Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)

Fast Feature Selection for Linear Value Function Approximation

Bahram Behzadian, Soheil Gharatappeh, Marek Petrik
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
{bahram, soheil, mpetrik}@cs.unh.edu

Abstract

Linear value function approximation is a standard approach to solving reinforcement learning problems with large state spaces. Since designing good approximation features is difficult, automatic feature selection is an important research topic. We propose a new method for feature selection that is based on a low-rank factorization of the transition matrix. Our approach derives features directly from high-dimensional raw inputs, such as image data. The method is easy to implement using SVD, and our experiments show that it is faster and more stable than alternative methods.

1 Introduction

Reinforcement learning (RL) methods typically use value function approximation to solve problems with large state spaces (Sutton and Barto 1998; Szepesvári 2010). The approximation makes it possible to generalize from a small number of samples to the entire state space. Perhaps the most common methods for value function approximation are neural networks and linear methods. Neural networks offer unparalleled expressibility in complex problems, but linear methods remain popular due to their simplicity, interpretability, ease of use, and low sample and computational complexity.

This work focuses on *batch reinforcement learning* (Lange, Gabel, and Riedmiller 2012). In batch RL, all domain samples are provided in advance as a batch, and it is impossible or difficult to gather additional samples. This is common in many practical domains. In medical applications, for example, it is usually too dangerous and expensive to run additional tests, and in ecological applications, it may take an entire growing season to obtain a new batch of samples.

Overfitting is a particularly difficult challenge in practical deployments of batch RL. Detecting that the solution overfits the available data can be complex. Using a regular test set does not work in RL because of the difference between the sampling policy and the optimized policy. Also, off-policy policy evaluation remains difficult in large problems (Jiang and Li 2015). As a result, a solution that overfits the training batch is often discovered only after it has been deployed and real damage has been done.

With linear approximation, overfitting occurs more easily when too many features are used. In this paper, we presents Fast Feature Selection (FFS), a new method that can effectively reduce the number of features in batch RL. To avoid confusion, we use the term *raw features* to refer to the natural features of a given problem. They could, for example, be the individual pixel values in video games or particular geographic observations in geospatial applications. Raw features are usually numerous, but each feature alone has a low predictive value. FFS constructs (rather than selects) a small set of useful features that are a linear combination of the provided raw features. The constructed features are designed to be used in concert with LSTD, LSPI, and other related batch RL methods.

FFS reduces the number of features by computing a low-rank approximation of the transition matrix after it is compressed using the available raw features. Low-rank matrix approximation and completion gained popularity from their use in collaborative filtering (Murphy 2012), but they have been also applied to reinforcement learning and other machine learning domains (Ong 2015; Cheng, Asamov, and Powell 2017; Rendle, Freudenthaler, and Schmidt-Thieme 2010). None of this prior work, however, computes a low-rank approximation of the compressed transition matrix.

Several feature selection methods for reducing overfitting in RL have been proposed previously, but none of them explicitly target problems with low-rank (compressed) transition probabilities. ℓ_1 regularization, popularized by the LASSO, has been used successfully in reinforcement learning (Kolter and Ng 2009; Petrik et al. 2010; Le, Kumaraswamy, and White 2017). ℓ_1 regularization assumes that only a few of the features are sufficient to obtain a good approximation. This is not a reasonable assumption when individual raw features are of a low quality.

Proto-value functions (Mahadevan and Maggioni 2007) use the spectral decomposition of the transition probability matrix or of a related random walk. Although the spectrum of a matrix is closely related to its rank, eigenvector-based methods provide weak approximation guarantees even when the majority of the eigenvalues are zero (Petrik 2007).

BEBFs and Krylov are other techniques that work well when the characteristic polynomial of the transition probability matrix is of a small degree (Parr et al. 2007; Petrik 2007); this property is unrelated to the matrix rank.

The closest prior method to FFS is LFD (Song et al. 2016). LFD works by computing 1) a linear encoder that maps the raw features of a state to a small-dimensional space and 2) a linear decoder that maps the small-dimensional representation back to the raw features. While LFD was not introduced as a low-rank approximation technique, we show that similarly to FFS, it introduces no additional error when the matrix of transition probabilities is low-rank. LFD, unfortunately, has several limitations. It involves solving a nonconvex optimization problem, is difficult to analyze, and provides no guidance for deciding on the right number of features to use.

As the main contribution, this paper proposes and analyzes FFS both theoretically and empirically. We derive new bounds that relate the singular values of the transition probability matrix to the approximation error. As a secondary contribution, we provide a new interpretation of LFD as a type of low-rank approximation method. We argue that FFS improves on LFD in terms of providing fast and predictable solutions, similar or better practical performance, and guidance on how many features should be selected.

The remainder of the paper is organized as follows. Section 2 summarizes the relevant properties of linear value function approximation in Markov decision processes. Section 3 describes FFS and new bounds that relate singular values of the compressed transition probability matrix to the approximation error. Section 4 then compares FFS with other feature construction algorithms, and, finally, the empirical evaluation in Section 5 indicates that FFS is a promising feature selection method.

2 Linear Value Function Approximation

In this section, we summarize the background on linear value function approximation and feature construction.

We consider a reinforcement learning problem formulated as a Markov decision process (MDP) with states $\mathcal{S}$, actions $\mathcal{A}$, transition probabilities $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, and rewards $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ (Puterman 2005). The value of $P(s, a, s')$ denotes the probability of transitioning to state s' after taking an action a in a state s. The objective is to compute a stationary policy π that maximizes the expected $\gamma-$discounted infinite-horizon return. It is well-known that the value function $\boldsymbol{v}^\pi$ for a policy π must satisfy the Bellman optimality condition (e.g., Puterman (2005)):

$$\boldsymbol{v}^\pi = \boldsymbol{r}^\pi + \gamma P^\pi \boldsymbol{v}^\pi \ , \tag{1}$$

where P^π and $\boldsymbol{r}^\pi$ are the matrix of transition probabilities and the vector of rewards, respectively, for the policy π.

Value function approximation becomes necessary in MDPs with large state spaces. The value function $\boldsymbol{v}^\pi$ can then be approximated by a linear combination of features $\phi_1, \ldots, \phi_k \in \mathbb{R}^{|\mathcal{S}|}$, which are vectors over states. Using the vector notation, an approximate value function $\tilde{\boldsymbol{v}}^\pi$ can be expressed as:

$$\tilde{\boldsymbol{v}}^\pi = \Phi \, \boldsymbol{w} \ ,$$

for some vector $\boldsymbol{w} = \{w_1, \ldots, w_k\}$ of scalar weights that quantify the importance of features. Here, Φ is the feature matrix of dimensions $|\mathcal{S}| \times k$; the columns of this matrix are the features ϕ_i.

Numerous algorithms for computing linear value approximation have been proposed (Sutton and Barto 1998; Lagoudakis and Parr 2003; Szepesvári 2010). We focus on fixed-point methods that compute the *unique* vector of weights $\boldsymbol{w}_\Phi^\pi$ that satisfy the projected Bellman equation (1):

$$\boldsymbol{w}_\Phi^\pi = \Phi^+ (\boldsymbol{r}^\pi + \gamma P^\pi \Phi \boldsymbol{w}_\Phi^\pi) \ , \tag{2}$$

where Φ^+ is the Moore-Penrose pseudo-inverse of Φ and $\Phi^+ = (\Phi^\intercal \Phi)^{-1} \Phi^\intercal$ when columns of Φ are linearly independent (e.g., Golub and Van Loan (2013)). This equation follows by applying the orthogonal projection operator $\Phi(\Phi^\intercal \Phi)^{-1}\Phi^\intercal$ to both sides of (1).

The following insight will be important when describing the FFS method. The fixed-point solution to (2) can be interpreted as a value function of an MDP with a *linearly compressed* transition matrix P_Φ^π and a reward vector $\boldsymbol{r}_\Phi^\pi$ (Parr et al. 2008; Szepesvári 2010):

$$\begin{aligned} P_\Phi^\pi &= (\Phi^\intercal \Phi)^{-1} \Phi^\intercal P^\pi \Phi = \Phi^+ P^\pi \Phi, \\ \boldsymbol{r}_\Phi^\pi &= (\Phi^\intercal \Phi)^{-1} \Phi^\intercal \boldsymbol{r}^\pi = \Phi^+ \boldsymbol{r}^\pi \ . \end{aligned} \tag{3}$$

The weights $\boldsymbol{w}_\Phi^\pi$ in (2) are equal to the value function for this compressed MDP. That is, $\boldsymbol{w}_\Phi^\pi$ satisfies the Bellman equation for the compressed MDP:

$$\boldsymbol{w}_\Phi^\pi = \boldsymbol{r}_\Phi^\pi + \gamma P_\Phi^\pi \boldsymbol{w}_\Phi^\pi \ . \tag{4}$$

In order to construct good features, it is essential to be able to determine their quality in terms of whether they can express a good approximate value function. The standard bound on the performance loss of a policy computed using, for example, approximate policy iteration can be bounded as a function of the Bellman error (e.g., Williams and Baird (1993)). To motivate FFS, we use the following result that shows that the Bellman error can be decomposed into the error in 1) the compressed rewards, and in 2) the compressed transition probabilities.

Theorem 1 (Song et al. 2016)**.** *Given a policy π and features Φ, the Bellman error of a value function $v = \Phi \boldsymbol{w}_\Phi^\pi$ satisfies:*

$$\mathrm{BE}_\Phi = \underbrace{(\boldsymbol{r}^\pi - \Phi \boldsymbol{r}_\Phi^\pi)}_{\Delta_r^\pi} + \gamma \underbrace{(P^\pi \Phi - \Phi P_\Phi^\pi)}_{\Delta_P^\pi} \boldsymbol{w}_\Phi^\pi \ .$$

We seek to construct a basis that minimizes both $\|\Delta_r^\pi\|_2$ and $\|\Delta_P^\pi\|_2$. These terms can be used to bound the ℓ_2 norm of Bellman error as:

$$\begin{aligned} \|\,\mathrm{BE}_\Phi\,\|_2 &\le \|\Delta_r^\pi\|_2 + \gamma \|\Delta_P^\pi\|_2 \|\boldsymbol{w}_\Phi^\pi\|_2 \le \\ &\le \|\Delta_r^\pi\|_2 + \gamma \|\Delta_P^\pi\|_F \|\boldsymbol{w}_\Phi^\pi\|_2 \ , \end{aligned} \tag{5}$$

where the second inequality follows from $\|X\|_2 \le \|X\|_F$.

The Bellman error (BE) decomposition in (5) has two main limitations. The first limitation is that it is expressed in terms of the ℓ_2 norm rather than ℓ_∞ norm, which is needed for standard Bellman residual bounds (Williams and Baird 1993). This can be addressed, in part, by using the weighted ℓ_2 norm bounds (Munos 2007). The second limitation of (5) is that it depends on $\|\boldsymbol{w}_\Phi^\pi\|_2$ besides the terms $\|\Delta_r^\pi\|_2, \|\Delta_P^\pi\|_2$ that we focus on. Since $\|\boldsymbol{w}_\Phi^\pi\|_2$ can be problem-dependent, the theoretical analysis of its impact on the approximation error is beyond the scope of this work.

3 FFS: A Fast Low-Rank Approximation for Feature Selection

In this section, we describe the proposed method for selecting features from a low-rank approximation of the transition probabilities. To simplify the exposition, we first introduce the method for the tabular case and then extend it to the batch RL setting with many raw features in Section 3.1.

Algorithm 1: TFFS: Tabular Fast low-rank Feature Selection

```
Input: Transition matrix $P$, rewards $\boldsymbol{r}$, and number of
       features $k+1$
1 Compute SVD decomposition of $P$: $P = U\Sigma V^\intercal$ ;
2 Assuming decreasing singular values in $\Sigma$, select the
    first $k$ columns of $U$: $U_1 \leftarrow [u_1, \ldots, u_k]$ ;
3 return Approximation features: $\Phi = [U_1, \boldsymbol{r}]$.
```

The Tabular Fast Feature Selection algorithm is summarized in Algorithm 1. Informally, the algorithm selects the top k left-singular vectors and the reward function for the features. Our error bounds show that including the reward function as one of the features is critical. It is not surprising that when the matrix P is of a rank at most k then using the first k left-singular vectors will result in no approximation error. However, such low-rank matrices are rare in practice. We now show that it is sufficient that the transition matrix P is close to a low-rank matrix for TFFS to achieve small approximation errors. In order to bound the error, let the SVD decomposition of P be $\text{SVD}(P) = U\Sigma V^\intercal$, where

$$U = [U_1 \quad U_2], \quad \Sigma = \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix}, \quad V = [V_1 \quad V_2] \ .$$

That implies that the transition probability matrix can be expressed as:

$$P = U_1\Sigma_1 V_1^\intercal + U_2\Sigma_2 V_2^\intercal \ .$$

Let matrix U_1 have k columns and let the singular values be ordered decreasingly. Then, Algorithm 1 generates $\Phi = [U_1, \boldsymbol{r}]$. The following theorem bounds the error regarding the largest singular value for a vector not included in the features.

Theorem 2. *Assuming k features Φ computed by Algorithm 1, the error terms in Theorem 1 are upper bounded as:*

$$\|\Delta_P\|_2 \le \|\Sigma_2\|_2, \\ \|\Delta_r\|_2 = 0 \ .$$

The proof of the theorem is deferred to Appendix A.1.

Theorem 2 implies that if we choose Φ in a way that the singular values in Σ_2 are zero (when the transition matrix is low rank), Δ_P would be zero. That means that for a matrix of rank k there is no approximation error because $\|\Delta_P\|_2 = 0$. More broadly, when the rank of the matrix is greater than k, the error is minimized by choosing the singular vectors with the greatest singular values. That means that TFFS chooses features Φ that minimize the error bound in Theorem 2.

3.1 Using Raw Features

Using TFFS in batch RL is impractical since the transition matrix and reward vector are usually too large and are not available directly. The values must instead be estimated from samples and the raw features.

Algorithm 2: FFS: Fast low-rank Feature Selection from raw features

```
Input: Sampled raw features $A$, next state of raw
       feature $A'$ , rewards $\boldsymbol{r}$, and number of features
       $k+1$
1 Estimate compressed transition probabilities
    $P_A = A^+ A'$ as in LSTD ;
2 Compute SVD decomposition of $P_A$: $P_A = U\Sigma V^\intercal$ ;
3 Compute compressed reward vector: $\boldsymbol{r}_A = A^+\boldsymbol{r}$ ;
4 Assuming decreasing singular values in $\Sigma$, select the
    first $k$ columns of $U$: $U_1 \leftarrow [u_1, \ldots, u_k]$ ;
5 return Approximation features: $\widehat{\Phi} = [U_1, \boldsymbol{r}_A]$.
```

As described in the introduction, we assume that the domain samples include a potentially large number of low-information raw features. We use A to denote the $n \times l$ *matrix of raw features*. As with Φ, each row corresponds to one state, and each column corresponds to one *raw* feature. The compressed transition matrix is denoted as $P_A = A^+ P A$ and compressed rewards are denoted as $\boldsymbol{r}_A = A^+\boldsymbol{r}$ and are computed as in (3). To emphasize that the matrix P is not available, we use $A' = PA$ to denote the expected value of features after one step. Using this notation, the compressed transition probabilities can be expressed as $P_A = A^+ A'$.

Algorithm 2 describes the FFS method that uses raw features. Similarly to TFFS, the algorithm computes an SVD of the transition matrix. Note that the features $\widehat{\Phi}$ are linear combinations of the raw features. To get the actual state features, it is sufficient to compute $A\widehat{\Phi}$ where $\widehat{\Phi}$ is the output of Algorithm 2. The matrix $\widehat{\Phi}$ represents features for P_A and is of a dimension $l \times k$ where l is the number of raw features in A. We omit the details for how the values are estimated from samples, as this is well known, and refer the interested reader to Lagoudakis and Parr; Johns, Petrik, and Mahadevan (2003; 2009).

Using raw features to compress transition probabilities and rewards is simple and practical, but it is also essential to understand the consequences of relying on these raw features. Because FFS computes features that are a linear combination of the raw features, they cannot express more complex value functions. FFS thus introduces additional error—akin to bias—but reduces sampling error—akin to variance. The following theorem shows that the errors due to our approximation and using raw features merely add up with no additional interactions.

Theorem 3. *Assume that the raw features A for P and computed features $\widehat{\Phi}$ for P_A are normalized, such that $\|A\|_2 = \|\widehat{\Phi}\|_2 = 1$. Then:*

$$\|\Delta_P^{A\widehat{\Phi}}\|_2 \le \|\Delta_P^A\|_2 + \|\Delta_{P_A}^{\widehat{\Phi}}\|_2 ,$$
$$\|\Delta_r^{A\widehat{\Phi}}\|_2 \le \|\Delta_r^A\|_2 + \|\Delta_{r_A}^{\widehat{\Phi}}\|_2 ,$$

where the superscript of Δ indicates the feature matrix for which the error is computed; for example $\Delta_{P_A}^{\widehat{\Phi}} = P_A\widehat{\Phi} - \widehat{\Phi}(P_A)_{\widehat{\Phi}}$.

The proof of the theorem is deferred to Appendix A.2.

Note that the normalization of features required in Theorem 3 can be achieved by multiplying all features by an appropriate constant, which is an operation that does not affect the approximate value function. Scaling features does, however, affect the magnitude of $\boldsymbol{w}_\Phi$, which, as we discuss above, is problem-specific and largely independent of the feature selection method used.

Perhaps one of the most attractive attributes of FFS is its simplicity and low computational complexity. Selecting the essential features only requires computing the singular value decomposition—for which many efficient methods exist—and augmenting the result with the reward function. As we show next, this simple approach is well-motivated by bounds on approximation errors.

We described FFS in terms of singular value decomposition and showed that when the (compressed) transition probability matrix has a low rank, the approximation error is likely to be small. Next, we describe the relationship between FFS and other feature selection methods in more detail.

4 Related Feature Selection Methods

In this section, we describe similarities and differences between FFS and related feature construction or selection methods.

Perhaps the best-known method for feature construction is the technique of *proto-value functions* (Mahadevan and Maggioni 2006; 2007). Proto-value functions are closely related to spectral approximations (Petrik 2007). This approximation uses the eigenvector decomposition of the transition matrix $P = S\Lambda S^{-1}$, where S is a matrix with eigenvectors as its columns and Λ is a diagonal matrix with eigenvalues that are sorted from the largest to the smallest. The first k columns of S are then used as the approximation features. As with our FFS method, it is beneficial to augment these features with the reward vector. We will refer to this method as EIG+R in the numerical results. Surprisingly, unlike with FFS, which uses top k left-singular vectors, using the top k eigenvectors does not guarantee zero Bellman residual even if the rank of P is less than k.

Using the Krylov subspace is another feature selection approach (Petrik 2007) which has also been referred to as BEBF (Parr et al. 2007; 2008). The Krylov subspace $\mathcal{K}$ is spanned by the images of $\boldsymbol{r}$ under the first k powers of P (starting from $P^0 = \mathbf{I}$):

$$\mathcal{K}_k(P, \boldsymbol{r}) = \operatorname{span}\{\boldsymbol{r}, P\boldsymbol{r}, \ldots, P^{k-1}\boldsymbol{r}\} .$$

Petrik (2007) shows that when k is equal to the degree of the minimal polynomial, the approximation error is zero. Krylov methods are more likely to work in different problem settings than either EIG+R or FFS and can be easily combined with them.

Algorithm 3: LFD: Linear Feature Discovery for a fixed policy π (Song et al. 2016).

```
1 $D_0 \leftarrow \mathrm{random}(k, l)$;
2 $i \leftarrow 1$;
3 while Not Converged do
4     $E_i \leftarrow A^+ A' D_{i-1}$ ;
5     $D_i \leftarrow (AE_i)^+ A'$;
6     $i \leftarrow i + 1$ ;
7 end
8 return $E_k$ // Same role as $\widehat{\Phi}$ in FFS.
```

Finally, Linear Feature Discovery (LFD) (Song et al. 2016) is a recent feature selection method that is closely related to FFS. Algorithm 3 depicts a simplified version of the LFD algorithm, which does not consider the reward vector and approximates the value function instead of the Q-function for a fixed policy π. Recall that A is the matrix of raw features and $A' = P^\pi$.

LFD is motivated by the theory of *predictive optimal feature encoding*. A low-rank encoder E^π is *predictively optimal* if there exist decoders D_s^π and D_r^π such that:

$$AE^\pi D_s^\pi = P^\pi A , \qquad AE^\pi D_r^\pi = r^\pi .$$

When an encoder and decoder are predictively optimal, then the Bellman error is 0 (Song et al. 2016). Unfortunately, it is almost impossible to find problems in practice in which a predictively optimal controller exists. No bounds on the Bellman error are known when a controller is merely close to predictively optimal. This is in contrast with the bounds in Theorems 2 and 3 that hold for FFS.

Although LFD appears to be quite different from FFS, our numerical experiments show that it computes solutions that are similar to the solutions of FFS. We argue that LFD can be interpreted as a coordinate descent method for computing the following low-rank approximation problem:

$$\min_{E\in\mathbb{R}^{l\times k}, D\in\mathbb{R}^{k\times l}} \|AED - A'\|_F^2 . \tag{6}$$

This is because the iterative updates of E_i and D_i in Algorithm 3 are identical to solving the following optimization problems:

$$E_i \leftarrow \arg\min_{E\in\mathbb{R}^{l\times k}} \|AED_{i-1} - A'\|_F^2$$
$$D_i \leftarrow \arg\min_{D\in\mathbb{R}^{k\times l}} \|AE_iD - A'\|_F^2$$

The equivalence follows directly from the orthogonal projection representation of linear regression. This kind of coordinate descent is a very common heuristic for computing low-rank matrix completions (Hastie et al. 2015). Unfortunately, the optimization problem in (6) is *non-convex* and coordinate descent, like LFD, may only converge to a local optimum, if at all. Simple algebraic manipulation reveals that any set of k singular vectors represents a local minimum of LFD. Finally, we are not aware of any method that can solve (6) optimally.

Similarly to LFD, FFS solves the following optimization problem:

$$\min_{E\in\mathbb{R}^{l\times k}, D\in\mathbb{R}^{k\times l}} \|ED - A^+A'\|_F^2 \,. \tag{7}$$

This fact follows readily from the SVD decomposition of A^+A' and the fact that the Frobenius norm is equal to the L_2 norm of the the singular values (Hastie, Tibshirani, and Friedman 2009; Golub and Van Loan 2013).

Note that when using tabular features ($A = \mathbf{I}$) the optimization problems (6) and (7) are identical. For any other raw features, there are two reasons for preferring (7) over (6). First, FFS is much easier to solve both in theory and in practice. Second, as Theorem 3 shows, the approximation error of FFS is simply additive to the error inherent to the raw features. No such property is known for LFD. In the next section, we compare the two methods numerically.

5 Empirical Evaluation

In this section, we empirically evaluate the quality of features generated by FFS both with and without using raw features. We focus on a comparison with LFD which was empirically shown to outperform radial basis functions (RBFs) (Lagoudakis and Parr 2003), random projections (Ghavamzadeh et al. 2010), and other methods (Song et al. 2016).

We first compare the quality of solutions on a range of synthetic randomly-generated problems. The goal is to ensure that the methods behave similarly regardless of the number of samples, or the type of raw features that are used. Then, we use an image-based version of the cart-pole benchmark, used previously by Song et al. (2016), to evaluate FFS in more complex settings. This problem is used to evaluate both the solution quality and the computational complexity of the methods.

5.1 Synthetic Problems

To compare FFS to other common approaches in feature selection, we start with small policy evaluation problems. Since the policy is fixed throughout these experiments, we omit all references to it. The data matrix $A \in \mathbb{R}^{n\times l}$ only contains the states where n denotes the number of states and l the length of each *raw* feature, with $\Phi \in \mathbb{R}^{n\times k}$ using k features.

The synthetic problems that we use throughout this section have 100 states. The rewards $\boldsymbol{r} \in \mathbb{R}^{100}$ are generated uniformly randomly from the interval of $[-500, 500)$. The stochastic transition probabilities $P \in [0,1)^{100\times 100}$ are generated from the uniform Dirichlet distribution. To ensure that the rank of P is at most 40, we compute P as a product $P = XY$, where X and Y are small-dimensional. The discount factor we use is $\gamma = 0.95$.

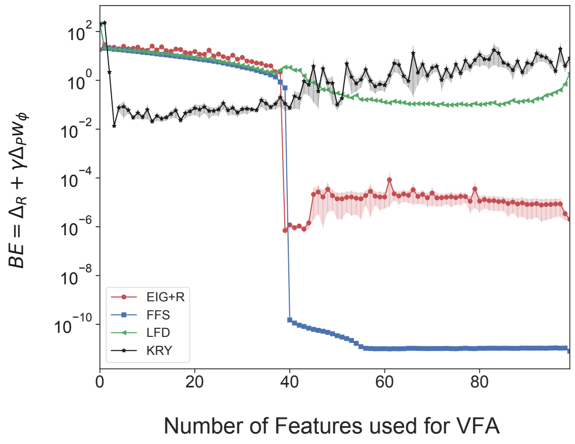

Figure 1: Bellman error for the exact solution. The transition matrix is 100×100 and has a low rank with $\operatorname{rank}(P) = 40$. The Input matrix is $A = \mathbf{I}$ an identity matrix.

We now proceed by evaluating FFS for both tabular and image-based features. For the sake of consistency, we use FFS to refer to both TFFS in a tabular case and FFS when raw features are available. To evaluate the quality of the value function approximation, we compute the Bellman residual of the fixed-point value function, which is a standard metric used for this purpose. Recall that the Bellman error can be expressed as

$$\mathrm{BE} = \Delta_{\boldsymbol{r}} + \gamma\Delta_P \boldsymbol{w}_\Phi,$$

where $\boldsymbol{w}_\Phi$ is the value-function given in (4). All results we report in this section are an average of 100 repetitions of the experiments. All error plots show the ℓ_2 norm of the Bellman error in logarithmic scale.

Case 1: Tabular raw features. In this case, the true transition probabilities P and the reward function $\boldsymbol{r}$ are known, and the raw features are an identity matrix: $A = \mathbf{I}$. Therefore all computations are made concerning the precise representations of the underlying MDP.

This is the simplest setting, under which SVD simply reduces to a direct low-rank approximation of the transition probabilities. That is, the SVD optimization problem reduces to:

$$\min_{U_1\in\mathbb{R}^{n\times k}} \min_{\Sigma_1 V_1^\mathsf{T}\in\mathbb{R}^{k\times n}} \|U_1\Sigma_1 V_1^\mathsf{T} - P\|_F^2 \,.$$

Similarly, the constructed features will be $\Phi = U_1$. In case of FFS, we can simply add the reward vector to feature's set $\Phi = [U_1, \boldsymbol{r}]$. EIG+R and KRY are implemented as described in Petrik; Parr et al. (2007; 2008). In case of EIG+R approach, we use the eigenvectors of P as basis functions, and then $\boldsymbol{r}$ is included. For Krylov basis we calculate $\Phi = \mathcal{K}_k(P, \boldsymbol{r})$.

Figure 1 depicts the Bellman error for the exact solution when the number of features used for value function varies from 0 to 100. Note that the Bellman error of FFS is zero for

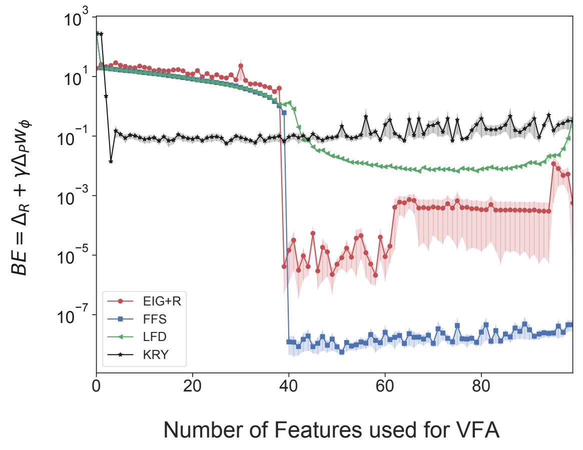

Figure 2: Bellman error for the approximate solution. The transition matrix is 100×100 and has a low rank with $\mathrm{rank}(P) = 40$. The Input matrix is A = random binary matrix.

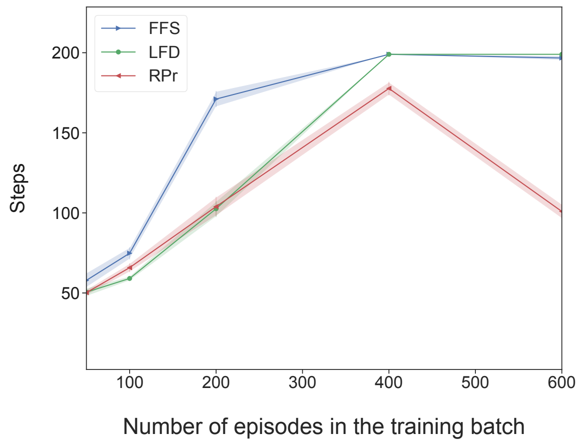

Figure 3: Average number of balancing steps with $k = 50$.

$k \geq 40$. This is because the rank of P is 40, and according to Theorem 2 the first 40 features obtained by FFS are sufficient to get $\|\mathrm{BE}\|_2 = 0$. This experiment shows FFS is robust and generally outperforms other methods. The only exception is the Krylov method which is more effective when few features are used but is not numerically stable with more features. The Krylov method could be combined relatively easily with FFS to get the best of both bases.

Case 2: Image-based raw features. In this case, the raw features A are not tabular but instead simulate an image representation of states. So the Markov dynamics are experienced only via samples and the functions are represented using an approximation scheme. The matrix A is created by randomly allocated zeros and ones similar to the structure of a binary image. We use LSTD to compute the approximate value function, as described in Section 2.

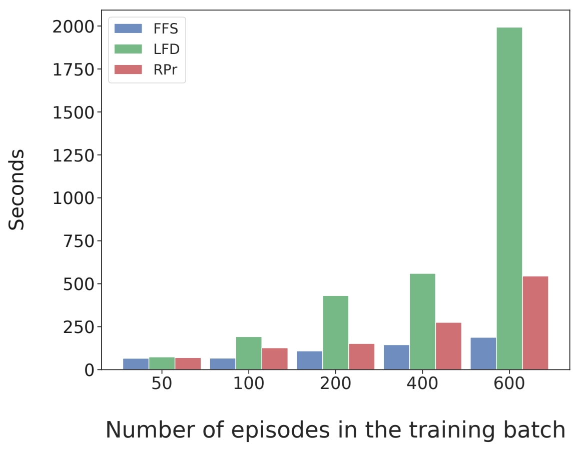

Figure 4: Mean running time for estimating the Q-function with $k = 50$.

The SVD optimization problem now changes as described in Section 3.1. The constructed features will be $\Phi = A\widehat{\Phi}$ and for FFS we include the reward predictor vector $[P_A, \boldsymbol{r}_A]$ in the optimization problem. In the case of the EIG+R method, we multiply the eigenvectors of P_A and $\boldsymbol{r}_A$ with the raw features. The Krylov basis is constructed as: $\Phi = A\,\mathcal{K}_k(P_A, \boldsymbol{r}_A)$ where $\mathcal{K}_k$ is the k-th order Krylov operator.

Figure 2 compares the Bellman error for the approximate solution. FFS again outperforms other methods. LFD is unstable when the number of features exceeds the rank of P, and sometimes it is not possible to obtain the pseudo-inverse of matrix AE.

It is worth noting that this section deals with very small MDPs with only about 100 states. It is expected to see a more significant gap in Bellman error of these methods when dealing with large MDPs with enormous and high-dimensional state spaces. In the next section, we compare LFD and FFS with random projection approach using a more significant and more challenging benchmark problem.

5.2 Cart-Pole

These experiments evaluate the similarity between the linear feature encoding (LFD) approach and the fast feature selection (FFS) method on a modified version of cart-pole, which is a standard reinforcement learning benchmark problem. We use random projections (Ghavamzadeh et al. 2010) as the baseline. The controller must learn a good policy by merely observing the *image* of the cart-pole without direct observations of the angle and angular velocity of the pole. This problem is large enough that the computational time plays an important role, so we also compare the computational complexity of the three methods.

Note that this is a control benchmark, rather than value approximation for a fixed policy. Since the goal of RL is to optimize a policy, results on policy optimization are often more meaningful than just obtaining a small Bellman residual which is not sufficient to guarantee that a good policy will be computed (Johns, Petrik, and Mahadevan 2009).

To obtain training data, we collect the specified number of trajectories with the starting angle and angular velocity sampled uniformly on $[-0.1, 0.1]$. The cart position and velocity are set to zero at each episode.

The algorithm was given three consecutive, rendered, gray-scale images of the cart-pole. Each image is down sampled to 39×50 pixels, so the raw state is a $39 \times 50 \times 3 = 5850-$dimensional vector. We chose three frames to preserve the Markov property of states without manipulating the cart-pole simulator in OpenAI Gym. We used $k = 50$ features for all methods.

We follow a setup analogous to Song et al. (2016) by implementing least-squares policy iteration (Lagoudakis and Parr 2003) to obtain the policy. The training data sets are produced by running the cart for $[50, 100, 200, 400, 600]$ episodes with a random policy. We then run policy iteration to iterate up to 50 times or until there is no change in the $A' = P^{\pi} A$ matrix.

The state of the pole in the classic cart-pole problem is described by its angle and angular velocity. However, in the image-based implementation, the agent does not observe this information. Song et al. (2016) chose two successive frames to show the state of the pole. To preserve the Markovian property of the state, they had to modify the simulator and force the angular velocity to match the change in angle per time step $\dot{\theta} = (\theta' - \theta)/\delta t$. We, instead, use the standard simulator from OpenAI Gym and choose the last three consecutive frames rather than two. Three consecutive frames are sufficient to infer θ and $\dot{\theta}$ and construct a proper Markov state. Intriguingly, no linear feature construction methods work well in the original problem definition when using only the last two frames.

The performance of the learned policy is reported for 100 repetitions to obtain the average number of balancing steps. Figure 3 displays the average number of steps during which the pole kept its balance using the same training data sets. For each episode, a maximum of 200 steps was allowed to run. This result shows that on the larger training sets the policies obtained from FFS and LFD are quite similar, but with small training sets, FFS shows a better performance. Both methods outperform random projection (RPr) significantly.

Figure 4 depicts the average running time of LFD and FFS for obtaining the value function with $k = 50$. The computation time of FFS grows very slowly as the number of training episodes increases; at 600 training episodes, the maximum number of episodes tested, FFS is 10 times faster than LFD. Therefore, LFD would likely be impractical in large problems with many training episodes.

Both FFS and LFD implementations use randomized SVD in all computations including the computation of pseudo-inverses. The result is usually very close to truncated singular value decomposition. Randomized SVD is fast on large matrices on which we need to extract only a small number of singular vectors. It reduces the time to compute k top singular values for an $m \times n$ matrix from $O(mnk)$ to $O(mn \log(k))$ (Halko, Martinsson, and Tropp 2011).

In comparison to black box methods such as neural networks, linear value functions are more interpretable: their behavior is more transparent from an analysis standpoint and

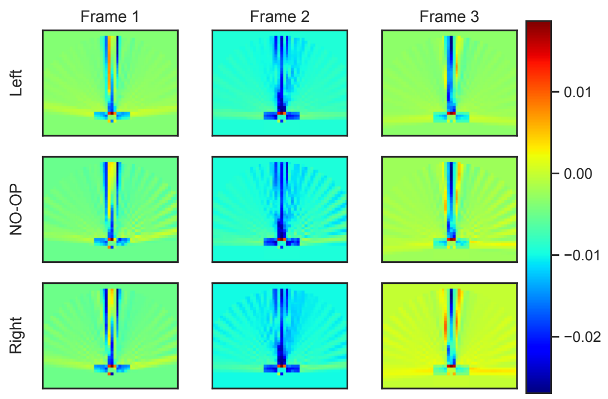

Figure 5: Value function in jet color-map.

feature engineering standpoint. It is comparatively simple to gain some insight into the reasons for which a particular choice of features succeeds or fails. When the features are normalized, the magnitude of each parameter is related to the importance of the corresponding feature in the approximation (Lagoudakis and Parr 2003).

Figure 5 shows the learned coefficients of Q-function for three actions (left, right and no-op) using color codes. The q-values are obtained by the inner product of raw features (3-frames of cart-pole) and these coefficients. They are computed by the FFS method from 400 training episodes with random policy. In this experiment, the raw images, taken from the cart-pole environment in OpenAI Gym toolkit, are preprocessed, converted to gray-scale and normalized. Therefore, the pole in the raw images is in black, and the value of black pixels are close to zero. Other areas in the raw features are in white, so these pixel values are closer to one. It is interesting to see how the linear value function captures the dynamics of the pole (the cart is stationary). If the pole is imbalanced, the value function is smaller since the blue area in Figure 5 represents negative scalars.

6 Conclusion

We propose FFS, a new feature construction technique that computes a low-rank approximation of the transition probabilities. We believe that FFS is a promising method for feature selection in batch reinforcement learning. It is very simple to implement, fast to run, and relatively easy to analyze. A particular strength of FFS is that it is easy to judge its effectiveness by singular values of features not included in the approximation (Theorem 2).

In terms of future work, it is important to study how common it is for reinforcement learning to exhibit the low-rank property exploited by FFS. This is most likely to vary depending on the domain, problem size, and amount of batch data. We suspect that in most applications, a combination of several methods, such as FFS and BEBF, is likely to yield the best and most robust results. Finally, it would be interesting to study the impact of FFS on finite-sample bounds and robustness in RL.

7 Acknowledgments

We thank the anonymous reviewers for helpful comments and suggestions. This work was supported by the National Science Foundation under Grant No. IIS-1717368.

A Technical Proofs

A.1 Proof of Theorem 2

Proof of Theorem 2. From the definition of Δ_P and P_Φ we get the following equality:

$$\Delta_P = U\Sigma V^\intercal U_1 - U_1(U_1^\intercal U_1)^{-1}U_1^\intercal U\Sigma V^\intercal U_1 \; .$$

Recall that singular vectors are orthonormal which implies that $(U_1^\intercal U_1)^{-1} = \mathbf{I}$ and $U_1^\intercal U = [\mathbf{I}_1 \quad 0]$. Substituting these terms into the equality above, we get:

$$\begin{aligned}\|\Delta_P\|_2 &= \|(U\Sigma V^\intercal - U_1\Sigma_1 V_1^\intercal)U_1\|_2 \\ &\le \|U\Sigma V^\intercal - U_1\Sigma_1 V_1^\intercal\|_2 \|U_1\|_2 \; .\end{aligned}$$

Simple algebraic manipulation shows that $\|U\Sigma V^\intercal - U_1\Sigma_1 V_1^\intercal\|_2 = \|\Sigma_2\|_2$ and $\|U_1\|_2 = 1$ because U is an unitary matrix. This establishes the inequality for Δ_P; the result for Δ_r follows directly from the properties of orthogonal projection since $\boldsymbol{r}$ itself is included in the features. □

A.2 Proof of Theorem 3

Proof of Theorem 3. We show the result only for Δ_P; the result for Δ_r follows similarly. From the definition of Δ,

$$\left\|\Delta_P^{A\widehat{\Phi}}\right\|_2 = \left\|PA\widehat{\Phi} - A\widehat{\Phi}P_{A\widehat{\Phi}}\right\|_2 .$$

Now, by adding a zero $(AP_A\widehat{\Phi} - AP_A\widehat{\Phi})$ and applying the triangle inequality, we get:

$$\begin{aligned}\left\|\Delta_P^{A\widehat{\Phi}}\right\|_2 &= \left\|PA\widehat{\Phi} - AP_A\widehat{\Phi} + AP_A\widehat{\Phi} - A\widehat{\Phi}P_{A\widehat{\Phi}}\right\|_2 \le \\ &\le \left\|PA\widehat{\Phi} - AP_A\widehat{\Phi}\right\|_2 + \left\|AP_A\widehat{\Phi} - A\widehat{\Phi}P_{A\widehat{\Phi}}\right\|_2 .\end{aligned}$$

Given $(A\widehat{\Phi})^+ = \widehat{\Phi}^+ A^+$ and the property of the compressed transition matrix in Equation (3) we can show:

$$\begin{aligned}(P_A)_{\widehat{\Phi}} - P_{A\widehat{\Phi}} &= \widehat{\Phi}^+ P_A \widehat{\Phi} - (A\widehat{\Phi})^+ PA\widehat{\Phi} \\ &= \widehat{\Phi}^+(P_A - A^+PA)\widehat{\Phi} = 0\end{aligned}$$

$$\begin{aligned}\left\|\Delta_P^{A\widehat{\Phi}}\right\|_2 &\le \|PA - AP_A\|_2 \left\|\widehat{\Phi}\right\|_2 + \\ &+ \left\|P_A\widehat{\Phi} - \widehat{\Phi}(P_A)_{\widehat{\Phi}}\right\|_2 \|A\|_2 \; .\end{aligned}$$

The theorem then follows directly from algebraic manipulation and the fact that the features are normalized. □

References

Cheng, B.; Asamov, T.; and Powell, W. B. 2017. Low-rank value function approximation for co-optimization of battery storage. *IEEE Transactions on Smart Grid* 3053.

Ghavamzadeh, M.; Lazaric, A.; Maillard, O.; and Munos, R. 2010. LSTD with random projections. In *Advances in Neural Information Processing Systems (NIPS)*, 721–729.

Golub, G. H., and Van Loan, C. F. 2013. *Matrix computations*.

Halko, N.; Martinsson, P.-G.; and Tropp, J. A. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review* 53(2):217–288.

Hastie, T.; Mazumder, R.; Lee, J.; and Zadeh, R. 2015. Matrix completion and low-rank svd via fast alternating least squares. *Journal of Machine Learning Research* 16:3367–3402.

Hastie, T.; Tibshirani, R.; and Friedman, J. 2009. *The elements of statistical learning*. 2nd edition.

Jiang, N., and Li, L. 2015. Doubly robust Off-policy value evaluation for reinforcement learning. In *International Conference on Machine Learning (ICML)*.

Johns, J.; Petrik, M.; and Mahadevan, S. 2009. Hybrid least-squares algorithms for approximate policy evaluation. *Machine Learning* 76(2):243–256.

Kolter, J. Z., and Ng, A. Y. 2009. Regularization and feature selection in least-squares temporal difference learning. In *International Conference on Machine Learning (ICML)*, 521–528. ACM.

Lagoudakis, M. G., and Parr, R. 2003. Least-squares policy iteration. *Journal of machine learning research* 4(Dec):1107–1149.

Lange, S.; Gabel, T.; and Riedmiller, M. 2012. Batch reinforcement learning. In *Reinforcement learning*. Springer. 45–73.

Le, L.; Kumaraswamy, R.; and White, M. 2017. Learning sparse representations in reinforcement learning with sparse coding. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2067–2073.

Mahadevan, S., and Maggioni, M. 2006. Value function approximation with diffusion wavelets and laplacian eigenfunctions. In *Advances in Neural Information Processing Systems (NIPS)*, 843–850.

Mahadevan, S., and Maggioni, M. 2007. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research* 8:2169–2231.

Munos, R. 2007. Performance bounds in Lp-norm for approximate value iteration. *SIAM journal on control and optimization* 46(2):541–561.

Murphy, K. P. 2012. *Machine learning: a probabilistic perspective*. MIT press.

Ong, H. Y. 2015. Value function approximation via low-rank models. *arXiv preprint arXiv:1509.00061*.

Parr, R.; Painter-Wakefield, C.; Li, L.; and Littman, M. 2007. Analyzing feature generation for value-function approximation. In *International Conference on Machine Learning (ICML)*, 737–744.

Parr, R.; Li, L.; Taylor, G.; Painter-Wakefield, C.; and Littman, M. L. 2008. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 752–759.

Petrik, M.; Taylor, G.; Parr, R.; and Zilberstein, S. 2010. Feature selection using regularization in approximate linear programs for Markov decision processes. In *International Conference on Machine Learning (ICML)*.

Petrik, M. 2007. An analysis of Laplacian methods for value function approximation in MDPs. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 35, 2574–2579.

Puterman, M. L. 2005. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, Inc.

Rendle, S.; Freudenthaler, C.; and Schmidt-Thieme, L. 2010. Factorizing personalized Markov chains for next-basket recommendation. In *International Conference on World Wide Web (WWW)*, 811–820.

Song, Z.; Parr, R. E.; Liao, X.; and Carin, L. 2016. Linear feature encoding for reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, 4224–4232.

Sutton, R. S., and Barto, A. 1998. *Reinforcement learning*.

Szepesvári, C. 2010. *Algorithms for Reinforcement Learning*. Morgan & Claypool Publishers.

Williams, R. J., and Baird, L. C. 1993. Tight performance bounds on greedy policies based on imperfect value functions. Technical report, College of Computer Science, Northeastern University.