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.

Method

Algorithm 1: DQN with Reward Backpropagation Experience Replay
Parameters: non-zero reward priority $J \geq 1$, priority decay rate $\lambda$, learning rate $\alpha$, minibatch size $n$.
Initialize DQN parameter $\theta$ 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 $\epsilon$-greedy policy
    Take action $a$, observe reward $r$ and next state $s'$
    if $s' = \text{terminal}$ or $r \neq 0$ then
      $p \leftarrow \beta$
    else
      $p \leftarrow 1$
    end if
    Store transition $(s, a, r, s')$ with priority $p$ in $M$
    $b \leftarrow$ sample a minibatch of transitions in $M$ with probabilities proportional to their priority $p$
  for each transition $(s, a, r, s')$ with $p_i \in b$ do
    if $s_i = \text{terminal}$ then
      $Q' \leftarrow 0$
    else
      $Q' \leftarrow \max_a Q(s', \alpha \theta^-)$
    end if
    $c \leftarrow c + r_i + Q' - Q(s_i, a_i, \theta)$
    $b_i \leftarrow \text{Predecessor}[(s_i, a_i, r_i, s')]$
    if $s_i \neq \text{terminal}$ and $r_0 = 0$ and $p_0 > 1$ then
      update $p_0 \leftarrow p_0$ in $M$
    else if $p_0 > 1$ then
      $(s_i, a_i, r_i, s')$, $p_0 \leftarrow$ the first transition after $(s_j, a_j, r_j, s_j)$ with $s_j = \text{terminal}$ or $r_j \neq 0$
      update $p_0 \leftarrow \max(s_0, 1)$ in $M$
    end if
    update $p_i \leftarrow 1$ in $M$
  end for
  $\theta \leftarrow \theta + \alpha \cdot \frac{c}{p_0}$
end for

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.

References