

Machine Learning Reinforcement Learning

dsai.asia

Asian Data Science and Artificial Intelligence Master's Program



Co-funded by the
Erasmus+ Programme
of the European Union



Readings for these lecture notes:

- Ng, A. (2017), *Reinforcement Learning and Control*. Lecture note set 12 for CS229, Stanford University.

These notes contain material © Ng (2017).

Outline

- 1 Introduction
- 2 Markov decision processes
- 3 Value iteration and policy iteration
- 4 Learning a model for a MDP
- 5 Continuous state MDPs
- 6 Finite-horizon MDPs
- 7 Linear quadratic regulation (LQR)
- 8 LQR for nonlinear dynamics
- 9 Linear quadratic Gaussian (LQG) regulation

Now we consider the problem of **reinforcement learning**.

In the supervised learning setting, we give the machine examples of what it should do in particular circumstances.

Sometimes it is not easy or even possible to come up with such a training set. Examples include robot motion control, game playing systems, stock trading policies, dynamic allocation of resources to a Web application, ...

In some of these domains, rather than providing a definite answer, we provide the machine with a **reward function** indicating when it is doing well and when it is doing poorly.

Introduction

Example reward functions:

- Robot control: give positive rewards for moving forward and negative rewards for falling over.
- Game playing: give positive rewards for winning, negative rewards for losing.
- Robotic exploration: give positive rewards for expanding the “frontier,” negative rewards for going over the same old ground.
- Robotic vacuum cleaning: give positive rewards for cleaning dirty areas, negative rewards for spending time in clean areas.
- Web application resource allocation: positive rewards for serving requests quickly, negative rewards for using excessive resources.

Usually, reinforcement learning problems are posed as **Markov decision processes**.

Outline

- 1 Introduction
- 2 Markov decision processes**
- 3 Value iteration and policy iteration
- 4 Learning a model for a MDP
- 5 Continuous state MDPs
- 6 Finite-horizon MDPs
- 7 Linear quadratic regulation (LQR)
- 8 LQR for nonlinear dynamics
- 9 Linear quadratic Gaussian (LQG) regulation

Markov decision processes

Formal definition

A Markov decision process is a tuple $(S, A, \{P_{sa}\}, \gamma, R)$, where

- S is a set of **states**.
- A is a set of **actions**.
- P_{sa} are the **state transition probabilities**. For each state $s \in S$ and action $a \in A$ P_{sa} is a distribution over S . $P_{s_i a_j}(s_k)$ gives the probability that we transition to state s_k when we perform action a_j in state s_i .
- $\gamma \in [0, 1)$ is called the **discount factor**.
- $R : S \times A \mapsto \mathbb{R}$ is the **reward function**. Sometimes we use $R : S \mapsto \mathbb{R}$.

Markov decision processes

Payoff

A MDP proceeds as follows:

- Start in some state s_0 .
- Choose action $a_0 \in A$.
- Sample $s_1 \sim P_{s_0 a_0}$.
- Choose action a_1 .
- Sample $s_2 \sim P_{s_1 a_1}$.
- Repeat...

We can represent the progress of the process as

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots$$

Given such a sequence, we can compute the **total payoff** as

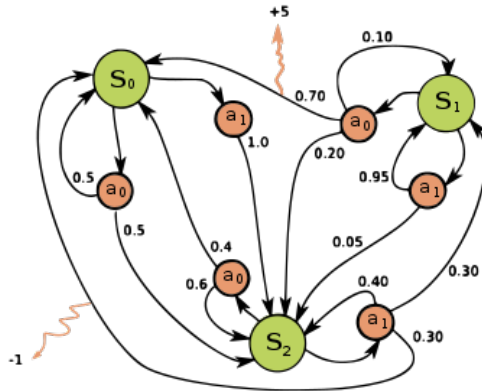
$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \dots$$

or

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$$

Markov decision processes

Simple MDP example



https://en.wikipedia.org/wiki/Markov_decision_process

Markov decision processes

Discount

The reward at timestep t is **discounted** by a factor γ^t .

In some applications, time might not matter and we would use $\gamma = 1$.

In many other applications, we want to accrue positive rewards as quickly as possible, so $\gamma < 1$.

When $R(\cdot)$ is the amount of money we make, γ would represent the effect of interest (money accrued today is more valuable in the long term than money accrued tomorrow).

Markov decision processes

Policy, value function

How should our learning agent behave?

A **policy** is a function $\pi : S \mapsto A$ mapping from states to actions.

Executing policy π means when we are in state s , we take action $a = \pi(s)$.

The **value function** for policy π is

$$V^\pi(s) = \mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots \mid s_0 = s; \pi],$$

i.e., the expected sum of discounted rewards upon starting in state s and taking actions according to π .

Markov decision processes

Bellman equations

Given a fixed policy π , the value function V^π satisfies the **Bellman equations**

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^\pi(s').$$

$R(s)$ is called the **immediate reward** we get for starting in state s .

The second term (the summation) can be rewritten $\mathbb{E}_{s' \sim P_{s\pi(s)}}[V^\pi(s')]$, i.e., the expected sum of discounted rewards for starting in state s' , where s' is distributed according to $P_{s\pi(s)}$, the distribution over states resulting from action $\pi(s)$ in state s .

Markov decision processes

Solving the Bellman equations

The Bellman equations give us a simple way to determine V^π for finite-state MDPs ($|S| < \infty$).

Simply write down the equation $V^\pi(s)$ for each $s \in S$.

When R , P_{sa} , and π are fixed, we have $|S|$ linear equations in $|S|$ unknowns.

[Exercise: find V^π for a small MDP such as a robot maze escape.]

Markov decision processes

Optimal value function and policy

We define the **optimal value function** as

$$V^*(s) = \max_{\pi} V^{\pi}(s).$$

We can write the Bellman equations for the optimal value function as

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s').$$

We have the immediate reward plus the maximum over all a of the expected future sum of discounted rewards we'll get from a .

We define the **optimal policy** $\pi^* : S \mapsto A$ as

$$\pi^*(s) = \operatorname{argmax}_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s') \quad (1)$$

Markov decision processes

Optimal policy

Some facts:

$$\forall s, V^*(s) = V^{\pi^*}(s).$$

$$\forall s, \pi, V^{\pi^*}(s) \geq V^{\pi}(s).$$

i.e., π^* as defined in Equation (1) is the optimal policy.

Note that π^* is optimal for *all* states s . So there is no need to modify π^* according to the start state.

So the question, then, is **how can we find π^* for a given MDP?**

Outline

- 1 Introduction
- 2 Markov decision processes
- 3 Value iteration and policy iteration**
- 4 Learning a model for a MDP
- 5 Continuous state MDPs
- 6 Finite-horizon MDPs
- 7 Linear quadratic regulation (LQR)
- 8 LQR for nonlinear dynamics
- 9 Linear quadratic Gaussian (LQG) regulation

Value iteration and policy iteration

Value iteration

For the time being, we assume our MDP has a finite state space and action space ($|S| < \infty$, $|A| < \infty$).

Algorithm VALUE-ITERATION

For each state s

$$V(s) \leftarrow 0$$

Repeat until convergence

For each state s

$$V(s) \leftarrow R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V(s')$$

The inner loop can be **synchronous** (compute all values before overwriting) or **asynchronous** (compute and update values one by one in some order).

Theorem: **under synchronous or asynchronous value iteration, V converges to V^* .**

Value iteration and policy iteration

Policy iteration

After running value iteration, we use Equation (1) to find π^* .

Policy iteration computes π^* somewhat more directly:

Algorithm POLICY-ITERATION

Initialize π randomly

Repeat until convergence

$$V \leftarrow V^\pi$$

For each state s

$$\pi(s) \leftarrow \operatorname{argmax}_{a \in A} \sum_{s' \in S} P_{sa}(s') V(s')$$

Theorem: after a finite number of iterations, V will converge to V^* and π will converge to π^* .

Value iteration and policy iteration

Which to use?

Policy iteration converges quickly for small MDPs.

However, computing V^π requires solving a prohibitively large linear system for large problems.

Value iteration is therefore more often used in practice.

Outline

- 1 Introduction
- 2 Markov decision processes
- 3 Value iteration and policy iteration
- 4 Learning a model for a MDP**
- 5 Continuous state MDPs
- 6 Finite-horizon MDPs
- 7 Linear quadratic regulation (LQR)
- 8 LQR for nonlinear dynamics
- 9 Linear quadratic Gaussian (LQG) regulation

Learning a model for a MDP

Assumptions thus far

So we see that reinforcement learning is straightforward when

- $|S| < \infty$
- $|A| < \infty$
- P_{sa} is given
- $R(s)$ is given
- The outcome s' obtained from executing action a in state s is observable.
- The reward $R(s)$ obtained from state s is immediately observable.
- The **horizon** is infinite (the total payoff calculation is an infinite sum).

We would want to relax each of these assumptions.

Relaxing assumptions 3 and 4 require **estimating P_{sa} and/or $R(s)$ from data.**

[Think about what is known in example real-world problems.]

Learning a model for a MDP

Estimating P_{sa}

When P_{sa} is unknown, we run the MDP multiple times in a simulator or the real world and observe the behavior of the system:

$$s_0^{(1)} \xrightarrow{a_0^{(1)}} s_1^{(1)} \xrightarrow{a_1^{(1)}} s_2^{(1)} \xrightarrow{a_2^{(1)}} s_3^{(1)} \xrightarrow{a_3^{(1)}} \dots$$

$$s_0^{(2)} \xrightarrow{a_0^{(2)}} s_1^{(2)} \xrightarrow{a_1^{(2)}} s_2^{(2)} \xrightarrow{a_2^{(2)}} s_3^{(2)} \xrightarrow{a_3^{(2)}} \dots$$

Examples:

- For an inverted pendulum, we run until the pendulum falls over.
- For a game, we try a large but finite number of moves.

Learning a model for a MDP

Estimating P_{sa}

With enough experience, the maximum likelihood estimate of the state transition probabilities are

$$P_{sa}(s') = \frac{\# \text{ of times we took action } a \text{ in state } s \text{ and got to } s'}{\# \text{ of times we took action } a \text{ in state } s}.$$

(When we obtain 0/0 we can assume $P_{sa}(s') = 1/|S|$.)

To do this efficiently, we just need to record the counts for the numerator and denominator over time and increment the appropriate counts each time we take an action and observe the result.

Estimating R is similar. Just keep track of the number of times we entered state s and what reward we received in order to calculate the expected reward.

Learning a model for a MDP

Estimating P_{sa}

Putting value iteration together with model learning for P_{sa} , we obtain

Algorithm VALUE-ITERATION-WITH-TRANSITION-LEARNING

Initialize π randomly

Repeat

 Execute π in the MDP for some number of trials

 Use accumulated experience to update P_{sa}

 Use value iteration to estimate V

 Update π to be the greedy policy for V

Value iteration will converge more quickly if we begin with the V obtained in the previous step.

Note that there is a related, more stochastic algorithm called **Q-learning** that learns the expected value of taking action a in state s .

Outline

- 1 Introduction
- 2 Markov decision processes
- 3 Value iteration and policy iteration
- 4 Learning a model for a MDP
- 5 Continuous state MDPs**
- 6 Finite-horizon MDPs
- 7 Linear quadratic regulation (LQR)
- 8 LQR for nonlinear dynamics
- 9 Linear quadratic Gaussian (LQG) regulation

Continuous state MDPs

Examples

Now that we can estimate models for P_{sa} and R , we have the following remaining assumptions:

- $|S| < \infty$
- $|A| < \infty$
- The outcome s' obtained from executing action a in state s is observable.
- The reward $R(s)$ obtained from state s is immediately observable.

What about **relaxing the finite state assumption**?

In a car or ground robot, we might have a state $(x, y, \theta, \dot{x}, \dot{y}, \dot{\theta})$. $S = \mathbb{R}^6$.

In the inverted pendulum problem, we have $(x, \theta, \dot{x}, \dot{\theta})$. $S = \mathbb{R}^4$.

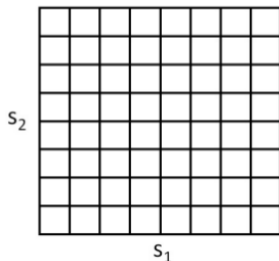
For an aircraft, we have $(x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi})$. $S = \mathbb{R}^{12}$.

Continuous state MDPs

Discretization

Depending on the situation, we may be able to **discretize** the state space. If so, we can simply run value iteration or policy iteration on the discrete approximation to the state space.

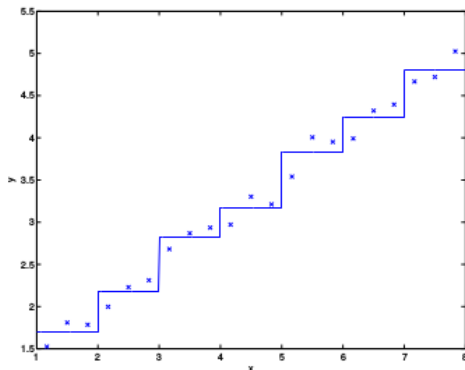
For example, with $S = \mathbb{R}^2$, we use a **grid**. Rather than $s = (s_1, s_2) \in \mathbb{R}^2$ we break up the range of s_1 into k bins and s_2 into l bins and use $\bar{S} = \{1, \dots, k\} \times \{1, \dots, l\}$.



Continuous state MDPs

Discretization

Problem 1 with discretization: **inaccuracy**. Suppose the value $V(s)$ was really linear in s . We might obtain something like this:



Ng (2017), CS 229 lecture notes, set 12

Continuous state MDPs

Discretization

Problem 2 with discretization: **the curse of dimensionality**.

The number of parameters we have to estimate is exponential in the number of variables in the state.

Ng:

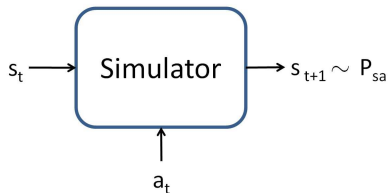
- Discretization is good for 1D and 2D problems.
- Discretization sometimes works with clever representations for 3D-6D problems.
- It never works for higher dimensional problems!

Continuous state MDPs

Value function approximation

The next technique in arsenal for dealing with continuous state spaces is **value function approximation**.

Value function approximation works best when we have a **simulator** for the MDP at hand or when running trials in the real world is fast and cheap.



Ng (2017), CS 229 lecture notes, set 12

[Simulating an inverted pendulum is straightforward if all parameters like the pendulum mass are known. Think about simulators for other tasks.]

Continuous state MDPs

Value function approximation

We first need to know P_{sa} . One possible form is

$$s_{t+1} = As_t + Ba_t + \epsilon_t,$$

where $\epsilon_t \sim \mathcal{N}(0, \Sigma)$.

A and B might be given, or we could estimate them.

Continuous state MDPs

Value function approximation

To estimate A and B, first we collect data from the simulation or real world MDP:

$$\begin{aligned} s_0^{(1)} &\xrightarrow{a_0^{(1)}} s_1^{(1)} \xrightarrow{a_1^{(1)}} s_2^{(1)} \xrightarrow{a_2^{(1)}} \dots \xrightarrow{a_{T-1}^{(1)}} s_T^{(1)} \\ s_0^{(2)} &\xrightarrow{a_0^{(2)}} s_1^{(2)} \xrightarrow{a_1^{(2)}} s_2^{(2)} \xrightarrow{a_2^{(2)}} \dots \xrightarrow{a_{T-1}^{(2)}} s_T^{(2)} \end{aligned}$$

The maximum likelihood estimator, if $\epsilon_t \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$, would be

$$\mathbf{A}^*, \mathbf{B}^* = \underset{\mathbf{A}, \mathbf{B}}{\operatorname{argmin}} \sum_{i=1}^m \sum_{t=0}^{T-1} \left\| s_{t+1}^{(i)} - \left(\mathbf{A} s_t^{(i)} + \mathbf{B} a_t^{(i)} \right) \right\|^2.$$

(If Σ is unknown, we could also estimate it.)

Continuous state MDPs

Value function approximation

Once we have A , B , and possibly Σ , we can either build a **deterministic** model in which we assume

$$s_{t+1} = As_t + Ba_t$$

precisely, or a **stochastic** model in which we treat s_{t+1} as random.

Other alternatives: add nonlinear mappings ϕ_s and ϕ_a to obtain

$$s_{t+1} = A\phi_s(s_t) + B\phi_a(a_t),$$

or use locally weighted linear regression to estimate s_{t+1} from s_t and a_t .

Continuous state MDPs

Value function approximation

Modeling the state transition is just the first step!

Next, we perform **fitted value iteration** to approximate the value function.

For simplicity, we'll assume $S = \mathbb{R}^n$ but $A = \{1, \dots, k\}$ for a small number k of possible discrete actions.

(Even if A is really continuous, discretizing A is often feasible due to lower dimensionality than S .)

We will essentially, then, approximate $V(s)$ as a linear or nonlinear function of the state:

$$V(s) = \theta^\top \phi(s)$$

where ϕ is an appropriate feature mapping.

For each state in each sampled state transition sequence, we will compute a training signal $y^{(i)}$ that approximates $R(s) + \gamma \max_a \mathbb{E}_{s' \sim P_{sa}}[V(s')]$, i.e., the value function.

Continuous state MDPs

Value function approximation

Here is the complete algorithm:

Algorithm FITTED-VALUE-ITERATION

Randomly sample m states $s^{(1)}, s^{(2)}, \dots, s^{(m)} \in \mathcal{S}$

$\theta \leftarrow 0$

Repeat

For $i \in 1..m$

For each action $a \in A$

Sample $s'_1, \dots, s'_k \sim P_{s^{(i)}a}$ (using a model of the MDP)

// Set $q(a)$ to an estimate of $R(s^{(i)}) + \gamma \mathbb{E}_{s' \sim P_{s^{(i)}a}}[V(s')]$

$q(a) \leftarrow \frac{1}{k} \sum_{j=1}^k R(s^{(i)}) + \gamma V(s'_j)$

// Set $y^{(i)}$ to an estimate of $R(s^{(i)}) + \gamma \max_a \mathbb{E}_{s' \sim P_{s^{(i)}a}}[V(s')]$

$y^{(i)} \leftarrow \max_a q(a)$

$\theta \leftarrow \operatorname{argmin}_{\theta} \frac{1}{2} \sum_{i=1}^m (\theta^\top \phi(s^{(i)}) - y^{(i)})^2$

Continuous state MDPs

Value function approximation

Rather than linear regression, we can use any appropriate regression algorithm, such as locally weighted linear regression.

Unfortunately, fitted value iteration is **not guaranteed to converge** like discrete-space value iteration.

In practice, it does converge for many problems.

If we have a deterministic model of the MDP, then $k = 1$ is sufficient.

The **approximately optimal policy** is determined by $V(s)$, which is an approximation to V^* :

$$\pi(s) = \operatorname{argmax}_a \mathbb{E}_{s' \sim P_{sa}} [V(s')].$$

Outline

- 1 Introduction
- 2 Markov decision processes
- 3 Value iteration and policy iteration
- 4 Learning a model for a MDP
- 5 Continuous state MDPs
- 6 Finite-horizon MDPs**
- 7 Linear quadratic regulation (LQR)
- 8 LQR for nonlinear dynamics
- 9 Linear quadratic Gaussian (LQG) regulation

Finite-horizon MDPs

Up to now, we always assumed an **infinite horizon** (our learning agent's total payoff is a infinite sum).

However, in the real world, we would often want to consider a **finite horizon** MDP $(\mathcal{S}, \mathcal{A}, P_{sa}, T, R)$ with $T > 0$ placing a **limit** on the time the agent has to accrue its payoff:

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \cdots + R(s_T, a_T)$$

This leads to some important consequences, especially that the optimal policy becomes **non-stationary** (different depending on the start state).

With some other generalizations (P_{sa} and R are dynamic, i.e., changing over time), value iteration turns into a **dynamic programming** algorithm for solving the Bellman equations.

Outline

- 1 Introduction
- 2 Markov decision processes
- 3 Value iteration and policy iteration
- 4 Learning a model for a MDP
- 5 Continuous state MDPs
- 6 Finite-horizon MDPs
- 7 Linear quadratic regulation (LQR)**
- 8 LQR for nonlinear dynamics
- 9 Linear quadratic Gaussian (LQG) regulation

Linear quadratic regulation (LQR)

Suppose $\mathcal{S} = \mathbb{R}^n$ and $\mathcal{A} = \mathbb{R}^d$.

Suppose also **linear transitions**

$$\mathbf{s}_{t+1} = \mathbf{A}\mathbf{s}_t + \mathbf{B}\mathbf{a}_t + \mathbf{w}_t$$

with $\mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, \Sigma_t)$.

Finally, suppose we have **quadratic rewards**

$$R^{(t)}(\mathbf{s}_t, \mathbf{a}_t) = -\mathbf{s}_t^\top \mathbf{U}_t \mathbf{s}_t - \mathbf{a}_t^\top \mathbf{W}_t \mathbf{a}_t,$$

where $\mathbf{U}_t \in \mathbb{R}^{n \times n}$ and $\mathbf{W}_t \in \mathbb{R}^{d \times d}$ are positive definite, forcing the reward to always be negative.

Linear quadratic regulation (LQR)

The quadratic rewards mean that we want to take **small, smooth actions** that keep the system **as close to $s_t = 0$** as possible.

Think about deviation of the inverted pendulum, deviation of our car from the center of the lane, etc.

The **LQR algorithm** first estimates A_t , B_t , and Σ_t , then uses dynamic programming to find the optimal policy.

Outline

- 1 Introduction
- 2 Markov decision processes
- 3 Value iteration and policy iteration
- 4 Learning a model for a MDP
- 5 Continuous state MDPs
- 6 Finite-horizon MDPs
- 7 Linear quadratic regulation (LQR)
- 8 LQR for nonlinear dynamics**
- 9 Linear quadratic Gaussian (LQG) regulation

LQR for nonlinear dynamics

Now we suppose that the dynamics, rather than being linear

$$s_{t+1} = As_t + Ba_t + w_t,$$

we instead have a nonlinear relationship:

$$s_{t+1} = f(s_t, a_t) + w_t.$$

An example is the inverted pendulum, in which the elements of the next position vector depend on the cosine or sine of the current angle and so on.

In this situation, we can obtain a Taylor expansion of $f(\cdot, \cdot)$ around s_t and a_t to **linearize** the function, then perform LQR as before.

There is a further generalization called **differential dynamic programming** for the case in which the reward function is more complicated than quadratic rewards (for example, approximating a gold standard trajectory for a rocket).

Outline

- 1 Introduction
- 2 Markov decision processes
- 3 Value iteration and policy iteration
- 4 Learning a model for a MDP
- 5 Continuous state MDPs
- 6 Finite-horizon MDPs
- 7 Linear quadratic regulation (LQR)
- 8 LQR for nonlinear dynamics
- 9 Linear quadratic Gaussian (LQG) regulation

Linear quadratic Gaussian (LQG) regulation

Sometimes we are not able to fully observe s_t .

Instead, we may only be able to take an **observation** related to s_t , such taking a picture with a camera or a distance scan with a laser or a GPS reading from a GPS device.

This leads to the idea of the **partially observable Markov decision process (POMDP)**.

We assume that at each timestep t we obtain an observation $z_t \in \mathcal{O}$ related to the state s_t by a distribution $P_{zs}(z \mid s)$.

A POMDP is thus given by a tuple $(\mathcal{S}, \mathcal{O}, \mathcal{A}, P_{sa}, P_{zs}, T, R)$.

In the POMDP setting, we will maintain a **belief** over the states $p(s_t \mid z_1, \dots, z_t)$, and the policy will map beliefs to actions rather than crisp states to actions.

Linear quadratic Gaussian (LQG) regulation

In the special case that the dynamics and observations are both linear with Gaussian noise, i.e.,

$$\begin{aligned}s_{t+1} &= A s_t + B a_t + w_t \\ z_t &= C s_t + v_t\end{aligned}$$

with $w_t \sim \mathcal{N}(0, \Sigma_s)$ and $v_t \sim \mathcal{N}(0, \Sigma_z)$, we obtain the **Kalman filter** for maintaining our belief $p(s_t \mid z_1, \dots, z_t)$.

The Kalman filter gives us a mean and covariance for the belief at each time t .

Based on mean at time t , we can use the regular LQR for determining the optimal action.