Approximate Linear Successor Representation

Clement A. Gehring
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institutes of Technology
Cambridge, MA 02139
gehring@csail.mit.edu

Abstract

The dependency of the value function on the dynamics and a fixed rewards function makes the reuse of information difficult when domains share dynamics but differ in their reward functions. This issue is present when attempting to reuse options to solve different tasks in the same domain. If instead of a value function, a successor representation is learned for some fixed dynamics, then any value function defined on any linear reward function can be computed efficiently. This setting can be particularly useful for reusing options in some hierarchical planning settings. Unfortunately, even linear parametrization of successor representation require a quadratic number of parameters with respect to the number of features and as many operations per temporal difference update step. We present a simple temporal difference-like algorithm for learning an approximate version of the successor representation with an amortized runtime $O(nk)$ for n features and the maximum rank-k of the approximation. Preliminary results indicate that the rank parameter can be much smaller than the number of features.

Keywords: successor representation, occupancy function, universal option model, incremental singular value decomposition, temporal difference

Acknowledgements

This work was supported in part by the NSF (grant 1420927). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We also gratefully acknowledge support from the ONR (grant N00014-14-1-0486), from the AFOSR (grant FA23861014135), and from the ARO (grant W911NF1410433).

1 Introduction

Hierarchical planning is a necessity for solving very large problems by breaking the problem into many short-horizon sub-problems. Options, a reinforcement learning (RL) formulation of macro-actions, can allow an agent to plan at a more abstract level [7]. In order to efficiently plan with options, it is necessary to build option models predicting the total reward from the start of an option in a given state to its completion. However, simply learning a value function for every option might not allow for the amount of generalization an efficient agent requires. This is in part due to the dependency of value functions on both the dynamics of the problem and the reward function.

As motivations for this work, we can think about a specific kind of hybrid RL planner. One can imagine a series of sub-tasks (i.e., reward functions), generated by an high-level planner, which are fed to an RL agent. In this context, we wish the RL agent to generate, off-line, and efficiently, new policies for the sub-tasks. This can be done with pre-defined options and their learned models. However, since every sub-task uses a different reward function, we need a generalized version of these models as proposed by Yao [8] in order to build models invariant to the reward function. Given these tools, the RL agent could then use model-based methods (e.g., dyna [4]) to efficiently find a good policy. This generalization of option models is achieved by learning a successor representation in lieu of value functions [3]. Successor representations (SR), which predict the expected discounted times future states are visited, have recently gained traction as models for animal behaviour. They offer an interesting addition to the model-free learning process (the *habitual* system). In addition, there seems to exist interesting shared characteristic between the successor representation and place cells as wells as well as the egiendecomposition of the SR and grid cells [5]. These results suggest that a compressed version of the SR might be useful for planning.

The SR solve an important issue with re-using options but come with a quadratic memory and computational cost. These drawbacks limit their usefulness as a tool for efficient planning. In this work, we present an approximate linear successor representation algorithm which would allow universal option models to be learned in much bigger feature spaces. In order to learn a rank-k approximation on n features, our temporal difference-like algorithm has an amortized cost $O(k^2 + nk)$ and requires $4nk + k$ parameters. This allows new value functions to be evaluated with a smaller matrix-vector product in $O(nk)$. Preliminary results indicate that k can be set much smaller than n. Furthermore, the performance degrades gracefully as the rank decreases, allowing a designer to choose a trade-off between computation time and accuracy.

We begin in Section 2 by presenting the reinforcement learning framework used as well as the definition of the successor representation. In Section 3, we define the rank-k approximation of the successor representation, after which, we present a novel algorithm for incrementally learning an approximate successor representation with a series of singular value decompositions. We present some preliminary results in Section 4 demonstrating the correctness of this approach in a small domain. We conclude with a discussion of the promising future work this approach might lead to. For brevity, we present the approximate successor representation algorithm as a standalone algorithm but the reader should keep in mind that this approach can easily be adapted to replace the "*accumulation* part" from Yao's work on learning universal option models [8].

2 Background

We model the problem as a Markov Decision Process (MDP) with states $\mathbb{S}$, actions $\mathbb{A}$, transition probabilities $\mathcal{P}^a(s, s')$, and reward function $r(s, a)$ for s and $s' \in \mathbb{S}$ and $a \in \mathbb{A}$. We consider actions sampled from a fixed policy π inducing transition probabilities $\mathcal{P}^\pi$. We seek to represent value functions V^π defined as

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \;\middle|\; S_0 = s\right],$$

where R_t is the reward received at time t, and $\gamma \in [0, 1)$ is a discount factor. Given that we only consider the fixed policy case, we omit the π superscript but it is important to keep in mind that the value function always depends on the policy.

In some cases, the states are discrete and few in number, allowing us to keep a separate value for each. We refer to this as the tabular case. However, in most interesting applications, the state space is large and continuous. In order to handle this case, we consider approximating the value functions with a linear combination of basis functions. This can be written as

$$\hat{V}(s) = \theta^T \phi(s),$$

where $\phi : \mathbb{S} \to \mathbb{R}^n$ corresponds to some feature representation of the state, and $\theta \in \mathbb{R}^n$ is a parameter vector. The successor representation is defined as the discounted total number of times a state is seen after starting from any other state [3]. More formally, in the tabular case, we define a matrix F, so that element $[F]_{i,j}$ denotes the expected discounted sum of indicator random variables with value 1 if the agent is in state j at that time, given that the trajectory starts in

state i.

$$[F]_{ij} = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathbb{1}_{S_t=j} \;\middle|\; S_0 = i\right],$$

where $\mathbb{1}_{S_t=j}$ has value one when the state at time t is j and zero otherwise. The successor representation can be naturally extended to linear function approximation in the following way

$$\phi(s)^\top \theta_i \approx \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t [\phi(S_t)]_i \;\middle|\; S_0 = s\right]$$
$$\Theta \triangleq [\theta_1 \quad \theta_2 \quad \dots \quad \theta_n],$$

where θ_i is the vector of parameters for a linear value function where feature i is accumulated instead of the rewards, and Θ is a concatenation of the columns θ_i. This means that $\phi(s)^\top \Theta$ is a row vector in which entry j is, approximately, the expected discounted accumulation of feature j's future value. It is important to note that the successor representation is invariant to the reward function. Computing the successor representation can be particularly helpful, as it allows us to easily compute any linear value function with arbitrary linear reward function (sharing the same transition probabilities). For a reward function of the form $r(s,a) = w^\top \phi(s)$, we compute a corresponding value function as $V(s) = \phi(s)^\top \Theta w$.

Once Θ is obtained, all that is required is to estimate w through linear regression, which is significantly easier to do than fitting a value function as it doesn't require any temporal abstraction. In addition, in many cases, it makes sense to assume w is given.

Since each column of Θ is analogous to a value function, the parameters Θ_t at time t can be updated using temporal difference with eligibility traces [6], giving the following update rule for a sample transition $\{s_t, r_t, s_{t+1}\}$:

$$\begin{aligned} e_t &\leftarrow \gamma\lambda e_{t-1} + \phi(s_t) \\ \delta_t &\leftarrow \phi(s_t) + \gamma\Theta_t^\top \phi(s_{t+1}) - \Theta_t^\top \phi(s_t) \\ \Theta_{t+1} &\leftarrow \Theta_t + \alpha e_t \delta_t^\top, \end{aligned} \tag{1}$$

where δ_t is a vector of temporal difference errors (for each feature), and e_t is an eligibility trace vector allowing updates to propagate through recently visited states. This results in update steps requiring $O(n^2)$ operations per complete update.

3 Approximate Linear Successor Representation

Since the successor representation is inherently quadratic in size, a natural approach is to consider some form of compression. Unfortunately, it is known that in the tabular case $F = (\mathbb{I} - \gamma\mathcal{P}^\pi)^{-1}$ and is therefore full rank. Instead, we consider a rank-k approximation of Θ of the form $\hat{\Theta} = U\Sigma V^\top$, where Σ is a $k \times k$ diagonal matrix and U and V are both $n \times k$ orthogonal matrices. The optimal such approximation (with respect to the reconstruction error) would be the singular value decomposition (SVD) of Θ. For this reason, we will consider an incremental SVD algorithm with truncation. The truncation will cause the algorithm to be sub-optimal (i.e., the result might not be the true SVD) but, empirically, this hasn't caused any problems for large enough k. In our experiments, we have observed the singular values of Θ decay very quickly which might suggest that k can often be set much smaller than n.

Instead of keeping a set of parameters Θ, our algorithm maintains five matrices, U, Σ, V, A, and B. The matrices U, Σ, V carry the same meaning as before. We include two extra $n \times k$ matrices which serve as temporary storage for updates. These buffers allow us to defer updates of the SVD in order to improve the runtime as well as the numerical accuracy. At all times, we can compute the most recent estimate of Θ as $\hat{\Theta} = U\Sigma V^\top + AB^\top$. The matrix-vector product $\hat{\Theta}^\top \phi(s)$ can be evaluated as a series of $n \times k$ matrix-vector operations. This algorithm is never required to build the quadratic $\hat{\Theta}$. Since every temporal-difference learning update has the form of an outer product, e_t's are stored as columns of A and δ_t's as columns of B. This allows updates of the SVD to be deferred while still updating the values of $\hat{\Theta}$. We store k updates in A and B before updating the SVD and zeroing A and B.

We use a simple algorithm presented in Brand et al. [2] for updating the SVD. It requires two QR decompositions on $n \times k$ matrices and one SVD on a $2k \times 2k$ matrix, as shown in Algorithm 1. This results in an amortized cost of $O(k^2 + nk)$ per update. The full pseudocode for our approximate linear successor representation (ASLR) update step is given by Algorithm 2. Note that in the case where $k = n$, this algorithm is exact (given infinite numerical precision) and will give the true SVD of Θ since every update step is identical to (1).

4 Experimental results

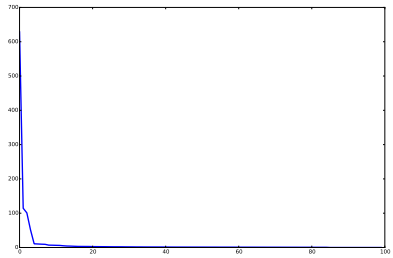
We offer a set of preliminary results on the behaviour of rank-k approximations of the successor representation. We refer to the temporal-difference algorithm for learning successor representation as SR and our approach as ALSR. In

Algorithm 1 SVD update

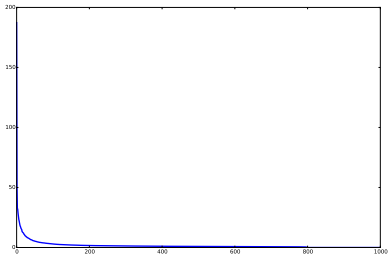
function UPDATE-SVD(U, Σ, V, A, B, k)
$Q_a, R_a \leftarrow$QR$((\mathbb{I} - UU^\top)A)$
$Q_b, R_b \leftarrow$QR$((\mathbb{I} - VV^\top)B)$
$K \leftarrow \begin{bmatrix} \Sigma & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} U^\top A \\ R_a \end{bmatrix} \begin{bmatrix} V^\top B \\ R_b \end{bmatrix}^\top$
$U', \Sigma', V' \leftarrow$ SVD(K)
$U \leftarrow \begin{bmatrix} U & Q_a \end{bmatrix} U'$
$V \leftarrow \begin{bmatrix} V & Q_b \end{bmatrix} V'$
$\Sigma \leftarrow \Sigma'$
return TRUNCATE(U, Σ, V, k)
end function

Algorithm 2 ALSR update

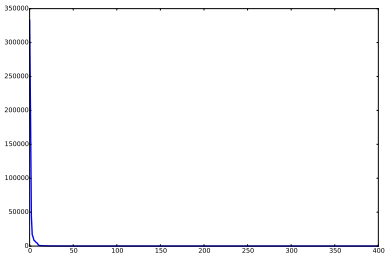
Given a sample transition $\{s_t, r_t, s_{t+1}\}$
$e_t \leftarrow \gamma\lambda e_{t-1} + \phi(s_t)$
$\delta_t \leftarrow \phi(s_t) + \gamma(V\Sigma U^\top + BA^\top)\phi(s_{t+1})$
$\quad -(V\Sigma U^\top + BA^\top)\phi(s_t)$
if $i \geq k$ **then**
$U, \Sigma, V \leftarrow$ UPDATE-SVD(U, Σ, V, A, B, k)
$A \leftarrow 0^{m \times k}$
$B \leftarrow 0^{m \times k}$
$i \leftarrow 0$
else
$A_{1:m,i} \leftarrow e_t$
$B_{1:m,i} \leftarrow \delta_t$
$i \leftarrow i + 1$
end if



(a) The sorted singular values of Θ in the 4-room domain with tabular representation and random actions. The successor representation was estimated from 1000 episodes each of 1000 steps.



(b) The sorted singular values of Θ in the Moutain car domain with a 20×20 radial basis encoding plus a bias term. The successor representation was estimated from 4000 episodes of experience following an energy pumping policy.



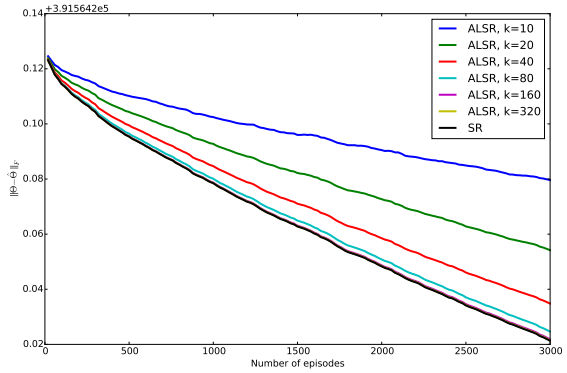
(c) The sorted singular values of Θ in the Moutain car domain with 10 tilings discretizing the state in a 10×10 grid plus a bias term. The successor representation was estimated from 4000 episodes of experience following an energy pumping policy.

Figure 1: The sorted singular values of Θ

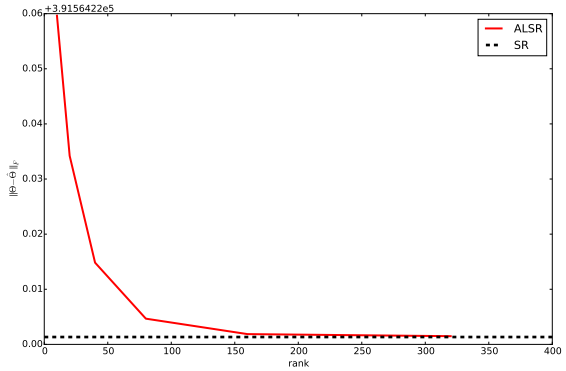
small domains, it is computationally feasible to approximate Θ fully using batched Least-Squares Temporal Difference (LSTD) [1] followed by a full SVD. In order to establish whether or not it is reasonable to approximate Θ with our approach, we plotted in Figure 1 the singular values of the successor matrix for the 4-room domain, defined in Sutton et al. [7], with tabular representation and random actions, and the Mountain car domain with a radial basis encoding and CMAC encoding under an energy pumping policy. These plots show that there are few important singular values. This result is encouraging since such a structure indicates that the matrix can be approximated with a relatively small rank.

We explored ALSR's performance more carefully on the Mountain car domain, in which an agent is tasked with driving a 1D under-actuated car back and forth in order to escape a valley. We used the dynamics described by Sutton and Barto [6], with random restart. The agent followed an energy pumping policy which always attempts to drive in the same direction as the velocity. With a 20×20 radial basis function encoding, we considered the effect of the parameter k with the approximated solution. With LSTD, we computed an empirical truth to which we compared $\hat{\Theta}$ by computing the Frobenius norm of their difference. Figure 2a shows the effect of the rank parameter, k, on the speed of convergence. Figure 2b highlights the effect of k on the solution after 3000 episodes. The dotted line represents the SR performance.

Our results in this experiment indicate that the performance of ALSR decreases as the rank is reduced, as one would expect. It is also encouraging to note that the same performance as SR is achieved with a rank significantly smaller than the number of features. This paves the way for much larger experiments where the computational gain of using ALSR could much greater. As a final observation, note that the required rank for achieving a reasonable performance with ALSR is higher than the required rank for a good approximation of Θ. This is most likely due to the greedy nature of the truncations in the incremental SVD. With our implementation, ALSR with $k = 40$ was an order of magnitude faster than SR when running on 900 features. This advantage decreased as the number of features was reduced, as expected. Though we haven't rigorously evaluated ALSR's runtime, this result hints that it would be advantageous in cases with a large number of features.



(a) The error of the estimated $\hat{\Theta}$ with respect to the number of episodes. The bottom lines represent the full successor representation and the approximate successor representation with high rank. Scores are averaged over 5 independent trials.



(b) The error of the estimated $\hat{\Theta}$ as a function of the rank parameter. The score represent the $\hat{\Theta}$ learned after 3000 episodes. Scores are averaged over 5 independent trials.

Figure 2: Estimation accuracy of Θ in the Mountain Car domain

5 Discussion

In this work, we proposed a novel approach for learning successor representation. Our ALSR algorithm is more efficient than the full SR temporal-difference algorithm and can potentially be applied to much large domains. We presented some preliminary results on small domains in which we have observed a graceful decay in performance as the rank is lowered and a comparable performance to that of the full successor representation.

Our results are preliminary and of limited scope. In the future, we hope to expand on this work by exploring ALSR's behaviour in larger domains, and settings with changing policies. In order to achieve better performance, we plan on adapting some of the modern temporal-difference algorithm variants to work with ALSR. This work focused on the successor representation used by the universal option model. There still remains a quadratic part in these models to optimize before they can be fully applied to large domains. It is possible that the model's expected next state functions can be similarly approximated by an incremental SVD approach. ALSR could easily be adapted to serve this goal in the future.

Finally, we would like to explore the link between the singular vectors found by ALSR and proto-value functions. The left singular vectors U can be seen as forming a basis over value function. It is possible that they can be used as proto-value functions (or could be equivalent in some way). This is an interesting avenue for upcoming research we hope to pursue.

References

[1] Steven J Bradtke and Andrew G Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1-3):33–57, 1996.

[2] Matthew Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear algebra and its applications*, 415(1):20–30, 2006.

[3] Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993.

[4] Jonathan Sorg and Satinder Singh. Linear options. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 31–38. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[5] Kimberly L Stachenfeld, Matthew Botvinick, and Samuel J Gershman. Design principles of the hippocampal cognitive map. In *Advances in Neural Information Processing Systems*, pages 2528–2536, 2014.

[6] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*. MIT Press, 1998.

[7] Richard S Sutton, Doina Precup, and Satinder P Singh. Intra-option learning about temporally abstract actions. In *ICML*, volume 98, pages 556–564, 1998.

[8] Hengshuai Yao, Csaba Szepesvari, Richard S Sutton, Joseph Modayil, and Shalabh Bhatnagar. Universal option models. In *Advances in Neural Information Processing Systems*, pages 990–998, 2014.