# Predictive Processing in Proximal Policy Optimization

Master Thesis
Walraaf Borkent

Supervisor
Prof. Marcel van Gerven
Radboud University

Department of Artificial Intelligence
Faculty of Social Sciences
Radboud University

April 26th 2021

**Abstract**

Advances in reinforcement learning have led to drastically more complex agents that require unrealistic amounts of compute resources. The human brain often achieves comparable results with a fraction of the energy requirements of these models. Therefore, we turn to insights from neuroscience on predictive processing and efficient coding to design an efficient agent. 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. The prediction error that results from subtracting the encoded observed state from the world model prediction is used as the primary signal in our model. We demonstrate that with this approach, predictive processing with a world model can be incorporated while reducing a model's biologically analogous energy footprint, thus supporting the efficient coding hypothesis. When we use the encoded state information only to inhibit the recurrent connections, rather than providing the prediction error separately as input, the number of neurons in the model can be drastically reduced. Moreover, this approach encourages activations in the model to remain centered around the zero point, analogous to a lower spiking rate in a biological system and reduced energy usage. Furthermore, the P4O agent far outperforms the original PPO algorithm on the Seaquest environment while retaining its efficiency and can be run on a single GPU. It also outperforms other model-based and model-free state-of-the-art single GPU agents on Seaquest given the same wall-clock time and exceeds human gamer performance in an initial performance comparison. Future research could extend the agent with additional uses of its world model, improve its performance through tuning, inspect its neural coding of competing goals on different timescales or investigate the use of our approach in modeling brain function in various scenarios. Altogether, our work underlines the synergistic benefits of the convergence of insights from the fields of neuroscience, artificial intelligence and cognitive science.

# 1 Introduction

Over the past decade, many advances have been made in the field of reinforcement learning. Initially, reinforcement learning models were still relatively biologically plausible to the extent that their performance was in large part due to the use of artificial neural networks. However, these models were notoriously difficult to train and unable to be generalized or used for transfer learning. Many techniques were introduced that attempted to improve the stability and performance of these models, but they remain typical drawbacks of reinforcement learning. Attention also focused on more complex problems to solve. For example, the 'Human Atari benchmark'; a set of 57 Atari games where the overall goal is to beat human performance across the entire library of games ((Bellemare, Naddaf, Veness, & Bowling, 2013)). Some of these games were solvable with typical reinforcement learning techniques such as deep Q learning (Mnih et al., 2015) or actor-critic models, while others required a different approach. As performance on this benchmark became more competitive, more elaborate models were designed to beat the state-of-the-art.

Current state-of-the-art for most of these games is the MuZero agent (Schrittwieser et al., 2020), a complex model which utilizes the biologically implausible Monte Carlo tree search (MCTS) to improve its policy with large amounts of computing power. The direct competition for MuZero comes mostly from similarly complex models, such as Agent57 (Badia et al., 2020), which combines a large number of innovative approaches into a single model, and GoExplore (Ecoffet, Huizinga, Lehman, Stanley, & Clune, 2019), which keeps an archive of trajectories to force exploration of promising unknown states. All three of these agents are distributed multi-GPU approaches that are too computationally expensive to be viable for most research groups, and their code has not been made publicly available. Each of these approaches has added significant complexity to their model compared to the earlier breakthrough approaches in reinforcement learning and none of these models appear particularly biologically plausible as is. Since we know that the human brain is able to solve these tasks with ease, there must exist a more biologically plausible way of tackling these problems. Similarly, there must be a more efficient and elegant solution, considering that the human brain requires a fraction of the power and sampling that these complex algorithms use.

To find such a solution, it would be preferable to start from a foundation that is comparatively simple and efficient. The best performing recent algorithm that fits this description is the proximal policy optimization (PPO) algorithm (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017). This algorithm uses an actor-critic approach whereby the change in the policy at every update is constrained by its divergence from the previous policy through a simple clipping operation. This ensures that the policy cannot change too drastically and undo its progress in a single update. With this foundation, we can look at biologically plausible modifications that improve its performance, inspired by advances in neuroscience.

To better estimate future states and plan actions in a biologically plausible way, the neural network at the heart of the algorithm would need to have recurrent connections. Recurrent neural networks (RNN) take inspiration from our brain's recurrent connections to implement complex behavior (Elman, 1990; Jordan, 1990; Rumelhart, Hinton, & Williams, 1985) and it has been shown that they are well suited to predict sensory information processing in the human brain (Güçlü & van Gerven, 2017). Biological neural networks are known to make ample use of recurrent processing to integrate information, for example in order to be able to navigate environments, as well as for memory (Maass, 2016; Moser, Rowland, & Moser, 2015). For any agent to be able to successfully complete complex tasks in dynamic environments they must be able to integrate evidence, both over time and space. Otherwise, the agent would not be able to distinguish between situations that may appear identical given the current perceptual input, but may differ substantially in broader context and thus in the required action. This issue, where many world states could map onto a single internal state, or vice versa, is called perceptual aliasing (Whitehead & Ballard, 1991). The recurrent connections in an RNN would allow it to integrate evidence, keep information in memory and thus seem to have the potential to incorporate advanced action planning. Traditional RNNs struggled with retaining information over longer periods of time due to vanishing gradients. Long short-term memories (LSTM) were invented to address this problem by enforcing a constant error flow (Hochreiter & Schmidhuber, 1997).

Related work by Ha and Schmidhuber (2018a) proposed the World Models algorithm, which incorporated an LSTM to allow the agent to learn through 'dreaming' by generating simulated future states after learning a model of the environment. A large number of recent studies investigated similar model-based approaches (Babaeizadeh, Finn, Erhan, Campbell, & Levine, 2017; Buckman, Hafner, Tucker, Brevdo, & Lee, 2018; Buesing et al., 2018; Chiappa, Racaniere, Wierstra, & Mohamed, 2017; Chua, Calandra, McAllister, & Levine, 2018; Denton & Fergus, 2018; Doerr et al., 2018; Gal, McAllister, & Rasmussen, 2016; Gemici et al., 2017; Gregor, Papamakarios, Besse, Buesing, & Weber, 2018; Ha & Schmidhuber, 2018b; Hafner et al., 2019; Henaff, Whitney, & LeCun, 2017; Higuera, Meger, & Dudek, 2018; Igl, Zintgraf, Le, Wood, & Whiteson, 2018; Kalweit & Boedecker, 2017; Karl, Soelch, Bayer, & Van der Smagt, 2016; Krishnan, Shalit, & Sontag, 2015; Kurutach, Clavera, Duan, Tamar, & Abbeel, 2018; A. X. Lee, Nagabandi, Abbeel, & Levine, 2019; Nagabandi, Kahn, Fearing, & Levine, 2018; Oh, Guo, Lee, Lewis, & Singh, 2015; Srinivas, Jabri, Abbeel, Levine, & Finn, 2018; Wang & Ba, 2019; Wang et al., 2019; Watter, Springenberg, Boedecker, & Riedmiller, 2015; Wayne et al., 2018; Weber et al., 2017). Many of these studies intended to use the model of the world to improve sample efficiency; by allowing the agent to train on imagined trajectories, in addition to rollouts in the actual environment, fewer steps in the real environment would be needed to reach similar performance. However, this often did not translate into more efficient use of computational resources than the model-free counterparts and the performance tended to be strongly affected by imperfections in the predicted trajectories. Furthermore, many of these studies limited the practical application to simpler tasks than Atari environments.

Recent work on DreamerV2 (Hafner, Lillicrap, Norouzi, & Ba, 2020) demonstrated that the world model introduced by Hafner et al. (2019) can be used to learn behavior exclusively from latent space predictions and can outperform current state-of-the-art model-free algorithms simultaneously in both absolute score and sample efficiency in a collection of 55 Atari games. The world model in DreamerV2 is trained entirely separately from its policy with an additional neural network and attains human-level performance. The work on DreamerV2 made significant strides to reduce complexity and computational requirements compared to models such as MuZero (Schrittwieser et al., 2020), which requires multiple TPUs to train; equivalent to 80 days of single GPU training. However, DreamerV2 still uses six separate components in its model, 22M trainable parameters and requires roughly 10 days to process 200M environment frames (Hafner et al., 2020). It generates 468B imagined latent states to train its policy, leaving us to question whether this is the only way to benefit from a model-based approach and whether more efficient methods can be designed.

A leading theory in neuroscience proposes that the brain processes sensory information through a process named predictive coding or predictive processing (Ciria, Schillaci, Pezzulo, Hafner, & Lara, 2021; Clark, 2013; Friston, 2005; Mumford, 1992; Srinivasan, Laughlin, & Dubs, 1982). This theory suggests that higher-level brain areas predict the activation of lower-level brain areas and inhibit neuronal activation when it matches the expectation. The remaining prediction error signal can be seen as a measure of surprise compared to its internal model of the world that was used to generate the predictions. This surprise signal can then be used to adjust behavior and update its internal understanding of the world. A number of studies have contributed to the growing experimental evidence for this theory (Alink, Schwiedrzik, Kohler, Singer, & Muckli, 2010; De Lange, Heilbron, & Kok, 2018; Dijkstra, Ambrogioni, Vidaurre, & van Gerven, 2020; Ekman, Kok, & de Lange, 2017; Hupé et al., 1998; Kok, Jehee, & De Lange, 2012; Murray, Kersten, Olshausen, Schrater, & Woods, 2002; Näätänen, Tervaniemi, Sussman, Paavilainen, & Winkler, 2001; H. M. Rao, Mayo, & Sommer, 2016; Schwiedrzik & Freiwald, 2017; Squires, Squires, & Hillyard, 1975; Summerfield, Trittschuh, Monti, Mesulam, & Egner, 2008), while others reproduced the experimentally observed phenomena in explicit computational models of predictive coding (Friston, 2005, 2010; T. S. Lee & Mumford, 2003; R. P. Rao & Ballard, 1999). It stands to reason then, that such an approach would be a key component of a biologically plausible reinforcement learning agent. Besides biological plausibility as a merit on its own, there may be additional benefits to employing this approach in terms of performance or efficiency.

Considering the limited energy available to the brain, efficiency is likely to be one of the most important features of its computational mechanisms. This was phrased by Barlow (1961) as the efficient coding hypothesis; the suggestion that the brain minimizes redundancy and overall neuronal activity, i.e. energy consumption, while maximizing the amount of sensory information it can represent. Previous

work has since expanded on this hypothesis (Bell & Sejnowski, 1995; Berkes & Wiskott, 2005; Bialek, Van Steveninck, & Tishby, 2006; Chalk, Marre, & Tkačik, 2018; Eckmann, Klimmasch, Shi, & Triesch, 2020; Olshausen & Field, 1996), while a particularly relevant study by Ali, Ahmad, de Groot, van Gerven, and Kietzmann (2021) showed that predictive coding may naturally emerge in energy-constrained neural networks. Vice versa, this relationship between predictive coding and efficient coding implies it is at least possible to apply predictive coding in an energy-efficient manner and may even contribute to energy efficiency. However, most current work on practical model-based reinforcement learning agents tends to require additional neuronal networks for the world model, instead leading to increased energy usage. Therefore, the present study aims to investigate whether we can use recurrent neural networks, enriched with insights from predictive coding theory, to improve upon the performance and efficiency of the PPO algorithm on discrete Atari environments.

Enhanced efficiency in solving these complex tasks could also open the door to solving even more complex tasks and more intelligent behavior from our artificial agents. Crucially, we would be further approaching the way the human brain is likely to solve these tasks through convergence of insights from the fields of cognitive science, neuroscience and artificial intelligence; which is referred to as the great convergence (Gershman, Horvitz, & Tenenbaum, 2015; Van Gerven, 2017). Thus, besides adding to the state-of-the-art of intelligent artificial agents, we could use these models to further our understanding of human intelligence and brain function with regards to action planning, predictive coding and efficient coding.

# 2 P4O

We present the Predictive Processing Proximal Policy Optimization (P4O) agent, an actor-critic reinforcement learning agent that incorporates ideas from predictive coding theory in a recurrent variant of the PPO algorithm. We build upon the work by Ali et al. (2021) that demonstrated how predictive processing can naturally emerge under energy constraints, incorporating predictions in the hidden state. In particular, we reserve part of the recurrent hidden state to integrate a world model, predicting the encoded state of the environment at every step. The prediction error that results from subtracting the encoded observed state from this prediction is used as the primary signal in our model. This allows the model to continuously adjust its predictions and outputs exclusively through prediction error. Additionally, by reserving part of the existing hidden state rather than utilizing a separate network, we show that this can be done without increasing the number of parameters in the model. In fact, when we provide the prediction error in place of the prediction before every recurrent cycle, rather than providing both separately, the number of neurons used can be drastically reduced compared to the typical model-based approach. Moreover, this approach strongly encourages activations in the model to remain centered around the zero point. Both these properties show that predictive processing with a world model can be incorporated while reducing a model's biologically analogous energy footprint, thus efficiently representing sensory information and supporting the efficient coding hypothesis.

We show that incorporating this use of predictive processing with a world model significantly improves the performance of the agent without drastically changing the way the original model-free PPO algorithm operates. The P4O agent far outperforms the original PPO algorithm on the Seaquest environment while retaining its efficiency and can be run on a single GPU. It also outperforms other current state-of-the-art single GPU agents on Seaquest, both model-based and model-free, and exceeds human gamer performance. The agent could easily be extended with more involved usages of its world model, such as learning from full predicted trajectories similarly to DreamerV2 (Hafner et al., 2020). The exact benefit of each such extension can then verified by comparing to our minimalist approach.

## 2.1 Predictive Processing

The basis of the P4O agent consists of a simple ResNet encoder and an LSTM main model for recurrent connections, generating state value predictions and action distributions. The simplest way to transform a model-free algorithm to model-based would be to add a separate network that is trained to encode a prediction of the input the main model will receive at a future point in time, in other words, a prediction

of the future sensory information in either original or encoded form. Then, this model of the world can be used in some way in the agent's behavioral decision-making process or state value estimation. However, this relatively straightforward approach to incorporating a model-based approach is not sufficient to be called predictive processing.

As mentioned, predictive processing suggests that the brain continuously updates its prediction of sensory information and inhibits sensory input activations that match the expectation, leaving only the prediction error as a surprise signal to update its beliefs. Therefore, rather than adding a prediction of the encoded next state as an additional output, it ought to be integrated in recurrent dynamics of the main model itself. Additionally, the primary input signal to the main model should not be the actual encoded sensory information, but instead the prediction error.

## 2.2 Base Model

To achieve this in our P4O agent, we reserve half of the hidden state of the LSTM-based main model and assume it contains the future state prediction (Figure 1). The LSTM hidden state $h$ therefore consists of an unrestricted part $z$ and the future state prediction $p$. Before each LSTM cycle, we take the current encoded state information $x_t$ and subtract it from the prediction $p_{t-1}$, generating the prediction error $e_t$:

$$e_t = p_{t-1} - x_t \tag{1}$$

Instead of providing $x_t$ as input to the gating functions, we only provide $e_t$, leading to the following gating functions:

$$\begin{aligned} f_t &= \sigma_g(W_f e_t + U_f h_{t-1} + b_f) \\ i_t &= \sigma_g(W_i e_t + U_i h_{t-1} + b_i) \\ o_t &= \sigma_g(W_o e_t + U_o h_{t-1} + b_o) \\ \tilde{c}_t &= \sigma_c(W_c e_t + U_c h_{t-1} + b_c) \end{aligned} \tag{2}$$

where $f_t$, $i_t$, $o_t$ and $\tilde{c}_t$ are the forget gate, input gate, output gate and cell input respectively, while $W$, $U$ and $b$ are the weight matrices and bias vector. The sigmoid activation function is represented as $\sigma_g$ and $\sigma_h$ denotes the tanh activation function. Similarly to a normal LSTM, the final cell state $c_t$ results from an element-wise product of the forget gate $f_t$ and the previous cell state $c_{t-1}$ added to the element-wise product of the input gate $i_t$ and the cell input $\tilde{c}_t$:

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \tag{3}$$

Finally, we can define the hidden state $h_t$ as the element-wise product of the output gate $o_t$ and the activated cell state $c_t$:

$$\begin{aligned} h_t &= o_t \circ \sigma_h(c_t) \\ &= \begin{bmatrix} z_t \\ p_t \end{bmatrix} \end{aligned} \tag{4}$$

where the resulting vector $h_t$ contains the new prediction $p_t$ and an unrestricted part of the hidden state $z_t$.

By minimizing $e$ during training, the model naturally learns to encode predictions in the reserved part of the hidden state $p$. Minimizing the prediction error also forces the activations to center around the zero point, reducing the analogous biological energy consumption in line with the efficient coding hypothesis. Similarly to the MuZero agent (Schrittwieser et al., 2020), we do not add a decoder with a pixel-wise loss based on the unencoded input, so that the latent space is free to arrange itself optimally for planning and behavioral choices, rather than for image reconstruction. We refer to this base variant of our agent as 'P4O Base' throughout the following sections.

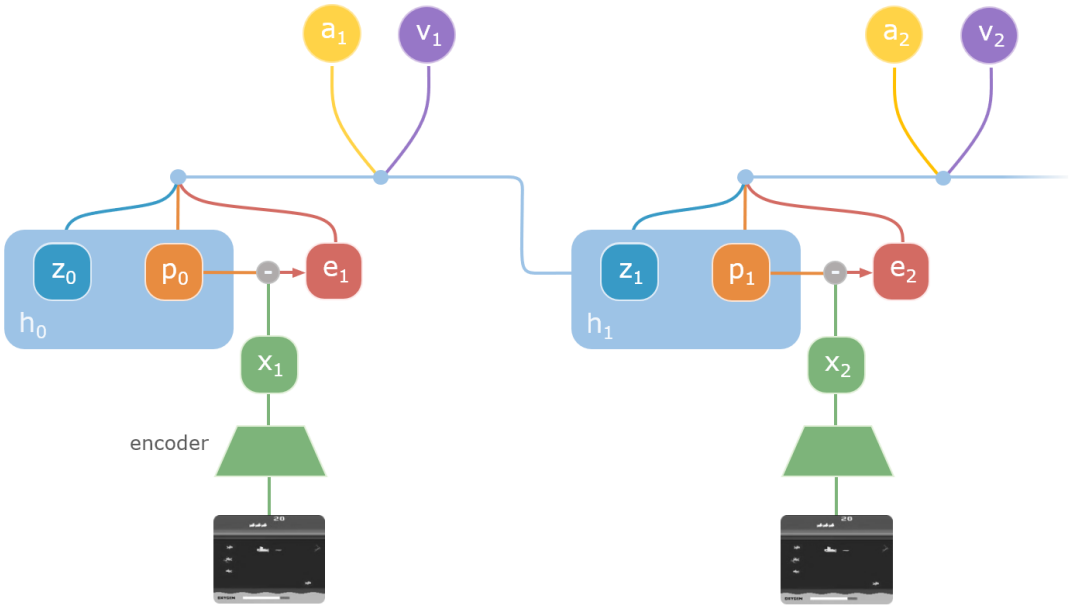


Figure 1: Visual representation of the P4O Base model. A game state observation is encoded to $x_t$, which is subtracted from the prediction $p_{t-1}$ contained in the hidden state $h_{t-1}$, generating the prediction error $e_t$. The model then uses $z_{t-1}$, $p_{t-1}$ and $e_t$ to generate action probabilities $a_t$, state value $v_t$ and the new hidden state $h_t$.

### 2.3 Integrated Prediction Error

Ideally, the main model learns to improve its predictions based purely on the information contained in the free part of hidden state $z$, and the prediction error $e$. To generate new predictions, state value estimations and action distributions, it would not necessarily need the previous prediction itself as an input to the LSTM. We can therefore take the integration one step further, as shown in Figure 2, and consider $e$ the inhibited form of $p$, and only provide $z$ and $e$ as the hidden state in each cycle:

$$\hat{h}_{t-1} = \begin{bmatrix} z_{t-1} \\ e_t \end{bmatrix} \tag{5}$$

Here $\hat{h}_{t-1}$ represents the partially inhibited form of the hidden state $h$ that resulted from inhibiting $p_{t-1}$ by $x_t$. The LSTM then requires no direct inputs besides its own recurrent connections with only an inhibitory signal on part of these connections, resulting in the following simplified gating functions:

$$\begin{aligned} f_t &= \sigma_g \left( U_f \hat{h}_{t-1} + b_f \right) \\ i_t &= \sigma_g \left( U_i \hat{h}_{t-1} + b_i \right) \\ o_t &= \sigma_g \left( U_o \hat{h}_{t-1} + b_o \right) \\ \tilde{c}_t &= \sigma_c \left( U_c \hat{h}_{t-1} + b_c \right) \end{aligned} \tag{6}$$

All other equations remain equal to the base model. This approach drastically reduces the number of parameters in the main model since only weights for the hidden state are required. This reduction in redundancy and fewer connections with low activation values nicely aligns with the efficient coding hypothesis. We refer to this variant of our agent as 'P4O Integrated' throughout the following sections.

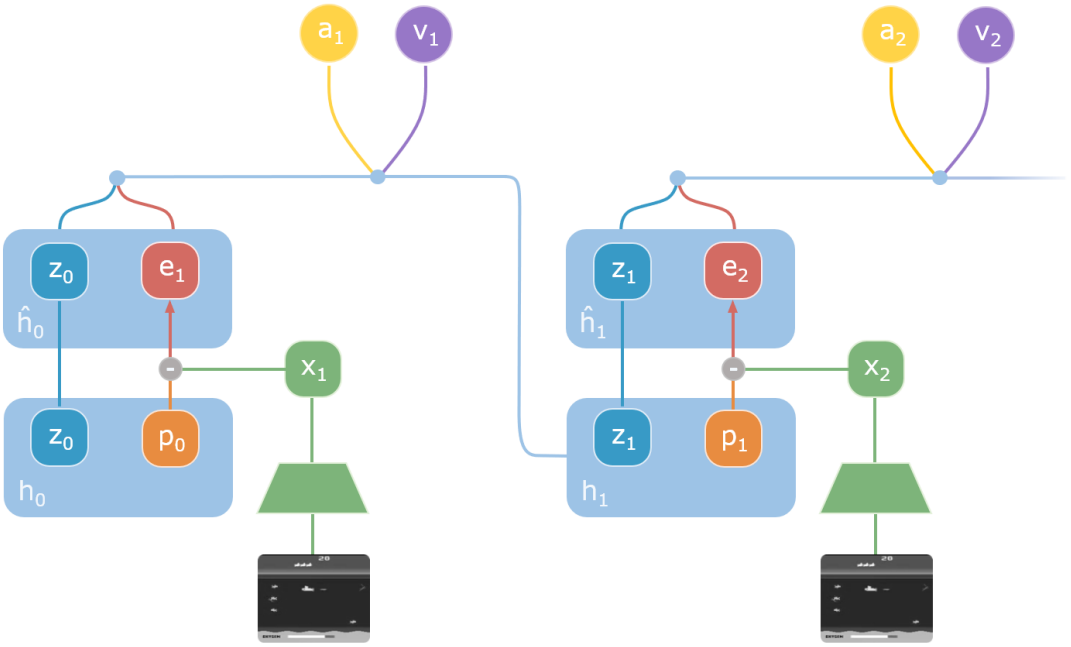


Figure 2: Visual representation of the P4O Integrated model. In this variant, we consider $e_t$ the inhibited form of $p_{t-1}$. The model only uses $z_{t-1}$ and $e_t$ to generate new action probabilities, the state value and the next hidden state.

### 2.4 Additional Efficient Coding Constraint

Although the previous variants improve the efficiency of the predictive processing part of the main model, the free part of the hidden state z is left unchanged. To force this part of the recurrent dynamics to be similarly efficient in its biologically analogous energy consumption, we can place an additional L1 norm on $z$. Although this is only an extra constraint on the model with no immediate benefit to performance, it would be interesting to see if the model is still able to perform well, even with such an energy constraint. It also provides another hyperparameter that can be tuned to optimize the model. We refer to this variant of our agent as 'P4O Constrained' throughout the following sections.

## 3 Algorithm

For the most part, the training procedure of our agents follows the standard PPO algorithm (Schulman et al., 2017). That is, the agent retrieves a batch of data by interacting with a number of parallel environments simultaneously and then updates the model by splitting the data in mini-batches and training for multiple epochs while constraining the divergence of the policy. However, since we use a recurrent main model, hidden states need to be retained during rollout to be able to update the model multiple times using the same data, and the mini-batches cannot be randomized the way they are in PPO or the temporal relationships would be lost. If we would store all hidden states for each step in the environment, without updating them during training, the hidden states would be stale after the first update. Updating multiple times in a row could then destabilize the model, since its gradients would be based on hidden states of a much older model. To minimize this effect, we only use the first hidden state of a mini-batch for each environment and generate fresh hidden states by unrolling the LSTM for the entire sequence of an environment's steps in the mini-batch at once.

For similar reasons, we also refresh the calculated advantages with the latest model before each update as suggested by Andrychowicz et al. (2020). The advantages are calculated with generalized advantage estimation (Schulman, Moritz, Levine, Jordan, & Abbeel, 2015) in its truncated form, as described by

Schulman et al. (2017). Furthermore, we add an optional action encoding as input to the main model in all variants, which can be used to inform the model of the taken action when generating future trajectories. Lastly, in standard PPO the first update is unconstrained because the batch was retrieved with the same policy that is being updated, leading to a ratio of always 1.0. To avoid the first update changing the policy too drastically, we retrieve data with the second last policy of the previous set of updates, rather than the latest policy, so that divergence from this policy can be clipped as normal even in the first update. For further details on our baseline PPO implementation, please refer to Appendix D.

For the model to naturally incorporate predictions within its hidden state, we minimize the prediction error $e$ during training. For our objective this is simply defined as follows:

$$L_t^{PP}(\theta) = \hat{E}_t[e_t^2], \tag{7}$$

where $e_t$ denotes the prediction error generated at timestep $t$. If we only use the prediction error of predicting a single step ahead, 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 process each mini-batch again, only now the model unrolls multiple steps ahead without input from the encoder, by assuming a prediction error of zero. The model can use the actions taken during rollout to adjust its predictions. We use the prediction errors from predictions at each step for the loss calculation. We then combine this predictive processing loss with the standard PPO loss components, starting with $L_t^{CLIP}(\theta)$, the typical PPO surrogate clipped actor loss (Schulman et al., 2017):

$$L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)] \tag{8}$$

where

$$r_t(\theta) = \frac{\pi_\theta(a_t s_t)}{\pi_{\theta_{old}}(a_t s_t)} \tag{9}$$

denotes the action probability ratio between the old policy and the current policy, $A_t$ is the advantage function estimator at timestep $t$ and $\epsilon$ determines the clip range. The loss component for our critic is also clipped to avoid strong divergence from the values $\hat{V}_t$ originally estimated under $\theta_{old}$:

$$V^{CLIP} = \text{clip}(V_\theta(s_t) - \hat{V}_t, -\epsilon, +\epsilon) \tag{10}$$

$$L_t^{VF}(\theta) = \hat{E}_t[|V^{CLIP}(s_t) - R_t|], \tag{11}$$

where $R_t$ denotes the return $(\hat{V}_t + A_t)$. The combined objective to minimize can then be defined as:

$$L_t(\theta) = \hat{E}_t[c_1 L_t^{CLIP}(\theta) + c_2 L_t^{VF}(\theta) + c_3 L_t^{PP}(\theta) - c_4 S[\pi_\theta](s_t)], \tag{12}$$

where $S[\pi_\theta](s_t)$ represents the entropy bonus and the loss coefficients are denoted by $c_1, c_2, c_3, c_4$. For the P4O Constrained variant we add an additional loss in the form of an L1 norm on the normal half of the hidden state $z$:

$$L_t^C(\theta) = \hat{E}_t[\|z_t\|_1], \tag{13}$$

resulting in the following total objective for the P4O Constrained variant:

$$L_t(\theta) = \hat{E}_t[c_1 L_t^{CLIP}(\theta) + c_2 L_t^{VF}(\theta) + c_3 L_t^{PP}(\theta) + c_5 L_t^C(\theta) - c_4 S[\pi_\theta](s_t)]. \tag{14}$$

# 4 Experiments

## 4.1 Experimental Setup

As mentioned before, we focus on the Atari games often used in model-free reinforcement learning research so that comparisons can easily be made. Due to computational resource limitations and the large number of variants to test, we select a single game from this set: Seaquest. Seaquest was chosen

for both its popularity among research, as well as its relatively high performance ceiling compared to games like Pong, which is easily solved by a large number of models. Additionally, 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 achieved only 25% of their human gamer normalized score, or roughly 12% of the human gamer used by Deepmind (Hafner et al., 2020).

We compare the performance of several predictive processing variants with the LSTM-PPO baseline algorithm. We evaluate the agents after 80M environment steps rather than the 40M in the original PPO paper, so that the differences between these agents can become even clearer. We use the typical frame stacking of four, 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, Wirbel, and Moutarde (2019). After training, we compare the average scores of the final 100 episodes. For most PPO-related hyperparameters we do not apply grid search, but instead use commonly reported hyperparameter values from other PPO implementations, also see Appendix C for specific hyperparameter values. We additionally run a single P4O Base agent for 10 days to compare performance with the current state-of-the-art in model-based and model-free single GPU agents.

## 4.2 P4O Base

As can be seen in Figure 3, our P4O algorithm significantly outperforms the baseline PPO algorithm ($p = .016$, $N = 14$) with a mean score of 6407 compared to our ResNet LSTM-PPO baseline's mean score of 2165, or a 196% increase in performance, without increasing the number of parameters. The original PPO paper reported a 1204 mean score on Seaquest at 40M frames, at which point both our PPO baseline implementation and the P4O agent perform better as well, as they ramp up in score much more quickly. The fact that our baseline PPO implementation outperforms the results in the original PPO paper is likely due to the modifications we made to the original implementation and the addition of a ResNet encoder and LSTM main model, as described in Appendix D. Further tuning of the hyperparameters for the P4O Base agent, such as the scaling of the predictive processing component of the loss, may lead to further performance improvements.

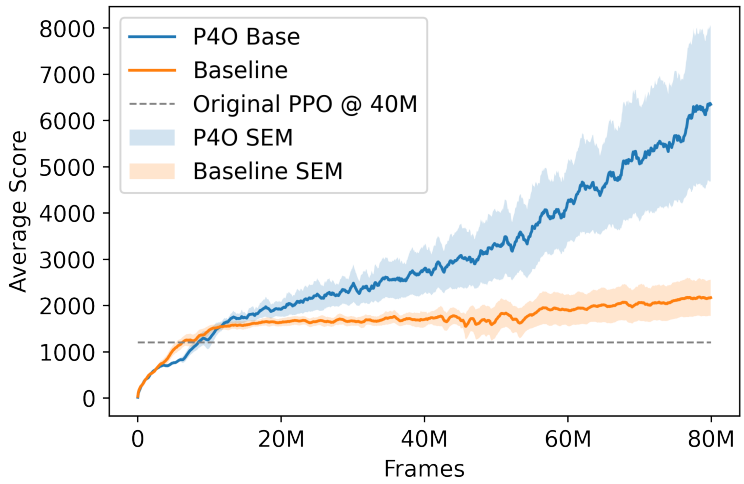


Figure 3: Average score over the last 100 episodes, comparison of the P4O Base agent, original PPO agent and our PPO baseline.

The final average score of the baseline PPO algorithm seems to be related to a particularly difficult barrier at a score of roughly 2000, preventing some agents from achieving higher scores (Figure 4B). When we look at the individual curves of each run of our P4O agent (Figure 4A), we can see this same barrier having a strong effect on the final score. Whether the agent breaks through this barrier

early or late into the run heavily influences the outcome. However, once it does, further learning of the environment is unobstructed, and scores continuously climb. Investigating the behavior of the agents around this barrier level shows that the agent at this point struggles to juggle the multiple competing goals on different temporal scales. Specifically, the agents play the game by only avoiding and destroying the enemy ships; frantically shooting while staying in the bottom half of the screen. These agents did not learn to tackle the other goals; rescuing the stranded divers and coming up for air when the oxygen level is low. However, once the P4O agents break through this barrier they learn to come up for air in time and can play for much longer, using the entire screen and rescuing divers in the process.

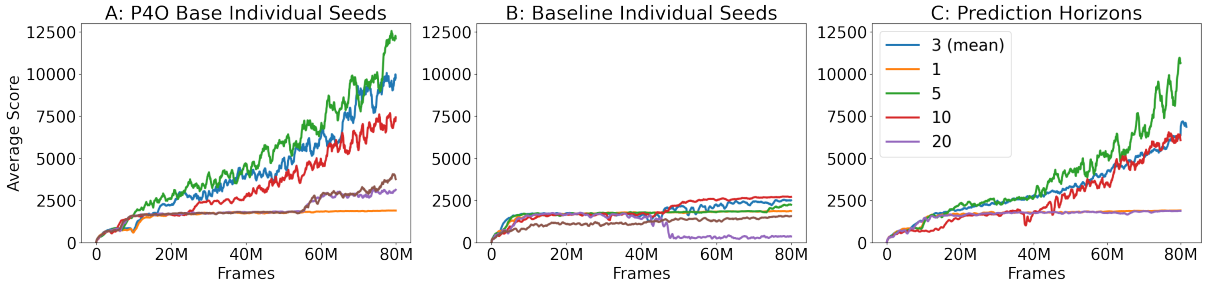


Figure 4: Comparison of individual performance curves.

## 4.3 Varying Prediction Horizons

Figure 4C shows individual runs of the P4O Integrated agent where we vary the number of steps to predict ahead during training and compare with the mean of predicting three steps ahead, the number of steps used in all other experiments. Although variance likely plays a large role in these results, it seems that predicting many more steps ahead during training of the world model does not provide consistently better results, while it comes at a slightly larger computational cost. Similarly, predicting only one step ahead seems to decrease performance, as the result is very similar to the baseline PPO algorithm. Based on this, we can infer that predicting three steps ahead is a decent choice as it seems to provide the benefits of predicting future sensory information with minimal computational impact.

## 4.4 P4O Integrated

As can be seen in Figure 5A, the P4O Integrated variant also outperforms the baseline PPO algorithm. This difference is nearly statistically significant ($p = .06$, $N = 14$) even with a sample size of 14. Even though it achieved a slightly lower average score, the P4O Integrated variant roughly approaches the performance of the P4O Base agent, despite requiring 21% fewer parameters.

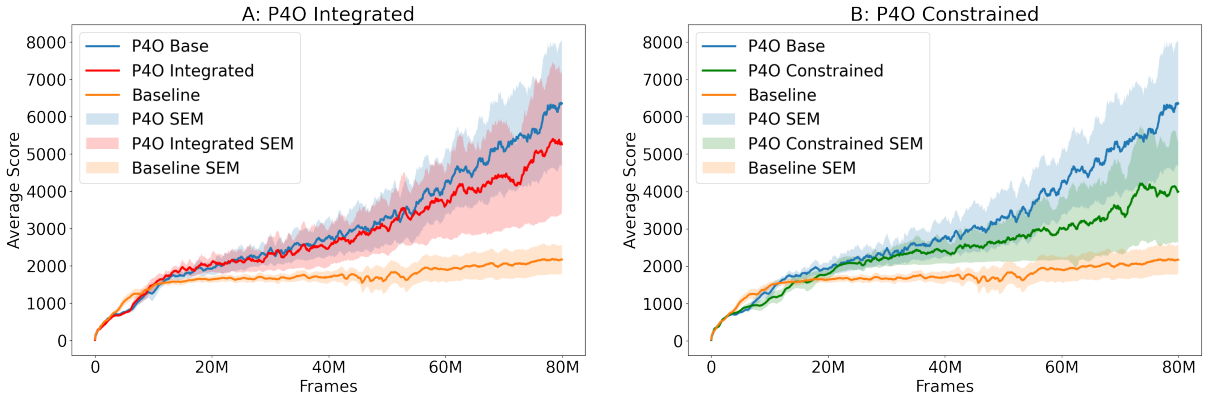


Figure 5: Comparison of different P4O variants and LSTM-PPO baseline.

Any difference in average score is well within the standard error of the mean so that many more runs would be required to determine whether this difference is statistically significant or not. This shows that predictive processing can be incorporated in a model while considerably reducing the number of parameters, in contrast to most model-based approaches that use a separate network or many additional parameters.

### 4.5 P4O Constrained

Figure 5B demonstrates how the P4O Constrained variant that applies an additional L1 norm loss component also outperforms the baseline PPO algorithm. However, this difference is not statistically significant with the current number of runs ($p = .16$, $N = 14$). The additional L1 norm used in the P4O Constrained variant results in a reduced average score compared to the P4O Base agent although this difference is also not statistically significant ($p = .30$) due to the low sample size ($N = 12$). We do note that the P4O Constrained variant only learned to go up for oxygen and break through the 2000 barrier two times out of six runs given 80M frames. On the other hand, the P4O base agent was able to break through this barrier in five out of six runs given the same amount of frames. This may indicate that the 0.1 scale used on the L1 norm loss component affects the learning speed of the model. Tuning this parameter might alleviate this negative effect. At the same time, in certain biological or neuromorphic systems, the benefit of an analogous energy constraint on neuronal firing might outweigh a difference in speed at which the model learns. Given that the agent still outperforms the baseline PPO algorithm despite such a constraint, we consider this is worth further investigation in the context of efficient coding.

### 4.6 Comparison with state-of-the-art

To place the performance of our agent in a broader context, a 10 days long run of the P4O Base agent is compared with a number of current state-of-the-art single GPU reinforcement learning agents in Figure 6.

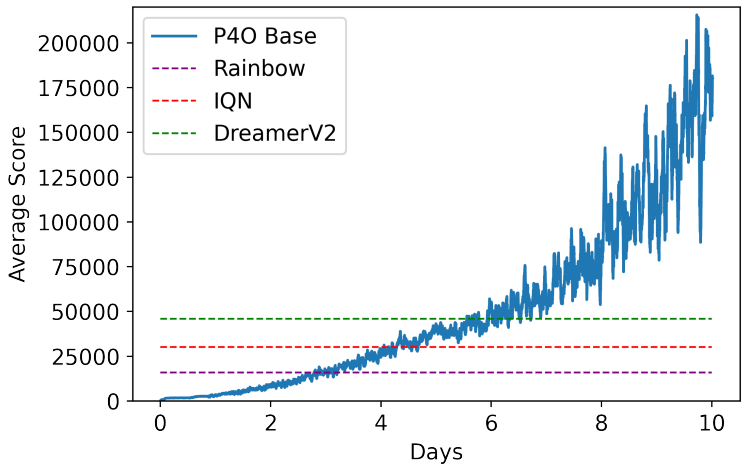


Figure 6: Initial single seed performance comparison with IQN, Rainbow and DreamerV2 after 10 days of accelerator time. Dashed lines represent final reported scores of the IQN, Rainbow and DreamerV2 agents after 10 days. Reported average score is a rolling mean of the last 100 episodes.

The model-based DreamerV2 agent and model-free Rainbow and IQN agents all report their performance after 10 accelerator days in wall-clock time (Dabney, Ostrovski, Silver, & Munos, 2018; Hafner et al., 2020; Hessel et al., 2018). Each of these three models has processed 200M Atari frames at this point, whereas our P4O agent is able to process 1.2B frames in 10 days with the same Nvidia Tesla V100 GPU as used by DreamerV2 (Table 1). Furthermore, the DreamerV2 agent uses 22M parameters, whereas our P4O Base agent uses 9.6M parameters, and 7.6M for the P4O Integrated variant. As can be seen in Figure 6, this single run of our agent surpasses the Rainbow agent's final score in less than 3 days, surpasses the IQN agent after 5 days, and the DreamerV2 average after roughly 6 days, achieving

a final average score of 180054 over the last 100 episodes, or 428% of the human gamer score reported by Hafner et al. (2020).

| Agent | Atari Frames | Accelerator Days | Average Score | Gamer-Normalized Score |
|---|---|---|---|---|
| Rainbow | 200M | 10 | 15898 | 0.37 |
| IQN | 200M | 10 | 30140 | 0.71 |
| DreamerV2 | 200M | 10 | 45898 | 1.09 |
| P4O | **1.2B** | 10 | **180054 (727033)** | **4.28 (17.28)** |

Table 1: Comparison of our P4O Base agent with top single GPU agents on Seaquest after 10 days of accelerator time (Dabney et al., 2018; Hafner et al., 2020; Hessel et al., 2018). 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).

The highest score achieved by the agent was 999999; the maximum score possible in the game. 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 high 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. To extract the maximum performance out of 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 this way in another 100 episodes leads to a much higher average score of 727033, or 1728% of the human gamer score. Further inspection shows that the agent achieved the maximum score in 70% of these episodes. Given this result, it may be beneficial to apply a decay factor to the entropy bonus to allow the model to already become more deterministic towards the end of training.

We note that these results are an initial comparison with a single long run on Seaquest due to limitations in compute resources. To exhaustively compare these algorithms, multiple runs on the entire set of Atari games would be required. One additional difference with the Rainbow and DreamerV2 agents in particular is that these agents enforce a frame limit for each episode equivalent to 30 minutes of gameplay. This could potentially affect scores, however, in Seaquest their reported scores should not be affected by this limit, since episodes with such a score are significantly shorter than 30 minutes. Another potential limitation is that Seaquest might be particularly suited to our model, as it forces the agent to deal with competing goals on different timescales. The performance benefit of our approach may therefore not directly translate to all Atari games, which can be investigated in future research. Finally, further tuning of the hyperparameters may lead to enhanced performance, for example by adjusting the learning rate, the entropy bonus or other loss coefficients.

## 4.7 Neural coding

When we inspect the effect of predicting future states on our neural coding, we can see a clear effect (Figure 7). In the traditional baseline PPO algorithm, the output of the encoder shows a typical tanh activation profile, in other words almost all values are grouped at the extremes; -1 and 1. On the other hand, when we look at the neural coding of our P4O algorithm, we observe a very different distribution. Despite the tanh activation, the values remain grouped around the zero point and do not approach the extremes. The prediction contained in the hidden state shows a very similar distribution with values roughly between -0.7 and 0.7. The resulting prediction error is thus even more centered around the zero point, with values mostly distributed between -0.05 and 0.05. The coefficient of determination ($R^2$) of the prediction with respect to the encoded input is 0.86 for the particular run used in Figure 7, 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.

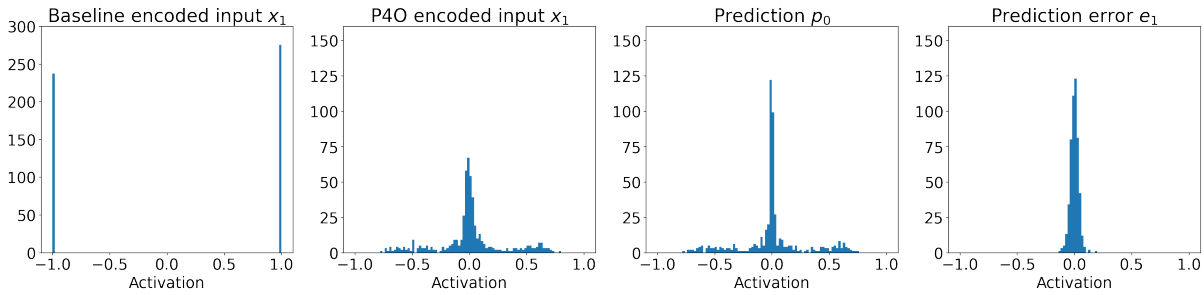


Figure 7: Comparison of activation distributions between the baseline latent encoding, the P4O latent encoding, the P4O prediction and the P4O prediction error. The distributions are a typical encoding of a randomly selected single state.

As larger activation values in neural coding can be related to energy consumption in biological neurons in terms of spiking frequency, one interpretation of this difference is that our P4O model is much more energy efficienct, in line with expectations from efficient coding literature.

# 5 Discussion

We have demonstrated that predictive processing with a world model can be integrated into an artificial agent without increasing its biologically analogous energy footprint while leading to significant performance gains. Furthermore, we showed that when we integrate the prediction error in the form of inhibition on recurrent connections, the number of neurons in the model can even be reduced compared to a model-free approach, in stark contrast to the typical model-based approaches that use an entirely separate network for their world models. The fact that our agent achieved significantly better performance compared to the model-free variant implies that the need to predict future states entices the model to extract features that are relevant in understanding temporal relationships and that these features are more informative for the policy than the features normally extracted. Our model also demonstrates that prediction error is all that is required as an input to the main model to achieve state-of-the-art performance. The model can operate unhindered by the lack of direct access to the actual encoded sensory information. These findings support predictive coding theory, as our approach works fundamentally similarly to what it proposes. The fact that it is possible to learn complex behavior in such an environment and exceed the performance of a human gamer further reinforces the idea that the human brain may use a similar approach. While we do not hard-code a hierarchy of multiple levels of predictive processing, the ability of the agent to deal with multiple competing goals on different temporal scales supports the suggestion that an LSTM can integrate information over long timescales (Lu, Hasson, & Norman, 2020) and suggests that a temporal hierarchy may have self-organized (Paine & Tani, 2005; Yamashita & Tani, 2008) within its neural dynamics through shifting neural states, as in the human brain (Geerligs, van Gerven, Campbell, & Güçlü, 2021; Geerligs, van Gerven, & Güçlü, 2021). Future studies could reveal how the model balances these competing goals at different timescales and whether such a temporal hierarchy has self-organized. The agent could also be modified in later research to encourage exploration through the use of its prediction error, for example by adding a bias towards areas that generate surprise.

Additionally, the P4O agent supports the efficient coding hypothesis in three distinct ways. First, the ability to incorporate a prediction of its sensory information inside existing recurrent connections without adding parameters, or extending the model in other ways, maximizes the information stored in the model. Second, when we use sensory information exclusively as an inhibitory signal on these recurrent connections, we can reduce the size of the model, since we remove the need to provide a separate input signal. Third, the minimization of prediction error entices the model to encode its sensory information in an efficient way, centering the activation values around the zero point, which would be biologically analogous to a lower spike firing rate and reduced energy usage. We should note, however, that our approach does require the number of neurons used in the hidden state of the LSTM to be large enough to encode its sensory information prediction while leaving space for normal usage of the rest of the hidden state.

In an initial single run performance comparison, the P4O agent outperformed other model-free and model-based state-of-the-art single GPU reinforcement learning agents' final scores reported after 10 days of wall-clock time on Seaquest. Wall-clock time is arguably the most limiting factor in reinforcement learning research, and therefore should be prioritized as a metric besides cumulative reward. On the other hand, although it requires six-fold the wall-clock time to process the same amount of Atari frames, the DreamerV2 agent (Hafner et al., 2020) achieves better sampling efficiency; performance per frames observed. It accomplishes this by making extensive use of its world model to predict entire trajectories, and train its policy on these imagined trajectories rather than the actual environment. Future research could extend our agent with such an approach and investigate how much sampling efficiency could be gained without sacrificing wall-clock time or requiring significantly more compute resources. A balance could likely be struck where most of the benefit in terms of sampling efficiency is combined with most of the benefit in terms of wall-clock time, overall efficiency and compute requirements of our agent. Future studies could also further investigate the performance of the P4O agent across the entire domain of Atari games and other environments.

As discussed, the current trend in state-of-the-art reinforcement learning has been excessively complex, biologically implausible and computationally intensive multi GPU agents (Badia et al., 2020; Ecoffet et al., 2019; Schrittwieser et al., 2020). 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 world model with biologically plausible elements that boost efficiency and performance. Improvements in performance without sacrificing efficiency, or even improving efficiency, are vital to future progress in the field and allow reinforcement learning to be more widely applicable. Furthermore, when reinforcement learning can be done more efficiently, this creates computational room for more intelligent systems, for example, to facilitate transfer learning, meta learning or behavior in much more complex environments. Besides improving the current artificial agents, the benefit of a more biologically plausible approach is that it is naturally more likely to accurately describe how the human brain tackles similar problems. This can help further understand human action planning and guide research and exploration in the field of neuroscience, highlighting the synergy between the fields of neuroscience, cognitive science and artificial intelligence.

## Acknowledgements

I would like to thank prof. Marcel van Gerven for his stellar supervision and suggestions that led to this thesis, Burcu Küçükoğlu for her in-depth feedback on the baseline code and my fiancée Hyemi for her continuous support and insightful comments.

## References

Ali, A., Ahmad, N., de Groot, E., van Gerven, M. A. J., & Kietzmann, T. C. (2021). Predictive coding is a consequence of energy efficiency in recurrent neural networks. *bioRxiv 2021.02.16.430904*.

Alink, A., Schwiedrzik, C. M., Kohler, A., Singer, W., & Muckli, L. (2010). Stimulus predictability reduces responses in primary visual cortex. *Journal of Neuroscience*, *30*(8), 2960–2966.

Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., ... others (2020). What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*.

Babaeizadeh, M., Finn, C., Erhan, D., Campbell, R. H., & Levine, S. (2017). Stochastic variational video prediction. *arXiv preprint arXiv:1710.11252*.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., & Blundell, C. (2020). Agent57: Outperforming the atari human benchmark. In *International conference on machine learning* (pp. 507–517).

Barlow, H. B. (1961). Possible principles underlying the transformation of sensory messages. In W. A. Rosenblith (Ed.), *Sensory communication* (p. 217-234). Cambridge, MA: MIT Press.

Bell, A. J., & Sejnowski, T. J. (1995). An information-maximization approach to blind separation and blind deconvolution. *Neural computation*, *7*(6), 1129–1159.

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

Berkes, P., & Wiskott, L. (2005). Slow feature analysis yields a rich repertoire of complex cell properties. *Journal of vision*, *5*(6), 9–9.

Bialek, W., Van Steveninck, R. R. D. R., & Tishby, N. (2006). Efficient representation as a design principle for neural coding and computation. In *2006 ieee international symposium on information theory* (pp. 659–663).

Buckman, J., Hafner, D., Tucker, G., Brevdo, E., & Lee, H. (2018). Sample-efficient reinforcement learning with stochastic ensemble value expansion. *arXiv preprint arXiv:1807.01675*.

Buesing, L., Weber, T., Racaniere, S., Eslami, S., Rezende, D., Reichert, D. P., ... others (2018). Learning and querying fast generative models for reinforcement learning. *arXiv preprint arXiv:1802.03006*.

Chalk, M., Marre, O., & Tkačik, G. (2018). Toward a unified theory of efficient, predictive, and sparse coding. *Proceedings of the National Academy of Sciences*, *115*(1), 186–191.

Chiappa, S., Racaniere, S., Wierstra, D., & Mohamed, S. (2017). Recurrent environment simulators. *arXiv preprint arXiv:1704.02254*.

Chua, K., Calandra, R., McAllister, R., & Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*.

Ciria, A., Schillaci, G., Pezzulo, G., Hafner, V. V., & Lara, B. (2021). Predictive processing in cognitive robotics: a review. *Neural Computation*, *33*(5), 1402–1432.

Clark, A. (2013). Whatever next? predictive brains, situated agents, and the future of cognitive science. *Behavioral and brain sciences*, *36*(3), 181–204.

Dabney, W., Ostrovski, G., Silver, D., & Munos, R. (2018). Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning* (pp. 1096–1105).

De Lange, F. P., Heilbron, M., & Kok, P. (2018). How do expectations shape perception? *Trends in*

*cognitive sciences*, *22*(9), 764–779.

Denton, E., & Fergus, R. (2018). Stochastic video generation with a learned prior. In *International conference on machine learning* (pp. 1174–1183).

Dijkstra, N., Ambrogioni, L., Vidaurre, D., & van Gerven, M. (2020). Neural dynamics of perceptual inference and its reversal during imagery. *Elife*, *9*, e53588.

Doerr, A., Daniel, C., Schiegg, M., Duy, N.-T., Schaal, S., Toussaint, M., & Sebastian, T. (2018). Probabilistic recurrent state-space models. In *International conference on machine learning* (pp. 1280–1289).

Eckmann, S., Klimmasch, L., Shi, B. E., & Triesch, J. (2020). Active efficient coding explains the development of binocular vision and its failure in amblyopia. *Proceedings of the National Academy of Sciences*, *117*(11), 6156–6162.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., & Clune, J. (2019). Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*.

Ekman, M., Kok, P., & de Lange, F. P. (2017). Time-compressed preplay of anticipated events in human primary visual cortex. *Nature Communications*, *8*(1), 1–9.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, *14*(2), 179–211.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., ... others (2018). Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning* (pp. 1407–1416).

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

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

Gal, Y., McAllister, R., & Rasmussen, C. E. (2016). Improving pilco with bayesian neural network dynamics models. In *Data-efficient machine learning workshop, icml* (Vol. 4, p. 25).

Geerligs, L., van Gerven, M., Campbell, K., & Güçlü, U. (2021). A nested cortical hierarchy of neural states underlies event segmentation in the human brain. *bioRxiv 2021.02.05.429165*.

Geerligs, L., van Gerven, M., & Güçlü, U. (2021). Detecting neural state transitions underlying event segmentation. *NeuroImage*, 118085.

Gemici, M., Hung, C.-C., Santoro, A., Wayne, G., Mohamed, S., Rezende, D. J., ... Lillicrap, T. (2017). Generative temporal models with memory. *arXiv preprint arXiv:1702.04649*.

Gershman, S. J., Horvitz, E. J., & Tenenbaum, J. B. (2015). Computational rationality: A converging paradigm for intelligence in brains, minds, and machines. *Science*, *349*(6245), 273–278.

Gregor, K., Papamakarios, G., Besse, F., Buesing, L., & Weber, T. (2018). Temporal difference varia-

tional auto-encoder. *arXiv preprint arXiv:1806.03107*.

Güçlü, U., & van Gerven, M. A. (2017). Modeling the dynamics of human brain activity with recurrent neural networks. *Frontiers in computational neuroscience*, *11*, 7.

Ha, D., & Schmidhuber, J. (2018a). Recurrent world models facilitate policy evolution. *arXiv preprint arXiv:1809.01999*.

Ha, D., & Schmidhuber, J. (2018b). World models. *arXiv preprint arXiv:1803.10122*.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., & Davidson, J. (2019). Learning latent dynamics for planning from pixels. In *International conference on machine learning* (pp. 2555–2565).

Hafner, D., Lillicrap, T., Norouzi, M., & Ba, J. (2020). Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*.

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

Henaff, M., Whitney, W. F., & LeCun, Y. (2017). Model-based planning with discrete and continuous actions. *arXiv preprint arXiv:1705.07177*.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., ... Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 32).

Higuera, J. C. G., Meger, D., & Dudek, G. (2018). Synthesizing neural network controllers with probabilistic model-based reinforcement learning. In *2018 ieee/rsj international conference on intelligent robots and systems (iros)* (pp. 2538–2544).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hupé, J., James, A., Payne, B., Lomber, S., Girard, P., & Bullier, J. (1998). Cortical feedback improves discrimination between figure and background by v1, v2 and v3 neurons. *Nature*, *394*(6695), 784–787.

Igl, M., Zintgraf, L., Le, T. A., Wood, F., & Whiteson, S. (2018). Deep variational reinforcement learning for pomdps. In *International conference on machine learning* (pp. 2117–2126).

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

Kalweit, G., & Boedecker, J. (2017). Uncertainty-driven imagination for continuous deep reinforcement learning. In *Conference on robot learning* (pp. 195–206).

Karl, M., Soelch, M., Bayer, J., & Van der Smagt, P. (2016). Deep variational bayes filters: Unsupervised learning of state space models from raw data. *arXiv preprint arXiv:1605.06432*.

Kok, P., Jehee, J. F., & De Lange, F. P. (2012). Less is more: expectation sharpens representations in the primary visual cortex. *Neuron*, *75*(2), 265–270.

Krishnan, R. G., Shalit, U., & Sontag, D. (2015). Deep kalman filters. *arXiv preprint arXiv:1511.05121*.

Kurutach, T., Clavera, I., Duan, Y., Tamar, A., & Abbeel, P. (2018). Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*.

Lee, A. X., Nagabandi, A., Abbeel, P., & Levine, S. (2019). Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *arXiv preprint arXiv:1907.00953*.

Lee, T. S., & Mumford, D. (2003). Hierarchical bayesian inference in the visual cortex. *JOSA A*, *20*(7), 1434–1448.

Lu, Q., Hasson, U., & Norman, K. A. (2020). Learning to use episodic memory for event prediction. *bioRxiv 2020.12.15.422882*.

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

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... others (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529–533.

Moser, M.-B., Rowland, D. C., & Moser, E. I. (2015). Place cells, grid cells, and memory. *Cold Spring Harbor perspectives in biology*, *7*(2), a021808.

Mumford, D. (1992). On the computational architecture of the neocortex. *Biological cybernetics*, *66*(3), 241–251.

Murray, S. O., Kersten, D., Olshausen, B. A., Schrater, P., & Woods, D. L. (2002). Shape perception reduces activity in human primary visual cortex. *Proceedings of the National Academy of Sciences*, *99*(23), 15164–15169.

Näätänen, R., Tervaniemi, M., Sussman, E., Paavilainen, P., & Winkler, I. (2001). 'primitive intelligence'in the auditory cortex. *Trends in neurosciences*, *24*(5), 283–288.

Nagabandi, A., Kahn, G., Fearing, R. S., & Levine, S. (2018). Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 ieee international conference on robotics and automation (icra)* (pp. 7559–7566).

Oh, J., Guo, X., Lee, H., Lewis, R., & Singh, S. (2015). Action-conditional video prediction using deep networks in atari games. *arXiv preprint arXiv:1507.08750*.

Olshausen, B. A., & Field, D. J. (1996). Natural image statistics and efficient coding. *Network: computation in neural systems*, *7*(2), 333–339.

Paine, R. W., & Tani, J. (2005). How hierarchical control self-organizes in artificial adaptive systems. *Adaptive Behavior*, *13*(3), 211–225.

Rao, H. M., Mayo, J. P., & Sommer, M. A. (2016). Circuits for presaccadic visual remapping. *Journal*

*of Neurophysiology*, *116*(6), 2624–2636.

Rao, R. P., & Ballard, D. H. (1999). Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience*, *2*(1), 79–87.

Rosenblith, W. A. (1961). *Sensory communication*. The MIT Press.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation* (Tech. Rep.). California Univ San Diego La Jolla Inst for Cognitive Science.

Salimans, T., & Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *arXiv preprint arXiv:1602.07868*.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., ... others (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, *588*(7839), 604–609.

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

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

Squires, N. K., Squires, K. C., & Hillyard, S. A. (1975). Two varieties of long-latency positive waves evoked by unpredictable auditory stimuli in man. *Electroencephalography and clinical neurophysiology*, *38*(4), 387–401.

Srinivas, A., Jabri, A., Abbeel, P., Levine, S., & Finn, C. (2018). Universal planning networks: Learning generalizable representations for visuomotor control. In *International conference on machine learning* (pp. 4732–4741).

Srinivasan, M. V., Laughlin, S. B., & Dubs, A. (1982). 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.

Summerfield, C., Trittschuh, E. H., Monti, J. M., Mesulam, M.-M., & Egner, T. (2008). Neural repetition suppression reflects fulfilled perceptual expectations. *Nature neuroscience*, *11*(9), 1004.

Toromanoff, M., Wirbel, E., & Moutarde, F. (2019). Is deep reinforcement learning really superhuman on atari? leveling the playing field. *arXiv preprint arXiv:1908.04683*.

Van Gerven, M. (2017). Computational foundations of natural intelligence. *Frontiers in computational neuroscience*, *11*, 112.

Wang, T., & Ba, J. (2019). Exploring model-based planning with policy networks. *arXiv preprint arXiv:1906.08649*.

Wang, T., Bao, X., Clavera, I., Hoang, J., Wen, Y., Langlois, E., ... Ba, J. (2019). Benchmarking

model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*.

Watter, M., Springenberg, J. T., Boedecker, J., & Riedmiller, M. (2015). Embed to control: A locally linear latent dynamics model for control from raw images. *arXiv preprint arXiv:1506.07365*.

Wayne, G., Hung, C.-C., Amos, D., Mirza, M., Ahuja, A., Grabska-Barwinska, A., ... others (2018). Unsupervised predictive memory in a goal-directed agent. *arXiv preprint arXiv:1803.10760*.

Weber, T., Racanière, S., Reichert, D. P., Buesing, L., Guez, A., Rezende, D. J., ... others (2017). Imagination-augmented agents for deep reinforcement learning. *arXiv preprint arXiv:1707.06203*.

Whitehead, S. D., & Ballard, D. H. (1991). Learning to perceive and act by trial and error. *Machine Learning*, *7*(1), 45–83.

Yamashita, Y., & Tani, J. (2008). Emergence of functional hierarchy in a multiple timescale neural network model: A humanoid robot experiment. *PLOS Computational Biology*, *4*(11).

# A 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.

# B Model Architecture

Our model consists of a ResNet encoder and an LSTM-based main model. The original Atari frames of 210 by 160 pixels in RGB color are converted to the commonly used 84 by 84 pixels grayscale format. After frame-stacking four times, the final input is four channels of 84 by 84.

In the ResNet encoder, we define a residual block as two convolutional layers each preceded by a ReLU activation, where the original input is added back to the output, in the preactivation configuration as suggested by He, Zhang, Ren, and Sun (2016). Note that we do not use batch normalization because Salimans and Kingma (2016) argued that batch normalization can destabilize the learning process in a reinforcement learning context. Our overall encoder configuration is very similar to the encoder used by Espeholt et al. (2018), albeit with a different number of channels and groups, as well as an additional ReLU activation function preceding each group and a tanh final activation function. The preceding ReLU is used to ensure there is an activation function between convolutional layers of consecutive groups. The ResNet model uses four groups of a convolutional layer, a max-pooling layer and two residual blocks, where the number of channels for all layers in a group is the same. The number of channels per group 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, except for the max-pooling layers, where the stride is two. Kernel size, padding and strides are the same in both dimensions. The output of the ResNet architecture is 128 channels with 6 by 6 dimensions. The final layer of the encoder is a densely connected layer containing 512 neurons with tanh activation.

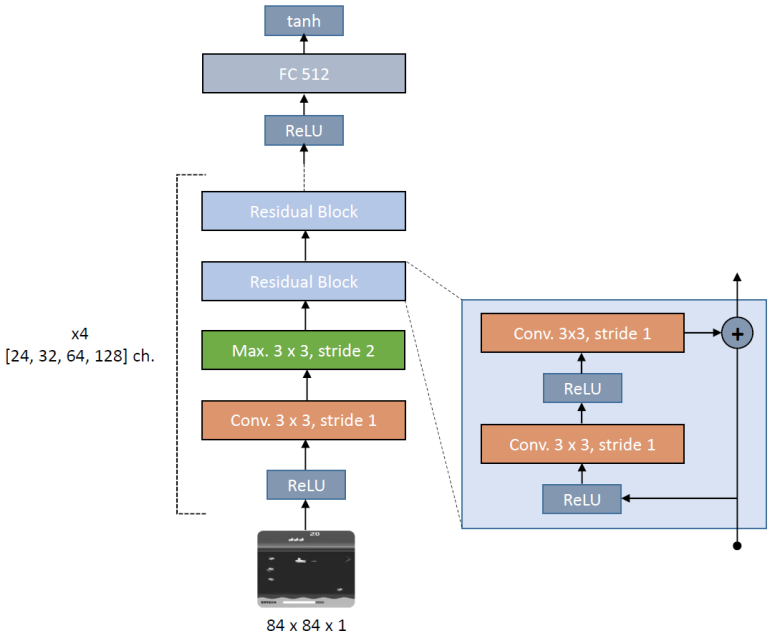


Figure 8: Encoder architecture, following a similar structure to the architecture used by Espeholt et al. (2018) with a few modifications. The encoder uses a total of 20 convolutional layers and 3.3M parameters.

For the baseline main model, we apply a simple LSTM with 1024 neurons and outputs for value prediction and softmax action probabilities. For our P4O models, we modify the LSTM significantly. First, we reserve half of the LSTM hidden state to represent input predictions. In the P4O Base model, we subtract the actual input from the input prediction and provide this prediction error signal as input to the model, along with a single action encoding. In the P4O Integrated variant, we again subtract the actual input from the input prediction, but this time transform this part of the hidden state to the prediction error signal. We replace the normal input in the LSTM with a vector that only provides space for a one-hot encoding of an action. The weight matrices and gating functions of the LSTM are therefore applied to the normal half of the hidden state, the prediction error and an action vector. Note that this means we do not provide the input on its own, thus the only way the model will receive information about its true state is through the prediction error incorporated in the hidden state. During training, we minimize the prediction error segment of the hidden state. The direct output hidden state of the LSTM then again consists of half an unrestricted hidden state and the other half represents the new input prediction. For the P4O Constrained variant, we use the same model as the P4O Integrated variant, but add an additional L1 loss component to the normal part of the hidden state during training.

# C Hyperparameters

| Hyperparameter | Value |
|---|---|
| Learning rate | $2.5 \times 10^{-4}$ |
| Optimizer | Adam |
| Adam ($\epsilon$) | $1 \times 10^{-5}$ |
| Num. parallel environments | 16 |
| Mini-batch size | 400 |
| Discount ($\gamma$) | 0.99 |
| GAE parameter ($\lambda$) | 0.95 |
| PPO clip range ($\epsilon$) | 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.

# D Baseline PPO Algorithm

While the standard PPO algorithm with a clipped objective as described in the original paper works well as is, due to its popularity a number of best practices and adjustments have appeared in other studies and online resources. We have incorporated a number of these improvements in our baseline model, as well as a few of our own modifications. The foundation of the algorithm is PPO with generalized advantage estimation (Schulman et al., 2015) combined with a ResNet encoder and an LSTM as our final layer. The final layer of the ResNet encoder is a densely connected layer with tanh activation to be compatible with our LSTM output. Besides clipping the actor loss, we also clip the critic to stabilize the critic further and avoid sudden large deviations from the previous model. We apply an entropy term to encourage exploration and avoid the model becoming too deterministic early on. As mentioned by Andrychowicz et al. (2020), in standard PPO the advantages are calculated as the last step of retrieving a batch of data, resulting in stale advantages after a few epochs with the same data. To avoid this, we recalculate the advantages for every update with the new model rather than once at the end of a batch rollout. We apply a similar approach to refresh the hidden states for the LSTM at every update. We also standardize the advantages to have a mean of zero and a standard deviation of one. For further stability, we clip the gradients before every update. The learning rate decays linearly relative to the total number of updates and ends at zero, except for the long run where we decay with a constant factor of 0.995 every 100 batches to a minimum learning rate of $5 \times 10^{-6}$.

An additional observation about the original PPO algorithm that we addressed in our implementation is that the first update in a set of updates on a batch of data is unconstrained. This occurs because the policy in the first update is the same as the policy that generated the data, and therefore the ratio between these policies is always one, allowing this update to be unconstrained. Since training should be more stable if we also constrain the first update, we have done so by rolling out with the second-last policy parameters from the previous set of updates, rather than the latest policy. The latest policy can then be used for the first update with the new set of data and will be constrained by its ratio compared to the previous model, as described in Algorithm 1.

Algorithm 1 Modified PPO

for iteration $= 1, 2, \ldots$ do
  for actor $= 1, 2, \ldots, N$ do
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for $T$ timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate $L$ wrt $\theta$, with $K$ epochs and minibatch size $M \leq NT$ for a total of $U$ updates, generating $\theta_1, \ldots, \theta_U$
  $\theta_{\text{old}} \leftarrow \theta_{U-1}$
  $\theta \leftarrow \theta_U$
end for

(15)