Reinforcement Learning

Shimon Whiteson
Dept. of Computer Science
University of Oxford

February 28, 2017
How can an intelligent agent learn from experience how to make decisions that maximise its utility in the face of uncertainty?
Artificial intelligence

Knowledge-Based Systems

Machine Learning

- Unsupervised Learning
- Reinforcement Learning
- Supervised Learning
Reinforcement learning

- In **reinforcement learning** an agent tries to solve a control problem by directly interacting with an unfamiliar environment.

- The agent must learn by trial and error, trying out actions to learn about their consequences.

- Applicable to robot control, game playing, system optimisation, advertising, and information retrieval.

- Part of machine learning, inspired by behavioural psychology, related to operations research, control theory, classical planning, and aspects of neuroscience.
Reinforcement learning vs. supervised learning

- No examples of correct or incorrect behaviour; instead only **rewards** for actions tried
- The agent is **active** in the learning process: it has partial control over what data it will obtain for learning
- The agent must learn **on-line**: it must maximise performance during learning, not afterwards
The **$K$-armed bandit problem** is as follows:

- Sit before a slot machine (bandit) with many arms.
- Each arm has an unknown stochastic payoff.
- Goal is to maximise cumulative payoff over some period.
Formalizing the $K$-armed bandit problem

- There are $K$ actions available at each timestep (also called plays or pulls).

- After the $t$-th action, the agent receives reward $r_i \sim R_{at}$.

- In a finite-horizon problem, the agent tries to maximise its total reward over $T$ actions: $\sum_{t=1}^{T} r_t$.

- In an infinite-horizon problem, the agent tries to maximise its discounted total reward: $\sum_{t=0}^{\infty} \gamma^t r_t$ where $\gamma \in [0, 1)$.

- $1 - \gamma$ can be interpreted as the probability of the game ending after each step.
Pop quiz

Would you prefer to receive 50 pounds today or 100 pounds a year from now?
Pop quiz

- Would you prefer to receive 50 pounds today or 100 pounds a year from now?

- Would you prefer to receive 50 pounds 5 years from now or 100 pounds six years from now?
Exponential vs. hyperbolic discounting

- **Exponential discounting**: $\gamma$ fixed; the percent decline in reward is constant

- Human behaviour is often irrational for any fixed $\gamma$

- Explained using **hyperbolic discounting**: percent decline decreases over time

(Figure from Wikipedia)
Id vs. ego: the war against yourself

Now
50 In 5 Years ✗
100 In 6 Years ✓
1 Year
From Now
50 Now ✓
100 In A Year ✗
5 Years
From Now
6 Years
From Now
50 Now ✓
100 In A Year ✗
Self-binding: tricking your id

Is the following behaviour rational?

- Buy a bottle of whiskey
- Take one drink
- Pour out the rest of the bottle
Self-binding: tricking your id

Is the following behaviour rational?

- Buy a bottle of whiskey
- Take one drink
- Pour out the rest of the bottle

Binding your sharply-discounting id is the key to success!
Exploration and exploitation

- The agent’s ability to get reward in the future depends on what it knows about the arms. Thus, it must **explore** the arms in order to learn about them and improve its chances of getting future reward.

- But the agent must also use what it already knows in order to maximise its total reward; Thus it must **exploit** by pulling the arms it expects to give the largest rewards.
Balancing exploration and exploitation

- The main challenge in a $k$-armed bandit is how to balance the competing needs of exploration and exploitation.

- If the horizon is finite, exploration should decrease as the horizon gets closer.

- If the horizon is infinite but $\gamma < 1$, exploration should decrease as the agent’s uncertainty about expected rewards goes down.

- If the horizon is infinite and $\gamma = 1$, there is an infinitely delayed splurge.
Action-value methods

- Based on observed rewards, maintain estimates of the expected value of each arm: \( Q_t(a) \approx E[r_t|a_t] \)

- Estimates are based on the sample average; If action \( a \) has been chosen \( k_a \) times, yielding rewards \( r_1, r_2, \ldots, r_{k_a} \), then:

\[
Q_t(a) = \frac{\sum_{i=1}^{k_a} r_i}{k_a}
\]

- Exploiting means taking the greedy action: \( a^* = \arg \max_a Q_t(a) \)

- Exploring means taking any other action
\(\epsilon\)-greedy exploration

In \(\epsilon\)-greedy exploration, the agent selects a random action with probability \(\epsilon\), and the greedy action otherwise.
Softmax exploration

In softmax exploration, the agent chooses actions according to a Boltzmann distribution

\[ p(a) = \frac{\exp\left(\frac{Q(a)}{\tau}\right)}{\sum_{a'} \exp\left(\frac{Q(a')}{\tau}\right)} \]
Optimism in the face of uncertainty

- Neither $\epsilon$-greedy nor softmax considers **uncertainty** in action-value estimates.
- Goal of exploration is to reduce uncertainty.
- So focus exploration on most uncertain actions.
- Principle of **optimism in the face of uncertainty**.
Upper confidence bound

- Compute **confidence interval** for each arm
- Select arm with largest **upper confidence bound**

Formally: \( a_t = \arg \max_i u_t(a_i) \) where \( u_t(a_i) = Q_t(a_i) + c_t(a_i) \)
  - \( Q_t(a_i) \): action-value estimate
  - \( c_t(a_i) \): optimism bonus

Defining optimism bonus:
  - \( c_t(a_i) = \sqrt{\frac{\alpha \ln t}{N_t(a_i)}} \)
  - Decreases with \# pulls of \( a_i \)
  - Increases with \( t \)
Contextual bandit problem

- Also called **associative search**
- At each play, agent receives a **state signal**, also called an **observation** or **side-information**
- Expected payoffs depend on that observation
- Suppose there are many bandits, each a different color; after each play, you are randomly transported to another bandit
- In principle, can be treated as multiple simultaneous bandit problems and estimate $Q(s, a)$
Ad placement

- Web page = state
- Actions = ads
- Environment = user
- Reward = pay per click
The full reinforcement learning problem

- **Sequential decision task**: actions affect both reward and next state (and thereby opportunities for future reward)
- Agent’s behavior determined by its policy $\pi : S \rightarrow A$
- Goal: find an optimal policy $\pi^*$ maximizing expected sum of rewards
The credit-assignment problem

Suppose an agent takes a long sequence of actions, at the end of which it receives a large positive reward?

How can it determine to what degree each action in that sequence deserves credit for the resulting reward?
Richard Bellman

- Father of decision-theoretic planning
- Formalized Markov decision processes, derived Bellman equation, invented dynamic programming
- “A towering figure among the contributors to modern control theory and systems analysis” - IEEE
- “The Bellman equation is one of the five most important ideas in artificial intelligence” - Bram Bakker
Return

- The goal of the agent is to maximize the expected return, a sum over the rewards received.
- In an infinite-horizon task, the return is defined as:

\[ R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \]

- In a finite-horizon task, this becomes a finite summation
- In an infinite-horizon task that is episodic instead of continuing, we represent episode termination as transition to an absorbing state with self-transitions and zero reward.
Value functions

- **Value functions**: the primary tool for reasoning about future reward
- The **state-value function** of a policy \( \pi \) is:

\[
V^\pi(s) = E_\pi \left[ R_t | s_t = s \right] = E_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right]
\]

- The **action-value** of a policy \( \pi \) is:

\[
Q^\pi(s, a) = E_\pi \left[ R_t | s_t = s, a_t = a \right] = E_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right]
\]
The definition of $V^\pi$ can be rewritten recursively by making use of the transition model, yielding the **Bellman equation**: 

$$V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V^\pi(s') \right]$$

This is a set of linear equations, one for each state, the solution of which defines the value of $\pi$.

A similar recursive definition holds for $Q$-values:

$$Q^\pi(s, a) = \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma \sum_{a'} \pi(s', a') Q^\pi(s', a') \right]$$
Optimal value functions

- Value functions define a partial ordering over policies:
  \[ \pi \succ \pi' \Rightarrow V^\pi(s) \geq V^{\pi'}(s), \forall s \in S \]

- There can be multiple optimal policies but they all share the same optimal state-value function:
  \[ V^*(s) = \max_\pi V^\pi(s), \forall s \in S \]

- They also share the same optimal action-value function:
  \[ Q^*(s, a) = \max_\pi Q^\pi(s, a), \forall s \in S, a \in A \]
Bellman optimality equations

- **Bellman optimality equations** express this recursively:

  $$V^* = \max_{a \in A} \sum_{s'} P^a_{ss'} \left[ R^a_{ss'} + \gamma V^*(s') \right]$$

  $$Q^*(s, a) = \sum_{s'} P^a_{ss'} \left[ R^a_{ss'} + \gamma \max_{a' \in A} Q^*(s', a') \right]$$

- An optimal policy is **greedy** with respect to $V^*$ or $Q^*$:

  $$\pi^*(s) \in \arg \max_a Q^*(s, a) = \arg \max_a \left[ R^a_{ss'} + \gamma \sum_{s'} P^a_{ss'} V^*(s') \right]$$
MDP planning

- MDPs give us a formal model of sequential decision making
- Given the optimal value function, computing an optimal policy is straightforward
- How can we find $V^*$ or $Q^*$?
- Algorithms for **MDP planning** compute the optimal value function given a complete model of the MDP
- Given a model, $V^*$ is usually sufficient
Dynamic programming approach

\[ \pi \rightarrow V \rightarrow V^\pi \rightarrow \text{evaluation} \]

\[ \pi \rightarrow \text{greedy}(V) \rightarrow \text{improvement} \]

\[ \pi^* \leftarrow V^* \rightarrow \pi^* \]
Policy evaluation

- Rather than estimating value of each state independently, use Bellman equation to exploit the relationship between states.
- Initial value function $V_0$ is chosen arbitrarily.
- Policy evaluation update rule:

$$V_{k+1}(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a \left( R_{ss'}^a + \gamma V_k(s') \right)$$

- Apply to every state in each sweep of the state space.
- Repeat over many sweeps.
- Converges to the fixed point $V_k = V^\pi$. 

Shimon Whiteson
Reinforcement Learning
February 28, 2017
Policy improvement (1)

- Policy evaluation yields $V^\pi$, the true value of $\pi$
- Use this to incrementally improve the policy by considering whether for some state $s$ there is a better action $a \neq \pi(s)$
- Is choosing $a$ in $s$ and then using $\pi$ better than using $\pi$, i.e.,

$$Q^\pi(s, a) = \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V^\pi(s') \right] \geq V^\pi(s)?$$

- If so, then the **policy improvement theorem** tells us that changing $\pi$ to take $a$ in $s$ will increase its value:

$$\forall s \in S, Q^\pi(s, \pi'(s)) \geq V^\pi(s) \Rightarrow \forall s \in S, V^{\pi'}(s) \geq V^\pi(s)$$

- In our case, $\pi = \pi'$ except that $\pi'(s) = a \neq \pi(s)$
Policy improvement (2)

- Applying this principle at all states yields the **greedy** policy with respect to $V^\pi$:

$$
\pi'(s) \leftarrow \arg \max_a Q^\pi(s, a) = \arg \max_a \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V^\pi(s') \right]
$$

- If $\pi = \pi'$, then $V^\pi = V^{\pi'}$ and for all $s \in S$:

$$
V^{\pi'} = \max_{a \in A} \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V^{\pi'}(s') \right]
$$

- This is equivalent to the Bellman optimality equation, implying that $V^\pi = V^{\pi'} = V^*$ and $\pi = \pi' = \pi^*$
Policy iteration

\[ V = V\pi \]

\[ \pi = \text{greedy}(V) \]
Value iteration

- Need not complete policy evaluation before doing policy improvement
- In extreme case, two steps are integrated in one update rule:

\[ V_{k+1}(s) \leftarrow \max_a \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V_k(s') \right] \]

- Turns Bellman optimality equation into an update rule
- This can also be written:

\[
V_{k+1}(s) \leftarrow \max_a Q_{k+1}(s, a), \\
Q_{k+1}(s, a) \leftarrow \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V_k(s') \right]
\]
Temporal difference methods

- **TD(0):** on-policy TD estimation of $V^\pi$
  - TD(0) estimates $V^\pi$ from samples
  - It regresses to an **update target** constructed via **bootstrapping**

\[
V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]
\]
Temporal difference methods

- **TD(0):** on-policy TD estimation of $V^\pi$
  - TD(0) estimates $V^\pi$ from samples
  - It regresses to an update target constructed via bootstrapping
    \[
    V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]
    \]

- **Sarsa:** on-policy TD estimation of $Q^\pi$
  - To learn $\pi^*$ with TD, we need to learn $Q^*$ instead of $V^*$
  - Sarsa updates $Q$ by bootstrapping off next $(s, a)$:
    \[
    Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]
    \]
Temporal difference methods

- TD(0): on-policy TD estimation of $V^\pi$
  - TD(0) estimates $V^\pi$ from samples
  - It regresses to an update target constructed via bootstrapping
    $$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

- Sarsa: on-policy TD estimation of $Q^\pi$
  - To learn $\pi^*$ with TD, we need to learn $Q^*$ instead of $V^*$
  - Sarsa updates $Q$ by bootstrapping off next $(s, a)$:
    $$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

- Q-learning: off-policy TD control
  - Make TD off-policy: bootstrap with best action, not actual action:
    $$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$
Model-based reinforcement learning

- Planning methods require prior knowledge of the MDP
- TD methods are model-free reinforcement learning methods
- **Model-based** reinforcement learning assumes no prior knowledge but learns a model of the MDP and then plans on it
- A **model** is anything the agent can use to predict how the environment will respond to its actions
Types of models

- A **full** or **distribution** model is a complete description of $P_{ss'}^a$ and $R_{ss'}^a$: space complexity is $O(|S|^2|A|)$
- A **sample** or **generative** model can be queried to produce samples $r$ and $s'$ given any $s$ and $a$
- A **trajectory** or **simulation** model can simulate a complete episode but cannot jump to an arbitrary state
Planning, learning, and acting

- Model-based methods make fuller use of experience: lower \textit{sample complexity}
- Model-free methods are simpler and not affected by modeling errors
- Can also be combined
Vanilla model-based reinforcement learning

- Repeat:
  - Take exploratory action (based on greedy policy)
  - Use resulting immediate reward and state to update a maximum-likelihood model:
    \[
    \hat{P}_{ss'}^a = \frac{n_{ss'}^a}{n_s^a}, \quad \hat{R}_{ss'}^a = \frac{1}{n_{ss'}^a} \sum_{i=1}^{n_{ss'}^a} r_i
    \]
  - Solve the model using value iteration
  - Update greedy policy

- Computationally expensive
- But don’t have to plan to convergence or plan on every step
Use vanilla model-based RL

However, for all \((s, a)\) for which \(n_s^a < m\):

- Remove all transitions from \((s, a)\) from model
- Add transition of prob. 1 to artificial, terminal jackpot state
- Immediate reward on this transition is \(R_{\text{max}}\)

Plan on altered model

Remove artificial transitions once \(n_s^a \geq m\)

Agent will plan how to visit insufficiently visited states: efficient exploration
Function approximation

- Represent $Q$-function as, e.g., a neural network
- Tabular Q-learning

\[ Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \]

- Q-learning with function approximation:

\[
\theta_{t+1} \leftarrow \theta_t - \alpha \nabla_{\theta_t} [q_t - Q(s_t, a_t; \theta_t)]^2
\]

where:

\[
q_t = r_{t+1} + \gamma \max_{a} Q(s_{t+1}, a; \theta_t)
\]
DQN

Keys: poor man’s **fitted Q-iteration** and randomised **experience replay**
Other topics

- Policy gradient and actor-critic methods
- Gradient-free policy search
- Hierarchical reinforcement learning
- Partial observability
- Inverse reinforcement learning
- Multi-agent reinforcement learning
- Transfer reinforcement learning
- Multi-objective reinforcement learning
- Exploration in deep reinforcement learning
Telepresence robots
Robust reinforcement learning
Multi-agent scaling & communication
Active perception