# Contingent Features for Reinforcement Learning

Nathan Sprague

Department of Computer Science
James Madison University
Harrisonburg, VA 22807
{spragunr@jmu.edu}

**Abstract.** Applying reinforcement learning algorithms in real-world domains is challenging because relevant state information is often embedded in a stream of high-dimensional sensor data. This paper describes a novel algorithm for learning task-relevant features through interactions with the environment. The key idea is that a feature is likely to be useful to the degree that its dynamics can be controlled by the actions of the agent. We describe an algorithm that can find such features and we demonstrate its effectiveness in an artificial domain.

## 1 Introduction

A longstanding challenge in the area of reinforcement learning is scaling up to handle problems with high-dimensional and continuous state spaces. Even in cases where the intrinsic dimensionality of a task may be relatively low, the relevant state information is often embedded in a high-dimensional stream of sensor data.

There has been progress over the last few years in developing feature learning algorithms that extract task-relevant state information. Some approaches explicitly search for features that improve value function estimation [10, 11, 8, 3], while others examine the dynamics of state transitions in order to discover low-dimensional features based on the temporal structure of the task [9, 12, 7].

Each of these general approaches has drawbacks. Approaches based on value function estimation are dependent on the quality of the reward signal. Where reward is infrequent or absent, little learning can occur. On the other hand, unsupervised approaches based on state dynamics may waste representational power on aspects of the state that are not relevant to reaching the agent's goals.

The idea explored in this paper is that features are likely to be useful to the degree that their dynamics can be controlled by the actions of the agent. Aspects of the environment that are outside of the agent's control are less likely to be task-relevant.

The contingent feature analysis algorithm (CFA) described in this paper quantifies this notion of action contingency by searching for features that have lower variance in their temporal derivative when the agent chooses *not* to act than when it chooses to act. We refer to these as contingent features because their values are contingent upon the agent's actions.

The remainder of this paper will describe the contingent feature optimization problem, describe an algorithm for minimizing the objective, and illustrate the effectiveness of the algorithm in an artificial environment.

## 2 Contingent Feature Analysis

The following description of the contingent feature analysis algorithm builds on the notation and structure introduced by Wiskott and Sejnowski [14] in describing slow feature analysis (SFA). Slow feature analysis discovers slowly varying features overall: features that have a small temporal derivative. In contrast, CFA searches for features with temporal derivatives that have higher variance when actions are taken than when they are not taken. While the two algorithms have a similar structure, the quantities being minimized, and the resulting features, are entirely different.

The CFA algorithm handles signal, action pairs of the form $(\boldsymbol{x}(t), a(t))$, where $t$ indicates time, the $\boldsymbol{x}$ values are are potentially high-dimensional signal vectors, and the action values are drawn from a discrete set of actions $\mathcal{A}$. We will assume that $\mathcal{A}$ contains a designated "No Operation" ($NOP$) action: an action that the agent may choose in order to avoid driving a state transition.

### 2.1 The CFA Objective

The optimization problem is to find a vector-valued feature function $\boldsymbol{g}(\boldsymbol{x})$ such that the output signals $y_j := g_j(\boldsymbol{x})$ minimize the objective

$$\langle (\dot{y_j} - \langle \dot{y_j} \rangle)^2 \rangle_{a(t)=NOP} \qquad \text{(variance of derivative after } NOP \text{ actions)} \tag{1}$$

Subject to the following three constraints:

$$\langle y_j \rangle = 0 \quad \text{(zero mean)}, \tag{2}$$

$$\langle (\dot{y_j} - \langle \dot{y_j} \rangle)^2 \rangle = 1 \quad \text{(unit variance of derivative overall)}, \tag{3}$$

$$\forall i < j, \quad \langle \dot{y_i}, \dot{y_j} \rangle - \langle \dot{y_i} \rangle \langle \dot{y_j} \rangle = 0 \quad \text{(decorrelated derivative)}, \tag{4}$$

Where $\langle \cdot \rangle$ indicates temporal averaging, and $\dot{y}$ is the derivative of $y$ with respect to time. We assume that signal values are sampled at discrete time intervals, and that temporal derivatives are approximated by finite differences.

Given constraint 3, the objective in 1 is minimal for features that have low variance in their temporal derivative after $NOP$ actions relative to the overall variance of their temporal derivative. In other words, contingent features change more when actions are taken than when they are not taken. Constraint 4 ensures that different features carry different information, and it imposes an ordering on the features: $y_1$ is the most contingent feature, $y_2$ is less contingent since it must obey an additional constraint, etc. Constraint 2 prevents the solution from being under-constrained.

In this paper we will only consider the linear case where $g_j(\boldsymbol{x}) = \boldsymbol{w}_j^T \boldsymbol{x}$ for some weight vectors $\boldsymbol{w}_j$. It would be straightforward to extend this to the non-linear case by expanding $\boldsymbol{x}$ using a fixed set of non-linear functions, and then searching for appropriate weight vectors in the expanded space.

### 2.2 The CFA Algorithm

The CFA objective may be minimized as described below. Raw training signals will be indicated with a tilde, test data with a hat, and symbols with neither a tilde nor a hat will refer to normalized training data.

1. Calculate the time derivative of the raw signal: $\dot{\tilde{\boldsymbol{x}}}(t)$.
2. Center and whiten the resulting derivatives:

$$\dot{\boldsymbol{x}}(t) = S(\dot{\tilde{\boldsymbol{x}}}(t) - \langle \dot{\tilde{\boldsymbol{x}}} \rangle) \tag{5}$$

   Here $S$ is a whitening matrix that rescales the derivatives so that they have an identity covariance matrix. The matrix $S$ can be found using principal components analysis. This step ensures that the final features will satisfy constraint 3.
3. Separate out the whitened signal derivatives that correspond to $NOP$ actions:

$$\dot{\boldsymbol{n}}(t) := \{\dot{\boldsymbol{x}}(t) \mid a(t) = NOP\} \tag{6}$$

4. Perform PCA on the $\dot{\boldsymbol{n}}(t)$ values. The contingent features are determined by the eigenvectors with the smallest eigenvalues. For example:

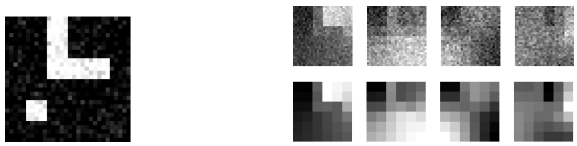
$$\begin{aligned} y_1(t) &= \boldsymbol{p}_1^T S(\tilde{\boldsymbol{x}}(t) - \langle \tilde{\boldsymbol{x}} \rangle) \\ &= \boldsymbol{w}_1^T (\tilde{\boldsymbol{x}}(t) - \langle \tilde{\boldsymbol{x}} \rangle) \end{aligned} \tag{7}$$

   Where $\boldsymbol{p}_1$ is the principal component with the smallest corresponding eigenvalue. The magnitude of each eigenvalue provides a measurement of the contingency of the corresponding feature.
5. The contingent features of test data may then be calculated by subtracting the training mean from the test signal, and multiplying the result by the appropriate weight vector:

$$\hat{y}_i(t) = \boldsymbol{w}_i^T (\hat{\boldsymbol{x}}(t) - \langle \tilde{\boldsymbol{x}} \rangle) \tag{8}$$

It may seem odd that the sphering matrix and principal components are calculated based on the time derivatives of the signal data, while the features themselves are calculated directly from the signal values. The resulting features do satisfy constraints 3 and 4. Since the derivative values are approximated by taking the difference between subsequent signal vectors, $\dot{\boldsymbol{x}}(t) \approx \boldsymbol{x}(t+1) - \boldsymbol{x}(t)$, multiplying the signal vectors by the weight vectors has the desired effect on the derivative as well.

The overall time complexity of this algorithm is the same as slow feature analysis: $O(NI^2 + I^3)$, where $N$ is the number of samples, and $I$ is the dimensionality of the input signal [2].



a b

**Fig. 1.** a. The grid navigation task b. *top*: Weight vectors discovered by slow feature analysis *bottom*: Feature activations at each grid position. The bottom row corresponds to the data from Figure 1 of [7].

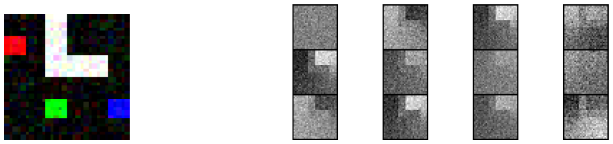
### 2.3 Experiment

In order to illustrate the CFA algorithm we will apply it to a variant of a task introduced by Lange and Riedmiller [6]. The task involves an agent moving in a $6 \times 6$ grid world containing an L-shaped wall. The agent's sensory input takes the form of a $30 \times 30$ image of the environment showing the position of the agent as a small square. Each pixel is corrupted with noise drawn from the distribution $\mathcal{N}(0, .01)$ and then clipped so that all values remain in the range [0,1]. There are four possible actions that each move the agent by exactly one grid square in one of the cardinal directions. Actions that would result in a collision with the wall or with a boundary have no effect. Figure 1a illustrates the task. This is an intrinsically low-dimensional task embedded in a high-dimensional sensor space.

Luciw and Schmidhuber used this same task as a test-bed for their incremental slow feature analysis algorithm [7]. It has been shown that slow feature analysis is a function approximation of Laplacian Eigenmaps, which provide the basis vectors for proto-value function reinforcement learning [13]. This suggests that slow feature analysis could be a useful approach to feature learning in high-dimensional RL tasks. Indeed, Luciw and Schmidhuber show that a linear reinforcement learning agent can reach near-optimal performance on this task using the four slowest features as a basis. Figure 1b shows the four slowest weight vectors discovered by (non-incremental) slow feature analysis for the environment described above. These features were learned from a sequence of 40,000 randomly generated actions[1].

Figure 2a illustrates a modified version of this task that will be used to demonstrate the CFA algorithm. In this version of the task there are three moving squares. The agent is represented by the small red square, while the green and blue squares are distractors that are not under the control of the agent. The agent's sensor data takes the form of $30 \times 30 \times 3$ color images. There are now five possible actions: the original four plus a $NOP$ action which does not result in a movement. Movements that would result in a collision with a wall or another agent have no effect. A reward of 1.0 is received when the agent enters a designated goal location in the upper-right region of the grid. All other actions result in zero reward.

[1] Slow feature analysis was performed using the MDP library [15].



a b

**Fig. 2.** a. Modified task, with distractors b. Weight vectors discovered by slow feature analysis.

In order to gather training data for the following experiments, actions are chosen uniformly at random over the course of 40,000 time steps. The green and blue squares move according to the same logic as the red square, but their movements will have no relationship to the actions chosen by the agent.

In this domain the sensor data associated with each square is segregated into a distinct color channel. This should make it possible to discover a set of features that encode the position of the red square while filtering out the positions of the distractors. This also makes it possible to assess the success of the algorithm by visually inspecting the feature weights. Since the dynamics of the green and blue squares are not under the control of the agent, there should be no structure in the corresponding feature weights.

Figure 2b illustrates the top four features discovered by slow feature analysis on this task. Each column represents a single weight vector, with the top box representing the red channel and the next two boxes representing the green and blue channels respectively. Not surprisingly, these features show no preference for any of the three color channels. Each feature represents a mixture of information from multiple channels. If the goal is to find a feature space that will enable the red agent to navigate in this environment, these features are not ideal.

Figure 3 shows the result of running the CFA algorithm on this data. The left side of Figure 3 shows the eigenvalues associated with the first 100 contingent features. It can be seen that there is abrupt transition between the contingent and non-contingent features in this task. The right-hand side of the figure shows the first thirty features, arranged from most to least contingent (or, equivalently, from lowest to highest eigenvalue).

The most notable attribute of these features is that they only carry information that is related to the agent's position: there is no structure in either the green or blue color channels.

It is also notable that the most contingent features in this task are features with high spatial frequency: as the degree of contingency goes down, the spatial frequency of the features goes down as well, with the last few features showing similarities to the weights discovered by slow feature analysis.

For this task, features with high spatial frequency will also have a large time derivative. The CFA algorithm does not explicitly optimize for features with large derivatives. Instead, the ordering of the features here is likely explained by the fact that the image noise causes background noise in the derivatives in

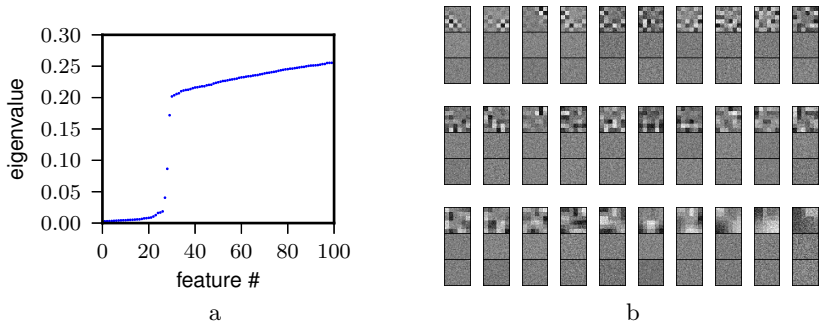



**Fig. 3.** a: Eigenvalues associated with the first 100 contingent features b: Weight vectors discovered by contingent feature analysis

every direction. When the derivative data is whitened, that noise will become proportionally larger in directions that had a low temporal derivative to start with. After the PCA step, those dimensions will then have proportionally larger eigenvalues since the same noise exists regardless of which actions are taken.

In algorithms such as PCA and SFA, it is common to extract the top two or three features and discard the rest. Figure 3 suggests that CFA shouldn't be used in that way. There is a clear demarcation between contingent and non-contingent features, but it appears that there are informative features at the lower end of the range of contingency values. This suggests a two-pass approach where CFA is used to extract a set of contingent features, while a second dimensionality reduction algorithm is applied in the resulting feature space.

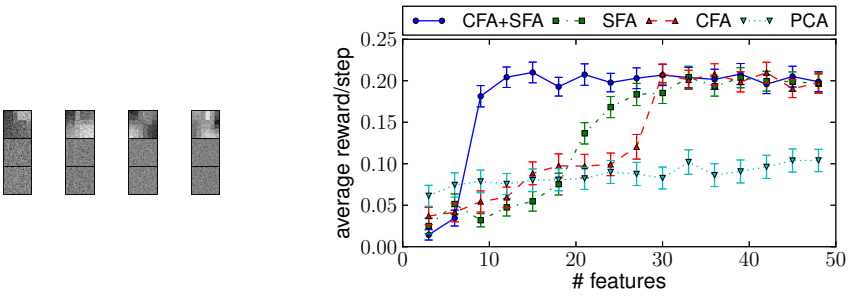
Figure 4a shows the result of performing slow feature analysis in the space defined by the 30 contingent features from Figure 3. The red channel here is strikingly similar to the single-agent slow features from Figure 1.

Figure 4b compares several feature types in terms of their effectiveness for reinforcement learning. Each data point represents the average reward received by a policy learned using the indicated number of features. Each policy is evaluated over the course of 1000 trials. Individual trials are terminated after 50 steps or when the agent reaches the goal location. Policies are learned using Least-Squares Policy Iteration (LSPI) applied to the same 40,000 training samples [5]. A small amount of L2 regularization is used to ensure convergence [4].

As expected, contingent feature analysis followed by slow feature analysis appears to produce useful features for this task. The agent is also able to learn the task using pure SFA or CFA features, but in each case it requires more than twice as many features to reach a similar level of performance.

## 3 Discussion

It should be noted that the contingent features for a domain will not necessarily be sufficient for task learning. As a simple example, imagine a version of the




a b

**Fig. 4.** a. Weight vectors discovered by performing slow feature analysis on the top-30 contingent features. b. Average reward values for different feature types on the example task. Error bars represent 95% confidence intervals.

three-square domain above in which the goal is to move the red square so that it comes into contact with the green square. The features from Figures 3 and 4a would clearly be inadequate for that task since they provide no information about the location of the green square.

The contingent features may be useful as part of a larger pool of features, even in cases where they are not themselves sufficient for task learning. Bellemare et. al. have illustrated that learned features based on contingency can significantly improve performance on a learning task that involves playing Atari 2600 games [1]. They explore a different notion of contingency that involves predicting regions of an image that will be changed by the actions of an agent. Those locations are then incorporated into a set of hand-designed features that are used for learning.

In the form described above, the CFA algorithm is only applicable in domains that have a natural notion of not acting. The algorithm could be extended to other domains by sequentially treating each action as the $NOP$ action and executing the CFA algorithm $|\mathcal{A}|$ times. Depending on the application, this version of the algorithm may provide more useful information than the single-$NOP$ version, since it provides a different set of contingent features for each action. Each contingent feature will be tied to the action that governs its dynamics.

## 4 Conclusions

Feature discovery for reinforcement learning presents a chicken and egg problem. We want features that enable accurate value function estimation, but until we have a value function estimate we lack the supervisory signal that could drive feature learning. Purely unsupervised approaches ignore potentially valuable information about how the agent's actions drive the dynamics of the environment. The CFA algorithm described in this paper addresses these challenges by including information about the agent's action choices in the feature discovery

process. The results above illustrate the effectiveness of the approach. We are able to discover relevant features even though the sensor data includes distractor information with exactly the same structure and dynamics as the information related to the agent.

## References


1. M. G. Bellemare, J. Veness, and M. Bowling. Investigating contingency awareness using Atari 2600 games. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.
2. A.N. Escalante-B and L. Wiskott. Slow feature analysis: Perspectives for technical applications of a versatile learning algorithm. *Künstliche Intelligenz*, 26(4):341–348, 2012.
3. A. Geramifard, T. Walsh, N. Roy, and J. How. Batch iFDD: A Scalable Matching Pursuit Algorithm for Solving MDPs. In *Proceedings of the 29th Annual Conference on Uncertainty in Artificial Intelligence*, 2013.
4. J. Kolter and A.Y. Ng. Regularization and feature selection in least-squares temporal difference learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 521–528. ACM, 2009.
5. M.G. Lagoudakis and R. Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149, 2003.
6. S. Lange and M. Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *Proceedings of the 2010 International Joint Conference on Neural Networks*, pages 1–8. IEEE, 2010.
7. M. Luciw and J. Schmidhuber. Low complexity proto-value function learning from sensory observations with incremental slow feature analysis. In *Artificial Neural Networks and Machine Learning–ICANN 2012*, pages 279–287. Springer, 2012.
8. S. Mahadevan, S. Giguere, and N. Jacek. Basis adaptation for sparse nonlinear reinforcement learning. 2013.
9. S. Mahadevan and M. Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 8(2169-2231):16, 2007.
10. R. Parr, C. Painter-Wakefield, L. Li, and M. Littman. Analyzing feature generation for value-function approximation. In *Proceedings of the 24th international conference on Machine learning*, 2007.
11. N. Sprague. Basis iteration for reward based dimensionality reduction. In *Proceedings of the 6th IEEE International Conference on Development and Learning*, 2007.
12. N. Sprague. Predictive projections. In *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence*, 2009.
13. H. Sprekeler. On the relation of slow feature analysis and laplacian eigenmaps. *Neural computation*, 23(12):3287–3302, 2011.
14. L. Wiskott and T.J. Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural computation*, 14(4):715–770, 2002.
15. T. Zito, Wilbert N., L. Wiskott, and P. Berkes. Modular toolkit for data processing (MDP): a Python data processing frame work. *Front. Neuroinform.*, 2(8), 2008.