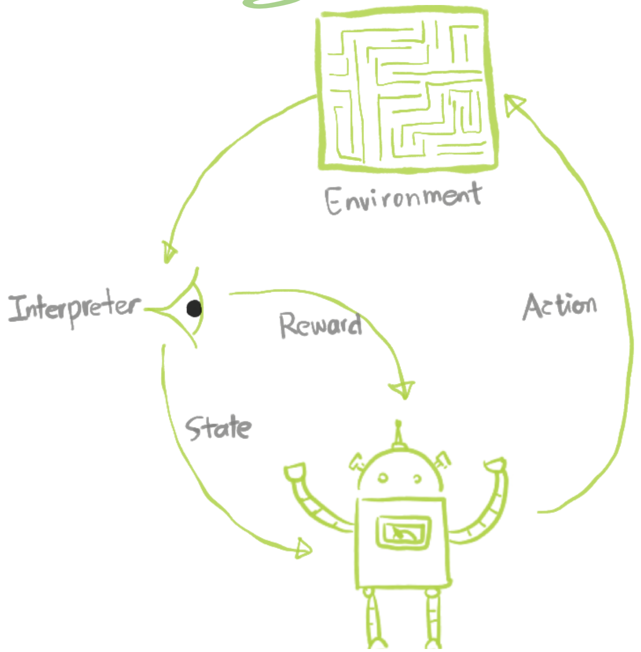
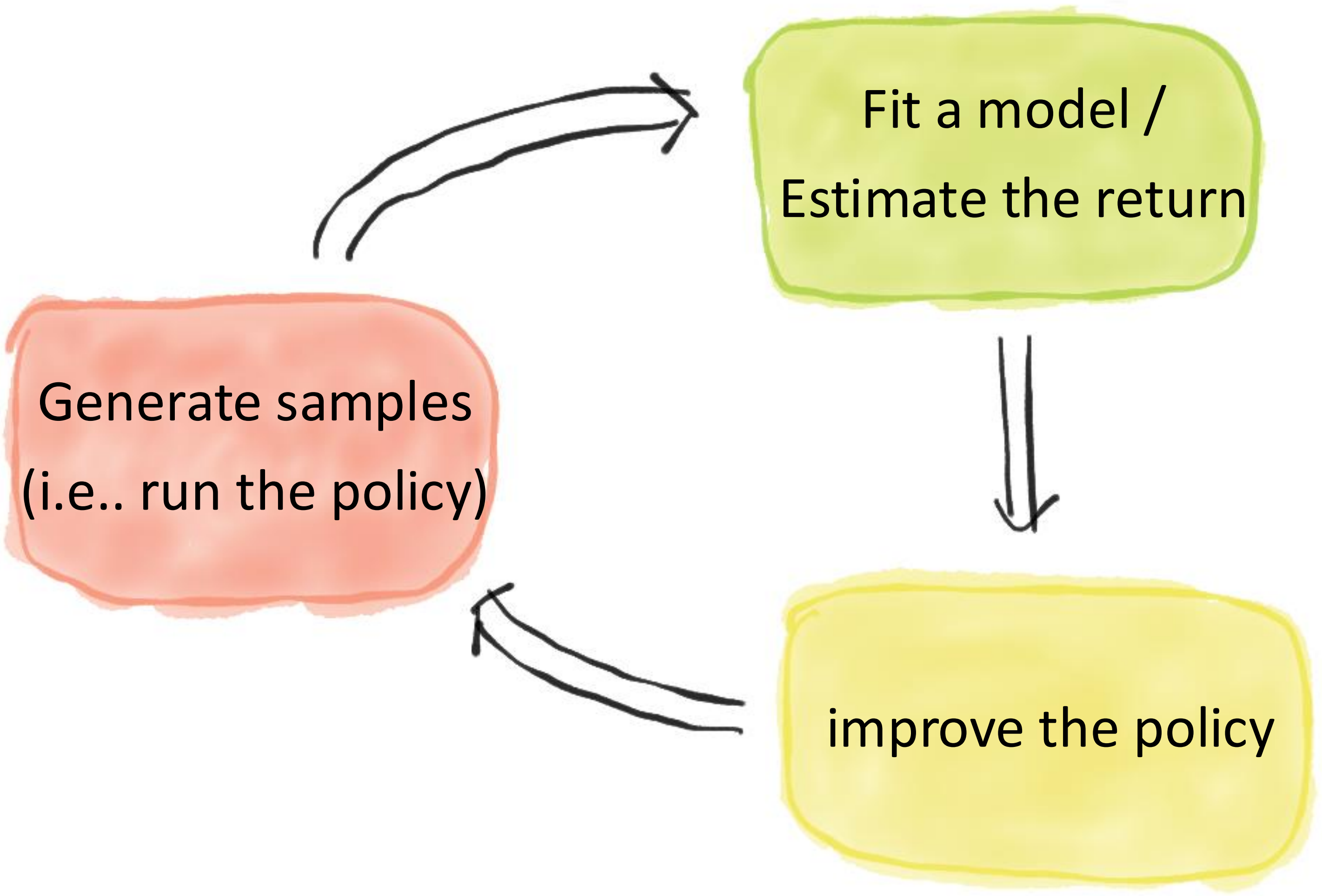
CS5491: Artificial Intelligence

Reinforcement Learning



Instructor: Kai Wang

Recap: Anatomy of Reinforcement Learning



Recap: Policy-gradient Algorithms

Fit a model /
Estimate the return

evaluate returns

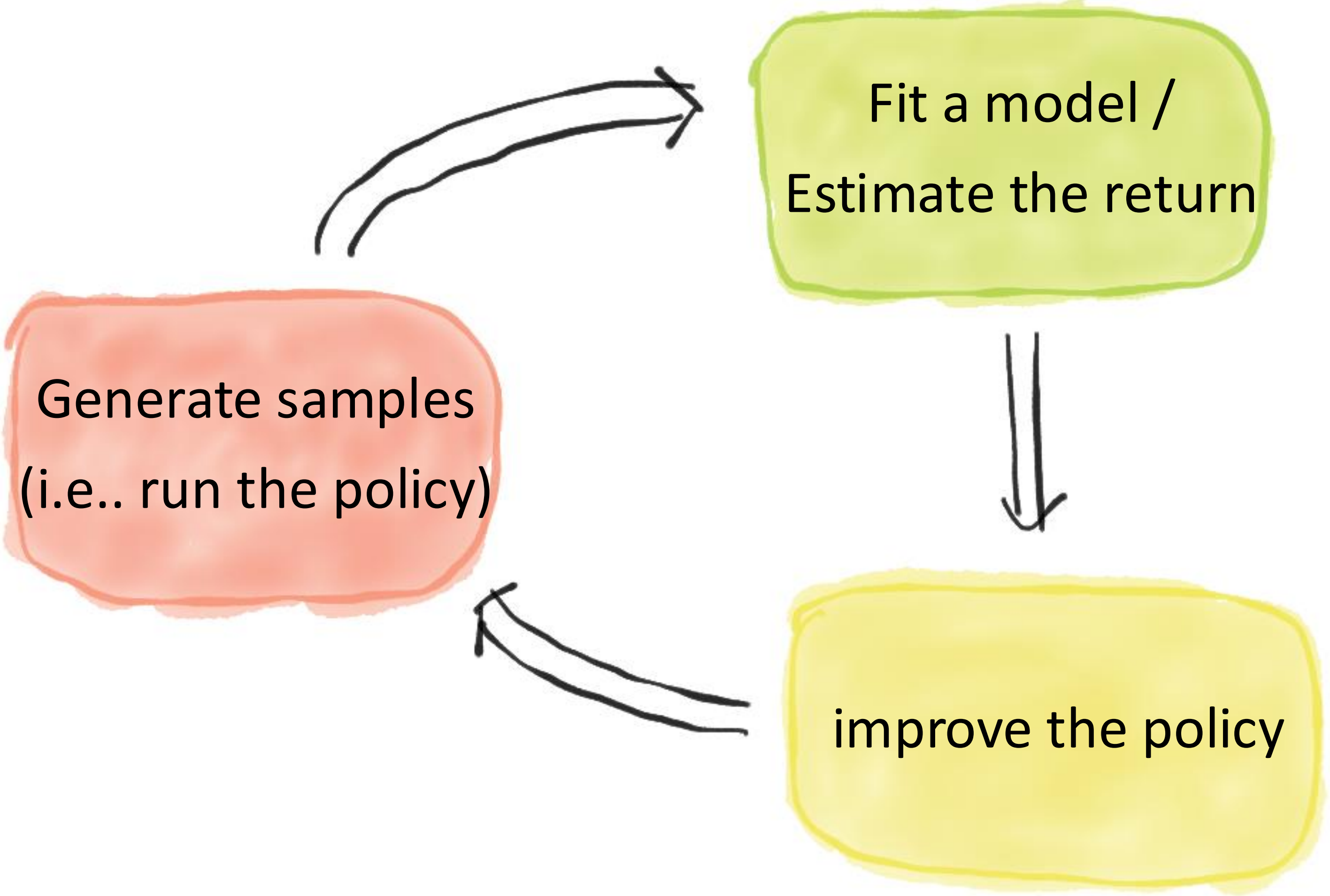
$$R_\tau = \sum_t r(s_t, a_t)$$

improve the policy

$$\theta \leftarrow \theta + \alpha \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

Generate samples
(i.e.. run the policy)

Recap: Value function-based Algorithms



fit $V(s)$ or $Q(s, a)$

set $\pi(s) = \arg\max_a Q(s, a)$

Recap: Value Function

Following a policy produces sample trajectories $(s_0, a_0, r_0, s_1, a_1, r_1, \cdots)$. The value function evaluates how good a state s is by measuring the expected cumulative reward from following the policy from state s.

$$V^{\pi}(s) = \mathbb{E}[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi]$$

Recap: Q-value Function

Following a policy produces sample trajectories $(s_0, a_0, r_0, s_1, a_1, r_1, \cdots)$. The Q-value function evaluates how good a state-action pair (s, a) is by measuring the expected cumulative reward from taking action a in state s and following the policy.

$$Q^{\pi}(s,a) = \mathbb{E}[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi]$$

Today

Q-Learning

Issues and Solutions

General view of Q-learning

Today

Q-Learning

Issues and Solutions

General view of Q-learning

Bellman Equation

- The optimal Q-value function Q^* is the maximum expected cumulative reward achievable from a given (state, action) pair:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi]$$

- Q^*satisfies the following Bellman Equation:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}[r + \gamma \max_{a'} Q^*(s',a') | s,a]$$

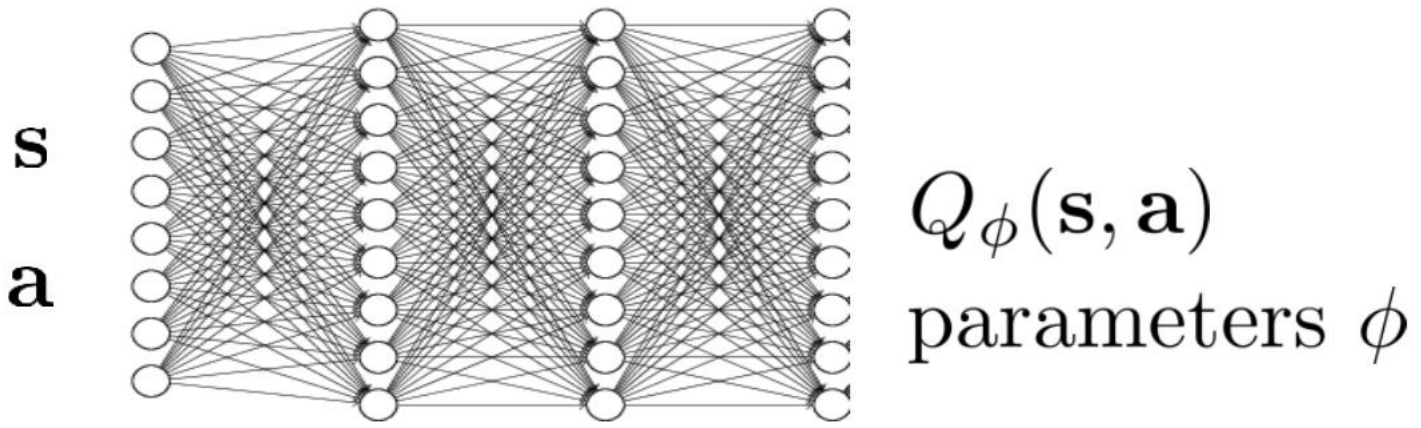
- Intuition: if the optimal state-action values for the next time step $Q^*(s',a')$ are known, then the optimal strategy is to take the action that maximizes the expected value of $r + \gamma Q^*(s',a')$.

Fitted Q-iteration

1. Collect dataset $\{(s_i, a_i, s_i', r_i)\}$ using some policy

$K \times$

2. Set $y_i = r(s_i, a_i) + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$
3. Set $\phi = \arg\min_\phi \frac{1}{2} \sum_i \| y_i - Q_\phi(s_i, a_i) \|^2$



Off-policy Property in Fitted Q-iteration

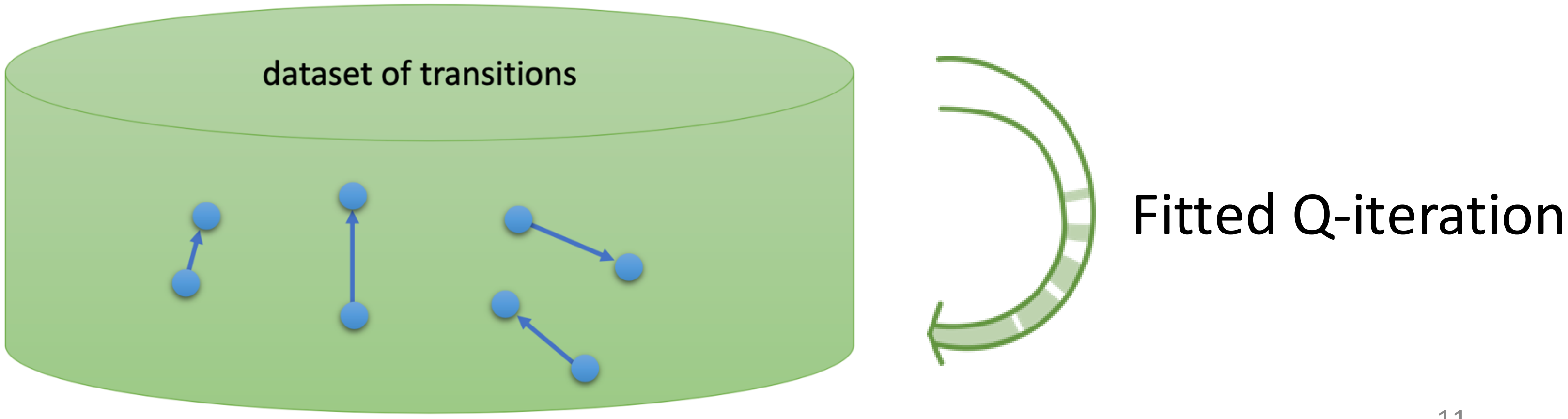
1. Collect dataset $\{(s_i, a_i, s_i', r_i)\}$ using some policy

$K \times$

2. Set $y_i = r(s_i, a_i) + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$

3. Set $\phi = \text{argmin}_\phi \frac{1}{2} \sum_i \left\| y_i - Q_\phi(s_i, a_i) \right\|^2$

Given $s_i, a_i, s_i{}'$, the transition is actually independent of the policy π



Online Q-learning

1. Take some action a_i, and observe $\{(s_i, a_i, s_i', r_i)\}$
2. Set $y_i = r(s_i, a_i) + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$
3. Set $\phi = \phi - \alpha \nabla_\phi Q(s_i, a_i)(y_i - Q_\phi(s_i, a_i))$

In fact, this is still off-policy; many options can be taken in Step 1.

Today

Q-Learning

Issues and Solutions

General view of Q-learning

Issue 1: Bad Policy in the Beginning

- Which policy should we use to sample the actions?
  - Eventually, we would use

$$\pi(a_t|s_t) = \begin{cases} 1 \; if \;\; a_t = \mathrm{argmax}_{a_t} Q_\phi(s_t, a_t) \\ 0 \; otherwise \end{cases}$$

  - In the beginning, however, the Q function could be very bad so that we stuck into bad and local transitions $\{(s_i, a_i, s_i', r_i)\}$

Solution 1: Exploration

- Epsilon-greedy exploration

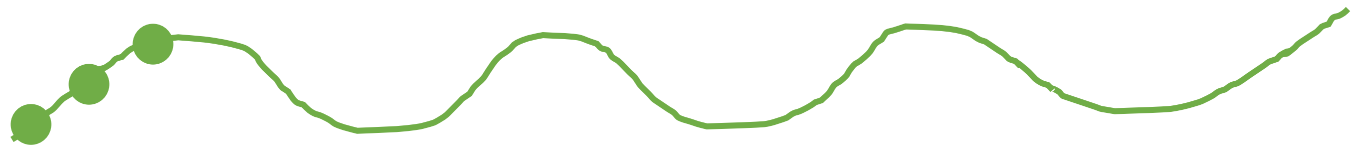
$$\pi(a_t|s_t) = \begin{cases} 1 - \epsilon \ if \ a_t = \text{argmax}_{a_t} Q_\phi(s_t, a_t) \\ \epsilon/(|\mathcal{A}| - 1) \ otherwise \end{cases}$$

- Boltzmann exploration

$$\pi(a_t|s_t) \propto \exp(Q_\phi(s_t, a_t))$$

Issue 2: Correlated Samples

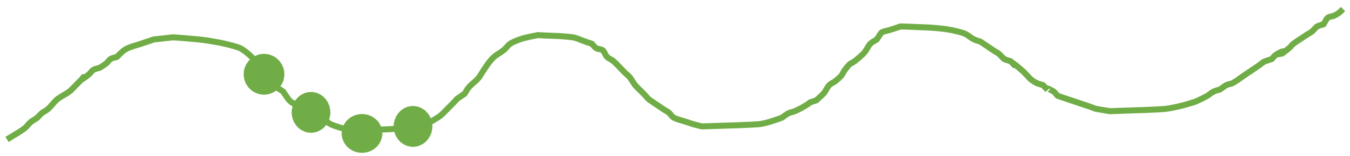
- Sequential states are strongly correlated



1. Take some action a_i, and observe $\{(s_i, a_i, s_i', r_i)\}$
2. Set $y_i = r(s_i, a_i) + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$
3. Set $\phi = \phi - \alpha \nabla_\phi Q(s_i, a_i)(y_i - Q_\phi(s_i, a_i))$

Issue 2: Correlated Samples

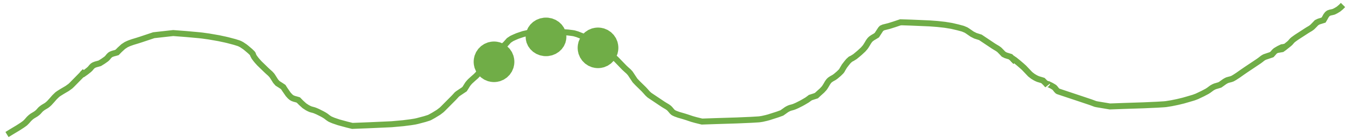
- Sequential states are strongly correlated



1. Take some action a_i, and observe $\{(s_i, a_i, s_i', r_i)\}$
2. Set $y_i = r(s_i, a_i) + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$
3. Set $\phi = \phi - \alpha \nabla_\phi Q(s_i, a_i)(y_i - Q_\phi(s_i, a_i))$

Issue 2: Correlated Samples

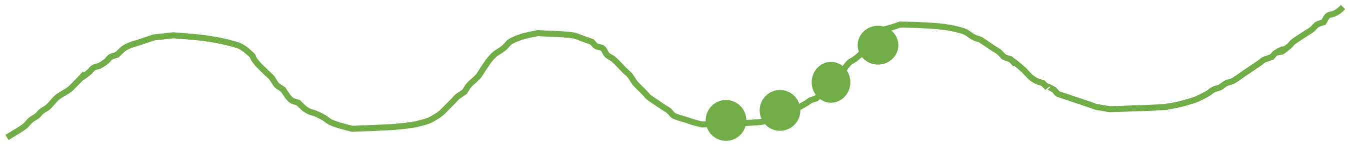
- Sequential states are strongly correlated



1. Take some action a_i, and observe $\{(s_i, a_i, s_i', r_i)\}$
2. Set $y_i = r(s_i, a_i) + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$
3. Set $\phi = \phi - \alpha \nabla_\phi Q(s_i, a_i)(y_i - Q_\phi(s_i, a_i))$

Issue 2: Correlated Samples

- Sequential states are strongly correlated

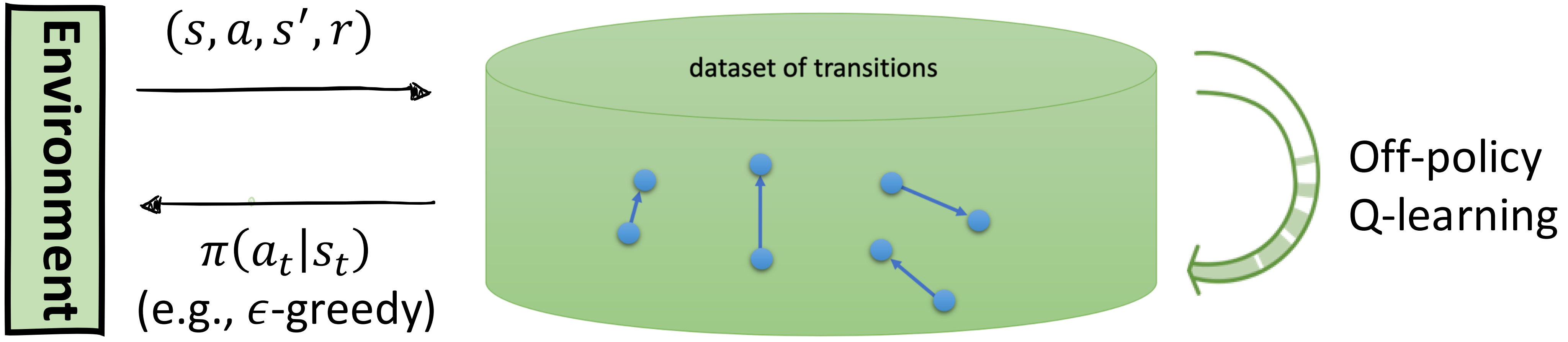


- This violates the identically and independently distributed (i.i.d.) assumption in supervised learning!

1. Take some action a_i, and observe $\{(s_i, a_i, s_i', r_i)\}$
2. Set $y_i = r(s_i, a_i) + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$
3. Set $\phi = \phi - \alpha \nabla_\phi Q(s_i, a_i)(y_i - Q_\phi(s_i, a_i))$

Solution 2: Experience Replay

- Just load the data from a replay buffer $\mathcal{B}$
- We need to periodically feed the replay buffer



Solution 2: Experience Replay

- Updated online Q-learning algorithm with experience replay

1. Take some action a_i and observe (s_i, a_i, s_i', r_i) , add it to $\mathcal{B}$
2. Sample a batch $\{(s_i, a_i, s_i', r_i)\}$ from $\mathcal{B}$
3. Set $y_i = r(s_i, a_i) + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$
4. Set $\phi = \phi - \alpha \nabla_\phi Q(s_i, a_i)(y_i - Q_\phi(s_i, a_i))$

Solution 2: Experience Replay

- Updated fitted Q-learning algorithm with experience replay

1. Collect dataset $\{(s_i, a_i, s_i', r_i)\}$ using some policy, add it to $\mathcal{B}$

$K \times$

2. Sample a batch $\{(s_i, a_i, s_i', r_i)\}$ from $\mathcal{B}$
3. Set $y_i = r(s_i, a_i) + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$
4. Set $\phi = \mathrm{argmin}_\phi \frac{1}{2} \sum_i \left\| y_i - Q_\phi(s_i, a_i) \right\|^2$

Deep Q-learning with Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    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 $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Issue 3: Moving Target and Poor Convergence

- This is not a regular regression problem.
- The target y_i is changing.
- There is no gradient through the target value.

1. Take some action a_i and observe (s_i, a_i, s_i', r_i) , add it to $\mathcal{B}$
2. Sample a batch $\{(s_i, a_i, s_i', r_i)\}$ from $\mathcal{B}$
3. Set $y_i = r(s_i, a_i) + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$
4. Set $\phi = \phi - \alpha \nabla_\phi Q(s_i, a_i)(y_i - Q_\phi(s_i, a_i))$

Solution 3: Target Networks

- Updated fitted Q-learning algorithm with target networks

$N \times$

$K \times$

1. Save target network parameters $\phi' \leftarrow \phi$
2. Collect dataset $\{(s_i, a_i, s_i', r_i)\}$ using some policy, add it to $\mathcal{B}$
3. Sample a batch $\{(s_i, a_i, s_i', r_i)\}$ from $\mathcal{B}$
4. Set $y_i = r(s_i, a_i) + \gamma \max_{a_i'} Q_{\phi'}(s_i', a_i')$
5. Set $\phi = \mathrm{argmin}_\phi \frac{1}{2} \sum_i \left\| y_i - Q_\phi(s_i, a_i) \right\|^2$

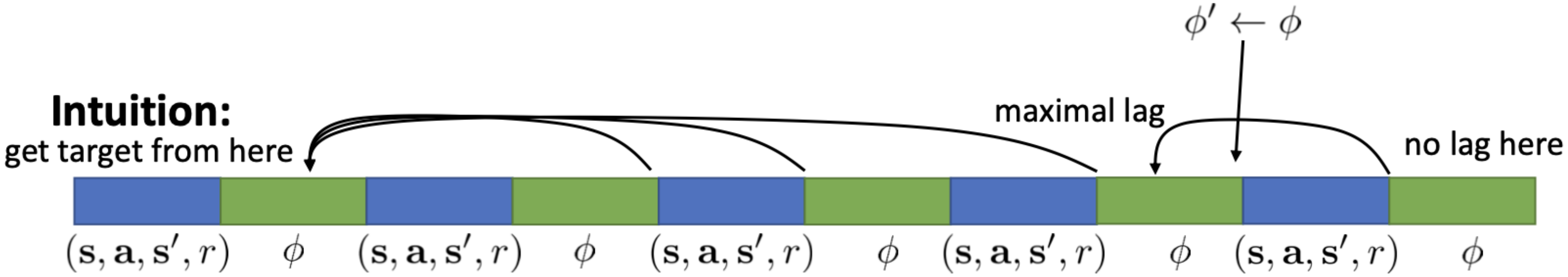
Solution 3: Target Networks

- Updated online Q-learning algorithm with target networks (DQN)

1. Take some action a_i and observe (s_i, a_i, s_i', r_i) , add it to $\mathcal{B}$
2. Sample a batch $\{(s_i, a_i, s_i', r_i)\}$ from $\mathcal{B}$
3. Set $y_i = r(s_i, a_i) + \gamma \max_{a_i'} Q_{\phi'}(s_i', a_i')$
4. Set $\phi = \phi - \alpha \nabla_\phi Q(s_i, a_i)(y_i - Q_\phi(s_i, a_i))$
5. Update ϕ': copy ϕ every N steps

Solution 3: Target Networks with Polyak Averaging

- Make the target network share the same lag always

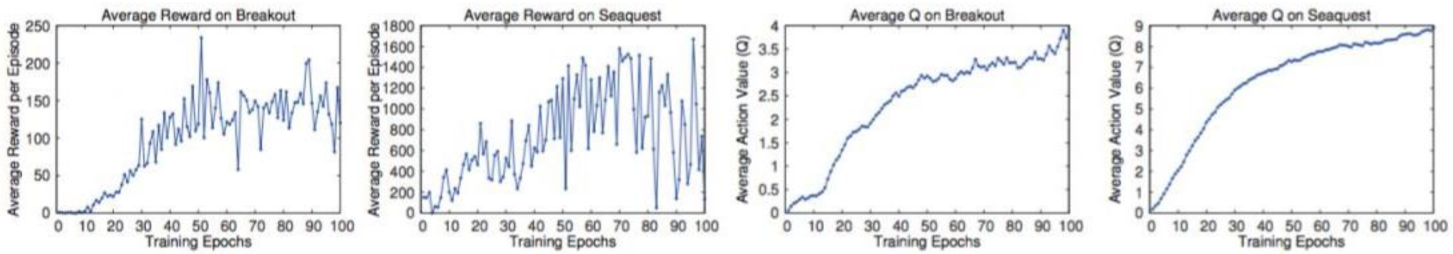


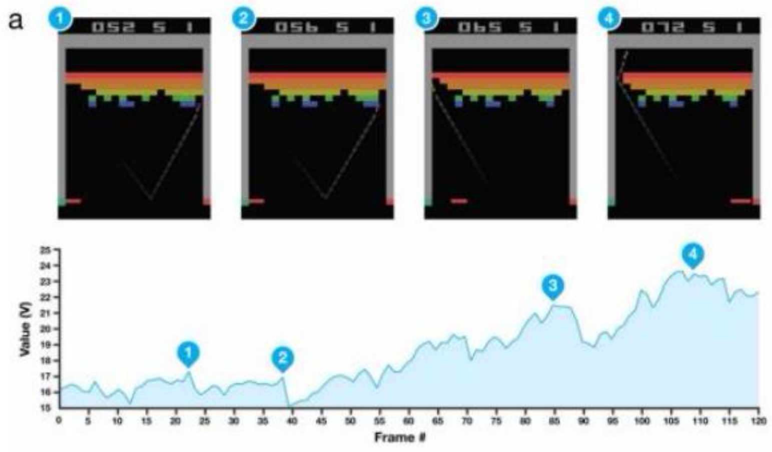
1. Take some action a_i and observe (s_i, a_i, s_i', r_i) , add it to $\mathcal{B}$
2. Sample a batch $\{(s_i, a_i, s_i', r_i)\}$ from $\mathcal{B}$
3. Set $y_i = r(s_i, a_i) + \gamma \max_{a_i'} Q_{\phi'}(s_i', a_i')$
4. Set $\phi = \phi - \alpha \nabla_\phi Q(s_i, a_i)(y_i - Q_\phi(s_i, a_i))$
5. Update ϕ': $\phi' = \tau\phi' + (1 - \tau)\phi$

$\tau = 0.999$ works well

Issue 4: Overestimation

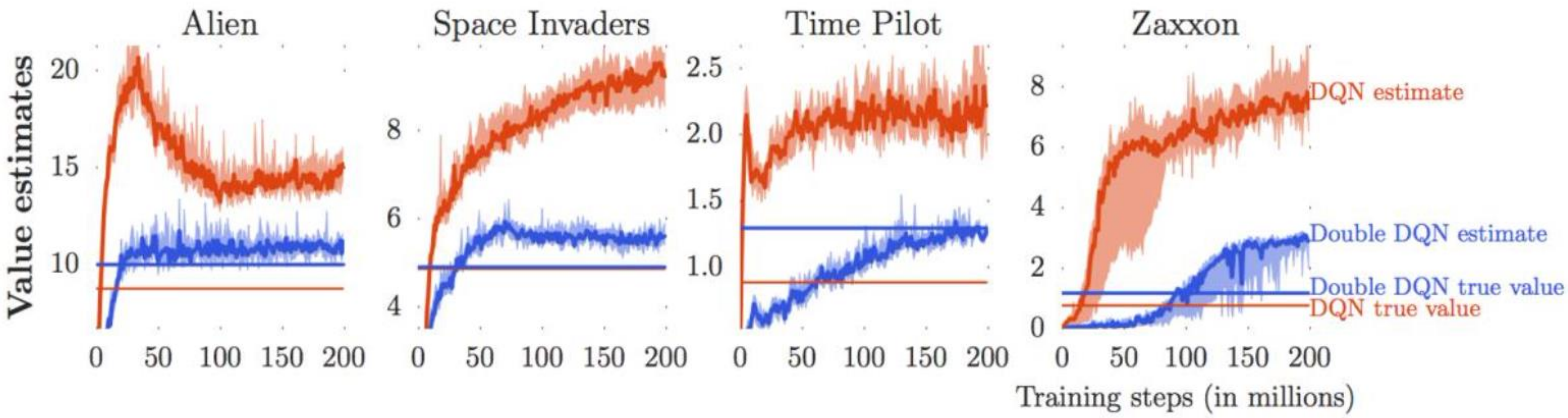
- The predicted Q-values share the same trend with the actual return (expected discounted rewards).





Issue 4: Overestimation

- Unfortunately, the absolute values are always way larger than the actual return.



Issue 4: Overestimation

$$y_i = r(s_i, a_i) + \gamma \max_{a_i'} Q_{\phi'}(s_i', a_i')$$
$$= r(s_i, a_i) + \gamma Q_{\phi'}\left(s_i', \text{argmax}_{a_i'} Q_{\phi'}(s_i', a_i')\right)$$

- $Q_{\phi'}(s_i', a_i')$ is not perfect – it looks noisy
- $E(\max(X_1, X_2)) > \max(E(X_1), E(X_2))$
- $\max_{a_i'} Q_{\phi'}(s_i', a_i')$ overestimates the next value

Solution 4: Double Q-Learning

- Use two different networks

$$y_i = r(s_i, a_i) + \gamma \max_{a_i'} Q_{\phi'}(s_i', a_i')$$
$$= r(s_i, a_i) + \gamma Q_{\phi_1'}\left(s_i', \operatorname{argmax}_{a_i'} Q_{\phi_2'}(s_i', a_i')\right)$$

If the noise in these is decorrelated into different ways, the problem goes away!

- Where can we get the second network?
  - Just use the current network ϕ as ϕ_2'

Issue 5: Q-learning with Continuous Actions

- How can we obtain the $\max_{a_i'} Q_{\phi'}(s_i', a_i')$ if the action space is continuous?

- Quick recipe: randomly/uniformly sample several actions from the continuous action space.

  not very accurate

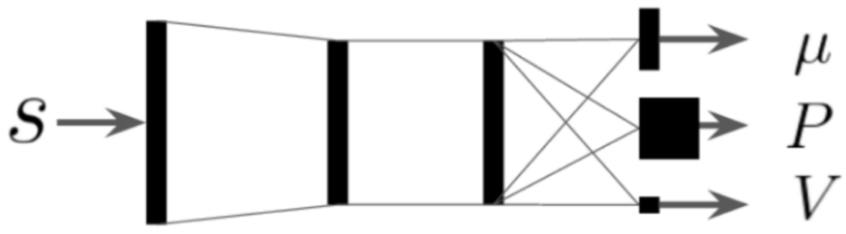
Solution 5a: Normalized Advantage Functions

- Use the function class of quadratic functions to easily optimize

$$Q_\phi(s,a) = -\frac{1}{2}\left(a - \mu_\phi(s)\right)^T P_\phi(s)\left(a - \mu_\phi(s)\right) + V_\phi(s)$$

- $\arg\max_{a_i'} Q_\phi(s_i', a_i') = \mu_\phi(s_i')$

- $\max_{a_i'} Q_\phi(s_i', a_i') = V_\phi(s)$



NAF Architecture.

Solution 5b: Maximizer Network

- Train another network $\mu_\theta(s)$ such that $\mu_\theta(s) \approx \arg\max_a Q_\phi(s,a)$
- How? Just solve $\theta = \theta + \beta \frac{dQ_\phi}{da} \frac{da}{d\theta}$

1. Take some action a_i and observe (s_i, a_i, s_i', r_i) , add it to $\mathcal{B}$
2. Sample a batch $\{(s_i, a_i, s_i', r_i)\}$ from $\mathcal{B}$
3. Compute $y_i = r(s_i, a_i) + \gamma Q_{\phi'}(s_i', \mu_{\theta'}(s_i'))$
4. Set $\phi = \phi - \alpha \nabla_\phi Q(s_i, a_i)(y_i - Q_\phi(s_i, a_i))$
5. Set $\theta = \theta + \beta \frac{dQ_\phi}{da} \frac{da}{d\theta}$
6. Update ϕ' and θ' : copy ϕ, θ every N steps

Today

Q-Learning

Issues and Solutions

General view of Q-learning

Deep Q-learning for Atari Games

- Objective: complete the game with the highest score
- State: raw pixel inputs of the game state
- Action: game controls (left, right, up, down)
- Reward: score increase/decrease at each time step

https://www.youtube.com/watch?v=V1eYniJ0Rnk

Q-network Architecture

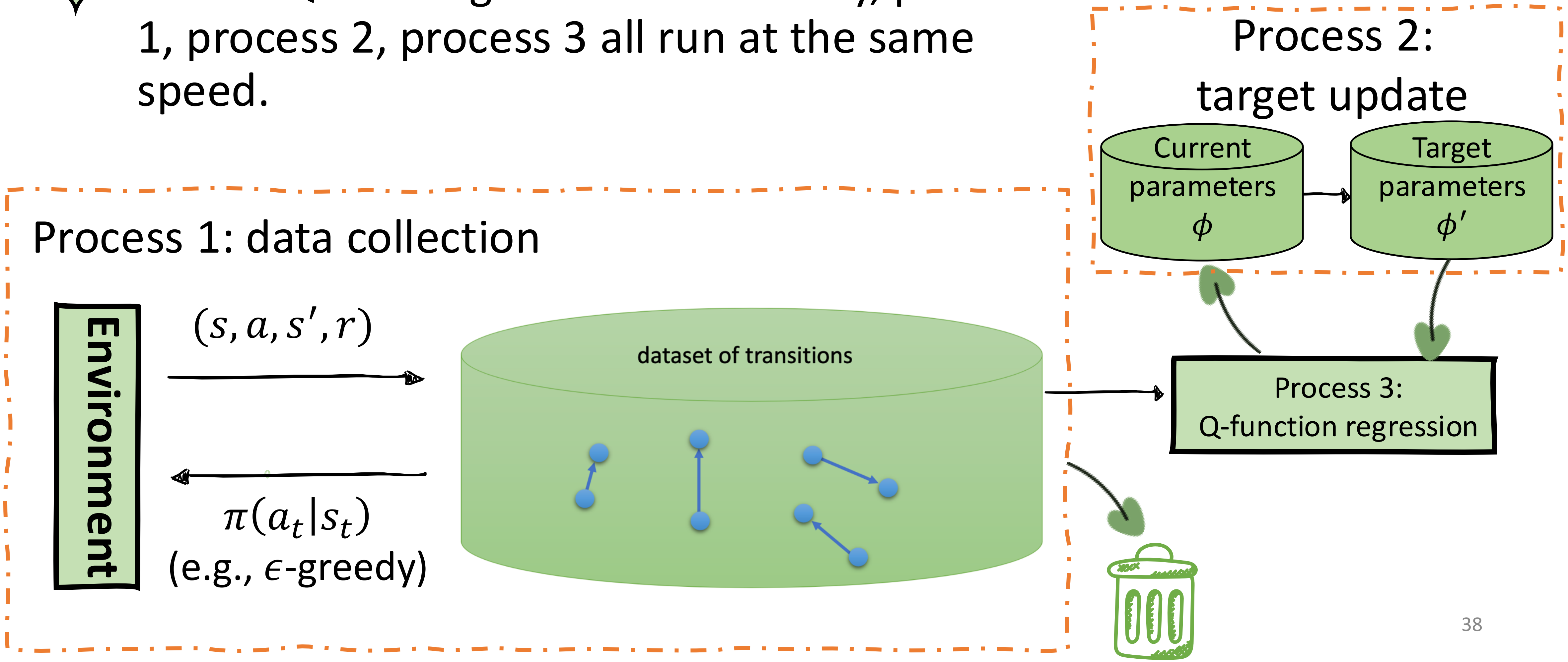
- $Q_\phi(s, a)$: the architecture of the neural network with weights ϕ

Current state s_t: 84x84x4 stack of last 4 frames

4-d output for 4 actions, corresponding to $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t, a_4)$

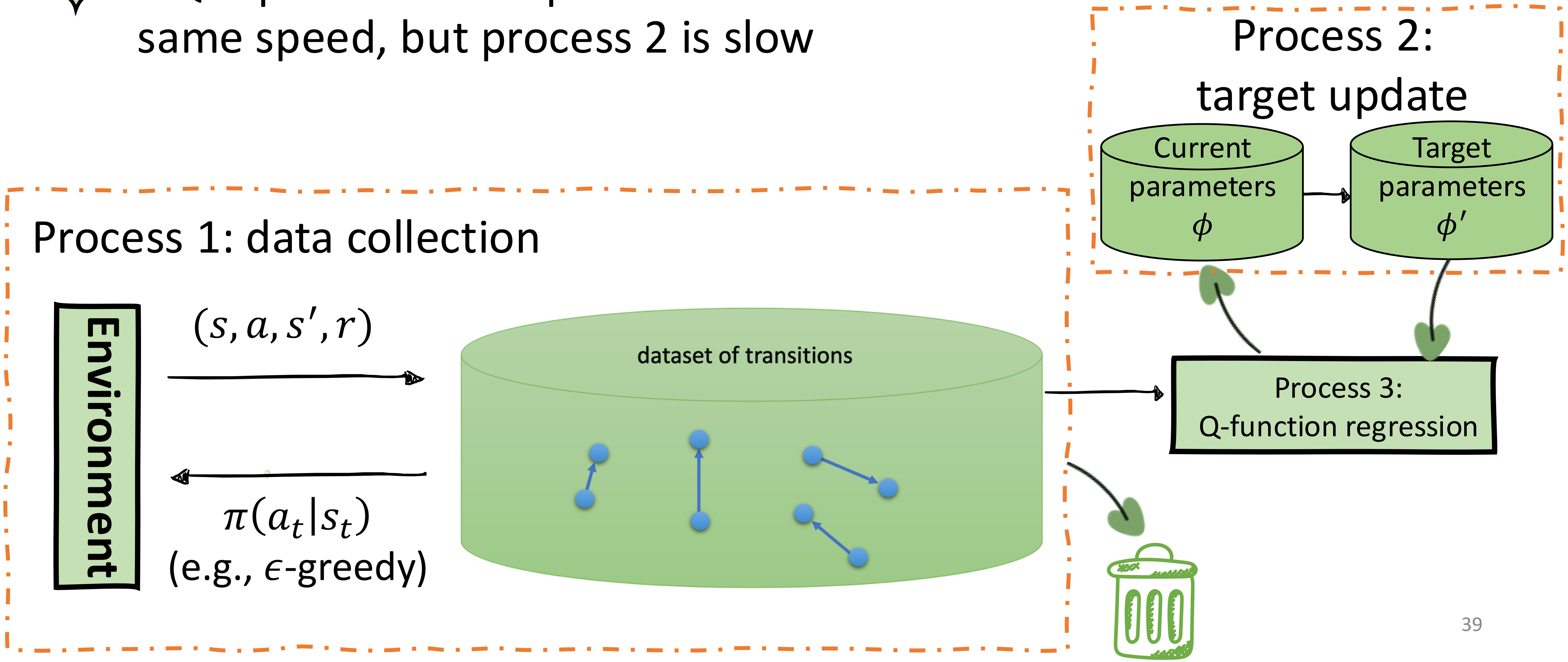
A General View of Q-learning

- Online Q-learning: evict immediately, process 1, process 2, process 3 all run at the same speed.



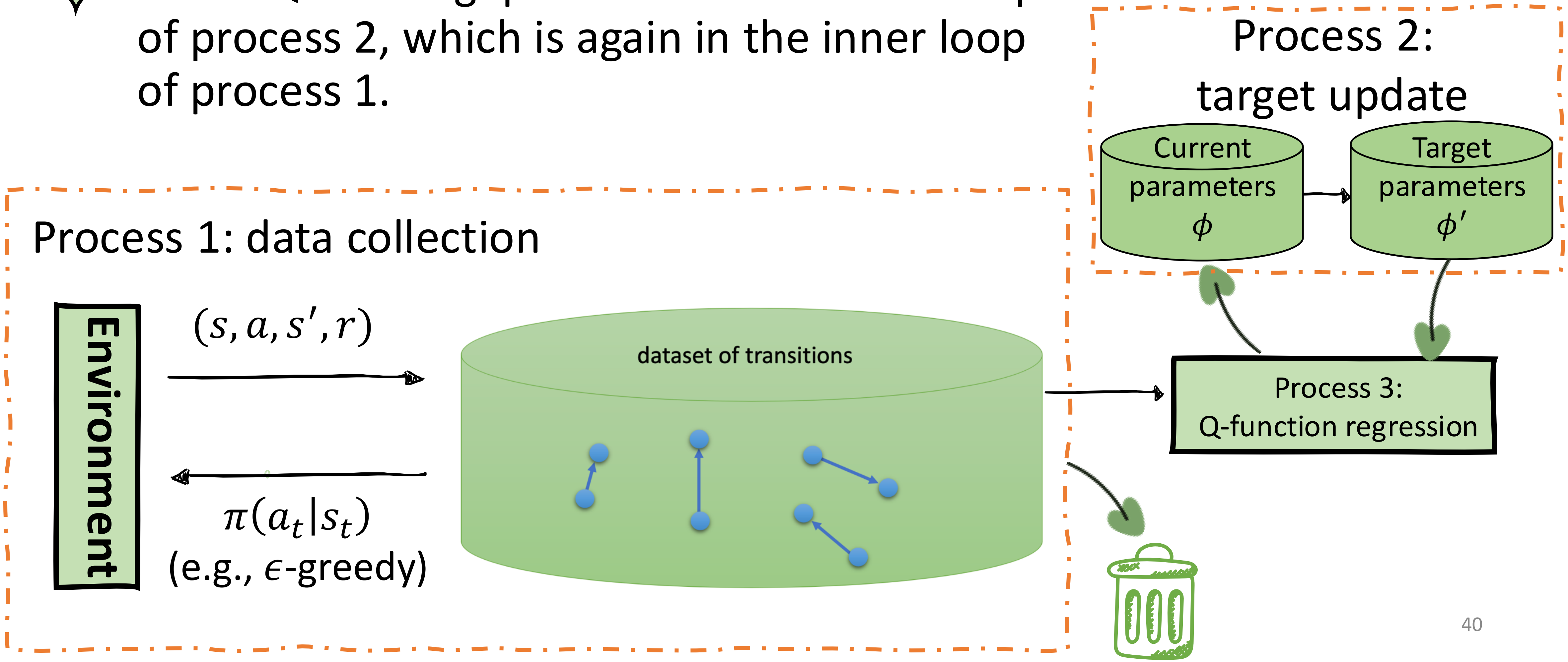
A General View of Q-learning

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



A General View of Q-learning

- Fitted Q-learning: process 3 in the inner-loop of process 2, which is again in the inner loop of process 1.



Comparison

Q-learning

- Sample-efficient
- Potentially poor exploration
- No optimality guarantees
- Does not always work

Policy gradients

- General and converges to a local minima of $J(\theta)$
- Suffer from high-variance
- Sample inefficient

Goals

- Understand basic value function-based methods.
- Understand the algorithmic insight and workflow behind Q-learning algorithm.
- Understand how to improve Q-learning algorithms to real-world problems.
- Learn how to use Keras to implement the Deep Q-learning.

Important This Week

- Read Reinforcement Learning: An Introduction.
- Know more about Reinforcement Learning algorithms here.
  http://rail.eecs.berkeley.edu/deeprlcourse/
- Know more about implementation issues of RL here.
  https://spinningup.openai.com/en/latest/