Long-Term Credit Assignment via Model-based Temporal Shortcuts

Michel Ma **Pierluca D'Oro** **Yoshua Bengio** **Pierre-Luc Bacon**
Mila, Université de Montréal
{michel.ma, pierluca.doro, yoshua.bengio, pierre-luc.bacon}@mila.quebec

Abstract

This work explores the question of long-term credit assignment in reinforcement learning. Assigning credit over long distances has historically been difficult in both reinforcement learning and recurrent neural networks, where discounting or gradient truncation respectively are often necessary for feasibility, but limit the model's ability to reason over longer time scales. We propose LVGTS, a novel model-based algorithm that bridges the gap between the two fields. By using backpropagation through a latent model and temporal shortcuts to directly propagate gradients, LVGTS assigns credit from the future to the possibly distant past regardless of the use of discounting or gradient truncation. We show, on simple but carefully-designed problems, that our approach is able to perform effective credit assignment even in the presence of distractions.

1 Introduction

In reinforcement learning (RL) [33], an agent executes actions in an environment to maximize future rewards. An arbitrarily long period of time can occur from the execution of a particular sequence of actions to the reception of the corresponding reward; the problem of understanding the relationship between actions and rewards regardless of this delay is called *temporal credit assignment* [34]. Effective assignment of credit over long time spans is an issue for most current RL methods and is exacerbated in realistic settings with partially observable environment states.

Zooming out, temporal credit assignment is also a fundamental problem is sequence-based supervised learning tasks. Recurrent Neural Networks (RNNs) [10, 29] are often the go-to solution to solve them but, in a similar way to basic RL algorithms, vanilla RNNs struggle with long-term temporal dependencies [4, 14, 26]. The root causes of this phenomena have been thoroughly investigated, and numerous intuitive solutions have been designed over the years to more accurately capture these temporal dynamics, with considerable performance gains [4, 19, 21, 26, 35].

In this paper, we propose to build on the connection between temporal credit assignment in RNNs and RL tasks, by leveraging ideas developed for the first as a natural solution to design algorithms for the second. In particular, we exploit the similarity between the computational graph in which Backpropagation Through Time (BPTT) operates for computing the gradients for RNNs and the one implied by the interactions of an agent with its environment. As core techniques in RNNs allow them to effectively propagate gradients in a more direct way to the distant past, their RL counterpart will allow agents to assign credit to actions even when they yield rewards in a non-immediate future.

We propose *Latent Value Gradients with Temporal Shortcuts* (LVGTS), a novel model-based policy optimization algorithm, which learns a latent model of the dynamics and uses it, together with reparameterization techniques, for the analytical computation of the policy gradient. During dynamics learning, we impose a specific attentive structure which allows the model to create shortcuts between potentially distant states and actions; then, during policy optimization, the gradient is computed by

Deep Reinforcement Learning Workshop at Neural Information Processing Systems, 2021

backpropagating through real trajectories of experience using the learned model, implicitly injecting nuanced credit assignment patterns into policy learning.

LVGTS is designed by starting from a Partially Observable Markov Decision Process formulation; in that context, we highlight that a blind application of an attention mechanisms to model-based policy optimization does not necessarily result in improved credit assignment capabilities. We then showcase, in small but focused experiments, that LVGTS is able to deal with delayed rewards even in the presence of distractions. To the best of our knowledge, this is the first model-based algorithm that performs effective long-term credit assignment without the need for heuristics or reward shaping, being simultaneously fully compatible with a discounted reinforcement learning formulation.

2 Background

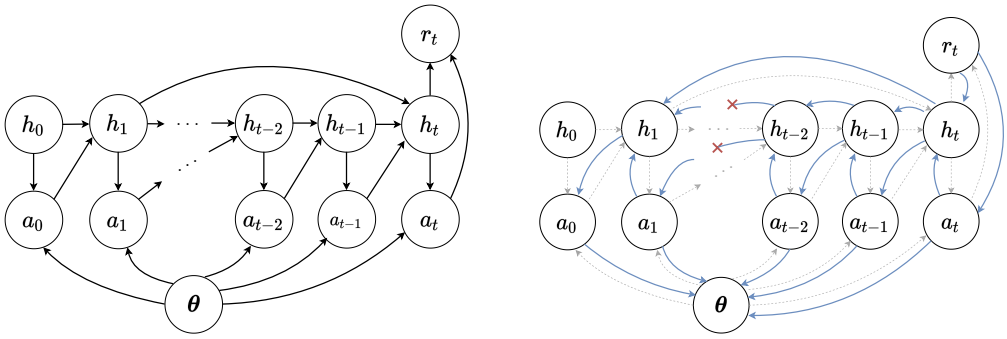
Problem Definition. A continuous finite Partially Observable Markov Decision Process (POMDP) is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{O}, \mathcal{A}, f, O, r, \mu, T)$, where $\mathcal{S}$ is the state space, $\mathcal{O} \subseteq \mathbb{R}^x$ is the observation space, $\mathcal{A} \subseteq \mathbb{R}^u$ is the action space, $f : \mathcal{S} \times \mathcal{A} \to \Delta(S)$ is the state transition function, $O : \mathcal{S} \to \Delta(\mathcal{O})$ is the observation function, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\mu \in \Delta(\mathcal{S})$ is the initial state distribution and T is the problem horizon. A sequence $\tau_t = (o_0, a_0, \ldots, s_t, a_t)$ of t observations and actions can be seen as taken from a space $\mathcal{T}_t$ of *trajectories*. At each discrete time step, the agent interacts with the environment by receiving an observation and acting according to a policy $\pi_{\boldsymbol{\theta}} : \mathcal{T}_t \to \Delta(\mathcal{A})$, which potentially depends on the entire history, and which comes from a space $\Pi_\Theta = \{\pi_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \Theta \subset \mathbb{R}^d\}$ of parameterized policies. To dissect the fundamental features of credit assignment in POMDPs, we are interested in solving so-called *type 1* and *type 2* information acquisition tasks as introduced in [20]. Type 2 tasks test the agent's capability to learn adequate state representations in a partially observable setting where past observations, however distant, are necessary for future reward predictions. The notion of *passive memory* is thus crucial to acting optimally in such an environment: to reach the optimal policy, the agent must be able remember a specific observation o_i. Conversely, type 1 tasks test an agent's ability to simultaneously learn adequate memory-based state representations and perform long-term temporal credit assignment. In this setting, the memory state is not passively observed, but needs to be sought out by the agent who is only rewarded for it in the future when it needs to be recalled.

Gradient-based Policy Optimization. The goal of the agent is to maximize the expected cumulative reward of trajectories induced by its policy:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\theta}} \left[\sum_{t=0}^{T} r_t \right] = \mathbb{E}_{s \sim \mu} \left[V^{\pi_{\boldsymbol{\theta}}}(s) \right] \quad , \tag{1}$$

where $V^{\pi_{\boldsymbol{\theta}}}$ is a state-value function, which gives for every state the expected cumulative reward of policy $\pi_{\boldsymbol{\theta}}$. To maximize this metric, we are interested in estimating $\nabla_{\boldsymbol{\theta}} V$. Under non-restrictive smoothness assumptions the reparameterization trick [13, 22] allows us to rewrite the above expectation, which concerns the distribution induced by policy and environment, with respect to another random variable, for instance $\epsilon \sim \mathcal{N}(0, 1)$, to compute the gradient by direct differentiation of a transformed function. In other words, finding $\nabla_{\boldsymbol{\theta}} V$ becomes a simple matter of estimating $\nabla_{\boldsymbol{\theta}} r_t$. If the transition and reward functions of the environment are known and differentiable, then the gradient of r_t can be found by backpropagating along the trajectory. The resulting gradient is estimated not unlike the traditional backpropagation through time (BPTT) mechanism in recurrent neural networks. The methods, either model-based or value-based, which build upon this techniques for policy optimization are referred to as *value gradients* [11, 18].

Discounting and Truncation. When dealing with large horizons, the practice of discounting the rewards during the computation of a value function has been found to be practically beneficial even in episodic reinforcement learning, where it would not be necessary on a theoretical level. By constraining a problem to an effective horizon of $\frac{1}{1-\gamma}$, the discount factor has been seen to reduce the variance of model-free RL. While the advantages of discounting is not as well understood in model-based learning, it is still employed in almost all prior works [16, 18, 32], and can intuitively control the bias of propagating a learned model through time. Similar to discounting, RNNs often employ truncated backpropagation through time (TBPTT) for long sequences. Again, by effectively reducing the horizon of the backwards pass, some of the major issues in long sequences can be



(a) Model Forward Pass (b) Policy Backward Pass

Figure 1: Model forward pass (Figure 1a) and policy backward pass (Figure 1b, gradient flow shown in blue) in LVGTS, with a gradient truncation length of $N = 3$; model parameters and other rewards apart from the last one are not shown for clarity. The model required an access to memory towards $m = 1$ while being at time t; in result, a temporal shortcut is opened from time t to time $m = 1$ during the computation of the policy gradient, allowing credit to be directly assigned from one time step to the other, regardless of their distance.

mitigated. In both cases, long-term credit assignment becomes difficult or impossible beyond the effective horizon implied by TBPPT and discounting in vanilla RNNs and standard RL algorithms.

3 Credit Assignment with Model-based Temporal Shortcuts

Value gradient methods lie at the intersection of RNN training and reinforcement learning. Given the popularity of discounting in RL and gradient truncation in RNNs, and their direct mathematical connection (see Appendix), it seems natural to propose an algorithm compatible with these methods, but at the same time featuring long-term credit assignment capabilities.

In this section, we present a value gradient method which directly exploits this connection, by employing gradient truncation together with a strategy to circumvent the myopia it implies. Our *Latent Value Gradients with Temporal Shortcuts* algorithm takes advantage of two main building blocks: reparameterization-based stochastic value gradients (SVG) [18] and attention-memory augmented RNNs [21]. Value gradients are used for policy optimization, and an attention-memory augmented RNN is used for state representation in POMDPs. Given its novel architecture, LVGTS has the capability to leverage the rich structure learned during model learning, by directly incorporating it into credit assignment. We first introduce a naive implementation of SVG with the specialized RNN, called *latent-memory stochastic value gradients* (LM-SVG). This will serve both as an introduction and as a baseline for LVGTS.

3.1 A Value Gradients Baseline

In the rest of the paper, we assume for clarity of presentation deterministic dynamics, but then use reparameterization in our implementation to accommodate for stochasticity. When value gradients are applied to the fully observable setting, the policy is directly improved given an (approximate) transition model $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ and reward model $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. In the partially observable setting, the true underlying states are not seen, and must be instead approximated through a latent (or belief) state $h_t := b(o_{1..t}a_{1..t-1}) \approx s_t$, where $b(\cdot)$ is some function approximator. It is also beneficial to introduce a decoder $d(h_t) \approx o_t$, to be employed as a regularizer. To summarize, the models learned for latent planning are therefore:

$$\begin{aligned} h_t &:= b(o_{1..t}a_{1..t-1}) && \text{Encoder} \\ o_t &:= d(h_t) && \text{Decoder} \\ \hat{h}_t &:= f(\hat{h}_{t-1}, a_{t-1}) && \text{Transition model} \\ r_t &:= r(h_t, a_t) && \text{Reward model} \end{aligned} \tag{2}$$

Given such models, we can train a policy $\pi_\theta(h_t) \rightarrow a_t$ following the value gradient methodology outlined in the previous section. LM-SVG implements this architecture by employing an encoder based on an attention-augmented recurrent network architecture. This form of attention-augmented recurrent network is similar to the one introduced in *Sparse Attentive Backtracking* [21]. The parameterization then becomes:

$$\begin{aligned} h_m^t &:= att(h_{i<t}) \ , \\ h_t &:= b(h_{t-1}, h_m^t, o_t a_{t-1}) \ , \end{aligned} \tag{3}$$

where $att(h_{i<t})$ uses a dot-product key-value based attention mechanism [35] to select k latent states from the past, summarize them, and passes it along the encoder b to better predict h_t. A more detailed description of this attention mechanism can also be found in Algorithm 2.

Despite being perhaps the most natural architecture for memory-based latent value gradients, this approach presents two important drawbacks:

1. **The resulting latent space is non-markovian.** The transition model used for value gradients assume that the given states are Markovian, such that $p(h_t|h_{i<t}) = p(h_t|h_{t-1})$. This is not the case given (3), which may make the transition model difficult or impossible to learn.
2. **Credit assignment remains unchanged during policy optimization.** Truncating the gradients will still result in a myopic agent, no matter the accuracy of the model employed. Long-term dependencies that are captured by the dynamics through repeated transitions are cut short with gradient truncation, preventing long-term reasoning. The memory in this case is only used for representation learning, and does not directly address the problem of credit assignment in RL.

In the following, we will present an algorithm able too overcome both of these shortcomings.

3.2 Latent Value Gradients with Temporal Shortcuts

We can solve the two problems of LM-SVG by implementing a simple change: LVGTS includes h_m^t as an argument to the transition model f. The approximate state transitions can then be rewritten with $\hat{h}_m^t$ which recycles the attention mechanism learned in (3), but collects the approximate states predicted by the transition model instead:

$$\begin{aligned} \hat{h}_m^t &:= att(\hat{h}_{i<t}) \ , \\ \hat{h}_t &:= f(\hat{h}_{t-1}, \hat{h}_m^t, a_{t-1}) \ . \end{aligned}$$

The advantages of this formulation are two-fold, conveniently addressing the two problems of LM-SVG:

1. **The resulting latent space is Markovian**. Since h_{t-1} and h_m^t are the only past states used for the encoder b to predict h_t, then the combination of the previous state and memory state provides all the necessary past information for the transition model to predict h_t.
2. **Credit assignment is improved during policy optimization.** The addition of $\hat{h}_m^t$ into the argument of f acts as a *temporal shortcut* in the computational graph of $r_{i \geq t}$, allowing credit to directly flow among distant time steps as seen in Figure 1.

Let us further develop the intuition behind the second point through the idea of temporal shortcuts. Assume that $\hat{h}_m^t$ is comprised of a single state at time step $m < t$. The attention mechanism itself has no bias towards selecting states that are as close to t as possible, so m can be arbitrarily far from t as long as it was deemed important in predicting h_t. Traditionally, BPTT is only able to reason of the effects of m on t by first reasoning about $m + 1$, then $m + 2$, then $m + 3$ until finally t is reached. Similar to a game of broken telephone, the true effects of m on t are often difficult to discern because of this phenomenon, whether it is due to noise or model approximations. The situation is made even worst when employing TBPTT, where the model has no way of understanding the effects of m on t if $t - m > N$. If a temporal shortcut exists between m and t, as is the case in LVGTS, a single step of reasoning separates h_t and h_m^t instead of $t - m$, allowing for a more direct path for credit to be assigned to distant events. Implemented with TBPTT, this allows the agent to reason about past important events, and their neighboring states, just as Figure 1 suggests.

The complete algorithm for LVGTS, including the reparameterization trick for stochastic dynamics, is detailed in Algorithm 1. In what follows, we will see that LVGTS is capable of solving both type 1 and type 2 information acquisition tasks even with gradient truncation, while LM-SVG fails in the former.

4 Experiments

We test LVGTS and LM-SVG on a few simple type 1 and type 2 information acquisition tasks. The goal of these experiments are to verify the algorithms' capabilities to perform long-term credit assignment beyond what gradient truncation permits. First, we formalize the two types of tasks into two respective concrete environments.

4.1 Environments

Passive Memory Task (Type 2 Information Acquisition). The passive memory task (PMT) is a simplification of the grid world environments in [20, 36], generalised to the continuous state-action space. Given two constant time stamps t_1 and t_2, the task is separated into three phases: remember, distractor, recall.

1. The remember phase ($t < t_1$): A random observation o_m is generated for the agent to see. Actions in this phase do not affect future observations. In the simple case, this phase can be characterized by the observation generation process: $o_m \sim \mathcal{N}(0, 1)$.
2. The distractor phase ($t_1 < t < t_2$): Any desired continuous task can be inserted here. It serves to distract the agent with an unrelated task, temporally distancing the first and second phase.
3. The recall phase ($t > t_2$): Random unrelated observations are shown here, but a reward is given to the agent for adequately remembering the randomly generated observation o_m seen during the remember phase. For some maximum reward c, the reward can be defined as: $r_{t>t_2} = (a_t - o_m)^2 + c$.

Active Memory Task (Type 1 Information Acquisition). The active memory task (AMT) uses the same three phases defined above for PMT, with a slight modification to the remember phase. Instead of always observing the correct memory observation, o_m is only fully observed when performing the correct action. Given this action to be a_m, defined in the environment, the observations in the remember phase for AMT are $o_{t<t_1} = o_m + (a_m - a_t)\epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma)$.

We consider three different instances of these two tasks: dummy-PMT, dummy-AMT, and pendulum-AMT. The prefix specifies what environment is used for the distractor phase. The *dummy* environment always returns $o_t = 0, r_t = 0$ for any $t_1 < t < t_2$. The *pendulum* environment is the `Pendulum-v0` environment for the OpenAI Gym library [6].

Algorithm 1: Latent Value Gradients with Temporal Shortcuts

Input : Episode length T, truncation length N, and models: $b(h_t, w_t|o_{1..t}, a_{1..t-1})$, $d(o_t|h_t)$, $f(h_{t+1}|h_t, h_m, a_t, \xi)$, $r(r_t|h_t, a_t)$, $\pi_{\boldsymbol{\theta}}(a_t|h_t, \eta)$.

while *not converged* **do**
  Observe initial observation o_1;
  for $t \leftarrow 1$ **to** T **do**
    $h_t \leftarrow b(o_{1..t}, a_{1..t-1})$ from Algorithm 2;
    $a_t \leftarrow \pi_\theta(h_t, \eta_t), \eta_t \sim \rho(\eta)$;
    Take action a_t and observe o_{t+1}, r_t from the environment ;
    Insert (o_t, a_t, r_t, o_{t+1}) into $\mathcal{D}$;
  end
  Train b, d, f, r with (7) using $\mathcal{D}$;
  Initialize latent state $\hat{h}_1 \leftarrow b(o_1 a_0)$;
  Initialize memory: $\mathcal{M} \in \mathbb{R}^{T \times |h|}$, $\mathcal{M}[1] \leftarrow \hat{h}_1$;
  Initialize return: $G \leftarrow 0$;
  for $t \leftarrow 1$ **to** T **do**
    Infer $\xi, w_t|o_{t-1}, a_{t-1}, o_t$ following Algorithm 2;
    $h_m \leftarrow \sum_{w_t[i] \in w_t} w_t[i]\ \mathcal{M}[i]$;
    $\hat{a}_t \leftarrow \pi_{\boldsymbol{\theta}}(\hat{h}_t, \eta_t)$, $\hat{r}_t \leftarrow r(\hat{h}_t, \hat{a}_t)$;
    $\hat{h}_{t+1} \leftarrow f(\hat{h}_t, h_m, \hat{a}_t, \xi)$;
    $G \leftarrow G + \hat{r}_t$;
    if $t + 1 \mod N == 0$ **then**
      $\hat{h}_t \leftarrow$`Detach(`$\hat{h}_t$`)`;
    end
    $\mathcal{M}[t + 1] \leftarrow \hat{h}_t$;
  end
  Update π_θ using $\nabla_\theta G$;
end

4.2 Results

LM-SVG and LVGTS are tested on all three instances of PMT and AMT with $t_1 = 3, t_2 = 10$, and $T = 15$. In all cases, $c = 0.5$, therefore the maximum possible return for each episode is 2. Both algorithms are subjected to a truncation length of $N = 4$, which means that long-term credit assignment from the recall phase to the remember phase is impossible without skip connections. We set $k = 3$ for all experiments, allowing for a maximum of three skip connections to be made to the past at every time step. The results in Figure 2 show the top fifteen best performing runs out of twenty.

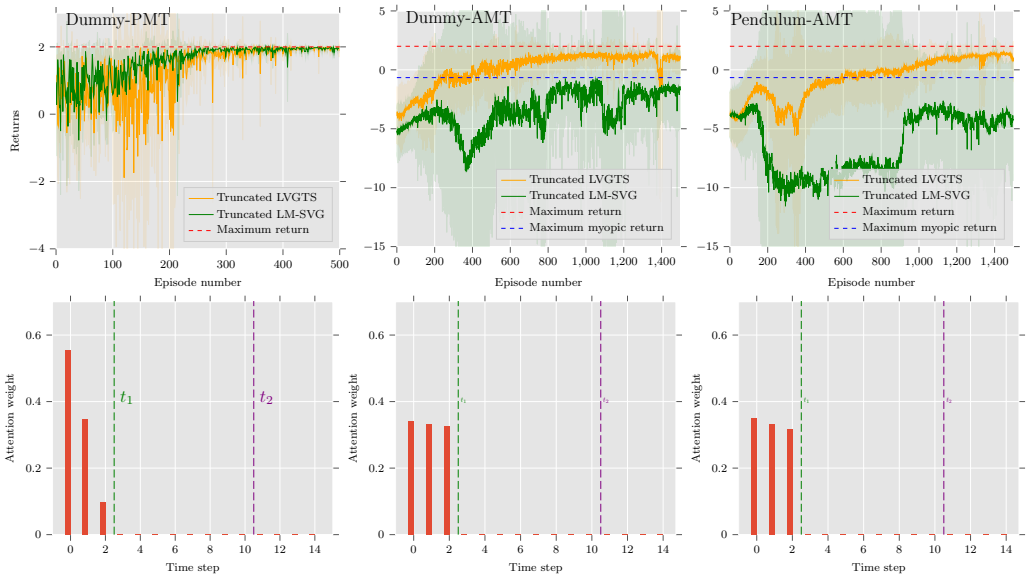


Figure 2: Results of truncated LVGTS and truncated LM-SVG on dummy-PMT, dummy-AMT and pendulum-AMT. On the top row, the return is plotted against the number of episodes during training using value gradient truncation of length 4. On the bottom row, the attention weights for LVGTS are shown with their corresponding tasks. In red are the points in time that are recalled during one step of the recall phase ($t = 12$). Attention is put from the recall phase to the remember phase, which allows truncated LVGTS to perform long-term credit assignment beyond the truncation length.

While both algorithms are able to solve the passive memory task, indicating a competency towards representation learning, only LVGTS is able to reliably solve both AMT instances, even in the presence of a distraction. Conversely, truncated LM-SVG struggles to surpass the performance of an ideal myopic agent even with minimal distractions (dummy-AMT). When distrations are present, in the form of the pendulum task, LM-SVG completely fails under truncation. The maximum myopic return is calculated based on a policy that acts perfectly in phases 2 and 3 according to the noisy observation in phase 1, but acts completely randomly in the first phase. Additionally, the bottom row of Figure 2 shows the agent forming the correct shortcuts to the past for credit to flow through in the third phase.

These results demonstrate that even though we truncate the gradients of LVGTS, a model-based algorithm, it is still able to perform long-term credit assignment in a POMDP well beyond its effective horizon without the need for heuristics.

5 Related Work

Long-Term Credit Assignment in RNNs. A fundamental problem faced while training recurrent neural networks is the assignment of credit across timesteps. The goal of facilitating this task inspired the development of nuanced architectures and training algorithms, which go well-beyond the primordial forms of RNNs, based on the simple flow of a latent state through time [10]. Indeed, vanilla RNNs are susceptible to exploding and vanishing gradients during backpropagation [19, 26],

which hinder the ability to model dynamics spanning over long time windows. To counterbalance this, *attention* [4] has been recently proposed as a particularly successful paradigm: instead of being constrained to only look at a single, most recent, state, the model is augmented with the capability of attending time frames from the past. Methods such as transformers [35] and SAB [21], which we use as a backbone for our algorithm, extensively leverage attention mechanisms.

Value Gradients. The intuitive approach of improving a control policy by differentiating through the dynamics and reward function was one of the first to be conceived in modern RL [30, 37]. Based on this idea, the resulting family of algorithmic tools encompasses both direct differentiation through the learned dynamics [1, 8, 11] and the use of the gradient from a learned value function [12, 15, 23, 31]. Value gradients have been recently revamped [2, 7, 18] due to their synergy with automatic differentiation, the workhorse of deep learning [5], which allows automatic computation of the policy gradient from the computational graph constructed by repeatedly evaluating dynamics, policy and reward. It is possiible to extend value gradients to partially observable and complex environments, by using latent models [16] and computing gradients through the latent space.

Overcoming delayed rewards in RL. Delayed rewards pose a great challenge to long-term credit assignment in reinforcement learning [3, 20]. Traditional use of the discount factor is especially problematic in these settings, as it is a direct contradiction to the reality of delayed rewards in many real world problems [24, 27, 36]. Previous work in addressing this issue has mostly been model-free, and includes methods around meta-learning [28, 38], reward reshaping [3, 28], re-weighting discounting factors [38], and hindsight reasoning [17, 25]. Of particular relevance is the work on Temporal Value Transport (TVT) [20]. TVT is a model-free approach that shares one important similarity with our work: they use an attention-based architecture for latent state representation. However, there are two major differences with LVGTS. Our algorithm is model-based, and directly reuses the attention weights for policy optimization, not requiring any reward shaping heuristic. We take full advantage of automatic differentiation and let this process naturally handle the *value transport* that is manually implemented in TVT.

6 Conclusion

In this paper, we presented a model-based method for long-term credit assignment, based on performing backpropagation through time in latent space. By allowing the agent to construct temporal shortcuts during model learning, credit can be assigned to distant events during policy optimization, while taking full advantage of truncated backpropagation through time. We show, through a simple set of experiments, that LVGTS is able solve both *type 2* and *type 1* information acquisition tasks. Value gradient algorithms have historically been shown to have difficulty scaling to larger more complex environments, and can benefit greatly from using an approximate value function [2, 18]. A natural followup to our work is to scale it to more difficult problems, and to see how critics and value functions might translate into this framework.

References

[1] Pieter Abbeel, Morgan Quigley, and Andrew Y Ng. Using inaccurate models in reinforcement learning. In *ICML*, 2006.

[2] Brandon Amos, Samuel Stanton, Denis Yarats, and Andrew Gordon Wilson. On the model-based stochastic value gradient for continuous reinforcement learning. *arXiv preprint arXiv:2008.12775*, 2020.

[3] Jose A. Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, and Sepp Hochreiter. RUDDER: return decomposition for delayed rewards. *CoRR*, abs/1806.07857, 2018.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[5] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18, 2018.

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[7] Ignasi Clavera, Yao Fu, and Pieter Abbeel. Model-augmented actor-critic: Backpropagating through paths. In *ICML*, 2020.

[8] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning*, pages 465–472, 2011.

[9] Pierluca D'Oro and Wojciech Jaśkowski. How to learn a useful critic? model-based action-gradient-estimator policy optimization, 2020.

[10] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[11] Michael Fairbank. *Value-gradient learning*. PhD thesis, City University London, 2014.

[12] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1582–1591, 2018.

[13] P. Glasserman. *Gradient Estimation via Perturbation Analysis*. Kluwer Academics, 1991.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

[15] T. Haarnoja, Aurick Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

[16] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020.

[17] Anna Harutyunyan, Will Dabney, Thomas Mesnard, Mohammad Gheshlaghi Azar, Bilal Piot, Nicolas Heess, Hado van Hasselt, Gregory Wayne, Satinder Singh, Doina Precup, and Rémi Munos. Hindsight credit assignment. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 12467–12476, 2019.

[18] Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015.

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

[20] 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.

[21] Nan Rosemary Ke, Anirudh Goyal, Olexa Bilaniuk, Jonathan Binas, Michael C Mozer, Chris Pal, and Yoshua Bengio. Sparse attentive backtracking: Temporal credit assignment through reminding. *arXiv preprint arXiv:1809.03702*, 2018.

[22] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[23] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

[24] Jukka Luoma, Sampsa Ruutu, Adelaide Wilcox King, and Henrikki Tikkanen. Time delays, competitive interdependence, and firm performance. *Strategic Management Journal*, 38(3):506–525, 2017.

[25] Thomas Mesnard, Théophane Weber, Fabio Viola, Shantanu Thakoor, Alaa Saade, Anna Harutyunyan, Will Dabney, Tom Stepleton, Nicolas Heess, Arthur Guez, Marcus Hutter, Lars Buesing, and Rémi Munos. Counterfactual credit assignment in model-free reinforcement learning, 2020.

[26] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. PMLR, 2013.

[27] Hazhir Rahmandad, Nelson Repenning, and John Sterman. Effects of feedback delay on learning. *System Dynamics Review*, 25(4):309–338, 2009.

[28] David Raposo, Sam Ritter, Adam Santoro, Greg Wayne, Theophane Weber, Matt Botvinick, Hado van Hasselt, and Francis Song. Synthetic returns for long-term credit assignment, 2021.

[29] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning internal representations by error propagation*. Institute for Cognitive Science, University of California, San Diego, 1986.

[30] Jürgen Schmidhuber. Making the world differentiable: On using self-supervised fully recurrent neural networks for dynamic reinforcement learning and planning in non-stationary environments. 1990.

[31] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[32] Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, July 1991.

[33] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[34] Richard Stuart Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, University of Massachusetts Amherst, 1984.

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[36] Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, Jack Rae, Piotr Mirowski, Joel Z. Leibo, Adam Santoro, Mevlana Gemici, Malcolm Reynolds, Tim Harley, Josh Abramson, Shakir Mohamed, Danilo Rezende, David Saxton, Adam Cain, Chloe Hillier, David Silver, Koray Kavukcuoglu, Matt Botvinick, Demis Hassabis, and Timothy Lillicrap. Unsupervised predictive memory in a goal-directed agent, 2018.

[37] Paul J Werbos. A menu of designs for reinforcement learning over time. *Neural networks for control*, pages 67–95, 1990.

[38] Zeyu Zheng, Risto Vuorio, Richard Lewis, and Satinder Singh. Pairwise weights for temporal credit assignment, 2021.

A Appendix

A.1 Truncated Value Gradients and Discounting

We formalize here the relationship of truncated value gradients with discounting. For the remainder of this section, we suppose a deterministic environment with Markov states s_t. The truncated value function can be written in the following form, where the complete proof can be found in [18]:

$$\begin{aligned}\nabla_\theta V^t_{n-trunc}(s_t;\theta) &= \sum_{k=t}^{T} \frac{\partial r_k}{\partial a_k}\frac{\partial a_k}{\partial \theta} + \frac{\partial s_{k+1}}{\partial a_k}\frac{\partial a_k}{\partial \theta}\frac{\partial V^{k+1}(s_{k+1};\theta)}{\partial s_{k+1}} \ , \\ \frac{\partial V^i(s_i;\theta)}{\partial s_i} &= \frac{\partial r_i}{\partial s_i} + \frac{\partial r_i}{\partial a_i}\frac{\partial a_i}{\partial s_i} + \left(\frac{\partial s_{i+1}}{\partial s_i} + \frac{\partial s_{i+1}}{\partial a_i}\frac{\partial a_i}{\partial s_i}\right)\frac{\partial V^{i+1}(s_{i+1};\theta)}{\partial s_{i+1}} \ , \ for \ i < T-n \\ \frac{\partial V^{T-n}(s_{T-n};\theta)}{\partial s_{T-n}} &:= \mathbf{0} \ .\end{aligned} \tag{4}$$

We can rewrite the recurrent relationship iteratively to better understand how it compares to the original value function:

$$\frac{\partial V^i_{n-trunc}(s_i;\theta)}{\partial s_i} = \frac{\partial r_i}{\partial s_i} + \frac{\partial r_i}{\partial a_i}\frac{\partial a_i}{\partial s_i} + \sum_{j=i+1}^{i+n}\left(\prod_{l=i}^{j}\left(\frac{\partial s_{l+1}}{\partial s_l} + \frac{\partial s_{l+1}}{\partial a_l}\frac{\partial a_l}{\partial s_l}\right)\right)\left(\frac{\partial r_j}{\partial s_j} + \frac{\partial r_j}{\partial a_j}\frac{\partial a_j}{\partial s_j}\right) \tag{5}$$

The original un-truncated and undiscounted value function being:

$$\begin{aligned}V^i(s_i;\theta) &= \sum_{k=i}^{T} r_k \\ \frac{\partial V^i(s_i;\theta)}{\partial s_i} &= \frac{\partial r_i}{\partial s_i} + \frac{\partial r_i}{\partial a_i}\frac{\partial a_i}{\partial s_i} + \sum_{j=i+1}^{T}\left(\prod_{l=i}^{j}\left(\frac{\partial s_{l+1}}{\partial s_l} + \frac{\partial s_{l+1}}{\partial a_l}\frac{\partial a_l}{\partial s_l}\right)\right)\left(\frac{\partial r_j}{\partial s_j} + \frac{\partial r_j}{\partial a_j}\frac{\partial a_j}{\partial s_j}\right) \ ,\end{aligned}$$

and an **n-step** value function being:

$$\begin{aligned}V^i_{n-step}(s_i;\theta) &= \sum_{k=i}^{i+n} r_k \\ \frac{\partial V^i_{n-step}(s_i;\theta)}{\partial s_i} &= \frac{\partial r_i}{\partial s_i} + \frac{\partial r_i}{\partial a_i}\frac{\partial a_i}{\partial s_i} + \sum_{j=i+1}^{i+n}\left(\prod_{l=i}^{j}\left(\frac{\partial s_{l+1}}{\partial s_l} + \frac{\partial s_{l+1}}{\partial a_l}\frac{\partial a_l}{\partial s_l}\right)\right)\left(\frac{\partial r_j}{\partial s_j} + \frac{\partial r_j}{\partial a_j}\frac{\partial a_j}{\partial s_j}\right) \ .\end{aligned} \tag{6}$$

We can now appreciate the equivalence between equations 5 and 6, demonstrating that the **n-truncated value gradient** is equivalent to a **full value gradient** using an **n-step value function**. Given that in traditional actor-critic methods, the critic $\hat{V}^i(s_i)$ is only used derivative estimation [9], the **n-truncated value gradient** is not so different than policy gradient algorithms with $\gamma = 1 - \frac{1}{n}$, where future rewards are discounted smoothly instead of discretely.

A.2 Training the Latent Models

Given a dataset of transition observations sampled from the environment $(o_{1..t}a_{1..t-1}, a_t, o_{t+1}, r_t)$ the environment model approximations are trained on the following loss function:

$$\begin{aligned}h_t &:= b(o_{1..t}a_{1..t-1}) \ , \\ h_{t+1} &:= b(o_{1..t+1}a_{1..t}) \ , \\ \mathcal{L}_{rew} &= \frac{1}{2}\left[r_t(h_t, a_t) - r_t\right]^2 \ , \\ \mathcal{L}_{decoder} &= \frac{1}{2}\left[d(h_t) - o_t\right]^2 \ , \\ \mathcal{L}_{trans} &= \frac{1}{2}||f(\text{StopGradient}(h_t), a_t) - \text{StopGradient}(h_{t+1})||^2 \ .\end{aligned}$$

The total loss sums all the above terms, and can be controlled by the constants $\alpha_{rew}, \alpha_{decoder}, \alpha_{trans}$,

$$\mathcal{L}_{models} = \alpha_{rew}\mathcal{L}_{rew} + \alpha_{decoder}\mathcal{L}_{decoder} + \alpha_{trans}\mathcal{L}_{trans} \ . \tag{7}$$

The decoder model acts as a regularizer, and provides a training signal beyond simple reward prediction for the latent model.

A.3 Attention Augmented RNN

The attention augmented RNN that uses key-value dot product attention, inspired by the work in [21] is detailed here. We assume a sparse recall to memory with k memory recalls, where k is a hyper-parameter set during training. Given a new observation $o_t a_{t-1}$, and the previous latent state h_{t-1} the procedure returns the next latent state h_t, and the attention weights $attn_k$ used for predicting h_t.

Algorithm 2: Procedure for attention-augmented RNN $b(o_t, a_{t-1}, h_{t-1})$

Input :

History	$o_{1..t}, a_{1..t-1} \in \mathbb{R}^{t \times d}$
Sparsity	k
Attention Key-gen	$\mathbf{W} \in \mathbb{R}^{d \times d_k}$
Multilayer Perceptron	$\texttt{MLP}_\psi : \mathbb{R}^{1 \times d} \to \mathbb{R}^{1 \times d_k}$

$K \leftarrow o_{1..t}, a_{1..t-1}\mathbf{W}$;
$\hat{h}_t \leftarrow \texttt{MLP}_\psi(o_t a_{t-1})$;
$attn_w \leftarrow \hat{h}_t K^T$;
$attn_k \leftarrow \text{sorted}(attn_w)[k+1]$;
$attn_k \leftarrow \text{ReLU}(attn_w - attn_k)$;
$h_m \leftarrow \frac{\sum_{i=1}^{t} attn_k[i] o_i a_{i-1}}{\sum_i attn_k[i]}$;
return $\hat{h}_t + h_m, \ \ attn_k$