Reinforcement Learning: Model-based

Chris R. Sims
Department of Brain & Cognitive Sciences
University of Rochester
Rochester, NY 14627, USA

July 24, 2012

Reference: Much of the material in this note is from Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press. An electronic copy of the book is freely available at `http://webdocs.cs.ualberta.ca/~sutton/book/the-book.html`.

1. Model-free versus model-based reinforcement learning

Reinforcement learning (RL) refers to a wide range of different learning algorithms for improving a behavioral policy on the basis of numerical reward signals that serve as feedback. In its basic form, reinforcement learning bears striking resemblance to 'operant conditioning' in psychology and animal learning: actions that are rewarded tend to occur more frequently; actions that are punished are less likely to be repeated.

Because of its simplicity, reinforcement learning is applicable to an incredibly wide range of different problems. Unfortunately, also because of its simplicity, RL often exhibits extremely slow learning in complex problems. An ongoing challenge in the field of machine learning is developing learning algorithms that share the generality and elegance of RL, but learn in a more 'intelligent' and sophisticated manner. This note considers one particular approach to extending the capabilities of RL algorithms, known as model-based reinforcement learning. Model-based RL is also of significant interest to researchers in cognitive science, as it is believed to more closely correspond to the type of learning that humans routinely demonstrate.

In model-based RL, as in the basic model-free approach, the primary goal of learning is the improvement of a behavioral policy in order to maximize a numerical reward signal. However, the approach differs from model-free RL in that simultaneously, the agent attempts to learn a model of its environment. Having such a model is a useful entity: it allows the agent to predict the consequences of actions before they are taken, allowing the agent to generate virtual experience, as well as perform mental search through a problem space to locate an efficient solution. Thus, model-based RL integrates both learning on the basis of past experience, and planning future actions. Further, a model of the environment is not directly tied to the task one is currently performing. For example, consider a mouse learning to navigate a maze to find a food reward. While performing this task, the mouse could acquire a mental representation of the structure of the maze. If the mouse's goals change, for example, from finding food to finding a safe place to hide, the previously learned policy (the sequence of turns to locate the food reward) is no help, but the learned model of the maze structure could still aid the mouse in achieving its new goal. The downside of model-based RL is that acquiring a model in the first place adds to the overall

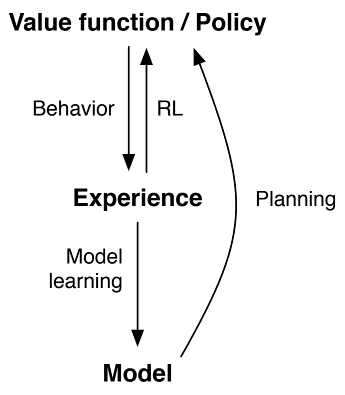

Figure 1: Relationship between a policy, experience, and model in reinforcement learning. Arrows indicate a causal influence (such as learning). Figure adapted from (Sutton & Barto, 1998).

complexity of the learning problem, and inaccurate models can actually impair performance, relative to model-free RL approaches.

Figure 1 illustrates the relationship between a behavioral policy, the experience it generates, and the role of a model of the environment in reinforcement learning. In standard RL, experience leads to an improved policy. However, this same experience can also be used to learn a model, which in turn can be used (via planning) to generate better actions. The additional challenges in model-based RL learning are learning the model, and using it intelligently to improve behavior. The next section briefly addresses both of these challenges.

2. Learning and using a model

If the task environment is deterministic, and consists of only a modest number of discrete states, then a simple approach to model learning is to implement a lookup table. This table stores, for each state and action, the resulting state and reward. Initially, the lookup table will be empty. Each time an action is taken and a reward and new state observed, this information is stored in the table. The table can be used to predict the consequences of actions, without actually executing them in the environment. Thus, a simple approach to using models to improve RL is to use the model to generate simulated experience. The agent can randomly sample a state and action, along with its predicted successor state and reward, and apply standard reinforcement learning algorithms (such as Q-learning) to improve the behavior policy. By including this additional 'virtual' experience, the backwards propagation of temporal difference (TD) errors occurs much more quickly from the point that rewards are received, to the past actions that preceded the rewards.

For tasks that involve probabilistic rewards or state transitions, the same general approach can be applied, but in this case the model should ideally capture the stochastic nature of the environment. One approach is to maintain a distribution over possible successor states and rewards, updating the distributions via Bayesian inference. Then, when using the model to generate virtual experience, transitions can be sampled from the learned distributions. The downside to this approach is that the model must learn separate distributions (each likely involving multiple parameters) for each state. For large state spaces, this approach will be computationally impractical. Even for simple problems, a model with many free parameters can be quite inaccurate at predicting the consequences of actions when given only limited experience. As a result, the performance of model-based RL algorithms can actually be worse than simple, model-free approaches. At present, there is no general, efficient solution to these challenges, and model-based RL algorithms are typically tailored or tuned to each task.

2.1 Prioritize sweeping

In the previous section we considered a simple approach to using a model to improve RL. This approach involved generating 'virtual experience' by sampling from previously observed transitions, and applying standard RL algorithms to this experience. As described, this approach sampled randomly from its past experience. There is a more intelligent way of using past experience, however.

Consider an agent attempting to learn to play checkers. The reward function is +1 if the agent wins a game, and -1 if it loses. Assume that the value function is initialized to zero for all states and actions. On the first game, all actions except the final move have zero reward; thus, sampling from this experience and performing RL updates will have no effect on the value function. Similarly, on the second game, only the state immediately before the end of the game will have non-zero utility. The simple approach to model-based learning ignores this fact, and would sample many stored actions that have little or no impact on the value function, because the vast majority of actions move an agent from a state with zero value to another state with zero value.

A more intelligent approach is to only sample previous transitions if applying RL updates to those transitions would have a large change on the value function. As applied to our checkers example, this approach would focus the model-based updates to the actions that immediately preceded the end of the game. After these updates are performed, the states and actions leading to the updated states would also benefit from model-based learning. Thus, one can imagine a 'wave' of updates starting at states with large or small reward, and working backwards through the states and actions leading to the reward. This corresponds to an approach to planning in which one first focuses on the goal, and works backwards to find a path for reaching it.

A further refinement to this approach is to prioritize the stored experience according to how big of an impact it would have to update each state-action transition. This 'impact' is simply defined as the temporal difference (TD) error for a given state-action pair. Using this approach, model-based learning is focused on the updates that would have the biggest TD error, and hence the largest change on the value function. In the reinforcement learning literature, this approach is known as 'prioritized sweeping'. The implementation of this algorithm requires a priority queue, a data structure that sorts its entries according to a numerical 'priority' score.

Algorithm 1 provides pseudo-code for the prioritized sweeping algorithm. The priority queue ranks all state-action transitions according to the TD error associated with performing an update on that state-action pair. A threshold parameter, ζ, is defined so that model-based updates are only applied to transitions that would have a sufficiently large impact on the value function. When each update is performed, it is necessary to re-compute the priority for every state leading to the updated state. In this way, updates are performed until nothing further can be learned from the experience stored in the model. Note that the code as presented in Algorithm 1 is only applicable to environments with discrete states and deterministic transitions. Extending this same basic approach to continuously defined states represents a difficult and ongoing challenge in reinforcement learning. Readers are encouraged to consult (Sutton & Barto, 1998) for more details.

Algorithm 1 Prioritized sweeping algorithm.

```
Inputs:
    $Q(s, a)$: State-action values, initialized arbitrarily
    $M(s, a)$: Model, predicting the reward and successor state for taking action $a$ in state $s$
    $s_0$: Initial state
Algorithm:
    Initialize the priority queue, $PQ$, to be empty
    $s \leftarrow s_0$
    Repeat
        Choose $a$ for the current state $s$ based on Q(s,a)
        Take action $a$, observe reward $r$ and resulting state $s'$
        $M(s, a) \leftarrow (s', r)$            // Update the model
        $p \leftarrow |r + \gamma \max_{a'} Q(s', a') - Q(s, a)|$
        Insert $(s, a)$ into $PQ$ with priority $p$
        Repeat
            $(s, a) \leftarrow PQ$            // Remove the highest priority entry from the queue
            $(s', r) \leftarrow M(s, a)$
            $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$
            Compute priority for all states $(\bar{s}, \bar{a})$ that lead to $s$, and add to $PQ$
        Until all entries in PQ have priority $p < \zeta$
        $s \leftarrow s'$
    Until the end of the learning episode
    Return $Q(s, a)$, the updated state-action values
```