Notes on Model-based RL: A Survey

Rom Parnichkun

November 24, 2022

- Difference between planning and RL:
  - **Planning** is a class of MDP algorithms that use a model and store a local solution.
  - **Reinforcement Learing** is a class of MDP algorithms that share a global solution.
- **Model-based reinforcement learning** is a class of MDP algorithms that use a model, and store a global solution.
- *Learning* can be done in three ways:
  - Model-based RL with a learned model, where we both learn a model and learn a global solution.
  - Model-based RL with known model, where we plan over a known model, and only use learning for the global solution.
  - Planning over a learned model, where we do lean a model, but subsequently locally plan over it (without learning a global solution).
- There are various types of **dynamics models**, which attempt to learn the transition probabilities between states.
  - **Forward model**: $(s_t, a_t) \rightarrow s_{t+1}$ [1]
  - **Backward/reverse model**: $s_{t+1} \rightarrow (s_t, a_t)$
  - **Inverse model**: $(s_t, s_{t+1}) \rightarrow a_t$
- There are also several estimation methods utilized in dynamics models:
  - Parametric: Model doesn't depend on size of observed dataset, instead, uses parameters to approximate the transition probabilities.
  - Non-parametric: Uses the observed dataset itself as the model. We can think of replay buffers as a form of non-parametric model, or in other cases, Gaussian processes can be used as well.
- Finally, another important consideration is the region in which the model is valid:
  - Globally valid models
  - Locally valid models: Only locally approximate the dynamics model (popular in the control community). Benefit is that we can use a linear model which is easier to analyze.
- **Stochasticity**: In a stochastic MDP, the transition function specifies a distribution over possible states.
- **Uncertainty**: When our observed output is limited, the model may be uncertain about the transition model of unobserved regions.
  - Bayesian methods: Gaussian process techniques (good but poor scalability)
  - Variational inference and variational dropout
- **Partial Observability**: When the observation does not provide all information necessary to infer a state in the MDP. Usually, this can be partially mitigated by incorporating observation from multiple timesteps.
  - Windowing: Concatenate n most recent observations and treat them together as a state.
  - Belief states: Partitions problem into $p(o \mid s)$ and $\mathcal{T}(s' \mid s, a)$.
  - Recurrency: Use recurrent neural networks. Suffers from vanishing and exploding gradient.
  - External memory: Use an external memory (triggered by an agent) to memorize historical information.
- **Non-stationarity** in an MDP occurs when the true transition and/or reward function changes over time.
- **Multi-step prediction**: We eventually intend to use these models in multi-step planning procedures.
  - Loss function for multi-step predictions
  - Leanrn a new dynamics model for different time steps. (i.e., $\mathcal{T}^3(\hat{s}_{t+3} \mid s_t, a_t, a_{t+1}, a_{t+2})$)
- **State abstraction**: In neural network-based approach, the dynamics model is typically factorized into three parts:
  - Encoding function: $z_t = f^{\text{enc}}(s_t)$
  - Latent dynamics function: $z_{t+1} = f^{\text{trans}}(z_t, a_t)$

[1] This is the most common type of model.

  - Decoder function: $s_{t+1} = f^{\text{dec}}(z_{t+1})$

  Additionally, there are 3 important additional these for state representation learning in dynamics models:

  - How do we ensure that we can plan at the more abstract level? If we can ensure this, we can learn in latent space (which is usually faster to roll-out). Some researchers find latent representations that follow a linear model (then use techniques like LQR to find the agent).
  - How may we better structure our models to emphasize objects and their physical interactions? (Object-oriented models?)
  - How may we construct loss functions that retrieve more information representations? (Predict relative effect $s_{t+1} - s_t$ helps focus on moving objects, contrasive loss, value equivalent models (create dynamics model that can correctly predict value functions)).

- **Temporal abstraction** (hierarchical reinforcement learning): Identifying a high level action space that extends over multiple timesteps. Two popular methods are the *options* method, discrete set of high level actions; and the **goal-conditioned policy/value functions** (GCVF), $Q(s, a, g)$ which is a universal value function. How to discover the subroutines?

  - Graph structure: Identify bottleneck states (which are like checkpoints).
  - State-space coverage: Cluster state-spaces, then identify policies to move between the centers of each cluster.
  - Compression: Associate state-space with noise distribution.
  - Reward relevancy: Relevant subroutines will help incur extra reward, and they should therefore automatically emerge from a black-box optimization approach.
  - Priors: Use prior knowledge to identify useful subroutines.

- Four main question of the integration of planning and learning:
  - At which state do we start planning?
    * *Uniform*: Select all states in the state space and formulate a plan for each state. Drawback is that it is very time-consuming and intractable for high dimensional spaces.
    * *Visited*: Only plan for visited states.
    * *Prioritized*: Prioritize which states likely would need planning.
    * *Current*: Only spend effort planning for the current state.
  - How much planning budget do we allocate?
  - How to plan?
    * Type: (Discrete planning): Table-based or tree-based planning. They do not require differentiability of the model. E.g., MCTS, minimax-search.
    * Type: (Differential planning): Requires model to be differentiable. We can directly find the derivative of the policy with respect to the cumulative predicted rewards of the environment model. E.g., Dreamer, PILCO, Guided policy search, iterative linear quadratic regulator planning.
    * Breadth and depth
    * Uncertainty: (data-close planning): Plan close to past observation.
    * Uncertainty: (uncertainty propagation): Explicitly estimate model uncertainty, so that planning prediction spread out (vanish) over long horizon.
  - How to integrate planning in the learning and acting loop?
    * Input to planning: Policy/value prior can help in planning.
    * Updating with planning: Planning can help in updating policy/value.
    * Output of planning: Planning can help provide the action to be performed.
  - Implicit Model-based RL: Take one or more aspects of model-based RL process and optimize these for the ultimate objective (optimal value or policy computation). For optimal value, simply train the "planning" part to predict the correct **optimal** value. Or planning itself can be some sort of differentiable procedure, where in a MCTS search, each action (backup, selection, etc.) is parameterized using a neural network.