10. Planning under realizability

PDF Version

The lesson from the last lecture is that efficient planners are limited to induce policies whose suboptimaly gap is polynomially larger than the misspecification error of the feature-map supplied to the planner. We have also seen) that if we accept this polynomial in the feature-space-dimension error amplification, a relatively straightforward adaptation of policy iteration gives rise to a computationally efficient (global) planner – at least, when the planner is furbished with the solution to an underlying optimal experimental design problem. In any case, the planner is query efficient.

All this was shown in the context when the misspecification error is relative to the set of action value functions underlying all possible policies. In this lecture we look into whether this error metric could be changed so that the misspecification error is measured by how well the optimal action-value function, , is approximated by the features, while still retaining the positive result. As the negative result already implies that there are no efficient planners unless the suboptimality gap of the induced policy is polynomially larger than the approximation error, we look into the case when **the optimal action-value function is perfectly representable** with the features supplied to the planner. This assumption is also known as " -realizability", or, " linear realizability", if we want to be more specific about the nature of the function approximation technique used.

Planning under realizability

We consider fixed horizon online planning in large finite MDPs . As usual, the horizon is denoted by and we consider planning with a fixed initial state , as in the previous lecture. Let us denote by the states that are reachable from in steps. As before, we assume that when . Recall that in this case the action-value functions depend on the number of steps left, of the current stage. For a fixed , let be the optimal action-value function with stages in the process, stages left. Since we do not need the values of outside of , we abuse notation by redefining it restricted to this set.

Important note: The indexing of used here is not consistent with the indexing used in the previous lecture, where it was more convenient to index value functions based on the number of stages left.

The planner will be given a feature map for every stage such that . The realizability assumption means that

Note that we demand that **the same parameter vector is shared between all stages**. As it turns out, this makes our result stronger. Regardless, at the price of increasing the dimension from to , one can always assume that the parameter vector is shared. Since we will give a negative result concerning the query-efficiency of planners, we allow the planners access to the full feature-map: The negative result still applies even if the planner is allowed to perform any sort of computation with the feature-map during or before the planning process.

For , we call an online planner **-sound for the -step criterion** if for any MDP and feature map pair such that the optimal action-value function of is realizable with the features in the sense that holds, the planner induces a policy that is -suboptimal or better when evaluated with the -horizon undiscounted total reward criterion from the designated start-state in MDP . Note that this is very much the same as the previous soundness criterion, except that the definition of the approximation error is relaxed, while we demand .

The result below uses MDPs where the immediate reward (obtained from the simulator) can be random. The random reward is used to make the job of the planners harder and it allows us to consider MDPs with deterministic dynamics. (The result could also be proven for MDPs with deterministic rewards and random transitions.)

The usual **definition of MDPs with random transitions and rewards** is in a way even simpler: Such a (finite) MDP is given by the tuple where is a collection of distributions over state-reward pairs. In particular, for all state-action pairs , . Letting (i.e., is drawn from at random), we can recover as the distribution of and as the expected value of . That the reward can be random forces a change to the notion of the canonical probability spaces, since histories now also show include rewards, incurred in each time step . With appropriate modifications, we can nevertheless still introduce and the corresponding expectation operator, , as well. The natural definition of the value of a policy at state , say, in the discounted setting is then . However, it is easy to see that for any , , and, as such, nothing changes in the theoretical results derived so far.

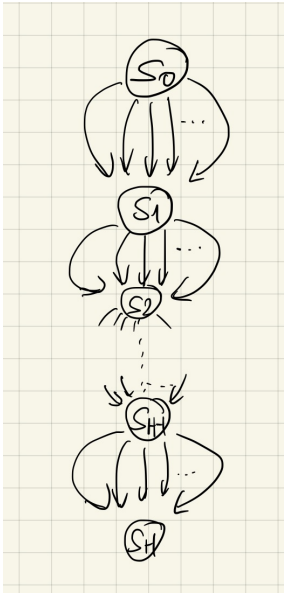
For reals, let . The main result of this lecture is as follows:

Theorem (worst-case query-cost is exponential under -realizability): For any large enough and any online planner that is -sound for the -horizon planning problem, there exists a triplet where is a finite MDP with random rewards taking values in and deterministic transitions, is a state of this MDP and is a -dimensional feature-map such that holds for the optimal action-value function and the expected number of queries that uses when interconnected with satisfies

---

Note that with random rewards with no control on their tail behavior (e.g., unbounded variance) it would not be hard to make the job of any planner arbitrarily hard. As such, it is quite important that the MDPs that are constructed for the result, the rewards, while random, lie in a fixed interval. Note that the specific choice of this interval does not matter: If there is a hard example with some interval, that example can be translated into another by shifting and scaling, and at the price of introducing an extra dimension in the feature map to account for the shifts. A similar comment applies to (which, nevertheless, needs to be scaled to the range of the rewards).

The main ideas of the proof



Rather than giving the full proof, we will just explain the main ideas behind it. At a high-level, the proof merges the ideas behind the lower bound for the small action-set case and the lower bound of the large action-set case. That is, we will consider an action set that is exponentially large in . In particular, we will consider action sets that have elements.

Note that because realizability holds, having a large action set but with a trivial dynamics (as in the lower bound in the last lecture) does not lead to the lower bound of the desired form. In particular, if the dynamics are trivial (i.e., , see the figure on the right) then the optimal action to be taken at does not depend on what actions are taken at later stages and can be efficiently found by just maximizing for the reward received in that stage, which can be done efficiently due to our realizability assumption, even in the presence of random rewards. Whether an example exists with only a few actions but with a more complicated dynamics remains open. With the construction provided here (which is based on tree dynamics and zero intermediate reward in the tree), this clearly fails, as we will make it clear below.

In any case, since the "chain dynamics" does not work, the next simplest approach is to have a tree, but with exponentially many actions in every node. Since this creates many many states (states at stage) the next question then is **how to ensure realizability**. There are two issues: We need to be able to keep the dimension fixed at at every stage and somehow we will need to have a way of controlling which action should be optimal at each state at each stage. Indeed, realizability means that we need to ensure that for all and

,

Here, stands for the state that is reached by taking action in state (in the tree, every node, or state is uniquely indexed by the action sequence that reaches it). Now, in the definition of , for all , we also have , which calls for the need to know the identity of the maximizing action. What is more, since the solution to the Bellman optimality equations is unique, if we guarantee that holds at all state-action pairs for with some features and parameter vectors, it also follows that for all , that is, is realizable with the features.

A simple approach to resolve all of these issues is to **let a fixed action be the optimal action at all the states**, together with using the JL features from the previous lecture (the identity of this action is of course hidden from the planner). In particular, the **JL feature-matrix lemma** from the previous lecture furnishes us with -dimensional unit vectors such that for ,

—

Fix these vectors. That should be optimal at all states is equivalent to that

In our earlier proof we used and . Will this still work? Unfortunately, it does not. The first observation is that from this it follows that for any , , ,

As such, for almost all the actions , we expect to be close to . Now, under this choice we also have that for all states and all stages . This creates essentially the same problem as what we saw above with the trivial chain dynamics. In particular, from we get that . As such, we expect to be close to either or (since is close to). Putting aside the issue that we wanted the immediate reward be in , we see that if the reward noise is not large, and thus the identity of can be obtained with just a few queries: The signal to noise ratio is just too good!

This problem replicates itself at the very last stage: Here, for any state , hence

for any pair. Unless we choose to be small, say, , a planner will succeed with fewer queries than in our desired bound.

This motivates us to introduce a **scaling of the features** (recall that the parameter vector is shared between the stages) with some scaling factors. For maximum generality, we allow for the scaling factor of the feature vector of to depend on itself (since states between stages are not shared, scaling can depend on the stage with this choice). Let be the scaling factor we intend to use with where we intend to **keep in a constant range** (so the scaling with the stage index works as intended) while we aim to use .

Now, we can explain the **need for many actions**. By the Bellman optimality equation we have that for any suboptimal action, ,

where uses that . From this we see that close to the initial state the reward gaps are of constant order. In particular, **if there were only a few actions per state**, a planner could identify the optimal action by finding the action whose reward is significantly larger than that of the others. By choosing to have many actions, the planner faces a "needle-in-a-haystack" situation, which makes their job hopeless even with perfect signal (no noise).

The next idea is to **force "clever" planners to only experiment with actions in the last stage**. Since here, the signal-to-noise ratio will be very poor, if we manage to achieve this, even clever planners will need to use a large number of queries. A simple way of forcing this is to **choose all the rewards while transitioning in the tree and taking suboptimal actions to be identically zero** except for stage , where, in accordance to our earlier plan, the rewards are chosen at random to ensure consistency but the signal to noise ratio will be poor.

Since the dynamics in the tree is known, and it is known that all rewards are zero with the possible exception of when using the optimal action (one of exponentially many actions and is thus hard to find), planners are either left with either solving the needle in a haystack problem of identifying the optimal action by randomly stumbling upon it, or they need to experiment with actions in the last stage. That the rewards are chosen to be identically zero is not critical: From the point of view of this argument, what is critical is that they are all the same.

It remains to be seen that consistency can be achieved and also that the optimal action at has a large value compared to the values of suboptimal actions at the same state. Here, we still face some challenges with consistency. Since we want the immediate rewards to belong to the

interval, all the action values have to be nonnegative. As such, it will be easier if we introduce an additional bias component in the feature vectors, which we allow to scale with the stage.

To summarize, we let

while we propose to use

—

It remains to show that and can be satisfied with , while also keeping the suboptimal gap of at large, and while the last stage rewards ( ) are in and are of size as planned.

Assume for a moment that is optimal in all states, i.e., that holds. Then, is also optimal in state , hence, under , for any is equivalent to

where we also used that by assumption because . Plugging in the definitions,

— —

Define so that

— —

with — — (i.e., is a decreasing geometric sequence) This has two implications: simplifies to

— —

and also for the last stage rewards, from we get

— — — —

Clearly, if , since for , , while also .

With this, to satisfy , on the one hand we choose to define with the following "downward recursion" in the tree: For any in the tree and actions ,

— —

Note that this is consistent with . The next challenge is to show that stays within a constant range. In fact, with the above definition, this will not hold. In particular, when , the right-hand side can be as large as , which means that the scaling coefficients will exponentially increase with a base of . Note, however, that if , then provided that (which can be ensured at the root by choosing for all actions),

— — — — — — —

and thus will also hold.

Hence, we modify the construction so that **the definition is never needed for** . This is achieved by changing the dynamics: We introduce a special set of states, , the **exit lane**. Once, the process gets into this lane, there is now return and in fact all the remaining rewards up the end are zero. Specifically, all the actions in lead to state and we set the feature vector of all states in the exit-lane zero:

This way, regardless the choice of the parameter vector, we ensure that the Bellman optimality equations hold at these state and the optimal values are correctly set to zero.

The exit lane is introduced to remove the need to use with repeat actions. In particular, for any with some , say, (i.e., is obtained by following these actions) then if for , the next state is . Since the optimal value of is zero and we don't intend to introduce an immediate reward, we set

making the value of repeat actions zero. The next complication is that this ruins our plan to keep optimal at all states: Indeed, could be applied multiply times in a path from to a leaf of the tree, and by the second application, the new rule forces the value of to be zero. Hence, we need to modify this rule when the action is .

Clearly, whether a suboptimal action, or is repeated is problematic for the recursive definition of . Hence, it is better if is also forced to use the exit lane. Thus, if is used in with , the next state is . However, we do not zero out , but keep the recursive definition and we rather introduce an immediate reward to match . It is not

hard to check that this reward is also in the range. Note that here if then by definition . This completes the description of the structure of the MDPs.

That the action gap at is large follows from the choice of the JL feature vectors.

It remains to be seen that is indeed the optimal action at any state. This boils down to checking that for , . When is a repeat action, this is trivial. When is not a repeat action, we have

— — — — — —

where we used that and and thus — by the choice of and since .

Let denote the MDP constructed this way when the optimal action is (the feature maps, of course, are common between these MDPs). For a formal proof, one also needs to argue that planners that do not use many queries cannot distinguish between these MDPs. Intuitively, this is because such planners will receive, with high probability, identical observations under different MDPs in this class. As such, these planners can at best randomly choose an action ("needle in a haystack") and since in MDP only action incurs high values, they cannot induce a policy with a near-optimal value.

Computation with many actions

In the construction given the number of actions was allowed to scale exponentially with the dimension. The above proof would show a separation between the query and computation complexity of planning, if one could demonstrate that there is a choice of the JL feature vectors when the optimization problems

admits a computationally efficient solver regardless of the choice of and (for simplicity, we suppress dependence on). Whether such a solver exist will depend on the choice of the feature-map and this is a fascinating question on its own. One approach to arrive at such a solver is to rewrite this problem as the problem of finding

where is the convex hull of the feature vectors . Provided that this problem admits an efficient solution and given any extreme point of , we can efficiently recover an action such that (this amounts to "inverting" the feature map), the first problem can also be solved efficiently.

Note that is a linear optimization problem over a convex set and the question whether this problem admits an efficient solver lies at the heart of computer science. The general lesson is that the answer can be expected to be yes when has some "convenient" description other than the one that is used to define it. The second problem of inverting the feature map is known as the "decomposition problem" and the same conclusions hold for this problem.

Notes

- It is possible to modify the construction to make it work in the discounted setting. The paper cited below shows how.
- Back to the finite horizon setting, for an upper bound, one can employ the **least-squares value iteration algorithm with -optimal** design (LSVI-G), which we have met in Homework 2. What results is that to get a -sound (global) planner with this approach,

---

queries are sufficient (and the compute cost is also of similar order). We see that as far as the exponents in the lower and upper bounds are concerned, in the upper bound the exponent is while in the lower bound it is . Thus, there remains a logarithmic gap between them when , while the **gap is unbounded** when , i.e., for **long horizon problems**. In particular, in the constant dimension and long-horizon featurized planning problem, the LSVI-G algorithm seems to be suboptimal because it calculates the optimal action-value function stage-wise. One conjectures that the upper bound for LSVI-G is tight, while the lower bound in this lecture is also essentially correct. This would means that there is an alternate algorithm that could perform much better than LSVI-G in large-horizon planning with constant feature-dimension. Clearly, for the specific construction used in this lecture, a planner that tries all actions, say at , will find the optimal action and the **cost of this planner is independent of the horizon**. Hence, at least in this case, the lower bound can be matched with an alternate algorithm. One may think that this problem is purely of theoretical interest. To counter this note that long-horizon planning is a really important practical question: Many applications require thousands of steps, if not millions, while perhaps the feature space dimension does not need to be very large. Whether there exist an algorithm that works better than LSVI-G thus remains to be a fascinating open problem with good potential for having a real impact on applications.

- For infinite horizon undiscounted problems and realizability, there is a simple example that shows that with actions and -dimensional features, any query efficient planner that guarantees a constant suboptimality gap needs queries per state. This is based on a shortest path problem on a regular grid. Here, the obstruction is simply algebraic: There is no noise in either the transitions or the rewards.

Bibliographical notes

This lecture is entirely based on the paper

- Weisz, Gellert, Philip Amortila, and Csaba Szepesvári. 2020. "Exponential Lower Bounds for Planning in MDPs With Linearly-Realizable Optimal Action-Value Functions.",

which is available on arXiv and which will also soon appear at ALT.

The second lower for the undiscounted setting mentioned in the notes is from

- Weisz, Gellert, Philip Amortila, Barnabás Janzer, Yasin Abbasi-Yadkori, Nan Jiang, and Csaba Szepesvári. 2021. "On Query-Efficient Planning in MDPs under Linear Realizability of the Optimal State-Value Function."

available on arXiv.

A beautiful book that is a very good source on reading about the linear optimization problem mentioned above is

- Grotschel, Martin, László Lovász, and Alexander Schrijver. 1993. Geometric Algorithms and Combinatorial Optimization. Vol. 2. Algorithms and Combinatorics. Berlin, Heidelberg: Springer Berlin Heidelberg.
