MLDS 2019 Spring HW4-2 - Deep Q Learning

2019/06/01

adlxmlds@gmail.com

Time Schedule

- June 11th 19:30 Deadline
- June 15th 10:30 Deadline

Outline

	HW4-1	HW4-2	HW4-3
Pong	PG		AC
Breakout		DQN	AC
Improved Version			

Outline

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

Environment

Breakout

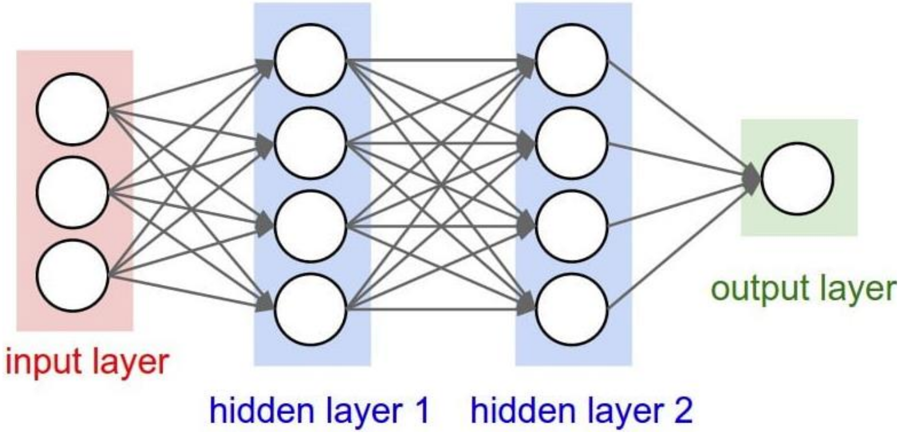
Deep Q-Learning (DQN)

	a1	a2
s1	-3	1
s2	-1	3
s3	3	3
s4	2	-2
...		

https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/

Deep Q-Learning (DQN)

	a1	a2
s1	-3	1
s2	-1	3
s3	3	37
s4	2	-20
...		



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 Q-Learning (DQN)

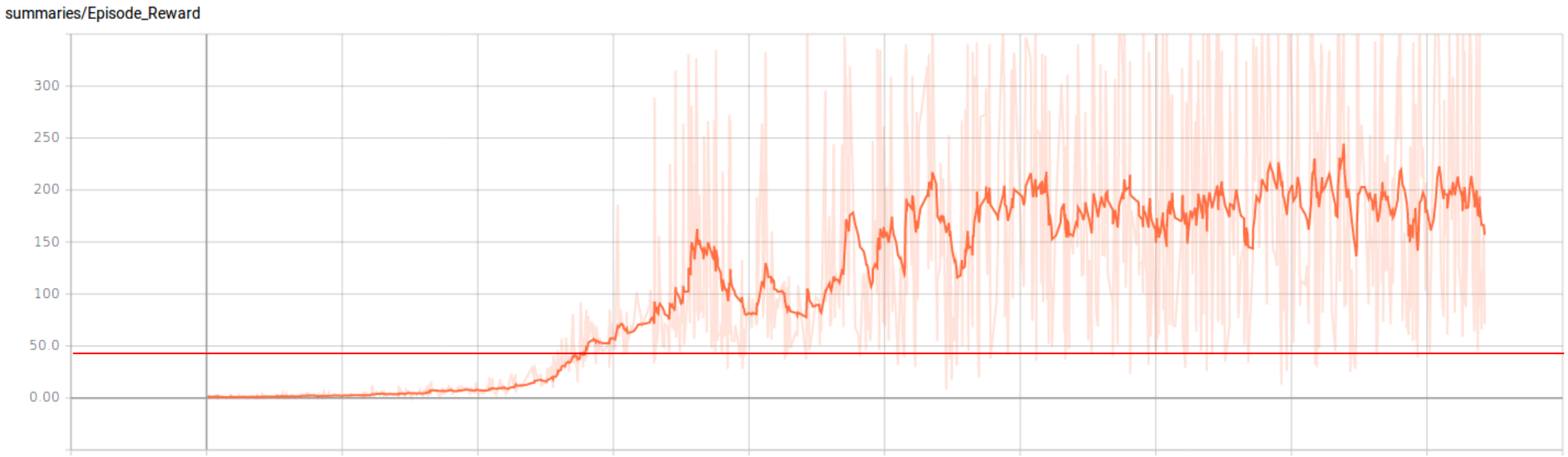
- The action should act ε -greedily
  - Random action with probability ε
- Linearly decline ε from 1.0 to some small value, say 0.025
  - Decline per step
  - Randomness is for exploration, agent is weak at start
- Hyperparameters
  - Replay Memory Size 10000
  - Perform Update Current Network Step 4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

Improvements to DQN

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

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

Training Plot



- X-axis : 1000 episodes/unit
- Y-axis : Unclipped reward per episode

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.

Asynchronous Update (Optional)

- 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.

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_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_dqn.py
- You can add your arguments in argument.py

Grading & Format

Submission

- Submit your presentation files to: google drive

Grading & Format

Baseline

- DQN
  - Getting averaging reward in 100 episodes over **40** in **Breakout**
  - With OpenAI’s Atari wrapper & reward clipping
    - You SHOULD will unclip the reward when testing

Grading & Format

Slides

- Describe your DQN model
- Plot the learning curve to show the performance of your Deep Q Learning on Breakout
  - X-axis: number of time steps
  - Y-axis: average reward in last 30 episodes
- Implement 1 improvement method on page 10
  - Describe your tips for improvement
  - Learning curve
  - Compare to origin Deep Q Learning

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

Double DQN

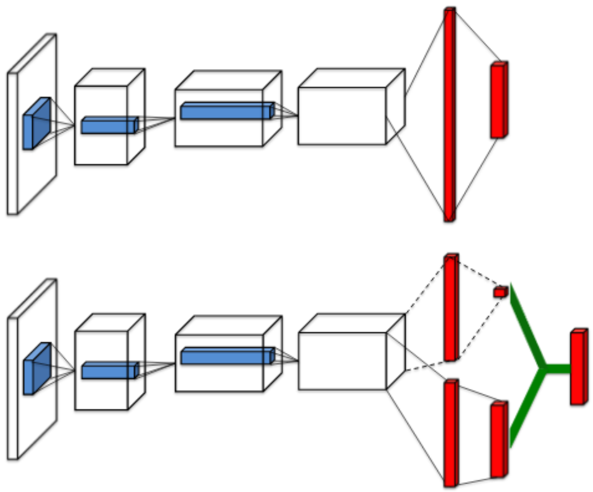
- The formula $$Y_t^{\mathrm{DQN}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_t^-) \,. \tag{3}$$
  often overestimates the maximum Q value.
- Thus instead choose the action of the max Q in the target network, choose the action of the max Q in the current network.

$$Y_t^{\mathrm{DoubleDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t), \boldsymbol{\theta}_t^-) \,.$$

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

Dueling Network

- In many state, action does not counts.
  - DQN trys 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)

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

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 ∝ |TD ERROR|^ω*
  - ω is a hyperprameter, 0.5 in Rainbow

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

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

Prioritized Replay Memory

- However, the resulting gradient estimator is biased, since we are sampling from a different distribution
    - Correct by inportance sampling weights

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

- With $\rho\_i = 1 / P(i)$, the IS weights

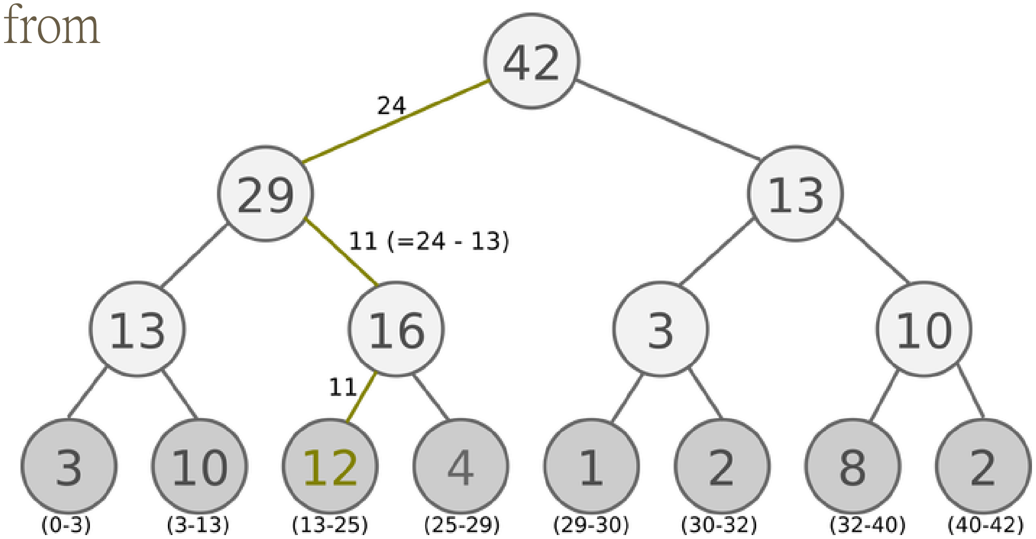
$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)} \right)^{\beta}$$

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

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

Prioritized Replay Memory

- Using array, the complexity of sampling is *O(n)*
  - Try another data structure
- Sum Tree, which prioiritized sampling can be *O(lgn)*
  - 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



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

Prioritized Replay Memory

Algorithm 1 Double DQN with proportional prioritization

Input: minibatch k, step-size η, replay period K and size N, exponents α and β, budget T.
Initialize replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
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 $\mathcal{H}$ with maximal priority $p_t = \max_{i<t} p_i$
  if $t \equiv 0 \mod K$ **then**
    for $j = 1$ **to** k **do**
      Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
      Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
      Compute TD-error $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
      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 $\Delta = 0$
    From time to time copy weights into target network $\theta_{\text{target}} \leftarrow \theta$
  end if
  Choose action $A_t \sim \pi_\theta(S_t)$
end for