Backtracking Restarts for Deep Reinforcement Learning

Zaid Marji and **John Licato**
Advancing Machine and Human Reasoning (AMHR) Lab
Department of Computer Science and Engineering
University of South Florida

Abstract

Manipulating the starting states of a Markov Decision Process to accelerate the learning of a deep reinforcement learning agent is an idea that has been proposed in several ways in the literature. Examples include starting from random states to improve exploration, taking random walks from desired goal states, and using performance-based metrics for starting states selection policy. In this paper, we explore the idea of exploiting the RL agent's trajectories generated during training for use as starting states. The main intuition behind this proposal is to focus the training of the RL agent to overcome its current weaknesses by practicing overcoming failure states by resetting the environment to a state in its recent past. We shall call the idea of starting from a fixed (or variable) number of steps back from recent terminal or failure states 'backtracking restarts'. Our empirical findings show that this modification yields tangible speedups in the learning process.

1 Introduction

Reinforcement learning (RL) is a powerful learning paradigm in the fields of machine learning and artificial intelligence. It is concerned with decision-making problems where the goal is to maximize the expectation of a specific objective function. Deep reinforcement learning (DRL), which incorporates deep neural networks into the reinforcement learning paradigm, has demonstrated tremendous promise at solving complex tasks. There are many examples, including board games (Schrittwieser et al. 2019), video games (Mnih et al. 2013), and autonomous driving simulations (Sallab et al. 2017). The performance on those tasks varies from adequate performance to super-human performance. All of those successes of DRL motivate exploring various techniques to enhance its performance in order to solve problems more efficiently and effectively.

In classical RL settings, tabular methods are used to evaluate policies from all states in the state space. While this is guaranteed to produce optimal results, it places a very insurmountable computational burden that can only be met for simple tasks where the number of states that need to be evaluated is reasonably small. To overcome this burden, approximate solutions were developed, most commonly deep reinforcement learning (DRL), that take advantage of the generalization abilities of deep neural networks.

In this paper, we propose a heuristic strategy for setting the initial states in an episode. The main idea is to let an agent perform a task until it reaches a terminal or failure state, after which the next episode is started at a point in the recent past during the previous episode. The intuition is that the agent will get multiple chances to explore alternative strategies to overcome their recent failure and allow it to focus its training on its current weaknesses. We empirically verify the effectiveness of this strategy in the game of 2048. This is a single-player game with simple rules which will be briefly explained. Our findings indicate that this heuristic does produce tangible speedups in the learning process.

2 Background

A standard Markov Decision Process (MDP) (Puterman 2014) is represented as a tuple of $\mathbf{M} = \langle \mathbf{S}, \mathbf{A}, \mathbf{P}, r, \gamma \rangle$, where $\mathbf{S}$ is the state space containing all the available states $s \in \mathbf{S}$. $\mathbf{P}$ is the state transition operator, representing the probability distribution of $p(s_{t+1}|s_t, a_t)$, which is the probability of the next state s_{t+1} after an action a_t has been executed at the current state s_t. $\mathbf{A}$ is the action space such that $a \in \mathbf{A}$, r is the reward function $r : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \to \mathbb{R}$ and γ is the reward discounting factor.

Reinforcement Learning (RL) uses MDP as the primary model of the world. An agent, the entity that makes decisions, chooses actions to perform in a given state. The MDP represents the environment in which the agent executes their decisions. RL is the task of finding decision policies that maximize the expected sum of discounted future rewards. Deep reinforcement learning (DRL) extends RL using deep learning techniques to allow RL to be usable in complex tasks with large state spaces that are intractable using less sophisticated techniques.

3 Related Work

The effects that start state distributions have on DRL have been studied before. One of the major contributions to the discussion was a paper by Kakade and Langford (2002) that discusses the advantages of what they called 'restart distributions,' which are separate from the start distribution associated with the original MDP. The main argument is that

when using function approximations (as opposed to tabular methods) the approximations will be insensitive to unlikely or infrequently visited states. They argue for a more uniform restart distribution that allows more exploration and more accurate function approximations for a wider range of the state space.

Various strategies have been proposed to manipulate the initial states of RL episodes to achieve better learning performance. A strategy called reverse curriculum generation (Florensa et al. 2017) proposes training an agent in reverse. The agent is trained on states that are few steps away from a goal state, and is then incrementally trained from states that are further and further from the goal states, until it eventually learns to reach the goal state from the initial states which are expected in the original task. Reverse curriculum generation avoids the need to incorporate domain knowledge into the reward function as it only requires providing at least one known goal state, which is cited as one of its main advantages.

A more recently proposed strategy uses some performance metrics (such as goal-reaching probability) that are dynamically estimated as part of the training process to set the restart distribution (Wöhlke, Schmitt, and van Hoof 2020). This technique combines rollouts from states based on current policies to get a measure of how likely it is to reach goal states, and artificial neural networks are used to generalize those measurements over the state space.

In (Ecoffet et al. 2019), the authors propose a method called 'Go-Explore.' One of its main principles is the idea of revisiting previously encountered promising states to enhance exploration and accelerate learning. They provide strong evidence that remembering and revisiting previously encountered states is a useful strategy. In a similar vein, the authors in (Tavakoli et al. 2018) propose three different strategies for restart distributions which are based on previously encountered states. The first is called 'uniform restart,' where recent states are stored in a visited states buffer, and the initial states are uniformly selected from the set of visited states and a sample of the start states of the original MDP. The second is called 'prioritized restart,' where states are stored in a buffer similar to the first approach. However, the states are prioritized based on the TD-error. The third approach is called 'episode restart,' where only states from episodes that received high total undiscounted rewards are stored. When the visited episodes buffer is full, the agent maintains the most rewarding episodes in its buffer.

4 Backtracking Restarts

In this paper, we propose a technique we shall call 'backtracking restarts.' The idea is to let an agent perform a task until it reaches a terminal state or a failure state. Once such a state is reached, we take a number of steps backward from that state on the trajectory of that specific episode or some recent episode. The number of steps backward can be a fixed number of steps, a fixed percentage of the trajectory length, a random number of steps, or dynamically decided. The number of steps is intended to be enough in most situations to change the outcome of the episode. In other words, it is intended to provide a meaningful second chance for the agent to make an improvement over its previous performance or to explore a different evolution of the trajectory from a previous decision point. We are assuming that either the environment has enough stochasticity for different outcomes to occur, or the agent has enough variance in its decision-making to affect the outcome (or both).

Algorithm 1: Pseudocode for Backtracking Restarts

Require: Backtracking probability p_b, Backtracking length L, Exploratory random start probability p_e
for *each episode* **do**
  if $backtracking = on$ *and* $length(buffer) \geq L$ *and* $random[0,1] \leq p_b$ **then**
    state ← buffer[length(buffer)-L]
    truncate(buffer, L)
  else if $exploring = on$ *and* $random[0,1] \leq p_e$ **then**
    state ← random state
    clear(buffer)
  else
    state ← normal start state
    clear(buffer)
  end
  for *each environment step* **do**
    Observe state s
    append(buffer, s)
    $a \sim \pi(\cdot|s)$
    Apply action a and observe s', r
    Update with $< s, a, s', r >$
  end
end

Unlike the previously discussed techniques that start the training from states close to the goal states and go backward, this technique moves the initial states forward as the agent becomes more capable of making progress in the environment. This means that for this technique to work, we need a dense reward function that signals the agent's progress towards its goals, or a task that is naturally defined in terms of dense reward functions. On the other hand, this means that we do not need to provide goal states or expert demonstrations as in the aforementioned techniques. This is especially useful for tasks with no natural goal states and problems with a more open-ended nature where the total reward achievable has no well-known upper limit.

Moreover, this strategy addresses the need to focus the training where the agent is lacking. In many tasks, the episodes might be lengthy, and the agent has more or less mastered the early phases of a task. As the task gets progressively harder, the RL agent will fail at the later phases. By focusing on the later phases, it should learn faster. It is still important to let the RL agent keep its skill of the early stages; therefore, it should start normally with some probability.

5 The Game of 2048

The game 2048 is a simple single-player game where the objective is to create a tile with the numeric value 2048.[1] The game is played on a 4x4 board. The game starts with two tiles randomly placed on the board. Each location on the board can be either an empty slot or contains a tile with a numeric face value. The initial tiles can be either 2s or 4s. The player has four moves to consider: Up, down, left, and right. All tiles slide towards the direction of the player's move whenever possible. If a tile slides towards another tile of the same face value, they will merge into a single tile that is the total sum of their individual values. If the player's move was a legal move that is at least one tile has moved or merged, then a new tile will be added that is either a 2-tile or a 4-tile at a randomly chosen empty slot. The game ends when no legal moves are possible. For additional details, please refer to the link in the footnotes.

In some variations of the game, the game ends in a winning state if a 2048-tile has been created, which is considered the goal of the game and where the name originates. In other variations, the game is open-ended, and the goal is to create the highest tile possible. In this paper, we consider the second open-ended variant. Creating 2048-tiles is a challenging task for most human players, let alone creating 4096-tiles or higher. However, our experiments show that the agent obtains a 2048-tile consistently after some training, and therefore the open-ended variant is well suited for our purposes.

6 Experimental Setup

In order to establish some empirical results, we ran several experiments. All experiments were performed with an algorithm that we shall refer to as 'DQN with lookahead.' This algorithm is very similar to standard DQN (Mnih et al. 2015). In all experiments, we used a replay buffer of size 1 million. We update the target network every 25 thousand agent steps. We perform an evaluation run every 250 thousand agent steps, and each evaluation is 20 thousand agent steps long. However, in our experiments, the DQN was unstable and failed to produce any meaningful results. We modified the DQN by adding 1 step of lookahead at decision time, which can be thought of as a form of truncated rollouts. The algorithm works as follows: When deciding which action to take, DQN would pass the current state through the Q-Network, which outputs an estimate for the action-values $Q(s, a)$ for all possible actions and selects the action with the highest estimated action-value. When acting greedily, DQN's unmodified decision policy is:

$$\pi_g(s) = \arg\max_a Q(s, a) \tag{1}$$

However, instead of using estimates based on the current state, we select all actions in turn. For each action $a_i \in \mathbf{A}$, we obtain 5 samples of the next state $s'_{i,j} \sim p(\cdot|s, a_i), j \in \{1, .., 5\}$, and obtain a state-value estimate by finding the maximum action-value for the next state $\max_{a'} Q(s'_{i,j}, a')$. Finally, we average over the state-value estimates and choose the action with the highest estimate. The following formula summarizes our modified decision policy:

$$\pi_g(s) \tag{2}$$
$$= \arg\max_a \left[\frac{\sum_j \left[r(s, a, s') + \gamma \max_{a'} Q(s', a') \right]_{s' \sim p(\cdot|s,a)}}{N} \right]$$

Since the game is played in an open-ended setting, the reward structure is designed to reflect progress. The higher valued tiles present on the board, the more the agent is rewarded. We decided to use a non-linear scoring system that favors higher valued tiles such that -for example- one 1024-tile is more valuable than two 512-tiles. We used the following formula to calculate the score:

$$score(s) = \sum_{v \in tiles(s)} v * \log_2 v \tag{3}$$

The reward is defined as the difference between the score of the current state and the next state.

$$r(s, a, s') = score(s') - score(s) \tag{4}$$

We performed the experiment in six configurations. The first parameter in those configurations is whether backtracking restarts are used and with what probability. We used three settings for this parameter: 98%, 80%, and not used. The probability defines how likely an episode will be started from the history buffer of the previous episode. If using the history buffer is chosen, the episode will start 50 steps back from the terminal state of the previous episode. The second parameter is whether exploratory random starts are used or not. If using the history buffer is not selected, then this setting will be selected with a fixed probability of 80%. If selected, the episode will start from a randomly generated state. Otherwise, the episode will start from an initial state according to the game's rules. All of those configurations only affect episodes used for training purposes. Algorithm 1 shows pseudocode for the backtracking algorithm in training mode. All episodes used for evaluation purposes start from an initial state according to the game's rules.

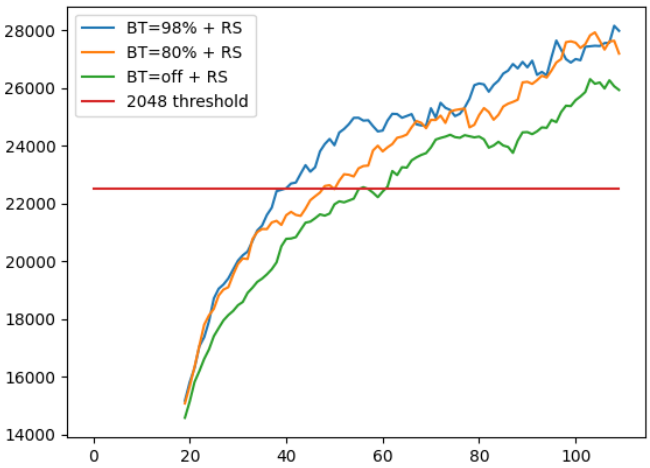
7 Empirical Results

Figure 1(a) shows the moving average of the undiscounted sum of rewards (ie. the score) during the evaluations on different configurations using exploratory random starts. The X-axis represents the number of evaluation runs (which happen after every 250 thousand agent steps in training mode). Each point on the Y-axis represents the average score of all episodes in the last 20 evaluation runs. As it can be observed, the backtracking restarts positively affect the RL agent's learning speed. Figure 1(b) shows similar results when exploratory random starts are not used.

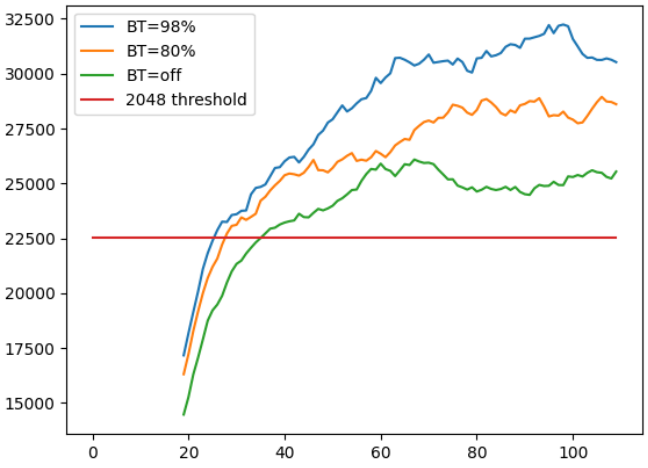
8 Conclusions and Future Work

Reinforcement learning is at the forefront of advancing machine capabilities beyond human performance. Supervised learning and imitation learning approaches

[1] The game can be played at: https://play2048.co/



(a) Using exploratory random starts



(b) Not using exploratory random starts

Figure 1: Summary of empirical results

are generally limited by human capabilities. For machines to become an indispensable and proactive asset to human goals, they need to be trusted to provide knowledge and insights that are at least on par, if not even exceed, human capabilities. Reinforcement learning is a computationally demanding learning paradigm. We need to find reliable ways to improve its effectiveness and efficiency to learn tasks within reasonable computational, financial, and time resource constraints. For this reason, it is essential to research ways to optimize its learning process.

This work presents some heuristics to accelerate the learning of DRL agents under certain assumptions, including the ability to start a MDP at an arbitrary state, and the availability of a reward function that signals progress towards a goal. However, it does not require knowledge of goal states or access to expert demonstrations. We have demonstrated the utility of this heuristic on the game of 2048 and verified it empirically, and demonstrated the potential for its use in other applications. Next steps include performing more experimentation to get a clearer idea of the conditions and applications where this approach works effectively and investigate means of optimizing the parameters for various applications.

9 Acknowledgements

This material is based upon work supported by the Air Force Office of Scientific Research under award numbers FA9550-17-1-0191 and FA9550-18-1-0052. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Air Force.

References

Ecoffet, A.; Huizinga, J.; Lehman, J.; Stanley, K. O.; and Clune, J. 2019. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*.

Florensa, C.; Held, D.; Wulfmeier, M.; Zhang, M.; and Abbeel, P. 2017. Reverse curriculum generation for reinforcement learning. In *Conference on robot learning*, 482–495. PMLR.

Kakade, S., and Langford, J. 2002. Approximately optimal approximate reinforcement learning. In *In Proc. 19th International Conference on Machine Learning*. Citeseer.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

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):529–533.

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

Sallab, A. E.; Abdou, M.; Perot, E.; and Yogamani, S. 2017. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging* 2017(19):70–76.

Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; et al. 2019. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*.

Tavakoli, A.; Levdik, V.; Islam, R.; Smith, C. M.; and Kormushev, P. 2018. Exploring restart distributions. *arXiv preprint arXiv:1811.11298*.

Wöhlke, J.; Schmitt, F.; and van Hoof, H. 2020. A performance-based start state curriculum framework for reinforcement learning. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 1503–1511.