# Summary - Chapter 4

Ameet Deshpande

December 19, 2017

## 1 Summaries

Dynamic Programming is generally used for finding optimal policies in Markov Decision Processes. These methods are natural for Discrete States, but they can also be used for continuous states by quantizing the states and actions. Dynamic Programming methods are somewhat like turning Bellman equations into assignment statements.

### 1.1 Section 4.1 - Policy Evaluation

- The first step is usually to estimate the value and value state functions, given the policy. The usual formula for value functions will be

$$V^{\pi}(s) = \sum \pi(s,a) \times \sum P^{a}_{ssf} \times (R^{a}_{ssf} + \gamma V^{\pi}(s^{f}))$$

- If the environment dynamics are known completely, it can be seen that it is easy to solve these equations. There are $|s|$ equations and unknowns. This might be tedious but simple calculation.

- But we generally resort to Iterative Policy Evaluation (**why?**) where we start off with an initial guess $V_o$ and use the bellman equations to perform updates.

- Backups done in DP algorithms are called *full backups* because they consider all possible next states and not a sample of them.

- If a program is being written for policy iteration, we could use two arrays, the second one to store the updated values of the value function. After completing a full sweep through all the states, the second array can be copied into the first array. Another *alternative* is to use the same array to store the updated values. This will allow the states using that value to have the most recent value and thus the convergence is naturally **faster**.

- The order in which the states are swept through has a large impact on the rate of convergence. This can be seen naturally because it would make sense to update from terminal states down up as terminal states will have its effect on more number of states in the topology. When writing algorithms, it could be a key step to figure out a good, if not optimal way to sweep through the states.

- A stopping condition for the number of iterations would in general be to check if $|V_{k+1}(s) - V_k(s)| < \epsilon$ for all the states.

## 1.2 Section 4.2 - Policy Improvement

- The process of making a new policy that improves over an original policy, by making it greedy with respect to the value function of the original policy, is called policy improvement
- Note that the policy improvement theorem will ensure that the new policy is better than the old policy *with respect to the current value function.*
- When actions are being chosen greedily, it could be possible that there are multiple actions for a state which have the same reward estimates. In this case, the probability of choosing could be distributed among these actions. Any such policy will work as long as sub-optimal actions get zero probabilities of being chosen.

## 1.3 Section 4.3 - Policy Iteration

- This is usually multiple iterations of *Policy Evaluation* and *Policy Improvement*. We will thus get a sequence of policies, each one better than the previous.

## 1.4 Section 4.4 - Value Iteration

- The *Policy Iteration* algorithm considered has a drawback that policy evaluation has to be done at each step, which in itself is an iterative algorithm taking several step. An alternative to this algorithm is the *Value iteration* algorithm which involves only 1 backup per state. $V_{k+1} = update\_rule$. It is thus just one sweep of the states in the Policy Evaluation step followed by one sweep of Policy Improvement.
- The algorithm's convergence can be improved by combining multiple sweeps of *Policy Evaluation* before the *Policy Improvement* step.
- *Trick:* By placing the reward of 1 on a win, calculating Value functions for different states will provide us with the "probabilities" of winning.

## 1.5 Section 4.5 - Asynchronous Dynamic Programming

- Asynchronous DP algorithms are in-place iterative DP algorithms that are not organized in terms of systematic sweeps of the state set. These algorithms back up the values of states in any order whatsoever, using whatever values of other states happen to be available. The values of some states may be backed up several times before the values of others are backed up once
- To guarantee convergence however, it is important to keep updating the value functions of all the states

- In essence, we thus do not have to update the values of all the states before proceeding to the next step of policy improvement. *Value Iteration* thus seems like a special case of this solution method. If the policy gets changed at some state $s$, we might want to update values of states which are "closer" to $s$ in some sense and thus reduce computation.

- Note that this algorithm does not actually guarantee lesser computation. It just ensures that the algorithm does not get locked in some entire sweep of Policy Evaluation before it can again improve. It can possibly reduce the unnecessary computation.

- For example, we can apply backups to states as the agent visits them. This makes it possible to focus the DP algorithm's backups onto parts of the state set that are most relevant to the agent. This kind of focusing is a repeated theme in reinforcement learning.

## 1.6 Section 4.6 - Generalized Policy Iteration

- There are multiple algorithms that were seen. They all follow a common theme of having a policy, evaluating it and then improving it with respect to that value function.

- We use the term generalized policy iteration (GPI) to refer to the general idea of letting policy evaluation and policy improvement processes interact, independent of the granularity and other details of the two processes. Almost all reinforcement learning methods are well described as GPI.

## 1.7 Section 4.7 Efficiency of Dynamic Programming

- In practice, DP methods usually converge much faster their theoretical worst-case run-times, particularly if they are started with good initial value functions or policies.

- For problems with large state spaces, *Asynchronous* DP methods might be useful as they don't require you to sweep through all the states.

- For some problems, even this much memory and computation (smaller sets) is impractical, yet the problem is still potentially solvable because only a relatively few states occur along optimal solution trajectories. Asynchronous methods and other variations of GPI can often be applied in such cases to find good or optimal policies much faster than synchronous methods.

- DP methods form estimates for a state based on the Value functions of successor states. They are in some sense updating one estimate based on another estimate. This general idea is called *Bootsrapping*.