Recent Emphasized Prioritized Experience Replay based on Deep Q-Network

Haoxin Liu[*] **Zhuo Chen**[†]

ShanghaiTech University
School of Information Science and Technology
`{liuhx, chenzhuo}@shanghaitech.edu.cn`

Abstract

Deep Q-Network proposed in 2013 is the first model which combines neural network and Q-learning in reinforcement learning, and it also maintains an experience replay buffer to utilize samples before with equal probability. We present a new model based on Prioritized Experience Replay proposed in 2016, which gave priority to each sample and drawn them using different probabilities. In our model called Recent Emphasized Prioritized Experience Replay(REPER), We take into account the time sequence of sample entry and dynamically assign higher priorities to recent samples, which takes the effect of emphasizing recent samples. We also apply our model to two kinds of Atari game (*Pong*, *Cart Pole*) and compare the results with several former work.

1 Introduction

Applying reinforcement learning (RL) to high-dimensional state space like game interface has been a challenge in artificial intelligent. One major problem to overcome is the time-consuming and ineffective training process on tasks with huge state spaces. To improve the effectiveness of the training process based on reinforcement learning and neural network (Deep Q Network, a combination of neural network and reinforcement learning that the neural network is to approximate the Q function in reinforcement learning), we propose a model named Recent Emphasized Prioritized Experience Replay (REPER), which put importance both on its donation to gradient descent and degree of recentness during the training process. What recentness means that, intuitively, the actions we take in the later period are always more intelligent than that we take previously.

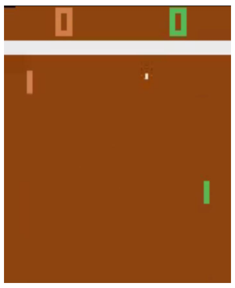


Figure 1: Pong Game

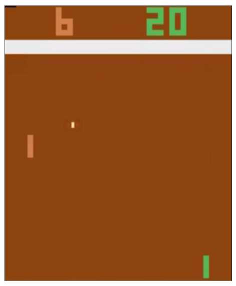


Figure 2: Green is winning

[*]make contribution equally.
[†]make contribution equally.

The task we focus on is a game called Pong, (Figure 1), the goal is to try your best to bounce the ball back to your opponent all the time until one of the players reach 21 points (Figure 2). In addition, to visualize the training process we also run our model on a simpler game, Cart Pole, in which we need to keep a pole from falling by moving its base. Our model is an extension from package "gym" in which each state is represented by an 84*84*4 tensor and we implement a "double Q network" to help us approximate the Q function.
Training and testing the network will take each frame as an input and output 14 numbers seen as a score respectively. All frames are also labeled with an action and a reward that specifies what the agent does and how it does under certain circumstance. Where all these experiences stored is called replay buffer. By sampling sufficient times from replay buffer, we can make our neural network closer to the real approximation of Q function of the game. Then the argmax of 14 scores will be the action we are taking. Our work focuses on the optimization on the sampling part. Specifically, how we sample from the replay buffer is a combination of its donation to gradient descent and its significance on the degree of recentness. Besides we also make the trade-off between these 2 factors flexible by a balancing hyper-parameter.

2 Background and Related Work

Deep reinforcement learning, in which neural network is used to model reinforcement learning([8]), has been a hot topic for several years and made many remarkable achievements such as *AlphaGo*, which defeated one of the most powerful human *go* players in the world and was the result of deep reinforcement learning. In this area, there have been many significant models such as DQN([4]), Prioritized Experience Replay([7]), Nature DQN([5]), etc. In this part, We will briefly introduce reinforcement learning and several work of predecessors that we mainly focus on, and compare our model with existing works.

Reinforcement learning. Reinforcement learning is the training of machine learning models to make a sequence of decisions. Usually, we focus on the interaction between an agent and an uncertain, potentially complex environment, in which the agent needs to learn to achieve a goal by exploring and exploiting its experience. The agent will update its state and choose the best action according to the **Bellman equation**:

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} [R^{a}_{ss'} + \gamma V^{\pi}(s^{'})]$$

$$Q^{\pi}(s,a) = \sum_{s'} P^{a}_{ss'} [R^{a}_{ss'} + \gamma \sum_{a'} Q^{\pi}(s^{'}, a^{'})]$$

Deep Q-Network. It is composed by a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. One significant improvement is that it designed a memory buffer to store the agent's experiences and used them later. The drawback, however, is that it treated all samples as equally weighted and drawn them with equal probability. We try to solve this problem by assigning quantified weights to different priors in our model.

Prioritized Experience Replay. This work is similar to our idea of improving the way samples are stored in the buffer by giving them quantified weights. Comparing with this model, we also take into account the sequence in which the samples are added to buffer, and assign higher weight to recent samples because recent experiences are often more important.

3 Our model: REPER on DQN

How to dynamically assign priorities to samples in the buffer while considering their the degree of recentness is the core problem to be solved. To tackle this issue, we split our work into two parts: 3.1. Use TD-Error to quantify the priority of the samples. 3.2. The time stamp of a sample is quantified, and the quantified result is introduced into the priority calculation, so as to give higher weight to the recent samples. Based on these, we propose our model named as **Recent Prioritized Experience Replay based on Deep Q-Network**, and it can also be called *REPER on DQN*.

Network Architecture

Before explaining the calculation of priorities, we also need to briefly describe our network architecture referred to the Nature-DQN proposed in [5]. Our network is composed of two sub-networks: predict Q-net and target Q-net. As you can see, a new network (called target-net) is introduced to generate the target-Q value, which keeps the target-Q value unchanged for a period of time, reducing the correlation between the current Q value and the target Q value to some extent, and improving the stability of the algorithm.

3.1 Basic criteria for priority calculation

The central component of prioritized replay is the criterion by which the importance of each transition is measured. In our model, we calculate the loss of each transition and use this value to indicate the priority of this transition sample. The loss function is:

$$L(w) = \frac{1}{m}\sum_{j=1}^{m} w_j (y_j - Q(\phi(S_j), A_j, w))^2 \qquad y_j = r_{t+1} + \lambda \max_{a^{'}} Q(s_{t+1}, a^{'}, w^{'})$$

Here $r_{t+1} + \lambda \max_{a'} Q(s_{t+1}, a^{'}, w^{'})$ is the target Q-value calculated in target Q-net and $Q(s, a, w)$ is the predict Q-value calculated in predict Q-net, so the loss is actually quantification of the difference between two network output values. And w_j is the priority weight of the j-th sample and it is computed by TD error $|\delta(t)|$ normalization:

$$w_j = \frac{(N * P(j))^{-\beta}}{\max_i(w_i)} = \frac{(N * P(j))^{-\beta}}{\max_i(N * P(j))^{-\beta}} = (\frac{p_j}{\min_i P(i)})^{-\beta}$$

Here $P(j)$ is the probability of each transition being sampled and they obey the uniform distribution at the beginning. After all parameters w of the Q network are updated by back propagation of the neural network in each step, we need to re-calculate the *TD-error* of all samples by $\delta_j = y_j - Q(\phi(S_j), A_j, w)$, and then update all samples' priority by $p_j = |\delta_j|$.

3.2 Quantification of degree of recentness

This part tells how we introduce degree of recentness into our training. In addition to represent each transition by their state, reward, next state and action, we also remember the time this transition shows up by an integer. To round this time number to a properly ranged logit, we design a function D to do the transformation:

$$\textit{degree of recentness:} D(x) = \frac{e^{log_2 x}}{1 + e^{log_2 x}}$$

The reason of applying this function is to prevent the recentness factor from dominating the priority because even though the recent experience is more useful, some of the former experience should not be abandoned.
Also, we set a hyper-parameter *balancing* b which balances the weight of priority based on TD error and the degree of recentness.

$$P(j) = (1 - b)P(j) + bD(j)$$

4 Experiment and Results

We have tested our model on two games. One is Pong[3] and the other is Cart Pole[4]. Compared Cart Pole with Pong, Cart Pole is a relatively simpler game in which you do not even have an opponent, thus the network structure we used for Cart Pole is also simpler than that of Pong, too. Testing our model on Cart Pole has two main reasons for which one is to test the stability and robustness of our model and the other is that to visualize the result easily (loss and reward curve, 4, 5, 6).
From the curve, it can be easily inferred that when we are doing recent emphasized prioritized

[3]`https://github.com/Hanson-Liuhx/REPER-on-DQN`

[4]`http://10.15.89.41:30303/notebooks/REPER-on-DQN/REPER-Pri-Random.ipynb` password: 123

sampling based on our REPER model, the model learns the game faster than all the other 2 methods brought by previous researchers. Specifically, REPER take about **180** rounds to master the game and simple prioritized model takes **280** rounds. For a random sampler, it has not mastered the game even after 300 rounds training. As for the reward shown in the figure, **136.6** for random sampling, **191.0** for simple prioritized sampling and **193.6** for our model, please note there may not be big difference between REPER and simple prioritized model, but we reach that level about 100 rounds earlier. Also note that the loss curve that the **absolute magnitude** of loss does not mean much here as we cannot evaluate the loss by the exactly same means in all three models. However, the fluctuation when we are doing random sampling tells that the model cannot approximate the Q function as well as REPER does.
There are 2 .gif files in our submission fold where you can watch the whole game of which Figure 3 are two screen shots. The orange player is the opponent of our agent (green player). Note that if we actually give them enough time to train both of them will master the game, however, if we control the resources they are available for when are trained, REPER-agent apparently does better. As a reminder, if the readers want to re-run the code to generate your own video, we would recommend using a machine with **multiple CUDA's** because each of them might take 4 hours to train.

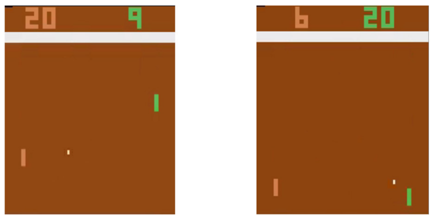


Figure 3: Random Sampling fails, REPER wins

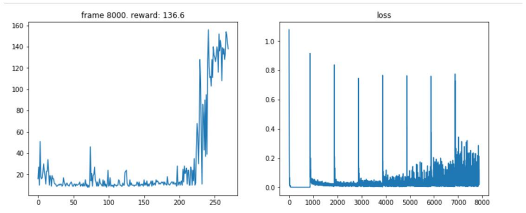


Figure 4: Random Sampling

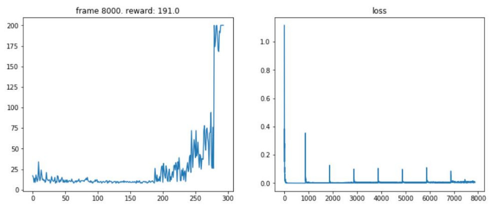


Figure 5: Prioritized Sampling

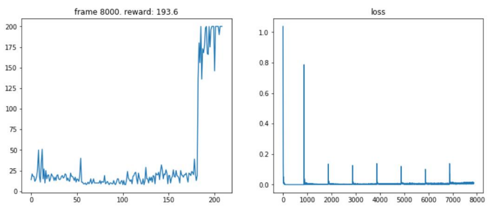


Figure 6: REPER

5 Conclusion and Future work

Our model maintains an experience replay buffer with different priorities for each sample based on DQN. Besides, we also include the degree of recentness of samples into the priority calculation, because recent samples often play a more important role in one agent's current decision-making. Generally speaking, our model shows a better performance in two Atari games comparing to plain DQN and prioritized DQN.
Q-learning is a kind of *value based* method, and it of course has some drawbacks such as low efficiency in large state space, so we will try to combine some other methods in reinforcement learning, such as *policy gradient* and *actor-critic*, to make more progress in deep reinforcement learning. By the way, introduce deep reinforcement learning into multi-agent system is also an interesting topic that we will continue to focus on.

References

[1] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Matteo Hessel, Ian Osband, Alex Graves, Volodymyr Mnih, Rémi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration. In *6th International Conference on Learning Representations (ICLR)*, 2018.

[2] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[3] A Rupam Mahmood, Hado P van Hasselt, and Richard S Sutton. Weighted importance sampling for off-policy learning with linear function approximation. In *Advances in Neural Information Processing Systems*, pages 3014–3022, 2014.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

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

[7] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2016.

[8] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[9] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[10] John N Tsitsiklis and Benjamin Van Roy. Analysis of temporal-diffference learning with function approximation. In *Advances in neural information processing systems*, pages 1075–1081, 1997.

[11] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[12] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning (ICML)*, pages 1995–2003, 2016.

[13] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. In *Machine Learning*, pages 279–292, 1992.

Appendix

The content in the appendix is only a supplement to the main text, you can choose whether to read it or not (not looking at it will not affect your understanding of the main text).

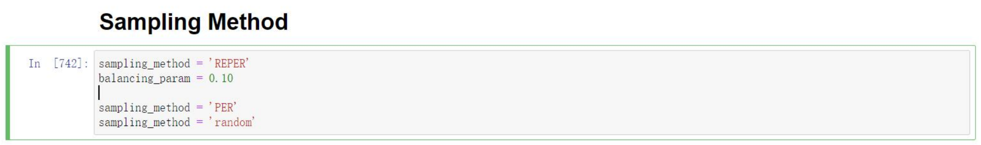
Pseudocode

Algorithm 1 REPER on DQN

```
Input: minibatch $k$, step-size $\eta$, replay period $K$ and size $N$, exponents $\alpha$ and $\beta$, budget $T$, $b$.
 1: Initialize replay memory $\mathcal{H} = \phi, \Delta = 0, p_1 = 1$
 2: Observe $S_0$ and choose $A_0 \sim \pi_\theta(S_0)$
 3: for $t$=1 to $T$ do
 4:     Observe $S_t, R_t, \gamma_t$
 5:     Compute degree of recentness $D_t = \frac{e^{\log_2 t}}{1+e^{\log_2 t}}$
 6:     Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with priority $p_t = (1-b) \cdot \max_{i<t} p_i + b \cdot D_t$
 7:     if $t = 0 \bmod K$ then
 8:         for $j$=1 to $k$ do
 9:             Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
10:             Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
11:             Compute TD-error $\delta_j = R_j + \gamma_j Q_{target}(S_j, argmax_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
12:             Update transition priority $p_j \leftarrow |\delta_j|$
13:             Accumulate weight-change $\Delta \leftarrow \Delta + w_i \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
14:         end for
15:         Update weights $\theta \leftarrow \theta + \eta$, reset $\Delta$=0
16:         From time to time copy weights into target network $\theta_{target} \leftarrow \theta$
17:     end if
18:     Choose action $A_t \sim \pi_\theta(S_t)$
19: end for
```

Operation manual for our code

1. To see a quick result curve on Jupyter:
   First, log into `http://10.15.89.41:30303/notebooks/REPER-on-DQN/REPER-Pri-Random.ipynb` via password 123. (If you fail to connect to the server, please contact chenzhuo@shanghaitech.edu.cn and we will open the port for Jupyter, because we cannot guarantee that the server is always in service.)
   Second, to compare the results with random sampling, prioritized sampling and our REPER model you might want to adjust *sampling_method* to be one of the three choices.
   Then, click "Cell" -> "Run All", and the curve will show up at the end.



Sampling Method

```
In [742]: sampling_method = 'REPER'
          balancing_param = 0.10

          sampling_method = 'PER'
          sampling_method = 'random'
```

Figure 7: Light weight training process

2. To train 2 players playing Pong Game:
   First, enter the folder REPER-on-DQN/framework.
   Then run

   $ python main.py –sample_method=random

   to train an agent based on random sampling run

```
$ python main.py –sample_method=PER
```

to train an agent based on simple prioritized sampling
run

```
$ python main.py –sample_method=REPER
```

to train an agent based on REPER model
Note that the training process may take about 4 hours on a machine with CUDA.

3. Prerequisites:

Name	Version	Build	Channel
gym	0.17.2	pypi_0	pypi
gym-wrappers	0.1.0	pypi_0	pypi
pytorch	1.4.0	py3.6_cuda9.2.148_cudnn7.6.3_0	pytorch
torchvision	0.5.0	py36_cu92	pytorch
numpy	1.15.4	pypi_0	pypi
opencv-python	4.2.0.34	pypi_0	pypi

Table 1: Prerequisites