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.

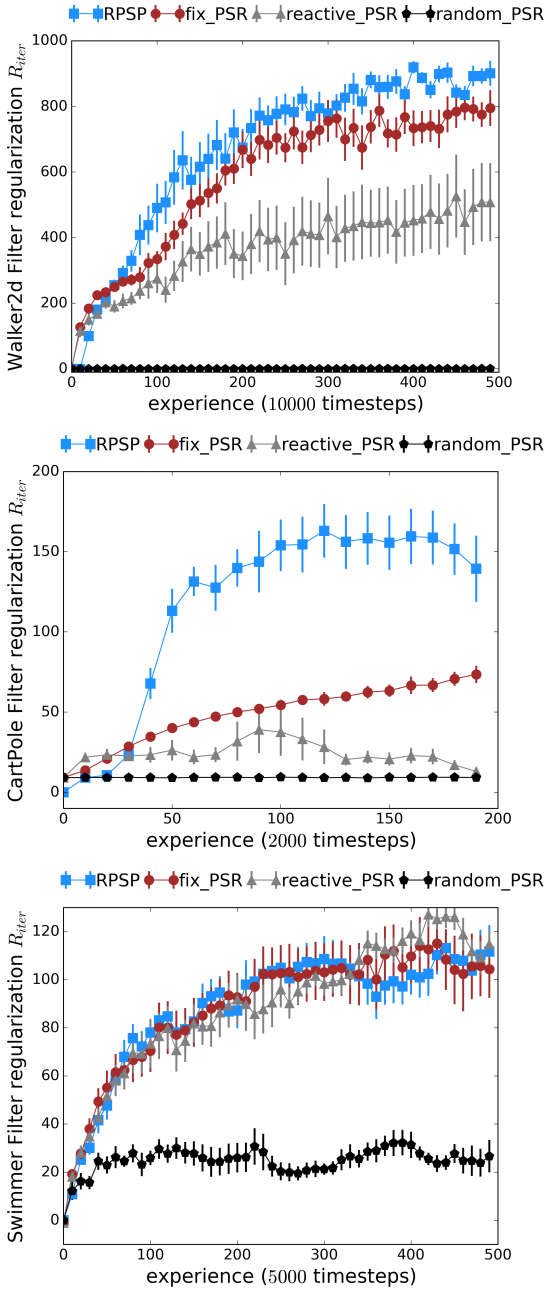

Figure 5: Predictive filter regularization effect for Walker2d, CartPole and Swimmer environments. RPSP with predictive regularization (RPSP:blue), RPSP with fixed PSR filter parameters (fix_PSR:red), RPSP without predictive regularization loss (reactive_PSR: grey).

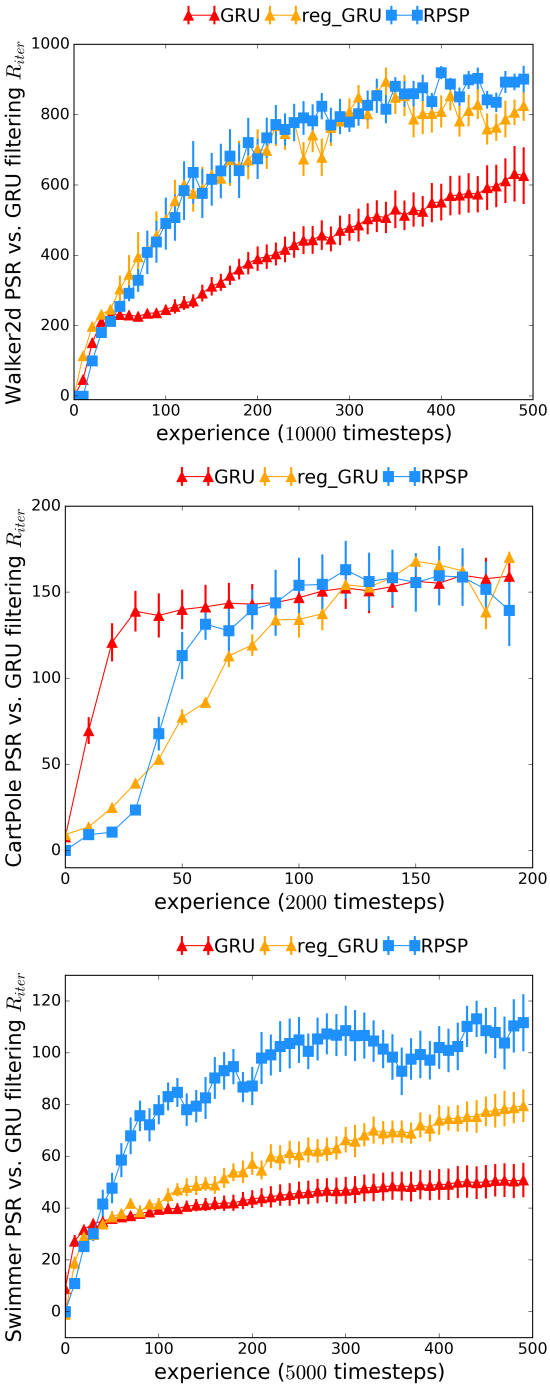

Figure 6: GRU vs. RPSP filter comparison for other Walker and CartPole environments. GRU filter without regularization loss (GRU:red), GRU filter with regularized predictive loss (reg_GRU: yellow), RPSP (RPSP:blue)

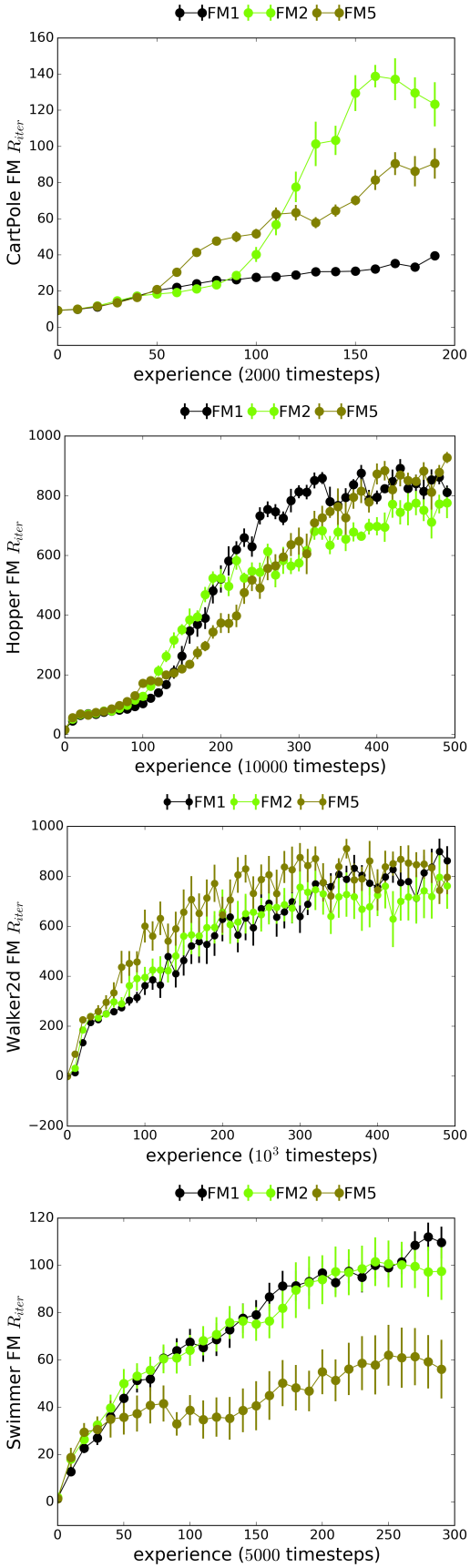

Figure 7: Empirical expected return using finite memory models of $w = 1$ (black), $w = 2$ (light green), $w = 5$ (brown) window sizes. (top-down) Walker, Hopper, CartPole, and Swimmer.

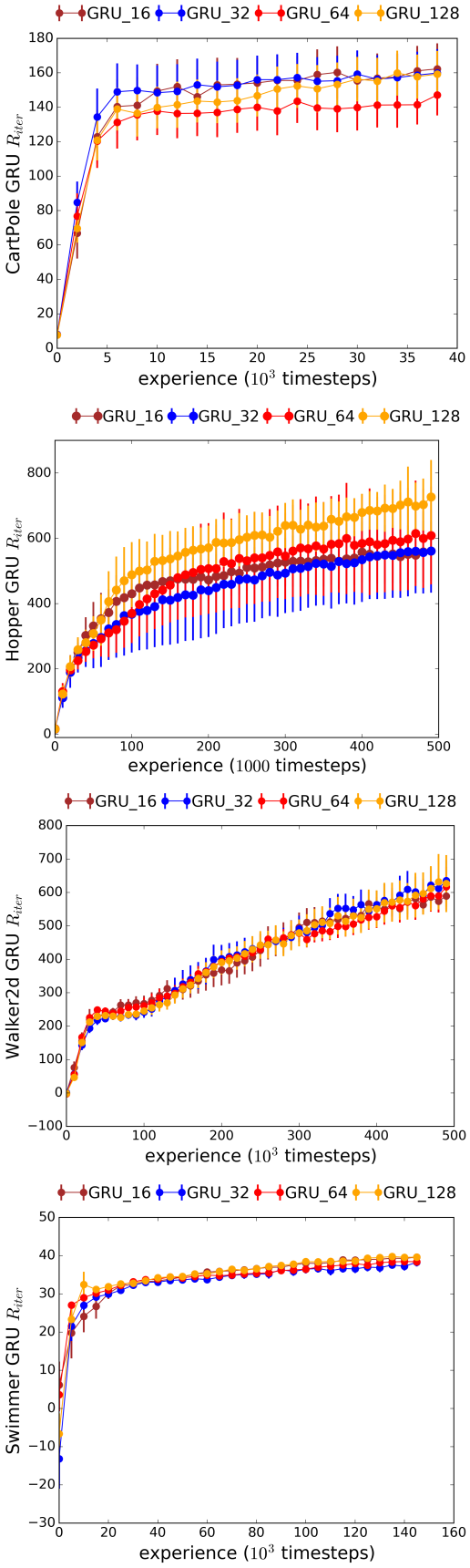

Figure 8: Empirical expected return using RNN with GRUs $d = 16$ (green), $d = 32$ (blue), $d = 64$ (red) and $d = 128$ (yellow) hidden units. (top-down) Walker, Hopper, CartPole, and Swimmer.

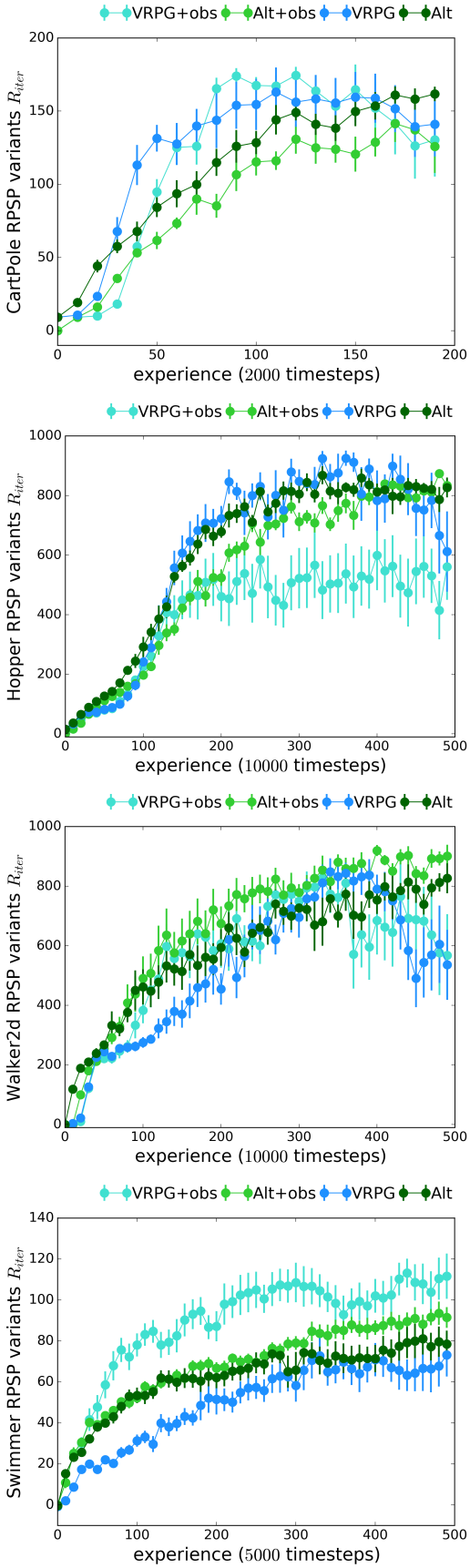

Figure 9: Reward over iterations for RPSP variants over a batch of $M = 10$ trajectories and 10 trials.