Proceedings of the Fifteenth AAAI Conference on Artificial
Intelligence and Interactive Digital Entertainment (AIIDE-19)

Terminal Prediction as an Auxiliary Task for Deep Reinforcement Learning

Bilal Kartal,* Pablo Hernandez-Leal,* Matthew E. Taylor
Borealis AI
Edmonoton, Canada
{bilal.kartal, pablo.hernandez, matthew.taylor}@borealisai.com

Abstract

Deep reinforcement learning has achieved great successes in recent years, but there are still open challenges, such as convergence to locally optimal policies and sample inefficiency. In this paper, we contribute a novel self-supervised auxiliary task, i.e., *Terminal Prediction* (TP), estimating temporal closeness to terminal states for episodic tasks. The intuition is to help representation learning by letting the agent predict how close it is to a terminal state, while learning its control policy. Although TP could be integrated with multiple algorithms, this paper focuses on Asynchronous Advantage Actor-Critic (A3C) and demonstrating the advantages of A3C-TP. Our extensive evaluation includes: a set of Atari games, the BipedalWalker domain, and a mini version of the recently proposed multi-agent Pommerman game. Our results on Atari games and the BipedalWalker domain suggest that A3C-TP outperforms standard A3C in most of the tested domains and in others it has similar performance. In Pommerman, our proposed method provides significant improvement both in learning efficiency and converging to better policies against different opponents.

Introduction

Deep reinforcement learning (DRL) combines reinforcement learning (Sutton and Barto 2018) with deep learning (LeCun, Bengio, and Hinton 2015), enabling better scalability and generalization for high-dimensional domains. DRL has been one of the most active areas of research in recent years with great successes such as mastering Atari games from only raw images (Mnih et al. 2015), a Go playing agent skilled well beyond any human player (Silver et al. 2017), and very recently, great successes in multiagent games (e.g., DOTA 2 and Starcraft II). Reinforcement learning has also been applied for interactive narrative generation (Wang et al. 2017) and learning companion NPC (Non-Player Character) behaviors (Sharifi, Zhao, and Szafron 2010). For a more general and technical overview of DRL, please see the recent comprehensive surveys (Arulkumaran et al. 2017; François-Lavet et al. 2018; Hernandez-Leal, Kartal, and Taylor 2018).

*Equal contribution

One of the biggest challenges for reinforcement learning is sample efficiency (Yu 2018). Data hungriness of model-free RL methods is only aggravated when the reward signals are sparse, delayed, or noisy. Standard RL problem formulations with non-linear function approximation (i.e., DRL) combine representation learning together with policy learning. In this case, the problem is further deepened when rewards are sparse since most of the collected experiences do not produce a learning signal for the agent, thus delaying representation learning for the environment. To address these issues, the concept of *auxiliary tasks* was introduced where an RL agent can learn from all experiences independent of external reward signals (Shelhamer et al. 2016; Sutton and Barto 2018). *Auxiliary tasks* can be any task that the RL agent can predict and observe from the environment in a self-supervised fashion such as reward prediction (Jaderberg et al. 2017), or predicting a game specific feature such as presence/absence of enemies in the current observation (Lample and Chaplot 2017). Also note that *auxiliary tasks* are different from model-based RL where the learned model is used for planning (Oh et al. 2015; Leibfried, Kushman, and Hofmann 2016). In contrast, auxiliary tasks were originally presented as *hints* that improved the network performance and learning time (Suddarth and Kergosien 1990). In a minimal example of a small neural network it was shown that adding an auxiliary task effectively removed local minima. Thus, the auxiliary losses are expected to give more ambient gradients, not necessarily yield a generative model (Shelhamer et al. 2016).

Different auxiliary tasks have been successfully evaluated such as: reward prediction (Jaderberg et al. 2017), modeling the inverse dynamics of the environment (Shelhamer et al. 2016), and depth prediction (Mirowski et al. 2016). In contrast, we propose *Terminal Prediction (TP)*, where the intuition is to help the representation learning by letting the agent predict how close it is to a terminal state while learning the standard actor policy. TP targets are similarly computed in a self-supervised fashion, but they are independent of reward sparsity of the game or any other domain dynamics that might render representation learning challenging such as drastic changes in domain visuals.

In this paper, we consider Asynchronous Advantage

Actor-Critic (A3C) (Mnih et al. 2016) as a baseline algorithm as it is one of the frontier approaches among asynchronous distributed deep RL techniques. Then, we make the following contributions:

- We propose a novel auxiliary task, namely *Terminal Prediction (TP)*, aiming to enhance the RL agent with a capability of predicting a measure of temporal closeness to expected terminal states, i.e. likely to be reached with the current agent policy, without extra signals from the environment. In this work, we propose A3C-TP, which results from integrating TP task to A3C with minimal refinements to ensure that it is still on-policy. Note that even though we showcase TP with A3C, it can be combined with other deep RL methods.
- We conduct experiments on a diverse set of Atari games and the BipedalWalker domain where A3C-TP either outperforms or performs similar to the standard A3C method. We also conduct experiments on a mini version of a recently released multi-agent domain, i.e. Pommerman (Resnick et al. 2018), showing that A3C-TP both learns faster and converges to better policies, compared to A3C, against different opponents.

Related Work

Reinforcement learning approaches mainly fall under three categories: value-based methods such as Q-learning (Watkins and Dayan 1992) or Deep-Q Network (Mnih et al. 2015); policy-based methods such as REINFORCE (Williams 1992); and a combination of value- and policy-based techniques, i.e. actor-critic methods (Konda and Tsitsiklis 2000). In particular, in the last category several distributed actor-critic based DRL algorithms have been recently proposed (Jaderberg et al. 2017). One notable example is A3C (Asynchronous Advantage Actor-Critic) (Mnih et al. 2016), which is an algorithm that employs a *parallelized* asynchronous training scheme (using multiple CPU cores) for efficiency.

Recently, auxiliary tasks have been proposed to improve representation learning in deep RL. For example, Mirowski et al. (2016) studied self-supervised tasks in a navigation problem. Their results show that augmenting the RL agent with auxiliary tasks supports representation learning which provides richer training signals that enhance data efficiency. Lample and Chaplot (2017) proposed to add an auxiliary task to a DRL agent in the Doom game; in particular, the agent was trained to predict the presence/absence of enemies in the current observation. Lastly, a concurrent work (Leibfried and Vrancx 2018) proposed a model based DRL architecture based on Deep-Q-Network which predicts: Q-values, next frame, rewards, and a binary terminal flag that predicts whether the episode will end or not. The terminal flag is similar to our *Terminal Prediction* except that (i) our method is not model-based, and (ii) we formulate the prediction problem as a regression problem rather than classification, thus we gather a lot more self-supervised signals that are automatically class-balanced (this is particularly essential for tasks with long episodes).

Another related work to ours is the UNREAL framework (Jaderberg et al. 2017) which is built on top of the A3C with several refinements and *auxiliary task* integration. In particular, UNREAL proposes to learn a reward prediction based task besides a pixel-control based task to speed up learning by improving representation learning. In contrast to on-policy A3C, UNREAL uses an experience replay buffer that is sampled with more priority given to positively rewarded interactions to improve the critic network. Our method, A3C-TP, differs from UNREAL in several ways: (i) We do not introduce the additional critic improvement step – to better isolate the gain of our auxiliary task over vanilla A3C. (ii) Even though we also integrate an auxiliary task, we keep the resulting method still on-policy with minimal refinements without an experience replay buffer which might require correction for stale experience data. (iii) UNREAL's reward-prediction requires class balancing of observed rewards in an off-policy fashion depending on the game reward sparsity and distribution whereas *Terminal Prediction* is balanced automatically, it can be applied within on-policy DRL methods, and it generalizes better for episodic tasks independently of the domain-specific reward distribution.

Preliminaries

We start with the standard reinforcement learning setting of an agent interacting in an environment over a discrete number of steps. At time t the agent in state s_t takes an action a_t and receives a reward r_t. The state-value function is the expected return (sum of discounted rewards, $R_{t:\infty} = \sum_{k=t}^{\infty} \gamma^k r_k$) from state s following a policy $\pi(a|s)$:

$$V^{\pi}(s) = \mathbb{E}[R_{t:\infty}|s_t = s, \pi],$$

and the action-value function is the expected return following policy π after taking action a from state s:

$$Q^{\pi}(s, a) = \mathbb{E}[R_{t:\infty}|s_t = s, a_t = a, \pi].$$

A3C (Asynchronous Advantage Actor-Critic) is an algorithm that employs a *parallelized* asynchronous training scheme (e.g., using multiple CPUs) for efficiency; it is an on-policy RL method that does not need an experience replay buffer. A3C allows multiple workers to simultaneously interact with the environment and compute gradients locally. All the workers pass their computed local gradients to a global network which performs the optimization and synchronizes the updated actor-critic neural network parameters with the workers asynchronously. A3C maintains a parameterized policy (actor) $\pi(a|s;\theta)$ and value function (critic) $V(s;\theta_v)$, which are updated as follows: $\triangle\theta = \nabla_\theta \log \pi(a_t|s_t;\theta) A(s_t, a_t;\theta_v)$ and $\triangle\theta_v = A(s_t, a_t;\theta_v)\nabla_{\theta_v} V(s_t)$ where

$$A(s_t, a_t;\theta_v) = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V(s_{t+n}) - V(s_t),$$

with $A(s, a) = Q(s, a) - V(s)$ representing the *advantage* function, commonly used to reduce variance.

The policy and the value function are updated after every t_{max} actions or when a terminal state is reached. It is

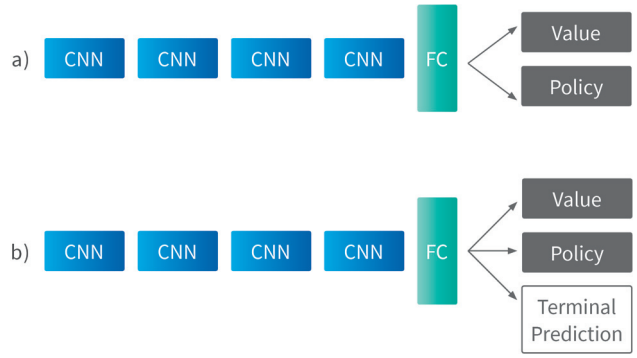


Figure 1: CNN represents convolutional neural network layers, and FC represents fully-connected layer. (a) A standard actor-critic neural network architecture for A3C with policy (actor) and value (critic) heads. (b) The neural network architecture of A3C-TP is identical to that of A3C, except the auxiliary TP head.

common to use a softmax output layer for the policy head $\pi(a_t|s_t;\theta)$ and one linear output for the value function head $V(s_t;\theta_v)$, with all non-output layers shared, see Figure 1 (a).

The loss function for A3C is composed primarily of two terms: policy loss (actor), $\mathcal{L}_\pi$, and value loss (critic), $\mathcal{L}_v$. An entropy loss for the policy, $H(\pi)$, is also commonly added, which helps to improve exploration by discouraging premature convergence to suboptimal deterministic policies (Mnih et al. 2016). Thus, the loss function is given by:

$$\mathcal{L}_{\text{A3C}} = \lambda_v \mathcal{L}_v + \lambda_\pi \mathcal{L}_\pi - \lambda_H \mathbb{E}_{s\sim\pi}[H(\pi(s,\cdot,\theta)]$$

with $\lambda_v = 0.5$, $\lambda_\pi = 1.0$, and $\lambda_H = 0.01$, being standard weighting terms on the individual loss components.

Terminal Prediction as an Auxiliary Task

In this section, we describe our main contribution, i.e., the *Terminal Prediction* (TP) auxiliary task. We name our method A3C-TP and present its neural network architecture in Figure 1 (b). The main motivation is to equip the RL agent with the capability to predict a measure of temporal closeness to terminal states that are more likely to be reached under the agent's current policy.[1] We present how simple to compute and integrate this auxiliary task is with deep RL methods, particularly to A3C.

For the proposed auxiliary task of *Terminal Prediction*, i.e., for each observation under the current policy, the agent predicts a temporal closeness value to a terminal state for the current episode under the agent's current policy. The neural network architecture of A3C-TP is identical to that of A3C, fully sharing parameters, except the additional terminal state prediction head.

[1]Sutton et al. (2011) mentioned a similar idea when discussing *general value functions*: "Suppose we are playing a game for which base terminal rewards are +1 for winning and -1 for losing (...) we might pose an independent question about how many more moves the game will last." Interestingly, we only learned about this connection after writing the paper.

We compute the loss term for the terminal state prediction head, $\mathcal{L}_{TP}$, by using mean squared error between the predicted value of closeness to a terminal state of any given state (i.e., y^p) and the target values approximately computed from completed episodes (i.e., y) as follows:

$$\mathcal{L}_{TP} = \frac{1}{N}\sum_{i=0}^{N}(y_i - y_i^p)^2$$

where N represents the average length of previous episode lengths during training. We assume that the target for ith state can be approximated with $y_i = i/N$ implying $y_N = 1$ for the actual terminal state and $y_0 = 0$ for the initial state for each episode, and intermediate values are linearly interpolated between $[0, 1]$.

We initially used the actual current episode length for N to compute TP labels. However, this delays access to the labels until the episode is over, similar to observing an external reward signal only when the episode ends. Furthermore, it did not provide significant improvement. To be able to have access to dense signals through our auxiliary task, we decided to use the *running average* of episode lengths computed from the most recent 100 episodes, to approximate the actual current episode length for N. We observed that the running average episode length does not abruptly change, but it can vary in correlation with the learned policy improvements. This approximation not only provides learning performance improvement, but also provides significant memory efficiency for distributed on-policy deep RL as CPU workers do not have to retain the computation graph, i.e. used within deep learning frameworks, until episode termination to compute terminal prediction loss.

Given $\mathcal{L}_{TP}$, we define the loss for A3C-TP as follows:

$$\mathcal{L}_{\text{A3C-TP}} = \mathcal{L}_{\text{A3C}} + \lambda_{TP}\mathcal{L}_{TP}$$

where λ_{TP} is a weight term that we set to 0.5 from experimental results (presented in the next section) to enable the agent to mainly focus on optimizing the policy.

The hypothesis is that the terminal state prediction provides some grounding to the neural network during learning with a denser signal as the agent learns not only to maximize rewards but it can also predict approximately how close it is to the end of episode.

Experiments

First, we describe our experimental domains, and then we present results comparing our proposed A3C-TP with A3C.

Domains and Setup

We trained on 6 Atari games (with discrete actions), the BipedalWalker domain (with continuous actions), and Pommerman (with discrete actions and two different opponents). In all cases, we trained both standard A3C and A3C-TP with 3 different random seeds. For Atari games and the BipedalWalker domain, we employed only 8 CPU workers. For the Pommerman domain, we increased to 24 CPU workers as Pommerman is very challenging for model-free RL.

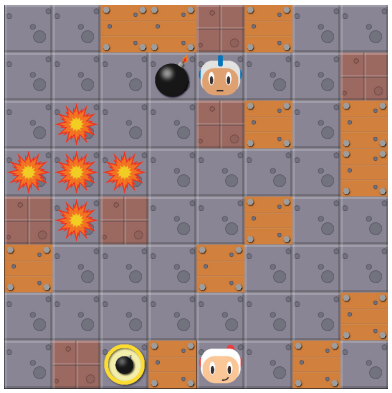


Figure 2: An example of the 8×8 Pommerman board, randomly generated by the simulator. Agents' initial positions are randomly selected among four corners at each episode.

Atari Games and BipedalWalker We conducted experiments using OpenAI Gym (Brockman et al. 2016) using sticky actions and stochastic frame-skipping. We did not perform any reward scaling across the games (Chrabaszcz, Loshchilov, and Hutter 2018) so that agents can distinguish the reward magnitudes rather than just being rewarded with $+1$ or -1.

The Atari benchmark provides several games with diverse challenges that render different methods more successful in different subset of games,[2] e.g., the game Frostbite is mastered by DQN, but A3C performs poorly on it. Thus, we aimed to select a small but diverse set of games with varying difficulties for computational cost reasons. The selected games are: Pong, Breakout, CrazyClimber, Q*bert, BeamRider, and SpaceInvaders. The first 3 games are simpler games where dense reward signals are ubiquitous and reactive strategies learned by the RL method provide high rewards to progress the policy learning; Q*bert, Seaquest, and SpaceInvaders are relatively harder as longer term non-reactive strategies are needed (Mnih et al. 2015; 2016; Chrabaszcz, Loshchilov, and Hutter 2018).

We also experimented with the BipedalWalker (available through OpenAI Gym), which takes only a few hours to train on a standard laptop. Main difference of this domain is that it is a continuous action-space one.

Pommerman The Pommerman environment (Resnick et al. 2018) is based off of the classic console game Bomberman, played with four agents on an 11x11 board. Our experiments use a mini version of the simulator with only two agents on an 8x8 board (see Figure 2). Each agent can execute one of 6 actions at every timestep: move in any of cardinal directions, stay put, or place a bomb. Each cell on the board can be a passage, a rigid wall, or wood. The maps are generated randomly, albeit there is always a guaranteed path between any two agents. Whenever an agent places a bomb it explodes after 10 timesteps, producing flames that have a lifetime of 2 timesteps. Flames destroy wood and kill any agents within their blast radius. When wood is destroyed either a passage or a power-up is revealed. Power-ups can be of three types: increase the blast radius of bombs, increase the number of bombs the agent can place, or give the ability to kick bombs. A single game is finished when an agent dies or when reaching 800 timesteps.

Pommerman is a very challenging benchmark for RL methods. The first challenge is that of sparse and delayed rewards: the game can last up to 800 timesteps and the environment only provides terminal reward per episode. A second issue is the complex environment since tile locations and agents' initial locations are randomized at the beginning of every game (episode). The game board changes within each episode too due to the (dis)appearance of the wood, power-ups, flames, and bombs. The last complication is the multiagent component since the agent needs to best respond to any type of opponent whose behaviour could change based on collected power-ups.

Also, the rewards in Pommerman game can be deceiving. We consider false positive episodes when our agent gets a positive reward because the opponent commits suicide (not due to our agent's combat skill), and false negative episodes when our agent gets a negative reward due to its own suicide (rather than getting killed by an enemy bomb). False negative episodes are a major bottleneck for learning reasonable behaviours with pure-exploration within the RL. Besides, false positive episodes also can reward agents for arbitrary passive survival policies such as camping rather than engaging actively with opponents.

For these reasons, generally a local optimum is commonly learned, i.e., not placing bombs (Resnick et al. 2018).

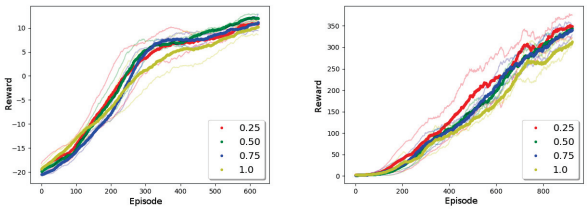
We considered two types of opponents in this domain:

- *Static* opponents: the opponent waits in the initial position and always executes the 'stay put' action. This opponent is the simplest possible opponent as it provides a more stationary environment.
- *Rule-based* opponents: the baseline agent that comes within the simulator. It collects power-ups and places bombs when it is near an opponent. It is skilled in avoiding blasts from bombs. It uses Dijkstra's algorithm on each time-step, resulting in longer training times. Its behaviour is highly stochastic based on the power-ups it has collected, e.g., if it collected certain power-ups, it can place many bombs triggering chain explosions (bombs explode earlier due to being on a flame zone created by another exploding bomb).

Implementation Details

NN Architecture and Hyperparameters: For Atari and BipedalWalker domains, our NN architecture uses 4 convolutional layers where the first two have 32 filters with a kernel size of 5×5, and the remaining ones have 64 filters with a stride and padding of 1 with a filter size of 4×4 and 3×3. This is followed with an LSTM layer with 128 units leading to standard actor-critic heads along with our *Terminal Prediction head*. For Pommerman, our NN architecture similarly uses 4 convolutional layers, each of which has 32 filters and 3×3 filters with a stride and padding size of 1,

[2] Emekligil et al. (2018) recently analyzed which factors contribute to the Deep RL methods' success within Atari games.



(a) Pong (b) Breakout

Figure 3: **Sensitivity Analysis on λ_{TP}:** Moving average over 100 games of the rewards (horizontal lines depict individual episodic rewards) is shown. Each training curve depicts the average and standard deviation of 3 experiments with different random seeds. Both games are trained for a day using 8 CPU cores.

followed with a fully connected layer with 128 hidden units which leads to the three output heads. For both domains, we employed the Adam optimizer with a learning rate of 0.0001. Neural network architectures are not tuned.

NN State Representation: For Atari games, the visual input is fed to the NN as an input whereas in Pommerman, similar to (Resnick et al. 2018), we maintain 22 feature maps that are constructed from the agent observation. These feature channels maintain location of walls, wood, power-ups, agents, bombs, and flames. Agents have different properties such as bomb kick, bomb blast radius, and number of bombs. We maintain 3 feature maps for these abilities per agent. We also maintain a feature map for the remaining lifetime of flames.

Results

In this section, we first present a sensitivity analysis on the *Terminal Prediction* loss weight term. Then, we show training results for both Atari and Pommerman domains to ablate the contribution of our proposed method, i.e., A3C-TP, against the standard A3C.

Sensitivity Analysis on λ_{TP} We conducted preliminary experiments on two (fast to train) Atari games, i.e., Pong and Breakout. We tested learning performance on 4 values $\{0.25, 0.5, 0.75, 1\}$ for the *Terminal Prediction* weight. Each curve is obtained from training with 3 random seeds. As Figure 3 shows, the learning performance gets worse when $\lambda_{TP} = 1$ as the agent puts too much emphasis on learning the auxiliary task compared to the main policy learning task. On the other hand, when $\lambda_{TP} = 0.25$ the variance is higher for both games. Given this preliminary analysis where intermediate values of $\{0.5, 0.75\}$ perform more reliably, we chose to employ $\lambda_{TP} = 0.5$ for all the experiments.

Atari Games We present comparisons of our proposed method, A3C-TP against standard A3C method in Figure 4. Our method significantly improves upon standard A3C in Q*bert and CrazyClimber games while performing similarly in the rest of the tested games. As we employed relatively easy games Pong and Breakout games for sensitivity analysis for A3C-TP, we omitted them from Figure 4. Our hypothesis is that A3C-TP does not hurt the policy learning performance, however it can provide improvement in some benchmarks.

BipedalWalker We also benchmark on a continuous action-space domain, BipedalWalker. The experimental results in Figure 4 (e) show that A3C-TP outperforms A3C, and it has relatively lower variance.

Pommerman We present comparison results based on the training performance in terms of converged policies and time-efficiency against *Static* and *Rule-based* opponents.

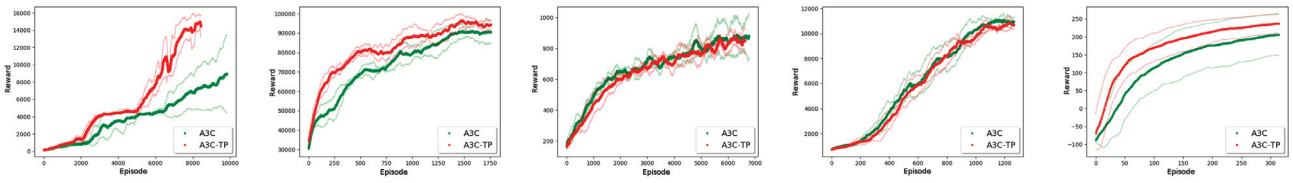
The training results for A3C and our proposed method, A3C-TP, against a *Static* opponent are shown in Figure 5 (a). Our method both converges much faster compared to the standard A3C. As *Static* opponents do not commit suicide, there are no false positives in observed positive rewards. However there are still false negatives due to our agent's possible suicides during training that negatively reinforces the essentially needed bombing skill. In this scenario, A3C takes almost one million episodes to converge because the agent needs to learn to execute a series of actions (without killing itself): get close to the enemy, place a bomb near the enemy and stay alive until the bomb explodes in 10 timesteps, eliminating the enemy. Only after successful execution of these actions, the agent obtains a non-zero external reward signal. One reason we present results and analysis against *Static* opponents is to convey how challenging the Pommerman domain is for purely model-free RL.

Against the *Rule-based* opponent our method learns faster, and it finds a better policy in terms of average rewards, see Figure 5 (b). Training against *Rule-based* opponents takes much longer, possibly due to episodes with false positive rewards.

Discussion and Future Work

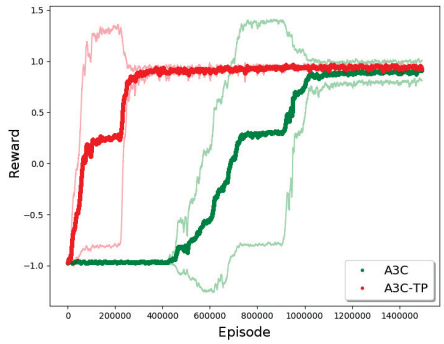
Atari games have been heavily used by RL researchers, but Pommerman was recently released and there are many open problems. Some recent works have used Pommerman as a test-bed, e.g. Zhou et al. (2018) proposed a hybrid method combining rule-based heuristics with depth-limited search. Resnick et al. (2019) proposed a framework that uses a single demonstration to generate a training curriculum for sparse reward RL problems, including Pommerman, assuming episodes can be started from arbitrary states. In this work, we use this highly challenging domain and show that our proposed method provides significant improvement for sample efficiency.

We want to note that there are already a variety of auxiliary tasks proposed in the literature, and they are in general heuristics proposed as additional self-supervised learning targets to help RL agents to improve its representation learning with more contextual information. A recent work shows that using some auxiliary tasks can result in worse performance for the main task (Du et al. 2018). Thus, depending on the domain properties, different auxiliary tasks are likely to have varying performances. However, we note

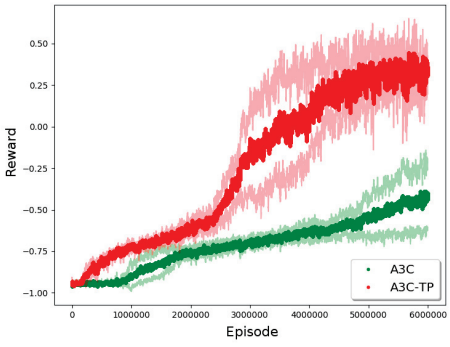


(a) Q*bert (b) CrazyClimber (c) SpaceInvaders (d) BeamRider (e) BipedalWalker

Figure 4: Moving average over 50 games of the rewards (horizontal lines depict individual episodic rewards) is shown. Atari games are trained for 3 days, BipedalWalker is trained for 5 hours, all using 8 CPU workers with 3 random seeds. Our method, A3C-TP, performed no worse than standard A3C in any tested games, and it outperformed A3C in Q*bert and CrazyClimber games, and BipedalWalker domain.



(a) Learning against a *Static* opponent.



(b) Learning against a *Rule-based* opponent

Figure 5: **Pommerman Domain Results:** Moving average over 5k games of the rewards (horizontal lines depict individual episodic rewards) is shown. Each training curve depicts average and standard deviation of 3 experiments with different random seeds. Our method, A3C-TP, outperforms the standard A3C in terms of both learning faster and converging to a better policy in learning against both *Static* and *Rule-based* opponents. The training times was 6 hours for (a) and 3 days for (b) both using 24 CPU cores.

that more theoretical understanding of auxiliary tasks and how to devise them automatically is still an open research avenue (Bellemare et al. 2019; Sutton and Barto 2018).

As future research, we think TP could be employed for safe exploration and within the context of learning from demonstrations. TP values could be used to bias the exploration strategy within the context of safe RL (Garcıa and Fernández 2015). For example, the RL agent could combine state-value function estimates with TP estimate: it can trigger exploitation when both state-value and TP estimates are high (e.g., a win case) whereas exploration could be triggered when state-value estimate is low and TP estimate is high, implying for example, a dead-end for the agent.

Another avenue we want to investigate is to use TP values for better integration of demonstrators and model-free RL within the context of imitation learning. For example, a (search-based, simulated) demonstrator can be asked for action-guidance when the TP value is high and state-value estimate is low, i.e., implying the episode is predicted to be close to a terminal state against the favor of the agent. TP estimates could also be used to determine a reward discounting schedule or the effective horizon length to consider future rewards (Fedus et al. 2019). One main limitation of *Terminal Prediction* auxiliary task is its applicability to only episodic tasks (in contrast to other existing auxiliary tasks).

Conclusions

Deep reinforcement learning has achieved great successes in recent years with the help of novel methods and higher compute power. However, DRL is often sample inefficient. One of the active research area to address this challenge is to use auxiliary tasks. Along this line of research, in this paper, we propose a novel self-supervised auxiliary task, i.e., *Terminal Prediction (TP)*, estimating the temporal closeness to terminal states for episodic tasks, which can be easily integrated to existing DRL methods to improve the learning efficiency. We experimented integrating this task with A3C proposing A3C-TP which improves the policy learning performance across different domains (e.g., Atari, BipedalWalker, and Pommerman) with diverse dynamics. Our TP task is general as can be easily integrated with other DRL methods and with any episodic domain.

References

Arulkumaran, K.; Deisenroth, M. P.; Brundage, M.; and Bharath, A. A. 2017. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine* 34(6):26–38.

Bellemare, M. G.; Dabney, W.; Dadashi, R.; Taiga, A. A.; Castro, P. S.; Roux, N. L.; Schuurmans, D.; Lattimore, T.; and Lyle, C. 2019. A geometric perspective on optimal representations for reinforcement learning. *arXiv preprint arXiv:1901.11530*.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI gym. *arXiv preprint arXiv:1606.01540*.

Chrabaszcz, P.; Loshchilov, I.; and Hutter, F. 2018. Back to basics: Benchmarking canonical evolution strategies for playing atari. *arXiv preprint arXiv:1802.08842*.

Du, Y.; Czarnecki, W. M.; Jayakumar, S. M.; Pascanu, R.; and Lakshminarayanan, B. 2018. Adapting auxiliary losses using gradient similarity. *arXiv preprint arXiv:1812.02224*.

Emekligil, E., and Alpaydın, E. 2018. What's in a game? the effect of game complexity on deep reinforcement learning. In *Workshop on Computer Games*, 147–163.

Fedus, W.; Gelada, C.; Bengio, Y.; Bellemare, M. G.; and Larochelle, H. 2019. Hyperbolic discounting and learning over multiple horizons. *arXiv preprint arXiv:1902.06865*.

François-Lavet, V.; Henderson, P.; Islam, R.; Bellemare, M. G.; Pineau, J.; et al. 2018. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning* 11(3-4):219–354.

Garcıa, J., and Fernández, F. 2015. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research* 16(1):1437–1480.

Hernandez-Leal, P.; Kartal, B.; and Taylor, M. E. 2018. Is multiagent deep reinforcement learning the answer or the question? A brief survey. *arXiv preprint arXiv:1810.05587*.

Jaderberg, M.; Mnih, V.; Czarnecki, W. M.; Schaul, T.; Leibo, J. Z.; Silver, D.; and Kavukcuoglu, K. 2017. Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representations*.

Konda, V. R., and Tsitsiklis, J. N. 2000. Actor-critic algorithms. In *Advances in neural information processing systems*, 1008–1014.

Lample, G., and Chaplot, D. S. 2017. Playing FPS Games with Deep Reinforcement Learning. In *AAAI*, 2140–2146.

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

Leibfried, F., and Vrancx, P. 2018. Model-based regularization for deep reinforcement learning with transcoder networks. *arXiv preprint 1809.01906*.

Leibfried, F.; Kushman, N.; and Hofmann, K. 2016. A deep learning approach for joint video frame and reward prediction in atari games. In *ICML 2017 Workshop on Principled Approaches to Deep Learning*.

Mirowski, P.; Pascanu, R.; Viola, F.; Soyer, H.; Ballard, A. J.; Banino, A.; Denil, M.; Goroshin, R.; Sifre, L.; Kavukcuoglu, K.; et al. 2016. Learning to navigate in complex environments. In *ICLR*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

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

Oh, J.; Guo, X.; Lee, H.; Lewis, R. L.; and Singh, S. 2015. Action-conditional video prediction using deep networks in atari games. In *NIPS*, 2863–2871.

Resnick, C.; Eldridge, W.; Ha, D.; Britz, D.; Foerster, J.; Togelius, J.; Cho, K.; and Bruna, J. 2018. Pommerman: A multi-agent playground. *AIIDE Multi-Agent Workshop*.

Resnick, C.; Raileanu, R.; Kapoor, S.; Peysakhovich, A.; Cho, K.; and Bruna, J. 2019. Backplay:" man muss immer umkehren". *AAAI-19 Workshop on RL in Games*.

Sharifi, A.; Zhao, R.; and Szafron, D. A. 2010. Learning companion behaviors using reinforcement learning in games. In *Sixth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Shelhamer, E.; Mahmoudieh, P.; Argus, M.; and Darrell, T. 2016. Loss is its own reward: Self-supervision for reinforcement learning. *arXiv preprint arXiv:1612.07307*.

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

Suddarth, S. C., and Kergosien, Y. 1990. Rule-injection hints as a means of improving network performance and learning time. In *Neural Networks*. Springer. 120–129.

Sutton, R. S., and Barto, A. G. 2018. *Reinforcement learning: An introduction*.

Sutton, R. S.; Modayil, J.; Delp, M.; Degris, T.; Pilarski, P. M.; White, A.; and Precup, D. 2011. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *AAMAS*.

Wang, P.; Rowe, J. P.; Min, W.; Mott, B. W.; and Lester, J. C. 2017. Interactive narrative personalization with deep reinforcement learning. In *IJCAI*, 3852–3858.

Watkins, C. J., and Dayan, P. 1992. Q-learning. *Machine learning* 8(3-4):279–292.

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

Yu, Y. 2018. Towards sample efficient reinforcement learning. In *IJCAI*, 5739–5743.

Zhou, H.; Gong, Y.; Mugrai, L.; Khalifa, A.; Nealen, A.; and Togelius, J. 2018. A hybrid search agent in pommerman. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*.