A Guided Tour of Chapter 14: Blending Learning and Planning

Ashwin Rao

ICME, Stanford University

March 1, 2022

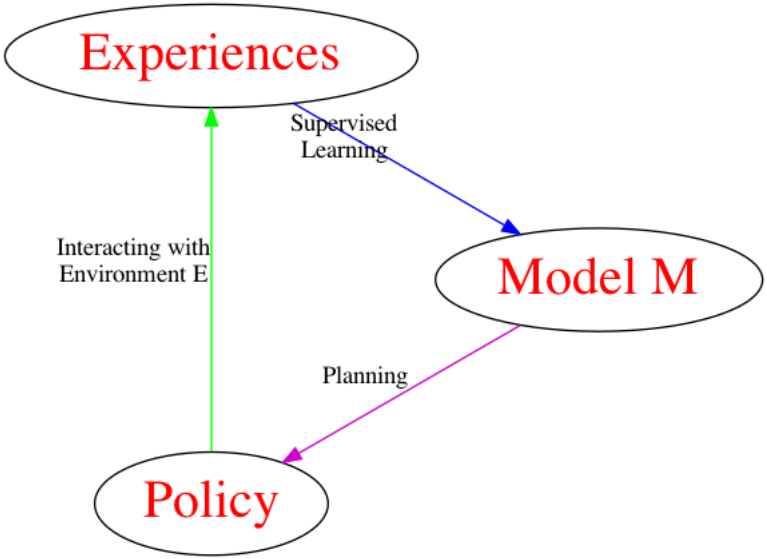
Planning versus Learning

- *Planning* and *Learning* refer to two different AI methodologies
- Let's understand what they mean for MDP Prediction and Control
- AI Agent has access to an MDP Environment E
- By interacting with E, AI Agent receives experiences data
- Goal is to estimate requisite VF/Policy using these experiences
- Broadly, there are two ways to do this:
  - *Planning* (Model-based approach): Build a model M from interaction experiences, then use M to estimate requisite VF/Policy
  - *Learning* (Model-free approach): Don't bother with a model, use interaction experiences to directly estimate VF/Policy (with RL)

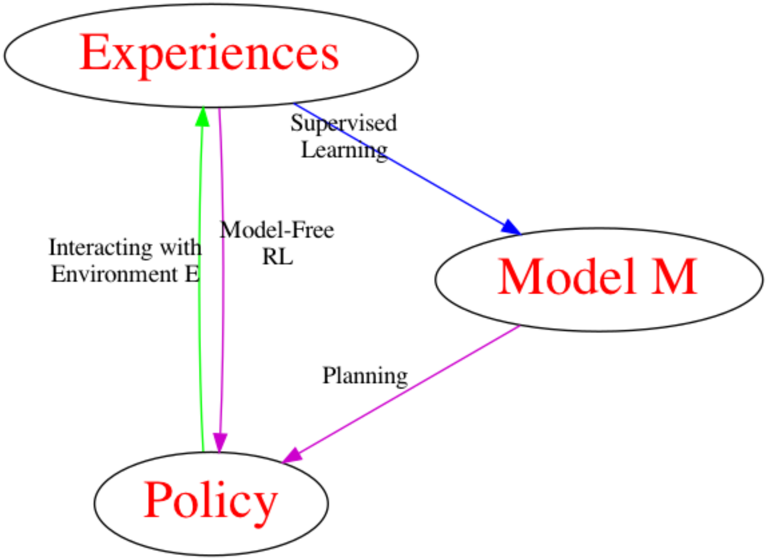
Two different approaches to *Planning*

- Building model M from experiences data is Supervised Learning
- Then estimate requisite VF/Policy using model M
- There are two different approaches to estimate VF/Policy from M
- The first approach is to build an explicit representation of $\mathcal{P}_R$
- Then employ Dynamic Programming or Tree-Search
- This approach doesn't require interactions with E (since we have M)
- The second approach is to build a sampling model of $\mathcal{P}_R$
- Sampling model gives us a Simulated Environment S
- Now interact with S (instead of E) and solve VF/Policy using RL
- This approach is known as Model-based RL

Planning with a Supervised-Learnt Model



Blending Planning and Learning

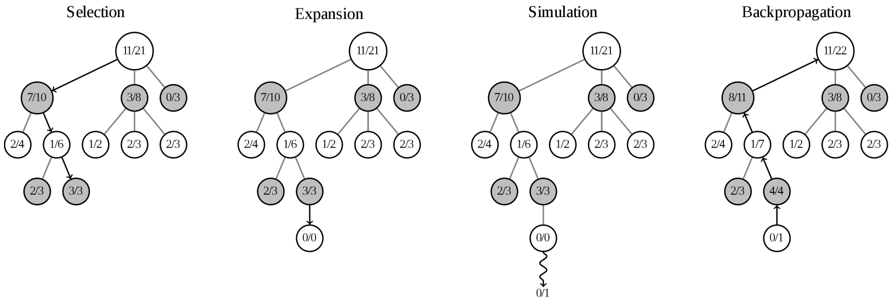


Decision-Time Planning

- *Background Planning*: Pre-compute requisite VF/Policy *for all states*
- So when it's time to perform an action for a given state, just refer to pre-computed Policy
- In the *background*, the VF/Policy is being constantly improved (irrespective of which state is to be acted on)
- *Decision-Time Planning*: The calculations to identify the best action for a state are started only upon reaching that state
- Do this when too many states make Background Planning infeasible
- However, this means we need sufficient time to perform the calculations to identify the best action upon reaching a specific state
- Decision-Time Planning typically looks deeper than a time step ahead
- By growing out a tree of future states/actions/rewards
- Tree rooted at the state of current interest (where the entire focus is)

Monte Carlo Tree Search (MCTS)

- MCTS was popularized a few years ago by Deep Mind's AlphaGo
- It is a simulation-based method to identify the best action in a state
- MCTS term was first introduced by Remi Coulom for game trees
- Each round of MCTS consists of four steps:
  - Selection: Successively select children from root R to leaf L
  - Expansion: Create node C as a new child of L
  - Simulation: Complete a random playout from C
  - Backpropagation: Use result of playout to update nodes from C to R



The Selection Step in MCTS

- Selection involves picking a child with “most promise”
- This means prioritizing children with higher success estimates
- For estimate confidence, we need sufficient playouts under *each* child
- This is our usual *Explore v/s Exploit* dilemma (Multi-armed Bandit)
- Explore v/s Exploit formula for games first due to Kocsis-Szepesvari
- Formula called *Upper Confidence Bound 1 for Trees* (abbrev. UCT)
- Most current MCTS Algorithms are based on some variant of UCT
- UCT is based on UCB1 formula of Auer, Cesa-Bianchi, Fischer
- However, MCTS and UCT concepts first appeared in the Adaptive Multistage Sampling algorithm of Chang, Fu, Hu, Marcus
- Adaptive Multistage Sampling (AMS) is a generic simulation-based algorithm to solve a finite-horizon Markov Decision Process (MDP)
- AMS can be considered as the “spiritual origin” of MCTS/UCT
- Hence, this lecture is dedicated to AMS

The Setting for the AMS Algorithm

- MDP with finite number of time steps $t = 0, 1, \ldots, T$
- State denoted $s_t \in \mathcal{S}$, where $\mathcal{S}$ is very large
- Action denoted $a_t \in \mathcal{A}$, where $\mathcal{A}$ is fairly small
- Reward $r_t \in \mathbb{R}$, with $\mathbb{E}[r_t|(s_t, a_t)]$ provided as a function $R(s_t, a_t)$
- Next time step's state s_{t+1} can be generated by invoking a random sampling function $SF(s_t, a_t)$, i.e., $s_{t+1} = SF(s_t, a_t)()$
- Discount factor denoted as γ, and $r_T = 0$
- The problem is to calculate the Optimal Value function $V_t^*(s_t)$
- Unlike tabular backward induction where state transition probabilities are given, here only a sampling function (for next state) is given
- Armed with the sampling function, can we do better than backward induction for the case where $\mathcal{S}$ is very large and $\mathcal{A}$ is small?

Outline of AMS Algorithm

- AMS Algorithm is based on a fixed allocation of action selections for each state in each time step
- Denote number of action selections per state in time step t as N_t
- Denote $\hat{V}_t^{N_t}(s_t)$ as the AMS Algorithm estimate of $V_t^*(s_t)$
- Let $N_t^{s_t,a_t}$ be the number of selections of a_t for s_t ($\sum_{a_t \in \mathcal{A}} N_t^{s_t,a_t} = N_t$)
- Proportions of $N_t^{s_t,a_t}$ based on Explore v/s Exploit UCT formula
- For each of the $N_t^{s_t,a_t}$ selections of a_t, *one* next-state s_{t+1} is sampled
- Each $s_{t+1} = SF(s_t, a_t)()$ sample leads to recursive call $\hat{V}_{t+1}^{N_{t+1}}(s_{t+1})$
- Optimal Action Value Function $Q_t^*(s_t, a_t)$ estimated as:

$$\hat{Q}_t^{N_t}(s_t, a_t) = R(s_t, a_t) + \gamma \cdot \frac{\sum_{j=1}^{N_t^{s_t,a_t}} \hat{V}_{t+1}^{N_{t+1}}(SF(s_t, a_t)())}{N_t^{s_t,a_t}}$$

- $V_t^*(s_t) = \max_{a_t} Q_t^*(s_t, a_t)$ approximated as:

$$\hat{V}_t^{N_t}(s_t) = \sum_{a_t} \frac{N_t^{s_t,a_t}}{N_t} \cdot \hat{Q}_t^{N_t}(s_t, a_t)$$

The AMS Algorithm

Algorithm 0.1: $\text{OptVF}(t, s, N_t)$

```
if $t == T$ return $(0)$
comment: Initialize VALS and CNTS by selecting each action once
for $a \leftarrow \mathcal{A}$
  do { $VALS[a] \leftarrow OptVF(t+1, SF(s,a)(), N_{t+1})$
       $CNTS[a] \leftarrow 1$
for $i \leftarrow |\mathcal{A}|$ to $N_t - 1$
  do { comment: Pick action based on UCB1 Explore v/s Exploit formula
       $a^* \leftarrow \text{argmax}_{a \in \mathcal{A}}(R(s,a) + \gamma \cdot \frac{VALS[a]}{CNTS[a]} + \sqrt{\frac{2 \ln i}{CNTS[a]}})$
       $VALS[a^*] \leftarrow VALS[a^*] + OptVF(t+1, SF(s,a^*)(), N_{t+1})$
       $CNTS[a^*] \leftarrow CNTS[a^*] + 1$
return $(\sum_{a \in \mathcal{A}} \frac{CNTS[a]}{N_t} \cdot (R(s,a) + \gamma \cdot \frac{VALS[a]}{CNTS[a]}))$
comment: $N_t$ next-state samplings and $N_t$ recursive calls to $OptVF$
```

Running Time, Bias, Convergence and Code

- Let $N = \max(N_0, N_1, \ldots, N_{T-1})$ and assume $N > |\mathcal{A}|$
- Running time of AMS Algorithm is of the order of $N^T \cdot |\mathcal{A}|$
- Compare this versus backward induction running time of $|\mathcal{S}|^2 \cdot |\mathcal{A}| \cdot T$
- So AMS is more efficient when $\mathcal{S}$ is very large (typical in real-world)
- AMS paper proves the estimate $\hat{V}_0^{N_0}(s_0)$ is asymptotically unbiased

$$\lim_{N_0 \to \infty} \lim_{N_1 \to \infty} \ldots \lim_{N_{T-1} \to \infty} \mathbb{E}[\hat{V}_0^{N_0}(s_0)] = V_0^*(s_0) \text{ for all } s_0 \in \mathcal{S}$$

- AMS paper also proves that the worst-possible bias is bounded by a quantity that converges to zero at rate $O(\sum_{t=0}^{T-1} \frac{\ln N_t}{N_t})$

$$0 \leq V_0^*(s_0) - \mathbb{E}[\hat{V}_0^{N_0}(s_0)] \leq O(\sum_{t=0}^{T-1} \frac{\ln N_t}{N_t}) \text{ for all } s_0 \in \mathcal{S}$$

- Here's some Python code for the AMS Algorithm you can play with