EVENT DISCOVERY FOR HISTORY REPRESENTATION IN REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Environments in Reinforcement Learning (RL) are usually only partially observable. To address this problem, a possible solution is to provide the agent with information about past observations. While common methods represent this history using a Recurrent Neural Network (RNN), in this paper we propose an alternative representation which is based on the record of the past events observed in a given episode. Inspired by the human memory, these events describe only important changes in the environment and, in our approach, are automatically discovered using self-supervision. We evaluate our history representation method using two challenging RL benchmarks: some games of the Atari-57 suite and the 3D environment Obstacle Tower. Using these benchmarks we show the advantage of our solution with respect to common RNN-based approaches.

1 INTRODUCTION

Deep Reinforcement Learning (RL) algorithms have been successfully applied to a range of challenging domains, from computer games (Mnih et al., 2013) to robotic control (OpenAI et al., 2018). These approaches use a Neural Network (NN) to both represent the current observation of the environment and to learn the agent's optimal policy, used to choose the next action. For instance, the state observation can be the current game frame or an image of the robot camera, and a Convolutional Neural Network (CNN) may be used to obtain a compact feature vector from it.

However, often RL environments are only partially observable and having a significant representation of the past may be crucial for the agent (Kapturowski et al., 2019). For instance, in the Atari 2600 Pong game, the state can be represented as two consecutive frames. In this way, the agent can determine both the position of the ball and the direction of its movement. More complex partially observable domains require a longer state history input to the agent. For instance, when navigating inside a 1-st person-view 3D labyrinth, the agent obtains little information from the current scene observation and needs several previous frames to localise itself.

A common solution to represent the observation history is based on the use of Recurrent Neural Networks (RNNs), where the RNN hidden-layer activation vector is input to the agent, possibly together with the current state observation (Mnih et al., 2016). However, RL is characterized by highly nonstationary data, which makes training unstable (Schulman et al., 2016) and this instability is exacerbated when a recurrent network needs to be simultaneously trained to extract the agent's input representation. We show in Sec. 5.2 that, *with a limited training budget*, an RNN-based history representation struggles to improve the agent's results over an input representation composed of only the instantaneous observations. In this paper we propose a different direction, in which the agent's observation history is represented using a set of discrete *events*, which describe important changes in the state of the world (Orr et al., 2018). Environment-specific events are automatically discovered during training by clustering past observations and are then used as landmarks in our history representation. Intuitively, this is inspired by the common human behaviour: when making decisions, humans do not keep detailed visual information of the previous steps. For instance, while navigating through the halls of a building, it is sufficient to recall a few significant landmarks seen during the walk, e.g. specific doors or furniture.

Following this idea, in this paper we propose an Event Discovery History Representation (EDHR) approach, composed of 3 iterative stages: *experience collection, policy optimisation* and *event discovery*. We discover events by clustering past state observations. In more detail, we maximize the

Mutual Information (MI) between the latent representations of temporally-close frames in order to cluster the frame semantics. This is a form of *self-supervision*, in which no additional annotation is needed for the event discovery: the higher the *predictability* of one frame with respect to the other, the larger the semantic information shared by the two frames (van den Oord et al., 2018; Hjelm et al., 2019; Anand et al., 2019; Ji et al., 2018).

Once clusters have been formed, the probability distribution of a given frame F_t with respect to the set of current clusters is used as the semantic representation of the state observation at time t and is recorded in a longer *history*. Finally, the history is input to the agent together with an *instantaneous* observation representation, obtained, following (Mnih et al., 2013), as a stack of the last 4 frames. This information is now used for policy optimisation. Note that our proposed history representation is independent of the specific RL approach. However, in all our experiments we use the PPO algorithm (Schulman et al., 2017) for the policy and the value function optimisation. The 3 stages are iterated during training, and thus past clusters can be modified and adapted to address new observations, while the agent is progressing through the environment.

We summarize the contribution of this paper below. First, we use a modified version of the Invariant Information Clustering (IIC) algorithm (Ji et al., 2018) in order to discover significant events in the agent's past observations. Second, we propose to replace common RNN-based history representations with a time-dependent probability distribution of the observations with respect to the events so far discovered. We evaluate our history representation (EDHR) using the PPO algorithm (Schulman et al., 2017) on several environments of the Atari-57 benchmark (Bellemare et al., 2012) and on the 3D environment Obstacle Tower (Juliani et al., 2019), showing that it provides a significant boost with respect to the most common input representation methods. Specifically, we show that EDHR outperforms RNN-based methods when both are trained with the same budget of interactions with the environment (steps), while simultaneously requiring $2\times$ less wall-clock time for training.

The source code of the method and all the experiments are publicly available at the anonymous link: https://github.com/iclr2020anon/EDHR and will be published after acceptance.

2 Related Work

Partially Observable Markov Decision Processes. We consider a setting in which an agent interacts with an environment in order to maximize a cumulative reward signal. The interaction process is defined as a Partially Observable Markov Decision Process (POMDP) $< S, A, T, R, \Omega, O, \gamma >$ (Monahan, 1982; Kae, 1998), where S is a finite state space, A is a finite action space, $R(s, a)$ is the reward function, $T(\cdot|s, a)$ is the transition function, Ω is a set of possible observations, O is a function which maps states to probability distributions over observations and $\gamma \in [0, 1)$ is a *discount factor* used to compute the cumulative discounted reward. Within this framework, at each step t, the environment is in some unobserved state $s \in S$. When the agent takes an action a, the environment transitates to the new state s' with probability $T(s'|s, a)$, providing the agent with a new observation $o' \in \Omega$ with probability $O(o'|s')$ and reward $r \sim R(s, a)$.

POMDPs are an active research area and several approaches have been recently proposed to address partial observability. DQN (Mnih et al., 2013) is one of the earliest deep RL models directly trained using the high-dimensional observation o_t as input (i.e., the raw pixels of the current frame). To deal with partial observability, the input of the model is composed of a stack of the 4 last grayscale frames, processed as 4 channels using a 2D CNN. We also use this observation representation which we call *instantaneous* to highlight the difference with respect to the past state history.

Hausknecht & Stone (2015) replace the fully-connected layer in DQN with an LSTM (Hochreiter & Schmidhuber, 1997) layer. At each step, the model receives only one frame of the environment together with the LSTM hidden-layer value which represents the past. To simulate a reduced observability and demonstrate the robustness of the method, the authors introduce the flickering Atari games, obscuring the observation with a certain probability at each step.

Recurrent layers (e.g., LSTM (Hochreiter & Schmidhuber, 1997) or GRU (Cho et al., 2014)), are a common choice for modern RL architectures. The A3C algorithm (Mnih et al., 2016) showed a high mean performance with an LSTM layer. R2D2 (Kapturowski et al., 2019) is specifically focused on processing long-history sequences (80 steps with additional 40 burn-in steps) using an LSTM. However, despite its widespread, RNNs have disadvantages within an RL framework because they

increase the training instability (Sec. 1). In this paper we propose an alternative history representation and we empirically show in Sec. 5.2 that, we the same training budget, our proposal largely outperforms RNN-based solutions.

Unsupervised and Self-supervised Learning. Unsupervised and self-supervised learning aim to extract useful representations from data (e.g., images, videos, etc.) without annotation and reward signal. While common approaches are covered in recent surveys (Jing & Tian, 2019; Kolesnikov et al., 2019), below we analyse the most relevant methods to this work.

A first class of unsupervised feature-extraction methods is based on a generative encoding-decoding scheme. The methods typically minimize the reconstruction error in the pixel space, thus the representation is encouraged to capture pixel-level details, which may differ from semantics. Ha & Schmidhuber (2018) combine Variational Autoencoders (Kingma & Welling, 2014) (VAEs) with RNNs in their *World Models*. A VAE is trained to reconstruct the current frame and simultaneously produce a latent, compressed frame representation which is input to an RNN predicting the next latent feature vector. The input of the RL agent consists of the latent vector of the current observation and the hidden state of the RNN.

An alternative to a reconstruction loss in the input space, is clustering in the feature space. DeepCluster (Caron et al., 2018) iteratively alternates between k-means clustering of the CNN features and updating the network's parameters with the pseudo-labels obtained by means of clustering.

Another important self-supervised paradigm is based on mazimizing the Mutual Information (MI) between latent representations. Typically these methods rely on collecting pairs of *positive* and *negative* samples. Two positive samples should share the same semantics, while negatives should be perceptually different. A discriminator is trained to distinguish between positive and negative pairs, so computing a lower-bound on the MI of the learned latent representations. In (van den Oord et al., 2018; Anand et al., 2019) positives are temporally close video frames, while in (Ji et al., 2018) simple data-augmentation techniques are applied on still images. In (Hjelm et al., 2019) this idea is extended including *locality*, and MI is computed also between global and local representations of the same image. (Hjelm et al., 2019) was applied to several games of the Atari 2600 suite (Bellemare et al., 2012) in (Anand et al., 2019), however the reported evaluation is based only the disentanglement degree of the obtained representations and does not consider the possible benefit for an RL task.

Our event discovery stage (Sec. 3) is based on IIC (Ji et al., 2018), but we use as positives temporally-close frames extracted from an agent trajectory instead of data-augmentation. Note that the main difference between our proposal and IIC is that our goal is not clustering, but clusters are used as a set of discrete events to represent the agent's history, in contrast to RNN and other continuous history representations.

3 Event Discovery and History Representation

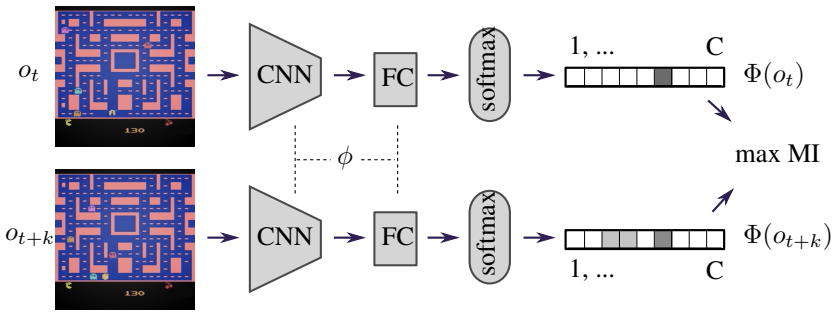

Figure 1: Event discovery. This figure shows two replicas of our *unsupervised encoder* architecture (Φ) which separately map two different observations in corresponding event probability distributions. The latter are used to maximize the MI over the latent assignment variable z, in this way obtaining a set of events Y and the supervision signal used to update the parameters (ϕ) of Φ.

Event discovery stage. As mentioned in Sec. 1-2, we use IIC (Ji et al., 2018) for our unsupervised event discovery. While we refer the reader to the original paper for the details, we show below how we use this clustering method to obtain a set of discrete events and simultaneously learn to encode our observations.

Let o_t be the observation of the environment at time step t, presented as an image. We want to learn a representation mapping $\Phi : \Omega \rightarrow [0,1]^C$, where $Y = \{1, ..., C\}$ is a set of C events and $\Phi(o_t)$ is the probability distribution of a discrete random variable z which (soft-)assigns o_t to the elements in Y and can be defined as: $\Phi_c(o_t) = P(z = c|o_t)$ (Ji et al., 2018). $\Phi(\cdot)$ is our *unsupervised encoder* and it is based on a CNN followed by a fully-connected layer and a softmax layer (Fig.. 1), parametrized by ϕ. The clustering process in IIC is based on pairs of images randomly transformed which contain a similar semantics. In our case we use two observations extracted from the same trajectory: o_t and o_{t+k}, where $k \sim U[1, L]$. L defines a temporal translation window in which observations are assumed to mostly like have the same semantic content. Small values of L (e.g. $L = 1$) reduces the uncertainty and thus improves the stability during training. On the other hand, larger values of L produce a higher intra-cluster variability, in which the clustering process is forced to extract a more abstract semantics in order to group similar frames. This parameter can vary for different RL environments, in our experiments we use $L = 3$.

Following (Ji et al., 2018), clustering is performed by computing a $C \times C$ matrix whose elements represent the joint probability $P(z = c, z' = c'|o_t, o_{t+k})$ and then maximizing the MI directly using this matrix.

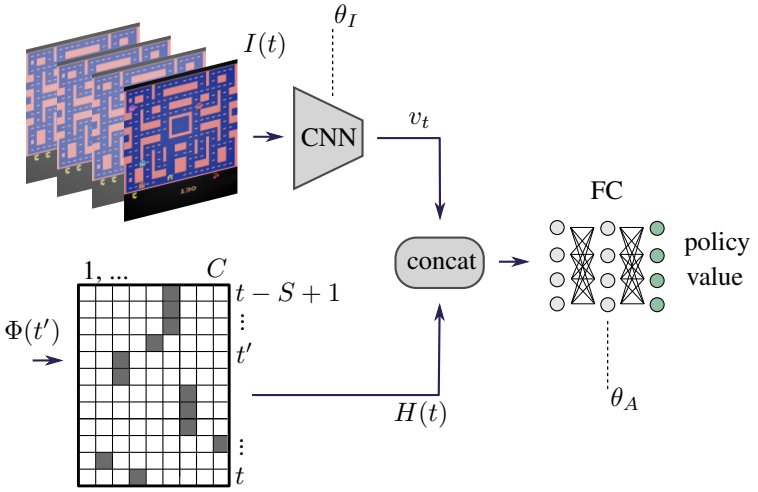

Figure 2: The full agent model and our history representation.

History Representation and Policy Optimisation. Once ϕ is updated (see Sec. 4), we use $\Phi(o_t; \phi)$ to represent an observation at time t and update the agent's policy. In more detail, we represent the agent with an Actor-Critic architecture approximated with a NN π parametrised by θ. We use the PPO algorithm (Schulman et al., 2017) to optimise the policy and the value function. At time step t, the agent receives two types of data as input, an *instantaneous* representation $I(t)$ consisting of 4 concatenated raw-pixel, grayscale frames $I(t) = \{o_t, o_{t-1}, o_{t-2}, o_{t-3}\}$ and an *history* $H(t)$ representing past events. Specifically, for each $t' \in \{t, ..., t - S + 1\}$ let $\Phi(o_{t'}; \phi)$ be the probability distribution of $o_{t'}$ with respect to the current set of events Y. Our proposed history representation is simply given by the concatenation of these cluster soft-assignment vectors, computed over the past S steps: $H(t) = \{\Phi(o_t; \phi), ..., \Phi(o_{t-S+1}; \phi)\}$, arranged as a $S \times C$ matrix.

We empirically observed that the history matrix $H(t)$ is very sparse (Sec.), being most of the elements in $\Phi(o_t)$ close to 0 ($\Phi(o_t)$ is usually very peaked). Thus, an even more compact history representation may be produced. Anyway, with a small value of S ($S = 32$ in all our experiments) and a small number of clusters C (we use $C = 16$), $H(t)$ is a low-dimensional, sparse matrix which represents significant information about the recent agent's past trajectory. Importantly, in contrast with an RNN-based history representation, which may be prone to forgetting past information, in

Algorithm 1 Training

1 Initialize ϕ and θ.
2 For $iteration = 1, ...N$ do:
3 For $actor = 1, 2, ...$ do:
4 For $t = 1, ..., r$ do:
Observation collection stage:
5 Run $\pi_\theta(I(t), H(t))$ in the environment and get o_{t+1}
6 Use $\Phi(o_{t+1}; \phi)$ to compute $H(t+1)$.
7 Collect $I(t+1)$, $H(t+1)$ and other PPO-specific information in a buffer B.
Policy optimisation stage:
8 Use B to update θ_I and θ_A in n_P epochs
Event discovery stage:
9 Use B to update ϕ in n_E epochs

our proposed history representation $H(t)$, all the past events are stored, together with their time-dependent evolution.

$I(t)$ is input to a second CNN (Ψ) which extracts a vector v_t representing the instantaneous visual information (v_t is obtained by flattening the last CNN feature map). We call the parameters of Ψ θ_I, to differentiate them from the agent's proper parameters θ_A which control the policy and the value function ($\theta = [\theta_I; \theta_A]$). Finally, v_t is concatenated with the (flattened) matrix $H(t)$ and input to the agent. Fig. 2 shows the full architecture of our agent. We highlight that the CNN of the unsupervised encoder (Φ) is different from Ψ, and the two CNNs do not share parameters ($\phi \neq \theta_I$). Moreover, while the encoder is trained in an unsupervised fashion, Ψ, used to extract visual information (v_t) from $I(t)$, is trained *using the RL reward*. This emphasizes the complementary between the knowledge of the two networks, being the former based on clustering visual similarities in data which do not depend on the specific RL task, while the latter being trained as in a standard RL framework.

4 TRAINING

We run the agent in several parallel environments for a predefined number of steps (rollout size r). The actions are chosen according to the current policy and encoder (*observation collection stage*). Next, using the collected experience, we update θ for n_P epochs (*policy optimisation stage*) and ϕ for n_E epochs (*event discovery stage*). The second and the third stages (described in Sec. 3) are each other independent and can be performed in parallel, which removes the event discovery runtime overhead. These three stages are iterated, so to adapt $\Phi(\cdot)$ (and Y) to the observation distribution change. The full algorithm is described in Alg. 1.

4.1 IMPLEMENTATION DETAILS

The weights ϕ and θ are initialized with a (semi-) orthogonal matrix, as described in (Saxe et al., 2013), apart from the head of Φ (i.e., the fully-connected layer on top of the CNN, see Fig. 1), which is initialized randomly. All the biases are initialized with zeros.

Following (Ji et al., 2018), to improve robustness, the head of Φ is duplicated 5 times with different initial random parameters. Moreover, Φ includes an auxiliary overclustering head with $C \times 5$ clusters. Each head produces a different representation with a potentially different degree of utility for the agent. A possible solution is the simple concatenation of the 5 different event probability distributions $\Phi_1(o), ...\Phi_5(o)$. However, this solution would increase the dimensionality of $H(t)$. We instead select the "best" head using the MI value as the selection criterion as follows. We split training in two phases. In the first phase ($N * p_{pretrain}$ iterations of the outmost loop in Alg. 1) the agent receives only $I(t)$ as input and θ and ϕ (and each head in Φ) are updated as usual (lines 8 and 9 in Alg. 1). Then, the head corresponding to the highest MI value (averaged with respect to the last iterations) is selected and in the second phase the agent receives the full input ($H(t)$ included).

In all our experiments we use the same set of hyperparameters and the same network architecture. We adopt the training procedure and the hyperparameters in (Schulman et al., 2017), including the

number of time steps equal to 10M. Our agent network architecture is similar to (Schulman et al., 2017), except from the additional history input ($H(t)$). Φ is composed of a CNN and a fully-connected layer (Sec. 3). Specifically, its CNN configuration is similar to Ψ, except the first layer, which takes only one channel as input. In Appendix A we list the all other hyperparameters.

5 EXPERIMENTAL EVALUATION

We evaluate EDHR on two challenging deep RL benchmarks: the Arcade Learning Environment (Bellemare et al., 2012) (ALE) and the Obstacle Tower (Juliani et al., 2019). Moreover, we compare the results of EDHR with PPO (Schulman et al., 2017) and PPO with RNN, where the latter is described below.

PPO with RNN. Ψ is the same as in EDHR and in PPO (Schulman et al., 2017). However, the past observations are represented as follows. The encoder Φ is replaced with a similar network (Υ) but without the softmax layer and having 256 neurons as output of the fully-connected layer instead of C. For each $t' \in \{t, ..., t-S+1\}$, $\omega_t = \Upsilon(o_{t'})$ is input to a GRU (Cho et al., 2014) with an hidden layer of size 512 (re-initiated with zeros each S steps). Finally, the hidden layer activation vector represents the history and replaces $H(t)$ in Fig. 2, keeping all the rest of the agent unchanged. Note that no self-supervision is used here to train the network components.

5.1 ENVIRONMENTS

In ALE we use 7 classic Atari 2600 video games: MsPacman, SpaceInvaders, Breakout, Gravitar, Qbert, Seaquest and Enduro. In all these environments, o_t is an RGB image. The action space A is discrete and the number of actions varies in each game from 4 to 18. Following closely the experimental protocol in (Schulman et al., 2017), we apply the same observation pre-processing (frame downscaling, etc.).

Obstacle Tower is a 3D 3-rd person game environment. It consists of a number of floors, each floor being composed of several rooms. The layout is procedurally generated based on an environment seed. The rooms can contain puzzles to solve, the complexity of which increases with the floor number, creating a natural curriculum learning for the agent. The environment has a high visual fidelity, realistic lighting, a variety of textures and geometry, physics-driven interactions. Possible actions include movements in four directions, camera rotations and jumps. The agent receives a reward of +1.0 after passing each floor and an auxiliary reward of +0.1 after solving an intermediate puzzle. An observation o_t is an 84×84 RGB image, converted to grayscale. The action space is discrete, with 54 actions, which are reduced to 12 after removing redundancy. This environment is especially challenging because of several reasons. Unlike the Atari games with simple sprite-based graphics, this environment has diverse and realistic observations. The sparse reward signal complicates training and favours a low-dimensional agent's input. The agent is trained using seeds 0-7 and it is evaluated with seeds 50-57, which shows the generalisation abilities of the algorithm (see the *Train* and *Test* rows in Tab. 2, respectively).

5.2 RESULTS

In all the experiments we report the final scores averaged over 100 episodes. Tab. 1-2 show the results. "PPO (Sch.)" refers to the results reported in (Schulman et al., 2017), while the results "PPO" are based on our reproduction of that algorithm. We include our implementation in the comparison to remove the impact of minor technical implementation details.

These results show the benefit of our history representation, which is able to significantly boost the scores in most of the games, although more reactive games, like Breakout, do not show any benefit. In Tab. 1 we also report the results of PPO with RNN (see above), which drastically underperforms our EDHR. Note that, in most of the games, PPO with RNN have a similar or even *worse* performance of plain PPO. We belive this is due to the difficulty of training an RNN simultaneously with the agent in an RL framework when the training budget is limited. Specifically, the training budget, adopted in all our experiments and uniformly with all the tested methods, consists of 10M of time steps (Sec. 4.1) which is equivalent to the number of observations o_t an agent receives from the environment, considering all the parallel actors and all the iterations. Moreover, if we consider the

Table 1: Scores on the ALE benchmark

	PPO (Sch.)	PPO	PPO with RNN	EDHR (Ours)
MsPacman	2096.5	2039.8	1956.1	**2575.8**
SpaceInvaders	942.5	916.1	923.1	**1248.9**
Breakout	274.8	**438.6**	410.6	410.2
Gravitar	737.2	604.5	**736.5**	636.5
Qbert	14293.3	10689	11202	**13718.8**
Seaquest	1204.5	950.8	1173.8	**1713**
Enduro	758.3	752.15	751.4	**787**

Table 2: Scores on the Obstacle Tower benchmark

	PPO	EDHR
Train	1.57	**2.35**
Test	1.08	**1.35**

wall-clock time (and with the same training budget), in all our experiments PPO with RNN is on average about two times *slower* then our EDHR.

The higher efficiency of our EDHR with respect to the training budget is also shown Fig. 3, where, for each game, we plot the scores reached by each method with respect to the training iterations. These plots show that EDHR effectively makes use of the knowledge acquired in the unsupervised training stages and that, using an event-based history representation, most of the times training is easier. We believe this is due to the fact that the agent, in making a decision, can use the events, recorded in $H(t)$, as a sort of "summary" of the important information in a past trajectory.

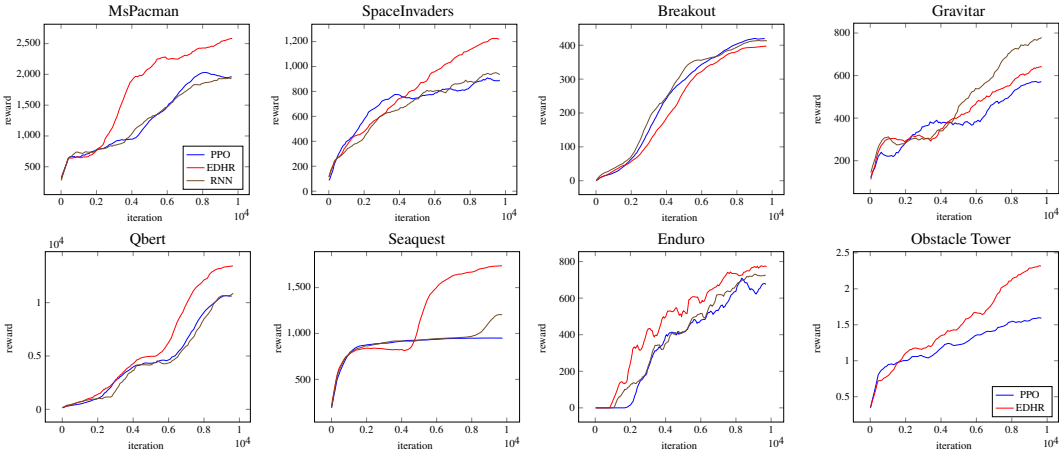

Figure 3: Training dynamics, smoothed for readability

5.3 Discovered Event Analysis

What type of events does the method discover? To answer this question, we visually examined the observations and the corresponding events of a trained agent in two environments, MsPacman and Obstacle Tower. Fig. 4 shows how the events are distributed during an episode of MsPacman. Specifically, for each time step t, we plot the predominant event c_t^* corresponding to:

$$c_t^* = \arg\max_{c \in Y} \Phi_c(o_t) = \arg\max_{c \in Y} P(z = c | o_t) \tag{1}$$

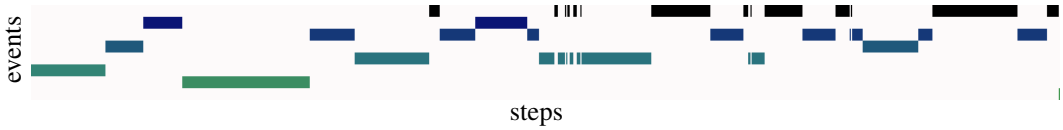

Figure 4: The event dynamics during an episode of MsPacman. Each color corresponds to a different predominant event computed using Eq. 1. For simplicity, the y-axis of the figure contains only a subset of the $C = 16$ events.

This figure shows that the predominant event tend to be constant for a while during an episode, which is coherent with the expectation that important information about the past (an "event") changes with a low frequency. Moreover, we observed that these events correspond to specific changes of the environment. For instance, when the agent collects an object called "Power Pellet", which produces a switching of the color of the ghosts to blue and it allows these ghosts to be eaten by the agent, then in that moment there is a predominant event change in $H(t)$.

Figure 5 shows some observations and the corresponding predominant events extracted from an episode of the Obstacle Tower environment. The observations have a sequential order (but they are not each other time-adjacent). New events correspond to the transition to a new room, or to different viewpoints of the same room. As shown in this figure, in the middle of the trajectory, the agent observes a green door, and when it later returns to it, the event changes back to the corresponding value. In Appendix B we plot another episode of the same environment, with high-frequency selected observations and the corresponding predominant events, which more clearly shows the semantics behind the clusters discovered by the encoder and used by the agent.

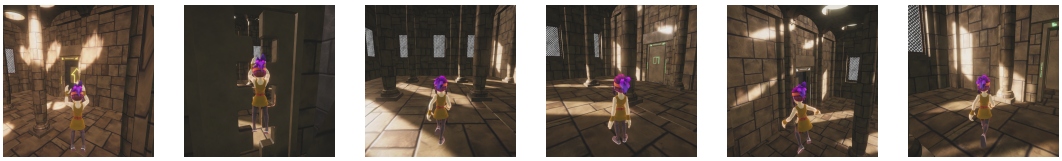

Figure 5: Obstacle Tower predominant events associated with sequentially ordered observations of an episode.

Sparsity of our history representation. Using Obstacle Tower we computed the probability that, in a trained agent, a random element in the $S \times C$ matrix $H(t)$ has a value q such that $0.1 < q < 0.9$. This probability is 0.00146. Using the range of values $0.01 < q < 0.99$, the probability is 0.0031. These empirical results show that our representation is very sparse and "binary", meaning that all the information is concentrated around the values 0 and 1.

6 Conclusion

We presented a method for history representation in RL which is an alternative to common solutions based on RNNs. Our EDHR is based on the idea that important information about the past can be "compressed" using events. Specifically, these events are automatically discovered using a modification of the ICC clustering method (Ji et al., 2018), which is performed jointly with the agent training and iterated through time, to adapt the discovered events to the new observations. In EDHR, visual information is represented using two different networks: Φ and Ψ. The latter is trained using a reward signal, so it presumably extracts task-specific information from the observations. On the other hand, the encoder Φ is trained using self-supervision, and thus it focuses on patterns which are repeated in the data stream, potentially leveraging a larger quantity of supervision signal. Although self-supervision has been explored in other RL and non-RL works, this is the first work to show how the discovered information can be exploited in the form of discrete events used for history representation.

Our results, based on the challenging deep RL benchmarks, ALE and Obstacle Tower, show that EDHR can more effectively represent information about the past in comparison with task-oriented representations such as those used in PPO (Schulman et al., 2017) and PPO with RNN.

REFERENCES

Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99 – 134, 1998.

Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Ct, and R Devon Hjelm. Unsupervised state representation learning in atari, 2019.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. 2012.

Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features, 2018.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

David Ha and Jrgen Schmidhuber. World models. 2018. doi: 10.5281/zenodo.1207631.

Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps, 2015.

R. Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Philip Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *ICLR*, 2019.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

Xu Ji, Joo F. Henriques, and Andrea Vedaldi. Invariant information clustering for unsupervised image classification and segmentation, 2018.

Longlong Jing and Yingli Tian. Self-supervised visual feature learning with deep neural networks: A survey, 2019.

Arthur Juliani, Ahmed Khalifa, Vincent-Pierre Berges, Jonathan Harper, Ervin Teng, Hunter Henry, Adam Crespi, Julian Togelius, and Danny Lange. Obstacle tower: A generalization challenge in vision, control, and planning, 2019.

Steven Kapturowski, Georg Ostrovski, Will Dabney, John Quan, and Remi Munos. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=r1lyTjAqYX`.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. Revisiting self-supervised visual representation learning, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

Volodymyr Mnih, Adri Puigdomnech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. 2016.

George E. Monahan. State of the art—a survey of partially observable markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):1–16, January 1982. doi: 10.1287/mnsc.28.1.1. URL `https://doi.org/10.1287/mnsc.28.1.1`.

OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation, 2018.

John Walker Orr, Prasad Tadepalli, and Xiaoli Z. Fern. Event detection with neural networks: A rigorous empirical evaluation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 999–1004, 2018.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks, 2013.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *ICLR*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding, 2018.

Table 3: Hyperparameters

Number of clusters C	16
History size S	32
Number of epochs n_E	10
Batch size used to update ϕ	512
Temporal window L	3
Learning rate	.0005
Pretraing iterations ratio $p_{pretrain}$	15%

A HYPERPARAMETERS

In Tab. 3 we list all our hyperparameters which are different from (Schulman et al., 2017).

B EVENT VISUALIZATION

Figure 6 shows a detailed sequence of observations and the corresponding predominant events from an episode of Obstacle Tower. We observe that some events correspond to semantically general classes and do not focus only on visual appearance. For instance, event 15 indicates when the agent passes from one room to another, despite each transition (room) is visually different. Events 2 and 12 correspond to specific rooms, event 8 detects the door, events 4 and 14 indicates specific views of a room.

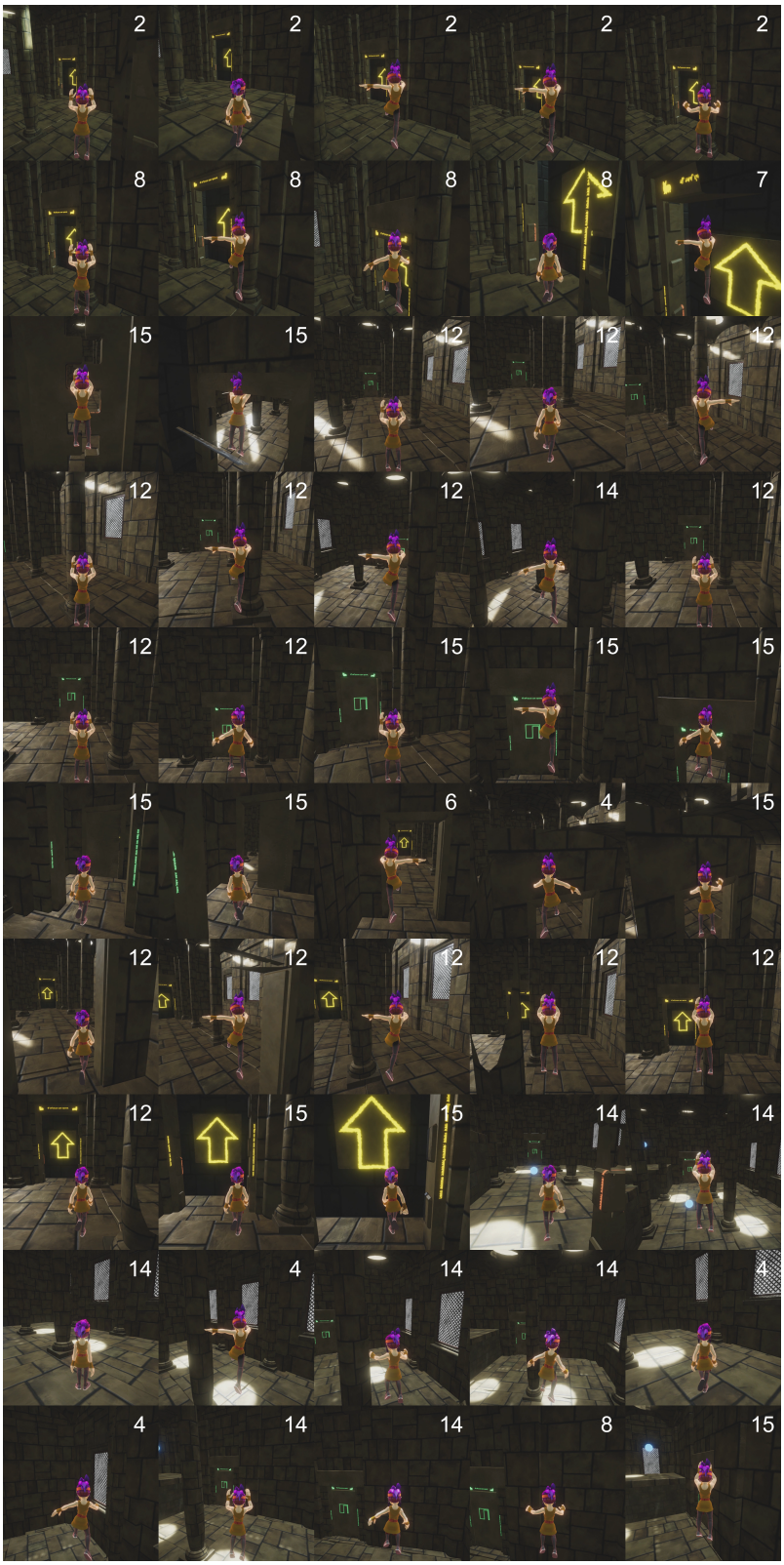

Figure 6: Detailed sequence of Obstacle Tower events