Learning to Act - Part3

Robotic Vision Summer School 2024

Pamela Carreno-Medrano and Dana Kulić
Monash University
dana.kulic@monash.edu

[1]The material covered in this lecture is based on David's Silver RL lectures at UCL, Mario Martin's RL lectures at UPC, and Sutton and Barton's Introduction to RL book

Activity 0: Notebook Setup

- Please open your Jupyter notebook environment and open the `LTASession3-Part1` notebook in the `Reinforcement_Learning` folder

Review of Session 2

Yesterday, we learned that:

- Reinforcement Learning problems can be formally defined as a Markov Decision Processes

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle,$$

  where $\mathcal{S}, \mathcal{A}$ indicate the state and action spaces, $\mathcal{T}$ and $\mathcal{R}$ are the transition and reward functions (i.e, our model of the environment), and γ is a discount factor.
- Solving a RL problem means finding the policy π^* that results in the largest expected return
- The state-value function $v(s)$ and the action-value function $q(s, a)$ are key when solving for the optimal policy
- If $\mathcal{T}$ and $\mathcal{R}$ are fully known, dynamic programming (DP) methods can be used to find the optimal policy π^*

Review of Session 2

Two Key Insights

- We can use the Bellman equation to estimate the value of a policy recursively

$$v_\pi(s) = \mathbb{E}_\pi\Big[r_{t+1} + \gamma v_\pi(s_{t+1}) \big| s_t = s\Big]$$

- We can iteratively improve a policy by making local improvements with respect to the value function

$$\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}} \mathcal{R}(s,a) + \gamma \textstyle\sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') v_\pi(s')$$

$$\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}} q(s,a)$$

New Twist - Incomplete MDP

Recall our definition of an MDP:

- A set of finite states $s \in \mathcal{S}$
- A set of finite actions $a \in \mathcal{A}$
- A model of the environment dynamics $\mathcal{T}(s_t, a_t, s_{t+1}) = \mathbb{P}(s_{t+1}|s_t, a_t)$
- A reward function $\mathcal{R}(s_t, a_t) = \mathbb{E}[r_{t+1}|s_t, a_t]$
- A discount factor $\gamma \in [0, 1]$

How can we can find an (approximately) optimal policy $\pi^*(s, a)$ when $\mathcal{T}(s_t, a_t, s_{t+1})$ and $\mathcal{R}(s_t, a_t)$ are **unknown**?

Model-Free RL

Model-Free RL

General idea: Estimate value function(s) and/or policies from **interaction experience**

Recall the solution we developed in Session 2 (when model is known)

Model-Free RL

General idea: Estimate value function(s) and/or policies from **interaction experience**

Recall the solution we developed in Session 2 (when model is known)

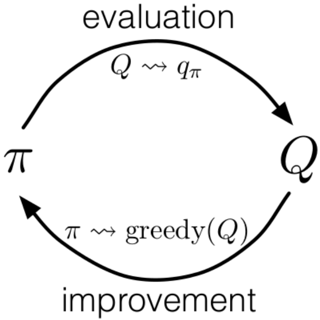


Figure: (Generalized) policy iteration

Model-Free RL

General idea: Estimate value function(s) and/or policies from **interaction experience**

Recall the solution we developed in Session 2 (when model is known)

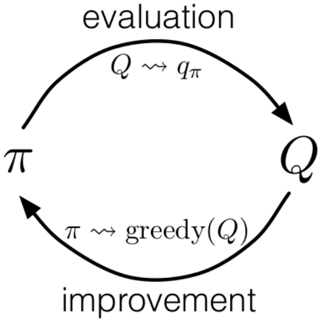


Figure: (Generalized) policy iteration

- Repeatedly alter $\hat{q}_\pi(s, a)$ to more closely approximate the *true* action-value function for π

Model-Free RL

General idea: Estimate value function(s) and/or policies from **interaction experience**

Recall the solution we developed in Session 2 (when model is known)

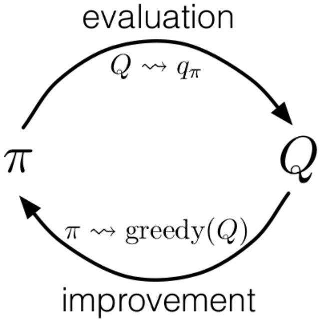


Figure: (Generalized) policy iteration

- Repeatedly alter $\hat{q}_\pi(s, a)$ to more closely approximate the *true* action-value function for π
- Repeatedly improve π with respect to the current $\hat{q}_\pi(s, a)$
  $\pi(s) = \arg\max_a \hat{q}_\pi(s, a)$

Review: Iterative Policy Evaluation

(when we have the model)

Given an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, we want to compute the *state-value* function for an arbitrary policy π.

Solution (intuition):

- Start with an arbitrary guess $v_0(s)$ that is an estimate of $v_\pi(s)$
- Improve estimate $v_i(s)$ by iteratively applying Bellman equation for all states until convergence

$$\underbrace{v_{i+1}(s)}_{\text{iteration i+1}} \leftarrow \underbrace{\sum_{a \in \mathcal{A}} \pi(a|s)\Big(\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') v_i(s')\Big)}_{\text{iteration i}}$$

- If I know the reward model $\mathcal{R}(s,a)$ and the transition model $\mathcal{T}(s,a,s')$, we can do this entirely in simulation
- What can we do if don't have any models?

Temporal Difference Learning

Goal: Given a policy π, learn $\hat{v}_\pi(s)$ from experience episodes

$$\{s_0, a_0, r_1, \ldots, s_T\} \sim \pi$$

Recall: State-value Bellman Equation

$$v(s_t) = \mathbb{E}_\pi \left[r_{t+1} + \gamma v_\pi(s_{t+1}) | s_t = s \right]$$

Approximation: At each time step t use *observed immediate* reward r_{t+1} and the estimated return $\hat{v}(s_{t+1})$ to update $\hat{v}(s_t)$

$$\hat{v}(s_t) \leftarrow \hat{v}(s_t) + \alpha \overbrace{[\underbrace{r_{t+1} + \gamma \hat{v}(s_{t+1})}_{\text{TD target}} - \hat{v}(s_t)]}^{\text{TD error}},$$

where α is a step-size paratemer.

We update our sample estimate $\hat{v}(s_t)$ in the direction of the TD error.

Temporal Difference Value Approximation

Algorithm 1: TD(0) learning for estimating v_π

Input : Policy π to evaluate, discount factor γ, step-size $\alpha \in (0, 1]$
Output: v_π
Initialize $v(s) = 0 \, \forall s \in \mathcal{S}$
Loop
    Sample episode $i = \{s_{i,0}, a_{i,0}, r_{i,1}, s_{i,1}, a_{i,1}, r_{i,2}, \dots, s_{i,T_i}\} \sim \pi$
    foreach $t \in \{0, \dots, T_i - 1\}$ **do**
        $s \leftarrow \mathrm{s}_{i,t}$
        $r \leftarrow \mathrm{r}_{i,t}$
        $s' \leftarrow \mathrm{s}_{i,t+1}$
        $v(s) \leftarrow v(s) + \alpha[r + \gamma v(s') - v(s)]$

Activity 1: Temporal Difference - Value Function Approximation

- Please open your Jupyter notebook environment and open the `LTASession3-Part1` notebook in the `Reinforcement_Learning` folder
- Take a look at Activity 1, Temporal Difference Learning - Policy Evaluation

Review: Optimal Policy Through Policy Iteration

A policy can be improved iif

$$\exists s \in \mathcal{S}, a \in \mathcal{A} \text{ such that } q_\pi(s, a) > q_\pi(s, \pi(a))$$

If this condition is met, how do we improve π?

Solution (intuition):

- Evaluate the policy π using *policy evaluation*
- Improve the policy by acting *greedily* with respect to $v_\pi(s)$ or $q_\pi(s)$

But now we will do policy evaluation using only data from our interaction with the environment.
What policy should we follow for collecting this interaction data?

Activity 2: ϵ-Greedy Policies

- Please open your Jupyter notebook environment and open the `LTASession3-Part1` notebook in the `Reinforcement_Learning` folder
- Take a look at Activity 2, Action Selection During Learning

Q-learning (1)

Off-line methods evaluate and improve a *target* policy $\pi(a|s)$ while using *behaviour* policy $\mu(a|s)$.

Q-learning is an off-policy method, where

- Target policy π is **greedy** with respect to $\hat{q}(s, a)$ (*deterministic)*
- The behaviour policy μ is e.g., ϵ-**greedy** with respect to $\hat{q}(s, a)$ (*stochastic*)

Q-learning target is then given by

$$q(s, a) \leftarrow q(s, a) + \alpha [r + \gamma \max_{a'} q(s', a') - q(s, a)]$$

Q-learning (2)

Algorithm 2: Q-learning algorithm

Input : Step-size $\alpha \in (0, 1]$, small $\epsilon > 0$

Output: $\hat{q}^*(s, a)$

Initialize $\hat{q}(s, a)$ arbitrarily $\forall s \in \mathcal{S}\ a \in \mathcal{A}$, $q(\text{terminal state}, \cdot) = 0$

Loop *for each episode*

- Initialize s
- **repeat**
  - Choose action a from s using ϵ-greedy policy derived from $\hat{q}(s, a)$
  - Take action a, observe r, s'
  - $\hat{q}(s, a) \leftarrow \hat{q}(s, a) + \alpha[r + \gamma \max_{a'} \hat{q}(s', a') - \hat{q}(s, a)]$
  - $s \leftarrow s'$
- **until** *s is terminal*

Activity 3: Q-Learning

- Please open your Jupyter notebook environment and open the `LTASession3-Part1` notebook in the `Reinforcement_Learning` folder
- Take a look at Activity 3, Q-Learning

Function Approximation:
Moving Away from Tabular Environments

Function Approximation

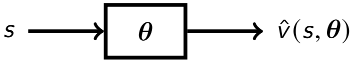
We have seen what to do when the model is **unknown**, what about continuous, high-dimensional environments (non-tabular cases)?

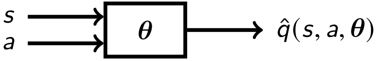
How can we scale up model-free methods for large environments?

Function Approximation

We have seen what to do when the model is **unknown**, what about continuous, high-dimensional environments (non-tabular cases)?

How can we scale up model-free methods for large environments?





$\hat{v}(s, \boldsymbol{\theta}) \approx v_\pi(s)$

$\hat{q}(s, a, \boldsymbol{\theta}) \approx q_\pi(s, a)$

How do we find the right parameters $\boldsymbol{\theta}$?

If we knew the true $q_\pi(s, a)$ this would be a standard supervised learning problem, we could find the best approximation by minimising:

$$J(\boldsymbol{\theta}) = \mathbb{E}_\pi[(q_\pi(s, a) - \hat{q}_\pi(s, a, \boldsymbol{\theta}))^2]$$

But RL only gives us access to rewards. What do we do in this case?

If we knew the true $q_\pi(s,a)$ this would be a standard supervised learning problem, we could find the best approximation by minimising:

$$J(\boldsymbol{\theta}) = \mathbb{E}_\pi[(q_\pi(s,a) - \hat{q}_\pi(s,a,\boldsymbol{\theta}))^2]$$

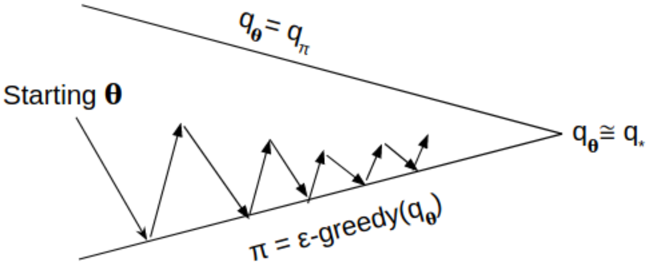
But RL only gives us access to rewards. What do we do in this case?

Intuition: Substitute a target for $q_\pi(s,a)$

- For TD, the target is the TD target $r_{t+1} + \gamma \hat{q}_\pi(s_{t+1}, a_{t+1}, \boldsymbol{\theta})$

$$\Delta\boldsymbol{\theta} = \alpha[\underbrace{r_{t+1} + \gamma \hat{q}_\pi(s_{t+1}, a_{t+1}, \boldsymbol{\theta})}_{\text{Target}} - \hat{q}_\pi(s_t, a_t, \boldsymbol{\theta})]\nabla_{\boldsymbol{\theta}} \hat{q}_\pi(s_t, a_t, \boldsymbol{\theta})$$

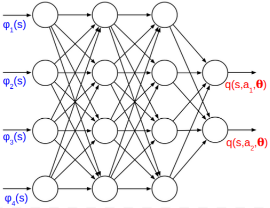
We apply the same generalized policy iteration algorithm we saw for the tabular-case.



Policy evaluation: Approximate policy evaluation $\hat{q}(\cdot,\cdot,\boldsymbol{\theta}) \approx q_\pi$ using TD
Policy improvement: ϵ-greedy policy improvement (exploration vs exploitation)

Function Approximation:

Neural Networks



Compute loss function (error) on forward pass

$$\mathcal{L}_i(\boldsymbol{\theta}_i) = [\underbrace{r + \gamma \max_{a'} q(s, a', \boldsymbol{\theta}_{i-1})}_{\text{Target}} - \overbrace{q(s, a, \boldsymbol{\theta}_i)}^{\text{Prediction}}]^2$$

When combined with function approximation, Q-learning is known to diverge due to:

- Correlation between samples (recall we have sequential non i.i.d data)
- Non-stationary targets
  - As the parameters $\boldsymbol{\theta}$ change, the target $r + \gamma \max_{a'} q(s, a', \boldsymbol{\theta})$ is also changing

What can we do in this case?

When combined with function approximation, Q-learning is known to diverge due to:

- Correlation between samples (recall we have sequential non i.i.d data)
- Non-stationary targets
  - As the parameters $\boldsymbol{\theta}$ change, the target $r + \gamma \max_{a'} q(s, a', \boldsymbol{\theta})$ is also changing

What can we do in this case? **DQN to the rescue**

Playing Atari with Deep Reinforcement Learning - V. Mnih, K. Kavakcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, NIPS DL Workshop, 2013.

Deep Q-learning (DQN) addresses both of these challenges by:

- Experience replay (replay buffer)
- Fixed q-targets

Deep Q-learning (DQN) addresses both of these challenges by:

- Experience replay (replay buffer)
- Fixed q-targets

How does experience replay work?

Deep Q-learning (DQN) addresses both of these challenges by:

- Experience replay (replay buffer)
- Fixed q-targets

How does experience replay work?

1. Store dataset $\mathcal{D}$ from prior experience

s_1, a_1, r_1, s_2
s_2, a_2, r_2, s_3
s_3, a_3, r_3, s_4
...
s_t, a_t, r_t, s_{t+1}

Function Approximation

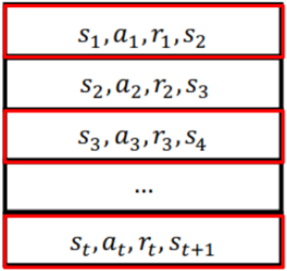
DQN

Deep Q-learning (DQN) addresses both of these challenges by:

- Experience replay (replay buffer)
- Fixed q-targets

How does experience replay work?

1. Store dataset $\mathcal{D}$ from prior experience
2. Sample a random batch from $\mathcal{D}$

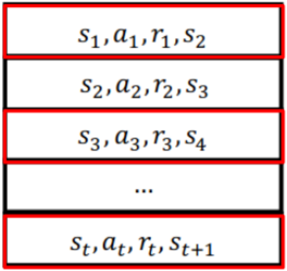


Deep Q-learning (DQN) addresses both of these challenges by:

- Experience replay (replay buffer)
- Fixed q-targets

How does experience replay work?

1. Store dataset $\mathcal{D}$ from prior experience
2. Sample a random batch from $\mathcal{D}$
3. Compute target value on sampled batch



Function Approximation

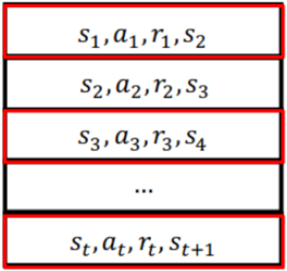
DQN

Deep Q-learning (DQN) addresses both of these challenges by:

- Experience replay (replay buffer)
- Fixed q-targets

How does experience replay work?

1. Store dataset $\mathcal{D}$ from prior experience
2. Sample a random batch from $\mathcal{D}$
3. Compute target value on sampled batch
4. Use stochastic gradient descent (SGD) to update network weights $\boldsymbol{\theta}$



To help improve stability, fix the **target weights** used in the calculation of $r + \gamma \max_{a'} q(s, a', \boldsymbol{\theta})$ for multiple updates.

How does the idea work in practice?

1. Define a *target* (with parameters $\boldsymbol{\theta}^-$) and a *policy* (with parameters $\boldsymbol{\theta}$) networks
2. Given a batch sampled from $\mathcal{D}$, compute target values using *target* network $r + \gamma \max_{a'} q(s, a', \boldsymbol{\theta}^-)$
3. Use SGD to update the *policy* network parameters

$$\boldsymbol{\theta_{i+1}} = \boldsymbol{\theta_i} + \alpha \Big[r + \gamma \max_{a'} q(s, a', \boldsymbol{\theta}^-) - q(s, a, \boldsymbol{\theta}_i) \Big] \nabla_{\boldsymbol{\theta_i}} q(s, a, \boldsymbol{\theta}_i)$$

4. Every C iterations $\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}_{i+C}$

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 $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    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 $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ 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

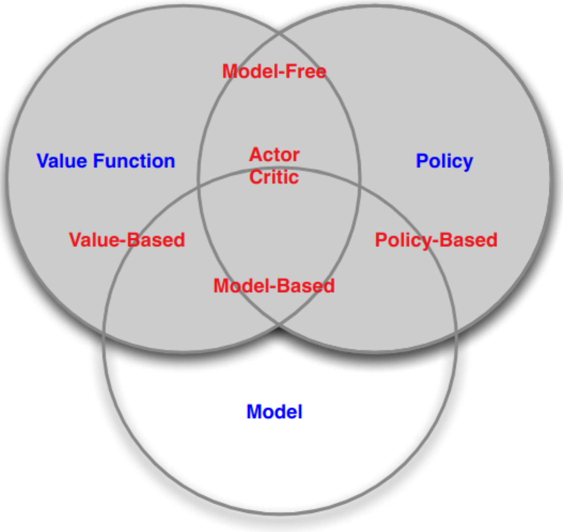
Post-lecture: DQN with Target Network

- Check out the `DeepRL_BasicDQN` and the `DeepRL_TargetDQN` and notebook (`Reinforcement_Learning` folder)

Summary

- We have introduced algorithms that can address some of the limitations of DP-based methods
  - Model-free methods can be used if model is **unknown**
  - Model-free based methods are online, **the agent actively interacts with the environment**
  - **Function approximation** allows to scale RL algorithms to larger (potentially continuous) domains
- We learned about the trade-off between exploration and exploitation
- We also covered on-policy and off-policy methods

Types of RL Algorithms



[2] Image courtesy of David Silver

Active RL topics

- Model-based vs. model-free methods
- Interacting with the real world may be dangerous
  - Sim2Real
  - Domain randomisation
  - Starting from good examples?
  - Safe RL
- How to specify the reward?
  - Reward shaping
  - Reward learning (e.g. inverse reinforcement learning)