Forward-Backward Reinforcement Learning

Ashley D. Edwards[1], Laura M. Downs[2], and James C. Davidson[3]

***Abstract*— Goals for reinforcement learning problems are typically defined through hand-specified rewards. To design solutions to such problems, developers of learning algorithms must inherently be aware of what the task goals are, yet we often require agents to discover them on their own without any supervision beyond these sparse rewards. While much of the power of reinforcement learning derives from the concept that agents can learn with little guidance, this requirement greatly burdens the training process. If we relax this one restriction and endow the agent with knowledge of the reward function, and in particular of the goal, we can leverage backwards induction to accelerate training. To achieve this, we propose training a model to learn to take imagined reversal steps from known goal states. Rather than training an agent exclusively to determine how to reach a goal while moving forwards in time, our approach travels backwards to jointly predict how we got there. We evaluate our work in Gridworld and Towers of Hanoi and we empirically demonstrate that it yields better performance than standard DDQN.**

I. INTRODUCTION

Reinforcement Learning (RL) problems are often formulated with the agent blind to the task reward of the environment. However, for many sparse reward problems, including goal-directed tasks such as point-to-point navigation, pick-and-place manipulation, assembly, etc., endowing the agent with knowledge of the reward function is both feasible and practical for learning generalizable behavior. In general, developers of these problems often know what the task goals are, but not necessarily how to solve them. In this paper, we will describe how we can leverage our knowledge of goals to enable learning of behaviors in these regions before the agent even reaches them. This formulation may be easier to solve than approaches that initialize learning from the start alone. For example, if we know the desired location, pose, or configuration of a task, then we can reverse the actions that brought us there, rather than forcing the agent to solve these difficult problems solely through random discovery.

In this paper, we introduce Forward-Backward Reinforcement Learning (FBRL), which introduces backward induction, to enable our agent to reason backwards in time. Through an iterative process, we both explore forwards from the start position and backwards from the target/goal. To achieve this we introduce a learned backwards dynamics model to explore in reverse from known goal states and update values within this local neighborhood. This has the effect of "spreading out" sparse rewards so that they are easier to discover, thus accelerating the learning process.

[1]Georgia Institute of Technology, work done as an intern at Google Brain aedwards8@gatech.edu

[2]Google Brain ldowns@google.com

[3]Google Brain jcdavidson@google.com

Standard model-based approaches aim to reduce the amount of experience necessary to learn good policies by imagining steps *forward* and using these hallucinated events to augment the training data. However, there is no guarantee that the projected states will lead to the goal, so these roll-outs may be inadequate. The ability to predict the result of an action does not necessarily provide guidance about which actions lead to the goal. In contrast, FBRL takes a more guided approach since, given an accurate model, we have confidence that each state visited in a backwards step has a path to the goal.

In the rest of the paper, we will describe the relevant background and related works. We will then formally introduce FBRL, followed by an empirical section in which we evaluate our approach in Gridworld and Towers of Hanoi, and show that it yields better results than standard Deep Double Q-Learning (DDQN) [12]. Finally, we will conclude with discussions for future work.

II. BACKGROUND

Reinforcement Learning (RL) problems are specified through a Markov Decision Process (MDP) $\langle S, A, R, T \rangle$[11]. Here, $s \in S$ describes the states in the environment, $a \in A$ defines the actions the agent can take, $r \in R(s)$ refers to the rewards an agent receives within state s, and $T(s, a, s')$ is a transition model that specifies the probability of entering state s' after taking action a in s. A policy π estimates the probability of taking action a in state s, and we are typically interested in learning an optimal policy that maximizes the expected long-term discounted return. Model-free approaches do not have access to T, and rather learn an action-value function $Q(s, a)$ that predicts the return after experiencing samples $\langle s, a, s', r \rangle$ in the environment:

$$L(\theta) = \mathbb{E}_{(s,a,s',r)\sim D}[r + \gamma \max_{a'} Q(s', a') - Q(s, a)] \quad (1)$$

Here, D is a replay buffer that stores experiences [5]. This loss aims to minimize the TD-error, or the difference between the expected return and current prediction.

Learning Q-values often requires a large quantity of samples. Rather than directly experiencing the states, an alternative method is to jointly use model-based planning to predict values. DYNA-Q [10] makes updates to values by using imagined experiences. In this case, the parameters $\langle s, a, r, s' \rangle$ from Equation 1 may also be obtained from imagined experiences.

III. RELATED WORK

When we have access to the true dynamics model, purely model-based approaches such as dynamic programming can

be used to compute values over all states [11]. Though when the state space is large or continuous, it may be intractable to iterate over the entire state-space. Q-Learning is a model-free approach and updates values in an online manner by directly visiting states, and function approximation techniques such as Deep Q-Learning enable generalizing to unseen ones [5]. Hybrid approaches that combine model-based and model-free information can also be used. DYNA-Q [10], for example, was an early approach that used imagined roll-outs to update the Q-values as if they had been experienced in the true environment. There are more recent approaches as well, for example NAF [4] and I2A [13]. But these approaches only use forward imagination.

A similar approach to our own does value iteration in reverse [14], but this is a purely model-based approach, and it does not learn a reverse model. A related approach performs bidirectional search from the start and goal [2], but that work learns values only, whereas we aim to learn action-values. Another comparable work solves problems by using a reverse curriculum near goal states [3]. However, that approach assumes the agent can be initialized near the goal. We do not make this assumption, as knowing what the goal state is does not mean that we know how to get to it.

Many works have used domain knowledge to help speed up learning, for example through reward shaping [7]. Another approach is to more efficiently use the experiences from the replay buffer. Prioritized experience replay [9] aims to replay samples that have high TD-error. Hindsight experience replay treats each state in an environment as a potential goal so that the system can learn even when it fails to reach the desired target.

The concept of using reverse dynamics is similar to inverse dynamics ([1], [8]). In those approaches, a system predicts the dynamics that yielded a transition between two states. In our approach, we use the state and action to predict the previous state. The purpose of this function is to reverse an action and use this unraveling to learn values near the goal.

IV. APPROACH

Algorithm 1: Forward-Backward RL

while *training* **do**
  /* Forward step */
  Take step from π and update D
  $d \sim D$
  Train $b(\cdot)$, $Q(\cdot)$ with d
  $\widehat{s}_{t+1} \sim G$
  /* Backward step */
  for *imagination steps* **do**
    $a_t \leftarrow$ random or greedy
    $r_t \leftarrow R(\widehat{s}_{t+1})$
    $\widehat{s}_t \leftarrow \widehat{s}_{t+1} - b(\widehat{s}_{t+1}, a_t)$
    $D.append(\widehat{s}_t, a_t, r_t, \widehat{s}_{t+1})$
    $\widehat{s}_{t+1} \leftarrow \widehat{s}_t$
  end
end

We now introduce our approach, Forward-Backward Reinforcement Learning (FBRL). In this work, we utilize both imagined and real experiences to learn values. A *forward* step uses samples of real experiences originating from the start state to update Q-values, and a *backward* step uses imagined states that are asynchronously predicted in reverse from known goal states. We hypothesize that this approach will improve our model of values in the vicinity of the goal, and thus expedite learning. We now describe the preliminaries for our approach.

A. Preliminaries

We specify FBRL problems through a modified MDP $\langle S, A, R, G \rangle$. As before, $s \in S$ corresponds to the states in the environment, $a \in A$ are the actions the agent can take, and $R(s)$ represents the rewards an agent receives in s. We assume that R does not distinguish between real and imagined inputs and can be queried at any time. Finally, $g \sim G$ is a distribution of goal states from which we can sample uniformly.

B. Backwards model

We aim to learn a backward transition model that captures what happens if we undo an action in a state. We use a tuple of experience $\langle s_t, a_t, r_t, s_{t+1} \rangle \sim D$ to learn the model. Rather than predicting the previous state directly, we aim to learn the difference between the two: $\Delta = s_{t+1} - s_t$. This allows the model to learn how states will change, rather than absolute positional information. It reduces the expected range of output values and generally centers them around zero, resulting in a more stable estimate. This formulation is appropriate since we are using states from the start of the problem to learn the backwards model, which is used near goal states that will initially have little training data.

The backwards model is a neural network that is trained to predict Δ, where $b(s_{t+1}, a_t) \rightarrow \widehat{\Delta}$. Now, we can predict the previous state as $\widehat{s}_t = s_{t+1} - b(s_{t+1}, a_t)$. The loss for the backward model then is: $\mathcal{L}_{\theta_b} = \|\Delta - b(s_{t+1}, a_t)\|$, where $\|\cdot\|$ denotes a Huber loss.

In some environments, it may be impossible to learn an accurate deterministic backward model, even if the problem has deterministic actions. For example, if an agent is next to a wall, we might not know if it previously bumped into the wall or if it took a step towards it. Additionally, for discrete-valued problems, it may be difficult to learn a network that can predict discrete values. These issues are compounded further in stochastic settings. To address this we formulate the problem using a variational approach. If we know the distribution over Δ, then we can predict a distribution over potential outcomes. In this formulation, $\widehat{\Delta}$ will represent a probability distribution for each state variable that can be trained using a cross-entropy loss from the true distribution.

C. Action sampling

Another important consideration is how to sample actions that lead to useful updates. Our approach either randomly samples actions or uses a more greedy step that aims to direct

the roll-outs towards the start by moving to states with high Q-values: $\arg\max_{a_t} Q(\widehat{s}_t, a_t)$.

D. Backwards Imagination

Algorithm 1 shows the pseudo-code for our approach. In the *forward step*, we train the agent using experiences from the replay buffer, according to whichever learning paradigm we choose. In this work, we use DDQN. We additionally use real experiences to update the backward model.

The *backward step* takes place asynchronously. During this process, we use backward imagination for a limited amount of steps. Starting from the goal state, the approach samples an action, uses the model to imagine backwards, and then repeats the process from the resulting state. These imagined experiences are used to augment the replay buffer.

It is important to note that initially the backwards model is unlikely to accurately predict the true dynamics model. The model starts by being trained on experience near the starting region. Often, the portion of the dynamics model exercised outside of this initial region will vary significantly, especially near the goal. For example, consider a maze for navigation task where the maze beyond is unknown or the difference in dynamics for a humanoid lying down versus standing up.

While the model may start out being inaccurate, it provides a constantly improving signal that helps formulate the value function, which is then used to guide exploration. In this way, it acts like an intrinsic reward to provide a predicted direction for exploration for the model. Consider again the navigation problem, where the model in the immediate region will learn a factored representation for locomotion, but cannot predict the walls of the maze further away. The hallucinated experience will likely predict movement through walls. While this is is inaccurate, it does provide a shape for the value function that will encourage traveling towards the goal until a wall is discovered. Once discovered, the model will update and the value function will shift to anticipate the presence of the wall. As training progresses, the system will capture larger regional dynamics and start to predict potential global dynamics, e.g., presence of walls beyond what has been directly observed. As the system approaches the goal, the backward model will converge to the real model.

V. EXPERIMENTS

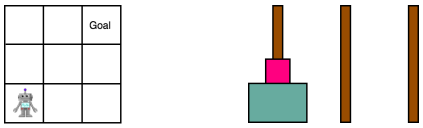

Fig. 1: Gridworld and Towers of Hanoi environments.

The purpose of our experiments is to demonstrate that FBRL can significantly speed up learning in environments with sparse rewards. We evaluate our approach in Gridworld and Towers of Hanoi, illustrated in Figure 1. For comparison we formulate FBRL by augmented DDQN, which we compare against a standard DDQN baseline.

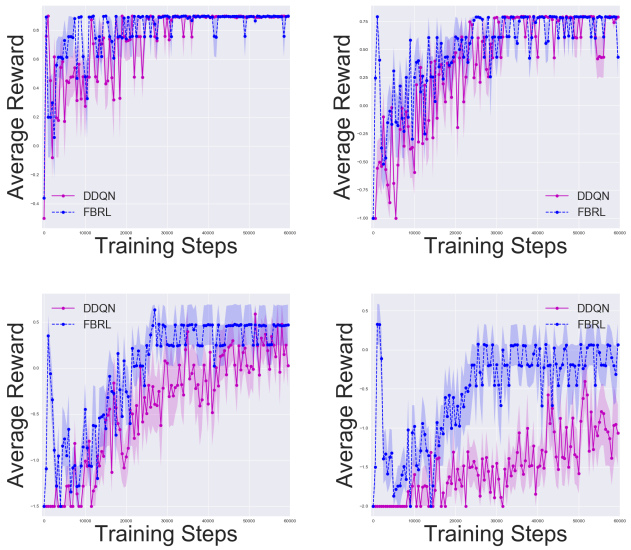

Fig. 2: Results for Gridworld where $n = 5, 10, 15, 20$. We use a fixed horizon of $50, 100, 150, 200$ steps, respectively. The results are averaged over 10 trials.

A. Gridworld

We first evaluate our approach in an nxn Gridworld. We use this environment as it allows us to easily show the benefits of our approach as the reward becomes more sparse. The agent's actions are to move up, down, left, and right by a single unit, and its state consists of its cartesian coordinates. The agent is initialized in the bottom left corner of the grid, and receives a reward of 1 when it reaches the top right. It receives a step cost of $-.01$ per time-step. The inputs to the backward model are x, y and it must learn to predict Δ_x, Δ_y. The model architecture is a fully-connected network with 100 outputs followed by RELU, followed by another fully-connected network with 2 outputs, one for each state dimension. For FBRL, we used 10 steps of imagination with 1 asynchronous stream.

Figure 2 shows the results for running different size gridworlds. The results show that as we increase the size of the grid, i.e., as the goal gets further away, there is a clear advantage for using reverse imagination. The gap between the performance of DDQN compared to FBRL increases as the size gets larger. This suggests the approach is better suited for longer horizon sparse reward environments–but still does not degrade performance for short horizon tasks.

B. Towers of Hanoi

The next environment we evaluate in is n-disc Towers of Hanoi. In this problem, the agent needs to move n discs from the first to the third pillar, but it is only able to place a disc on top of another one if it is smaller than it. The actions are to move each disc to the first, second, or third pillar. It receives a reward of 1 when all discs are in the third pillar and a step cost of $-.01$ per time-step. The inputs to the backward model are bit-strings indicating which pillars each disc is on. For

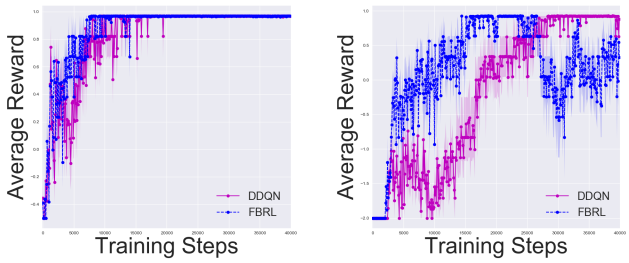

Fig. 3: Results for Towers of Hanoi where $n = 2, 3$. We use a fixed horizon of $50, 100$ steps, respectively. The results are averaged over 10 trials.

example, the environment in Figure 1 has a representation of $[1, 0, 0, 1, 0, 0]$ since the small disc is on the first pillar and the large disc is on the third pillar. The backward model predicts a distribution for each bit over possible Δ values: $P(\Delta = -1), P(\Delta = 0), P(\Delta = 1)$. The model architecture is a fully-connected network with 100 outputs followed by RELU, followed by another fully-connected network with $9n$ outputs, representing the distribution over each bit. For FBRL, we used 5 steps of imagination with 3 asynchronous streams.

Figure 3 shows the results for running Towers of Hanoi with a different number of discs. We again see an advantage for using FBRL as the goal gets further away. When we increase the number of discs, FBRL outperforms DDQN. We did find though that the performance of FBRL degraded for 3 discs, which may be due to overfitting.

VI. CONCLUSION

In this paper, we have introduced an approach for speeding up learning in problems with sparse rewards. We introduced FBRL, which takes imagined steps in reverse from the goal. We demonstrated that this approach can perform better than DDQN in Gridworld and Towers of Hanoi. There are many directions for extending this work. We were interested in evaluating a backward planner, but we could also train using both forward and backward imagination. Another improvement would be to improve the planning policy. We used a exploratory and greedy approach, but did not evaluate how to balance the two. We could also use prioritized sweeping [6], which chooses actions that lead to states with high TD-error.

VII. ACKNOWLEDGEMENTS

We thank Anoop Korattikara, Himanshu Sahni, Sergio Guadarrama, and Shixiang Gu for useful discussions and feedback about this work.

REFERENCES

[1] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems*, pages 5074–5082, 2016.

[2] G. Baldassarre. Forward and bidirectional planning based on reinforcement learning and neural networks in a simulated robot. In *Anticipatory behavior in adaptive learning systems*, pages 179–200. Springer, 2003.

[3] C. Florensa, D. Held, M. Wulfmeier, and P. Abbeel. Reverse curriculum generation for reinforcement learning. *arXiv preprint arXiv:1707.05300*, 2017.

[4] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[6] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning*, 13(1):103–130, 1993.

[7] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

[8] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, volume 2017, 2017.

[9] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[10] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the seventh international conference on machine learning*, pages 216–224, 1990.

[11] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[12] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 16, pages 2094–2100, 2016.

[13] T. Weber, S. Racanière, D. P. Reichert, L. Buesing, A. Guez, D. J. Rezende, A. P. Badia, O. Vinyals, N. Heess, Y. Li, et al. Imagination-augmented agents for deep reinforcement learning. *arXiv preprint arXiv:1707.06203*, 2017.

[14] P. Zang, A. Irani, and C. L. Isbell Jr. Horizon-based value iteration. Technical report, Georgia Institute of Technology, 2007.

APPENDIX

For each experiment, we used a batch-size of 100. The discount factor was .99. The exploration parameter ϵ was initialized to 1 and decayed to .1. The replay memory had a size of 10000 and we collected 10000 initial samples before training DDQN. The architectures for the backwards models are described in the main text.

A. Gridworld

The learning rate for DDQN was $1e^{-3}$. The architecture for DDQN was a fully-connected network with 32 outputs followed by RELU, followed by another fully-connected network with 4 outputs, one for each action. We updated the target network every 100 steps. FBRL had the same settings except we increased the learning rate to $5e^{-3}$.

B. Towers of Hanoi

The learning rate for DDQN was $5e^{-4}$. The architecture for DDQN was a fully-connected network with 32 outputs followed by RELU, followed by another fully-connected network with 9 outputs, one for each action. We updated the target network every 500 steps. Like with Gridworld, we had the same architecture as DDQN, but we found we obtained better results when the learning rate was reduced to $1e^{-4}$.