Confidence-Based Aggregation of Multi-Step Returns for Reinforcement Learning

Girish Raguvir Jeyakumar[1], Balaraman Ravindran[1,2]
[1]Department of Computer Science and Engineering, [2]Robert Bosch Centre for Data Science and AI (RBCDSAI)
Indian Institute of Technology Madras, Chennai, India
girishraguvir@gmail.com, ravi@cse.iitm.ac.in

Abstract

Reinforcement Learning (RL) enables modelling of complex behavioral patterns for sequential decision making tasks with well-defined goals. Many successful RL algorithms rely on Temporal Difference (TD) Learning and consequently, accurate estimation of the value function becomes important. In this respect, λ-returns (LR) have proven effective in the past as they enable aggregation over multiple multi-step returns and help in faster propagation of delayed rewards. Though the exponentially decaying weighing scheme of λ-returns does garner validity from such theoretical and empirical results in the literature, it's important to observe that it is a static weighing scheme with limited flexibility. We propose a new paradigm of dynamically weighted returns called *Confidence-based Returns* wherein the multiple many-step look-aheads are explicitly valued based on the *confidence* in the estimates. We propose a simple and efficient way to model confidence and use it to derive Confidence-based Returns. We incorporate Confidence-based Returns (CR) into the Asynchronous Advantage Actor Critic (A3C) algorithm to obtain a new variant of A3C called CRA3C and demonstrate the efficacy of CRA3C by showcasing results on multiple high-dimensional visual input tasks in the Atari 2600 domain.

Introduction

Reinforcement Learning (RL) (Sutton and Barto 1998) has recently seen immense progress with tremendous success in many domains which had previously seemed insurmountable (Mnih et al. 2015; Mnih et al. 2016; Silver et al. 2017b; Todorov, Erez, and Tassa 2012). RL enables modelling of complex behavioral patterns for sequential decision making tasks with well-defined goals where no explicit guidance, as in supervised learning, is feasible. In RL, tasks are modelled as Markov Decision Processes (MDPs) (Puterman 2014).

Traditional RL algorithms typically solve the MDP by maintaining tabular estimates for the value function of each state and updating it using the Bellman Optimality Equation (Puterman 2014). However, as is the case with many games, when one moves to problems with exponentially large or continuous state spaces, these traditional tabular methods are rendered moot. Complex computer games like Atari, Doom, Dota 2 etc. have high-dimensional visual inputs, posing a very difficult challenge. Such limitations faced by traditional RL methods along with tremendous progress in representation learning using deep neural networks (Bengio and others 2009; LeCun, Bengio, and Hinton 2015) lead to the advent of Deep Reinforcement Learning (DRL). Deep Neural Networks provided a way to learn abstract representations in an end-to-end manner, directly from data. They enabled RL algorithms to achieve generalization over large state spaces. This motivated a lot of work in recent years and the rapid utilization of parallel advances in Deep Neural Networks (Krizhevsky, Sutskever, and Hinton 2012; Hochreiter and Schmidhuber 1997) in RL has led to vast advancements in multiple complex domains - Atari 2600 (Bellemare et al. 2013; Mnih et al. 2015; Mnih et al. 2016; Jaderberg et al. 2017), Chess (Silver et al. 2017a), Go (Silver et al. 2017b) etc.

Many of these successful RL algorithms utilize Temporal Difference (TD) Learning (Sutton 1988). In TD learning, we use n-step returns wherein we bootstrap from the value function estimate of the n^{th} future state to obtain an estimate for the current state. Consequently, accurate estimation of the value function becomes important. Precise estimates of value functions would not only lead to better policy estimates but would also result in faster learning. In this respect, λ-returns (LR) (Sutton and Barto 1998) have been observed to be effective as they enable aggregation over multiple multi-step returns and help in faster propagation of delayed rewards.

However, while the weights assigned by λ-returns do garner some validity from the bias-variance trade-off associated with the choice of shorter versus longer returns in-addition to positive empirical results in literature, it is a static weighing scheme with limited flexibility and little sophistication. We propose the use of a more dynamic and explicit choice of weights based on *confidence* in the bootstrapped estimates. Though vaguely similar notions have been considered previously (White and White 2016; Thomas et al. 2015), the body of work is largely limited and the results shown are minimal. One reason behind dearth of such works is the difficulty in quantifying and optimizing such a metric. Recently, to address this in the DRL setting, the concept of *Autodidactic Returns* (AR) (Sharma et al. 2017) was proposed wherein the agent is also equipped to learn the weights it wants to assign to each of the $n-$step returns. In a specific derivative called *Confidence-based Autodidactic*

Returns (CAR), they discuss a theoretically weak confidence-based interpretation using limited post-training analyses but they provide no explicit guarantee of it's generalization. The agent isn't explicitly constrained to learn confidences and their formulations are largely restricted to the DRL setting. Their results on the Atari 2600 domain are only on par with baseline λ-returns - a model which is much more simple, efficient and easier to implement. The idea is well-justified, but we believe that the problem is due to an inherent difficulty in the agent learning certainty/confidence on it's own and we want to address it.

We propose a more explicit and widely applicable paradigm of weighted returns called *Confidence-based Returns* (CR). In CR, the multiple many-step look-aheads are valued based on the maintained confidence in their estimates. To model this, we formulate a simple and efficient way to estimate confidence and then use it derive CR. We believe that CR's explicit importance to "usefulness" of the bootstrapped estimates by specifically modelling confidence differentiates it's from λ-returns, AR and all other similar works. To demonstrate the efficacy of CR, we incorporate CR into the Asynchronous Advantage Actor Critic (A3C) (Mnih et al. 2016) algorithm to obtain a new variant of A3C called CRA3C and showcase results on multiple complex tasks in the Atari 2600 domain.

Background

In this section, we briefly discuss a few RL concepts which are essential to understanding our contribution.

Preliminaries

Any reinforcement learning (RL) problem is typically formulated as a Markov Decision Process (MDP) (Puterman 2014), specified as $\langle \mathcal{S}, \mathcal{A}, r, \mathcal{P}, \gamma \rangle$ where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ is the action space, $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is the reward function, $\mathcal{P} : (\mathcal{S} \times \mathcal{A}) \times \mathcal{S} \mapsto [0, 1]$ is the transition probability distribution and $\gamma \in [0, 1)$ is the discount factor. An RL agent engages with an environment $\mathcal{E}$ over a certain number of discrete time steps to achieve a specific goal guided by rewards. The agent follows a policy $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ which, given the state, is a probability distribution over the set of actions $\mathcal{A}$. At any given time step t, the agent in state $s_t \in \mathcal{S}$, chooses an action $a_t \in \mathcal{A}$ according to $\pi(s_t)$ and moves to state s_{t+1} while receiving a reward r_{t+1} from the environment $\mathcal{E}$. The goal is to identify a policy π that maximizes the expected discounted sum of future rewards. The state-value $V^\pi(s)$ of a state s under a policy π is the expected sum of discounted future rewards obtained by starting at s and following π: $V^\pi(s) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s]$.

Multi-step Returns

Monte Carlo methods estimate the value function of a state based on the entire trajectory ranging from that state until end of the episode. On the other hand, one-step TD methods utilize just the immediate reward and then bootstrap from the next state's value function to substitute for future rewards. These form two ends of a spectrum of methods that can be used for estimating the value function and the concept of multi-step returns bridges the gap between them. Multi-step returns compute an estimate using an intermediate set of rewards (typically more than 1 and less than the entire episode) from the trajectory and then bootstrapping for the rest. Specifically, a n-step return uses the first n rewards and bootstraps for the rest using the value-function estimate of $(n+1)^{th}$ state. If T is the last time step of the episode, the n-step return estimate $G_t^{(n)}$, where $1 \leq n \leq (T - t)$, for state s_t is given by

$$G_t^{(n)} = \sum_{i=1}^{n} \gamma^{i-1} r_{t+i} + \gamma^n V(s_{t+n}) \tag{1}$$

$G_t^{(n)}$ can then be used for iterative learning of the value function by using the TD(0) update equation

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t^{(n)} - V(s_t))$$

where $G_t^{(n)}$ serves as the TD-target and α is learning rate.

Weighted Multi-step Returns

Weighted multi-step returns are linear combinations of multiple n-step returns with varying n's. Considering the bias-variance trade-off associated with the choice of n, when one considers multi-step returns, a logical solution is to utilize a weighted average of n-step return estimates instead. Such an estimate can be used as a TD-target in any TD learning method as long as the sum of the weights is 1 (Sutton and Barto 1998).

Given n-step returns $G_t^{(1)}, G_t^{(2)}, \cdots, G_t^{(h)}$ for $n \in \{1, 2, \cdots, h\}$ and associated weights $w^{(1)}, w^{(2)}, \cdots, w^{(h)}$ such that $\sum_{i=1}^{h} w^{(i)} = 1$, we can define a TD-target

$$G_t^w = \sum_{n=1}^{h} w^{(n)} G_t^{(n)}$$

λ-Returns

λ-returns arises from the forward view understanding of eligibility traces. For every state, we look ahead in time to future rewards and value function of states and then decide how best to combine them. Specifically, TD(λ) can be considered as an instance of weighted multi-step returns with the weight $w^{(n)}$ for the n-step backup being proportional to λ^{n-1}, with the normalization term being $(1 - \lambda)$ (for ensuring that weights sum upto 1). The resulting combination can serve as a TD-target and is given by

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

As one can see, the weights decay exponentially with each step. Once a terminal state is reached at T, all further n-step returns become G^t (Monte Carlo estimate). So we can rewrite G_t^λ as

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{(n)} + \lambda^{T-t-1} G_t$$

We can also define a truncated form of λ-returns for shorter time horizons of say h steps.

$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{h-1}\lambda^{n-1}G_t^{(n)} + \lambda^{h-1}G_t^{(h)}$$

λ-returns have been in the RL literature for a long time (Peng and Williams 1996; Sutton and Barto 1998; Seijen and Sutton 2014) and they have recently been used in the DRL setting as well (Schulman et al. 2015; Gruslys et al. 2017; Sharma et al. 2017).

Confidence-based Returns

In this section, we provide a complete description of our proposed model of Confidence-based Returns (CR).

Definition

A $n-$step return estimate for $V(s_t)$ is obtained by bootstrapping from $V(s_{t+n})$. But the key aspect to note is that, $V(s_{t+n})$ is in itself an estimate. With this in mind, when one considers a weighted multi-step return derived from bootstrapped estimates $G_t^{(1)}, G_t^{(2)}, \cdots, G_t^{(h)}$, a natural way to designate weights would be based on the *confidence* we have on the corresponding estimates (Sharma et al. 2017).

We choose to weigh $G_t^{(n)}$ based on how *confident* we are about our estimate for $V(s_{t+n})$. Let's denote the confidence on $V(s_{t+n})$ as $c(s_{t+n})$. Given the confidences $c(s_{t+1}), c(s_{t+2}), \cdots, c(s_{t+h})$ we define the weights $w^{(1)}, w^{(2)}, \cdots, w^{(h)}$ by taking a softmax (to ensure $\sum_{i=1}^{h} w^{(i)} = 1$)

$$w^{(i)} = \frac{e^{c(s_{t+i})}}{\sum_{n=1}^{n=h} e^{c(s_{t+n})}} \tag{2}$$

We then define the confidence-based TD-target as

$$G_t^{w} = \sum_{i=1}^{h} w^{(i)} G_t^{(i)} \tag{3}$$

Modelling Confidence

"Confidence" or "certainty" is typically quantified by the width of the confidence interval associated with an estimate. Using "uncertainty" to motivate exploration has long been studied in RL. Most algorithms are based on the OFU heuristic - "optimism in the face of uncertainty" (Strehl, Li, and Littman 2009) and use state (or state-action) visit counts to model "uncertainty". The agent is motivated to move to less frequently visited regions of the state space by assigning larger reward bonuses to lower counts. They have traditionally been tabular in the sense that they define *exploration bonuses* by maintaining a table of state (or state-action) visit counts.

UCB1 bandit algorithm (Lai and Robbins 1985) is one of the best known algorithms derived from this fundamental idea. UCB1 chooses an action a such that the associated upper confidence bound $\hat{Q}_t(a) + \sqrt{\frac{2\log(t)}{N(a)}}$ is maximum, where $\hat{Q}_t(a)$ is empirical reward mean and $N(a)$ is the visit-count. In the more generalized MDP setting, MBIE-EB (Strehl and Littman 2008) proposes solving the augmented Bellman equation

$$V(x) = \max_{a\in A}[R(x,a) + \gamma E_p[V(x^{'})] + \beta N(x,a)^{\frac{-1}{2}}]$$

with the exploration bonus being proportional to $N(x,a)^{\frac{-1}{2}}$ and $\hat{R}$ and $\hat{P}$ being the empirical reward and transition function respectively. This bonus term is based on the theoretical guarantee that the confidence intervals associated with both the rewards and transition functions shrink as $N(x,a)^{\frac{-1}{2}}$, where $N(x,a)$ is the state-action visit count. This dependence stems from Chernoff bounds and is provably optimal (Kolter and Ng 2009).

When the policy is sufficiently random enough that most actions would be tried at any new state, we can replace that state-action visit count with just the state visit count. Since the uncertainty term is now independent of action taken, we move $N(x)^{\frac{-1}{2}}$ out of the maximization, giving us a simpler augmented Bellman equation.

$$V(x) = \max_{a\in A}[R(x,a) + \gamma E_p[V(x^{'})]] + \beta N(x)^{\frac{-1}{2}} \tag{4}$$

Equation 4 motivates the agent to explore new states by using a bonus term proportional to the "uncertainty in the state" modelled by $N(x)^{\frac{-1}{2}}$. This simplifying reduction has been used extensively and it is found to be experimentally just as sound (Tang et al. 2017; Martin et al. 2017; Bellemare et al. 2016).

We pivot on this specific result to address our problem of modelling confidence in value function estimates. Considering that the value function is a direct function of the state, $N(x)^{\frac{-1}{2}}$ becomes an effective proxy for the uncertainty in the value function as well. Since larger uncertainty would mean lower confidence and vice-versa, we propose the utility of $N(x)^{\frac{1}{2}}$ as a measure of confidence $c(x)$ on an estimate of $V(x)$. Simply put, this would mean value function estimates of states with larger visit-counts are considered to be more "confident" estimates. This choice also derives support from the intuition that more frequently recurring states would enable the network to learn more "confident" estimates for their value functions.

Maintaining State-Visit Counts

For small discrete MDPs, simple tabular based methods which already fit conveniently in the framework of tabular RL algorithms (typically used for solving such MDPs) can be used to maintain visit counts. These methods have been well utilized in literature, especially for addressing the problem of efficient exploration (Strehl and Littman 2008). However, these become ineffective when one moves to high-dimensional state spaces. Considering most recent works in RL have focused on such complex tasks, we direct our focus to such domains as well.

A fundamental problem which arises in high-dimensional continuous state spaces is that most states are never visited during training and consequently the visit-counts remain

close to zero for a majority of the states. To overcome this, we need a mechanism by which visit-counts can be generalized to unseen states. This concept of *generalized state visit-counts* has been studied and people have proposed multiple ways to compute and maintain such counts (Tang et al. 2017; Martin et al. 2017; Bellemare et al. 2016; Ostrovski et al. 2017). We derive from one such work by Tang et al. where they use a simple yet effective generalization of classic tabular methods through hashing.

Hashing for High-Dimensional State Spaces The counts are maintained using hashing . The state-space is discretized by using a hash function ϕ which maps states to integers and a counter $n(.)$ is maintained over these hashed entities. The counter is initially set to zero and for every state s encountered, $n(\phi(s))$ is incremented by one. At any instant, the state-visit count of any state s is obtained by simply accessing $n(\phi(s))$.

The choice of the hash function ϕ is clearly important. ϕ should be such that "similar" states are mapped to the same bucket while "dissimilar" states are mapped to different buckets. With this in mind, a popular and efficient type of locality-sensitive hashing (LSH) called SimHash (Charikar 2002). LSH is widely used for answering nearest neighbour queries in high-dimensional spaces based on different similarity metrics (Andoni and Indyk 2006). SimHash, in specific, utilizes similarity based on angular distance. For given state $s \in \mathcal{S}$, SimHash computes the binary hash-code as

$$\phi(s) = sgn(Ag(s)) \in \{-1, 1\}^k$$

where $A \in \mathbb{R}^{k \times D}$ with i.i.d. samples drawn from $\mathcal{N}(0, 1)$ and $g : S \to \mathbb{R}^D$ is a preprocessing function (can be identity as well). As one can infer, the choice of k determines the length of the hash code and consequently the granularity. Suitable choice of k is necessary for ϕ to appropriately distinguish between states.

Algorithm 1 summarizes how we maintain state-visit counts.

Algorithm 1 Maintaining state visit-counts

Define $g : S \to \mathbb{R}^D$ ▷ state pre-processing function
Initialize $A \in \mathbb{R}^{k \times D}$ with i.i.d. samples from $\mathcal{N}(0, 1)$.
Initialize all values $n(.)$ to 0.

function ADD_COUNT(s)
    $\phi(s) \leftarrow sgn(Ag(s))$
    $n(\phi(s)) \leftarrow n(\phi(s)) + 1$

function GET_COUNT(s)
    $\phi(s) \leftarrow sgn(Ag(s))$
    return $n(\phi(s))$ ▷ $N(x) \leftarrow$ GET_COUNT(x)

Experimental Setup

We showcase the efficacy of our model of Confidence-based Returns (CR) by incorporating it in the A3C algorithm (Mnih et al. 2016). The choice of A3C is due to its immense success in Deep RL, especially in high-dimensional visual input domains. We evaluate CRA3C (Confidence-based Returns + A3C) in it's ability to learn game-play in the Arcade Learning Environment (Bellemare et al. 2013). ALE is a important benchmark in Deep RL with it's wide variety of Atari 2600 video games in high-dimensional state-spaces.

Using Weighted Multi-step Returns in A3C

Considering that the A3C algorithm already uses multi-step returns, the extension to using weighted multi-step returns is trivial. In fact, incorporation of any weighted multi-step return variant is quite simple as they, by definition, already form a valid TD-target.

We simply use the weighted multi-step return estimate, G_t^w, in place of G_t. The actor and critic gradient sample estimates consequently become:

$$\nabla_{\theta_{actor}} \log \pi_{\theta_{actor}}(a_t|s_t)(G_t^w - V(s_t))$$
$$\nabla_{\theta_{critic}}(G_t^w - V_{\theta_{critic}}(s_t))^2$$

Computing the TD-target

For a given state s_t, we compute the TD-target for $V(s_t)$ by aggregating look-aheads upto $h-$step returns. By doing a forward pass through the A3C network for states s_{t+1}, s_{t+2}, ..., s_{t+h}, we obtain value function and policy estimates : $(V(s_{t+1}), \pi(s_{t+1}))$, $(V(s_{t+2}), \pi(s_{t+2}))$, ..., $(V(s_{t+h}), \pi(s_{t+h}))$. We also obtain the rewards r_{t+1}, r_{t+2}, ..., r_{t+h} from the environment by taking actions as dictated by $\pi(s_t)$, $\pi(s_{t+2})$, ..., $\pi(s_{t+h-1})$ respectively. Using the value function estimates and the rewards we calculate n-step returns $G_t^{(1)}, G_t^{(2)}, ..., G_t^{(h)}$ using Equation 1. In parallel, we use the pipeline described in Section to calculate $c(s_{t+1})$, $c(s_{t+2})$, ..., $c(s_{t+h})$. We then use Equations 2 and Equation 3 to compute the TD-target for $V(s_t)$.

Preprocessing function g

We experiment with two non-trivial preprocessing functions: BASS (Bellemare et al. 2013) and Feature-Space Transformation (Martin et al. 2017). The former, though not restricted to, is specifically modelled for tasks in Atari 2600 while the latter is more generic.

BASS When it comes to tasks in the Atari 2600 domain the input to any RL algorithm is typically the game screen itself. Thus, given the image, we divide it into square cells and compute the average pixel intensity of each color channel within a cell. These mean values would now serve as representatives for the corresponding pixels. We then normalize the values to lie in [0,1]. Mathematically,

$$I_{BASS}(i, j, z) = \left\lfloor \frac{1}{255C^2} \sum_{(x,y) \in cell(i,j)} I(x, y, z) \right\rfloor$$

where z is the channel, C is cell size, (x, y, z) is the pixel location in the original image I, (i, j, z) is the the pixel location in the processed image I_{BASS} and (i, j) is the location of the cell. Using this, we move from a $A \times A \times K$ image to a $\lceil \frac{A}{C} \rceil \times \lceil \frac{A}{C} \rceil \times K$ image.

BASS is essentially a preprocessing that provides us with a reduced representation which captures the essence of the game-screen and is invariant to small object motions. Being easy to implement and also light-weight, it enables us to update and retrieve counts faster as we now have much smaller input to hash (a reduction factor of $O(C^2)$).

Feature-Space Transformation This preprocessing can be used with any value-based RL algorithm that makes use of a function approximator. The idea is to use the representation learnt by the function approximator in the feature space instead of the state itself. This preprocessing will make states that "share features" to have similar counts. In addition, since the representation learnt in the feature space is typically of much smaller dimension than the original state representation, we would incur lower computational overhead for maintaining state-visit counts.

It's important to note that this preprocessing ensures that the same lower dimensional feature representation is used to estimate both the value function and the confidence. Considering that we want to essentially use these counts as a proxy for the uncertainty in the value function estimate, the motivation is to treat states with similar features used in estimating value function, to be similar. In-addition, this preprocessing requires no additional computations as we already compute the representation for estimating the value function.

In A3C, the policy and the value function is estimated directly from the final LSTM layer outputs (Mnih et al. 2016). Consequently, for our A3C experiments, the output of this LSTM layer serves as the feature-space representation of the state.

Additional Tricks

We utilize a few additional tricks tailored for our model definition.

Count-based Exploration Exploration is vital for success of any RL algorithm. It's important for the agent to explore the environment and identify opportunities with high rewards and long-term gains. Typically most methods rely of simple rules like $\epsilon-$greedy. But recent works (Tang et al. 2017; Martin et al. 2017; Bellemare et al. 2016) have shown success in using more sophisticated count-based exploration methods. Motivated by such works, we use a count-based exploration bonus for our experiments.

Exploration is motivated by adding a exploration bonus $r^+ : \mathcal{S} \mapsto \mathbb{R}$ to the reward function, given by

$$r^+(s) = \frac{\beta_{exp}}{\sqrt{n(\phi(s))}}$$

where $\beta_{exp} \geq 0$. The agent is trained by using rewards given by $(r + r^+)$ but performance evaluation is done devoid of such bonus rewards.

Count Decay As emphasized earlier the value-function targets for the agent are estimates themselves. Hence there is an inherent problem of non-stationarity - the value function targets keep changing with learning. In such a setting, when the target changes, previously accumulated counts may no longer be relevant. In order to take this into account, we decay the counts over time.

$$n(\phi(s)) = \beta_{count} n(\phi(s)) + 1$$

where $0 < \beta_{count} \leq 1$.

Confidence Decay In our current formulation, the importance of $G_t^{(j)}$ is derived solely from the confidence $c(s_{t+j})$ in the estimate of $V(s_{t+j})$. But as emphasized by λ-returns, it is also important to keep the bias-variance trade-offs in mind. Even though the agent maybe be confident of the value function estimates used by longer returns, the large variance arising from using a longer chain of rewards obtained from a stochastic environment maybe detrimental. So we experiment with more "return-sensitive" measure of confidence which takes this into account.

We decay confidences based on the length of the return. For $k-$step return $G_t^{(k)}$ bootstrapped from $V(s_{t+k})$, we use $c(s_{t+k})$ as

$$c(s_{t+k}) = \beta_{conf}^{k-1} N(s_{t+k})^{\frac{1}{2}}$$

where $0 < \beta_{conf} \leq 1$. This can be viewed as a generalization as $\beta_{conf} = 1$ would give us our original formulation.

Algorithms Overview

Table 1: Overview of algorithms

Pre-processing (g)	β_{conf}	Algorithm
Identity	1	CRA3C-Pixel
Feature-Space Tranform	1	CRA3C-FS
BASS	1	CRA3C-BASS
BASS	0.9	CRA3C-BASS-CD

We experiment with 4 different algorithms as shown in Table 1. All algorithms utilize $\beta_{exp} = 0.02$, $\beta_{count} = 0.99$ and we use two different values of β_{conf}: 0.9 and 1. The hyper-parameters for our algorithms were chosen based on performance on 4 tasks: Asterix, Bowling, Seaquest and Space Invaders. The same set of hyper-parameters were then used for all experiments on rest of the tasks.

Results

We evaluate our algorithms on 10 Atari 2600 games and we compare our results against 3 competitive and relevant benchmarks: A3C, CARA3C (Confidence-based Autodidactic Returns + A3C) and LRA3C (λ-returns + A3C) (Mnih et al. 2016; Sharma et al. 2017).

Each agent was trained for 50 million time steps. All evaluations were done using the best agent obtained after 50 million time-steps of training. All reported values are averages of results corresponding to experiments performed twice with different random seeds. This was done to ensure robustness of our results to random initialization. The scores for A3C, CARA3C and LRA3C, under identical experimental settings of 50 million training steps and two different random seeds, were obtained from Sharma et al..

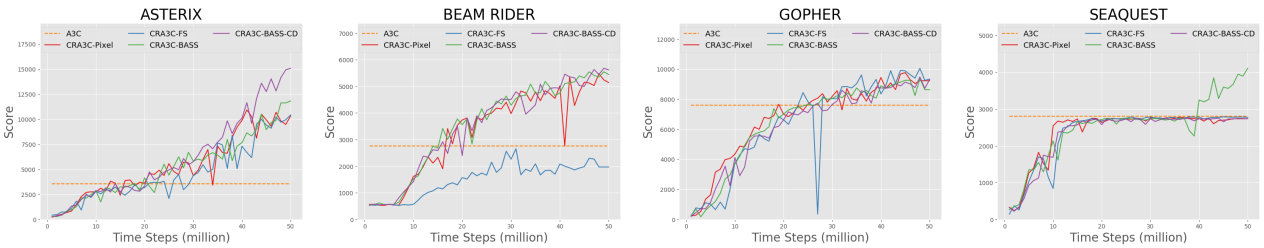

Figure 1: Training curves showing evolution of performance over time.

Gameplay Performance

Table 2: Mean and median scores across 10 Atari games computed as percentages of A3C baseline, along with number of games improved over A3C.

Algorithm	Mean	Median	> A3C
CARA3C	150%	115%	8
LRA3C	212%	119%	7
CRA3C-FS	132%	120%	7
CRA3C-BASS-CD	222%	128%	9
CRA3C-Pixel	230%	138%	8
CRA3C-BASS	**257%**	**155%**	**10**

Figure 1 shows the evolution of performance of our algorithms - CRA3C-Pixel, CRA3C-FS, CRA3C-BASS and CRA3C-BASS-CD - during training for four of our best performing games: Asterix, Beam Rider, Gopher and Seaquest.

Table 2 shows the mean and median scores computed as percentages of A3C. For each algorithm, we also present the number of games (out of 10) in which that algorithm performed better than the A3C baseline.

As one an see, all our algorithms achieve significant improvements over A3C with three of them - CRA3C-Pixel, CRA3C-BASS and CRA3C-BASS-CD - outperforming all 3 benchmarks. The raw scores obtained by our methods are discussed in sections below. From Table 2, it's clear that CRA3C-BASS performs the best, with it achieving over 2.5$\times$ the scores of A3C on average. Importantly, in-addition to performing considerably well, CRA3C-BASS is also consistent. It achieves improvement over A3C on every single task unlike any other.

Table 3 shows the benchmark scores and also the highest score achieved among benchmarks, MB, for each game.

Without Confidence Decay

Table 4 summarizes our results. Without any preprocessing, CRA3C-Pixel already outperforms many of the baselines with it achieving the best score in 5 out of 10 games and it being in top two in 7 out of 10 games. CRA3C-BASS achieves further improvements on top of this, with it being the best in 7 out 10 games and it being in the top two in 9 out 10 games! Keeping in mind that BASS is just a static preprocessing function, designed based on a generic idea of Atari game screens, strong performance of BASS draws

Table 3: Scores attained on Atari 2600 games: Benchmark Scores. MB represents the maximum score among benchmarks.

Game	A3C	CARA3C	LRA3C	MB
Asterix	3548.50	6043.25	8552.25	8552.25
Beam Rider	2764.60	3270.96	2516.79	3270.96
Bowling	54.69	43.17	50.14	54.69
Breakout	409.15	582.37	596.94	596.94
Freeway	20.58	20.63	20.81	20.81
Frostbite	366.50	407.75	450.50	450.50
Gopher	7611.00	8322.10	8766.00	8766.00
Kangaroo	46.00	177.00	445.00	445.00
Seaquest	2798.60	2726.40	2792.00	2798.60
Space Inv.	766.40	1457.05	1125.15	1457.05

focus to how important preprocessing can be. Thus using more task-oriented and adaptive preprocessing can possibly lead to better hash functions and in turn help achieve more improvement.

Based on Table 2, one might feel that performance CRA3C-FS is not on par with the other two but notably, as shown in Table 4, it achieves top scores in Gopher & Freeway and beats A3C in 7 out 10 games. One major reason for it's poor reflection in Table 2 is because of it's stats being severely pulled down by poor performance in Kangaroo. Based on some initial experiments, we believe that the drop in performance in games like Kangaroo and Beam Rider might be due to over-compression of information by feature-space transformation as compared to the other two and consequent prevention of derived the hash function from suitably distinguishing between different states in the game.

With Confidence Decay

To understand the impact of confidence decay, we chose the best performing algorithm from the previous section - CRA3C-BASS - and incorporated confidence-decay (i.e $\beta_{conf} < 1$) to obtain CRA3C-BASS-CD. Table 5 summarizes our results.

As one can see, confidence decay leads to significant improvement in performance, with it boosting the already the best scores of CRA3C-BASS on 6 out 7 games! But oddly, in games in which CRA3C-BASS doesn't perform upto the mark, CRA3C-BASS-CD performs worse. This leads to lower mean improvement across games as seen in Table 2.

Table 4: Scores attained on Atari 2600 games: MB represents the maximum score among benchmarks. Bold highlights scores $>$ MB. Asterisk ($*$) indicates best among proposed methods.

Game	MB	Pixel	BASS	FS
Asterix	8552.25	**10932.00**	**12238.00***	**10423.00**
BeamRider	3270.96	**5474.28**	**5603.63***	2654.42
Bowling	54.69	**84.88***	**54.94**	**72.92**
Breakout	596.94	513.17	526.73	543.82*
Freeway	20.81	**21.04**	**21.02**	**21.19***
Frostbite	450.50	288.30	**1125.95***	325.70
Gopher	8766.00	**9770.00**	**9574.90**	**10053.80***
Kangaroo	445.00	440.00*	435.00	52.00
Seaquest	2798.60	2776.20	**4112.80***	2786.40
Space Inv.	1457.05	1126.95	1250.48*	975.70
$>$ MB	-	5/10	7/10	4/10

Table 5: Scores attained on Atari 2600 games: MB represents the maximum score among benchmarks. Bold highlights scores $>$ MB. Asterisk ($*$) indicates best among proposed methods.

Game	MB	BASS	BASS-CD
Asterix	8552.25	**12238.00**	**15088.50***
Beam Rider	3270.96	**5603.63**	**5761.59***
Bowling	54.69	**54.94**	**65.96***
Breakout	596.94	526.73*	475.40
Freeway	20.81	**21.02**	**21.12***
Frostbite	450.50	**1125.95**	**1330.65***
Gopher	8766.00	**9574.90**	**9742.30***
Kangaroo	445.00	435.00*	245.00
Seaquest	2798.60	**4112.80***	2763.30
Space Inv.	1457.05	1250.48*	974.00
$>$ MB	-	7/10	6/10

For this, we believe a more exhaustive tuning of β_{conf} might alleviate the problem. However, performance of CRA3C-BASS-CD stands to emphasize the importance of modelling confidence and how better models can lead to much better results.

Conclusion and Future Work

We propose a novel paradigm of weighted returns called Confidence-based Returns (CR). In CR, the agent weighs the various bootstrapped n-step returns dynamically based on the confidence it has on the corresponding estimates. We propose a simple and efficient way to model *confidence* in value function estimates and then describe how they can be used to obtain TD-targets using CR. We showcase the efficacy of CR by incorporating it in A3C and presenting state-of-the-art results in Atari 2600 domain. We believe that the dynamicity of CR and it's explicit importance to "usefulness" of the bootstrapped estimates by modelling confidence is key to it's success.

The concept of Confidence-based Returns is quite generic. It's core idea is to essentially use a more sophisticated aggregation of bootstrapped estimates based on the notion of confidence, to come up with better value function targets. We believe that this way of modelling TD-targets is powerful and can lead to very successful learning in a variety of tasks. We have proposed one way of modelling confidence but there could multiple other. Using more sophisticated counting models in the current framework is also be another path for further exploration.

We want to emphasize that our proposed idea can be combined with any RL algorithm wherein the TD-target is modelled using n-step returns. TD-learning is fundamental to RL. Thus the problem of learning better value function estimates is always going to be important and we believe that we have taken significant steps towards it with this work.

References

[Andoni and Indyk] Andoni, A., and Indyk, P. 2006. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, 459–468. IEEE.

[Bellemare et al.] 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.

[Bellemare et al.] Bellemare, M.; Srinivasan, S.; Ostrovski, G.; Schaul, T.; Saxton, D.; and Munos, R. 2016. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, 1471–1479.

[Bengio and others] Bengio, Y., et al. 2009. Learning deep architectures for ai. *Foundations and trends® in Machine Learning* 2(1):1–127.

[Charikar] Charikar, M. S. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, 380–388. ACM.

[Gruslys et al.] Gruslys, A.; Azar, M. G.; Bellemare, M. G.; and Munos, R. 2017. The reactor: A sample-efficient actor-critic architecture. *arXiv preprint arXiv:1704.04651*.

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

[Jaderberg et al.] Jaderberg, M.; Mnih, V.; Czarnecki, W. M.; Schaul, T.; Leibo, J. Z.; Silver, D.; and Kavukcuoglu, K. 2017. Reinforcement learning with unsupervised auxiliary tasks. *To appear in 5th International Conference on Learning Representations*.

[Kolter and Ng] Kolter, J. Z., and Ng, A. Y. 2009. Near-bayesian exploration in polynomial time. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 513–520. ACM.

[Krizhevsky, Sutskever, and Hinton] Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems (NIPS)* 1097–1105.

[Lai and Robbins] Lai, T. L., and Robbins, H. 1985. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics* 6(1):4–22.

[LeCun, Bengio, and Hinton] LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *Nature* 521(7553):436–444.

[Martin et al.] Martin, J.; Sasikumar, S. N.; Everitt, T.; and Hutter, M. 2017. Count-based exploration in feature space for reinforcement learning. *arXiv preprint arXiv:1706.08090*.

[Mnih et al.] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature*.

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

[Ostrovski et al.] Ostrovski, G.; Bellemare, M. G.; Oord, A. v. d.; and Munos, R. 2017. Count-based exploration with neural density models. *arXiv preprint arXiv:1703.01310*.

[Peng and Williams] Peng, J., and Williams, R. J. 1996. Incremental multi-step q-learning. *Machine learning* 22(1):283–290.

[Puterman] Puterman, M. L. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[Schulman et al.] 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*.

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

[Sharma et al.] Sharma, S.; J, G. R.; Ramesh, S.; and Ravindran, B. 2017. Learning to mix n-step returns: Generalizing lambda-returns for deep reinforcement learning. *CoRR* abs/1705.07445.

[Silver et al.] Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2017a. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

[Silver et al.] Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017b. Mastering the game of go without human knowledge. *Nature* 550(7676):354.

[Strehl and Littman] Strehl, A. L., and Littman, M. L. 2008. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences* 74(8):1309–1331.

[Strehl, Li, and Littman] Strehl, A. L.; Li, L.; and Littman, M. L. 2009. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research* 10(Nov):2413–2444.

[Sutton and Barto] Sutton, R. S., and Barto, A. G. 1998. Introduction to reinforcement learning. *MIT Press*.

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

[Tang et al.] Tang, H.; Houthooft, R.; Foote, D.; Stooke, A.; Chen, O. X.; Duan, Y.; Schulman, J.; DeTurck, F.; and Abbeel, P. 2017. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 2750–2759.

[Thomas et al.] Thomas, P. S.; Niekum, S.; Theocharous, G.; and Konidaris, G. 2015. Policy evaluation using the ω-return.

[Todorov, Erez, and Tassa] Todorov, E.; Erez, T.; and Tassa, Y. 2012. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–5033. IEEE.

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