The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)

Deriving Subgoals Autonomously to Accelerate Learning in Sparse Reward Domains

Michael Dann, Fabio Zambetta, John Thangarajah
Computer Science
RMIT University, Australia
{michael.dann, fabio.zambetta, john.thangarajah}@rmit.edu.au

Abstract

Sparse reward games, such as the infamous *Montezuma's Revenge*, pose a significant challenge for Reinforcement Learning (RL) agents. Hierarchical RL, which promotes efficient exploration via subgoals, has shown promise in these games. However, existing agents rely either on human domain knowledge or slow autonomous methods to derive suitable subgoals. In this work, we describe a new, autonomous approach for deriving subgoals from raw pixels that is more efficient than competing methods. We propose a novel intrinsic reward scheme for exploiting the derived subgoals, applying it to three *Atari* games with sparse rewards. Our agent's performance is comparable to that of state-of-the-art methods, demonstrating the usefulness of the subgoals found.

Introduction

In recent years, videogames have become an increasingly popular domain for artificial intelligence research. This is in large part due to the famous *Deep Q-Network (DQN)* agent of Mnih et al. (2015), which learned to play many *Atari 2600* games to human level from only raw pixel input and a feed of the game score. Impressive as this was, DQN fared far better in reflex-driven games with frequent rewards, such as *Video Pinball*, than in adventure games with sparse rewards. Notoriously, it failed to learn a path to the first key in *Montezuma's Revenge* after more than a month of experience. DQN's weakness in sparse reward *Atari* games has driven much subsequent research (Bellemare et al. 2016a; Kulkarni et al. 2016; Osband et al. 2016; Martin et al. 2017; Ostrovski et al. 2017; Vezhnevets et al. 2017; Roderick, Grimm, and Tellex 2018), as it is symptomatic of a broader difficulty in applying reinforcement learning (RL) to long-term, real-time planning problems.

The reason DQN struggles in sparse reward games is that until the agent has discovered some reward, it sees no incentive to favour one course of action over any other. It responds to this predicament in a manner typical of many learning-based agents: by simply choosing actions at random. In games with dense rewards, such as *Video Pinball*, random action selection is often sufficient for the agent to find rewards and start improving. However, in sparse reward games the agent can become stuck in a "chicken-egg" scenario, where it cannot improve its policy until it finds some reward, but it cannot discover any rewards until it improves its policy.

It has long been thought that *hierarchical reinforcement learning* (Parr and Russell 1998; Sutton, Precup, and Singh 1999; Dietterich 2000), which splits drawn-out tasks into subtasks, may be key to solving sparse reward problems efficiently. In fact, it has already been established that hierarchical agents can make rapid progress in sparse reward *Atari* games, so long as they are provided with appropriate subgoals by a human expert (Kulkarni et al. 2016; Roderick, Grimm, and Tellex 2018). Of course, the major deficiency of such agents is that they do not address subgoal identification, which is a very challenging part of the problem. Despite decades of research (Vezhnevets et al. 2017; Thrun, Schwartz, and others 1995; Digney 1998; Şimşek, Wolfe, and Barto 2005; Konidaris and Barto 2009; Machado, Bellemare, and Bowling 2017; Machado et al. 2018b), identifying subgoals from high-dimensional input, such as *Atari* pixel representations, remains a major open problem.

The main contribution of this work is a new, autonomous method for deriving subgoals in such domains. The method operates by partitioning the state space according to a novel heuristic distance measure, which we term *exploration effort*. The approach can readily be applied to raw visual input, and the subgoals it identifies in *Atari* games resemble those identified by human experts in previous work.

To leverage the subgoals found, we propose a novel intrinsic reward scheme, dubbed *pellet rewards*. The scheme encourages the agent to explore via a mechanism similar to the collectable pellets in *Ms. Pacman* and the coins in *Super Mario Bros.* We apply this approach to three *Atari* games with sparse rewards (*Venture*, *Freeway* and *Montezuma's Revenge*), achieving similar performance to state-of-the-art methods based on visual density models (Bellemare et al. 2016a; Ostrovski et al. 2017). This makes ours the first fully-autonomous subgoal-oriented agent to be truly competitive with state-of-the-art agents on these games. In addition, pellet rewards appear to have relatively little distortion on the original task objective, causing no discernible harm in two games (*Battlezone* and *Robot Tank*) where Ostrovski et al.'s (2017) method was detrimental.

Background

In this section we briefly introduce, and discuss some open issues with, two lines of work that have shown promise in addressing reward sparsity in videogames: count-based novelty bonuses derived from visual density models, and hierarchical reinforcement learning. While not all recent work on the problem falls into one of these categories (Osband et al. 2016; Machado, Srinivasan, and Bowling 2015; Stadie, Levine, and Abbeel 2015), to the best of our knowledge no alternative methods have achieved near the same level of success on the most challenging games.

Count-Based Novelty from Visual Density Models

The idea behind count-based novelty bonuses is to maintain a visit count for each state, $n(s)$, and pay the agent an *intrinsic reward* (Bellemare et al. 2016a) for encountering rarely-visited states. While simple conceptually, a major challenge in applying count-based methods to *Atari* games is that it is impractical to maintain a visit count for each individual state. Bellemare et al. (2016a) address this issue by deriving a *pseudo-count* from a state density model. Roughly speaking, this method generalises states' visit counts based on their visual similarity. Later work explores the use of alternative density models (Martin et al. 2017; Ostrovski et al. 2017), though these too can be broadly classified as visual similarity methods.

The density model approach has yielded state-of-the-art performance on several sparse reward games (Bellemare et al. 2016a; Ostrovski et al. 2017). However, it struggles notably in the following scenarios: (1) When the agent's learning progression requires "risky" exploration. (2) When visual novelty is a poor proxy for true progress in the game. We shall now describe these issues in some detail, since to date they have received little attention in the literature (barring Roderick, Grimm, and Tellex (2018), who note one aspect of the second issue).

Issue #1: Risky Exploration. In videogames where the player has a certain number of lives, the question arises of what should constitute the end of an episode: (A) When the player loses a life, or (B) When the player has lost *all* lives. In most work on *Atari* games to date, including the original DQN paper (Mnih et al. 2015), choice (A) was made. This setting usually yields better performance, as it helps the agent learn to avoid life loss. Interestingly though, this choice was heavily detrimental to Bellemare et al.'s (2016a) agent, which achieved far higher scores under setting (B). This phenomenon was likely due in part to the form of the novelty bonus. In all novelty-based work we have cited (Bellemare et al. 2016a; Martin et al. 2017; Ostrovski et al. 2017), the agent is paid an intrinsic reward of the following form at each time step:

$$r^{+}(s) = \frac{\beta}{\sqrt{n(s)}} \tag{1}$$

where β is a reward scaling factor. Under this scheme, the agent can accumulate large returns in rarely-visited states by remaining stationary and "soaking up" the novelty bonus. If episodes are deemed to terminate upon life loss, the agent is discouraged from exploring risky manoeuvres in such states. Thus, the scheme may actually have an anti-exploratory effect on tasks that require the agent to risk life loss in order to learn, such as jumping over the first skull in *Montezuma's Revenge*.

Issue #2: Visual Novelty $\neq$ True Progress. In some videogames, minor visual differences between states may indicate important strategic differences regarding the task at hand. Conversely, large visual differences may be inconsequential. For example, on the first screen of *Venture*, the enemy sprites are far larger than that of the protagonist. Therefore, under visual novelty schemes, the agent may receive large rewards for inconsequential enemy movements, but receive little reward for navigating to rarely explored parts of the map. The same type of issue caused Martin et al.'s (2017) agent to struggle on *Freeway*, where it was "awed" by the continually changing traffic.

Similarly, Roderick, Grimm, and Tellex (2018) note that in *Montezuma's Revenge*, visual novelty schemes discourage the agent from retracing its steps back to the locked doors after obtaining the first key. This is because visual density models fail to recognise that pre- and post-key acquisition states are fundamentally different, instead seeing all states where the protagonist is located near the start position as being relatively "boring". Electing not to terminate episodes with life loss masks this issue, as suiciding to reach the doors becomes the optimal solution[1].

Hierarchical Learning Approaches

In hierarchical reinforcement learning, agents are trained to perform temporally extended subtasks in the hope of simplifying longer-term tasks. In a videogame context, subtasks might include "reach the bottom left room" in *Venture*, or "descend the central ladder" in *Montezuma's Revenge*. This approach seems well-suited to sparse reward environments because, by increasing the time scale over which the agent acts, the number of decisions the agent must make to reach a distant reward is effectively reduced. Also note that hierarchical RL introduces abstraction; a property that seems lacking from visual density methods. Specifically, it introduces *temporal abstraction* (as the time taken to complete a subtask may be uncertain) and *state abstraction* (as the subtask definitions may omit certain state details).

Given suitable subgoal definitions, hierarchical RL may progress much faster than ordinary RL. Kulkarni et al. (2016) demonstrated that an agent equipped with high-level waypoints for the first room of *Montezuma's Revenge* could learn to exit the room faster than the strongest novelty-driven agents. Roderick, Grimm, and Tellex (2018) took this further by providing their agent with a factored state abstraction, including information such as "the player has the key". From this, their agent learned to retrace its steps and exit the first room using only a single life.

[1]Videos of Bellemare et al.'s (2016a) agent and Ostrovski et al.'s (2017) agent demonstrating the suicide approach are available at https://youtu.be/0yI2wJ6F8r0 (skip to 0:51) and http://youtu.be/232tOUPKPoQ.

The downside of these approaches is that there currently exists no efficient, autonomous method for identifying subgoals in state spaces the size of *Atari* pixel representations. Zahavy et al. (2016) were able to identify subgoals for some games by clustering the Q-network's hidden layer activations. However, this method requires a trained Q-network, which makes it unhelpful during the critical learning phase in sparse reward games. Vezhnevets et al.'s (2017) *FeUdal Networks* did eventually identify useful subgoals for *Montezuma's Revenge*, but took an extremely long time to reach competitive performance (around half a year of experience). To the best our knowledge, the recent approach of *eigenoptions* (Machado, Bellemare, and Bowling 2017; Machado et al. 2018b) has not yet yielded a competitive agent for sparse reward *Atari* games.

State Space Partitioning via a Measure

Inspired by some prominent historical work in hierarchical RL (Dietterich 2000; Hengst 2002), our approach to subgoal identification centres around a partitioning of the state space. The idea is to reward the agent for reaching rarely-visited partitions, thus incentivising exploration. Under this approach, entering a partition is synonymous with achieving a subgoal.

To perform the partitioning, we make use of a heuristic distance measure $d : \mathcal{S} \times \mathcal{S} \to \mathbb{R}^{+} \cup \{0\}$ that maps a pair of states to a measure of their dissimilarity. In the next section, we propose a specific heuristic for generating partitions that contain "strategically similar" states. For now though, merely note that there are multiple ways by which one can derive partitions from a distance measure. One such method is sketched as follows:

Partition via a fixed radius: Create a node at the initial state and define a fixed partition radius. As soon as the agent encounters a state outside this radius, add another node at that point. Keep adding nodes whenever the current state is outside all existing nodes' radii. The nodes can be thought of as "representative states" for the set of partitions. The partition, p, to which a state belongs is the one whose representative state, $s_p \in \mathcal{R}$, is closest.

Unfortunately, under the distance measure introduced in the next section, preliminary experiments revealed the efficacy of this method to be highly sensitive to the partition radius. Without careful tuning, it frequently yielded too few or far too many partitions. Therefore, we opted for the following alternative, which fixes the number of partitions that exist at any given time and does not require a partition radius:

Partition via a schedule: Create a node at the initial state then act according to some policy for a number of time steps. During this period, keep track of the farthest state discovered from the set of existing nodes. Periodically add that state to the set of nodes and restart the process. Again, treat the nodes as representative states for partitions.

More explicit details can be found within our full pseudocode at the end of this paper (see Algorithm 1).

Exploration Effort (EE)

Under the approach outlined so far, different partitionings of the state space may promote exploration to differing degrees. For example, consider the game *Freeway*, where the aim is to navigate a chicken to the other side of a busy road. If one were to partition the state space via a visual dissimilarity measure, the resulting partitions would most likely contain states with similar traffic configurations, as traffic accounts for most of the visual variety in *Freeway*. In that case, rewarding the agent for reaching rarely-visited partitions would be unlikely to help, as the player has no control over the traffic. On the other hand, if the state space were partitioned according to the chicken's position, the agent would be incentivised to reach the rarer, middle-of-the-road and top-of-the-road positions.

Based on this reasoning, it seems we require a distance measure that regards states like top-of-the-road and bottom-of-the-road positions in *Freeway* as being far apart. Indeed, from an exploration perspective, there is an important sense in which such states truly are "far apart": During the early stages of training, a decaying ϵ-greedy policy is unlikely to take the chicken from the bottom of the road to the top, because the policy will be near-uniform random at this stage and thus unlikely to oversample *up* actions sufficiently. This suggests that we can derive a suitable distance measure from the amount of action over/undersampling required to transition between states.

To be clear, we do not claim that this an appropriate heuristic for all problems. For example, in a "combination safe" task where only one sequence of actions will successfully open the safe, the heuristic described does not provide a meaningful measure of progress. However, a broad problem class to which the heuristic does seem well-suited is domains where the agent is controlling a physical entity. Indeed, similar intuition underlies the use of autocorrelated noise in continuous control tasks (Wawrzynski 2015).

Formal Definition of EE

Continuing the above line of thought, we seek a function $\mathcal{E}^{\pi} : \mathcal{S} \times \mathcal{S} \to \mathbb{R}^{n}$ that takes two states as input and returns, for each of the n actions available[2], a measure of how much that action must be oversampled (relative to the agent's current policy, π) in order to transition from the former state to the latter. To this end, we begin by defining an n-dimensional auxiliary reward scheme. At each time step, the auxiliary reward vector is:

$$\hat{r}^{\pi}(s,a) = \kappa \langle \hat{r}^{\pi}_{1}(s,a),\ \hat{r}^{\pi}_{2}(s,a),\ \ldots\ ,\hat{r}^{\pi}_{n}(s,a) \rangle \qquad (2)$$

where $\kappa > 0$ is a reward scaling factor and

$$\hat{r}^{\pi}_{i}(s,a) = \begin{cases} 1 - \pi(s,a_i), & \text{if } a = a_i \\ -\pi(s,a_i), & \text{if } a \neq a_i \end{cases} \qquad (3)$$

The form of the reward is deliberately chosen so that the expected reward vector under the agent's current policy is zero. However, if an action is over/undersampled for a period of time, the sum of rewards in the corresponding

[2]We assume in this work that the action space is discrete.

dimension will be positive/negative. To clarify this point, we now provide a worked example:

Example 1. Let $\mathcal{A} = \langle$*up, down, left, right*$\rangle$, and suppose that an agent transitioned from s_0 to s_4 by pressing *up, right, right, down* under a uniform random policy. Let $\gamma = 0.99$ and $\kappa = 1$. Then:

$$\hat{r}^\pi(s_0, a_0) = \langle \tfrac{3}{4},\ -\tfrac{1}{4},\ -\tfrac{1}{4},\ -\tfrac{1}{4} \rangle$$
$$\hat{r}^\pi(s_1, a_1) = \langle -\tfrac{1}{4},\ -\tfrac{1}{4},\ -\tfrac{1}{4},\ \tfrac{3}{4} \rangle$$
$$\hat{r}^\pi(s_2, a_2) = \langle -\tfrac{1}{4},\ -\tfrac{1}{4},\ -\tfrac{1}{4},\ \tfrac{3}{4} \rangle$$
$$\hat{r}^\pi(s_3, a_3) = \langle -\tfrac{1}{4},\ \tfrac{3}{4},\ -\tfrac{1}{4},\ -\tfrac{1}{4} \rangle$$

Thus the sampled (Monte Carlo) auxiliary return is:

$$\begin{aligned}&\hat{r}^\pi(s_0, a_0) + \gamma \hat{r}^\pi(s_1, a_1) + \gamma^2 \hat{r}^\pi(s_2, a_2) + \gamma^3 \hat{r}^\pi(s_3, a_3)\\ =&\langle 0.015,\ -0.015,\ -0.985,\ 0.985 \rangle\end{aligned}$$

This reflects the fact that the *left* action was undersampled while *right* was oversampled.

△

Continuing in this vein, we define the *exploration effort function* as follows:

Definition 1. The exploration effort from s to s' (under time limit m) is the expected auxiliary return when the agent transitions from s to s' within m steps, via policy π:

$$\mathcal{E}^\pi_m(s, s') = \mathbb{E}_\pi[\sum_{k=0}^{T-1} \gamma^k \hat{r}^\pi(s_k, a_k) \,|\, s_0 = s, s_T = s', T < m] \tag{4}$$

△

In the absence of prior knowledge, the auxiliary rewards in Equation 3 have zero expectation under π. However, knowledge of s' means that the expectation in Equation 4 may be non-zero. Returning to the example in *Freeway*, if we knew that the chicken was at the bottom of the road in s, then at the top of the road in s', we would expect the interleaving auxiliary return in the *up* dimension to be positive (assuming π is untrained and does not yet favour the up button).

To train an estimator for the exploration effort function, we sample state pairs from the agent's replay memory that are less than m time steps apart, as well as the interleaving auxiliary rewards. In addition to calculating Monte Carlo targets, as per Example 1, we also calculate one-step targets as follows:

$$\mathcal{E}^\pi_m(s_t, s')_{\text{one-step target}} = \hat{r}(s_t, a_t) + \gamma \mathcal{E}^\pi_{m-1}(s_{t+1}, s')$$

This allows us to train towards a *mixed Monte Carlo target* (Bellemare et al. 2016a), using proportion η of the Monte Carlo target and proportion $(1 - \eta)$ of the one-step target. Using a mixed return helps mitigate the variance of the Monte Carlo estimates, while also propagating distant rewards faster than pure one-step updating. To avoid maintaining separate estimators for $\mathcal{E}^\pi_m$ and $\mathcal{E}^\pi_{m-1}$, we bootstrap one-step targets from $\mathcal{E}^\pi_m$, with the justification that the functions should be very similar for large enough m.

Deriving a Distance Measure from EE

In certain situations, the exploration effort between strategically similar states may be large. For example, suppose that an agent is playing *Montezuma's Revenge* via a uniform random policy, and that the protagonist was positioned against a wall in s, then later against the same wall in s'. Given this knowledge, it is likely the policy oversampled actions that ran the protagonist into the wall versus those that would have escaped the wall. Therefore, simply treating the magnitude of the exploration effort vector as distance is inappropriate. In the worst case, it could cause the state partitioning algorithm to generate duplicate representative states.

Fortunately, we can address this issue via a mathematical method. Let $\hat{s} \in S$ be an arbitrary *reference point*. Then, define the distance between s and s' relative to $\hat{s}$ as:

$$\begin{aligned}d^{\hat{s}}_{\pi,m}(s, s') = \max(&\|\mathcal{E}^\pi_m(\hat{s}, s) - \mathcal{E}^\pi_m(\hat{s}, s')\|,\\ &\|\mathcal{E}^\pi_m(s, \hat{s}) - \mathcal{E}^\pi_m(s', \hat{s})\|)\end{aligned} \tag{5}$$

To paraphrase, this measure finds the displacement of both s and s' from the reference point, treating the magnitude of the difference as distance. Observe that the distance from a state to itself is always zero, and the two-way maximum ensures that the measure is invariant to the order of the arguments. Taking this one step further, we define the distance between s and s' relative to a *reference set*, $\hat{S} \subseteq S$, as:

$$d^{\hat{S}}_{\pi,m}(s, s') = \max_{\hat{s} \in \hat{S}} d^{\hat{s}}_{\pi,m}(s, s') \tag{6}$$

In our implementation we use this measure, with $\hat{S}$ equal to the set of representative states, $\mathcal{R}$.

Pellet Rewards

The approach presented thus far does in fact yield partitions for sparse reward *Atari* games that resemble intuitive subgoals. (Skip to Figure 3 for a preview.) However, we have not yet shown how these partitions can be exploited, nor demonstrated that such an approach actually aids exploration. To enable us to address these points, in this section we propose a partition-based intrinsic reward scheme, dubbed *pellet rewards*. Our approach takes its name and inspiration from the collectable pellets in *Ms. Pacman* which, from an exploration perspective, have a number of desirable effects:

1. They compel the player to perform an approximate depth-first search of the environment, as it is generally more efficient to continue forward rather than to retreat over a trail of consumed pellets. However, they do not disincentivise the player from backtracking upon hitting a dead-end.
2. The player has no incentive to remain stationary after collecting a pellet (which address the issue of "soaking up" novelty bonuses, as discussed in the background section).
3. The player's incentive to avoid death depends on how many pellets they have already collected. Once the player has collected all low-risk pellets, there is no reason not to attempt high-risk pellets.

We mimic the pellet mechanic by paying the agent a novelty bonus on its first transition (per episode) into a partition. Visited partitions become "collected", meaning no further novelty bonus is paid for reaching those partitions until the next episode. Similarly to previous schemes (Bellemare et al. 2016a; Martin et al. 2017; Ostrovski et al. 2017), the bonus takes the form $\beta/\sqrt{n_p}$, where β is a constant scale factor and n_p is partition's visit count in episodes.

One downside of creating partitions on a schedule is that sometimes the algorithm creates seemingly redundant partitions in regions that have already been thoroughly explored. This has the potential to cause destabilising novelty bonuses. Further, since the distance measure is continually changing, visits that were assigned to a partition at an earlier time may no longer be valid. To address these issues, we calculate n_p as an *inferred visit count*. That is, we calculate a moving average of the partition's visit rate, r_p, and set $n_p = r_p \times totalEpisodeCount$. This ensures that new partitions that are frequently visited quickly attain large counts, and partitions whose borders change significantly will also have their counts shift accordingly.

Experimental Configuration

In this section, we focus on explaining non-standard or otherwise pertinent aspects of our configuration for *Atari* games that warrant some comment. All other settings can be found in our source code[3].

Environment. We used version 0.6 of the *Arcade Learning Environment (ALE)* (Bellemare et al. 2013), injecting stochasticity via *sticky actions* (Machado et al. 2018a) with a stickiness of 0.25. To expose the agent to "risky exploration" scenarios (as explained in the background), we considered episodes to be terminated upon life loss.

Architecture. Since the exploration effort function takes two screens as input and acts as a kind of similarity measure, we approximated it via a *Siamese* architecture (Bromley et al. 1994). First, the screens are passed through two parallel encoders, whose weights are shared. The structure of the encoders is identical to the combined preprocessing and convolutional section of Mnih et al.'s (2015) network, except that we reduce the number of convolutional filters from (32, 64, 64) to (16, 16, 16) and input only a single screen rather than a four frame history. The encoders feed into a parallel mean and subtraction layer, followed by a fully connected layer of 128 units, then finally into an output layer with one node per action. Our Q-network architecture matches that of Mnih et al. (2015), except that we additionally feed the values of the collected pellets to the fully connected layer. As in Bellemare et al. (2016a), the Q-function is trained via mixed Monte Carlo updates.

Exploration Effort Training Parameters. The auxiliary reward scale factor, κ, was set to 1. The time separation constant, m, was set to 100. Both the EE and Q-function were trained via mixed Monte Carlo updates with $\eta = 0.1$. To further mitigate the large variance of the EE targets, the Monte Carlo deltas were clipped to 0.5. Prior to commencing Q-learning, the EE function was trained for 8 million frames (2 million samples) on experience generated via a uniform random policy. From that point on, it was trained in tandem with the Q-function.

Partition / Pellet Configuration. Pellet rewards were calculated using partition visits at sample time rather than the time of insertion into the replay memory. We set the pellet reward scale factor, β, to 1, but clipped bonuses to a maximum of 0.1. The time between partition additions was initially set to 80,000 frames, then increased by 20% with each addition. Experience collection for Q-learning commenced once there were 5 partitions in existence.

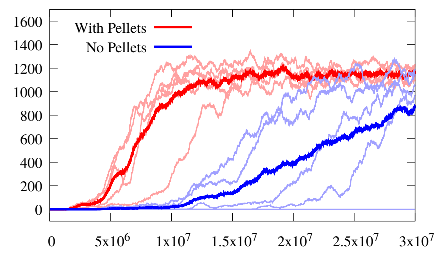
Improved Baseline Exploration. The usual ϵ-greedy decay schedule, whereby ϵ is annealed to a small value over the first 4 million frames (Mnih et al. 2015), risks "dooming" agents that have discovered few rewards by the end of the schedule. In essence, favouring the greedy action 90+% of the time *before* the agent has learned anything is too committal and may harm the agent's chances of ever learning. To mitigate this issue, we fix $\epsilon = 1$ for the remainder of the episode whenever the agent exceeds 500 actions without receiving a reward (either a pellet or an extrinsic reward). Our rationale is that DQN's default discount of $\gamma = 0.99$ effectively limits the agent's planning horizon to around 500 actions (as $0.99^{500} \approx 0.01$). Assuming there is some way to achieve a positive return within this time frame, rewardless trajectories of greater length indicate that the agent may not have explored the current region sufficiently. As we shall see in the next section, this tweak yields a more competitive benchmark and helps contextualise certain results.

Results and Discussion

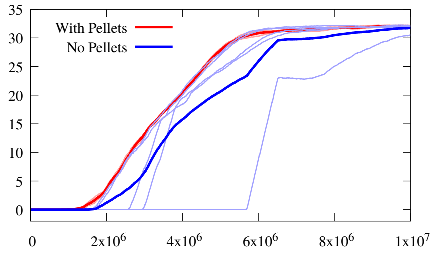
We applied our approach to three sparse reward *Atari* games: *Venture*, *Freeway* and *Montezuma's Revenge*. In addition, we tested it on two games with dense rewards: *Battlezone* and *Robot Tank*. The reason for including the latter games was not to see whether pellet rewards would be beneficial, but rather to see if they would be detrimental in games where standard DQN already performs well. In previous work (Ostrovski et al. 2017), novelty bonuses had an adverse affect in the two games chosen.

We benchmarked our method against a configuration that was identical in every respect except that pellet rewards were turned off. For each game, we conducted 5 training runs per agent. To make it easier to see the effect of pellet rewards on learning progress, we have plotted all training curves in Figures 1 and 2 from the point where Q-learning commenced. However, it should be noted that the pellet rewards agent was given an additional 8 million frames to pre-train the exploration effort function, plus a further 460,000 frames to generate the first 5 partitions. For *Freeway*, the graph's time scale has been shrunk to emphasise the early learning phase. At 30 million frames, the agents' average scores were virtually identical (pellet rewards: 33.4, baseline: 33.3).

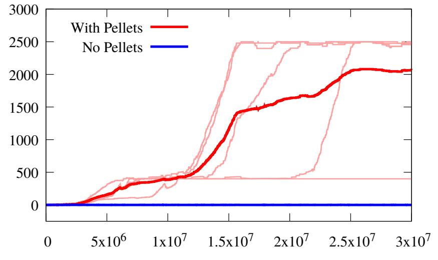
[3]https://bitbucket.org/mchldann/aaai2019



(a) *Venture*

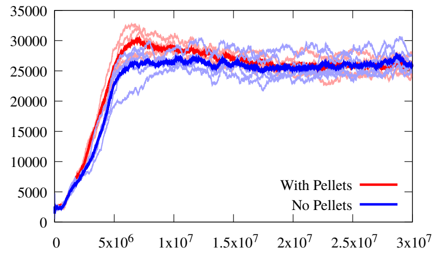


(b) *Freeway*

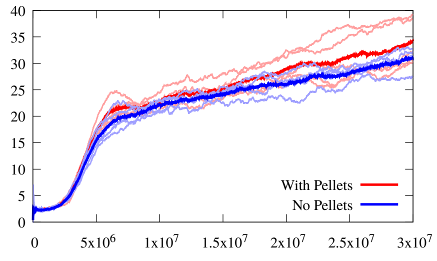


(c) *Montezuma's Revenge*

Figure 1: Average score versus training frames on sparse reward games (averaged over the last 100 episodes).



(a) *Battlezone*



(b) *Robot Tank*

Figure 2: Average score versus training frames on dense reward games (averaged over the last 100 episodes).

In all three sparse reward games, our approach identified meaningful subgoals. In *Venture*, it placed subgoals in all the main rooms. In *Freeway*, it generated representative states resembling intermediate waypoints for the chicken crossing the road. For *Montezuma's Revenge*, the first 20 representative states found on a particular run are shown in Figure 3. Like the human experts in Kulkarni et al.'s (2016) work, our method generated subgoals corresponding to the bottom-right ladder (#2 and #6), bottom-left ladder (#8), top-right door (#5), top-left door (#3) and middle-ladder (#4 and #7).

The resultant pellet rewards clearly aided extrinsic reward discovery in these games, as evidenced by the training curves in Figure 1. In *Venture*, all runs of the pellet agent reached an average of ≈1000 points by 15 million frames, while the baseline agent's training curves were slower and more staggered, indicating that it relied more on luck to discover extrinsic rewards. By 30 million frames, one of the baseline runs remained stuck on zero score. Results in *Freeway* were similar, with the baseline exhibiting staggered starts while the pellet agent progressed so reliably that its five runs are virtually indistinguishable in Figure 1b.

On *Montezuma's Revenge*, none of the baseline runs made progress within the time given. By contrast, the pellet rewards agent learned to reach the key, open the right-hand door, descend a ladder to reach a sword then climb back up the ladder to kill an enemy with the sword for a total of 2,500 points, achieving this consistently in 4 out of 5 runs by the end of training[4]. Impressively, the agent learned to retrace its steps from the key to reach a door without suiciding. As far as we are aware, the only previous agent to display this behaviour is that of Roderick, Grimm, and Tellex (2018), but it exploited a handcrafted state representation.

One surprising result is the performance of the baseline in *Venture*, as baselines in previous work average less than 100 points in this game (Bellemare et al. 2016a; Martin et al. 2017; Ostrovski et al. 2017). The fact that our baseline began improving well after 4 million frames (i.e. after ϵ had fully decayed) suggests that premature annealing of ϵ was a significant problem for previous agents.

Relation to Previous Novelty-Driven Methods

The previous methods most comparable to ours are those of Bellemare et al. (2016a; 2016b) and Ostrovski et al. (2017). Individual comparisons are provided below, but broadly speaking, these methods differ from ours in three main respects: (1) They do not identify subgoals. (2) They pay an intrinsic reward at every frame, whereas we pay bonuses only on pellet collection. (3) They consider episodes to be terminated after the loss of all lives (which mitigates the "risky exploration" problem discussed in the background), whereas we terminate episodes after individual life loss.

Bellemare et al.'s (2016a; 2016b) agent, which is based on the *CTS density model*, is currently the strongest agent for *Montezuma's Revenge* (averaging 3439 points after 100 million frames). While our agent never scored more than

[4]On the other training run, the agent formed a hard-to-unlearn preference for exiting the first room via the left door, from where it is difficult to score. Bellemare et al.'s (2016a) state-of-the-art agent also experienced this issue; see Figure 2 in their work, noting the minimum score line that persists until around 60 million frames.

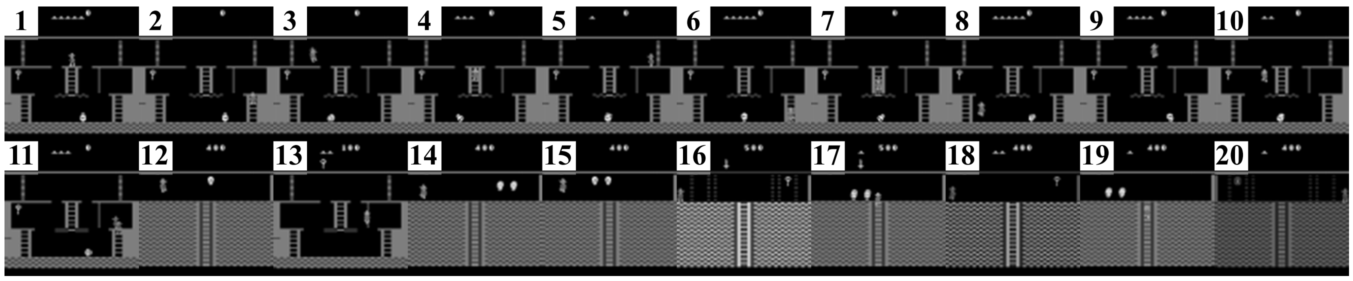


Figure 3: The first 20 representative states found over one training run on *Montezuma's Revenge*.

2500 points, we trained under the harder episode termination condition. Bellemare et al.'s *A3C+* agent averaged only 143 points with this setting[5], suggesting that it was more susceptible to the risky exploration problem. In addition, their strongest agent achieved only ≈350 points in *Venture*, which was far surpassed by even our baseline configuration.

Ostrovski et al.'s (2017) agent, which is the strongest for sparse reward *Atari* games overall, uses a more sophisticated, neural density model in place of CTS. It is difficult to compare our results directly with theirs because they try a number of different configurations and it is unclear which constitutes the "main" agent, but on the three sparse reward games tested here, their results were broadly comparable to ours (≈1000–1200 in *Venture*, ≈30 in *Freeway* and ≈1500–3000 in *Montezuma's Revenge*). However, on the dense reward games we tested (*Battlezone* and *Robot Tank*) their scheme was heavily detrimental, reducing scores by 30–50% over the course of training[6]. For this to have occurred, the cumulative effect of their per-frame intrinsic rewards must have been large enough to distract the agent from the extrinsic rewards. By contrast, it seems that pellet rewards had a milder effect on the task objective, as they caused no noticeable impact in these games (see Figure 2).

Relation to Previous Subgoal-Based Methods

The only previous approach we are aware of that derives subgoals autonomously and reaches a competitive score on *Montezuma's Revenge* is that of Vezhnevets et al. (2017). However, their agent took somewhere in the order of *one billion* frames to match our agent's 30 million frame score. Machado et al.'s (2018b) *eigenoption* method autonomously discovered some useful options on the first screen of *Montezuma's Revenge*, but to the best of our knowledge has not yet yielded a successful game playing agent. Further, the options showcased in their paper were selected from amongst hundreds of derived options. While our method did identify some seemingly redundant subgoals (e.g. see subgoals #9 and #10 in Figure 3), it was more selective than the eigenoption method overall.

[5]They do not report results for their non-AC3+ agent with termination after life loss, but it was also harmed by this setting.

[6]See Figure 17 in their appendix, noting that the agents with mixed Monte Carlo returns are most comparable to ours.

Relation to Novelty-Driven Planning

Beyond Deep RL, the idea of favouring novel states has also been applied in *Atari* planning agents (Lipovetzky, Ramirez, and Geffner 2015). However, the strongest *Atari* planning agents rely on the provision of an exact model, and we are not aware of any planning agents that attain competitive scores on games such as *Montezuma's Revenge* and *Venture* whilst also operating in real-time.

Future Work

In the future, we believe it will be valuable to incorporate uncertainty estimates into the exploration effort function, perhaps leveraging recent work by Gal (2016) and others. Our reason is that, ideally, the approach should account for changes to the state distribution as the agent learns. When the agent first experiences states with large visual novelty (e.g. when it first reaches beyond the first room in *Montezuma's Revenge*), the EE estimates are liable to be inaccurate. On one hand this is not a major problem, since inaccurate EE values will usually result in large distance estimates (due to the way Equations 5 and 6 are structured), meaning the agent will be inclined to place subgoals in such areas. Intuitively, this should be beneficial. However, to ensure the reliability of the distance estimates, it would be preferable for Equation 6 to ignore values with large uncertainty.

Conclusion

In this paper, we proposed a new approach to the important problem of identifying subgoals autonomously in high-dimensional state spaces. Our experiments in the *Atari* domain showed that our method was capable of identifying meaningful subgoals from raw pixels. We proposed a novel intrinsic reward scheme for exploiting the subgoals and used it to train an agent that was competitive with state-of-the-art methods on three sparse reward games. In addition, our approach mitigated the "risky exploration" problem, and performed gracefully on two games where a previous intrinsic reward scheme was detrimental.

Algorithm Pseudocode

A sketch of our full approach is provided in Algorithm 1. For comprehensive implementation details and a complete list of parameter settings, please refer to our source code (https://bitbucket.org/mchldann/aaai2019).

Algorithm 1 Q-learning with Pellet Rewards

```
**var:** current set of rep. states, $\mathcal{R} = \{s_{p_1}, s_{p_2}, \ldots, s_{p_n}\}$
**var:** time elapsed since last partition added, $t_{\text{partition}}$
**var:** time gap between partition additions, $T_{\text{add}}$
**var:** current candidate for the next rep. state, $\tilde{s}_{p_{n+1}}$
**var:** max dist. veered since last partition addition, $D_{\text{max}}$
**var:** the set of partitions visited in the episode so far, $v$
**var:** Monte Carlo mixing coefficient for Q-learning, $\eta_Q$
**var:** Monte Carlo mixing coefficient for EE, $\eta_E$

**procedure** MAINLOOP()

    RESET()

    // Add representative state for first partition
    $s_{p_1} \leftarrow s$
    $\mathcal{R} \leftarrow \{s_{p_1}\}$
    $t_{\text{partition}} \leftarrow 0$

    **for** each episode **do**

        **while** $s$ is not terminal **do**

            $\pi \leftarrow \epsilon$-greedy policy derived from Q
            Select $a \sim \pi(s, \cdot)$

            Calculate aux. reward $\hat{r}^{\pi}$ as per Eq. 3
            Take action $a$, observe $r$, $s'$

            // Determine the current partition
            $s_{p_c} \leftarrow \text{argmin}_{s_{p_i} \in \mathcal{R}}\, d(s', s_{p_i})$

            // Update the set of visited partitions
            $v' \leftarrow v \cup s_{p_c}$

            // Update the best candidate according
            to the
            // distance measure defined by Equation 6
            **if** $d(s', s_{p_c}) > D_{\text{max}}$ **then**
                $\tilde{s}_{p_{n+1}} \leftarrow s'$
                $D_{\text{max}} \leftarrow d(s', s_{p_c})$
            **end if**

            Store transition info $\{s, v, a, \hat{r}^{\pi}, r, s', v'\}$
            in the replay memory

            // Add a new rep. state every $T_{\text{add}}$ steps
            $t_{\text{partition}} \leftarrow t_{\text{partition}} + 1$
            **if** $t_{\text{partition}} > T_{\text{add}}$ **then**
                $\mathcal{R}.add(\tilde{s}_{p_{n+1}})$
                $D_{\text{max}} \leftarrow 0$
                $t_{\text{partition}} \leftarrow 0$
            **end if**

            QLEARN()
            EELEARN()
            $s \leftarrow s'$
            $v \leftarrow v'$
        **end while**

        Update all partitions' visit counts based on $v$
        RESET()

    **end for**
**end procedure**


**procedure** RESET()
    Reset the game and set $s$ equal to the initial state
    $v \leftarrow \{\}$
**end procedure**


**procedure** QLEARN()

    Sample random minibatch of transitions
    $\{s, v, a, r, s', v'\}$ from replay memory

    **if** $v \neq v'$ **then**
        $r^{+} \leftarrow$ pellet reward for the partition visited
        (i.e. the single partition in $v' \setminus v$)
    **else**
        $r^{+} \leftarrow 0$
    **end if**
    $targ_{\text{one-step}} \leftarrow r + r^{+} + \gamma \max_a Q(s', v', a)$

    Calculate extrinsic and intrinsic returns, $R$ and $R^{+}$,
    via the remaining history in the replay memory

    $targ_{\text{MC}} \leftarrow R + R^{+}$
    $targ_{\text{mixed}} \leftarrow (1 - \eta_Q) targ_{\text{one-step}} + \eta_Q targ_{\text{MC}}$
    Update $Q(s, v, a)$ towards $targ_{\text{mixed}}$

**end procedure**


**procedure** EELEARN()

    Sample a minibatch of state pairs and interleaving
    auxiliary rewards $\{s_t, s_{t+k}, \{\hat{r}^{\pi}_t, \ldots, \hat{r}^{\pi}_{t+k-1}\}\}$
     from the replay memory with $k < m$

     $targ_{\text{one-step}} \leftarrow \hat{r}^{\pi}_t + \gamma \mathcal{E}^{\pi}_m(s_{t+1}, s_{t+k-1})$

     $targ_{\text{MC}} \leftarrow \sum_{i=0}^{k-1} \gamma^i \hat{r}^{\pi}_{t+i}$

     $targ_{\text{mixed}} \leftarrow (1 - \eta_E) targ_{\text{one-step}} + \eta_E targ_{\text{MC}}$

     Update $\mathcal{E}^{\pi}_m(s_t, s_{t+k-1})$ towards $targ_{\text{mixed}}$

 **end procedure**
```

References

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research.*

Bellemare, M.; Srinivasan, S.; Ostrovski, G.; Schaul, T.; Saxton, D.; and Munos, R. 2016a. Unifying Count-Based Exploration and Intrinsic Motivation. In *Advances in Neural Information Processing Systems*, 1471–1479.

Bellemare, M. G.; Srinivasan, S.; Ostrovski, G.; Schaul, T.; Saxton, D.; and Munos, R. 2016b. Unifying Count-Based Exploration and Intrinsic Motivation. *arXiv preprint arXiv:1606.01868.*

Bromley, J.; Guyon, I.; LeCun, Y.; Säckinger, E.; and Shah, R. 1994. Signature Verification using a "Siamese" Time Delay Neural Network. In *Advances in Neural Information Processing Systems*, 737–744.

Dietterich, T. G. 2000. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research (JAIR)* 13:227–303.

Digney, B. L. 1998. Learning Hierarchical Control Structures for Multiple Tasks and Changing Environments. In *Proceedings of the 5th International Conference on Simulation of Adaptive Behavior*, volume 5, 321–330.

Gal, Y. 2016. *Uncertainty in Deep Learning*. Ph.D. Dissertation, University of Cambridge.

Hengst, B. 2002. Discovering Hierarchy in Reinforcement Learning with HEXQ. In *Proceedings of the 19th International Conference on Machine Learning*, volume 2, 243–250.

Konidaris, G., and Barto, A. 2009. Skill Discovery in Continuous Reinforcement Learning Domains Using Skill Chaining. In *Advances in Neural Information Processing Systems*, 1015–1023.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. In *Advances in Neural Information Processing Systems*, 3675–3683.

Lipovetzky, N.; Ramirez, M.; and Geffner, H. 2015. Classical Planning with Simulators: Results on the Atari Video Games. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 15, 1610–1616.

Machado, M. C.; Bellemare, M. G.; and Bowling, M. 2017. A Laplacian Framework for Option Discovery in Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, 2295–2304.

Machado, M. C.; Bellemare, M. G.; Talvitie, E.; Veness, J.; Hausknecht, M.; and Bowling, M. 2018a. Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents. *Journal of Artificial Intelligence Research.*

Machado, M. C.; Rosenbaum, C.; Guo, X.; Liu, M.; Tesauro, G.; and Campbell, M. 2018b. Eigenoption Discovery through the Deep Successor Representation. In *International Conference on Learning Representations.*

Machado, M. C.; Srinivasan, S.; and Bowling, M. H. 2015. Domain-Independent Optimistic Initialization for Reinforcement Learning. In *AAAI Workshop: Learning for General Competency in Video Games.*

Martin, J.; Sasikumar, S. N.; Everitt, T.; and Hutter, M. 2017. Count-Based Exploration in Feature Space for Reinforcement Learning. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI).*

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-Level Control Through Deep Reinforcement Learning. *Nature* 518(7540):529–533.

Osband, I.; Blundell, C.; Pritzel, A.; and Van Roy, B. 2016. Deep Exploration via Bootstrapped DQN. In *Advances in Neural Information Processing Systems*, 4026–4034.

Ostrovski, G.; Bellemare, M. G.; van den Oord, A.; and Munos, R. 2017. Count-Based Exploration with Neural Density Models. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, 2721–2730.

Parr, R., and Russell, S. 1998. Reinforcement Learning with Hierarchies of Machines. *Advances in Neural Information Processing Systems* 1043–1049.

Roderick, M.; Grimm, C.; and Tellex, S. 2018. Deep Abstract Q-Networks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 131–138. International Foundation for Autonomous Agents and Multiagent Systems.

Şimşek, Ö.; Wolfe, A. P.; and Barto, A. G. 2005. Identifying Useful Subgoals in Reinforcement Learning by Local Graph Partitioning. In *Proceedings of the 22nd International Conference on Machine Learning*, 816–823. ACM.

Stadie, B. C.; Levine, S.; and Abbeel, P. 2015. Incentivizing Exploration in Reinforcement Learning with Deep Predictive Models. *arXiv preprint arXiv:1507.00814.*

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence* 112(1):181–211.

Thrun, S.; Schwartz, A.; et al. 1995. Finding Structure in Reinforcement Learning. *Advances in Neural Information Processing Systems* 385–392.

Vezhnevets, A. S.; Osindero, S.; Schaul, T.; Heess, N.; Jaderberg, M.; Silver, D.; and Kavukcuoglu, K. 2017. FeUdal Networks for Hierarchical Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, 3540–3549.

Wawrzynski, P. 2015. Control Policy with Autocorrelated Noise in Reinforcement Learning for Robotics. *International Journal of Machine Learning and Computing* 5(2):91.

Zahavy, T.; Ben-Zrihem, N.; and Mannor, S. 2016. Graying the black box: Understanding DQNs. In *Proceedings of the 33rd International Conference on Machine Learning (ICML-16)*, 1899–1908.