MLMI7: Reinforcement Learning and Decision Making

Temporal-difference methods

Presented by David Krueger and Carl Rasmussen

Slides prepared by Milica Gašić

Lent Term

In this lecture...

Introduction to temporal-difference learning

SARSA: On-policy TD control

Q-learning: Off-policy TD control

Planning and learning with tabular methods

Temporal-difference (TD) learning

Dynammic programming update estimates based in part on other learned estimates, without waiting for the final outcome (they bootstrap)

Monte Carlo methods learn directly from raw experience without a model of the environment's dynamics

TD learning uses both bootstrapping and sampling to estimate value.

TD prediction

- TD methods only wait until the next time step to update the value estimates.
- At time $t+1$ they immediately form a target and make an update using the observed reward r_{t+1} and the current estimate $V(s_{t+1})$.

$$V(s_t) \leftarrow V(s_t) + \alpha \left(r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right),$$

  where $\alpha > 0$ is a step-size parameter.
- Note that this is similar to the MC update except that it takes place at every step.
- Similar to DP methods, the TD method bases its update in part on an existing estimate – a bootstrapping method.

TD error

TD error arises in various forms through-out reinforcement learning

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

The TD error at each time is the error in the estimate made at that time. Because the TD error at step t depends on the next state and next reward, it is not actually available until step $t+1$. Updating the value function with the TD-error is called a **backup**. The TD error is related to the Bellman equation.

SARSA: On-policy TD control

- TD prediction for control ie action-selection
- A generalised policy iteration method
- Balances between exploration and exploitation
- Learns tabular Q-function

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right)$$

This update is done after every transition from a non-terminal state s_t. If s_{t+1} is terminal, then $Q(s_{t+1}, a_{t+1})$ is defined as zero. This rule uses every element of the quintuple of events, $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, hence the name.

SARSA: On-policy TD control

Algorithm 1 SARSA

Initialise Q arbitrarily, $Q(terminal, \cdot) = 0$
repeat
  Initialize s
  Choose a ϵ-greedily
  repeat
    Take action a, observe r, s'
    Choose a' ϵ-greedily
    $Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma Q(s', a') - Q(s, a)\right)$
    $s \leftarrow s'$,$a \leftarrow a'$
  until s is terminal
until convergence

Properties of SARSA

- SARSA is an **on-policy** algorithm which means that while learning the optimal policy it uses the current estimate of the optimal policy to generate the behaviour.
- SARSA converges to an **optimal policy** as long as all state-action pairs are visited an infinite number of times and the policy converges in the limit to the greedy policy ($\epsilon = \frac{1}{t}$).

Q-learning: Off-Policy TD Control

In Q-learning the learned action-value function, Q, directly approximates the optimal action-value function, independent of the policy being followed.

$$Q(s_t, a_t) \leftarrow (s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right)$$

This dramatically simplifies the analysis of the algorithm and enabled early convergence proofs: all that is required for correct convergence is that all pairs continue to be updated.

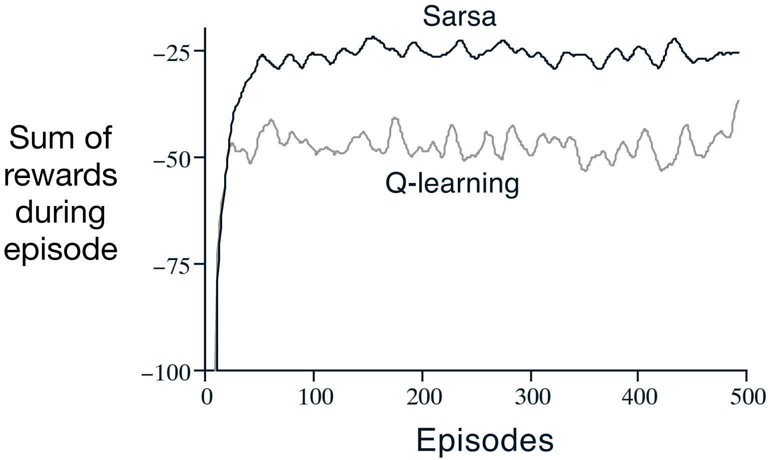
Q-learning: Off-policy TD control

Algorithm 2 Q-learning

Initialise Q arbitrarily, $Q(terminal, \cdot) = 0$
repeat
    Initialize s
    repeat
        Choose a ϵ-greedily
        Take action a, observe r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right)$
        $s \leftarrow s'$
    until s is terminal
until convergence

SARSA vs Q-learning

Comparison of the SARSA and the Q-learning algorithm on the cliff-walking task (a variant of grid-world). The results show the advantage of on-policy methods during the learning process.



Expected Sarsa

- An alternative to taking a random action and using the estimate of the Q-function for that action in TD-error (as in SARSA) is to use the expected value of the Q-function.

$$
\begin{aligned}
Q(s_t, a_t) &\leftarrow Q(s_t, a_t) + \alpha \left(E[Q(s_{t+1}, a_{t+1}) \mid s_{t+1}] - Q(s_t, a_t) \right) \\
&= Q(s_t, a_t) + \\
&\alpha \left(r_{t+1} + \gamma \sum_{a'} \pi(a'|s_{t+1}) Q(s_{t+1}, a') - Q(s_t, a_t) \right)
\end{aligned}
$$

- Although computationally more complex, this method has a lower variance.
- Generally performs better and it can be either on-policy or off-policy.

Summary

- Prediction: the value function must accurately reflect the policy
- Improvement: the policy must improve locally (eg ϵ-greedy) with respect to the current value function
- SARSA is an on-policy TD method
- Q-learning is an off-policy TD method
- Expected SARSA can be either an on-policy or an off-policy method
- They can be applied on-line, with a minimal amount of computation, to learn from interaction with an environment

Planning and learning with tabular methods

A unified view of

Planning Methods which require the model of the environment

Learning Methods which do not require the model of the environment

Models and planning

Model of the environment – anything that an agent can use to predict how the environment will respond to its actions. Models can be used to *simulate* experience: given a starting state and action, the model produces a possible transition.

Planning – any computational process that takes a model as input and produces or improves a policy for interacting with the modelled environment.

$$\textbf{Model} \xrightarrow{\quad\textbf{Planning}\quad} \textbf{Policy}$$

Planning

Planning is based on two basic ideas:

1. all state-space planning methods involve computing value functions as a key intermediate step toward improving the policy
2. they compute their value functions by backup operations (TD updates) applied to simulated experience.

Dyna: integrating planning, acting, and learning

A planning agent can be used to:

model-learning improve the model (to match the real environment)

reinforcement learning directly improve the value function and policy

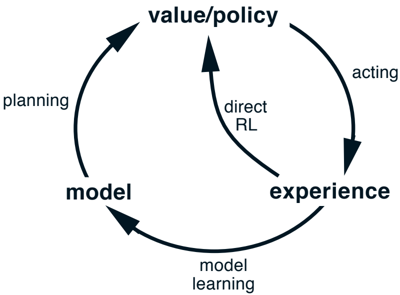


Figure 1: Planning agent

Dyna-Q

Dyna-Q includes all of the processes shown in Figure 1: planning, acting, model-learning, and direct RL – all occurring continually.

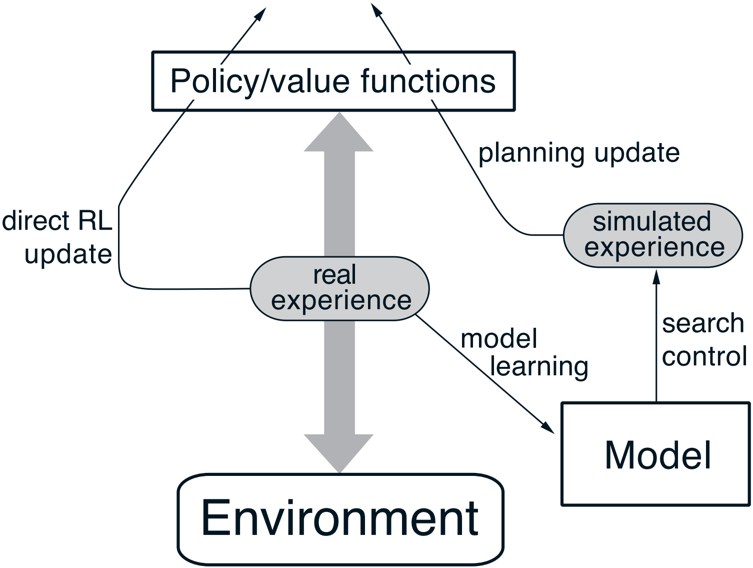
Planning the Q-learning applied to samples from the model

Model-learning table-based and assumes the world is deterministic

RL after each transition $s_t, a_t; \rightarrow r_{t+1}, s_{t+1}$, the model records in its table entry for s_t, a_t the prediction that r_{t+1}, s_{t+1} will deterministically follow.

The planning algorithm randomly samples only from state-action pairs that have previously been experienced, so the model is never queried with a pair about which it has no information.

Dyna architecture



Tabular Dyna-Q

Algorithm 3 Tabular Dyna-Q

Initialise $Q(s, a)$ and $Model(s, a)$ arbitrarily
repeat
    Initialize s
    Choose a ϵ-greedily
    Take action a, observe r, s' {real experience}
    $Q(s, a) \leftarrow Q(s, a) + \alpha\,(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$ {RL}
    $Model(s, a) \leftarrow r, s'$ {model learning deterministically}
    repeat
        s, a random previously observed state-action pair {search control}
        $r, s' \leftarrow Model(s, a)$ {simulated experience}
        $Q(s, a) \leftarrow Q(s, a) + \alpha\,(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$ {planning}
        $s \leftarrow s'$
    until n times
until convergence

Dyna: properties

- Learning and planning are accomplished by exactly the same algorithm, operating on real experience for learning and on simulated experience for planning.
- Planning proceeds incrementally, it is trivial to intermix planning and acting.
- The agent responds instantly to the latest sensory information and yet always plans in the background.
- As new information is gained, the model is updated to better match reality.

Prioritised sweeping

- Upto now simulated transitions in state-action pairs are selected uniformly at random from all previously experienced pairs.
- The number of updates grows rapidly but not all updates are equally useful.
- The value of some state-action pairs have changed a lot while the value of other state-action pairs has changed little.
- In a stochastic environment, variations in estimated transition probabilities also contribute to the magnitude of the change.

Prioritised sweeping

Prioritize the backups according to a measure of their urgency

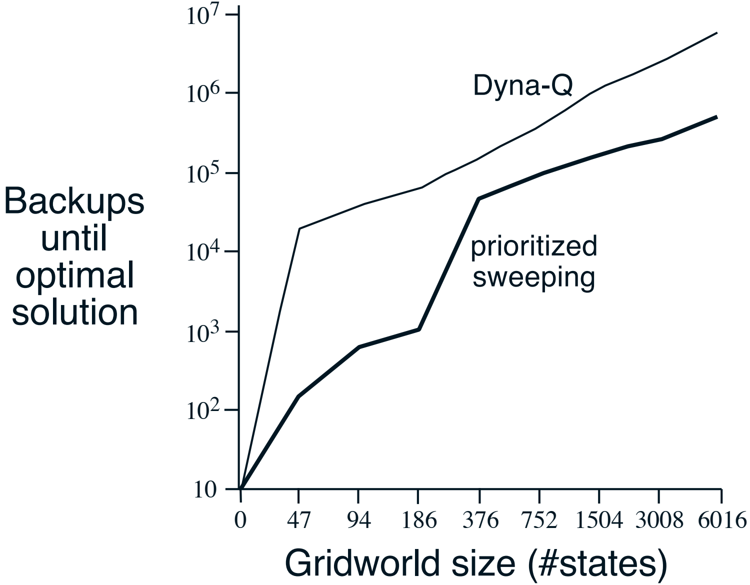
- Base urgency on the TD-error
- If a state-action pair was updated all state-actions preceding this pair must be updated too.
- Perform the backups in order of priority

Prioritised sweeping for a deterministic env.

Algorithm 4 Prioritised sweeping

Initialise $Q(s,a)$, $Model(s,a)$ arbitrarily and $PQueue$ to empty
repeat
  Initialize s; choose a ϵ-greedily
  Take action a, observe r, s'; $Model(s,a) \leftarrow r, s'$
  $P \leftarrow |r + \gamma \max_{a'} Q(s',a') - Q(s,a)|$ {TD-error}
  if $P > \theta$ then insert s, a into $PQueue$ with priority P
  repeat
    $s, a \leftarrow first(PQueue)$, $r, s' \leftarrow Model(s,a)$
    $Q(s,a) \leftarrow Q(s,a) + \alpha\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$
    for all $\hat{s}, \hat{a}$ predicted to lead to s **do**
      $\hat{r}$ reward for $\hat{s}, \hat{a}, s$; $P \leftarrow |\hat{r} + \gamma \max_{a} Q(s,a) - Q(\hat{s},\hat{a})|$
      if $P > \theta$ then insert $\hat{s}, \hat{a}$ into $PQueue$ with priority P
    end for
  until $PQueue$ empty
until convergence

Benefit of prioritised sweeping



Summary

- Planning optimal behaviour and learning optimal behaviour involve estimating the same value functions.
- Any of the learning methods can be converted into planning methods simply by applying them to simulated (model-generated) experience rather than to real experience.
- Prioritized sweeping orders the updates according to the urgency and it can lead to the optimal solution with less updates
- Prioritized sweeping focuses backward on the predecessors of states whose values have recently changed significantly.

Next lecture

- Approximate solutions and function approximation