Background Planning

Christopher Mutschler

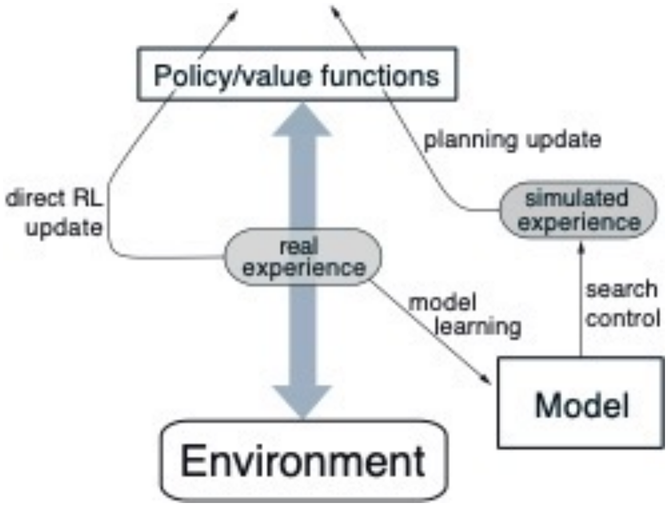
Outline

- Motivation: why model-based RL?
- What is a model? What are its inputs? What is a good model?
- **How can we use a model?**
  - **Background Planning**
    - **Environment data augmentation / simulation**
    - Sample-efficient policy learning
  - Online Planning
    - Discrete Actions
    - Continuous Actions
  - Auxiliary tasks
- Real-world application

Environment Data Augmentation

Dyna Architecture: Dyna-Q

- Use collected data to learn a transition and reward model
- Train a traditional RL algorithm (e.g., Q-Learning) using both environment data (real experience) and data generated from the learned model (simulated experience)



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) $A \leftarrow \varepsilon$-greedy(S, Q)
(c) Take action A; observe resultant reward, R, and state, S'
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
(f) Loop repeat n times:
$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previously taken in S
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Environment Data Augmentation

Model-Based Policy Optimization

- Use collected data to learn $p_\theta(s', r|\, s, a)$, i.e., a predictive model of the environment (transition model)
- Apply traditional policy gradient methods on synthetic model rollouts
- Take action in real environment

Algorithm 2 Model-Based Policy Optimization with Deep Reinforcement Learning

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

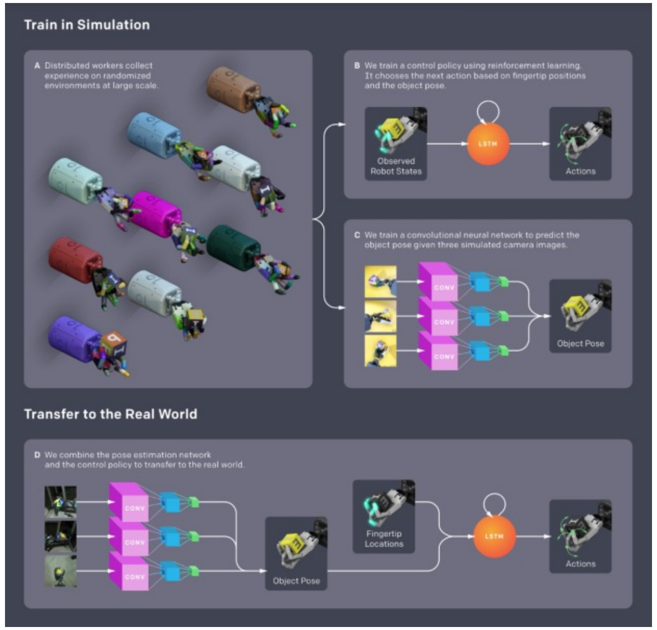
Janner et al.: When to Trust Your Model: Model-Based Policy Optimization. NeurIPS 2019.

Environment Data Augmentation

Domain Randomization & Sim2Real

- If we have an available simulator (model), we can train an RL agent there
- But the simulation will always be different compared to the real system
- Solution: learn a good policy on a *“distribution of similar environments”*, differing in some physical parameters (e.g., masses or image textures)
- This way, the real system will be “another variation” for the policy

- *Note: seems super-simple but works remarkably in practice!*



Andrychowicz, OpenAI: Marcin, et al. "Learning dexterous in-hand manipulation." The International Journal of Robotics Research 39.1 (2020): 3-20.

Outline

- Motivation: why model-based RL?
- What is a model? What are its inputs? What is a good model?
- **How can we use a model?**
  - **Background Planning**
    - Environment data augmentation / simulation
    - **Sample-efficient policy learning**
  - Online Planning
    - Discrete Actions
    - Continuous Actions
  - Auxiliary tasks
- Real-world application

Sample-efficient Policy Learning

Idea: train model and policy jointly end-to-end

- In other words: do what successfully worked in other domains (such as computer vision, speech recognition, etc.)

- Goal: maximize reward of parametric policy:

$$J(\theta) = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t), \qquad \text{with } a_t = \pi_\theta(s_t) \text{ and } s_{t+1} = T(s_t, a_t)$$

- Just apply gradient ascent on policy gradient $\nabla_\theta J$.

- But how to calculate $\nabla_\theta J$?
- Remember REINFORCE
  - High-variance
  - Requires stochastic policy

Sample-efficient Policy Learning

We can do more!

- Smooth models offer derivatives:

$$s_{t+1} = f_s(s_t, a_t) \qquad r_t = f_r(s_t, a_t)$$

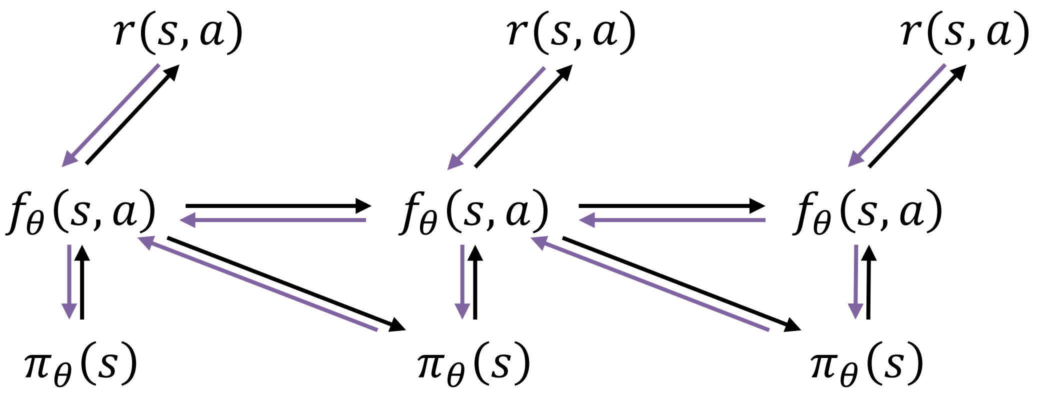
$$\nabla_{s_t}(s_{t+1}), \nabla_{a_t}(s_{t+1}), \nabla_{s_t}(r_t), \nabla_{a_t}(r_t), \dots$$

- How do small changes in action affect the next state?
- How do small changes in states affect the rewards?
- …

→ Allows end-to-end differentiation via backpropagation!

Policy Backprop

Back-propagate through the model to optimize the policy



Backprop:

$$\max_\theta \sum_t \gamma^t R(s_t, a_t)$$

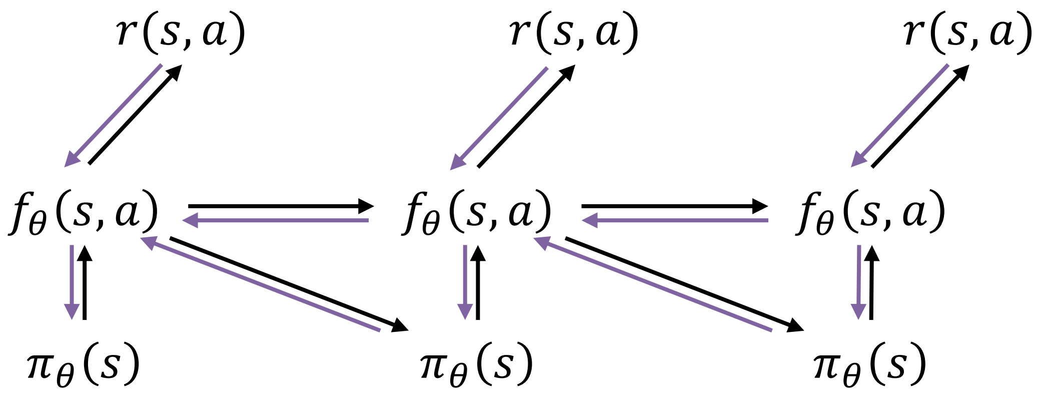
Simple Algorithm:

1. Run a base policy $\pi_0(a_t|s_t)$ (e.g., a random policy) to collect data samples $\mathcal{D}\{(s,a,s')_i\}$
2. Learn a dynamics model $f_\theta(s,a)$ by minimizing $\sum_i \|f_\theta(s_i,a_i) - s_i'\|^2$
3. Backpropagate through $f_\theta(s,a)$ into policy to optimize $\pi_\theta(a_t|s_t)$

Remember: Distribution Mismatch!

Policy Backprop

Back-propagate through the model to optimize the policy



Backprop:

$$\max_\theta \sum_t \gamma^t R(s_t, a_t)$$

Better Algorithm:

1. Run a base policy $\pi_0(a_t|s_t)$ (e.g., a random policy) to collect data samples $\mathcal{D}\{(s,a,s')_i\}$
2. Learn a dynamics model $f_\theta(s,a)$ by minimizing $\sum_i \|f_\theta(s_i,a_i) - s_i'\|^2$
3. Backpropagate through $f_\theta(s,a)$ into policy to optimize $\pi_\theta(a_t|s_t)$
4. Run $\pi_\theta(a_t|s_t)$
5. Append visited tuples (s,a,s') to $\mathcal{D}$

Policy Backprop

Back-propagate through the model to optimize the policy

1. Approximate transitions and rewards with differentiable models
2. Calculate policy gradient via back-prop-through-time (BPTT)

- **Pros:**
  - Long-term credit assignment
  - Differentiable transitions and rewards models → sample efficiency
  - Principles behind BPTT well understood
  - Deterministic & no variance involved
- **Cons:**
  - Similar problems to training long RNNs with BPTT → poor conditioning
    - Vanishing and exploding gradients
    - Unlike LSTM, we cannot just “choose” simple dynamics as dynamics are chosen by nature.
  - Prone to local minima