

CS 277: Control and Reinforcement Learning

Winter 2022

Lecture 12: Partial Observability

Roy Fox

Department of Computer Science

Bren School of Information and Computer Sciences

University of California, Irvine



Logistics

assignments

- Assignment 3 is due **Tuesday**
- Assignment 4 to be published soon

quizzes

- Quiz 5 to be published soon

Today's lecture

Model-predictive control

Partially Observable MDPs

Belief-state MDPs

RNNs

Issues with approximate models (1)

- In large state / action spaces, we can only **approximate** the dynamics
- **No guarantees** outside of training distribution
 - We can't be too far off-policy
- Solution: **keep interacting** using learner policy and updating the model

Issues with approximate models (2)

- Model inaccuracy **accumulates**
 - If $\|p_\phi(s' | s, a) - p(s' | s, a)\|_1 \leq \epsilon$ then $\|p_\phi(s_t) - p(s_t)\|_1 \leq \epsilon t$
 - We have to plan far enough ahead to realize the **consequences** of actions
 - But we don't have to **execute** those plans far ahead!

Algorithm Model-Predictive Control (MPC)

$\mathcal{D} \leftarrow$ collect data

repeat

$\hat{M} \leftarrow$ train model \hat{p}, \hat{r} from \mathcal{D}

repeat

$\pi \leftarrow$ plan in \hat{M} from current state s to horizon H

Take *one action* a according to π

Add empirical (s, a, r, s') to \mathcal{D}

How to use a learned model

- Recall how **planning** benefitted from access to a model:
 - As a fast simulator
 - As an arbitrary-reset simulator
 - As a **differentiable model**

Local models

- Can we use a **learned model** for iLQR?
 - Idea 1: learn **global** model, linearize locally \Rightarrow **wasteful**
 - Idea 2: directly learn **local** linearizations:

Algorithm Local Models

Initialize a policy $\pi(u_t|x_t)$

repeat

Roll out π to horizon T for N trajectories

Fit $p(x_{t+1}|x_t, u_t)$

Plan new policy π

How to fit local dynamics

- Idea 1: linear regression

- ▶ Find $(A_t, B_t)_{t=0}^{T-1}$ such that $x_{t+1} \approx A_t x_t + B_t u_t$

- ▶ Do we care about the process noise ω_t ?

- If we assume it's Gaussian, doesn't affect policy; but could help evaluate the method

- Idea 2: Bayesian linear regression

- ▶ Learn global model, use it as prior for local model

- ▶ More data efficient across time steps and across iterations

How to plan with local models

- Idea 1: as in iLQR, find **optimal control** sequence \hat{u} and its trajectory \hat{x}
 - Problem: model errors will cause actual trajectory to **diverge** from \hat{x}
- Idea 2: find \hat{x} by executing the optimal policy directly **in the environment**
 - Problem: need **spread** for linear regression, dynamics may be **too deterministic**
- Idea 3: make control stochastic by injecting Gaussian noise
 - E.g., have $\epsilon_t \sim \mathcal{N}(0, R^{-1})$, shaped by the **control cost**
 - Optimal for the incurred **costs**, not for the **spread** needed for regression

Recap

- Model-based RL schemes:
 - Plan in a learned model
 - Improve model-free RL using a learned model
- Good theory for how to explore optimally for learning a model

Today's lecture

MPC, Local Models

Partially Observable MDPs

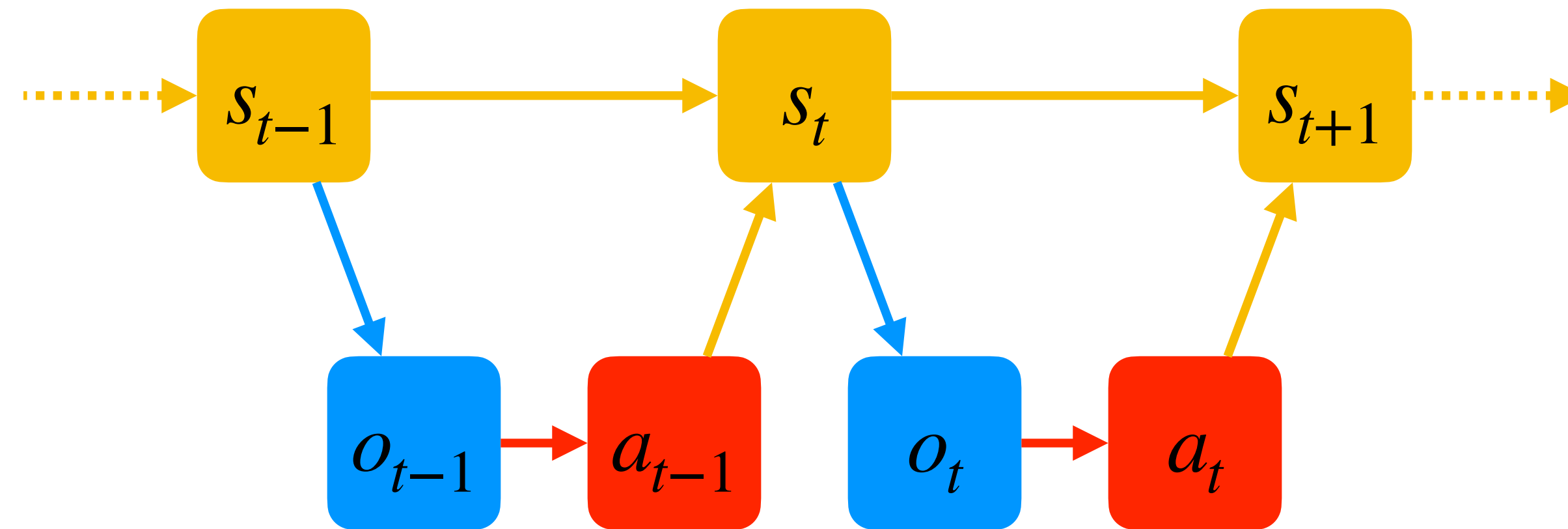
Belief-state MDPs

RNNs

What does the policy depend on?

- Minimally: **nothing**
 - Just an **open-loop** sequence of actions a_0, a_1, \dots
 - Except, even this depends on a **clock** $a_t = \pi(t)$
- Typically: the **current state** $\pi(a_t | s_t)$
- What if the state is not fully observable to the agent's **sensors**?
 - Completely **unobservable** → forced open loop
 - **Partially observable** → $\pi(a_t | o_t)$?

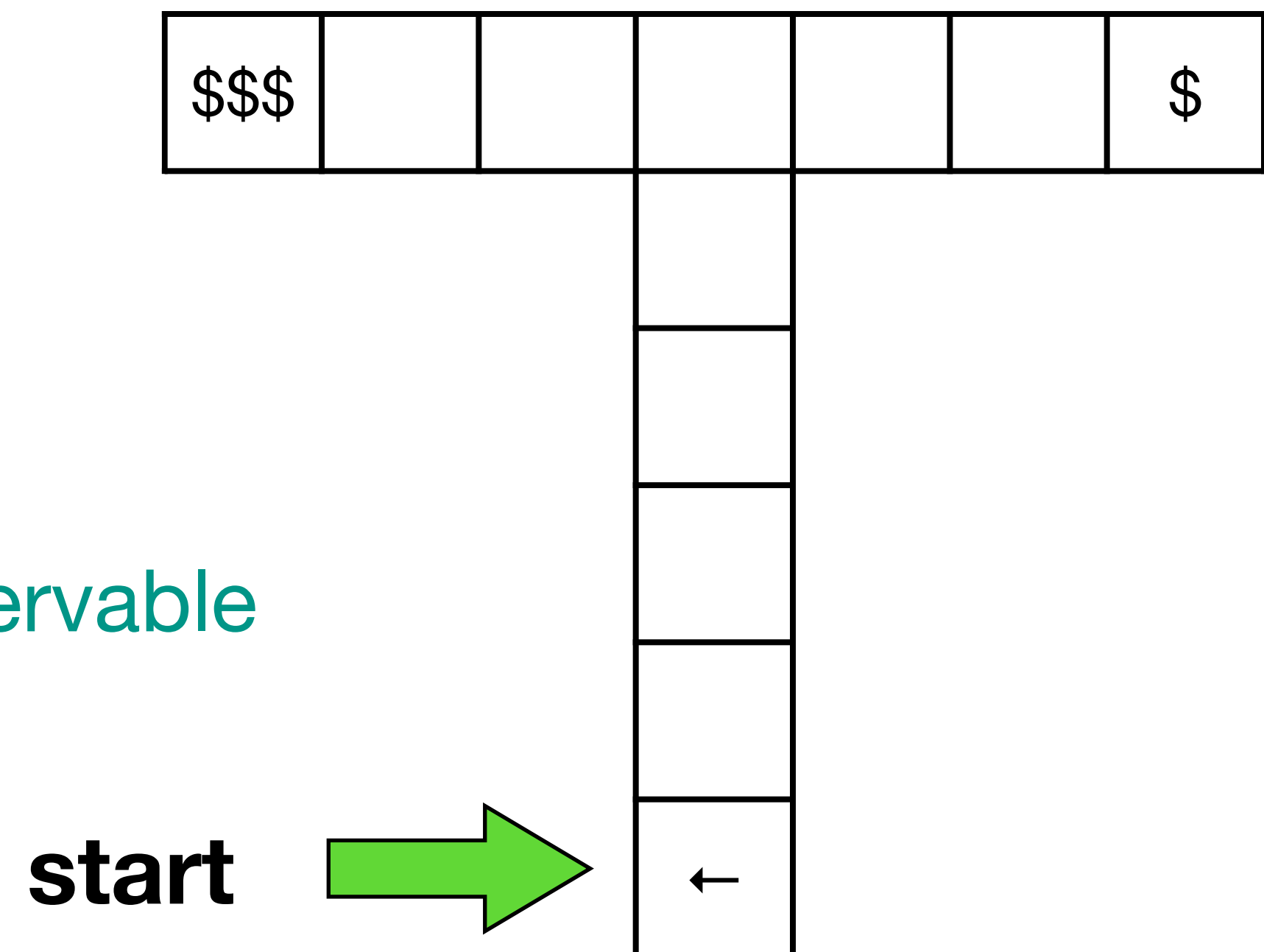
Partially Observable Markov Decision Process (POMDP)



- States \mathcal{S}
- Actions \mathcal{A}
- Observations \mathcal{O}
- Transitions $p(s_{t+1} | s_t, a_t)$
- Emissions (observation model) $p(o_t | s_t)$
- Rewards $r(s_t, a_t)$

T-maze domain

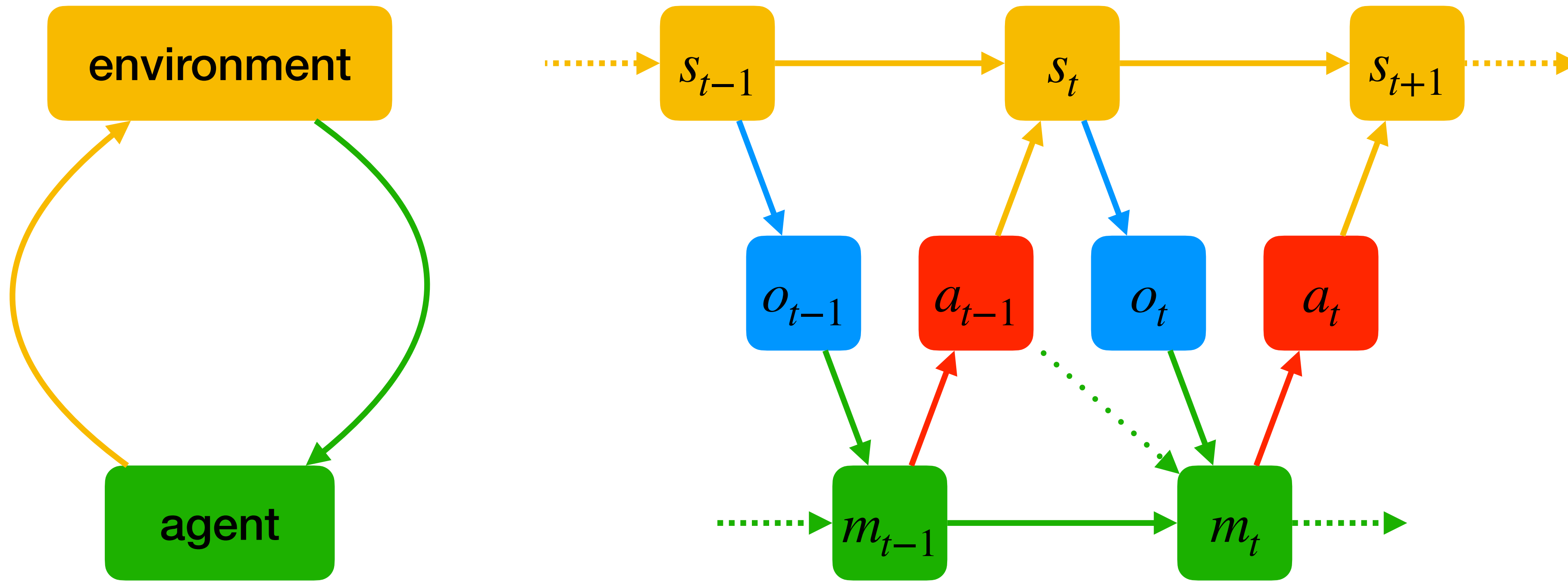
- **Observation**: current cell
- Observe **cue** at start
 - **Decision** at T-junction — cue no longer **observable**
- **Memory** is needed



What does the policy depend on? (revisited)

- Maximally: the entire **observable history** $\pi(a_t | h_t = (o_0, o_1, \dots, o_t))$
 - Should we remember past **actions**?
 - In a **stochastic policy** $\pi(a_t | h_t)$, yes: $h_t = (o_0, a_0, o_1, a_1, \dots, o_t)$
 - In a **deterministic policy** $\pi : h_t \mapsto a_t$, we could regenerate a_{t-1} from h_{t-1} (more compute)
- Problem: we can't have **unbounded memory** that grows with t
- Solution 1: keep a **window** of k last observations $\pi(a_t | o_{t-k+1}, \dots, o_t)$ (**frame stacking**)
- Solution 2: keep a **statistic** $m_t = \pi(h_t)$ or $\pi(m_t | h_t)$ of the observable history, use $\pi(a_t | m_t)$
 - **Memory** must allow sequential updates: $m_t = f(m_{t-1}, o_t)$ or $m_t = f(m_{t-1}, a_{t-1}, o_t)$

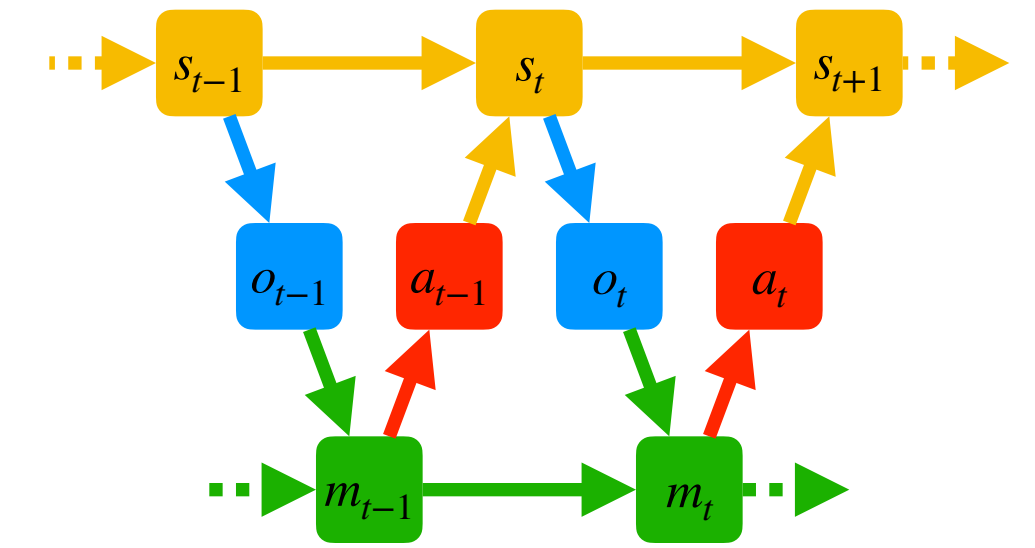
Agent–environment interaction



- Agent **policy**: $\pi(m_t, a_t | m_{t-1}, o_t) = \pi(m_t | m_{t-1}, o_t)\pi(a_t | m_t)$
- For simplicity, no edge $a_{t-1} \rightarrow m_t$
 - Can make a_{t-1} explicitly **observable** in o_t , or explicitly **remembered** in m_{t-1}

So what is memory?

- There's **no Markov property** in the observable process alone
 - All **past observations** may be informative of **future actions**
- **Filter** the observable past to provide more information about the hidden state
- No less important: **plan** for the future
- Previously, we needed to trade off **short-term** with **long-term** rewards
 - Now we also need to trade off with information-gathering = **active perception**
- In multi-agent: state of the world is incomplete without **other agent's memory**
 - **Theory of mind**



Tiger domain

- 2 states: which door leads to a tiger (-100 reward) and which to \$\$\$ (+10)
- You can stop and listen: $p(o_t = s_t | s_t) = 0.8$



$$p(s_0 = s_{\text{left}}) = 0.5$$

$$\mathbb{E}[r(s_0, a_{\text{left}})] = -45$$

$$o_1 = o_{\text{right}}$$

$$p(s_1 = s_{\text{left}}) = 0.2$$

$$\mathbb{E}[r(s_1, a_{\text{left}})] = -12$$

$$o_2 = o_{\text{left}}$$

$$p(s_2 = s_{\text{left}}) = 0.5$$

$$\mathbb{E}[r(s_2, a_{\text{left}})] = -45$$

$$o_3 = o_{\text{right}}$$

$$p(s_3 = s_{\text{left}}) = 0.2$$

$$\mathbb{E}[r(s_3, a_{\text{left}})] = -12$$

$$o_4 = o_{\text{right}}$$

$$p(s_4 = s_{\text{left}}) = \frac{0.04}{0.04 + 0.64} \approx 0.06 \quad \mathbb{E}[r(s_4, a_{\text{left}})] = -3.5$$

$$o_5 = o_{\text{right}}$$

$$p(s_5 = s_{\text{left}}) \approx 0.015$$

$$\mathbb{E}[r(s_4, a_{\text{left}})] = -8.3$$

Today's lecture

MPC, Local Models

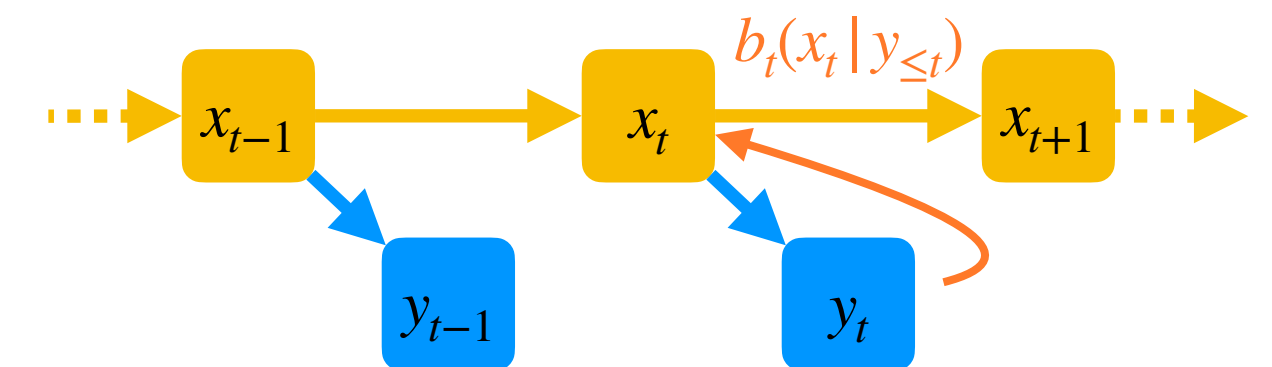
Partially Observable MDPs

Belief-state MDPs

RNNs

Belief

- **Belief** = distribution over the state $b(s)$
 - If the agent reaches belief b after history h , that **does not imply** $s \sim b$
- **Bayesian belief** $b_h(s) = p(s | h)$: a sufficient statistic of h for s
 - For a Bayesian belief: $s \sim b_h$ after history h
- In the linear–Gaussian case: the **Kalman filter**
 - Bayesian belief is **Gaussian** $p(x_t | h_t = y_{\leq t}) = \mathcal{N}(x_t; \hat{x}_t, \Sigma_t)$
 - Covariance can be **precomputed** $\mathbb{V}(x_t | h_t) = \Sigma_t$ (independent of h_t)
 - Mean can be **updated linearly**: $\hat{x}'_t = A\hat{x}_{t-1} + Bu_{t-1}$ $e_t = y_t - C\hat{x}'_t$ $\hat{x}_t = \hat{x}'_t + K_t e_t$



Computing the Bayesian belief

- **Predict** s_{t+1} from $h_t = (o_0, a_0, o_1, a_1, \dots, o_t)$ and a_t :

$$b'_t(s_{t+1} | h_t, a_t) = \sum_{s_t} p(s_t | h_t) p(s_{t+1} | s_t, a_t) = \sum_{s_t} b_t(s_t) p(s_{t+1} | s_t, a_t)$$

total probability over s_t
 previous belief b_t
 dynamics needs to be known

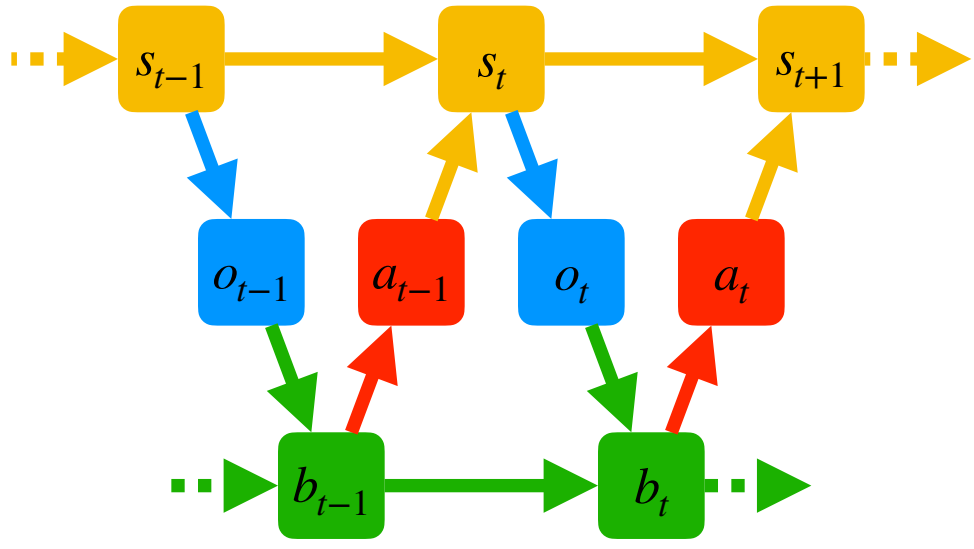
- **Update** belief of s_t after seeing $h_t = (h_{t-1}, a_{t-1}, o_t)$:

$$b_t(s_t | h_t) = \frac{p(s_t | h_{t-1}, a_{t-1}) p(o_t | s_t)}{p(o_t | h_{t-1}, a_{t-1})} = \frac{b'_{t-1}(s_t) p(o_t | s_t)}{\sum_{\bar{s}_t} b'_{t-1}(\bar{s}_t) p(o_t | \bar{s}_t)}$$

previous prediction
 observation model
Bayes' rule on o_t
 $o_t - s_t - (h_{t-1}, a_{t-1})$
 normalizer

- A **deterministic, model-based** update:

▶ $b_{t-1}(s_{t-1}) \rightarrow$ use a_{t-1} to **predict** $b'_{t-1}(s_t) \rightarrow$ use o_t to **update** $b_t(s_t)$



Belief-state MDP

- In the linear–quadratic–Gaussian case: **certainty equivalence**

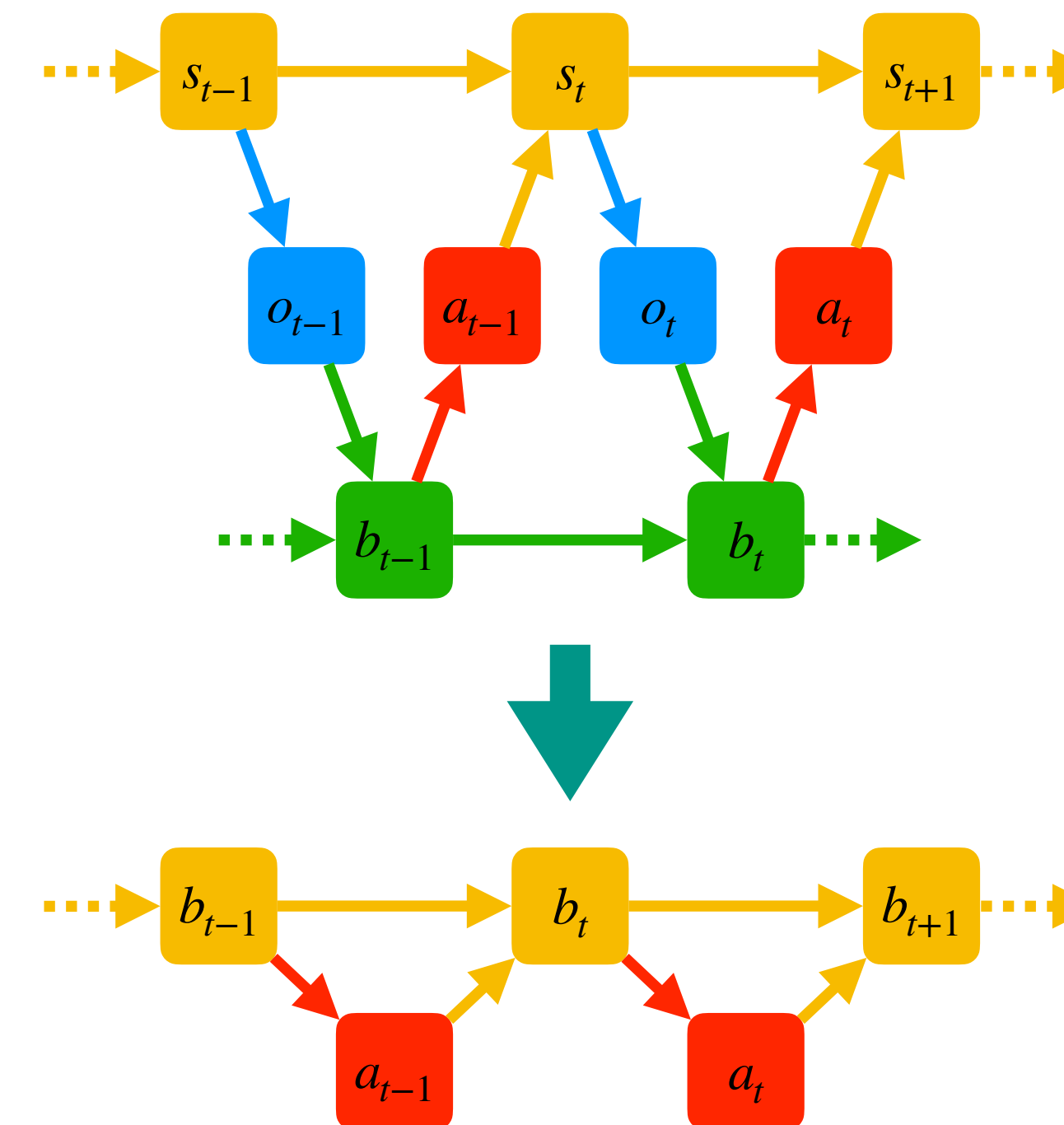
- Plan using \hat{x}_t as if it was x_t

- More generally (though vastly less useful): **belief-state MDP**

- States:** $\Delta(\mathcal{S})$ **Actions:** \mathcal{A} **Rewards:** $r(b_t, a_t) = \sum_{s_t} b_t(s_t)r(s_t, a_t)$

- Transitions:** each possible observation o_{t+1} contributes its probability

$$p(o_{t+1} | b_t, a_t) = \sum_{s_t, s_{t+1}} b_t(s_t)p(s_{t+1} | s_t, a_t)p(o_{t+1} | s_{t+1})$$



to the total probability that the belief that follows (b_t, a_t, o_{t+1}) is the **Bayesian belief**

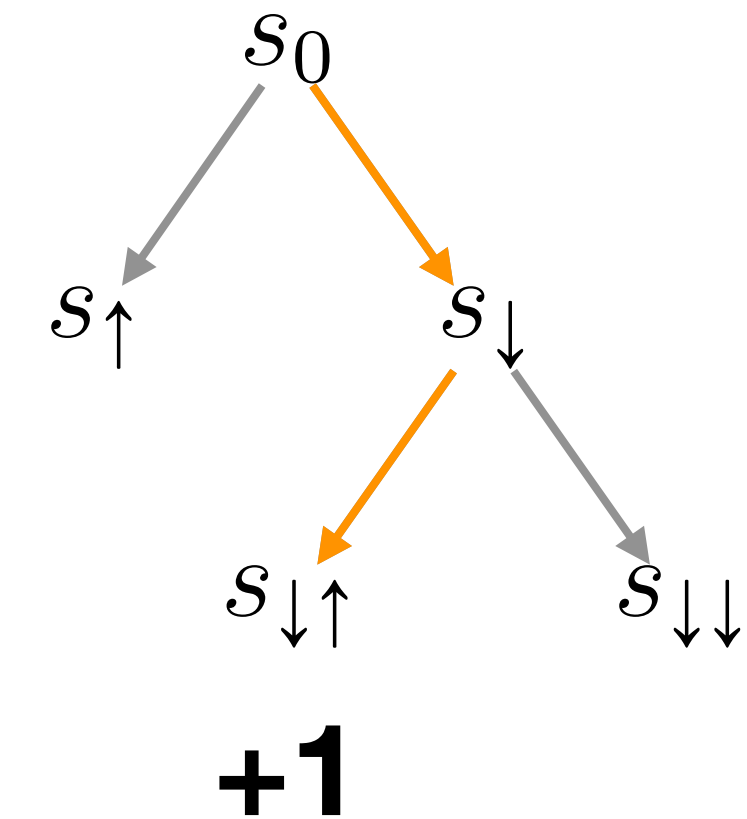
$$b_{t+1}(s_{t+1}) = p(s_{t+1} | b_t, a_t, o_{t+1}) = \frac{\sum_{s_t} b_t(s_t)p(s_{t+1} | s_t, a_t)p(o_{t+1} | s_{t+1})}{p(o_{t+1} | b_t, a_t)}$$

Learning to use memory is hard

- Belief space $b(s_t)$ is continuous and high-dimensional (dimension $|\mathcal{S}|$)
 - Curse of dimensionality
 - Beliefs are naturally multi-modal — how do we even represent them?
- The number of reachable beliefs may grow exponentially in t (one per h_t)
 - Curse of history
- Belief-value function can be very complex, hard to approximate
- There may not be optimal stationary deterministic policy \Rightarrow instability

Stationary deterministic policy counterexample

- Assume **no observability**
- Stationary deterministic policies gets **no reward**
- **Non-stationary** policy: \downarrow, \uparrow ; expected return: $+1$



▸ But non-stationary = observability of a clock t

- Stationary **stochastic policy**: \downarrow / \uparrow with equal prob.; expected return: $+0.25$

- Open problem: **Bellman optimality** is inherently stationary and deterministic

no dependence on t → $V(s) = \max_a r(s, a) + \gamma \mathbb{E}_{(s'|s,a) \sim p} [V(s')]$ **maximum achieved for some action**

Today's lecture

MPC, Local Models

Partially Observable MDPs

Belief-state MDPs

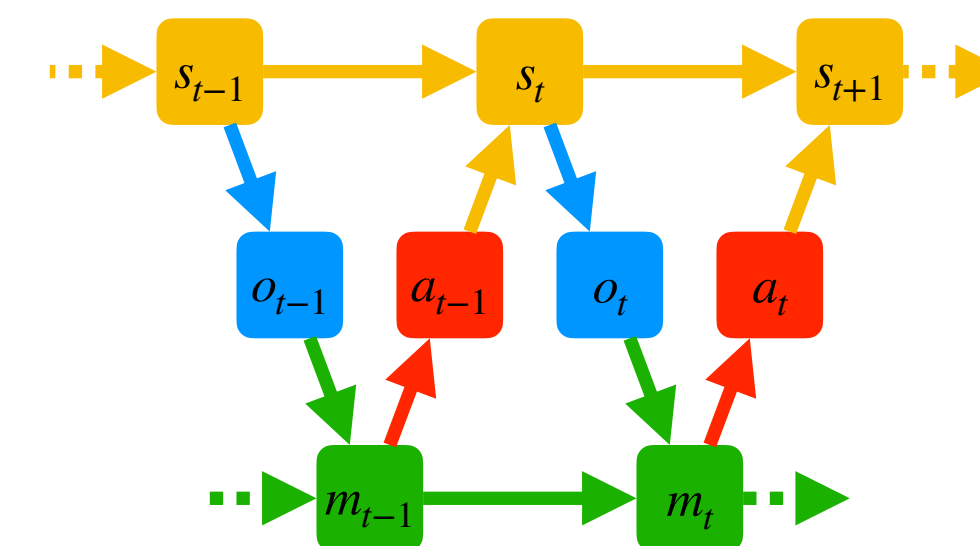
RNNs

Filtering with function approximation

- Instead of Bayesian belief: **memory update** $m_t = f_\theta(m_{t-1}, o_t)$ (a_{t-1} optional)

- ▶ **Action policy**: $\pi_\theta(a_t | m_t)$

- ▶ Sequential structure = **Recurrent Neural Network (RNN)**



- **Training**: back-propagate gradients through the whole sequence

- ▶ **Back-propagation through time (BPTT)**

- Unfortunately, gradients tend to **vanish** $\rightarrow 0$ / **explode** $\rightarrow \infty$

- ▶ **Long term coordination** of memory updates + actions is challenging

- ▶ RNN **can't use** information not remembered, but backup **no gradient** unless used

RNNs in on-policy methods

- Training RNNs with **on-policy methods** is straightforward (and backward)

- ▶ **Roll out policy**: parameters of a_t distribution are determined by $\pi_\theta(m_t)$ with

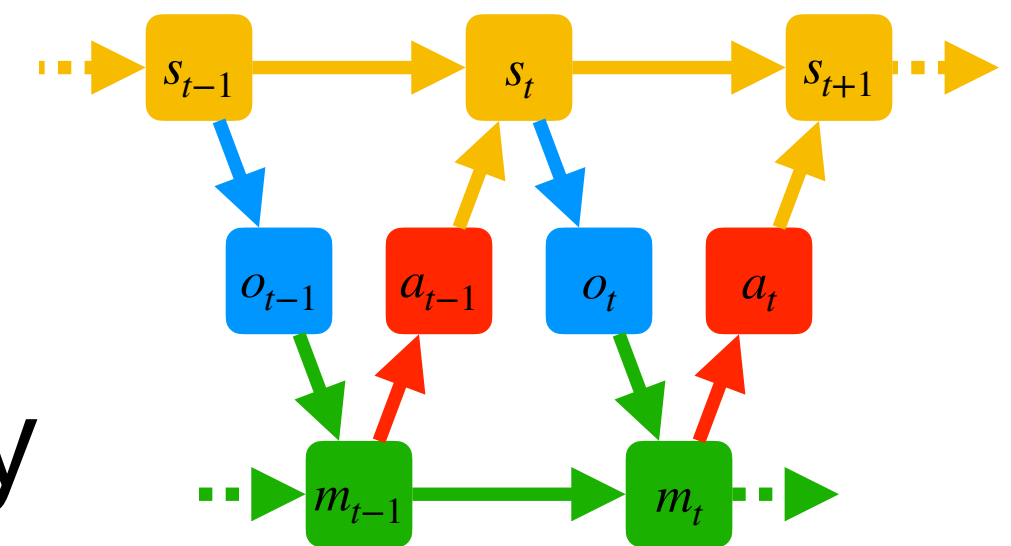
$$m_t = f_\theta(\dots f_\theta(f_\theta(o_0), o_1), \dots o_t)$$

- ▶ Compute $\nabla_\theta \log \pi_\theta(a_t | m_t)$ with **BPTT** all the way to initial observation o_0

- **Problems**: computation graph > **RAM**; **vanishing / exploding** grads

- ▶ **Solutions**: **stop gradients** every k steps; use **attention**

- **Problem**: cannot learn **longer memory** — but that's hard anyway



RNNs in off-policy methods

- **Problem:** RNN states in replay buffer disagree with current RNN params
- **Solution 1:** use n -step rollouts to reduce mismatch effect

$$Q_{\theta}(o_t, m_t, a_t) \rightarrow r_t + \gamma r_{t+1} + \dots + \gamma^{n-1} r_{t+n-1} + \gamma^n \max_{a'} Q_{\theta}(o_{t+n}, m_{t+n}, a')$$

- **Solution 2:** “burn in” m_t from even earlier stored steps
 - Same target, but m_t is initialized from $(o_{t-k}, \dots, o_{t-1})$
- **In practice:** RNNs rarely used
 - **Stacking k frames** every step (o_{t-k+1}, \dots, o_t) may help with short-term memory

Deep RL as partial observability

- Memory-based policies fail us in **Deep RL**, where we need them most:
 - Deep RL is inherently **partially observable**
- Consider what **deeper layers** get as input:
 - High-level / action-relevant state features are **not Markov!**
- **Memory management** is a huge open problem in Deep RL
 - Actually, in other areas of ML too: NLP, time-series analysis, video processing, ...

Recap and further considerations

- Let policies depend on **observable history** through **memory**
- **Memory update**: Bayesian, approximate, or learned
 - **Learning to update memory** is one of the biggest open problems in all of ML
- Let policy be **stochastic**
 - Should memory be stochastic? interesting research question...
- Let policies be **non-stationary** if possible, otherwise learning may be unstable
 - **Time-dependent** policies for finite-horizon tasks
 - **Periodic** policies for periodic tasks