DS-GA 3001 001 | Lecture 8

Reinforcement Learning

Jeremy Curuksu, PhD
NYU Center for Data Science
jeremy.cur@nyu.edu

April 9, 2025

DS-GA 3001 RL Curriculum

Reinforcement Learning:

- Introduction to Reinforcement Learning
- Multi-armed Bandit
- Dynamic Programming on Markov Decision Process
- Model-free Reinforcement Learning
- Value Function Approximation (Deep RL)
- Examples of Industrial Applications
- Policy Function Approximation (Actor-Critic)
- **Planning from a Model of the Environment (AlphaZero)**
- Aligning AI systems to Human Preferences (ChatGPT)
- Advanced Topics

Reinforcement Learning

Last week: Learning a Policy Function

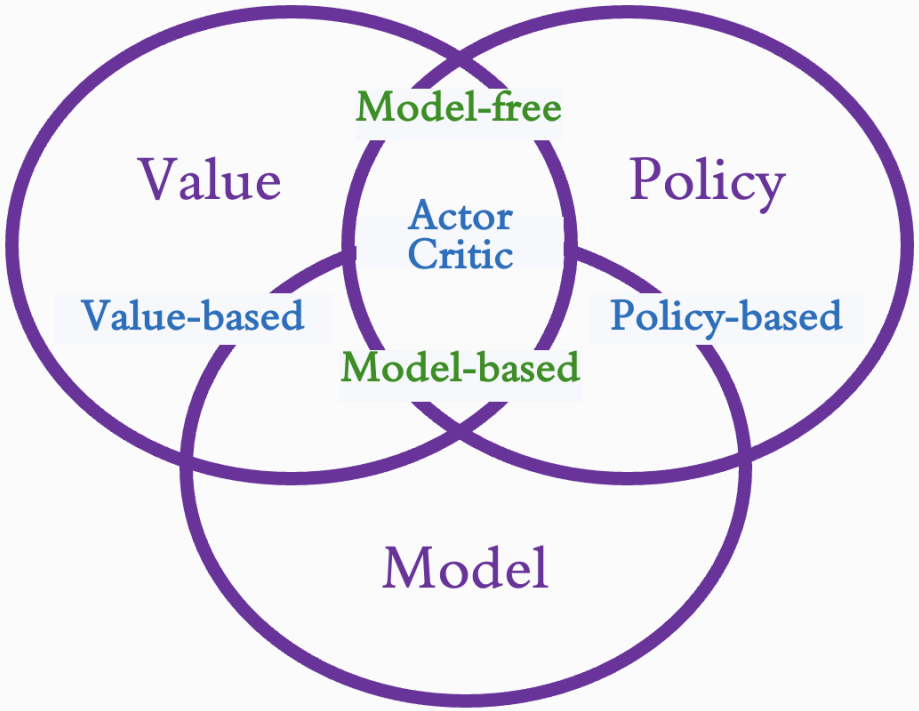
- Policy Gradient Reinforcement Learning
- Advanced Sampling of Policy Gradient (Actor-Critic)
- RL for Continuous Action Space (skipped in 2025)

Today: Planning a Policy from a Model

- **Learning and planning from a MDP model**
- **Reinforcement Learning with a local MDP model**
- **Case studies: AlphaGo, AlphaZero**

Learning and planning from a MDP model

Model-based RL



Categories of RL agents:

- **Model-based**: Use a model to learn policy and/or values
- **Model-free**: Learn policy and/or values without model
- **Value-based**: Learn value function, not policy function
- **Policy-based**: Learn policy function, not value function
- **Actor-Critic**: Learn policy and value functions

Model-, Value-, or Policy-based RL ?

- **Model-based RL:**
  - ✓ Learns 'all there is to know' from the data
  - ✓ Very well understood method (supervised learning)
  - × Objective captures irrelevant information
  - × May focus compute/capacity on irrelevant details
  - × Deriving a policy from a model (planning) can be hard

- **Challenges and Opportunities:**
  - How best to approximate a MDP model from experience?
  - How best to simulate experience from a model?
  - Can we combine learning policy and/or value functions with planning from a model? Should we?
  - Is it best to use simulated, real experience, or both? Why?
  - Can we focus a model on what matters most? How?

What is a RL Model?

What is a RL Model?

Online Reinforcement Learning

Learn by interacting with an environment (trial & error)

Offline Reinforcement Learning

Simulate the environment using a model

What is a RL Model?

A RL model represents the environment dynamics

- **A Bandit (single-state) model is a reward function:**

$$p(r \mid a) = p(r_{t+1} = r \mid a_t = a) \iff r(a) = \mathbb{E}(r \mid a)$$

- **A Generalized RL model predicts the next state and reward:**

$$p(s', r \mid s, a) = p(s_{t+1} = s', r_{t+1} = r \mid s_t = s, a_t = a)$$

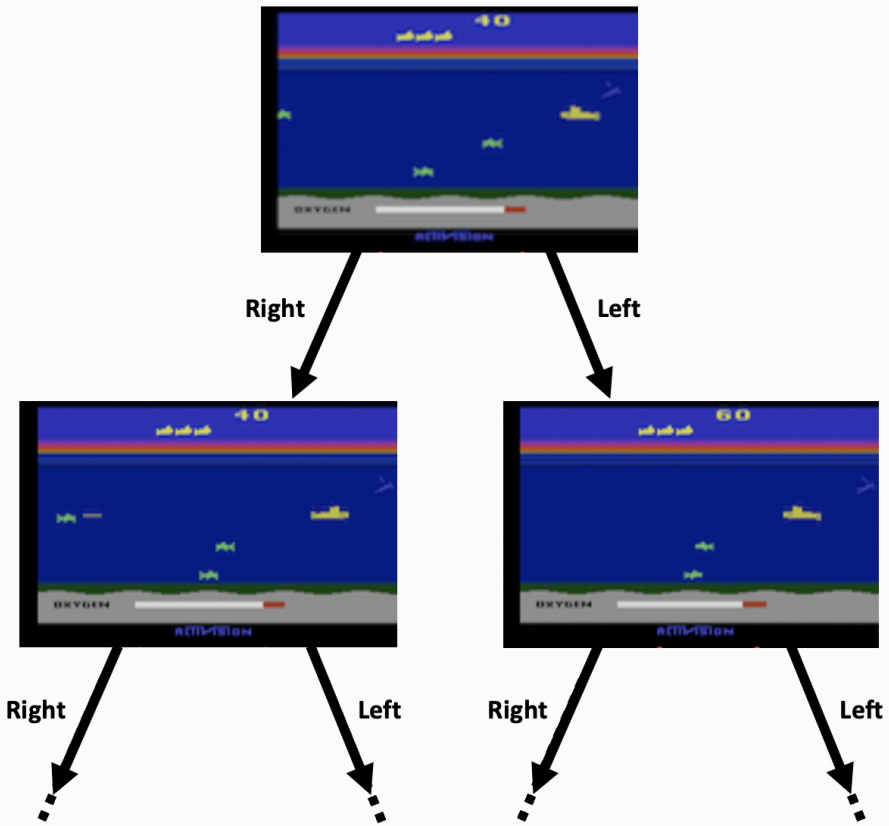
  It is also called a Markov Decision Process' transition function. The model itself does not predict actions. But using this model, an RL agent can reason and take actions.

- **The dynamics of reward vs. state can be modelled separately**

Model of Environment Dynamics

Example of model:

- Querying a video game console is an example of "perfect model" (rules of the game are perfectly defined by the console)



Learning a Model from Experience

Categories of RL models

1. Table Lookup models
2. Parametric Expectation models
3. Parametric Generative models
4. Non-Parametric models

Learning a Model from Experience

Table Lookup RL model

- Explicit MDP model approximated by counting visits to each state action pair (s, a) stored individually in a lookup table:

$$\forall (s,a): \quad \hat{p}_t(s' \mid s,a) = \frac{\sum_{k=0}^{t-1} \mathcal{I}(s_k = s, a_k = a, s_{k+1} = s')}{\sum_{k=0}^{t-1} \mathcal{I}(s_k = s, a_k = a)}$$

$$\hat{r}_t(s,a) = \frac{\sum_{k=0}^{t-1} [\mathcal{I}(s_k = s, a_k = a)\, r_{k+1}]}{\sum_{k=0}^{t-1} \mathcal{I}(s_k = s, a_k = a)}$$

- Feasible only for relatively small state-action space
- For large MDP, a parametric model can be learned instead

Learning a Model from Experience

Parametric RL models (supervised learning)

- **Expectation model** = Predictive model based on linear approximation of state and reward distributions:

  Learn function $f_\eta : \; s, a \longmapsto r, s'$

  such that $$\eta \sim \arg\min_{\eta} \mathbb{E}_{s,a} \left[\left((\hat{r}, \hat{s'}) - (r, s') \right)^2 \right]$$

- **Expectation models are often used to predict rewards.**

Learning a Model from Experience

Parametric RL models (supervised learning)

- **Generative (Stochastic) model** = Predictive model that directly approximates state and reward probability distributions:

  Learn function $g_\eta : \; s, a \longmapsto p(r),\; p(s')$

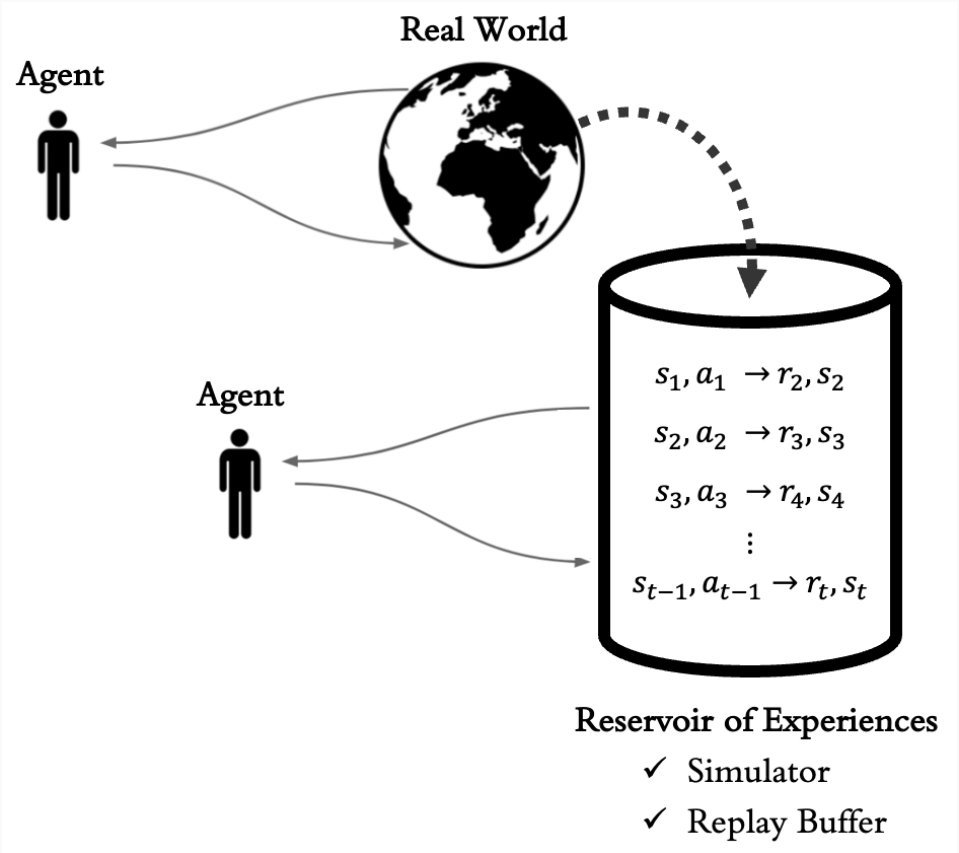
  such that $\eta \sim \arg\min_{\eta} \mathbb{E}_{s,a} \left[-\sum p\left(r, s'\right) log\left(\hat{p}\left(r, s'\right)\right) \right]$

- **Generative models can be queried to sample new experience**, but predicting the probability of each (r, s') requires more data and compute relative to predicting their expected values.
- **Generative models are often used to predict states.**

Learning a Model from Experience

Non-parametric RL model

- Any other form of experience storage (replay buffer, simulator) that provides access to information on the environment



Planning from a RL Model

Planning uses a model as input and produces or improves a policy for interacting with the modelled environment:

- Explicit models of the environment can be used directly to plan

- Simulators can be queried to sample experience. Then, any of the model-free RL methods seen in this course can be used to plan from the simulated experience

Practice: MDP Planning Algorithm

MDP Planning selects an action and evaluates a value and/or policy function at every step by querying a model of the environment

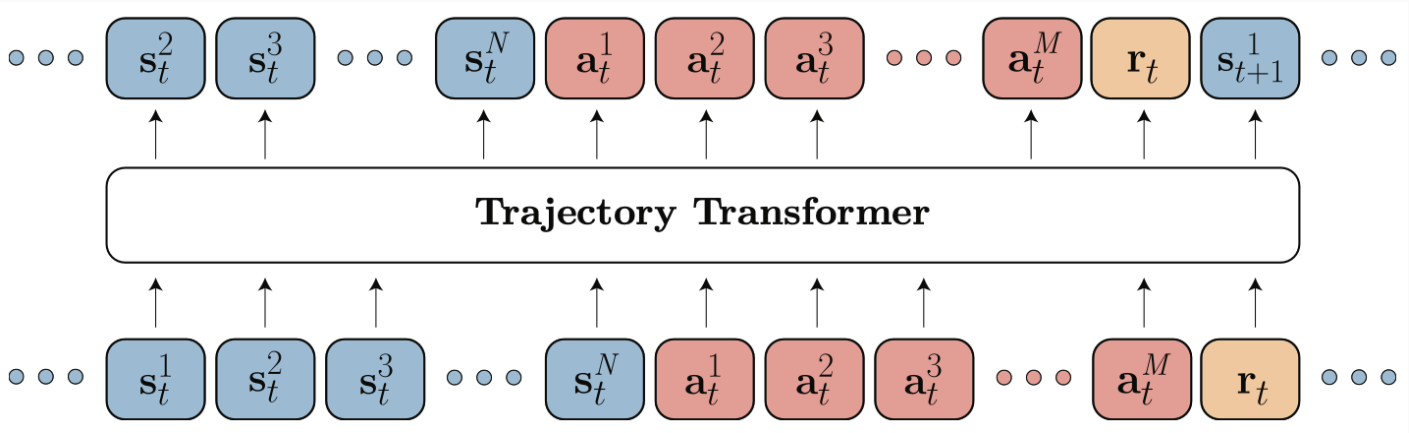
Initialize model $\mathcal{M}$, value function v, and policy function π, arbitrarily

Loop forever:

- Initialize s
- Select a from s following $\pi(s)$
- Send (s, a) to the model $\mathcal{M}$ and receive (r, s') from the model $\mathcal{M}$
- Update v for π at s
- Update π given value function v
- $s = s'$

Quantization of state-action space

Trajectory Transformer (Janner et al., 2021)



- Given a trajectory of length T: $\left(\vec{s_1}, \vec{a_1}, r_2, \vec{s_2}, \vec{a_2}, r_3, \ldots, \vec{s_T}, \vec{a_T}, r_T\right)$ where $\vec{s} \in \mathbb{R}^N$, $\vec{a} \in \mathbb{R}^M$, and $r \in \mathbb{R}$, define:

  $$\tau = \left(\ldots, s_t^1, s_t^2, \ldots, s_t^N, a_t^1, a_t^2, \ldots, a_t^M, r_t, \ldots\right) \qquad t = 1, \ldots, T$$

- s_t^i is the i^{th} dimension of the state at time t.
- a_t^j is the j^{th} dimension of the action at time t.
- τ is a trajectory of length $T(N + M + 1)$.

Quantization of state-action space

Trajectory Transformer (Janner et al., 2021)

- Pre-train TT by supervised learning maximizing this objective:

$$\sum_{t=1}^{T}\left(\sum_{i=1}^{N}\log p_\eta\left(s_t^i \mid \vec{s}_t^{<i}, \tau_{<t}\right)+\sum_{j=1}^{M}\log p_\eta\left(a_t^j \mid \vec{a}_t^{<j}, \vec{s}_t, \tau_{<t}\right)+\log p_\eta\left(r_t \mid \vec{a}_t, \vec{s}_t, \tau_{<t}\right)\right)$$

Dataset	Environment	BC	MBOP	BRAC	CQL	DT	TT (uniform)	TT (quantile)
Med-Expert	HalfCheetah	59.9	105.9	41.9	91.6	86.8	40.8 ±2.3	95.0 ±0.2
Med-Expert	Hopper	79.6	55.1	0.9	105.4	107.6	106.0 ±0.28	110.0 ±2.7
Med-Expert	Walker2d	36.6	70.2	81.6	108.8	108.1	91.0 ±2.8	101.9 ±6.8
Medium	HalfCheetah	43.1	44.6	46.3	44.0	42.6	44.0 ±0.31	46.9 ±0.4
Medium	Hopper	63.9	48.8	31.3	58.5	67.6	67.4 ±2.9	61.1 ±3.6
Medium	Walker2d	77.3	41.0	81.1	72.5	74.0	81.3 ±2.1	79.0 ±2.8
Med-Replay	HalfCheetah	4.3	42.3	47.7	45.5	36.6	44.1 ±0.9	41.9 ±2.5
Med-Replay	Hopper	27.6	12.4	0.6	95.0	82.7	99.4 ±3.2	91.5 ±3.6
Med-Replay	Walker2d	36.9	9.7	0.9	77.2	66.6	79.4 ±3.3	82.6 ±6.9
Average		47.7	47.8	36.9	77.6	74.7	72.6	78.9

- Offline RL with TT performs better than with other models

Planning from Simulated Experience *vs.* Learning from Real Experience

Limits of Planning with an Inaccurate Model

- Model-based RL is only as good as the estimated model
- Approx model may have biases or focus on irrelevant details
- Approximate a policy from a model $=$ *"estimate from estimate"*
- **One solution: Learn both from real-world and simulations**
  - **Data source 1:** Real Experience (unbiased, scarce)

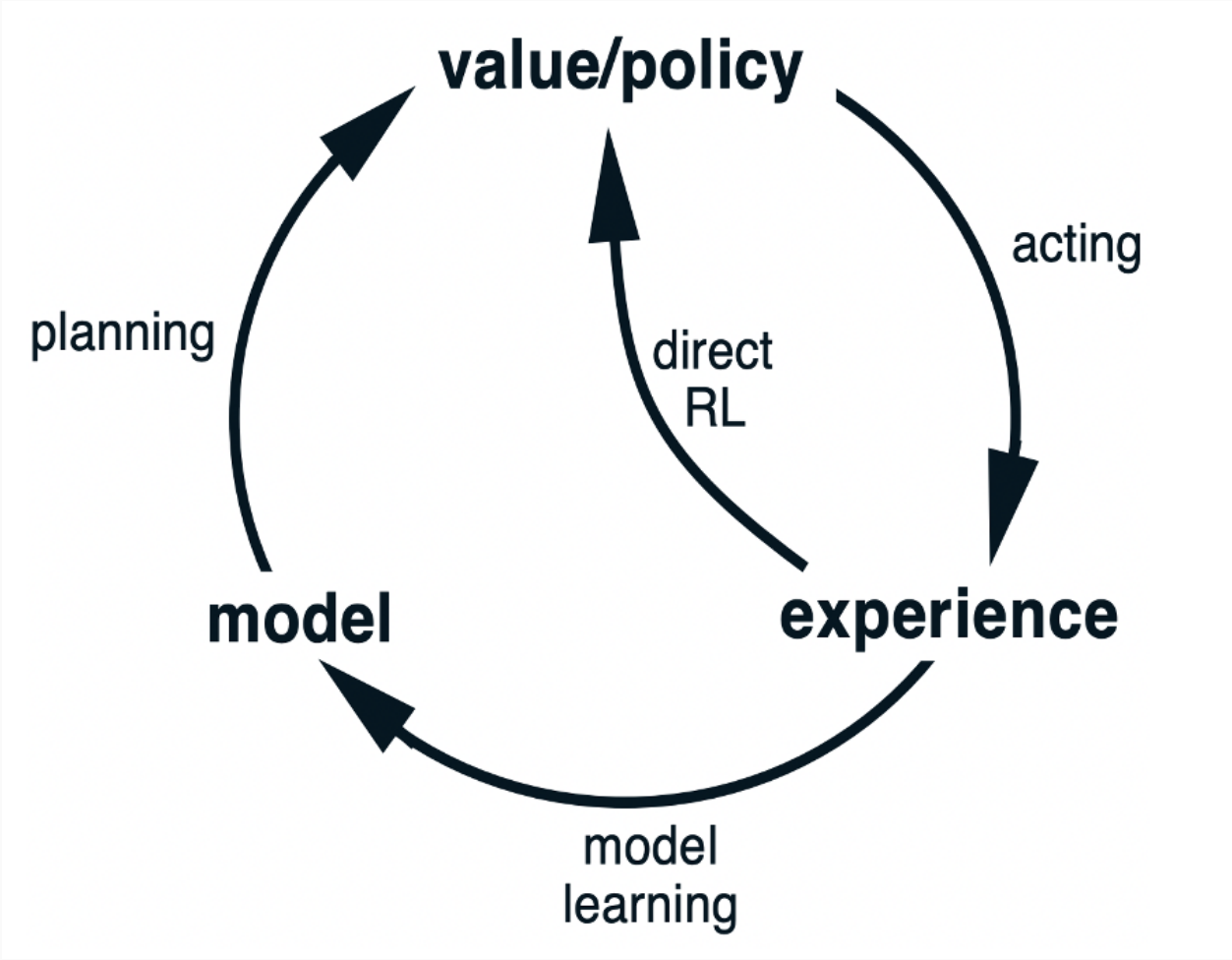
$$r, s' \sim p_{\text{unbiased}}$$

  - **Data source 2:** Simulated Experience (biased, abundant)

$$r, s' \sim \hat{p}_{\eta}$$

Integrated Learning and Planning

Model-based RL + Model-free RL

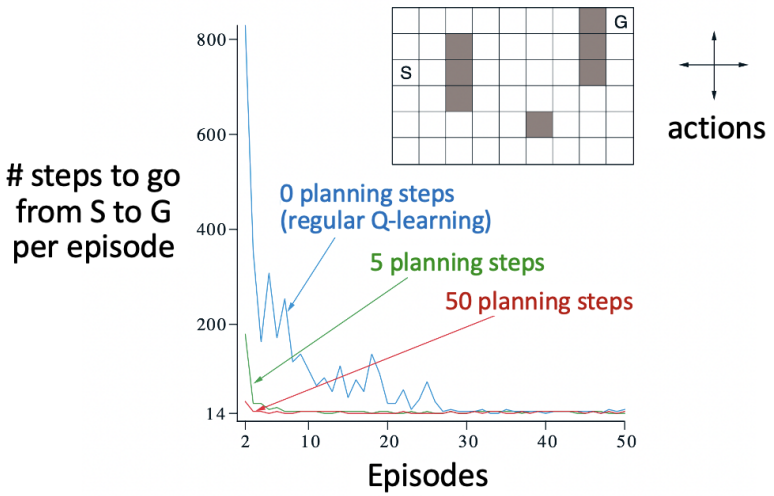


Sutton, Barto (1997)

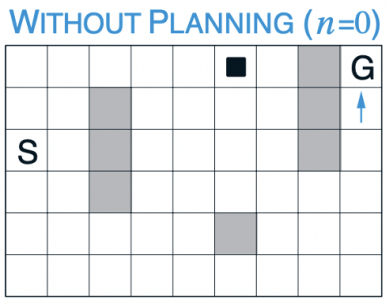
Integrated Learning and Planning

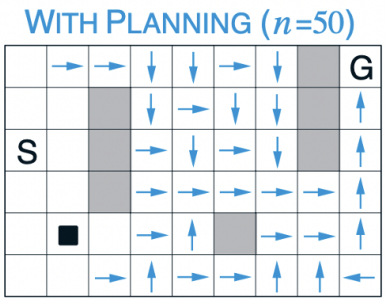
Dyna: Integrated Model-based and Model-free RL

- Learn values/policy from real experience
- Learn a model from real experience
- Plan values/policy from simulated experience
- Treat real and simulated experience equivalently



$r = -1$
in every state except G





Sutton, Barto (1997)

Practice: Dyna-Q Algorithm

Dyna-Q refines *q(s,a)* at every step by Q-learning, and also refines *q(s,a)* by querying a model of the environment

Initialize $\mathcal{M}$, $q(s,a)$, and $\pi(s)$, arbitrarily

Loop forever:

Initialize s

Select and take a from s following $\pi(s)$, observe r and s'

$$q(s,a) = q(s,a) + \alpha \left(r + \gamma \max_{a'} q(s',a') - q(s,a) \right)$$

Update π at s given $q(s,a)$

$s = s'$

Update $\mathcal{M}$ with tuple (s,a,s',r)

Repeat *n* times:

Randomly select a pair (s,a) previously observed

Send (s,a) to $\mathcal{M}$ and receive (r,s') from $\mathcal{M}$

$$q(s,a) = q(s,a) + \alpha \left(r + \gamma \max_{a'} q(s',a') - q(s,a) \right)$$

Reinforcement Learning with a local MDP model

Planning at Decision Time

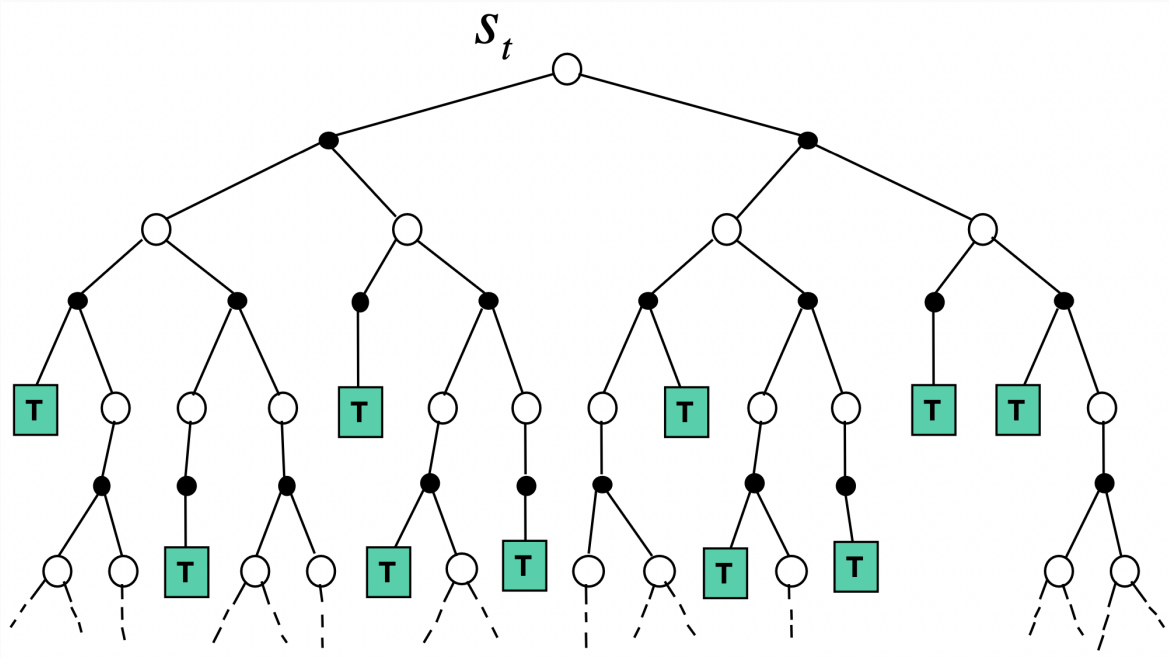
Learn a model starting from current state

- **Models can be defined *globally*** for entire MDPs to learn value and policy functions optimal for the entire state-action space
- **Models can also be defined *locally*** i.e., just for the near future, starting from the current state $=>$ *just to select the next action*
- **The distribution of states that may be encountered soon ("local" model) can often be approximated more accurately because it is often much smaller than a global model**

Forward Search Tree Model

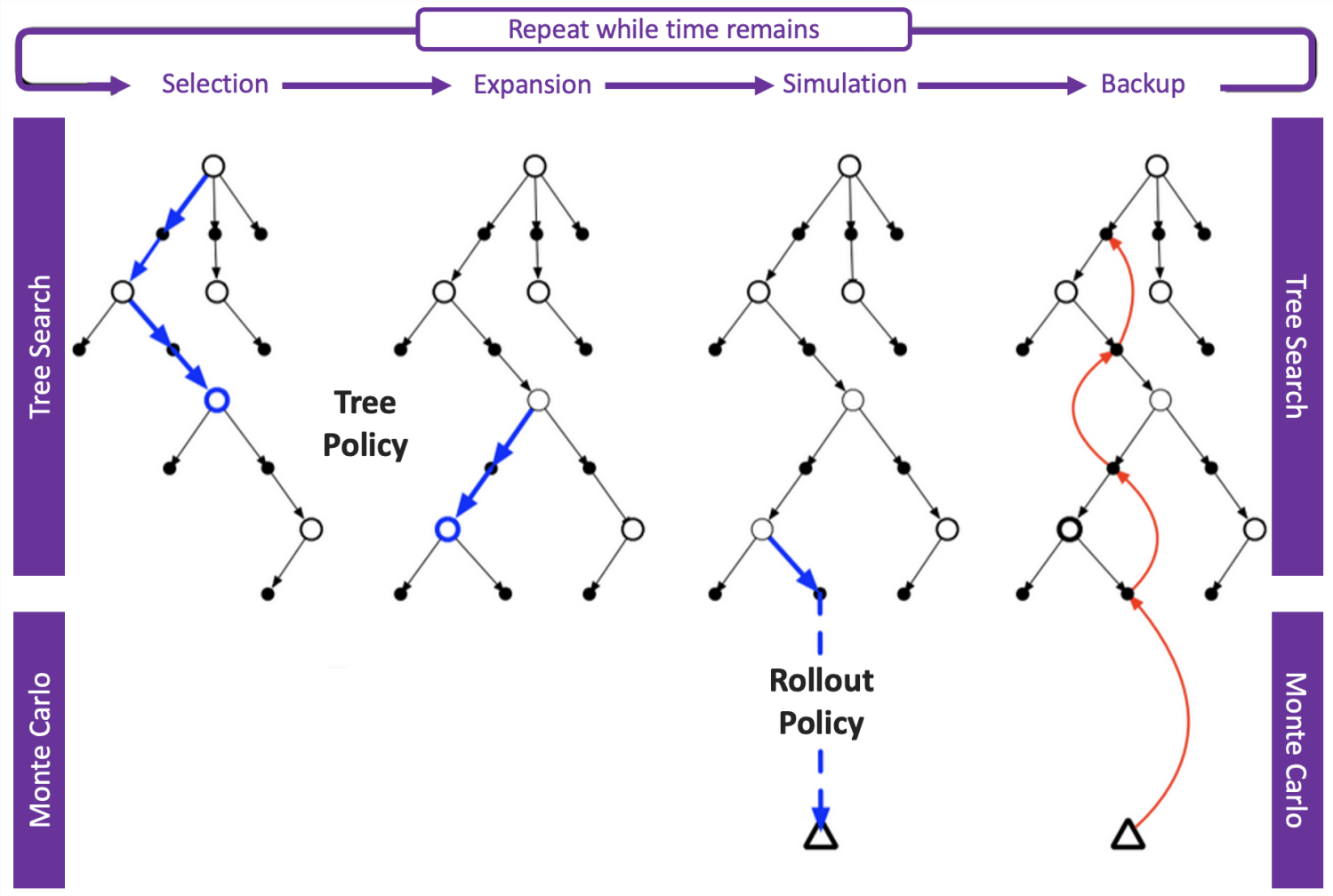
Partial instantiation of a lookup table

- **Forward search algorithms are local models** that look ahead at next possible states and actions
- **A forward search tree is a table lookup model** building a search tree with the current state s_t at the root



Monte Carlo Tree Search (MCTS)

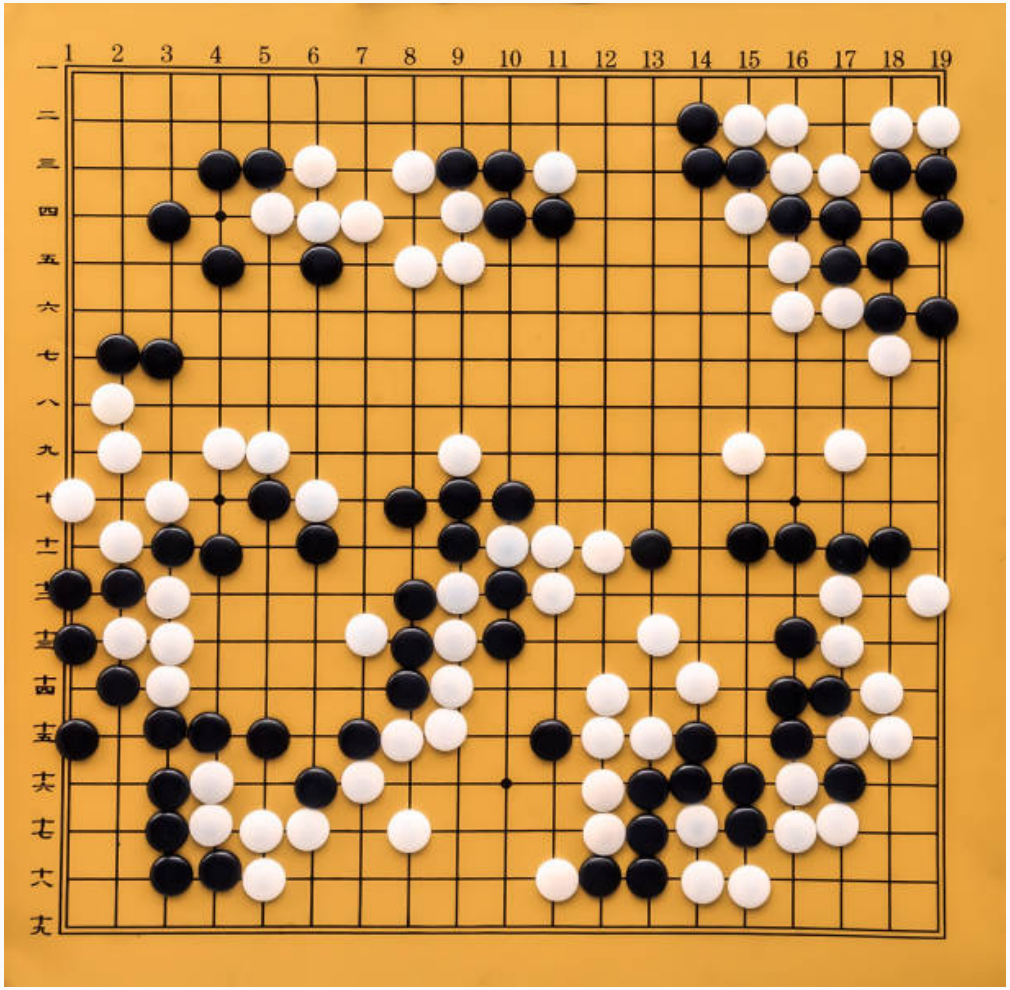
Plan locally + sample rest of trajectory, at decision time



Case studies: AlphaGo, AlphaZero

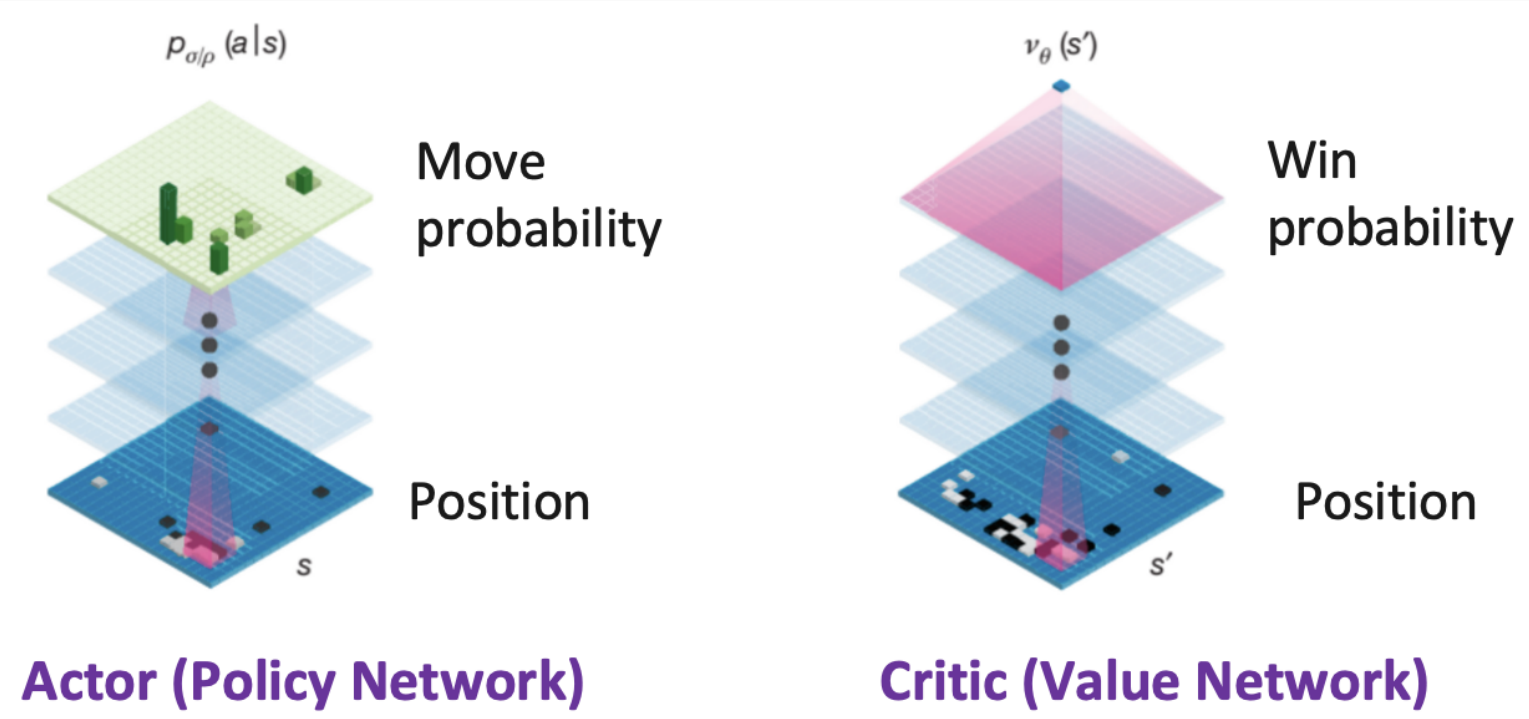
Case Study: Go

2-player board game, 10^{170} possible moves



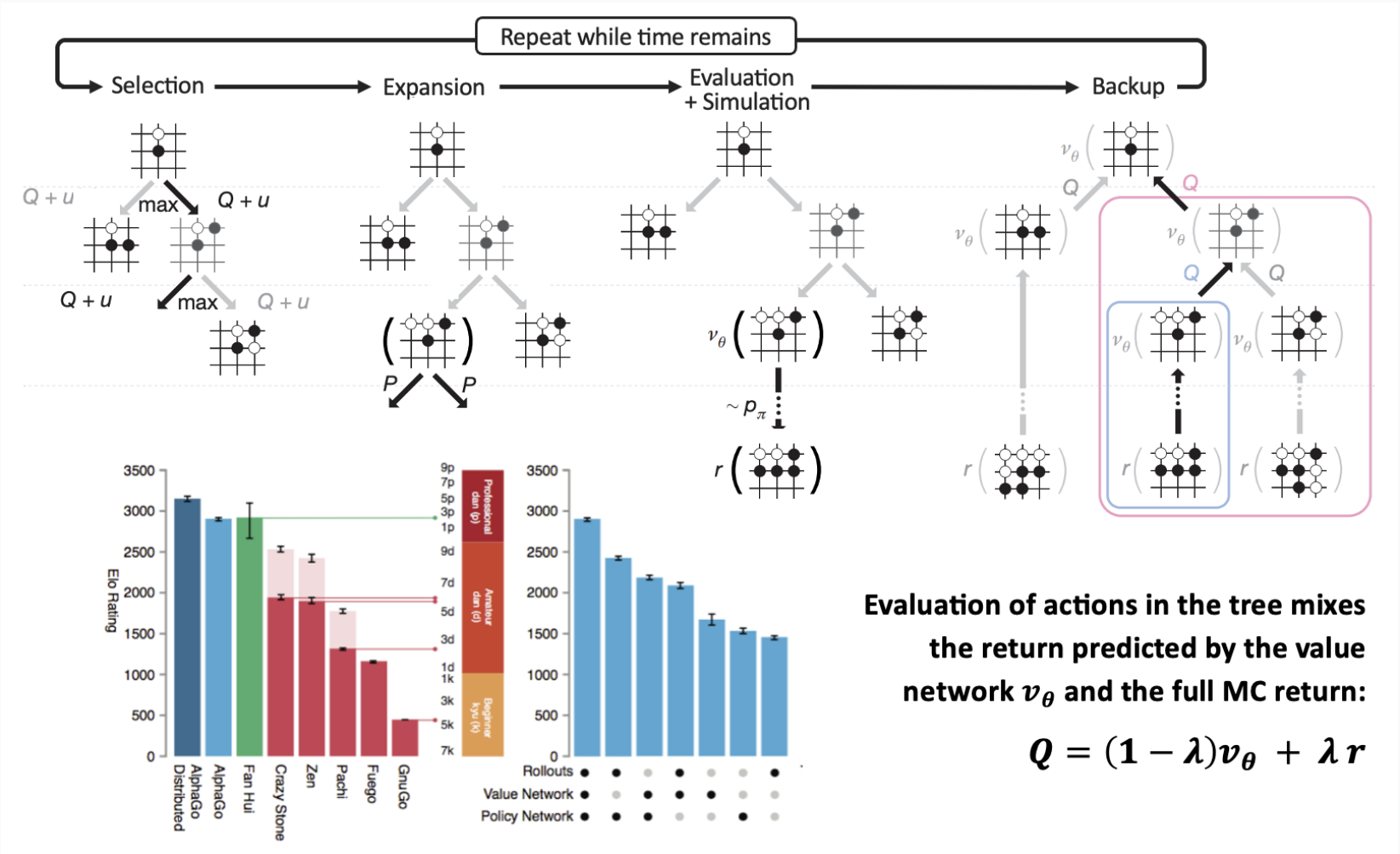
AlphaGo (Silver et al., 2016)

- **Actor (Policy Network)**: CNN initially trained by supervised learning on 30 millions human expert positions
- **Critic (Value Network)**: CNN initially trained by self-play using the actor to simulate both players



AlphaGo (Silver et al., 2016)

AlphaGo MCTS used both policy and value networks



Evaluation of actions in the tree mixes the return predicted by the value network v_θ and the full MC return:

$$Q = (1 - \lambda) v_\theta + \lambda r$$

Beyond human feedback: Superalignment

AlphaZero (Silver et al., 2017)

Use AlphaGo MCTS self-played games to parameterize a new generation of rollout policy and value network(s)

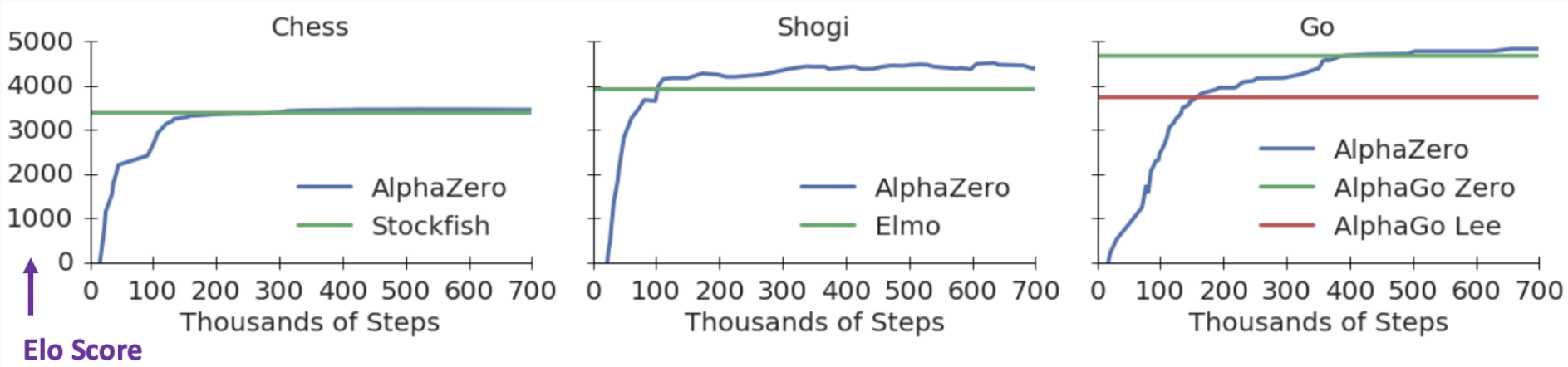
1. **Play self-games with MCTS** guided by policy & value networks (start with random parameters i.e, not pre-trained)

2. **Use the millions of games simulated by MCTS to train a new network** that predicts the policy (moves played by self-played MCTS) and the values (winner for any position)

   Repeat...

AlphaZero (Silver et al., 2017)

Learn AlphaGo generations that require only 1 network, from first principles... no human required



	Chess	Shogi	Go
Mini-batches	700k	700k	700k
Training Time	9h	12h	34h
Training Games	44 million	24 million	21 million
Thinking Time	800 sims 40 ms	800 sims 80 ms	800 sims 200 ms

Thank you!