Chapter 4

Tuesday, January 8, 2019 4:54 PM

Dynamic Programming:
It refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as Markov Decision Process (MDP).

Classical DP Algorithms are limited in RL because of the following reasons:

1. Assumption of a perfect model and
2. Great Computational Expense

The environment is assumed as a finite MDP. So, state, action and reward sets, S, A, and R, are finite. Then dynamics are given by a set of probabilities p(s', r | s, a), for all s ∈ S, a ∈ A(s), r ∈ R and s' ∈ S^+. (S^+ is S plus a terminal state if the problem is episodic).

Application of the DP for continuous methods:
Common way of obtaining approximate solutions for tasks with continuous states and actions is to quantize the state and action spaces and then apply finite-state DP methods.

Value functions are used to organize and structure the search for good policies. Optimal policies can be easily found out once the optimal value functions can be found, v_* or q_*, which satisfy the Bellman optimality equations.

$$\begin{aligned}
v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_*(s')\Big], \text{ or} \\
q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1},a') \;\Big|\; S_t = s, A_t = a\Big] \\
&= \sum_{s',r} p(s',r|s,a)\Big[r + \gamma \max_{a'} q_*(s',a')\Big],
\end{aligned}$$

for all s ∈ S, a ∈ A(s) and s' ∈ S^+. Bellman equations are turned into assignments, into update rules for improving approximations of the desired value functions.

4.1. Policy Evaluation (Prediction)
The aim of "Policy evaluation" is to compute the state-value function v_Π for an arbitrary policy Π. This is called as *the prediction problem*.

The value function of a policy 'Π' is given using the formula below. The subscripts of (Π) in the formula states that the values are obtained when the actions are chosen based on the policy - Π.

$$\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big],
\end{aligned}$$

Π(a | s) defines the probability of taking an action 'a' in state 's' under policy 'Π'.

Iterative Solution methods to find the value function:
A sequence of approximate value functions v_0,v_1, v_2, …. and each mapping from S^+ to R. The initial approximation v_0 is chosen randomly, value of 0 should be given for the terminal state, then each

successive approximation is obtained by using the Bellman Equation for v_Π.

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_k(s')\Big] \end{aligned}$$

for all s ∈ S.

Iterative Policy Evaluation:
Iterative policy evaluation is used to produce each successive approximation, i.e. v_{k+1} from v_k. So in each iteration the method replaces the old value of 's' with a new value obtained from the old values of the successor states of 's' and the expected immediate rewards, along all the one-step transitions possible under the policy being evaluated. The sequence $\{v_k\}$ in general converge to v_Π as k --> ∞. This is called as "expected update", since they are based on an expectation over all possible next states rather than on a sample next state.

<u>Sequential program with two arrays:</u>
Iterative policy evaluation can be done using two arrays and sequential program. One array is for the old values, $v_k(s)$ and another array for $v_{k+1}(s)$. So, the new values can be computed one by one from the old values without changing the old values.

<u>One Array approach:</u>
It is possible and easy to use one array and update the values "in place", so with each new value immediately overwriting the old one. So in this approach depending on the order in which the states are updated, sometimes new values are used instead of old ones on the right hand side of the bellman equation. So in-place algorithm converges to v_Π more quickly since the new values are used as soon as they were available for the update.

Updates are being done in a sweep through the state space. The order of update plays a prominent role in the in-place update algorithm's convergence rate towards the v_Π .

The following pic gives the pseudo code for the Iterative Policy Evaluation to estimate $V \approx v_\Pi$.

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

```
Input $\pi$, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
    $\Delta \leftarrow 0$
    Loop for each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$
```

4.2. Policy Improvement:
Reason for computing the value function for a policy is to help find the better policies.
So once we determined the value function v_Π of an *arbitrary deterministic* policy, we would like to know whether or not we should change the policy to deterministically choose an action a ≠ Π(s).
To know if the new policy is better or not from the previous policy.

One way to answer this is to choose the 'a' in 's' according to new policy and thereafter following the existing policy (Π).
The value obtained by behaving like this can be formulated as below:

$$\begin{aligned} q_\pi(s,a) &\doteq \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big]. \end{aligned}$$

The criteria is whether this is greater than or less than $v_\Pi(s)$. If the value is greater, then every time the agent encounters the state 's' the action 'a' can be chosen and this policy is better one overall.

This is a special case of a general result called "Policy Improvement".

So if Π and Π' be any pair of deterministic policies which follows the following relationship for all s ∈ S

$$q_\pi(s, \pi'(s)) \geq v_\pi(s)$$

This denotes that the policy Π' is better than or as good as Π.

So this leads to the following relationship which holds for all the states s ∈ S.

$$v_{\pi'}(s) \geq v_\pi(s)$$

So if $q_{\Pi'}(s,a) > v_\Pi(s)$ then the policy Π' is better than Π.

The following picture shows the proof behind the policy improvement theorem .

$$\begin{aligned} v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\ &= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(s)] \qquad (\\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \qquad (\\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})] \mid S_t = s] \\ &= \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \mid S_t = s\big] \\ &\leq \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \mid S_t = s\big] \\ &\vdots \\ &\leq \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \mid S_t = s\big] \\ &= v_{\pi'}(s). \end{aligned}$$

It is a natural extension to consider changes at all states and to all possible actions, selecting at each state the action that appears best according to $q_\Pi(s,a)$ i.e. to consider the new greedy policy, Π' given by the following formula.

$$\begin{aligned} \pi'(s) &\doteq \underset{a}{\arg\max}\, q_\pi(s,a) \\ &= \underset{a}{\arg\max}\, \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \underset{a}{\arg\max} \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big], \end{aligned}$$

where argmax_a denotes the value of a at which the expression that follows is maximized (with ties broken arbitrarily).

The greedy policy takes the action that looks best in the short term—after one step of lookahead—

according to v_Π.

The process of making a new policy that improves on an original policy by making it greedy with respect to the value function of the original policy , is called "policy improvement".

Suppose the new greedy policy Π' is as good as, but not better than the previous policy Π. Then $v_\Pi = v_{\Pi'}$ for all the states s ∈ S.

$$\begin{aligned} v_{\pi'}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_{\pi'}(s')\Big]. \end{aligned}$$

This is same as the Bellman Optimality Equation. Therefore v_Π must be v_*, and both Π and Π' must be optimal policies.
So the policy improvement must strictly give better policies except when the original policy is already optimal.

So far the discussions were about the *deterministic policies*.

Policy Improvement for Stochastic Policies:
If we consider a stochastic policy Π, it specifies the probabilities , $\Pi(a \mid s)$ for taking each action 'a' in state 's'. The policy improvement theorem stated before also holds for the stochastic case. If there are ties in the policy improvement steps that is if there are several actions at which the maximum is achieved, in stochastic case, there is no need to select a single action from them, instead each maximizing action can be given a portion of the probability being selected in the new greedy policy. So any apportioning scheme is allowed as long as all the suboptimal actions are given zero probability.

4.3. Policy Iteration
A policy Π, has been improved using v_Π to yield a better policy, Π', then $v_{\Pi'}$ is computed and improve it again to yield an even better Π''.
The sequence of monotonically improving policies and value functions is obtained:

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$

where $\xrightarrow{E}$ denotes a *policy evaluation* and $\xrightarrow{I}$ denotes a *policy improvement*. Each policy is guaranteed to be a strict improvement over the previous one (unless it is already optimal).
Since finite MDP has only a finite number of policies, this process will converge to optimal policy and optimal value function in finite number of iterations. This way of finding an optimal policy is called as *policy iteration*.

The complete algorithm for the policy iteration is shown below.
Each policy evaluation itself is an iterative computation, which started with the value of the previous policy.

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

```
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
       $\Delta \leftarrow 0$
       Loop for each $s \in \mathcal{S}$:
           $v \leftarrow V(s)$
           $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\big[r + \gamma V(s')\big]$
           $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   $policy\text{-}stable \leftarrow true$
   For each $s \in \mathcal{S}$:
       $old\text{-}action \leftarrow \pi(s)$
       $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
       If $old\text{-}action \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
   If $policy\text{-}stable$, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2
```

4.4. Value Iteration

Drawback to policy iteration is each of its iterations involves policy evaluation, which itself is a protracted iterative computation that require multiple sweeps through the state set. Iterative policy evaluation converge exactly to v_Π only in the limit.

The policy evaluation can be truncated in several ways without losing the convergence guarantees of policy iteration.

Value Iteration:

It is a special case of truncating the policy evaluation step. In Value Iteration, the policy evaluation is stopped after just one sweep, which is one update of each state. The following figure explains about the value iteration formula.

$$\begin{aligned} v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_k(s')\Big], \end{aligned}$$

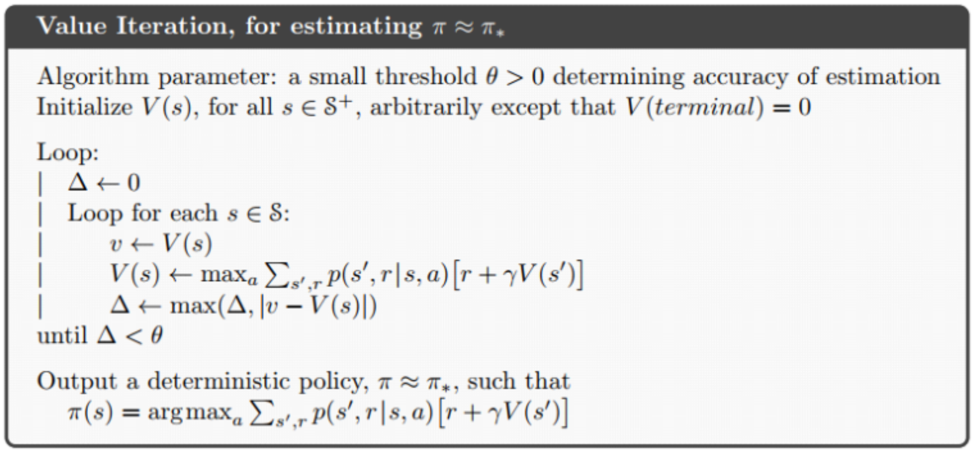
for all s ∈ S.

For arbitrary v_0, the sequence $\{v_k\}$ can be shown to converge to v_* under the same conditions that guarantee the existence of v_*.

Value iteration is obtained simply by turning the Bellman Optimality Equation into an update rule. Value iteration update is identical to the policy evaluation update except that it requires the maximum to be taken over all actions.

Termination of value iteration:

Like policy iteration the value iteration also requires an infinite number if iterations to converge exactly to v_*. The iteration is stopped once the value function changes only by a small amount in sweep. The following pic defines about the value iteration algorithm.



Value Iteration, for estimating $\pi \approx \pi_*$

```
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
|   $\Delta \leftarrow 0$
|   Loop for each $s \in \mathcal{S}$:
|       $v \leftarrow V(s)$
|       $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
|       $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
    $\pi(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
```

Value iteration effectively combines, in each of its sweeps, one sweep of policy evaluation and one sweep of policy improvement.
Faster convergence can be achieved by doing multiple policy evaluation sweeps between each policy improvement sweep.

In general, the entire class of truncated policy iteration algorithms can be thought of as sequences of sweeps, some of which use policy evaluation updates and some of which use value iteration updates.

The only difference between the policy iteration and the value iteration is that the max operation is added to some sweeps of policy evaluation.

All of these algorithms converge to an optimal policy for discounted finite MDPs.

4.5. Asynchronous Dynamic Programming
The methods that have discussed before have a major drawback because they involve operations over the entire state set of the MDP, i.e., they require sweeps of the state set.

If the state set is very large, then even a single sweep can be prohibitively expensive.

For example the game of backgammon has 10^{20} states and even value iteration update on a million state per second can be carried out, it would take over a thousand years to complete a single sweep.

Asynchronous DP algorithms are in-place iterative DP algorithms that are not organized in terms of systematic sweeps of the state set. They update the values of states in any order, using whatever the values of other states available. Value of some states may be updated several times before the values of others are updated once.
To converge correctly, an asynchronous algorithm must continue to update the values of all the states, it can't ignore any state after some point in the computation.
These algorithms offer flexibility in selecting the state to update.

It is possible to intermix policy evaluation and value iteration updates to produce a kind of asynchronous truncated policy iteration.

We can take advantage of this flexibility by selecting the states to which we apply updates so as to improve the algorithm's rate of progress.
We can also order the updates to let value information propagate from state to state in an efficient way. Some states may not need their values updated as often as others. Some states can be skipped entirely if they are not relevant to optimal behavior.

Asynchronous algorithms also make it easier to intermix computation with real-time interaction. An

iterative DP can be run at the same time that an agent is actually experiencing the MDP. By using the agent's experience the states to which the DP algorithm applies updates can be determined.
The same time the latest value and policy information from the DP algorithm can be used to guide the agent's decision making.

For example:
The updates can be applied to the states as the agent visits them. This makes the DP algorithm to focus the updates only on the parts of the state set that are most relevant to the agent.

4.6. Generalized Policy Iteration
Two simultaneous interacting processes in policy iteration:
1. Policy Evaluation (making the value function consistent with the current policy)
2. Policy Improvement (making the current policy greedy with respect to current value function)

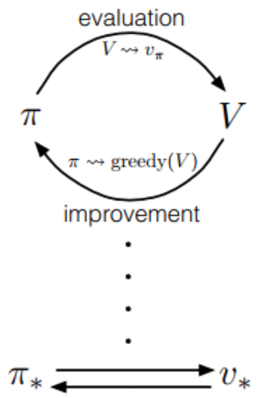
In policy iteration these two processes alternate, each completing before the other begins, but it is not necessary.

In value iteration, only single iteration of policy evaluation is performed in between each policy improvement.

In asynchronous DP, these processes are interleaved at even finer grain.

The term Generalized Policy Iteration (GPI) can be used to refer to the general idea of letting the policy evaluation and policy-improvement processes interact, independent of the granularity and other details of the two processes.

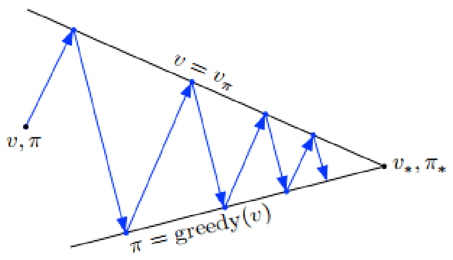
All of the algorithms described above have identifiable policies and value functions, with the policy always being improved with respect to the value functions and the value function always being driven toward the value function for the policy. This is represented by the diagram below:


evaluation
$V \rightsquigarrow v_\pi$
π
V
$\pi \rightsquigarrow \text{greedy}(V)$
improvement
π_*
v_*

The value function stabilizes only when it is consistent with the current policy, and the policy stabilizes only when it is greedy with respect to the current value function.
Both processes stabilize only when a policy has been found that is greedy with respect to its own evaluation function. This implies that the Bellman optimality equation can be applied and thus the policy and the value function are optimal.

The evaluation and improvement processes in GPI can be viewed as both competing and cooperating. Making the policy with respect to the value function typically makes the value function incorrect for the change policy and making the value functions consistent with the policy typically causes that policy no longer to be greedy. In the long run, these two processes interact to find a single joint solution: the optimal value function and an optimal policy.

The diagram below suggests what happens in the real case. Each process drives the value function or policy toward one of the lines representing a solution to one of the two goals. Driving towards one goal causes movement away from the other goal. Inevitably, the joint process is brought closer to the overall goal of optimality. The arrows in the diagram correspond to the behavior of policy iteration in that each takes the system all the way to achieving one of the two goals completely. In GPI, it is also possible to take smaller, incomplete steps toward each goal. In either case, the two processes together achieve the overall goal of optimality even though neither is attempting to achieve it directly.



4.7 Efficiency of Dynamic Programming
DP may not be practical for very large problems but they are actually quite efficient in solving MDPs.

The worst case time DP methods take to find an optimal policy is polynomial in the number of states and actions. If 'n' and 'k' denote the number of states and actions, this means that a DP methods takes a number of computational operations that is less than some polynomial function of 'n' and 'k'.

DP method is guaranteed to find an optimal policy in polynomial time even though the total number of (deterministic) policies is k^n.

DP is limited in applications because of "the curse of dimensionality". The number of states often grow exponentially with the number of state variables.

Large state sets create difficulties, these are inherent difficulties of the problems and not of DP as a solution method.

DP is better suited to handling large state spaces than competing methods such as direct search and linear programming.

DP methods can be used with today's computers to solve MDPs with million states. These methods usually converge much faster than their theoretical worst-case run times, particularly when they are started with good initial value functions or policies.

Asynchronous DP methods are used to solve the problems with large state spaces. To complete even one sweep of a synchronous method requires computation and memory for every state. In some problems, this much memory and computation is impractical, yet they are still potentially solvable since relatively few states occur along optimal solution trajectories. Asynchronous methods and other variations of GPI can be applied in such cases and may find good or optimal policies much faster than synchronous methods can.

4.8 Summary
Policy evaluation refers to the iterative computation of the value functions for a given policy.
Policy improvement refers to the computation of an improved policy given the value function for that policy.
Policy Iteration and *Value Iteration* are the two important DP methods.
Either of these can be used to reliably compute optimal policies and value functions for finite MDPs given complete knowledge of MDP.

Classical DP Methods operate in sweeps through the state set, performing an update operation on each state. Each update of a state depends on the values of all possible successor states and their probabilities of occurring. Expected updates are closely related to Bellman Equations. When the updates no longer result in any changes in value, then the convergence has occurred to values that satisfy the corresponding Bellman Equation.

Generalized Policy Iteration (GPI) is the general idea of two interacting processes revolving around an approximate policy and an approximate value function. One process takes the policy as given and performs the policy evaluation, changing the value function to be more like the true value function of that policy. The other process takes the value function as given and performs some form of policy improvement, changing the policy to make it better, assuming that the value functions is its value function. Each process changes the basis for the other, overall they work together to find a joint solution: a policy and value function that are unchanged by either process and consequently, are optimal.

Asynchronous DP methods are in-place iterative methods that update states in an arbitrary order, perhaps stochastically determined and using the out-of-date information. These methods can be viewed as fine grained forms of GPI.

Bootstrapping:
All the DP methods discussed above update estimates of the value of states based on estimates of the values of successor states. They update estimates based on other estimates. This idea is called as 'bootstrapping'.