7. Function Approximation

Our lower bound for online planners show that there are no online planners that lead to good policies in all MDPs while satisfying the following three requirements

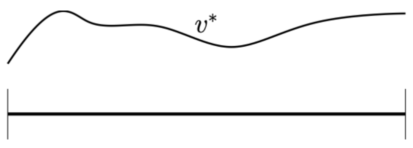
1. the planner induces policies that achieve some positive fraction of the optimal value in all MDPs;
2. the per-state runtime shows polynomial dependence on the planning horizon H and
3. it shows a polynomial dependence on the number of actions and
4. it shows no dependence on the number of states in the MDP.

Thus, one is left with no choice than to give up on one of the requirements. Since efficiency is clearly nonnegotiable (otherwise the runner just would not be practical), the only requirement that can be replaced is the first one. In what follows we will look at ways of relaxing this requirement.

In all the relaxations we will look at, we will essentially restrict the set of MDPs that the planner is expected to work on. However, we will do this in such a way that no MDP will be ever ruled out. We achieve this by giving the planner some **extra hint** about the MDP and we demand good performance only when the hint is correct. Since the hint will take a **general form**, some hint is always correct for **any** MDP. Hence, no MDP is left behind and the planner can again demanded to be efficient and effective.

Hints on value functions

The hints that we start with will concern the value functions. In particular, they state that either the optimal value, or the value function of all policies are effectively compressible.



For motivation, consider the figure on the right. Imagine the state space is an interval of the real line and the optimal value function in an MDP looks like as shown on the figure: It is a nice, smooth function over the interval. As is well known, such relatively slowly changing functions can be well approximated by using the linear combination of a few fixed basis functions, like an appropriate polynomial, or Fourier basis, or using splines. Then, one hopes that even though the state space is large or even infinite as in this example, there could perhaps be a method that calculates the few coefficients needed get a good approximation to v^* with a runtime that depends polynomially on the horizon, the number of actions and the number of coefficients that one needs to calculate. Given the knowledge of v^* and simulator access to the MDP, good actions can then be efficiently obtained by performing one-step lookahead computations.

Linear function approximation

If the basis functions mentioned are $\phi_1, \dots, \phi_d : \mathcal{S} \to \mathbb{R}$ then, formally, the hope is that with some coefficients $\theta = (\theta_1, \dots, \theta_d)^\top \in \mathbb{R}^d$, we will have

$$v^*(s) = \sum_{i=1}^{d} \theta_i \phi_i(s) \qquad \text{for all } s \in \mathcal{S}\,. \tag{1}$$

In the reinforcement learning literature, the vector $(\phi_1(s), \dots, \phi_d(s))^\top$ is called the feature vector assigned to state s. For a more compact notation we also use ϕ to be a map from $\mathcal{S}$ to $\mathbb{R}^d$ which assigns the feature vectors to the states:

$$\phi(s) = (\phi_1(s), \dots, \phi_d(s))^\top\,.$$

Conversely, given $\phi : \mathcal{S} \to \mathbb{R}^d$, its component are denoted using $\phi_1, \dots, \phi_d$. It will also be useful to introduce a matrix notation: Recall that the number of states is S and without loss of generality we may assume that $\mathcal{S} = [\mathrm{S}]$. Then, we can treat each of $\phi_1, \dots, \phi_d$ as S-dimensional vectors: The ith component of ϕ_j is $\phi_j(i)$. Then, we can stack $\phi_1, \dots, \phi_d$ next to each other to form a matrix:

$$\Phi = \begin{array}{cccc} | & | & \dots & | \\ \phi_1 & \phi_2 & \dots & \phi_d \\ | & | & \dots & | \end{array} \in \mathrm{R}^{\mathrm{S}\times d}\,.$$

That is, Φ is a $\mathrm{S} \times d$ matrix. The set of real-valued functions over the state space that can be described with the linear combination of the basis functions is

$$\mathcal{F} = \{f : \mathcal{S} \to \mathbb{R} \,:\, \exists \theta \in \mathbb{R}^d \text{ s.t. } f(s) = \langle \phi(s), \theta \rangle\}\,.$$

Identifying the space of real-valued functions with the vector space $\mathbb{R}^{\mathrm{S}}$ in the natural way, $\mathcal{F}$ is a d-dimensional subspace of $\mathbb{R}^{\mathrm{S}}$, which is the same as the "column space", or the span, or the range space of Φ:

$$\mathcal{F} = \{\Phi\theta \,:\, \theta \in \mathbb{R}^d\} = \mathrm{span}(\Phi)$$

If we need to indicate the dependence of $\mathcal{F}$ on the choice of features, we will write either $\mathcal{F}_\phi$ or $\mathcal{F}_\Phi$.

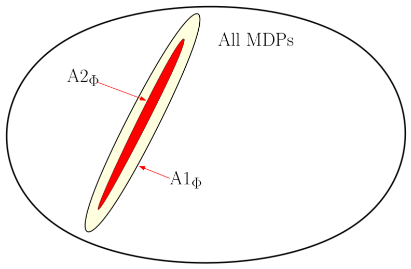
Now, we have three equivalent ways of specifying the "features", either by specifying the basis functions $\phi_1, \dots, \phi_d$, or the feature-map ϕ, or the feature matrix Φ, and we have a four equivalent way of specifying the functions that can be obtained via the linear combination of features.

Delivering the hint

Note that in the above problem description it is tacitly assumed that the feature-map, in some form or another, is available to the planner. In fact, the feature map can be made available in multiple ways. When we argue for lower bounds, especially for query complexity, we often assume that the whole feature-map is available for the algorithm. For upper bounds with online planning, the most natural assumption is that the planner gets from the simulator the feature vector of the states that it encounters. In particular, when it comes to online planning, the natural assumption is that the planner gets the feature vector of the initial state together with the state and with any subsequent calls to the simulator, the simulator returns the feature vector of the next states, together with the next states.

Typical hints

In what follows we will study planning under a number of different hints (or assumptions) that connect the MDP and a feature-map. The simplest of this just states that (1) holds:



Assumption A1 (v^*-realizibility): The MDP M and the featuremap ϕ are such that $v^* \in \mathcal{F}_\phi$

A second variation is when all value functions are realizable:

Assumption A2 (universal value function realizibility) The MDP M and the featuremap ϕ are such that for any memoryless policy π of the MDP, $v^\pi \in \mathcal{F}_\phi$.

Clearly, A2 implies A1, because by the fundamental theorem of MDPs, there exists a memoryless policy π such that $v^\pi = v^*$. The figure on the right illustrates the set of all finite MDPs with some state space and within those the set of those MDPs that satisfy A1 with a specific feature map ϕ (denoted by A1$_\phi$ on the figure), as well as those MDPs that satisfy A2 with the same feature map (denoted by A2$_\phi$). Both of these sets represent a very small fraction of all MDPs. However, of one changes the feature map, the union of all these sets clearly covers the set of all MDPs: The hint is general.

There are many variations of these assumptions. Often, we will find it useful to relax the assumption value functions are exactly realizable. Under the modified assumptions the value function does not need to lie in the span of the feature-map, but only in some vicinity of it. The natural error metric to be used is the maximum norm for reasons that will become clear later. To help with stating these assumptions in a compact form, introduce the notation

$$v \in_\varepsilon \mathcal{F}$$

to denote that

$$\inf_{f \in \mathcal{F}} \|f - v\|_\infty \le \epsilon \,.$$

That is, $v \in_\varepsilon \mathcal{F}$ means that the best approximator to v from $\mathcal{F}$ approximates it within a uniform error of ε.

Fixing $\varepsilon \ge 0$ and replacing $\in$ with $\in_\varepsilon$ in the above two assumptions gives the following:

Assumption A1$_\varepsilon$ (approximate v^* realizability): The MDP M and the featuremap ϕ are such that $v^* \in_\varepsilon \mathcal{F}_\phi$

Assumption A2$_\varepsilon$ (approximate universal value function realizibility) The MDP M and the featuremap ϕ are such that for any memoryless policy π of the MDP, $v^\pi \in_\varepsilon \mathcal{F}_\phi$.

Action-value hints

We obtain new variants if we consider feature-maps that map state-action pairs to vectors. Concretely, (by abusing notation) let $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$. Then, the analog of A1 is as follows:

Assumption B1 (q^*-realizibility): The MDP M and the featuremap ϕ are such that $q^* \in \mathcal{F}_\phi$

Here, as expected, $\mathcal{F}_\phi$ is defined as the set of functions that lie in the span of the feature-map. The analog of A2 is as follows:

Assumption B2 (universal value function realizibility) The MDP M and the featuremap ϕ are such that for any memoryless policy π of the MDP, $q^\pi \in \mathcal{F}_\phi$.

We can also introduce positive approximation errors $\varepsilon > 0$, which lead to B1$_\varepsilon$ and B2$_\varepsilon$:

Assumption B1$_\varepsilon$ (approximate q^*-realizibility): The MDP M and the featuremap ϕ are such that $q^* \in_\varepsilon \mathcal{F}_\phi$

Assumption B2$_\varepsilon$ (approximate universal value function realizibility) The MDP M and the featuremap ϕ are such that for any memoryless policy π of the MDP, $q^\pi \in_\varepsilon \mathcal{F}_\phi$.

One may wonder why not choose one of these assumptions? When one assumption implies another, then clearly there is a preference to choose the weaker assumption. But often, there is going to be a price and sometimes the assumptions are just not comparable.

Notes

Origin

The idea of using value function approximation in planning dates back to at least the 1960s if not earlier. I include some intriguing early references at the end. That these ideas already appeared at the down of computing where computers hardly even existed is quite intriguing.

Infinite spaces

Function approximation is especially appealing when the state space, or the action space, or both are "continuous" (i.e., they are a subset of a Euclidean space). In this case, the compression is "infinite". Experimental evidence suggests that function approximation can work quite well in the context of MDP planning in a surprisingly large number of different scenarios. When the spaces are infinite, all the "math" will still go through, except that occasionally one has to be a bit more careful. For example, one cannot clearly say that Φ is a matrix, but Φ can clearly be defined as a linear operator mapping $\mathbb{R}^d$ to the vector space of all real-valued functions over the (say) state space (when the feature map is also over states).

Nonlinear value function approximation

The most successful use of the idea of compressing value functions uses **neural networks**. Readers are most likely are already familiar with the ideas underlying neural networks. The hope here is that whatever we find in the case of linear function approximation will have implications in how to use nonlinear function approximation in MDP planning. In a way, the very first question is whether one can **decouple** the design of the planning algorithm from what function approximation technique it is used with. We will study this question by asking for planners that work with any feature map. If we find that we can identify planners that are performant no matter the feature map, the decoupling is successful and we can hope that the ideas will generalize to nonlinear function approximation. However, if we find that successful planners need to use intricate properties of the feature maps, then this is must be taken as a warning that complications may arise when the results are generalized to nonlinear function approximation. In any case, it appears to be a prudent strategy to first investigate the simpler, more straightforward linear case, before considering the nonlinear case.

Computation with advice/Non-uniform Computation

Computation with advice is a general approach in computer science where a problem of computing a map is changed to computing a map which has an additional input, the advice. Clearly, the approach taken here can be seen as a special case of computation with advice. There is also the closely related notion of non-uniform computation studied in computability/complexity theory. In non-uniform computation, the Turing machine, in addition to its input, also receives some “advice” string.

References

- Richard Bellman, Robert Kalaba and Bella Kotkin. 1963. Polynomial Approximation—A New Computational Technique in Dynamic Programming: Allocation Processes. Mathematics of Computation, 17 (82): 155-161
- Daniel, James W. 1976. “Splines and Efficiency in Dynamic Programming.” Journal of Mathematical Analysis and Applications 54 (2): 402–7.
- Schweitzer, Paul J., and Abraham Seidmann. 1985. “Generalized Polynomial Approximations in Markovian Decision Processes.” Journal of Mathematical Analysis and Applications 110 (2): 568–82.
- Brattka, Vasco, and Arno Pauly. 2010. Computation with Advice. arXiv [cs.LO].