REINFORCEMENT LEARNING (David Silver, DeepMind / UCL)

How does it compare to other machine learning paradigms
- No supervisor, just reward signal
- Feedback delayed
- Time matters (sequential)
- Agent's actions affect subsequent observations

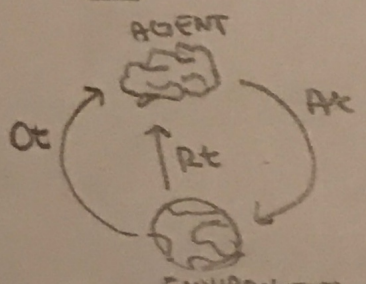
Rewards

A reward R_t is a scalar feedback signal indicating how well it is doing at step t. The goal is to maximise cumulative reward in RL.

Reward hypothesis
All goals can be described by the maximisation of expected cumulative reward

Sequential decision making

The goal is then to select actions to maximise the total future reward. Actions can have consequences which delay the reward. It may be better to sacrifice immediate reward to gain more long term reward.



At each step t the agent
- executes action A_t
- receives observation O_t
- receives scalar reward R_t

The environment
- Receives action A_t
- Emits observation O_{t+1}
- Emits scalar reward R_{t+1}

t increases at each env. step

The **history** is the sequence of observations, actions, rewards up to step t → $H_t = O_1, R_1, A_1 \dots A_{t-1}, O_t, R_t$

What happens next depends on the history. We define the **state** as information about the history.

environment state S^e_t is the environment's private representation used to pick the next reward and observation. This is usually not visible to the agent

agent state S^a_t is the agent's internal representation used to pick the next action

A **Markov state** contains all useful info from the history

A state is Markov iff $P[S_{t+1} | S_t] = P[S_{t+1} | S_1 \dots S_t]$

the future is independent from the past given the present. S_t is sufficient to use as $S_1 \dots S_t$

↓ MDP

An environment is fully observable when $O_t = S^a_t = S^e_t$ ← the agent directly observes the environment state

It is partially observable when $S^a_t \neq S^e_t$ ← the agent indirectly observes the environment state

↑ POMDP

Inside an RL agent

Policy

A policy is the ~~a~~ map from state to action

deterministic policy $a = \pi(s)$

stochastic ~ $\pi(a|s) = P[A_t = a \mid S_t = s]$

Value Function

A value function is a prediction of future rewards

value function v_π = $E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s]$

v given policy π

discount factor (0,1) makes future rewards less and less relevant

Model

A model predicts what the environment will do next

next state s' given action a and state s → $P^a_{ss'} = P[S_{t+1} = s' \mid S_t = s, A_t = a]$

$R^a_s = E[R_{t+1} \mid S_t = s, A_t = a]$

↖ next reward given action a and state s

Types of RL agents

Value based
- just value function uses

Policy based
- just policy

Actor Critic
- uses both value function and policy

Model Free
- Policy ~~or~~ and value function but no model

Model based
- Policy and/or value function, and model

Problems in sequential decision making

- Reinforcement learning
  the environment is initially unknown and the agent improves its policy by interacting with the environment
- Planning
  A model of the environment is known and the agent performs computations on its model (without doing any actions) in order to improve its policy

In RL

The agent should discover a good policy by exploring without losing too much reward along the way

Exploitation, exploit known info to maximise reward

Exploration, find more information about the environment

(both important)

Prediction, evaluate the future given a policy

Control, optimise the future by finding the best policy

Markov decision process

they formally describe an environment, where the environment is fully observable. Almost all RL problems can be formalized as MDPs

[Talked about Markov property already]

- a state with Markov property

For a Markov state s and a successor state s', the state transition probability is defined as

$$P_{ss'} = P[S_{t+1} = s' \mid S_t = s]$$

The state transition matrix defines transition probabilities for every transition from any state to another

$$P = \begin{matrix} \\ \text{from} \end{matrix} \begin{matrix} \begin{matrix} \text{to} \\ \end{matrix} \\ \begin{bmatrix} P_{11} & \dots & P_{1n} \\ \vdots & & \\ P_{n1} & \dots & P_{nn} \end{bmatrix} \end{matrix} \leftarrow \text{each row sums to 1}$$

A Markov Process is a sequence of random states $S_1, S_2 \dots$ with the Markov property. (also called Markov chain)

It is defined as a tuple $\langle S, P \rangle$ (S: finite set of states; P: transition matrix)

A Markov Reward Process is a Markov chain with values.

defined as a tuple (S, P, R, γ) (γ: discount factor; R: reward function) $R_s = E[R_{t+1} \mid S_t = s]$

The return is the total discounted reward from time-step t

given a Markov chain $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

γ avoids infinite cycles

uncertainty about the future

immediate rewards are better than delayed rewards (e.g. Finance)

The state value function $v(s)$ of an MRP is the expected return starting from state s

$$v(s) = E[G_t \mid S_t = s]$$

Bellman Equation for MRPs: decompose value function into R_{t+1} and $\gamma v(S_{t+1})$

$$\begin{aligned} v(s) &= E[G_t \mid S_t = s] \\ &= E[\underbrace{R_{t+1}}_{\text{immediate reward}} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s] \\ &= E[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= E[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s] \end{aligned}$$

$$v(s) = R_s + \gamma \sum_{s' \in S} P_{ss'} v(s')$$

A Markov decision process is a Markov reward process with decisions (all states are Markov here as well)

It is a tuple $\langle S, A, P, R, \gamma \rangle$ where A is a set of finite actions

FOR MDPs:

A policy π is a distribution over actions given states

$$\pi(A \mid s) = P[A_t = a \mid S_t = s]$$

It defines the agents actions and depends on the current state

The state value function $v_\pi(s)$ is the expected return starting from state s and following policy π $\quad v_\pi(s) = E_\pi[G_t \mid S_t = s]$

The action value function $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and the following policy π

$$q_\pi(s, a) = E_\pi[G_t \mid S_t = s, A_t = a]$$

Bellman expectation equation

$v_\pi(s) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$ For state value function

$q_\pi(s,a) = E_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$ For action value function

v_π: how good it is to be in s

q_π: how good it is to be in s and take action a

$$v_\pi(s) = \sum_{a \in A} \pi(a|s)\, q_\pi(s,a) \qquad q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s')$$

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right)$$

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a'|s')\, q_\pi(s',a')$$

Optimal value function

- The optimal state-value function $v_*(s)$ is the maximum value function over all policies. $v_*(s) = \max_\pi v_\pi(s)$
- The optimal action-value function $q_*(s,a)$ is the maximum action value function over all policies. $q_*(s,a) = \max_\pi q_\pi(s,a)$

The optimal value function specifies the best possible performance in the MDP (The MDP is "solved")

For any MDP there exists a policy π_* that is better than or equal to all other policies $\pi_* \geq \pi, \forall \pi$, given the partial ordering $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s)\ \forall s$

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in A} q_*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

There's always an optimal policy for any MDP

Bellman optimality equation (often known as Bellman equation)

$$v_*(s) = \max_a q_*(s,a)$$

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$$

$$v_*(s) = \max_a R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$$

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} q_*(s',a')$$

There are no closed form solutions and we usually use iterative method to solve the equations

POMDPs

A partially observable Markov decision process is an MDP with hidden states. It is a hidden Markov model with actions

A POMDP is a tuple $\langle S, A, O, P, R, Z, \gamma \rangle$

where O is a finite set of observations and Z is an observation function $Z_{s'o}^a = P[O_{t+1} = o \mid S_{t+1} = s', A_t = a]$

A belief state $b(h)$ is a probability distribution over states conditioned on the history h

$$b(h) = (P[S_t = s^1 \mid H_t = h] \ldots P[S_t = s^n \mid H_t = h])$$

The history H_t satisfies the Markov property

The belief state $b(H_t)$ satisfies the Markov property

Planning by Dynamic Programming

What is DP? Dynamic sequential or temporal component of the problem

Programming optimising a program i.e policy

It's a method for solving complex problems by breaking them down into subproblems. Then solve the subproblems and combine the solutions to find the solution to the original problem

Problems must have two properties

- Optimal substructure. the principle of optimality applies (the optimal solution to the subproblems tells you how to get the optimal solution to the overall problem), and the optimal solution can be decomposed into subproblems
- Overlapping subproblems. The subproblems occur many times and solutions can be cached and reused

For MDP the Bellman equation gives recursive decomposition and the value function stores and reuses solutions.
Both properties are satisfied

DP assumes the full knowledge of the MDP (transition and rewards) and is used for planning. Two special cases for planning: prediction & control

- prediction. We're given and MDP $\langle S, A, P, R, \gamma \rangle$ and π (or a MRP) and we get v_π (how good π is)
- control. We get the MDP $\langle S, A, P, R, \gamma \rangle$ and we get v_* and π_*

Iter. Policy evaluation

Evaluate a given policy π using the Bellman expectation equation

$\rightarrow v_1 \rightarrow v_2 \rightarrow \dots v_\pi \leftarrow$ after many iterations we get the true value function

arbitrary initial value function — one look-ahead using Bellman eq.

We'll use synchronous backups

- For each iteration $k+1$, for all states $s \in S$, update $v_{k+1}(s)$ from $v_k(s')$ successor state from s

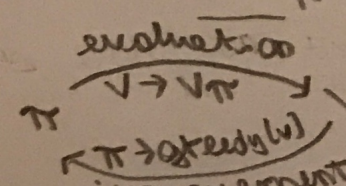
$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$

How to improve a policy (policy iteration)

Given a policy π, evaluate it (1) (as seen before) and act greedily with respect to v_π to improve it (2)

(1) $v_\pi(s) = E[R_{t+1} + \gamma R_{t+2} + \dots | S_t = s]$

(2) $\pi' = \text{greedy}(v_\pi)$ ← chose the action with the lowest $v_\pi(s)$ for π'



Keep iterating until v_π stops improving at that point we converged to π_* and v_* (optimal π and v)

Policy improvement

Consider a deterministic policy $a = \pi(s)$, we can improve the policy by acting greedily $\pi'(s) = \arg\max_{a \in A} q_\pi(s,a)$

$q_{\pi'} \geq q_\pi$ every time we act greedily

If improvement stops $q_\pi(s, \pi'(s)) = \max_{a \in A} q_\pi(s,a) = q_\pi(s, \pi(s)) = v_\pi(s)$ then the Bellman optimality equation $v_\pi(s) = \max_{a \in A} q_\pi(s,a)$ has been satisfied (that means v_π is optimal)

Therefore $v_\pi(s) = v_*(s)$ for all $s \in S$

Instead of converging, we can stop early
- When some condition for stopping is met
- Stop after k iterations of policy evaluation (does not converge with $k=1$)

Value iteration

Principle of Optimality

Any optimal policy has an optimal first action A_* followed by an optimal policy for successor state S'

Theorem (principle of optimality)
A policy $\pi(a|s)$ achieves optimal value from s, $v_\pi(s) = v_*(s)$ iff for any state s' reachable from s, π achieves the optimal value from state s', $v_\pi(s') = v_*(s')$

Deterministic value iteration

If we know the solution to subproblems $v_*(s')$, then solution $v_*(s)$ can be found by one-step lookahead

$$v_*(s) \leftarrow \max_{a \in A} R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$$

The idea of value iteration is to apply these updates iteratively
Intuition: start with final rewards and work backwards
Still works with loops stochastic MDPs

Value iteration

Problem: find optimal policy π
Solution: iterative application of Bellman optimality backup
$v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_*$

Using synchronous backups at each iteration $k+1$ for all states $s \in S$, update $v_{k+1}(s)$ from $v_k(s')$

Unlike policy iteration, there's no explicit policy as intermediate value functions may not correspond to any policy

$$v_{k+1} \max_{a \in A} \left(R_s^a + \gamma \sum_{s \in S} P_{ss'}^a v_k(s') \right)$$

$$v_{k+1} = \max_{a \in A} R^a + \gamma P^a v_k$$

Synchronous DP algorithms

Prediction	Bellman expectation equation	Iterative policy evaluation
Control	Bellman expectation equation + Greedy policy improvement	Policy Iteration
Control	Bellman optimality equation	Value Iteration

- Algorithms are based on state-value function $v_\pi(s)$ or $v_*(s)$
- Complexity $O(mn^2)$ per iteration. m actions n states
- Could also apply to action value function $q_\pi(s,a)$ or $q_*(s,a)$
- Complexity $O(m^2n^2)$ per iteration

Asynchronous DP

DP methods described use synchronous backups (ie all states are backed up in parallel)

Asynchronous DP backs up states individually, in any order. For each state, apply the appropriate backup. Can significantly reduce computation and it is guaranteed to converge if all states continue to be selected

In-place DP

Synchronous value iteration stores two copies of value function

for all s in S $v_{new}(s) \leftarrow \max_{a \in A}\left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_{old}(s')\right)$

$v_{old} \leftarrow v_{new}$

In-place value iteration only stores one copy of value function

for all s in S $v(s) \leftarrow \max_{a \in A}\left(R_s^a + \gamma \sum P_{ss'}^a v(s')\right)$

Prioritised sweeping

Use magnitude of Bellman error to guide state selection

eg. $\left|\max_{a \in A}\left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v(s')\right) - v(s)\right|$

- Backup the state with the largest remaining Bellman error
- Update Bellman error of affected states after each backup
- Requires knowledge of reverse dynamics (predecessor states)
- Can be implemented efficiently by maintaining a priority q

Real-time programming

- Idea only states that are relevant to agent
- Use agent's experience to guide the selection of states
- After each time-step S_t, A_t, R_{t+1}
- Backup the state S_t

$$v(s_t) \leftarrow \max_{a \in A}\left(R_{s_t}^a + \gamma \sum_{s' \in S} P_{s_t s'}^a v(s')\right)$$