Markov Decision Processes

Ryan P. Adams*
COS 324 – Elements of Machine Learning
Princeton University

We now turn to a new aspect of machine learning, in which agents take *actions* and become active in their environments. Up until this point in the course, agents have been passive learners: hanging out and analyzing data. We are heading towards the interesting concept of *reinforcement learning* (RL), in which an agent learns to act in an uncertain environment by training on data that are sequences of state, action, reward, state, action, reward, etc. The fun part of RL is that the training data depend on the actions of the agent, and this will lead us to a discussion of *exploitation* vs. *exploration*.

But for now we will consider *planning* problems in which the agent is given a probabilistic model of its environment and should decide which actions to take. We begin with a brief introduction to utility theory and the maximum expected utility principle (MEU). This provides the foundation for what it means to be a rational agent: we will take a rational agent as one that tries to maximize its expected utility, given its uncertainty about the world.

Our focus in Markov decision processes (MDPs) will be on the more realistic situation of repeated interactions. The agent gets some information about the world, chooses an action, receives a reward, gets some more information, chooses another action, receives another reward, and so on. This type of agent needs to consider not only its immediate reward when making its decision, but also its utility in the long run.

1 Decision Theory

Here is a basic overview of the decision theoretic framework:

- We use *probability* to model uncertainty about the domain.
- We use *utility* to model an agent's objectives.
- The goal is to design a *decision policy*, describing how the agent should act in all possible states in order to maximize its expected utility.

Decision theory can be viewed as a definition of *rationality*. The *maximum expected utility principle* states that a rational agent is one that chooses its policy so as to maximize its expected

*These notes are adapted from Harvard CS181 course notes by Avi Pfeffer and David Parkes.

utility. Decision theory provides a precise and concrete formulation of the problem we wish to solve and a method for designing an intelligent agent.

1.1 Problem Formulation

The reason probability is central to decision theory is because the world is a noisy and uncertain place; rational agents need to make decisions even when it is unknown what exactly will happen. There are various sources of uncertainty that an agent faces when trying to achieve good performance in complex environments:

- The agent may not know the current state of the world. There can be a number of reasons for this:
  - Its sensors only give it partial information about the state of the world. For example, an agent using a video camera cannot see through walls.
  - Its sensors may be noisy. For example, if the camera is facing the sun, it may incorrectly perceive objects due to lighting effects.
- The effects of the agent's actions might be unpredictable. For example, if the agent tries to pick up a plant and eat it, it might accidentally crush it instead.
- There are exogenous events – things completely outside the agent's control – that it might have to deal with. For example, if it has planned a path from one room to another going through a door, someone might shut the door before it gets through.
- Over and above the agent's uncertainty in a particular situation, we might also have uncertainty about the correct model of the world.

In the decision theoretic framework, all these different kinds of uncertainty can be modeled using probabilities. For the most part we will ignore the last kind of uncertainty, uncertainty over the model itself. However, this can be handled through the Bayesian approach that we know from our earlier discussions of learning, in which we use the marginal likelihood to perform model selection. For now we will also be studying problems in which the first problem is ignored and the agent knows the current state. This will put us in the space of *Markov decision processes* (MDPs).

Continuing our elaboration of the problem definition, "performs well" means that an agent gets good utility. Of course, because of uncertainty, we can't devise a policy that guarantees that it will get good utility in all circumstances. Instead, we need to try to design an agent so that it gets good utility on average, i.e., in expectation. We can make this notion a little more precise. For now, we will focus on the "open-loop" setting, in which the agent has to make a single decision and then it is done. We will soon get to MDPs and the "closed-loop" setting, in which the agent repeatedly interacts with is environment.

1.2 What are Utilities?

We've been blithely throwing around the term "utility" without stopping to think what exactly it means. The simplest answer is that utility is a measure of happiness. But what does that mean? What is a happiness of 1, 3.27, or -15?

Can we measure utility by money? This works well for small gains and losses, but is not necessarily appropriate in general. Consider the following game: you have 99% chance of losing \$10,000, and 1% chance of winning \$1,000,000. Many people would be averse to playing this game, but the expectation of the game is to win \$100. This seems to indicate that having \$1,000,000 is less than 99 times as good as losing \$10,000 is bad, so money is not an ideal measure of utility.

Rather, utility is an encoding of preferences. Given a choice between outcomes A, B and C, you might rank them as A being better than B which is better than C. But by how much do you prefer A to B and B to C? If you had to choose between two *lotteries*, one that gets you B for sure, while the other gets you 50% chance of A and 50% chance of C, which would you choose? We denote such a lottery $L = [0.5 : A,\ 0.5 : C]$, and write $L \succ L'$ if you prefer lottery L over L', and $L \sim L'$ if you are indifferent.

The foundation of utility theory is based on the following idea. I can ask you the same type of question about all sorts of different lotteries. If you're rational, your answers to these questions should satisfy certain fundamental properties. These axioms are:

- **Orderability**: For every pair of lotteries L and L', then $L \succ L'$ or $L' \succ L$ or $L \sim L'$.
- **Substitutability**: If you are indifferent between two lotteries L and L', then you should be indifferent between two more complex lotteries that are the same except that L' replaces L in one part.
- **Transitivity**: Given any three lotteries, if you prefer L to L' and L' to L'' then you should prefer L to L''.
- **Continuity**: If L' is between L and L'' in preference ordering, then there is some probability p on a mix of L and L'' for which you are indifferent between this new lottery and L'.
- **Monotonicity**: If you prefer L to L' then you should prefer a new lottery that provides L with probability p and L' with probability $1 - p$ than one that provides L with probability $q < p$ and L' with probability $1 - q$.
- **Decomposability**: Compound lotteries can be made equivalent to simpler lotteries, e.g.,

$$[p : A,\ 1 - p : [q : B,\ 1 - q : C]] \sim [p : A,\ (1 - p)q : B,\ (1 - p)(1 - q) : C]\,.$$

Von Neumann and Morgenstern established that if your preferences satisfy these axioms, then there is some function $U(\cdot)$ that can be assigned to outcomes, such that you prefer one lottery to another if and only if the expectation of $U(\cdot)$ under the first lottery is greater than the expectation under the second. *This function $U(\cdot)$ is your utility function.* The expected utility of a

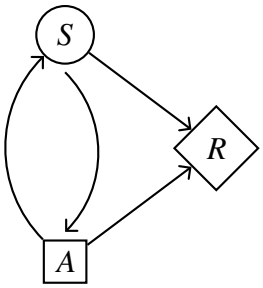

Figure 1: Example of agent interaction. S is the current state, A is its action, and R is the reward.

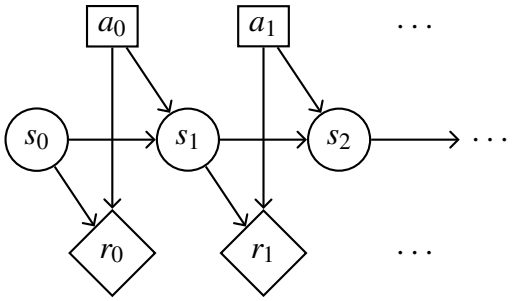

Figure 2: Two steps of a Markov decision process. The agent starts in state s_0, then takes action a_0, receiving reward r_0 and transitioning to state s_1. From state s_1, the agent takes action a_1, receives reward r_1 and transitions to state s_2.

lottery $L = [p : A,\ 1 - p : B]$ is just $p \times U(A) + (1 - p) \times U(B)$. That is, we have that there exists a utility function $U(\cdot)$ such that

$$L \succ L' \Leftrightarrow \mathbb{E}[U(L)] > \mathbb{E}[U(L')]\,.$$

An interesting point to note is that the function $U(\cdot)$ encoding preferences is not unique! In fact, it can be shown that if $U(\cdot)$ encodes an agent's preferences, then so does $V = aU + b$ where $a > 0$, and b is arbitrary.

The moral of this is that the utility numbers themselves don't mean anything. One can't say, for example, that positive utility means you're happy, and negative utility means you're unhappy, because the utility numbers could just be shifted in the positive or negative direction. All utilities achieve is a way of encoding the relative preferences between different outcomes.

2 Markov Decision Processes

The basic framework for studying utility-maximizing agents that have repeated interactions with the world and take a sequence of actions is called a *Markov Decision Process*, abbreviated MDP. We can illustrate the agent's interaction with the environment via Figure 1, where S is the current state of the world, observed by the agent for the moment, A is its action (drawn as a box because it is a decision variable rather than a random variable) and R is its reward or utility, drawn as a diamond. We are interested in "closed loop" problems in which the agent affects the state of the environment through its action. This is contrasted with "open loop" problems in which only a single action is taken and any effect on the subsequent state is irrelevant.

The MDP framework consists of the following elements $(\mathcal{S}, \mathcal{A}, R, P)$:

- A set $\mathcal{S} = \{1, \ldots, N\}$ of possible states.
- A set $\mathcal{A} = \{1, \ldots, M\}$ of possible actions.
- A *reward model* $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, assigning reward to state tuples of state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$.

- A *transition model* $P(s' \mid s, a)$, where $P(s' \mid s, a) \geq 0$ and $\sum_{s' \in \mathcal{S}} P(s' \mid s, a) = 1$ for all s, a, and defining the probability of reaching state s' in the next period given action a in state s. This is a probability mass function over the next state, given the current state and action.

This framework describes an agent with repeated interactions with the world. The agent starts in some state $s_0 \in \mathcal{S}$. The agent then gets to choose an action $a_0 \in \mathcal{A}$. The agent receives reward $R(s_0, a_0)$, while the world transitions to a new state s_1, according to the probability distribution $P(s_1 \mid s_0, a_0)$. Then the cycle continues: the agent chooses another action $a_1 \in \mathcal{A}$, and the world transitions to s_2 according to the distribution $P(s_2 \mid s_1, a_1)$, the agent receiving $R(s_1, a_1)$, and so on.

An agent in an MDP goes through $s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, s_3, \ldots$ where s_t, a_t, r_t indicate the state, action and reward in period t. In writing this model, we have adopted a stationary and deterministic (non-random) reward model $R(s, a)$. Taking the same action in the same state provides the same reward irrespective of time. Furthermore, we have adopted a stationary transition model, so that $P(s_{t+1} \mid s_t, a)$ does not depend on t. This can be relaxed as necessary by folding some temporal features into the state.

The framework makes a fundamental assumption: that the reward model and transition model depend only on the current state and current action, and not on previous history. This assumption is known as the *Markov* assumption (hence the name Markov Decision Process), which is a basic assumption used in reasoning about temporal models. The Markov assumption is often stated as "The future is independent of the past given the present."

The general idea of the MDP is shown in as a *decision network* in Figure 2. Actions are again in boxes, random variables (the state) in circles as in Bayesian networks (which you might encounter in later machine learning and statistics courses), and rewards in diamonds. This decision network uses the same conditional independence semantics for arrows that are used for Bayesian networks, although the details of these structures are outside the scope of this course. The agent's proceeds through time periods $t \in \{0, 1, \ldots\}$ and perhaps for an infinite sequence of periods.

Another component that we require in order to solve MDPs is the concept of a *policy*,

$$\pi_t : \mathcal{S} \to \mathcal{A}$$

which produces an action $a \in \mathcal{A}$ in every possible state $s \in \mathcal{S}$, here allowing the policy to also depend on time t. As we'll see, dependence on time is required in settings with a finite decision horizon because as the deadline approaches then different actions become useful. A policy describes the solution to an MDP: it is a complete description of how an agent will act in all possible states of the world.

The assumptions made in the MDP formalism are:

- **Fully Observable**: The state s_t in period t is known to the agent. This is relaxed in moving to partially observable MDPs.

- **Known Model**: The $(\mathcal{S}, \mathcal{A}, R, P)$ are known to the agent. This is relaxed in reinforcement learning, where $\mathcal{S}$ and $\mathcal{A}$ are known, but R and P are learned as the agent acts.

- **Markov Property**: The future is independent of the past, given the present.

- **Finite State and Action Spaces**: $\mathcal{S}$ and $\mathcal{A}$ are both finite sets.
- **Bounded Reward**: There is some $R_{\max}$ such that $R(s, a) \leq R_{\max}$ for all s, a.

The MDP formulation above is general. For example, it is sometimes convenient to describe the reward model as $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, which includes both the current and *next* state in the reward computation. However, this is equivalent to $R(s, a) = \sum_{s' \in \mathcal{S}} P(s' \mid s, a) R(s, a, s')$. Similarly, it is sometimes simpler to associate rewards with states, rather than with states and actions. When we do that, we will mean that the reward for taking *any* action in the state is the reward associated with the state.

The MDP framework describes how the agent interacts with the world, but we haven't yet described what the goals of the agent actually are, i.e., what problem we are trying to solve. Here are some variants of the problem:

Finite Horizon We are given a number T, called the *horizon*, and assume that the agent is only interested in maximizing the reward accumulated in the first T steps. In a sense, T represents the degree to which the agent looks ahead. $T = 1$ corresponds to a greedy agent that is only trying to maximize its next reward. We have

$$\textit{Utility} = \sum_{t=0}^{T-1} R(s_t, a_t) \tag{1}$$

Total Reward The agent is interested in maximizing its reward from now until eternity. This makes most sense if we know the agent will die at some point, but we don't have an upper bound on how long it will take for it to die. If it is possible for the agent to go on living forever, its total reward might be unbounded, so the problem of maximizing the total reward is ill-posed. We have

$$\textit{Utility} = \sum_{t=0}^{\infty} R(s_t, a_t) \tag{2}$$

Total Discounted Reward A standard way to deal with agents that might live forever is to say that they are interested in maximizing their total reward, but future rewards are *discounted*. We are given a discount factor $\gamma \in [0, 1)$. The reward received at time t is discounted by γ^t, so the total discounted reward is

$$\textit{Utility} = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \tag{3}$$

Provided $0 \leq \gamma < 1$ and the rewards are bounded, this sum is guaranteed to converge.[1] Discounting can be motivated either by assumption catastrophic failure can occur with some small probability $\epsilon > 0$ in every period, by acknowledging that the model $(\mathcal{S}, \mathcal{A}, R, P)$ may not be accurate all the way into the future, or by inflation or opportunity cost arguments that it is better to receive reward now because it can be used for something.

[1] For example, if a constant reward is received in every period t, then the discounted sum is $\frac{t}{1-\gamma}$.

Long-run Average Reward Another way to deal with the problems with the total reward formulation, while still considering an agent's reward arbitrarily far into the future, is to look at the *average reward* per unit time over the (possibly infinite) lifespan of the agent. This is defined as

$$Utility = \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} R(s_t, a_t) \tag{4}$$

While this formulation is theoretically appealing, and bounded, the limit in the definition makes it mathematically intractable in many cases.

In this course, we will focus on the finite horizon and total discounted reward cases (which we will simply call the *infinite horizon* case).

3 Examples

Let's look at some example MDPs.

3.1 Auction Bidding

For the first example, suppose you are participating in an internet auction. We'll use a somewhat artificial model to keep things simple. The bidding starts at \$100, and in each round you get a chance to bid. If your bid is successful, you will be recorded as the current bidder of record, and the asking price will go up \$100. Someone else might also bid, but only one successful bid is accepted on each round. The bidding closes if a bid of \$200 is received, or no bid is received for two successive rounds. You value the item being sold at \$150. If you do not buy the item, you receive 0 reward. If you buy it for $\$x$, you receive a reward of $150 - x$.

Formally, we specify the problem as follows:

- The state space consists of tuples $\langle x, y, z \rangle$, where x is the current highest bid, y specifies whether or not you are the current bidder, and z is the number of rounds since the last bid was received. The possible values for x are \$0, \$100, and \$200. Whether or not you are the current bidder is a binary variable. The number of rounds since the last bid was received is 0, 1 or 2. So the total number of states is $3 \times 2 \times 3 = 18$.

- There are two possible actions: to bid or not to bid, denote b and $\neg b$.

- Since you only get a non-zero reward when the auction terminates and you have bought the item, the reward function is defined as follows. While the action is normally an argument to the reward function, it has no effect on the reward in this case, so we drop it.

$$R(\langle x, y, z \rangle) = \begin{cases} 150 - x & \text{if } y = \textit{true} \text{ and } z = 2 \text{ or } x = 200 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

- The transition model is specified as follows. States where $x = \$200$ or $z = 2$ are terminal states — the auction ends at those states, and there is no further action. We only need to specify what happens if $x < \$200$ and $z < 2$. Let's say that if you do not bid, there is a 50% probability that someone else will successfully bid. If you do bid, your bid will be successful 70% of the time, but 30% of the time, someone else will beat you to the punch.

  Formally, we define the transition model as follows.

$$P(\langle x + 100, \mathit{false}, 0\rangle \mid \langle x, y, z\rangle, \neg b) = 0.5 \quad \text{[You don't bid. Someone else does.]}$$
$$P(\langle x, y, z + 1\rangle \mid \langle x, y, z\rangle, \neg b) = 0.5 \quad \text{[You don't bid. Nobody else does either.]}$$
$$P(\langle x + 100, \mathit{true}, 0\rangle \mid \langle x, y, z\rangle, b) = 0.7 \quad \text{[You bid and are current to bidder.]}$$
$$P(\langle x + 100, \mathit{false}, 0\rangle \mid \langle x, y, z\rangle, b) = 0.3 \quad \text{[You bid and get beaten to the punch.]}$$

3.2 Robot Navigation

Another common example of an MDP is robot navigation. Suppose that your robot is navigating on a grid. It has a goal to get to, and perhaps some dangerous spots like stairwells that it needs to avoid. Unfortunately, because of slippage problems, its operators are not deterministic, and it needs to take that into account when planning a path to the goal.

Here, the state space consists of possible locations of the robot, and the direction in which it is currently facing. The total number of states is four times the number of locations, which is quite manageable. The actions might be to move forward, turn left or turn right. The reward model gives the agent a reward if it gets to the goal, and a punishment if it falls down a stairwell. The transition model states that in most cases actions have their expected effect, e.g., moving forward will normally successfully move the robot forward one space, but with some probability the robot will stick in the same place, and with some probability it will move forward two spaces.

4 Expectimax Search: Finite Horizon

Given an MDP, the task is to find a policy that maximizes the agent's utility. How do we find an optimal policy? Let's concentrate on the finite horizon case first.

One approach is to view an MDP as a *game against nature* where an opponent called "nature" behaves in a specific probabilistic manner. One way to solve this problem is to build a search tree. The tree will have two kinds of nodes: those where you get to move, and those where nature moves. Nodes where you get to move are associated with the current state s; nodes where nature moves are associated with the pair (s, a), where s is the current state and a is your chosen action. The root of the search tree is the initial state s_0. If you choose action a at node s, then the we transition to tree node (s, a). From node (s, a), then we can transition to any tree node s' such that $P(s' \mid s, a) > 0$. The edge from (s, a) to s' is annotated with probability $P(s' \mid s, a)$. This models the move by nature, which is probabilistic and could take the state in the world to any one of the possible next states.

How do we adopt search to solve and find the optimal decision policy? In *expectimax search* we will solve from the end of time to the start of time, propagating a *value* for every tree node up

Algorithm 1 Expectimax Search

function EXPECTIMAX(s) ▷ Takes a state as an input.
  if s is terminal **then**
    Return 0
  else
    for $a \in \mathcal{A}$ **do** ▷ Look at all possible actions.
      $Q(s,a) \leftarrow R(s,a) + \sum_{s' \in \mathcal{S}} P(s' \mid s,a)\,\text{EXPECTIMAX}(s')$ ▷ Compute expected value.
    end for
    $\pi^\star(s) \leftarrow \arg\max_{a \in \mathcal{A}} Q(s,a)$ ▷ Optimal policy is value-maximizing action.
    Return $Q(s, \pi^\star(s))$
  end if
end function

from the leaves to the root. The value of a tree node is simply the expected total reward that an agent can expect to get forward from that node *under its optimal policy.* The name corresponds to the fact that the algorithm alternates between taking expectations and maximizations. Algorithm 1 shows the recursion.

We adopt $Q(s,a)$ to denote the value at a tree node (s,a), just before Nature is about to take an action and determine the next state. This is immediate reward for action a in state s plus the expectation over the values of the children s' of (s,a), where the expectation is taken according to the transition model. The algorithm works from the leaves towards the root, calling EXPECTIMAX on each new state. For a given state it considers the possible actions, determining the $Q(s,a)$ value based on the work done at the children of the state. The optimal action $\pi^\star(s)$ is then stored, and the Q-value under that optimal action returned as the expectimax value for the state.

Figure 3 shows the expectimax tree for the Internet auction example. The left subtree, corresponding to the initial action of bid, has been expanded completely and analyzed. The right subtree has only been partially expanded and dashed edges show the redundant nodes that also appear on the left subtree. Nature's edges are annotated with probabilities, while nodes whose values have been computed are annotated with the value. From the analysis, we see that the auction is worth at least \$8.75 to the agent. The agent can use the policy of bidding right away, hoping that the bid succeeds, and hoping that no-one bids for the next two rounds. As it happens, the agent cannot do as well by passing in the first round. If the agent passes, it will hope that no-one else bids, and then have to bid immediately in the next round, hoping its bid will be successful. It will then have to hope that no-one bids for the next two rounds. This policy has smaller chance of succeeding than bidding in the first round, because there is a 50% chance that someone else bids after the agent's first pass.

How expensive is the expectimax algorithm? It is proportional to the size of the tree, which is exponential in the horizon. What is the base of the exponent? The number of actions times the number of possible transitions for nature. To be precise, let the horizon be T, the number of actions be M, and the maximum number of transitions for any state and action be L. Then the cost of the algorithm is $O((ML)^T)$. How large is L? It depends on the model. L is the maximum number of non-zero entries in any row of the transition matrix for any action. In general, if the

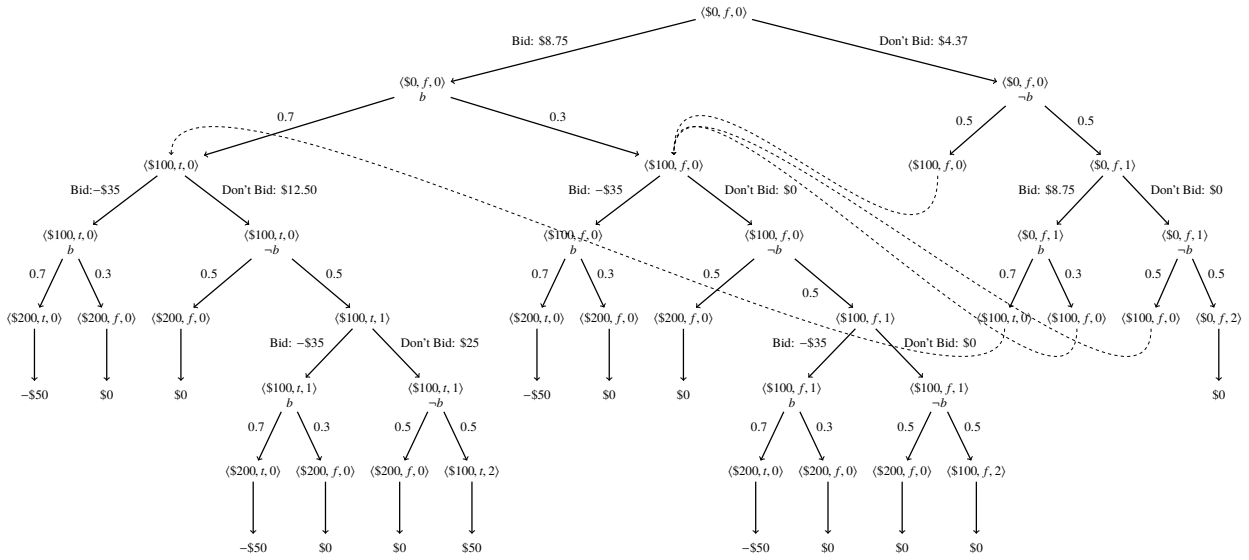

Figure 3: This figure shows the game tree, with shared structured shown via dashed arrows. Paths down the tree alternate between agent actions (Bid vs. Don't Bid) and Nature actions, i.e., probabilistic state transitions. The leaves on the tree are outcomes of the auction when we value the item at $150.

matrices are sparse, the branching factor will be small and the algorithm will be more efficient. As an alternative, we will turn to value iteration, which uses dynamic programming.

5 Value Iteration: Finite Horizon

Notice that in expectimax the same state may be reached by many different paths, a property we use to make the tree fit into Figure 3. This is true even in deterministic settings, but is especially true in probabilistic settings – Nature tends to act as an equalizer, negating the results of your actions, especially actions in the distant past. If there are states that can be reached in multiple ways, there's a tremendous amount of wasted work in the expectimax algorithm, because the entire tree beneath the repeated states is searched multiple times.

An alternative approach is to work backwards from the decision horizon but memorize the optimal thing to do from each possible state reached and reuse this computation. This is an approach of *dynamic programming*. It recognizes that the optimal policy forward from state s with k periods to go is the same irrespective of the way s is reached, and moreover can be found by taking the best subsequent action and then following the optimal k-to-go policy from the next state reached.

To flesh this out, we will denote by $V_k(s)$ the k-step-to-go value of state s. That is, the value that you can expect to get if you start in state s and proceed for k steps *under the optimal policy*. So, $V_0(s)$ is just 0 for all s. Similarly, let $Q_k(s,a)$ denote the k-step-to-go value of taking a now and then following by the optimal policy. Finally, let $\pi_k^\star(s)$ denote the optimal k-to-go action.

Algorithm 2 Value Iteration

function ValueIteration(T) ▷ Takes a horizon as input.
  for $s \in \mathcal{S}$ **do** ▷ Loop over each possible ending state.
    $V_0(s) \leftarrow 0$ ▷ Horizon states have no value.
  end for
  for $k \leftarrow 1 \ldots T$ **do** ▷ Loop backwards over time.
    for $s \in \mathcal{S}$ **do** ▷ Loop over possible states with k steps to go.
      for $a \in \mathcal{A}$ **do** ▷ Loop over possible actions.
        $Q_k(s,a) \leftarrow R(s,a) + \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V_{k-1}(s')$ ▷ Compute Q-function for k.
      end for
      $\pi_k^\star(s) \leftarrow \arg\max_{a \in \mathcal{A}} Q_k(s,a)$ ▷ Find best action with k to go in state s.
      $V_k(s) \leftarrow Q_k(s, \pi_k^\star(s))$ ▷ Compute value for state s with k steps to go.
    end for
  end for
end function

The base case is

$$V_0(s) = 0, \quad \forall s \tag{6}$$

The inductive case is

$$Q_k(s,a) = R(s,a) + \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V_{k-1}(s') \tag{7}$$

$$\pi_k^\star(s) = \arg\max_{a \in \mathcal{A}} Q_k(s,a) \tag{8}$$

$$V_k(s) = Q_k(s, \pi_k^\star(s)) \tag{9}$$

Look carefully at Equation 7. This holds that the value of each action a taken now is the immediate reward plus the expected continuation value from the next state s' reached, given the $V_{k-1}(s')$ values already computed. Also, note that $\pi_k^\star(s)$ is the optimal action with k steps to go and not the optimal action in time period k.

This suggests an algorithm, where we solve this series of equations layer by layer, from the bottom up. The algorithm, called *value iteration*, is as follows:

What is the meaning of $\pi_k^\star(\cdot)$ and $V_k(\cdot)$? The function $\pi_k^\star(\cdot)$ is the optimal policy for the kth step before the end, assuming you have only k steps to go. The function $V_k(\cdot)$ is the value you get in the last k steps, assuming you play optimally. As k grows larger, the nature of the optimal policy changes. For $k = 1$, the policy will be greedy, considering only the reward in the last step. For small k, only the next few rewards will be considered, and there will be no long-term planning. For large k, there is incentive to sacrifice short-term gain for long run benefit, because there is plenty of time to reap the reward of your investment.

What is the complexity of finite horizon value iteration? If the number of states is N, the number of actions is M, and the horizon is T, then there are NM Q-values to be updated every

time through the loop, each Q-value update is L work for the s' next states that occur with non-zero probability, and there are T iterations of the loop. Altogether, we have $O(NMLT)$, which is *linear* in the horizon, instead of exponential as we found for expectimax! Great.

However, expectimax still has one advantage over dynamic programming: it only has to compute values for states that are actually reached. If states can generally not be reached in multiple ways, and there are many states that are not reached at all, expectimax may be better. There is a simple algorithm that combines the advantages of both, using an idea called *reachability analysis*. The first stage is simply to determine which of the possible states are reachable. Then, value iteration is performed over this reduced set of states.

Changelog

- 23 November 2018 – Initial version converted from Harvard CS181 notes.