Lecture 12: Algorithms for nonlinear function approximation

Akshay Krishnamurthy
akshay@cs.umass.edu

April 21, 2022

In the last lecture, we developed structural conditions that permit statistically efficient reinforcement learning with nonlinear function approximation. The basic condition is that the way that the function class interacts with the MDP dynamics admits some "bilinear structure." More specifically we saw the notion of the Bellman rank, where the average Bellman errors admit a factorization as:

$$\mathcal{E}_h(\pi, f) := \mathbb{E}_{\substack{s_h \sim d_h^\pi \\ a_h \sim \pi_f(s_h)}} [(f - \mathcal{T}f)(s_h, a_h)] = \langle w_h(\pi), v_h(f) \rangle ,$$

where $w_h : \Pi \to \mathbb{R}^d$ and $v_h : \mathcal{F} \to \mathbb{R}^d$ are d-dimensional embeddings associated with the policies and value functions. The Bellman rank of the problem is the dimensionality of this embedding.

We also briefly discussed many models that admit low Bellman rank. Two illustrative examples that highlight the two distinct aspects of the decomposition are the linear Bellman complete setting and the block MDP setting. The former admits low Bellman rank simply because $(f - \mathcal{T}f)$ is a linear function. On the other hand, the latter admits low Bellman rank because d_h^π can be embedded into a low-dimensional space, capturing a "bottleneck" structure in the dynamics.

Finally, our proof that problems with bilinear structure are statistically tractable was actually constructive. We developed an algorithm and showed that this algorithm has sample complexity scaling polynomially with all of the relevant quantities. Thus it can addresses all three of the challenges we have been focusing on in the course.

The main downside is that this algorithm is not computationally efficient in any sense; it should be thought of as the analog of exponential weights for contextual bandits (recall that this algorithm achieves the optimal regret guarantee for contextual bandits, but requires enumerating over the policy class). Continuing with this analogy, in contextual bandits we saw that we can address computation and statistics simultaneously through the framework of *oracle efficient* methods, of which SquareCB is a notable example. Thus, we can ask how to develop oracle efficient algorithms for Bilinear classes if at all possible, or, more narrowly, for interesting subclasses. While this is still relatively immature as a research program, there have been some interesting ideas and connections to the Deep RL literature, which we will cover today.

1 Basic framework: representation learning

So far, we don't have an efficient algorithm that works for general Bilinear classes, and we believe that this is not possible. But we can make some progress in some models and perhaps the most interesting models are the ones with bottleneck structure like the block and low rank MDPs. For these problems, the basic approach for developing algorithms is through *representation learning*, where we construct an auxiliary supervision signal that reveals some useful aspect of the bottleneck structure. The auxiliary supervision is used to learn a concise representation of the dynamics and this representation is used to form exploration bonuses that allows us to plan to acquire new information. Typically representation learning, exploration bonus construction, and planning to explore, are interleaved until no more information can be acquired. At this point reward maximization is usually quite straightforward.

Block MDP formulation. To fix the ideas of how representation learning might work, consider the block MDP model. We'll change the notation a bit to align with the literature. The block MDP is defined by a tuple $(\mathcal{X}, \mathcal{Z}, \mathcal{A}, P, R)$ with horizon H. Here $\mathcal{X}$ is the *observation space*, $\mathcal{Z}$ is the *latent state space*, and $\mathcal{A}$ is the action

space. Both $\mathcal{Z}$ and $\mathcal{A}$ are finite. Each observation $x \in \mathcal{X}$ is associated with a *unique* latent state $z \in \mathcal{Z}$ and we use $\phi^\star : \mathcal{X} \to \mathcal{Z}$ to denote this mapping. The dynamics $P : \mathcal{X} \times \mathcal{A} \to \Delta(\mathcal{X})$ can be expressed in terms of a latent transition operator $P_Z : \mathcal{Z} \times \mathcal{A} \to \Delta(\mathcal{Z})$ and an emission distribution $P_X : \mathcal{Z} \to \Delta(\mathcal{X})$ as

$$P(x' \mid x, a) = \sum_z P_X(x' \mid z) \cdot P_Z(z \mid \phi^\star(x), a)$$

In other words, the transition operates by first "decoding" the current latent state $\phi^\star(x)$, then transitioning to a new latent state, and finally emitting an observation. We often also assume that the reward function R only depends on the latent state.

In this setup, $\phi^\star$ can be thought of as a "planted" representation that concisely describes the MDP dynamics. Specifically, if we know $\phi^\star$ then, we could ignore the observations completely and instead work with the latent states, which reduces the problem to a tabular MDP. So for representation learning, one idea is to consider a function class $\Phi : \mathcal{X} \to \mathcal{Z}$ of candidate decoders/representations and assume realizability of the ground-truth representation, that is $\phi^\star \in \Phi$. We hope to explore and find a near optimal policy with sample complexity scaling as $\log |\Phi|$.

We should point out that this setting admits low Bellman rank if you use Φ to define the Q-function class. Consider the class defined as all "lookup tables" on top of all candidate decoders:

$$\mathcal{F} := \{f_{M,\phi} : (x, a) \mapsto M(\phi^\star(x), a) : \phi \in \Phi, M : \mathcal{Z} \times \mathcal{A} \to [0, H]\}.$$

This function class has statistical complexity scaling as $|\mathcal{Z}||\mathcal{A}| + \log |\Phi|$, and contains $\vec{Q}^\star$ as long as $\phi^\star \in \Phi$. As we saw previously, the Block MDP has Bellman rank $|\mathcal{Z}|$ for *any* function class, and so this representation learning formulation is statistically tractable. The question is how to develop computationally efficient methods.

Forward algorithm. In this setup, one scheme we can consider is a forward-algorithm that operates one step at a time. In the h^{th} iteration, we do three things:

1. collect some data and learn a representation $\hat{\phi}_h$ for layer h;
2. define reward functions of the form $\hat{R}_{h,z}(x_h) := \mathbf{1}\{\hat{\phi}_h(x_h) = z\}$;
3. plan to find policies $\hat{\pi}_{h,z}$ that optimize $R_{h,z}$.

The idea is that this set of policies $\Psi_h = \{\hat{\pi}_{h,z} : z \in \mathcal{Z}\}$ will cover all of the states at layer h. More specifically, we can proceed inductively, assuming good coverage over the true latent states at time $h-1$ and showing that Ψ_h somehow provides similar coverage over time step h. Then we can use Ψ_h to collect data for the next iteration.

2 Block MDP approaches

To instantiate an algorithm we primarily need to specify items (1) and (3) above, as item (2) is already specified. In typically instantiations, many of the techniques we've already learned can work for item (3). For example, if you have good coverage, then the approximate value iteration procedure we saw in the homework can be used for planning, but we may require a completeness assumption.

Let us turn to item (1), where the key question is how to learn a useful representation. This would be easy if we observed the latent states, as we could just solve a supervised learning problem with our decoder class, e.g., train $\hat{\phi} = \operatorname{argmin}_{\phi \in \Phi} \mathbb{E}[\mathbf{1}\{\phi(x) \neq z\}]$ using the *labeled* dataset. The issue is that we never observe the latent states, so we don't have the labels! On the other hand, we may be able to use just the observations and actions for representation learning. This is a fairly popular idea in the empirical literature and some things one can try are:

Inverse dynamics. Inverse dynamics broadly refers to learning to predict actions given observations. We may consider collecting a dataset (x_{h-1}, a_{h-1}, x_h) where $a_{h-1} \sim \text{Unif}(\mathcal{A})$, and training a representation via

$$\hat{\phi}, \hat{f} \leftarrow \operatorname*{argmin}_{f, \phi \in \Phi} \mathbb{E}\left[-\log f(a_{h-1} \mid \phi(x_{h-1}), \phi(x_h))\right].$$

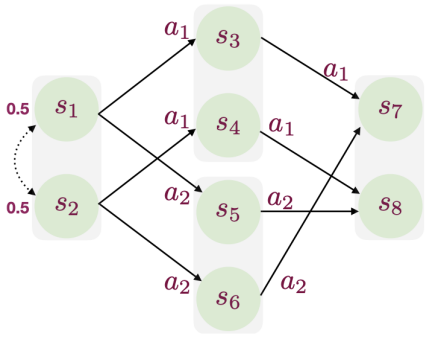

Figure 1: Counterexamples for inverse dynamics and backward prediction. In the figure, both approaches group $\{s_3, s_4\}, \{s_5, s_6\}, \{s_7, s_8\}$ which results in an exploration failure, since one can visit $\{s_7, s_8\}$ maximally but completely avoiding s_8.

Approaches based on this idea are fairly successful in Deep RL settings. In theory, we can understand this procedure by considering the global minimizer of the objective. Via Bayes rule, the global minimizer can be written as

$$f^\star(a \mid x, x') = \Pr[a \mid x, x'] = \frac{\Pr[x' \mid x, a] \Pr[a \mid x]}{\Pr[x' \mid x]} = \frac{P(x' \mid x, a)(1/A)}{\sum_{a'} P(x' \mid x, a')(1/A)} = \frac{P_Z(\phi^\star(x') \mid \phi^\star(x), a)}{\sum_{a'} P_Z(\phi^\star(x') \mid \phi^\star(x), a')}.$$

From, here we can see that the global minimizer only depends on the latent state $\mathcal{Z}$, so the objective encourages us to filter irrelevant information. However, you can construct examples where it results in a "over-abstraction" in which we also filter important information. An example is displayed in Figure 1. So while inverse dynamics may work in some cases, for theory one needs to make assumptions to rule out these counterexamples.

Backward prediction. If we have a learned representation for time $h-1$, we could consider predicting the previous latent state $\hat{\phi}_{h-1}(x_{h-1})$ and previous action a_{h-1} given the current observation. Again this can be justified intuitively by reasoning about the global optimizer when we have the correct representation for the previous layer.

$$\hat{\phi}_h, \hat{f} \leftarrow \operatorname*{argmin}_{f, \phi \in \Phi} \mathbb{E}\left[-\log f(a_{h-1}, \hat{\phi}_{h-1}(x_{h-1}) \mid \phi(x_h))\right].$$

If $\hat{\phi}_{h-1} = \phi^\star$ then the global minimizer here is

$$f^\star(a, z_{h-1} \mid x_h) = \Pr[a, z_{h-1} \mid x_h] = \frac{\Pr[x_h \mid a, z_{h-1}] \Pr[a, z_{h-1}]}{\Pr[x_h]} = \frac{P_Z(z_h \mid a, z_{h-1}) \Pr[z_{h-1}]}{\sum_{z'_{h-1}, a'} P_Z(z_h \mid a', z'_{h-1}) \Pr[z'_{h-1}]}$$

As with inverse dynamics, this objective also encourages filtering irrelevant information. This can can be analyzed under fairly strong conditions on the dynamics, in particular the fact that we won't have the $\phi^\star$ for the previous time step can be addressed. However, the approach is subject to similar counterexamples as inverse dynamics, in that it throws away too much information.

Contrastive learning. Another possibility is based on contrastive learning, which is a popular technique for learning pre-trained representations in a self-supervised manner outside of RL. In the self-supervised setting, the idea behind contrastive learning is to use some notion of semantic similarity between inputs to create classification problems where we try to distinguish between similar and dissimilar items. For example, when working with text data, we might take two sentences from the same paragraph to be "similar" and sentences from different paragraphs to be different. If we train a model in this fashion, we hope that the representation learned captures some underlying structure of text, which would be useful for other tasks.

In RL, we can use temporal information to define similarity. For example, we can collect two triples $(x^{(1)}_{h-1}, a^{(1)}_{h-1}, x^{(1)}_h)$ and $(x^{(2)}_{h-1}, a^{(2)}_{h-1}, x^{(2)}_h)$, and we can swap the second observation to create a dissimilar example. More formally, let

D_{h-1} be some distribution over observations at time $h-1$. We consider the sampling process:

$$x_{h-1}^{(i)} \sim D_{h-1}, a_{h-1}^{(i)} \sim \text{Unif}(\mathcal{A}), x_h^{(i)} \sim P(\cdot \mid x_h, a_h), \text{ for } i \in \{1,2\}, y \sim \text{Unif}(\{0,1\})$$
$$(x, a, x') = \begin{cases} (x_{h-1}^{(1)}, a_{h-1}^{(1)}, x_h^{(1)}) \text{ if } y = 1 \\ (x_{h-1}^{(1)}, a_{h-1}^{(1)}, x_h^{(2)}) \text{ if } y = 0 \end{cases}$$

and we return the tuple (x, a, x') with y as the label. Then we train a binary classifier as

$$\hat{f}, \hat{\phi} \leftarrow \operatorname*{argmin}_{f, \phi \in \Phi} \mathbb{E}_{(x,a,x',y)} \left[-\log f(y \mid \phi(x), a, \phi(x')) \right],$$

where the expectation is over data sampled from the above process. In this way, the classifier is trained to predict "real" transition triples that may occur via the dynamics, from "fake" transition triples that we manually spliced together. Again, reasoning about the global optimizer, we find that

$$f^\star(1 \mid x, a, x') = \frac{\Pr[x, a, x' \mid y = 1]}{\Pr[x, a, x' \mid y = 1] + \Pr[x, a, x' \mid y = 0]} = \frac{P(x' \mid x, a)(1/A)D_{h-1}(x)}{P(x' \mid x, a)(1/A)D_{h-1}(x) + D_h(x')(1/A)D_{h-1}(x)}$$
$$= \frac{P(x' \mid x, a)}{P(x' \mid x, a) + D_h(x')} = \frac{P(\phi^\star(x') \mid \phi^\star(x), a)}{P(\phi^\star(x') \mid \phi^\star(x), a) + D_h(\phi^\star(x'))}.$$

Here D_h is the marginal distribution over x_h induced by the above sampling process. Again here we find that the objective encourages filtering irrelevant information. As with the other approaches, this approach may suffer from the over-abstraction issue, but it does not lead to an exploration failure. To understand why, suppose we find a near-optimizer $\hat{f}, \hat{\phi}$ where $\hat{\phi}$ groups two observations x' and $\tilde{x}'$ at time step h together. Then even if x and x' come from different latent states, we can show that

$$\forall x, a : f^\star(1 \mid x, a, x') \approx f^\star(1 \mid x, a, x') \text{ which implies } \frac{P(x' \mid x, a)}{P(\tilde{x}' \mid x, a)} \approx \frac{D_h(\tilde{x'})}{D_h(x')}.$$

Put another way the ratio of transitioning to x' or $\tilde{x'}$ actually does not depend on the previous (x, a) pair. Since a policy we might use to roll-in induces a distribution over (x, a) this implies that

$$\forall \pi : \frac{d_h^\pi(x')}{d_h^\pi(\tilde{x}')} \approx \text{constant independent of } \pi.$$

This does not show that x' and $\tilde{x}'$ come from the same latent state, even if we had exact equality here. In particular, it could be the case that the above equality is satisfied, but two latent states have drastically different "forward" dynamics, in the sense that $T(\cdot \mid z_1, a) \neq T(\cdot \mid z_2, a)$. However, the argument above shows that if x' and $\tilde{x}'$ come from different latent states (say z_1, z_2), then the two latent states are coupled as far as the "backward dynamics" are concerned. This means that if we plan to reach one of the states z_1 then we will also hit z_2. So setting the reward function as $\mathbf{1}\{\hat{\phi}(x_h) = z\}$ and planning to optimize this reward function will succeed.

3 Low rank MDP approaches

We can also consider representation learning in the low rank or linear MDP. The "model-free" formulation is that we have a linear MDP with unknown feature maps $\phi^\star, \mu^\star$ and we have a function class Φ containing the true feature mapping $\phi^\star$. In the "model-based" formulation, we have two classes Φ, Υ where $\phi^\star \in \Phi, \mu^\star \in \Upsilon$. The model-based assumptions are stronger, since we require realizability for both embeddings. As you will show on the homework, both formulations have Bellman rank d, the feature dimension.

The model-based formulation admits a very elegant algorithm that combines many of the techniques we have seen previously. It is an iterative algorithm that operates in the discounted setting, starting with π_0 as the uniform random policy and two empty datasets D_0, D_0'. In iteration t we

1. Collect a tuple (s, a, s', a, s'') where $s \sim d^{\pi_{t-1}}$ (the stationary measure for the previous policy), $a, a' \sim \text{Unif}(\mathcal{A})$, and $s' \sim P(s, a), s'' \sim P(s', a')$. Recall that we can use geometric stopping to obtain a sample from $d^{\pi_{t-1}}$ with a single trajectory. Add (s, a, s') to D_{t-1} to form D_t and add (s', a', s'') to D_{t-1}' to form D_t.

2. Fit a model $(\hat{\phi}_t, \hat{\mu}_t)$ using maximum likelihood estimation:

$$(\hat{\phi}_t, \hat{\mu}_t) = \operatorname*{argmax}_{\phi \in \Phi, \mu \in \Upsilon} \sum_{D_t \cup D'_t} \log(\langle \phi(s,a), \mu(s') \rangle)$$

3. Using the new learned features $\hat{\phi}_t$, update the empirical covariance matrix as $\Sigma_t = \sum_{D_t} \hat{\phi}_t(s,a)\hat{\phi}_t(s,a)^\top + \lambda I$ and set the reward bonus as

$$b_t(s,a) := \beta \sqrt{\hat{\phi}_t(s,a)^\top \Sigma_t^{-1} \hat{\phi}_t(s,a)}$$

4. Compute π_t to maximize $r + b_t$ *in the learned model* $\hat{P}_t = (\hat{\phi}_t, \hat{\mu}_t)$. This can be done by running LSVI-UCB, but does not require any samples from the environment.

This algorithm is also reasonably close to model-based approaches that work quite well in practice. These empirical algorithms fit dynamics models (typically not by MLE but using a variational auto-encoder), create an exploration bonus (perhaps elliptical potential but sometimes using ensembles), and plan to maximize the exploration bonus in the model. So while not all of the parts are the same, you can view this algorithm/analysis as providing some justification to these approaches.

While this algorithm certainly has familiar elements, like the elliptical bonus, it is quite challenging to analyze. The first issue is that, even though we use the elliptical bonus for exploration, the features $\hat{\phi}_t$ change from iteration to iteration, so it is not clear how to use $\hat{\phi}_t$ to track progress. Related to this, it is not clear that using the bonus defined by the learned features leads to any sort of optimism, which we have seen is quite crucial for exploration.

The first thing to point out is that we can get a generalization guarantee for the maximum likelihood estimation procedure. Let ρ_t be the uniform mixture of $\{d^{\pi_i} \circ \mathrm{Unif}(\mathcal{A})\}_{i=0}^{t-1}$ and let ρ'_t be defined as the next step distribution. Then we can get that

$$\forall t : \max\{\mathbb{E}_{(s,a)\sim\rho_t}\left[\|\hat{P}_t(s,a) - P(s,a)\|_{\mathrm{TV}}^2\right], \mathbb{E}_{(s,a)\sim\rho'_t}\left[\|\hat{P}_t(s,a) - P(s,a)\|_{\mathrm{TV}}^2\right]\} \lesssim \frac{\log|\Phi||\Upsilon|}{t} =: \varepsilon_{\mathrm{stat}}.$$

Using this MLE guarantee, it turns out that we can establish global near-optimism for this algorithm by applying the simulation lemma in the learned model $\hat{P}_t$ (which has dynamics $\left\langle \hat{\phi}_t, \hat{\mu}_t \right\rangle$). In particular, we can show that

$$\forall t, \forall \pi : J(\pi; \hat{P}_t, r + b_t) - J(\pi; P, r) \gtrsim -\sqrt{\frac{A}{1-\gamma} \cdot \varepsilon_{\mathrm{stat}}}, \tag{1}$$

Then we can use the regret decomposition for globally optimistic algorithms to show that

$$J(\pi^\star; P, r) - J(\pi_t; P, r) \lesssim \frac{1}{1-\gamma}\mathbb{E}_{s,a\sim d_P^{\pi_t}}\Bigg[\underbrace{b_t(s,a) + \frac{1}{1-\gamma}\|\hat{P}_t(s,a) - P^\star(s,a)\|_{\mathrm{TV}}}_{\text{bound on } Q^{\pi_t} - \mathcal{T}Q^{\pi_t}}\Bigg] + \underbrace{\sqrt{\frac{A}{1-\gamma} \cdot \varepsilon_{\mathrm{stat}}}}_{\text{penalty for "near" optimism}} \tag{2}$$

From here, we want to perform a change-of-measure/distribution-shift argument, to translate from $d_P^{\pi_t}$ to the training distribution under which we collected the data. If we can do this, the quadratic form in the bonus term will be poly(d) since we'll get something like $\mathrm{tr}(\Sigma^{-1}\Sigma) = d$. Meanwhile the total variation term will be bounded by $\varepsilon_{\mathrm{stat}}$ since we will be on the training distribution.

The distribution shift argument is quite similar to what is needed to prove (1), except that for (1) we work primarily in the learned model $\hat{P}_t$ while for (2) we work in the true environment. Let us focus on (1) for now. Proving this essentially requires two steps. First, by simulation lemma we have

$$\begin{aligned} J(\pi; \hat{P}_t, r + b_t) - J(\pi; P, r) &\geq \frac{1}{1-\gamma}\mathbb{E}_{(s,a)\sim d^\pi_{\hat{P}_t}}\left[b_t(s,a) + \gamma\mathbb{E}_{s'\sim\hat{P}_t(s,a)}[V^\pi_{P,r}(s')] - \gamma\mathbb{E}_{s'\sim P(s,a)}[V^\pi_{P,r}(s')]\right] \\ &\geq \frac{1}{1-\gamma}\mathbb{E}_{(s,a)\sim d^\pi_{\hat{P}_t}}\left[b_t(s,a) - \|\hat{P}_t(s,a) - P(s,a)\|_{\mathrm{TV}}\right] \end{aligned}$$

Notice how this is quite similar to the bound in (2), except that the expectation is in the learned model $\hat{P}_t$. The key idea is to use that $\hat{P}_t$ itself is a low rank MDP to apply a distribution shift argument on the second term:

$$\begin{aligned}\left|\mathbb{E}_{(s,a)\sim d^{\pi}_{\hat{P}_t}}\left[\|\hat{P}_t(s,a)-P(s,a)\|_{\mathrm{TV}}\right]\right| &\lesssim \left|\mathbb{E}_{(\tilde{s},\tilde{a})\sim d^{\pi}_{\hat{P}_t,\mathrm{prev}}}\left\langle \hat{\phi}_t(\tilde{s},\tilde{a}), \int \hat{\mu}_t(s)\pi(a\mid s)\|\hat{P}_t(s,a)-P(s,a)\|_{\mathrm{TV}}\right\rangle\right| \\ &\lesssim \left|\mathbb{E}_{(\tilde{s},\tilde{a})\sim d^{\pi}_{\hat{P}_t,\mathrm{prev}}}\|\hat{\phi}_t(\tilde{s},\tilde{a})\|_{\Sigma_t^{-1}}\cdot\|\int \hat{\mu}_t(s)\pi(a\mid s)\|\hat{P}_t(s,a)-P(s,a)\|_{\mathrm{TV}}\|_{\Sigma_t}\right|\end{aligned}$$

Now we can bound the second term by the model error on the training distribution, since

$$\begin{aligned}\|\int \hat{\mu}_t(s)\pi(a\mid s)\|\hat{P}_t(s,a)-P(s,a)\|_{\mathrm{TV}}\|^2_{\Sigma_t} &\lesssim \sum_{D_t}\left\langle \hat{\phi}_t(s,a), \int \hat{\mu}_t(s')\pi(a'\mid s')\|\hat{P}_t(s',a')-P(s',a')\|_{\mathrm{TV}}\right\rangle^2 \\ &= \sum_{D_t}\left(\mathbb{E}_{s'\sim\hat{P}_t(s,a)}\pi(a'\mid s')\|\hat{P}_t(s',a')-P(s',a')\|_{\mathrm{TV}}\right)^2 \\ &\lesssim \sum_{D_t}\left(\mathbb{E}_{s'\sim P(s,a)}\pi(a'\mid s')\|\hat{P}_t(s',a')-P(s',a')\|_{\mathrm{TV}}\right)^2 + O(t\varepsilon_{\mathrm{stat}}) \\ &\lesssim \sum_{D_t}\mathbb{E}_{s'\sim P(s,a)}\pi(a'\mid s')\|\hat{P}_t(s',a')-P(s',a')\|^2_{\mathrm{TV}} + O(t\varepsilon_{\mathrm{stat}}) \\ &\lesssim A\sum_{D_t}\mathbb{E}_{s'\sim P(s,a),a'\sim\mathrm{Unif}(\mathcal{A})}\|\hat{P}_t(s',a')-P(s',a')\|^2_{\mathrm{TV}} + O(t\varepsilon_{\mathrm{stat}}) \\ &= A\sum_{D'_t}\|\hat{P}_t(s,a)-P(s,a)\|^2_{\mathrm{TV}} + O(t\varepsilon_{\mathrm{stat}}) = O(At\varepsilon_{\mathrm{stat}})\end{aligned}$$

Here the first inequality expands the Σ_t norm (we are omitting the regularization term) and then in the second inequality we use that $\hat{P}_t$ and P are similar on the training set D_t. The third inequality is a direct application of Jensen's inequality, then we perform an importance weighting argument, and finally we use that $\hat{P}_t$ and P are similar on the training set D'_t (which is one step ahead). Note that collecting data from two different steps is quite important here. Finally, we can establish (1) by setting the coefficient β in the elliptical bonus to be roughly $At\varepsilon_{\mathrm{stat}}$, so that it cancels out with the covariance-norm on $d^{\pi}_{\hat{P}_{t,\mathrm{prev}}}$. (The error term in (1) captures all the regularization effects as well as the fact that $d^{\pi}_{\hat{P}_{t,\mathrm{prev}}}$ differs from $d^{\pi}_{\hat{P}_t}$ by roughly the initial state.)

The above derivation highlights the key steps in this part of the analysis, but is certainly not tracking all of the terms. If we do this more carefully, we'll obtain (1) and to obtain (2), we perform a similar sequence of steps, which will result in a elliptical potential *in the true features* $\phi^\star$. This potential only appears in the analysis, but it will allow us to track progress in a manner that is consistent across iterations. For this part, we'll get something like

$$\begin{aligned}&\mathbb{E}_{(s,a)\sim d^{\pi_t}_P}\left[b_t(s,a)+\frac{1}{1-\gamma}\|\hat{P}_t(s,a)-P(s,a)\|_{\mathrm{TV}}\right] \\ &\lesssim \mathbb{E}_{(s,a)\sim d^{\pi_t}_{P,\mathrm{prev}}}\|\phi^\star(s,a)\|_{\Sigma_t^{\star,-1}}\cdot\sqrt{t|\mathcal{A}|\mathbb{E}_{s,a\sim\rho_t}\left[b_t^2(s,a)+\frac{1}{(1-\gamma)^2}\|\hat{P}_t(s,a)-P(s,a)\|^2_{\mathrm{TV}}\right]}\end{aligned}$$

where $\Sigma_t^\star$ is the covariance matrix in the true features $\phi^\star$. The two terms under the square root are both relatively small, since the distribution ρ_t is precisely (a) how we calculate the bonus b_t and (b) how we fit the model $\hat{P}_t$. Putting things together we can show that this algorithm has a sample complexity that scales as

$$\tilde{O}\left(\frac{d^4A^2\log(|\Phi||\Upsilon|/\delta)}{(1-\gamma)^5\epsilon^2}\right),$$

which is quite reasonable.