Classic control in reinforcement learning

Yu-Tong Shen

June 12, 2018

Algorithm

- Double Deep Q Network With Prioritized Experience Replay
  - **Input:** minibatch k, step-size η, replay period K and size N, exponents α and β, budget
  - Initialize replay memory H = Φ, Δ = 0, $p_1 = 1$
  - Observe S_0 and choose $A_0 \sim \pi_\theta(S_0)$
  - **for** t = 1 **to** T **do**
    - Observe S_t, R_t, γ_t
    - Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in H with maximal priority $p_t = \max_{i<t} p_i$
    - **if** t ≡ 0 mod K **then**
      - **for** j = 1 **to** k **do**
        - Sample transition $j \sim P(j) = \frac{p_j^\alpha}{\sum_i p_i^\alpha}$
        - Compute importance-sampling weight $w_j = \frac{(N \cdot P(j))^{-\beta}}{\max_i w_i}$
        - Compute TD-error $\delta_j = R_j + \gamma_j Q_{target}\left(S_j, argmax_a Q(S_j, a)\right) - Q(S_{j-1}, A_{j-1})$
        - Update transition priority $p_j \leftarrow |\delta_j|$
        - Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
      - **end for**
      - Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset Δ = 0
      - From time to time copy weights into target network $\theta_{target} \leftarrow \theta$
    - **end if**
    - Choose action $A_t \sim \pi_\theta(S_t)$
  - **end for**

Setting Reward

- According to OpenAI gym, we have:
    - **Observation**

Num	Observation	Min	Max
0	position	-1.2	0.6
1	velocity	-0.07	0.07

- So we update agent with $\mathrm{R} \leftarrow \frac{|p+0.52|}{1.12} + \frac{|v|}{0.07} - 1.0$
    - The range of reward will be -1 to +1
    - it makes car moving with higher velocity and closer edge

Architecture Analysis

- Prioritized experience replay have 2 issues
    - **lead to a loss of diversity**:
        - Using stochastic prioritization, it makes all transition will be sampled with some probability. Therefor, this problem will alleviate.
    - **Introduce bias**:
        - This algorthm using importance sampling weight to prevent that some high priority transitions become the main of updated transitions.
- In `src/rl.py`, we have 3 classes include `SumTree`, `Memory` and `RL`
    - **SumTree**
        - We store transition and priority in this class.
        - It assist non-uniform sampling
    - **Memory**
        - This is replay memory, there is sum-tree in this class.
        - `Memory` should calculate priority for `SumTree` and importance-sampling weight for `RL`, and maintain priorities of each transitions.

- **RL**
  - This is main part of the agent.
  - Choocing action via `actor()`. Using ε-greedy method, that is, the agent randomly chooce action with ε probability. As times of updating increass ε will become smaller.
  - Learning via `learn()`. The agent update self by the transitions that sampling in replay memory, and need product with importance-sampling weight when computing loss.

Configuration

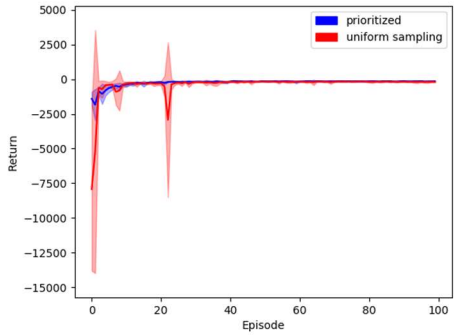
- Epsilon: **0.5 to 0.1**. This value decides greedily or randomly chooce actions. We set 0.5 at the begin, it makes the agent will explore environment. As time passes, the agent become smarter. So we don't need explore with high prabobilty.
- Replay memory size: **10,000**
- Parameters of target network update: In the DQN agent, We have 2 neural networks, target network and predict network. In this fomula $\delta_j = R_j + \gamma_j Q_{target}\left(S_j, argmax_a Q(S_j, a)\right) - Q(S_{j-1}, A_{j-1})$, target network is Q_{target} and predict network is Q. Then we use **hard update** and **the period is 500**. Means target network will update when predict network training 500 times.

Performance

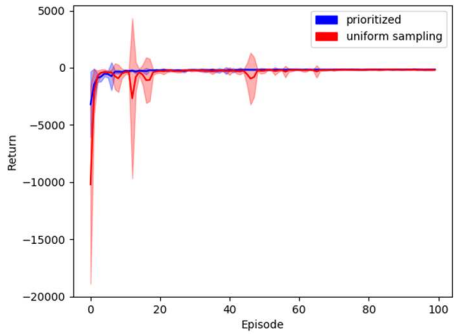
- note 1: The value of y-axis is original reward in episode, it is not used to update agent.
  - **Original reward**: -1 for each time step, until the goal position of 0.5 is reached.
- note 2: No matter which red or blue line are using double deep Q network.
- note 3: Both its replay memory size are 10,000
  - In this experiment, replay memory contain same 10,000 transitions that using randomly chooce action at the begining.

Changing the batch size

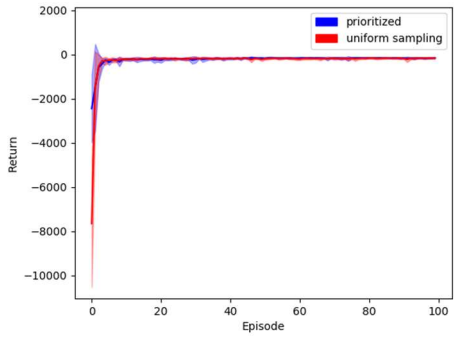
- **batch size: 16**, learning rate: 0.0005



- **batch size: 32**, learning rate: 0.0005



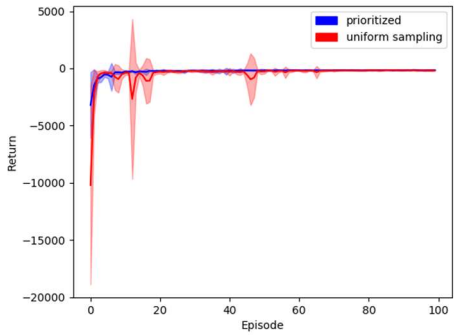
- **batch size: 64**, learning rate: 0.0005



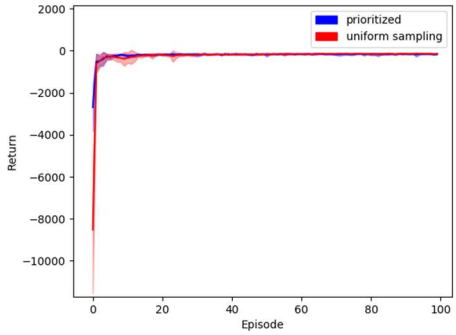
- As the batch size increases, double DQN without prioritized experience replay will be more stable (with lower standard deviation).
- However, double DQN with prioritized experience replay remains stable throughout.

Changing the learning rate

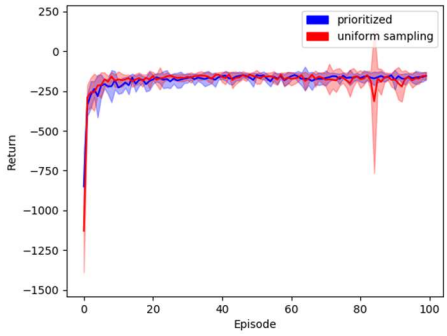
- batch size: 32, **learning rate: 0.0005**



- batch size: 32, **learning rate: 0.001**

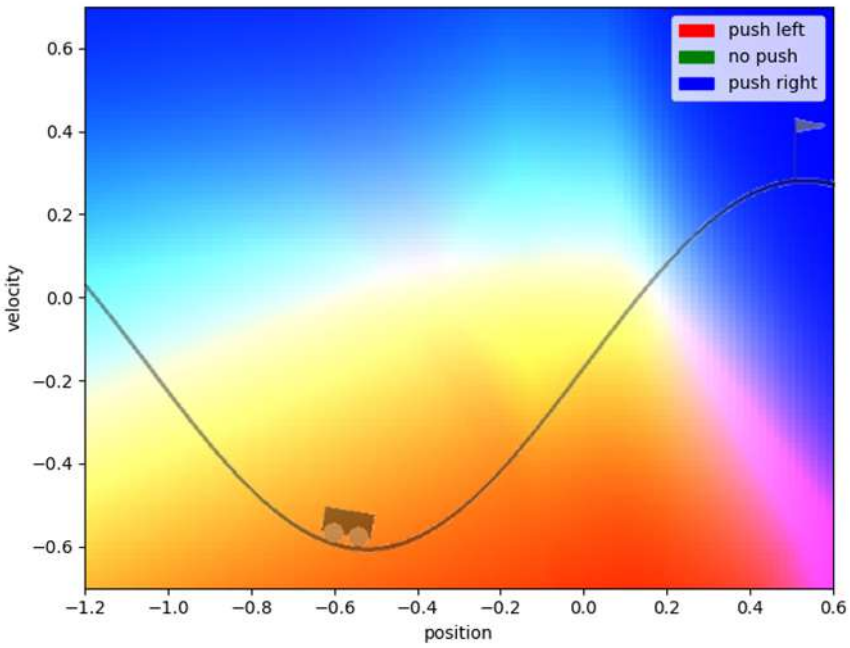


- batch size: 32, **learning rate: 0.01**



- Although the uniform sampling agent will get higher standard deviation sometime, overall, As the learning rate increases, both its average reward are higher.

Action-value diagram (prioritized replay)



- This diagram tell us the agent had learned that it want to push right if speed is positive, conversely, it will want to push left. But the blue (push right) area is larger than red (push left) at right hand side, because the agent knows the goal is not far.

Summary

- **When we change the batch size**
  - As the batch size increases, double DQN without prioritized experience replay will be more stable (with lower standard deviation).
  - However, double DQN with prioritized experience replay remains stable throughout.
- **When we change the learning rate**
  - Although the uniform sampling agent will get higher standard deviation sometime, overall, As the learning rate increases, both its average reward are higher.

- **Prioritized experience replay is better than uniform sampling**
    - **More robust**:
        - In a prioritized experience replay agent, all new transitions arrive without a known TD-error, so we put them at maximal priority in order to guarantee that all experience is seen at least once. But uniform sampling agent is not so lucky, maybe some important transitions leave replay memory without update.Therefor, its performance will get higher standard deviation sometime.
    - **Has better score at the beginning**:
        - In the replay memory, there are almost redundant transitions. Using prioritized experience replay method makes rare and task-relevant transitions are sampled more easily. But uniform spamling method usually sample redundand transitions, lead uniform spamling method will be slower growth. So we can say prioritized experience replay method will adapt to the environment earlier than uniform sampling.

Conclusion

- **DQN belong to value-base**:
    - The DQN agent always choose action via its action-value function (neural network). the action-value function (neural network) will tell which action is best, when it want to choose action. In addition, This agent use ε-greedy method, so that it will explore the environment (randomly choose action) with ε probability.
- **This algorithms is off-policy**:
    - An on-policy agent update self based on its current action derived from the current policy, whereas its off-policy counterpart update self based on the action obtained from another policy. In this algorithm, the agent sample transitions in the replay memory, but the replay memory contains a lot of different policies and those are almost not the current policy. So this algorithms is off-policy.
- **The replay memory method break the temporal correlations**:
    - In a deep Q network agent, it often randomly sample transition in replay memory. It means old and new transitions will be mixed, so that the temporal correlations will be broken. In addition, The replay memory method also makes rare experience will be used for more then just signle update.