Proto Value Functions in continuous domains

Adrien Vacher
ENS Paris-Saclay
adrien.vacher@ensae.fr

Abstract

In this project, we implemented an efficient LSPI algorithm and a Proto-Value-Function (PVF) generator for continuous domains. We tried to reproduce some results of Mahadevan (5), benchmarking our solution on 3 classic control problems and comparing the performance of PVFs against Radial Basis Functions, a popular choice of basis function. As expected, PVFs were superior to RBF.

1 Introduction

We place ourselves in the context of an MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. Our goal is to find a policy π which maximizes the sum of discounted rewards $\sum r_t^\gamma$. In this project we focus on a particular *offline* resolution method of the problem called the Least Square Policy Iteration. The LSPI algorithm as presented by Lagoudakis and Parr (4) requires a set of *basis* functions ϕ to linearly approximate the Q function. The choice of features significantly affects perfomance (3) and thetrefore, they need to be carefully "hand-crafted", depending of the dynamics and structure of the problem. However, in the case where the environment is not well known, choosing a proprer basis becomes more delicate and can lead to bad results. To face this issue, Mahadevan (5) proposed a framework called Proto-Value functions aiming to produce a basis automatically and without prior knowledge. This framework exploits the diffusion properties of the environment throught the discretized Laplacian operator.

In this project, we will start by presenting in Section 2 the PVF framework and provide an experimental insight of their efficiency to represent the environment. In Section 3, we will review the LSPI algorithm in details and propose a slight modification to improve computationnal efficiency. Finally, in Section 4, we will present the results obtained by LSPI with PVF on 3 classic control tasks in comparasion with Radial Basis Function, a popular basis. Even if PVF proved to be more performant than RBF, it eventually failed to solve the Mountain Car problem.

Our main contributions were:

- the implementation of PVF for continuous domains
- the implementation of a highly parallelised LSPI algorithm which can be applicated to a wider setting where basis functions do not explicitly depend on the actions

2 Proto-Value Functions

In this section, we first describe the Laplacian operator and its link to our problem. Then we provide details about the implementation of such an operator when dealing with the ineherent discrete nature of obeservations. Finally we show with numerical experiments how it can capture the state space manifold $\mathcal{S}$.

2.1 The Laplacian operator

We place ourselves on a smooth manifold $\mathcal{M}$. The laplacian operator $\mathcal{L}$ operates on regular functions of $\mathcal{M}$ and is such that for every real valued function $\phi \in \mathcal{C}^\infty, \mathcal{L}(\phi) = \text{div}(\nabla \phi)$. If $\mathcal{M} = \mathbb{R}^n$, we

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

have that $\mathcal{L}(\phi) = \sum_{i=1}^{n} \frac{\partial^2 \phi}{\partial^2 x_i}$, which can be seen as a form of measure of convexity. This operator shows strong link with the Fourrier operator $\mathcal{F}$ and for cause: they are both self adjoint and compact (see Saito (6)). So, just as sine and cosine functions with "standard" functions, every $\phi \in L^2(\mathcal{M})$ can be well approximated by a linear combination of Laplacian's eigenfunctions. Pushing the comparison further, the smaller the associated eigenvalue is, the smoother the corresponding eigenfunction is.

Now back to our problem. Recall that LSPI aims to approximate $Q(s, a)$ (see Section 3). If a is fixed, $Q(., a)$ becomes a real valued function on the state manifold $\mathcal{S}$. Therefore, if we assume a certain smoothness of the state-action function, it seems natural to approximate $Q(., a)$ by $\sum_{i=1}^{M} \theta_i^a \phi_i$ where the ϕ_i are the ordered laplacian's eigenvectors, corresponding to the lowest "frequencies". At this point, we have to introduce a way to compute those ϕ_i.

2.2 Computing the Proto basis

In an offline setting as ours, we have a sample (s_i) of states from wich we must compute the laplacian's eigenfunctions. We have a double problem: not only we ignore the true manifold $\mathcal{S}$ but also, we must resolve numerically the functionnal equation $\mathcal{L}(\phi) = \lambda\phi$.

To approximate the manifold, we build a similarity graph on top of our N sampled states which is computed in the following manner: for each pair (s_i, s_j), we compute $K(s_i, s_j)$ where K is a kernel supposed to compute the manifold distance between s_i and s_j. For the implementation, we took $K(x, y) = \exp(-\frac{||x-y||^2}{\sigma^2})$ (standard kernel choice). Afterwards, we connect s_i to s_j if s_i is among the k nearest neighbors of s_j with k an hyperparameter. We store the weighted connections between states in a similarity matrix W that we symetrize ; W encodes the state manifold $\mathcal{S}$.

Now we define L the normalized Laplace-Beltrami operator (1) as $L = I - D^{-0.5} W D^{-0.5}$ where D is a diagonal matrix whose coefficients are the column-wise (or row-wise) sum of W. This operator is the discrete analogue of $\mathcal{L}$ and thanks to this duality, we have that the coordinates the $i - th$ eigenvector $v_i = (a_1, \cdot, a_N)$ correspond to the pointwise evaluation $(\phi_i(s_1), ..., \phi_i(s_N))$ of the states (s_i) by the $i - th$ eigenfunction of $\mathcal{L}$.

Finally, in order to approximate ϕ_i on all the state space, we use the fact that the operator $\mathcal{K}(\phi)(x) = \int K(x, y)\phi(y)dy$ commutes with $\mathcal{L}$ such that they share the same eigenfunctions and eigenvalues (see Saito (6)). The ϕ_i are such that $\int K(x, y)\phi_i(y)dy = \lambda_i \phi_i$. Using the Nyström interpolation method (see Mahadevan (5)) it comes that:

$$\phi_i(x) = \frac{1}{1 - \lambda_i} \sum_{s_j \sim x} \frac{K(x, s_j)}{\sqrt{d(x)d(s_j)}} \phi_i(s_j) \tag{1}$$

where $s_j \sim x$ denotes the neighbors of x and $d(x) = \sum_{s_j \sim x} K(x, s_j)$.

Thanks to these resutls, we were able to compute a continuous Proto-Value Function basis using only the NumPy library for linear algebra and SciPy for nearest neighbors computation. In the following paragraph, we demonstrate the abitilty of this impleted basis to capture structural properties of the environment, hoping to give a more practical insight of the formal statements above.

2.3 Numerical experiments

We place ourselves in a 100x100 grid space with thick cross-like walls and 4 doors in the middle of the walls. The doors can be seen as bottlenecks, separating 4 rooms.

On Figure 1, we plotted the first three eigenfunctions of the laplacian. They clearly separate the 4 rooms. It is interesting to point out that they are associated with almost nul eigenvalues. The fact that the eigenvalues are very low is because they are almost constant by blocks. The fact that they are very close to each other yields an intersting remark. As Canzani (2) mentionned, the eigenvalues encode for geometrical properties of the manifold. Since the eigenfunctions refer to the same property, they have the same value. The same remark holds on Figure 2 where the represented eigenfunctions encode the bottlenecks of the environment. Once again, their eigenvalues, $0.16, 0.17$ and 0.19, are close because they represent the same property.

Now that we have described the Proto-Value basis and its implementation, we can now plug them into the LSPI framework.

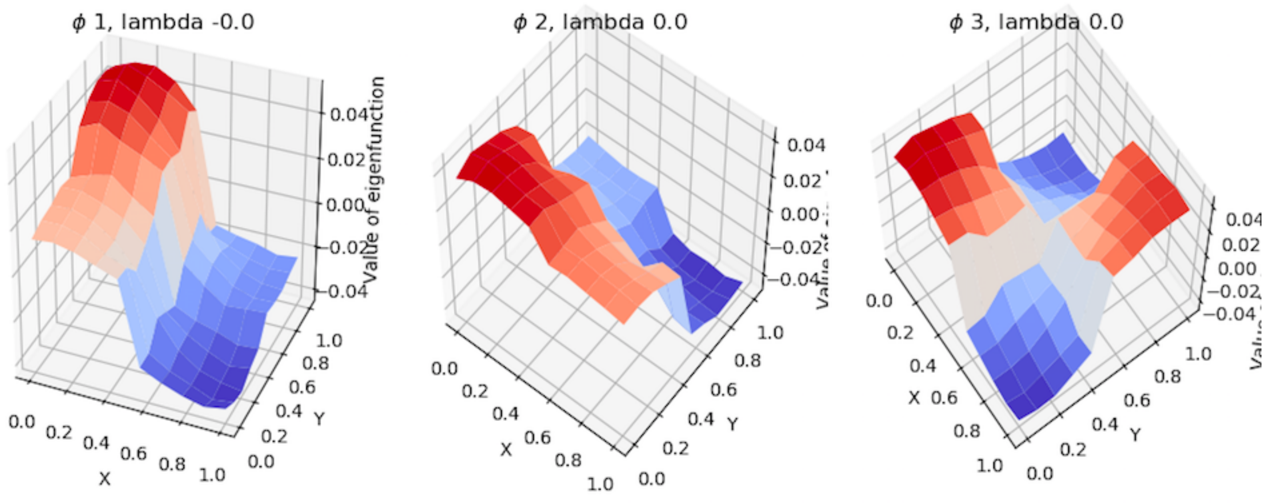

Figure 1: Rooms

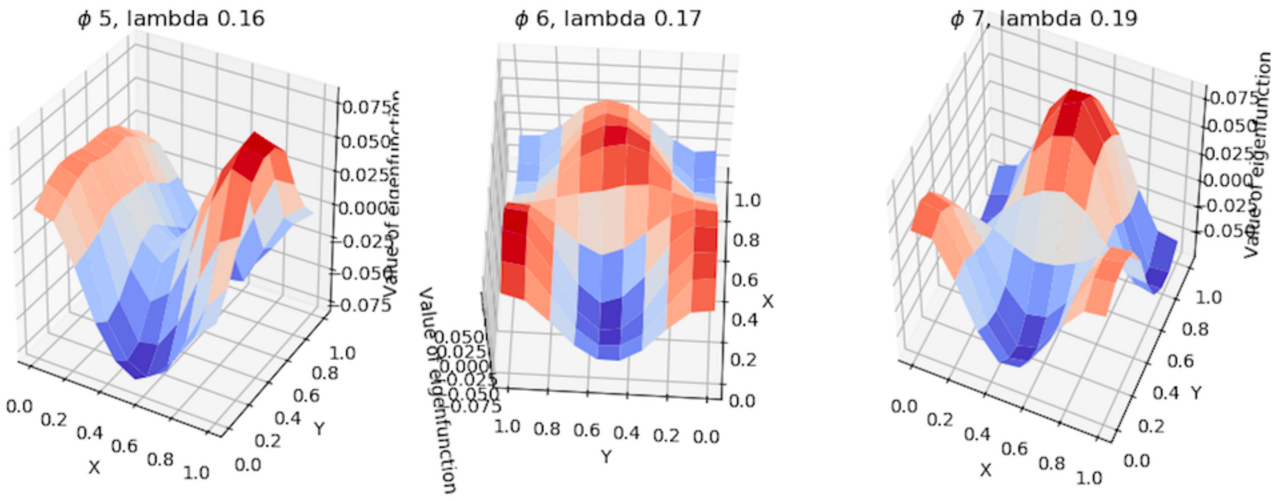

Figure 2: Bottlenecks

3 Least Squares Policy Iteration

3.1 Overview

LSPI belongs to the class of approximate Policy Iteration algorithms. The core idea of PI with a greedy policy is to iteratively compute a greedy policy $\pi_i(s) = \operatorname{argmax}_a Q(s,a)$ by performing a series of 2 steps:

1. evaluate the policy for each state-action pair and compute $Q^\pi(s,a) = r(s,a) + \gamma * \sum r(y,\pi(y))p(y|s,\pi(s))$
2. improve the policy $\pi_i(s) = \operatorname{argmax}_a Q(s,a)$

Two problems can occur with PI: first, the transitions $p(.|.,.)$ are usually unknown. Second, we cannot compute $Q(s,a)$ for every state-action pair when the state space is continuous.

The Least-Squares PI adresses these issues simultaniously. It assumes that $Q(s,a)$ can be approximated by a linear combination w of some basis $\phi(s,a)$. The evaluation step is transformed into a fitting step: we cannot compute $Q^w(s,a) = r(s,a) + \gamma * \sum r(y,\pi(y))p(y|s,\pi(s))$ but we can "force" Q to fit in the optimal Bellman equation that Q should enforce: $Q(s,a) = r(s,a) + \sum p(s'|s,a) \max_a Q(s',a)$. Given our batch $B = (s_i, a_i, r_i, s'_i)_i$, we can estimate the quantity $\sum p(s'|s,a) \max_a Q(s'|s,a)$ by $r_i + \max_a Q^w(s',a)$. Enventually, the LPSI boils down to the following computations (repeated until w_i converges):

- Policy evaluation: minimize wrt w_i the quantiy $\sum_i (\phi(s_i,a_i)w_i - r_i + \max_a \phi(s'_i,a)w_i$. This minimization can be computed in a close form.

The policy improvement is not explicitly made, it translates itself by the call $\max_a \phi(s'_i,a)w_i$.

3.2 Efficient LSPI for action independent basis

In our case and for many other popular basis choice (3), ϕ doesn't depend on a. Therefore, Q can be re-written $Q(s,a) = \sum \lambda_i^a \phi_i(s)$, the weight parameter λ is now a matrix. Note that the action space must be finite and preferably small.

This can yield serious computation improvements. The calls $\max_a \phi(s'_i,a)w_i$ and $\phi(s_i,a_i)w_i$ are made at each iteration and for the whole batch. Plus, the computation of $\phi(s)$ can be expensive, in our case we have to compute the neighbors of s and calculate the pairwise distances. This is why we propose the following scheme: at the begining of the algorithm, compute the tensors $\Phi_S = [[\phi(s)], ..., [\phi(s)]]$ with as many copies of $\phi(s)$ as the number of possible actions and $\Phi'_S = [[\phi(s')], ..., [\phi(s')]]$ for s and s' in the batch. This step can be standardly parallelized on python using the joblib library. Then at each step, one can heavily parallelize the calls mentionned in above using NumPy: for $\phi(s'_i,a)w_i$ use $\Phi_S \Lambda_i$ action-wise, at the whole batch level and for $\max_a \phi(s'_i,a)w_i$, use $\max \Phi'^{\Lambda}_{S}{}_{i}$.

These slight changes were necessary to pursue the rest of the project. Due to our low computational ressources, we would not have been able to test for many different hyperparameters in the experiments that follow.

4 Experiments

To assess the quality of approximations provided by the PVF basis, we compared the perfomance of PVFs against the one of RBFs, a popular basis based on gaussian kernels (also independent of the actions), using our efficient LSPI algorithm on 3 classic controls problem. Concerning the methodoldgy, we averaged our results on 5 independent runs for each basis and problem. For a given run, the performance was estimated doing 30 simulations. An extensive overview of the algorithm using PVF is provided in 16.

We added two things we did not mention before: PVF are learned on a sub-data set. The ϵ covering sampling (see Mahadevan (5)) aims to capture the manifold at best. If we did a random sampling, we would end up with highly covered sub spaces corresponding to attraction points of the MDP and sparse sub spaces corresponding to the areas hard to attain. The second thing is the implementation

Algorithm 1 PVF Policy Iteration

```
Require: n episode, horizon, k, nn, $\sigma$, $\epsilon$, tol
  1: LEARN PVF
  collect dataset D(n episodes, horizon), random walks
  sumbsample S from D to have an $\epsilon$ covering of the state space
  Compute Laplacian L from S, using nn neighbors, $\sigma$ in the kernel and store the k first eigenvalues
  $\phi \leftarrow$ NYSTRÖM(L)
  2: LSPI
  Compute $\Phi_s$ and $\Phi'_s$ (joblib parallel)
  $d \leftarrow \infty$
  $\Theta \leftarrow$ random matrix
  while d > tol do
    a $\leftarrow \operatorname{argmax} \Phi'_s\Theta$
    A $\leftarrow \Sigma(\Phi_s\Theta, \Phi_s\Theta - \gamma\Phi'_s[a]\Theta)$
    b $\leftarrow \Sigma r\Phi_s$
    d $\leftarrow$ ||$\Theta$ - solve(A, b)||
    $\Theta \leftarrow$ solve(A, b)
  end while
```

of LSPI. We chose to implement the method involving outer products rather than the closed-form solution to avoid direct matrix inversion. This being said, we are going to describe the problems we tried to solve by order of difficulty and the specific parameters we used.

Generic parameters For all the experiments, we selected 25 eigenfunctions for PVFs and 25 kernels for RBF (computation of RBFs was found on GitHub), we used 20 neighbors and $\sigma = 0.2$. For the LSPI algorithm, we used a .05 tolerance. Finally we used a .95 discount rate.

CartPole The cartpole problem aims to stablelize un unstable system. The system is a pole attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The perfomance metrics is mesured as a number of steps during which the cart remained stable. Since the system is unstable, we sampled many short episodes. The ϵ covering parameter was set to 0.08. The reward was 0 while the system was stable. We had to introduce a custom reward of -100 when the system stopped since none was provided by the environment.

Mountain Car A car is on a one-dimensional track, positioned between two "mountains". The goal is to drive up the mountain on the right; however, the car's engine is not strong enough to scale the mountain in a single pass. Therefore, the only way to succeed is to drive back and forth to build up momentum. Here, by opposition to the previous problem, the lesser steps, the better. We used an covering of 0.01. Rewards were -1 or 100 (custom, none provided by gym) for a terminal state.

Acrobot The acrobot system includes two joints and two links, where the joint between the two links is actuated. Initially, the links are hanging downwards, and the goal is to swing the end of the lower link up to a given height. Likewise, for the performance, the lower the number of steps, the better. We sampled few long walks for the training. The ϵ covering parameter was set to 0.9 ; the state space is 6 dimensions and much wider than the previous two. The rewards were -1 and 0 when the goal was reached.

The results are presented in the chart 4. First we denote that our LSPI did not converge in the case the Mountain Car problem whereas considered as "simpler" than the Acrobot, in terms of state space. This can be due to the fact that we need very long random walks to reach the goal so very few positive rewards in the training dataset. Perhaps an importance sampling procedure would have been usefull. The other result is that as expected, PVFs perform better in average than RBFs. However there is a higher variance for PVFs perfomance (confidence intervals don't overlap though). And for cause: when a good sampling is made, the PVFs are better built and the batch is better exploited. On the contrary, when poor sampling happens, all gets worse.

	Random	RBF	PVF
CartPole	>15	95 ± 9	$\mathbf{160} \pm 54$
Mountain Car	< 10000	none	none
Acrobot	< 3000	473 ± 41	$\mathbf{176} \pm 82$

Overall, our results seem quite far from the state of the art. However, our goal was simply to show that using PVF could benefit the approximation and outperform other basis and that they are worth being used. Also, in our defense, the LSPI framework is not the state-of-the-art anymore.

References

[1] A. I. Bobenko and B. A. Springborn. A discrete laplace–beltrami operator for simplicial surfaces. *Discrete & Computational Geometry*, 38(4):740–756, Dec 2007. ISSN 1432-0444. doi: 10.1007/s00454-007-9006-1. URL `https://doi.org/10.1007/s00454-007-9006-1`.

[2] Y. Canzani. Analysis on manifolds via the laplacian, 2013. URL `http://www.math.harvard.edu/canzani/docs/Laplacian.pdf`.

[3] G. Konidaris, S. Osentoski, and P. Thomas. Value function approximation in reinforcement learning using the Fourier basis. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence*, pages 380–385, August 2011. URL `http://lis.csail.mit.edu/pubs/konidaris-aaai11a.pdf`.

[4] M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *J. Mach. Learn. Res.*, 4:1107–1149, Dec. 2003. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=945365.964290`.

[5] S. Mahadevan. Proto-value functions: Developmental reinforcement learning. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 553–560, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5. doi: 10.1145/1102351.1102421. URL `http://doi.acm.org/10.1145/1102351.1102421`.

[6] N. Saito. Geometric harmonics as a statistical image processing tool for images on irregularly-shaped domains. *IEEE/SP 13th Workshop on Statistical Signal Processing, 2005*, pages 425–430, 2005.