We compare with both joint optimization (VRPG+obs) and an alternating approach (Alt+obs). Extended predictive states with a window of observations ($w = 2$) provide better results in particular for joint optimization. This extension might mitigate prediction errors, improving information carried over by the filtering states.

Finite Memory models: Next, we present all finite memory models used as baselines in §6. Figure 7 shows finite memory models with three different window sizes $w = 1, 2, 5$ for all environments. We report in the main comparison the best of each environment ($FM2$ for Walker, Swimmer, Cart-Pole, and $FM1$ for Hopper).

GRU baselines: In this section we report results obtained for RNN with GRUs using the best learning rate $\eta = 0.01$. Figure 8 shows the results using different number of hidden units $d = 16, 32, 64, 128$ for all the environments.

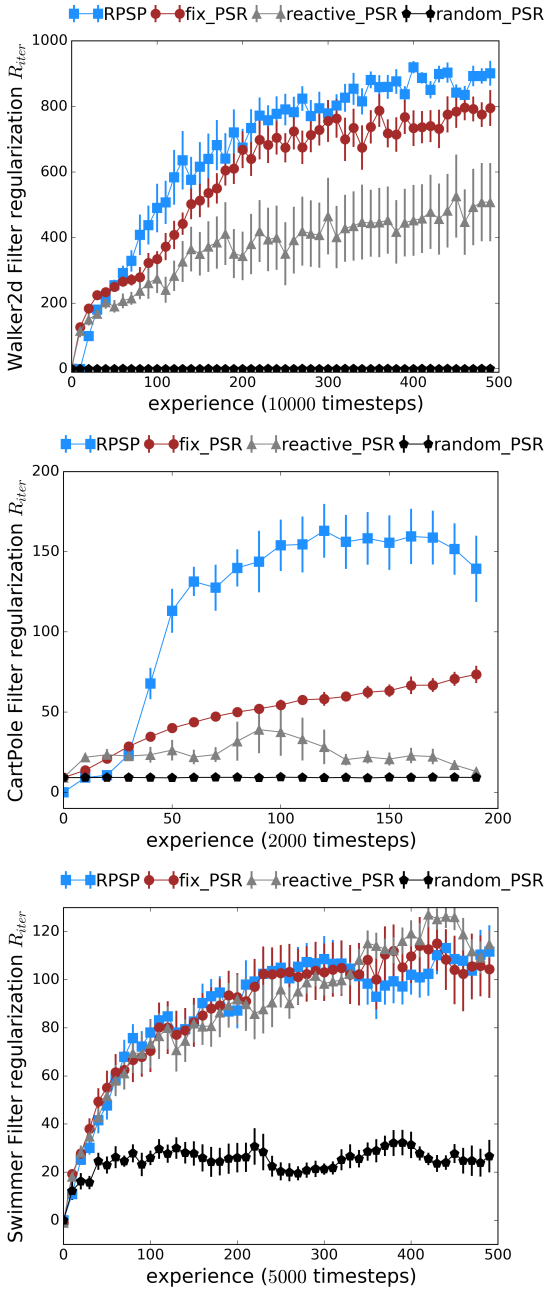


Figure 5: Predictive filter regularization effect for Walker2d, CartPole and Swimmer environments. RPSP with predictive regularization (RPSP:blue), RPSP with fixed PSR filter parameters (fix_PSR:red), RPSP without predictive regularization loss (reactive_PSR: grey).

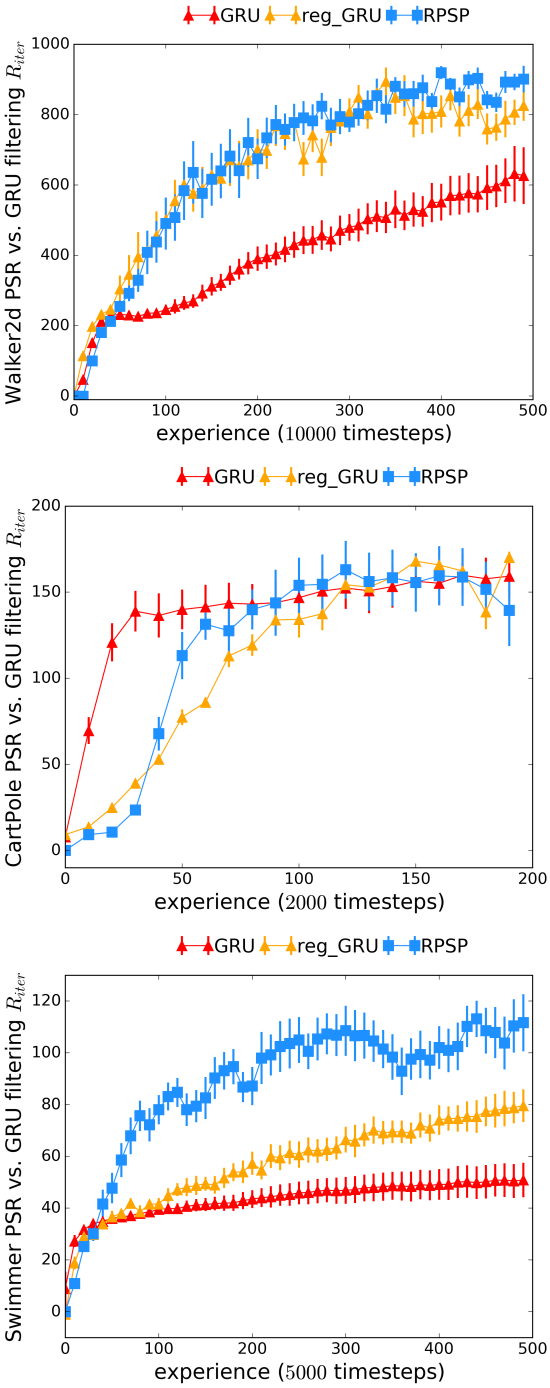


Figure 6: GRU vs. RPSP filter comparison for other Walker and CartPole environments. GRU filter without regularization loss (GRU:red), GRU filter with regularized predictive loss (reg_GRU: yellow), RPSP (RPSP:blue)

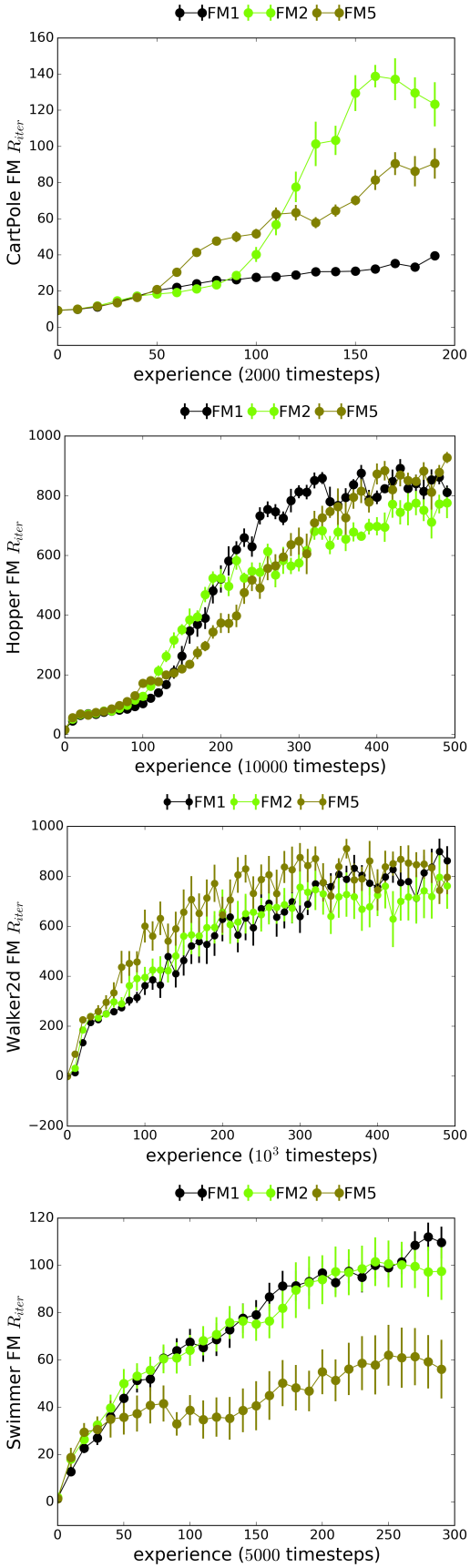


Figure 7: Empirical expected return using finite memory models of $w = 1$ (black), $w = 2$ (light green), $w = 5$ (brown) window sizes. (top-down) Walker, Hopper, CartPole, and Swimmer.

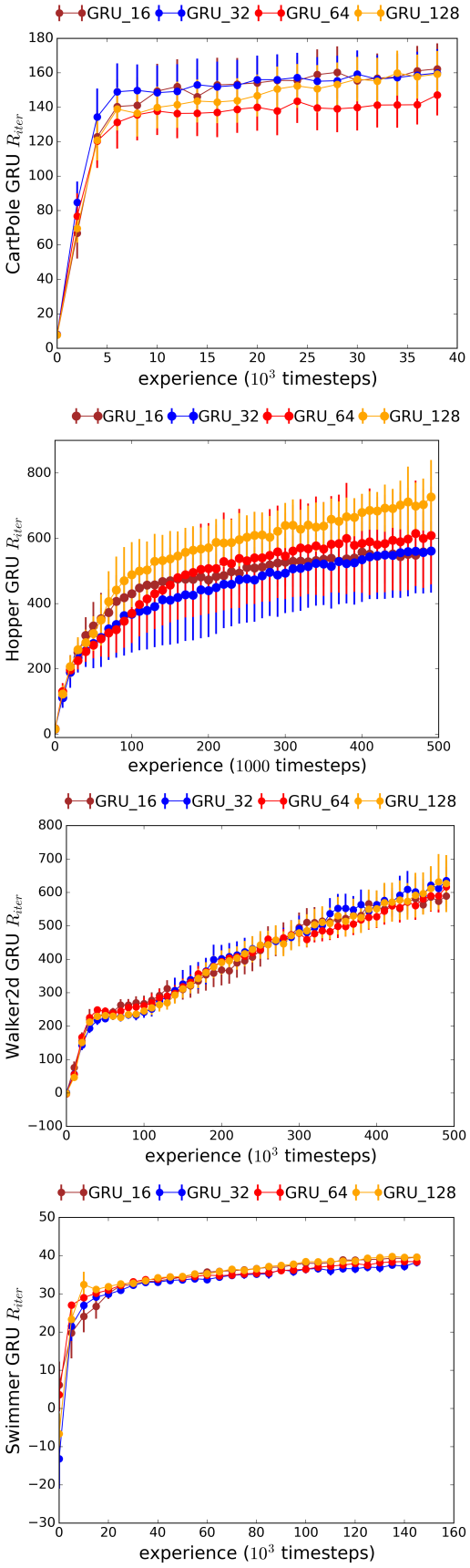


Figure 8: Empirical expected return using RNN with GRUs $d = 16$ (green), $d = 32$ (blue), $d = 64$ (red) and $d = 128$ (yellow) hidden units. (top-down) Walker, Hopper, CartPole, and Swimmer.

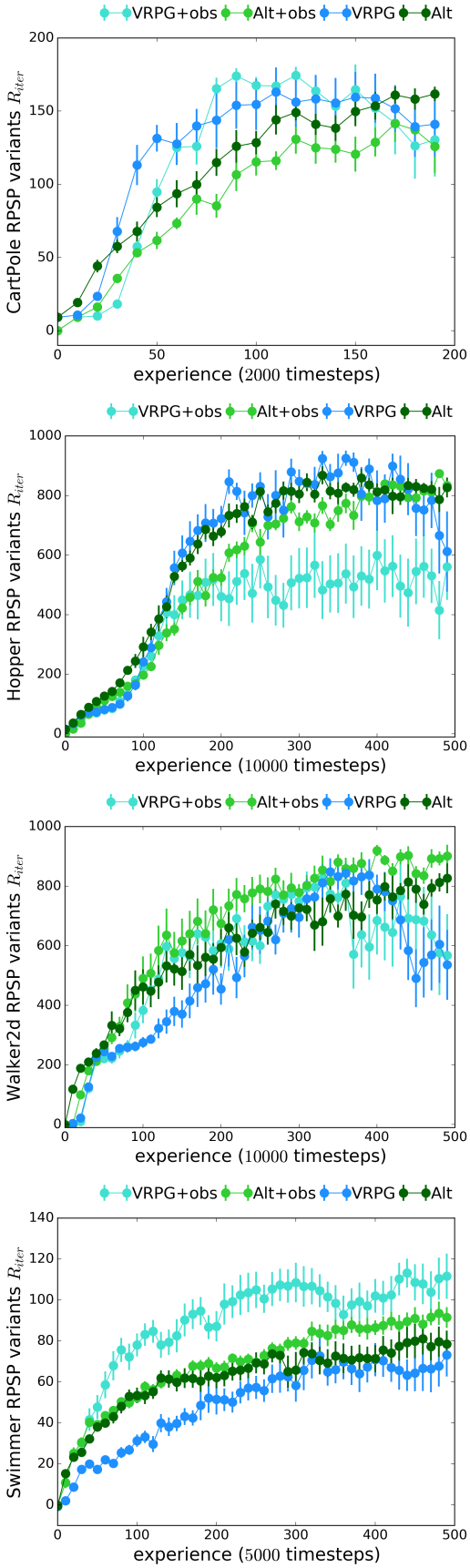


Figure 9: Reward over iterations for RPSP variants over a batch of $M = 10$ trajectories and 10 trials.