# Investigating Contingency Awareness Using Atari 2600 Games

**Marc G. Bellemare** and **Joel Veness** and **Michael Bowling**
University of Alberta, Edmonton, Canada
{mg17,veness,bowling}@cs.ualberta.ca

**Abstract**

Contingency awareness is the recognition that some aspects of a future observation are under an agent's control while others are solely determined by the environment. This paper explores the idea of contingency awareness in reinforcement learning using the platform of Atari 2600 games. We introduce a technique for accurately identifying contingent regions and describe how to exploit this knowledge to generate improved features for value function approximation. We evaluate the performance of our techniques empirically, using 46 unseen, diverse, and challenging games for the Atari 2600 console. Our results suggest that contingency awareness is a generally useful concept for model-free reinforcement learning agents.

## 1 Introduction

Contingency awareness is the recognition that components of a future observation can be affected by one's choice of action. Within the cognitive science literature, contingency awareness is considered a crucial step in the intellectual development of children, and is widely believed to be acquired during early infancy (Watson and Ramey 1972). Experience with contingent stimulation appears to transfer to efficient learning and competency in new situations (Finkelstein and Ramey 1977). Furthermore, contingent observations seem to encourage the repetition of causal behavior irrespective of extrinsic reward (White 1959). While it is not yet clear what mechanisms produce contingency awareness in humans, it seems plausible that some form of contingency awareness could play an important role in the construction of artificially intelligent agents.

When using function approximation with popular model-free reinforcement learning algorithms such as SARSA($\lambda$), it is well known (Sutton 1996; Tesauro 1995) that good performance hinges on having access to an appropriate set of basis functions or features. While automatic feature construction methods (Parr et al. 2007; Konidaris, Osentoski, and Thomas 2011) and generic feature sets (Schweitzer and Seidmann 1985) have been proposed, current techniques do not explicitly incorporate any notion of contingency awareness. Our work proposes a complementary feature generation technique that accurately captures this notion for a wide class of domains. To this end, we formalize the notion of *contingent regions*: the parts of an observation whose immediate future value depends on the the agent's choice. We then propose a mechanism to identify the contingent regions from a single stream of interaction with the environment. Finally, we describe how we use these contingent regions to generate contingency-aware features for value function approximation. The general applicability of our methods is evaluated in over 50 different Atari 2600 games.



## 2 Background

We begin by describing the Atari 2600 platform and reviewing related reinforcement learning work on video games. The Atari 2600 was a popular second generation game console, originally released in 1977. Over 900 games were developed for the Atari 2600, spanning a diverse range of genres such as shooters, beat'em ups, puzzle, sports and action-adventure games. Many popular arcade games, including *Space Invaders*, *Pac-Man*, and *Donkey Kong* were also ported to the Atari 2600. While these games are considerably simpler than modern video games, they remain challenging for game-independent techniques (Naddaf 2010).

Prior work on Reinforcement Learning on the Atari 2600 platform has outlined the challenges in finding good state features for this domain. Diuk, Cohen, and Littman (2008) applied their DOORMAX algorithm to a restricted version of the game of Pitfall. Their method extracts objects from the displayed image. These objects are then converted into a first-order logic representation of the world, the Object-Oriented Markov Decision Process (OOMDP). Their results show that DOORMAX can discover the optimal behavior for this OOMDP within one episode. Wintermute (2010) proposed a method that also extracts objects from the displayed image and embeds them into a logic-based architecture, SOAR. Their method uses a forward model of the scene to improve the performance of the Q-Learning algorithm (Watkins and Dayan 1992). They showed that by using such a model, a reinforcement learning agent could learn to play a restricted version of the game of Frogger. More recently, Cobo et al. (2011) investigated automatic feature discovery in the games of Pong and Frogger, using their own simulator. Their proposed method takes advantage of human trajectories to identify state features that are important for playing console games. Once such features are identified, they are

used by a reinforcement learning agent to abstract the game state. Their empirical results show increased performance when using the reduced feature set.

The methods discussed above all require significant human intervention, either in designing the first-order logic predicates and model or in obtaining expert human trajectories. The need for human intervention makes it difficult to apply algorithms from scratch to new problems. One of the aims of this paper is to show that contingency awareness can be used to automatically augment a feature set. Furthermore, the technique is effective for a broad range of Atari 2600 games, with no additional human knowledge needed for new games.

Lastly, Naddaf (2010) studied how one could develop game-independent AI agents for the Atari 2600. These agents were designed to perform well across a broad set of games. He studied search-based methods, using the Atari simulator as a forward model, and explored the performance of different handcrafted feature sets for linear function approximation in the reinforcement learning setting. His work illustrated the challenges involved in developing game-independent agents in both search and reinforcement learning settings.

## 3 Black Box RL

For this work, we adopt a discrete action, finite horizon, bounded reward, deterministic black-box reinforcement learning setup. This involves two entities, an agent and an environment, which communicate over a fixed number $N \in \mathbb{N}$ of discrete time cycles. Interaction begins with the agent selecting an action from the *action space*, a finite set $\mathcal{A}$. The environment then responds with a *percept*, which is an observation-reward pair. Each observation is an element from a set $\mathcal{O}$ known as the *observation space*. Each reward is a scalar value from a compact set $\mathcal{R} \subset \mathbb{R}$, the *reward space*. We denote the *percept space* by $\mathcal{X} := \mathcal{O} \times \mathcal{R}$. A *history* is an alternating string of action-percepts from the history space, which is defined as $\mathcal{H} := \cup_{i=0}^{N} (\mathcal{A} \times \mathcal{X})^i \cup (\mathcal{A} \times \mathcal{X})^i \times \mathcal{A}$. A history contains all of the information available to an agent at a particular point in time.

The goal of the agent is to act in such a way as to maximize its accumulated reward over the $N$ cycles. The behaviour of an agent is summarized by a *policy*: a set of probability distributions $\pi(\cdot|h)$ over the action space $\mathcal{A}$, one for each distinct history $h \in \mathcal{H}$ that ends in a percept. The only additional property of our black box environments we require is that they have the option to be *reset* to a starting configuration. This allows us to meaningfully evaluate the performance of a given policy by averaging across multiple runs of $N$ cycles for any given environment.

## 4 Contingency Awareness

Broadly speaking, *contingency awareness* is the recognition that a future observation is under an agent's control and not solely determined by the environment. From an AI perspective, one of the main challenges faced when attempting to exploit this natural idea comes from having to translate the abstract notion of contingency into mathematical terms. In this section we describe a notion of contingency awareness for the Atari 2600 platform.

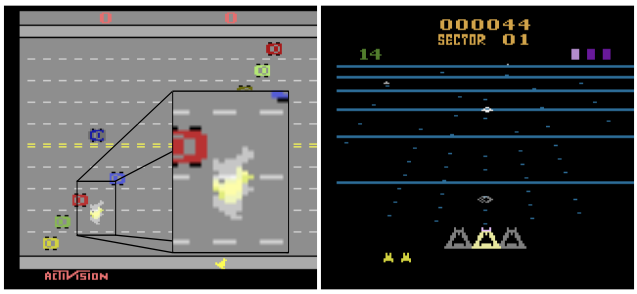


Figure 1: The contingent regions, shown by a transparent overlay, for *Freeway* (left) and *Beam Rider* (right).

### 4.1 Contingency within Atari 2600

Intuitively, the contingent regions of an observation are the components whose value is dependent on the most recent choice of action. In the context of the Atari domain, this corresponds to a set of pixel locations. This set changes over time and depends on the exact sequence of actions made by the agent. We now formalize this intuition by introducing some notation to describe the Atari 2600 observation space.

Each observation represents an image structured as a two-dimensional array of pixels. We use $\mathcal{D}_x \subset \mathbb{N}$ and $\mathcal{D}_y \subset \mathbb{N}$ to denote the set of row and column indices respectively, and $\mathcal{D} := \mathcal{D}_x \times \mathcal{D}_y$ to denote the joint index space. The color space $\mathcal{C}$ is a finite set of possible pixel colors. A pixel is a tuple $(x, y, c) \in \mathcal{D}_x \times \mathcal{D}_y \times \mathcal{C}$, where $x$ and $y$ denote the pixel's row and column location and $c$ denotes the color of this location. Each observation is therefore a set of $|\mathcal{D}|$ pixels; the observation space $\mathcal{O}$ is the set of all possible observations.

We can now define the notion of *contingent regions* in the black-box Atari 2600 reinforcement learning setup.

**Definition 1** *$\forall n \in \mathbb{N}$, given a history $h \in (\mathcal{A} \times \mathcal{X})^n$, the contingent regions $C(h)$ of history $h$ is defined as*

$$C(h) := \left\{ (x, y) \in \mathcal{D} : \exists a, a' \in \mathcal{A}, o^a_{x,y}(h) \neq o^{a'}_{x,y}(h) \right\},$$

*where $o^a_{x,y}(h)$ denotes the color of the pixel at location $(x, y)$ within the observation that follows history $ha \in (\mathcal{A} \times \mathcal{X})^n \times \mathcal{A}$.*

Thus the contingent regions of history $h$ is the set of pixels within the next observation whose value is dependent on the action that follows $h$. Figure 1 depicts examples of contingent regions in *Freeway* and *Beam Rider*. Because of the discrete and highly non-linear nature of Atari games, contingent regions are not necessarily connected: for example, bullets and missiles are separate from the region surrounding the player's avatar.

### 4.2 Learning the Contingent Regions

We now discuss some general approaches for learning a classifier that can predict whether or not a pixel belongs to the contingent regions within an arbitrary Atari 2600 game.

The most direct approach is to construct a training set and employ supervised learning methods. We can exploit

the determinism of the Atari, as well as its ability to reset, to compute the contingent regions $C(h)$ for any particular history $h$. Therefore, from a given set of histories $H := \{h_1, h_2, \dots\}$, we can construct a set of training examples

$$\big\{\big(h, l, t\big) \in H \times \mathcal{D} \times \{0,1\} : t = \mathbb{I}\big[l \in C(h)\big]\big\}. \quad (1)$$

An alternate method is to construct the training set online. An online approach is appealing because, unlike the offline approach, it can be applied to no-reset, stochastic domains. Our online learning method works by building an approximate model of the game's dynamics. Although the Atari domain is deterministic, from the agent's perspective it is reasonable to view the system as a stochastic process. We capture the notion of contingency within the stochastic setting by generalizing Definition 1.

**Definition 2** *$\forall n \in \mathbb{N}$, given a history $h \in (\mathcal{A} \times \mathcal{X})^n$, the contingent regions $C(h)$ of history $h$ is defined as*

$$C(h) := \Big\{(x,y) \in \mathcal{D} : \exists a, a' \in \mathcal{A}, T^a_{x,y}(\cdot \,|\, h) \neq T^{a'}_{x,y}(\cdot \,|\, h)\Big\},$$

*where each $T^a_{x,y}(\cdot \,|\, h)$ describes a pixel-level transition model: a distribution over the color of the pixel at location $(x,y)$ within the observation that follows history $ha \in (\mathcal{A} \times \mathcal{X})^n \times \mathcal{A}$.*

Although we do not have access to the true underlying pixel-level transition models, we can approximate Definition 2 by replacing each $T^a_{x,y}(\cdot \,|\, h)$ term with a learned approximation $\tilde{T}^a_{x,y}(\cdot \,|\, h)$ and the non-equality test with a divergence threshold. As both transition models are approximate, we use a symmetric form of the KL divergence,

$$D(p \,||\, q) \;:=\; D_{KL}(p \,||\, q) + D_{KL}(q \,||\, p),$$

to measure the distance between each pixel-level transition model, where $D_{KL}(p \,||\, q) := \sum_{s \in \mathcal{S}} p(s) \log p(s)/q(s)$ and $p$ and $q$ are discrete probability distributions over a common domain $S$. Our approximate notion of the contingent regions of history $h$ now becomes

$$\tilde{C}(h) := \Big\{(x,y) \in \mathcal{D} : \max_{a,a' \in \mathcal{A}} D\left(\tilde{T}^a_{x,y}(\cdot \,|\, h) \,||\, \tilde{T}^{a'}_{x,y}(\cdot \,|\, h)\right) \geq \delta\Big\}, \quad (2)$$

where $\delta \in \mathbb{R}^+$ is a positive threshold parameter. Thus the effects of an action on a pixel need to cause a divergence of at least $\delta$ before this pixel is considered part of the approximate contingent regions of a history $h$.

### 4.3 Implementation Details

This section describes how we translate the ideas of Sections 4.1 and 4.2 into an implementation fast enough to run in real-time. The main obstacle lay in the complexity introduced by the 7 bits per pixel, $210 \times 160$ pixel observation space.

**Block Decomposition.** In many cases, predicting the contingent regions at the pixel level is unnecessary. To reduce the method's computational cost, we divide the observation space into $k \times k$ blocks, $k \in \mathbb{N}$, and assume that pixels within a block share the same contingency behavior. This leads to the following definition of the contingent regions using a block size of $k$:

$$B_k(h) := \Big\{(x,y) \in \mathcal{D} : S_k(x,y) \cap C(h) \neq \emptyset\Big\},$$

where $S_k(x,y) := \big\{(x',y') \in \mathcal{D} : r_k(x,y) = r_k(x',y')\big\}$, $r_k(x,y) := (f_k(x), f_k(y))$ and $f_k(x) := k\lceil x/k \rceil - \lfloor k/2 \rfloor$. The approximate block contingent regions $\tilde{B}_k(h)$ is defined similarly, with $\tilde{C}(h)$ replacing $C(h)$ in the above definition. This leads to a speedup proportional to $k^2$ since only one prediction is made for every $k \times k$ block of pixels. We found $k = 5$ to provide a good trade-off between performance and accuracy. Figure 2 shows an example of the exact contingent regions of a *Seaquest* frame and its corresponding $5 \times 5$ block decomposition.

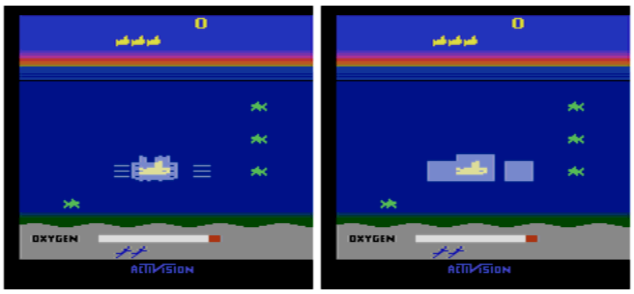


Figure 2: Exact (left) and 5x5 Block Decomposed (right) contingent regions in *Seaquest*.

**Contingency Learning.** We now describe how we use supervised learning to generate a probabilistic predictor of the contingent regions. We first construct a training set in the form of Equation 1 with $C(h)$ replaced by $B_5(h)$. We then fit a parametrized logistic regression model to this data by finding an approximate maximum likelihood solution. Under our logistic model, the likelihood that a pixel at location $l \in \mathcal{D}$ is part of the contingent regions of history $h$ is $p(l \,|\, h; \mathbf{w}) := \sigma\left(\mathbf{w}^T \mathbf{\Phi}_l(h)\right)$, where $\mathbf{w} \in \mathbb{R}^{m \times 1}$ is a vector of weights, $\sigma(x) := (1 + \exp(-x))^{-1}$ is the logistic function and $\mathbf{\Phi}_l(h) \in \mathbb{R}^{m \times 1}$ is a column vector of features that depend on the current history $h$ and the location $l$ of interest. Thus, given a training set $\mathcal{T}$ of the form specified by Equation 1, the likelihood of the observed data under our logistic model is

$$p(\mathcal{T}; \mathbf{w}) := \prod_{(h,l,t) \in \mathcal{T}} p(l \,|\, h; \mathbf{w})^t (1 - p(l \,|\, h; \mathbf{w}))^{1-t}. \quad (3)$$

Equation 3 can be maximized by minimizing the *cross entropy error function* $E(\mathbf{w}) := -\log(p(\mathcal{T}; \mathbf{w}))$, which upon rearranging and simplifying becomes

$$E(\mathbf{w}) := -\sum_{(h,l,t) \in \mathcal{T}} t \log p(l \,|\, h; \mathbf{w}) + (1-t) \log\big(1 - p(l \,|\, h; \mathbf{w})\big).$$

We minimize $E(\mathbf{w})$ with stochastic gradient descent as it allows us to avoid having to keep the entire training set in memory. This gives, for the $k$th training example $(h_k, l_k, t_k) \in \mathcal{T}$, the following update equation

$$\mathbf{w}_k \leftarrow \mathbf{w}_{k-1} + \eta\left[t_k - \sigma\left(\mathbf{w}_{k-1}^T \mathbf{\Phi}_{l_k}(h_k)\right)\right] \mathbf{\Phi}_{l_k}(h_k),$$

where $\eta \in \mathbb{R}$ is the learning rate parameter.

**Color Change Prediction.** Although Equation 2 gives a natural criterion for detecting contingent regions with a learned probabilistic color model, learning such a model over the full 7-bit Atari color space requires a prohibitive number of samples. An acceptable level of accuracy can be achieved by predicting whether a pixel changes color rather than predicting the color to which it changes. In effect, we reduce learning a full pixel-level transition model (the $\tilde{T}^a_{x,y}$ term) to learning a binary predictor. This also reduces the number of summands needed to compute the symmetrized KL divergence by a factor of $2^6$. We also used logistic regression in combination with stochastic gradient descent to predict color changes.

# 5 Contingency-based Feature Generation

To illustrate how contingent regions can be used, we describe one way to integrate them into a reinforcement learning agent. Since value function approximation holds a prominent role in most large-scale RL solutions (Sutton 1996), we focus on incorporating contingency awareness within a function approximation architecture. We restrict our attention to linear function approximation, a commonly used scheme in practice.

Our method relies on two mild assumptions: (1) the player controls an avatar that exists at some location and generates contingencies; (2) the player's avatar moves smoothly across the screen. Our method works by first tracking the centroid of the avatar using a standard Bayes filter (Russell and Norvig 2010). We implement this by using a truncated Gaussian motion model and a Gaussian observation model for which the probability of observing contingency decreases with distance from the player location. At the start of a game or when the game is reset, the prior over the player location is set to be uniform over $\mathcal{D}$. Since we never observe contingency directly, we instead provide the observation model with the distribution output by the learned contingency predictor.

We extract the maximum a posteriori location of the player avatar from the Bayes' filter to enhance an existing RL feature map $\boldsymbol{\Upsilon} : \mathcal{H} \rightarrow \mathbb{R}^{m \times 1}$. First, the estimated player location is mapped into a particular cell of an $i$ by $j$ discretization of $\mathcal{D}$. The extended feature map $\boldsymbol{\Psi} : \mathcal{H} \rightarrow \mathbb{R}^{mij \times 1}$ is then constructed by taking the Cartesian product of the set of tiled locations and the features used by $\boldsymbol{\Upsilon}$. In effect, the resulting extended feature map represents each discretized player location using $m$ independent features.

# 6 Evaluation

Sections 4 and 5 propose a framework for incorporating a notion of contingency awareness into a reinforcement learning agent. We now experimentally evaluate the ability of this framework to both predict the contingent regions and to generate an improved set of features for value function approximation.

## 6.1 Environment Description

We used our publicly available Arcade Learning Environment (2012) to map games into reinforcement learning environments. Each environment generates observations that consist of $160 \times 210$ pixels, with each pixel taking on one of $2^7$ colors; a single observation does not usually contain all of the Atari state information. The action space consists of 18 distinct actions, which reflect the different possible x/y locations of the joystick and whether the fire button is pressed or not. In all games, the reward is given by the difference in score from one time step to the next. The range of rewards is not known in advance and varies from game to game. Each environment is episodic, terminating when the game is over (e.g., when the player is out of lives). A single reinforcement learning step consists of five frames of emulation, during which the agent's chosen action is repeated. Since the Atari simulator runs at 60 frames per second, to run in real-time an agent needs to transmit an action at the relatively fast rate of 12 times per second. All of our agents run in real-time or faster.

To carry out our evaluation, we used a total of 51 Atari games[1]. Of these games, five were selected as a *training set*. The training set was used to design and test our algorithms, including parameter optimization. The remaining 46 games constitute the *test set*. Games in the test set were solely used for evaluation purposes. During the testing phase, each algorithm uses a fixed set of parameters across all games.

## 6.2 Learning the Contingent Regions

We first studied how well our models could learn to predict the contingent regions across all games, using both the offline and online techniques described in Section 4.2. In the offline setting we constructed the training set described in Equation 1 using the emulator's load/save state functions. In the online setting we first learn a pixel-level change model (Section 4.3), then generate the training set using a KL divergence test (Section 4.2).

In both settings, training data was generated by sampling game situations that occurred near a human-provided trajectory. For each game we collected one human trajectory lasting at least two minutes. These human trajectories do not need to involve expert policies, but rather help provide representative coverage of the histories. Hybrid trajectories were then constructed by following the human-provided trajectory for a variable number of steps and then taking 300 additional, uniformly random actions. Pixel-level change models were learned from 2,000,000 samples drawn at five-frame intervals. Predictors for both settings were then trained on 200,000 samples. The contingent region predictor's input $\boldsymbol{\Phi}_l(h)$ consisted of the color of pixels in a 21x21 neighborhood around pixel location $l$, for both the current and last time step. The divergence threshold parameter in Equation 2 was set to $\delta = 4.0$.

Table 1 reports relevant statistics for our learned models. The offline learning procedure yielded a highly accurate model. In practice, we found that for all games with

[1]See `http://arcadelearningenvironment.org` for a complete list.

| | Accuracy | | Precision | | Recall | |
|---|---|---|---|---|---|---|
| | Offline | Online | Offline | Online | Offline | Online |
| Mean | 0.969 | 0.919 | 0.396 | 0.217 | 0.573 | 0.356 |
| Min. | 0.800 | 0.365 | 0.016 | 0.000 | 0.000 | 0.000 |
| Max. | 1.000 | 0.998 | 0.769 | 0.836 | 0.985 | 0.985 |

Table 1: Prediction statistics for the learned contingent region models. Each model was learned once; minimum and maximum reflect prediction accuracy across the 51 games.

avatars, the contingent regions predictor correctly identified its surrounding contingent region. The low precision and recall values can be explained by our use of the block decomposition (Section 4.3). As illustrated in Figure 2, block decomposition yields accurate results for tracking purposes but cannot precisely capture the boundaries of the contingent regions. The accuracy of the online learning procedure varied widely across games. While the pixel change models were generally accurate, no single value of $\delta$ provided a good approximation of the contingent regions (Equation 2) for all games: *Space Invaders* worked best with $\delta = 0.1$, while *Beam Rider* required $\delta = 10.0$. The online results of Table 1 reflect our choice of $\delta = 4.0$ as a compromise.

### 6.3 Feature Generation Methods

We now describe two Atari-specific feature generation techniques and their subsequent contingency-aware extensions.

The *Basic* method, derived from BASS (Naddaf 2010), encodes the presence of colors on the Atari screen. The background is first removed using a histogram method, described in Naddaf's work (2010). Each game background is precomputed offline, using a sample trajectory containing 18,000 observations collected from a uniformly random policy. The screen is then divided into $16 \times 14$ tiles. Basic generates one binary feature for each of the 128 colors and each of the tiles, giving a total of 28,672 features.

*MaxCol* also divides the screen into tiles but encodes only its *two* most frequent colors. Unlike the Basic method, MaxCol does not rely on background detection, which can prove problematic in games where the background scrolls or changes color. In many cases, one of encoded colors is part of the background and the other indicates relevant state information. We used a $32 \times 30$ grid and the full 128-color set for a total of 122,880 features.

The *Extended* method first generates the Basic features, and then applies the technique described in Section 5. The avatar location, estimated from the output of our contingent regions predictor, is discretized into a $10 \times 10$ grid. The Extended feature set is then generated by jointly tiling the Basic features with each particular grid cell, yielding a total of 2,867,200 features. The *Extended MaxCol* method works similarly, except that it jointly tiles the avatar location with the MaxCol features, giving a total of 12,288,000 features.

### 6.4 Reinforcement Learning Setup

We trained agents using the well-known SARSA($\lambda$) reinforcement learning algorithm. We approximate the value function using linear approximation with one of the four feature sets described in Section 6.3. An $\epsilon$-greedy policy (Sutton and Barto 1998) is used, which takes exploratory actions with probability $\epsilon$, and otherwise chooses the action estimated to have highest value. Exploratory actions are chosen uniformly at random from $\mathcal{A}$.

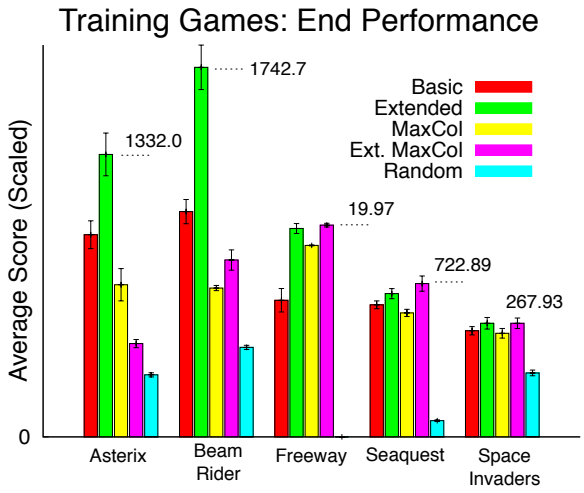


Figure 3: Average score over the last 500 episodes for each of the training games. Each result is an average over 30 trials; error bars show 99% confidence intervals.

We set the discount factor $\gamma = 0.999$, the eligibility trace parameter $\lambda = 0.9$ and the exploration rate $\epsilon = 0.05$. The learning rate $\alpha$ was tuned for each specific feature generation method by using parameter sweeps over the training games. The best values were 0.2, 0.1, 0.5, 0.5 for Basic, MaxCol, Extended and Extended MaxCol respectively. Note that these values are not directly comparable since the learning rates are normalized with respect to the number of active features (Sutton 1996).

### 6.5 Training Evaluation

We first evaluated our methods on the training games. Each feature set was tested using 30 independent trials per game. Each trial consisted in training the agent on a particular game for 10,000 episodes. The performance within a trial was obtained by taking the average score obtained during the last 500 episodes of training. The overall performance estimate we report is the average performance across all 30 trials.

The results we obtained on the training games are shown in Figure 3. Extended performs statistically better than Basic on 4/5 games, while Extended MaxCol performs better than MaxCol on 3/5 games and worse on *Asterix*. These results illustrate the advantage of using contingency awareness. We also observed (not shown here) that the performance of Basic agents reached a plateau within 10,000 episodes. In contrast, for *Asterix*, *Beam Rider* and *Space Invaders*, agents using Extended were still learning at the end of the trial. In *Asterix* we believe the lower performance of Extended MaxCol was a result of the agent controlling the appearance of the remaining lives icons; as such, the player avatar is only approximately tracked as long as the agent has more than one life remaining, resulting in a deteriorated feature set for

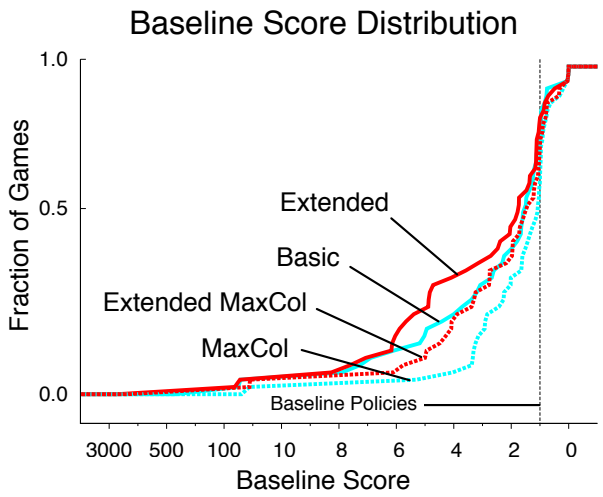


Figure 4: Score distribution for the testing games. Each line indicates the fraction of testing games that achieve a score at least the value on the $x$ axis.

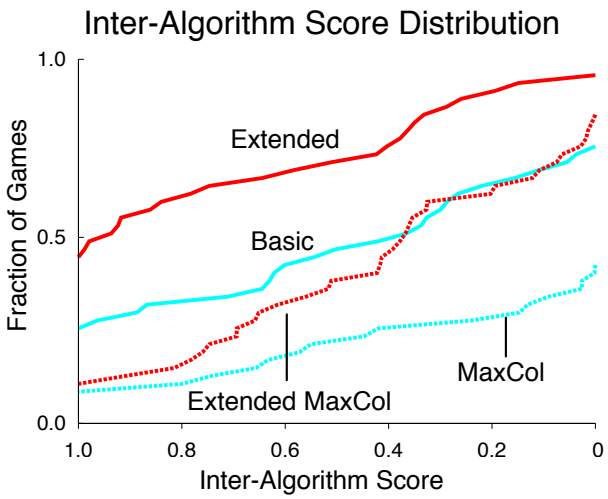


Figure 5: Inter-algorithm score distribution for the testing games.

a portion of the game. Despite this issue, Extended outperforms Basic in *Asterix*. All learning agents perform better than the random policy.

### 6.6 Testing Evaluation

We now evaluate the feature generation methods across the 46 previously unseen games in the test set. We used the same experimental setup as for the training games; the results of this section are based on a total of $30 \times 46 \times 4 = 5,520$ trials.

**Normalized Scores.** Comparing different methods over a large and diverse set of Atari games poses a challenge: each game uses its own scale for scores (human-level scores range from single digits to many thousands), and different game dynamics make some games harder to learn than others. We solve this problem by normalizing the scores for each game. We examine two different normalized scores: the *baseline score* and the *inter-algorithm score*. Both scores are computed using a score range $[r_{g,\min}, r_{g,\max}]$. Given the average score $s_{g,i}$ for game $g$ and method $i$, we compute the normalized score $z_{g,i} := (s_{g,i} - r_{g,\min})/(r_{g,\max} - r_{g,\min})$. Once we have normalized scores we display the results in the form of a *score distribution*: a graph showing the fraction of testing games for which a method achieves a certain normalized score or better. The graph can be thought of as the cumulative distribution function of the normalized score. If one curve is above another on a score distribution then that method generally has higher normalized scores.

**Baseline Score Distribution.** The first scoring metric we use normalizes with a set of 19 baseline policies: 18 single-action policies and the uniform random policy. Each single-action policy selects a fixed action with probability 0.95 and otherwise acts uniformly randomly. We obtained the average score $b_{g,i}$ achieved by each baseline policy $i$ in game $g$. The baseline score range was computed as $r_{g,\min} := \min_i\{b_{g,i}\}$ and $r_{g,\max} := \max_i\{b_{g,i}\}$. Under this metric, a baseline score greater than 1.0 indicates that a method performs better than all baseline policies, and less than 0.0 indicates performance worse than all baseline policies. Figure 4 shows the score distribution of the baseline scores for all four feature sets. The graph demonstrates that learning takes place across all methods in the majority of the games, as most games have scores at least as large as 1.0. Furthermore, for games where learning occurred, incorporating contingency information generally gave an improvement.

**Inter-algorithm Score Distribution.** An alternate way to compute the performance is to normalize using the scores achieved by the algorithms themselves. Given a set of scores $\{s_{g,i}\}$, we define the inter-algorithm score range as $r_{g,\min} = \min_i\{s_{g,i}\}$ and $r_{g,\max} = \max_i\{s_{g,i}\}$. By construction, each inter-algorithm score lies within the $[0, 1]$ interval. Figure 5 shows the inter-algorithm score distribution over the testing games for our four algorithms. The differences observed in Figure 4 are clearer: Extended performs better than Basic, and Extended MaxCol performs better than MaxCol. In particular, Extended achieves the highest score on nearly half of the games.

We also computed the proportion of games for which the extended methods provide a statistically significant (i.e. non-overlapping 99% confidence intervals) advantage. Extended did better than Basic in 17 games and worse in 7, while Extended MaxCol did better than MaxCol in 22 games and worse in 6. These results highlight the benefits of using contingency information for feature construction.

### 6.7 Online Contingency Learning

We now evaluate contingency-based feature generation using the online contingent regions method of Section 4.2. Figure 6 shows the resulting score distributions (normalized using the baseline policies) for MaxCol, Extended Max-

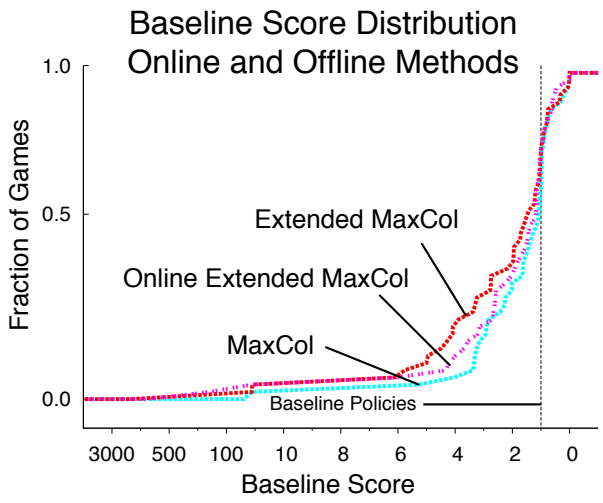


Figure 6: Score distribution comparing the offline and online methods on the testing games.

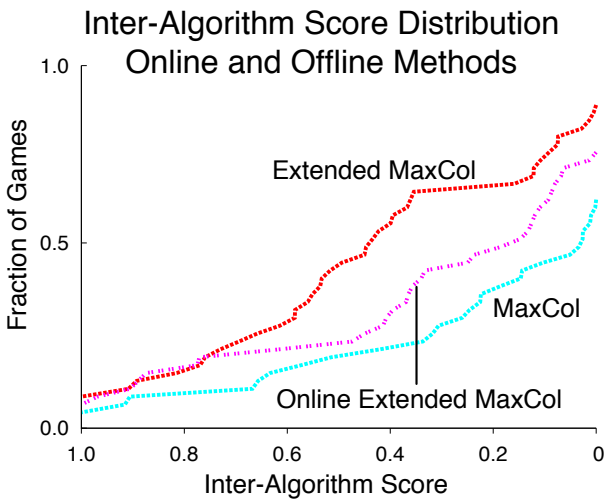


Figure 7: Inter-algorithm score distribution comparing the offline and online methods on the testing games.

Col and Online Extended MaxCol. Across the 46 testing games, the online method showed significant improvements over MaxCol. Unsurprisingly, the offline method performed better than its online counterpart. We omit the comparison between Basic and Online Extended as our results did not exhibit a statistically significant change in performance. As discussed in Section 6.2, our single choice of divergence threshold led in many cases to a loss of accuracy in the contingent regions predictor, which made it difficult to track the player avatar. We feel this explains the lack of improvement using the Extended feature set in the online setting, though further investigation is needed.

We also evaluated the performance of our online method using inter-algorithm scores. Figure 7 shows the score distribution for MaxCol, Extended MaxCol and Online Extended MaxCol. The score range was computed using all six algorithms to facilitate comparisons with Figure 5. Figure 7 shows that Online Extended MaxCol outperforms MaxCol, but does not achieve the performance level of the offline method.

## 7 Limitations

There are three main sources of error that can affect the performance of our contingency-aware agents. The first is due to learning the contingent regions model. As with any feature-based learning technique, the chosen set of features can have a significant influence on the final performance. The second source of error is related to the avatar tracking procedure. Here we rely on the assumption that the location of the largest contingent region provides useful information. This assumption does not always hold in side-scrolling and first-person games, in which large portions of the screen are always under the agent's control. Where the assumption fails, the contingency aware features provide no useful information; this seems however to have only a minor negative effect, as Extended performed significantly worse than Basic in only one of the 15 side-scrolling and first-person games. Finally, online contingency learning suffers from additional error introduced by using the learned pixel transition model.

## 8 Discussion

Although this paper has focused on exploiting contingency awareness within Atari 2600 games, we feel the idea has relevance to other settings. For example, our notion of contingency awareness naturally extends to domains in which the observation space is large and factored (Degris, Sigaud, and Wuillemin 2006). It also seems plausible that contingency awareness could be useful for option discovery (McGovern 2002), for example by restricting option policies to only use contingent regions as state information.

The Atari 2600 domain holds many challenges not addressed in this work. In particular, side-scrolling games (15 out of our 51 total games) rarely benefit from the extended feature sets; we are currently investigating alternate ways of tracking the player avatar. In some games, the screen contains multiple important contingent regions. Here, avatar tracking only exploits part of the information contained in the contingent regions model. Finally, six of the games investigated were simply too difficult for any of our methods. Solving these harder games may require more elaborate reinforcement learning techniques, such as smarter exploration (Brafman and Tennenholtz 2003), temporally extended actions (Sutton, Precup, and Singh 1999) or Monte-Carlo Tree Search (Kocsis and Szepesvári 2006). We feel the Atari platform is an ideal testbed for empirically evaluating the general applicability of both present and future reinforcement learning algorithms.

## 9 Conclusion

We have investigated a notion of contingency awareness using the Atari 2600 domain. We laid out a general mathematical framework for learning contingent regions in Atari, and subsequently described a way to exploit these notions in value function approximation for reinforcement learning.

We evaluated our techniques by dividing a set of 51 Atari games into a training and a test set, which allowed us to accurately measure the performance of our techniques on a diverse range of previously unseen games. We found that contingency awareness in combination with a Bayes filter led to a robust solution for tracking the player avatar in a domain-independent manner. Furthermore, our results show that contingency awareness can significantly improve the performance of existing feature construction methods by adding contingency-specific features.

**Acknowledgments** We would like to thank Yavar Naddaf, Adam White, and Anna Koop for helpful discussions. This research was supported by the Alberta Innovates Technology Futures and the Alberta Innovates Centre for Machine Learning at the University of Alberta. Invaluable computational resources were provided by Westgrid, the Réseau Québécois de Calcul de Haute Performance and Compute Canada.

## References

Brafman, R. I., and Tennenholtz, M. 2003. R-max — a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research* 3:213–231.

Cobo, L. C.; Zang, P.; Isbell, C. L.; and Thomaz, A. L. 2011. Automatic state abstraction from demonstration. In *Proceedings of the 22nd Second International Joint Conference on Articial Intelligence (IJCAI)*.

Degris, T.; Sigaud, O.; and Wuillemin, P. 2006. Learning the structure of factored markov decision processes in reinforcement learning problems. In *Proceedings of the 23rd International Conference on Machine learning*. ACM.

Diuk, C.; Cohen, A.; and Littman, M. L. 2008. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th International Conference on Machine learning (ICML)*, 240–247.

Finkelstein, N. W., and Ramey, C. T. 1977. Learning to control the environment in infancy. *Child Development* 48(3):806–819.

Kocsis, L., and Szepesvári, Cs. 2006. Bandit based Monte-Carlo planning. In *Proceedings of 17th European Conference on Machine Learning (ECML)*.

Konidaris, G. D.; Osentoski, S.; and Thomas, P. S. 2011. Value function approximation in reinforcement learning using the fourier basis. In *Proceedings of the 25th Conference on Artificial Intelligence (AAAI)*.

McGovern, A. E. 2002. *Autonomous discovery of temporal abstractions from interaction with an environment*. Ph.D. Dissertation, University of Massachusetts.

Naddaf, Y. 2010. Game-Independent AI Agents for Playing Atari 2600 Console Games. Master's thesis, University of Alberta.

Parr, R.; Painter-Wakefield, C.; Li, L.; and Littman, M. 2007. Analyzing feature generation for value-function approximation. In *Proceedings of the 24th International Conference on Machine learning*.

The Arcade Learning Environment: A Platform for AI Research. 2012. http://arcadelearningenvironment.org.

Russell, S. J., and Norvig, P. 2010. *Artificial Intelligence - A Modern Approach (3rd internat. ed.)*. Pearson Education.

Schweitzer, P. J., and Seidmann, A. 1985. Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications* 110(2):568–582.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT Press.

Sutton, R. S.; Precup, D.; and Singh, S. P. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112(1-2):181–211.

Sutton, R. S. 1996. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8 (NIPS)*.

Tesauro, G. 1995. Temporal Difference Learning and TD-Gammon. *Communications of the ACM* 38(3).

Watkins, C., and Dayan, P. 1992. Q-learning. *Machine Learning* 8:279–292.

Watson, J. S., and Ramey, C. T. 1972. Reactions to response-contingent stimulation in early infancy. *Merrill-Palmer Quarterly: Journal of Developmental Psychology* 18(3):219–227.

White, R. W. 1959. Motivation reconsidered: The concept of competence. *Psychological Review* 66(5):297–333.

Wintermute, S. 2010. Using imagery to simplify perceptual abstraction in reinforcement learning agents. In *Proceedings of the the 24th Conference on Artificial Intelligence (AAAI)*.