Reinforcement learning

Master CPS, Year 2 Semester 1

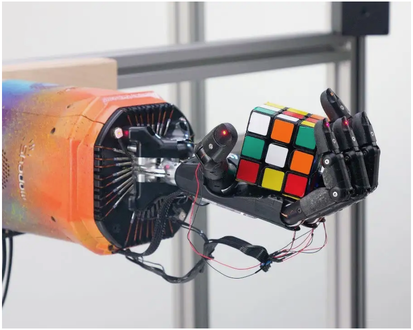
Lucian Buşoniu, Florin Gogianu

Part V

Online approximate reinforcement learning

Recap: need for approximation

- In real applications, x, u often **continuous** (or discrete with very many values)



- Tabular representation **impossible**
- **Approximate** functions of interest $Q(x, u)$, $V(x)$, $h(x)$

Recap: Part 4 – Offline approximate DP and RL

Given either:
– a model f, ρ
– data (x_s, u_s, r_s, x'_s), $s = 1, \ldots, n_s$

1. find an approximate solution $\widehat{Q}(x, u)$, $\widehat{h}(x)$, etc.
2. control the system using the solution found

Algorithms discussed:

- Q-iteration with interpolation
- Fitted Q-iteration

Part V in plan

- Reinforcement learning problem
- Optimal solution
- Exact dynamic programming
- Exact reinforcement learning
- Approximation techniques
- Approximate dynamic programming
- Offline approximate reinforcement learning
- **Online approximate reinforcement learning**

Algorithm landscape

By model usage:

- Model-based: f, ρ known a priori
- Model-free: f, ρ unknown (reinforcement learning)

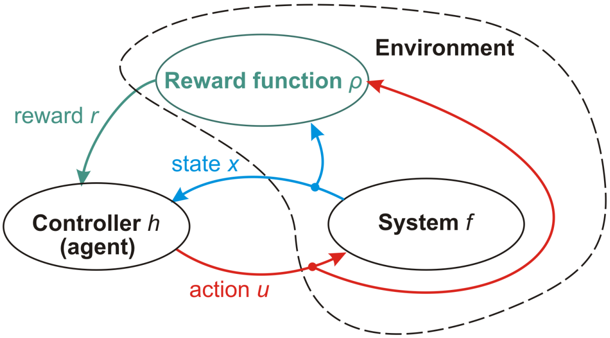
By interaction level:

- Offline: algorithm runs in advance
- Online: algorithm runs with the system

Exact vs. approximate:

- Exact: x, u small number of discrete values
- Approximate: x, u continuous (or many discrete values)

RL principle



We are now truly following, online, the RL interaction scheme

Many algorithms exist; we discuss just a few

Contents of part V

Recall classical SARSA

SARSA with ε-greedy

for each trajectory **do**
  initialize x_0

$$u_0 = \begin{cases} \arg\max_u Q(x_0, u) & \text{w.p. } (1 - \varepsilon_0) \\ \text{unif. random} & \text{w.p. } \varepsilon_0 \end{cases}$$

  repeat at each step k
    apply u_k, measure x_{k+1}, receive r_{k+1}

$$u_{k+1} = \begin{cases} \arg\max_u Q(x_{k+1}, u) & \text{w.p. } (1 - \varepsilon_{k+1}) \\ \text{unif. random} & \text{w.p. } \varepsilon_{k+1} \end{cases}$$

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot [r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)]$$

  until trajectory finished
end for

Recall derivation of SARSA (on-policy) update

- Update:

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k[r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)]$$
$$= Q(x_k, u_k) + \alpha_k[\hat{R}_k - Q(x_k, u_k)]$$

- $\hat{R}_k$ is a bootstrapped estimate (which exploits the Bellman equation) of the Monte-Carlo return R_k from (x_k, u_k) under the current policy h
- R_k is itself a sample of $Q^h(x_k, u_k)$, so in the end we are running an estimated version of the ideal update:

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k[Q^h(x_k, u_k) - Q(x_k, u_k)]$$

Stochastic gradient descent for approximate case

- Extend this idea to a parametric approximator $\hat{Q}(x, u; \theta)$
- We can no longer update Q directly, instead we update θ using **stochastic gradient descent (SGD)** on the square approximation error:

$$\theta_{k+1} = \theta_k - \frac{1}{2}\alpha_k \nabla_\theta \left[Q^h(x_k, u_k) - \widehat{Q}(x_k, u_k; \theta_k) \right]^2$$
$$= \theta_k + \alpha_k \nabla_\theta \widehat{Q}(x_k, u_k; \theta_k) \left[Q^h(x_k, u_k) - \widehat{Q}(x_k, u_k; \theta_k) \right]$$

- Replace $Q^h(x_k, u_k)$ by Monte Carlo sample R_k, then R_k by its bootstrapped estimate $\hat{R}_k = r_{k+1} + \gamma \widehat{Q}(x_{k+1}, u_{k+1}; \theta_k)$:

$$\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta \widehat{Q}(x_k, u_k; \theta_k) \left[r_{k+1} + \gamma \widehat{Q}(x_{k+1}, u_{k+1}; \theta_k) - \widehat{Q}(x_k, u_k; \theta_k) \right]$$

Semigradient

$$\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta \widehat{Q}(x_k, u_k; \theta_k) \left[r_{k+1} + \gamma \widehat{Q}(x_{k+1}, u_{k+1}; \theta_k) - \widehat{Q}(x_k, u_k; \theta_k) \right]$$

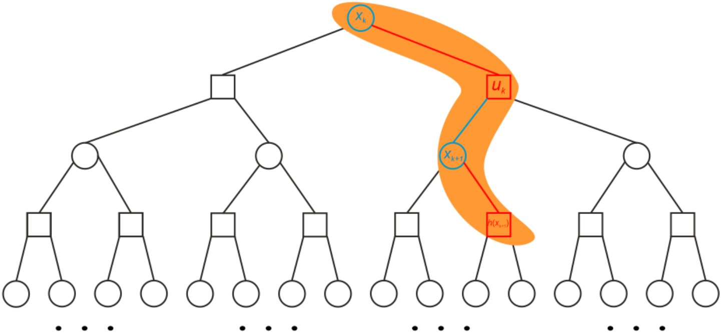
- The final update is not a full gradient descent, because $\hat{R}_k$ depends on θ_k via $\widehat{Q}(x_{k+1}, u_{k+1}; \theta_k)$, but only the second term of the error is differentiated!

$\Rightarrow$ Such methods are called **semigradient**.

- The term in squared brackets is an **approximate temporal difference**.

Illlustration

Graphical illustration is similar to the classical case:



but now approximation requires use of gradients

Objective and comparison to fitted methods

$$\theta_{k+1} \leftarrow \theta_k + \alpha_k \nabla_\theta \widehat{Q}(x_k, u_k; \theta_k) \left[Q^h(x_k, u_k) - \widehat{Q}(x_k, u_k; \theta_k) \right]$$

minimizes the following objective under the distribution arising from (x_k, u_k) samples:

$$\mathcal{L}_{\text{eval}} = \mathrm{E}\left\{ \left[Q^h(x, u) - \widehat{Q}(x, u; \theta) \right]^2 \right\}$$

- Compare to fitted Q-iteration objective, where the distribution is implicitly also that of the samples:

$$\sum_{s=1}^{n_\mathrm{s}} \left[\hat{R}_s - \widehat{Q}(x_s, u_s; \theta) \right]^2$$

- If $\sum_{k=0}^{\infty} \alpha_k^2$ is finite and $\sum_{k=0}^{\infty} \alpha_k \to \infty$, SGD converges to a local minimum of $\mathcal{L}_{\text{eval}}$.
- This still holds when $Q^h(x_k, u_k)$ is replaced by Monte Carlo R_k, but not anymore for bootstrapped $\hat{R}_k$, due to the semigradient updates!

Semigradient, approximate SARSA

Approximate SARSA

for each trajectory **do**
  initialize x_0
  choose u_0 (e.g., ε-greedy from $Q(x_0, \cdot; \theta_0)$)
  repeat at each step k
    apply u_k, measure x_{k+1}, receive r_{k+1}
    choose u_{k+1} (e.g., ε-greedy from $Q(x_{k+1}, \cdot; \theta_k)$)

$$\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta \widehat{Q}(x_k, u_k; \theta_k) \cdot \left[r_{k+1} + \gamma \widehat{Q}(x_{k+1}, u_{k+1}; \theta_k) - \widehat{Q}(x_k, u_k; \theta_k) \right]$$

  until trajectory finished
end for

Exploration is of course necessary in the approximate case as well.

1. Approximate TD methods
   - Approximate SARSA
   - Approximate Q-learning
   - Maximization and discussion
2. Policy gradient
3. Outlook

Semigradient Q-learning update

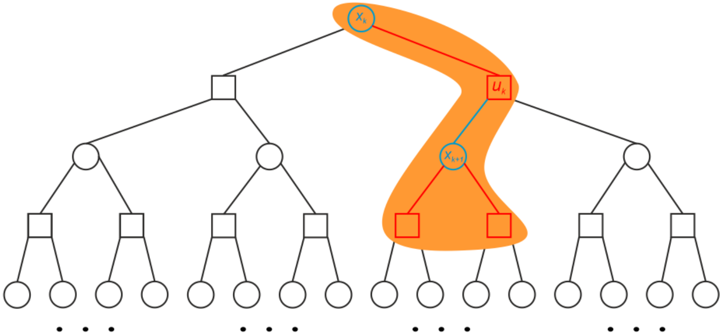
Recall classical Q-learning update:

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k [r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)]$$
$$\approx Q(x_k, u_k) + \alpha_k [Q^*(x_k, u_k) - Q(x_k, u_k)]$$

In the approximate case, perform gradient descent:

$$\theta_{k+1} = \theta_k - \frac{1}{2}\alpha_k \nabla_\theta \left[Q^*(x_k, u_k) - \widehat{Q}(x_k, u_k; \theta_k) \right]^2$$
$$= \theta_k + \alpha_k \nabla_\theta \widehat{Q}(x_k, u_k; \theta_k) \left[Q^*(x_k, u_k) - \widehat{Q}(x_k, u_k; \theta_k) \right]$$
$$\approx \theta_k + \alpha_k \nabla_\theta \widehat{Q}(x_k, u_k; \theta_k) \cdot \left[r_{k+1} + \gamma \max_{u'} \widehat{Q}(x_{k+1}, u'; \theta_k) - \widehat{Q}(x_k, u_k; \theta_k) \right]$$

Illlustration



Semigradient, approximate Q-learning

Approximate Q-learning

for each trajectory **do**
  initialize x_0
  repeat at each step k
    choose u_k (e.g., ε-greedy from $Q(x_k, \cdot; \theta_k)$)
    apply u_k, measure x_{k+1}, receive r_{k+1}

$$\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta \widehat{Q}(x_k, u_k; \theta_k) \cdot \left[r_{k+1} + \gamma \max_{u'} \widehat{Q}(x_{k+1}, u'; \theta_k) - \widehat{Q}(x_k, u_k; \theta_k) \right]$$

  until trajectory finished
end for

1. Approximate TD methods
   - Approximate SARSA
   - Approximate Q-learning
   - Maximization and discussion
2. Policy gradient
3. Outlook

Recall: Maximization

Solution 1: Implicit greedy policy

Solution 2: Explicitly represented (approximate) policy

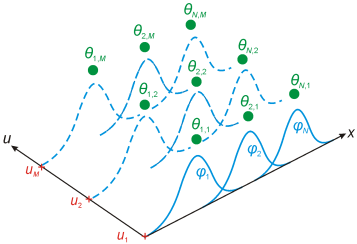
Maximization in approximate TD methods

- Greedy actions are computed on demand from $\widehat{Q}$:

$$\dots \color{red}{\arg\max_u} \widehat{Q}(x, u; \theta) \dots$$

⇒ Solution 1: The policy is implicitly represented

- Q-function approximator must ensure **efficient solution for** $\arg\max$
- Ex. discrete actions & features in x



Demo: robot walking (E. Schuitema)

Method: Approximate Q-learning
Approximator: Tile coding

Discussion of approximate TD methods

- Convergence guaranteed for **modified versions**
- Low complexity
- Exploration and learning rates must be **carefully tuned** for all methods
- Just like in the classical case, approximate TD methods learn slowly, so they must be **accelerated**
- Experience replay and n-step returns are nearly directly applicable (the latter for SARSA, but can be extended to off-policy Q-learning)

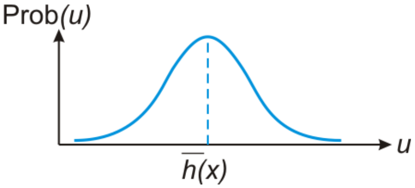
Policy representation

- Type 2: Policy **explicitly approximated**
- Recall advantages: easier to handle continuous actions, prior knowledge
- For example, feature-based representation:

$$\bar{h}(x;\mu) = \sum_{i=1}^{n} \phi_i(x)\mu_i$$

Policy with exploration

- Online RL $\Rightarrow$ policy gradient must explore



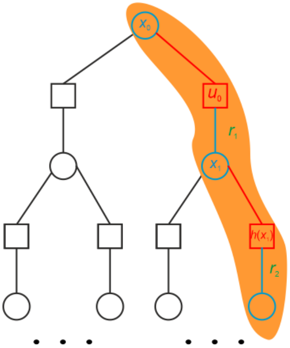
- **Gaussian exploration** applies u in x with probability:

$$P(u|x) = \mathcal{N}(\overline{h}(x;\mu), \sigma) =: \widehat{h}(x, u; \vartheta)$$

with ϑ containing μ as well as the covariances in matrix σ

- So a stochastic policy is represented, directly including random exploration in the parameterization

Trajectory



- Trajectory $\tau := (x_0, u_0, \dots, x_k, u_k, \dots)$ generated with $\widehat{h}$; and resulting rewards $r_1, \dots, r_{k-1}, \dots$
- Take deterministic MDP for simplicity. Return along the trajectory:

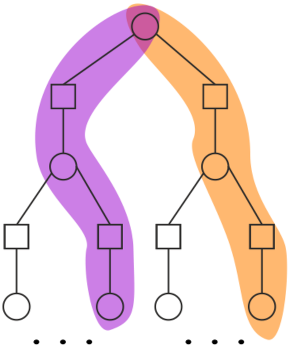
$$R(\tau) = \sum_{k=0}^{\infty} \gamma^k r_{k+1} = \sum_{k=0}^{\infty} \gamma^k \rho(x_k, u_k)$$

- Probability of trajectory τ under policy parameters ϑ:

$$P_\vartheta(\tau) = \prod_{k=0}^{\infty} \widehat{h}(x_k, u_k; \vartheta)$$

where $x_{k+1} = f(x_k, u_k)$

Objective



Take x_0 fixed, for simplicity

Objective

Maximize expected return from x_0 under policy $\widehat{h}(\cdot, \cdot; \vartheta)$:

$$J_\vartheta := \mathrm{E}_\vartheta \{R(\tau)\} = \int R(\tau) P_\vartheta(\tau) d\tau$$

Main idea

Gradient ascent on J_ϑ:

$$\vartheta \leftarrow \vartheta + \alpha \nabla_\vartheta \boldsymbol{J}_\vartheta$$

Gradient derivation

$$\begin{aligned}
\nabla_\vartheta J_\vartheta &= \int R(\tau)\nabla_\vartheta P_\vartheta(\tau)d\tau \\
&= \int R(\tau)P_\vartheta(\tau)\nabla_\vartheta \log P_\vartheta(\tau)d\tau \\
&= \mathrm{E}_\vartheta\left\{R(\tau)\nabla_\vartheta \log\left[\prod_{k=0}^{\infty}\widehat{h}(x_k,u_k;\vartheta)\right]\right\} \\
&= \mathrm{E}_\vartheta\left\{R(\tau)\sum_{k=0}^{\infty}\nabla_\vartheta \log\widehat{h}(x_k,u_k;\vartheta)\right\}
\end{aligned}$$

Where we:

- used “likelihood ratio trick” $\nabla_\vartheta P_\vartheta(\tau) = P_\vartheta(\tau)\nabla_\vartheta \log P_\vartheta(\tau)$
- replaced integral by expectation, and substituted $P_\vartheta(\tau)$
- replaced log of product by sum of logs

Gradient implementation

- Many methods exist to estimate gradient, based e.g. on Monte-Carlo
- E.g. REINFORCE uses current policy to execute n_τ sample trajectories, each of finite length K, and estimates:

$$\widehat{\nabla_\vartheta J_\vartheta} = \frac{1}{n_\tau} \sum_{s=1}^{n_\tau} \left[\sum_{k=0}^{K-1} \gamma^k r_{s,k} \right] \left[\sum_{k=0}^{K-1} \nabla_\vartheta \log \widehat{h}(x_{s,k}, u_{s,k}; \vartheta) \right]$$

  (with possible addition of a baseline to reduce variance)
- Compare with exact formula:

$$\nabla_\vartheta J_\vartheta = \mathrm{E}_\vartheta \left\{ R(\tau) \sum_{k=0}^{\infty} \nabla_\vartheta \log \widehat{h}(x_k, u_k; \vartheta) \right\}$$

- Gradient $\nabla_\vartheta \log \widehat{h}$ preferably computable in closed-form

Power-assisted wheelchair (with Feng et al.)

- Hybrid power source: human and battery
- Goal: follow reference velocity, optimizing assistance to:
  - (i) attain desired user fatigue level
  - (ii) minimize battery usage
- Challenge: user has **unknown dynamics**

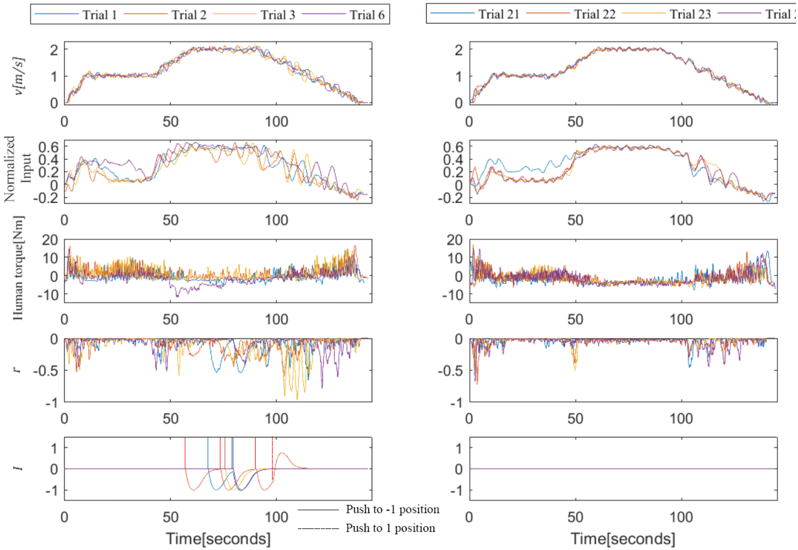
PAW: Experiment setup

- User sets velocity, pulls/pushes joystick when too tired/wants more exercise
- Reward penalizes velocity error, joystick signal I, and assistance magnitude (to save energy)

$$r = -w_1(v - v_{\mathrm{ref}})^2 - w_2 I^2 - w_2 u^2$$

- PI-type control with gains tuned by policy gradient (POWER)

PAW: Results


Trial 1
Trial 2
Trial 3
Trial 6
Trial 21
Trial 22
Trial 23
v[m/s]
Normalized Input
Human torque[Nm]
r
I
Push to -1 position
Push to 1 position
Time[seconds]
Time[seconds]

Open problems

RL research is **ongoing**

Open problems:

- Safety and stability guarantees
- States that cannot be measured (output feedback)
- Exploration strategies
- Multi-agent systems
- Multi-task learning

Deep reinforcement learning

A way to handle high-dimensional variables *when they are images* (or image-like); or relatively high-dimensional variables ($\sim$10) when they are numerical.

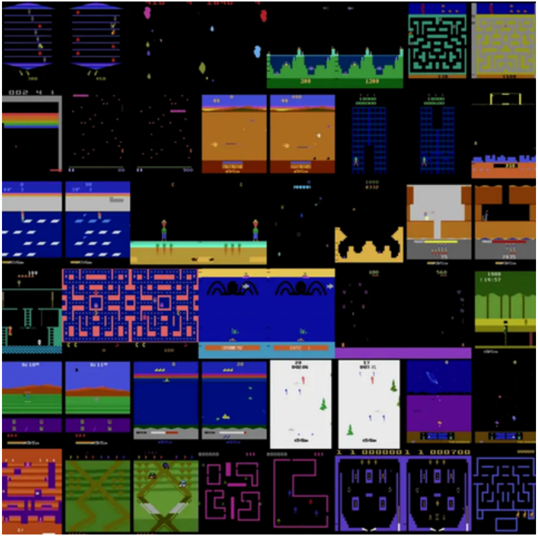
Deep Q-networks, DQN (DeepMind)

- Q-function represented via a deep neural network using e.g. convolutional layers to process images
- All data added to a replay buffer
- Network trained by SGD to reduce temporal differences, like semigradient Q-learning...

  ... but on mini-batches of transitions from the replay buffer, like fitted Q-iteration

$\Rightarrow$ algorithm combines online and offline approximate RL

Deep Q-networks, DQN (DeepMind)



Key terms in this part

- stochastic gradient descent
- semigradient
- approximate temporal difference
- policy gradient
- likelihood ratio trick

wiki

Exercises

1. Derive semigradient updates of the parameters θ to approximate V^h and V^*.
2. Generalize the derivation of the semigradient update in SARSA to n-step returns.
3. Try to derive a full gradient descent formula for SARSA, which takes into account the dependence of the bootstrapped estimate on the parameter vector.