Adaptive Pairwise Weights for Temporal Credit Assignment

Zeyu Zheng[1]*, Risto Vuorio[2]*, Richard Lewis[1], Satinder Singh[1]

[1]University of Michigan
[2]University of Oxford
zeyu@umich.edu, risto.vuorio@cs.ox.ac.uk, rickl@umich.edu, baveja@umich.edu

Abstract

How much credit (or blame) should an action taken in a state get for a future reward? This is the fundamental temporal credit assignment problem in Reinforcement Learning (RL). One of the earliest and still most widely used heuristics is to assign this credit based on a scalar coefficient, lambda (treated as a hyperparameter), raised to the power of the time interval between the state-action and the reward. In this empirical paper, we explore heuristics based on more general pairwise weightings that are functions of the state in which the action was taken, the state at the time of the reward, as well as the time interval between the two. Of course it isn't clear what these pairwise weight functions should be, and because they are too complex to be treated as hyperparameters we develop a metagradient procedure for learning these weight functions during the usual RL training of a policy. Our empirical work shows that it is often possible to learn these pairwise weight functions during learning of the policy to achieve better performance than competing approaches.

1 Introduction

The following *umbrella problem* (Osband et al. 2019) illustrates a fundamental challenge in most reinforcement learning (RL) problems, namely the *temporal credit assignment* (TCA) problem. An RL agent takes an umbrella at the start of a cloudy morning and experiences a long day at work filled with various rewards uninfluenced by the umbrella, before needing the umbrella in the rain on the way home. The agent must learn to credit the take-umbrella action in the cloudy-morning state with the very delayed reward at the end of the day, while also learning to not credit the action with the many intervening rewards, despite their occurring much closer in time. More generally, the TCA problem is how much credit or blame should an action taken in a state get for a future reward. One of the earliest and still most widely used heuristics for TCA comes from the celebrated TD(λ) (Sutton 1988) family of algorithms, and assigns credit based on a scalar coefficient λ raised to the power of the time interval between the state-action and the reward. Note that this is a recency and frequency heuristic, in that it assigns credit based on how recently and how frequently a state-action pair has occurred prior to the reward. It is important, however, to also note that this heuristic has not in any way shown to be the "optimal" way for TCA. In particular, in the umbrella problem the action of taking the umbrella on a cloudy morning will be assigned credit for the rewards achieved during the workday early on in learning and it is only after a lot of learning that this effect will diminish. Nevertheless, the recency and frequency heuristic has been adopted in most modern RL algorithms because it is so simple to implement, with just one hyperparameter, and because it has been shown to allow for asymptotic convergence to the true value function under certain circumstances.

In this empirical paper, we present two new families of algorithms for addressing TCA: one that generalises TD(λ) and a second that generalises a Monte-Carlo algorithm. Specifically, our generalisation introduces pairwise weightings that are functions of *the state in which the action was taken*, *the state at the time of the reward*, and *the time interval between the two*. Of course, it isn't clear what this pairwise weight function should be, and it is too complex to be treated as a hyperparameter (in contrast to the scalar λ in TD(λ)). We develop a metagradient approach to learning the pairwise weight function at the same time as learning the policy of the agent. Like other metagradient algorithms, our algorithm has two loops: an outer loop that periodically updates the pairwise weight function in order to optimize the usual RL loss (policy gradient loss in our case) and an inner loop where the policy parameters are updated using the pairwise weight function set by the outer loop.

Our main contribution in this paper is a family of algorithms that contains within it the theoretically well understood TD(λ) and Monte-Carlo algorithms. We show that the additional flexibility of our algorithms can yield benefit analytically in a simple illustrative example intended to build intuition and then empirically in more challenging TCA problems. A second contribution is the metagradient algorithm to learn such the pairwise-weighting function that parameterises our family of algorithms. Our empirical work is geared towards answering two questions: (1) Are the proposed pairwise weight functions able to outperform the best choice of λ and other baselines? (2) Is our metagradient algorithm able to learn the pairwise weight functions fast enough to be worth the extra complexity they introduce?

*These authors contributed equally.

2 Related Work

Several heuristic methods have been proposed to address the long-term credit assignment problem in RL. RUDDER (Arjona-Medina et al. 2019) trains a LSTM (Hochreiter and Schmidhuber 1997) to predict the return of an episode given the entire state and action sequence and then conducts contribution analysis with the LSTM to redistribute rewards to state-action pairs. Synthetic Returns (SR) (Raposo et al. 2021) directly learns the association between past events and future rewards and use it as a proxy for credit assignment. Different from the predictive approach of RUDDER and SR, Temporal Value Transport (TVT) (Hung et al. 2019) augments the agent with an external memory module and utilizes the memory retrieval as a proxy for transporting future value back to related state-action pairs. We compare against TVT by using their published code, and we take inspiration from the core reward-redistribution idea from RUDDER and implement it within our policy gradient agent as a comparison baseline (because the available RUDDER code is not directly applicable). We do not compare to SR because their source code is not available.

We also compare against two other algorithms that are more closely related to ours in their use of metagradients. Xu et al. (Xu, van Hasselt, and Silver 2018) adapt λ via metagradients rather than tuning it via hyperparameter search, thereby improving over the use of a fixed-λ algorithm. The Return Generating Model (RGM) (Wang, Ye, and Liu 2019) generalizes the notion of return from exponentially discounted sum of rewards to a more flexibly weighted sum of rewards where the weights are adapted via metagradients during policy learning. RGM takes the entire episode as input and generates one weight for each time step. In contrast, we study pairwise weights as explained below.

Some recent works address counterfactual credit assignment where classic RL algorithms struggle (Harutyunyan et al. 2019; Mesnard et al. 2020; van Hasselt et al. 2020). Although they are related to our work in that they also address the TCA problem, we do no compare to them because our work does not focus on the counterfactual aspect.

3 Pairwise Weights for Advantages

At the core of our contribution are new parameterizations of functions for computing advantages used in policy gradient methods. Therefore, we briefly review advantages in policy gradient methods and TD(λ) as our points of departure.

Background on Advantages, Policy Gradient, and TD(λ). We assume an episodic RL setting. The agent's policy π_θ, parameterized by θ, maps a state S to a probability distribution over the actions. Within each episode, at time step t, the agent observes the current state S_t, takes an action $A_t \sim \pi_\theta(\cdot|S_t)$, and receives the reward R_{t+1}. The return is denoted by $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ where γ is the discount factor and T denotes the length of the episode. The state-value function V^π is defined as

$$V^\pi(s) = \mathbb{E}_\pi[G_t|S_t = s], \tag{1}$$

and the action-value function Q^π is defined as

$$Q^\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]. \tag{2}$$

The notation $\mathbb{E}_\pi[\cdot]$ denotes the expected value of a random variable given that the agent follows the policy π. Because the policy is parameterized by θ, we will use $\mathbb{E}_\pi[\cdot]$ and $\mathbb{E}_\theta[\cdot]$ interchangeably. The advantage function is defined as

$$\Psi^\pi(s,a) = Q^\pi(s,a) - V^\pi(s,a). \tag{3}$$

For brevity, we will omit the superscript π on V, Q, and Ψ.

The performance measure for the policy π_θ, denoted by $J(\theta)$, is defined as the expected sum of the rewards when the agent behaves according to π_θ, i.e.,

$$J(\theta) = \mathbb{E}_\theta[\sum_{t=1}^{T} \gamma^{t-1} R_t], \tag{4}$$

The gradient of $J(\theta)$ w.r.t the policy parameters θ is (Sutton et al. 2000; Williams 1992)

$$\nabla_\theta J(\theta) = \mathbb{E}_\theta\Big[\big(G_t - b(S_t)\big)\nabla_\theta \log \pi_\theta(A_t|S_t)\Big], \tag{5}$$

where $b(S_t)$ is a baseline function for variance reduction. If we choose the state-value function V as the baseline function, then Eq. 5 can be rewritten as (Schulman et al. 2015)

$$\nabla_\theta J(\theta) = \mathbb{E}_\theta\Big[\Psi(S_t, A_t)\nabla_\theta \log \pi_\theta(A_t|S_t)\Big]. \tag{6}$$

Since the true state-value function V is usually unknown, an approximation v is used in place of V, which leads to a Monte-Carlo (MC) estimation of Ψ:

$$\hat{\Psi}_t^{\text{MC}} = G_t - v(S_t). \tag{7}$$

However, $\hat{\Psi}^{\text{MC}}$ usually suffers from high variance. To reduce variance, the approximated state-value function v is used to estimate the return as in the TD(λ) algorithm using the eligibility trace parameter λ; specifically the new form of the return, called λ-return is a weighted sum of n-step truncated corrected returns where the correction uses the estimated value function after n-steps. The corresponding λ-estimator is (see (Schulman et al. 2015) for a full derivation)

$$\hat{\Psi}_t^{(\lambda)} = \sum_{k=t+1}^{T} (\gamma\lambda)^{k-t-1}\delta_k, \tag{8}$$

where $\delta_t = R_t + \gamma v(S_t) - v(S_{t-1})$ is the TD-error at time t. As a special case, when $\lambda = 1$, it recovers the MC estimator (Schulman et al. 2015). As noted above, the value for λ is usually manually tuned as a hyperparameter. Adjusting λ provides a way to tradeoff bias and variance in $\hat{\Psi}^{(\lambda)}$ (this is absent in $\hat{\Psi}^{\text{MC}}$). Below we present two new estimators that are analogous in this regard to $\hat{\Psi}^{(\lambda)}$ and $\hat{\Psi}^{\text{MC}}$.

Proposed Heuristic 1: Advantages via Pairwise Weighted Sum of TD-errors. Our first new estimator, denoted PWTD for **P**airwise **W**eighted **TD**-error, is a strict generalization of the λ-estimator above and is defined as follows:

$$\hat{\Psi}_{\eta,t}^{\text{PWTD}} = \sum_{k=t+1}^{T} f_\eta(S_t, S_k, k-t)\delta_k, \tag{9}$$

where $f_\eta(S_t, S_k, k-t) \in [0,1]$, parameterized by η, is the scalar weight given to the TD-error δ_k as a function of the state to which credit is being assigned, the state at which the TD-error is obtained, and the time interval between the two. Note that if we choose $f(S_t, S_k, k-t) = (\gamma\lambda)^{k-t-1}$, it recovers the usual λ-estimator $\hat{\Psi}^{(\lambda)}$.

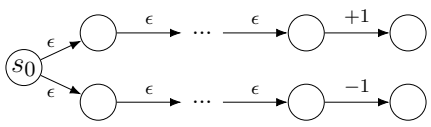

Figure 1: A simple illustrative MDP. The initial action determines the final reward but does not impact the intermediate rewards. The consequence of the initial action is delayed.

Proposed Heuristic 2: Advantages via Pairwise Weighted Sum of Rewards. Instead of generalizing from the λ-estimator, we can also generalize from the MC estimator via pairwise weighting. Specifically, the new pairwise-weighted return is defined as

$$G^{\text{PWR}}_{\eta,t} = \sum_{k=t+1}^{T} f_\eta(S_t, S_k, k-t) R_k, \tag{10}$$

where $f_\eta(S_t, S_k, k-t) \in [0,1]$ is the scalar weight given to the reward R_k. The corresponding advantage estimator, denoted PWR for **P**airwise **W**eighted **R**eward, then is:

$$\hat{\Psi}^{\text{PWR}}_{\eta,t} = G^{\text{PWR}}_{\eta,t} - v^{\text{PWR}}(S_t), \tag{11}$$

where $V^{\text{PWR}}(s) = \mathbb{E}_\theta[G^{\text{PWR}}_{\eta,t} | S_t = s]$ and v^{PWR} is an approximation of V^{PWR}. Note that if we choose $f(S_t, S_k, k-t) = \gamma^{k-t-1}$, we can recover the MC estimator $\hat{\Psi}^{\text{MC}}$.

The benefit of the additional flexibility provided by these new estimators highly depends on the choice of the pairwise weight function f. As we will demonstrate in the simple example below, the new estimators can yield lower variance and benefit policy learning if the function f captures the underlying credit assignment structure of the problem. On the other hand, the new estimators may not even be well-defined in the infinite-horizon setting if the pairwise weight function is chosen wrongly because the weighted sum of TD-errors/rewards could be unbounded. Designing a good pairwise weight function by hand is challenging because it requires both domain knowledge to capture the credit assignment structure and careful tuning to avoid harmful consequences. Thus we propose a metagradient algorithm to *learn* the pairwise weight function such that it benefits policy learning, as detailed in § 4.

An Illustrative Analysis of the Benefit of the PWR Estimator. Consider the simple-MDP version of the umbrella problem in Figure 1. Each episode starts at the leftmost state, s_0, and consists of T transitions. The only choice of action is at s_0 and it determines the reward on the last transition. A noisy reward ϵ is sampled for each intermediate transition independently from a distribution with mean μ and variance $\sigma^2 > 0$. These intermediate rewards are independent of the initial action. We consider the undiscounted setting in this example. The expected return for state s_0 under policy π is

$$V(s_0) = \mathbb{E}_\pi[G_0] = (T-1)\mu + \mathbb{E}_\pi[R_T].$$

For any initial action a_0, the advantage is

$$\Psi(s_0, a_0) = \mathbb{E}_\pi[G_0 | a_0] - V(s_0) = \mathbb{E}_\pi[R_T | a_0] - \mathbb{E}_\pi[R_T].$$

Consider pairwise weights for computing $\hat{\Psi}^{\text{PWR}}(s_0, a_0)$ that place weight only on the final transition, and zero weight on the noisy intermediate rewards, capturing the notion that the intermediate rewards are not influenced by the initial action choice. More specifically, we choose f such that for any episode, $w_{0T} = 1$ and $w_{ij} = 0$ for other i and j. The shorthand w_{ij} denotes $f(S_i, S_j, j-i)$ for brevity. The expected parameterized reward sum for the initial state s_0 is

$$V^{\text{PWR}}(s_0) = \mathbb{E}_\pi[G_{\eta,0}] = \mathbb{E}_\pi[\sum_{i=t}^{T} w_{0t} R_t] = \mathbb{E}_\pi[R_T].$$

If v^{PWR} is correct, for any initial action a_0, the pairwise-weighted advantage is the same as the regular advantage:

$$\begin{aligned}
\mathbb{E}_\pi[\hat{\Psi}^{\text{PWR}}_\eta(s_0, a_0)] &= \mathbb{E}_\pi[G_{\eta,0} - v^{\text{PWR}}(s_0) | a_0] \\
&= \mathbb{E}_\pi[\sum_{t=1}^{T} w_{0t} R_t] - V^{\text{PWR}}(s_0) \\
&= \mathbb{E}[R_T | a_0] - \mathbb{E}_\pi[R_T] = \Psi(s_0, a_0).
\end{aligned}$$

As for variance, for any initial action a_0, $[G_{\eta,0} | a_0]$ is deterministic because of the zero weight on all the intermediate rewards and thus $\hat{\Psi}^{\text{PWR}}_\eta(s_0, a_0)$ has zero variance. The variance of $\hat{\Psi}^{\text{MC}}(s_0, a_0)$ on the other hand is $(T-1)\sigma^2 > 0$. Thus, in this illustrative example $\hat{\Psi}^{\text{PWR}}$ yields an unbiased advantage estimator with far lower variance than $\hat{\Psi}^{\text{MC}}$.

Our example exploited knowledge of the domain to set weights that would yield an unbiased advantage estimator with reduced variance, thereby providing some intuition on how a more flexible return might in principle yield benefits for learning. Of course, in general RL problems will have the umbrella problem in them to varying degrees. But how can these weights be set by the agent itself, without prior knowledge of the domain? We turn to this question next.

4 A Metagradient Algorithm for Adapting Pairwise Weights

Recently metagradient methods have been developed to learn various kinds of parameters that would otherwise be set by hand or by manual hyperparameter search; these include discount factors (Xu, van Hasselt, and Silver 2018; Zahavy et al. 2020), intrinsic rewards (Zheng, Oh, and Singh 2018; Rajendran et al. 2019; Zheng et al. 2020), auxiliary tasks (Veeriah et al. 2019), constructing general return functions (Wang, Ye, and Liu 2019), and discovering new RL objectives (Oh et al. 2020; Xu et al. 2020). We use the metagradient algorithm from (Xu, van Hasselt, and Silver 2018) for training the pairwise weights. The algorithm consists of an outer loop learner for the pairwise weight function, which is driven by a conventional policy gradient loss and an inner loop learner driven by a policy-gradient loss based on the new pairwise-weighted advantages. An overview of the algorithm is in the appendix. For brevity, we use $\hat{\Psi}_\eta$ to denote $\hat{\Psi}^{\text{PWTD}}_\eta$ or $\hat{\Psi}^{\text{PWR}}_\eta$ unless it causes ambiguity.

Learning in the Inner Loop. In the inner loop, the pairwise-weighted advantage $\hat{\Psi}_\eta$ is used to compute the policy gradient. We rewrite the gradient update from Eq. 5 with

the new advantage as

$$\nabla_\theta J_\eta(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T-1} \hat{\Psi}_{\eta,t} \nabla_\theta \log \pi_\theta(A_t|S_t)], \quad (12)$$

where τ is a trajectory sampled by executing π_θ. The overall update to θ is

$$\nabla_\theta J^{\text{inner}}(\theta) = \nabla_\theta J_\eta(\theta) + \beta^{\mathcal{H}} \nabla_\theta \mathcal{H}(\pi_\theta), \quad (13)$$

where $\mathcal{H}(\theta)$ is the usual entropy regularization term (Mnih et al. 2016) and $\beta^{\mathcal{H}}$ is a mixing coefficient. We apply gradient ascent to update the policy parameters and the updated parameters are denoted by θ'.

Computing $\hat{\Psi}_\eta^{\text{PWR}}$ with Equation 11 requires a value function predicting the expected pairwise-weighted sums of rewards. We train the value function, v_ψ with parameters ψ, along with the policy by minimizing the mean squared error between its output $v_\psi(S_t)$ and the pairwise-weighted sum of rewards $G_{\eta,t}$. The objective for training v_ψ is

$$J_\eta^v(\psi) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T-1} (G_{\eta,t} - v_\psi(S_t))^2]. \quad (14)$$

Note that $\hat{\Psi}_\eta^{\text{PWTD}}$ does not need this extra value function.

Updating η via Metagradient in the Outer Loop. To update η, the parameters of the pairwise weight functions, we need to compute the gradient of the usual policy loss w.r.t. η through the effect of η on the inner loop's updates to θ.

$$\nabla_\eta J^{\text{outer}}(\eta) = \nabla_{\theta'} J(\theta') \nabla_\eta \theta'. \quad (15)$$

where,

$$\nabla_{\theta'} J(\theta') = \mathbb{E}_{\tau'\sim\pi_{\theta'}}[\sum_{i=0}^{T-1} \Psi_t \nabla_{\theta'} \log \pi_{\theta'}(A_t|S_t)], \quad (16)$$

τ' is another trajectory sampled by executing the updated policy $\pi_{\theta'}$ and Ψ_t is the regular advantage.

Note that we need two trajectories, τ and τ', to make one update to the meta-parameters η. The policy parameters θ are updated after collecting trajectory τ. The next trajectory τ' is collected using the updated parameters θ'. The η-parameters are updated on τ'. In order to make more efficient use of the data, we follow (Xu, van Hasselt, and Silver 2018) and reuse the second trajectory τ' in the next iteration as the trajectory for updating θ. In practice we use modern auto-differentiation tools to compute Equation 15 without applying the chain rule explicitly. Computing the regular advantage requires a value function for the regular return. This value function is parameterized by ϕ and updated to minimize the squared loss analogously to v_ϕ.

5 Experiments

We present three sets of experiments. The first set (§5.1) uses simple tabular MDPs that allow visualization of the pairwise weights learned by Meta-PWTD and -PWR. The results show that the metagradient adaptation both *increases* and *decreases* weights in a way that can be interpreted as reflecting explicit credit assignment and variance reduction. In the second set (§5.2) we test Meta-PWTD and -PWR with neural networks in the benchmark credit assignment task *Key-to-Door* (Hung et al. 2019). We show that Meta-PWTD and -PWR outperform several existing methods for directly addressing credit assignment, as well as TD(λ) methods, and show again that the learned weights reflect domain structure in a sensible way. In the third set (§5.3), we evaluate Meta-PWTD and -PWR in two benchmark RL domains, `bsuite` (Osband et al. 2019) and Atari, and show that our methods do not hinder learning when environments do not pose idealized long-term credit assignment challenges.

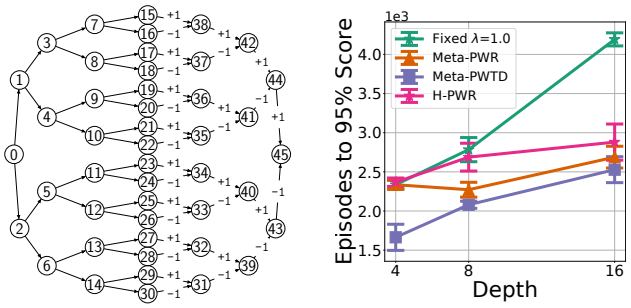

Figure 2: (Left) Depth 8 DAG environment with choice of two actions at each state and rewards along transitions. (Right) Learning performance of regular return, handcrafted weights, and fixed meta-learned weights. Results are averaged over 5 independent runs. Low is good.

5.1 Learned Pairwise Weights in A Simple MDP

Consider the environment represented as a DAG in Figure 2 (left). In each state in the left part of the DAG (states 0–14, the *first phase*), the agent chooses one of two actions but receives no reward. In the remaining states (states 15–44, the *second phase*) the agent has only one action available and it receives a reward of $+1$ or -1 at each transition. Crucially, the rewards the agent obtains in the second phase are a consequence of the action choices in the first phase because they determine which states are encountered in the second phase. There is an interesting credit assignment problem with a nested structure; for example, the action chosen at state 1 determines the reward received later upon transition into state 44. We refer to this environment as the *Depth 8 DAG* and also report results below for depths 4 and 16.

In the DAG environments we use a tabular policy, value function, and meta-parameter representations. The parameters θ, ψ, ϕ, and η represent the policy, baseline for the weighted return, baseline for the regular return, and meta-parameters respectively. The η parameters are a $|S| \times |S|$ matrix. The entry on the ith row and the jth column defines the pairwise weight for computing the contribution of reward at state j to the return at state i. A sigmoid is used to bound the weights to $[0, 1]$, and the η parameters are initialized so that the pairwise weights are close to 0.5. Note that even in a tabular domain such as the DAG, setting the credit assignment weights by random search would be infeasible due to the number of possible weight combinations. This problem is exacerbated by larger domains discussed in the following sections. For this reason, the metagradient al-

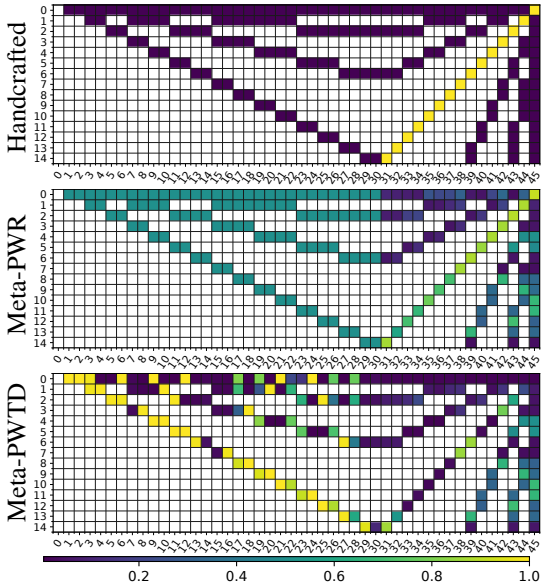

Figure 3: Inner loop-reset weight visualization: Top: Handcrafted pairwise weights for Depth 8 DAG; rows and columns correspond to states in Fig. 2. Middle: Meta-learned weights for Depth 8 DAG for Meta-PWR and Bottom: Meta-PWTD.

gorithm is a promising candidate for setting the weights.

Visualizing the Learned Weights via Inner-loop Reset. To clearly see the most effective weights that metagradient learned for a random policy, we repeatedly reset the policy parameters to a random initialization while continuing to train the meta-parameters until convergence. More specifically: the meta-parameters η are trained repeatedly by randomly initializing θ, ψ, and ϕ and running the inner loop for 16 updates for each outer loop update. Following existing work in metagradient (Veeriah et al. 2019; Zheng et al. 2020), the outer loop objective is evaluated on all 16 trajectories sampled with the updated policies. The gradient of the outer loop objective on the ith trajectory with respect to η is backpropagated through all of the preceding updates to θ. Hyperparameters are provided in the appendix.

What pairwise weights would accelerate learning in this domain? Figure 2 (top) visualizes a set of *handcrafted* weights for $\hat{\Psi}^{\text{PWR}}$ in the Depth 8 DAG; each row in the grid represents the state in which an action advantage is estimated, and each column the state in which a future reward is experienced. For each state pair (s_i, s_j) the weight is 1 (yellow) only if the reward at s_j depends on the action choice at s_i, else it is zero (dark purple; the white pairs are unreachable). Figure 2 (middle) shows the corresponding weights learned by Meta-PWR. Importantly, the learned pairwise weights have been *increased* for those state pairs in which the handcrafted weights are 1 and have been *decreased* for those state pairs in which the handcrafted weights are 0. As in the analysis of the simple domain in §3, these weights will result in lower variance advantage estimates.

The same reset-training procedure was applied to Ψ^{PWTD}.

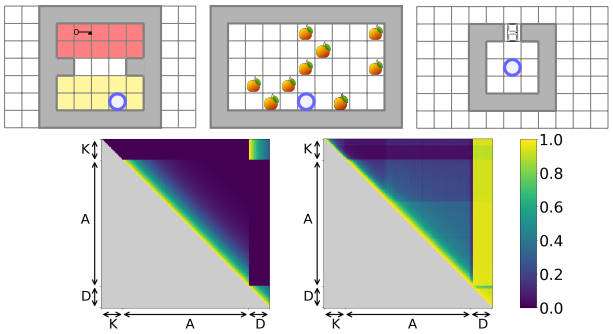

Figure 4: (Top) The three phases in KtD. The blue circle denotes the agent. (Bottom left) Visualization of Handcrafted weights in the KtD experiment. (Bottom right) Weights learned by Meta-PWR in the $\mu = 5, \sigma = 5$ configuration.

Figure 2 (bottom) visualizes the resulting weights. Since the TD-errors depend on the value function which are nonstationary during agent learning, we expect different weights to emerge at different points in training; the presented weights are but one snapshot. But a clear contrast to reward weighting can be seen: high weights are placed on transitions in the first phase of the DAG, which yield no rewards—because the TD-errors at these transitions do provide signal once the value function begins to be learned. In the appendix, we explicitly probe the adaptation of Ψ^{PWTD} to different points in learning by modifying the value function in reset experiments, and show that the weights indeed adapt sensibly to differences in the accuracy of the value function.

Evaluation of the Learned Pairwise Weights. After the θ-reset training of the pairwise-weights completed, we used them to train a new set of θ parameters, fixing the pairwise weights during learning. Figure 2 (right) shows the number of episodes to reach 95% of the maximum score in each DAG, for policies trained with regular returns, handcrafted weights (H-PWR), and meta-learned weights. Using the meta-learned weights learned as fast as (indeed faster than) using the handcrafted weights, and both were faster than the regular returns, with the gap increasing for larger DAG-depth. We conjecture that the learned weights performed even better than the handcrafted weights because the learned weights adapted to the dynamics of the inner-loop policy learning procedure whereas the handcrafted weights did not. Learning curves in the appendix show that all method achieved the optimal performance in the end.

5.2 The Key-to-Door Experiments

We evaluated Meta-PWTD and -PWR the Key-to-Door (KtD) environment (Hung et al. 2019) that is an elaborate umbrella problem that was designed to show-off the TVT algorithm's ability to solve TCA. We varied properties of the domain to vary the credit assignment challenge. We compared the learning performance of our algorithms to a version of $\hat{\Psi}^{\text{PWR}}$ that uses fixed handcrafted pairwise weights and no metagradient adaptation, as well as to the following **five** baselines (see related work in §2): **(a)** best

fixed-λ: Actor-Critic (A2C) (Mnih et al. 2016) with a best fixed λ found via hyperparameter search; **(b)** TVT (Hung et al. 2019) (using the code accompanying the paper); **(c)** A2C-RR: a reward redistribution method *inspired* by RUDDER (Arjona-Medina et al. 2019); **(d)** Meta-$\lambda(s)$ (Xu, van Hasselt, and Silver 2018): meta-learning a state-dependent function $\lambda(s)$ for λ-returns; and **(e)** RGM (Wang, Ye, and Liu 2019): meta-learning a *single* set of weights for generating returns as a linear combination of rewards.

Environment and Parametric Variation. KtD is a fixed-horizon episodic task where each episode consists of three phases (Figure 4 top). In the *Key phase* (15 steps in duration) there is no reward and the agent must navigate to the key to collect it. In the *Apple phase* (90 steps in duration) the agent collects apples; apples disappear once collected. Each apple yields a noisy reward with mean μ and variance σ^2. In the *Door phase* (15 steps in duration) the agent starts at the center of a room with a door but can open the door only if it has collected the key earlier. Opening the door yields a reward of 10. Crucially, picking up the key or not has no bearing on the ability to collect apple rewards. The apples are the noisy rewards that *distract the agent from learning that picking up the key early on leads to a door-reward later*. In our experiments, we evaluate methods on 9 different environments representing combinations of 3 levels of apple reward mean and 3 levels of apple reward variance.

Implementation. The agent observes the top-down view of the current phase rendered in RGB and a binary variable indicating if the agent collected the key or not. The policy (θ) and the value functions (ψ and ϕ) are implemented by separate convolutional neural networks. The *meta-network* (η) computes the pairwise weight w_{ij} as follows: First, it embeds the observations s_i and s_j and the time difference $(j-i)$ into separate latent vectors. Then it takes the element-wise product of these three vectors to fuse them into a vector h_{ij}. Finally it maps h_{ij} to a scalar output that is bounded to $[0, 1]$ by sigmoid. We tuned hyperparameters for each method on the mid-level configuration $\langle \mu = 5, \sigma = 25 \rangle$ and kept them fixed for the other 8 configurations. Each method has a distinct set of parameters (e.g. outer-loop learning rates, λ). We referred to the original papers for the parameter ranges searched over. More details are in the appendix.

Empirical Results. Figure 5 presents learning curves for Meta-PWTD, Meta-PWR, and baselines in three KtD configurations (the remaining configurations are in the appendix). Learning curves are shown separately for the *total episode return* and the *door phase reward*, the latter a measure of success at the long-term credit assignment. Not unexpectedly, H-PWR which uses handcrafted pairwise weights performs the best. The gap in performance between H-PWR and the best fixed-λ shows that this domain provides a credit assignment challenge that the pairwise-weighted advantage estimate can help with. The TVT and A2C-RR methods used a low discount factor and so relied solely on their heuristics for learning to pick up the key, but neither appears to enable fast learning in this domain. In the door phase, Meta-PWR is generally the fastest learner after H-PWR. Meta-PWTD, though slower, achieves optimal performance. Although RGM performs third best in the door phase, it does not perform well overall, suggesting that the inflexibility of its single set of reward weights (vs. pairwise of Meta-PWR) forces a trade off between short and long-term credit assignment. In summary, Meta-PWR outperforms all the other methods and Meta-PWTD is comparable to the baselines.

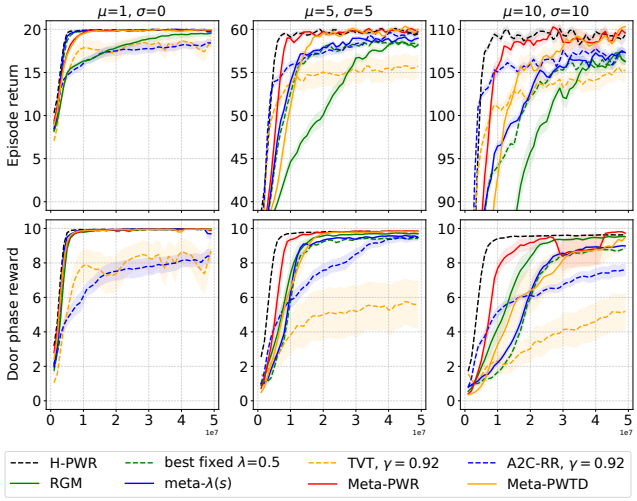

Figure 5: Learning curves for the KtD domain. Each column corresponds to a different configuration. The x-axis denotes the number of frames. The y-axis denotes the episode return in top row and the door phase reward in bottom row. The solid curves show the average over 10 independent runs and the shaded area shows the standard errors.

Figure 4 presents a visualization of the handcrafted weights for H-PWR (bottom left) and weights learned by Meta-PWR (bottom right). In each heatmap, the element on the i-th row and the j-th column denotes w_{ij}, the pairwise weight for computing the contribution of the reward upon transition to the j-th state to the return at the i-th state in the episode. In the heatmap of the handcrafted weights, the top-right area has non-zero weights because the rewards in the door phase depend on the actions selected in the key phase. The weights in the remaining part of the top rows are zero because those rewards do not depend on the the actions in the key phase. For the same reason, the weights in the middle-right area are zero as well. The weights in the rest of the area resemble the exponentially discounted weights with a discount factor of 0.92. This steep discounting helps fast learning of collecting apples. The learned weights largely resemble the handcrafted weights, which indicate that the meta-gradient procedure was able to simultaneously learn (1) the important rewards for the key phase are in the door phase, and (2) a quick-discounting set of weights within the apple phase that allows faster learning of collecting apples.

5.3 Experiments on Standard RL Benchmarks

Both the DAG and KtD domains are idealized credit assignment problems. However, in domains outside this idealized class, Meta-PWTD and -PWR may learn slower than

	Catch	Catch Noise	Catch Scale	Umbr. Length	Umbr. Distract	Cartpole	Discount Chain
A2C	5975	42221	56800	38050	37524	76874	3554
A2C-RR	**5950**	42295	57033	38083	37433	71506	3548
RGM	7849	48268	54421	40397	40159	119102	2444
Meta-PWTD	6096	**41106**	**48199**	**37973**	37226	65945	1040
Meta-PWR	5967	43076	49361	38168	**36554**	**61752**	**161**

Table 1: Total regret on selected `bsuite` domains (low is good).

baseline methods due to the additional complexity they introduce. To evaluate this possibility we compared them to baseline methods on `bsuite` (Osband et al. 2019) and Atari (Bellemare et al. 2013), both standard RL benchmarks. For these experiments, we did not compare to Meta-$\lambda(s)$ because it performed similarly to the fixed-λ baseline in previous experiments as noted in the original paper (Xu, van Hasselt, and Silver 2018).

`bsuite` is a set of unit-tests for RL agents: each domain tests one or more specific RL-challenges, such as exploration, memory, and credit assignment, and each contains several versions varying in difficulty. We selected all domains that are tagged by "credit assignment" and at least one other challenge. These domains are not designed solely as idealized credit assignment problems. We ran all methods for $100K$ episodes in each domain except *Cartpole*, which we ran for $50K$ episodes. Each run was repeated 3 times with different random seeds. Table 1 shows the total regret. Overall Meta-PWTD or -PWR achieved the lowest total regret in all domains except for *Catch*. It shows that Meta-PWTD and -PWR perform better than or comparably to the baseline methods even in domains without the idealized umbrella-like TCA structure.

To test scalability to high-dimensional environments, we conducted experiments on Atari. Atari games often have long episodes of more than 1000 steps thus episode truncation is required. However, the returns in RGM and Meta-PWR are not in a recursive additive form thus the common way of correcting truncated trajectories by bootstrapping from the value function is not applicable. TVT also requires full episodes for value transportation. Therefore, we excluded RGM, TVT, and Meta-PWR and only ran Meta-PWTD, A2C-RR and A2C. For each method we conducted hyperparameter search on a subset of 6 games and ran each method on 49 games with the fixed set of hyperparameters; see appendix for details. An important hyperparameter for the A2C baseline is λ, which was set to 0.95.

Figure 6 (inset) shows the median human-normalized score during training. Meta-PWTD performed slightly better than A2C over the entire period, and both performed better than A2C-RR Figure 6 shows the relative performance of Meta-PWTD over A2C. Meta-PWTD outperforms A2C in 30 games, underperforms in 14, and ties in 5. These results show that Meta-PWTD can scale to high-dimensional environments like Atari. We conjecture that Meta-PWTD provides a benefit in games with embedded umbrella problems but this is hard to verify directly.

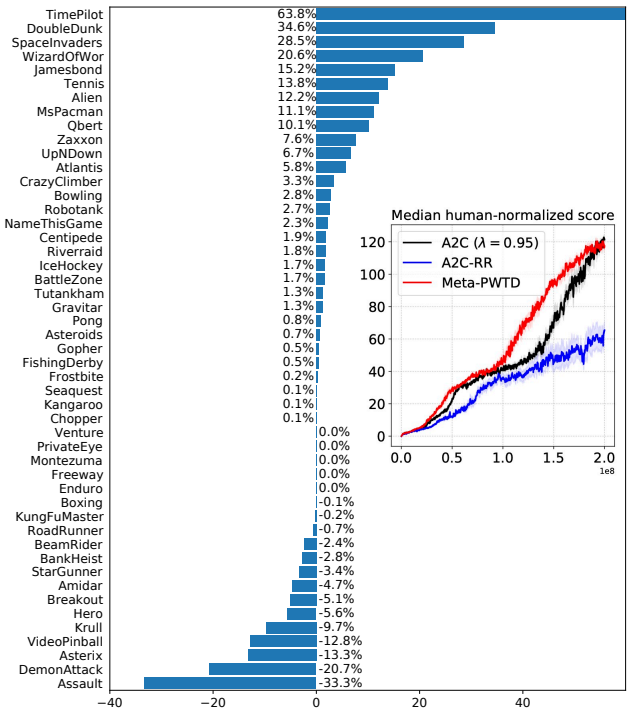

Figure 6: Relative performance of Meta-PWTD over A2C ($\lambda = 0.95$). All scores are averaged over 5 independent runs with different random seeds. Inset: Learning curves of median human normalized score of all 49 Atari games. Shaded area shows the standard error over 5 runs.

6 Conclusion

We presented two new advantage estimators with pairwise weight functions as parameters to be used in policy gradient algorithms, and a metagradient algorithm for learning the pairwise weight functions. Simple analysis and empirical work confirmed that the additional flexibility in our advantage estimators can be useful in domains with delayed consequences of actions, e.g., in umbrella-like problems. Empirical work also confirmed that the metagradient algorithm can learn the pairwise weights fast enough to be useful for policy learning, even in large-scale environments like Atari.

Acknowledgements

This work was supported by DARPA's L2M program as well as a grant from the Open Philanthropy Project to the Center for Human Compatible AI. Any opinions, findings, conclusions, or recommendations expressed here are those of the authors and do not necessarily reflect the views of the sponsors.

References

Arjona-Medina, J. A.; Gillhofer, M.; Widrich, M.; Unterthiner, T.; Brandstetter, J.; and Hochreiter, S. 2019. Rudder: Return decomposition for delayed rewards. In *Advances in Neural Information Processing Systems*, 13544–13555.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279.

Harutyunyan, A.; Dabney, W.; Mesnard, T.; Azar, M. G.; Piot, B.; Heess, N.; van Hasselt, H. P.; Wayne, G.; Singh, S.; Precup, D.; et al. 2019. Hindsight credit assignment. In *Advances in neural information processing systems*, 12467–12476.

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

Hung, C.-C.; Lillicrap, T.; Abramson, J.; Wu, Y.; Mirza, M.; Carnevale, F.; Ahuja, A.; and Wayne, G. 2019. Optimizing agent behavior over long time scales by transporting value. *Nature communications*, 10(1): 1–12.

Mesnard, T.; Weber, T.; Viola, F.; Thakoor, S.; Saade, A.; Harutyunyan, A.; Dabney, W.; Stepleton, T.; Heess, N.; Guez, A.; et al. 2020. Counterfactual Credit Assignment in Model-Free Reinforcement Learning. *arXiv preprint arXiv:2011.09464*.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 1928–1937.

Oh, J.; Hessel, M.; Czarnecki, W. M.; Xu, Z.; van Hasselt, H. P.; Singh, S.; and Silver, D. 2020. Discovering Reinforcement Learning Algorithms. *Advances in Neural Information Processing Systems*, 33.

Osband, I.; Doron, Y.; Hessel, M.; Aslanides, J.; Sezener, E.; Saraiva, A.; McKinney, K.; Lattimore, T.; Szepezvari, C.; Singh, S.; et al. 2019. Behaviour suite for reinforcement learning. *arXiv preprint arXiv:1908.03568*.

Rajendran, J.; Lewis, R.; Veeriah, V.; Lee, H.; and Singh, S. 2019. How Should an Agent Practice? *arXiv preprint arXiv:1912.07045*.

Raposo, D.; Ritter, S.; Santoro, A.; Wayne, G.; Weber, T.; Botvinick, M.; van Hasselt, H.; and Song, F. 2021. Synthetic Returns for Long-Term Credit Assignment. *arXiv preprint arXiv:2102.12425*.

Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; and Abbeel, P. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

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

Sutton, R. S.; McAllester, D. A.; Singh, S. P.; and Mansour, Y. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, 1057–1063.

van Hasselt, H.; Madjiheurem, S.; Hessel, M.; Silver, D.; Barreto, A.; and Borsa, D. 2020. Expected eligibility traces. *arXiv preprint arXiv:2007.01839*.

Veeriah, V.; Hessel, M.; Xu, Z.; Rajendran, J.; Lewis, R. L.; Oh, J.; van Hasselt, H. P.; Silver, D.; and Singh, S. 2019. Discovery of useful questions as auxiliary tasks. In *Advances in Neural Information Processing Systems*, 9306–9317.

Wang, Y.; Ye, Q.; and Liu, T.-Y. 2019. Beyond Exponentially Discounted Sum: Automatic Learning of Return Function. *arXiv preprint arXiv:1905.11591*.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4): 229–256.

Xu, Z.; van Hasselt, H.; Hessel, M.; Oh, J.; Singh, S.; and Silver, D. 2020. Meta-gradient reinforcement learning with an objective discovered online. *arXiv preprint arXiv:2007.08433*.

Xu, Z.; van Hasselt, H. P.; and Silver, D. 2018. Meta-gradient reinforcement learning. In *Advances in neural information processing systems*, 2396–2407.

Zahavy, T.; Xu, Z.; Veeriah, V.; Hessel, M.; Oh, J.; van Hasselt, H. P.; Silver, D.; and Singh, S. 2020. A self-tuning actor-critic algorithm. *Advances in Neural Information Processing Systems*, 33.

Zheng, Z.; Oh, J.; Hessel, M.; Xu, Z.; Kroiss, M.; Van Hasselt, H.; Silver, D.; and Singh, S. 2020. What Can Learned Intrinsic Rewards Capture? In *International Conference on Machine Learning*, 11436–11446. PMLR.

Zheng, Z.; Oh, J.; and Singh, S. 2018. On learning intrinsic rewards for policy gradient methods. In *Advances in Neural Information Processing Systems*, 4644–4654.