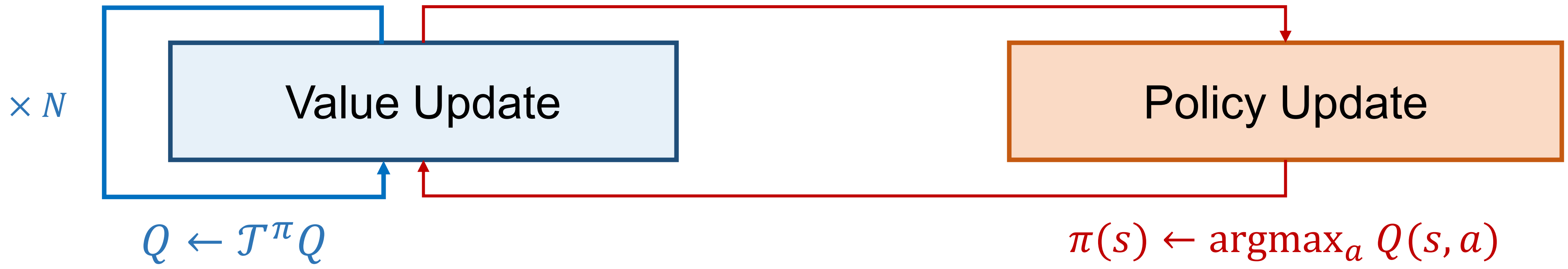
Model-Based RL

Chen-Yu Wei

Recap: Model-Free RL



$Q \leftarrow \mathcal{T}^{\pi} Q$ means $\quad Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s',a'} P(s'|s,a)\,\pi(a'|s')Q(s',a')$ $\quad$ for all s, a

Recap: Generalized Policy Iteration with Samples

For $k = 1,\ 2, \dots$

  For $i = 1, 2, \dots, N$:

    Choose action a_i with the current policy

    Receive reward $r_i \sim R(s_i, a_i)$ and $s_i' \sim P(\cdot\,|s_i, a_i)$

    $s_{i+1} = s_i'$ if episode continues, $s_{i+1} \sim \rho$ if episode ends

    Push (s_i, a_i, r_i, s_i') to $\mathcal{B}$

Data collection

  Draw (s, a, r, s') from $\mathcal{B}$, and use them to update policy/value

  Empty $\mathcal{B}$ if under on-policy training

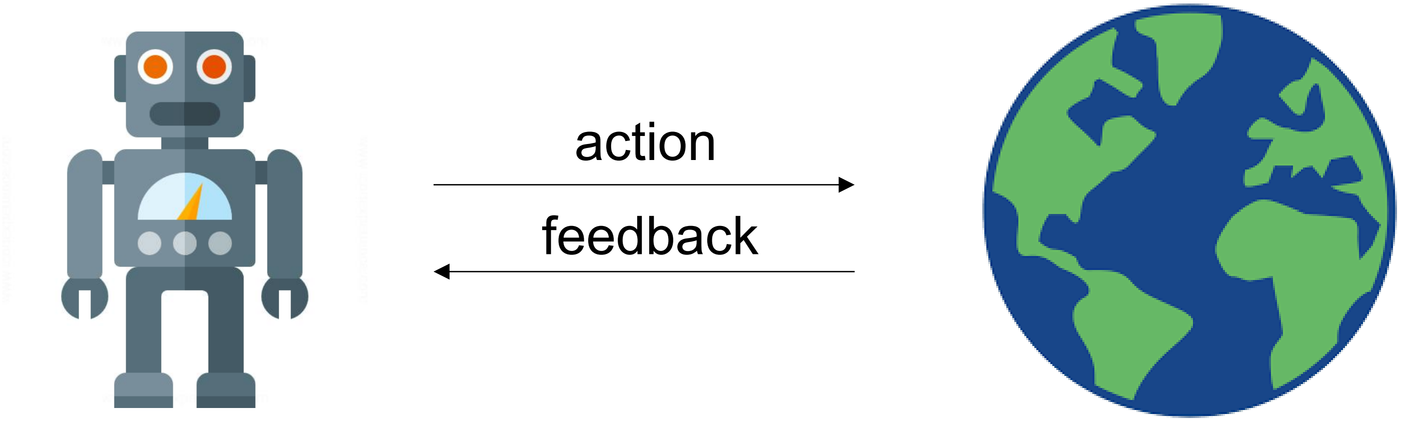
Policy / value update

When Is Model-Based Method Helpful?

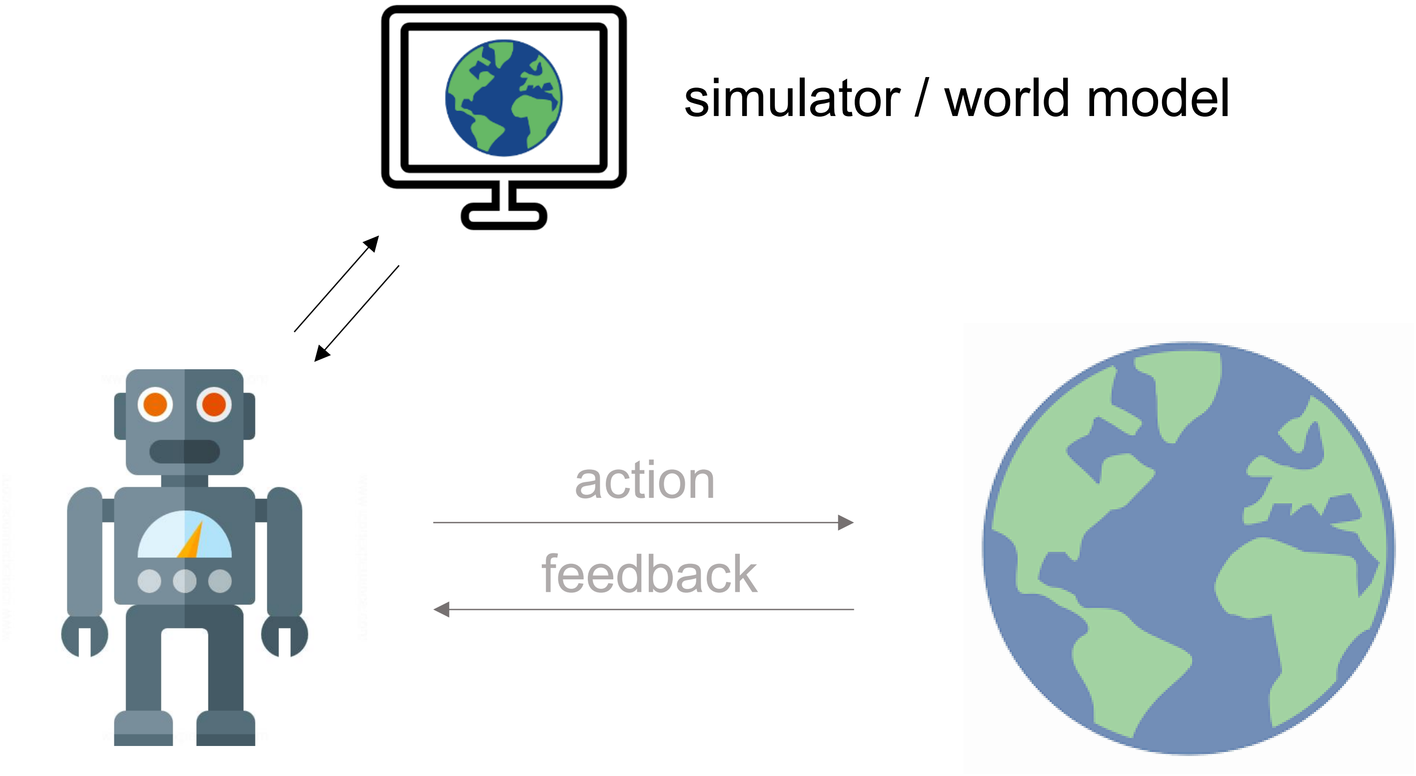
- Model (transition) is easy to learn
  - Deterministic transition could be easier to learn to stochastic one
  - System identification: known parameterized model with unknown parameters

- Model is known
  - The space/action space is too large for full policy/value iteration (Go, Chess)

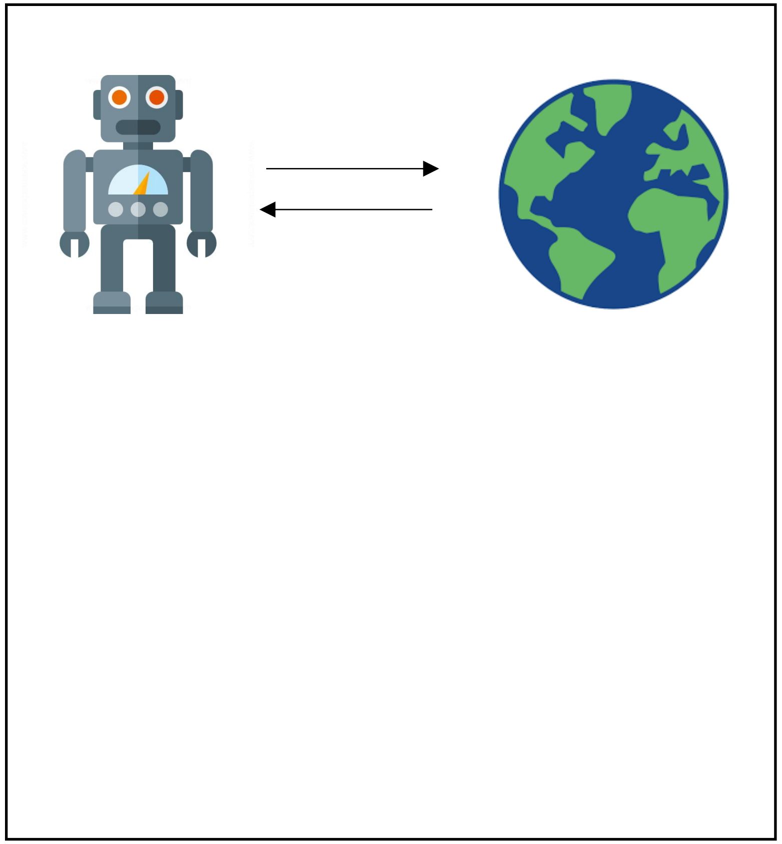
Model-Based RL



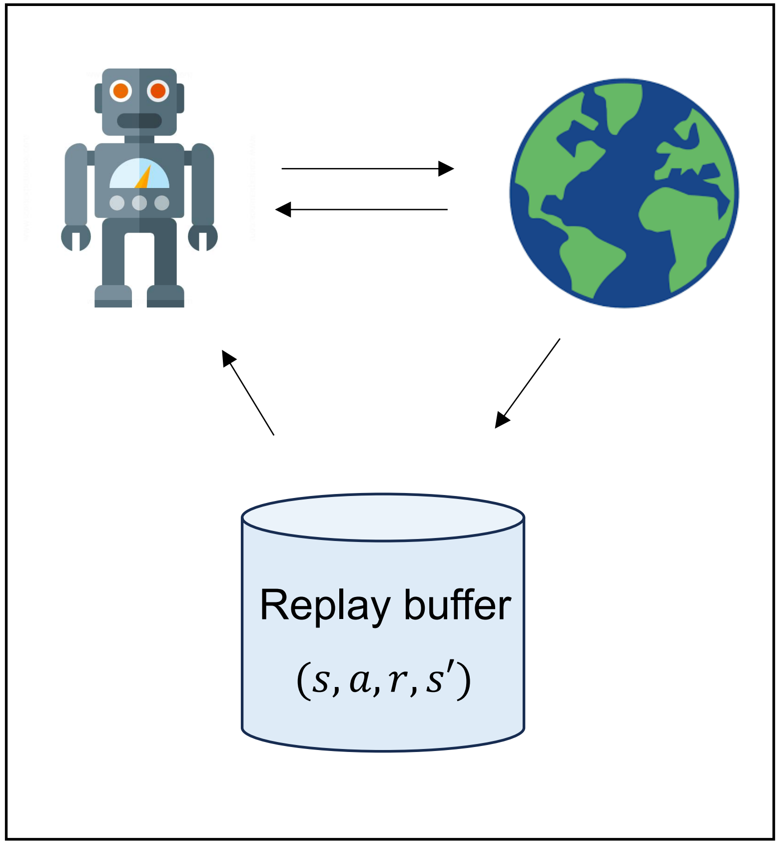
Model-Based RL



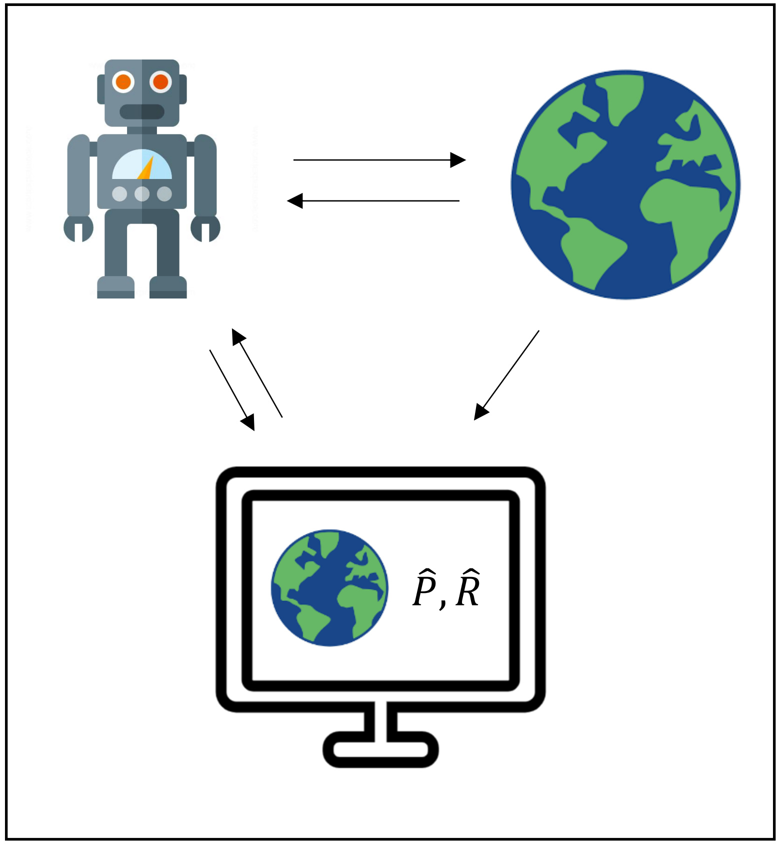
Comparison between training methods



Model-free On-policy



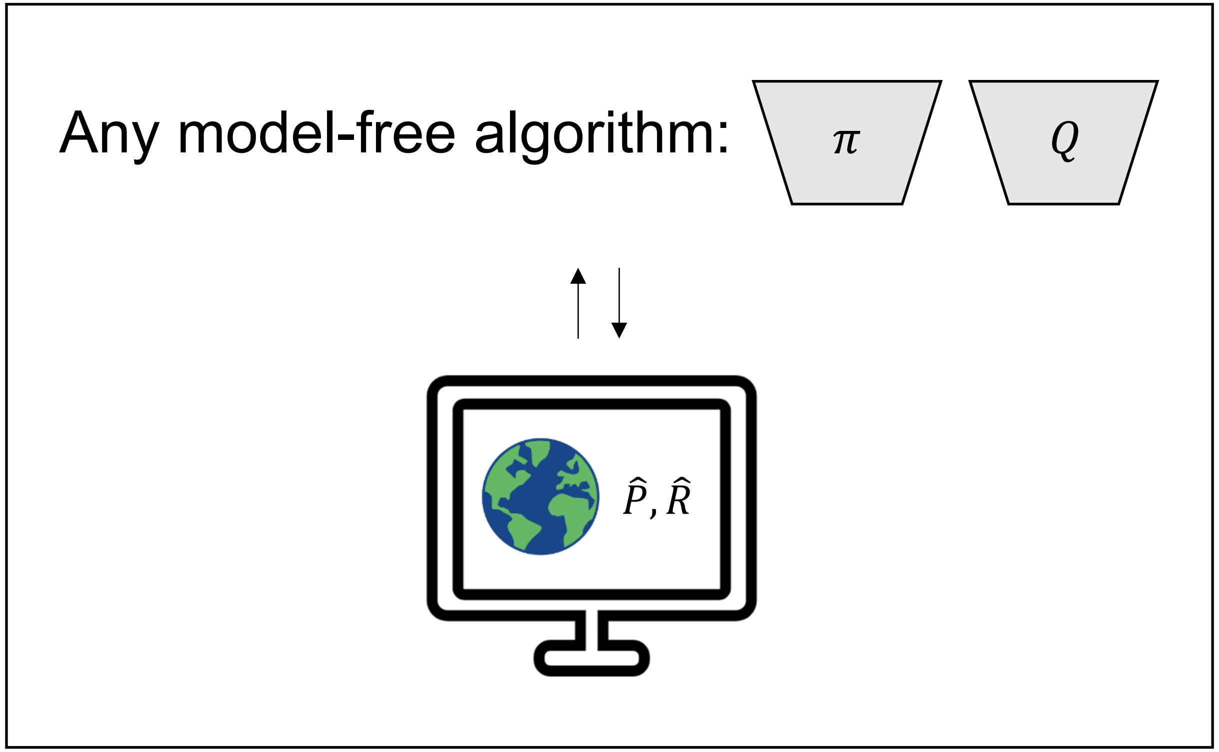
Model-free Off-policy



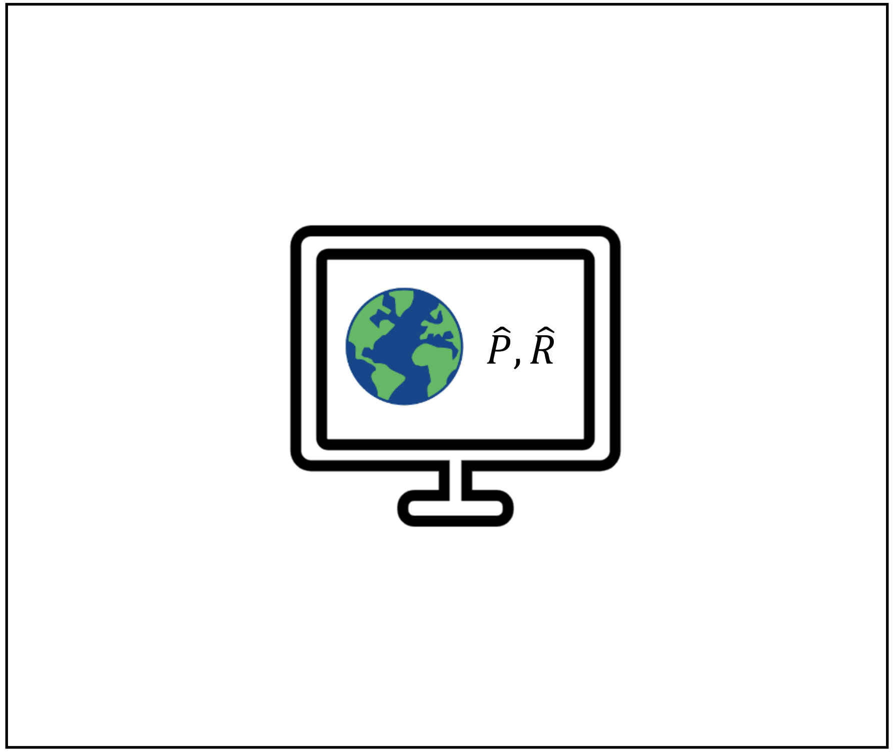
Model-based

Model-Based RL

Two ways to use the simulator / world model / model:



Model-assisted model-free learning



Planning with a model

1. Model-Assisted Model-Free Learning (Dyna-style)

For $k = 1,\ 2, \ldots$

For $i = 1, 2, \ldots, N$:

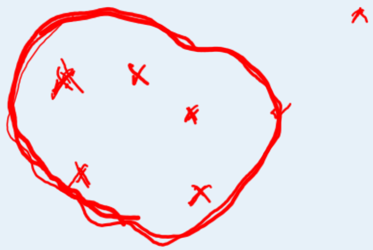
Choose action a_i with the current policy π_k

Receive reward $r_i \sim R(s_i, a_i)$ and $s_i' \sim P(\cdot\,|s_i, a_i)$

$s_{i+1} = s_i'$ if episode continues, $s_{i+1} \sim \rho$ if episode ends

Push (s_i, a_i, r_i, s_i') to $\mathcal{B}$

Data collection



Update model $\hat{P}, \hat{R}$ with data in $\mathcal{B}$

Model update

Repeat several times:

Sample $(s, a) \sim \mathcal{B}$, or sample $s \sim \mathcal{B}$ and $a \sim \pi_k(\cdot\,|s)$ or uniform

Let $r \sim \hat{R}(s, a)$ and $s' \sim \hat{P}(\cdot\,|s, a)$

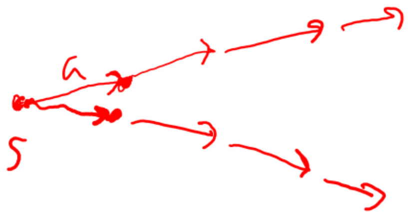
Update policy / value with sample (s, a, r, s')

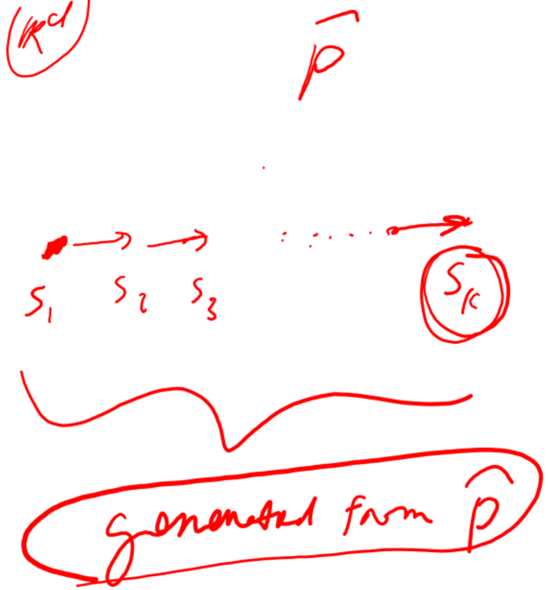
Policy / value update

Lecture 12, Deep Reinforcement Learning by Sergey Levine

1. Model-Assisted Model-Free Learning (Dyna-style)

Why still sample s from the buffer? Why not generate s randomly or generate it from the trained model?





1. Model-Assisted Model-Free Learning (Dyna-style)

Some Dyna-style algorithms:

Gu et al., Continuous deep Q-Learning with model-based acceleration, 2016. (MBA)
Feinberg et al., Model-based value expansion, 2018. (MVE)
Janner et al., When to trust your model: model-based policy optimization. 2019. (MBPO)

The performance of MB-RL is heavily influenced by how to represent and train $\hat{P}$ efficiently, while making it predictive and scalable:

Hafner et al., Mastering Diverse Domains through World Models. 2023. (DreamerV3)

2. Planning with A Model

If we have a model / simulator, how to decide the next action without having a trained policy / value network?

Exact / closed-form solution: finite-state-finite-action, linear system

$$s_{t+1} = A s_t + B u_t + w_t$$

$\in \mathbb{R}^k$ noise

$\in \mathbb{R}^d$

$$\text{loss} = s_t^T Q s_t + a_t^T R a_t$$

2. Planning with A Model

If we have a model / simulator, how to decide the next action without having a trained policy / value network?

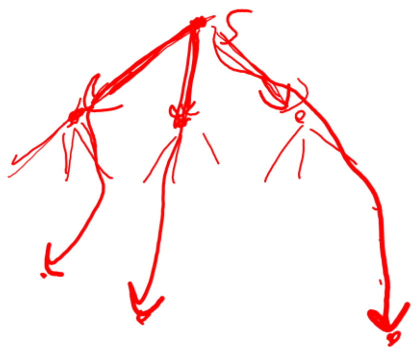
Search (for large state space without structure):

"Create the policy on the fly": decide $\pi(\cdot \mid s)$ only when reaching s

This is often used when we want to enhance a **default** policy on the fly.

Monte-Carlo Tree Search (MCTS)

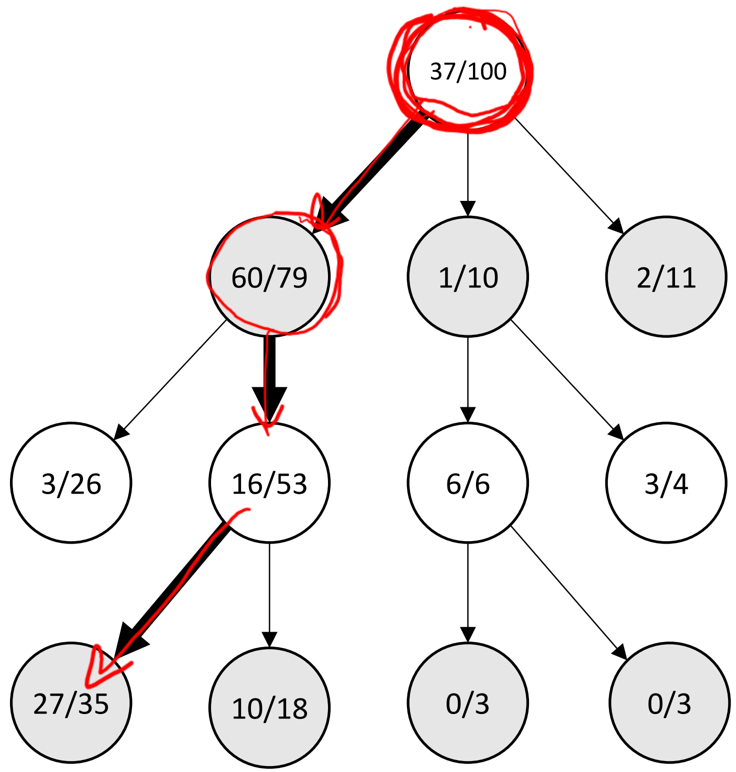
game of Go



Repeat while time remains

Selection → Expansion → Simulation → Backup

Monte-Carlo Tree Search (MCTS)



Selection

- Starting from the root node, execute **tree policy** until reaching a leaf node
- One effective tree policy is given by UCB1, which chooses an action based on

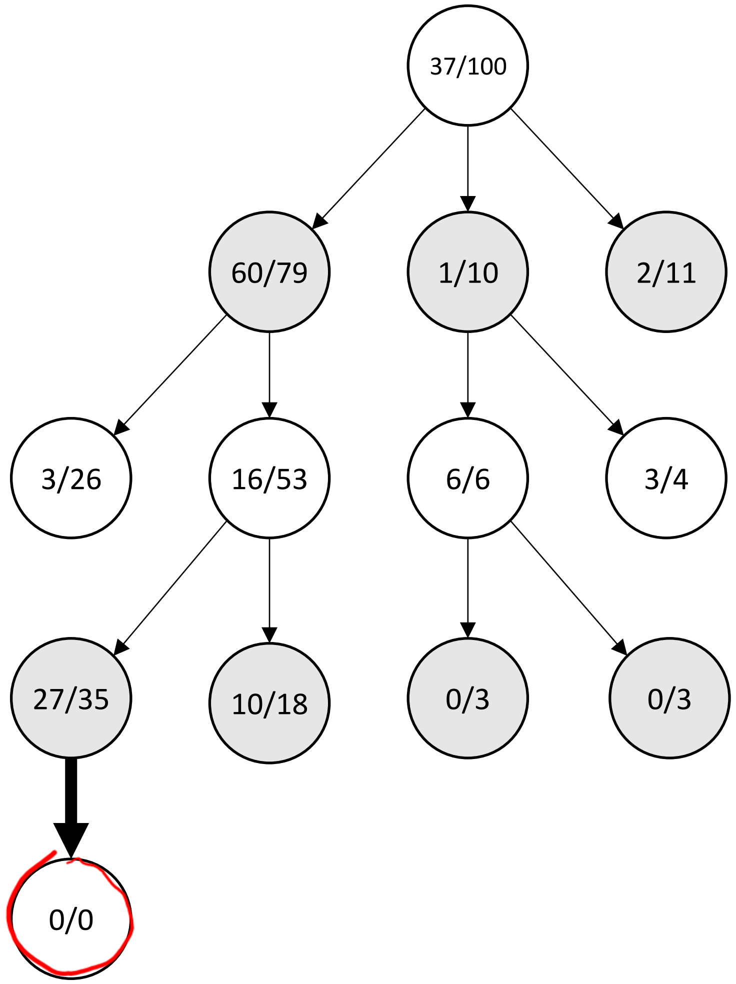
$$\frac{W(n)}{N(n)} + C \times \sqrt{\frac{\log N(\text{parent}(n))}{N(n)}}$$

$W(n)$: total #wins of all playouts that went through node n
$N(n)$: total #playouts that went through node n

Monte-Carlo Tree Search (MCTS)

Expand

- On some iterations, grow the search tree from selected leaf nodes by adding one or more child nodes

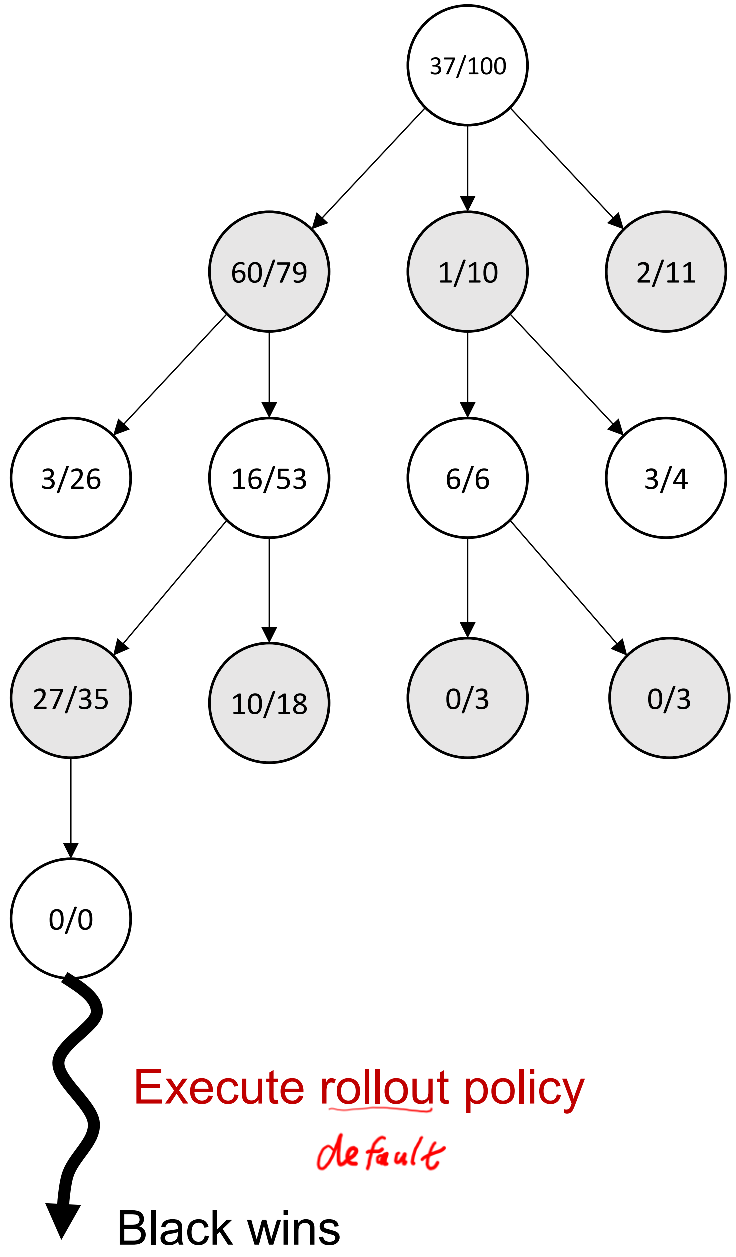


Monte-Carlo Tree Search (MCTS)

Simulation

- From the selected or expanded node (if any), execute the **rollout policy** to the end of the game

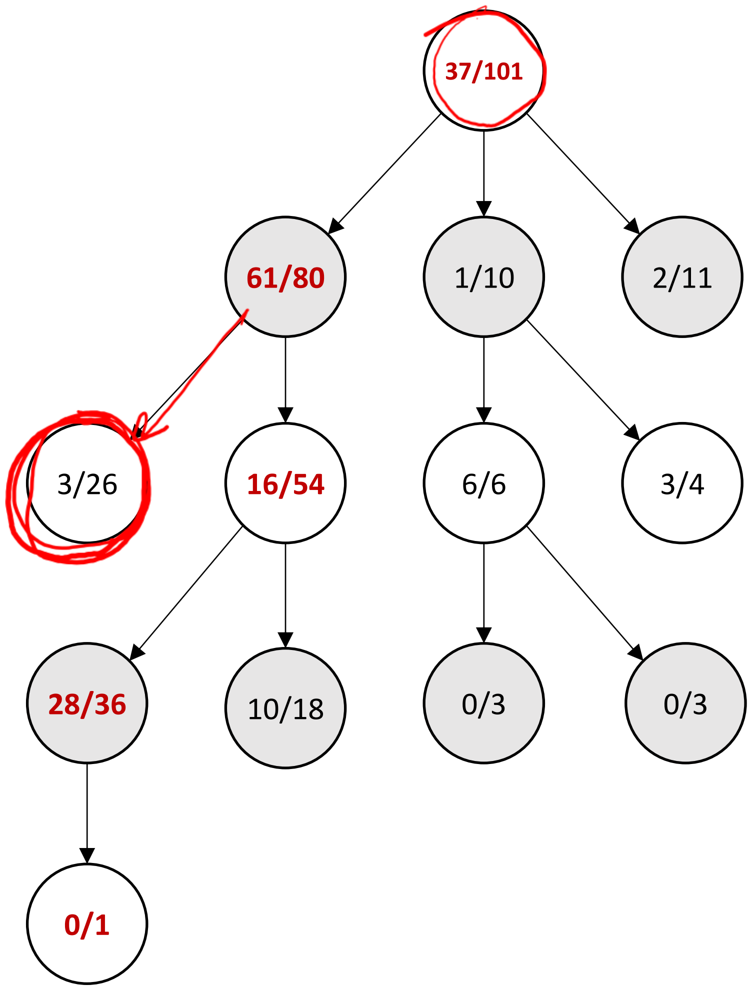
- Rollout policy
  - Could be heuristics, such as “consider capture moves” in chess
  - Could be learned through neural networks by self-play



Monte-Carlo Tree Search (MCTS)

Backup

- Update the #wins and #playouts on nodes along the tree policy



Monte-Carlo Tree Search (MCTS)

Finally,

- Choose the action from the root node that has the largest visit count.
- After the opponent’s move, start the same procedure from the new state (can keep the statistics from the previous state)

2. Planning with A Model

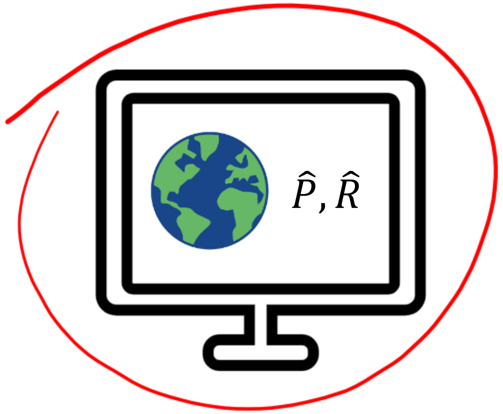
If we have a model / simulator, how to decide the next action without having a trained policy / value network?

Search (for large state space without structure):

"Create the policy on the fly": decide $\pi(\cdot\,|s)$ only when reaching s

This is often used when we want to enhance a **default** policy on the fly.

Plan for multiple steps, but execute only the first step.



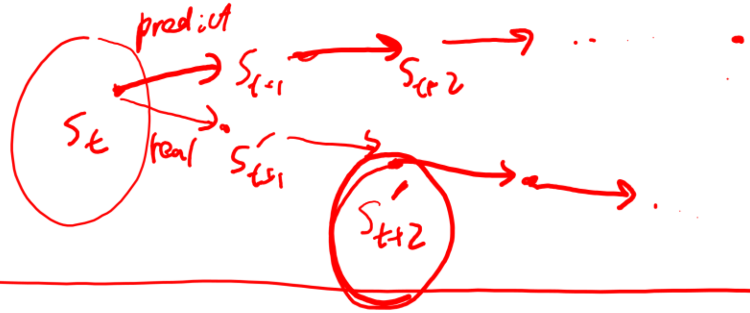
linear system $\boxed{s_{t+1} = A s_t + B a_t}$

$s_t \in \mathbb{R}^d, \; a_t \in \mathbb{R}^k$

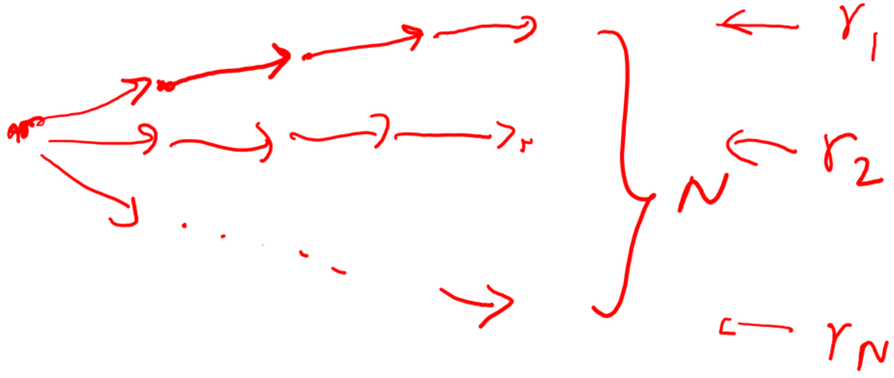
$$\underset{\mathbb{R}^d}{a} = \overbrace{\left[K^*(A,B)\right]}^{d \times k} \underset{\mathbb{R}^k}{s}$$

$s = (\theta, \omega)$

$a = \text{force}$



Best-of-N



$i = \arg\max_i r_i$

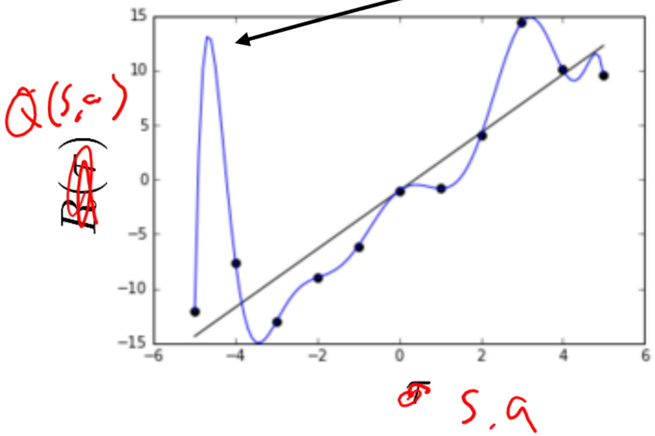
Uncertainty in Model-Based RL

Model Error / Failure of Generalization

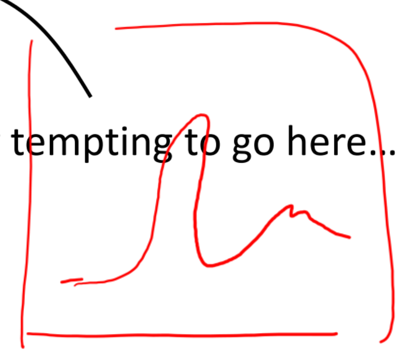
As the model is learned with only finite samples, they could have large errors in uncovered areas.

DDQN TD3.

We can encounter the same issue in off-policy methods like DQN, DDPG.



very tempting to go here...



Solution: train 2 target networks and avoid the $\max_a$ operator to exploit the error of a single network

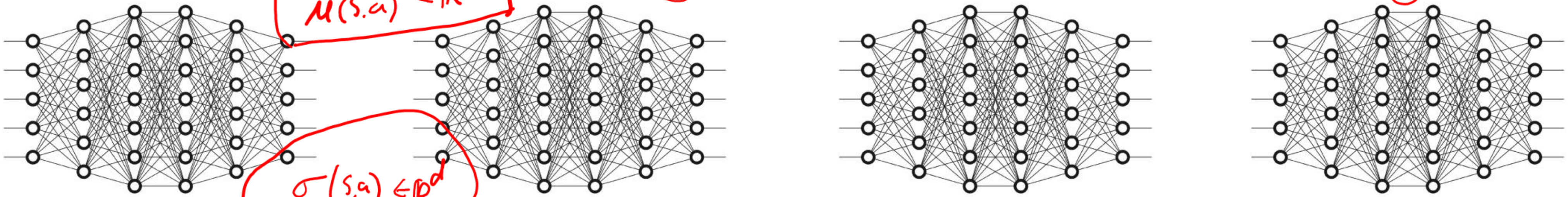
$\boxed{\max_a Q(s,a)}$

Ensemble Models

$\in \mathbb{R}^d$

s, a

$\mu(s,a) \in \mathbb{R}^d$



$\sigma(s,a) \in \mathbb{R}^d$

$P_{\theta_1}(s'|s,a)$ $\quad$ $P_{\theta_2}(s'|s,a)$ $\quad$ $P_{\theta_3}(s'|s,a)$ $\quad$ $P_{\theta_4}(s'|s,a)$

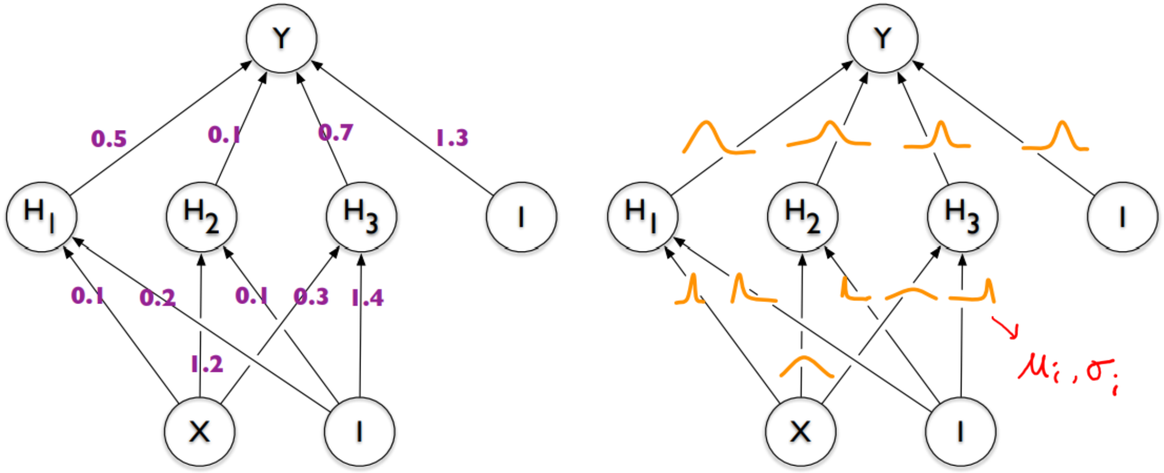
$\mathcal{N}\left(\mu_i(s,a), \sigma_i(s,a)\right)$

$$P(s'|s,a) \approx \frac{1}{N}\sum_{i=1}^{N} P_{\theta_i}(s'|s,a)$$

$$\neq \mathcal{N}\left(\frac{1}{N}\sum_{i=1}^{N}\mu_i(s,a), \frac{1}{N}\sum_{i=1}^{N}\sigma_i(s,a)\right)$$

Chua et al. Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models. 2018.
Buckman et al. Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion. 2018.

Bayesian Neural Network



expected weight　　uncertainty of the weight

Common approximation: $p(\theta) = \prod_i p(\theta_i)$ where $p(\theta_i) = \mathcal{N}(\mu_i, \sigma_i)$

Blundell et al., Weight Uncertainty in Neural Networks. 2015.
Gal et al., Concrete Dropout. 2017.

Aleatoric and Epistemic Uncertainty

- **Aleatoric uncertainty**
  - Comes from inherent randomness or noise in the data (e.g., sensor noise, coin flips)
  - **Irreducible** — cannot be removed even with more data
- **Epistemic uncertainty**
  - Comes from lack of data or limited model capacity
  - **Reducible** — can shrink with more data or better models

The “model uncertainty” here refers to **Epistemic uncertainty**.