Reinforcement Learning

Niccolò Consigli n.consigli@studenti.unipi.it
Luca Seggiani l.seggiani@studenti.unipi.it

Università di Pisa

November 25, 2024

A new kind of learning

- Supervised learning: matching features to labels

A new kind of learning

- Supervised learning: matching features to labels
- Unsupervised learning: clustering data

A new kind of learning

- Supervised learning: matching features to labels
- Unsupervised learning: clustering data
- **Reinforcement learning**: learning from rewards

What is reinforcement learning (RL)?

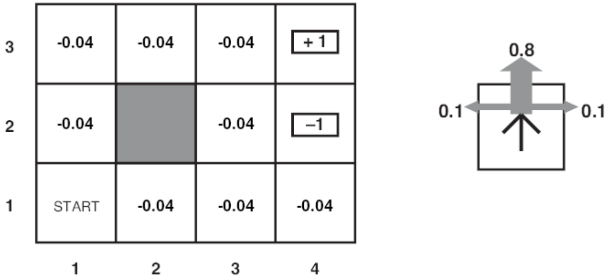
- We give the agent **rewards** based on its performance

What is reinforcement learning (RL)?

- We give the agent **rewards** based on its performance
- Markov property: we can view problems as **Markov decision processes** (MDPs)

A framework: Markov decision processes

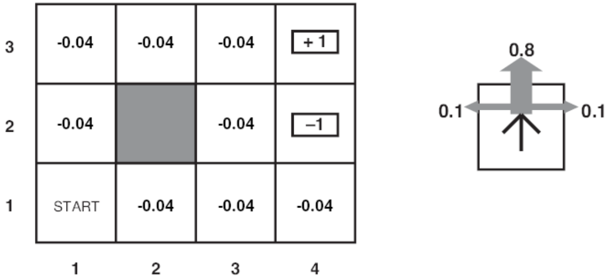
- Actions map states to states with a probability distribution
- Transition reward: $R(s, a, s')$
- Transition probability: $P(s' \mid s, a)$



(a) (b)

A framework: Markov decision processes

- Actions map states to states with a probability distribution
- Transition reward: $R(s, a, s')$
- Transition probability: $P(s' \mid s, a)$



(a) (b)

Bellman's equation

Given a discount factor $\gamma \in [0, 1]$, the utility is given by:

$$U(s) = \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + \gamma U(s') \right]$$

Model-based reinforcement learning

We maintain a transition model (an **MDP**):

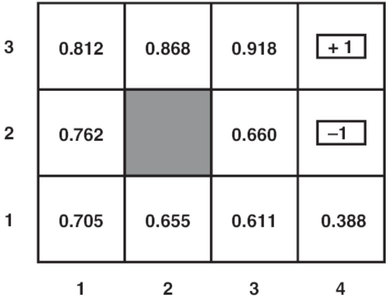
- Reward function: $R(s, a, s')$
- Probability function: $P(s' \mid s, a)$
- Utility function: U(s), maps states to **utility**

Model-based reinforcement learning

We maintain a transition model (an **MDP**):

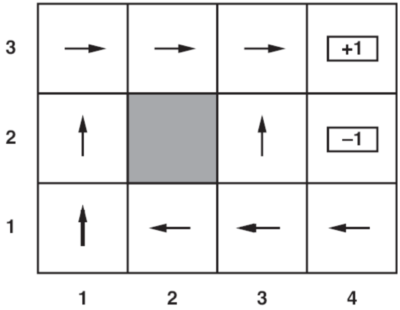
- Reward function: $R(s, a, s')$
- Probability function: $P(s' \mid s, a)$
- Utility function: U(s), maps states to **utility**

Once the model is learned, we can **maximize utility**



Model-free reinforcement learning

- Action-utility learning: we learn an action-utility function $Q(s, a)$ that maps **actions** to **utility**
- Policy search: maps **states** to **actions**, essentialy a reflex agent



Passive reinforcement learning

We can't modify the policy, but we can figure out the utilities $U(s)$

- We use **policy iteration**:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{+\infty} \gamma^t R(s_t, \pi(s_t), s_{t+1})\right]$$

Passive reinforcement learning

We can't modify the policy, but we can figure out the utilities $U(s)$

- We use **policy iteration**:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{+\infty} \gamma^t R(s_t, \pi(s_t), s_{t+1})\right]$$

Three approaches for approximation:

- Direct utility estimation
- Adaptive dynamic programming (ADP)
- Temporal difference learning (TD)

Direct utility estimation

The goal is to approximate state utility under the given policy

- Make **trials** and take the **reward-to-go** for each state

Direct utility estimation

The goal is to approximate state utility under the given policy

- Make **trials** and take the **reward-to-go** for each state

Example

We calculate utilities for $(1, 2)$ given the trial:

$$(1,1) \xrightarrow[\text{Up}]{-0.4} (1,2) \xrightarrow[\text{Up}]{-0.4} (1,3) \xrightarrow[\text{Right}]{-0.4} (1,2) \xrightarrow[\text{Up}]{-0.4} (1,3) \xrightarrow[\text{Right}]{-0.4} (2,3) \xrightarrow[\text{Right}]{-0.4} (3,3) \xrightarrow[\text{Right}]{+1} (4,3)$$

Direct utility estimation

The goal is to approximate state utility under the given policy

- Make **trials** and take the **reward-to-go** for each state

Example

We calculate utilities for $(1, 2)$ given the trial:

$$(1,1) \xrightarrow[\text{Up}]{-0.4} (1,2) \xrightarrow[\text{Up}]{-0.4} (1,3) \xrightarrow[\text{Right}]{-0.4} (1,2) \xrightarrow[\text{Up}]{-0.4} (1,3) \xrightarrow[\text{Right}]{-0.4} (2,3) \xrightarrow[\text{Right}]{-0.4} (3,3) \xrightarrow[\text{Right}]{+1} (4,3)$$

$$U(1,2) = -0.04 - 0.04 - 0.04 - 0.04 - 0.04 + 1 = 0.8$$

Direct utility estimation

The goal is to approximate state utility under the given policy

- Make **trials** and take the **reward-to-go** for each state

Example

We calculate utilities for $(1,2)$ given the trial:

$$(1{,}1) \xrightarrow[\text{Up}]{-0.4} (1{,}2) \xrightarrow[\text{Up}]{-0.4} (1{,}3) \xrightarrow[\text{Right}]{-0.4} (1{,}2) \xrightarrow[\text{Up}]{-0.4} (1{,}3) \xrightarrow[\text{Right}]{-0.4} (2{,}3) \xrightarrow[\text{Right}]{-0.4} (3{,}3) \xrightarrow[\text{Right}]{+1} (4{,}3)$$

$$U(1,2) = -0.04 - 0.04 - 0.04 - 0.04 - 0.04 + 1 = 0.8$$

$$U(1,2)' = -0.04 - 0.04 - 0.04 + 1 = 0.88$$

Direct utility estimation

The goal is to approximate state utility under the given policy

- Make **trials** and take the **reward-to-go** for each state

Example

We calculate utilities for $(1,2)$ given the trial:

$$(1{,}1) \xrightarrow[\text{Up}]{-0.4} (1{,}2) \xrightarrow[\text{Up}]{-0.4} (1{,}3) \xrightarrow[\text{Right}]{-0.4} (1{,}2) \xrightarrow[\text{Up}]{-0.4} (1{,}3) \xrightarrow[\text{Right}]{-0.4} (2{,}3) \xrightarrow[\text{Right}]{-0.4} (3{,}3) \xrightarrow[\text{Right}]{+1} (4{,}3)$$

$$U(1,2) = -0.04 - 0.04 - 0.04 - 0.04 - 0.04 + 1 = 0.8$$

$$U(1,2)' = -0.04 - 0.04 - 0.04 + 1 = 0.88$$

- This is inefficient! We can **exploit** the Markov property

Adaptive dynamic programming

We can approximate the transition reward and probability functions (P and R) and apply **simplified Bellman's equation**:

$$U(s) = \sum_{s'} P(s' \mid s, \pi(s)) \left[R(s, \pi(s), s') + \gamma U(s') \right]$$

When the policy is fixed, this gives a **linear equation**

Adaptive dynamic programming

We can approximate the transition reward and probability functions (P and R) and apply **simplified Bellman's equation**:

$$U(s) = \sum_{s'} P(s' \mid s, \pi(s)) \left[R(s, \pi(s), s') + \gamma U(s')\right]$$

When the policy is fixed, this gives a **linear equation**

- Calculating P and R is easy when the environment is **fully observable**

Temporal difference learning

Still taking trials, but we **update** the utility to match frequently observed transitions

- We don't need a model!

Temporal difference learning

Still taking trials, but we **update** the utility to match frequently observed transitions

- We don't need a model!

Example

We calculate utilities for transiton $(1,3) \rightarrow (2,3)$ given the trial:

$$(1{,}1) \xrightarrow[\text{Up}]{-0.4} (1{,}2) \xrightarrow[\text{Up}]{-0.4} (1{,}3) \xrightarrow[\text{Right}]{-0.4} (2{,}3) \xrightarrow[\text{Right}]{-0.4} (3{,}3) \xrightarrow[\text{Right}]{-0.04} (3{,}2) \xrightarrow[\text{Up}]{-0.04} (3{,}3) \xrightarrow[\text{Right}]{+1} (4{,}3)$$

and the previous utility estimates:
$U^{\pi}(1,3) = 0.88$, $U^{\pi}(2,3) = 0.96$

Temporal difference learning

Still taking trials, but we **update** the utility to match frequently observed transitions

- We don't need a model!

Example

We calculate utilities for transiton $(1,3) \rightarrow (2,3)$ given the trial:

$$(1{,}1) \xrightarrow[\text{Up}]{-0.4} (1{,}2) \xrightarrow[\text{Up}]{-0.4} (1{,}3) \xrightarrow[\text{Right}]{-0.4} (2{,}3) \xrightarrow[\text{Right}]{-0.4} (3{,}3) \xrightarrow[\text{Right}]{-0.04} (3{,}2) \xrightarrow[\text{Up}]{-0.04} (3{,}3) \xrightarrow[\text{Right}]{+1} (4{,}3)$$

and the previous utility estimates:
$U^{\pi}(1,3) = 0.88,\ \ U^{\pi}(2,3) = 0.96$

$$U^{\pi}(1,3)' = -0.04 + U^{\pi}(2,3)$$

- This gives $U^{\pi}(1,3)' = 0.92$, which means 0.88 might be a low estimate

Temporal difference learning (continued)

We want to match U^π to $U^{\pi\prime}$ with learning rate α:

$$U^\pi(1,3) \leftarrow U^\pi(1,3) + \alpha \left[U^\pi(1,3)' - U^\pi(1,3)\right]$$

Which generalizes to:

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha \left[R(s,\pi(s),s') + \gamma U^\pi(s') - U^\pi(s)\right]$$

- From TD's point of view, ADP is TD with *simulated experiences*

Active reinforcement learning

Utility functions give the best (expected) policy

We can decide to improve utility (**explore**) or keep maximizing utility (**exploit**)

- This is a **multi-armed bandit problem** (set of Markov reward processes)

Active reinforcement learning

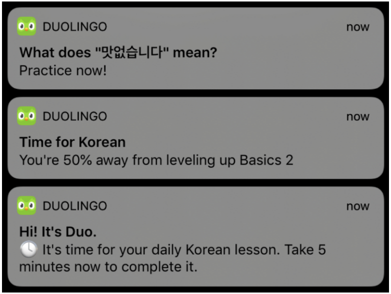
Utility functions give the best (expected) policy

We can decide to improve utility (**explore**) or keep maximizing utility (**exploit**)

- This is a **multi-armed bandit problem** (set of Markov reward processes)

Example

Duolingo's recurring notifications use a bandit problem solver:



Template	Eligibility Criteria
You're on fire Continue your 109 day Spanish streak on Duolingo.	*Must have a 3+ day streak.*[1]
Streak wager reminder You're on day 2 of your 7-day streak wager! Now get to day 3!	*Must have a streak wager.*[2]
Ready for your trip? Take 5 minutes to practice Italian now	*User's profile must indicate travel motivation.*

Greedy in the limit of infinite exploration (GLIE)

A greedy algorithm could ignore higher utility actions. We need a GLIE algorithm:

- Each eaction in a each state is tried an unbounded amount of times
- In the limit ($\to \infty$) the algorithm becomes truly greedy

Greedy in the limit of infinite exploration (GLIE)

A greedy algorithm could ignore higher utility actions. We need a GLIE algorithm:

- Each eaction in a each state is tried an unbounded amount of times
- In the limit ($\to \infty$) the algorithm becomes truly greedy

We can try different approaches:

- Random sampling
- Minimum trials

Random sampling

At step t, do the following:

- Choose a random action with probability $\frac{1}{t}$
- Otherwise, follow the greedy policy

This **does** eventually converge, but can be slow: a better approach would use some heuristic on state utility

Minimum trials

Have an **optimistic estimate** of utility ($U^+(s)$):

$$U^+(s) \leftarrow \max_a f\left(\sum_{s'} P(s' \mid s, a)\left[R(s, a, s') + \gamma U^+(s')\right], N(s, a)\right)$$

- $N(s, a)$ is the number of times action a was tried from state s
- f is defined as:

$$f(u, n) = \begin{cases} R^+, & n < N_e \\ u, & n \geq N_e \end{cases}$$

Here, R^+ is an optimistic utility for actions tried less than N_e times

Basically, we are estabilishing an optimistic **prior** that assigns higher utilities to actions tried less than N_e times

Safe exploration

In the real world, we have to look out for:

- Large negative rewards
- Absorbing states
- Irreversible actions

Neither the greedy nor the exploratory approach are optimal!

Safe exploration

In the real world, we have to look out for:

- Large negative rewards
- Absorbing states
- Irreversible actions

Neither the greedy nor the exploratory approach are optimal!
Three ways to solve the problem:

- Bayesian reinforcement learning
- Partially Observable MDPs
- Robust control theory approach

Bayesian reinforcement learning

We take a probabilistic approach, starting with a **prior** probability $P(h)$ on h hypotheses about what the model **is**: the policy is then given by the posterior $P(h \mid e)$:

$$\pi^* = \operatorname*{argmax}_{\pi} \sum_h P(h \mid e) U_h^{\pi}$$

Bayesian reinforcement learning

We take a probabilistic approach, starting with a **prior** probability $P(h)$ on h hypotheses about what the model **is**: the policy is then given by the posterior $P(h \mid e)$:

$$\pi^* = \operatorname*{argmax}_{\pi} \sum_h P(h \mid e) U_h^{\pi}$$

- Note that this approach assumes that the prior contains good estimates of reality
- We can turn this into an exploration POMDP where belief states are distributions over models

The robust control theory approach

We define a set of models $\mathcal{H}$, and always take the policy that maximizes utility in the **worst case** over $\mathcal{H}$:

$$\pi^* = \operatorname*{argmax}_{\pi} \min_{h} U_h^{\pi}$$

The robust control theory approach

We define a set of models $\mathcal{H}$, and always take the policy that maximizes utility in the **worst case** over $\mathcal{H}$:

$$\pi^* = \operatorname*{argmax}_{\pi} \min_{h} U_h^{\pi}$$

- This is closely tied to bayesian reinforcement learning: the set $\mathcal{H}$ is the set of models that exceed some likelihood threshold on $P(h \mid e)$
- In both cases we assume the models to be good estimates of reality

Temporal difference Q-learning

- With TD learning, we estimated the utility under a given policy

Temporal difference Q-learning

- With TD learning, we estimated the utility under a given policy
- Now we can estimate the action-utility functions in order to improve the policy

Temporal difference Q-learning

- With TD learning, we estimated the utility under a given policy
- Now we can estimate the action-utility functions in order to improve the policy

We substitute $Q(s, a)$ into Bellman's equation:

$$Q(s, a) = \sum_{s'} P(s'|s, a) \left[R(s, a, s') + \gamma \max_{a'} Q(s', a') \right]$$

Now we can derive an update:

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

SARSA learning

SARSA is a variation of TD Q-learning: instead of assuming the best possible action under known action-utility functions to be taken (**off-policy**), we wait for the model to actually execute actions (**on-policy**)

- We act after s, a, r, s', a' quintuplets

The update is given by:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R(s,a,s') + \gamma Q(s',a') - Q(s,a) \right]$$

SARSA learning

SARSA is a variation of TD Q-learning: instead of assuming the best possible action under known action-utility functions to be taken (**off-policy**), we wait for the model to actually execute actions (**on-policy**)

- We act after s, a, r, s', a' quintuplets

The update is given by:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R(s,a,s') + \gamma Q(s',a') - Q(s,a) \right]$$

TD and SARSA are subtly different:

- TD learning asks: "What does this action give if i stop using my policy, and take the best action (according to my estimates) from there onwards?"
- SARSA asks: "What did I get when i followed my policy and took this action?"