P4O: EFFICIENT DEEP REINFORCEMENT LEARNING WITH PREDICTIVE PROCESSING PROXIMAL POLICY OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Advances in reinforcement learning (RL) often rely on massive compute resources and remain notoriously sample inefficient. In contrast, the human brain is able to efficiently learn effective control strategies using limited resources. This raises the question whether insights from neuroscience can be used to improve current RL methods. Predictive processing is a popular theoretical framework which maintains that the human brain is actively seeking to minimize surprise. We show that recurrent neural networks which predict their own sensory states can be leveraged to minimise surprise, yielding substantial gains in cumulative reward. Specifically, we present the Predictive Processing Proximal Policy Optimization (P4O) agent; an actor-critic reinforcement learning agent that applies predictive processing to a recurrent variant of the PPO algorithm by integrating a world model in its hidden state. P4O significantly outperforms a baseline recurrent variant of the PPO algorithm on multiple Atari games using a single GPU. It also outperforms other state-of-the-art agents given the same wall-clock time and exceeds human gamer performance on Seaquest, which is a particularly challenging environment in the Atari domain. Altogether, our work underscores how insights from the field of neuroscience may support the development of more capable and efficient artificial agents.

1 INTRODUCTION

The goal of reinforcement learning (RL) is to learn effective control policies based on scalar reward signals provided by the environment. Temporally sparse and delayed reward signals make training an RL agent notoriously slow and unstable. Over the past decade, however, RL has been successfully applied to increasingly complex tasks. This progress is afforded by the use of deep neural networks in combination with algorithmic advances in RL. Research in RL has also been accelerated by the availability of simulation benchmarks that allow rapid testing and comparison of RL algorithms (Bellemare et al., 2013).

The current state of the art in RL is achieved by distributed multi-GPU approaches such as MuZero (Schrittwieser et al., 2020), utilizing Monte Carlo tree search (MCTS), Agent57 (Badia et al., 2020), combining a large number of innovative approaches into a single model, and GoExplore (Ecoffet et al., 2019), which keeps an archive of trajectories to force exploration of promising unknown states. However, the high computational cost of these approaches make them infeasible in many research settings.

On the other hand, methods have also been developed for efficient training on single GPU contexts. These deploy a myriad of strategies from across the range of modern RL research. For example, Rainbow (Hessel et al., 2018) integrates a number of recent developments from Q-learning into a single model, DreamerV2 (Hafner et al., 2020) relies upon world models combined with 'imagined' outcomes through predictions of future states, and IQN (Dabney et al., 2018) efficiently integrates distributional RL techniques with deep Q-learning. These approaches show promise, however RL remains a sample inefficient and expensive paradigm.

Motivated by the efficiency with which our own brain is able to solve challenging control problems, we ask if we can use brain-inspired algorithms to go beyond the performance of these existing RL

approaches. *Predictive coding*, an established theory of sensory information processing in the brain (Srinivasan et al., 1982; Mumford, 1992; Friston, 2005; Clark, 2013; Ciria et al., 2021), proposes that higher-level brain areas attempt to predict the activation of lower-level brain areas and use this prediction to inhibit incoming activity. The remaining signal, a prediction error, can be seen as a measure of surprise of the internal model of the world that generated the predictions. This surprise signal can be used to adjust an agent's behavior and update its internal model of the world. A number of studies have contributed to the growing experimental evidence for this theory (Alink et al., 2010; Näätänen et al., 2001; Summerfield et al., 2008; Squires et al., 1975; Hupé et al., 1998; Murray et al., 2002; Rao et al., 2016; Kok et al., 2012; Ekman et al., 2017; De Lange et al., 2018; Dijkstra et al., 2020; Schwiedrzik & Freiwald, 2017), while others reproduced experimentally observed phenomena in explicit computational models of predictive coding (Rao & Ballard, 1999; Lee & Mumford, 2003; Friston, 2005; 2010).

We hypothesize that simultaneously minimizing sensory surprise and maximizing return (expected cumulative reward) yields more effective and biologically plausible control algorithms. Specifically, we suppose that by minimizing surprise the agent is forced to learn an internal model which may facilitate learning of more effective control laws. To test this hypothesis, we leverage recurrent neural network (RNN) models, commonly used to to capture temporal, discrete-time, state evolutions for machine learning and neuroscience (Jordan, 1990; Elman, 1990; Sussillo, 2014; Maass, 2016; Vyas et al., 2020). We next investigate whether RNNs that implement predictive processing can learn to efficiently and effectively solve complex RL tasks. To this end, we employ a subset of environments in the Human Atari benchmark, which consists of 57 games where the goal is to beat human-level performance (Bellemare et al., 2013).

Our results show that game performance of a recurrent variant of the Proximal Policy Optimization (PPO) algorithm (Schulman et al., 2017) is strongly improved by including a predictive processing mechanism, yielding a novel Predictive Processing Proximal Policy Optimization (P4O) algorithm. P4O achieves results that are competitive with the current state of the art while using only a fraction of the computational resources.

2 METHODS

2.1 P4O ARCHITECTURE

Our aim is to investigate how predictive processing can aid learning of more effective control policies. For this purpose we augment PPO with a recurrent network and loss function that incorporate a predictive processing error. This approach is motivated by the work of Ali et al. (2021) which demonstrated that error and prediction units as proposed in predictive coding may naturally emerge in energy-constrained neural networks that implement an efficient coding constraint (Barlow, 1961).

A P4O agent consists of three components: an encoder model, a recurrent neural network, and an actor-critic model. The encoder transforms a sensory input into a latent representation. We use a multi-layer CNN with residual connections for this purpose. The recurrent network consists of a modified LSTM layer that incorporates a predictive coding mechanism. The actor-critic component contains one fully connected layer for action selection and one for state value prediction. The agent's objective function is enhanced by a term minimizing the prediction error. The overall architecture is shown in Figure 1. The following subsections describe each of these components in more detail.

2.1.1 ENCODER MODEL

Our encoder is a residual convolutional neural network inspired by Espeholt et al. (2018). It consists of four layer groups; each group contains a convolutional layer, a max-pooling layer and two residual blocks with two convolution layers each. The final group is followed by a fully-connected layer with 512 neurons and a $\tanh$ nonlinearity. Layers within a group have the same number of channels; the number of channels across groups increases with the depth of the model (24, 32, 64, and 128 channels, respectively). We use a kernel size of three in all layers, a padding of one and a stride of one. See Appendix A for a visualization of the encoder architecture.

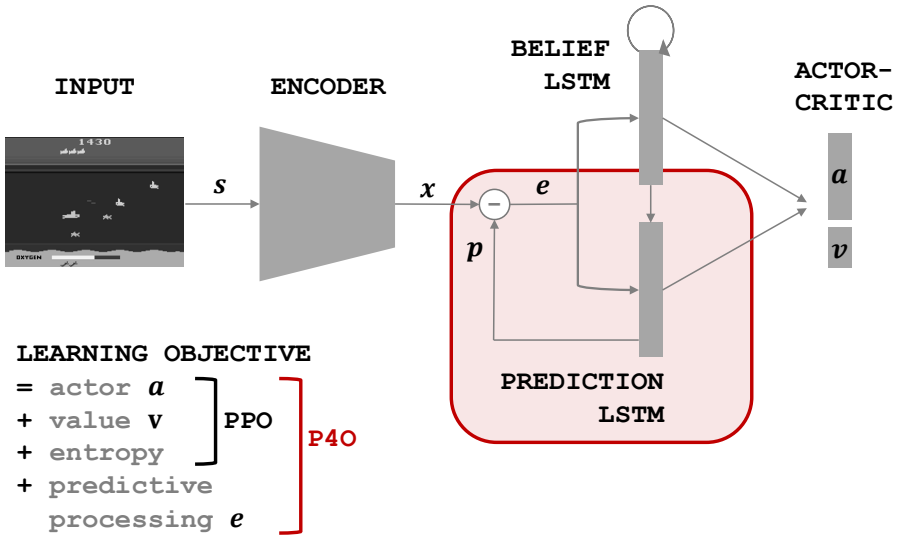

Figure 1: Components of the P4O architecture. A sensory input s is encoded into a low-dimensional latent representation x. This encoded game state is subtracted from a prediction p generated by an LSTM layer. The resulting prediction error is passed into both the prediction layer and a second LSTM layer representing the agent's belief states. The LSTM outputs are used by an actor-critic model to select an action a and compute a corresponding state value v. The red box highlights the main architectural contribution of P4O. Minimization of the prediction error is added to the P4O agent's objective function.

2.1.2 RNN MODEL

Our RNN consists of two populations of LSTM cells (Hochreiter & Schmidhuber, 1997) which together form the world model of the P4O agent. One population generates a prediction, p_{t-1}, of the upcoming latent sensory representation, x_t. The other population can be interpreted as a more persistent *belief* state of the agent. The prediction population's structure is inspired by predictive coding architectures (Ali et al., 2021; Rao & Ballard, 1999), in which a population of error neurons and prediction neurons are separately structured. The prediction population provides a feedback loop which converts the encoded sensory input into a prediction error signal, $e_t = p_{t-1} - x_t$. This prediction error is then used as input to both LSTM cell populations providing the necessary information to update the internal world model. The prediction LSTM units, acting as information integrators, additionally receive input from the belief LSTM states.

To make the LSTM update rules explicit, we define the states of our belief and prediction LSTM units as h_t and p_t, respectively. These LSTM outputs are controlled by gating variables, $g_{j,t} = \sigma(z_{j,t})$, for the input, output and forget gates, $j \in \{\mathrm{i}, \mathrm{o}, \mathrm{f}\}$. Here, $z_{j,t} = [z^h_{j,t}, z^p_{j,t}]$ are the internal states of the belief and prediction LSTM unit gates. In P4O, the gates of the belief population are updated as in regular LSTMs based on error input and the previous hidden state:

$$z^h_{j,t} = W^h_j e_t + U^h_j h_{t-1} + b^h_j. \tag{1}$$

Here, W_j denotes the input weights, U_j the recurrent weights, and b_j the bias of the corresponding gate. The prediction population likewise receives the prediction error as external input together with the belief states, as discussed above:

$$z^p_{j,t} = W^p_j e_t + U^p_j h_{t-1} + b^p_j. \tag{2}$$

Notably, the gate variables for the prediction population receive both input from the belief population and indirectly from their own previous state via the prediction error loop.

2.1.3 ACTOR-CRITIC MODEL

The combined hidden states $[h_t, p_t]$ of the world model are passed as input to an actor-critic model which uses two fully-connected layers to select actions a_t and predict state values v_t. The actor layer contains one neuron for each possible action and implements a policy $\pi(a_t \mid h_t)$ by applying a softmax on the network output, resulting in a probability distribution over the action space. The agent chooses an action by sampling from this action distribution. The critic layer consists of a single neuron that outputs the state value v_t. The objective function minimized while training the agent is described in the next section.

2.2 P4O ALGORITHM

The learning algorithm is based on a modification of the standard PPO algorithm. This modification makes it suitable for training recurrent models and jointly minimizes prediction error while optimizing for action. Similar to PPO, the agent retrieves a batch of data by interacting with a number of parallel environments simultaneously. The data batch then updates the model by splitting the data into mini-batches and training for multiple epochs while constraining the divergence of the policy. However, since we use a recurrent model, hidden states are retained during rollout in order to update by backpropagation through time. A difficulty then arises due to the parameter updates within epochs. Hidden states become 'stale' after any parameter update as they no longer represent the states of the updated RNN. To avoid this issue, we generate new hidden states by re-running the LSTM after each parameter update.

The loss L_t at time t with respect to the parameters θ decomposes into a predictive processing loss combined with the other PPO loss terms. The predictive processing loss is given by

$$L_t^P(\theta) = \sum_{i=1}^{H} \text{MSE}(e_{t+i}) \,, \tag{3}$$

where e_t denotes the prediction error at time t, $\text{MSE}(\cdot)$ is a mean squared error, and H is a prediction horizon, allowing prediction error minimization over multiple consecutive timepoints. If we only use the prediction error of predicting a single step ahead ($H = 1$), the model might be tempted to copy the previous state, since the difference in the environment after a single step can be very small. To force the world model to learn temporal relationships, we let the model unroll multiple steps ahead ($H > 1$) during training. This is accomplished by fixing the prediction errors to zero, thus allowing an 'unaffected' longer time prediction sequence. We can then use the true encoder inputs during this rollout to measure the prediction errors, thereafter used to compute the loss as per Equation 3.

We combine the predictive processing loss, $L_t^P(\theta)$, with standard PPO loss components (Schulman et al., 2017) which include an actor loss, $L_t^A(\theta)$, a critic loss, $L_t^V(\theta)$, and a loss which penalises low entropy policies (facilitating exploration), $L_t^H(\theta)$. We sum these to form the combined objective to minimise, defined as

$$L_t(\theta) = \hat{E}_t \left[c_1 L_t^A(\theta) + c_2 L_t^V(\theta) + c_3 L_t^P(\theta) + c_4 L_t^H \right] \,, \tag{4}$$

where the loss coefficients are denoted by c_i and $\hat{E}_t$ is the empirical mean over a set of samples. The actor loss is clipped to avoid strong divergences, as in the original PPO implementation (Schulman et al., 2017). The critic loss is modified by clipping of the change in value estimate and replacement of the squared difference with an absolute difference. We found these changes to provide an empirical benefit in stability during training.

Unlike the standard PPO, we refresh the calculated advantages with the latest model before each update as suggested by Andrychowicz et al. (2020), to prevent basing calculations on old data as in the case with hidden states. The advantages are calculated with generalized advantage estimation (Schulman et al., 2015) in its truncated form, as described by Schulman et al. (2017). Lastly, in standard PPO the first update is unconstrained because the batch was retrieved with the same policy that is being updated, leading always to an action probability ratio of 1. To prevent the first update from changing the policy too drastically, we ignore the latest policy and instead update based upon the second-to-last policy of the previous batch.

2.3 EXPERIMENTAL SETUP

We utilize the Atari 2600 benchmark commonly used in model-free reinforcement learning research for ease of comparison with state-of-the-art methods. Due to computational resource limitations, we focus on six environments spanning a wide range of difficulty levels. The selected games, from most to least difficult, include Seaquest, Riverraid, Q*bert, Beamrider, SpaceInvaders and Breakout. The original Atari frames of 210 by 160 pixels in RGB color were converted to the commonly used 84 by 84 pixels in grayscale format. We apply the typical four-frame stacking, and therefore the final input is four channels of 84×84. We use sticky actions, the full action space, and train with 16 environments in parallel. We do not enforce a time or frame limit per episode, as suggested by Toromanoff et al. (2019). The encoder transforms the input $s_t \in \mathbb{R}^{84\times84}$ into a low-dimensional input representation $x_t \in \mathbb{R}^p$ with $p = 512$. We use a belief LSTM population with state $h_t \in \mathbb{R}^q$ where $q = 512$. Overall, P4O operates with combined predictive and belief LSTM population states $[h_t, p_t] \in \mathbb{R}^k$ with $k = p + q$. Finally, the actor-critic model selects one out of 4-18 possible actions (depending on the game) and generates one state value per time step.

We compare the performance of the P4O algorithm against a recurrent variant of the original PPO algorithm, which we call the LSTM-PPO baseline algorithm. This baseline algorithm also uses a ResNet encoder and is similar to the P4O algorithm in most ways, yet lacks predictive processing. We used two variants of LSTM-PPO. First, a model which uses $k = 1024$ hidden states, matching the number of hidden states as in the LSTM layer of the P4O architecture. Second, a model which uses $k = 800$ units such that the total number of parameters is comparable with that of the P4O architecture. Hence, the former keeps the size of the hidden state equal to that of P4O whereas the latter controls for model complexity. Notably, the discrepancy between scaling of hidden state dimension and model complexity is due to architectural differences between the LSTM-PPO architecture, which has k^2+kp+k parameters per gate, versus the P4O architecture, with only $kp+kq+k$ parameters per gate.

For most PPO-related hyperparameters we do not apply grid search, but instead use commonly reported hyperparameter values from other PPO implementations (see Appendix B). For predictive processing, a prediction horizon of 3 was chosen by evaluating the trade-off between improved agent performance and increased computational cost.

We additionally ran a single P4O agent for 10 days to compare performance with the current state-of-the-art in model-based and model-free single GPU agents. Due to computational resource limitations, this longer experiment was run only in the Seaquest environment. To perform well at Seaquest, an agent must effectively juggle multiple goals on different timescales, requiring more complex planning behavior than, for example, Breakout. This makes it one of the more difficult games in the Atari collection to achieve superhuman performance in, as demonstrated by Mnih et al. (2015), who reported a DQN achieving only 25% of their human gamer normalized score, or roughly 12% of the human gamer used by Hafner et al. (2020).

For additional implementation details, please refer to Appendix C. All code required to reproduce the simulations described above, is available at <REDACTED FOR ANONYMOUS SUBMISSION – see additional submission zip for code >.

3 RESULTS

We here investigate if predictive processing improves the learning of control policies by comparing performance of P4O against both a baseline model and other state-of-the-art RL algorithms.

3.1 BASELINE COMPARISON

As Figure 2 demonstrates, the P4O algorithm significantly outperforms the baseline LSTM-PPO algorithm ($k = 1024$) in 5 out of 6 games tested. The difference between the mean learning curves of the two algorithms are statistically significant in all environments, based on one-tailed t-tests ($p < 0.05$, $N = 16$), with an even stronger effect ($p < 0.001$) for Riverraid, Q*bert and Breakout. The effect is similar for the LSTM-PPO baseline with $k = 800$, which has a comparable number of parameters to the P4O algorithm, confirming that the difference in performance can indeed be attributed to the predictive processing impact, rather than an increase in efficiency due to reduced

model complexity. Also, the inclusion of the predictive processing loss in the optimization of the P4O agents yields a significant contribution to performance in all environments ($p < 0.05$, $N = 16$), justifying its incorporation into the P4O algorithm. In all environments where the P4O algorithm outperforms the baseline LSTM-PPO algorithm, predictive processing loss was successfully minimized. In SpaceInvaders P4O actually performed worse, which is associated with a failure to minimize predictive processing loss. In Seaquest, the most difficult game in our chosen set, P4O achieves a mean score of 6216 compared to our LSTM-PPO (k=1024) baseline's mean score of 2166, implying a 2.9X increase in performance with the inclusion of predictive processing, while requiring 22% fewer parameters in total. The P4O agent also surpasses the mean score of 1204 reported by the PPO paper (Schulman et al., 2017) at 40 million frames in Seaquest. Further tuning of the hyperparameters for the P4O agent, such as the scaling of the predictive processing component of the loss, may lead to greater performance gains.

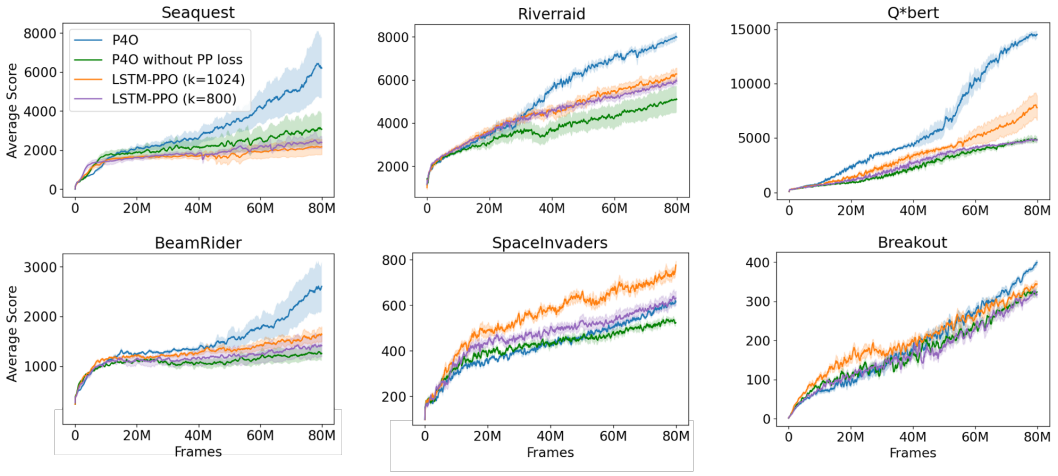

Figure 2: Comparison of P4O algorithm against the baselines LSTM-PPO ($k = 1024$), LSTM-PPO ($k = 800$) and the P4O model optimized without predictive processing loss (P4O without PP loss). Results are based on average score over the last 100 episodes. Shaded areas show standard error of the mean.

3.2 Individual runs

When inspecting the mean learning curves, we observe a distinct gap between the P4O and baseline agents, especially in Seaquest, Q*bert and BeamRider. This finding indicates some fundamental learning barrier once a certain score level is reached. Investigating the behavior of the baseline agents around this saturation point reveals that the agents struggle to integrate the variety of competing goals on different temporal scales. In Seaquest for instance, the agents initially play the game by merely avoiding and destroying the enemy ships while restricting their shooting to the bottom half of the screen. The agents fail to learn to tackle other goals like rescuing the stranded divers or moving up for air when the oxygen level is low. The P4O agents appear to cross this learning barrier, begin to move up for air in time and thus play for much longer, using the entire screen and rescuing divers in the process. Similarly, games like Q*bert and BeamRider require fulfillment of new goals in each stage before moving on to the next stages, unlike the remaining games. In Q*bert, the agents that learn to jump on each cube twice rather than once at a later stage of the game manage to cross this barrier towards better performance. In BeamRider, high-scoring agents reach Stage 9 by also learning to avoid various enemies, while those only learning to shoot targets get stuck at Stage 2. Figure 3 shows the score of individual runs for the P4O and LSTM-PPO agents on a variety of games. Notably, in many cases we can observe that the LSTM-PPO agents reach plateaus in performance due to the above issues around strategy. Ultimately, this results in a strong negative effect on the final score relative to P4O.

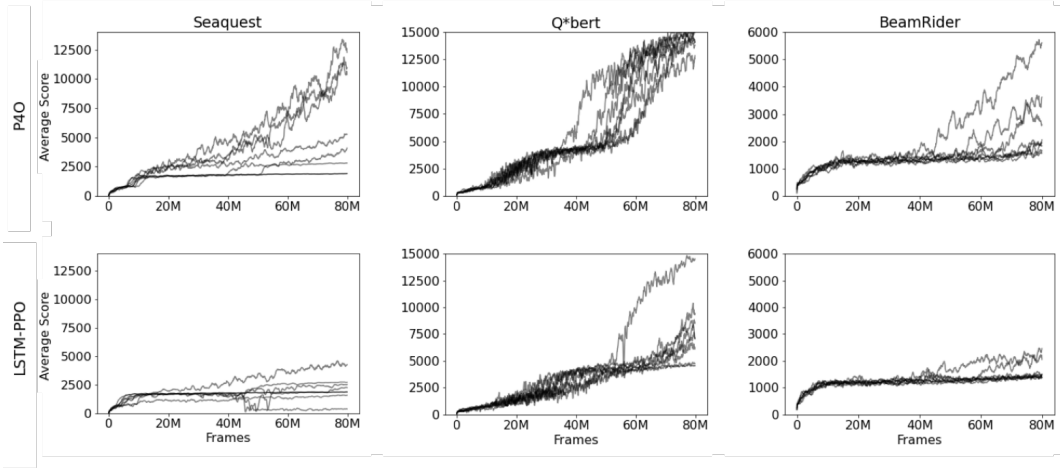

Figure 3: Comparison of individual performance curves. P4O results are shown in the top row and LSTM-PPO baseline ($k = 1024$) results are shown in the bottom row.

3.3 Input Encoding

The inclusion of a predictive coding loss term during training has a clear effect on how inputs are encoded (Figure 4). In the LSTM-PPO baseline algorithm, the output of the encoder follows a typical post-tanh activation profile, with most values being grouped at the extremes (-1 and 1). In contrast, input activations of the P4O algorithm develop a peaked distribution about the origin, despite the final tanh activation layer. Similarly, the prediction p_t and prediction error, e_{t+1}, distributions are centered around the zero point. The coefficient of determination (R^2) of the prediction with respect to the encoded input is 0.86 for the data shown in Figure 4, meaning much of the variance in the input is explained by the prediction of the model. While the exact score varies from agent to agent, most P4O agents achieve a similar result.

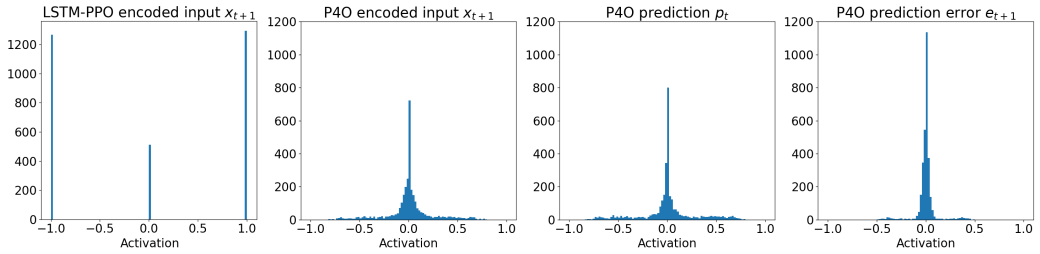

Figure 4: Comparison of activation distributions between the LSTM-PPO (k=1024) baseline latent encoding, the P4O latent encoding, the P4O prediction and the P4O prediction error. The distributions are drawn from five states aggregated from trained agents.

3.4 Comparison with State-of-the-Art

To place the performance of our agent in a broader context, a 10 day-long training run of the P4O agent in Seaquest is compared with a number of current state-of-the-art single GPU reinforcement learning agents in Figure 5.

The model-based DreamerV2 agent and model-free Rainbow and IQN agents all report their performance after 10 accelerator days in wall-clock time (Hessel et al., 2018; Dabney et al., 2018; Hafner et al., 2020). Each of these three models has processed 200M Atari frames at the final test time, whereas our P4O agent is able to process 1.2B frames in 10 days with the same Nvidia Tesla V100

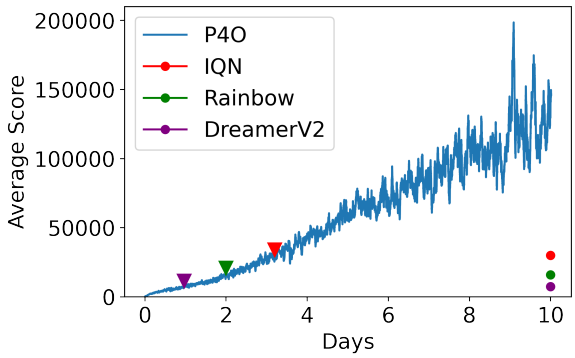

Figure 5: Single seed performance comparison on Seaquest with IQN, Rainbow and DreamerV2 after 10 days of accelerator time. Circles represent final reported scores of the IQN, Rainbow and DreamerV2 agents after 10 days. The moment where P4O exceeds their respective endpoint score is marked with triangles. Reported average score is a rolling mean of the last 100 episodes.

GPU as used by DreamerV2 (Table 1). Furthermore, the DreamerV2 agent uses 22M parameters, whereas the P4O agent uses 7.5M parameters. As can be seen in Figure 5, this single run of our agent surpasses the DreamerV2 average after 1 day, surpasses the Rainbow agent's final score after roughly 2 days, and the IQN agent in less than 4 days, achieving a final average score of 151750 over the last 100 episodes, or 361% of the human gamer score reported by Hafner et al. (2020). On balance, it appears that the sample efficiency of our P4O agent is lower than both IQN and Rainbow agents, as can be seen when examining performances after training on a matched number (200M) of Atari frames, though for P4O this takes only 1.7 accelerator days.

Agent	Atari Frames	Accelerator Days	Average Score	Gamer-Normalized Score
DreamerV2	200M	10	7480	0.18
Rainbow	200M	10	15898	0.38
IQN	200M	10	30140	0.72
P4O	200M	1.7	10299	0.24
P4O	**1.2B**	**10**	**151750 (319095)**	**3.61 (7.59)**

Table 1: Comparison of our P4O agent with top single GPU agents on Seaquest (Hessel et al., 2018; Dabney et al., 2018; Hafner et al., 2020). Gamer-normalized score based on the human gamer score reported by Hafner et al. (2020). Scores in parentheses for P4O are achieved when running the trained agent in deterministic mode (only exploitation). P4O's performance is reported in two cases: when trained with the same number of Atari frames and when trained for the same number of accelerator days as the competitors.

The performance curve shows no sign of tapering off at the end of the run, suggesting that the agent would still benefit from additional time to further approach perfect play. The relatively large gap between the highest score and average score indicates that the agent is still exploring through action sampling with the entropy bonus, although it has already beaten the game multiple times reaching the maximum score (999999). To extract the maximum performance from our agent, we can take the trained agent and run it in a deterministic mode by no longer sampling from the action distribution, but instead always selecting the highest probability action. Testing the trained agent in this way for another 100 episodes leads to a much higher average score of 319095, or 759% of the human gamer score. Further inspection shows that the agent achieved the maximum score in 16% of these episodes. Given this result, it may be beneficial to apply a decay factor to the entropy bonus to allow the model to become more deterministic towards the end of training.

4 DISCUSSION

We have here demonstrated that learning of control policies can be significantly improved by incorporating predictive processing within reinforcement learning agents. To this end, we introduced the P4O algorithm, which combines predictive processing with proximal policy optimization in recurrent neural networks. Results show that through the incorporation of a measure of prediction error which is minimised during training, the accumulated game score is significantly improved in challenging Atari environments.

The current trend in state-of-the-art reinforcement learning has been excessively complex, biologically implausible and computationally intensive multi GPU agents (Schrittwieser et al., 2020; Badia et al., 2020; Ecoffet et al., 2019). The DreamerV2 agent already made great strides reversing this trend by reducing complexity and demonstrating what is possible with a single GPU agent (Hafner et al., 2020). Our P4O agent continues this line of research by incorporating a predictive processing element in a standard control architecture. In a ten-day (wall-clock time) single-GPU training comparison, our P4O agent outperformed other model-free and model-based state-of-the-art reinforcement learning agents. Wall-clock time is arguably the most limiting factor in reinforcement learning research, and therefore should be considered as a metric besides cumulative reward. For example, both the DreamerV2 agent (Hafner et al., 2020) and the IQN agent (Dabney et al., 2018) require six-fold the wall-clock time to process the same number of Atari frames as P4O without a proportionally greater sample efficiency. In fact, the DreamerV2 agent is worse on sampling efficiency based on performance per frames observed, despite utilizing a similar approach with a world model that is differently used to predict entire imagined trajectories, and having 193% more parameters compared to the P4O agent. On the other hand, the IQN and Rainbow agents achieve better sampling efficiency than P4O at the cost of significantly greater computational expense.

The question remains why RNNs that also minimize their prediction error perform so well. We hypothesize that it encourages the RNN to learn an internal representation (world model) of the causes of its sensations (Ha & Schmidhuber, 2018). Such an induced world model may provide a better basis for control. Furthermore, the use of a predictive processing loss may serve as a regularizer that decorrelates the inputs, which is similar to whitening of sensory input in early visual areas (Graham et al., 2006) and has been shown to have a positive impact on generalization performance in classification tasks (Huang et al., 2018). We expect that a deeper understanding and more widespread adoption of brain-inspired mechanisms such as the one proposed here, will yield more efficient and effective neural controllers in artificial intelligence.

REFERENCES

Abdullahi Ali, Nasir Ahmad, Elgar de Groot, Marcel A. J. van Gerven, and Tim C. Kietzmann. Predictive coding is a consequence of energy efficiency in recurrent neural networks. *bioRxiv 2021.02.16.430904*, 2021.

Arjen Alink, Caspar M Schwiedrzik, Axel Kohler, Wolf Singer, and Lars Muckli. Stimulus predictability reduces responses in primary visual cortex. *Journal of Neuroscience*, 30(8):2960–2966, 2010.

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, et al. What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*, 2020.

Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*, pp. 507–517. PMLR, 2020.

Horace B Barlow. Possible principles underlying the transformation of sensory messages. In W. A. Rosenblith (ed.), *Sensory Communication*, pp. 217–234. MIT Press, Cambridge, MA, 1961.

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

Alejandra Ciria, Guido Schillaci, Giovanni Pezzulo, Verena V Hafner, and Bruno Lara. Predictive processing in cognitive robotics: a review. *Neural Computation*, 33(5):1402–1432, 2021.

Andy Clark. Whatever next? predictive brains, situated agents, and the future of cognitive science. *Behavioral and Brain Sciences*, 36(3):181–204, 2013.

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *International Conference on Machine Learning*, pp. 1096–1105. PMLR, 2018.

Floris P De Lange, Micha Heilbron, and Peter Kok. How do expectations shape perception? *Trends in Cognitive Sciences*, 22(9):764–779, 2018.

Nadine Dijkstra, Luca Ambrogioni, Diego Vidaurre, and Marcel van Gerven. Neural dynamics of perceptual inference and its reversal during imagery. *Elife*, 9:e53588, 2020.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

Matthias Ekman, Peter Kok, and Floris P de Lange. Time-compressed preplay of anticipated events in human primary visual cortex. *Nature Communications*, 8(1):1–9, 2017.

Jeffrey L Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pp. 1407–1416. PMLR, 2018.

Karl Friston. A theory of cortical responses. *Philosophical transactions of the Royal Society B: Biological sciences*, 360(1456):815–836, 2005.

Karl Friston. The free-energy principle: a unified brain theory? *Nature Reviews Neuroscience*, 11 (2):127–138, 2010.

Daniel J Graham, Damon M Chandler, and David J Field. Can the theory of "whitening" explain the center-surround properties of retinal ganglion cell receptive fields? *Vision Research*, 46(18): 2901–2913, September 2006.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. *arXiv preprint arXiv:1809.01999*, 2018.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

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

Lei Huang, Dawei Yang, Bo Lang, and Jia Deng. Decorrelated batch normalization. *arXiv preprint arXiv:1804.08450*, April 2018.

JM Hupé, AC James, BR Payne, SG Lomber, P Girard, and J Bullier. Cortical feedback improves discrimination between figure and background by v1, v2 and v3 neurons. *Nature*, 394(6695): 784–787, 1998.

Michael I Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Artificial neural networks: concept learning*, pp. 112–127. 1990.

Peter Kok, Janneke FM Jehee, and Floris P De Lange. Less is more: expectation sharpens representations in the primary visual cortex. *Neuron*, 75(2):265–270, 2012.

Tai Sing Lee and David Mumford. Hierarchical bayesian inference in the visual cortex. *Journal of the Optical Society of America A*, 20(7):1434–1448, 2003.

Wolfgang Maass. Searching for principles of brain computation. *Current Opinion in Behavioral Sciences*, 11:81–92, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

David Mumford. On the computational architecture of the neocortex. *Biological Cybernetics*, 66 (3):241–251, 1992.

Scott O Murray, Daniel Kersten, Bruno A Olshausen, Paul Schrater, and David L Woods. Shape perception reduces activity in human primary visual cortex. *Proceedings of the National Academy of Sciences*, 99(23):15164–15169, 2002.

Risto Näätänen, Mari Tervaniemi, Elyse Sussman, Petri Paavilainen, and István Winkler. 'primitive intelligence'in the auditory cortex. *Trends in Neurosciences*, 24(5):283–288, 2001.

Hrishikesh M Rao, J Patrick Mayo, and Marc A Sommer. Circuits for presaccadic visual remapping. *Journal of Neurophysiology*, 116(6):2624–2636, 2016.

Rajesh PN Rao and Dana H Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2(1):79–87, 1999.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Caspar M Schwiedrzik and Winrich A Freiwald. High-level prediction signals in a low-level area of the macaque face-processing hierarchy. *Neuron*, 96(1):89–97, 2017.

Nancy K Squires, Kenneth C Squires, and Steven A Hillyard. Two varieties of long-latency positive waves evoked by unpredictable auditory stimuli in man. *Electroencephalography and Clinical Neurophysiology*, 38(4):387–401, 1975.

Mandyam Veerambudi Srinivasan, Simon Barry Laughlin, and Andreas Dubs. Predictive coding: a fresh view of inhibition in the retina. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 216(1205):427–459, 1982.

Christopher Summerfield, Emily H Trittschuh, Jim M Monti, M-Marsel Mesulam, and Tobias Egner. Neural repetition suppression reflects fulfilled perceptual expectations. *Nature Neuroscience*, 11 (9):1004, 2008.

David Sussillo. Neural circuits as computational dynamical systems. *Current Opinion in Neurobiology*, 25:156–163, 2014. ISSN 09594388. doi: 10.1016/j.conb.2014.01.008.

Marin Toromanoff, Emilie Wirbel, and Fabien Moutarde. Is deep reinforcement learning really superhuman on atari? leveling the playing field. *arXiv preprint arXiv:1908.04683*, 2019.

Saurabh Vyas, Matthew D. Golub, David Sussillo, and Krishna V. Shenoy. Computation through Neural Population Dynamics. *Annual Review of Neuroscience*, 43:249–275, 2020. ISSN 15454126. doi: 10.1146/annurev-neuro-092619-094115.

A ENCODER MODEL ARCHITECTURE

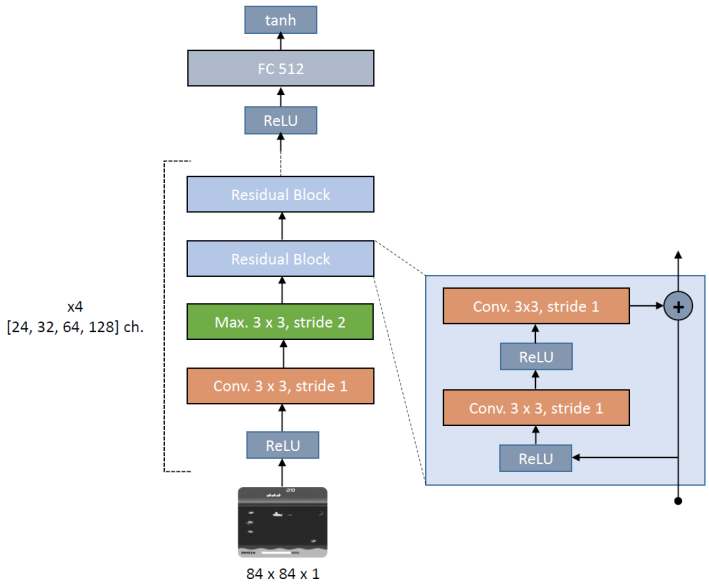

Figure 6: Encoder architecture, following a similar structure to the architecture used by (Espeholt et al., 2018) with a few modifications. The encoder uses a total of 20 convolutional layers and 3.3M parameters.

B HYPERPARAMETERS

Hyperparameter	Value
Learning rate	2.5×10^{-4}
Optimizer	Adam
Adam (ϵ)	1×10^{-5}
Learning rate decay	0.995 every 100 batches, min 5×10^{-6}
Num. parallel environments	16
Mini-batch size	400
Discount (γ)	0.99
GAE parameter (λ)	0.95
PPO clip range (ϵ)	0.1
Epochs per batch	4
Num. mini-batches	5
Actor loss coefficient (c_1)	1.0
Critic loss coefficient (c_2)	0.5
Predictive processing loss coefficient (c_3)	1.0
Entropy term coefficient (c_4)	0.02
L1 norm loss coefficient (c_5)	0.1
Hidden units in final encoder layer	512
LSTM hidden units	1024
ResNet channels	[24,32,64,128]
Image width, height, channels	84, 84, 1
Frame stacking	4

Table 2: Hyperparameters used in the P4O agents.

C HARDWARE AND IMPLEMENTATION DETAILS

We programmed our implementation in Python using the MxNet framework. Because our model is relatively small and efficient, it can be run on a single GPU requiring roughly 6GB of GPU memory. We used a combination of Google Cloud instances with Nvidia Tesla V100 and T4 GPUs and consumer hardware ranging from Nvidia GTX 1060 to RTX 2080TI graphics cards with typical multi-core CPUs to run our experiments. The fact that the agent can be run on an Nvidia GTX 1060 with 6GB of GPU memory demonstrates the small footprint of our model. The choice of CPU did not seem to affect the speed of the model significantly, considering that the largest bottleneck during training was GPU speed.