Model-based Reinforcement Learning

Ayça Özçelikkale

Dept. of Electrical Engineering, Uppsala University

Overview of the Course

- Planning by Dynamic Programing
- Model-free prediction and control with tabular methods
- Function Approximation: We can use function approximation
  - to approximate value functions (last lecture)
  - to parametrize policies (next lecture)
  - to model the environment†
- Model-based RL
  - Tabular Model-based RL Methods: Dyna-Q and Dyna-Q+ (today's lecture)
  - General Model-based RL Methods:†
    - Motivating Example: PILCO
- Policy-Gradient Methods

5hp course: You're not responsible to know the details of algorithms/methods but you should be aware of the general ideas and advantages/disadvantages

Reminder: Function Approximation

Linear Approximation

- Value Function: $\hat{v}(s, \boldsymbol{w}) = \sum_{i=1}^{d} w_i x_i(s)$
- Action-Value Function: $\hat{q}(s, a, \boldsymbol{w}) = \sum_{i=1}^{d} w_i x_i(s, a)$

How to find the "best" approximation? We combine two ideas:

- Stochastic Gradient Descent
  - Update: step size $\times$ prediction error $\times$ gradient of (action-)value function approx.

$$\Delta \boldsymbol{w} = \alpha((v_\pi(S) - \hat{v}(S, \boldsymbol{w}))\nabla \hat{v}(S, A, \boldsymbol{w})$$
$$\Delta \boldsymbol{w} = \alpha((q_\pi(S, A) - \hat{q}(S, A, \boldsymbol{w}))\nabla \hat{q}(S, A, \boldsymbol{w})$$

- Replace $v_\pi(s, \boldsymbol{w})$ or $q(s, a, \boldsymbol{w})$ with an appropriate update target:
  - MC for predicting $v_\pi(s, \boldsymbol{w})$: G_t
  - TD(0) for predicting $v_\pi(s, \boldsymbol{w})$: $R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w})$
  - Sarsa for predicting $q_\pi(s, a, \boldsymbol{w})$: $R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w})$

Group Exercise: Function Approximation

Consider a corridor environment of 2 rooms. The environment is represented with 2 non-terminal states $\mathcal{S} = \{1, 2\}$ and one terminating state 3. The actions are LEFT (ActL) and RIGHT (ActR).
We obtain the following episode:

t	S_t	A_t	R_{t+1}
0	Room 1	ActR	$+1$
1	Room 2	ActR	$+3$
2	Room 3 (terminate)	-	-

The features are given by

$$x_1(s) = 0.5$$

$$x_2(s) = s$$

Let $\gamma = 1$, $\alpha = 1$. The weight vector is initialized as $\boldsymbol{w} = \mathbf{0}$. Using the above episode data and gradient Monte-Carlo with function approximation determine $\boldsymbol{w}_{final}$ after the updates.

What is a "Model"?

Model: Anything the agent can use to predict how the environment will respond to a given action

Given a state s and an action a,

- A model produces a prediction of the next state and the next reward: $(s, a) \rightarrow (\hat{s'}, \hat{r})$.
- A model for state-transition produces a prediction of the next state: $(s, a) \rightarrow \hat{s'}$.

Classification of Models:

Distribution Model (Probabilistic Models): Models that provide the **probabilities** for all possibilities, i.e. $p(s', r|s, a) \forall s, a$.

Sample Model: Models that provide one sample from the possible outcomes for a given (s, a) (according to the associated probabilities).

Self-study Exercise: Consider the experiment of throwing a fair die. What is the distribution model? How does the output of the sample model look like?

Comparison of Distribution Models and Sample Models

- Both type of models can be used to create "simulated" experience
- Sample models are easier to obtain in most cases.
- Distribution models are stronger in the sense that they can be always used to produce samples but the vice-a-versa is typically more difficult.

Self-study Exercise: Consider an environment where $\forall s, a$, there is only one s', r pair possible, i.e. the environment is deterministic. You know all possible (s, a) pairs. Given this a priori information, can you find the distribution model from the sample model?

What is Model-based RL?

A RL approach that uses predictions of the environment response explicitly is called a **model-based RL** approach.

- Prediction can be just a prediction of next state and next sample but it can also be the expected next reward or full distribution of the states and rewards.
- Only sampling from the experience such as in Q-learning does not count as using a prediction explicitly.

The computational process that uses a model to create/improve a policy is called **planning.**

Motivation: Why model-based RL?

Why should we be interested in model-based RL?

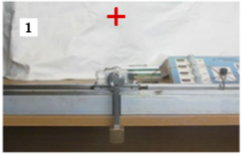
- Intuition: Explicitly predicting the future "should" help!
- Model-based approaches have been used successfully before in classic control in various scenarios including robotic control tasks and industrial process control in factories.
  - Model building in RL $\leftrightarrow$ system identification in classical control
  - Models can be easy to construct for scenarios where a representation of the system can be build using laws of physics where only a few parameters need to be found using data.
- Model-based RL approaches are "sample" efficient

Example: Illustration of the Sample Efficiency of Model-based Approaches -PILCO (probabilistic inference for learning control)

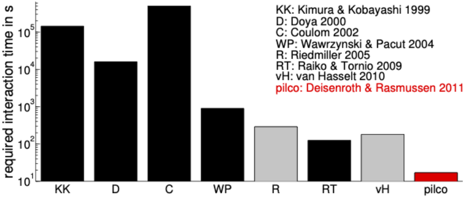
Main Ideas for PILCO: Model-based approach with uncertainty quantification + policy optimization

Example: Cart Pole Task

- swing the pendulum up and balance it on upright position by applying a horizontal force to the cart



Cart Pole



Interaction time required to balance the cart pole (Note the logarithmic scale for time!)

Archetypical Model-based RL Approach

Setting and Notation:
We focus on the state-transitions $(s, a) \to s'$ in the deterministic case.

Suppose that $Model(S, A)$ is the model that the agent uses for predicting the new state: $\hat{s'} = Model(S, A)$

Let $d(x, y)$ measure how much x is different from y, for instance $d(x, y) = \|x - y\|^2$.

Algorithm:

`Initialization:` `Collect the dataset` $\mathcal{D} = \{(s_i, a_i, s_i')\}$ `by running, for instance, a random policy.`
`repeat:`

`Learn model` $Model(s, a)$ `using` $\min \sum_i d(\hat{s_i'}, s_i')$

`Plan using` $Model(s, a)$ `to find the action` a_j `for the current state` s_j.

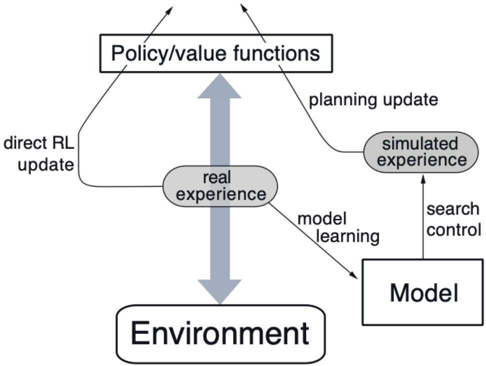
`Execute the action` a_j `and get` s_j'.

`Add` (s_j, a_j, s_j') `to` $\mathcal{D}$.

Archetypical Model-based RL Approach: Remarks

- Variations where the `Learn model` and `Plan` steps are not repeated at every step can be used.
- Stochastic case: We can work with $p_{\boldsymbol{w}}(s'|s,a)$ instead of a deterministic mapping.
- These ideas can be also extended to the cases where $reward(s,a)$ is also estimated.

Overview of Dyna-Q



Tabular Dyna-Q for deterministic environments

Initialize $Q(s,a)$ and $Model(s,a)$ for all s,a.
Loop forever

(a) S $\leftarrow$ current (nonterminal) state

(b) Decide on action A using an exploration policy, such as ϵ-greedy.

(c) Take action A, observe the resulting reward R and state S'

(d) Q-update: $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$

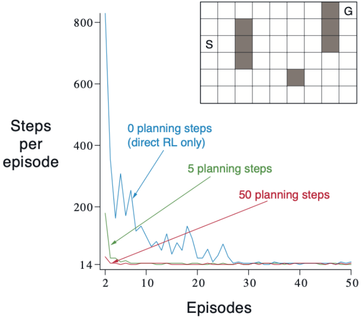
(e) Update model: $Model(S,A) \leftarrow R, S'$

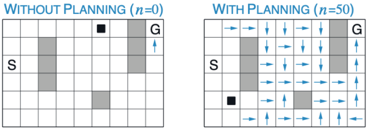
(f) Planning Update: Repeat n times

- Choose a random previously observed state S
- Choose a random action A previously taken in S
- Get the resulting reward R and state S' from the model: $R, S' \leftarrow Model(S,A)$
- Q-update: $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$

Example: Dyna Maze

- Actions: up, down, right, and left
  - if obstacle/edge, the agent remains where it is.
- Reward is zero on all transitions, except those into the goal state, on which it is +1.

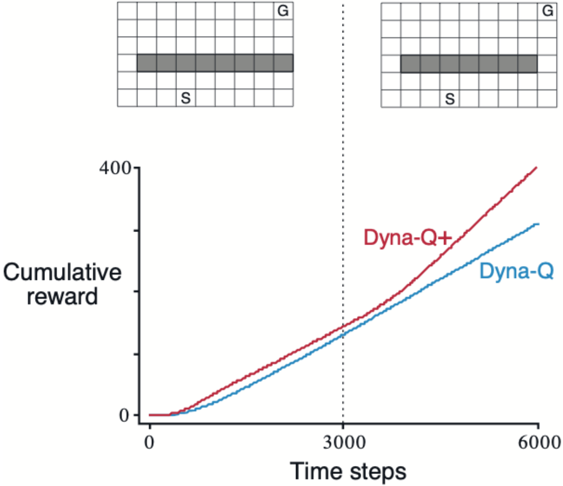




Policy at halfway during the second episode

Motivation: What if the environment changes?

Model bias: (Naive) model based methods inherently assume that the learned model is an accurate description of the real environment.



Dyna-Q+

Main Idea for Dyna-Q+: In the planning update, change reward from R to $R+\kappa\sqrt{\tau}$, where τ is the time passed since the last time this state-action pair is tried.

- This encourages the agent to test all admissible state transitions even if the reward observed for them were low previously. (exploration!)
- Trying all these state transitions has a "cost" but "curiosity" may help, especially if the model changes as in the previous example.

Dyna-Q+

Initialize $Q(s,a) = 0$ for all s, a. Initialize $Model(s,a) \leftarrow r = 0, s$ for all s, a. Initialize all s, a as previously observed/taken with $t_v(S,A) = 0$.
Initialize agent's internal time $t = 0$.
Loop forever

(a) S $\leftarrow$ current (nonterminal) state

(b) Decide on action A using an exploration policy, such as ϵ-greedy.

(c) Take action a, observe the resulting reward R and state S'

(d) Q-update: $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$

(e) Update model: $Model(S,A) \leftarrow R, S'$. Record the last time of the visit $t_v(S,A)$

(f) Planning Update: Repeat n times

- Choose a random previously observed state S
- Choose a random action A previously taken in S
- Get the resulting reward R and state S' from the model: $R, S' \leftarrow Model(S,A)$
- Let $\tau = t - t_v(S,A)$
- Q-update: $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \kappa\sqrt{\tau} + \gamma \max_a Q(S',a) - Q(S,A)]$

(g) Increment t.

Self-study Exercise: This implementation uses an internal time variable t. Can you implement the Dyna-Q+ idea without explicitly defining such a variable?

A Quick and Dirty Look at PILCO

Main Ideas for PILCO: Model-based RL with uncertainty quantification + policy optimization

Dynamical Model:

$$s_t = f(s_{t-1}, a_{t-1})$$

- $s_t \in \mathbb{R}^D$: continuous valued states
- $a_t \in \mathbb{R}^F$: continuous valued controls/actions
- $f(\cdot)$: the unknown transition dynamics, i.e. latent function

Policy†:
Action is given by $a_t = \pi(s_t, \theta)$ where π is the policy/controller and θ is the unknown parameters of the policy.

Objective:
Find the policy π that minimizes the expected cost of following π for T steps:

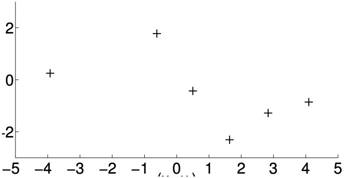
$$J^{\pi}(\theta) = \sum_{t=0}^{T} \mathbb{E}_{s_t}[c(s_t)]$$

Example cost function: $c(s) = 1 - \exp(\|s - s_{target}\|^2/\sigma_c^2)$

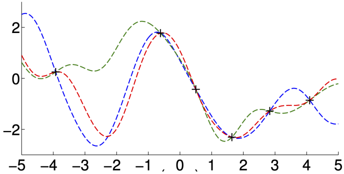
Model Uncertainty Quantification in PILCO

In PILCO, a probabilistic function approximator is used to model the uncertainty about the latent function. Why is this a good idea?

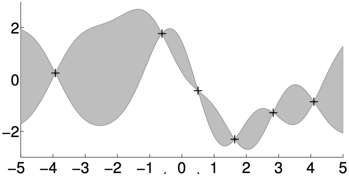
Figure Axes:
y- axis: Latent function $f(s_t, a_t)$
x- axis: State-action pairs (s_t, a_t)



Data set of observed transitions



There are multiple plausible deterministic function approximators



Probabilistic function approximator encodes our uncertainty about $f(.)$

PILCO: Algorithm

Initialization: Initialize dataset $\mathcal{D}$ by running a random policy. Parametrize policy with θ, i.e. $\pi(\theta)$. Initialize θ.

repeat:

Learn model: Using the tuples (s_t, s_{t-1}, a_{t-1}) from $\mathcal{D}$ find $\hat{p}(s_t|s_{t-1}, a_{t-1})$.

repeat: *Model-based policy search*

Using $\hat{p}(s_t|s_{t-1}, a_{t-1})$, perform policy evaluation

Perform policy improvement.

until: convergence; return θ^*.

Execute actions for trial/episode using $\pi(\theta^*)$.

Add new data to $\mathcal{D}$.

Exploration vs Exploitation Trade-offs

- **Exploration:** Gather more information
- **Exploitation:** Make the best decision given the current information

Example: What to cook for tonight?

- Exploration: "Try a new recipe" versus Exploitation: "Cook your favorite dish"

Exploration ideas we have seen so far:

- Dyna-Q+
- ϵ-greedy
- optimistic initialization

Question: How does this trade-off relate to model uncertainty?

Concluding Remarks

- **Main idea for Model-based RL:** Having an explicit model for the environment may help.
- Advantages -Model based approaches are typically promising from the following aspects:
  - sample efficiency
  - transfer learning (ex: same dynamics with different reward or same task under similar dynamics)
  - explanability
- Disadvantages
  - tend to have lower asymptotic performance
  - possibly can be unstable due errors in learned model (uncertainty quantification!)
  - additional computational complexity/memory, possibly more tunable parameters

Concluding Remarks

- Different Model-Based Approaches:
  - Dyna type of approaches: use “imagined” data from the model to improve policy (a model-free method is used on the “imagined” data from the model)
  - Use model and its derivatives directly to optimize the RL objective
  - Model-based predictive control type of approaches (directly predict what will happen in the next time steps and choose the best)
- In general, for model based approaches we need to think about: Exploration vs Exploitation Trade-offs, Exploration for the Model vs Task, Model-bias, Quantification of model uncertainty

References

- Sutton, Barto, Reinforcement Learning: An Introduction
- Sutton, R. S., Integrated architectures for learning, planning, and reacting based on approximating dynamic programming., Proc. of International Workshop on Machine Learning (ICML), 1990.
- Marc Peter Deisenroth and Carl Edward Rasmussen. PILCO: a model-based and data-efficient approach to policy search, Proc, of International Conference on Machine Learning (ICML) 2011.