Aalto University
School of Electrical
Engineering

ELEC-E8125 Reinforcement Learning Interleaved learning and planning in model-based RL

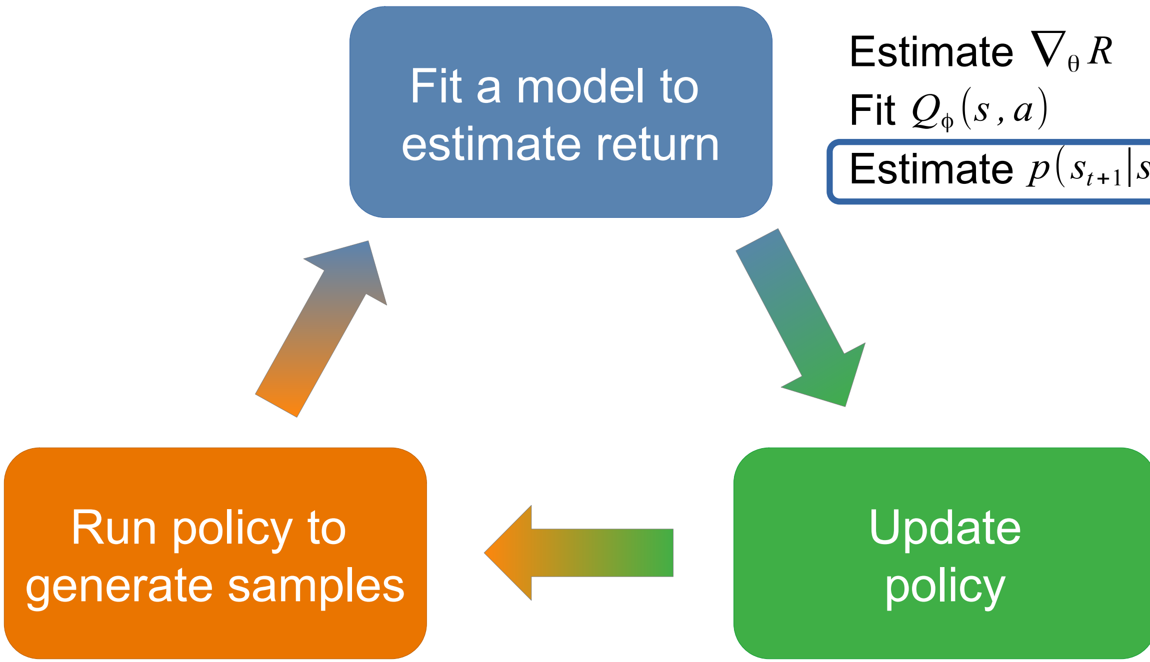
Joni Pajarinen

1.11.2022

Learning goals

- Understand how learning and planning are used together in model-based reinforcement learning

Anatomy of reinforcement learning
Model-based



Estimate $\nabla_\theta R$ (policy gradient)
Fit $Q_\phi(s, a)$ (Q-learning, actor-critic)
Estimate $p(s_{t+1}|s_t, a_t)$ (model-based)

$\theta \leftarrow \theta + \alpha \nabla_\theta R(\theta)$ (policy gradient)
$arg\,max_u Q_\phi(s, a)$ (Q-learning)
Optimize $\pi_\theta(a|s)$ (model-based)

Adopted from Sergey Levin.

Motivation (partial recap)

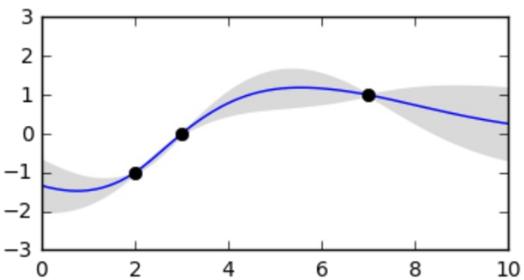
- Reinforcement learning has limited sample efficiency
- Learned policies are task(reward-function)-specific, learned policies cannot be directly reused
- Learned dynamics model is reusable and can be used to reason about potential futures
- Sometimes we know the model, e.g. in games!

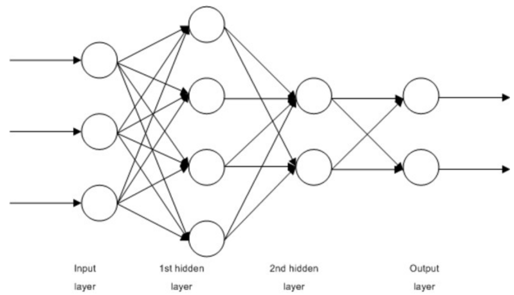
How to specify the model? Are there other situations where we know the model?

Model definition and types

- Dynamics model $\boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \boldsymbol{a}_t)$ or $f(\boldsymbol{s}_{t+1} | \boldsymbol{s}_t, \boldsymbol{a}_t)$
- Reward model $r_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t)$ or $r(r_t | \boldsymbol{s}_t, \boldsymbol{a}_t)$
- Models are usually learned
  - Parametric regression (e.g. neural net) common
- May be also known (e.g. games, simulators)
  - Even physics based models need to be often calibrated
- Also other possibilities (active research area)
  - Latent variable models, graph neural networks, non-parametric regression models such as Gaussian processes, ...

Which model to use?





$$Y_i = \beta_0 + \beta_1 \phi_1(X_{i1}) + \cdots + \beta_p \phi_p(X_{ip})$$

Gaussian process (GP)

- Data-efficient
- Slow with big datasets
- May be too smooth for non-smooth dynamics

Neural networks (NNs)

- Expressive
- Unpredictable with sparse data (overfit)
  - NN ensembles estimate uncertainty

Linear models

- May be used locally
- Do not overfit

Domain specific parametric models (e.g. physics parameters) can also be used.
→ Traditional control engineering approach of model identification + control.

Spectrum of model-based RL

how to act in
current situation
(choose action)

Time of planning

learn to act in
any situation
(learn policy)

On-line (every time step)

- Act on current state
- Act without learning
- Better in unfamiliar situations

Off-line (use single / multiple episodes)

- Fast online computation
- Predictable within familiar situations

Spectrum of model-based RL

Time of planning

- **On-line**
  - **Discrete actions**
    - MCTS
    - ...
  - **Continuous actions**
    - iLQR
    - CEM
    - ...
- **Off-line**
  - **Simulate environment**
    - DYNA
    - Latent dynamics
    - ...
  - **Assist learning**
    - Policy backprop
    - ...

Spectrum of model-based RL

Time of planning

- **On-line**
  - **Discrete actions**
    - MCTS
    - ...
  - **Continuous actions**
    - iLQR
    - CEM
    - ...
- **Off-line**
  - **Simulate environment**
    - DYNA
    - Latent dynamics
    - ...
  - **Assist learning**
    - Policy backprop
    - ...

We kind of saw this already last week.

Continuous on-line planning: iLQR + learned model

Input: base policy π_0

Run base policy to collect data $D \leftarrow \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')_i\}$

Repeat

Fit dynamics model $f(\boldsymbol{s}, \boldsymbol{a})$ to minimize $\sum_i \|f(\boldsymbol{s}_i, \boldsymbol{a}_i) - \boldsymbol{s}_i'\|^2$

Use model to plan (e.g. iLQR, CEM) actions

Execute first planned action, observe resulting state $\boldsymbol{s}'$

Update dataset $D \leftarrow D \cup \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')\}$

Continuous on-line planning: iLQR + learned model

Input: base policy π_0

Run base policy to collect data $D \leftarrow \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')_i\}$

Repeat

Fit dynamics model $f(\boldsymbol{s}, \boldsymbol{a})$ to minimize $\sum_i \| f(\boldsymbol{s}_i, \boldsymbol{a}_i) - \boldsymbol{s}_i' \|^2$

Use model to plan (e.g. iLQR, CEM) actions

Execute first planned action, observe resulting state $\boldsymbol{s}'$

Update dataset $D \leftarrow D \cup \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')\}$

- Sample efficient
- Computationally expensive for two reasons
  - Dynamics fitting costly → model may be fitted only periodically (every n steps)
  - Planning costly for long horizons
- Robust to moderate model errors
- Choice of regression model is an important design parameter

This is model-predictive control (MPC) with learned dynamics.
MPC horizon length is limited, can we do something?

Continuous on-line planning: iLQR + learned model

Input: base policy π_0

Run base policy to collect data $D \leftarrow \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')_i\}$

Repeat

Fit dynamics model $f(\boldsymbol{s}, \boldsymbol{a})$ to minimize $\sum_i \|f(\boldsymbol{s}_i, \boldsymbol{a}_i) - \boldsymbol{s}_i'\|^2$

Use model to plan (e.g. iLQR, CEM) actions

Execute first planned action, observe resulting state $\boldsymbol{s}'$

Update dataset $D \leftarrow D \cup \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')\}$

- Sample efficient.
- Computationally expensive for two reasons
  - Dynamics fitting costly → model may be fitted only periodically (every n steps)
  - **Planning costly for long horizons**
- Robust to moderate model errors
- Choice of regression model is an important design parameter

This is model-predictive control (MPC) with learned dynamics.
MPC horizon length is limited, can we do something?

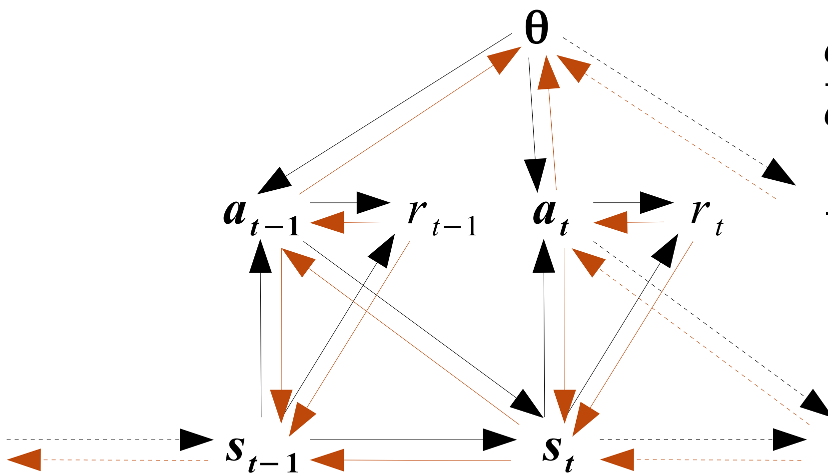
Spectrum of model-based RL

Time of planning

- **On-line**
  - **Discrete actions**
    - MCTS
    - ...
  - **Continuous actions**
    - iLQR
    - CEM
    - ...
- **Off-line**
  - **Simulate environment**
    - DYNA
    - Latent dynamics
    - ...
  - **Assist learning**
    - Policy backprop
    - ...

Backprop ~ chain rule of partial derivatives

Combining parametric policy with learned dynamics by backpropagation



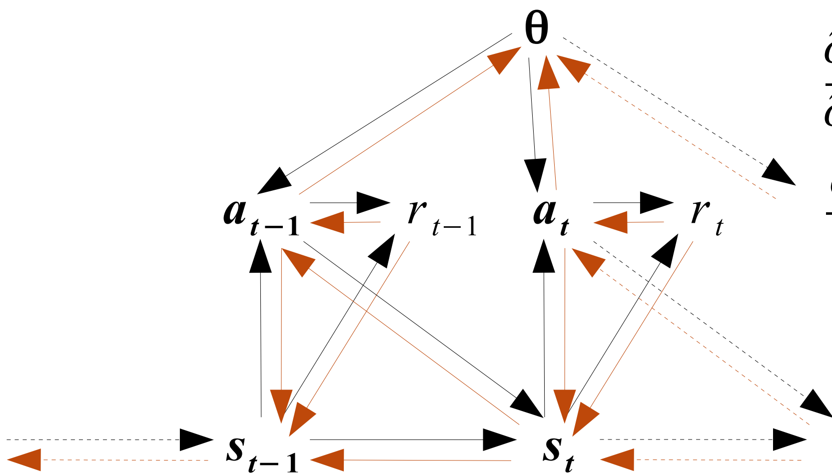
$$\frac{\partial r_t}{\partial \theta} = \frac{\partial r_t}{\partial a_t}\frac{\partial a_t}{\partial \theta} + \frac{\partial r_t}{\partial s_t}\frac{\partial s_t}{\partial \theta}$$

$$\frac{\partial s_t}{\partial \theta} = \frac{\partial s_t}{\partial s_{t-1}}\frac{\partial s_{t-1}}{\partial \theta} + \frac{\partial s_t}{\partial a_{t-1}}\frac{\partial a_{t-1}}{\partial \theta}$$

policy	reward	dynamics
$\nabla_\theta \pi(s_t)$	$\nabla_a r(s_t, a_t)$	$\nabla_s f(s_{t-1}, a_{t-1})$
	$\nabla_s r(s_t, a_t)$	$\nabla_a f(s_{t-1}, a_{t-1})$

Backprop ~ chain rule of partial derivatives

Combining parametric policy with learned dynamics by backpropagation



$$\frac{\partial r_t}{\partial \theta} = \frac{\partial r_t}{\partial a_t}\frac{\partial a_t}{\partial \theta} + \frac{\partial r_t}{\partial s_t}\frac{\partial s_t}{\partial \theta}$$

$$\frac{\partial s_t}{\partial \theta} = \frac{\partial s_t}{\partial s_{t-1}}\frac{\partial s_{t-1}}{\partial \theta} + \frac{\partial s_t}{\partial a_{t-1}}\frac{\partial a_{t-1}}{\partial \theta}$$

Run base policy to collect data $D \leftarrow \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')_i\}$

Repeat

- Fit dynamics model $f_\phi(\boldsymbol{s}, \boldsymbol{a})$ to minimize $\sum_i \|f_\phi(\boldsymbol{s}_i, \boldsymbol{a}_i) - \boldsymbol{s}_i'\|^2$
- Calculate policy gradient update by backpropagating through dynamics
- Execute updated policy (1 or more steps), collect data
- Update dataset $D \leftarrow D \cup \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')\}$

Tools handle this automatically by automatic differentiation.

Continuous on-line planning: iLQR + learned model

Input: base policy π_0

Run base policy to collect data $D \leftarrow \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')_i\}$

Repeat

Fit dynamics model $f(\boldsymbol{s}, \boldsymbol{a})$ to minimize $\sum_i \|f(\boldsymbol{s}_i, \boldsymbol{a}_i) - \boldsymbol{s}_i'\|^2$

Use model to plan (e.g. iLQR, CEM) actions

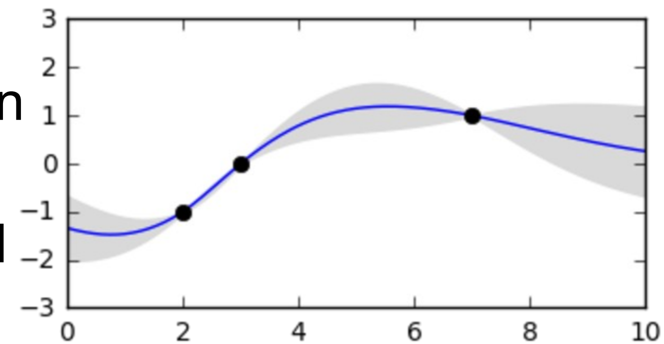
Execute first planned action, observe resulting state $\boldsymbol{s}'$

Update dataset $D \leftarrow D \cup \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')\}$

- Sample efficient
- Computationally expensive for two reasons
  - Dynamics fitting costly → model may be fitted only periodically (every n steps)
  - Planning costly for long horizons
- Robust to moderate model errors
- $\boxed{\textbf{Choice of regression model is an important design parameter}}$

Example
PILCO (Deisenroth&Rasmussen, 2011)

- Dynamics learning: Use Gaussian process models to include model uncertainty. Known quadratic reward
- Simulation: Simulate trajectory with learned model, including uncertainty
- Policy: Radial basis function
- Policy update: Calculate analytically policy gradient using learned dynamics and optimize with quasi-Newton optimizer (BFGS)
- GP → Very sample efficient. Cannot handle a large dataset



Reward function can also be learned using GP, e.g. BlackDROPS (2017).

Spectrum of model-based RL

Time of planning

On-line

Discrete actions
- MCTS
- ...

Continuous actions
- iLQR
- CEM
- ...

Off-line

Simulate environment
- DYNA
- Latent dynamics
- ...

Assist learning
- Policy backprop
- ...

Simulate environment to generate additional data: DYNA

Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Loop forever:

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

(b) $A \leftarrow \varepsilon\text{-greedy}(S, Q)$

(c) Take action A; observe resultant reward, R, and state, S'

(d) $\boxed{Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]}$

(e) $\boxed{Model(S,A) \leftarrow R, S' \text{ (assuming deterministic environment)}}$

(f) Loop repeat n times:

$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

$\boxed{R, S' \leftarrow Model(S,A)}$

$\boxed{Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]}$

Update using experience

Learn dynamics model

Generate data by simulating dynamics

Update using simulated experience

Simulate environment to generate additional data: latent dynamics motivation

- Dynamics $f\left(\boldsymbol{s}_{t+1} \middle| \boldsymbol{s}_t, \boldsymbol{a}_t\right)$
- Reward model $r\left(r_t \middle| \boldsymbol{s}_t, \boldsymbol{a}_t\right)$

- Do we need to find an exact dynamics model that is valid for every possible state and action?
- What about learning only a model that allows us to perform the task?
- Some states may share identical optimal policies. Can we take advantage of this somehow?

Simulate environment to generate additional data: latent dynamics

- Real dynamics $f(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$
- Real reward model $r(r_t|\boldsymbol{s}_t, \boldsymbol{a}_t)$

- Latent state $\boldsymbol{q}_t$
- Latent dynamics model $f(\boldsymbol{q}_t|\boldsymbol{q}_{t-1}, \boldsymbol{a}_{t-1}, \boldsymbol{o}_{t-1})$ and $f(\boldsymbol{q}_t|\boldsymbol{q}_{t-1}, \boldsymbol{a}_{t-1})$
- Latent reward model $r(r_t|\boldsymbol{q}_t)$
- Policy $\pi(\boldsymbol{a}_t|\boldsymbol{q}_t)$
- Value function $v(\boldsymbol{q}_t)$

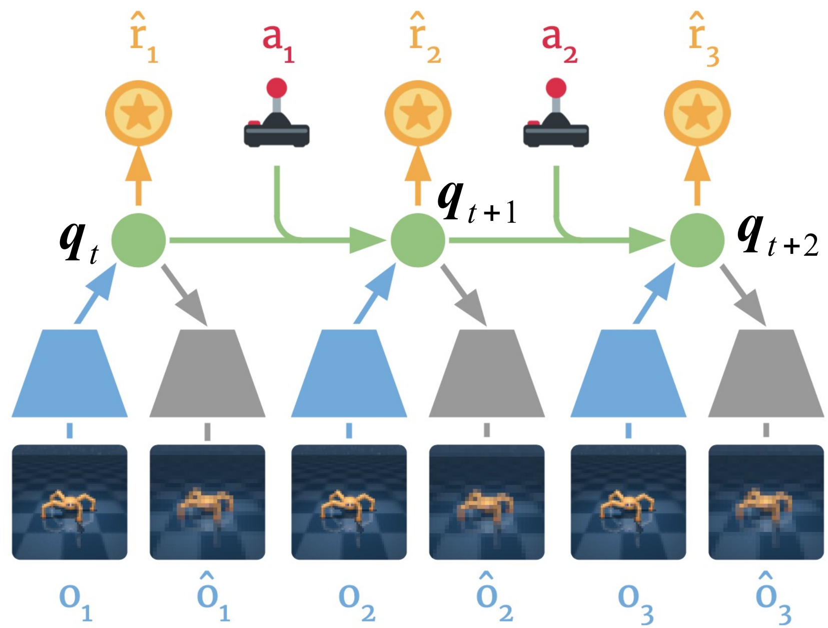
Observation of the state

Model from Dreamer [Hafner et al., ICLR 2019]

Dreamer: learn latent dynamics

- For real world data tuples $(\boldsymbol{o}_t, \boldsymbol{a}_t, \boldsymbol{r}_t)$ update latent state using $f(\boldsymbol{q}_t | \boldsymbol{q}_{t-1}, \boldsymbol{a}_{t-1}, \boldsymbol{o}_{t-1})$
- and to match real world data update latent models:

$$f(\boldsymbol{q}_t | \boldsymbol{q}_{t-1}, \boldsymbol{a}_{t-1}, \boldsymbol{o}_{t-1})$$
$$f(\boldsymbol{q}_t | \boldsymbol{q}_{t-1}, \boldsymbol{a}_{t-1})$$
$$r(r_t | \boldsymbol{q}_t)$$



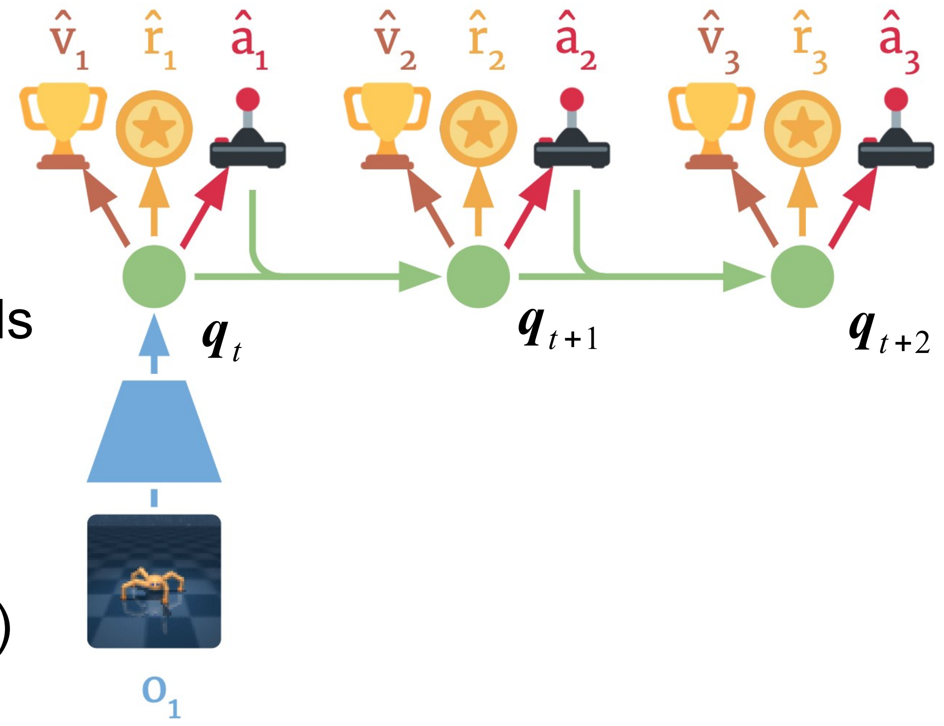
Picture adapted from Dream to Control: Learning Behaviors by Latent Imagination [Hafner et al., ICLR 2019]

Dreamer: learn behavior by policy backprop

- Simulate dynamics using $f(\boldsymbol{q}_t | \boldsymbol{q}_{t-1}, \boldsymbol{a}_{t-1})$
- Estimate value $v(\boldsymbol{q}_t)$ and rewards
- Update policy $\pi(\boldsymbol{a}_t | \boldsymbol{q}_t)$

  to maximize value

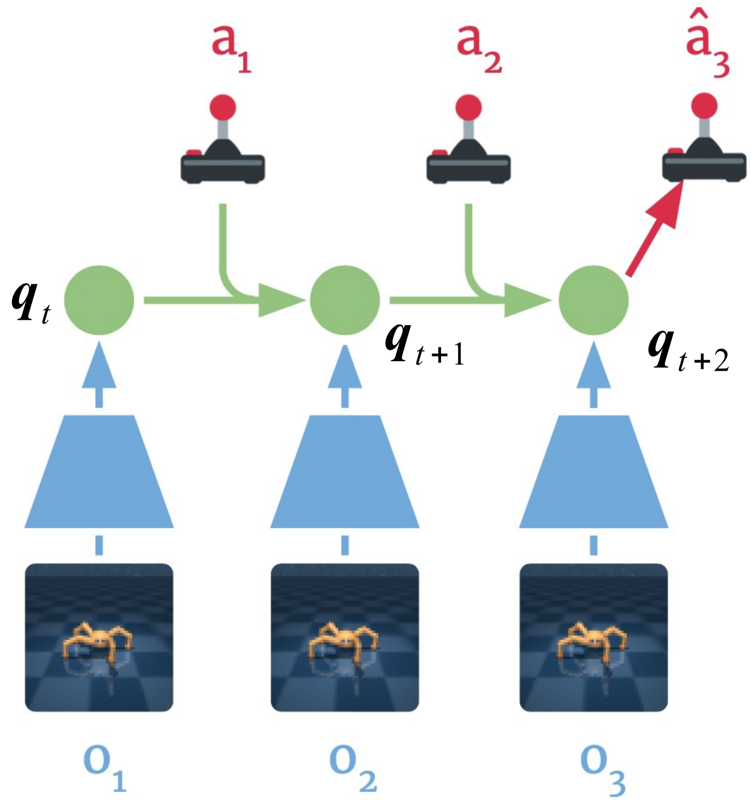
  using policy backprop through dynamics (discussed on slide 14)



Picture adapted from Dream to Control: Learning Behaviors by Latent Imagination [Hafner et al., ICLR 2019]

Dreamer: act in the real world

- To collect real world data sample actions from policy $\pi(\boldsymbol{a}_t|\boldsymbol{q}_t)$
  and update latent state using $f(\boldsymbol{q}_t|\boldsymbol{q}_{t-1},\boldsymbol{a}_{t-1},\boldsymbol{o}_{t-1})$



Picture adapted from Dream to Control: Learning Behaviors by Latent Imagination [Hafner et al., ICLR 2019]

Spectrum of model-based RL

Time of planning

- **On-line**
  - **Discrete actions**
    - MCTS
    - ...
  - **Continuous actions**
    - iLQR
    - CEM
    - ...
- **Off-line**
  - **Simulate environment**
    - DYNA
    - Latent dynamics
    - ...
  - **Assist learning**
    - Policy backprop
    - ...

The ideas can also be combined!

Summary

- Model-based RL requires typically less data than value-based or policy gradient approaches
- Sometimes learned dynamics can be transferred across tasks
- Potentially suboptimal: policy optimization with approximate models may lead to suboptimal solutions and approximate methods to local minima
- Sometimes models are harder to learn than policy
- Often explicit choices required (e.g. time horizon)

Next: exploration / exploitation

- Next week: how to choose actions to find optimal policy?
  - Choose always the best action?
  - But we do not know the best action before we try actions out!
  - How to balance exploration (trying out) with exploitation (choosing what seems the best at the moment)?
  - Monte Carlo tree search (MCTS): balancing exploration vs. exploitation in model-based planning