
MLDS 2018 Spring

HW4-2 - Deep Q Learning

2018/6/8
ntu.mldsta@gmail.com

Notification

- If you want to present in class, please start your hw4
ASAP

Time Schedule

- June 1st 4-1 announce
 - Policy Gradient
- June 8th 4-2 announce
 - Deep Q learning
- June 15th 4-3 announce
 - Actor-Critic
- July 6th 23:59 Deadline (all in one)

Outline

- Introduction
 - Game Playing : Breakout
- Deep Reinforcement Learning
 - Deep Q-Learning (DQN)
 - Improvements to DQN
- Grading & Format
 - Grading Policy
 - Code Format
 - Report
 - Submission

Introduction

Environment

Breakout




Deep Reinforcement Learning

Deep Q-Learning (DQN)

“classic” deep Q-learning algorithm:

Replay buffer

- 
1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to \mathcal{B}
 2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from \mathcal{B} uniformly
 3. compute $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ using *target* network $Q_{\phi'}$
 4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
 5. update ϕ' : copy ϕ every N steps

Deep Reinforcement Learning

Deep Q-Learning (DQN)

- The action should act ϵ -greedily
 - Random action with probability ϵ
 - Also in testing
- **Linearly decline ϵ** from 1.0 to some small value, say 0.025
 - Decline per step
 - Randomness is for exploration, agent is weak at start
- Hyperparameters
 - Replay Memory Size 10000
 - Perform Update Current Network Step 4
 - Perform Update Target Network Step 1000
 - Learning Rate $1.5e-4$
 - Batch Size 32

Deep Reinforcement Learning

Improvements to DQN

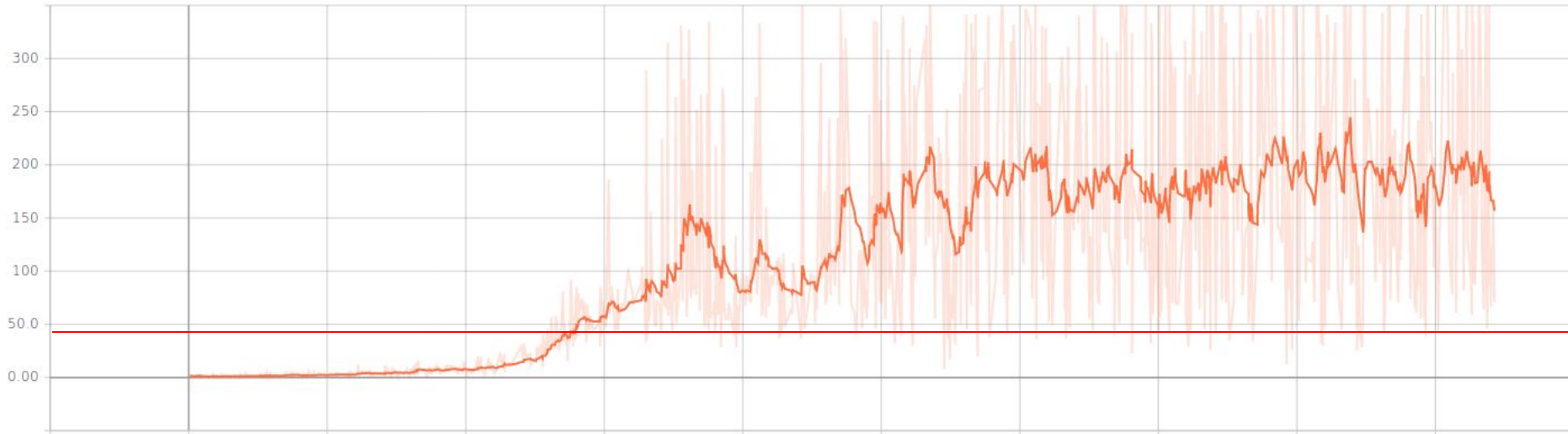
- Double Q-Learning
- Dueling Network
- Prioritized Replay Memory
- Noisy DQN
- Distributional DQN

<https://arxiv.org/pdf/1710.02298.pdf>

Training Tips

Training Plot

summaries/Episode_Reward



- X-axis : 1000 episodes/unit
- Y-axis : **Unclipped** reward per episode
- Baseline is achieved within an hour of training

Training Tips

Why Reward is clipped

- Performing the same action for 4 frames
 - To use data more efficiently
- Reward may be up to 4
 - If positive, clip to 1 → reduce variance
- How to see your unclipped reward
 1. Use the *test* function
 2. Turn off the *clip_reward* option of your environment and do the clipping by yourself.

Training Tips

Asynchronous Update (Optional, for Tensorflow)

- In tensorflow, *feed_dict* does the copy thing
 - Upon updating, the agent have to wait for it to continue exploring.
- Try run the update asynchronously
 - Main thread : Collect data
 - The other thread : Copy data to GPU
 - GPU : Training
 - Using the thread/multiprocessing module
- This is totally **not necessary for you to get baseline**, just some speed-up you can try.
 - This can go wrong and annoying if you're not familiar with threading, thus I recommend not to try it unless you are confident enough.

Grading Policy

- Code Baseline (5%)
- Report (5%)

Grading & Format

Baseline (5%)

- DQN (5%)
 - Getting averaging reward in 100 episodes over **40** in **Breakout**
 - With **OpenAI's Atari wrapper** & reward clipping
 - We will unclip the reward when testing

Grading & Format

Code Format

- Please download the sample files from [github](#)
- Follow the instructions in README to install required packages
- **Four** functions you should implement in [agent_\[pg|dqn\].py](#)
 1. `__init__(self, env, args)`
 2. `init_game_setting(self)`
 3. `train(self)`
 4. `make_action(self, state, test)`
- **DO NOT** add any parameter in `__init__()`, `init_game_setting()` and `make_action()`
- You can add new methods in the [agent_\[pg|dqn\].py](#)
- You can add your arguments in [argument.py](#)

Grading & Format

Report (10%)

- Up to 6 pages (4-1 + 4-2 + 4-3), in Chinese
- Describe your DQN model (1%)
- Plot the learning curve to show the performance of your Deep Q Learning on Breakout (1%)
 - X-axis: number of time steps
 - Y-axis: average reward in last 30 episodes
- Implement 1 improvement method on page 6
 - Describe your tips for improvement (1%)
 - Learning curve (1%)
 - Compare to origin Deep Q Learning(1%)

Grading & Format

Late submission

- Please fill the late submission form first **only if you will submit HW late**
- Please push your code before you fill the form
- **There will be 25% penalty per day for late submission,** so you get 0% after four days
- You get 0% if the required files has bug.
 - If the error is due to the format issue, please come to fix the bug at the announced time, or you will get 10% penalty afterwards.

Grading & Format

Submission

- Deadline: **2018/7/6 23:59 (GMT+8)**
- Your github **MUST** have 5 files under directory hw4/
 - agent_dir/agent_pg.py
 - agent_dir/agent_dqn.py
 - [saved_model_file] * 2
 - report.pdf
 - argument.py (optional)
 - README (optional)
 - download.sh (optional)
 - other files you need
- If your model is too large for github, upload it to a cloud space and write download.sh to download the model
- Do not upload any file named the same with other sample codes

Grading & Format

Grading

- Please use Python with version ≥ 3.5
- The TAs will execute `'python3 test.py --test_pg --test_dqn'` to run your code on **ubuntu**
- The execution for both model should be done within 10 minutes respectively, excluding model download
- Allowed packages
 - PyTorch v**0.3.0**
 - Tensorflow r**1.6** (CUDA 9.0)
 - Numpy
 - Scipy
 - Pandas
 - Python Standard Lib
- **No keras !!!! No keras !!!! No keras !!!! No keras !!!! No keras !!!!**
- **If you use other packages, please ask for permission first**

Related Materials

- Course & Tutorial:
 - [Berkeley Deep Reinforcement Learning, Fall 2017](#)
 - [David Silver RL course](#)
 - [Nips 2016 RL tutorial](#)
- Blog:
 - [Andrej Karpathy's blog](#)
 - [Arthur Juliani's Blog](#)
- Text Book:
 - [Reinforcement Learning: An Introduction](#)
- Repo:
 - <https://github.com/williamFalcon/DeepRLHacks>

Deep Reinforcement Learning

Double DQN

- The formula $Y_t^{\text{DQN}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t^-)$. (3)
often overestimates the maximum Q value.
- Thus instead choose the action of the max Q in the target network, choose the action of the max Q in the **current network**.
- $Y_t^{\text{DoubleDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t), \theta_t^-)$.

Deep Reinforcement Learning

Dueling Network

- In many state, action does not counts.
 - DQN tries to find out the max Q in each state
- Use same network to output *Value* and *Advantage*

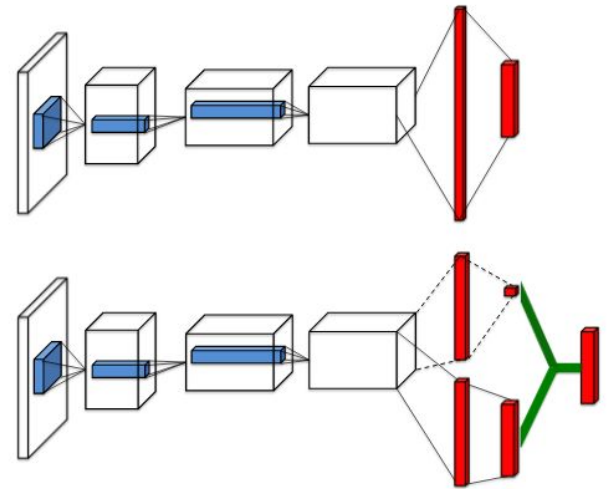
- Why should it be the *Advantage*?

- Add loss constraint

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \max_{a' \in |\mathcal{A}|} A(s, a'; \theta, \alpha) \right)$$

- Alternative Q function, more stable (more used)

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a'; \theta, \alpha) \right)$$



Deep Reinforcement Learning

Prioritized Replay Memory

- DQN : Sample from replay memory uniformly
- We can sample the replays with large loss more often
- Thus we sample with the probability
 - $TD\ ERROR = R_j + \gamma_j Q_{\text{target}}(S_j, \arg \max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
 - $p_t \propto |TD\ ERROR|^\omega$
 - ω is a hyperparameter, 0.5 in Rainbow

$$p_t \propto \left| R_{t+1} + \gamma_{t+1} \max_{a'} q_{\bar{\theta}}(S_{t+1}, a') - q_{\theta}(S_t, A_t) \right|^\omega$$

<https://arxiv.org/pdf/1511.05952.pdf>

Deep Reinforcement Learning

Prioritized Replay Memory

- However, the resulting gradient estimator is biased, since we are sampling from a different distribution
 - Correct by importance sampling weights
- With $\rho_i = 1 / P(i)$, the IS weights

$$\tilde{v}_g \doteq \frac{\sum_{k=1}^n \rho_k Y_k}{n}.$$

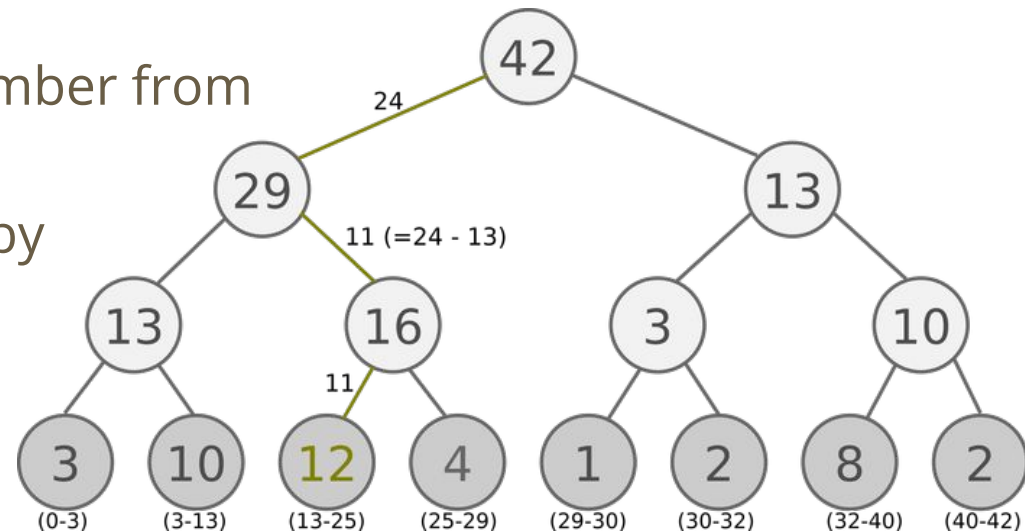
$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)} \right)^\beta$$

- β is linearly declined to 1
 - $\beta = 1 \rightarrow$ Unbiased
 - Try to learn quicker \rightarrow Try to converge correctly

Deep Reinforcement Learning

Prioritized Replay Memory

- Using array, the complexity of sampling is $O(n)$
 - Try another data structure
- Sum Tree, which prioritized sampling can be $O(\lg n)$
 - Devide the priorities into k groups(batch size) by the max priority
 - That is if the max is 42, batch size = 6, we devide them into [1, 7], [8, 14],, [36, 42]
 - Randomly sample a number from each interval
 - Go down the sum tree by the priority to retrieve the data at the leaf



Deep Reinforcement Learning

Prioritized Replay Memory

Algorithm 1 Double DQN with proportional prioritization

```
1: Input: minibatch  $k$ , step-size  $\eta$ , replay period  $K$  and size  $N$ , exponents  $\alpha$  and  $\beta$ , budget  $T$ .
2: Initialize replay memory  $\mathcal{H} = \emptyset$ ,  $\Delta = 0$ ,  $p_1 = 1$ 
3: Observe  $S_0$  and choose  $A_0 \sim \pi_\theta(S_0)$ 
4: for  $t = 1$  to  $T$  do
5:   Observe  $S_t, R_t, \gamma_t$ 
6:   Store transition  $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$  in  $\mathcal{H}$  with maximal priority  $p_t = \max_{i < t} p_i$ 
7:   if  $t \equiv 0 \pmod K$  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  $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg \max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$ 
12:      Update transition priority  $p_j \leftarrow |\delta_j|$ 
13:      Accumulate weight-change  $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$ 
14:    end for
15:    Update weights  $\theta \leftarrow \theta + \eta \cdot \Delta$ , reset  $\Delta = 0$ 
16:    From time to time copy weights into target network  $\theta_{\text{target}} \leftarrow \theta$ 
17:  end if
18:  Choose action  $A_t \sim \pi_\theta(S_t)$ 
19: end for
```
