Low Complexity Proto-Value Function Learning from Sensory Observations with Incremental Slow Feature Analysis

Matthew Luciw and Juergen Schmidhuber

IDSIA-USI-SUPSI,
Galleria 2, 6928, Manno-Lugano, Switzerland

Abstract. We show that Incremental Slow Feature Analysis (IncSFA) provides a low complexity method for learning Proto-Value Functions (PVFs). It has been shown that a small number of PVFs provide a good basis set for linear approximation of value functions in reinforcement environments. Our method learns PVFs from a high-dimensional sensory input stream, as the agent explores its world, without building a transition model, adjacency matrix, or covariance matrix. A temporal-difference based reinforcement learner improves a value function approximation upon the features, and the agent uses the value function to achieve rewards successfully. The algorithm is local in space and time, furthering the biological plausibility and applicability of PVFs.

Keywords: Proto-Value Functions, Incremental Slow Feature Analysis, Biologically Inspired Reinforcement Learning

1 Introduction

A reinforcement learning [21] agent, which experiences the world from its continuous and high-dimensional sensory input stream, is exploring an unknown environment. It would like be able to predict future rewards, i.e., learn a value function (VF), but, due to its complicated sensory input, VF learning must be precluded by learning a simplified perceptual representation.

There has been a plethora of work on learning representation for RL, specifically Markov Decision Processes (MDPs); we can outline four types. **1. Top-Down Methods.** Here, the representation/basis function parameter adaptation is guided by the VF approximation error only [13, 16]. **2. *Spatial* Unsupervised Learning (UL).** An unsupervised learner adapts to improve its own objective, which treats each sample independently, e.g., minimize per-sample reconstruction error. The UL feeds into a reinforcement learner. UL methods used have included nearest-neighbor type approximators [17] or autoencoder neural nets [11]. **3. Hybrid Systems.** Phases of spatial UL and top-down VF-based feedback are interleaved [5, 11]. **4. *Spatiotemporal* UL.** Differs from the spatial UL type by using a UL objective that takes into account how the samples change through time. Such methods include the framework of Proto-Reinforcement Learning (PRL) [15], and Slow Feature Analysis (SFA) [22, 12].

There are some potential drawbacks to types 1,2 and 3. The top-down techniques bias their representation for the reward function. They also require the reward information for any representation learning to take place. In the spatial UL techniques, the encoding need not capture the information important for reward prediction — the underlying Markov Process dynamics. The spatiotemporal UL do not have these drawbacks. These capture the state-transition dynamics, the representation is not biased by any particular reward function, and it can learn when the reward information is not available.

In PRL, the features are called Proto-Value Functions (PVFs); theoretical analysis shows just a few PVFs can capture the global characteristics of some Markovian processes [3, 4] and that just a few PVFs can be used as building blocks to approximate value functions with low error. Sprekeler recently showed how SFA can be considered a function approximation to learning PVFs [20], so slow features (SFs) can have the same set of beneficial properties for representation learning for general RL. Kompella, Luciw and Schmidhuber recently developed an incremental method for updating a set of slow features (IncSFA; [10, 9]), with linear computational and space complexities.

The new algorithm in this paper is the combination of IncSFA and RL — here we use a method based on temporal-differences (TD) for its local nature, but other methods like LSTD [1] are possible — for incrementally learning a good set of RL basis functions for value functions, as well as the value function itself. The importance is twofold. First, the method gives a way to approximately learn PVFs directly from sensory data. It doesn't need to build a transition model, adjacency matrix, or covariance matrix, and in fact does not need to ever know what state its in. Second, it has linear complexity in the number of input dimensions. The other methods that derive such features — batch SFA and graphical embedding (Laplacian EigenMap) — have cubic complexity and don't scale up well to a large input dimension. Therefore our method is suited to autonomous learning on sensory input streams (e.g., vision), which the other methods are not suited for due to their computational and space complexities.

2 Slow Features as Proto-Value Functions

Due to space limits, we just skim over the background. See elsewhere [21, 15, 3, 22, 20] for further details.

Value Function Approximation for MDPs An MDP is a five-tuple: $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{P}^a_{s,s'}$ is the probability of transition from state s to s' when taking action a, $\mathcal{R}^a_s$ is the expected immediate reward when taking action a in state s, and $0 < \gamma \leq 1$ is the discount factor. RL often involves learning a value function on $\mathcal{S}$. Values are future expected cumulative discounted rewards. A complication: in our case, the agent does not know s. Instead it gets an observation vector: $\mathbf{x} \in \mathbb{R}^I$. The dimension I is large so it relies on its sensory mapping Φ to map $\mathbf{x}$ to $\mathbf{y} \in \mathbb{R}^J$, where $J << I$. Then, values are approximated as a linear combination of these J mappings:

$$V(\mathbf{x};\phi,\theta) = \sum_{j}^{J} \phi_j(\mathbf{x})\theta_j. \tag{1}$$

We're not so concerned with learning θ since there are good methods to do this given suitable basis functions. Here we are interested in learning ϕ. We'd like a compact set of mappings, which can deliver a reasonable approximation of different possible value functions, and which could be learned in an unsupervised way, i.e., without requiring the reward information. PVFs suit all these criteria, but require that the state is known, so they do not fit Eq. 1.

Proto-Value Functions. PVFs capture global dynamic characteristics of the MDP in a low dimensional space. The objective is to find a Φ that preserves similarity relationships between each pair of states s_t and $s_{t'}$, with a small set of basis functions, formally to minimize $\Psi(\phi_j) = \sum_{t,t'} A_{t,t'} \left(\phi_j(s_t) - \phi_j(s_{t'})\right)^2$ with unit norm and orthogonality constraints. In general $A_{t,t'}$ is a matrix of similarities, for MDPs typically a binary adjacency matrix, where a one means the states are connected (i.e., transition probability higher than some threshold). The objective penalizes differences between mapped outputs of adjacent states, e.g., if states s_t and $s_{t'}$ are connected, and a good ϕ_j will have $(y_t - y_{t'})^2$ small.

Laplacian EigenMap (LEM) procedure Φ can be solved for through an eigenvalue problem [2, 19]. The eigenvectors of the combinatorial Laplacian $\mathbf{L}$,

$$L_{i,j} = \begin{cases} degree(s_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \;\&\; s_i \text{ is adjacent to } s_j \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

are used, ordered from smallest to largest eigenvalue, for Φ.

Slow Features. SFA's objective is to find a few instantaneous functions g_j on the input that generate orthogonal output signals varying *as slowly as possible* (but not constant) [22]. The slow features do not require the true state to be known. The SFA objective can be solved via eigendecomposition of the covariance matrix $\dot{\mathbf{C}}$ of the temporal differences of the (whitened) observations. The slow features are low-order eigenvectors of $\dot{\mathbf{C}}$. The batch-SFA (BSFA) technique involves constructing such a covariance matrix from a batch of data.

Equivalence of PVFs and SFs. Sprekeler showed that the two objectives, of SFA (slowness) and LEM (nearness), are mathematically equivalent under some reasonable assumptions [20] (which are fulfilled when we collect data from a random walk on an MDP). For intuition, note that saying two observations have a high probability of being successive in time is another way of saying that the two underlying states have a high probability of being connected. In the LEM formulation, the neighbor relationships are explicit (through the adjacency matrix), but in SFA's they are implicit (through temporal succession).

The main reason the slow features (SFs) are *approximations* of the PVFs depends on the relation of observations to states. If the state is not extractable from each single observation, the problem becomes partially-observable (and out of scope here). Even if the observation has the state information embedded

Algorithm 1: IncSFA-TD$(J, \eta, \gamma, \alpha, T)$

```
//Autonomously learn $J$ slow features and VF approximation
  coefficients from *whitened* samples $\mathbf{x} \in \mathcal{R}^I$
$\{\mathbf{W}, \mathbf{v}^\beta, \theta\} \leftarrow$ Initialize ()
// $\mathbf{W}$ : Matrix of slow feature column vectors $\mathbf{w}_1, ..., \mathbf{w}_J$
// $\mathbf{v}^\beta$ : First Principal Component in difference space, with
  magnitude equal to eigenvalue
// $\theta$ : Coefficients for the VF
for $t \leftarrow$ *1 to* $\infty$ do
    $(\mathbf{x}_{prev} \leftarrow \mathbf{x}_{curr})$ //After $t = 1$
    $\mathbf{x}_{curr} \leftarrow$ GetWhitenedObsv()
    $r \leftarrow$ ObserveReward()
    if $t > 1$ then
        $\dot{\mathbf{x}} \leftarrow (\mathbf{x}_{curr} - \mathbf{x}_{prev})$
        $\mathbf{v}^\beta \leftarrow$ CCIPCA-Update $(\mathbf{v}^\beta, \dot{\mathbf{x}})$ //For seq. addition parm.
        $\beta \leftarrow \mathbf{v}^\beta / \|\mathbf{v}^\beta\|$
         //Slow features update
        $\mathbf{l}_1 \leftarrow 0$
        for $i \leftarrow$ *1 to* $J$ do
            $\mathbf{w}_i \leftarrow (1 - \eta)\mathbf{w}_i - \eta \left[(\dot{\mathbf{x}} \cdot \mathbf{w}_i)\ \dot{\mathbf{x}} + \mathbf{l}_i\right].$
            $\mathbf{w}_i \leftarrow \mathbf{w}_i / \|\mathbf{w}_i\|.$
            $\mathbf{l}_{i+1} \leftarrow \beta\ \sum_j^i (\mathbf{w}_j \cdot \mathbf{w}_i)\mathbf{w}_j$
        end
        $(\mathbf{y}_{prev} \leftarrow \mathbf{y}_{curr})$ //After $t = 2$
        $\mathbf{y}_{curr} \leftarrow \mathbf{x}_{curr}^T \mathbf{W}$
        if $t > T$ then
            $\delta \leftarrow r + (\gamma \mathbf{y}_{curr} - \mathbf{y}_{prev})\,\theta$ //TD-error
            $\theta \leftarrow \theta + \alpha\ \delta\ \mathbf{y}_{prev}$ //TD update
        end
    end
    $a \leftarrow$ SelectAction()
end
```

within, there may not be a linear mapping. Expanded function spaces [22] and hierarchical networks [8] are typically used with SFA to deal with such cases, and they can be used with IncSFA as well [14].

3 PVF Learning with IncSFA for VFs

Incremental Slow Feature Analysis updates slow features, incrementally and covariance-free, eventually converging to the same features as BSFA. It is detailed elsewhere [10, 9]. We want to use it to develop ϕ in Eq. 1, but we also need something to learn θ. As a motivation behind this work is to move towards biologically plausible, practical, RL methods, we use TD learning, a simple local learning method of value function coefficient adaptation. The resulting algorithm, IncSFA-TD (see Alg. 1) is biologically plausible to the extent that it is local in space and time [18], and its updating equation (Line 12) has an anti-Hebbian form [6]. The input parameters: J, the number of features to learn, η, the IncSFA learning rate, γ, the discount factor, α, the TD learning rate, and T, the time to start adapting the VF coefficients. For simplicity, the algorithm requires the observation to be drawn from a whitened distribution. Note the original IncSFA also provides a method for incrementally doing this whitening.

On Complexity. The following table compares the time and space complexities of three methods that will give approximately the same features — LEM (Laplacian EigenMap), BSFA (Batch SFA), and IncSFA — in terms of number of samples n and input dimension I.

	Computational Complexity	Space Complexity
LEM	$O(n^3)$	$O(n^2)$
BSFA	$O(I^3)$	$O(n + I^2)$
IncSFA	$O(I)$	$O(I)$

The computational burden on BSFA and LEM is the one time cost of matrix eigendecomposition, which has cubic complexity [7]. SFA uses covariance matrices of sensory input, which scale with input dimension I. However LEM's graph Laplacian scales with the number of data points n. So the computational complexity of batch SFA can be quite a bit less than LEM, especially for agents that collect a lot of samples (since typically $I << n$). IncSFA has linear updating complexity since it avoids batch-based eigendecomposition altogether. However, as an incremental method, it will be less efficient with each data point. The space burden in BSFA and LEM involves collecting the data and building the matrices, which IncSFA avoids.

4 Experiment

Environment. We use a vision-based Markovian environment (Fig 1), introduced by Lange and Reidmiller, 2010 [11]. The observation at any time is a 30×30 (slightly noisy) image, which shows a top-down view of the agent's position in a room. The agent is the 5 x 5 white square in the image. It can move up, down, left, or right, each by 5 pixels. The white "L" are walls. The environment borders on the image edges are also impassable. **Reward.** During an initial phase, there are no rewards. During a second phase, one point in the image will be associated with positive reward; all other places have zero reward. It has to learn features during the first phase (without any rewards) and approximate (and use) the value function in the second.

Setup. The agent explores the environment via random walk for $40,000$ steps. After each step, the slow features are updated, with learning rate $\eta = 0.0002$. At $t = 40,000$, the reward appears, and the agent continues its random walk for $3,000$ more steps, while learning the value function coefficients, with learning rate $\alpha = 0.0001$. After $t = 43,000$, the agent enters exploitation mode,

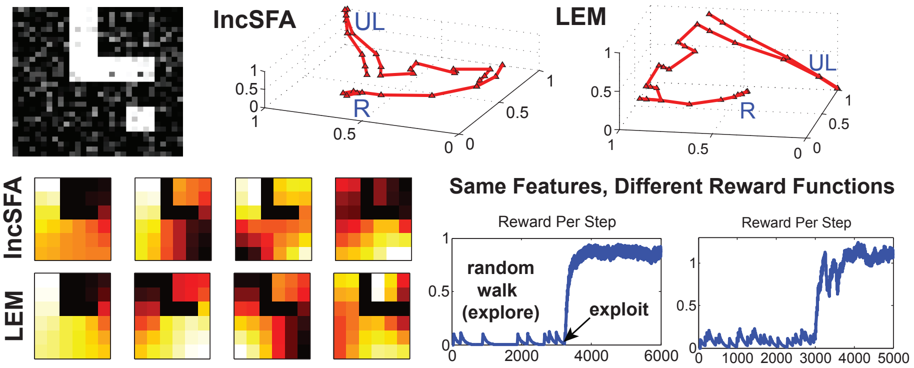

Fig. 1. Upper left: sample observation (30 × 30 image) from the environment. The IncSFA features developed from the exploration sequence directly from these observations are approximately the same as LEM features learned from eigendecomposition of the graph Laplacian of the true transition model. Upper center/right: embedding of a trajectory through the environment for both LEM features and IncSFA features (UL refers to the upper left corner of the room and R refers to the small inner room). Lower left: feature responses upon a grid of different images, where the agent is at different possible positions, for each of LEM and IncSFA (best viewed in color). Lower right: the agent goes to exploitation mode (maximize reward) using its value function learned upon the incrementally developed slow features. The performance shoots up to a nearly optimal level, for two different reward positions.

where it picks the action that will take it to the most valuable possible next state (using its current VF approximation), but with a 5% random action chance. To avoid the agent staying at the reward in exploitation mode, when the reward is reached, the agent teleports away. The features and coefficients continue to adapt. To show some generality, the reward will be placed in two different places (in two different instances) — inside the room or at the bottom center of the image.

Results are shown in Fig. 1. First, we want to show the features learned incrementally from sensory data actually deliver a reasonable LEM embedding. Visually compare the features of IncSFA, developed online and incrementally on the high-dimensional noisy images, to eigenvectors of the graph Laplacian, using the actual underlying transition model. Also note the similarity of the graphical embeddings of a single trajectory through the entire room upon the first three (non-constant) features for each of LEM and IncSFA. After going into exploitation mode, the agent quickly reaches a near optimal level of reward accumulation, for both reward functions. The features did not change significantly in the roughly $3,000$ exploitative steps.

Discussion. We can discuss some other methods that might apply to this setting. As mentioned, this environment was first developed elsewhere [11]. In that work, deep autoencoder neural nets are trained to compress the observations, and the bottleneck layer (with the fewest number of neurons) output becomes the state representation for the agent. Neural-fitted reinforcement learning (NFQ) learns the Q-function upon this state representation, and the NFQ net error (the TD-error) backpropagates throughout the autoencoder, causing the state representation to conform to a map-like embedding. This effect only emerges when the Q-error is backpropagated; otherwise the autoencoder representation does not resemble a map. In our case, the slow features learn a map representation in the unsupervised phase and therefore do not need the reward information to learn such a representation.

Another type of method that would apply is a nearest-neighbor prototype state quantization, where new prototypes/states are added when the distance of an observation from all existing prototypes exceeds some threshold. This type of method provides distinct states for RL but does not provide an embedding. Additionally, this method can lead to a large number of states, which increases the search space for the RL.

One might want to try an incremental Principal Component Analysis (PCA), which like SFA will also give a compressed code in a few features, but captures directions of highest variance (a spatial encoding). SFA uses the temporal information to learn spatial features, i.e., it casts the data into a low-dimensional space where similarity information is preserved. A low-dimensional map is quite useful for planning and control, but PCA's encoding does not necessarily have these properties (it will be good for reconstructing the input).

5 Conclusions

A real-world reinforcement learning agent doesn't get clean states, but messy observations. Learning to represent its perceptions in such a way that will aid its future reward prediction capabilities is just as (if not more) important than its method for learning a value function. For biological plausibility, the methods for learning representation and learning value need to be incremental and local in space and time. IncSFA and TD fulfill these criteria. We hope this method and the background we provided here influences autonomous real-world reinforcement learners.

Acknowledgments. We thank the anonymous reviewers and Sohrob Kazerounian for their useful comments. This work was funded by Swiss National Science Foundation grant CRSIKO-122697 (Sinergia project), and through the 7th framework program of the EU under grant #270247 (NeuralDynamics project).

References

1. S.J. Bradtke and A.G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1):33–57, 1996.
2. F.R.K. Chung. *Spectral graph theory.* AMS Press, Providence, Rhode Island, 1997.

3. R.R. Coifman, S. Lafon, A.B. Lee, M. Maggioni, B. Nadler, F. Warner, and S.W. Zucker. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *Proceedings of the National Academy of Sciences of the United States of America*, 102(21):7426, 2005.
4. R.R. Coifman and M. Maggioni. Diffusion wavelets. *Applied and Computational Harmonic Analysis*, 21(1):53–94, 2006.
5. A. da Motta Salles Barreto and C.W. Anderson. Restricted gradient-descent algorithm for value-function approximation in reinforcement learning. *Artificial Intelligence*, 172(4-5):454–482, 2008.
6. P. Dayan and L.F. Abbott. Theoretical neuroscience: Computational and mathematical modeling of neural systems. 2001.
7. G.E. Forsythe and P. Henrici. *The cyclic Jacobi method for computing the principal values of a complex matrix.* Applied Mathematics and Statistics Laboratories, Stanford University, 1958.
8. M. Franzius, H. Sprekeler, and L. Wiskott. Slowness and sparseness lead to place, head-direction, and spatial-view cells. *PLoS Computational Biology*, 3(8):e166, 2007.
9. V. R. Kompella, M. D. Luciw, and J. Schmidhuber. Incremental slow feature analysis: Adaptive low-complexity slow feature updating from high-dimensional input streams. *Neural Computation*, 2012. Accepted and to appear.
10. V.R. Kompella, M. Luciw, and J. Schmidhuber. Incremental slow feature analysis. In *International Joint Conference of Artificial Intelligence*, 2011.
11. S. Lange and M. Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *International Joint Conference on Neural Networks, Barcelona, Spain*, 2010.
12. R. Legenstein, N. Wilbert, and L. Wiskott. Reinforcement learning on slow features of high-dimensional input streams. *PLoS Computational Biology*, 6(8), 2010.
13. L.J. Lin. *Reinforcement learning for robots using neural networks.* School of Computer Science, Carnegie Mellon University, 1993.
14. M. Luciw, V. R. Kompella, and J. Schmidhuber. Hierarchical incremental slow feature analysis. In *Workshop on Deep Hierarchies in Vision*, 2012.
15. S. Mahadevan. Proto-value functions: Developmental reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 553–560. ACM, 2005.
16. I. Menache, S. Mannor, and N. Shimkin. Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research*, 134(1):215–238, 2005.
17. J.C. Santamaria, R.S. Sutton, and A. Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive behavior*, 6(2):163, 1997.
18. J. Schmidhuber. A local learning algorithm for dynamic feedforward and recurrent networks. *Connection Science*, 1(4):403–412, 1989.
19. J. Shi and J. Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.
20. H. Sprekeler. On the relation of slow feature analysis and laplacian eigenmaps. *Neural Computation*, pages 1–16, 2011.
21. R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press, 1998.
22. Laurenz Wiskott and Terrence Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural Computation*, 14(4):715–770, 2002.