Low Complexity Proto-Value Function Updating with Incremental Slow Feature Analysis

Matthew Luciw* MATTHEW@IDSIA.CH and Juergen Schmidhuber JUERGEN@IDSIA.CH
IDSIA-USI-SUPSI, Galleria 2, 6928, Manno-Lugano, Switzerland

Abstract
We show that Incremental Slow Feature Analysis provides a low complexity online and incremental method for learning and adapting Proto-Value Functions (PVFs). PVFs capture important, often global, characteristics of a graph-based representation of the environment. Since they provide a good basis for learning different reward functions, they are good building blocks for general reinforcement learning. Our incremental method learns PVFs from high-dimensional observation vectors, as the agent explores its world. Upon the incrementally learned features, a temporal-difference based coefficient learning method improves a value function approximation, and the agent uses the value function to achieve rewards successfully. IncSFA’s updates are local in space and time, and use anti-Hebbian learning, furthering the biological plausibility of PVFs.

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

1. Introduction
A reinforcement learning (Sutton and Barto, 1998) agent with continuous or high-dimensional sensory inputs is exploring its initially unknown environment, and trying to approximate the current value function (VF), which, when good enough, it will proceed to exploit to achieve rewards. But it needs to also learn the parameters of its representation layer — sensory mapping — $\Phi$ which maps sensory input $x$ to internal representation $y$, upon which the VF approximation is defined.

There has been a plethora of work on representation learning for RL, specifically Markov Decision Processes (MDPs). 1. Top-Down Methods. Representation parameters are adapted to reduce the VF approximation error (Lin, 1993; Menache et al., 2005). 2. Spatial Unsupervised Learning (UL). The objective for measuring the quality of the UL, e.g., per-sample reconstruction error, does not take into account the order of the samples. Such methods include nearest-neighbor type approximators (Santamaria et al., 1997) or autoencoder neural nets (Lange and Riedmiller, 2010). 3. Hybrid Systems. Steps or phases of spatial UL and top-down VF-based feedback are interleaved (da Motta Salles Barreto and Anderson, 2008; Lange and Riedmiller, 2010). 4. Spatiotemporal UL. While the resulting encoding will be instantaneous (per-sample), the measure of its quality involves how samples relate to one another through time. Such methods include the framework of Proto-Reinforcement Learning (PRL) (Mahadevan, 2005), and Slow Feature Analysis (SFA) (Wiskott and Sejnowski, 2002; Legenstein et al., 2010).

* This work was funded by Swiss National Science Foundation grant CRSIKO-122697 (Sinergia project).
The methods above are all suitable for certain domains. Here, we are interested in learning a general representation, e.g., not one biased by a single reward function. In this regard, the top-down techniques may have difficulty overcoming bias of the representation for each particular reward function. And in the spatial UL techniques, the encoding need not capture the information truly necessary to learn values — the underlying Markov Process dynamics. But the spatiotemporal UL techniques can capture the state-transition dynamics, and they are not biased by any particular reward function.

For example, theoretical analysis of PRL shows that just a few Proto-Value Functions (PVFs) can capture the global characteristics of some Markovian processes (Coifman et al., 2005; Coifman and Maggioni, 2006) and that just a few PVFs can be used as building blocks to approximate different value functions.

Sprekeler recently showed how SFA can be considered a function approximation to learning PVFs (Sprekeler, 2011), so slow features (SFs) can have the same set of beneficial properties for representation learning for general RL. Anonymous et al. recently developed an incremental method for updating a set of slow features (IncSFA; Kompella et al. (2011)), with low (linear) computational and space complexities. In this paper, we show how IncSFA can be used as a low complexity method for an autonomous RL agent to learn features of a spectral (graph)-embedding of its environment, which it experiences indirectly through high-dimensional observations (images). These features are used in value function approximation for different reward functions, and the value functions are exploited to achieve rewards.

This method is biologically inspired. Batch methods are not supported as biologically-plausible. Biological systems seem to learn incrementally. IncSFA is local (Schmidhuber, 1989) in space and time (one step of memory is required). Incremental methods lead to more robust behavior in complex and changing environments, and are good for open-ended learning. Additionally, IncSFA’s updating equations are related to anti-Hebbian learning (Dayan and Abbott, 2001) (although not completely biologically-plausible).

2. Slow Features as Proto-Value Functions

Preliminary: Value Function Approximation for MDPs An MDP is a five-tuple: \((S, A, P, R, \gamma)\), where \(S\) is a set of states, \(A\) is a set of actions, \(P^a_{s,s'}\) is the probability of transitioning from state \(s\) to \(s'\) when taking action \(a\), \(R^a_s\) is the expected immediate reward when taking action \(a\) in state \(s\), and \(0 < \gamma \leq 1\) is the temporal discount factor, an implicit cost on each action.

Values are the future expected cumulative discounted rewards associated with each state. When there are many states, the values cannot be simply stored in a table. Instead they can be approximated as a linear combination of \(J\) basis functions

\[
\hat{V}(s; \theta) = \sum_j \phi_j(s)\theta_j. \tag{1}
\]

We’d like as small a set of basis functions as possible, which can deliver a reasonable approximation of different possible value functions.

Further, the agent does not necessarily know \(s\) at any time. Instead it gets an observation vector: \(x \in \mathbb{R}^I\). In this case \(\Phi\) maps \(x\) to \(y \in \mathbb{R}^J\).
Proto-Value Functions. Through spectral embedding, PVFs capture global dynamic characteristics of the MDP in a low dimensional space. Imagine the MDP is represented as a graph, e.g., as $A$, an adjacency matrix of size $|S| \times |S|$\(^1\), and $D$, a diagonal matrix with $D_{i,i} = \text{degree}(s_i)$.

The objective is to find a $\Phi$ that preserves similarity relationships between each pair of states $s_t$ and $s_{t'}$, with a small set of basis functions, formally to minimize

$$\Psi(\phi_j) = \sum_{t,t'} A_{t,t'} \left( \phi_j(x(t)) - \phi_j(x(t')) \right)^2.$$  \hspace{1cm} (2)

under normalization (to control the scale of the feature outputs) and orthogonality (to ensure we get different mappings for $\phi_1, \phi_2, ..$) constraints:

$$\sum_{t,t'} D_{t,t'} \phi_i(x(t)) \phi_j(x(t')) = \delta_{i,j}$$  \hspace{1cm} (3)

where $\delta_{i,j} = 1$ if $i = j$ and is 0 otherwise.

Eq. 2 penalizes differences between mapped outputs of adjacent states. If states $t$ and $t'$ are connected, $A_{t,t'} = 1$, and a good $\phi_j$ will have $(\phi_j(x(t)) - \phi_j(x(t')))^2$ small.

$\Phi$ can be solved for through an eigenvalue problem (Chung, 1997; Shi and Malik, 2000). In the PRL framework, the eigenvectors of the combinatorial Laplacian $L$,

$$L_{i,j} = \begin{cases} 
\text{degree}(s_i) & \text{if } i = j \\
-1 & \text{if } i \neq j \& \text{s}_i \text{ is adjacent to } s_j \\
0 & \text{otherwise}
\end{cases}$$  \hspace{1cm} (4)

are used, ordered from smallest to largest eigenvalue, for $\Phi$. This is the Laplacian EigenMap (LEM) procedure.

Slow Features. SFA’s objective is to find a few instantaneous functions $g_j$ on the input that generate different output signals varying as slowly as possible. Formally:

Given a function space $F$ and an $I$-dimensional sequential input signal $x(t) = [x_1(t), ..., x_I(t)]^T$, find a set of $J$ instantaneous real-valued functions $g(x) = [g_1(x), ..., g_J(x)]^T$, which together generate a $J$-dimensional output signal $y(t) = [y_1(t), ..., y_J(t)]^T$ with $y_j(t) := g_j(x(t))$, such that for each $j \in \{1, ..., J\}$

$$\Delta_j := \Delta(y_j) := \langle \dot{y}^2_j \rangle \text{ is minimal}$$  \hspace{1cm} (5)

under the constraints

$$\langle y_j \rangle = 0 \text{ (zero mean),}$$  \hspace{1cm} (6)

$$\langle y^2_j \rangle = 1 \text{ (unit variance),}$$  \hspace{1cm} (7)

$$\forall i < j : \langle y_i y_j \rangle = 0 \text{ (decorrelation and order),}$$  \hspace{1cm} (8)

with $\langle \cdot \rangle$ and $\dot{\cdot}$ indicating temporal averaging and the derivative of $y$, respectively.

---

1. One way to do this is via a threshold on the transition probabilities — if $P_{s,s'} > \varepsilon$, states $s$ and $s'$ are marked adjacent. In this case, connections are assumed bidirectional, i.e., $P_{s,s'} \approx P_{s',s}$.
The constraints (6) and (7) together avoid a trivial constant output solution. The decorrelation constraint (8) ensures that different functions \( g_j \) do not code for the same features.

Like the PVFs, the SFA objective can be solved via eigendecomposition, of the covariance matrix \( \hat{\mathbf{C}} \) of the temporal differences of the (whitened) observations. The slow features are low-order eigenvectors of \( \hat{\mathbf{C}} \). Constructing such a covariance matrix from a batch of data is done by the batch-SFA (BSFA) technique.

**Equivalence of PVFs and SFs.** Sprekeler (2011) showed that the two objectives, of SFA (slowness) and LEM (nearness), are mathematically equivalent under some reasonable (i.e., when we collect data from a random walk on an MDP) assumptions. 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).

Note that SFA additionally depends on function space \( \mathcal{F} \). The main reason the slow features (SFs) are approximations of the PVFs is due to this function space, and what information it captures and what information it lacks. An agent’s observation may not so transparently capture the state, such as the case of an agent with egocentric high-dimensional images as observations. Expanded function spaces and hierarchical networks are typically used with SFA to deal with such cases.

3. **Approximate Low-Complexity Adaptive PVF Updating with IncSFA**

**IncSFA.** Incremental Slow Feature Analysis (Kompella et al., 2011, 2012) updates slow features, incrementally and online, eventually converging to the same features as BSFA. A key to its low complexity is that it is a covariance-free method, meaning it does not ever need to compute or build a covariance matrix, even in passing. The feature updating takes the form of Covariance-Free Incremental Minor Component Analysis (CIMCA; Peng and Yi (2006); Kompella et al. (2012)) upon the approximate derivatives (i.e., differences) of the inputs. CIMCA gives the lowest-order eigenvectors of the covariance matrix of the approximate derivatives.

CIMCA can provide approximate PVFs. To see this, take the following example. Here, the observation vector \( \mathbf{x} \) has a dedicated element for each state — this is a “tabular” observation. In this case, an observation \( \mathbf{x} \in \{0,1\}^{|\mathcal{S}|} \), with all elements zero except \( x_i(t) = 1 \), when \( s(t) = i \). The approximate derivative measurement \( \dot{\mathbf{x}}(t) = \mathbf{x}(t) - \mathbf{x}(t-1) \). Note that a covariance sample \( \dot{\mathbf{x}} \dot{\mathbf{x}}^T \) represents a bidirectional connection in the combinatorial Laplacian \( \mathbf{L} \). For example, if \( \mathbf{x}(t-1) = [1 0 0]^T \) and \( \mathbf{x}(t) = [0 1 0]^T \), then \( \dot{\mathbf{x}}(t) = [-1 1 0]^T \) and

\[
\dot{\mathbf{x}} \dot{\mathbf{x}}^T = \begin{pmatrix} 1 & -1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}
\]  (9)

So \( \mathbf{L} \) can be approximated by the covariance of \( \dot{\mathbf{x}} \), if we collect the samples from a random walk of length \( T \). Then, \( \mathbf{L} \approx E[\dot{\mathbf{x}} \dot{\mathbf{x}}^T] \approx \frac{1}{T-1} \sum_{t} \dot{\mathbf{x}} \dot{\mathbf{x}}^T. \)
CIMCA’s updating equation 10 has an anti-Hebbian form (Dayan and Abbott, 2001) \( w_i \leftarrow w_i - \eta g(w_i, \dot{x}) \dot{x} \). The update for a feature vector \( w_i \) given new sample \( \dot{x} \) is

\[
\begin{align*}
    w_i &\leftarrow (1 - \eta^{SFA}) w_i - \eta^{SFA} \left( (\dot{x} \cdot w_i) \dot{x} + \gamma \sum_{j=1}^{i-1} (w_j \cdot w_i) w_j \right), \\
    w_i &\leftarrow w_i / \|w_i\|.
\end{align*}
\]

(10)

(11)

where \( \eta_{SFA} \) is a learning rate. The Gram-Schmidt term inside the summation\(^2\) enforces features to be orthogonal. \( w_j \) are the “lower-order” minor components, and \( \gamma \) is larger than the first eigenvalue of \( \dot{z} \). We compute the most significant PC of \( \dot{x} \) with an incremental PCA technique to get this scalar. After updating, a feature is normalized for stability. The feature output will be \( \phi_i(x) = x^T w_i \).

**Comparison.** The following table compares the time and space complexities of the three methods under consideration — LEM (Laplacian EigenMap, used to get the PVFs), BSFA (Batch SFA), and IncSFA — in terms of number of samples \( n \) and input dimension \( I \).

<table>
<thead>
<tr>
<th></th>
<th>Computational Complexity</th>
<th>Space Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEM</td>
<td>( O(n^3) )</td>
<td>( O(n^2) )</td>
</tr>
<tr>
<td>BSFA</td>
<td>( O(I^3) )</td>
<td>( O(n+I^2) )</td>
</tr>
<tr>
<td>IncSFA</td>
<td>( O(I) )</td>
<td>( O(I) )</td>
</tr>
</tbody>
</table>

The computational burden on BSFA and LEM is the one time cost of matrix eigendecomposition, which has cubic complexity (Forsythe and Henrici, 1958). 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 \( I \ll n \)). 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. IncSFA does not need to store covariance or graph Laplacian matrices, since it updates its features directly, without ever computing an outer product. Further it updates its estimates after each sample; it is possible to discard the sample after that.

Finally, we note that, in the PRL framework or via BSFA, it is not possible to take an existing set of features and modify them based on the new samples, while it is possible with IncSFA.

**Updating VF Coefficients.** It is simple to estimate the value-function on top of the developing slow features. For biological plausibility, it is important that the VF learning is also local in space in time. Temporal difference-based updating is a simple local learning method of value function coefficient adaptation (from Eq. 1). The incremental update uses the TD-error, \( \delta_t = r_t + (\gamma \Phi(x_{t+1}) - \Phi(x_t))^T \theta_t \), and the form of the update is \( \theta_t = \theta_{t-1} + \alpha \Phi(x_t) \delta_t \), where \( \alpha \) is the learning rate.

\(^2\) We note this is not biologically plausible. A biologically plausible version might use mutually lateral inhibitory connections between SFs.
4. Experiment and Results

Experiment 1. 20-State Closed Chain. As a proof of concept, we take an agent in a small environment, with a different dedicated observation component to each state in the MDP. This is a 20-state, 2-action, chain environment, where state $1 < i < 20$ is connected to state $i - 1$ (through action one) and $i + 1$ (through action two). This is a closed chain — states (1) and (20) are connected. The observation is a 20 dimensional vector with a one at the current state and zeros elsewhere. Rewards of one are placed at state five and 15; the reward is zero elsewhere. The discount factor is 0.85.

Experiment. The agent explores by selecting action one or two randomly. Both CIMCA and TD start learning immediately and continue learning simultaneously. CIMCA has constant learning rate 0.01, and the TD learning rate is 0.05.

Results. Results are shown in Fig 1. (a) shows the similarity (at times {2000, 10000, 20000}) of the first eight slow features to the PVFs that result from eigendecomposition of the combinatorial graph Laplacian matrix, showing they are functionally equivalent. (b) measures...
IncSFA: Low Complexity Proto-Value Function Updating

Figure 2: The observation vectors are pixels of slightly-noisy 30 × 30 images, where the agent can move up, down, left, right from 4 to 6 pixels. It can wrap around to the opposite side. (a) Example observation. (b) Responses of 20 incrementally developed slow features over the entire range of agent positions, after 40,000 samples. Best viewed in color. (c) Value function approximations, after each 1,000 steps. The reward function changes over time from [5 5] to [10 10], and so on to [25 25]; the times of these changes are indicated by the black arrows.

the value function error (sum of squared errors) over all states; VF progress must wait until the SFs become stable enough. As a reference, VF learning upon the table of states is shown... of course, this tabular method will not scale to a large number of states. (c) shows the embedding of the 20 environment observations on developed features 2-4. The embedding captures the circular nature of the environment.

Experiment 2. Vision-Based High-Dimensional Continuous Environment. The range of true \( \{x, y\} \) positions goes from 1 to 30. But the observation vector is a 30 × 30 image where the agent is represented as a 5 × 5 white square centered on the true position. There are no walls here, parts of the observed agent phase through to the other side. There is slight gaussian noise on each pixel. The four actions take the agent up, left, down, or right in the image, with a 60% chance of traveling 5 pixels, and a 20% chance of going four or six pixels, respectively. The reward function has no reward except \( r = 1 \) when any agent pixel touches pixel 5,5 before \( t = 15,000 \), at which point it the rewarding pixel shifts to position 10,10 (at 20,000), 15, 15, (at 25,000) 20, 20 (at 30,000), then 25, 25 (at 35,000).

Experiment. At each \( t \), the agent selects an action randomly. It uses IncSFA to learn 20 SFs and the TD-method to learn the 20 value function coefficients. This experiment uses the full IncSFA algorithm, which includes spatial compression with CCIPCA (20 PCs). The sample mean and CCIPCA learning rate schedule are set to stabilize to a low constant learning rate at \( t = 10,000 \). The MCA learning rate is 0.001, and the TD learning rate 0.005.

Results. Refer to Fig. 2. Once the PCs stabilize, the SFs are able to learn, and once the SFs stabilize, the VF approximation becomes good. We can maintain a constant TD
learning rate and we observe the 20 coefficients adapt quickly and correctly when the reward function changes. The SFs do not change significantly after the reward function changes.

Figure 3: Upper left: the environment. Upper center/right: embedding of a trajectory through the environment for 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: results of going from exploration to exploitation for two different reward functions, but using the same incrementally developed slow features. The performance is nearly optimal.

High-Dimensional Environment with an Inner Room. This is an environment introduced by Lange and Reidmiller, 2010 (Lange and Riedmiller, 2010). The observation at any time has pixels of a $30 \times 30$ noisy image, which shows a top-down view of the agent’s position in a room. The room has a larger outer area and a small inner room, which can be reached through the passage on the right side. The agent can move up, down, left or right, each by 5 pixels. The reward will be placed in two different places: inside the room or at the bottom center of the image.

Experiment. The agent does a random walk in the room for 43,000 steps. After each step, the 32 features are updated through CIMCA, as are the 32 value function coefficients. Learning rates of $\eta^{MCA} = 0.0002$ and $\alpha = 0.0001$ were used. After $t = 43,000$, the value function is used to achieve rewards. The same features are used to approximate the VF for two different reward functions. In one case, a reward is placed inside the room, and the agent is teleported to the upper left corner every time it reaches the reward. In the other case, the reward is placed in the lower center and the agent is teleported to either the upper left or inside the room (random selection of the two). These are two separate experiments, however they use the same features from $t = 40,000$. In both cases, the agent goes into exploitation mode at $t = 43,000$, 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. The features and coefficients remain able to adapt.
Results are shown in Fig. 3. We 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 graphical structure. We also compare the graphical embeddings of a single trajectory through the entire room upon the first three (non-constant) features for each of LEM and IncSFA. This result shows the similarity of the low-complexity covariance and graph laplacian-free IncSFA to LEM. The results of using learned value functions are also quite good. After going into exploitation mode, the agent quickly reaches a near optimal level of reward accumulation. We did not notice the features change significantly in the roughly 5000 steps during which the agent collected data from the exploitative policy.

5. Conclusions

Incremental Slow Feature Analysis provides a method for an agent to learn useful features for different reward functions while exploring an initially unknown environment. The slow features are approximately proto-value functions, therefore graphical embedding can be done with linear complexity via IncSFA without constructing a graph (or a covariance matrix). Since IncSFA is incrementally and local in space and time, this paper’s results add to the biological plausibility of graph embedding objectives in feature learning for potentially autonomous agents that learn from rewards.

Acknowledgments

Sohrob Kazerounian provided helpful comments on a draft of this paper.

References


R. Legenstein, N. Wilbert, and L. Wiskott. Reinforcement learning on slow features of high-dimensional input streams. *PLoS Computational Biology*, 6(8), 2010. ISSN 1553-734X.


