COSC 689: Deep Reinforcement Learning

Spring 2019

Prof. Grace Hui Yang
Department of Computer Science
Georgetown University

Midterm

- Apr 3, in class, open book
- Written questions, covering:
  - MDP
  - Dynamic Programming (Value iteration, policy iteration, policy improvement, policy evaluation)
  - PG and AC (REINFORCE, REINFORCE w/baseline, AC, A3C, TRPO, PPO, GAE)
  - TD (SARSA, Q-learning, expected SARSA, double Q-learning, DQN, DDPG)

Outline

- Deep Q-learning
- Deep Q-Network (DQN)
- Deep Deterministic Policy Gradient (DDPG)
- What are working?
  - Experience replay
  - Target network
  - Double Q-Learning
  - N-Step Q-learning
- Practical considerations for Deep Q-learning

Q-learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

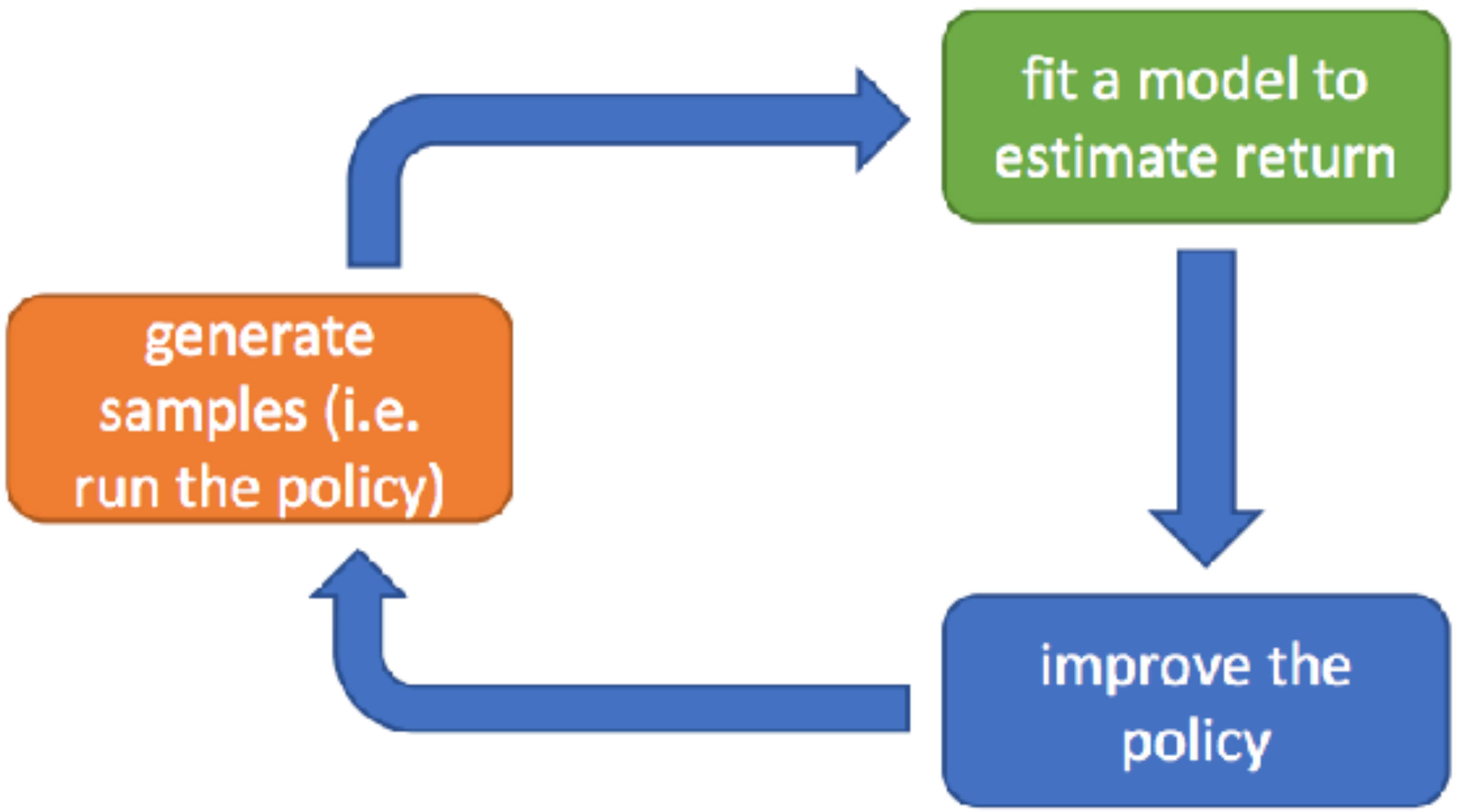
Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  until S is terminal

Deep Q-Learning

- Use a deep neural network to learn the Q-function
- How:
  - Parameterize Q function with ϕ
  - Use SGD to find the optimal ϕ and the optimal Q_ϕ

Deep Q-Learning

$$Q_\phi(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}', \mathbf{a}')$$



$$\mathbf{a} = \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a})$$

Image from UC Berkeley CS294-112

Naive DQN

- Also known as Online Q-iteration algorithm:

  Repeat:

  Take some action a_i and observe (s_i, a_i, s'_i, r_i)

  Calculate TD error's target $y_i = r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i)$

  Update $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i)[Q_\phi(s_i, a_i) - y_i]$

Naive DQN

- Also known as Online Q-iteration algorithm:

  Repeat:

  Take some action a_i and observe (s_i, a_i, s_i', r_i)

  Calculate TD error's target $y_i = \underbrace{r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s_i', a_i')}_{\text{Target}}$

  Update $\phi \leftarrow \phi - \alpha \dfrac{dQ_\phi}{d\phi}(s_i, a_i)[\underbrace{Q_\phi(s_i, a_i) - y_i}_{\text{TD error}}]$

Naive DQN

- Suffers from two major issues:
  - Correlated samples in sequential states
    - The neural network used to learn Q is too easily to overfit to with current episodes
    - Once it is overfitted, it won't be able to generate varying experiences
    - It gets stuck!

Image from UC Berkeley CS294-112

  - A moving target
    - It is because the target $y_i = r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s_i', a_i')$ depends on the same parameters that we are trying to learn — ϕ
    - It becomes a chicken-and-egg problem and the learning becomes super unstable
    - And when we use SGD to seek for ϕ, the gradient is not though the target function part (see next slide)

Naive DQN

- Also known as Online Q-iteration algorithm:

Repeat:

Take some action a_i and observe (s_i, a_i, s_i', r_i)

Calculate TD error's target $y_i = r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s_i', a_i')$

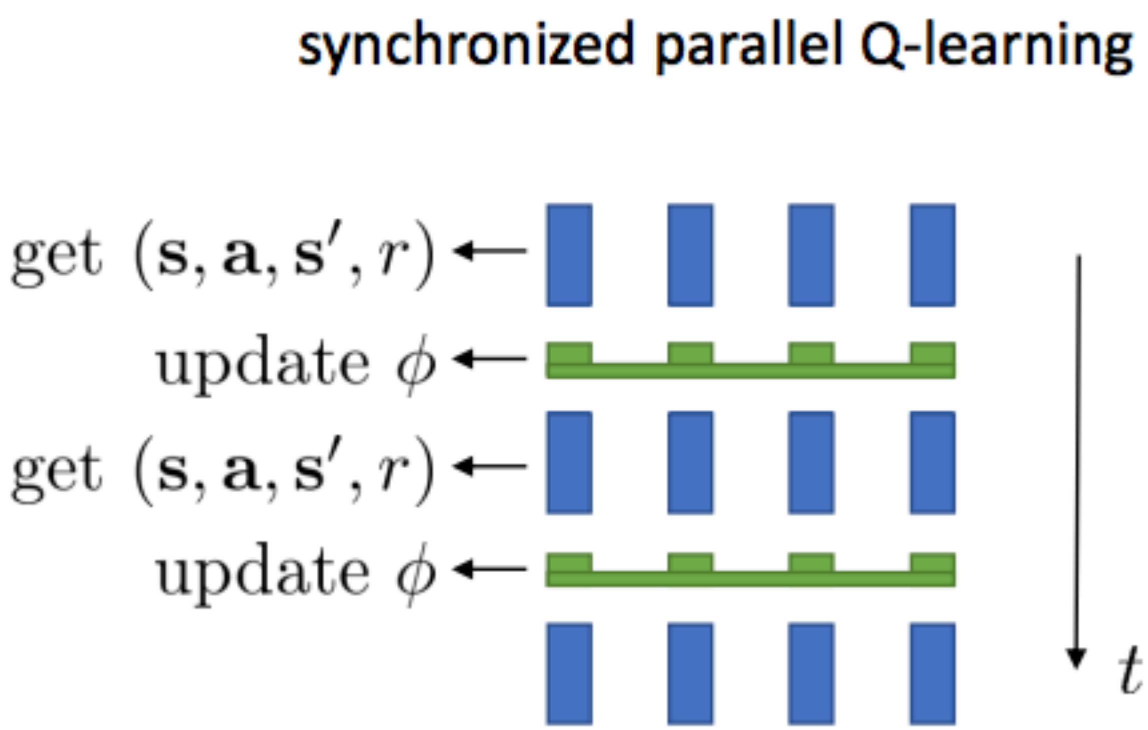
Update $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i)[Q_\phi(s_i, a_i) - \underline{y_i}]$

$$= \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i)[Q_\phi(s_i, a_i) - \underline{(r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s_i', a_i'))}]$$

gradient is not through this part

Solutions to Correlated samples - asynchronous learning

- Solution 1: asynchronous learning to break the correlations



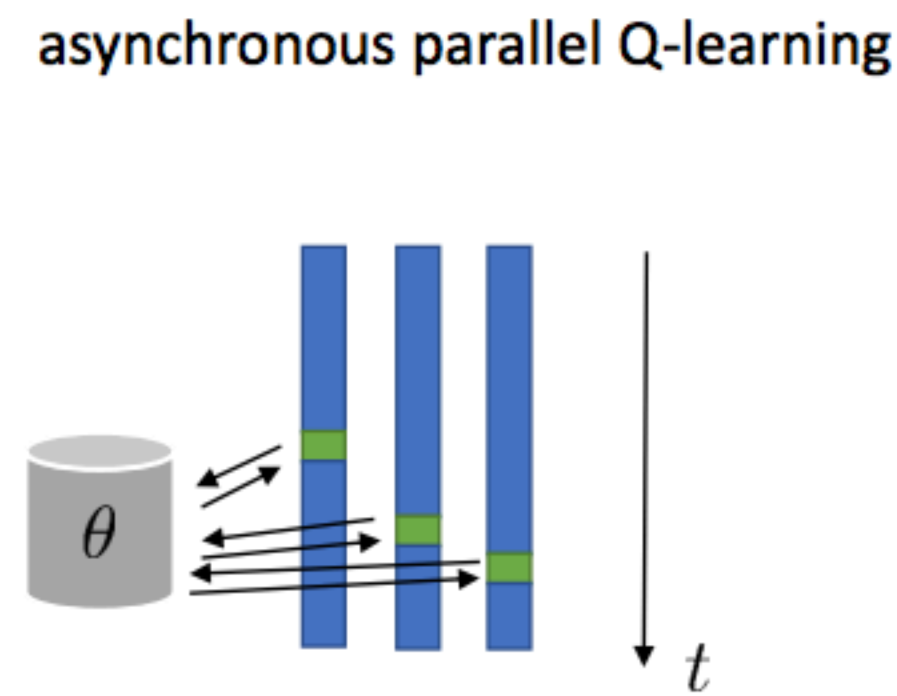


Image from UC Berkeley CS294-112

Solutions to Correlated samples - Experience Replay

- Solution 2: sample the recent states from a buffer

- This technique is called “experience replay”

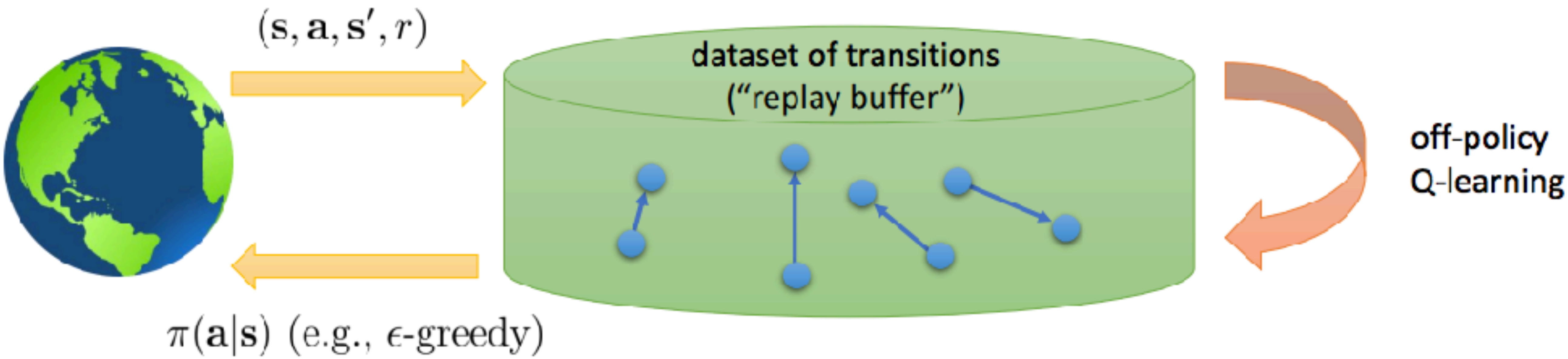


Image from UC Berkeley CS294-112

Experience Replay

- Idea:
  - Stores the experiences, including state transitions, rewards, actions, into a buffer
    - What are stored are those that are essential to learn a Q-function
      - state transitions (from s to s’; note it is not the entire trajectory, but pairs of Markov transitions)
      - rewards
      - actions
  - Then, random samples (mini-batches) from this buffer to update the neural network

Experience Replay

- Benefits:
  - It reduces the correlation between recent experiences when updating the NN
  - With a good size of mini-batches, it can speed up the learning
  - reuses past experience to avoid sudden change/forgetting

Experience Replay

- How big the buffer should be?
  - the replay buffer should be big enough to contain a wide range of experiences
  - however, not keep everything. It would be very inefficient
- How recent the experiences should be?
  - If you use very recent date only, then it again easily overfit to recent samples
  - If you use a lot of past experiences, which means to keep more data in the buffer, it will slow you down
- Tuning is thus important

Naive DQN + Experience Replay

Repeat:

collect experiences $\{(s_i, a_i, s_i', r_i)\}$, add them to replay buffer B

Repeat:

sample a batch of (s_i, a_i, s_i', r_i) from B

calculate TD error's target $y_i = r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s_i', a_i')$

update $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i)[Q_\phi(s_i, a_i) - y_i]$

Issue - Moving Target

Repeat:

Take some action a_i and observe (s_i, a_i, s'_i, r_i)

Calculate TD error's target $y_i = r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i)$

Update $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i)[Q_\phi(s_i, a_i) - (r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i))]$

gradient is not through this part

- The targets don't change in the inner loop
- Q-learning (Naive DQN) is not gradient descent
- The learning is super unstable

Solution to moving target - Target Network

- Idea:
  - Use a separate target network (parameterized by ϕ') in Q-value fitting
  - In the new target network, we use a set of parameters that is close to ϕ, but not exactly it, with a time delay
  - Ways to do it:
    - Directly copy over the learned ϕ from Q-network to the target network (as in DQN):

$$\phi' \leftarrow \phi$$

    - Use Polyak averaging (as in DDPG)

$$\phi' \leftarrow \tau\phi' + (1-\tau)\phi \qquad \tau = 0.999 \text{ for instance}$$

Naive DQN + Experience Replay + Target Network

Repeat:

Update target network parameter by copying: $\phi' \leftarrow \phi$

Repeat (N times):

collect experiences $\{(s_i, a_i, s_i', r_i)\}$, add them to B

Repeat (K times):

sample a batch of $\{(s_j, a_j, s_j', r_j)\}$ from B

compute TD error's target using target network parameters $y_j = r(s_j, a_j) + \gamma \max_{a'} Q'_\phi(s_j', a_j')$

update $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_j, a_j)[Q_\phi(s_j, a_j) - y_j]$

Naive DQN + Experience Replay + Target Network

Repeat:

Update target network parameter by copying: $\phi' \leftarrow \phi$

Repeat (N times):

collect experiences $\{(s_i, a_i, s_i', r_i)\}$, add them to B

Repeat (K times):

sample a batch of $\{(s_j, a_j, s_j', r_j)\}$ from B

compute TD error's target using target network parameters

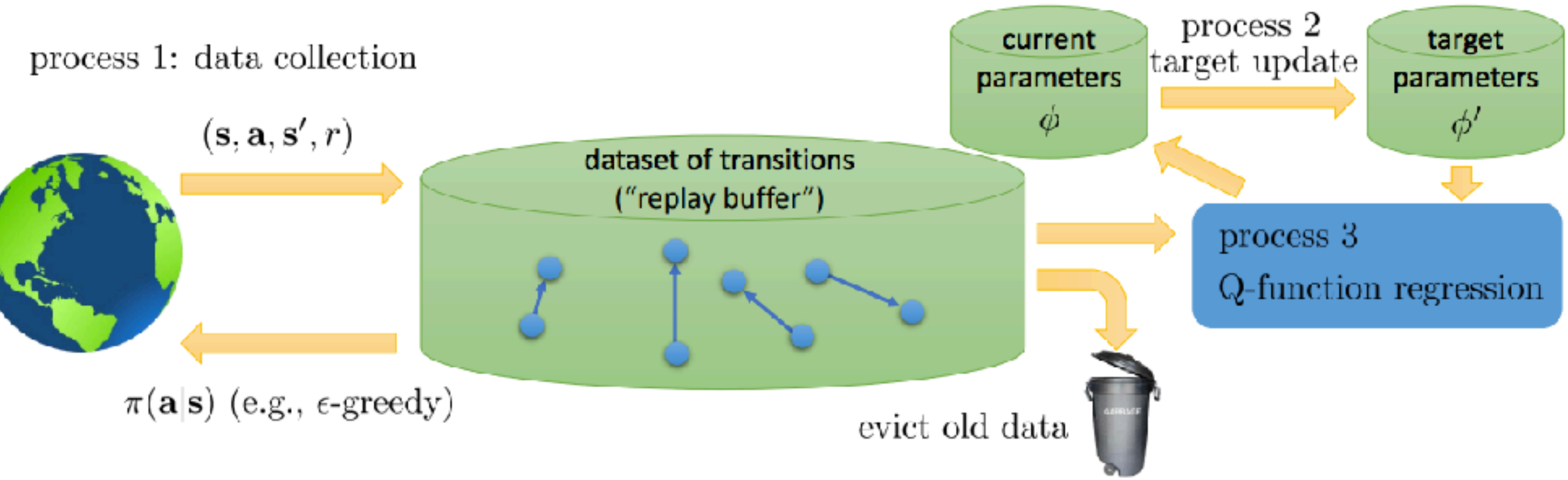
$$y_j = r(s_j, a_j) + \gamma \max_{a'} Q'_\phi(s_j', a_j')$$

update $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_j, a_j)[Q_\phi(s_j, a_j) - y_j]$

N: skipping steps
K: times of random sampling

Note the index changes from i to j

Illustration



- Online Q-learning (last lecture): evict immediately, process 1, process 2, and process 3 all run at the same speed •

- DQN: process 1 and process 3 run at the same speed, process 2 is slow

Image from UC Berkeley CS294-112

Deep Q-Network (DQN)

- It is Naive DQN + Experience Replay + Target Network, when N=1 and K=1.
- DQN Paper: Human-level control through deep reinforcement learning: Q-learning with convolutional networks for playing Atari. https://www.nature.com/articles/nature14236

DQN

Repeat:

Update target network parameter by copying: $\phi' \leftarrow \phi$

Repeat (N times): **N=1 and K=1**

collect experiences $\{(s_i, a_i, s_i', r_i)\}$, add them to B

Repeat (K times):

sample a batch of $\{(s_j, a_j, s_j', r_j)\}$ from B

compute TD error's target using target network parameters $y_j = r(s_j, a_j) + \gamma \max_{a'} Q'_\phi(s_j', a_j')$

update $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_j, a_j)[Q_\phi(s_j, a_j) - y_j]$

It is your Homework 3.

DQN Algo in the Paper

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Further Improvement: Double Q-learning

- Overestimation in Q-learning
- (We mentioned in last lecture) It is because Q-learning is a greedy algorithm
    - Or at least \epsilon-greedy
    - It can be biased towards overestimation of the next value

$$\text{target value } y_j = r_j + \gamma \max_{\mathbf{a}'_j} \underline{Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)}$$

Overestimation in Q-Learning

$$\text{target value } y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$$

$$\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}') = \underbrace{Q_{\phi'}}_{\text{evaluate Q-value}}(\mathbf{s}', \underbrace{\arg\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}')}_{\text{action selection}})$$

they both depend on $Q_{\phi'}$

Double Q-Learning

- Idea:
  - Do not use the same network for choosing actions and evaluating values
  - Instead, use two different networks to do the above (with 0.5 probably for each)

$$Q_{\phi_A}(\mathbf{s}, \mathbf{a}) \leftarrow r + \gamma Q_{\phi_B}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi_A}(\mathbf{s}'))$$

$$Q_{\phi_B}(\mathbf{s}, \mathbf{a}) \leftarrow r + \gamma Q_{\phi_A}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi_B}(\mathbf{s}'))$$

  - In practice, we use target network to evaluate Q-value and current network to choose action

Double Q-Learning

- In practice, we use target network to evaluate Q-value and *use current network to choose action*:

$$y_j = r(s_j, a_j) + \gamma Q_{\phi'}(s_j', \arg\max_{a'} Q_{\phi}(s_j', a_j'))$$

Double Q-Learning

- In practice, we use target network to evaluate Q-value and *use current network to choose action*:

$$y_j = r(s_j, a_j) + \gamma Q_{\phi'}(s'_j, \arg\max_{a'} Q_{\phi}(s'_j, a'_j))$$

Not $Q_{\phi'}$

Further Improvement: N-Step Return

- Q-learning's target is high in bias and low in variance.
- It is because it only looks ahead 1 step and 1 reward

$$y_j = r(s_j, a_j) + \gamma Q_{\phi'}(s_j', \arg\max_{a'} Q_{\phi}(s_j', a_j'))$$

- How to improve (to make it low in both)?
    - We can use N-step returns as in actor-critic

Q-learning + N-Step Return

- Q-learning target becomes:

$$y_{j,t} = \sum_{t'=t}^{t'+N-1} r_{j,t'} + \gamma^N \max_{\mathbf{a}_{j,t+N}} Q_{\phi'}(\mathbf{s}_{j,t+N}, \mathbf{a}_{j,t+N})$$

- When we do action selection,

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 \text{ otherwise} \end{cases}$$

we will need transitions $\mathbf{s}_{j,t'}, \mathbf{a}_{j,t'}, \mathbf{s}_{j,t'+1}$ all to come from the same policy π for all $t' - t < N - 1$

- It is hard to do though - so we sometimes ignore the problem

Summary: DQN

- What are working?
  - Experience replay
  - Use a Target network
  - Double Q-Learning
  - N-Step Q-learning

Deep Q-learning with Continuous Actions

- DQN select actions based on the following or \epsilon-greedy:

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 \text{ otherwise} \end{cases}$$

- This works well with discrete actions

- However, when use DQN with continuous actions, we cannot exhaustively evaluate all actions in the space to find out the max

- Moreover, solving this optimization (the max) is in the inner loop of calculating the target, which makes the optimization highly nontrivial

Deep Q-learning with Continuous Actions

- DQN select actions based on the following or \epsilon-greedy:

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 \text{ if } \mathbf{a}_t = \underline{\arg\max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t)} \\ 0 \text{ otherwise} \end{cases}$$

a painfully expensive subroutine

- This works well with discrete actions

- However, when use DQN with continuous actions, we cannot exhaustively evaluate all actions in the space to find out the max

- Moreover, solving this optimization (the max) is in the inner loop of calculating the target, which makes the optimization highly nontrivial

Deep Q-learning with Continuous Actions

- Solution:
  - Parameterize the entire argmax part $\arg\max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t)$ by θ
  - Then, learn a maximizer (another network) μ_θ to approximate it - $\arg\max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t)$
  - Assumption: the Q-function is differentiable w.r.t the action $\frac{dQ_\phi}{d\theta} = \frac{d\mathbf{a}}{d\theta}\frac{dQ_\phi}{d\mathbf{a}}$
  - This leads to DDPG

Deep Deterministic Policy Gradient (DDPG)

- train another network $\mu_\theta(\mathbf{s})$ such that $\mu_\theta(\mathbf{s}) \approx \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a})$
- Then, solve $\theta \leftarrow \arg\max_\theta Q_\phi(\mathbf{s}, \mu_\theta(\mathbf{s}))$
- After this action selection, the new target becomes

$$y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mu_\theta(\mathbf{s}'_j))$$

- DDPG paper: Continuous control with deep reinforcement learning: continuous Q-learning with actor network for approximate maximization.

DDPG

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 $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mu_{\theta'}(\mathbf{s}'_j))$ using *target* nets $Q_{\phi'}$ and $\mu_{\theta'}$
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. $\theta \leftarrow \theta + \beta \sum_j \frac{d\mu}{d\theta}(\mathbf{s}_j) \frac{dQ_\phi}{d\mathbf{a}}(\mathbf{s}_j, \mathbf{a})$
6. update ϕ' and θ' (e.g., Polyak averaging)

- DDPG paper: Continuous control with deep reinforcement learning: continuous Q-learning with actor network for approximate maximization.

Adapted from UC Berkeley CS294-112

DDPG in the Paper

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

DDPG

- It is an actor-critic method
- The value learning part (critic) is similar to DQN
- The action section (actor) is using Policy Gradient
- It also uses tricks including
  - experience replay
  - target network

Practical Tips for Q-learning

- Test on easy, reliable tasks first, make sure your implementation is correct
- Large replay buffers help improve stability
- Start with high exploration (\epsilon) and gradually reduce
- Start with high learning rate (\alpha) and gradually reduce
- Q-learning could be slow, so be patient
- Double Q-learning helps a lot in practice, simple and no downsides
- N-step returns also help a lot, but have some downsides
- Run multiple random seeds, it could be inconsistent between runs

Summary: Deep TD-Learning

- Widely used value-based method family
- Connects well with psychology and neuroscience
  - TD error
- DQN is a state-of-art RL method
- DDPG can be used for continuous action space; and works as an actor-critic method
- Key techniques to increase training stability:
  - experience replay
  - target network

References

- Main Textbook: RL book chapters 6.
  - Note that some images or formulas from the main textbook are not individually cited. You should assume they are from the textbook.
- DQN Paper: Human-level control through deep reinforcement learning: Q-learning with convolutional networks for playing Atari.
- DDPG paper: Continuous control with deep reinforcement learning: continuous Q-learning with actor network for approximate maximization.
- UC Berkeley CS294-112 Deep Reinforcement Learning