02465: Introduction to reinforcement learning and control

Q-learning and deep-Q learning

Tue Herlau

DTU Compute, Technical University of Denmark (DTU)

DTU Compute
Department of Applied Mathematics and Computer Science

Lecture Schedule

Dynamical programming

1. The finite-horizon decision problem
   31 January
2. Dynamical Programming
   7 February
3. DP reformulations and introduction to Control
   14 February

Control

4. Discretization and PID control
   21 February
5. Direct methods and control by optimization
   28 February
6. Linear-quadratic problems in control
   7 March
7. Linearization and iterative LQR
   14 March

Reinforcement learning

8. Exploration and Bandits
   21 March
9. Policy and value iteration
   4 April
10. Monte-carlo methods and TD learning
    11 April
11. Model-Free Control with tabular and linear methods
    18 April
12. Eligibility traces and value-function approximations
    25 April
13. **Q-learning and deep-Q learning**
    2 May

Syllabus: `https://02465material.pages.compute.dtu.dk/02465public`
Help improve lecture by giving feedback on DTU learn

Reading material:

- [SB18, Chapter 6.7-6.9; 8-8.4; 16-16.2; 16.5; 16.6]

Learning Objectives

- Double-Q learning
- Dyna-Q and the replay buffer
- Deep-Q learning

Housekeeping

- Unofficial exam Q/A about one week before the exam (the 20th?). Please put wishes on blackboard.
- I have added a survey on the course (what went well/ less well /what can be improved). You can find it in the menu to the right on DTU Learn.
- I have updated the video on preparing for the exam, `https://www2.compute.dtu.dk/courses/02465/exam.html`, and uploaded solutions to the previous exams.
- Exam is planned to be in English as last year (only one language). Please let me know before Tuesday the 7th if this is not acceptable.
- Test exam at `https://eksamen.dtu.dk/studerende/proeve/7482/tilmeld/3a1b13368489ef57c103c1e4642d6ff2` (Hopefully this works!)

Recap: Q-learning

- **Bellman optimality condition:**

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a\right]$$

- **Theorem:** q_* satisfies the above recursions if (and only if) it corresponds to the **optimal value function**
- **Value iteration**: Replace q_* arbitrary Q and iterate:

$$Q(s,a) \leftarrow \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') | S_t = s, A_t = a\right]$$

- **Theorem:** Q will converge to q_*
- **Q-learning:** Given $(S_t, A_t, R_{t+1}, S_{t+1}) = (s, a, r, s')$ transition, update

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

Uses that red expression is a **biased** but **consistent** estimate of Q

Q-learning

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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop 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

Convergence of Q-learning

- All s, a pairs visited infinitely often
- Robbins-Monro sequence of step-sizes α_t

$$\sum_{t=1}^{\infty} \alpha_t = \infty, \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

Learning and planning

- Value iteration uses a **model** of the environment to **plan** a policy

$$Q(s,a) \leftarrow \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') | S_t = s, A_t = a\right]$$

- Q-learning uses **samples** from the environment (s, a, r, s') to **learn** a policy

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

- Both uses value functions and backups
- Can we combine these ideas?

Learning and planning

- A **distributional model** is an estimate of the MDP $p(s', r|s, a)$
- A **sample model** is a mechanism to generate samples (s, a, r, s') from the MDP (**weaker assumption**)
- Idea: Learn sample model and use it to improve value function by regular backups
- Allows re-use of data for faster convergence (**sample efficiency**)

Tabular planning

Random-sample one-step tabular Q-planning

Loop forever:

1. Select a state, $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(S)$, at random
2. Send S, A to a sample model, and obtain
   a sample next reward, R, and a sample next state, S'
3. Apply one-step tabular Q-learning to S, A, R, S':
   $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$

Dyna-Q planning

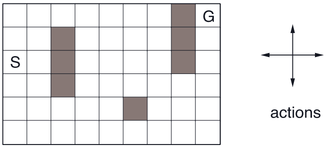
Tabular Dyna-Q

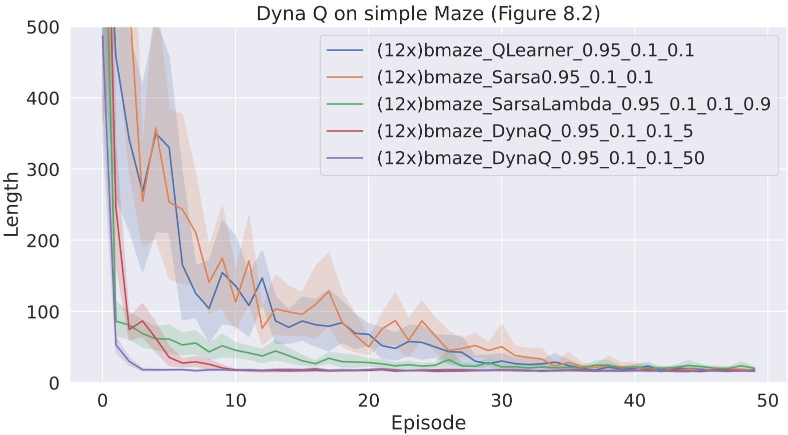
Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:

(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow \varepsilon\text{-greedy}(S, Q)$
(c) Take action A; observe resultant reward, R, and state, S'
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
(f) Loop repeat n times:

$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previously taken in S
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Dyna-Q on deterministic Maze environment





`lecture_13_Q_maze.py`, `lecture_13_dyna_q_5_maze.py`, `lecture_13_sarsa_lambda_maze.py`

Dyna-Q implementation

Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:

(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow \varepsilon$-greedy(S, Q)
(c) Take action A; observe resultant reward, R, and state, S'
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
(f) Loop repeat n times:

$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previously taken in S
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

- The **model** is simply a list of experience (a **replay buffer**)
- Deterministic assumption not used

Double-Q learning

- Target for the Q-values can be considered noisy (random)

$$r + \max_{a'} Q(s', a').$$

- Q-update is

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \max_{a'} Q(s', a') - Q(s, a) \right)$$

- By chance some of the $Q(s', a')$ values are likely to be unusually large
- This leads to over-estimate $Q(s, a)$:

$$\mathbb{E}[\max(X_1, X_2)] \geq \max(\mathbb{E}[X_1], \mathbb{E}[X_2])$$

- **Conclusion:**
  - Q-values systematically over-estimated
  - the worse the estimate of a state, the more we will prefer it

Double-Q learning

Given transition $(S_t, A_t, R_{t+1}, S_{t+1}) = (s, a, r, s')$

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') \, Q\left(s', \arg\max_{a} Q(s', a)\right) Q_2\left(s', \arg\right.\right.$$

- Where Q_2 is another Q-function
- Q_2 is independent of Q which avoids systematic over-estimation

Double-Q learning

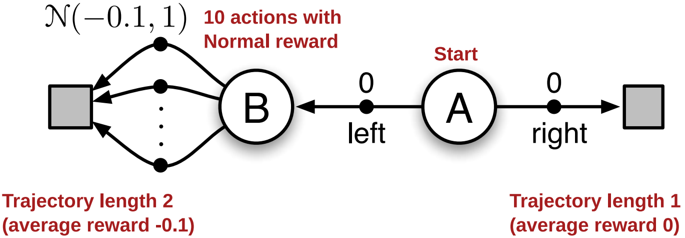
Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

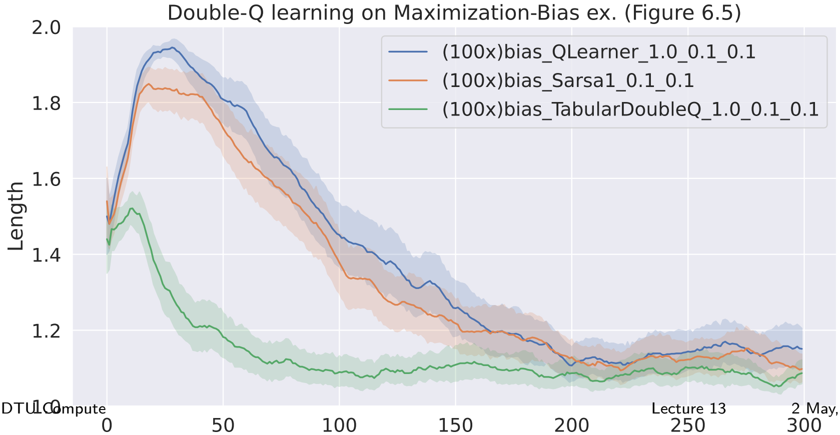
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:
      $$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S', a)\big) - Q_1(S, A)\Big)$$
    else:
      $$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S', a)\big) - Q_2(S, A)\Big)$$
    $S \leftarrow S'$
  until S is terminal

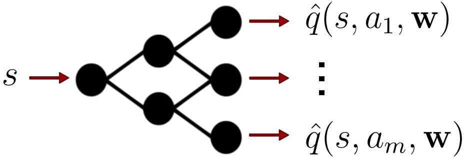
- Twice as slow to learn

Double-Q learning on bias-example environment





Q-learning with function approximators



- We want an approximation of the Q-values $Q(s, a)$
- Assume $\boldsymbol{y} = \hat{q}_\phi(s)$ is a vector of dimension $|\mathcal{A}|$ such that

$$y_a \approx Q(s, a)$$

  is our approximation of the Q-value
- In practice, $\hat{q}_\phi : \mathbb{R}^d \mapsto \mathbb{R}^{|\mathcal{A}|}$ is a deep network
  - Input-dimension is dimension of each state $s \in \mathcal{S} = \mathbb{R}^d$
  - Output dimension $|\mathcal{A}|$

Q-learning with function approximators

Regular Q-learning:

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

Regular Q-learning with function approximators

- Given $(S_t, A_t, R_{t+1}, S_{t+1}) = (s, a, r, s')$ update:

$$\phi \leftarrow \phi + \alpha \left(r + \gamma \max_{a'} \hat{q}_\phi(s',a') - \hat{q}_\phi(s,a) \right) \nabla_\phi \hat{q}_\phi(s,a)$$

- Defining $y = r + \gamma \max_{a'} \hat{q}_\phi(s',a')$ this can be written as

$$\phi \leftarrow \phi - \alpha \frac{1}{2} \nabla_\phi \left(y - \hat{q}_\phi(s,a) \right)^2$$

Fitted Q-iteration algorithm

Fitted Q-iteration algorithm

1. At step t observe $(s_t, a_t, r_{t+1}, s_{t+1})$
2. $y_t = r_{t+1} + \gamma \max_{a'} \hat{q}_\phi(s_{t+1}, a')$
3. Repeat fit step one or more times:
   - $\phi \leftarrow \phi - \alpha \nabla_\phi \left[\frac{1}{2} \left(y_t - \hat{q}_\phi(s_t, a_t) \right)^2 \right]$

- The use of a **single** sample gives a high variance in the gradient estimate
- The samples are only used once

Q-learning with a replay buffer

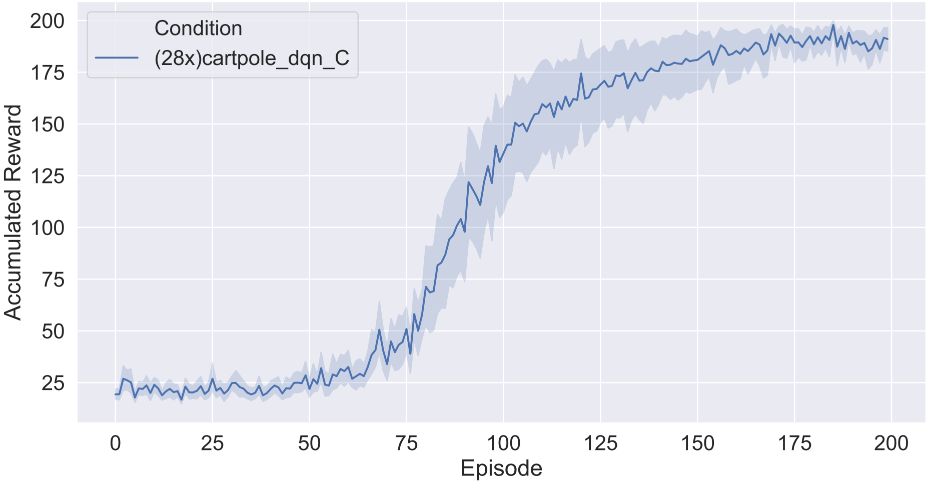
Initialize a **replay buffer** $\mathcal{B}$

Q-learning with a replay buffer

1. At step t observe $(s_t, a_t, r_{t+1}, s_{t+1})$ and add it to $\mathcal{B}$
2. Repeat K times:
   1. Sample a **batch** $(s_i, a_i, r_i, s_i')_{i=1}^{B}$ from $\mathcal{B}$
   2. Set $y_i = r_i + \gamma \max_{a'} \hat{q}_\phi(s_i', a')$
   3. $\phi \leftarrow \phi - \alpha \nabla_\phi \left[\frac{1}{2B} \sum_{i=1}^{B} (y_i - \hat{q}_\phi(s_i, a_i))^2 \right]$

- Similar to dyna-Q
- Lower gradient variance, quicker convergence
- Replay buffer should be large (thousands to a few millions)
- **You can implement this in the exercises**

Basic deep Q learning on Cartpole



An issue with deep Q learning

- Consider the target

  1. $y = r_{t+1} + \gamma \max_{a'} \hat{q}_\phi(s_{t+1}, a')$
  2. $\phi \leftarrow \phi - \alpha \nabla_\phi \left[\frac{1}{2} \left(y - \hat{q}_\phi(s, a) \right)^2 \right]$

- We don't compute gradients through y
- This is to a great extend why deep-Q sometimes do not converge: We adapt towards y, without taking into account that y changes during the adaption
- **Idea 1:** Use an alternative weight network ϕ'

$$y = r_{t+1} + \gamma \max_{a'} \hat{q}_{\phi'}(s_{t+1}, a')$$

- **Idea 2:** Let ϕ' be an old version of ϕ

Deep-Q learning

Initialize $\mathcal{B}$ and make a copy $\phi' \leftarrow \phi$ of the weights

Deep-Q learning

1. At step t observe $(s_t, a_t, r_{t+1}, s_{t+1})$ and add it to $\mathcal{B}$
2. Repeat K times:
    1. Sample a batch $(s_i, a_i, r_i, s'_i)_{i=1}^B$ from $\mathcal{B}$
    2. Set $y_i = r_i + \gamma \max_{a'} \hat{q}_{\phi'}(s'_i, a')$
    3. $\phi \leftarrow \phi - \alpha \nabla_\phi \left[\frac{1}{2B} \sum_{i=1}^B (y_i - \hat{q}_\phi(s_i, a_i))^2 \right]$
3. Update $\phi' \leftarrow \phi' + \tau(\phi - \phi')$ (Slow changes, e.g. $\tau = 0.08$ or less)

- Can we also address the over-estimation problem of the Q-values?

Double-Q learning

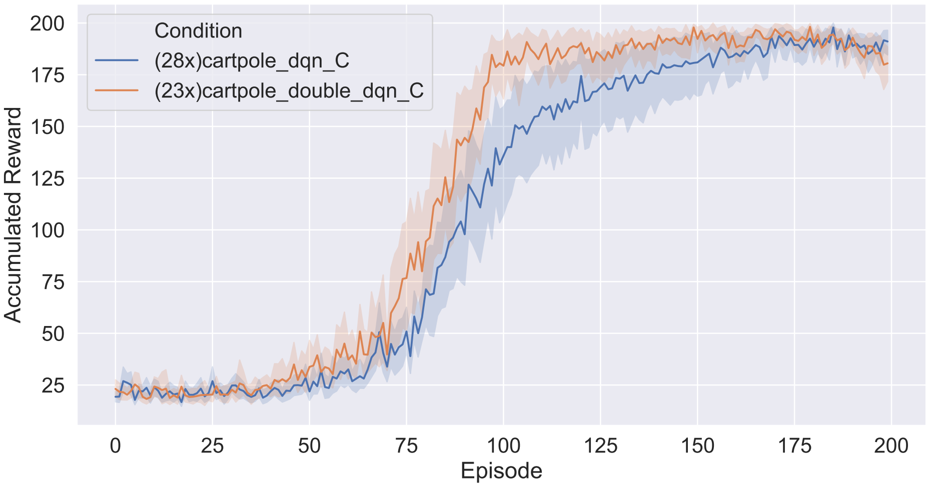
Initialize $\mathcal{B}$ and make a copy $\phi' \leftarrow \phi$ of the weights

Double-Q learning

1. At step t observe $(s_t, a_t, r_{t+1}, s_{t+1})$ and add it to $\mathcal{B}$
2. Repeat K times:
   1. Sample a batch $(s_i, a_i, r_i, s_i')_{i=1}^{B}$ from $\mathcal{B}$
   2. Set $y_i = r_i + \gamma \hat{q}_{\phi'}(s_i', \arg\max_{a'} \hat{q}_\phi(s', a'))$
   3. $\phi \leftarrow \phi - \alpha \nabla_\phi \left[\frac{1}{2B} \sum_{i=1}^{B} (y_i - \hat{q}_\phi(s_i, a_i))^2 \right]$
3. Update $\phi' \leftarrow \phi' + \tau(\phi - \phi')$

- Double-Q: Select actions according to ϕ, but evaluate according to ϕ'
- **We will implement this in the exercises**

Double-deep Q learning on Cartpole



The buffer

The buffer is a list with a sample function

```python
# deepq_agent.py
self.memory = BasicBuffer(replay_buffer_size) if buffer is None else buffer
self.memory.push(s, a, r, sp, done) # save current observation
""" First we sample from replay buffer. Returns numpy Arrays of dimension
> [self.batch_size] x [...]]
for instance 'a' will be of dimension [self.batch_size x 1].
"""
s,a,r,sp,done = self.memory.sample(self.batch_size)
```

First dimension is batch dimension

$$(\texttt{batch\_size} \times d)$$

The network

Implemented in separate class

```
# irlc/ex13/lecture_12_examples.py
    # Initialize a network class
    self.Q = Network(env, trainable=True) # initialize the network
    """ Assuming s has dimension [batch_dim x d] this returns a float numpy Array
    array of Q-values of [batch_dim x actions], such that qvals[i,a] = Q(s_i,a) """
    qvals = self.Q(s)
    actions = env.action_space.n # number of actions
    """ Assume we initialize target to be of dimension [batch_dim x actions]
    > target = [batch_dim x actions]
    The following function will fit the weights in self.Q by minimizing
    > ||self.Q(s)-target||^2
    (averaged over Batch dimension) using one step of gradient descent
    """
    self.Q.fit(s, target)
```

I.e. select `target` appropriately to implement loss

$$\frac{1}{B}\sum_{i=1}^{B}\left(\hat{q}_\phi(s_i, a_i) - y_i\right)^2$$

The network (for double-Q)

```
# irlc/ex13/lecture_12_examples.py
    self.Q2 = Network(env, trainable=True)
    """ Update weights in self.Q2 (target, phi') towards those in Q (source, phi)
    with a factor of tau. tau=0 is no change, tau=1 means overwriting weights
    (useful for initialization) """
    self.Q2.update_Phi(Q2, tau=0.1)
```

Updates weights ϕ' in `Q2` towards ϕ in `Q`

$$\phi' = \phi' + \tau(\phi - \phi')$$

Q-learning, additional tricks

- Parameters: Decrease exploration rate ε_t and learning rate α_t through training
- Networks
  - Clip gradients or use Huber loss
  - Batch normalization
  - Tune parameters; linear $\rightarrow$ shallow $\rightarrow$ deep
- Methods:
  - Double-Q learning always a good idea
  - Replay buffer always a good idea
  - Prioritizing samples (PER) improves convergence speed
  - Check out **Rainbow** for current(ish) state of the art(ish) [HMVH$^+$18]
- Lots of training and results highly variable across seeds

FIN!

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver.
Rainbow: Combining improvements in deep reinforcement learning.
In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Richard S. Sutton and Andrew G. Barto.
Reinforcement Learning: An Introduction.
The MIT Press, second edition, 2018.
(Freely available online).