RECURRENT LEARNING FOR REINFORCEMENT LEARNING

Pierre Thodoroff[*]
McGill University

Nishanth Anand[*]
McGill University

Lucas Pagé-Caccia
McGill University

Joelle Pineau
McGill University
Facebook AI Research

Doina Precup
McGill University

ABSTRACT

In sequential modelling, exponential smoothing is one of the most widely used techniques to maintain temporal consistency in estimates. In this work, we propose Recurrent Learning, a method that estimates the value function in reinforcement learning using exponential smoothing *along the trajectory*. We establish its asymptotic convergence properties under some smoothness assumption on the reward. The proposed algorithm yields a natural way to learn a state dependent emphasis function that selectively learns to emphasize or ignore states based on trajectory information. We demonstrate the potential for this selective updating on a partially observable domain and several continuous control tasks.

1 INTRODUCTION

Reinforcement Learning is a mathematical framework designed to model sequential decision making. They are widely applied on a range of tasks but they suffer from several issues. In particular, we describe two issues that Recurrent Learning attempts to mitigate. The first one considers the *variance of the value estimates along the trajectory*. Most algorithms in Reinforcement Learning estimate the value at every time step without necessarily *explicitly* enforcing temporal coherence nor considering previous estimates. This can lead to erratic and temporally inconsistent behaviors, particularly in tabular and discrete settings. The second issue considers the limited capacity of the brain to process information and make decision. Bounded rationality argues that human brain has a limited capacity to learn and can't store all the information. In this work we argue that the capacity to ignore or emphasize chosen state is a key component for the success of decision making algorithms.
Exponential smoothing (1) is one of the most widely used technique to reduce the variance of point estimates using *previous observations*. In time series data it is used to approximate the mean of a stream of data. This kind of smoothing has various names (2; 3) depending on the field it is used in. Most quantities in reinforcement learning (Q values, state values, actions, etc) are estimated as point estimate ignoring previous estimates from the trajectory. In this work, we use exponential smoothing in reinforcement learning on the trajectory. In other words, we propose to smooth the estimate of the present state using the estimates of past states. Intuitively, states that are temporally close to each other should have similar value.
However, exponential averaging can be biased if a sharp change(non-stationarity), like falling off a cliff, is encountered along the trajectory. To alleviate this issue a common technique used is to set β_t, the exponential smoothing factor, as a state or time dependent. It is possible to view LSTM (4) and GRU (5) as several non-linear exponential smoothing functions. The key ingredient is the gating mechanism(state dependent β_t) that ignores the information allowing the cell to focus only on important information. In this work we explore a new method that attempts to learn a state dependent smoothing factor β. We show how it relates to existing methods and exploit similar properties such as emphatic TD (6; 7). The contributions of the paper are as follows:

- Propose a new way to estimate value function in reinforcement learning by exploiting the estimates along the trajectory.

[*]Equal contribution

- Provide asymptotic convergence guarantee in the tabular setting under some smoothness assumption.
- Derive a learning rule for a state dependent β.
- Perform a set of experiments in tabular and continuous settings to evaluate its strengths and weaknesses.

2 TECHNICAL BACKGROUND

A Markov Decision Process (MDP), as defined in (8), consists of a discrete set of states $\mathcal{S}$, a transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$, and a reward function $r : \mathcal{S} \times \mathcal{A} \mapsto \mathsf{R}$. On each round t, the learner observes current state $s_t \in \mathcal{S}$ and selects action $a_t \in \mathcal{A}$, after which it receives reward $r_t = r(s_t, a_t)$ and moves to a new state $s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t)$. We define a stationary policy π as a probability distribution over actions conditioned on states $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$, such that $a_t \sim \pi(\cdot|s_t)$. When performing policy evaluation, the goal is to find the optimal value function V^π that estimates the discounted expected return of policy π at a state $s \in \mathcal{S}$, $V^\pi(s) = \mathsf{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s]$, with discount factor $\gamma \in [0, 1)$. In this paper we only consider policy evaluation and simplify the notation: $r(s_t) = r(s_t, a_t)$.

In practice V^π is approximated using Monte Carlo rollouts (9) or TD methods (10). In reinforcement learning the aim is to find a function $V_\theta : \mathsf{S} \to \mathsf{R}$ parametrized by θ that approximates V^π. We can fall back to the tabular setting by representing the states in a one hot vector form with $\theta \in \mathsf{R}^{|\mathbb{S}|}$. The goal is to find a set of parameters θ that minimizes the squared loss:

$$\mathcal{L}(\theta) = \mathop{\mathsf{E}}_{\pi}[(V^\pi - V_\theta)^2] \tag{1}$$

which yields the following update by taking the derivative with respect to θ:

$$\theta_{t+1} = \theta_t + \alpha(V^\pi(s_t) - V_{\theta_t}(s_t))\nabla_{\theta_t} V_\theta(s_t) \tag{2}$$

where α is a learning rate.

3 RECURRENT LEARNING IN REINFORCEMENT LEARNING

In this work, we propose to estimate the value function using the estimates along the trajectory. The value of a state s at time step t is given by:

$$V_\theta^\beta(s_t) = \beta_t V_\theta(s_t) + (1 - \beta_t)(V_\theta^\beta(s_{t-1})) \tag{3}$$

where V_θ^β is an exponential smoothing of V_θ. V_θ^β is a function parametrized by θ and $\beta \in [0, 1]$. We consider β to be fixed for each state in this section and relax the assumption later. For a given set of state dependant $\beta(s_t) = \beta_t$, the goal is to find a set of parameters θ minimizing,

$$\min_\theta \mathcal{L}(\theta) = \min_\theta \mathop{\mathsf{E}}_{\pi}[(V^\pi - V_\theta^\beta)^2] \tag{4}$$

In contrast to traditional methods that attempts to minimize Eq. 1 we explicitly enforce temporal consistency by finding θ that minimizes Eq. 4. By taking the derivative with respect to Eq. 4 we obtain the following update:

$$\theta = \theta + \alpha\delta_t \nabla_\theta V_\theta^\beta(s_t) \tag{5}$$

where $\delta_t = V^\pi(s_t) - V_\theta^\beta(s_t)$. The gradients of V_θ^β can be expressed in a recursive form:

$$\nabla_\theta V_\theta^\beta(s_t) = \beta_t \nabla_\theta V_\theta + (1 - \beta_t)\nabla_\theta V_\theta^\beta(s_{t-1}) \tag{6}$$

For computational reason, it is possible to approximate the gradient $\nabla_\theta V_\theta^\beta(s_t)$ using a recursive *eligibility* trace:

$$e_t = \beta_t \nabla_\theta V_\theta(s_t) + (1 - \beta_t)e_{t-1} \tag{7}$$

This technique is computationally inexpensive compared to the recursive backpropogation as the gradient from the past is accumulated. But this kind of an update is biased. This bias comes from

the fact that the true gradient obtained by applying the chain rule at future step is different than the trace if the parameters are updated at each time step. This aspect is well studied in (11) and (12). We now present Recurrent Temporal Difference(RTD(0)) in Algorithm 1. The algorithm is referred to as RTD(0) because we have a one step target. This algorithm is compatible with any targets such as Monte Carlo, n-step and even a λ return. In fact, there exists an important relationship between λ and β. β can choose to ignore a state if the past is different from the future. We could get a better estimate of the future using λ returns when compared against one step methods. Hence, a need for long horizon to learn accurate β. The convergence proof is presented in the appendix 6.1.

Algorithm 1 Recurrent Temporal Difference, RTD(0)

Input: π,β,γ,θ
Initialize: $V_\theta^\beta(s_0) = V_\theta(s_0)$ and $e_0 = \nabla_\theta V_\theta(s_0)$
Output: θ
for all t **do**
  Choose $a \sim \pi(s_t)$
  Take action a, observe $r(s_t), s_{t+1}$
  $V_\theta^\beta(s_t) = \beta_t V_\theta(s_t) + (1-\beta_t)(V_\theta^\beta(s_{t-1}))$
  $e_t = \beta_t \nabla_\theta V_\theta(s_t) + (1-\beta_t)e_{t-1}$
  $\delta_t = r(s_t) + \gamma V_\theta(s_{t+1}) - V_\theta^\beta(s_t)$
  $\theta = \theta + \alpha \delta_t e_t$
end for

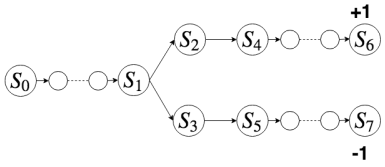
In practice, $V_\theta^\beta(s_0)$ is initialized with the value of the first state $V_\theta(s_0)$. We can understand our algorithm better when we intuitively interpret its effect in the *extreme cases* ($\beta = \{0, 1\}$). We consider TD(0) in a tabular setting for simplicity. First, consider a state s_t where $\beta(s_t) = 0$. The value of this state is frozen at the initialization point and is never updated. This is because the trace as defined in Eq. (7) is $e_t = e_{t-1}$ making such a state *ineligible* for the update. The error received at this state is used to update the previous states as per their *eligibility* in e_{t-1}. This means that for a state s_t with $\beta(s_t) = 1$, $V_\theta(s_t)$ is updated at every time step $t + n$ until another state with $\beta(s_{t+n}) = 1$ is encountered.

State dependent β: State dependent β are responsible for the success of several techniques. For instance, the success of LSTM (4) can be attributed to the gating(state dependent smoothing) mechanism. In this section we consider β_ω where β_t is estimated with a set of parameters ω. There is a natural way to estimate β in our formulation $\min_\beta \mathsf{E}_\pi \left\| V^\pi - V_{\theta,\omega}^\beta \right\|_2$. In practice, this compares the estimate $V_\theta(s_t)$ and $V_{\theta,\omega}^\beta(s_{t-1})$ to V^π and gives more weight to the one that is closer among the two. This differs from the greedy approach(13) to set λ as we don't explicitly need to consider the variance of the return.

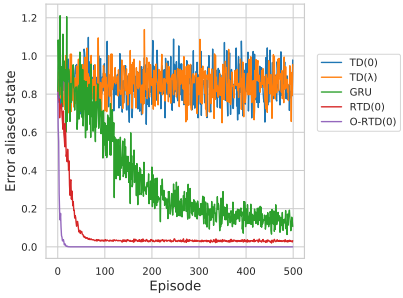
Adjusting for the reward: In practice many environments in reinforcement learning have either a constant negative reward or a constant positive reward at every time step. In order to account for those rewards we propose an alternative formulation where V_θ^β accounts for the reward that was just seen, $V_\theta^\beta(s_t) = \beta_t V_\theta(s_t) + (1-\beta_t)(V_\theta^\beta(s_{t-1}) - r_{t-1})$. The decision on the choice of formulation to use will depend on the environment considered.

4 EXPERIMENTS

We consider the simple chain MDP described in figure 1(a) to demonstrate our method. This MDP has three chains connected together to form a *Y*. Each of the 3 chains (left of S_1, right of S_2, right of S_3) is made up of a sequence of states. The agent starts at S_0 and navigates through the chain. At the intersection, S_1, there is a 0.5 probability to go top or bottom. The chain on the top has a reward of $+1$ at the last state and the chain on the bottom has a reward of -1. Every other transition has a reward of 0, unless specified otherwise.



(a) Simple chain MDP. The dotted line in the figure indicates multiple states in between shown states.



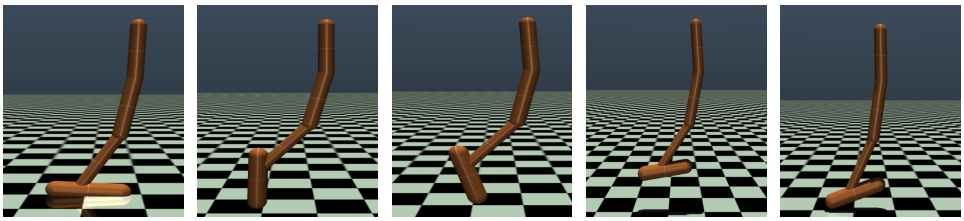
(b) TD versus Recurrent TD on aliased Y-chain

Figure 1: Toy MDP and POMDP performance

4.1 Partially observable setting

We explore the capacity of recurrent learning to solve a partially observable task. In particular, we consider the case where some states are aliased (share a common representation). The representation of the state following S_4 and S_5 in figure 1 (a) is aliased. The goal of this environment is to correctly estimate the value of the aliased state $V^\pi(S_4) = 0.81, V^\pi(S_5) = -0.81$(due to the discounting and the length of each chain being 5). When TD(λ) is used to estimate the values for these states, the values of states S_4 and S_5 is close to 0 as the reward at the end of the chain is $+1$ and -1. However, when learning β, recurrent learning achieve almost no error in the estimate of the aliased state as illustrated in figure 1 (b). To understand this result the first thing to realize is that $\beta = 0$ on the aliased state as the previous values along the trajectory are a better estimate of the future along the same chain compared to the actual estimate of the aliased state. As $\beta \to 0$, $V^\beta_\theta(S_4)$ and $V^\beta_\theta(S_5)$ tends to rely on the previous estimate $V(S_2)$ and $V(S_3)$ which are accurate. We see that learning to ignore certain states can sometimes be enough to solve correctly an aliased tasks in contrast to a traditional POMDP method that would attempt to infer the belief state. The results displayed in figure 1(b) are averaged over 20 random seeds. The learning rate used is 0.05, $\gamma = 0.9$ and $\lambda = 0.9$.

4.2 Deep Reinforcement Learning



Phase 1: high β | Phase 2: low β | Phase 3: high β | Phase 4: low β | Phase 1: high β

Figure 2: Visual illustration of the cyclical behavior of β on Hopper

In this experiment, we modify the critic of Proximal Policy Optimization (PPO) (14) to use recurrent learning. We modify the critic to estimate the value function parametrized by θ using recurrent learning. We add a separate network parametrized by ω (same architecture as PPO) to learn a state dependant β. The loss for learning β is $\min_{\beta,\omega} \mathsf{E}_\pi(V^\lambda_\theta - V^\beta_\theta)^2)$ where the target V^λ_θ is the generalized advantage function (15). Using an automatic differentiation library (Pytorch (16)) we differentiate the loss through the modified estimate to learn the θ and ω. The hyperparameters of PPO are not modified. Due to the batch updates in PPO, obtaining the trajectory information to create the computational graph can be costly. As a result, we cut the backpropagation after N timesteps in a similar manner to truncated backpropagation through time (12). The learning rate for the β network, L1 regularization coefficient of β and number of backpropagation steps were obtained using hyperparameter search. We used a truncated backprop of $N = 5$ in our experiments as we found

no empirical improvements for $N = 10$. The motivation to regularize β to be sparse comes from bounded rationality as updating the value at key states is a desirable property. This parameter plays a similar role as the deliberation cost in option (17). The best performing set of parameters included a regularization parameter of 0.5 supporting this hypothesis. The performance reported are averaged over 20 different random seeds[1].

4.3 PERFORMANCE

The algorithm was tested on several games of the Mujoco suite (19). As demonstrated in the figure 3 in appendix, we observe an increase in performance on tasks - Swimmer, Hopper, Inverted Pendulum, Double Inverted Pendulum. The performances are averaged over 20 seeds and a confidence interval of 95% is used. Detailed plots of performance and β behaviour is found in appendix (6, 7, 8) The performances were found to be comparable on MountainCar and the detailed discussion is in appendix (5). One interesting finding in the figure 3 concerns inverted-pendulum in the context of generalization. Though the convergence is quick for PPO, severe performance drops (reward drop by more than 100) can be observed. We found that recurrent learning stabilizes the learning and suffer significantly less drops during training. We notice that on an average PPO will suffer from 5.91 drops below 900 points over 500k steps. In contrast, recurrent learning will only drop 3.53 times. This represent a 40% drop in *catastrophic forgetting* underlying the potential robustness of recurrent learning.

4.4 QUALITATIVE INTERPRETATION OF β

Hopper: At the end of training, we qualitatively analyze β through the trajectory and observe a cyclical behavior as shown in appendix 4. One intuitive way to look at β is: *if I were to give a different value to a state would that alter my policy significantly?* We observe an increase in β value when the agent has to take an important decision like jumping or landing. We see a decrease in β when the agent has trivial actions to perform. This pattern is illustrated in the figure 2. This behaviour is cyclic and repetitive and a video of the same can be found at the following link[2]. One surprising fact was that this behavior was obtained without any regularization on β. Similar behavior can be observed with regularization although the mean and variance of β diminishes.

5 DISCUSSIONS

β as an interest function: One interesting result of this work is that the β network learns to ignore some state without any restrictions imposed on it. It does so in order to reduce the variance. Furthermore, this β can be interpreted as an *interest* function. In reinforcement learning, having access to a function quantifying the *interest* (7) of a state can be helpful. For example, one could decide to explore from those states, prioritize experience replay based on those states. Further work can be done to study how β may impact performance. We also believe β can be related to the concepts of bottleneck state (20). Finally, the concept of interest state in recurrent learning aligns well with the notion of interest state for the λ return. Indeed bootstrapping on states with similar values than the one estimated will only result in variance. The most informative updates comes from bootstrapping on state with different value.

Recurrent learning for action: We have not explored the concept of recurrent learning for actions (Q values and policy gradient). This is a promising area as constraining actions to be temporally coherent is a natural prior to induce on a function approximator. This technique could also be interesting in the context of exploration as this could yield a *structured exploration*. Finally, this framework with learning β can be cast as a *vanilla* version of options (21).

Conclusion: This paper proposes a technique to address two important aspects of reinforcement learning algorithms namely - temporal coherence, and selective updating. First, we propose a new formulation of temporal difference. Then, we provide experiments to corroborate the application of

[1]The base code used to develop this algorithm can be found here (18)

[2]`https://youtu.be/0bzEcrxNwRw`

our method in a continuous control domain. Finally, we demonstrate interesting properties that result while we emphasize and de-emphasize updates on states during learning.

REFERENCES

[1] Everette S Gardner Jr. Exponential smoothing: The state of the art. *Journal of forecasting*, 4(1):1–28, 1985.

[2] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

[3] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[5] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[6] Richard S Sutton, A Rupam Mahmood, and Martha White. An emphatic approach to the problem of off-policy temporal-difference learning. *The Journal of Machine Learning Research*, 17(1):2603–2631, 2016.

[7] A Rupam Mahmood, Huizhen Yu, Martha White, and Richard S Sutton. Emphatic temporal-difference learning. *arXiv preprint arXiv:1507.01569*, 2015.

[8] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 1994.

[9] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction.

[10] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[11] Harm Seijen and Rich Sutton. True online td (lambda). In *International Conference on Machine Learning*, pages 692–700, 2014.

[12] Ronald J Williams and David Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. *Backpropagation: Theory, architectures, and applications*, 1:433–486, 1995.

[13] Martha White and Adam White. A greedy approach to adapting the trace parameter for temporal difference learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 557–565. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[14] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[15] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[16] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[17] Jean Harb, Pierre-Luc Bacon, Martin Klissarov, and Doina Precup. When waiting is not an option: Learning options with a deliberation cost. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[18] Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms. `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr`, 2018.

[19] Emanuel Todorov, Tom Erez, and Yuval 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.

[20] Naftali Tishby and Daniel Polani. Information theory of decisions and actions. pages 601–636, 2011.

[21] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[22] John N Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Machine learning*, 16(3):185–202, 1994.

[23] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[24] Peter Dayan. The convergence of td (λ) for general λ. *Machine learning*, 8(3-4):341–362, 1992.

[25] Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[26] Chia-Chun Hung, Timothy Lillicrap, Josh Abramson, Yan Wu, Mehdi Mirza, Federico Carnevale, Arun Ahuja, and Greg Wayne. Optimizing agent behavior over long time scales by transporting value, 2018.

[27] Pierre Thodoroff, Audrey Durand, Joelle Pineau, and Doina Precup. Temporal regularization for markov decision process. In *Advances in Neural Information Processing Systems*, pages 1782–1792, 2018.

[28] Zhongwen Xu, Joseph Modayil, Hado P van Hasselt, Andre Barreto, David Silver, and Tom Schaul. Natural value approximators: Learning when to trust past estimates. In *Advances in Neural Information Processing Systems*, pages 2120–2128, 2017.

6 APPENDIX

6.1 CONVERGENCE PROOF

We consider the following assumptions to prove convergence. The first assumption deals with the ergodic nature of the Markov chain. It is a common assumption in theoretical reinforcement learning that guarantees infinitely many visits to all states thereby avoiding chains with transient states.

Assumption 1. *The Markov chain is ergodic.*

The second assumption concerns the relative magnitude of the maximum and the minimum reward. It is a key element that allow us to bound the magnitude of the regularization term.

Assumption 2. *We define R_{max} and R_{min} as the maximum and minimum reward in a MDP. The relative magnitude between them is bounded by a factor of D such that:*

$$\begin{aligned} DR_{max} &\leq R_{min} \\ D &> \gamma \\ R_{min} &\geq 0 \end{aligned} \tag{8}$$

where $D \in (0.5, 1]$ is a constant to be defined based on γ.

This is the most restrictive assumption of the proof. Although assumption 2 is restrictive in theory we never observed a divergence in tabular and deep RL settings that we considered.

The key component of the proof is to control the magnitude of the term $\Delta_t(s_i) = (1 - C_t(s_i))(V_\theta(s_i) - \widetilde{V_{\theta t}}(s_i))$. As the eligibility of this update gets smaller the magnitude of the term gets bigger. This suggests that not updating certain states whose eligibility is less than the threshold *C* can help mitigate biased updates. Depending on the values of γ and D we may need to set a threshold C to guarantee convergence.

Theorem 1. *Let's define $V_{\mathsf{max}} = \frac{R_{\mathrm{max}}}{1-(\gamma+(1-D))}$ and $V_{\mathsf{min}} = \frac{R_{\mathrm{min}}}{1-(\gamma-(1-D))}$. If the following holds*

- *Let X be the set of V_θ functions such that $\forall s \in \mathsf{S} \quad V_{\mathsf{min}} \leq V_\theta(s) \leq V_{\mathsf{max}}$. We assume the functions are initialized in X.*
- *For a given D and γ we select C such that $(1 - C)(V_{\mathsf{max}} - V_{\mathsf{min}}) \leq (1 - D)V_{\mathsf{min}}$*

then $\mathcal{T}^\beta : X \to X$ is a contractive operator.

Proof. The first step is to prove that $\mathcal{T}^\beta$ maps to itself for any noisy update $\widetilde{\mathcal{T}^\beta}$. From 2) we know that $(1 - C)V_{\mathsf{max}} - V_{\mathsf{min}} < DV_{\mathsf{min}} \leq DV_{\mathsf{max}}$ we can then deduce that

$$\begin{aligned} \widetilde{\mathcal{T}^\beta} V_\theta(s) &\leq R_{\mathsf{max}} + \gamma V_{\mathsf{max}} + (1 - C)(V_{\mathsf{max}} - V_{\mathsf{min}}) \\ &\leq R_{\mathsf{max}} + (\gamma + (1 - D))V_{\mathsf{max}} \\ &\leq V_{\mathsf{max}} \end{aligned} \tag{9}$$

and

$$\begin{aligned} \widetilde{\mathcal{T}}^\beta V_\theta(s) &\geq R_{\mathsf{min}} + \gamma V_{\mathsf{min}} + (1 - C)(V_{\mathsf{min}} - V_{\mathsf{max}}) \\ &\geq R_{\mathsf{min}} + (\gamma - (1 - D))V_{\mathsf{min}} \\ &\geq V_{min} \end{aligned} \tag{10}$$

The next step is to show that $\mathcal{T}^\beta$ is a contractive operator:

$$\begin{aligned} &\left\| \mathcal{T}^\beta V - \mathcal{T}^\beta U \right\|_\infty \\ &\leq \max_{s,s'} \mathop{\mathsf{E}}_{\pi}[\gamma V(s) + \Delta^V(s') - (\gamma U(s) + \Delta^U(s'))] \\ &\leq \max_{s,s'} \mathop{\mathsf{E}}_{\pi}[\gamma(V(s) - U(s)) + (1 - D)(V(s') - U(s'))] \\ &\leq \max_{s} \mathop{\mathsf{E}}_{\pi}[((1 - D) + \gamma)(V(s) - U(s))] \\ &\leq ((1 - D) + \gamma) \left\| V - U \right\|_\infty \end{aligned} \tag{11}$$

and from the assumption we know that $(1 - D) + \gamma < 1$. □

6.2 SELECTING C

To select C based on γ and D it suffice to solve analytically for:

$$\begin{aligned}
&(1-C)(V_{\max}-V_{\min}) \leq (1-D)V_{\min} \\
&\equiv (1-C)\frac{R_{\max}}{1-(\gamma+(1-D)} \leq ((1-D)+(1-C))\frac{R_{\min}}{1-(\gamma-(1-D)} \\
&\equiv \frac{(1-C)(1-(\gamma-(1-D)))}{(1-(\gamma+(1-D))((1-D)+(1-C)}R_{\max} \leq R_{\min} \\
&\equiv \frac{D(1-C)(1-(\gamma-(1-D)))}{(1-(\gamma+(1-D))((1-D)+(1-C)}R_{\min} \leq R_{\min}
\end{aligned} \tag{12}$$

which is satisfied only if:

$$\frac{D(1-C)(1-(\gamma-(1-D)))}{(1-(\gamma+(1-D))((1-D)+(1-C)} \leq 1 \tag{13}$$

As an example for $D = 0.8$ and $\gamma = 0.5$ any $C \geq 0.33$ satisfies this inequality.

6.3 ASSUMPTION ASYNCHRONOUS STOCHASTIC APPROXIMATION

We now discuss the assumptions of theorem 3 in (22)

Assumption 1: Allows for delayed update that can happen in distributed system for example. In this algorithm all V_θ's are updated at each time step t and is not an issue here.

Assumption 2: As described by (22) assumption 2 allows for the possibility of deciding whether to update a particular component x_i at time t, based on the past history of the process. This assumption is defined to accommodate for ϵ-greedy exploration in Q-learning. In this paper we only consider policy evaluation hence this assumptions holds.

Assumption 3: The learning rate of each state $s \in \mathsf{S}$ must satisfy Robbins Monroe conditions such that there exists $C \in \mathsf{R}$:

$$\begin{aligned}
\sum_{i=0}^{\infty} \alpha_t(s)e_t(s) = \infty \quad \text{w.p.1} \\
\sum_{i=0}^{\infty} (\alpha_t(s)e_t(s))^2 \leq C
\end{aligned} \tag{14}$$

This can be verified by assuming that each state gets visited infinitely often and an appropriate decaying learning rate based on $\#_s$ (state visitation count) is used (linear for example).

Assumption 5: This assumption requires $\mathcal{T}$ to be a contraction operator. This has been proven in theorem 1 of this paper.

6.4 DERIVATION OF β UPDATE RULE

As an example, we derive an analytic form of gradient of $\beta_\omega = \sigma(\omega_{s_t})$ where ω_{s_t} is a scalar. Taking the derivative of loss mentioned in section 3, gives the following update rule:

$$\begin{aligned}
\omega_{s_t} = \omega_{s_t} + \alpha(\sigma(\omega_{s_t})(1-\sigma(\omega_{s_t}) \\
(V^\pi(s_t) - V_\theta^\beta(s_t))(V_\theta(s_t) - V_\theta^\beta(s_{t-1}))).
\end{aligned} \tag{15}$$

A full derivation is below:
We wish to find $\beta = \sigma(\omega)$ minimizing the loss :

$$\min \ \frac{1}{2}(V^\pi(s_t) - V_{\theta,\omega}^\beta(s_t))^2 \tag{16}$$

$$\tag{17}$$

Table 1: Behavior of β based on the loss

	$V^\pi(s_t) > V^\beta_\theta(s_t)$	$V^\pi(s_t) < V^\beta_\theta(s_t)$
$V_\theta(s_t) > V^\beta_\theta(s_{t-1})$	$\beta \uparrow$	$\beta \downarrow$
$V_\theta(s_t) < V^\beta_\theta(s_{t-1})$	$\beta \downarrow$	$\beta \uparrow$

Taking the derivative of the R.H.S of 2 gives

$$\frac{d}{d\omega_{s_t}} \left(\frac{1}{2}(V^\pi(s_t) - V^\beta_{\theta,\omega}(s_t))^2\right) = (V^\pi(s_t) - V^\beta_{\theta,\omega}(s_t))\left(\frac{d(V^\pi(s_t) - V^\beta_{\theta,\omega}(s_t))}{d\omega_{s_t}}\right)) \quad \text{by chain rule} \tag{18}$$

We know that $\frac{d}{d\omega}\sigma(\omega_{s_t}) = \sigma(\omega_{s_t})(1 - \sigma(\omega_{s_t}))$
and $\frac{d}{d\sigma(\omega_{s_t})}(V^\pi(s_t) - V^\beta_{\theta,\omega}(s_t)) = \frac{d}{d\sigma(\omega_{s_t})}\Big(\sigma(\omega_{s_t})V_\theta(s_t) + \big(1 - \sigma(\omega_{s_t})\big)V^\beta_{\theta,\omega}(s_{t-1}) - V^\pi(s_t)\Big) = V_\theta(s_t) - V^\beta_{\theta,\omega}(s_{t-1})$

Therefore,

$$\frac{d}{d\omega} \left(\frac{1}{2}(V^\pi(s_t) - V^\beta_{\theta,\omega}(s_t))^2 + \lambda\sigma(\omega_{s_t})\right) = \tag{19}$$

$$(V^\pi(s_t) - V^\beta_{\theta,\omega}(s_t))(V_\theta(s_t) - V^\beta_{\theta,\omega}(s_{t-1}))\Big(\sigma(\omega_{s_t})(1 - \sigma(\omega_{s_t}))\Big) + \lambda\Big(\sigma(\omega_{s_t})(1 - \sigma(\omega_{s_t}))\Big) = \tag{20}$$

$$\Big(\sigma(\omega_{s_t})(1 - \sigma(\omega_{s_t}))\Big)\Big((V^\pi(s_t) - V^\beta_{\theta,\omega}(s_t))(V_\theta(s_t) - V^\beta_{\theta,\omega}(s_{t-1}))\Big) \tag{21}$$

Finally, the update rule is simply a gradient step using the above derivative.

6.5 Related Work

Recurrent learning shares similarities with many algorithms in reinforcement learning. The most important similarity is with respect to λ return (23; 24). There is often a debate of explicit versus implicit modelling in reinforcement learning and supervised learning from a conceptual perspective. λ return is a way to implicitly enforce temporal coherence through the trajectory. In this paper, we propose a method to *explicitly* enforce the temporal coherence using recurrent learning. Furthermore, recurrent learning yields a natural way to estimate an emphasis function whereas setting λ efficiently still remains an open problem. In practice both λ return and recurrent learning may be needed to properly enforce temporal coherence. The capacity to ignore states also share some similar motivations to semi-Markov decision process (25) and Temporal Value Transport (26). Temporal Value Transport attempts to exploit similar ideas that recurrent learning does. It does so in a different manner. In particular, Temporal Value Transport is based on a discrete attention mechanism using threshold values in contrast to our continuous β *attention* mechanism. This yields very different algorithms and theory in practice. As mentioned earlier, there exists similarities with emphatic TD (7) in the sense that it emphasizes or de-emphasizes the update done to a state based on β. One key difference with emphatic TD is that the interest is decayed across all the states using γ, whereas in this work it is based on the interest of the future states. Our formulation is driven by the success of *forgetting* and *ignoring* in supervised learning. Furthermore, learning the emphasis function is not explored in (7). Our work also relates to Temporal Regularization (27) that smooth the target of temporal difference methods using previous values of the trajectory. Although sharing similar motivation, the algorithms proposed are different. In particular, learning *smoothing* coefficient is not considered. Finally, (28) proposes a mechanism to adaptively learn to use previous estimates. In

practice, this is done by considering an *auxiliary loss* between the target and the previous estimate in contrast with the setup considered here. Furthermore, the theoretical properties(convergence) of their algorithm is not considered.

6.6 Deep Reinforcement Learning

The following values were considered for the learning rate $\{3E{-}05, 6E{-}05, 9E{-}05, 3E{-}04, 6E{-}04\}$, regularization coefficient $\{0, 0.5, 1\}$ and $N = \{2, 5, 10\}$. The hyperparameter were selected on 10 random seeds. The optimal values for learning rate is $6E-05$, regularization coefficient is 0.5 and $N = 5$.

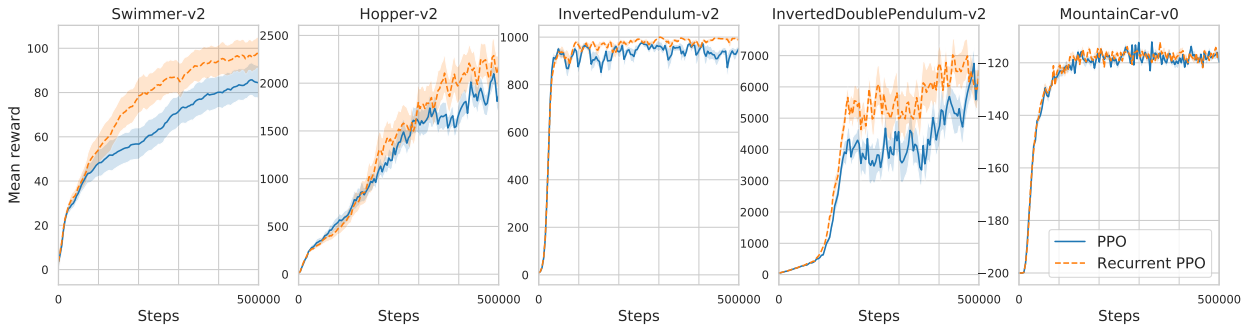


Figure 3: Performance on Mujoco environment. The X-axis represents the training steps and the Y-axis represents the mean reward

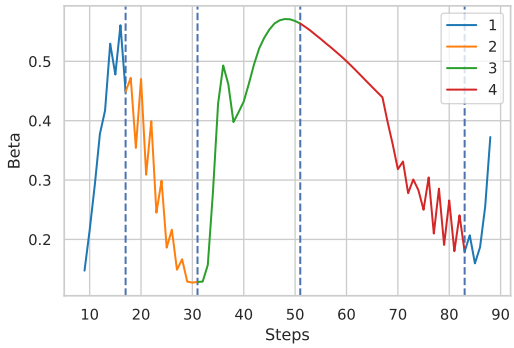


Figure 4: Behavior of β through the trajectory on Hopper with the different phase described in figure 6-10.

Mountain car: Two scenarios may happen when the agent is climbing up the hill on the right side. Either the agent has enough velocity to finish the game and obtain a high reward, or it doesn't have enough velocity and goes back down the hill. During early stages of training, the function approximator is confused about the scenarios mentioned earlier, resulting a drop in value function around step 100 as shown in figure 5. The value increases again once the agent climbs the hill with more velocity. In PPO, we can obtain a more accurate target by setting τ to a high value, thereby eliminating a drop in value. This enables the β network to learn to trust its past estimate rather than the noisy point estimate, hence a significant drop in the β value. As a result, V_θ^β becomes a better estimate of the target than V_θ in this scenario. After training PPO for a while this drop disappears and the β mean goes to 1. This experiment shows the potential of β to smooth out noisy estimates in the trajectory. One caveat to consider is the feedback loop induced by ignoring a state in control. When the policy changes a state that can be ignored at the beginning may be essential later on. One way to address this is to avoid saturating β such that learning remains possible later on.

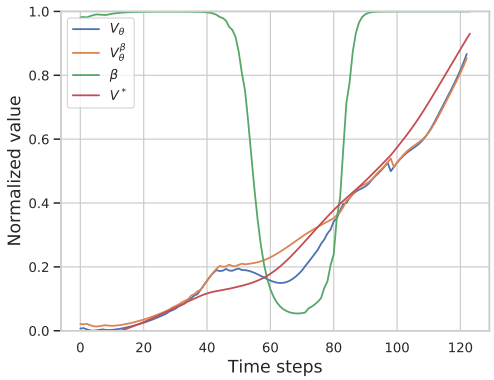


Figure 5: Behavior of β and the value function on Mountain-Car

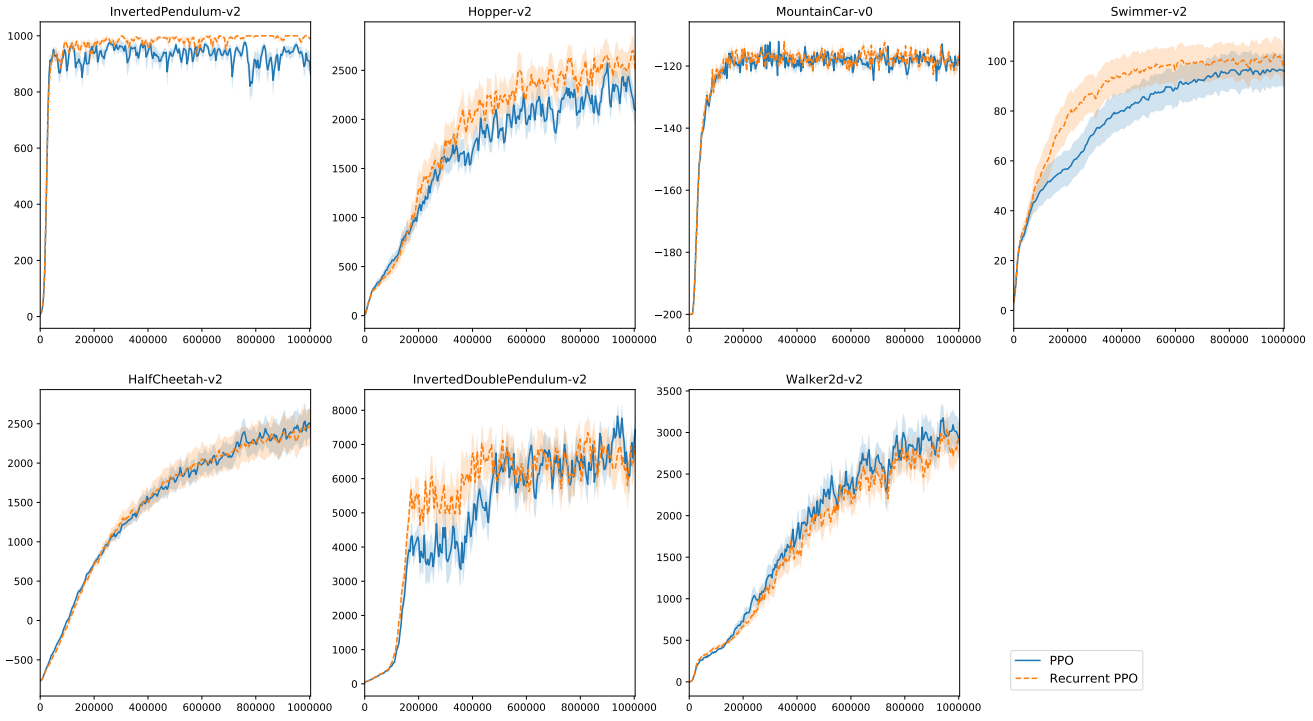


Figure 6: Mean reward using recurrent PPO on Mujoco domains

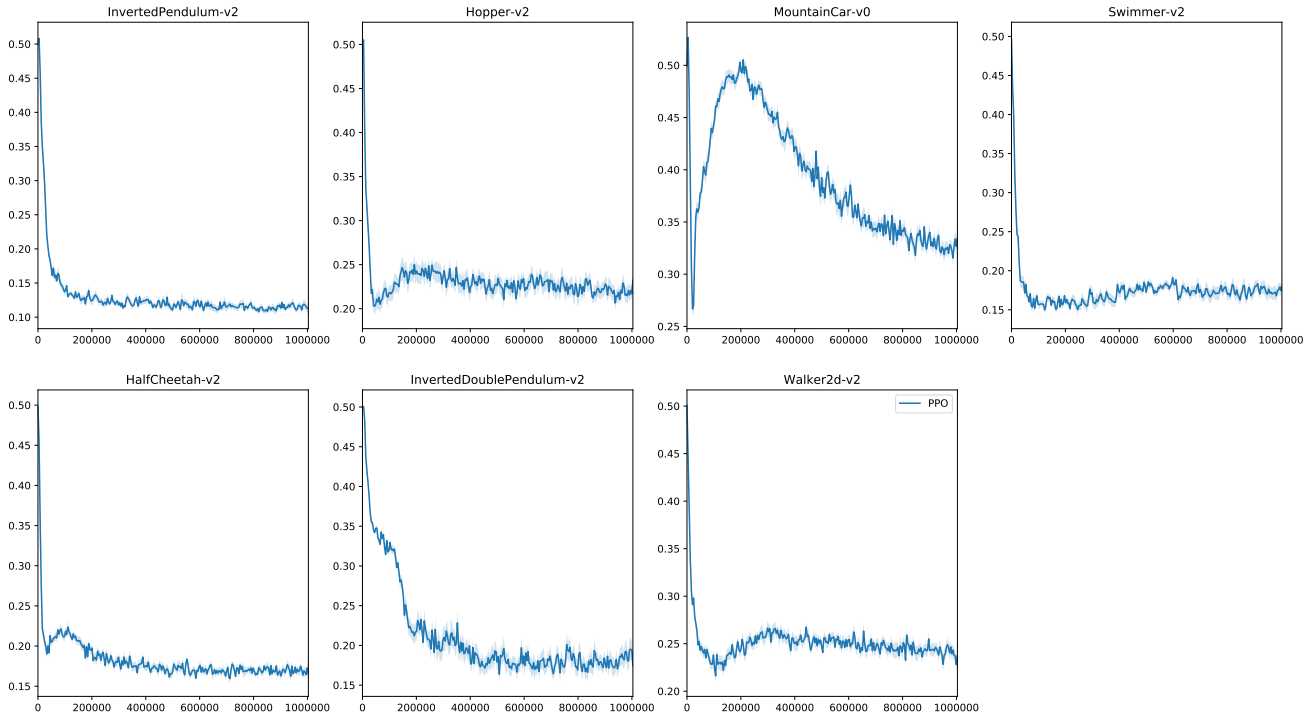


Figure 7: Mean beta values using recurrent PPO on Mujoco domains

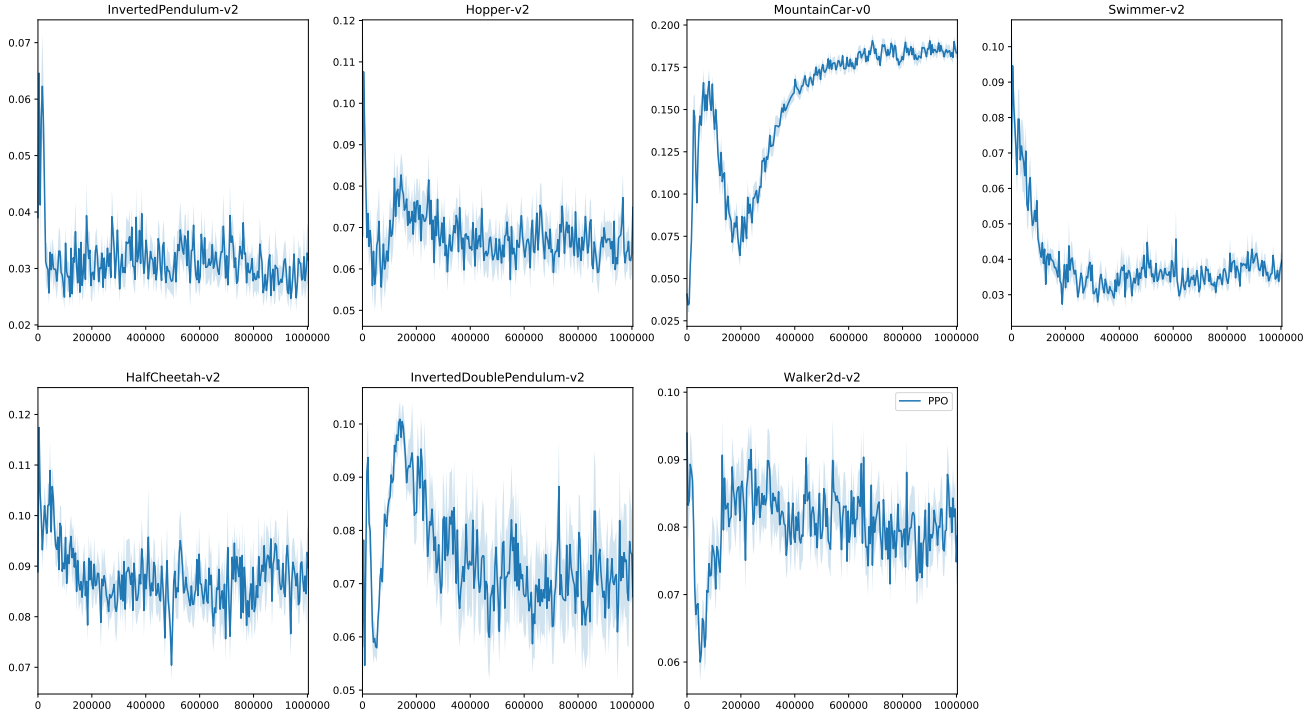


Figure 8: Standard variation of beta using recurrent PPO on Mujoco domains