Metatrace Actor-Critic: Online Step-Size Tuning by Meta-Gradient Descent for Reinforcement Learning Control

Kenny Young[1,2] , **Baoxiang Wang**[1] and **Matthew E. Taylor**[1]
[1]Borealis AI, Edmonton, Alberta, Canada
[2]University of Alberta, Edmonton, Alberta, Canada
kjyoung@ualberta.ca, {baoxiang.wang, matthew.taylor}@BorealisAI.com

Abstract

Reinforcement learning (RL) has had many successes, but significant hyperparameter tuning is commonly required to achieve good performance. Furthermore, when nonlinear function approximation is used, non-stationarity in the state representation can lead to learning instability. A variety of techniques exist to combat this — most notably experience replay or the use of parallel actors. These techniques stabilize learning by making the RL problem more similar to the supervised setting. However, they come at the cost of moving away from the RL problem as it is typically formulated, that is, a single agent learning online without maintaining a large database of training examples. To address these issues, we propose Metatrace, a meta-gradient descent based algorithm to tune the step-size online. Metatrace leverages the structure of eligibility traces, and works for both tuning a scalar step-size and a respective step-size for each parameter. We empirically evaluate Metatrace for actor-critic on the Arcade Learning Environment. Results show Metatrace can speed up learning, and improve performance in non-stationary settings.

1 Introduction

In the supervised learning setting, there are a variety of optimization methods that build on stochastic gradient descent (SGD) for tuning neural network parameters. Applying ordinary SGD is difficult due to the sensitivity to the step-size parameter, which controls the magnitude of each update, as well as potential for wide variation in the gradient magnitude across different dimensions. Methods like RMSProp [Tieleman and Hinton, 2012] and ADAM [Kingma and Ba, 2014] aim to accelerate learning and improve stability by approximately accounting for higher order gradients of a fixed loss surface. While these methods are effective in the supervised case, they do not translate straightforwardly to online reinforcement learning (RL), where targets incorporate future estimates and subsequent observations are correlated.

The eligibility trace method is a key component used in RL to control the degree of bootstrapping from future estimates. However, this further complicates the analogy with supervised learning, as individual updates do not correspond to a gradient descent step toward any target on their own. Instead, the error signal is broken up into a series of updates such that only the sum of updates over time optimizes it. The appropriate analogy to the previously mentioned SGD techniques from supervised learning is not clear in this setting.

To apply these SGD techniques in the RL setting, a common strategy is to make the RL problem as close to the supervised learning problem as possible. Techniques that help to achieve this include multiple actors [Mnih *et al.*, 2016], large experience replay buffers [Mnih *et al.*, 2015], and separate online and target networks [Van Hasselt *et al.*, 2016]. These all help smooth gradient noise and mitigate non-stationarity. However, they do so at the cost of losing the advantages of online updating. These advantages include lower computation and memory use, learning immediately from new experience, and allowing a single agent to learn from its own trajectories.

We propose Metatrace, a novel set of algorithms that tune the step-size parameter online for the actor-critic algorithm. Metatrace applies meta-gradient descent, propagating gradients through the optimization algorithm itself, to derive step-size tuning algorithms specifically for the online RL control problem. These algorithms build on the incremental delta-bar-delta (IDBD) approach [Sutton, 1992] while solving an open issue of previous work. To be exact, we use an eligibility trace-like mechanism to allow the meta-objective to account for time-dependent weights in an online manner. We apply this insight to define algorithms for tuning a scalar step-size, as well as a vector of step-sizes. Finally, we propose a mixed version that leverages the benefits of both, and integrate the algorithm with entropy regularization and normalization of the metaupdate to improve performance in practice.

We evaluate Metatrace for actor-critic with eligibility traces ($AC(\lambda)$). Our experiments include a classic mountain car problem to demonstrate the efficiency of the algorithms and a drifting version of mountain car to explore the performance of meta-gradient descent with a non-stationary state representation. Finally, we evaluate Metatrace for training a deep neural network online in the original training-set games of the Arcade Learning Environment (ALE) [Bellemare *et al.*, 2013]. Without the use of either multiple actors or experience replay, our method learns faster and achieves better performance than an optimal fixed step-size in most of the games.

2 Related Work

Meta-learning has a long history dating back to Schmidhuber's thesis [Schmidhuber, 1987], where a genetic programming algorithm was applied to evolve better genetic programming algorithms. Our work is closely related to IDBD [Sutton, 1992] and autostep [Mahmood *et al.*, 2012], which are meta-gradient descent procedures for step-size tuning in the supervised learning case. Also closely related are SID and NOSID [Dabney, 2014], which analogize meta-gradient descent procedures for the *SARSA*(λ) method [Sutton and Barto, 2018]. Our approach differs primarily by explicitly accounting for time-varying weights in the optimization objective for the step-size. We also extend the approach to *AC*(λ) and to vector-valued step-sizes, as well as a mixed version which combines scalar and vector step-sizes.

Meta-gradient RL [Xu *et al.*, 2018] applies meta-gradient descent to optimize γ and λ in *AC*(λ). They use multiple parallel agents, deviating from the online RL case we focus on.

It is worth noting that previous works such as TIDBD and its extension AutoTIDBD [Kearney *et al.*, 2019] tune vector step-sizes online. That work focuses on *TD*(λ) for prediction, and explores both vector and scalar step-sizes. They demonstrate that both scalar and vector AutoTIDBD outperform ordinary TD(λ), while vector AutoTIDBD outperforms a variety of scalar step-size adaptation methods. Aside from focusing on control rather than prediction, our methods differ from TIDBD in the meta-objective optimized by the step-size tuning: they use one step TD error, while we use a more general multi-step objective.

Our neural network experiments draw inspiration from the work of Elfwing et al. [Elfwing *et al.*, 2018], to our best knowledge the only prior work to apply online RL with eligibility traces to train a deep neural network. In their case, the neural network is used as a function approximator in *SARSA*(λ). They do not explore step-size tuning.

3 Background

We consider the RL problem, where a learning agent interacting with an environment strives to maximize a reward signal. The problem is formalized as a Markov decision process $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$. We consider stochastic environments and stochastic policies. At each time-step the agent observes the state $S_t \in \mathcal{S}$ and selects an action $A_t \in \mathcal{A}$. Based on S_t and A_t, the next state S_{t+1} is generated by $p(S_{t+1}|S_t, A_t)$. The agent additionally observes a reward R_{t+1}, generated by $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. We focus on the control problem, where an agent learns, through interaction with the environment, a policy π that maximizes the expected return $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$, with discount factor $\gamma \in [0, 1]$.

Action-value methods like Q-learning are often used for RL control [Mnih *et al.*, 2015, Watkins and Dayan, 1992]. However, actor-critic (AC) methods, which more directly optimize the expected return, are becoming increasingly popular in Deep RL [Espeholt *et al.*, 2018]. We will focus on AC, but our proposed method is general enough to be applied to other RL algorithms. AC methods learn a state value function for the current policy, as well as a policy which attempts to maximize that value function. In particular, we will derive Meta-trace for actor-critic with eligibility traces, *AC*(λ) [Degris *et al.*, 2012, Schulman *et al.*, 2015].

We define the objective of *AC*(λ) in terms of the lambda return, which is recursively defined as

$$G^{\lambda}_{w,t} = R_{t+1} + \gamma \left((1-\lambda) V_w(S_{t+1}) + \lambda G^{\lambda}_{w,t+1} \right).$$

$G^{\lambda}_{w,t}$ bootstraps future evaluations to a degree controlled by λ. Here we define $G^{\lambda}_{w,t}$ for a fixed weight vector w; in Section 4, we will extend this to a time varying w_t. Also, define the TD-error $\delta_t = R_t + \gamma V_w(S_{t+1}) - V_w(S_t)$. Then we can expand $G^{\lambda}_{w,t}$ as the current state value estimate plus the sum of future discounted δ_t values as

$$G^{\lambda}_{w,t} = V_w(S_t) + \sum_{k=t}^{\infty} (\gamma\lambda)^{k-t} \delta_k. \tag{1}$$

TD(λ) can be understood as performing gradient descent along the mean squared error $\left(G^{\lambda}_{w,t} - V_w(S_t)\right)^2$ between the value function V_w and the lambda return $G^{\lambda}_{w,t}$. In the derivation of *TD*(λ), the target $G^{\lambda}_{w,t}$ is taken as a constant despite its dependence on w. For this reason, *TD*(λ) is often called a *semi-gradient* method. Intuitively, this reflects our desire to modify our current estimate to match our future estimates, not the other way around. *AC*(λ), combines this mean squared error objective with a policy improvement term, such that the combined objective represents a trade-off between the quality of our value estimates and the performance of our policy:

$$\mathcal{J}_{\lambda}(w) = \frac{1}{2} \left(\sum_{t=0}^{\infty} \left[G^{\lambda}_{w,t} - V_w(S_t) \right]^2 - \sum_{t=0}^{\infty} \log(\pi_w(A_t|S_t)) \left[G^{\lambda}_{w,t} - V_w(S_t) \right] \right). \tag{2}$$

As is in *TD*(λ), we aim to perform semi-gradient descent to minimize equation (2). Along with $G^{\lambda}_{w,t}$, $V_w(S_t)$ in the right sum is taken to be constant. Similar to previous arguments, the intuition is that we wish to improve our actor under the evaluation of our critic, instead of modifying our critic to make our actor's performance look better. With this caveat in mind, by the policy gradient theorem [Sutton *et al.*, 2000], the expectation of the gradient of the right term in equation (2) is approximately equal to the negated gradient of the expected return. This approximation is accurate to the extent that our advantage estimate $\left(G^{\lambda}_{w,t} - V_w(S_t)\right)$ is accurate. Descending the gradient of the right half of $\mathcal{J}_{\lambda}(w)$ is then ascending the gradient of an estimate of expected return. Taking the semi-gradient of equation (2) yields

$$\frac{\partial}{\partial w} \mathcal{J}_{\lambda}(w) = -\sum_{t=0}^{\infty} \left(\frac{\partial V_w(S_t)}{\partial w} \left[G^{\lambda}_{w,t} - V_w(S_t) \right] + \frac{1}{2} \frac{\partial \log(\pi_w(A_t|S_t))}{\partial w} \left[G^{\lambda}_{w,t} - V_w(S_t) \right] \right)$$

Now, define for compactness

$$U_w(S_t) \doteq V_w(S_t) + \frac{1}{2} \log(\pi_w(A_t|S_t))$$

so that, applying equation (1), we can rewrite

$$\frac{\partial}{\partial w}\mathcal{J}_\lambda(w) = -\sum_{t=0}^{\infty}\frac{\partial U_w(S_t)}{\partial w}\sum_{k=t}^{\infty}(\gamma\lambda)^{k-t}\delta_k$$
$$= -\sum_{t=0}^{\infty}\delta_t\sum_{k=0}^{t}(\gamma\lambda)^{t-k}\frac{\partial U_w(S_k)}{\partial w}$$

and define the eligibility trace at time t as $z_t = \sum_{k=0}^{t}(\gamma\lambda)^{t-k}\frac{\partial U_w(S_k)}{\partial w}$, such that

$$\frac{\partial}{\partial w}\mathcal{J}_\lambda(w) = -\sum_{t=0}^{\infty}\delta_t z_t. \tag{3}$$

Offline *AC*(λ) makes updates in the direction of equation (3). Online *AC*(λ), analogous to online *TD*(λ), is an approximation to this offline version (exact in the limit of quasi-static weights) that updates weights after every time-step. Advantages of the online version include making immediate use of new information, and being applicable to continual learning[1]. Online *AC*(λ) is defined by the following set of equations:

$$z_t = \gamma\lambda z_{t-1} + \frac{\partial U_{w_t}(S_t)}{\partial w_t}, \tag{4}$$
$$w_{t+1} = w_t + \alpha z_t\delta_t.$$

Where α is the step-size parameter, controlling the magnitude of each update. In practice, performance can be highly sensitive the the value of α, hence tuning it online as we do in this work is a worthwhile goal.

4 Algorithm

We present three variations of Metatrace for control using *AC*(λ): scalar (single α for all model weights), vector (one α per model weight), and finally a "mixed" version that attempts to leverage the benefits of both. Additionally, we discuss two practical improvements over the basic algorithm: normalization that helps to mitigate parameter sensitivity across problems and avoid divergence; and entropy regularization, which is commonly employed in AC to avoid premature convergence [Mnih *et al.*, 2016].

4.1 Scalar Metatrace for *AC*(λ)

For convenience, we define our step-size as $\alpha = e^\beta$, similar to IDBD [Sutton, 1992]. We desire to optimize α to allow our weights to efficiently track the non-stationary *AC*(λ) objective. To account for non-stationarity in the representation, we define the objective with time-dependent weights

$$\mathcal{J}^\beta_\lambda(w_0..w_\infty) = \frac{1}{2}\left(\sum_{t=0}^{\infty}\left[G^\lambda_t - V_{w_t}(S_t)\right]^2 - \sum_{t=0}^{\infty}\log(\pi_{w_t}(A_t|S_t))\left[G^\lambda_t - V_{w_t}(S_t)\right]\right). \tag{5}$$

G^λ_t with no subscript w is defined as $G^\lambda_t = V_{w_t}(S_t) + \sum_{k=t}^{\infty}(\gamma\lambda)^{k-t}\delta_k$, where $\delta_k = R_k + \gamma V_{w_k}(S_{k+1}) - V_{w_k}(S_k)$.

[1]Continual learning is the setting where an agent interacts with an environment indefinitely, with no distinct episodes.

Our derivation is similar to Section 4.3.1 of Dabney's thesis [Dabney, 2014], but explicitly accounts for the time-dependency of w_t. Regarding each w_t as a function of β, differentiating equation (5) under the same semi-gradient treatment used in *AC*(λ) yields

$$\frac{\partial}{\partial\beta}\mathcal{J}_\lambda(w_0..w_\infty)$$
$$= -\sum_{t=0}^{\infty}\frac{\partial U_{w_t}(S_t)}{\partial\beta}\left[G^\lambda_{w,t} - V_{w_t}(S_t)\right]$$
$$= -\sum_{t=0}^{\infty}\left\langle\frac{\partial U_{w_t}(S_t)}{\partial w_t}, \frac{\partial w_t}{\partial\beta}\right\rangle\sum_{k=t}^{\infty}(\gamma\lambda)^{k-t}\delta_k$$
$$= -\sum_{t=0}^{\infty}\delta_t\sum_{k=0}^{t}(\gamma\lambda)^{t-k}\left\langle\frac{\partial U_{w_t}(S_t)}{\partial w_t}, \frac{\partial w_t}{\partial\beta}\right\rangle.$$

To handle the above objective, we define an auxiliary eligibility trace as

$$z_{\beta,t} \doteq \sum_{k=0}^{t}(\gamma\lambda)^{t-k}\left\langle\frac{\partial U_{w_k}(S_k)}{\partial w_k}, \frac{\partial w_k}{\partial\beta}\right\rangle \tag{6}$$

such that

$$\frac{\partial}{\partial\beta}\mathcal{J}_\lambda(w_0..w_\infty) = -\sum_{t=0}^{\infty}\delta_t z_{\beta,t}. \tag{7}$$

As z itself is a sum of first-order derivatives with respect to w, we approximate $\frac{\partial z_t}{\partial w} = 0$. Since $\log(\pi_{w_t}(A_t|S_t))$ necessarily involves some non-linearity, this is a first-order approximation even in the linear case. Furthermore, in the control case, the weights affect action selection. Action selection, in turn, affects expected weight updates. Hence, there are additional higher order effects of modifying β on the expected weight updates. As is in Dabney's thesis [Dabney, 2014], we do not account for this effect. We leave the question open of how to account for this interaction in the online setting, and to what extent it would make a difference in the algorithm. With the above reasoning, let $h(t)$ denote $\partial w_t/\partial\beta$, we update $h(t)$ as

$$h(t+1) = \frac{\partial}{\partial\beta}[w_t + \alpha\delta_t z_t]$$
$$\approx h(t) + \alpha\delta_t z_t + \alpha z_t\frac{\partial\delta_t}{\partial\beta}$$
$$= h(t) + \alpha z_t\left[\delta_t + \left\langle\frac{\partial\delta_t}{\partial w_t}, \frac{\partial w_t}{\partial\beta}\right\rangle\right]$$
$$= h(t) + \alpha z_t\left[\delta_t + \left\langle\frac{\partial\delta_t}{\partial w_t}, h(t)\right\rangle\right]. \tag{8}$$

Following equation (6)-(8), our Scalar Metatrace for *AC*(λ) is described by

$$z_\beta \leftarrow \gamma\lambda z_\beta + \left\langle\frac{\partial U_{w_t}(S_t)}{\partial w_t}, h\right\rangle,$$
$$\beta \leftarrow \beta + \mu z_\beta\delta_t,$$
$$h \leftarrow h + e^\beta z\left[\delta_t + \left\langle\frac{\partial\delta_t}{\partial w_t}, h\right\rangle\right].$$

Where μ is the meta-step-size parameter, controlling the update magnitude for the step-size parameter itself. The update

Algorithm 1 Normalized Scalar Metatrace for actor-critic

$h \leftarrow 0$, $\beta \leftarrow \beta_0$, $v \leftarrow 0$
for each episode **do**
  $z_\beta \leftarrow 0$, $u \leftarrow 0$
  while episode not complete **do**
    receive $V_{w_t}(S_t), \pi_{w_t}(S_t|A_t), \delta_t, z$
    $U_{w_t} \leftarrow V_{w_t} + \frac{1}{2}\log(\pi_{w_t}(A_t|S_t))$
    $z_\beta \leftarrow \gamma\lambda z_\beta + \left\langle \frac{\partial U_{w_t}}{\partial w_t}, h \right\rangle$
    $\Delta_\beta \leftarrow z_\beta\delta_t + \psi\left\langle \frac{\partial H_{w_t}(S_t)}{\partial w_t}, h \right\rangle$
    $v \leftarrow \max(|\Delta_\beta|, v + \mu(|\Delta_\beta| - v)$
    $\beta \leftarrow \beta + \mu\frac{\Delta_\beta}{(v \text{ if } v>0 \text{ else } 1)}$
    $u \leftarrow \max(e^\beta \left|\frac{\partial U_{w_t}}{\partial w_t}\right|^2, u+(1-\gamma\lambda)(e^\beta \left|\frac{\partial U_{w_t}}{\partial w_t}\right|^2 - u))$
    $M \leftarrow \max(u, 1)$
    $\beta \leftarrow \beta - \log(M)$
    $h \leftarrow h + e^\beta(z(\delta_t + \left\langle \frac{\partial \delta_t}{\partial w_t}, h \right\rangle) + \psi\frac{\partial H_{w_t}(S_t)}{\partial w_t})$
    output $\alpha = e^\beta$
  end while
end for

to z_β is an online computation of equation (6). The update to β is exactly analogous to the $AC(\lambda)$ weight update but with equation (7) in place of equation (3). The update to h computes equation (8) online. The full algorithm is detailed in Algorithm 1, including entropy regularization and normalization to be described shortly.

4.2 Vector Metatrace for *AC*(λ)

For the vector case, we denote $\alpha_i = e^{\beta_i}$ to be the i^{th} element of a vector of step-sizes. Each element α_i corresponds to one weight such that the update for each weight element in $AC(\lambda)$ will use the associated α_i. Having a separate α_i for each weight enables the algorithm to individually adjust how quickly it tracks each feature. This is particularly important when the state representation is non-stationary, including the case where neural network-based models are used. We expect our algorithm to assign high step-size to fast-changing, useful features while annealing the step-size of features that are either mostly stationary or not useful to avoid tracking noise.

Taking each w_t to be a function of β_i for all i, and following IDBD [Sutton, 1992] in using the approximation $\frac{\partial w_{i,t}}{\partial \beta_j} = 0$ for all $i \neq j$, differentiating equation (2) with respect to β_i yields

$$\begin{aligned}\frac{\partial}{\partial \beta_i}\mathcal{J}_\lambda^\beta(w_0..w_\infty) &= -\sum_{t=0}^{\infty}\frac{\partial U_{w_t}(S_t)}{\partial \beta_i}\left[G_t^\lambda - V_{w_t}(S_t)\right] \\ &\approx -\sum_{t=0}^{\infty}\frac{\partial U_{w_t}(S_t)}{\partial w_{i,t}}\frac{\partial w_{i,t}}{\partial \beta_i}\sum_{k=t}^{\infty}(\gamma\lambda)^{k-t}\delta_k \\ &= -\sum_{t=0}^{\infty}\delta_t\sum_{k=0}^{t}(\gamma\lambda)^{t-k}\frac{\partial U_{w_k}(S_k)}{\partial w_{i,k}}\frac{\partial w_{i,k}}{\partial \beta_i}.\end{aligned}$$

Similarly we define an auxiliary eligibility trace, which is vectorized in this section, with elements

$$z_{\beta,i,t} \doteq \sum_{k=0}^{t}(\gamma\lambda)^{t-k}\frac{\partial U_{w_k}(S_k)}{\partial w_{i,k}}\frac{\partial w_{i,k}}{\partial \beta_i},$$

such that

$$\frac{\partial}{\partial \beta_i}\mathcal{J}_\lambda(w_0..w_\infty) \approx -\sum_{t=0}^{\infty}\delta_t z_{\beta,i,t}.$$

To compute $h(t) \doteq \frac{\partial w_{i,t}}{\partial \beta_i}$ we use the immediate generalization of the scalar case and the approximation $\frac{\partial w_{i,t}}{\partial \beta_j} = 0$, which yields

$$\begin{aligned}&h_i(t+1) \\ &\approx h_i(t) + \alpha_i z_{i,t}\left[\delta_t + \left\langle \frac{\partial \delta_t}{\partial w_t}, \frac{\partial w_t}{\partial \beta_i} \right\rangle\right] \\ &\approx h_i(t) + \alpha_i z_{i,t}\left[\delta_t + \frac{\partial \delta_t}{\partial w_{i,t}}\frac{\partial w_{i,t}}{\partial \beta_i}\right] \\ &= h_i(t) + \alpha_i z_{i,t}\left[\delta_t + \frac{\partial \delta_t}{\partial w_{i,t}}h_i(t)\right].\end{aligned}$$

Therefore, Vector Metatrace for $AC(\lambda)$ is described by

$$\begin{aligned}z_\beta &\leftarrow \gamma\lambda z_\beta + \frac{\partial U_{w_t}(S_t)}{\partial w_t} \odot h, \\ \beta &\leftarrow \beta + \mu z_\beta \delta_t, \\ h &\leftarrow h + e^\beta \odot z \odot \left(\delta_t + \frac{\partial \delta_t}{\partial w_t} \odot h\right),\end{aligned}$$

where $\odot$ denotes the Hadamard product. The full algorithm is detailed in Algorithm 2.

Algorithm 2 Normalized Vector Metatrace for actor-critic

$h \leftarrow 0$, $\overrightarrow{\beta} \leftarrow \beta_0$, $\overrightarrow{v} \leftarrow 0$
for each episode **do**
  $\overrightarrow{z_\beta} \leftarrow 0$, $u \leftarrow 0$
  while episode not complete **do**
    receive $V_{w_t}(S_t), \pi_{w_t}(A_t|S_t), \delta_t, z$
    $U_{w_t} \leftarrow V_{w_t} + \frac{1}{2}\log(\pi_{w_t}(A_t|S_t))$
    $\overrightarrow{z_\beta} \leftarrow \gamma\lambda\overrightarrow{z_\beta} + \frac{\partial U_{w_t}}{\partial w_t} \odot \overrightarrow{h}$
    $\Delta_{\overrightarrow{\beta}} \leftarrow \overrightarrow{z_\beta}\delta_t + \psi\frac{\partial H_{w_t}(S_t)}{\partial w_t} \odot \overrightarrow{h}$
    $\overrightarrow{v} \leftarrow \max(|\Delta_{\overrightarrow{\beta}}|, \overrightarrow{v} + \mu(|\Delta_{\overrightarrow{\beta}}| - \overrightarrow{v})$
    $\overrightarrow{\beta} \leftarrow \overrightarrow{\beta} + \mu\frac{\Delta_{\overrightarrow{\beta}}}{(\overrightarrow{v} \text{ where } \overrightarrow{v}>0 \text{ elsewhere } 1)}$
    $u \leftarrow \max\left(\left\langle e^{\overrightarrow{\beta}} \left|\frac{\partial U_{w_t}}{\partial w_t}^2\right.\right\rangle, u + (1-\gamma\lambda)\left(\left\langle e^{\overrightarrow{\beta}} \left|\frac{\partial U_{w_t}}{\partial w_t}^2\right.\right\rangle - u\right)\right)$
    $M \leftarrow \max(u, 1)$
    $\overrightarrow{\beta} \leftarrow \overrightarrow{\beta} - \log(M)$
    $\overrightarrow{h} \leftarrow \overrightarrow{h} + e^{\overrightarrow{\beta}} \odot\left(\overrightarrow{z} \odot\left(\delta_t + \frac{\partial \delta_t}{\partial w_t} \odot \overrightarrow{h}\right) + \psi\frac{\partial H_{w_t}(S_t)}{\partial w_t}\right)$
    output $\overrightarrow{\alpha} = e^{\overrightarrow{\beta}}$
  end while
end for

We now describe three variations based on Scalar Meta-

trace and Vector Metatrace that can be helpful in practice.

Mixed Metatrace

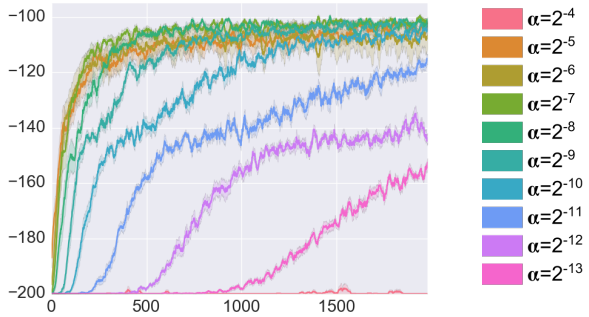
We explore a mixed algorithm where a vector correction to the step-size $\overrightarrow{\beta}$ is learned for each weight and added to a global value $\hat{\beta}$ which is learned collectively for all weights. The idea is that through $\hat{\beta}$ Mixed Metatrace can make use of all available data to learn a good base step-size, while $\overrightarrow{\beta}$ can be adjusted per parameter where it is useful. We omit the algorithm for this case for brevity: it is a straightforward combination of the scalar and vector algorithms.

Entropy Regularization

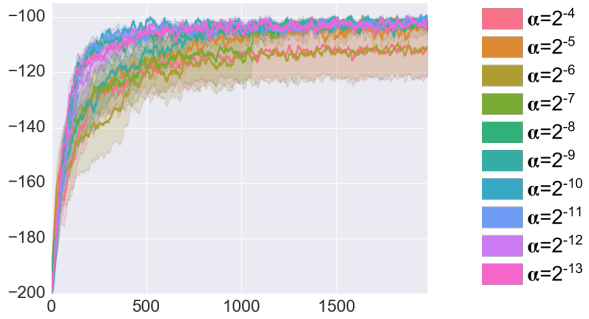
In practice, it is often helpful to add an entropy bonus, $-\psi \sum_{t=0}^{\infty} H_w(S_t)$, to the objective function to discourage premature convergence [Mnih *et al.*, 2016]. We found it helpful to also include the entropy bonus in the meta-objective, which requires modification to the step-size tuning procedure. This modification is included in Algorithms 1 and 2.

Normalization

As discussed in autostep [Mahmood *et al.*, 2012], even in the supervised learning case, the meta-gradient descent algorithms discussed so far can be unstable and sensitive to the parameter μ. To combat this, we normalize the meta-update, and clip the step-size to avoid overshooting in a manner analogous to the procedure introduced in autostep.



(a) No step-size tuning



(b) Normalized Scalar Metatrace with $\mu = 2^{-7}$

Figure 1: Return v.s. training episodes on mountain car for a variety of initial α values with and without step-size tuning. Each curve shows the average of 10 random seeds, smoothed over 20 episodes.

5 Experiments and Results

5.1 Mountain Car

We begin by testing Scalar Metatrace on the mountain car domain from OpenAI Gym [Brockman *et al.*, 2016]. A reward of -1 is given for each time-step until the goal is reached or 200 steps are taken. In this experiment, we use linear function approximation with tile-coding for state representation. The tiling size is 10×10 where 16 tilings create a feature vector of size 1600. The learning algorithm is based on $AC(\lambda)$ with γ fixed to 0.99 and λ fixed to 0.8 in all experiments. For mountain car, we use no entropy regularization.

Figure 1a shows the results for a variety of α values without step-size tuning. α values were chosen as powers of 2 which range from excessively low to excessively high (where the algorithm diverges). Figure 1b shows the results for Normalized Scalar Metatrace with different initial α values. In contrast to the untuned baseline, the learning curves of different αs ares very similar. Though the μ value has to be specified, similar results hold for a broad range of μ values below $\mu \geq 2^{-6}$, above which the algorithm becomes unstable.

5.2 Drifting Mountain Car

Motivated by the noisy, non-stationary experiment from IDBD [Sutton, 1992], we create drifting mountain car by adding noise and non-stationarity to the state representation. The environment is intended to provide a proxy for the issues inherent in representation learning such as using a neural network function approximator. We use the same tiling features as in the original mountain car experiment. However, at each time-step, each feature has a chance to randomly flip from indicating activation with 1 to -1 and vice-versa. We define a uniform flipping probability per time step across all features that we refer to as the drift rate, which is 6×10^{-6} in our experiment. Additionally, we add 32 noisy features which are 1 or 0 with probability 0.5 for every time-step.

Due to the non-stationarity, an arbitrarily small α is not asymptotically optimal, as it is unable to track the changing features. Because of the noise, a scalar α will be suboptimal as well. Nonzero α_i values for noisy features lead to weight updates when that feature is active in a particular state. This random fluctuation adds noise to the learning process. The noise can be avoided by annealing associated α_is to zero, which is what Vector and Mixed Metatrace are expected to achieve. We demonstrate the effectiveness in Figure 2, for Scalar, Vector, and Mixed Metatrace methods along with a baseline with no online tuning. The best μ value for each method was selected from $\{2^{-i} | i \in \{6, ..., 11\}\}$. We observe that Mixed Metatrace maintains the most stable performance in the presence of non-stationarity.

To better understand the algorithm, we illustrate the evolution of various elements of the log step-size β under Mixed Metatrace, in Figure 3. Value βs refer to the log step-sizes of weights associated with the value function while policy βs refer to those associated with the policy. Noisy βs refer to the weights attached to noisy features, while informative βs refer to those associated with the tile-coding which are expected

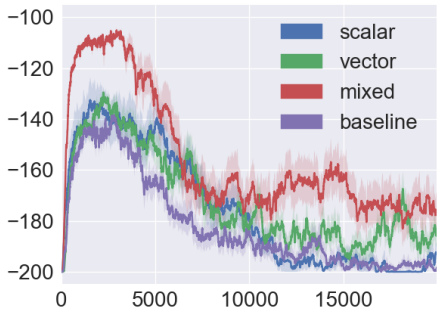


Figure 2: Return v.s. training episodes on drifting mountain car. Each curve shows the average of 20 random seeds, smoothed over 40 episodes.

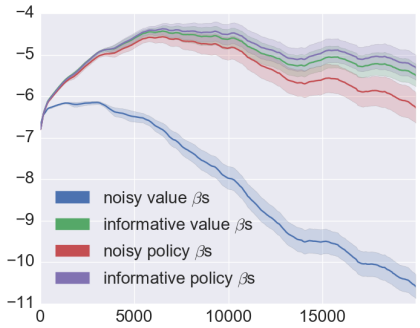
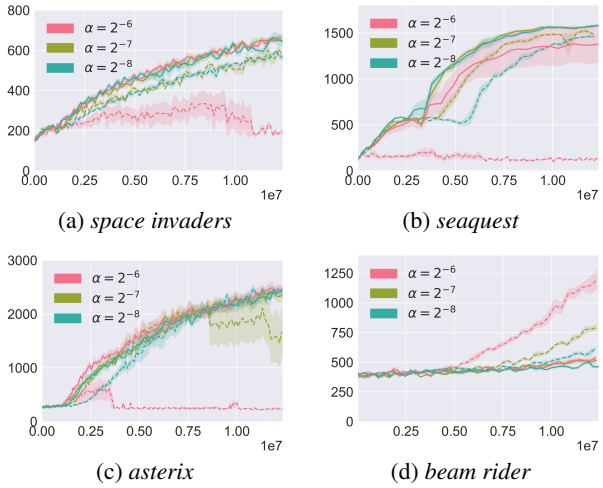


Figure 3: Evolution of average β values for various weights on drifting mountain car for initial $\alpha = 2^{-10}$, $\mu = 2^{-10}$. Each curve shows the average of 20 repeats, smoothed over 40 episodes.

to be useful. While all the step-sizes associated with noisy features are suppressed compared to the informative features, we observe a much stronger separation between the noisy and informative features for the value function than for the policy. One possible reason for this is that small errors in the value function have far more impact on optimizing the objective J_λ^β in mountain car than small imperfections in the policy. Learning a good policy requires fine-grained ability to distinguish the value of similar states, as individual actions will have a relatively minor impact on the car. On the other hand, errors in the value function in one state have a large negative impact on both the value learning of other states and the ability to learn a good policy.

5.3 Arcade Learning Environment

Finally, we test our algorithm with the 5 original training-set games of the ALE [Bellemare *et al.*, 2013]. We use a convolutional neural network architecture similar to that used in DQN [Mnih *et al.*, 2013], with SiLU and dSiLU activations to improve the performance in the online setting [Elfwing *et al.*, 2018]. The policy and value network share all but the final output layer. We fix $\mu = 1 \times 10^{-3}$ and run all experiments up to 1.25×10^7 observed frames. We use sticky actions to add stochasticity to the environments [Machado *et al.*, 2017]. Figure 4 shows the results of the ALE experiments.



(a) *space invaders* (b) *seaquest*

(c) *asterix* (d) *beam rider*

Figure 4: Return v.s. learning steps for the ALE games. Each curve is the average of 5 random seeds, smoothed over 40 episodes. Solid lines correspond to Metatrace with meta-step-size parameter $\mu = 1 \times 10^{-3}$ while dotted lines correspond to the fixed step-size baseline.

Metatrace significantly decreases the sensitivity to the initial choice of α. In most cases, Metatrace also improves the performance while accelerating learning. In *space invaders*, each α value tested outperforms the best α without online tuning. In *seaquest*, two of the three do. In *asterix*, the final performance of all initial αs with Metatrace was similar to the best-untuned α value, but the learning process is much faster. In *beam rider*, however, using no online tuning results in faster learning, especially for a well-chosen initial α. This result is possibly due to the high α sensitivity and high sample complexity in the game. The impact of changes in α only occurs after a very long period, making the online tuning especially difficult. In future work, it would be interesting to explore how Metatrace could be made robust to this issue.

6 Conclusion

We propose Metatrace, a novel set of algorithms based on meta-gradient descent for reinforcement learning, which performs step-size tuning for $AC(\lambda)$. Our result extends the methods of IDBD [Sutton, 1992] and autostep [Mahmood *et al.*, 2012] from supervised learning to reinforcement learning, which involves several technical challenges. To achieve this, we derive an eligibility trace-like mechanism to allow the meta-objective to account for time-dependent weights. We show that that Scalar Metatrace improves robustness to initial step-size choice, while Mixed Metatrace facilitates learning in an RL problem with non-stationary state representation. We also empirically test our approach for training a neural network online for several games in the Arcade Learning Environment, with nonlinear function approximation, where the state representation is inherently unstable. In most of the games, Metatrace allows a range of initial step-sizes to learn faster. Metatrace can also achieve better performance compared to the best-fixed choice of the step-size.

References

[Bellemare *et al.*, 2013] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[Brockman *et al.*, 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[Dabney, 2014] William C Dabney. Adaptive step-sizes for reinforcement learning. *PhD thesis, University of Massachusetts Amherst*, 2014.

[Degris *et al.*, 2012] Thomas Degris, Patrick M Pilarski, and Richard S Sutton. Model-free reinforcement learning with continuous action in practice. In *American Control Conference (ACC), 2012*, pages 2177–2182. IEEE, 2012.

[Elfwing *et al.*, 2018] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 2018.

[Espeholt *et al.*, 2018] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

[Kearney *et al.*, 2019] Alex Kearney, Vivek Veeriah, Jaden Tavnik, Patrick M Pillarski, and Richard S Sutton. Learning feature relevance through step size adaptation in temporal-difference learning. *arXiv preprint arxiv:1903.03252*, 2019.

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

[Machado *et al.*, 2017] Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *arXiv preprint arXiv:1709.06009*, 2017.

[Mahmood *et al.*, 2012] Ashique Rupam Mahmood, Richard S Sutton, Thomas Degris, and Patrick M Pilarski. Tuning-free step-size adaptation. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 2121–2124. IEEE, 2012.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[Mnih *et al.*, 2016] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[Schmidhuber, 1987] Jürgen Schmidhuber. Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook. *PhD thesis, Technische Universität München.*, 1987.

[Schulman *et al.*, 2015] 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.

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

[Sutton *et al.*, 2000] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[Sutton, 1992] Richard S Sutton. Adapting bias by gradient descent: An incremental version of delta-bar-delta. In *AAAI*, pages 171–176, 1992.

[Tieleman and Hinton, 2012] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-RMSProp: Divide the gradient by a running average of its recent magnitude. *Coursera: Neural networks for machine learning*, 4(2):26–31, 2012.

[Van Hasselt *et al.*, 2016] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 2, page 5. Phoenix, AZ, 2016.

[Watkins and Dayan, 1992] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[Xu *et al.*, 2018] Zhongwen Xu, Hado P van Hasselt, and David Silver. Meta-gradient reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 2402–2413. Curran Associates, Inc., 2018.