Reward Backpropagation Prioritized Experience Replay

Yangxin Zhong, Borui Wang, Yuanfang Wang
CS234 Final Project
Stanford University

Stanford | ENGINEERING
Computer Science

The Problem

Sample efficiency is important to reinforcement learning.

- Real world problems with large state spaces suffer from sparse and delayed rewards
- Exploration and learning are sometimes very expensive

With limited data and experience, how can we converge to a good policy more quickly?

Prior Work

DQN[1] uses the deep convolutional neural network as state-action pair function approximation and achieved promising results, but still suffers from low training efficiency resulting from sparse and delayed rewards. Then the model is further combined with Q(λ) method[2] in the DQ(λ)N model[3]. However, this algorithm uses a sequence of transitions from the replay memory in every update step, which may cause the sample correlated issue mentioned in original DQN paper.

Prioritized experience replay (PER) is another state-of-art in sample efficiency topic[4]. It samples each transition with a probability proportional to its priority weight, to replay important transitions more frequently and learn more efficiently.

References

[1] Mnih, V., et al. Human-level con- trol through deep reinforcement learning. *Nature*, 518 (7540):529–533, 2015.

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

[3] Mousavi, S. S., et al. Applying q (λ)-learning in deep reinforcement learning to play atari games. 2017.

[4] Schaul, T., et al. Prioritized experience replay. In *Proceedings of the 4th International Conference on Learning Representations (ICLR 2016).*

Method

A new experience replay method called **Reward Backpropagation**

- Converges 1.5x faster than vanilla DQN and also has a higher performance
- Higher minibatch sampling priority to (s, a, r, s') transitions with non-zero rewards
- Followed by propagating the priorities backward to previous experiences once being sampled and so on.
- Two extra parameters: β, priority weight for non-zero reward; λ, priority decay rate when we start a new round of propagation

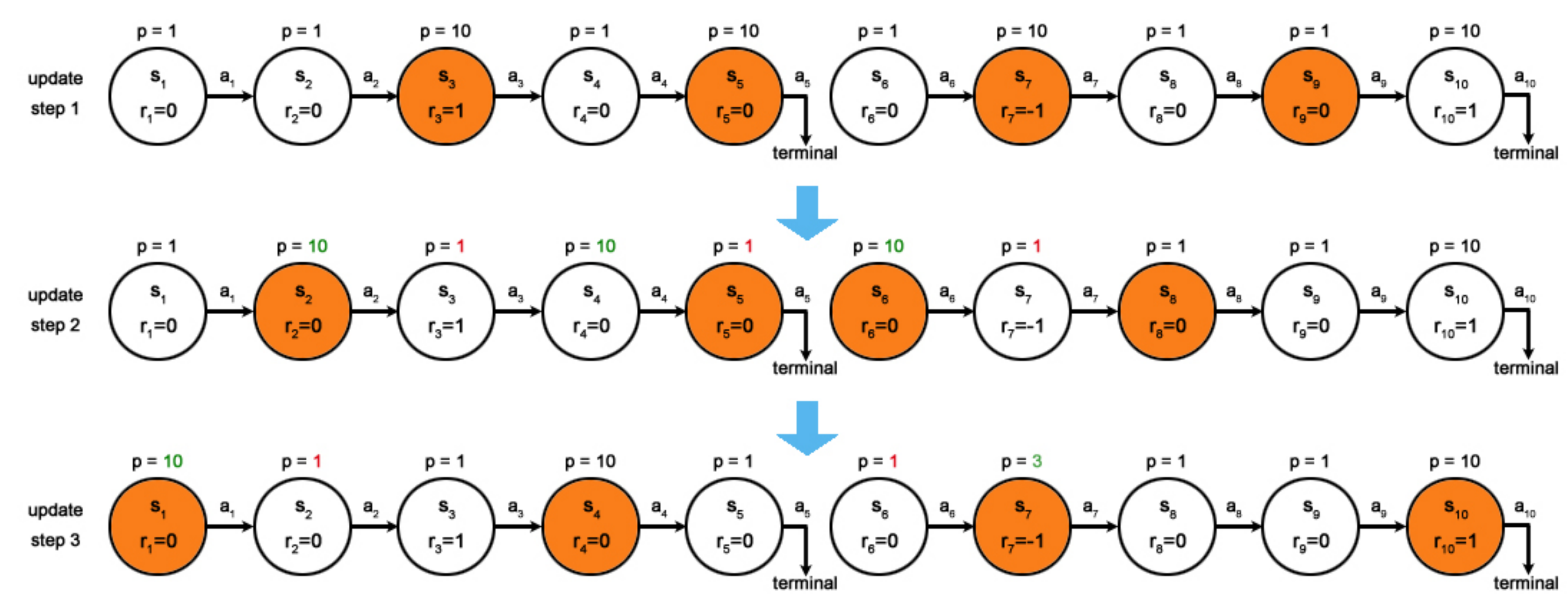

Fig 1. Illustration of priority backpropagation

Each transition observed will be stored in replay memory along with a priority weight p. p=β >> 1 if the transition is terminal or has non-zero reward; otherwise, p=1. When sampling minibatch of transitions, each transition in replay memory will be chosen with a probability proportional to its priority weight.
High priority β is propagated backward identically from a non-zero reward transition or a terminal transition to its previous zero transitions. After the propagation, the high priority of the original transition will be decayed by rate λ. In this loopy way, we propagate the high priority backward again and again until it is decayed down to 1, which is the normal priority.

Algorithm 1 DQN with Reward Backpropagation Experience Replay

Parameters: non-zero reward priority $\beta \gg 1$, priority decay rate λ, learning rate α, minibatch size n.
Initialize DQN parameter θ with random values
Initialize replay memory M with capacity N
for each episode **do**
  Initialize state s
  for each step in the episode **do**
    Choose an action a according to ϵ-greedy policy
    Take action a, observe reward r and next state s'
    if $s' =$ terminal **or** $r \neq 0$ **then**
      $p \leftarrow \beta$
    else
      $p \leftarrow 1$
    end if
    Store transition (s, a, r, s') with priority p in M
    $s \leftarrow s'$
    $b \leftarrow$ sample a minibatch of transitions in M with probabilities proportional to their priority p
    $e \leftarrow 0$
    for each transition (s_i, a_i, r_i, s'_i) with p_i in b **do**
      if $s'_i =$ terminal **then**
        $Q' \leftarrow 0$
      else
        $Q' \leftarrow \max_a Q(s'_i, a; \theta^-)$
      end if
      $e \leftarrow e + [r_i + Q' - Q(s_i, a_i; \theta)] \frac{\partial Q(s_i, a_i; \theta)}{\partial \theta}$
      $(s_j, a_j, r_j, s'_j), p_j \leftarrow \mathit{Predecessor}[(s_i, a_i, r_i, s'_i)]$
      if $s'_j \neq \mathit{terminal}$ **and** $r_j = 0$ **and** $p_i > 1$ **then**
        update $p_j \leftarrow p_i$ in M
      else if $p_i > 1$ **then**
        $(s_k, a_k, r_k, s'_k), p_k \leftarrow$ the first transition after (s_j, a_j, r_j, s'_j) with $s'_k =$ terminal **or** $r_k \neq 0$
        update $p_k \leftarrow \max(\lambda p_i, 1)$ in M
      end if
      update $p_i \leftarrow 1$ in M
    end for
    $\theta \leftarrow \theta + \alpha \cdot \frac{e}{n}$
  end for
end for

Evaluation

We tested our algorithm in the Atari 2600 games. Pong, Breakout and Ice hockey in our experiments since they are less complex compared to other environments and the agent can converge much faster. Experiments show that DQN model combined with our method converges 1.5x faster than vanilla DQN and also has a higher performance.