Prioritized Experience Replay

2022020425 최선묵

Dept. of Mathematics
Korea University

Experience Replay

Experience Replay (or Replay Buffer) is used to address the following two issues:

- strongly correlated updates that break the i.i.d. assumption of stochastic gradient algorithms.
- the rapid forgetting of possibly rare experiences that would be useful later on.

Using a replay memory leads to design choices at two levels:

- which transitions to store; and
- which transitions to replay (and how to do so).

Original experience replay stores all transitions, and they are replayed by sampling uniformly.

Prioritized Experience Replay addresses the latter by prioritizing which transitions are replayed. The priority is measured by the magnitude of their temporal-difference (TD) error.

Prioritized Experience Replay (PER)

- Prioritized Experience Replay is a way to sample the transition i with probability $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$, measured by their TD error, so as to replay important transitions more frequently.

- Before achieving an optimal action-value function $Q^*(s, a)$,
TD-error of a transition will indicate how 'surprising' or unexpected the transition is:
specifically, how far the value is from its next-step bootstrap estimate.

- For TD-error $\delta_i = r_{t+1} + \gamma \max_a \hat{Q}(s_{t+1}, a; \hat{\theta}) - Q(s_t, a_t; \theta)$, the priority p_i is measured as
  * $p_i = |\delta_i| + \epsilon$ in proportional prioritization (ϵ-soft: non-zero probability) ;
  * $p_i = \frac{1}{rank(i)}$ in rank-based prioritization, where
  $rank(i)$ is the rank of transition i when the replay memory is sorted according to $|\delta_i|$.
  It seems to be more robust as it is insensitive to outliers.
  Its heavy-tail property seems to guarantee that samples will be diverse.

  In practice, however, both variants perform similarly.

- The exponent α determines how much prioritization is used ($\alpha = 0$: uniform case).

Annealing the Bias

- The prioritization introduces bias due to the change of sampling distribution.
- It would change the solution that the estimates will converge to.

  We can correct this bias by using importance-sampling (IS) weights $w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)} \right)^{\beta}$

  that fully compensates for the non-uniform probabilities $P(i)$ if $\beta = 1$.
- Q-learning update uses $w_i \delta_i$ instead of δ_i (*weighted* IS, not ordinary IS).
  For stability reasons, the weights are normalized by $1/\max_i w_i$ (only scale downwards).
- To ensure the unbiasedness of updates at the end of training,
  we define a schedule on β that reaches 1 only at the end of learning (*e.g.* linear scheduler).

Importance sampling

If samples $\{x_n\}$ are drawn from $p_X(\cdot)$, then the true mean is estimated by $\mathbb{E}_p[f(X)] \approx \sum_n f(x_n)$.
Instead of sampling from $p_X(\cdot)$, if they are sampled from another distribution $q_X(\cdot)$, we can estimate $\mathbb{E}_p[f(X)]$ by

$$\mathbb{E}_p[f(X)] = \int_X f(x)\, p_X(x) dx = \int_X f(x)\, \frac{p_X(x)}{q_X(x)} q_X(x)\, dx \approx \sum_n \frac{p_X(x_n)}{q_X(x_n)} f(x_n) = \sum_n w_n f(x_n) \quad \text{where } w_n = \frac{p_X(x_n)}{q_X(x_n)}.$$

Double DQN with proportional prioritization

Hyperparameters: minibatch B, replay period K and size N, exponents α and β
Initialize replay buffer $\mathcal{R}$ to capacity N, $\Delta = 0$, $p_1 = 1$
Observe state s_1
for $t = 1, T$ **do**
    Select action $a_t = \arg\max_a Q(s_t, a; \theta)$ ϵ-greedy CNN θ
    Execute a_t in emulator and observe reward r_{t+1} and state s_{t+1}
    Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in $\mathcal{R}$ with maximal priority $p_t = \max_{i<t} p_i$
    if $t \equiv 0 \bmod K$ **then**
        for $i = 1, B$ **do**
            Sample transition $i \sim P(i) = p_i^\alpha / \sum_k p_k^\alpha$ prioritized replay + stochastic sampling
            Compute $w_i = (N \cdot P(i))^{-\beta} / \max_k w_k$ importance sampling
            Compute TD error $\delta_i = r_{i+1} + \gamma\, \hat{Q}(s_{i+1}, \arg\max_a Q(s_{i+1}, a; \theta); \hat{\theta}) - Q(s_i, a_i; \theta)$ CNN $\hat{\theta}$
            Update transition priority $p_i \leftarrow |\delta_i|$ Double DQN
            Accumulate weight-change $\Delta \leftarrow \Delta + w_i\, \delta_i\, \nabla_\theta Q(s_i, a_i; \theta)$
        end
        Update behavior network weights $\theta \leftarrow \theta + \eta\, \Delta$ and reset $\Delta = 0$
        Every C steps, update target network weights $\hat{\theta} \leftarrow \theta$
    end
end

Extensions

Prioritized Supervised Learning

- Analogous to PER, we can sample each data using a priority based on its last-seen error.
- This can help focus the learning on those samples that can still be learned from.

Off-policy Replay

- Apply the replay probability and the IS-correction to importance sampling or rejection sampling.

Feedback for Exploration

- M_i: the total number that the ith transition will end up being replayed.
  - This varies widely, and this gives a rough indication of how useful it was to the agent.
- We can monitor usefulness of the experience via M_i and update the distribution toward generating more useful experience (feedback to exploration strategy).

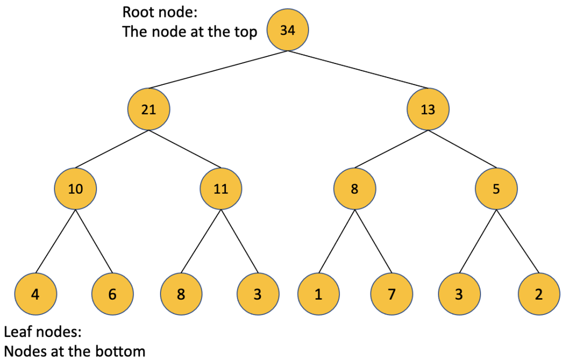
Prioritized Memories

- Priority can help determine which transitions to store and when to erase them.

How to implement Prioritized Experience Replay?

Proportional Prioritization

- An efficient implementation can be made using a ‘sum-tree’ data structure.
- The value written on a leaf node is the frequency that each leaf node is randomly sampled at.

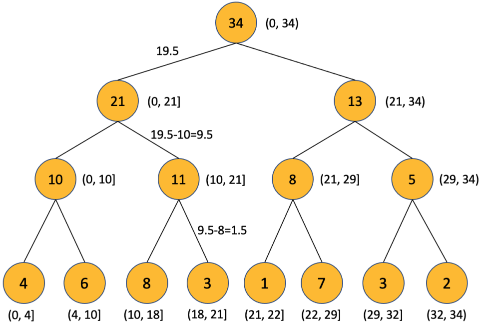


A binary tree is a tree data structure where each parent node has a maximum of 2 child nodes.

A sum tree is a binary tree where every node is the sum of its children.

PER can be implemented as a sum tree with the priorities in the leaf nodes.

How to sample a leaf node in Sum Tree



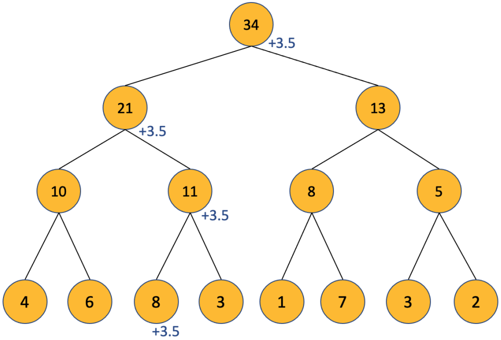
V = the value of the root node
Sample a number x from $\mathit{Unif}(0, V)$.

repeat until we reach a leaf node.
  v_1, v_2 = the value of left and right child nodes, resp.
  if $x < v_1$
    choose the left child node.
  else choose the right child node
    $x = x - v_1$
end
return the chosen node

- We split the interval $(0, V)$ into the sub-intervals of the length of the values in the leaf nodes. Sampling a leaf node = finding a sub-interval that contains the sampled number x from $\mathit{Unif}(0, V)$.
- To sample a minibatch of size B, the range $(0, V)$ is divided equally into B ranges. A value is uniformly sampled from each range to sample a minibatch.

How to Update the Value of a leaf node in Sum Tree

- Using a sum tree structure, the value of a leaf node can be updated by the following algorithm.
- Notice that the algorithm has a complexity of $O(\log_2 N)$, instead of $O(N)$.



s, v : a leaf node and its value.
Δ : the change of value of a leaf node s.

$v = v + \Delta$

repeat until we reach the root node.

    if there is a parent node s' of s,
        $v' = v' + \Delta$ (v': the value of s')
        $s = s'$

end