Forward Actor-Critic for Nonlinear Function Approximation in Reinforcement Learning

Vivek Veeriah*
Dept. of Computing Science
University of Alberta
Edmonton, Canada
vivekveeriah@ualberta.ca

Harm van Seijen†
Dept. of Computing Science
University of Alberta
Edmonton, Canada
harm.vanseijen@ualberta.ca

Richard S. Sutton
Dept. of Computing Science
University of Alberta
Edmonton, Canada
rsutton@ualberta.ca

ABSTRACT

Multi-step methods are important in reinforcement learning (RL). Eligibility traces, the usual way of handling them, works well with linear function approximators. Recently, van Seijen (2016) had introduced a delayed learning approach, *without* eligibility traces, for handling the multi-step λ-return with nonlinear function approximators. However, this was limited to action-value methods. In this paper, we extend this approach to handle n-step returns, generalize this approach to policy gradient methods and empirically study the effect of such delayed updates in control tasks. Specifically, we introduce two novel forward actor-critic methods and empirically investigate our proposed methods with the conventional actor-critic method on mountain car and pole-balancing tasks. From our experiments, we observe that forward actor-critic dramatically outperforms the conventional actor-critic in these standard control tasks. Notably, this forward actor-critic method has produced a new class of multi-step RL algorithms without eligibility traces.

Keywords

Reinforcement Learning, Actor-Critic, Policy Gradient, Nonlinear Function Approximation, Incremental Learning

1. INTRODUCTION

Reinforcement Learning (RL)[11, 12] is a problem setting where a learner learns to map actions to situations, in order to maximize a numerical reward signal. Temporal-Difference (TD)[10] learning is the popular algorithm for handling RL problems. Typically, a TD method estimates the quality of an action-selection strategy from online experiences and continually improves them by performing certain sample updates.

These update targets usually contain information extending multiple time steps into the future and often play a key role in the speed and efficiency of learning. λ-return is a multi-step update target that is typically used in TD learning and it is defined as a weighted average of rewards and value estimates. The λ parameter determines the weights for averaging between these components and can be seen as a parameter for controlling the bias-variance tradeoff.

TD learning, from a theoretical standpoint, can be defined as learning from stochastic gradient descent updates towards an update target, like the λ-return. Using such an update target will require the learning algorithm to gather information from time steps extending into the future, which implies that this method needs to look *forward* in time, and so this approach is called the *forward view*[14]. Naïvely implementing this will result in an algorithm that does not learn continually from online experiences — which seriously undermines the usability of this method in many real-world ongoing applications. Eligibility traces were introduced specifically to address this challenge.

With eligibility traces, TD learning can utilize multi-step update targets, like λ-return, *and* still learn incrementally from online experiences. Learning with eligibility traces propagates information *back* through time, and for this reason this approach is called the *backward view*[11].

Eligibility traces work well with linear function approximators. Recently, however, van Seijen (see [13]) showed that eligibility traces can cause divergence while learning with nonlinear function approximators, like neural networks. The underlying cause behind this instability is because of the deviations between the updates of forward and backward views. Specifically, eligibility traces in the case of nonlinear function approximators can produce updates that deviate from that of the forward view, thereby violating the equivalence between forward and backward views. Also, it is not prudent to avoid using nonlinear function approximators — they play a key role for scaling up reinforcement learning approaches to many difficult domains, particularly, in robotics.

Towards addressing this challenge, van Seijen introduced a multi-step action-value learning algorithm *without* eligibility traces. This is called forward Sarsa(λ), where the learning takes place incrementally through delayed multi-step updates and it occupies a middle ground between forward and backward views. In the case of nonlinear function approximators, this action-value learning algorithm performs substantially better than a conventional Sarsa(λ) algorithm. However, this work focused on designing an action-value method rather than a general policy gradient method. This is a drawback that needs to be addressed, because policy gradient methods with nonlinear function approximators are important for many control applications, like intelligent prosthesis, self-driving cars etc.

*Corresponding author
†Work done while the author was with University of Alberta

Appears in: *Proc. of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), S. Das, E. Durfee, K. Larson, M. Winikoff (eds.), May 8–12, 2017, São Paulo, Brazil.*
Copyright © 2017, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

In this paper, we extend van Seijen's approach to handle n-step returns, generalize this for policy gradient methods and empirically study the effect of delayed updates in two standard control tasks, mountain car and pole-balancing. Specifically, we introduce a policy gradient algorithm called the forward actor-critic, which utilizes multi-step update targets without any complications arising from eligibility traces. Moreover, this forward actor-critic method can be generalized to handle *any* complex backup, since it directly implements the forward view. As a direct consequence, this leads to the creation of a new class of forward multi-step policy gradient algorithms *without* eligibility traces. Unlike the conventional forward view methods, this proposed forward actor-critic method is computationally efficient and can learn *incrementally from online experiences*. Though the learning updates happen after a certain delay, it is outweighed by the substantial performance improvements obtained over the conventional actor-critic method with non-linear function approximators. However, for linear function approximators, there is no real advantage for this forward actor-critic method.

2. BACKGROUND

2.1 Reinforcement Learning Framework

RL tasks are formulated as *Markov Decision Processes* (MDPs) which are usually described as a tuple $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, consisting of $\mathcal{S}$, set of all states; $\mathcal{A}$, set of all actions; $p(s'|s,a)$, a transition probability function, which gives the probability of transition to the next state $s' \in \mathcal{S}$ from a given current state $s \in \mathcal{S}$ by selecting an action $a \in \mathcal{A}$; $r(s,a,s')$, the reward function that gives the expected reward for a transition from state $s \in \mathcal{S}$ to $s' \in \mathcal{S}$ by taking action $a \in \mathcal{A}$; γ is the discount factor, that specifies the relative importance between immediate and long term rewards. In episodic problems, the MDP can be viewed as having special states called *terminal states*, which terminate an episode. Such states ease the mathematical notations as they could be viewed as a state with a single action that results in a reward of 0 and transition to itself. The return G_t at a time instance t is defined as the discounted sum of rewards after time t:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$
$$G_t = \sum_{i=1}^{\infty} \gamma^{i-1} R_{t+i}$$

where R_{t+1} denotes the reward received after taking an action $A_t = a$ in state $S_t = s$.

Actions are taken at discrete time steps $t = 0, 1, 2, \cdots$ according to a *policy* $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0,1]$ which defines a selection probability for each action conditioned on the state. Each policy π has a corresponding state-value function $v_\pi(s)$, that maps each state $s \in \mathcal{S}$ to the expected return G_t from that state by following the policy π,

$$v_\pi(s) = \mathbb{E}_\pi \Big[G_t | S_t = s \Big]$$

2.2 Temporal-Difference Learning

Temporal-Difference (TD) learning aims at learning the state-value (or the action-value function) based on an update rule derived from stochastic gradient descent[2]. Let $\hat{V}(s|\boldsymbol{w})$ be an estimate of the true value function $v_\pi(s)$ parametrized by a weight vector $\boldsymbol{w} \in \mathbb{R}^n$. For a given set of samples $\{S_1, S_2, \cdots, S_n\}$ experienced by the learning algorithm, a squared error function can be defined as:

$$L(\boldsymbol{w}) := \frac{1}{2} \sum_{i=1}^{n} \big[v_\pi(S_i) - \hat{V}(S_i|\boldsymbol{w}) \big]^2$$

Minimizing this error function by stochastic gradient descent for an online sequence of samples experienced by the learning algorithm will yield the following update rule:

$$\begin{aligned} \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \alpha_v \frac{1}{2} \nabla_{\boldsymbol{w}} \big[v_\pi(S_t) - \hat{V}(S_t|\boldsymbol{w}_t) \big]^2 \\ &= \boldsymbol{w}_t + \alpha_v \big[v_\pi(S_t) - \hat{V}(S_t|\boldsymbol{w}_t) \big] \nabla_{\boldsymbol{w}} \hat{V}(S_t|\boldsymbol{w}_t) \end{aligned}$$

where $\alpha_v > 0$ is the step-size parameter and controls the magnitude of this update rule.

Generally, $v_\pi(S_t)$ is an unknown quantity. In TD learning, an estimate of $v_\pi(S_t)$, like G_t^λ, $G_t^{(n)}$ or G_t, is typically used. This estimate can be viewed as an *update target* U_t:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha_v \big[U_t - \hat{V}(S_t|\boldsymbol{w}_t) \big] \nabla_{\boldsymbol{w}} \hat{V}(S_t|\boldsymbol{w}_t) \qquad (1)$$

As these update targets require information spanning into the future time-steps, the methods that utilize this update rule are called *forward view* methods. In order to learn incrementally from experiences, the idea of *eligibility traces* were introduced. The learning methods that learn incrementally from ongoing experiences using eligibility traces are called *backward view* methods.

2.3 Actor-Critic Methods

Actor-Critic methods are special forms of policy gradient methods[1, 9] and are usually viewed as an architecture for solving RL problems.

The actor learns an action-selection policy $\pi(A_t|S_t, \boldsymbol{\theta})$, parameterized by $\boldsymbol{\theta} \in \mathbb{R}^m$, that maximizes the total expected reward in a RL control task. On the other hand, the critic learns a value function $\hat{V}(S_t|\boldsymbol{w})$ with respect to this policy and is parameterized by $\boldsymbol{w} \in \mathbb{R}^n$. At every time step, the critic critiques the actor for selecting an action by generating a TD error $\delta_t = R_{t+1} + \gamma \hat{V}(S_{t+1}|\boldsymbol{w}) - \hat{V}(S_t|\boldsymbol{w})$. This δ_t drives the learning in both actor and critic modules.

Typically, the critic utilizes the TD learning algorithm and its update rules are the same as described in the previous section.

For a given set of experienced states $\{S_1, S_2, \cdots, S_n\}$ by a learning agent, an optimization function (Bhatnagar et. al., 2009) can be defined for the actor module as:

$$L(\boldsymbol{\theta}) := \mathbb{E}_\pi \Big[q_\pi(S_t, A_t) - \hat{V}(S_t|\boldsymbol{w}) \Big]$$

Minimizing this error function by stochastic gradient descent for an online sequence of samples experienced will result in the following update rule:

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha_\pi \big[q_\pi(S_t, A_t) - \hat{V}(S_t|\boldsymbol{w_t}) \big] \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)} \\ &= \boldsymbol{\theta}_t + \alpha_\pi \big[U_t - \hat{V}(S_t|\boldsymbol{w_t}) \big] \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)} \end{aligned} \qquad (2)$$

where $\alpha_\pi > 0$ is the step-size parameters for the actor module. In order to learn incrementally from online experiences, eligibility traces can be used for both the actor and

Algorithm 1 Conventional Actor-Critic

INPUT: $\alpha_v, \alpha_\pi, \lambda, \gamma, \boldsymbol{w}_{init}, \boldsymbol{\theta}_{init}$

$\boldsymbol{w} \leftarrow \boldsymbol{w}_{init}$ ▷ $\boldsymbol{w}$ is the weight vector for the critic
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{init}$ ▷ $\boldsymbol{\theta}$ is the weight vector for the actor
$\boldsymbol{e}_w \leftarrow 0$ ▷ $\boldsymbol{e}_{\boldsymbol{w}}$ is the eligibility trace for the critic
$\boldsymbol{e}_\theta \leftarrow 0$ ▷ $\boldsymbol{e}_\theta$ is the eligibility trace for the actor
for num. of episodes **do**
  obtain initial state S
  while S is not terminal **do**
    select action A for state S
    take action A, observe S' and R
    $v_{current} \leftarrow \hat{V}(S|\boldsymbol{w})$
    $v_{next} \leftarrow 0$
    if S' is not terminal **then**
      $v_{next} \leftarrow \hat{V}(S'|\boldsymbol{w})$
    end if
    $\delta \leftarrow R + \gamma v_{next} - v_{current}$
    $\boldsymbol{e}_w \leftarrow \gamma\lambda\boldsymbol{e}_w + \nabla_{\boldsymbol{w}}\hat{V}(S|\boldsymbol{w})$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha_v\delta\boldsymbol{e}_w$
    $\boldsymbol{e}_\theta \leftarrow \gamma\lambda\boldsymbol{e}_\theta + \nabla_{\boldsymbol{\theta}} log[\pi(A|S,\boldsymbol{\theta})]$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_\pi\delta\boldsymbol{e}_\theta$
    $S \leftarrow S'$
  end while
end for

critic parameters. The conventional actor-critic algorithm with eligibility traces is summarized in Algorithm. 1.

3. FORWARD ACTOR-CRITIC

3.1 λ-return, n-step return and the conventional forward views

The forward views of both actor and critic (eqns. [1, 2]) uses multi-step update targets, like G_t^λ, $G_t^{(n)}$ or G_t, in place of U_t. These update targets are used as estimates of the true value function $v_\pi(S_t)$.

Usually, λ-return is often used in the update rules of both critic (eqn. [1]) and actor (eqn. [2]). This λ-return, $G_t^\lambda(\boldsymbol{w})$, is defined as an infinite, weighted sum over n-step returns as follows:

$$G_t^\lambda(\boldsymbol{w}) = (1-\lambda)\sum_{i=1}^{\infty}\lambda^{i-1}G_t^{(i)}(\boldsymbol{w}_t) \tag{3}$$

where the n-step return $G_t^{(n)}(\boldsymbol{w})$ is defined as:

$$G_t^{(n)}(\boldsymbol{w}) = \sum_{j=1}^{n}\gamma^{j-1}R_{t+j} + \gamma^n\hat{V}(S_{t+n}|\boldsymbol{w})$$

n-step returns are intermediate update targets that are based on an intermediate number of rewards extending into the future, more than one but less than all of them until termination. On the other hand, λ-return is a weighted combination of n-step returns where the weightings are determined by the λ parameter. Subsequently, we can also use the n-step return in place of λ-return.

It is important to point out here that the algorithms based on λ-return cannot learn incrementally from online experiences.

3.2 K-bounded λ - return

The λ-return is defined as an infinite weighted sum which contains information extending multiple time steps into the future. This cannot be accurately and incrementally learned from online experiences without eligibility traces. In order to approximately estimate this λ-return without using eligibility traces, Cichosz (see [3]) had proposed the idea of truncating this infinite sum so that it results in a K-bounded λ-return, which sums over the first K n-step returns:

$$G_t^{\lambda|K} = (1-\lambda)\sum_{i=1}^{K-1}\lambda^{i-1}G_t^{(i)}(\boldsymbol{w}_{t+i-1}) + \lambda^{K-1}G_t^{(K)}(\boldsymbol{w}_{t+K-1})$$

The time indices of $\boldsymbol{w}$ are different from that of the conventional λ-return defined in eqn. [3]. This allows in incrementally computing $G_t^{\lambda|K}$. Cichosz assumed that the value of K will be arbitrarily chosen. Subsequently, van Seijen introduced a heuristic for selecting this value of K. This heuristic was designed using γ and λ such that there was minimal error in approximating the actual λ-return with the truncated return. This heuristic is defined as:

$$K = \frac{ceil(log(0.01))}{log(\gamma\lambda)} \tag{4}$$

The K-bounded λ-return can be recursively and incrementally computed from an incoming stream of online experiences. The recursive equations along with their derivations are comprehensively shown in [13]. The final recursive equations that are necessary for computing this truncated return are defined as follows:

$$G_t^{\lambda|K} = G_t^{\lambda|K-1} + (\gamma\lambda)^{K-1}\delta'_{t+K-1} \tag{5}$$

$$G_t^{\lambda|K} = R_{t+1} + \gamma(1-\lambda)\hat{V}_{t+1} + \gamma\lambda G_{t+1}^{\lambda|K-1} \tag{6}$$

$$G_{t+1}^{\lambda|K-1} = \frac{(G_t^{\lambda|K} - \rho_t)}{\gamma\lambda} \tag{7}$$

where δ'_i and ρ_t are defined as:

$$\delta'_i = R_{i+1} + \gamma\hat{V}(S_{i+1}|\boldsymbol{w}_i) - \hat{V}(S_i|\boldsymbol{w}_{i-1}) \tag{8}$$

$$\rho_t = R_{t+1} + \gamma(1-\lambda)\hat{V}(S_{t+1}) \tag{9}$$

These equations are the necessary recursive update equations that allow in efficient implementation of forward view methods which are based on the K-bounded λ-return.

Using this return will result in an algorithm that waits for the first K time steps before making the first learning update. Specifically for this reason, these forward view based algorithms are called *delayed* learning algorithms. Additionally, this delay is determined through a heuristic based on the values of γ and λ such that there is minimal approximation error between the actual and the K-bounded λ-return.

3.3 n-step return

In order to untangle the effects of these learning delays in control tasks, it is necessary to analyze these forward view methods with the simplest return available, which is the n-step return $G_t^{(n)}$. For the K-bounded λ-return, the delay depends on the value of λ and γ. But λ also controls the amount of bootstrapping and the weighted averaging between multiple n-step returns making it difficult to clearly identify the factor responsible for the agent's performance. Therefore, we argue that it is reasonable to analyze these

truncated return based forward view methods with their n-step counterparts.

From the recursive update eqns. [5, 6, 7], the final recursive equations for computing n-step returns can be derived to be the following:

$$G_t^{(n)} = G_t^{(n-1)} + (\gamma)^{n-1}\delta'_{t+n-1} \tag{10}$$

$$G_t^{(n)} = R_{t+1} + \gamma G_{t+1}^{(n-1)} \tag{11}$$

$$G_{t+1}^{(n-1)} = \frac{(G_t^{(n)} - \rho_t)}{\gamma} \tag{12}$$

where δ'_i and ρ_t is defined as:

$$\delta'_i = R_{i+1} + \gamma\hat{V}(S_{i+1}|\boldsymbol{w}_i) - \hat{V}(S_i|\boldsymbol{w}_{i-1}) \tag{13}$$

$$\rho_t = R_{t+1} \tag{14}$$

3.4 The Algorithm

For the K-bounded λ-return, the actor and critic update equations can be rewritten as follows:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha_v\big[G_t^{\lambda|K} - \hat{V}(S_t|\boldsymbol{w}_t)\big]\nabla_{\boldsymbol{w}}\hat{V}(S_t|\boldsymbol{w}_t) \tag{15}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_\pi\big[G_t^{\lambda|K} - \hat{V}(S_t|\boldsymbol{w}_t)\big]\frac{\nabla_{\boldsymbol{\theta}}\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)} \tag{16}$$

Similarly, for the n-step return, the forward view update equations can be rewritten as:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha_v\big[G_t^{(n)} - \hat{V}(S_t|\boldsymbol{w}_t)\big]\nabla_{\boldsymbol{w}}\hat{V}(S_t|\boldsymbol{w}_t) \tag{17}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_\pi\big[G_t^{(n)} - \hat{V}(S_t|\boldsymbol{w}_t)\big]\frac{\nabla_{\boldsymbol{\theta}}\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)} \tag{18}$$

where $\boldsymbol{w} \in \mathbb{R}^n$ is the parameter vector for the critic and $\boldsymbol{\theta} \in \mathbb{R}^m$ is the parameter vector for the actor.

Algorithm. 2 summarizes the forward actor-critic method. This can handle both kind of returns that were defined previously. Importantly, when using K-bounded λ-return, the learning algorithm waits for the first K time steps before making its first learning update where K is computed based on the above described heuristic eqn. [4]. Similarly, when using n-step return, the algorithm waits for the first n time steps before making its first update. After this initial delay in learning, the algorithm continues to learn incrementally from online experiences without any additional delay.

Initially, when the time-step $t = 1$, $G_0^{\lambda|1}$ is computed from the following equation (reduced from δ' since $v_{current}$ is initialized as 0):

$$\begin{aligned} G_t^{\lambda|1} &= G_t^{(1)}(\boldsymbol{w}_t) \\ &= R_{t+1} + \gamma\hat{V}(S_{t+1}|\boldsymbol{w}_t) \end{aligned}$$

For subsequent time steps $2 \leq t \leq k-1$, $G_0^{\lambda|t}$ is computed using eqn. [5]. At time step $t = k$, the following occurs: $G_0^{\lambda|K}$ is computed from eqn. [5]; the critic and actor representations are updated using the state-action pairs (S_0, A_0) based on the eqns. [15]; $G_1^{\lambda|K-1}$ is computed from eqn. [6]. These three updates are repeated for subsequent time steps until the end of an episode. After the episode has terminated, the remaining state-action pairs are updated sequentially. As δ'_t is 0 for $t > T$, eqn. [5] need not be used further.

From an algorithmic perspective, the above update equations would essentially form the forward actor-critic algorithm. However, because of small rounding errors, repeatedly computing $G_{t+1}^{\lambda|K}$ from $G_t^{\lambda|K}$ based on eqns. [7 and

Algorithm 2 Forward Actor-Critic

INPUT: $\alpha_v, \alpha_\pi, \lambda, \gamma, K, n, \boldsymbol{w}_{init}, \boldsymbol{\theta}_{init}$
$\boldsymbol{w} \leftarrow \boldsymbol{w}_{init}$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{init}$
$c_{final} \leftarrow (\gamma\lambda)^{K-1}$ (or) $c_{final} \leftarrow (\gamma)^{n-1}$
for num. of episodes **do**
  $Q \leftarrow \emptyset$ ▷ Q is a Queue of length K
  $U_{sync} \leftarrow 0$
  $i \leftarrow 0$
  $c \leftarrow 1$
  $v_{current} \leftarrow 0$
  $is_ready \leftarrow$ FALSE
  obtain initial state S
  while S is not terminal **do**
    select action A for state S
    take action A, observe S' and R
    $v_{next} \leftarrow 0$
    if S' is not terminal **then**
      $v_{next} \leftarrow \hat{V}(S'|\boldsymbol{w})$
    end if
    $\rho \leftarrow$ Eqn. [9] (or) Eqn. [14]
    push the tuple (S, A, ρ) into Q
    $\delta' \leftarrow R + \gamma v_{next} - v_{current}$
    $v_{current} \leftarrow v_{next}$
    if $i = K - 1$ **then**
      $U \leftarrow U_{sync}$
      $U_{sync} \leftarrow v_{current}$
      $i \leftarrow 0$
      $c \leftarrow 1$
      $is_ready \leftarrow$ TRUE
    else
      $U_{sync} \leftarrow U_{sync} + c\delta'$
      $i \leftarrow i + 1$
      $c \leftarrow \gamma\lambda c$ (or) $c \leftarrow \gamma c$
    end if
    if $is_ready =$ TRUE **then**
      $U \leftarrow U + c_{final}\delta'$
      pop (S_q, A_q, ρ_q) from Q
      update $\boldsymbol{w}$ using S_q and U
      update $\boldsymbol{\theta}$ using (S_q, A_q) and U
      $U \leftarrow$ Eqn. [7] (or) Eqn. [12]
    end if
    $S \leftarrow S'$
    if is_ready = FALSE **then**
      $U \leftarrow U_{sync}$
    end if
    while Q is not empty **do**
      pop (S_q, A_q, ρ_q) from Q
      update $\boldsymbol{w}$ using S_q and U
      update $\boldsymbol{\theta}$ using (S_q, A_q) and U
      $U \leftarrow$ Eqn. [7] (or) Eqn. [12]
    end while
  end while
end for

5] causes errors in update targets. To avoid these practical issues, forward actor-critic method computes $G_t^{\lambda|K}$ from scratch every K time-steps. This is achieved by resetting U to U_{sync}. These observations were inaccordance with the observations made by [13].

4. RESULTS

For all our experiments, we used a single hidden layer neural network for representing the critic and actor modules separately. The critic and actor networks each consisted of 50 hidden units with *tanh* activation function. As the critic network estimated the value for a given state, it had only one output unit. Additionally, the actor network estimated the probabilities for selecting an action given a state representation. So, the actor network had output nodes equal to the number of actions available in a given domain. A *softmax* activation function was used on these output units.

The different values for α_v and α_π are { 0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1}. For λ, the different values considered were: {0.0, 0.2, 0.4, 0.6, 0.8, 0.9, 0.95} and for n, wherever applicable, the values were: { 1, 2, 4, 8, 16, 32, 64, 128, 256, 512 }. Moreoever, all the plots in this paper were generated by averaging the results obtained from 100 independent runs and standard errors are reported on the plots, wherever relevant.

4.1 Experiment 1: Mountain car domain

Some of the experimental results presented here (Fig. 1 (a, b, c, d, e, f) and Fig. 2 (a, b, c)) are based on the standard mountain car domain[11]. The objective here is to make an underpowered car climb up a hill.

The Fig. 1 (a, b, c, d, e, f) were obtained from experiments involving the K-bounded λ-return based forward actor-critic method, while the Fig. 2 (a, b, c) were obtained through experiments involving the n-step forward actor-critic method. Both, K-bounded λ-return and n-step based forward actor-critic methods were introduced in this paper.

The state space consists of the position and velocity of this underpowered car. There are three actions available in this domain and the reward is -1 for every time step. This task is undiscounted (i.e., γ is set to 1.0). Each episode of this task was limited to a maximum of 1000 time steps after which the domain was reseted.

K-bounded λ-return forward actor-critic: For the plots in Fig. 1 (a, b, d, e), we selected the α_v values which gave the best performance in terms of the average return accumulated by the learning agent over the first 50 episodes, by both conventional and K-bounded λ-return forward actor-critic methods, on the mountain car domain. This was obtained for two different λ values ($\lambda = 0.2, 0.9$), to show the effect of bootstrapping with respect to the agent's learning performance.

The plot on Fig. 1 (c) was obtained by selecting the best performance value for these two methods, which is in turn obtained by scanning over different α_v and α_π values. Also, the plot on Fig. 1 (f) shows the learning curves for these methods obtained with the best $\lambda, \alpha_v, \alpha_\pi$ settings.

Specifically, the forward actor-critic method using this λ-return simply performs much better than the conventional method, for these $\lambda = 0.2, 0.9$ settings. Additionally, from the Fig. 1 (c), the forward actor-critic method has the overall best performance in terms of the accumulated return over the first 50 episodes, from the parameters optimized. From the Fig. 1 (f), it can also be seen that the learning process of the conventional actor-critic tends to deteriorate over time. One possible reason for this can be inferred to be due to the eligibility traces with nonlinear function approximators.

n-step forward actor-critic: The experiments for this were based on the n-step return forward actor-critic method. Similarly, for the plots in Fig. 2 (a, b, c), we selected the α_v that gave the best performances for many n settings. For the plot in Fig. 3 (b), the values of α_v and α_π values are scanned over to obtain the best performance in the mountain car domain.

In this n-step return based forward actor-critic, it is clear that, during control, there needs to be as little delay in learning as possible. For example, in this domain, for values of $n = 1, 2, 4$ seem to give much better performance compared to the other settings. As this value of n determines the delay that exists for the first learning update to take place, the plots suggests something that is very much interesting — they suggest that the learning update needs to happen as soon as possible, irrespective of its accuracy or its variance.

4.2 Experiment 2: Pole-balancing domain

We also performed experiments on the standard pole-balancing task[1] and their results are presented in Fig. 1(g, h, i, j, k, l) and in Fig. 2 (d, e, f). The objective here is to balance a pole on a cart for as many time steps as possible.

The plots in Fig. 1(g, h, i, j, k, l) were based on the K-bounded λ-return based forward actor-critic method whereas the plots in Fig. 2 (a, b, c) were based on the n-step forward actor-critic method, both of which were introduced in this paper.

The state space for this task consists of position and linear velocity of the cart and the position and angular velocity of the pole. There are two actions available in this domain. The reward is +1 for every time step until termination. At termination, the reward is -1 and the state space is reseted. This task is undiscounted (i.e., γ is set to 1.0). Each episode of this task was limited to a maximum of 1000 time steps after which the domain was reseted.

K-bounded λ-return forward actor-critic: For the plots in Fig. 1 (g, h, j, k), we selected the α_v values which gave the best performance, for both conventional and K-bounded λ-return forward actor-critic methods, on the pole-balancing domain. The performance in these experiments, were measured on the average number of steps achieved by the learning agent over the first 1000 episodes. This was again repeated for two different λ values, to show that role of bootstrapping in affecting the performance for these two methods. Specifically, the forward actor-critic method using this λ-return simply performs much better than the conventional method, for these λ settings.

The plot on Fig. 1 (i) was obtained by selecting the best performance value for these two methods, which is in turn obtained by scanning over different α_v and α_π values. Again, the plot on Fig. 1 (l) shows the learning curves for these methods obtained with the best $\lambda, \alpha_v, \alpha_\pi$ settings. From this figure, it can be seen that the performance of the conventional actor-critic tends to deteriorate over time, as does in the mountain car task.

n-step forward actor-critic: Similarly, for the plots in Fig. 2 (a, b, c), we selected the α_v that gave the best performances for many n settings. The plot in Fig. 3 (b), the values of α_v and α_π values are scanned over to obtain the

best performance in the mountain car domain. In this n-step return based forward actor-critic, it is clear that, during control, there needs to be as little delay as possible for learning. For example, in this domain, for values of $n = 1, 2, 4$ seem to give much better performance compared to the other settings. As this value of n determines the delay that exists for the first learning update to take, the plots suggests something that is very much interesting — to make the learning update as soon as possible, irrespective of its accuracy. However, for the pole-balancing domain, it seems that some small delay actually improves the performance. For the $n = 8, 16$, the performance of the forward actor-critic is near to the best performance obtained by this method.

5. DISCUSSIONS

Typically, multi-step methods in reinforcement learning involve the use of eligibility traces. These work well for linear function approximators. However, in order to scale reinforcement learning methods for real-world control tasks, it is necessary to use non-linear function approximators (e.g.: neural networks) as these can express complicated real-valued functions. In such interesting scenarios, eligibility traces tend to deviate from their corresponding forward view methods. Specifically, using eligibility traces along with non-linear function approximators will not be equivalent to its forward view thereby leading to divergence issues.

Recently, this divergence issue has been comprehensively studied in TD methods (see [13]). However, even before this was studied, there was a general reluctance towards using eligibility traces (see [7, 6, 4, 15, 5]) , which could have been to avoid this specific issue. These methods specifically use the complete return G_t for learning. Moreover, as these methods use batch learning schemes, they lose the advantage of learning *incrementally* and often this leads to slower learning.

Learning incrementally is also the key focus of reinforcement learning and the methods introduced in this paper are promising, because they can learn incrementally from ongoing experiences *and* achieves this *without* eligibility traces. Also, this produces a new class of *forward* methods with better learning abilities compared to conventional methods, specifically for the case of nonlinear function approximation.

In this paper, we introduced two novel policy gradient algorithm which incrementally learning from complex multi-step update targets, specifically using the λ-return and n-step return. The strength of these approaches is that, it uses complex returns for learning *without eligibility traces* and achieves this with incremental computation. We evaluate K-bounded λ-return and the n-step based forward actor-critic methods on mountain car and pole-balancing domains using non-linear function approximation. Actor-Critic methods have the inherent ability to represent stochastic action-selection policies. Also, they use separate memory structures for representing the state-value function and action-selection policy separately. Decoupling the policy from the value function gives the freedom in selecting different learning algorithms for the actor and critic. Moreover, actor-critic methods have shown tremendous promise in many real-world control tasks as well[5, 15].

Additionally, the experiments conducted in this paper suggests that it is important to bootstrap, even while learning to control. Earlier, it was believed that bootstrapping is necessary only for improving the predictions[11]. From our forward actor-critic methods, it can be established that bootstrapping is necessary to improve performance in control tasks also. Subsequently, our results show that, to achieve the best level of performance in control tasks, it is important to perform learning updates as quick as possible. In other words, the agent learns to achieve much higher performance when it makes incremental but noisy updates.

From our experiments, we also conclude that multi-step methods can ideally improve learning performance, even in non-linear function approximations. Recently, Mnih et al.[8] showed that actor-critic methods achieve far better performance on Atari games compared with deep Q-network. As a future work, it would be interesting to compare their actor-critic method with forward actor-critic.

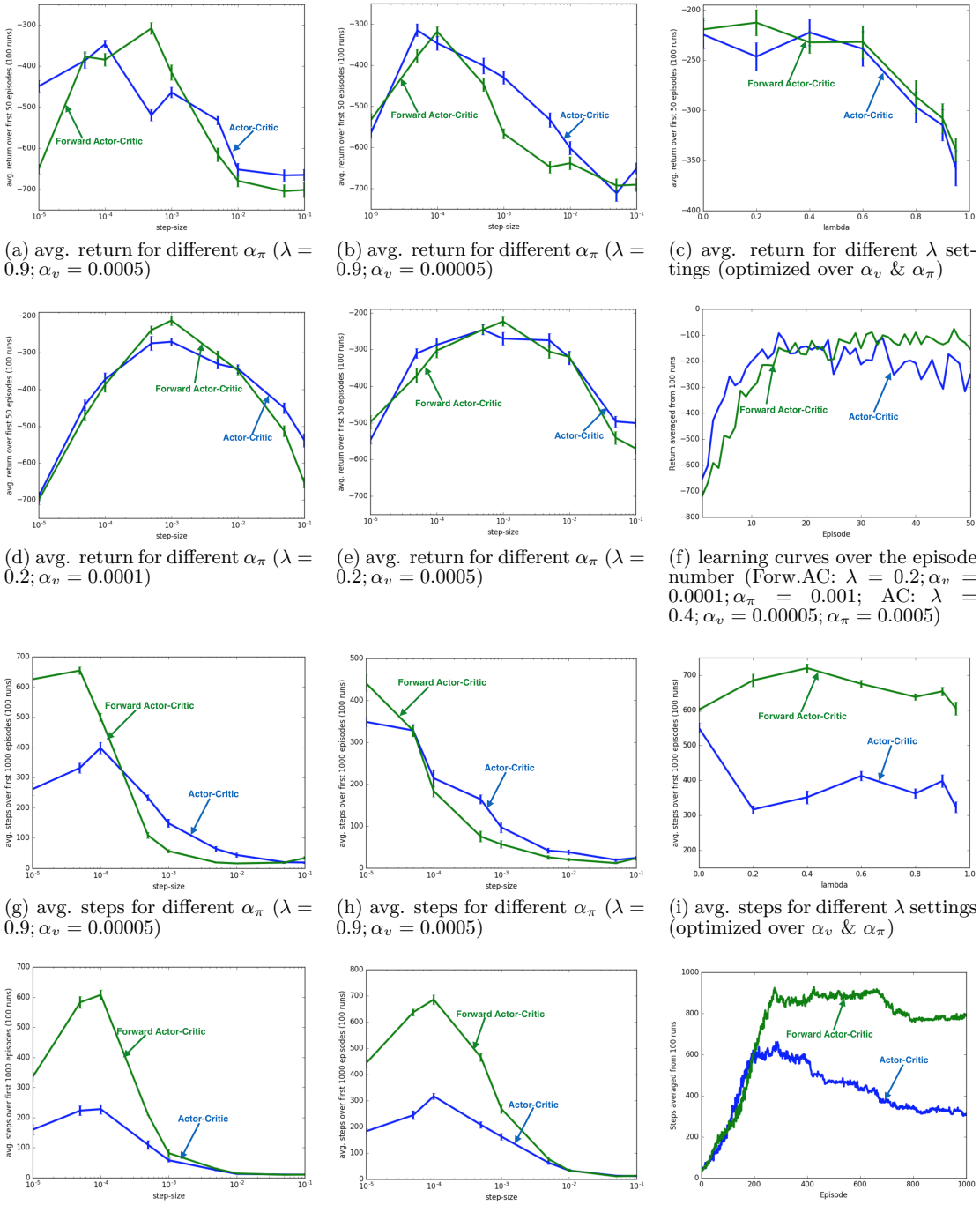
Further, there have been several recent successes with actor-critic methods but most of them either learned using the full return which is obtained after an episode terminates, after which the learning updates are made. Our method can definitely make significant contributions on those domains and those would be our future work, in order to show the complete potential for forward methods.

6. CONCLUSIONS

We introduced and empirically investigated two novel forward actor-critic methods, which as a result has produced a new class of multi-step learning algorithms without eligibility traces. This forward actor-critic algorithm produced substantial and dramatic performance improvements in the pole-balancing domains when compared against the conventional actor-critic method. Specifically, forward actor-critic learns to balance the pole for twice the length of an episode completed by the actor-critic method. In the mountain car domain, this forward actor-critic method produced small performance improvements which were not significant. Additionally, from the forward actor-critic methods, it can be inferred that bootstrapping is necessary to improve control performance which had not been studied until now. Moreover, we also showed that learning delays need to be as minimal as possible in order to achieve a higher level of performance, especially in control tasks. These forward actor-critic methods can learn incrementally from online experiences and with non-linear function approximators. As a result, they can produce immediate impacts in many real-world control domains.

Acknowledgments

The authors thank Kenny Young, Pooria Joulani and Gautham Vasan for insights and discussions contributing to the results presented in this paper. We also gratefully acknowledge funding from Alberta Innovates — Technology Futures and from the Natural Sciences and Engineering Research Council of Canada.



(a) avg. return for different α_π ($\lambda = 0.9; \alpha_v = 0.0005$)

(b) avg. return for different α_π ($\lambda = 0.9; \alpha_v = 0.00005$)

(c) avg. return for different λ settings (optimized over α_v & α_π)

(d) avg. return for different α_π ($\lambda = 0.2; \alpha_v = 0.0001$)

(e) avg. return for different α_π ($\lambda = 0.2; \alpha_v = 0.0005$)

(f) learning curves over the episode number (Forw.AC: $\lambda = 0.2; \alpha_v = 0.0001; \alpha_\pi = 0.001$; AC: $\lambda = 0.4; \alpha_v = 0.00005; \alpha_\pi = 0.0005$)

(g) avg. steps for different α_π ($\lambda = 0.9; \alpha_v = 0.00005$)

(h) avg. steps for different α_π ($\lambda = 0.9; \alpha_v = 0.0005$)

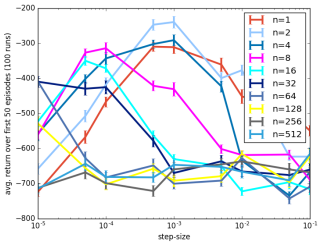
(i) avg. steps for different λ settings (optimized over α_v & α_π)

(j) avg. steps for different α_π ($\lambda = 0.2; \alpha_v = 0.00005$)

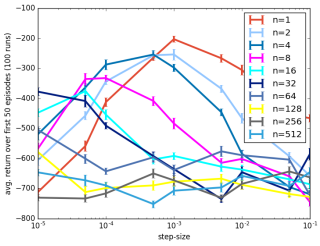
(k) avg. steps for different α_π ($\lambda = 0.2; \alpha_v = 0.0001$)

(l) learning curves over the episode number (Forw. AC: $\lambda = 0.4; \alpha_v = 0.0001; \alpha_\pi = 0.0001$; AC: $\lambda = 0.0; \alpha_v = 0.00005; \alpha_\pi = 0.0001$)

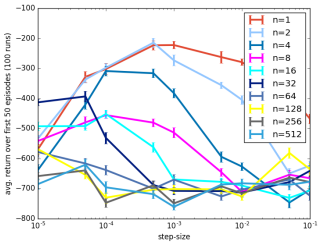
Figure 1: Forward actor-critic and actor-critic - comparison plots: (a, b, c, d, e, f) are obtained from the mountain car domain whereas (g, h, i, j, k, l) are from the pole-balancing domain. The forward actor-critic method shown here is based on the K-bounded λ-return in which the learning delay depended indirectly on the γ and λ values. This is compared against a conventional actor-critic method with eligibility traces. Both these methods use similar neural networks. Additionally, from the plots (c) and (i), it can be inferred that higher values of λ do improve the performance in these control tasks, which implies that some level of bootstrapping is required for good control performance. Also, from the plots (f) and (l), it can be observed that the actor-critic method begins to deteriorate as time progresses within a task.



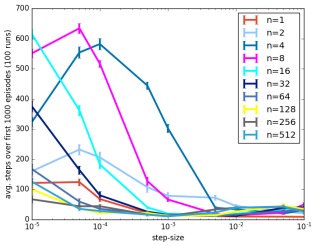
(a) avg. return for different α_π ($\alpha_v = 0.00005$)



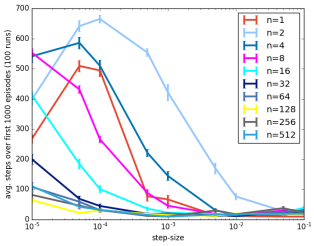
(b) avg. return for different α_π ($\alpha_v = 0.0001$)



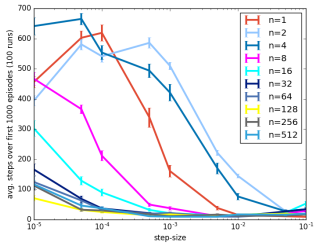
(c) avg. return for different α_π ($\alpha_v = 0.0005$)



(d) avg. steps for different α_π ($\alpha_v = 0.00001$)

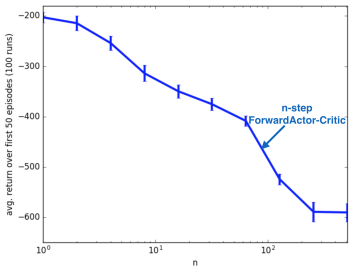


(e) avg. steps for different α_π ($\alpha_v = 0.00005$)

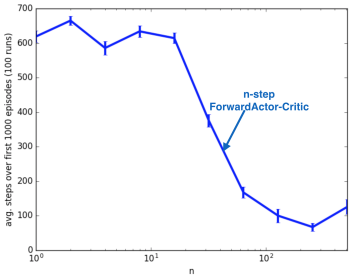


(f) avg. steps for different α_π ($\alpha_v = 0.0001$)

Figure 2: n-step forward actor-critic - step-size plots: (a, b, c) are from the mountain car domain and (d, e, f) are from the pole-balancing domain. From these step-size plots, it can be observed that the forward actor-critic method is relatively robust to a wide range of step-sizes as long as n takes a smaller value. Additionally, from (a, b, c), the parameters $n = 1, 2$ seem to provide a significant performance improvement in the mountain car domain. Similarly, from (d, e, f), the same inference can also be made from the pole-balancing domain, i.e., the delay in performing learning updates needs to be minimum. Overall, from all these plots, it can be inferred that for optimal control performance, the learning updates need to be performed as quickly as possible and such parameter settings are relatively robust to their step-sizes. However, small values of n also implies that these updates might be sometimes unreliable or biased.



(a) mountain car domain: avg. return for different n settings (optimized over α_v & α_π)



(b) pole-balancing domain: avg. steps for different n settings (optimized over α_v & α_π)

Figure 3: n-step forward actor-critic - performance plots: the final performances for both mountain car and pole-balancing domains are plotted for multiple settings of n after scanning over the different values for α_v and α_π. The parameter n controls the delay in making a learning update to the actor-critic method. For the mountain car task, it is clear that the value of n needs to be small, probably $n = 1, 2$. In the case of pole-balancing, the performance does not begin to deteriorate until n reaches 32. This implies that the n-step forward actor-critic method is relatively robust, atleast in this domain, in terms of its learning delays.

REFERENCES

[1] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

[2] D. P. Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.

[3] P. Cichosz. Truncating temporal differences: On the efficient implementation of td () for reinforcement learning. *Journal of Artificial Intelligence Research*, 2:287–318, 1995.

[4] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.

[5] C. Finn, S. Levine, and P. Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48, 2016.

[6] G. Lever. Deterministic policy gradient algorithms. 2014.

[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[8] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016.

[9] R. S. Sutton. Temporal credit assignment in reinforcement learning. 1984.

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

[11] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, in prep.

[12] C. Szepesvári. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103, 2010.

[13] H. van Seijen. Effective multi-step temporal-difference learning for non-linear function approximation. *arXiv preprint arXiv:1608.05151*, 2016.

[14] C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

[15] M. Zhang, Z. McCarthy, C. Finn, S. Levine, and P. Abbeel. Learning deep neural network policies with continuous memory states. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 520–527. IEEE, 2016.