Overview of reinforcement learning

Lucian Buşoniu
Technical University of Cluj-Napoca, Romania

Timişoara Workshop on Machine Learning
23 February 2019

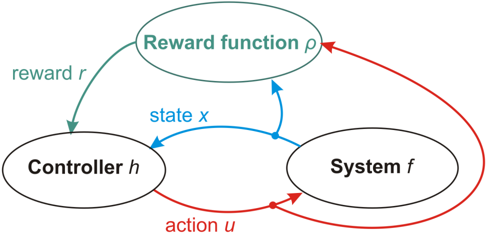
Reinforcement learning (RL)

Learn a sequential decision policy

to optimize the cumulative performance

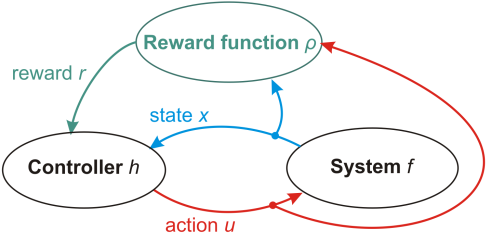
of an unknown system

RL framework



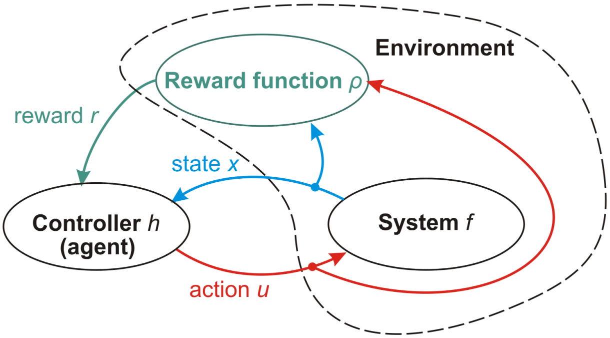
- At each step k, observe state x_k, apply action u_k
- System evolves with dynamics $x_{k+1} = f(x_k, u_k)$ (stochastic version exists)
- Receive reward $r_{k+1} = \rho(x_k, u_k)$, immediate performance
- **Objective:** Find policy $u_k = h(x_k)$ to maximize long-term return: $\sum_{k=0}^{\infty} \gamma^k r_{k+1}$, $\forall x_0$; $\gamma \in (0, 1)$ discount

RL framework



- At each step k, observe state x_k, apply action u_k
- System evolves with dynamics $x_{k+1} = f(x_k, u_k)$ (stochastic version exists)
- Receive reward $r_{k+1} = \rho(x_k, u_k)$, immediate performance
- **Objective:** Find policy $u_k = h(x_k)$ to maximize long-term return: $\sum_{k=0}^{\infty} \gamma^k r_{k+1}$, $\forall x_0$; $\gamma \in (0, 1)$ discount

AI view

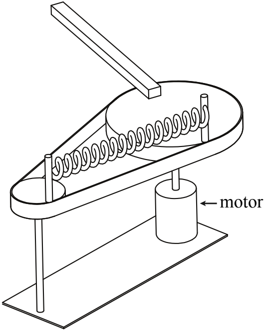


- Agent embedded in an environment that feeds back states and rewards

Markov decision process

- Set of possible states X
- Set of possible actions U
- Transition function (dynamics) $f(x, u)$
- Reward function $\rho(x, u)$

Example: Resonating robot arm



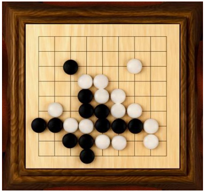
- Designed for pick & place tasks; spring stores energy between moves
- $x = [\text{angle } \alpha, \text{ velocity } \dot{\alpha}]^\top \in X = [-2, 2]\,\text{rad} \times [-2\pi, 2\pi]\,\text{rad/s}$
- u = motor torque $\in U = [-2, 2]\,\text{Nm}$
- Plane inclined at 0.4 rad
- Dynamics f discrete-time with $T_s = 0.05$

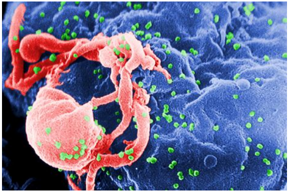
Objective: move to $\alpha_g = 0.85$ (often from -0.85):

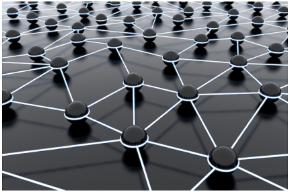
- $\rho(x, u) = 1 - (\alpha - \alpha_g)^2 \frac{1}{\Delta_{max}}$
- Discount factor $\gamma = 0.95$

Applications

Artificial intelligence, control, medicine, multiagent systems, economics etc.







RL on the machine learning spectrum

Supervised learning | Reinforcement learning | Unsupervised learning

← more informative feedback | less informative feedback

- Supervised: for each training sample, **correct output** known
- Unsupervised: only input samples, **no outputs**; find patterns in the data
- Reinforcement: correct actions not available, **only rewards**

But note: RL finds **dynamical optimal control**!

RL on the machine learning spectrum

Supervised learning — Reinforcement learning — Unsupervised learning

← more informative feedback — less informative feedback

- Supervised: for each training sample, **correct output** known
- Unsupervised: only input samples, **no outputs**; find patterns in the data
- Reinforcement: correct actions not available, **only rewards**

But note: RL finds **dynamical optimal control**!

Solution using Q-functions

- **Q-function** measures quality of policy h for each state-action pair x_0, u_0:

$$Q^h(x_0, u_0) = \rho(x_0, u_0) + \sum_{k=1}^{\infty} \gamma^k \rho(x_k, h(x_k))$$

i.e. return on applying u_0 in x_0 and then following h

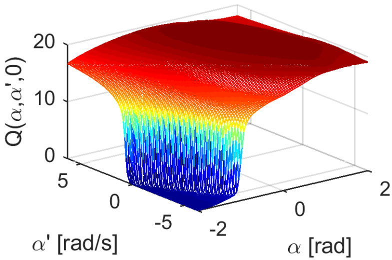
- **Optimal Q-function**: $Q^* = \max_h Q^h$
- "Greedy" policy in Q^*: $h^*(x) = \arg\max_u Q^*(x, u)$ is **optimal**, i.e. achieves maximal returns

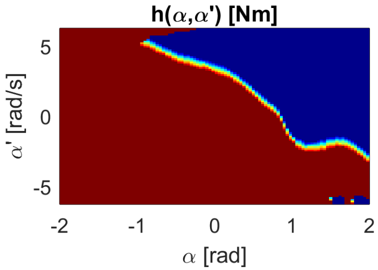
Bellman optimality equation

$$Q^*(x, u) = \rho(x, u) + \gamma \max_{u'} Q^*(f(x, u), u')$$

Resonating arm: Near-optimal solution

Left: slice $\hat{Q}^*(x, u)$ for $u = 0$ Right: near-optimal policy





Controlled trajectory

Algorithm landscape

By model usage:

- Model-based, DP: f, ρ known a priori
- Model-free RL: f, ρ unknown, learn solution from data
- Model-learning RL: f, ρ found from data

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)

Algorithm landscape

By model usage:

- Model-based, DP: f, ρ known a priori
- Model-free RL: f, ρ unknown, learn solution from data
- Model-learning RL: f, ρ found from data

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)

Algorithm landscape

By model usage:

- Model-based, DP: f, ρ known a priori
- Model-free RL: f, ρ unknown, learn solution from data
- Model-learning RL: f, ρ found from data

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)

Algorithm selection

Four basic algorithms:

1. Exact (discrete)
2. Approximate (continuous)

x

1. Offline
2. Online

There are many more!

Q-iteration

Transforms Bellman optimality equation:

$$Q^*(x,u) = \rho(x,u) + \gamma \max_{u'} Q^*(f(x,u),u')$$

into an **iterative procedure**:

Q-iteration

repeat at each iteration ℓ
  for all x, u **do**
    $Q_{\ell+1}(x,u) \leftarrow \rho(x,u) + \gamma \max_{u'} Q_\ell(f(x,u),u')$
  end for
until convergence to Q^*
Once Q^* available: $h^*(x) = \arg\max_u Q^*(x,u)$

- Offline, model-based; a type of value iteration
- Major contenders: policy iteration, policy search

Q-iteration

Transforms Bellman optimality equation:

$$Q^*(x, u) = \rho(x, u) + \gamma \max_{u'} Q^*(f(x, u), u')$$

into an **iterative procedure**:

Q-iteration

repeat at each iteration ℓ
  for all x, u **do**
    $Q_{\ell+1}(x, u) \leftarrow \rho(x, u) + \gamma \max_{u'} Q_\ell(f(x, u), u')$
  end for
until convergence to Q^*
Once Q^* available: $h^*(x) = \arg\max_u Q^*(x, u)$

- Offline, model-based; a type of value iteration
- Major contenders: policy iteration, policy search

Q-iteration

Transforms Bellman optimality equation:

$$Q^*(x,u) = \rho(x,u) + \gamma \max_{u'} Q^*(f(x,u),u')$$

into an **iterative procedure**:

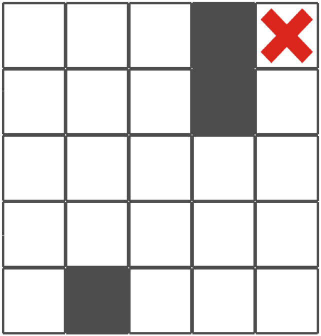
Q-iteration

repeat at each iteration ℓ
  for all x, u **do**
    $Q_{\ell+1}(x,u) \leftarrow \rho(x,u) + \gamma \max_{u'} Q_\ell(f(x,u),u')$
  end for
until convergence to Q^*
Once Q^* available: $h^*(x) = \arg\max_u Q^*(x,u)$

- Offline, model-based; a type of value iteration
- Major contenders: policy iteration, policy search

Gridworld example: Q-iteration

Task: Navigate to goal "X"
Reward 10 on reaching it, -0.1 otherwise; discount $\gamma = 0.95$
Actions: cardinal directions



Q-learning

1. Start from Q-iteration:

$$Q_{\ell+1}(x,u) \leftarrow \rho(x,u) + \gamma \max_{u'} Q_\ell(f(x,u),u')$$

2. Instead of model, use at each step k **observed transition** $(x_k, u_k, x_{k+1}, r_{k+1})$:

$$Q(x_k,u_k) \leftarrow r_{k+1} + \gamma \max_{u'} Q(x_{k+1},u')$$

Note: $x_{k+1} = f(x_k,u_k)$, $r_{k+1} = \rho(x_k,u_k)$

3. Turn into **incremental** update:

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

$\alpha_k \in (0,1]$ learning rate

Q-learning

1. Start from Q-iteration:
$$Q_{\ell+1}(x,u) \leftarrow \rho(x,u) + \gamma \max_{u'} Q_\ell(f(x,u),u')$$

2. Instead of model, use at each step k **observed transition** $(x_k, u_k, x_{k+1}, r_{k+1})$:
$$Q(x_k,u_k) \leftarrow r_{k+1} + \gamma \max_{u'} Q(x_{k+1},u')$$
Note: $x_{k+1} = f(x_k,u_k)$, $r_{k+1} = \rho(x_k,u_k)$

3. Turn into **incremental** update:
$$Q(x_k,u_k) \leftarrow Q(x_k,u_k) + \alpha_k \cdot \left[r_{k+1} + \gamma \max_{u'} Q(x_{k+1},u') - Q(x_k,u_k)\right]$$
$\alpha_k \in (0,1]$ learning rate

Q-learning

1. Start from Q-iteration:
$$Q_{\ell+1}(x,u) \leftarrow \rho(x,u) + \gamma \max_{u'} Q_\ell(f(x,u),u')$$

2. Instead of model, use at each step k **observed transition** $(x_k, u_k, x_{k+1}, r_{k+1})$:
$$Q(x_k,u_k) \leftarrow r_{k+1} + \gamma \max_{u'} Q(x_{k+1},u')$$
Note: $x_{k+1} = f(x_k,u_k)$, $r_{k+1} = \rho(x_k,u_k)$

3. Turn into **incremental** update:
$$Q(x_k,u_k) \leftarrow Q(x_k,u_k) + \alpha_k \cdot \left[r_{k+1} + \gamma \max_{u'} Q(x_{k+1},u') - Q(x_k,u_k)\right]$$
$\alpha_k \in (0,1]$ learning rate

Q-learning

Q-learning

for each trial **do**
  initialize state x_0
  repeat at each step k
    choose and apply u_k, measure x_{k+1}, receive r_{k+1}
    $Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot$
    $$[r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)]$$
  until trial finished
end for

- Online, model-free (RL);
  a type of temporal-difference learning
- Major contender: SARSA

Q-learning

Q-learning

for each trial **do**
  initialize state x_0
  repeat at each step k
    choose and apply u_k, measure x_{k+1}, receive r_{k+1}
    $$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot \left[r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)\right]$$
  until trial finished
end for

- Online, model-free (RL);
  a type of temporal-difference learning
- Major contender: SARSA

Convergence

Q-iteration converges to optimal solution Q^* with rate γ

Q-learning converges to Q^* if:

1. α_k satisfies some technical conditions
2. all pairs (x, u) continue to be updated

How to ensure condition 2? Key requirement: **exploration**

Convergence

Q-iteration converges to optimal solution Q^* with rate γ

Q-learning converges to Q^* if:

1. α_k satisfies some technical conditions
2. all pairs (x, u) continue to be updated

How to ensure condition 2? Key requirement: **exploration**

Convergence

Q-iteration converges to optimal solution Q^* with rate γ

Q-learning converges to Q^* if:

1. α_k satisfies some technical conditions
2. all pairs (x, u) continue to be updated

How to ensure condition 2? Key requirement: **exploration**

Exploration-exploitation dilemma

- **Exploration** needed:
  actions different from what currently seems best
- **Exploitation** of current knowledge also needed,
  to behave well

This dilemma is essential in all RL algorithms

ε-greedy strategy

- Simple solution to the exploration-exploitation dilemma: **ε-greedy**

$$u_k = \begin{cases} h(x_k) = \arg\max_u Q(x_k, u) & \text{with probability } (1 - \varepsilon_k) \\ \text{a uniformly random action} & \text{w.p. } \varepsilon_k \end{cases}$$

- Exploration probability $\varepsilon_k \in (0, 1)$ usually decreased over time
- Main disadvantage: when exploring, actions are fully random, leading to poor performance

ε-greedy strategy

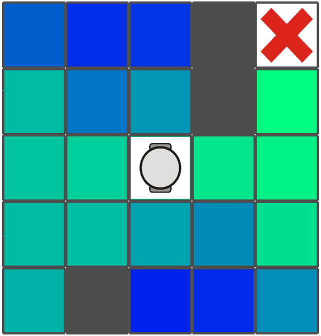
- Simple solution to the exploration-exploitation dilemma: **ε-greedy**

$$u_k = \begin{cases} h(x_k) = \arg\max_u Q(x_k, u) & \text{with probability } (1 - \varepsilon_k) \\ \text{a uniformly random action} & \text{w.p. } \varepsilon_k \end{cases}$$

- Exploration probability $\varepsilon_k \in (0, 1)$ usually decreased over time
- Main disadvantage: when exploring, actions are fully random, leading to poor performance

Gridworld: Q-learning with ε-greedy exploration

Settings: constant $\alpha = 0.5$, ε starts at 0.9 and decays to 0.95 of its value after each trial



Softmax strategy

- Action selection:

$$u_k = u \text{ w.p. } \frac{e^{Q(x_k,u)/\tau_k}}{\sum_{u'} e^{Q(x_k,u')/\tau_k}}$$

  where $\tau_k > 0$ is the **exploration temperature**
- Taking $\tau \to 0$, greedy selection recovered;
  $\tau \to \infty$ gives uniform random
- Compared to ε-greedy, better actions are more likely to be applied even when exploring

Many other options, including mathematically well-founded, e.g. bandit theory, Bayesian exploration

Softmax strategy

- Action selection:

$$u_k = u \text{ w.p. } \frac{e^{Q(x_k,u)/\tau_k}}{\sum_{u'} e^{Q(x_k,u')/\tau_k}}$$

  where $\tau_k > 0$ is the **exploration temperature**
- Taking $\tau \to 0$, greedy selection recovered;
  $\tau \to \infty$ gives uniform random
- Compared to ε-greedy, better actions are more likely to be applied even when exploring

Many other options, including mathematically well-founded, e.g. bandit theory, Bayesian exploration

1. Introduction and framework
2. Solution
3. Discrete-case algorithms
4. Exploration
5. Approximation and fitted Q-iteration
6. Approximate Q-learning

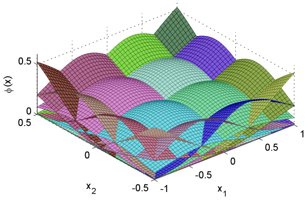
Need for approximation

- Classically, x, u discrete
  $Q(x, u)$ and $h(x)$ exactly represented, e.g. via tables with x on rows and u on columns (hence “tabular methods”)
- In e.g. robotics and control, x, u typically **continuous**
- **Approximation** over x, u necessary

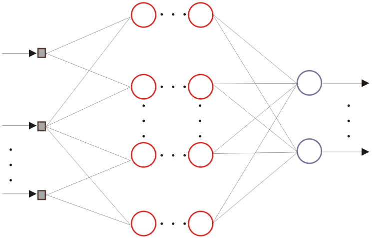
Approximators

- Parametric: fixed form, # of parameters

Linear:
$\widehat{Q}(x, u; \theta) = \sum_i \phi_i(x, u)\theta_i$
E.g. RBFs



Nonlinear:
E.g. (deep) neural net

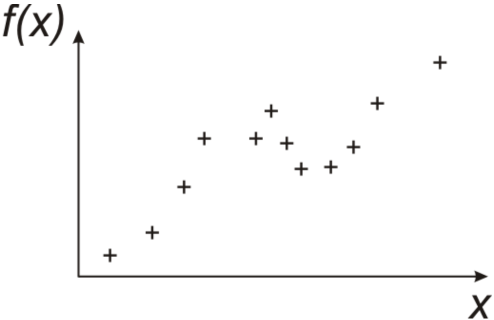


- Nonparametric: form, # of parameters derived from data
  E.g. local linear regression

Nonparametric example: Local linear regression

Local linear regression, LLR:

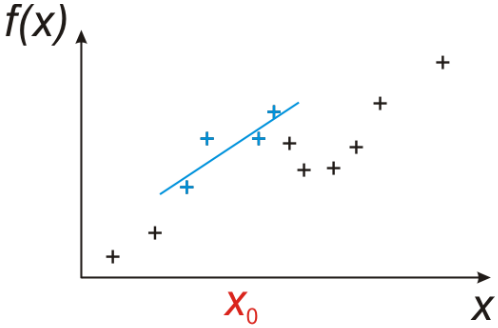
- Database of points $(x, f(x))$ (e.g. the training data)
- For given x_0, finds the k **nearest neighbors**
- Result found with **linear regression** on neighbors



Nonparametric example: Local linear regression

Local linear regression, LLR:

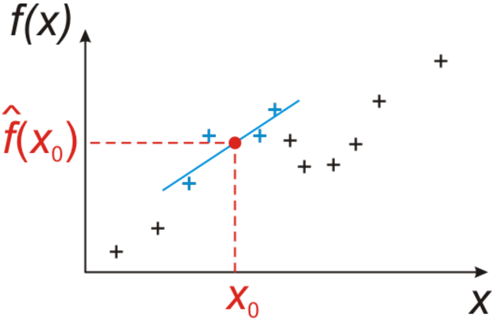
- Database of points $(x, f(x))$ (e.g. the training data)
- For given x_0, finds the k **nearest neighbors**
- Result found with **linear regression** on neighbors



Nonparametric example: Local linear regression

Local linear regression, LLR:

- Database of points $(x, f(x))$ (e.g. the training data)
- For given x_0, finds the k **nearest neighbors**
- Result found with **linear regression** on neighbors



Fitted Q-iteration: Idea

Recall Q-iteration:

for all x, u $\quad Q_{\ell+1}(x,u) \leftarrow \rho(x,u) + \gamma \max_{u'} Q_\ell(f(x,u),u')$ **end for**

1. Use an **approximator** $\widehat{Q}_\ell$ instead of exact Q-function
2. Use batch of **transition samples** (x_s, u_s, x'_s, r_s) instead of model (and of iterating over all x and u)
3. Train approximator $\ell + 1$ to recover Q-value targets $q_s = r_s + \gamma \max_{u'} \widehat{Q}_\ell(x'_s, u')$ computed from samples

Fitted Q-iteration: Idea

Recall Q-iteration:

for all x, u $\quad Q_{\ell+1}(x,u) \leftarrow \rho(x,u) + \gamma \max_{u'} Q_{\ell}(f(x,u),u')$ **end for**

1. Use an **approximator** $\widehat{Q}_{\ell}$ instead of exact Q-function
2. Use batch of **transition samples** (x_s, u_s, x'_s, r_s) instead of model (and of iterating over all x and u)
3. Train approximator $\ell + 1$ to recover Q-value targets $q_s = r_s + \gamma \max_{u'} \widehat{Q}_{\ell}(x'_s, u')$ computed from samples

Fitted Q-iteration: Idea

Recall Q-iteration:

for all x, u $\quad Q_{\ell+1}(x,u) \leftarrow \rho(x,u) + \gamma \max_{u'} Q_\ell(f(x,u),u')$ **end for**

1. Use an **approximator** $\widehat{Q}_\ell$ instead of exact Q-function
2. Use batch of **transition samples** (x_s, u_s, x'_s, r_s) instead of model (and of iterating over all x and u)
3. Train approximator $\ell + 1$ to recover Q-value targets $q_s = r_s + \gamma \max_{u'} \widehat{Q}_\ell(x'_s, u')$ computed from samples

Fitted Q-iteration: Idea

Recall Q-iteration:

for all x, u $\quad Q_{\ell+1}(x,u) \leftarrow \rho(x,u) + \gamma \max_{u'} Q_\ell(f(x,u),u')$ **end for**

1. Use an **approximator** $\widehat{Q}_\ell$ instead of exact Q-function
2. Use batch of **transition samples** (x_s, u_s, x'_s, r_s) instead of model (and of iterating over all x and u)
3. Train approximator $\ell + 1$ to recover Q-value targets $q_s = r_s + \gamma \max_{u'} \widehat{Q}_\ell(x'_s, u')$ computed from samples

Fitted Q-iteration: Algorithm

Fitted Q-iteration

given samples (x_s, u_s, r_s, x'_s), $s = 1, \ldots, S$
repeat at each iteration ℓ
  for $s = 1, \ldots, S$ **do**
    $q_s \leftarrow r_s + \gamma \max_{u'} \widehat{Q}_\ell(x'_s, u')$
  end for
  train $\widehat{Q}_{\ell+1}$ so that $\widehat{Q}_{\ell+1}(x_s, u_s) \approx q_s$ for all s
until finished

(Ernst et al., 2005)

- Offline, model-free (RL) if samples obtained from system
- Algorithm works for stochastic case

Fitted Q-iteration: Algorithm

Fitted Q-iteration

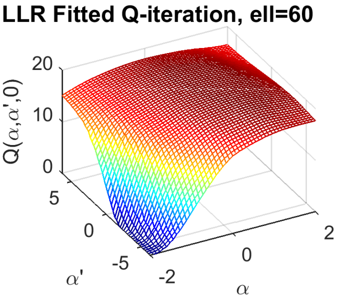
given samples (x_s, u_s, r_s, x'_s), $s = 1, \ldots, S$
repeat at each iteration ℓ
  for $s = 1, \ldots, S$ **do**
    $q_s \leftarrow r_s + \gamma \max_{u'} \widehat{Q}_\ell(x'_s, u')$
  end for
  train $\widehat{Q}_{\ell+1}$ so that $\widehat{Q}_{\ell+1}(x_s, u_s) \approx q_s$ for all s
until finished

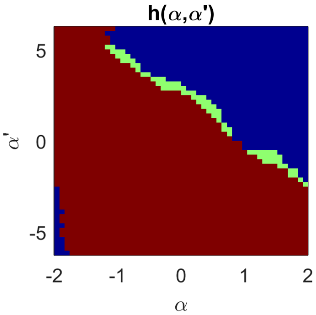
(Ernst et al., 2005)

- Offline, model-free (RL) if samples obtained from system
- Algorithm works for stochastic case

Resonating arm: LLR Fitted Q-iteration

Approximation: LLR, $k = 10$ nearest neighbors over X;
u discretized in $\{-2, 0, 2\}$ V to keep maximization simple
Samples: Grid of 31×15 on X, $\times$ all 3 discretized actions





Recall: Classical Q-learning

Q-learning

for each trial **do**
  initialize state x_0
  repeat at each step k
    choose and apply u_k, measure x_{k+1}, receive r_{k+1}
    $$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot \\ [r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)]$$
  until trial finished
end for

Approximate Q-learning: Idea

Use parametric approximator $\widehat{Q}(x, u; \theta)$, update **parameters** θ

- **Gradient descent** on the error $[Q^*(x_k, u_k) - \widehat{Q}(x_k, u_k; \theta)]$:

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

$$= \theta_k + \alpha_k \frac{\partial}{\partial\theta} \widehat{Q}(x_k, u_k; \theta_k) \cdot \left[Q^*(x_k, u_k) - \widehat{Q}(x_k, u_k; \theta_k)\right]$$

- Use available **estimate** of $Q^*(x_k, u_k)$:

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

Approximate Q-learning: Idea

Use parametric approximator $\widehat{Q}(x, u; \theta)$, update **parameters** θ

- **Gradient descent** on the error $[Q^*(x_k, u_k) - \widehat{Q}(x_k, u_k; \theta)]$:

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

- Use available **estimate** of $Q^*(x_k, u_k)$:

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

Approximate Q-learning: Idea

Use parametric approximator $\widehat{Q}(x, u; \theta)$, update **parameters** θ

- **Gradient descent** on the error $[Q^*(x_k, u_k) - \widehat{Q}(x_k, u_k; \theta)]$:

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

- Use available **estimate** of $Q^*(x_k, u_k)$:

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

Approximate Q-learning: Algorithm

Approximate Q-learning

for each trial **do**
  init x_0
  repeat at each step k
    choose and apply u_k, measure x_{k+1}, receive r_{k+1}

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

  until trial finished
end for

- Online, model-free (RL); exploration needed
- Many variants exist

(Sutton & Barto, 2018)

Approximate Q-learning: Algorithm

Approximate Q-learning

for each trial **do**
  init x_0
  repeat at each step k
    choose and apply u_k, measure x_{k+1}, receive r_{k+1}

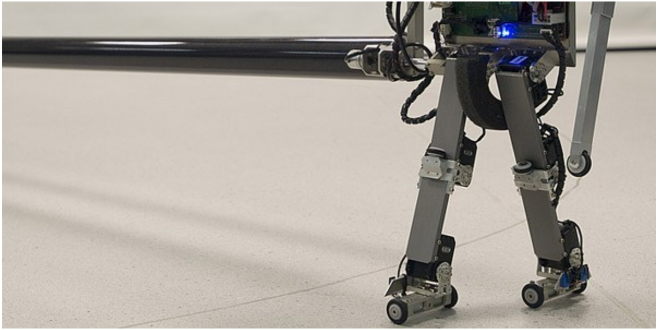
$$\theta_{k+1} = \theta_k + \alpha_k \frac{\partial}{\partial \theta} \widehat{Q}(x_k, u_k; \theta_k) \cdot \left[r_{k+1} + \gamma \max_{u'} \widehat{Q}(x_{k+1}, u'; \theta_k) - \widehat{Q}(x_k, u_k; \theta_k) \right]$$

  until trial finished
end for

- Online, model-free (RL); exploration needed
- Many variants exist

(Sutton & Barto, 2018)

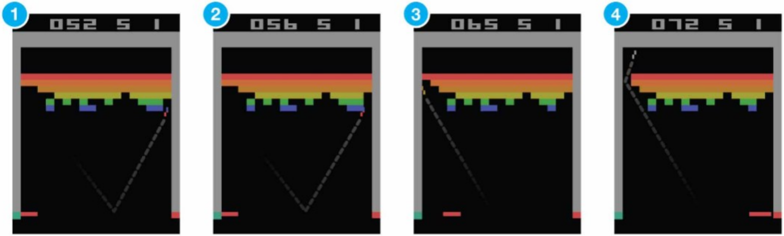
Approx. Q-learning: robot walking demo



(Schuitema, 2012)

DQN

Deep Q-Network algorithm is a few steps away,
in-between Q-learning and fitted Q-iteration



(Mnih et al., 2015)

Conclusion

Reinforcement learning =
learn how to **near-optimally** act on
an unknown system

Further topics:

- Policy search, policy iteration, deep RL, robot learning, safety & stability, etc. ...

Thank you!

Conclusion

Reinforcement learning =
learn how to **near-optimally** act on
an unknown system

Further topics:

- Everything
- Policy search, policy iteration, deep RL, robot learning, safety & stability, etc. ...

Thank you!

Conclusion

Reinforcement learning =
learn how to **near-optimally** act on
an unknown system

Further topics:

- Everything
- Policy search, policy iteration, deep RL, robot learning, safety & stability, etc. . . .

Thank you!

Conclusion

Reinforcement learning =
learn how to **near-optimally** act on
an unknown system

Further topics:

- Everything
- Policy search, policy iteration, deep RL, robot learning, safety & stability, etc. . . .

Thank you!

Selected books and cited references

- Sutton & Barto, *Reinforcement Learning: An Introduction*, 2nd ed., 2018.
- Bertsekas, *Dynamic Programming and Optimal Control*, vol II 4th ed., 2012.
- Szepesvári, *Algorithms for RL*, 2010.
- Buşoniu, Babuška, De Schutter, & Ernst, *RL and DP Using Function Approximators*, 2010.
- Powell, *Approximate DP: Solving the Curses of Dimensionality*, 2nd ed., 2011.
- Lewis & Liu (eds.) *RL and Approximate DP for Feedback Control*, 2012.

- Buşoniu, Babuška, De Schutter, & Ernst, *RL & DP Using Function Approximators*, 2010.
- Ernst, Geurts, Wehenkel, *Tree-Based Batch Mode RL*, JMLR 2005.
- Schuitema, *RL on Autonomous Humanoid Robots*, PhD thesis, 2012.
- Mnih et al. (DeepMind), *Human-level Control through Deep RL*, Nature, 2015.