Unsupervised Object-Level Deep Reinforcement Learning

William Agnew **Pedro Domingos**
Paul G. Allen School of Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350, U.S.A.
{wagnew3, pedrod}@cs.washington.edu

Abstract

Deep reinforcement learning has achieved superhuman performance in many challenging environments, but its practicality is limited by the high sample cost of current algorithms. Many have argued that sample-efficient reinforcement learning must be undergirded by significant unsupervised and supervised learning. It has also been noted that current Deep RL algorithms begin learning with very little information about the world, and that by including priors for environments of practical interest we could achieve gains in sample efficiency. In particular, the visual world is composed of objects, for which humans have modelling, exploration, and causality priors that allow them to learn in visual environments many orders of magnitude faster than current deep RL algorithms. We use these observations as inspiration for a new object oriented reinforcement learning framework. Our framework learns a succinct object representation of the environment from pixels and trains models of those objects' behaviors from observations. We then use the prior that important events occur when objects contact to develop a reinforcement learner with object contact-focused exploration, planning, and value and reward estimators. Our system achieves DQN-level performance with up to 10,000x fewer samples, learning faster than even humans.

1 Introduction

Deep reinforcement learning has achieved impressive performance in many environments (Mnih, Kavukcuoglu, et al. 2015; Mnih, Badia, et al. 2016; Silver et al. 2017). However, the sample inefficiency of deep reinforcement learning algorithms limits their applicability to real-world domains. These algorithms are very general, in theory capable of learning in all environments. However, we are only interested in real-world environments, suggesting that current deep reinforcement learning algorithms could be improved by incorporating additional priors or heuristics.

In most real-world environments, humans are able to achieve good performance after much less experience than machines, pointing to the incorporation of human-inspired priors as a means of improving sample efficiency. As extensively documented in psychology, humans perceive the world in terms of objects: groups of percepts that behave similarly across time (Spelke 1990). Humans are then able to use this powerful state representation to model, plan, and formulate goals. In contrast, most deep reinforcement learning algorithms rely on end-to-end trained deep neural networks to choose actions from pixels, forcing the agent to learn approximations of objects, object dynamics, and planning from scratch for each environment.

We incorporate human object priors into a reinforcement learner by developing a compact object-level representation and model of the environment from pixels with no supervision by oversegmenting, modeling segment dynamics, and then combining segments with dynamics that can be described

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

by a single model equally well as separate models into objects. In addition to being compact and powerful, this environment representation allows our agent to explore at an object level, exploring contacts between an agent-controlled object and other objects instead of taking ϵ-random actions. We combine these object priors with a simple linear reinforcement learner and brute-force planner to create an Object-Level Reinforcement Learner (OLRL). We compare this OLRL with humans, deep reinforcement learners, and the current state-of-the-art object feature set, Blob-PROST (Liang et al. 2016) on Atari environments, and show that it yields large improvements in sample and computational efficiency, as well as interpretability. Our object detection and tracking module can be used as a general preprocessor for reinforcement learning in domains like Atari video games, potentially greatly increasing the pace of experiments in this area.

2 Related Work

Deep reinforcement learners are the current state-of-the art for many complex and high-dimensional environments. At a high level, they use an end-to-end neural network policy to choose actions from raw inputs. DQN (Mnih, Kavukcuoglu, et al. 2015) established the efficacy of these methods on Atari 2600 games, and has seen many improvements including A3C (Mnih, Badia, et al. 2016), Double Q-Learning (Van Hasselt, Guez, and Silver 2016), and Prioritized Replay (Schaul et al. 2016), and Dueling Networks (Wang et al. 2016). However, even the current best deep agents for Atari games, Rainbow (Hessel et al. 2017) and NEC (Pritzel et al. 2017), take millions of experiences to match humans.

Our work is most closely related to Blob-PROST (Liang et al. 2016, Machado et al. 2018), BASS, (Naddaf 2010), and other feature sets which specify millions of features in an attempt to capture object-level relations. Blob-PROST is the current state of the art, and functions by first segmenting an image into monocolor blobs. It then approximates relative object distance by dividing an image into tiles and having binary features $\phi_{k_1,k_2,(i,j)}$ to represent a pair of blobs of colors k_1, k_2 offset by (i, j) tiles. A similar set of binary features are used to approximate relative velocities, $\phi_{k_1,k_2,(i,j),t}$, indicating whether the relative position feature $\phi_{k_1,k_2,(i,j)}$ was true t timesteps in the past. We improve on these feature sets by demonstrating how to track percepts such as monocolor blobs across frames and combine these percepts into higher-level objects without any supervision. This allows object relative position and velocity to be obtained exactly and represented much more naturally and compactly with a few hundred scalar values rather than several million binary features. In addition, we are able to create of new features such as accelerations and contacts.

There has been much work on model-based deep reinforcement learning and predicting future states or values with deep networks. Oh, Guo, et al. 2015 and Chiappa et al. 2017 use deep neural networks to accurately predict Atari frames hundreds of steps into the future. However, they rely on hundreds of thousands of training frames, computationally intensive deep architectures, do not consider environment stochasticity, and predict at the pixel level rather than the more efficient object level. Dosovitskiy and Koltun 2017 predict future state for Doom, specifically health, ammunition, and monster kills and use this to plan actions. However, unlike objects, these features Doom-specific and have complex dynamics. Garnelo, Arulkumaran, and Shanahan 2016 does reinforcement learning on a high-level symbolic world representation learned with no supervision, but must still learn what good world representations are and is only successful on very simple environments. Xue et al. 2016; Ebert et al. 2017; Oh, S. Singh, and Lee 2017; Higuera, Meger, and Dudek 2018 all learn to predict future state, but still predict either very low-level features difficult to learn on or high-level features that require many sample to effectively model. AlphaZero (Silver et al. 2017) uses a model of its environment, Go, a deep neural network to estimate state value, and Monte Carlo tree search to plan Go moves remarkably well. However, its model relies on strong self-play developed with an immense number of experiences and human specified game rules. In contrast, we use highly general priors to quickly learn a model from data with no supervision or human engineering.

Relational MDPs (Guestrin et al. 2003) propose generalizing across objects to improve planning. Diuk, Cohen, and Littman 2008 propose reinforcement learning on Object Oriented MDPs, and Scholz et al. 2014 extends this by learning parameters to physics based models. Kansky et al. 2017 learn more general probabilistic object models and demonstrate how to plan on such models using inference. demonstrate the power of object-like features and linear policies to rival the performance of deep reinforcement agents, but sacrifice sample efficiency, interpretability, and the potential to model and plan by using millions of features. By learning the parameters of physics-based environments

models. Woof and K. Chen 2018 develop an object embedding network to achieve great performance in complex environments, but require labeled objects, which we demonstrate how to obtain with no supervision.

Reward shaping is another way to incorporate human priors into reinforcement learning (Ng, Harada, and Russell 1999). Hybrid Reward Architectures (Van Seijen et al. 2017) show the potential of object-level reinforcement learning by using per-object reward functions to achieve approximately 10x the score of Rainbow, the current state-of-the-art model-free agent (and 400x when using a simple form of memorization/planning), but relies on humans to label objects in its domain, Ms. Pacman.

There has been much work on augmenting agents using imitation learning (Ziebart et al. 2008; Hester et al. 2018; Le et al. 2018), where agents learn from demonstrations, often by humans, which may be viewed as another technique of incorporating human priors, albeit priors that are much more task-specific. However, demonstrations are typically very limited in number, making generalization difficult.

3 Background

Reinforcement learning attempts to solve the problem of taking actions in some environment to maximize some reward. Formally, let $a \in A$ be actions, $s \in S$ be environment states, $t \in 0, 1, \ldots, \infty$ be time steps, $T(s', s, a) = P(s'|s, a)$ be a stochastic transition function, $R(s, a)$ be a stochastic reward function, and $\gamma \in [0, 1]$ be a discount factor. These define a Markov decision process where the goal at each time t_i is to take the action a_i that maximizes the expected discounted reward, $\sum_{t=0}^{\infty} R(s_t, a_t)\gamma^t$. Reinforcement learning solves this MDP by learning a policy π which specifies the probability of the agent choosing each action given the current state. Define the state value function to be $V_\pi(s) = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}(s_{t+k+1}, a_{t+k})|s_t = s]$ and the state-action value function to be $Q_\pi(s, a) = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}(s_{t+k+1}, a_{t+k}))|s_t = s, a_t = a]$, where $E_\pi[\cdot]$ denotes the expected value of the random variable given that the agent follows π. Then the optimal state value and state-action value functions are $V_*(s) = \max_\pi V_\pi(s)$ and $Q_*(s, a) = \max_\pi Q_\pi(s, a)$ for all $s \in S$ and $a \in A$, and an optimal policy π is one that is greedy with respect to the optimal state value or state-action value function. A near-optimal π is typically created by learning approximations of $V_*(s)$ or $Q_*(s, a)$ and implementing π by greedily choosing the best state or state-action, taking random actions with probability ϵ to achieve exploration.

Model-based reinforcement learning allows the agent access to a model of the world. The model can take several forms, such as only allowing sampling possible future states, or outputting all possible future states and their probabilities given the current state and action. When the model is perfect and the MDP small, the problem may be solved using dynamic programming. However, this is rarely the case and the model must often be learned from experience on very large state spaces. For this case algorithms such as Dyna (Sutton 1991) and Real-time Dynamic Programming (Barto, Bradtke, and S. P. Singh 1995) learn policies by using the model as a simulator to generate additional training experiences.

Model-based reinforcement learning algorithms use their models to generate simulated trajectories to train the policy on. One of the most effective algorithms for such planning is Monte Carlo Tree Search (MCTS), which repeatedly samples the most promising action sequences according to a tree policy, approximating the return of a state by playing to the end of the episode from that state using a rollout policy. A learned approximation of state or state-action return may be used in place of this rollout. One common tree policy is UCT: a new action sequence is sampled by, at each action decision, selecting an unexplored action and returning the new action sequence, or, if none, selecting the child action A_i that maximizes the UCB1 confidence bound, $\text{UCB1} = \bar{A}_i + c\sqrt{\frac{2 \log n}{n_i}}$, where n and n_i are the number of times the parent and child actions, respectively, have been sampled. $\bar{A}_i$ is the mean terminal value of all sampled action sequences that take A_i, where the terminal value is the sum of rewards plus the value of the last sampled state.

4 Object-Level Reinforcement Learning

In this section we describe our Object-Level Reinforcement Learning (OLRL) framework which uses standard tools from computer vision, planning, and reinforcement learning to develop an agent

that incorporates human object priors. At a high level, our agent learns an object representation of the environment from pixels by oversegmenting images, modeling segment dynamics, and then combining segments with similar dynamics into objects. Our agent then learns state reward and value estimators as functions of object features. Finally, our agent determines optimal actions by planning using the dynamics models and reward and value estimators.

4.1 Object Recognition, Tracking, and Dynamics Modeling

We define objects as groups of percepts with very similar behavior. This definition provides a path to unsupervised object recognition, tracking, and modeling from pixels through coupled object recognition and dynamics modeling. Our agent first oversegments an input image into percepts using a standard image segmentation algorithm such as the Watershed algorithm (Haris et al. 1998) or SLIC (Achanta et al. 2012). These percepts are then tracked across images using a tracking algorithm such as SemiBoost (Grabner, Leistner, and Bischof 2008) or median flow (Kalal, Mikolajczyk, and Matas 2010) biased to under-track. This information allows us to represent the state of the world in terms of powerful object level features, such a positions, velocities, accelerations, and collisions. Our agent then trains a supervised model to predict the next game state given the current and past game states.

While any tracking algorithm could work, we developed the following for fast, reliable performance even in environments with complex object dynamics. First we segment the image into objects $o \in \mathcal{O}$, where each object o is a set of pixels, and then track objects across frames by scoring each object in the current frame, $o_c \in \mathcal{O}_c$, against the last seen instance of all previous objects, $o_p \in \mathcal{O}_p$, using a linear combination of several intuitive scores. Let $P_{dm}(x)$ be the probability of event x according to the dynamics model. Let $e_1, \ldots, e_i = \mathcal{E}$ be the i experiences the agent has encountered. Let $loc(o)$ for $o \in \mathcal{O}$ be the mean coordinates of the pixels composing object o.

$$\mathcal{S}(o_c, o_p) = c_0\mathcal{S}_{disp} + c_1\mathcal{S}_{shape} + c_2\mathcal{S}_{perm} + c_3\mathcal{S}_{size} + c_4\mathcal{S}_{motion}$$

$\mathcal{S}_{disp}$ is the total change in relative distance between o_c and o_p and all objects contacting o_p, capturing the notion that over small changes in time the positions of objects generally do not change too much.

$$\mathcal{S}_{shape} = \sum_{i=1}^{25} |\log(Z_i(o_c)) - \log(Z_i(o_p))|$$

where $Z_i(o) = i$th Zernike moment (Khotanzad and Hong 1990) of o, encoding object shape consistency.

$\mathcal{S}_{perm}$ captures the intuition that objects generally do not appear or disappear and is the number of experiences since o_p was last seen, clipped at 5,

$$\mathcal{S}_{size} = \max(\frac{|o_c|}{|o_p|}, \frac{|o_p|}{|o_c|}) - 1.$$

$\mathcal{S}_{motion}$ encodes that many objects are background objects and are unlikely-move:

$$\mathcal{S}_{motion} = \log(\max(P_{motion}(o_c, o_p), \epsilon)),$$

where ϵ is some small positive constant.

$$P_{motion}(o_c, o_p) = \begin{cases} P_{dm}(\text{Moves}(o_p)|e_1, \ldots, e_i), & \text{if } loc(o_c) \neq loc(o_p) \\ P_{dm}(\text{NotMoves}(o_p)|e_1, \ldots, e_i), & \text{if } loc(o_c) = loc(o_p) \end{cases}$$

We track objects by assigning o_c to o_p where $o_c, o_p = \arg\min_{o_c \in \mathcal{O}_c \setminus \mathcal{O}_{ac}, o_p \in \mathcal{O}_p \setminus \mathcal{O}_{ap}} \mathcal{S}(o_c, o_p)$ and $S(o_c, o_p) \leq t$, the threshold and adding o_c to $\mathcal{O}_{ac}$ and o_p to $\mathcal{O}_{ap}$ until no more such assignments can be made. Any objects remaining in $\mathcal{O}_c \setminus \mathcal{O}_{ac}$ are considered new objects.

As long as the object recognition and tracking always oversegments, so no two objects are perceived as one, and under-tracks, so no two objects are tracked as one across frames, we argue that this state representation is functionally as expressive as raw pixels. The requirements of oversegmenting and under-tracking are generally easily met by adjusting segmentation and tracking algorithm thresholds. However, an object-level representation must be succinct in addition to fully expressive in order to be useful, and our algorithm of always oversegmenting and under-tracking will produce many more

objects than necessary. To remedy this we use our definition of an object as a group of percepts with similar behavior: we combine two percepts into one object by testing if the error of a dynamics model with the two percepts as one object is not more than the error of the current dynamics model with the two percepts as separate objects. In addition to creating a more compact object representation, this percept combining gives us labels for when our tracking algorithm correctly and incorrectly recognized and tracked objects. These labels allow us to train supervised learners to fit the tracking to the environment, quickly allowing our perceptual system to segment and track the environment as a succinct set of objects in real time rather than requiring observations of percepts to correct its errors by combining those precepts into objects.

We learn an improved tracker by training a set of XGBoost (T. Chen and Guestrin 2016) trees, one per known object, on the history of observed attempted trackings. That is, for every pair of objects o_p and o_c considered, where o_c would be the next observation of o_p, we trained the tree to output a 1 if o_c was actually determined to be the next observation of o_p, and a 0 if not, given the tracking scores for o_p and o_c described above as inputs. If we combine two separate objects, $o_0 \in \mathcal{O}$ and $o_1 \in \mathcal{O}$, we search through the history of observed experiences $\{\mathcal{O}_0, \mathcal{O}_1, ...\}$, and everywhere we see $o_0 \in \mathcal{O}_n$ and $o_1 \in \mathcal{O}_{n+1}$, or vice versa, we retroactively mark the objects as predecessor and successor.

An intuitive and powerful way to model object dynamics is by training a set of models, $m_{o,d} \in \mathcal{M}$, one for each object o_i in dimension d_j, to output a distribution over future velocities for o_i in d_j. We then sampled from this velocity distribution to obtain a prediction of o_i's velocity in d_j. Let

$$\textit{Err}(m_{o,d},\, o',\, \mathcal{E}) = \sum_{e \in \mathcal{E}} \int_{v_{pred}=v_{min}}^{v_{max}} |v_{obs}(e, o') - v_{pred}| P_{m_{o,d}}(v_{pred})$$

We attempt to combine two objects, o_i and o_j into a new object, o_k, by training a new model, M' over the new set of objects, $\mathcal{O}'$, and comparing its error to the current model M. When evaluating m', we desire that the error for the combined objects has not increased. Let $\text{freq}(o, \mathcal{E})$ be the number of times object o appears in experiences $\mathcal{E}$. We define $\textit{Err}(M, \mathcal{E}, o) = \frac{\text{error}(m_{o,d}, o, \mathcal{E})}{\text{freq}(o, \mathcal{E})}$ and $\textit{Err'}(M', \mathcal{E}, o, o_k) = \frac{\text{error}(m'_{o_k,d}, o, \mathcal{E})}{\text{freq}(o, \mathcal{E})}$. We combine two objects only if $\textit{Err'}(M', \mathcal{E}, o, o_k) \leq (1 + r)\textit{Err}(M, \mathcal{E}, o)\ \forall o \in \{o_i, o_k\}$, where r is a regularization parameter.

By coupling object recognition. tracking, and modeling we learn a succinct object representation of environments with no supervision. Object recognition and tracking provide data for training a dynamics model, which in turn allows low-level percepts to be combined into higher-level objects.

4.2 Learning

We use our object-level representation to generate powerful features for learning: we represent the environment as object absolute and relative distances, velocities, accelerations, and contacts to the agent. We then learn a state-action value function in terms of this representation using one of the many available reinforcement learning algorithms, such as Monte Carlo or Temporal Difference methods (Sutton 1988). We also learn a state-reward estimator in terms of these features.

4.3 Planning

A state value and state reward estimator along with a predictive environment model allows us to utilize many powerful planning algorithms, including UCT and its many variants (Browne et al. 2012), to plan optimal action sequences.

4.4 Object Exploration

Rather than explore using ϵ-random actions, we use our object prior to develop a much more powerful exploration method. Objects generally affect each other when making contact or otherwise in close proximity (Spelke 1990). Therefore, a good way to obtain interesting experiences is to make objects contact. We do this by choosing a pair of objects, at least heavily one influenced by agent actions, and modifying the value function to proximity and contact between the two objects very highly for some number of future steps.

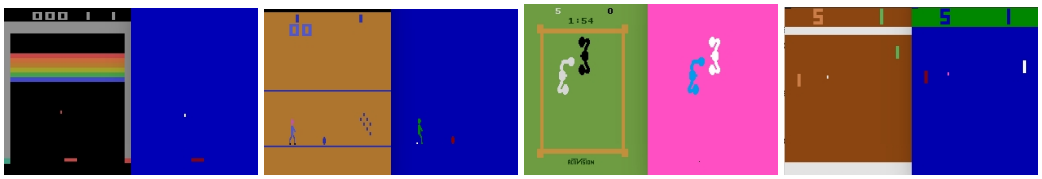

Figure 1: Object-level representations of Atari games. On the left we show the normal Atari game frames. On the right we show the same frame with each high-level object our agent perceives colored differently.

5 Experiments and Results

We first demonstrate that our object-level framework is capable of creating succinct and accurate object-level models on a variety of Atari games with very few samples, and little time and computational resources. We then demonstrate the potential of our representation by using simple reinforcement learning and planning algorithms to learn faster than humans and achieve better performance than DQN and Blob-PROST.

5.1 OLRL Implementation

We oversegment images into monocolor blobs, then track, model, and combine those low-level percepts as described in section 4.1. As shown in in Figure 1, our method of combining low-level percepts by searching for smaller yet still accurate models reduces the number of objects from dozens to a core few. We set the regularization parameter to 0.15, causing scores and other numbers to be combined with the background; reducing the regularization would prevent this if undesirable.

We modeled object dynamics by training a set of Lightgbm (Ke et al. 2017) trees to output a distribution over discretized velocities clipped to be in [-30, 30] given object position, position relative to other objects, velocity, acceleration, current contacts with other objects, and predicted future velocities for o_i in dimensions $d_{1 \leq k < j}$. We then sampled from this velocity distribution to obtain a prediction of o_i's velocity in d_j. The Lightgbm classifiers were trained to minimize multiclass log loss.

We use our object prior to build estimator architectures that allow for faster learning while still being general. We first use the prior that most tasks of interest can be described as combinations of pairwise relations between objects to justify a deep neural network architecture where the features for each pair of objects are first processed by several hidden layers which output a single scalar. These scalars, one for each object pair, are then input to a final fully connected layer which outputs the final estimate, as shown in Figure 2. The second object prior we use is that interesting events occur when objects contact, and, conversely, that interesting events do not occur when objects are not touching. This allows us to break our state-action value estimator into two parts.

The first is a simple linear regressor over the relative positions between objects, which learns which objects should contact and which objects should not contact.

The second is a deep neural network with the same object-pair architecture as the reward estimator (Figure 2), but with each object-pair network's output zeroed when the objects in the corresponding pair are not close to each other. This network learns how each pair of objects should interact, for example, how to hit a ball with a paddle to score a point, or how to successfully grasp an object. We train the object-pair value and reward neural networks on relative positions, relative velocities, and collisions between each object pair.

We follow the planning algorithm in (Oh, S. Singh, and Lee 2017) to search all possible actions sequences of length three into the future, choosing the sequence that maximizes the sum of state values at each future step. We implement object-level exploration by, at each timestep with chance $\epsilon = 0.04$, choosing a pair of previously unexplored objects, choosing the first object with probability equal to the importance of actions in its corresponding models relative to that of the other objects, and the other uniformly at random.

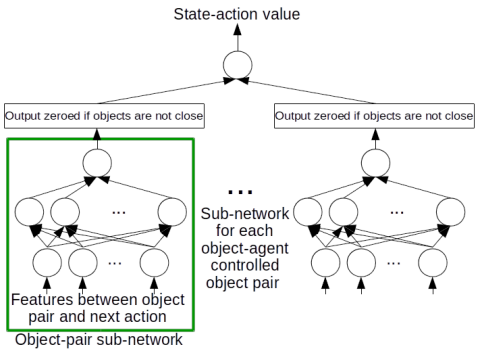

Figure 2: Object-Pair Value Network. Reward network is identical, except sub-network outputs are not zeroed and actions are not included in inputs.

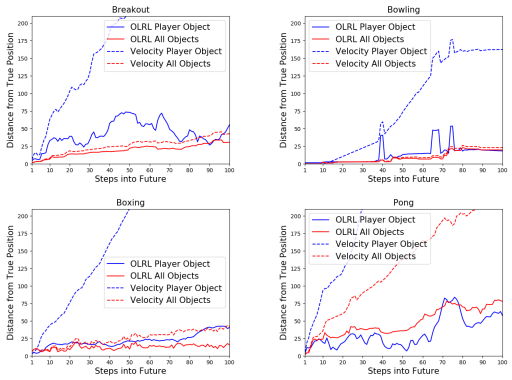

Figure 3: Model Error on Atari Games. Distances are in pixels; for comparison, the Atari game screen is 210x160 pixels.

5.2 Model Accuracy

We demonstrate the speed and accuracy of our object-level perception and modeling on a variety of Atari 2600 games. For each Atari game, we trained our model on only 2000 steps, where each step is 4 game frames, collected while playing random actions. We then evaluated our models by first playing n random actions, where n was sampled uniformly from [50, 150], and then predicting the positions of objects for the next 100 steps. Figure 3 shows prediction error as the average Euclidean distance between the predicted and actual object positions. Since many Atari objects do not move, also show prediction error for the player-controlled object, which has complex dynamics. For the same reason, we use predicting object velocity will remain constant as a natural baseline.

Even with very little training data, our models have learned object dynamics with high accuracy tens of frames into the future, effective for short-term planning. Unlike previous work, model prediction error generally does not explode for longer prediction horizons, allowing the agent to consider the approximate positions over objects even 100 steps into the future.

5.3 Comparison to Humans

In Figure 4 we compare the median number of samples required to achieve Breakout scores for humans and OLRL. Human learning curves were collected from Amazon Mechanical Turk using the same methodology as in Tsividis et al. 2017. All human experiments were conducted with no frame skip and no action repeat chance. We divide the number of human frames played by four to allow comparison to the OLRL tests which used a frameskip of 4; note that this makes the human environments strictly easier than the OLRL environments, with humans experiencing four frames for every OLRL experience. We say that humans have learned Breakout when they have achieved at least the expert player Breakout score given in Mnih, Kavukcuoglu, et al. 2015, and therefore, our OLRL learns slightly faster than humans.

5.4 Comparison to DQN and Blob-PROST

We compared OLRL to DQN and Blob-PROST in Figure 5 using the ALE parameters recommended in Machado et al. 2018, with the exception of using the smaller OpenAI Gym action sets (Brockman et al. 2016). DQN and Blob-PROST data was drawn from Machado et al. 2018. Using our object-level features, our agent reaches the performance of DQN and Blob-Prost with approximately 10,000x fewer experiences. We note that the data for DQN and Blob-PROST was collected using a larger action space than we did, so we include the peak average score of Blob-PROST trained Breakout with our smaller action set and also no action repeat from Liang et al. 2016, which OLRL also easily surpassed. Figure 7 shows how quickly our OLRL trains; on only a desktop CPU it achieves DQN-level performance in under two hours.

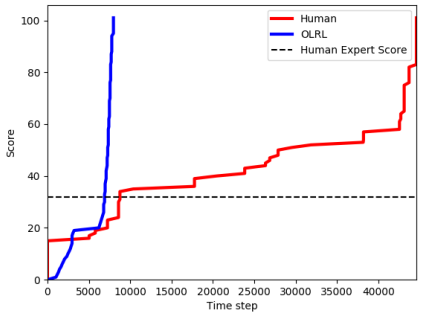

Figure 4: Comparison of OLRL and humans on Breakout.

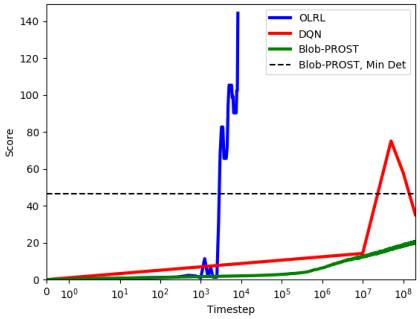

Figure 5: Comparison of OLRL and DQN on Breakout. Note timesteps are in log-scale.

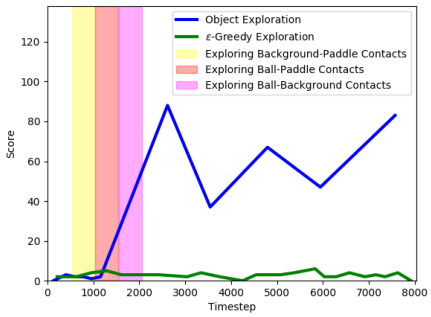

Figure 6: Comparison of OLRL agent with object exploration and OLRL agent with ϵ-greedy exploration on Breakout.

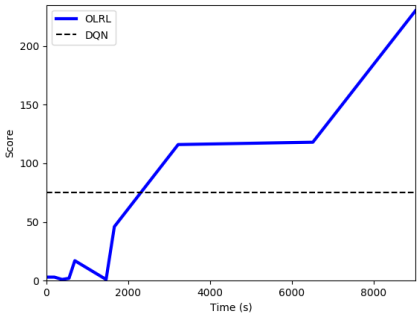

Figure 7: Learning time for OLRL on Breakout.

5.5 Object Exploration

We evaluated the effectiveness of our object-level exploration in Figure 6 by comparing it against an OLRL agent only using ϵ-greedy exploration, with $\epsilon = 0.01$. The goal of Breakout may be roughly summarized as hitting the ball with the paddle. The two agents had similar performance until the object-exploring agent explored contacts between the ball and paddle, when it rapidly learned the importance of contacts between the two objects. As these results show, our object prior provides a powerful new exploration technique.

6 Conclusion and Future Work

In this paper, we have shown that incorporating simple and general human object priors can greatly improve the sample efficiency of reinforcement learning. In particular, our OLRL system is able to reach human-level performance on Atari games with up to 10,000x fewer samples than DQNs. OLRL learns in just hours on a single core, which is dramatically more efficient than previous approaches. OLRL is also the the first system to learn an Atari game learn in fewer frames than humans. In addition, we observe that the use of object-level representations makes the results of learning significantly more interpretable. We plan to open-source the OLRL code, so that it can be used as a preprocessor and simple baseline to greatly speed up experimentation in this area. The next step is to integrate our object-level representation with current deep reinforcement learning algorithms and planning techniques. Currently, OLRL only uses dynamics to determine which objects are the same. However, this is not failproof: some objects that never move relative to the background

have high value, and some objects that do move relative to the background have low value. We propose also using the learned value and reward estimators when determining which percepts may be treated as a single object. Finally, we plan to apply our approach to real-world problems like robot manipulation and navigation.

7 Acknowledgements

The first author was supported by an NDSEG Fellowship. This research was partly funded by ONR grants N00014-18-1-2826 and N00014-16-1-2697. The GPU machine used for this research was donated by Nvidia. We would like to thank Sidd Srinivasa, Liyiming Ke, and Kevin Jamieson for their support, ideas, and feedback. We would also like to thank Pedro Tsividis and Josh Tenenbaum for providing data on human Atari learning.

References

Achanta, Radhakrishna, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, Sabine Süsstrunk, et al. (2012). "SLIC superpixels compared to state-of-the-art superpixel methods". In: *IEEE transactions on pattern analysis and machine intelligence* 34.11, pp. 2274–2282.

Barto, Andrew G, Steven J Bradtke, and Satinder P Singh (1995). "Learning to act using real-time dynamic programming". In: *Artificial Intelligence* 72.1-2, pp. 81–138.

Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba (2016). "Openai gym". In: *arXiv:1606.01540*.

Browne, Cameron B, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton (2012). "A survey of monte carlo tree search methods". In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1, pp. 1–43.

Chen, Tianqi and Carlos Guestrin (2016). "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, pp. 785–794.

Chiappa, Silvia, Sébastien Racaniere, Daan Wierstra, and Shakir Mohamed (2017). "Recurrent environment simulators". In: *arXiv preprint arXiv:1704.02254*.

Diuk, Carlos, Andre Cohen, and Michael L Littman (2008). "An object-oriented representation for efficient reinforcement learning". In: *International Conference on Machine learning*. ACM, pp. 240–247.

Dosovitskiy, Alexey and Vladlen Koltun (2017). "Learning to act by predicting the future". In: *International Conference on Learning Representations*.

Ebert, Frederik, Chelsea Finn, Alex X Lee, and Sergey Levine (2017). "Self-Supervised Visual Planning with Temporal Skip Connections". In: *Conference on Robot Learning*, pp. 344–356.

Garnelo, Marta, Kai Arulkumaran, and Murray Shanahan (2016). "Towards deep symbolic reinforcement learning". In: *arXiv:1609.05518*.

Grabner, Helmut, Christian Leistner, and Horst Bischof (2008). "Semi-supervised on-line boosting for robust tracking". In: *European Conference on Computer Vision*. Springer, pp. 234–247.

Guestrin, Carlos, Daphne Koller, Chris Gearhart, and Neal Kanodia (2003). "Generalizing plans to new environments in relational MDPs". In: *International Joint Conference on Artificial intelligence*. Morgan Kaufmann Publishers Inc., pp. 1003–1010.

Haris, Kostas, Serafim N Efstratiadis, Nikolaos Maglaveras, and Aggelos K Katsaggelos (1998). "Hybrid image segmentation using watersheds and fast region merging". In: *IEEE Transactions on Image Processing* 7.12, pp. 1684–1699.

Hessel, Matteo, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver (2017). "Rainbow: combining improvements in deep reinforcement Learning". In: *arXiv preprint arXiv:1710.02298*.

Hester, Todd, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, et al. (2018). "Deep Q-learning from demonstrations". In: *Association for the Advancement of Artificial Intelligence*.

Higuera, Juan Camilo Gamboa, David Meger, and Gregory Dudek (2018). "Synthesizing neural network controllers with probabilistic model based reinforcement learning". In: *arXiv:1803.02291*.

Kalal, Zdenek, Krystian Mikolajczyk, and Jiri Matas (2010). "Forward-backward error: Automatic detection of tracking failures". In: *International Conference on Pattern Recognition*. IEEE, pp. 2756–2759.

Kansky, Ken, Tom Silver, David A Mely, Mohamed Eldawy, Miguel Lázaro-Gredilla, Xinghua Lou, Nimrod Dorfman, Szymon Sidor, Scott Phoenix, and Dileep George (2017). "Schema Networks: Zero-shot Transfer with a Generative Causal Model of Intuitive Physics". In: *International Conference on Machine Learning*, pp. 1809–1818.

Ke, Guolin, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu (2017). "Lightgbm: A highly efficient gradient boosting decision tree". In: *Advances in Neural Information Processing Systems*, pp. 3146–3154.

Khotanzad, Alireza and Yaw Hua Hong (1990). "Invariant image recognition by Zernike moments". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.5, pp. 489–497.

Le, Hoang M, Nan Jiang, Alekh Agarwal, Miroslav Dudik, Yisong Yue, and Hal Daume III (2018). "Hierarchical imitation and reinforcement learning". In: *arXiv:1803.00590*.

Liang, Yitao, Marlos C Machado, Erik Talvitie, and Michael Bowling (2016). "State of the art control of atari games using shallow reinforcement learning". In: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, pp. 485–493.

Machado, Marlos C, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling (2018). "Revisiting the Arcade Learning Environment: evaluation protocols and open problems for general agents". In: *Journal of Artificial Intelligence Research* 61, pp. 523–562.

Mnih, Volodymyr, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu (2016). "Asynchronous methods for deep reinforcement learning". In: *International Conference on Machine Learning*, pp. 1928–1937.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. (2015). "Human-level control through deep reinforcement learning". In: *Nature* 518.7540, p. 529.

Naddaf, Yavar (2010). "Game-independent ai agents for playing Atari 2600 console games". In:

Ng, Andrew Y, Daishi Harada, and Stuart Russell (1999). "Policy invariance under reward transformations: Theory and application to reward shaping". In: *International Conference on Machine Learning*.

Oh, Junhyuk, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh (2015). "Action-conditional video prediction using deep networks in atari games". In: *Advances in Neural Information Processing Systems*, pp. 2863–2871.

Oh, Junhyuk, Satinder Singh, and Honglak Lee (2017). "Value prediction network". In: *Advances in Neural Information Processing Systems*, pp. 6120–6130.

Pritzel, Alexander, Benigno Uria, Sriram Srinivasan, Adria Puigdomenech, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell (2017). "Neural episodic control". In: *arXiv preprint arXiv:1703.01988*.

Schaul, Tom, John Quan, Ioannis Antonoglou, and David Silver (2016). "Prioritized experience replay". In: *International Conference on Learning Representations*.

Scholz, Jonathan, Martin Levihn, Charles Isbell, and David Wingate (2014). "A physics-based model prior for object-oriented mdps". In: *International Conference on Machine Learning*, pp. 1089–1097.

Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. (2017). "Mastering the game of Go without human knowledge". In: *Nature* 550.7676, p. 354.

Spelke, Elizabeth S (1990). "Principles of object perception". In: *Cognitive science* 14.1, pp. 29–56.

Sutton, Richard S (1988). "Learning to predict by the methods of temporal differences". In: *Machine learning* 3.1, pp. 9–44.

– (1991). "Dyna, an integrated architecture for learning, planning, and reacting". In: *ACM SIGART Bulletin* 2.4, pp. 160–163.

Tsividis, Pedro A, Thomas Pouncy, Jacqueline L Xu, Joshua B Tenenbaum, and Samuel J Gershman (2017). "Human learning in Atari". In:

Van Hasselt, Hado, Arthur Guez, and David Silver (2016). "Deep Reinforcement learning with double q-learning." In: *Association for the Advancement of Artificial Intelligence*. Vol. 16, pp. 2094–2100.

Van Seijen, Harm, Mehdi Fatemi, Joshua Romoff, Romain Laroche, Tavian Barnes, and Jeffrey Tsang (2017). "Hybrid reward architecture for reinforcement learning". In: *Advances in Neural Information Processing Systems*, pp. 5392–5402.

Wang, Ziyu, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas (2016). "Dueling Network Architectures for Deep Reinforcement Learning". In: *International Conference on Machine Learning*, pp. 1995–2003.

Woof, William and Ke Chen (2018). "Learning to play general video-games via an object embedding network". In: *arXiv:1803.05262*.

Xue, Tianfan, Jiajun Wu, Katherine Bouman, and Bill Freeman (2016). "Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks". In: *Advances in Neural Information Processing Systems*, pp. 91–99.

Ziebart, Brian D, Andrew L Maas, J Andrew Bagnell, and Anind K Dey (2008). "Maximum entropy inverse reinforcement learning." In: *AAAI*. Vol. 8. Chicago, IL, USA, pp. 1433–1438.

8 Appendix

8.1 Reward and Value Estimation Details

The hidden layers of the object-pair sub-networks had 16 neurons. Each layer used a leaky ReLU activation function, with the exception of the output layer, which used a sigmoid activation of the reward prediction network (rewards were normalized to between 0 and 1) and a ReLU activation function for the value network. The value network was trained with n-step TD(λ) with $n = 100$, $\lambda = 0.99$ and $\gamma = 0.99$, and the reward network was trained to minimize the L2 error. Each was trained with RMSprop with a learning rate of $5 \times 10^(- 3)$, alpha of 0.99, and eps of 10^{-5}, and an early stopping parameter of 20 epochs, holding out the most recent 25% of experiences as a validation set. We trained the linear regressor using TD(λ) with $\lambda = 0.99$ and $\gamma = 0.99$.