Markov-Style State Machines

Nathaniel Hamovitz, Ron Kibel, Rianna Alers
University of California Santa Barbara

Reinforcement Learning

Reinforcement learning (RL) is a technique for training an agent to act favourably in an arbitrary environment. Environments consist of states, actions, observations, and rewards. The agent's job is to choose actions that maximize the total reward.
There are many approaches to solving this task. The agent can try to predict the long-term consequences of its actions and maximize its long-term benefits (value-based approach). It can also try to learn the environment itself (model-based approach). Here we will look at what is known as a **policy-based** approach to RL, which refers to optimizing the sequence of actions we take to maximize our current and future rewards.

Markov Models

These models come in different variants, with different tradeoffs. The following four classes of models are all "stochastic, discrete state, discrete time" [1] finite state machines with Markovian dynamics – that is, the next state depends only on the current state and the action. We restrict our discussion here to **MDPs (Markov Decision Processes)** and **POMDPs (Partially Observable Markov Decision Processes)**, but you may be familiar with other Markovian processes:

States Completely Observable?	Control over State Transitions?	
	NO	YES
YES	Markov Chain	MDPs
NO	Hidden Markov Model	POMDPs

MDPs and POMDPs

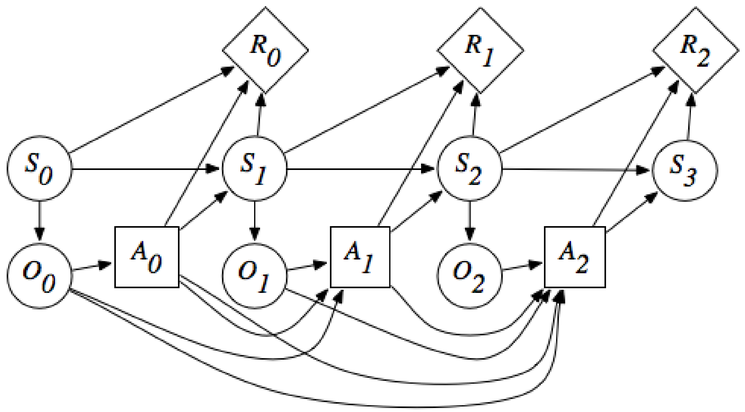
Both MDPs and POMDPs are especially helpful in the RL setting because they can represent the relationship between the agent and the environment.

- In an MDP, we assume the environment is fully observable. As such, the states fully capture all relevant information for decision-making (state transitions contain $P_a(s \mid s')$, the probability that taking action a at time t in state s will lead us to state s' at time $t+1$).
- We can generalize an MDP into a POMDP by assuming that the environment is only partially observable. This implies that states in our model only capture part of the true environment—observations give us information that are only partially informative, which makes it more challenging! We thus need to introduce what is known as the **belief state**, which is a probability distribution over all states encoding how closely a state resembles the true environment [3].

Formally, a POMDP is a tuple of the following items:

- S, a set of states
- A, a set of actions
- $P(s_t \mid a, s_{t-1})$, state-action transition probabilities
- $R : S \times A \to \mathbb{R}$, state-action reward function
- O, a set of observations
- $P(o \mid s)$, conditional observation probabilities
- $\gamma \in [0, 1)$, the discount factor

Diagram of a POMDP



Adaptive State Aggregation for MDPs

MDPs can be solved by iterating the operator

$$T_s(\mathbf{V}) = \min_{a \in \mathcal{A}} \left(r(s,a) + \gamma \cdot \mathbf{P}_{s,a}^{\top} \mathbf{V} \right).$$

Value iteration is simple and guarantees convergence, but is computationally expensive. For large state spaces solving an MDP with this technique becomes infeasible! We can use **state aggregation** to reduce these costs by dynamically grouping states with similar cost-to-go values [2].

This algorithm alternates between two phases: a global update phase and an aggregated update phase. The global update phase performs value iteration on S and the aggregated update phase groups together states with similar cost-to-go values.
We need both phases because the aggregated update phase will require updated knowledge of V* to perform aggregation. In the following algorithms, A_i reference our state-aggregation and B_i reference global iterations.

Value Iteration

We must use some form of value-iteration to obtain a value function for our aggregation. We observe an algorithm using a pre-specified aggregation where W is the value function generated by the mega-states and $\tilde{V}$ is the incuced value function. In the following algorithm α_t is the step size of the learning algorithm (α_t = 1 recovers the formula for value iteration). This has been proven to converge and our alterations maintain a similar convergence bound [2].

Algorithm 1 Random Value Iteration with Aggregation

Input: $\mathbf{P}, r, \gamma, \mathbf{\Phi}, \{\alpha_t\}_{t=1}^{\infty}$
Initialize $\mathbf{W}_0 = \mathbf{0}$
for $t = 1, \ldots, n$ **do**
  for $j = 1, \ldots, K$ **do**
    Sample state s uniformly from set S_j
    $W_{t+1}(j) = (1 - \alpha_t) W_t(j) + \alpha_t T_s \mathbf{V}(\mathbf{W}_t)$
  end for
end for
Output: $\tilde{\mathbf{V}}_n$

State Aggregation

Divide the state space S into K subsets and view these subsets as mega-states. The value function generated by each mega-state can be used to find the optimal value V^*. The algorithm is below:

Algorithm 2 Value-based Aggregation [2]

Input: $\varepsilon, V = (V(1), ..., V(|S|))^T$
$b_1 = \min_{s \in |S|} V(s), b_2 = \max_{s \in |S|} V(s), \Delta = (b_2 - b_1)/\varepsilon$
for $i = 1, ..., \Delta$ **do**
  $\hat{S}_i = \{s | V(s) \in [b_1 + (i-1)\epsilon, b_1 + i\epsilon)\}$
  $W(i) = b_i + (i - \frac{1}{2})\varepsilon$
end for
Return $\{S_i\}_{i=1}^{K}$ and W

Adaptive State Aggregation Algorithm

Combining the two techniques above, we get the following algorithm where A_i are our state-aggregation and B_i are our global iterations.
This method is separate from other aggregation techniques because it learns the cost-to-go values continuously, which aggregation methods need to generate mega-states.

Algorithm 3 Value Iteration with Adaptive Aggregation [2]

Input: $\mathbf{P}, r, \varepsilon, \gamma, \{\alpha_t\}_{t=1}^{\infty}, \{\mathcal{A}_i\}_{i=1}^{\infty}, \{\mathcal{B}_i\}_{t=1}^{\infty}$
Initialize $W_0 = \mathbf{0}, V_1 = \mathbf{0}, t_{sa} = 1$
for $t = 1, \ldots, n$ **do**
  if $t \in B_i$ **then**
    if $t = \min\{B_i\}$ **then**
      $\mathbf{V_{t-1}} = \tilde{\mathbf{V}}(\mathbf{W_{t-1}})$
    end if
    for $j = 1, \ldots, |S|$ **do**
      State $V_t(j) = T_j V_{t-1}$
    end for
  else
    Find current i s.t. $t \in \mathcal{A}_i$
    if $t = \min\{A_i\}$ **then**
      Run our Value-based Aggregation algorithm with input ε, V_{t-1}
      Set the $\{S_i\}_{i=1}^{K}$ and W_t to be the output of our Algorithm
    end if
    for $j = 1, ..., K$ **do**
      Sample state s uniformly from set S_j
      $W_t(j) = (1 - \alpha t_s a) W_{t-1}(j) + \alpha_{t_{sa}} T_s \mathbf{V}(\mathbf{W_{t-1}})$
    end for
    $t_{sa} = t_{sa} + 1$
  end if
end for
if $n \in \mathcal{B}_i$ **then** return V_n
end if
return $\tilde{\mathbf{V}}(\mathbf{W_n})$

Acknowledgements

We are grateful to Charles Kulick for his mentorship and guidance.

References

[1] Anthony R. Cassandra. *POMDP FAQ*. URL: https://pomdp.org/faq.html.

[2] Guanting Chen et al. "An Adaptive State Aggregation Algorithm for Markov Decision Processes". In: (2021). arXiv: 2107.11053 [cs.LG].

[3] Milos Hauskrecht. "Value-Function Approximations for Partially Observable Markov Decision Processes". In: *Journal of Artificial Intelligence Research* 13 (Aug. 2000).