

Reinforcement Learning

Lecture 9: Model-based methods

Robert Lieck

Durham University



Lecture covers chapter 8 in Sutton & Barto [1] and examples from David Silver [2]

1 Model-based reinforcement learning

- taxonomy
- overview
- the simulation cycle
- characteristics

2 Integrated learning and planning

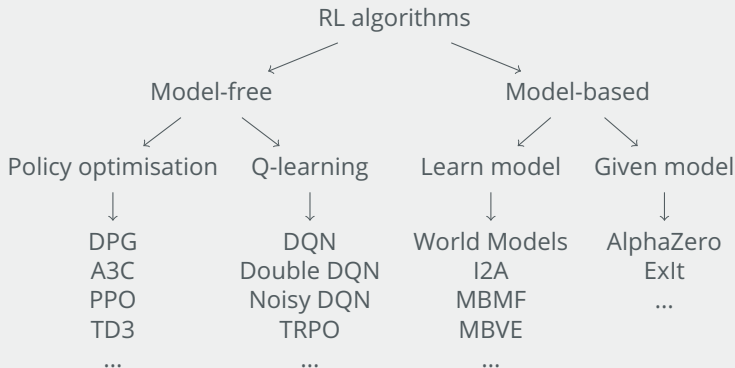
- Dyna-Q
- characteristics
- Monte Carlo tree search

3 Model-based policy optimization

- Model ensembles
- Model rollouts

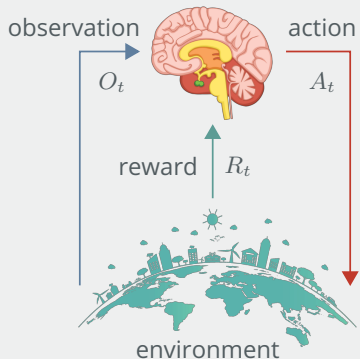


Taxonomy of reinforcement learning algorithms



This figure does not capture overlap, for example between policy optimisation and Q-learning algorithms

RL Agents

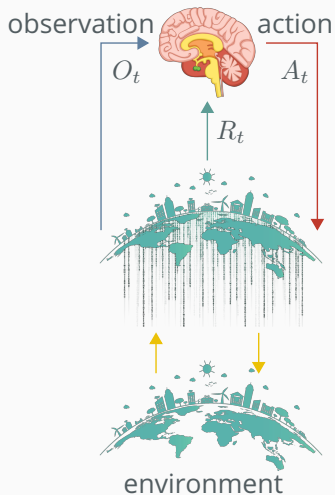


In **model-free** RL:

- No model
- **Learn** the value function $q(s, a)$ and/or the policy $\pi(a|s)$ from experience

In **model-based** RL:

- Learn the model from experience
- **Plan** the value function and/or the policy from the model



Model-based RL cycle:

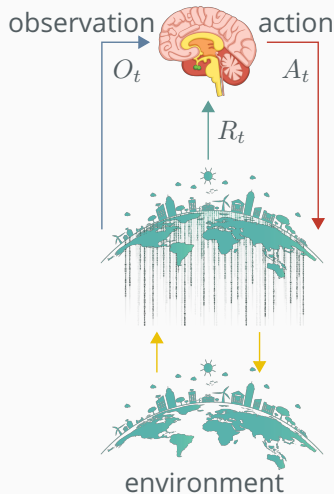
- The agent experiences the real environment
- We learn a model to predict what the real environment does (when you take an action)
- We then use this simulated model to plan
- This allows us to estimate the value function and/or policy without directly interacting with the real environment
- But we use this policy to take real actions again

Model-based RL advantages:

- The model may be much simpler than a value function
- Can be learnt by supervised learning
- Can reason about model uncertainty
- Can provide additional information (e.g. probabilities)
- Can change the reward function but keep the model

Model-based RL disadvantages:

- Another component that may introduce errors
- We can only be as good as our model
- Execution requires planning



Definition: model

A model $\mathcal{M} = \langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$ is a parameterised η representation of an MDP: $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$. It approximates state transitions $\mathcal{P}_\eta \approx \mathcal{P}$ and rewards $\mathcal{R}_\eta \approx \mathcal{R}$, learning a distribution over the next states and rewards:

$$S_{t+1} \sim \mathcal{P}(S_{t+1} | S_t, A_t)$$

$$R_{t+1} = \mathcal{R}(R_{t+1} | S_t, A_t),$$

which typically are conditionally independent of each other:

$$P(S_{t+1}, R_{t+1} | S_t, A_t) = \\ P(S_{t+1} | S_t, A_t) P(R_{t+1} | S_t, A_t)$$

Example: environment model





Learning the model

We learn the model \mathcal{M}_η from experience $\{S_1, A_1, R_2, \dots, S_T\}$ using **supervised learning**.

- We receive a stream of actual experiences
- This gives us a dataset:

$$S_1, A_1 \rightarrow R_2, S_2$$

$$S_2, A_2 \rightarrow R_3, S_3$$

...

- $s, a \rightarrow r$ is a regression problem
- $s, a \rightarrow s'$ is a density estimation problem

Experience can be simulated and real

Simulated experience sampled from \mathcal{M}_η

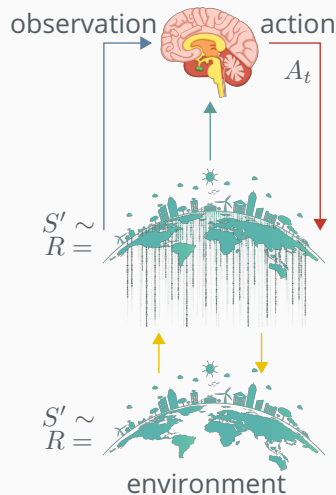
$$S' \sim \mathcal{P}_\eta(S'|S, A)$$

$$R = \mathcal{R}_\eta(R|S, A)$$

Real experience sampled from the true MDP

$$S' \sim \mathcal{P}_{s,s'}^a$$

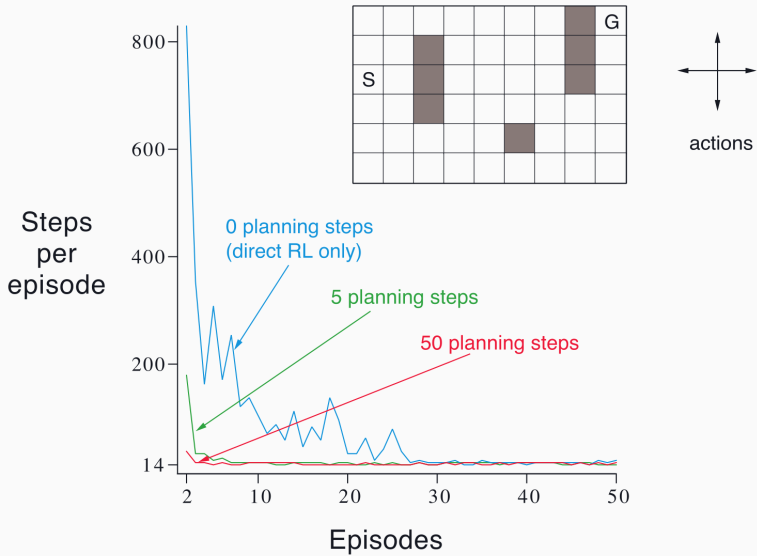
$$R = \mathcal{R}_s^a$$

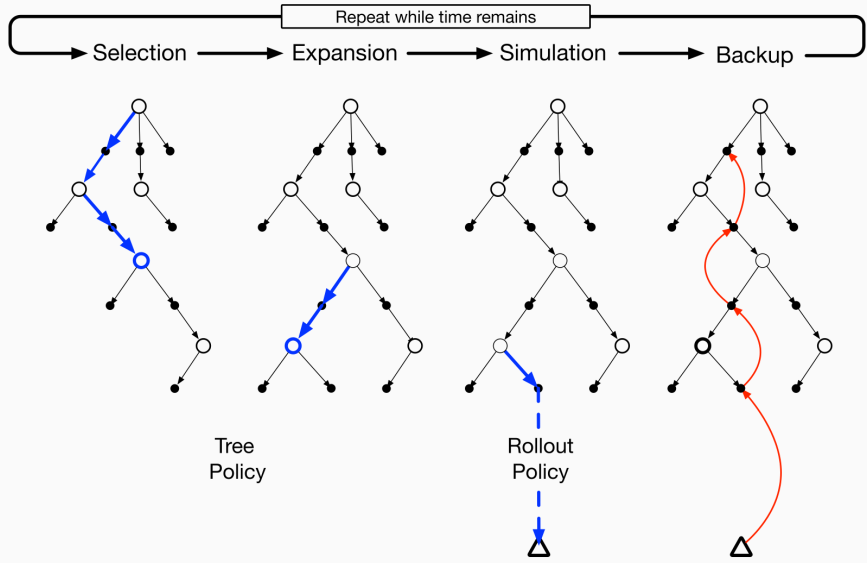




Algorithm: Dyna-Q [3, 4]

```
initialise  $Q(s, a)$  and model  $\mathcal{M}(s, a)$  for all  $s \in \mathcal{S}$  and  $a \in \mathcal{A}(s)$ 
while True:
     $s \leftarrow$  current (nonterminal) state
     $a \leftarrow \epsilon$ -greedy( $s, Q$ )
     $r, s' \leftarrow \text{env.step}(s, a)$ 
     $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{\hat{a}} Q(s', \hat{a}) - Q(s, a))$ 
     $\mathcal{M}(s, a) \leftarrow r, s'$  (assuming deterministic environment)
    for  $i$  in  $\text{range}(n)$ :
         $s \leftarrow$  random previously observed state
         $a \leftarrow$  random action previously taken in  $s$ 
         $r, s' \leftarrow \mathcal{M}(s, a)$ 
         $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{\hat{a}} Q(s', \hat{a}) - Q(s, a))$ 
```







Algorithm Model-ensemble trust-region policy optimization (ME-TRPO) [5]

- 1: Initialize a policy π_θ and all models $\hat{f}_{\phi_1}, \hat{f}_{\phi_2}, \dots, \hat{f}_{\phi_K}$.
- 2: Initialize an empty dataset \mathcal{D} .
- 3: **repeat**
- 4: Collect samples from the real system f using π_θ and add them to \mathcal{D} .
- 5: Train all models using \mathcal{D} .
- 6: **repeat** ▷ Optimize π_θ using all models.
- 7: Collect fictitious samples from $\{\hat{f}_{\phi_i}\}_{i=1}^K$ using π_θ .
- 8: Update the policy on the fictitious samples.
- 9: Estimate the performances $\hat{\eta}(\theta; \phi_i)$ for $i = 1, \dots, K$.
- 10: **until** the performances stop improving.
- 11: **until** the policy performs well in real environment f .



Algorithm Model-based policy optimization (MBPO) [6]

- 1: Initialize policy π_ϕ , predictive model p_θ , environment dataset \mathcal{D}_{env} , model dataset $\mathcal{D}_{\text{model}}$
- 2: **for** N epochs **do**
- 3: Train model p_θ on \mathcal{D}_{env} via maximum likelihood
- 4: **for** E steps **do**
- 5: Take action in environment according to π_ϕ ; add to \mathcal{D}_{env}
- 6: **for** M model rollouts **do**
- 7: Sample s_t uniformly from \mathcal{D}_{env}
- 8: Perform k -step model rollout starting from s_t using policy π_ϕ ; add to $\mathcal{D}_{\text{model}}$
- 9: **for** G gradient updates **do**
- 10: Update policy parameters on model data: $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi, \mathcal{D}_{\text{model}})$




Summary

In summary, model-based methods:

- are trained with supervised learning methods
- allow for planning ahead without interacting with the environment
- can be very data efficient by generating simulated experience
- but the values and policies can only be as good as the model
- they can be combined with model-free methods
- allow for changing the reward function/task without relearning the model



- [1] Richard S Sutton and Andrew G Barto.
Reinforcement learning: An introduction (second edition). Available online  MIT press, 2018.
- [2] David Silver. Reinforcement Learning lectures.
<https://www.davidsilver.uk/teaching/>. 2015.
- [3] Richard S Sutton. “Integrated architectures for learning, planning, and reacting based on approximating dynamic programming”. In:
Machine learning proceedings 1990. Elsevier, 1990, pp. 216–224.
- [4] Baolin Peng et al. “Deep Dyna-Q: Integrating planning for task-completion dialogue policy learning”. In: arXiv preprint arXiv:1801.06176 (2018).
- [5] Thanard Kurutach et al. “Model-Ensemble Trust-Region Policy Optimization”. In: International Conference on Learning Representations. 2018. URL:
<https://openreview.net/forum?id=SJJinbWRZ>.



- [6] Michael Janner et al. "When to trust your model: Model-based policy optimization". In: arXiv preprint arXiv:1906.08253 (2019).