

NOVA

IMS

Information
Management
School

Reinforcement Learning

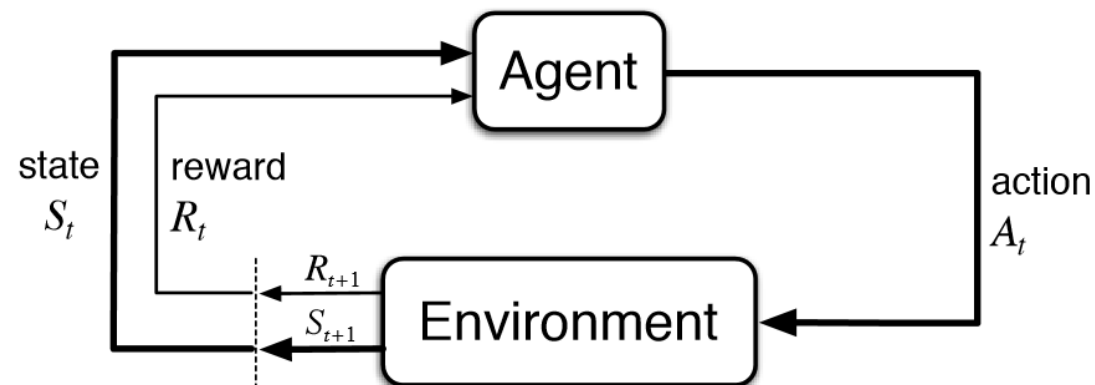
Lecture 5

Model Free Control

May 2025

Nuno Alpalhão
nalpalhao@novaims.unl.pt

- All components are functions
 - Policies $\pi : S \rightarrow A$ (or to probabilities over A)
 - Value functions $v : S \rightarrow R$
 - Q functions $q : S \times A \rightarrow R$
 - Models $m : S \rightarrow S$ and/or $r : S \rightarrow R$
 - State update $u : S \times O \rightarrow S$



First visit Monte Carlo

In Monte Carlo methods, **we approximate the value function by taking the average return.**

In the **first visit Monte Carlo method**, we average the return only the first time the state is visited in an episode.

If the agent **revisits the state**, **we don't consider an average return.** We consider an average return only when the agent visits the state for the first time.

Every visit Monte Carlo

In every visit Monte Carlo, we average the return **every time the state is visited in an episode.**

Update $v(s)$ **incrementally** after each episode:

For each state S_t with return G_t :

- $N'(S_t) \leftarrow N(S_t) + 1$
- $v'(S_t) \leftarrow v(S_t) + \frac{1}{N(S_t)} (G_t - v(S_t))$

In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes:

$$v'(S_t) \leftarrow v(S_t) + \alpha (G_t - v(S_t))$$

Generally high variance estimator:

- Reducing variance can require a lot of data.
- In cases where data is very hard or expensive to acquire, or the stakes are high, Monte Carlo may be impractical.

Requires episodic settings:

- Episode must end before data can be used to update the value function.

“If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-difference (TD) learning.” —
Sutton and Barto 2017

Temporal Difference methods **learn immediately** from each step of an **episode**.

Temporal Difference **is model-free**: no knowledge of MDP transitions / rewards.

Temporal Difference learns from incomplete episodes, by **bootstrapping**.

Temporal Difference can be used in episodic or **infinite-horizon** non-episodic settings.

Bootstrapping: update involves an estimate:

- Monte Carlo does not bootstrap.
- Dynamic Programming bootstraps.
- Temporal Differences learning bootstraps.

Sampling: update samples an expectation:

- Monte Carlo samples.
- Dynamic Programming does not sample.
- Temporal Differences learning samples.

We can apply the same idea to the Q function (action values)

Temporal-difference learning for action values:

- Update value $q(S_t, A_t)$
- Towards estimated return $R_{t+1} + \gamma \cdot q(S_{t+1}, A_{t+1})$

$$q'(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha [R_{t+1} + \gamma \cdot q(S_{t+1}, A_{t+1}) - q(S_t, A_t)]$$

This algorithm is known as **SARSA**.

Temporal Difference is model-free (no knowledge of MDP) and learn directly from experience.

Temporal Difference can learn from incomplete episodes, by bootstrapping.

Temporal Difference can learn during each episode.

Temporal Difference can learn **before knowing** the final outcome:

- Temporal Difference can learn online after every step.
- Monte Carlo must wait until end of episode before return is known.

Temporal Difference can learn **without the final outcome**:

- Temporal Difference can learn from incomplete sequences.
- Monte Carlo can only learn from complete sequences.
- Temporal Difference works in continuing (non-terminating) environments.
- Monte Carlo only works for episodic (terminating) environments.

Temporal Difference is independent of the temporal span of the prediction:

- Temporal Difference can learn from single transitions.
- Monte Carlo must store all predictions (or states) to update at the end of an episode.

Temporal Difference needs reasonable value estimates.

Temporal Differences **exploits Markov** property:

- Can help in **fully-observable** environments.

Monte Carlo **does not exploit Markov** property:

- Can help in **partially-observable** environments.

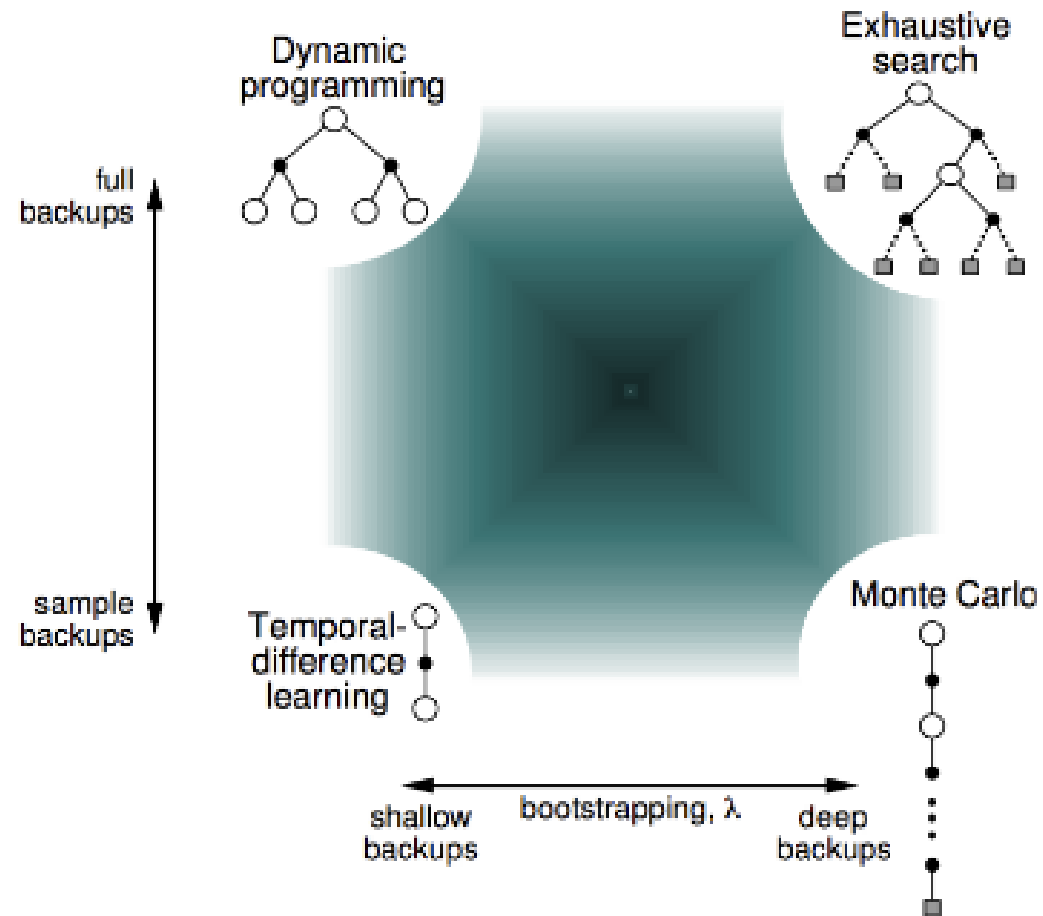
With finite data, or with function approximation, **the solutions may differ.**

Monte Carlo has **high variance, zero bias**:

- Good convergence properties.
- Not very sensitive to initial value.
- Very simple to understand and use.

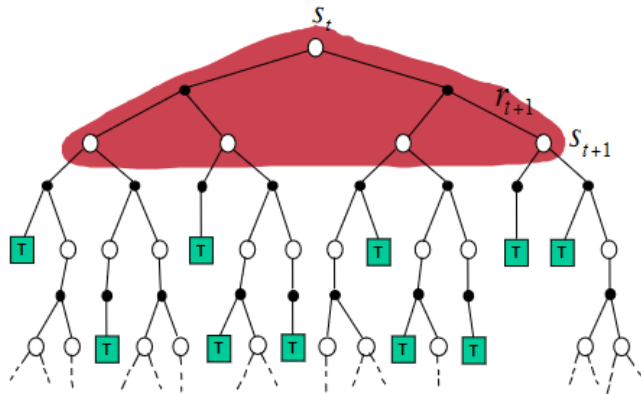
Temporal Differences **has low variance, some bias**:

- Usually more efficient than Monte Carlo.
- Temporal Differences usually converges to $v_{\pi}(s)$.
- More sensitive to initial value.



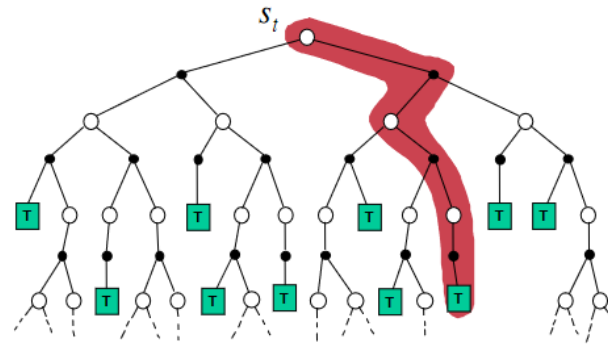
Visual Interpretation

$$v(S_t) \leftarrow \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) \mid A_t \sim \pi(S_t)]$$



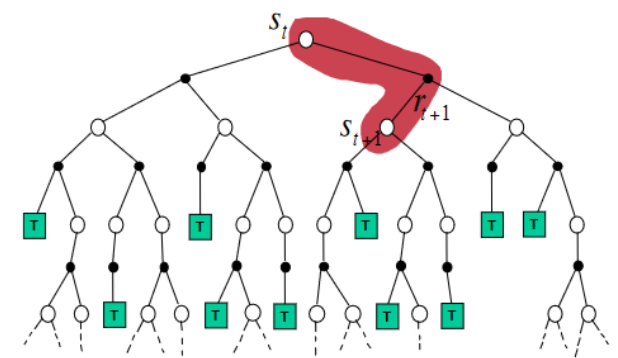
Dynamic Programming

$$v(S_t) \leftarrow v(S_t) + \alpha (G_t - v(S_t))$$



Monte Carlo

$$v(S_t) \leftarrow v(S_t) + \alpha (R_{t+1} + \gamma v(S_{t+1}) - v(S_t))$$

Temporal Differences
TD(0)

Let's discuss **control** methods!

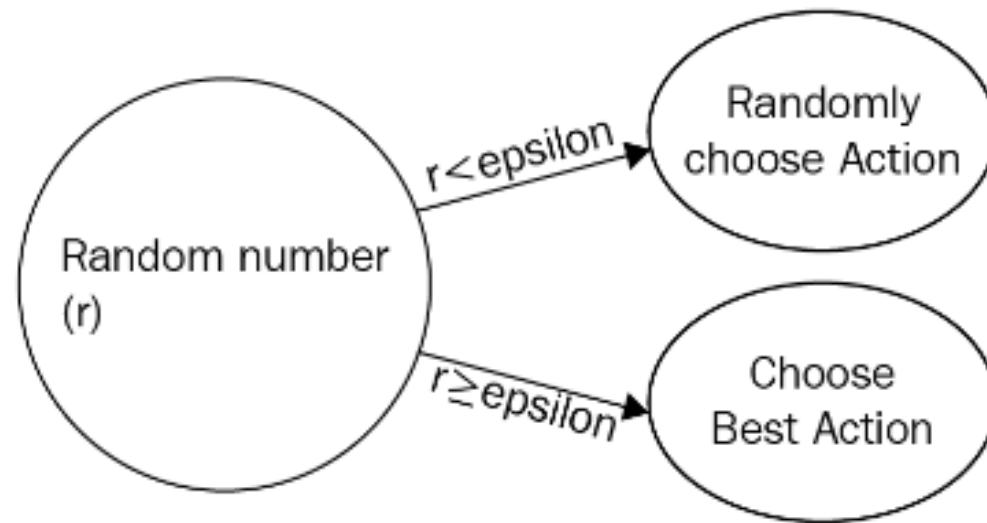
Remember two of the fundamental problems in reinforcement learning

- **Exploration** finds more information about the environment.
- **Exploitation** exploits known information to maximise reward.
- It is usually important to explore as well as exploit.

Simplest idea for ensuring continual exploration.

All actions are tried with non-zero probability.

With a given probability choose the greedy action.

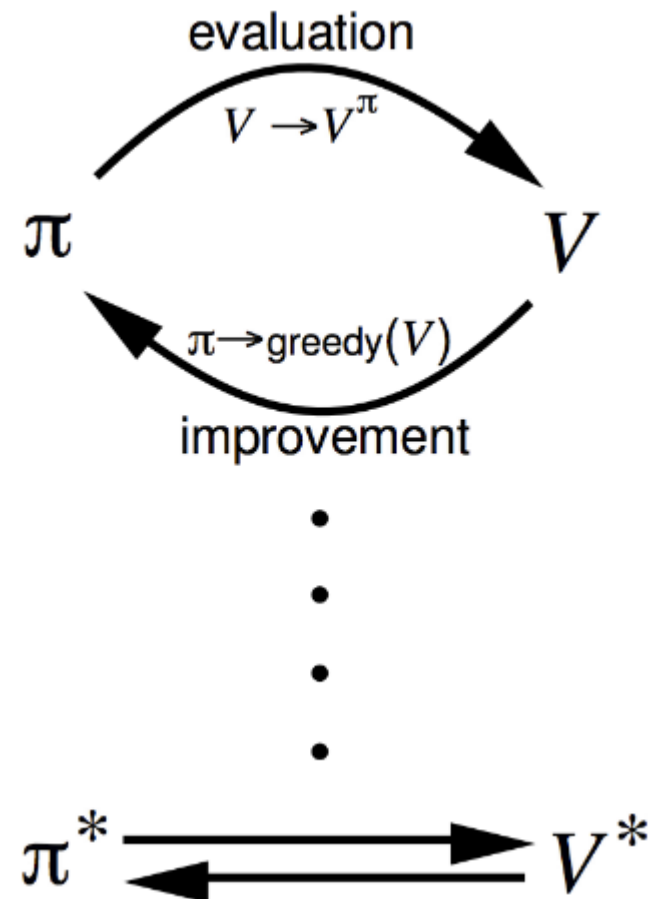
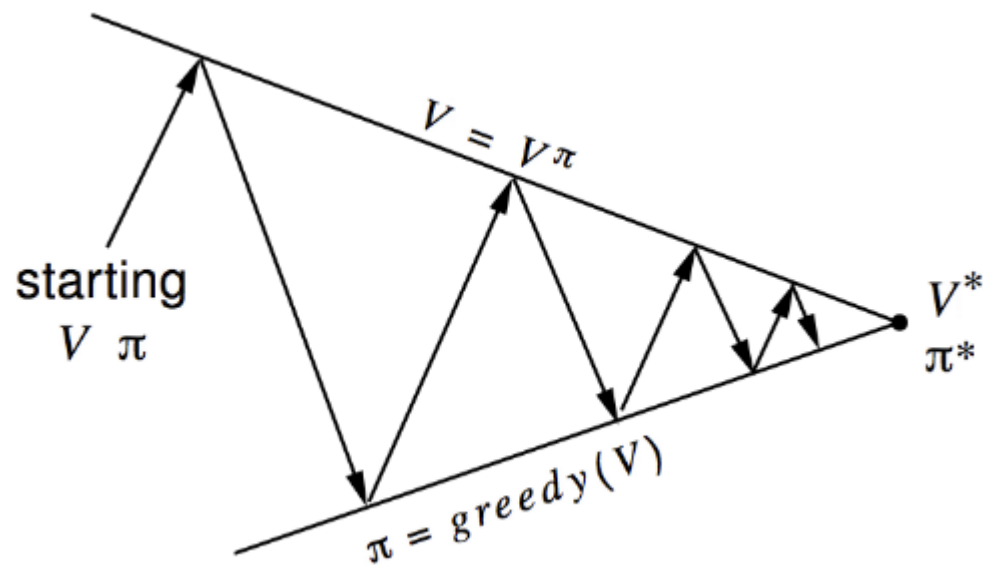


Model Free Control

Model Free Policy Iteration

Policy Iteration is seen as a **greedy** approach.

The idea is to sequentially improve a value function (**policy evaluation**) and then a policy (**policy improvement**).



Greedy policy improvement over $v(s)$ requires model of MDP:

$$\pi'(s) = \operatorname{argmax}_a E [\mathbf{R}_{t+1} + \gamma \cdot v(\mathbf{S}_{t+1}) \mid S_t = s, A_t = a]$$

Greedy policy improvement over $q(s, a)$ is **model free**:

$$\pi'(s) = \operatorname{argmax}_a q(s, a)$$

This makes action values convenient.

Definition

Greedy in the Limit with Infinite Exploration (GLIE)

- All state-action pairs are explored infinitely many times:

$$\forall s, a \quad \lim_{t \rightarrow \infty} N(s, a) = \infty$$

- The policy converges to a greedy policy:

$$\lim_{t \rightarrow \infty} \pi_t(a|s) = I[a = \operatorname{argmax}_{a'} q(s, a')]$$

Theorem

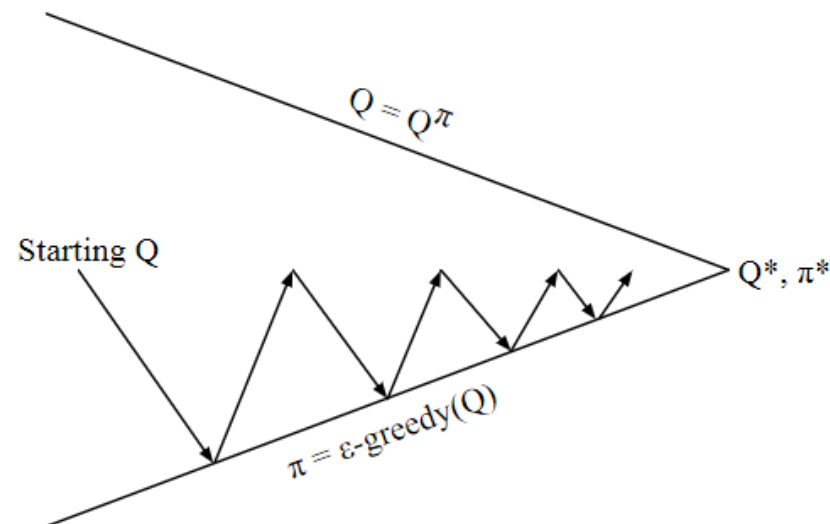
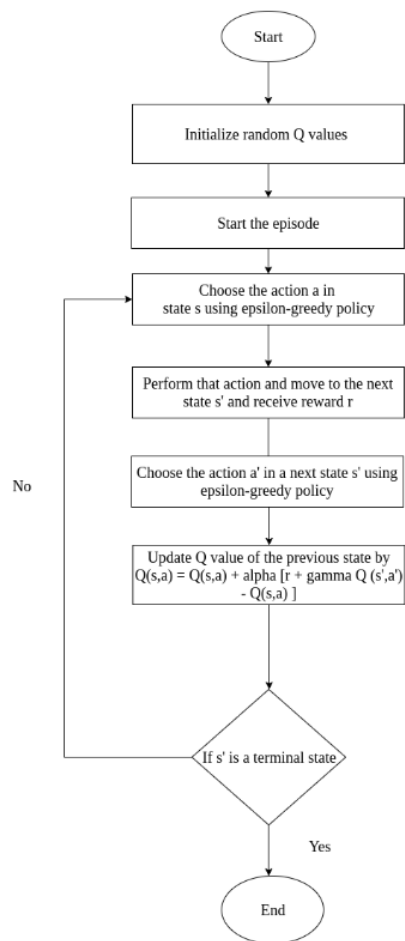
GLIE Model-free control **converges to the optimal action-value function**, $q \rightarrow q_*$

Temporal-difference learning has several advantages over Monte-Carlo:

- Lower variance.
- Online.
- Can learn from incomplete sequences.

Model Free Control

Updating Action-Value Functions with SARSA



Every time-step:

Policy evaluation SARSA, $q \approx q_\pi$

Greedy policy improvement

$$q'(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha [R_{t+1} + \gamma \cdot q(S_{t+1}, A_{t+1}) - q(S_t, A_t)]$$

Theorem

Tabular SARSA **converges to the optimal action-value function**,
 $q \rightarrow q_*$, if the policy is GLIE.

Dynamic programming

- Policy Iteration
- Value Iteration

Monte Carlo

Temporal Differences

- Temporal Differences (SARSA)
- Q-Learning

On-policy learning

The agent learns the policy it is currently following.

The behaviour policy (used to generate actions) is the same as the target policy (the one being improved).

Off-policy learning

The agent learns the value of a different policy from the one it is currently following.

The behaviour policy is different from the target policy.

Q-learning estimates the value of the greedy policy

$$q'(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \cdot \max_a q(S_{t+1}, a) - q(S_t, A_t) \right]$$

Q-learning is off-policy while SARSA is on-policy.

Reminder of **SARSA**:

$$q'(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha [R_{t+1} + \gamma \cdot q(S_{t+1}, A_{t+1}) - q(S_t, A_t)]$$

Theorem

Q-learning control **converges to the optimal action-value function**, $q \rightarrow q_*$, as long as we take each action in each state infinitely often.

Classical Q-learning has potential issues.

Uses same values to select and to evaluate.

Values are approximate:

- more likely to select overestimated values.
- less likely to select underestimated values.

This causes upward bias.

Q-learning overestimates because it uses the same values to select and to evaluate.

$$\max_{a'} q(S_{t+1}, a') = q\left(S_{t+1}, \operatorname{argmax}_{a'} q(S_{t+1}, a')\right)$$

Solution: decouple selection from evaluation

- Store two action-value functions: q and q^*

$$\max_{a'} q(S_t, a') = \mathbf{q}^* \left(S_t, \operatorname{argmax}_{a'} q(S_t, a') \right) \quad (1)$$

$$\max_{a'} \mathbf{q}^*(S_t, a') = q \left(S_t, \operatorname{argmax}_{a'} \mathbf{q}^*(S_t, a') \right) \quad (2)$$

- Each *step*, pick q or \mathbf{q}^* and update using (1) for q or (2) for \mathbf{q}^* .
- Can use both to act (e.g., use policy based average of both).

[1509.06461 \(arxiv.org\)](https://arxiv.org/abs/1509.06461)

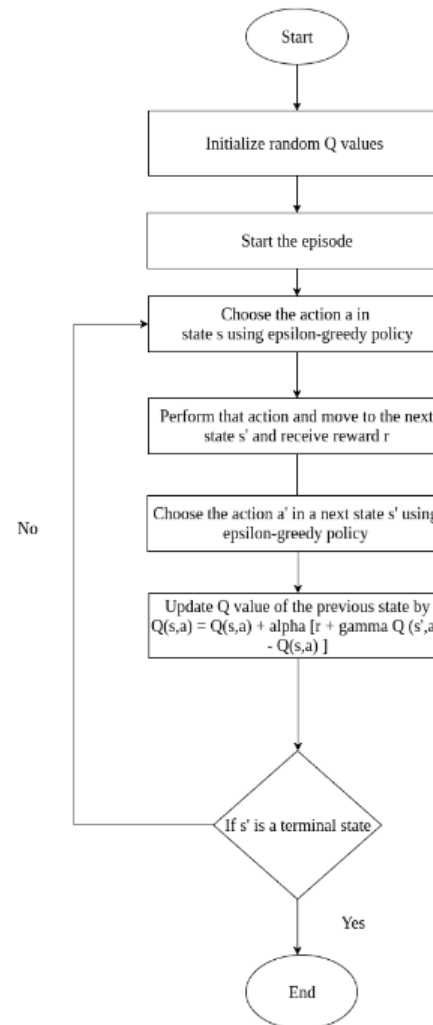
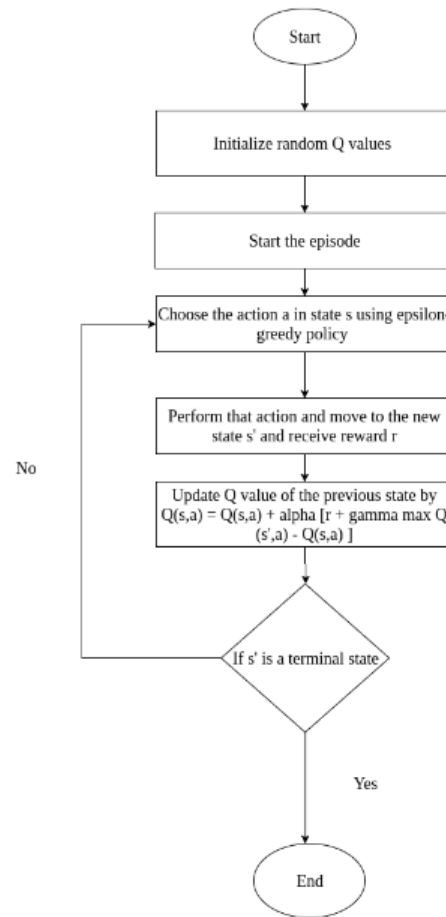
Double Q-learning **also converges to the optimal policy** under the same conditions as Q-learning.

The idea of double Q-learning can be generalised to other updates:

- Double SARSA.

Model Free Control

SARSA vs Q-learning



Off-policy learning means learning about one policy π from experience generated according to a different policy μ .

Q-learning is one example, but there are other options.

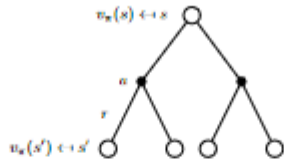

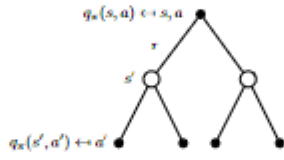
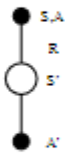
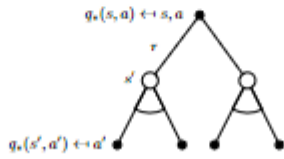
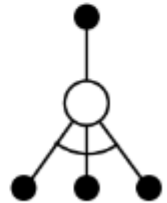
Q-learning uses a greedy target policy.

SARSA uses a stochastic sample from the behaviour as target policy.

Double learning uses a separate value function to evaluate the policy (for any policy).

Double learning is not necessary if there is no correlation between target policy and value function (e.g., pure prediction).

When using a greedy policy (Q-learning), there are strong correlations. Then double learning (Double Q-learning) can be useful.

	<i>Full Backup (DP)</i>	<i>Sample Backup (TD)</i>
Bellman Expectation Equation for $v_{\pi}(s)$	 <p>Iterative Policy Evaluation</p>	 <p>TD Learning</p>
Bellman Expectation Equation for $q_{\pi}(s, a)$	 <p>Q-Policy Iteration</p>	 <p>Sarsa</p>
Bellman Optimality Equation for $q_{*}(s, a)$	 <p>Q-Value Iteration</p>	 <p>Q-Learning</p>

<i>Full Backup (DP)</i>	<i>Sample Backup (TD)</i>
Iterative Policy Evaluation $V(s) \leftarrow \mathbb{E}[R + \gamma V(S') \mid s]$	TD Learning $V(S) \stackrel{\alpha}{\leftarrow} R + \gamma V(S')$
Q-Policy Iteration $Q(s, a) \leftarrow \mathbb{E}[R + \gamma Q(S', A') \mid s, a]$	Sarsa $Q(S, A) \stackrel{\alpha}{\leftarrow} R + \gamma Q(S', A')$
Q-Value Iteration $Q(s, a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S', a') \mid s, a\right]$	Q-Learning $Q(S, A) \stackrel{\alpha}{\leftarrow} R + \gamma \max_{a' \in \mathcal{A}} Q(S', a')$

where $x \stackrel{\alpha}{\leftarrow} y \equiv x \leftarrow x + \alpha(y - x)$

The policy, value function, model, and agent state update are all functions.

We want to learn these from experience.

If there are too many states, we need to approximate.

It is often called deep reinforcement learning, when using neural networks to represent these functions.

So far we mostly considered lookup tables:

- Every state s has an entry $v(s)$.
- Or every state-action pair s, a has an entry $q(s, a)$.

Problem with large MDPs:

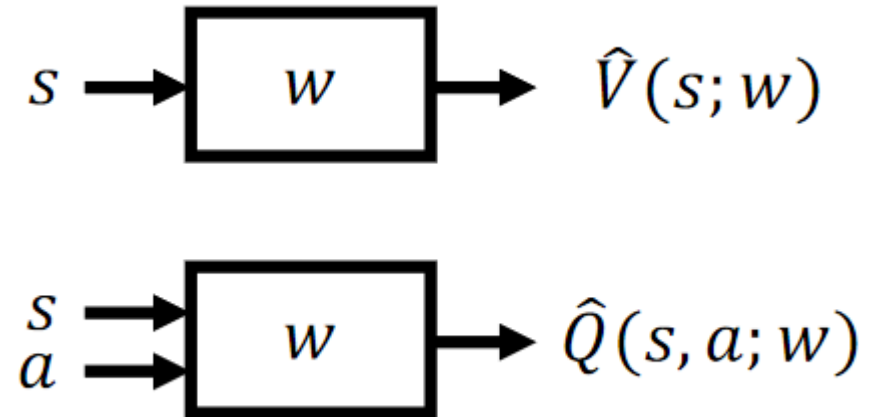
- There are too many states and/or actions to store in memory.
- It is too slow to learn the value of each state individually.
- Individual environment states are often not fully observable.

Want more compact representation that generalizes across state or states and actions.

Solution for large MDPs:

- Estimate value function with function approximation:

$$\begin{aligned} v_w(s) &\approx v_\pi(s) && \text{(or } v_*(s)) \\ q_w(s, a) &\approx q_\pi(s, a) && \text{(or } q_*(s, a)) \end{aligned}$$



- Update parameter w (e.g., using MC or TD learning).
- Generalise to unseen states.

Many possible function approximators including:

- **Linear combinations of features.**
- **Neural networks.**
- Decision trees.
- Nearest neighbours.
- Fourier/ wavelet bases.

We will focus on function approximators that are **easily differentiable**.

Tabular:

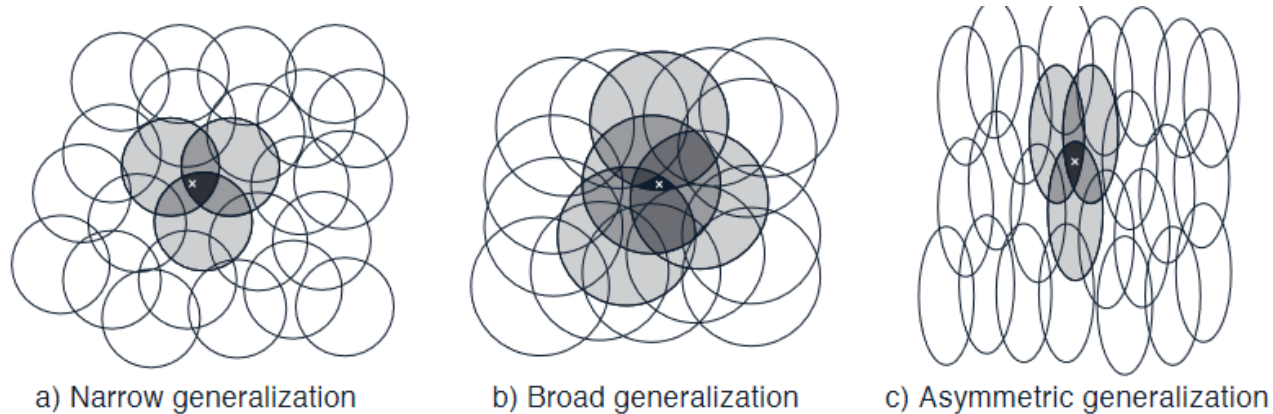
- A table with an entry for each MDP state.

State aggregation:

- Partition environment states (or observations) into a discrete set.

State aggregation Methods (i.e. Coarse Coding)

Consider a task in which the state set is continuous and two-dimensional. A state in this case is a point in 2-space, a vector with two real components.



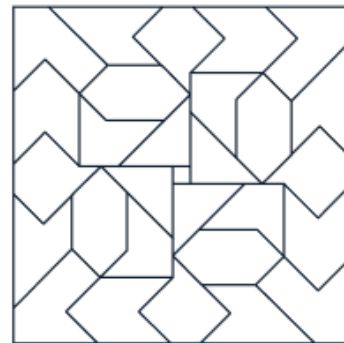
If the state is inside a circle, then the corresponding feature has the value 1 and is said to be present; otherwise the feature is 0 and is said to be absent (Binary feature).

State aggregation Methods

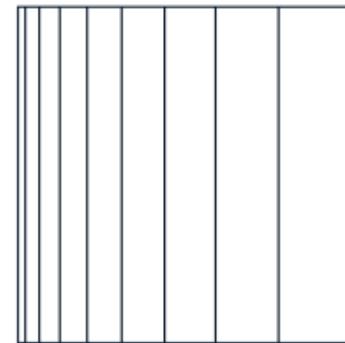
Tile Coding (We've already mentioned this approach)

Radial Basis Functions

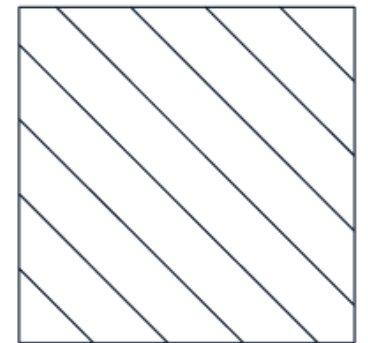
Kanerva Coding



a) Irregular



b) Log stripes



c) Diagonal stripes

Linear function approximation

- Consider fixed agent state update (e.g., $S_t = O_t$).
- Fixed feature map $x : S \rightarrow R^n$
- Values are linear functions of features.

Note: state aggregation and tabular are special cases of linear Function Approximation.

Differentiable function approximation

- $v_w(s)$ is a differentiable function of w , could be non-linear.
- E.g., a convolutional neural network that takes pixels as input.

Another interpretation: features are not fixed, but learnt.

In principle, any function approximator can be used, but RL has specific properties:

- **Experience is not i.i.d.** - successive time-steps are correlated.
- Agent's policy affects the data it receives.

Regression targets can be **non-stationary**:

- Changing policies (which can change the target and the data!).
- Bootstrapping.
- Non-stationary dynamics (e.g., other learning agents).
- Problem is large (never quite arriving to the same state again).

Which function approximation should you choose?

- **Tabular**: good theory but does not scale/generalise.
- **Linear**: reasonably good theory but requires good features.
- **Non-linear**: less well-understood, but scales well.

Flexible, and less reliant on picking good features first (e.g., by hand).

- **(Deep) neural networks** often perform quite well and remain a popular choice.

Thank You!

Morada: Campus de Campolide, 1070-312 Lisboa, Portugal

Tel: +351 213 828 610 | **Fax:** +351 213 828 611

Acreditações e Certificações da NOVA IMS

Cofinanciado por



UNIGIS



Computing
Accreditation
Commission

