We learn → policy
We learn → value function
We learn → model

FROM EXPERIENCE

plan with the model to construct a value function or policy

Model-free ML

① No model

② learn value function / policy from experience

Model-based RL

① learn a model from experience

Or given a model

② plan value function / policy from model

Markov Decision Processes

⇑ Solve

Full Backups

Dynamic Programming

One-step updates
(bellman equations)

Exhaustive Search

Couple of steps
(Model + sample)

Sample Backups

Temporal Difference Learning

Monte Carlo

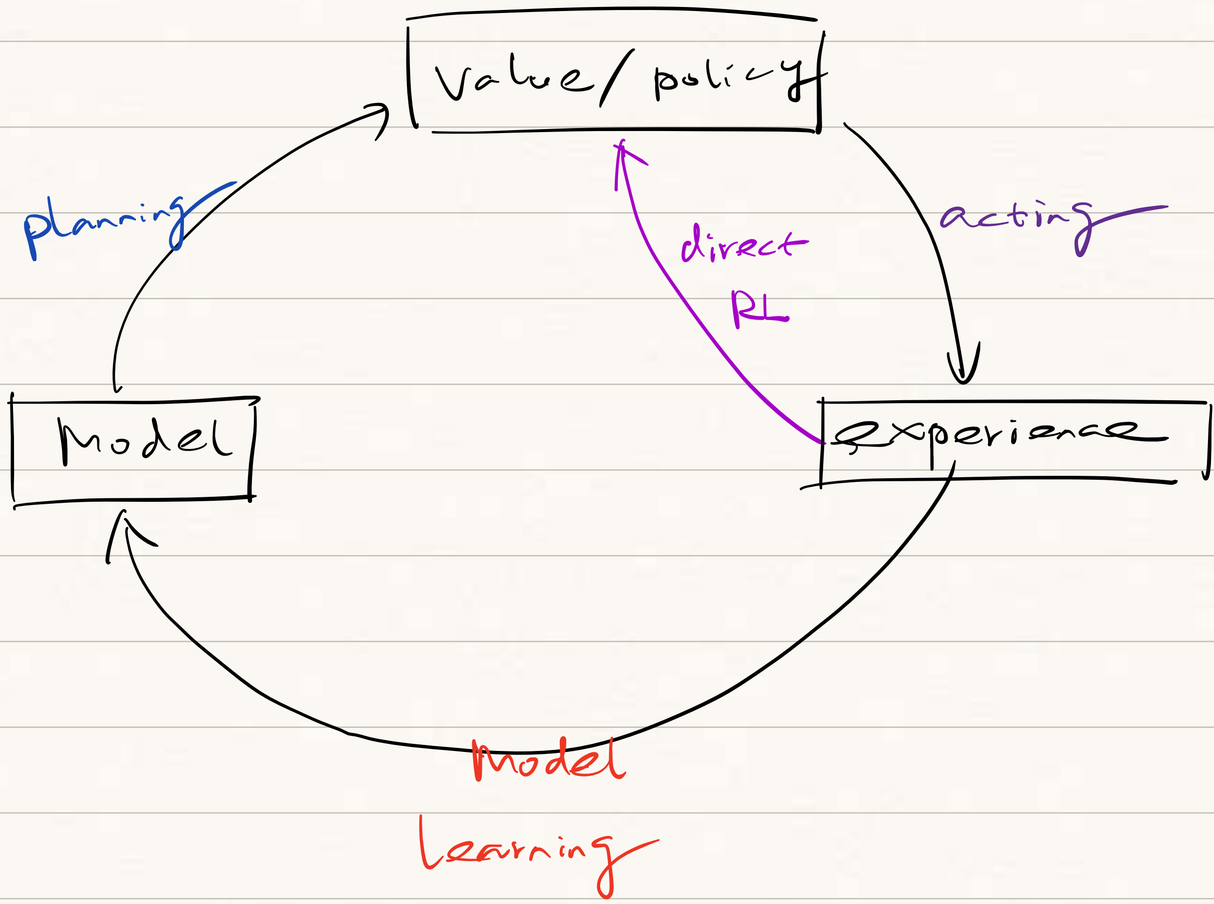
Shallow Backups

bootstrapping, λ

deep Backups

sample
Whole trajectory

Sample-based version of DP
(sample one step)


Value/policy
planning
acting
direct RL
Model
experience
Model learning

What is a Model ?

$\mathcal{M}_\eta$: an MDP $\langle S, A, \hat{P}_\eta \rangle$ reward

parameters: η

rewards : $\hat{P}_\eta \approx P$

State transitions :

$$R_{t+1}, S_{t+1} \sim \hat{P}_\eta(r, s' \mid S_t, A_t)$$

Model Learning

Goal: estimate $\mathcal{M}_\eta$ from experience

$$\{S_1, A_1, R_2, \ldots, S_T\}$$

$$S_1, A_1 \rightarrow R_2, S_2$$

$$\vdots \qquad \vdots$$

$$\boxed{S_{T-1}, A_{T-1} \rightarrow R_T, S_T}$$

Learn a function: $f(s, a) = r, s'$

Expectation Model

$$f(s,a) = r, s'$$

$$s' \approx \mathbb{E}[S_{t+1} \mid s = S_t,\ a = A_t]$$

With linear model and values:

$$P \qquad\qquad V_\theta(S_t) = \theta^T \phi_t$$

$$\mathbb{E}[\phi_{t+1}] = P\phi_t$$

then:

$$\mathbb{E}[V_\theta(S_{t+n}) \mid S_t = s]$$

$$= \mathbb{E}[\theta^T \phi_{t+n} \mid S_t = s]$$

$$= \mathbb{E}[\theta^T P \phi_{t+n-1} \mid S_t = s]$$

$$= \mathbb{E}[\theta^T P^n \phi_t \mid S_t = s]$$

$$= \theta^T P^n \phi(s)$$

$$= V_\theta(P^n \phi(s))$$

$$= V_\theta(\mathbb{E}[\varphi_{t+n} \mid S_t = s])$$

Stochastic Models

(generative models)

$$\hat{R}_{t+1}, \hat{S}_{t+1} = \hat{p}(S_t, A_t, w)$$

w: noise term

(can be chained)

Full models

branching : Model the complete transition dynamics, including stochasticity

$$\mathbb{E}[v(S_{t+1}) \mid S_t = s]$$

$$= \sum_a \pi(a \mid s) \sum_{s'} \hat{p}(s, a, s') (\hat{r}(s, a, s') + \gamma v(s'))$$

$$\mathbb{E}[V(S_{t+n} \mid S_t = s)$$

$$= \sum_a \pi(a|s) \sum_{s'} \hat{P}(s,a,s') \cdot \Big(\hat{r}(s,a,s') +$$

$$\gamma \sum_{a'} \pi(a'|s') \sum_{s''} \hat{P}(s',a',s'') \Big(\hat{r}(s',a',s'') +$$

$$\gamma^2 \sum_{a''} \pi(a''|s'') \sum_{s'''} \hat{P}(s'',a'',s''') \Big(\hat{r}(s'',a'',s''') +$$

$$\cdots \Big) \Big) \Big)$$

Models:

- Table Lookup
- Linear Expectation
- Linear Gaussian
- DNN

Table Lookup Model

$N(s,a)$

$$\hat{P}_t(s'|s,a) = \frac{1}{N(s,a)} \sum_{k=0}^{t-1} I(S_k = s, A_k = a, S_{k+1} = s')$$

$$\mathbb{E}_{\hat{P}_t}[R_{t+1} \mid S_t = s, A_t = a] = \frac{1}{N(s,a)} \sum_{k=0}^{t-1} I(S_k = s, A_k = a) R_{k+1}$$

Time t, $\langle S_t, A_t, R_{t+1}, S_{t+1} \rangle$

Planning with a Model

Model $\hat{P}_\eta$

Solve MDP : $\langle S, A, \hat{P}_\eta \rangle$

Algorithm : $\begin{cases} \text{Value iteration} \\ \text{policy iteration} \\ \text{tree search} \\ \dots \end{cases}$

Sample-based Planning

Model : generate samples

"Sample experience from model"

$$S, R \sim \hat{P}_\eta(\cdot \mid s, a)$$

Model-free RL to Sample :

$\begin{cases} \text{Monte-Carlo control} \\ \text{Sarsa} \\ \text{Q-learning} \end{cases}$

Idea: PLANNING.

1. Construct a lookup table model from real experience

2. Apply model-free RL to <u>sampled experience</u>

full-batch data

Batch Learning

learn

(Planning)

(Non-parameteric Model)

Conventional model-based and model-free methods

Traditional RL algorithms did not explicitly store their experiences, and were often placed into one of two groups.

- Model-free methods update the value function and/or policy and do not have explicit dynamics models.
- Model-based methods update the transition and reward models, and compute a value function or policy from the model.

say to learn a value function or a policy but

Moving beyond model-based and model-free labels

The sharp distinction between model-based and model-free methods is becoming somewhat less useful.

1. For tabular RL there is an exact output equivalence between some conventional model-based and model-free algorithms.
2. When the agent stores transitions in an *experience replay buffer* and learns from it (as in DQN), we can think of this stored experience as an implicit model.
3. More generally, an agent can store its experience in other forms. In those cases it is unclear if either or both labels apply.

The terms are still used to describe whether an algorithm is explicitly modeling the environmental dynamics (to a greater or lesser extent), and how the agent is generalizing from past experience.

distinction anymore
so

Using experience in the place of a model

Recall prioritized sweeping from tabular dynamic programming.

- Update the value function of the states with the largest magnitude Bellman errors using a priority queue.

A related idea is prioritized experience replay (Schaul et al, 2015) which works from experience for general function approximation.

- The experience replay buffer maintains a priority for each transition, with the priority given by the magnitude of the Bellman error.
- Minibatches are sampled using this priority to quickly reduce errors.
- Weighted importance sampling corrects for bias from non-uniform sampling.

is sometimes called prioritize sweeping
proactive sweeping is a more general

MC samples multi steps Model one-step

High Varience (input)

easy to learn

TD off-policy learning (Q-learning)

Limits of Planning with an Inaccurate Model

- Given an imperfect model $\hat{p}_\eta \neq p$
- Performance is limited to optimal policy for approximate MDP $\langle \mathcal{S}, \mathcal{A}, \hat{p}_\eta \rangle$
- Model-based RL is only as good as the estimated model
- When the model is inaccurate, planning process will compute a suboptimal policy (not covered in these slides)
  - Approach 1: when model is wrong, use model-free RL
  - Approach 2: reason explicitly about model uncertainty over η (e.g. Bayesian methods)
  - Approach 3: Combine model-based and model-free methods in a safe way.

free reinforcement learning
additionally or alternatively

Real and Simulated Experience

Real Experience: sampled from environment

(true MDP)

$$r, s' \sim p$$

Simulated Experience: sampled from model

(approximate MDP)

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

Integrating Learning and Planning

- Model-Free RL
  - No model
  - Learn value function (and/or policy) from real experience
- Model-Based RL (using Sample-Based Planning)
  - Learn a model from real experience
  - Plan value function (and/or policy) from simulated experience
- Dyna
  - Learn a model from real experience
  - Learn AND plan value function (and/or policy) from real and simulated experience
  - Treat real and simulated experience equivalently. Conceptually, the updates from learning or planning are not distinguished.

and this is sometimes called dinah this
is how how it's called in the

Dyna-Q Algorithm (Dyna + Q-learning)

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

(a) $s \leftarrow$ current (nonterminal) state
(b) $a \leftarrow \varepsilon\text{-greedy}(s, Q)$
(c) Execute action a; observe resultant state, s', and reward, r
(d) $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
(e) $Model(s,a) \leftarrow s', r$ (assuming deterministic environment)
(f) Repeat N times:

$s \leftarrow$ random previously observed state
$a \leftarrow$ random action previously taken in s
$s', r \leftarrow Model(s,a)$
$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

MacBook Pro

Simulation-based Model

Given (fixed) model,

$$\text{Global Value Function} \xleftarrow{\text{improve}} \text{planning}$$

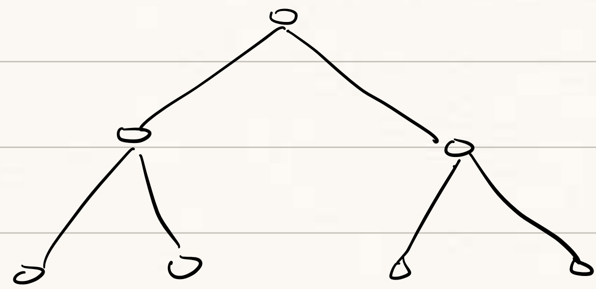
$$\text{planning} \xrightarrow{\text{select}} \text{Next Action}$$

Forward Search

Select the best action by lookahead

Search Tree

S_t current state



Use a MDP model to look ahead

Simulation-based Search

Forward search uses sample-based planning

Simulated episodes

Model-free

RL

$$\boxed{\{ S_t^k, A_t^k, R_{t+1}^k, \dots, S_T^k \}_{k=1}^K \sim \hat{P}_\eta}$$

- Monte-Carlo control $\longrightarrow$ MC Search
- Sarsa $\longrightarrow$ TD search

Search tree vs. value function approximation

- Search tree is a table lookup approach
- Based on a partial instantiation of the table
- For model-free reinforcement learning, table lookup is naive
  - Can't store value for all states
  - Doesn't generalise between similar states
- For simulation-based search, table lookup is less naive
  - Search tree stores value for easily reachable states
  - But still doesn't generalise between similar states
  - In huge search spaces, value function approximation is helpful

so for mobile free reinforce

Reinforcement Learning 7: Planning and Models

10,164 views • Nov 23, 2018

Monte-carlo Simulation

Parameterized model M_η

Simulation policy π

① Simulate K episodes from current state S_t

$$\{ S_t^k = S_t, A_t^k, R_{t+1}^k, S_{t+1}^k, \dots, S_T^k \}_{k=1}^{K} \sim \hat{P}_\eta, \pi$$

$$S_t^k = s, \quad A_t^k = a$$

② Evaluate state by mean return

(Monte-Carlo evaluation)

$$v(S_t) = \frac{1}{K} \sum_{k=1}^{K} G_t^k \rightsquigarrow V_\pi(S_t)$$

③ Evaluate action by mean return

$$q(s, a) = \frac{1}{K} \sum_{k=1}^{K} G_t^k \rightsquigarrow q_\pi(s, a)$$

④ Select current (real) action with maximum value

$$A_t = \underset{a \in A}{\text{argmax}}\ q(S_t, a)$$

Monte-Carlo Tree Search (Evaluation)

Given Model M_η

1. Simulate K episodes from current state S_t using current simulation policy π

$$\{S_t^k = S_t, A_t^k, R_{t+1}^k, S_{t+1}^k, \ldots, S_T^k\}_{k=1}^K \sim M_\eta, \pi$$

2. Build Search Tree containing Visited states and actions

3. Evaluate states $q(s, a)$ by mean return of episodes from s, a

$$q(s,a) = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{u=t}^{T} I(S_u^k, A_u^k = s, a)\, G_u^k \rightsquigarrow q_\pi(s,a)$$

4. After searching, select current (real) action with maximum value in search tree

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \; q(S_t, a)$$

Monte-Carlo Tree Search (Simulation)

- In MCTS, the simulation policy π improves
- The simulation policy π has two phases (in-tree, out-of-tree)
  - Tree policy (improves): pick actions from $q(s, a)$ (e.g. $\epsilon -$ greedy$(q(s, a))$)
  - Rollout policy (fixed): e.g., pick actions randomly
- Repeat (for each simulated episode)
  - Select actions in tree according to tree policy.
  - Expand search tree by one node
  - Rollout to termination with default policy
  - Update action-values q(s,a) in the tree
- Output best action when simulation time runs out.
- With some assumptions, converges to the optimal values, $q(s, a) \rightarrow q_*(s, a)$

going to consider a tree that we've
built ourselves and

MacBook Pro

Game Go

AlphaGo