Bootstrap Latent-Predictive Representations for Multitask Reinforcement Learning

Zhaohan Daniel Guo [*1] **Bernardo Avila Pires** [*1] **Bilal Piot** [1] **Jean-Bastien Grill** [2] **Florent Altché** [2]
Rémi Munos [2] **Mohammad Gheshlaghi Azar** [1]

Abstract

Learning a good representation is an essential component for deep reinforcement learning (RL). Representation learning is especially important in multitask and partially observable settings where building a representation of the unknown environment is crucial to solve the tasks. Here we introduce Predictions of Bootstrapped Latents (PBL), a simple and flexible self-supervised representation learning algorithm for multitask deep RL. PBL builds on multistep predictive representations of future observations, and focuses on capturing structured information about environment dynamics. Specifically, PBL trains its representation by predicting latent embeddings of future observations. These latent embeddings are themselves trained to be predictive of the aforementioned representations. These predictions form a bootstrapping effect, allowing the agent to learn more about the key aspects of the environment dynamics. In addition, by defining prediction tasks completely in latent space, PBL provides the flexibility of using multimodal observations involving pixel images, language instructions, rewards and more. We show in our experiments that PBL delivers across-the-board improved performance over state of the art deep RL agents in the DMLab-30 and Atari-57 multitask suites.

1. Introduction

Deep reinforcement learning (RL) has seen many successes in the recent years (Mnih et al., 2015; Levine et al., 2016; Silver et al., 2017; Vinyals et al., 2019). However, there are still many improvements to be made in complex, multitask and partially observable environments. Previous work (Mirowski et al., 2017; Jaderberg et al., 2017; Hessel et al., 2019; Gregor et al., 2019) has demonstrated that augmenting deep RL agents with auxiliary tasks for representation learning helps improve RL task performance. Representation learning has also had broader impact in deep RL, as representations have been used to generate intrinsic motivation signal for exploration (Houthooft et al., 2016; Pathak et al., 2017; Burda et al., 2019; Azar et al., 2019; Puigdomènech Badia et al., 2020) and as the basis for search models in planning agents (Oh et al., 2017; Amos et al., 2018; Hafner et al., 2019; Schrittwieser et al., 2019).

A common approach for representation learning is through training a (recurrent) neural network (RNN) to make predictions on future observations in a self-supervised fashion (Oh et al., 2017; Amos et al., 2018; Moreno et al., 2018; Guo et al., 2018; Hafner et al., 2019; Schrittwieser et al., 2019; Gregor et al., 2019). Predictive models for representation learning have been widely used as auxiliary tasks for deep RL (Jaderberg et al., 2017; Oord et al., 2018; Zhang et al., 2019; Aytar et al., 2018; Gregor et al., 2019; Song et al., 2020). However, to learn a rich and useful representation, these methods may demand a high-degree of accuracy in multistep prediction of future observations (Gregor et al., 2015; 2019). This degree of accuracy can be difficult to achieve in many problems, especially in partially observable and multitask settings, where uncertainty in the agent state and complex, diverse observations make prediction more challenging. In these settings, instead of trying to predict everything accurately, we would ideally want to predict a latent embedding of future observations that focuses on the key, structural information of the tasks.

In this paper, we introduce Predictions of Bootstrapped Latents (PBL, "pebble"), a new representation learning technique for deep RL agents. PBL learns a representation of the history (*agent state*, Sutton & Barto, 2018) by predicting the future *latent* embeddings of observations. The novel contribution is in how we train these latent embeddings, which we do by having the latent observation embedding at each timestep be predictive of the agent state at the same timestep. This leads to a bootstrapping effect, where the

[*]Equal contribution [1]DeepMind, London, UK [2]DeepMind Paris, France. Correspondence to: Mohammad Gheshlaghi Azar <mazar@google.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

agent states learn to predict future latent observations, and future latent observations learn to predict the corresponding future agent states. As an example, consider a task where an agent must find a key and use it to unlock a door. All histories that lead to an unlocked door must also contain the observation of finding the key. PBL trains the latent embedding of observations of the unlocked door to predict the agent states that capture these histories, which means the latent embedding is encouraged to encode information pertaining to the key. This example illustrate how representations trained with PBL may capture contextual, structured information about the dynamics of the environment. This is especially beneficial in settings with diverse and complex observations. Furthermore, PBL allows for bootstrapping information from the far future, since the predicted agent state is also predictive of future. Finally, by making predictions solely in latent space, PBL makes it easier to incorporate observations with multiple modalities such as images, voice and language instruction.

We evaluate PBL in the challenging partially observable, multitask setting, where the task indices are kept hidden from the agent (Brunskill & Li, 2013). The multitask setting is the standard setting to evaluate deep RL agents as it is important for an AI agent to perform well, across the board, on a variety of tasks (Finn et al., 2017; Rothfuss et al., 2018; Espeholt et al., 2018; Hessel et al., 2019; Song et al., 2020). As the benchmark, we use DMLab-30 (Beattie et al., 2016) task suite as a standard partially observable and multitask RL domain, and compare the performance of various representation learning techniques when they are used as auxiliary representation learning tasks for the PopArt-IMPALA agent (Hessel et al., 2019), i.e., both RL and representation learning techniques train the same RNN. In particular we report the performance of PBL, pixel control (state-of-the-art), DRAW (Gregor et al., 2019) and contrastive predictive coding (CPC) (Oord et al., 2018) in this setting. We also evaluate our approach on multi-task Atari-57 suite (Bellemare et al., 2013), comparing its performance with the RL baseline (PopArt-IMPALA) with various auxiliary representation learning task. We show, in section 4, that PBL, is able to outperform all these methods both in terms of final performance and sample efficiency in DMLab-30 suite. PBL also slightly improves the performance in comparison with the other methods in Atari-57 suite. This result suggests that PBL can be used as an alternative for the existing representation learning methods in the context of multitask deep RL.

2. Background

Partially Observable Environments. We model environments as partially observable Markov decision processes (POMDP, Cassandra et al., 1994) where an agent does not directly perceive the underlying states but only a partial view of the underlying states. More formally, a POMDP is a tuple $(\mathbb{S}, \mathbb{A}, \mathbb{O}, p_{\mathbb{S}}, p_{\mathbb{O}}, r, \gamma)$, where $\mathbb{S}$ is the state space, $\mathbb{A}$ the action space and $\mathbb{O}$ the observation space. The transition probability kernel $p_{\mathbb{S}}$ determines the probability of the next state S_{t+1} given the current state and action (S_t, A_t): $p_{\mathbb{S}}(S_{t+1} = s' | S_t = s, A_t = a)$. The observation kernel $p_{\mathbb{O}}$ dictates the observation process: $p_{\mathbb{O}}(O_t = o | S_t = s)$. $r \in \mathbb{R}^{\mathbb{S}\times\mathbb{A}}$ represents the reward function and γ is the discount factor. [1]

Policies dictate the action process: A policy is a mapping from sequences of observations and actions to a distribution over actions, that is, $\pi : \{\mathbb{O} \times (\mathbb{A} \times \mathbb{O})^n : n \in \mathbb{N}\} \to \Delta(\mathbb{A})$, where $\Delta(\mathbb{A})$ denotes the space of action distributions. For a fixed policy π and for all $t \geq 0$, we define recursively, the states S_t^π, the observations O_t^π, the histories H_t^π and actions A_t^π starting form some initial distribution ρ and following π:

$$\textit{Initialization:} \begin{cases} S_0^\pi \sim \rho, \quad O_0^\pi \sim p_{\mathbb{O}}(S_0^\pi), \\ H_0^\pi \doteq O_0^\pi, \quad A_0^\pi \sim \pi(H_0^\pi). \end{cases}$$

$$\textit{Recurrence:} \begin{cases} S_{t+1}^\pi \sim p_{\mathbb{S}}(S_t^\pi, A_t^\pi), \quad O_{t+1}^\pi \sim p_{\mathbb{O}}(S_{t+1}^\pi), \\ H_{t+1}^\pi \doteq (H_t^\pi, A_t^\pi, O_{t+1}^\pi), \\ A_{t+1}^\pi \sim \pi(H_{t+1}^\pi). \end{cases}$$

The *partial history* $H_{t,k}^\pi \doteq (H_t^\pi, A_t^\pi, \ldots, A_{t+k-1}^\pi)$ is formed by the history H_t^π and the k subsequent actions $A_t^\pi, \ldots, A_{t+k-1}^\pi$.

The RL (Sutton & Barto, 2018; Szepesvári, 2010) problem in the POMDP setting is finding the policy that maximizes the expected and discounted sum of rewards:

$$\max_\pi E\left(\sum_{t=0}^{\infty} \gamma^t r(S_t^\pi, A_t^\pi)\right).$$

A common strategy for RL in partially observable domains is to compress the full history H_t^π as the *agent state* B_t^π (Sutton & Barto, 2018) using a neural network and solve

$$\max_\pi E\left(\sum_{t=0}^{\infty} \gamma^t r(B_t^\pi)\right),$$

where $r(B_t^\pi) \doteq E(r(S_t^\pi, A_t^\pi) | B_t^\pi)$. Accordingly we consider the policy π as a mapping from the agent state B_t^π to distributions over actions as well. To avoid clutter, we will drop the π superscript from the POMDP variables.

Predictive Representations. Deep RL methods use neural networks to produce the agent state, and learn it from

[1] We note that the multitask learning setting (Brunskill & Li, 2013), where the identity of task is kept hidden from the agent, is absorbed into the POMDP formulation, since the unobserved state can also index different per-task dynamics and reward functions.

RL and self-supervised auxiliary tasks (Mnih et al., 2016; Jaderberg et al., 2017). A common approach to learn agent states is to learn predictive representations (Oh et al., 2015; Jaderberg et al., 2017; Oh et al., 2017; Amos et al., 2018; Guo et al., 2018; Ha & Schmidhuber, 2018; Moreno et al., 2018; Oord et al., 2018; Gregor et al., 2019; Hafner et al., 2019; Schrittwieser et al., 2019). One way to do so is to train representations to predict statistics of future observations conditioned on partial histories (Guo et al., 2018; Gregor et al., 2019). This technique uses RNNs to compress full histories H_t as B_t and partial histories $H_{t,k}$ as $B_{t,k}$, as inputs to auxiliary prediction tasks.

Figure 1 outlines the RNN architecture for compressing histories. The horizontal direction shows the RNN

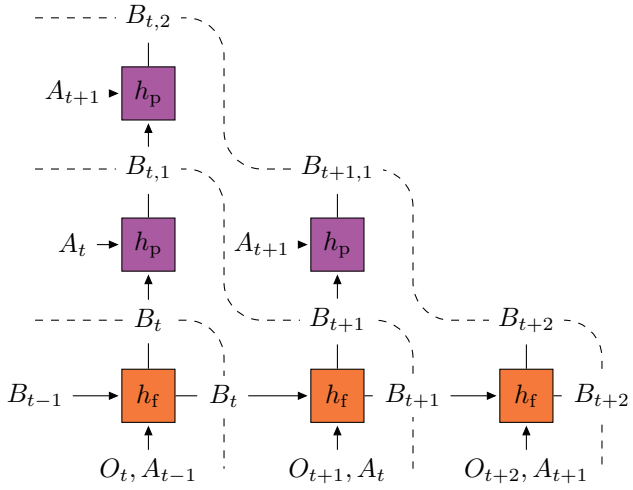


Figure 1. Recurrent architecture for compressing partial histories. Networks used for processing observations and actions have been omitted, and dashed lines connect histories and partial histories aligned in time.

h_f compressing full histories, with actions and observations as inputs. The agent state is updated as $B_{t+1} \doteq h_\mathrm{f}(B_t, O_{t+1}, A_t)$ (with B_0 equal zero).

The vertical direction shows the RNN h_p compressing partial histories, starting from agent states, and continuing with actions as the only inputs. The state of h_p is updated as

$$B_{t,1} \doteq h_\mathrm{p}(B_t, A_t)$$
$$B_{t,k+1} \doteq h_\mathrm{p}(B_{t,k}, A_{t+k}),$$

One can use the agent state B_t as the basis (input) for the decision making of the RL algorithm (Jaderberg et al., 2017; Espeholt et al., 2018; Hessel et al., 2019). The partial history RNN h_p is only used for predictions about the future.

3. Predictions of Bootstrapped Latents (PBL)

Our method, Predictions of Bootstrapped Latents (PBL, "pebble"), consists of two auxiliary prediction tasks: 1) a forward, action-conditional prediction from compressed partial histories to future latent observations; and 2) a reverse prediction from latent observations to agent states.

The forward, action-conditional prediction task of PBL consists of predicting latent embeddings, $Z_{t+k} = f(O_{t+k})$, of future observations, from a compressed partial history $B_{t,k}$:

$$E(Z_{t+k}|B_{t,k}) \approx E(Z_{t+k}|H_{t,k}) \quad (1)$$
$$= E(Z_{t+k}|H_t, A_t, \ldots, A_{t+k-1}). \quad (2)$$

To learn rich representations we make many predictions into the future—we take k to range from one to a maximum index in the future, which we call the *horizon*.

To predict Z_{t+k}, we solve:

$$\min_{h\in\mathbb{H}, g\in\mathbb{G}} \sum_{t,k} \|g(B_{t,k}) - Z_{t+k}\|_2^2, \quad (3)$$

where $\mathbb{H}$ and $\mathbb{G}$ are hypothesis spaces induced by neural networks, g is a feed-forward neural network, and $h = (h_\mathrm{f}, h_\mathrm{p})$ are the RNN networks that compute B_t and $B_{t,k}$. We call this problem a *forward prediction* (from compressed partial histories to latents), and its schematic is given in fig. 2.

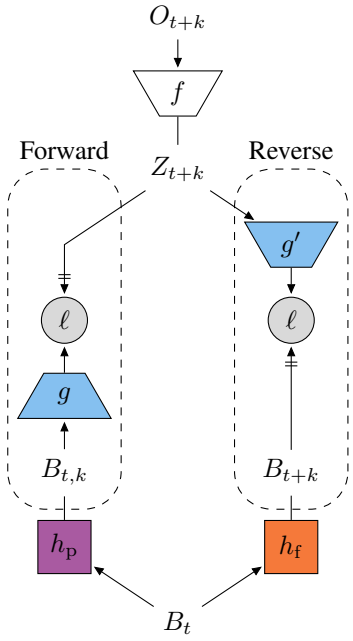


Figure 2. Architecture diagram for PBL, forward prediction (left) and reverse prediction (right). Crossed arrows block gradients and $\ell(x, y) \doteq \|x - y\|_2^2$.

The choice of Z_t is an essential part of our representation

learning. By picking Z_t to be a *latent* embedding of observations, we are able to learn meaningful observation features, as well as combine multiple modalities of observations (images and natural language sentences, for example) into a single representation. Specifically, PBL trains the latent embedding neural network f by predicting the learned B_t from Z_t. Concretely, we solve

$$\min_{f \in \mathbb{F}, g' \in \mathbb{G}'} \sum_t \|g'(\overbrace{f(O_t)}^{Z_t}) - B_t\|_2^2, \tag{4}$$

where $\mathbb{F}$ and $\mathbb{G}'$ are hypothesis spaces induced by neural networks, and g' is another feed-forward neural network. We call this problem *reverse prediction*, from latents to compressed histories, and its schematic is also given in fig. 2. See algorithm 1 for a high-level overview of the algorithm.

Forward prediction trains agent states to predict future latent observations. Reverse prediction trains latent observations to predict immediate agent states. Together, they can form a cycle, bootstrapping useful information between compressed histories and latent observations.

Because of the bootstrapping effect, the reverse prediction has the potential to train meaningful latent embeddings of observations that encode structural information about the paths that the agent takes to reach these observations. Furthermore, since we are training agent states to be predictive, this bootstrapping effect may also bootstrap useful information from far into the future.

Note that both forward and reverse predictions are one-way, i.e., we do not pass gradients into the targets of the predictions. In practice, we optimize both objectives together, as they optimize disjoint sets of parameters. A natural question that arises is what makes the bootstrapped predictions work, as it seems like there is the possibility that the prediction tasks collapse into trivial solutions. We hypothesize that due to forward and reverse predictions being one-way, the training dynamics actively move away from the trivial solution. Initially, the latent embedding of observations and agent states are the output of randomly initialized neural networks, which can be considered as random, non-linear projections. The forward prediction encourages the agent state to move away from collapsing in order to accurately predict future random projections of observations. Similarly, the reverse prediction encourages the latent observation away from collapsing in order to accurately predict the random projection of a full history. As we continue to train forward and reverse predictions, this seems to result in a virtuous cycle that continuously enriches both representations. In section 4.3, we empirically analyze this issue.

4. Experiments

In this section, we present our main experimental results (section 4.1) where we show that PBL, when used as the auxiliary task to shape the representation, outperforms other representation learning auxiliary tasks in DMLab 30 benchmark. Then we empirically investigate the scalability of PBL with the horizon of prediction and the effect of the reverse prediction in section 4.2, the stability of PBL in section 4.3, the neural network architecture choice in section 4.4, and what is captured by the learned representation in section 4.6. We discuss implementation details in section 4.7. Additionally, we performed experiments in the standard Atari-57 domain (Bellemare et al., 2013), which we report in section 4.5.

4.1. DMLab 30

We evaluated PBL in the DMLab 30 task set (Beattie et al., 2016). In this benchmark the agent must perform navigation and puzzle-solving tasks in 3D environments. There are a total of 30 tasks, each with a number of procedural variations over episodes. In the multitask setup of DMLab 30 each episode may come from a different task, and the agent *does not* receive the task identifier as an input. The only input the agent receives is an image of the first-person view of the environment, a natural language instruction (in some tasks), and the reward.

We compared PBL with different representation learning techniques, when they are used as auxiliary tasks for the deep RL agent. Pixel control (Jaderberg et al., 2017) is the current state-of-the-art self-supervised representation learning technique used in DMLab 30, having been applied to PopArt-IMPALA (Hessel et al., 2019) and, more recently, V-MPO (Song et al., 2020). CPC has been successful for representation learning in different settings (Oord et al., 2018). One advantage of CPC is its simplicity; it is lightweight and easy to incorporate in different applications. Simcore DRAW (Gregor et al., 2019) has achieved strong performance in some of the DMLab 30 tasks in the single-task regime. It is also a good method to compare with PBL, since it learns the representation with multistep predictions of the future. In contrast to CPC (which uses a contrastive approach for representation learning), Simcore DRAW uses a strong generative model of frames (Gregor et al., 2015).

All the representation learning techniques have been combined with PopArt-IMPALA (Hessel et al., 2019) as the base RL algorithm. PopArt-IMPALA has been shown to perform well in DMLab 30, where the use of PopArt makes the agent suitable for handling different reward scales across tasks. For PBL, Simcore DRAW, and CPC, we predict up to 20 steps into the future.

The most common performance measure for DMLab 30

Algorithm 1 Training Step Pseudocode for PBL

Require: Minibatch of trajectories $B = \{O_t^{(i)}, A_t^{(i)}, R_t^{(i)}\}$, RNN h_p, RNN h_f, MLPs g, g', f, future prediction horizon k, RLLoss (reinforcement learning loss)
Encode observation $Z_t^{(i)} \doteq f(O_t^{(i)})$
Let $B_0^{(i)} \doteq \mathbf{0}$ and $B_t^{(i)} \doteq h_\mathrm{f}(B_{t-1}^{(i)}, O_t^{(i)}, A_{t-1}^{(i)}) \doteq B_{t,0}^{(i)}$ and $B_{t,k}^{(i)} \doteq h_\mathrm{p}(B_{t,k-1}^{(i)}, A_{t+k-1}^{(i)})$
$\mathrm{Forward}(B_t^{(i)}) \doteq \frac{1}{k}\sum_{j=1}^{k} \|g(B_{t,j}^{(i)}) - \mathrm{StopGradient}(Z_{t+j}^{(i)})\|_2^2$
$\mathrm{Reverse}(Z_t^{(i)}) \doteq \|g'(Z_t^{(i)}) - \mathrm{StopGradient}(B_t^{(i)})\|_2^2$
Take gradient step of $\min \frac{1}{|B|}\sum_{i,t}\Big(\mathrm{Forward}(B_t^{(i)}) + \mathrm{Reverse}(Z_t^{(i)}) + \mathrm{RLLoss}(B_t^{(i)}, R_t^{(i)})\Big)$

is the mean over tasks of the capped human normalized score. The human normalized score of an agent on a task i is calculated as $100 \cdot (a_i - u_i)/(h_i - u_i)$, where a_i is the agent's expected return on that task, u_i is the expected return of an agent that selects actions uniformly at random, and h_i is the expected return of a human performing the task. Reference values for the random and human scores can be found in the work of Espeholt et al. (2018). The *capped* human normalized score of an agent on a task is the human normalized score capped at 100. Therefore, the mean capped human normalized emphasizes performance gains toward attaining human performance across tasks and disregards improvements that are superhuman. In this work, we primarily report this score when aggregating results across tasks, but the mean human normalized score can be found in appendix B.

Figure 3 shows the mean capped human normalized score for PBL and our baselines, averaged across 16 independent runs, and with 95% confidence intervals. Appendix A.10 gives additional details on the plotting protocol for our experiments. We see that PBL outperforms all other methods

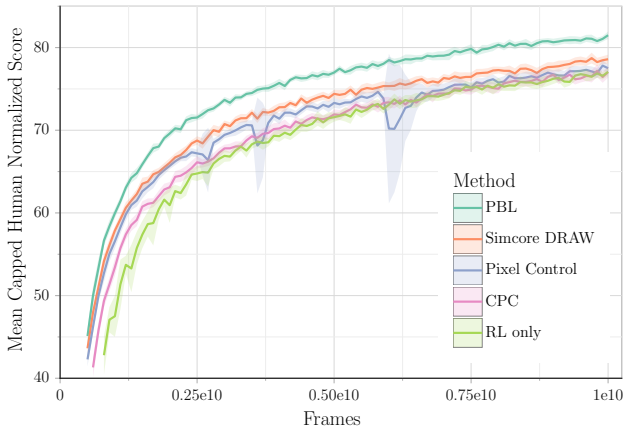


Figure 3. Mean capped human normalized score for compared methods.

throughout the whole training process by a significant margin. Since we are comparing how representation learning techniques lead to performance improvement, it is worth comparing how each method improves over the RL only baseline. The mean capped human normalized score of this baseline, over the last 5% ($5 \cdot 10^8$ frames) of training, is 76.70. Pixel control, the previous published state of the art in DMLab 30, improves final performance from to 77.59. PBL achieves 81.80, which is 5.31 times the gain with pixel control. By comparison, Simcore DRAW, the second best in fig. 3, attains 78.62—2.16 times the gain with pixel control, and thus only half of the gains obtained by using PBL.

While Simcore DRAW has been show to deliver strong improvements when training in single tasks, we believe that it may struggle with the large variety of tasks. The large diversity of images in multitask environments greatly increases the challenge of pixel prediction. Also, losses in pixel space can overlook details that are small but important for solving the tasks. Similarly, diverse images produced by different tasks are easy to distinguish from each other, meaning that CPC may not require a rich and informative representation to distinguish between the positive and negative examples, resulting in worse overall performance.

We show a per-task breakdown of PBL final performance relative to pixel control in fig. 4. We see that PBL is able to boost performance across the majority of the tasks, while achieving parity with pixel control on the rest of the tasks. This result shows that the representation learned by PBL encodes meaningful information which is useful to improve the performance of the agent across the board.

4.2. Scalability with the Horizon of Prediction

To have a better understanding of the effect of horizon on the performance of PBL we tried different horizon lengths for forward prediction in DMLab-30. Figure 5 shows that indeed, the performance improves monotonically as we predict farther into the future. However, as we increase the horizon, we get diminishing performance gains. Interestingly, predicting only one step into the future matches the performance of pixel control. This shows that multistep prediction enables PBL to encode crucial information for solving the tasks in its representation. To gain a better

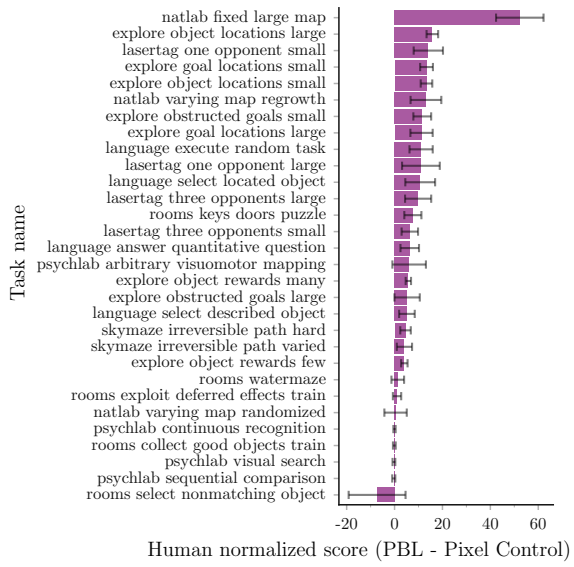


Figure 4. Difference in Human Normalized Score per Task between PBL and pixel control. Error bars denote 95% confidence intervals.

understanding on the effect of the prediction horizon, we investigate the case where we disable reverse prediction (see fig. 2), which means the latents are mere random projections of the observations. Interestingly, we found that increasing the horizon makes no difference to performance, so we only report the performance for horizon 20 in fig. 5. Overall, it is slightly worse than pixel control during training, but achieves the same final performance. These results emphasize that it is not enough to only try to predict farther into the future, but that it is crucial to predict meaningful latents about the future. This suggests that reverse prediction in PBL is able to learn a meaningful representation of future observations that does extends far into the future.

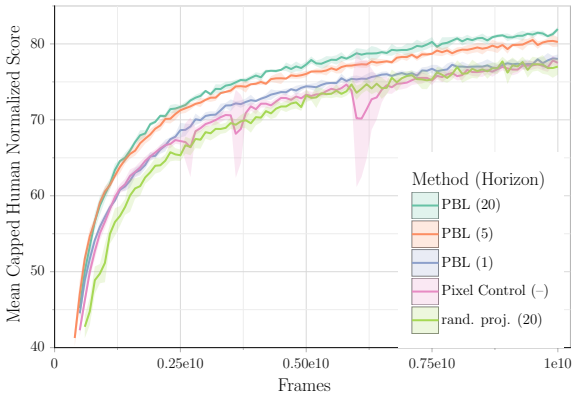


Figure 5. PBL Performance Across Forward Prediction Horizon, compared to pixel control and random projection.

4.3. Stability of PBL

From our experimental results, PBL is perfectly stable, and indeed does not collapse into a trivial solution. To further investigate this issue, we have tested a variant of PBL combined with random projection—we predict two separate latent observation embeddings where one is learned as usual and the other is a random projection. This variant grounds the forward prediction, which explicitly prevents collapsing. Interestingly, this variant achieves identical performance to PBL. This suggests that the training dynamics coupled with randomly initialized neural networks provide enough signal at the beginning to move away from collapse. As mentioned in section 3, we suspect that the forward and reverse predictions seem to form a virtuous cycle that continuously enriches both the compressed full history and the latent observations, instead of driving them to a trivial solution.

One subtle note is that for these experiments in DMLab-30, the agent state B_t is simultaneously trained with the RL loss as well as with PBL. However we have performed alternative experiments that also do not collapse where we have tried training two separate agent states where one is trained solely through PBL, and another is trained solely through RL but with the PBL agent state inputted as an extra observation (without passing gradients back). Also, in the upcoming section 4.6, the experiments have no RL at all, and the representations are purely trained using PBL.

While it might seem from these experiments that it is easy to avoid collapse, we have also tried to see what happens if instead of predicting future latent observations, we directly try predict future agent states using the forward prediction, i.e. predict B_{t+k} from B_t (without passing gradients through to the target). Unfortunately, even with the the additional signal from the RL loss, the representation collapses and the agent performance is completely flat.

4.4. Architecture Choice

In our experiments, we used a variant of the architecture used by Gregor et al. (2019); Song et al. (2020) (see table 1 in appendix A for full details). We compared the base RL method (PopArt-IMPALA with no auxiliary tasks) with this larger architecture, against the same method with the architecture used by Hessel et al. (2019).

Figure 6 shows the mean capped human normalized score of the base RL method with the two architectures, across four different independent runs. We chose to adopt the larger architecture because it gives a significant boost in overall performance.

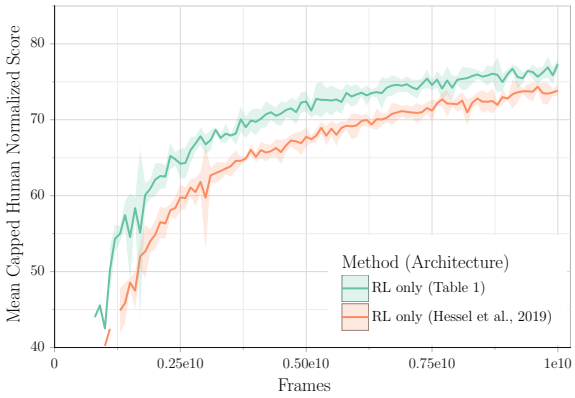


Figure 6. Mean capped human normalized score for compared architectures.

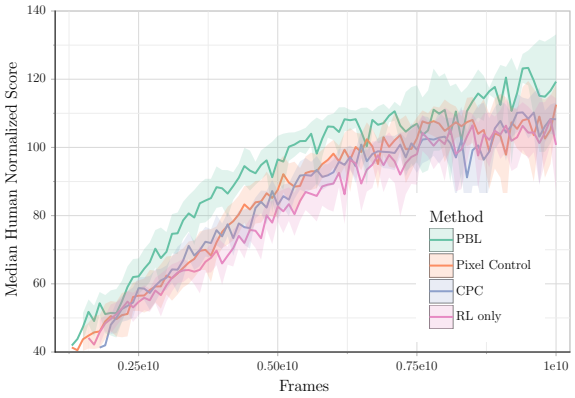


Figure 7. Median human normalized score for compared methods on Atari57.

4.5. Atari-57

We also ran PBL along with pixel control, CPC and RL-only on the Atari-57 (Bellemare et al., 2013) in a *multitask* setup (Espeholt et al., 2018; Hessel et al., 2019). The main goal of this experiment is to show that PBL is a general representation approach that can be useful in benchmarks other than DMLab 30. We choose Atari-57 since it is a standard benchmark in deep RL.

The result from Simcore DRAW is omitted as it is designed to output RGB images in the DMLab 30 format, and is incompatible with standard stacked, greyscale Atari frames format. We only changed two hyperparameters from DMLab 30: one is the entropy cost, which is now set to 0.01; the second is the unroll length, which is set to 30. In Espeholt et al. (2018), the unroll length was set to 20, but we increased it to 30 which facilitates multistep prediction in the case of PBL and CPC. We also clip the rewards to be between -1 and 1 which is standard for Atari-57 benchmark Espeholt et al. (2018).

Figure 7 shows the human normalized score (median across all 57 levels) for the compared approaches. We see that PBL is able to improve the overall performance in this benchmark. Interestingly, it seems that adding auxiliary tasks for representation learning only offers at most minimal improvements in performance. This may be due to Atari being essentially a fully observable domain as opposed to the partially observable nature of DMLab.

Table 10 shows the performance of different methods across the 57 Atari tasks with 95% confidence intervals. PBL outperforms the other methods, including pixel control, for seven tasks. In the other tasks, no method statistically outperforms PBL.

4.6. Decoding the PBL Representation

In this section, we try to shed some light on what information is being learned by the representation trained with PBL. We designed a simple DMLab-like 3D environment in which the agent is in a small room. There is a single red cube that spawns in a random location in the room. We use a fixed, uniformly random policy and only train the representation without any reinforcement learning.

Our criteria for a good representation is one that is able to encode the position of the red cube once the agent has seen it. To measure this, we train (in tandem with the agent's representation) a predictor from the agent's representation to the indicator of the position of the object in a top-down view of the environment. This position predictor does not affect the agent's representation (that is, no gradients pass from the predictor to the representation)—this type of predictor is known as a probe (Amos et al., 2018) or a glass-box evaluator (Guo et al., 2018). Figure 8 shows a first-person view of the room and a top-down grid-view of the environment indicating the ground-truth position of the object. We

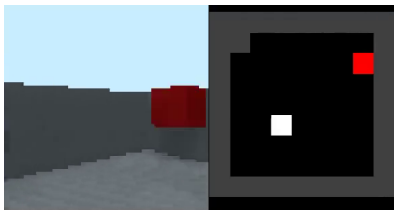


Figure 8. 3D room example: Agent's first-person view (left) and top-down grid-view indicating the object position (right).

compared PBL against the random projection version where reverse prediction is disabled. Figure 9 shows the loss for the object position predictor between these two methods, and clearly indicates that PBL is learning a representation that is significantly better at encoding the position of the

object. Taking a qualitative look at what happens during

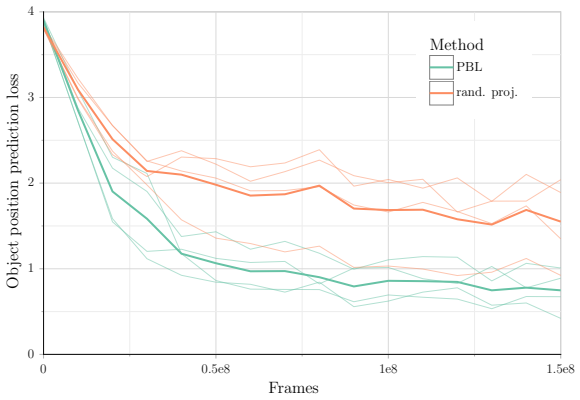


Figure 9. Object position prediction loss for random projection vs PBL. Light lines denote different independent runs.

individual episodes, we see that in videos of the glass-box evaluation, all methods are able to accurately localize the object right after the agent sees the object. However after the agent turns away, PBL's representation is able to remember the position of the object, while RP's representation quickly forgets. This shows an intimate connection between predicting the future and effectively compressing the past history—if the agent can predict that at some point far in the future, it will turn around and see the object again, then that means the representation has remembered the position of the object.

4.7. Implementation Overview

In this section, we give an overview of the most relevant implementation details. A full account of our method, with details on architecture choices and parameters, can be found in appendix A.

For the agent architecture (used in all methods), we use the same as Hessel et al. (2019) except for the following changes: 1) We increased the LSTM from one 256 unit hidden layer to two 512 unit hidden layers; 2) We changed the linear policy and value heads to have each an additional 512 unit hidden layer. This is in fact akin to Gregor et al. (2019) but with wider layers. We duplicate the LSTM architecture to use for the partial history unroll. Our larger architecture resulted in strictly better performance for all methods.

For PBL, we duplicate the agent's observation-processing architecture to use for the latent embedding (f in fig. 2). For the forward and reverse prediction networks (g and g' in fig. 2) we use MLPs with 2 hidden layers of 512 units each.

Most hyperparameters (e.g. PopArt hyperparameters) match the ones in Hessel et al. (2019). However, one significant difference is that we do not use population-based training (PBT). Instead, we picked the optimizer parameters and entropy penalty that worked best for the base RL agent (without any representation learning). For all representation learning methods, the only hyperparameter we tuned was the scale of the representation loss relative to the RL loss.

Computationally, predicting every future latent from one step ahead to the horizon from every timestep can be very expensive. We mitigate this issue by subsampling from which timesteps we do the forward prediction, and which future latents we predict (Gregor et al., 2019). This does not change the objective, and in practice can greatly speed up computation with minimal loss of performance.

5. Related Work

Littman et al. (2001); Singh et al. (2003) introduced predictive state representations (PSRs) and an algorithm for learning such representations. PSRs are akin to action-conditional predictions of the future where the prediction tasks are indicators of events on the finite observation space. The power of PSRs is the ability to act as a compact replacement for the state that can be constructed from observable quantities. That is, PSRs are compact sufficient statistics of the history. Inspired by this line of research, we focus on learning neural representations that are predictive of observations multiple timesteps into the future, conditioned on the agent's actions.

A number of other works investigated learning predictive representations (with action-conditional predictions of the future) to improve RL performance. Oh et al. (2015) use generated observations in addition to the actions when compressing partial histories. Amos et al. (2018) learned representations by maximizing the likelihood of future proprioceptive observations using a PreCo model, which interleaves an action-dependent (predictor) RNN with an observation dependent (corrector) RNN for compressing full histories, and uses the predictor RNN for the action-only sequences in partial histories (*cf.* fig. 1, where one RNN is used for the full histories, and another for the partial histories). Oh et al. (2017); Schrittwieser et al. (2019) learn to predict the base elements needed for Monte Carlo tree search conditioned on actions (Schrittwieser et al., 2019) or options (Oh et al., 2015): Rewards, values, logits of the search prior policy (in the case of Schrittwieser et al., 2019), and termination/discount (in the case of Oh et al., 2017). Guo et al. (2018); Moreno et al. (2018); Gregor et al. (2019) used an architecture similar to fig. 1 with a particular focus on learning representations of belief state.

Ha & Schmidhuber (2018); Hafner et al. (2019) used variational autoencoders (VAEs, Kingma & Welling, 2014; Gregor et al., 2015) to shape the full history RNN representation, and trained latent models from the partial histories

to maximize likelihood of the respective full histories. The Simcore DRAW technique (Gregor et al., 2019) is a demonstrably strong VAE-based representation learning technique for single task deep RL. DRAW trains the representation by trying to autoencode observations. In contrast PBL tries to learn meaningful features of observations related to important dynamics, rather than a generative model of pixel images.

Jaderberg et al. (2017) introduced pixel control as a representation learning task for reinforcement learning. This technique applies Q-learning for controlling the change in pixels, which can be considered as a technique for learning predictive representations based on future pixel changes.

(CPC Oord et al., 2018) is a powerful and lightweight alternative to generative models in representation learning. Similar to PBL, CPC operates in latent space. In contrast to our approach, which is based on predicting latent embeddings, CPC uses a contrastive approach to representation learning. (For an overview of CPC, see appendix A.5).

Other work that strives to learn a latent agent state representation include (DeepMDP Gelada et al., 2019) and (CRAR François-Lavet et al., 2019) that try to learn a transition model entirely in latent space. However DeepMDP relies on learning the reward function to ground its representation, making it very difficult to learn in sparse reward settings, and CRAR explicitly depends on a form of entropy maximisation to prevent collapse.

The idea of using bootstrapped latent embeddings to train the representations have been used recently in the context self-supervised learning of image representations (Grill et al., 2020). In this case Grill et al. (2020) show that one can improve the quality of the learned representation by using only one network for the latent embedding. The prediction task then consists of an *online* embedding network aiming at predicting the output of a *target* embedding network. The target network in this case is essentially a moving average of the previous copies of the online network. Thus Grill et al. (2020) remove the need to train separate embeddings of the observations.

6. Conclusion

We considered the problem of representation learning for deep RL agents in partially observable, multitask settings. We introduced Predictions of Bootstrapped Latents (PBL), a representation learning technique that provides a novel way of learning meaningful future latent observations through bootstrapped forward and reverse predictions. We demonstrated that PBL outperforms the state-of-the-art representation learning technique in DMLab-30.

Our results show that agents in multitask setting significantly benefit from learning to predict a *meaningful* representation of the future in response to its actions. This is highlighted in the fact that when we simply used random projections of future observations, there was no benefit in trying to predict farther into the future. On the other hand, by learning meaningful latent embeddings of observations, PBL continued to improve as we increased the horizon. Going forward, a promising direction would be to investigate ways to predict more relevant and diverse information, over long horizons.

Latent-space prediction is an easy way to combine multiple observation modalities. It would be worth seeing how far we can push the quality of learned representations by incorporating richer sources of observation. To name a few, proprioception in robotics and control tasks (Levine et al., 2016; Tassa et al., 2018), more sophisticated natural language, and even other sensory inputs like touch, smell, sound and temperature. In fact, the ability to learn representations from multimodal data makes PBL a good candidate for evaluation in other machine learning domains, beyond RL, such as the settings considered by Oord et al. (2018).

A promising extension to PBL may be to generalize the forward and reverse predictions from learning mean predictions (with squared loss) to learning generative models of latents. While this may not necessarily help in DMLab-30 where dynamics are deterministic, it may help in other settings where the dynamics are stochastic. Having a generative model may also be used to generate rollouts for Monte Carlo planning (similar to Schrittwieser et al., 2019).

We think it would be fruitful to evaluate PBL in a transfer learning setting. Since PBL improves performance virtually across all DMLab-30 tasks, it may have learned a representation that better captures the shared, latent structure across tasks, with the potential to generalize to new unseen tasks

Interesting future work also includes a more in-depth study of what predictive representations learn to encode, perhaps using a glass-box approach (Guo et al., 2018; Amos et al., 2018). In particular, it is worth finding out how much information about the POMDP *state* is captured and preserved by these representations.

Acknowledgements

We thank Hamza Merzic, Hubert Soyer and Satinder Singh Baveja for their support and useful feedback, and the ICML 2020 anonymous reviewers and program committee for their feedback and discussions.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard,

M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

Amos, B., Dinh, L., Cabi, S., Rothörl, T., Colmenarejo, S. G., Muldal, A., Erez, T., Tassa, Y., de Freitas, N., and Denil, M. Learning awareness models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

Aytar, Y., Pfaff, T., Budden, D., Paine, T. L., Wang, Z., and de Freitas, N. Playing hard exploration games by watching YouTube. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pp. 2935–2945, 2018.

Azar, M. G., Piot, B., Pires, B. A., Grill, J.-B., Altché, F., and Munos, R. World discovery models. *arXiv preprint arXiv:1902.07685*, 2019.

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., Schrittwieser, J., Anderson, K., York, S., Cant, M., Cain, A., Bolton, A., Gaffney, S., King, H., Hassabis, D., Legg, S., and Petersen, S. DeepMind lab, 2016.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Brunskill, E. and Li, L. Sample complexity of multi-task reinforcement learning. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, UAI'13, pp. 122–131, Arlington, Virginia, USA, 2013. AUAI Press.

Burda, Y., Edwards, H., Storkey, A. J., and Klimov, O. Exploration by random network distillation. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

Cassandra, A. R., Kaelbling, L. P., and Littman, M. L. Acting optimally in partially observable stochastic domains. In *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 2*, pp. 1023–1028, 1994.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. IMPALA: scalable distributed deep-RL with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 1406–1415, 2018.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

François-Lavet, V., Bengio, Y., Precup, D., and Pineau, J. Combined reinforcement learning via abstract representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3582–3589, 2019.

Gelada, C., Kumar, S., Buckman, J., Nachum, O., and Bellemare, M. G. Deepmdp: Learning continuous latent space models for representation learning. *arXiv preprint arXiv:1906.02736*, 2019.

Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., and Wierstra, D. DRAW: A recurrent neural network for image generation. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pp. 1462–1471, 2015.

Gregor, K., Jimenez Rezende, D., Besse, F., Wu, Y., Merzic, H., and van den Oord, A. Shaping belief states with generative environment models for RL. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 13475–13487. Curran Associates, Inc., 2019.

Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P. H., Buchatskaya, E., Doersch, C., Pires, B. A., Guo, Z. D., Azar, M. G., et al. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv preprint arXiv:2006.07733*, 2020.

Guo, Z. D., Azar, M. G., Piot, B., Pires, B. A., Pohlen, T., and Munos, R. Neural predictive belief representations. *arXiv preprint arXiv:1811.06407*, 2018.

Ha, D. and Schmidhuber, J. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pp. 2455–2467, 2018.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. In Chaudhuri, K. and Salakhutdinov, R.

(eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2555–2565, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 770–778, 2016a. doi: 10.1109/CVPR.2016.90.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016b.

Hessel, M., Soyer, H., Espeholt, L., Czarnecki, W., Schmitt, S., and van Hasselt, H. Multi-task deep reinforcement learning with PopArt. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3796–3803, 2019.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., Turck, F. D., and Abbeel, P. VIME: variational information maximizing exploration. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 1109–1117, 2016.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. Reinforcement learning with unsupervised auxiliary tasks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(1):1334–1373, January 2016. ISSN 1532-4435.

Littman, M. L., Sutton, R. S., and Singh, S. P. Predictive representations of state. In *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pp. 1555–1561, 2001.

Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., Kumaran, D., and Hadsell, R. Learning to navigate in complex environments. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

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. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529, 2015.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 1928–1937, 2016.

Moreno, P., Humplik, J., Papamakarios, G., Pires, B. A., Buesing, L., Heess, N., and Weber, T. Neural belief states for partially observed domains. In *NeurIPS 2018 workshop on Reinforcement Learning under Partial Observability*, 2018.

Nair, V. and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pp. 807–814, 2010.

Oh, J., Guo, X., Lee, H., Lewis, R. L., and Singh, S. P. Action-conditional video prediction using deep networks in Atari games. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 2863–2871, 2015.

Oh, J., Singh, S., and Lee, H. Value prediction network. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 6118–6128, 2017.

Oord, A. v. d., Li, Y., and Vinyals, O. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pp. 2778–2787, 2017.

Puigdomènech Badia, A., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., and Blundell, C. Never

give up: Learning directed exploration strategies. In *International Conference on Learning Representations*, 2020.

Rothfuss, J., Lee, D., Clavera, I., Asfour, T., and Abbeel, P. Promp: Proximal meta-policy search. *arXiv preprint arXiv:1810.06784*, 2018.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of Go without human knowledge. *nature*, 550(7676):354–359, 2017.

Singh, S. P., Littman, M. L., Jong, N. K., Pardoe, D., and Stone, P. Learning predictive state representations. In *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pp. 712–719, 2003.

Song, H. F., Abdolmaleki, A., Springenberg, J. T., Clark, A., Soyer, H., Rae, J. W., Noury, S., Ahuja, A., Liu, S., Tirumala, D., Heess, N., Belov, D., Riedmiller, M., and Botvinick, M. M. V-MPO: On-policy maximum a posteriori policy optimization for discrete and continuous control. In *(to appear) 8th International Conference on Learning Representations, ICLR 2020*, 2020.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Szepesvári, C. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103, 2010.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. DeepMind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575 (7782):350–354, 2019.

Zhang, M., Vikram, S., Smith, L., Abbeel, P., Johnson, M. J., and Levine, S. SOLAR: Deep structured representations for model-based reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pp. 7444–7453, 2019.

A. Implementation Details

In this section, we describe our implementation in more detail, especially architecture and parameter choices. For hyperparameters picked from sweeps, we describe our protocol and the values considered. We also indicate software libraries used and describe our protocol for plotting.

A.1. Agent Architecture

Table 1 collects the different architecture parameters for the "main agent networks". These include the observation processing, the RNN for full histories, and the MLPs for value estimates and policies. Our choices follow Hessel et al. (2019) with a few exceptions that are given in bold in table 1. The differences are limited to increases in network size. The RNNs used are LSTMS (Hochreiter & Schmidhuber, 1997) and the networks used for image processing are ResNets (He et al., 2016a) The DMLab30 observations we use and the way we process them is the same as Hessel et al. (2019) (with the exception of

Hyperparameter	Value
Convolutional Stack	
Number of sections	3
Channels per section	(16, 32, 32)
Activation function	ReLU
ResNet section	
Number of sections	1 / 3x3 / 1
Max-Pool	1 / 3x3 / 2
Conv	2 / 3x3 / 1
Skip	Identity
Conv	2 / 3x3 / 1
Skip	Identity
Post-ResNet	
Fully connected layer	**512**
Output activation	ReLU
Language preprocessing	
Word embeddings	20
Sentence embedding	LSTM 64
Vocabulary size	1000
Max. instruction length	15
Other input preprocessing	
Action	one-hot
Reward	see eq. (5)
Full history compression	
h_f	LSTM **(512, 512)**
Skip connections	**Yes**
RL heads	
Hidden layer activation	ReLU
Value	**(512, 30)**
Policy (softmax)	**(512, 15)**

Table 1. Main architecture parameters. These are the same as Table 7 of Hessel et al. (2019), except for parameters in bold.

the size of the post-resnet layer). The agent's first-person view is processed through a ResNet (He et al., 2016b), which is then flattened and input to a fully connected layer, with ReLU (Nair & Hinton, 2010) output activations. The language instructions are processed as follows. Each word is embedded into a 20-dimensional vector, where the vocabulary size is

100 and the embedding is learnable. The first 15 embedded words (or less if the instruction is shorter) are fed into an LSTM with 64 hidden units. The outputs are summed into the final instruction embedding. The previous action (which resulted in the given first-person view) is encoded as a one-hot, out of the 15 possible actions (from the action set given in table 7 of Hessel et al. (2019)). The reward observed along with the first-person view (resulting from the "previous action") is transformed according to

$$r \mapsto 0.3 \cdot \tanh(r/5)_- + 1.5 \cdot \tanh(r/5)_+, \tag{5}$$

where $(\cdot)_-$ denotes the negative part and $(\cdot)_+$ the positive part. The processed first-person view, instruction, previous action and reward are concatenated and fed as input to the full history RNN h_f. The output of the full history RNN is input to the value estimate MLP, which has 30 outputs because PopArt maintains one value estimate per task. Note that PopArt uses the task index for updating these estimates Hessel et al. (2019). This task index is not an input to any network. The output of the full history RNN is also input to the policy head, which takes a softmax over 15 logits corresponding to the actions in the discretized action set introduced by Hessel et al. (2019).

A.2. Action-conditional predictions of the future

The procedure for compressing partial histories is the same for PBL, CPC and Simcore DRAW, and is detailed in table 2. Our implementation reproduces the Simcore of Gregor et al. (2019).

Hyperparameter	Value
h_p	`LSTM (512, 512)`
Future prediction horizon	20
Time subsampling	to size 6
Batch subsampling	None
Future subsampling	to size 2

Table 2. Parameters for action-conditional predictions of the future.

The compressed full histories are built by unrolling h_f (see table 1). Specifically, when B_t is used as an input to MLPs, we use the LSTM output. When B_t is used as an input to an RNN (for unrolling h_f and h_p), we use the LSTM state. The LSTM for partial histories has the same architecture as the full history LSTM, and hence the same state size. So we can generate $B_{t,k}$ by unrolling h_p with the action sequence $A_t, \ldots, A_{t+k-1}$ (one-hot encoded), from the initial state provided by h_f, namely the state resulting from the RNN update $h_\mathrm{f}(B_t, O_t, A_{t-1})$.

Our minibatches have size $\mathtt{T} \times \mathtt{B}$, where $\mathtt{T}$ is the sequence length and $\mathtt{B}$ is the batch size (see table 8). In addition to the sequences of length $\mathtt{T}$, we have the $\mathtt{B}$ LSTM states of the main architecture at the start of these sequences (Espeholt et al., 2018).

We do not use the initial states as compressed full histories, so there are $\mathtt{T} \times \mathtt{B}$ possible compressed full histories to be used. Before we unroll h_p, we subsample the time indices and use only $6 \times \mathtt{B}$ compressed full histories (the randomly selected time indices are the same across the batch axis, but change in each minibatch). From these $6 \times \mathtt{B}$ initial states, we generate $\mathtt{H} \times 6 \times \mathtt{B}$ partial histories. We used $\mathtt{H} = 20$, except in the scalability experiments, where we varied $\mathtt{H}$ across $(1, 5, 20)$. We then subsampled these partial histories, taking $2 \times 6 \times \mathtt{B}$ to be used for representation learning (PBL, CPC or DRAW). The randomly selected future indices are the same across the batch and time axis, but change in each minibatch.

A.3. Picking Hyperparameters for Representation Learning

The weight of the representation learning losses for all methods (PBL, Simcore DRAW, CPC and pixel control) were chosen for top performance for each method, chosen among powers of 10 from -2 to 2.

The prediction horizon (how far we predict into the future) for PBL, Simcore DRAW, CPC has been fixed at 20. We have tried changing this horizon to 1 and to 5, for all methods, similar to the study for PBL presented in section 4. With the exception of PBL, predicting up to twenty steps into the future did not yield improvements compared to only up to five steps, whereas predicting up to five steps into the future was marginally better (for Simcore DRAW) or the same (for CPC) than using a horizon of one step. In all cases, therefore, 20 was a best performing choice for the horizon.

A.4. PBL

The representation learning in PBL is setup as follows. The forward prediction, from partial histories to future latents, uses the procedure described in appendix A.2. The reverse prediction is from latents to full histories without subsampling, over the whole $\texttt{T} \times \texttt{B}$ minibatch.

Table 3 gives the breakdown of parameters for PBL. We use the architecture detailed in table 1 to process observations

Hyperparameter	Value
Latent Embedding	
f	same as in table 1
Shared f	`No`
Forward prediction	
Loss weight	`1`
g	`(512, 512, 592)`
Normalize latents (targets)	`Yes`
Normalize predictions	`Yes`
Prediction regularizer	$v \mapsto 0.02 \cdot (\|v\|_2^2 - 1)^2$
Reverse prediction	
Loss weight	`1`
g'	`(512, 512, 1024)`
Normalize latents (g' inputs)	`Yes`
Normalize predictions	`No`
Prediction regularizer	None
Latent regularizer	$v \mapsto 0.02 \cdot (\|v\|_2^2 - 1)^2$
All hidden layer activations	`ReLU`

Table 3. PBL parameters.

(f), but in a separate network from the one used by the agent architecture. The output of f is a vector of size 592 (512 for first-person view, 64 for language, 15 for the previous action and 1 for the reward), which is the latent embedding of the observation. The output of g is a vector of size 592, the output size of f. The output of g' is a vector of size 1024, the output size of h_{f} (which has two layers of size 512 with skip connections, as given in table 1).

The latent embeddings are always normalized with $v \mapsto \frac{v}{\|v\|_2 + 10^{-8}}$, both when used as targets in the forward prediction and as inputs to g' in the reverse prediction. For the forward prediction, the predictions (outputs of g) are also normalized in this way, and regularized to have to have unit ℓ_2 norm (the regularization is applied before normalization). In the reverse prediction, the embeddings are regularized (outputs of f) to have unit ℓ_2 norm (again, the regularization is applied before normalization). The compressed histories (outputs of h_{f} and h_{p}) and predictions (outputs of g') are neither normalized nor regularized.

A.5. CPC

CPC aims to learn a discriminator between jointly distributed random variables and random variables sampled from the marginals. A diagram of CPC is given in fig. 10, illustrating discrimination between jointly distributed partial histories and observations, from independent ones. The parameters are given in table 4. To sample negative examples for timestep $t + k$, we take twenty observations at random, with replacement, from the minibatch, uniformly over the indices different from $t + k$. The losses over positive and negative examples are balanced—the total loss is the binary (sigmoid) cross-entropy over the positive example plus the mean over all negative examples of the binary (sigmoid) cross-entropy.

A.6. DRAW

Table 5 outlines parameters for DRAW and GECO. Parameters not reported in tables 1, 2 and 5 are the same as reported by Gregor et al. (2019). our implementation of DRAW uses the same architecture as fig. 1 to compress histories, but trains h_f

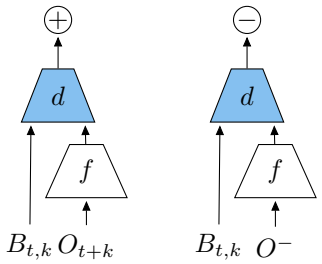


Figure 10. Architecture diagram for contrastive predictive coding.

Hyperparameter	Value
Loss weight	`0.1`
Negative examples	`20`
d	`(512, 512, 1)`
f	see table 1
Shared f	`Yes`

Table 4. CPC parameters.

and h_p by trying to autoencode observations. Taking $Z'_{t+k} \sim P_\phi(Z'_{t+k}|O_{t+k})$ to be the DRAW latent with parameters ϕ, the technique maximizes the (parametric) likelihood $P_\phi(O_{t+k}|Z'_{t+k})$ subject to the posterior $P_\phi(Z'_{t+k}|O_{t+k})$ being close (in KL divergence) to the prior $P_\phi(Z'_{t+k}|B_{t+k})$.

Hyperparameter	Value
Loss weight	**`1e-1`**
GECO κ	`0.015`

Table 5. DRAW loss weight. See Gregor et al. (2019) for full details, and tables 1 and 2.

A.7. Pixel Control

Table 6 shows the parameters for pixel control. Parameters not reported as as used by Hessel et al. (2019) (see section "Pixel Control" in their appendix). Differently from Hessel et al. (2019), we compute the total loss over 4×4 cells by taking the mean loss instead of the sum of losses.

A.8. VTrace and PopArt

The RL and other training parameters are given in table 8 and PopArt in table 7. We use the same parameters as Hessel et al. (2019) except where designated in bold. We did not use population-based training (PBT), so we tuned the entropy cost with a over $\{0.01, 0.005, 0.001, 0.0005\}$. We chose the parameter that worked best for the base RL algorithm (IMPALA PopArt) with no representation learning, and used it for all the representation learning techniques.

A.9. Libraries

We used TensorFlow (Abadi et al., 2015) and Sonnet (`github.com/deepmind/sonnet`). TRFL (`github.com/deepmind/trfl`) provides the pixel control implementation we used. In addition to implementations of convolutional layers, LSTMs and MLPs, Sonnet also provides the implementation we used for the language embedding.

A.10. Plotting

For our plots, we binned the observed per-episode returns into bins of size 10^8 frames (centered at multiples of 10^8) and took the mean performance in each bin. These averaged returns were then mapped to the mean human normalized score, capped where applicable, and then averaged over tasks. At this point we computed any confidence intervals if applicable,

Hyperparameter	Value
Loss weight	**1e-1**
Total loss	**mean** over cells

Table 6. Pixel control parameters. See Hessel et al. (2019) for full details. Different parameters are designated in bold.

Hyperparameter	Value
Statistics learning rate	3e-4
Scale lower bound	1e-4
Scale upper bound	1e6

Table 7. PopArt parameters, the same as used by Hessel et al. (2019).

Hyperparameter	Value
RL	
Unroll length	100
Batch size	32
γ	0.99
V-trace λ	**0.99**
Baseline loss weight	**0.4**
Entropy cost	**5e-3**
DMLab 30	
First-person view (height, width)	(72, 96)
Action repeats	4
Optimizer	**Adam**
Learning rate	**1e-4**
β_1	**0**
β_2	**0.999**
ϵ	**1e-6**

Table 8. RL and other training parameters. These match the parameters used by Hessel et al. (2019) with differences in bold.

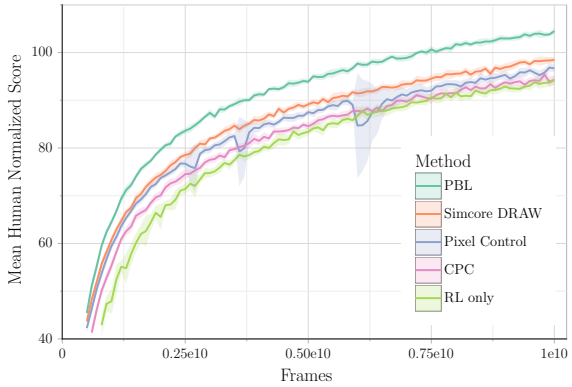
and finally averaged over independent runs. For plots with confidence bands and table 9, we assumed the mean estimates follow Student's t distribution, and reported 95% confidence intervals (two-sided).

The mean estimates in fig. 4 and table 9 were taken by averaging the performances on the final 500M frames of training. The confidence intervals in fig. 4 were computed by adding the confidence intervals of the estimates for PBL and pixel control.

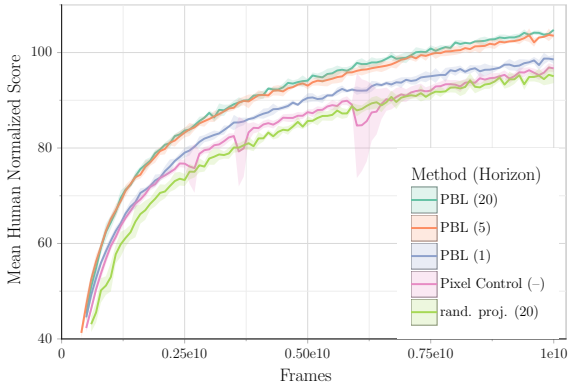
B. Supplementary results

B.1. DMLab-30

In this section, we present some additional results to provide more context to our results in DMLab-30. Table 9 shows the final performance breakdown across all tasks in DMLab-30 for all compared methods. We also show the uncapped mean human normalized score for different methods (fig. 11a), for PBL across different horizons (fig. 11b), and for the architecture comparison (fig. 12).



(a) Mean uncapped human normalized score for compared methods.



(b) Mean uncapped human normalized score for PBL across different prediction horizons.

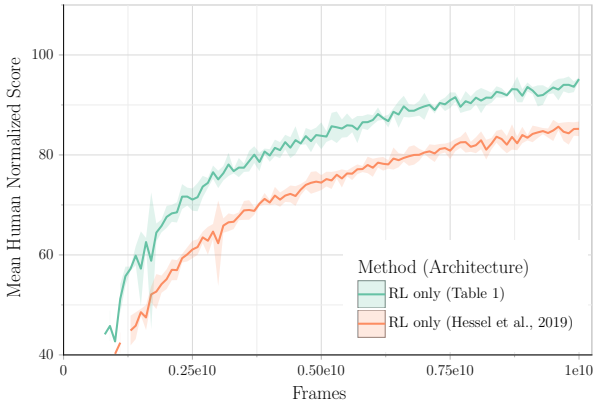


Figure 12. Mean uncapped human normalized score for compared architectures.

	RL only	Pixel Control	CPC	SC DRAW	PBL
explore goal locations large	77.0 ± 3.4	79.9 ± 2.2	76.9 ± 1.3	83.0 ± 1.4	$\mathbf{90.7 \pm 3.0}$
explore goal locations small	123.6 ± 3.2	124.6 ± 1.4	124.5 ± 2.1	130.8 ± 1.4	$\mathbf{137.7 \pm 1.3}$
explore object locations large	93.3 ± 3.1	97.5 ± 1.5	94.2 ± 1.5	102.1 ± 1.3	$\mathbf{112.9 \pm 0.9}$
explore object locations small	106.0 ± 2.6	111.3 ± 1.4	108.7 ± 1.2	115.5 ± 0.9	$\mathbf{124.8 \pm 1.0}$
explore object rewards few	57.9 ± 1.1	60.9 ± 0.7	59.8 ± 0.7	61.5 ± 0.9	$\mathbf{64.9 \pm 0.5}$
explore object rewards many	59.7 ± 0.7	61.9 ± 0.6	61.3 ± 0.7	63.2 ± 0.4	$\mathbf{67.7 \pm 0.5}$
explore obstructed goals large	72.5 ± 3.8	79.9 ± 2.5	72.4 ± 2.3	78.9 ± 2.1	84.8 ± 2.7
explore obstructed goals small	114.8 ± 3.7	121.9 ± 2.4	118.7 ± 2.3	124.2 ± 2.0	$\mathbf{133.3 \pm 1.2}$
language answer quantitative question	159.5 ± 6.1	159.0 ± 2.5	158.9 ± 2.6	162.6 ± 1.2	165.0 ± 1.4
language execute random task	139.2 ± 2.9	141.2 ± 3.1	138.8 ± 3.0	145.4 ± 2.1	$\mathbf{152.5 \pm 1.5}$
language select described object	152.0 ± 1.1	154.5 ± 1.9	154.2 ± 1.1	158.2 ± 0.7	159.7 ± 1.3
language select located object	241.8 ± 6.9	252.8 ± 4.9	243.9 ± 5.6	262.0 ± 1.5	263.9 ± 1.1
lasertag one opponent large	101.3 ± 4.3	112.4 ± 4.6	107.9 ± 4.7	111.7 ± 2.2	$\mathbf{124.0 \pm 3.2}$
lasertag one opponent small	163.2 ± 4.0	168.5 ± 3.5	166.1 ± 3.8	171.2 ± 2.6	$\mathbf{182.5 \pm 2.8}$
lasertag three opponents large	152.4 ± 5.6	158.1 ± 2.7	153.6 ± 3.5	158.0 ± 1.7	$\mathbf{167.2 \pm 2.1}$
lasertag three opponents small	143.9 ± 3.8	148.9 ± 1.9	145.0 ± 3.2	147.4 ± 1.7	$\mathbf{154.9 \pm 1.5}$
natlab fixed large map	41.7 ± 3.1	39.4 ± 2.0	38.9 ± 2.2	49.6 ± 4.1	$\mathbf{93.3 \pm 7.9}$
natlab varying map randomized	84.4 ± 2.8	85.2 ± 2.8	82.9 ± 3.4	82.5 ± 2.9	86.0 ± 2.0
natlab varying map regrowth	90.2 ± 3.9	87.6 ± 4.3	88.4 ± 4.4	92.4 ± 2.1	$\mathbf{100.4 \pm 1.9}$
psychlab arbitrary visuomotor mapping	38.3 ± 5.0	41.7 ± 4.0	43.4 ± 2.9	43.9 ± 3.1	47.9 ± 3.0
psychlab continuous recognition	52.2 ± 0.3	52.2 ± 0.3	52.2 ± 0.2	52.1 ± 0.3	52.1 ± 0.2
psychlab sequential comparison	75.6 ± 0.6	76.0 ± 0.3	75.9 ± 0.9	75.8 ± 0.4	75.7 ± 0.3
psychlab visual search	101.2 ± 0.4	101.3 ± 0.5	100.3 ± 1.0	101.1 ± 0.2	101.0 ± 0.1
rooms collect good objects test	95.0 ± 1.0	96.6 ± 0.3	95.7 ± 0.3	96.5 ± 0.3	96.6 ± 0.2
rooms exploit deferred effects test	36.9 ± 1.2	36.4 ± 1.1	37.7 ± 0.6	37.3 ± 0.8	37.7 ± 0.6
rooms keys doors puzzle	48.6 ± 1.5	49.2 ± 1.7	47.8 ± 1.9	55.1 ± 1.3	56.2 ± 1.9
rooms select nonmatching object	62.9 ± 11.1	65.3 ± 9.2	70.6 ± 8.9	65.7 ± 9.2	60.8 ± 5.8
rooms watermaze	42.2 ± 2.7	43.3 ± 1.6	42.1 ± 1.7	45.6 ± 2.1	45.0 ± 1.4
skymaze irreversible path hard	25.5 ± 2.2	27.5 ± 1.1	24.6 ± 1.6	27.8 ± 1.1	$\mathbf{32.2 \pm 1.1}$
skymaze irreversible path varied	49.0 ± 2.0	49.2 ± 2.2	45.0 ± 2.5	48.1 ± 1.5	$\mathbf{53.3 \pm 1.3}$

Table 9. Human normalized scores across tasks in the last 5% of training—the last 500M out of 10B frames. Statistically significant performance improvements in bold.

B.2. Atari-57

	RL only	Pixel Control	CPC	PBL
alien	15.8 ± 1.3	16.3 ± 1.3	17.2 ± 0.7	18.9 ± 2.0
amidar	8.6 ± 0.9	9.4 ± 0.8	9.5 ± 0.5	$\mathbf{12.0 \pm 1.2}$
assault	524.0 ± 47.5	658.2 ± 44.2	557.1 ± 48.4	764.7 ± 105.7
asterix	29.7 ± 8.4	32.8 ± 9.1	30.9 ± 4.3	43.5 ± 6.1
asteroids	3.6 ± 0.1	3.7 ± 0.1	3.8 ± 0.1	4.1 ± 0.2
atlantis	5033.8 ± 328.2	5180.9 ± 404.8	4761.6 ± 362.4	5448.9 ± 729.6
bank heist	157.9 ± 11.2	167.7 ± 6.3	166.9 ± 10.1	169.3 ± 21.3
battle zone	73.1 ± 5.9	78.3 ± 1.7	80.3 ± 4.9	88.5 ± 8.1
beam rider	13.8 ± 1.2	21.3 ± 1.1	13.0 ± 0.6	19.2 ± 1.8
berzerk	14.6 ± 1.0	15.4 ± 0.9	14.8 ± 0.7	16.2 ± 1.6
bowling	3.4 ± 1.4	4.8 ± 2.0	5.4 ± 1.2	7.3 ± 1.5
boxing	781.9 ± 14.1	765.9 ± 45.2	793.7 ± 8.5	807.3 ± 2.0
breakout	934.1 ± 68.6	1034.1 ± 58.1	997.5 ± 43.2	$\mathbf{1203.0 \pm 68.0}$
centipede	-7.6 ± 0.8	-8.5 ± 0.8	-6.8 ± 0.9	$\mathbf{-2.6 \pm 1.8}$
chopper command	105.0 ± 6.2	107.7 ± 7.7	111.2 ± 6.6	114.5 ± 13.7
crazy climber	280.5 ± 15.8	299.9 ± 12.3	293.4 ± 13.6	292.0 ± 29.3
defender	278.0 ± 41.5	285.7 ± 36.9	342.1 ± 41.9	347.1 ± 60.4
demon attack	549.9 ± 101.4	750.6 ± 108.1	510.3 ± 54.0	786.1 ± 148.7
double dunk	691.7 ± 108.6	815.8 ± 148.7	745.1 ± 38.8	862.9 ± 162.2
enduro	113.9 ± 11.3	129.5 ± 4.8	121.9 ± 8.0	141.8 ± 14.6
fishing derby	179.5 ± 12.1	182.7 ± 3.9	186.8 ± 9.8	187.0 ± 23.0
freeway	103.4 ± 3.5	108.7 ± 0.6	107.5 ± 1.8	102.8 ± 12.6
frostbite	4.4 ± 0.1	4.7 ± 0.2	4.6 ± 0.2	4.6 ± 0.3
gopher	935.0 ± 156.1	989.4 ± 277.7	838.7 ± 142.4	1080.0 ± 235.3
gravitar	44.7 ± 3.7	48.0 ± 3.7	47.5 ± 4.1	$\mathbf{54.6 \pm 1.0}$
hero	37.1 ± 1.9	40.1 ± 0.8	39.2 ± 1.0	38.8 ± 4.6
ice hockey	149.7 ± 6.0	159.2 ± 5.8	152.9 ± 4.6	158.6 ± 18.0
jamesbond	141.2 ± 13.0	166.4 ± 11.3	153.4 ± 10.0	155.4 ± 28.1
kangaroo	50.5 ± 2.6	53.2 ± 1.4	67.2 ± 27.7	112.9 ± 42.3
krull	669.1 ± 26.4	704.9 ± 14.4	698.0 ± 9.3	719.7 ± 42.8
kung fu master	104.0 ± 6.7	112.4 ± 7.4	112.1 ± 9.4	$\mathbf{150.5 \pm 17.4}$
montezuma revenge	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
ms pacman	24.2 ± 1.2	25.7 ± 1.0	26.8 ± 0.8	28.2 ± 3.3
name this game	137.7 ± 13.2	153.7 ± 17.9	110.4 ± 17.6	165.9 ± 10.2
phoenix	111.9 ± 8.9	134.7 ± 14.6	124.1 ± 7.5	143.3 ± 13.7
pitfall	3.2 ± 0.1	3.3 ± 0.1	3.3 ± 0.0	3.3 ± 0.1
pong	108.2 ± 5.1	109.7 ± 8.2	114.5 ± 2.5	116.9 ± 0.4
private eye	0.1 ± 0.0	0.1 ± 0.0	0.1 ± 0.0	0.1 ± 0.0
qbert	37.9 ± 11.5	47.9 ± 15.2	16.4 ± 6.5	72.8 ± 20.3
riverraid	27.4 ± 5.5	37.3 ± 5.3	18.9 ± 6.5	23.0 ± 8.7
road runner	321.0 ± 40.2	382.0 ± 44.2	359.4 ± 57.5	460.2 ± 81.3
robotank	218.8 ± 24.6	282.5 ± 13.6	207.6 ± 9.2	274.1 ± 32.6
seaquest	5.0 ± 0.4	6.3 ± 1.5	5.8 ± 0.2	8.9 ± 2.6
skiing	17.4 ± 2.5	18.1 ± 1.9	13.5 ± 2.7	21.0 ± 2.9
solaris	8.9 ± 0.9	8.0 ± 0.7	10.0 ± 1.1	9.1 ± 0.9
space invaders	40.2 ± 3.9	48.3 ± 6.8	41.3 ± 2.9	$\mathbf{106.1 \pm 13.0}$
star gunner	377.1 ± 37.3	436.4 ± 47.7	415.0 ± 37.9	477.5 ± 62.8
surround	73.9 ± 7.7	86.8 ± 10.2	83.4 ± 8.8	94.2 ± 12.4
tennis	287.6 ± 10.7	275.1 ± 28.9	255.3 ± 36.8	231.4 ± 48.2
time pilot	849.3 ± 59.3	910.9 ± 79.5	783.6 ± 71.0	999.7 ± 148.4
tutankham	144.1 ± 6.4	148.0 ± 6.5	148.8 ± 5.0	154.3 ± 13.2
up n down	2388.1 ± 238.8	2573.1 ± 150.9	2579.7 ± 142.9	2832.4 ± 284.6
venture	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
video pinball	1139.6 ± 101.6	1121.1 ± 79.4	1072.3 ± 109.4	1234.1 ± 185.6
wizard of wor	93.7 ± 10.0	107.6 ± 6.9	96.7 ± 6.5	108.5 ± 14.6
yars revenge	19.3 ± 2.3	36.1 ± 14.3	24.7 ± 2.4	$\mathbf{83.8 \pm 17.9}$
zaxxon	136.5 ± 12.2	176.0 ± 10.9	159.3 ± 7.9	169.4 ± 27.9

Table 10. Human normalized scores across tasks in the last 5% of training—the last 500M out of 10B frames. Statistically significant performance improvements in bold.