DS-GA 3001 007 | Lecture 8

Reinforcement Learning

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

April 6, 2023

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)
- Policy Function Approximation (Actor-Critic)
- **Planning from a Model of the Environment**
- Examples of Industrial Applications
- Advanced Topics and Development Platforms

Reinforcement Learning

Last week: Learning a Policy Function

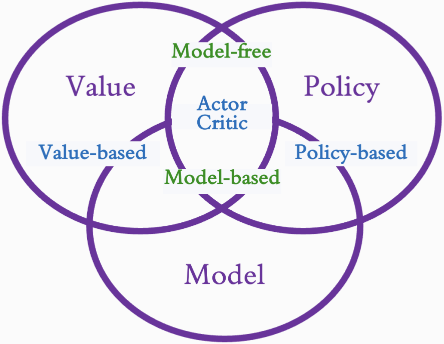
- Policy Gradient Reinforcement Learning
- Advanced Sampling of Policy Gradient
- RL for Continuous Action Space

Today: Planning a Policy from a Model

- **Learning a MDP model for planning**
- **Learning a local MDP model**
- **State-of-the-art RL: AlphaGo, AlphaZero**

Learning a MDP model for planning

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?

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:**

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

  MDP transition function does not infer policy (it does not *plan*)

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

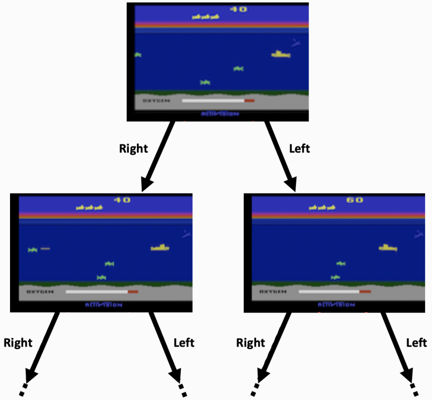
$$\text{E.g.,} \quad p\,(s' \mid s, a) = p\,(s_{t+1} = s' \mid s_t = s, a_t = a)$$

$$r(s, a) = \mathbb{E}(r \mid s_t = s, a_t = a)$$

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. Expectation models
3. Stochastic models
4. Non-Parametric (Experience Replay) 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

Expectation RL model

- **Predictive (supervised learning) parametric model** based on linear approximation of state and reward distributions:

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

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

from experience: $s_1, a_1 \rightarrow r_2, s_2 \,, \; \ldots \;, \; s_{T-1}, a_{T-1} \rightarrow r_T, s_T$

- **Scale to large MDP**: No need to store data for each (s, a)
- **Predicted states may not correspond to valid states** e.g., average of *left* and *right* actions is to go *straight*...
- **Commonly used to predict rewards**, not to predict states

Learning a Model from Experience

Stochastic (Generative) RL model

- **Predictive parametric model that directly approximates state and reward probability distributions:**

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

- **Generative models can be queried to sample new experience:**

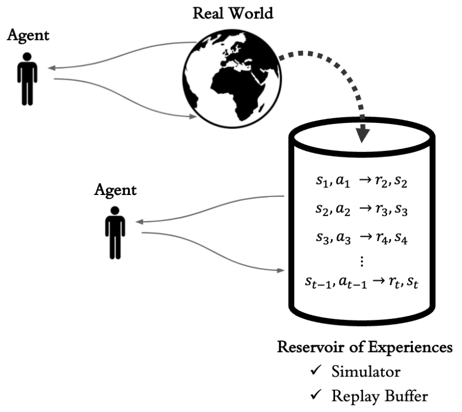
$$\text{Sample experience: } \hat{r}_{t+1}, \hat{s}_{t+1} \; = \; g_\eta(s, a) \, + \, \omega$$

- **Predicted states are generally valid states**
- **Predicting the probability of each (r, s') requires more data and compute** relative to predicting their expected values
- **Commonly used to sample states**, not to sample rewards

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 (randomly or following π)
- 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
- $s = s'$

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
- A model is an approximation thus may have biases
- Approx policy from approx model $=$ "*estimate from estimate*"
- **One solution: Learn both from real-world and simulations**
  - **Data source 1:** Real Experience (unbiased, scarce)

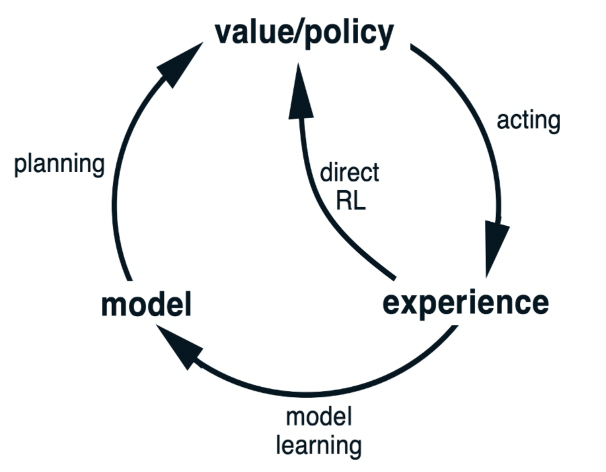
$$r, s' \sim p_{\text{unbiased}}$$

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

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

Integrated Learning and Planning

Model-based RL + Model-free RL

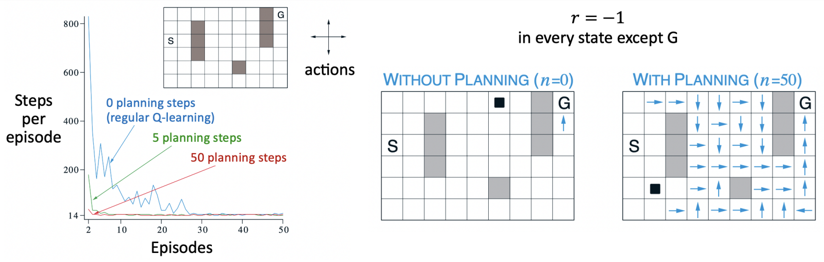


Sutton, Barto (1997)

Integrated Learning and Planning

Dyna: Integrated Model-based and Model-free RL

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



Sutton, Barto (1997)

Practice: Dyna-Q Algorithm

Dyna-Q computes *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 $\mathcal{M}$

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

Version using tabular Q-learning

Learning a local MDP model

Planning at Decision Time

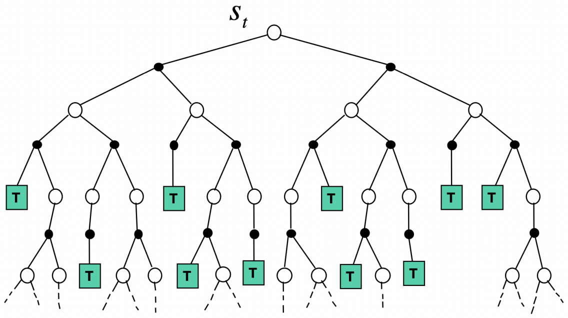
Learn a model starting from current state

- **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

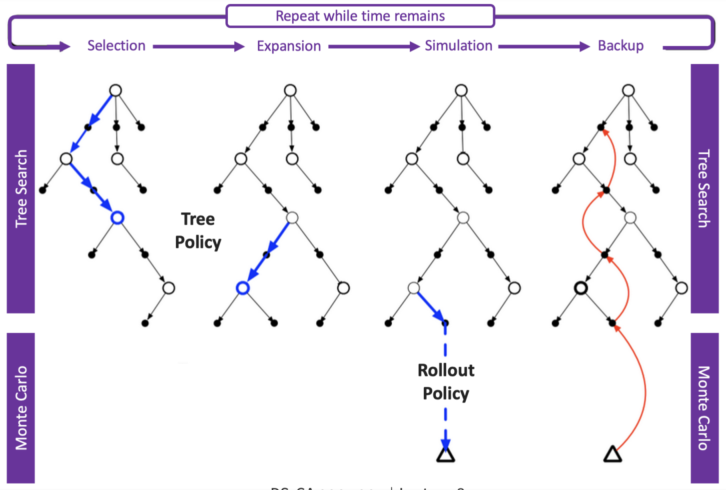
Partial instantiation of a lookup table

- **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)

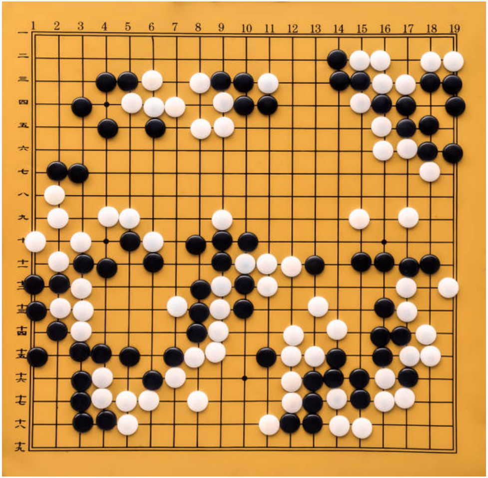
Plan locally + sample rest of trajectory, at decision time



State-of-the-art RL:
AlphaGo, AlphaZero

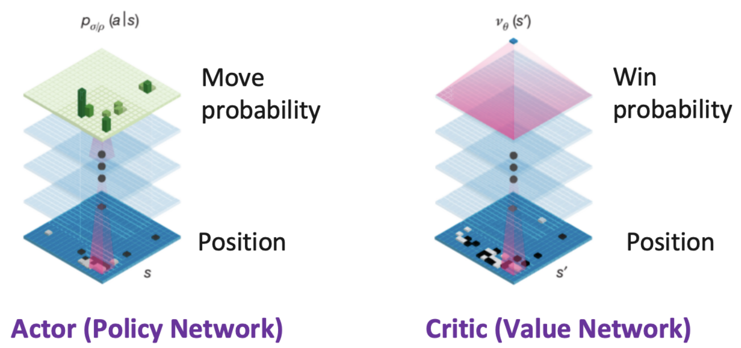
Case Study: Go

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



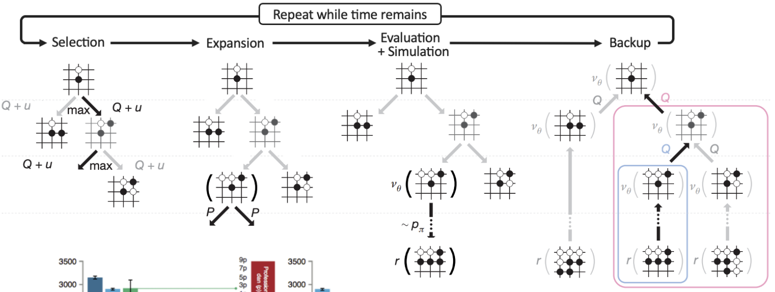
AlphaGo (Silver et al., 2016)

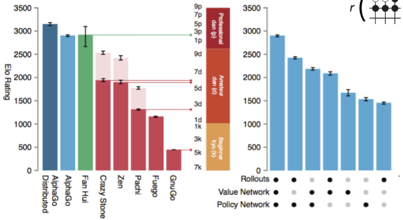
- **Actor (Policy Network)**: CNN initially trained by supervised learning on 30 millions human expert positions
- **Critic (Value Network)**: CNN initially trained by self-play using the actor to simulate both players



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:

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

AlphaZero (Silver et al., 2017)

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

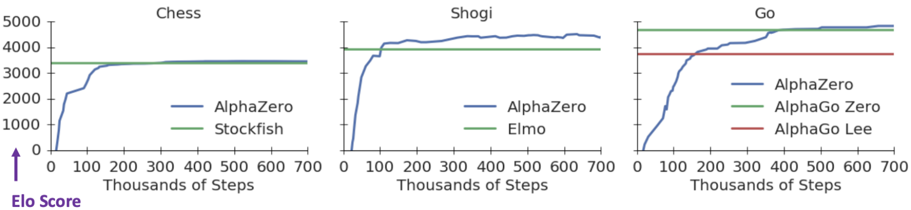
1. **Play self-games with MCTS** guided by policy & value networks (start with random parameters i.e, not pre-trained)

2. **Use the millions of games simulated by MCTS to train a new network** that predicts both 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	800 sims	800 sims
	40 ms	80 ms	200 ms

Thank you!