Approximate Policy Iteration with Demonstration Data

Beomjoon Kim
School of Computer Science
McGill University
Montreal, Quebec, Canada

Amir-massoud Farahmand
School of Computer Science
McGill University
Montreal, Quebec, Canada

Joelle Pineau
School of Computer Science
McGill University
Montreal, Quebec, Canada

Doina Precup
School of Computer Science
McGill University
Montreal, Quebec, Canada

Abstract

We propose an algorithm to solve uncertain sequential decision-making problems that utilizes two different types of data sources. The first is the data available in the conventional reinforcement learning setup: an agent interacts with the environment and receives a sequence of state transition samples alongside the corresponding reward signal. The second data source, which differentiates the setup of this work from the usual reinforcement learning framework, is in the form of expert's demonstrations, that is, a set of states with the expert's suggested actions.

Benefitting from both sources of data, which are available in many real-world application domains, allows the agent to perform well even with few data points. The algorithm is couched in the framework of Approximate Policy Iteration. Its approximate policy evaluation step is formulated as a convex optimization problem in which the expert demonstration data act as a set of linear constraints. In a real robotic navigation task, we show that the algorithm outperforms both pure approximate policy iteration and supervised learning.

Keywords: Reinforcement Learning, Learning from Demonstration, Approximate Policy Iteration, Large-Margin Algorithm, Regularization, Robotics

Acknowledgements

This work was supported by the Natural Sciences and Engineering Research Council (NSERC) through the Discovery program, the NSERC Canadian Field Robotics Network (NCFRN), and the NSERC Postdoctoral Fellowship as well as the Canadian Institutes for Health Research (CIHR) through the CanWheel team.

1 Introduction

Solving uncertain sequential decision-making and reinforcement learning (RL) [1] problems with large state spaces can be quite difficult. These are, however, the problems that appear most often in real-world applications, e.g., in robotics, designing treatment strategy for chronic patients, etc. There are two key insights that help us solve these problems. The first is that the algorithm has to benefit from the intrinsic regularities of the problem in hand, and preferably does this adaptively. Even though this idea has been known for a long time in statistics and supervised machine learning, researchers have only recently started to develop such algorithms for RL problems, e.g., Farahmand et al. [2], Taylor and Parr [3], Kolter and Ng [4], Ghavamzadeh et al. [5], Farahmand and Szepesvári [6]. This paper introduces another idea: For some problems, we might occasionally be able to provide the agent with some extra information to guide its learning. In particular, we might provide the agent with the information about what actions are good or close to optimal in a few states.

This extra information, which we call "expert data", is common in many application domains and not using it limits the range of sequential decision-making problems that can be solved. As an example in robot learning, it is a common practice to solicit suggestions from an expert to learn complex behaviour, as done in the Learning from Demonstration (LfD) framework. In robotics and other complex control problems it is important to achieve good performance from relatively little data. It is also particularly crucial to limit the risk involved in learning by trial-and-error (as is done in RL), which could lead to catastrophic failures. A combination of trial-and-error RL data and expert data (i.e., mixing RL and LfD) offers a tantalizing way to effectively address challenging real-world policy learning problems.

Our primary contribution is a new large-margin algorithm that allows us to benefit from the demonstration data in an Approximate Policy Iteration (API) framework. The method is formulated as a coupled convex optimization. The key insight is that one can incorporate the expert demonstration data as a set of linear constraints. The optimization is formulated in a way that permits mistakes in the data provided by the expert, and also accommodates variable availability of expert data (i.e., just an initial batch or continued demonstrations). The algorithm has a theoretical guarantee in the form of an upper bound on the Bellman error, but we do not report that result in this extended abstract (cf. Kim et al. [7]).

We evaluate the algorithm's practicality in a real robot path finding task, where there are a few demonstrations, and trial-and-error data is expensive due to limited time. In all of the experiments, our method performed better than Least-Square Policy Iteration (LSPI) [8], using fewer trial-and-error data points and exhibiting significantly less variance. More empirical studies are reported in [7].

2 APID Algorithm

We consider a *continuous-state, finite-action discounted MDP* $(\mathcal{X}, \mathcal{A}, P, \mathcal{R}, \gamma)$, where $\mathcal{X}$ is a measurable state space (e.g., a subset of $\mathbb{R}^d$), $\mathcal{A}$ is a finite set of actions, $P : \mathcal{X} \times \mathcal{A} \to \mathcal{M}(\mathcal{X})$ is the transition model, $\mathcal{R} : \mathcal{X} \times \mathcal{A} \to \mathcal{M}(\mathbb{R})$ is the reward model, and $\gamma \in [0, 1)$ is a discount factor.[1] Let $r(x, a) = \mathbb{E}[\mathcal{R}(\cdot|x, a)]$, and assume that r is uniformly bounded by $R_{\max}$. A measurable mapping $\pi : \mathcal{X} \to \mathcal{A}$ is called a (deterministic) *policy*. As usual, V^π and Q^π denote the value and action-value function for π, while V^* and Q^* denote the corresponding value functions for the optimal policy π^*.

Our algorithm is couched in the framework of Approximate Policy Iteration (API) [9]. A standard API algorithm starts with an initial policy π_0. At the $(k+1)^{\text{th}}$ iteration, given a policy π_k, the algorithm approximately evaluates π_k to find $\hat{Q}_k$, usually as an approximate fixed point of the Bellman operator T^{π_k}: $\hat{Q}_k \approx T^{\pi_k}\hat{Q}_k$.[2] This is called the approximate *policy evaluation* step. Then, a new policy π_{k+1} is computed, which is greedy with respect to $\hat{Q}_k$. There are several variants of API that mostly differ on how the approximate policy evaluation is performed, a challenging problem for continuous state spaces. Most methods attempt to exploit structure in the value function [2, 3, 5], but in some problems one might have extra information about the structure of good or optimal policies as well, which would ideally be incorporated in the algorithm. This is precisely our case, since we have expert demonstrations.

To develop the algorithm, we start with regularized Bellman error minimization, which is a common flavour of policy evaluation used in API. Suppose that we want to evaluate policy π and we are given a batch $\mathcal{D}_{\text{RL}} = \{(X_i, A_i)\}_{i=1}^n$ containing n examples, and that we know the exact Bellman operator T^π. Then, the new value function $\hat{Q}$ is computed as:

$$\hat{Q} \leftarrow \operatorname*{argmin}_{Q \in \mathcal{F}^{|\mathcal{A}|}} \|Q - T^\pi Q\|_n^2 + \lambda_Q J^2(Q) \qquad (1)$$

where $\mathcal{F}^{|\mathcal{A}|}$ is a set of possible action-value functions, the first term is the squared Bellman error evaluated on data,[3] $J^2(Q)$ is a regularization penalty, which can prevent overfitting when $\mathcal{F}^{|\mathcal{A}|}$ is complex, and $\lambda_Q > 0$ is the regularization

[1]For a space Ω with σ-algebra σ_Ω, $\mathcal{M}(\Omega)$ denotes the set of all probability measures over σ_Ω.

[2]For discrete state spaces, $(T^{\pi_k}Q)(x, a) = r(x, a) + \gamma \sum_{x'} P(x'|x, a)Q(x', \pi_k(x'))$.

[3]$\|Q - T^\pi Q\|_n^2 \triangleq \frac{1}{n}\sum_{i=1}^n |Q(X_i, A_i) - (T^\pi Q)(X_i, A_i)|^2$ with (X_i, A_i) from $\mathcal{D}_{\text{RL}}$.

coefficient. The regularizer $J(Q)$ measures the complexity of function Q in the function space $\mathcal{F}^{|\mathcal{A}|}$. Different choices of $\mathcal{F}^{|\mathcal{A}|}$ and J lead to different notions of complexity, e.g., various definitions of smoothness, sparsity in a dictionary, etc. As a large class of examples, $\mathcal{F}^{|\mathcal{A}|}$ could be a reproducing kernel Hilbert space (RKHS) and J^2 its corresponding norm, i.e., $J^2(Q) = \|Q\|_{\mathcal{H}}^2$.

Now suppose that, in addition to $\mathcal{D}_{\mathrm{RL}}$, we have a set of expert examples $\mathcal{D}_{\mathrm{E}} = \{(X_i, \pi_E(X_i))\}_{i=1}^m$, which we would like to take into account in the optimization process. The intuition behind our algorithm is that we want to use the expert examples to "shape" the value function where they are available, while using the trial-and-error data to improve the policy everywhere else. Hence, even if we have few demonstration examples, we can still obtain good generalization everywhere due to the trial-and-error data.

To incorporate the expert examples in the algorithm one might require that at the states X_i belonging to $\mathcal{D}_{\mathrm{E}}$, the demonstrated action $\pi_E(X_i)$ be optimal, which can be expressed as a large-margin constraint: $Q(X_i, \pi_E(X_i)) - \max_{a \in \mathcal{A} \setminus \pi_E(X_i)} Q(X_i, a) \geq 1$. Nevertheless, this might not always be feasible, or desirable (if the expert itself is not optimal), so we add slack variables $\xi_i \geq 0$ to allow occasional violations of the constraints (similar to soft vs. hard margin in the large margin literature). The policy evaluation step can then be written as the following soft-constrained optimization problem:

$$\begin{aligned} \hat{Q} \leftarrow & \operatorname*{argmin}_{Q \in \mathcal{F}^{|\mathcal{A}|}, \xi \in \mathbb{R}_+^m} \|Q - T^\pi Q\|_n^2 + \lambda_Q J^2(Q) + \frac{\alpha}{m} \sum_{i=1}^m \xi_i \\ & \text{s.t.} \quad Q(X_i, \pi_E(X_i)) - \max_{a \in \mathcal{A} \setminus \pi_E(X_i)} Q(X_i, a) \geq 1 - \xi_i. \qquad \text{for all } (X_i, \pi_E(X_i)) \in \mathcal{D}_{\mathrm{E}} \end{aligned} \tag{2}$$

The parameter α balances the influence of the data obtained by the RL algorithm (generally by trial-and-error) vs. the expert data. When $\alpha = 0$, we obtain (1), while when $\alpha \to \infty$, we essentially solve a structured classification problem based on the expert's data [10]. In the latter case, the action-value function $\hat{Q}$ would be such that it imitates the expert demonstration data as much as possible.

Note that the above constrained optimization problem is equivalent to the following unconstrained optimization:

$$\hat{Q} \leftarrow \operatorname*{argmin}_{Q \in \mathcal{F}^{|\mathcal{A}|}} \|Q - T^\pi Q\|_n^2 + \lambda_Q J^2(Q) + \frac{\alpha}{m} \sum_{i=1}^m \left(1 - \left(Q(X_i, \pi_E(X_i)) - \max_{a \in \mathcal{A} \setminus \pi_E(X_i)} Q(X_i, a)\right)\right)_+, \tag{3}$$

where $(1 - z)_+ = \max\{0, 1 - z\}$ denotes the hinge loss.

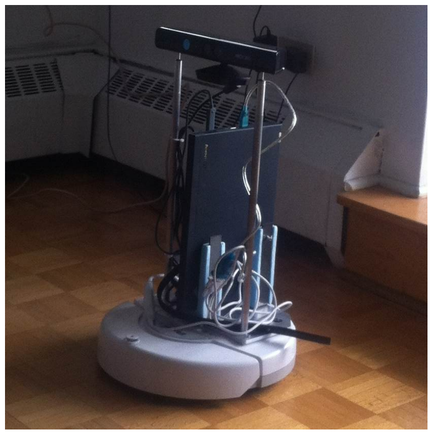
In many problems, we do not have access to the exact Bellman operator T^π, but only to samples $\mathcal{D}_{\mathrm{RL}} = \{(X_i, A_i, R_i, X_i')\}_{i=1}^n$ with $R_i \sim \mathcal{R}(\cdot|X_i, A_i)$ and $X_i' \sim P(\cdot|X_i, A_i)$. In this case, one might want to use the empirical Bellman error $\|Q - \hat{T}^\pi Q\|_n^2$ (with $(\hat{T}^\pi Q)(X_i, A_i) \triangleq R_i + \gamma Q(X_i', \pi(X_i'))$ for $1 \leq i \leq n$) instead of $\|Q - T^\pi Q\|_n^2$. It is known, however, that this is a biased estimate of the Bellman error, and does not lead to proper solutions [11]. One approach to address this issue is to use the modified Bellman error [11]. Another approach is to use Projected Bellman error, which leads to an LSTD-like algorithm [2]. Using the latter idea, we formulate our optimization as:

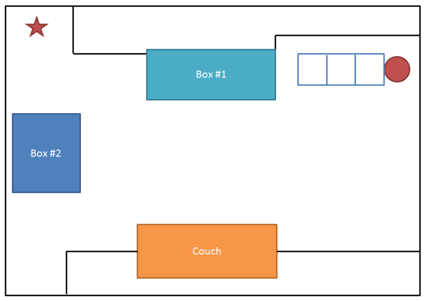
$$\begin{aligned} \hat{Q} \leftarrow & \operatorname*{argmin}_{Q \in \mathcal{F}^{|\mathcal{A}|}, \xi \in \mathbb{R}_+^m} \left\|Q - \hat{h}_Q\right\|_n^2 + \lambda_Q J^2(Q) + \frac{\alpha}{m} \sum_{i=1}^m \xi_i \\ & \text{s.t.} \quad \hat{h}_Q = \operatorname*{argmin}_{h \in \mathcal{F}^{|\mathcal{A}|}} \left[\left\|h - \hat{T}^\pi Q\right\|_n^2 + \lambda_h J^2(h)\right] \\ & \qquad Q(X_i, \pi_E(X_i)) - \max_{a \in \mathcal{A} \setminus \pi_E(X_i)} Q(X_i, a) \geq 1 - \xi_i. \qquad \text{for all } (X_i, \pi_E(X_i)) \in \mathcal{D}_{\mathrm{E}} \end{aligned} \tag{4}$$

Here $\lambda_h > 0$ is the regularization coefficient for $\hat{h}_Q$, which might be different from λ_Q. For some choices of the function space $\mathcal{F}^{|\mathcal{A}|}$ and the regularizer J, the estimate $\hat{h}_Q$ can be found in closed-form. For example, one can use linear function approximators $h(\cdot) = \boldsymbol{\phi}(\cdot)^\top \boldsymbol{u}$ and $Q(\cdot) = \boldsymbol{\phi}(\cdot)^\top \boldsymbol{w}$ where $\boldsymbol{u}, \boldsymbol{w} \in \mathbb{R}^p$ are parameter vectors and $\boldsymbol{\phi}(\cdot) \in \mathbb{R}^p$ is a vector of p linearly independent basis functions defined over the space of state-action pairs. Using simple l_2-regularization, $J^2(h) = \boldsymbol{u}^\top \boldsymbol{u}$ and $J^2(Q) = \boldsymbol{w}^\top \boldsymbol{w}$, the best parameter vector $\boldsymbol{u}^*$ can be obtained as a function of $\boldsymbol{w}$ by solving a ridge regression problem:

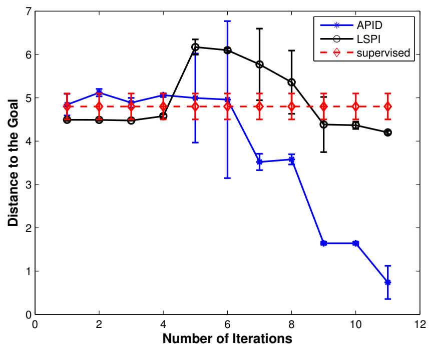
$$\boldsymbol{u}^*(\boldsymbol{w}) = \left(\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + n\lambda_h \mathbf{I}\right)^{-1} \boldsymbol{\Phi}^\top (\boldsymbol{r} + \gamma \boldsymbol{\Phi}' \boldsymbol{w}),$$

where $\boldsymbol{\Phi}$, $\boldsymbol{\Phi}'$ and $\boldsymbol{r}$ are feature and reward matrices for the trial-and-error dataset: $\boldsymbol{\Phi} = (\boldsymbol{\phi}(Z_1), \ldots, \boldsymbol{\phi}(Z_n))^\top$, $\boldsymbol{\Phi}' = (\boldsymbol{\phi}(Z_1'), \ldots, \boldsymbol{\phi}(Z_n'))^\top$, $\boldsymbol{r} = (R_1, \ldots, R_n)^\top$, with $Z_i = (X_i, A_i)$ and $Z_i' = (X_i', \pi(X_i'))$ (for data belonging to $\mathcal{D}_{\mathrm{RL}}$). More generally, as discussed above, we might choose the function space $\mathcal{F}^{|\mathcal{A}|}$ to be a reproducing kernel Hilbert space (RKHS)


(a)



(b)



(c)

Figure 1: (a) Picture of the robot. (b) Hand-drawn top-down view of the environment. The star represents the goal, circle represents the initial position, black lines indicate walls, and three grid cells represents the vicinity of Kinect. (c) Distance to the goal for LSPI, APID and Supervised learning with a random forest.

and J to be its corresponding norm, which provides the flexibility of working with a nonparametric representation while still having a closed-form solution for $\hat{h}_Q$.

The approach presented so far tackles the policy evaluation step of the API algorithm. We call our approach Approximate Policy Iteration with Demonstration (APID). This step would be alternated with greedification, as in usual API. So far we have left open the problem of how the datasets $\mathcal{D}_{\text{RL}}$ and $\mathcal{D}_{\text{E}}$ are generated. These datasets might be regenerated at each iteration, or they might be reused, depending on the availability of the expert and the environment. In practice when the expert data is rare, $\mathcal{D}_{\text{E}}$ will be a single fixed batch, but $\mathcal{D}_{\text{RL}}$ could be increased by e.g., running the most current policy (possibly with some exploration) to collect more data. The generation of these datasets is application-dependent.

Note that the values of the regularization coefficients as well as α should ideally change from iteration to iteration as a function of the number of samples as well as the value function Q^{π_k}.

3 Robot Path Finding

We evaluate APID on a real robot navigation task, when $\mathcal{D}_{\text{RL}}$ and $\mathcal{D}_{\text{E}}$ are *both* expensive to obtain. We have also conducted some experiments on a simulated domain, but we do not report those results here (cf. [7]). We compare APID with LSPI and supervised LfD (Random Forest), with small $|\mathcal{D}_E|$ and only one demonstrated trajectory. We do not assume that the expert is optimal (and/or abundant).

In this task, the robot needs to get to the goal in an unmapped environment (i.e., the robot does not know where the obstacles are). We use an iRobot Create equipped with Kinect RGB-depth sensor and a laptop. The Kinect perceptual module produces a point-cloud where each point, corresponding to a pixel in the RGB image, has horizontal, vertical, and depth coordinate information. We encode the Kinect observations with a 1×3 grid cells (1m $\times$ 1m). The robot also has three bumpers to detect a collision from the front, left, and right. Figures 1a-1b show a picture of the robot and its environment. In order to reach the goal, the robot needs to turn left to avoid a first box and wall on the right, while not turning too much, to avoid the couch. Next, the robot must turn right to avoid a second box, but make sure not to turn too much or too soon to avoid colliding with the wall or first box. Then, the robot needs to get into the hallway, turn right, and move forward to reach the goal position; the goal position is set to 6m forward and 1.5m right from the initial position.

The state is represented with 3 non-negative integer features (densities in each cell) and 2 continuous features (robot position). Densities are computed by counting the number of Kinect points in a cell. The robot has three discrete actions: turn left, turn right, and move forward. The reward is minus the distance to the goal. If the robot's front bumper is pressed and the robot moves forward, it receives a penalty equal to 2 times the current distance to the goal, and if the robot's left bumper is pressed and the robot does not turn right, and vice-versa, it again receives 2 times the current distance to the goal. The robot outputs actions at a rate of 1.7Hz. We used a linear Radial Basis Function (RBF) approximator as the function approximator architecture for the value function. To solve (4), we used CVX, a package for specifying and solving convex programs [12, 13].

We started from a single trajectory of demonstration, then incrementally added trial-and-error data while fixing the expert data. The number of data points added varied at each iteration, but the average was 160 data points, which is around 1.6 minutes of exploration, gathered using an ϵ-greedy exploration policy (decreasing ϵ over iteration). Over the 11 iterations, training time was approximately 18 minutes. Initially, $\frac{\alpha}{m}$ was set to 0.9, then decreased as new data was acquired. To evaluate the performance of each algorithm, we ran each iteration's policy for a task horizon of 100 ($\sim$1 min.), and repeated 5 times, to compute the mean and standard deviation.

As seen in Figure 1c, APID outperformed both LSPI and supervised LfD. The supervised LfD method kept running into the couch. Its poor performance is due to the difference in state distribution induced by the expert and the one induced by the agent's policy [14]. LSPI had a problem of exploring unnecessary states - when ϵ-greedy exploration policy was used, it explored regions of state space that are not relevant in learning the optimal plan, such as exploring the far left areas from the initial position. APID was able to leverage the expert data to efficiently explore most relevant states and avoid unnecessary collisions. For example, it learned to avoid the first box in the first iteration, then explored states near the couch where Supervised LfD failed. Finally, Table 1 gives the time it took for the robot to get to the goal (within 1.5m). The goal was reached only in the initial expert demonstration trajectory, and in iterations 9, 10 and 11 of APID. Note that the times achieved by APID (iteration 11) are similar to the expert

Table 1: Average time to reach the goal

Average Vals	**Demonstration**	**APID-9th**	**APID-10th**	**APID-11th**
Time To Goal(s)	35.9	38.4 ± 0.81	37.7 ± 0.84	36.1 ± 0.24

4 Conclusion

We proposed a regularized algorithm that allows us to benefit from expert's demonstrations in the reinforcement learning framework. This leads to policies that perform very well even with a few data samples and gradually improve when more trial-and-error samples are collected. This extension increases the range of real-world sequential decision-making problems that can efficiently be solved. In future work, we will explore more applications of APID.

References

[1] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, 1998.

[2] A.-m. Farahmand, M. Ghavamzadeh, Cs. Szepesvári, and S. Mannor. Regularized policy iteration. In *NIPS 21*. 2009.

[3] G. Taylor and R. Parr. Kernelized value function approximation for reinforcement learning. In *ICML*, 2009.

[4] J. Z. Kolter and A. Y. Ng. Regularization and feature selection in least-squares temporal difference learning. In *ICML*, 2009.

[5] M. Ghavamzadeh, A. Lazaric, R. Munos, and M. Hoffman. Finite-sample analysis of Lasso-TD. In *ICML*, 2011.

[6] A.-m. Farahmand and Cs. Szepesvári. Model selection in reinforcement learning. *Machine Learning Journal*, 85(3): 299–332, 2011.

[7] Beomjoon Kim, Amir-massoud Farahmand, Joelle Pineau, and Doina Precup. Learning from limited demonstrations. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[8] M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

[9] D. P. Bertsekas. Approximate policy iteration: A survey and some new methods. *Journal of Control Theory and Applications*, 9(3):310–335, 2011.

[10] I. Tsochantaridis, T. Joachims, T. Hofmann, Y. Altun, and Y. Singer. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6(2):1453, 2006.

[11] A. Antos, Cs. Szepesvári, and R. Munos. Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 71:89–129, 2008.

[12] M. C. Grant and S. P. Boyd. Graph implementations for nonsmooth convex programs. In *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008.

[13] CVX Research, Inc. CVX: Matlab software for disciplined convex programming, version 2.0. http://cvxr.com/cvx, August 2012.

[14] S. Ross, G. Gordon, and J. A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.