Value-driven Hindsight Modelling

Arthur Guez [1] Fabio Viola [1] Théophane Weber [1] Lars Buesing [1]
Steven Kapturowski [1] Doina Precup [1] David Silver [1] Nicolas Heess [1]

Abstract

Value estimation is a critical component of the reinforcement learning (RL) paradigm. The question of how to effectively learn predictors for value from data is one of the major problems studied by the RL community, and different approaches exploit structure in the problem domain in different ways. Model learning can make use of the rich transition structure present in sequences of observations, but this approach is usually not sensitive to the reward function. In contrast, model-free methods directly leverage the quantity of interest from the future but have to compose with a potentially weak scalar signal (an estimate of the return). In this paper we develop an approach for representation learning in RL that sits in between these two extremes: we propose to learn what to model in a way that can directly help value prediction. To this end we determine which features of the *future* trajectory provide useful information to predict the associated return. This provides us with tractable prediction targets that are directly relevant for a task, and can thus accelerate learning of the value function. The idea can be understood as reasoning, in hindsight, about which aspects of the *future* observations could help *past* value prediction. We show how this can help dramatically even in simple policy evaluation settings. We then test our approach at scale in challenging domains, including on 57 Atari 2600 games.

1. Introduction

Consider a baseball player trying to perfect their pitch. The player performs an arm motion and releases the ball towards the batter, but suppose that instead of observing where the ball lands and the reaction of the batter, the player only gets told the result of the play in terms of points or, worse, only gets told the final result of the game. Improving their pitch from this experience appears hard and inefficient, yet this is essentially the paradigm we employ when optimizing policies in model-free reinforcement learning. The scalar feedback that estimates the return from a state (and action), encoding *how well things went*, drives the learning while the accompanying observations that may explain that result (e.g., flight path of the ball or the way the batter anticipated and struck the incoming baseball) are ignored. To intuitively understand how such information could help value prediction, consider a simple discrete Markov chain $X \to Y \to Z$, where Z is the scalar return and X is the observation from which we are trying to predict Z. If the space of possible values of Y is smaller than X, then it may be more efficient to estimate both $P(Y|X)$ and $P(Z|Y)$ rather than directly estimating $P(Z|X)$.[1] In other words observing and then predicting Y can be advantageous to directly estimating the signal of interest Z. Model-based RL approaches would duly exploit the observed Y (by modelling the transition $Y|X$), but Y would, in general scenarios, contain information that is irrelevant to Z and hard to predict. Building a full high-dimensional predictive model to indiscriminately estimate all possible future observations, including potentially chaotic details of the ball trajectory and the spectators' response, is a challenge that may not pay off if the task-relevant predictions (e.g., was the throw accepted, was the batter surprised) are error-ridden. Model-free RL methods directly consider the relation X to Z, and focus solely upon predicting and optimising this goal, rather than attempting to learn the full dynamics. These methods have recently dominated the literature, and have attained the best performance in a wide array of complex problems with high-dimensional observations (Mnih et al., 2015; Schulman et al., 2017; Haarnoja et al., 2018; Guez et al., 2019).

In this paper, we propose to augment model-free methods with a lightweight model of future quantities of interest. The motivation is to model only those parts of the future observations (Y) that are needed to obtain better value predictions. The major research challenge is to learn, from observational data, which aspects of the future are important to model (i.e. what Y should be). To this end, we propose to learn

[1]DeepMind. Correspondence to: Arthur Guez <aguez@google.com>.

[1]In the discrete case, this follows from a counting argument from the size of the probability tables involved. See appendix A.1.

a special value function *in hindsight* that receives future observations as an additional input. This learning process reveals features of the future observations that would be most useful for value prediction (e.g., flight path of the ball or the reaction of the batter), if provided by an oracle. These important features are then predicted, in advance, using only information available at test time (at the time of releasing the baseball, we knew the identity of the batter, the type of throw and spin given to the ball). Learning these value-relevant features can help representation learning for an agent and provide an additional useful input to its value and policy functions. Experimentally, hindsight value functions surpassed the performance of model-free RL methods in a challenging association task (Portal Choice, see section 5.1). When hindsight value functions were added to the prior state-of-the-art model-free RL method for Atari games, they significantly increased median performance from 833% to 965%.

2. Background and Notation

We consider a reinforcement learning setting whereby an agent learns from interaction in a sequential decision-making environment (Sutton & Barto, 2011). An agent's policy π, mapping states to an action distribution, is executed to obtain a sequence of rewards and observations as follows. At each step t, after observing state s_t, the policy outputs an action a_t, sampled from $\pi(A|s_t)$, and obtains a scalar reward r_t and the next-state s_{t+1} from the environment. The sum of discounted rewards from state s is the return denoted by $G = \sum_{t=0}^{\infty} \gamma^t R_t$, with $\gamma < 1$ denoting the discount factor. Its expectation, as a function of the starting state, is called the value function, $v^\pi(s) = \mathbb{E}_\pi[G|S_0 = s]$. An important related quantity is the action-value, or Q-value, which corresponds to the same expectation with a particular action executed first: $q^\pi(s,a) = \mathbb{E}_\pi[G|S_0 = s, A_0 = a]$. The learning problem consists in adapting the policy π in order to achieve a higher value v^π. This usually entails learning an estimate of v^π for the current policy π, this is the problem we focus on in this paper.

Note that in practice we are interested in partially-observed environments where the state of the world is not directly accessible. For this case, we can think of replacing the observed state s in the case of the fully-observed case by a learned function that depends on past observations.

3. Value Learning

3.1. Direct Learning

A common approach to estimate v (or q) is to represent it as a parametric function v_θ (or q_θ) and directly update its parameters based on sample returns of the policy of interest. Value-based RL algorithms vary in how they construct a value target Y from a single trajectory. They may regress v_θ towards the Monte-Carlo return ($Y_t = G_t$), or exploit sequentiality in the reward process by relying on a form of temporal-difference learning to reduce variance (e.g., the TD(0) target $Y_t = R_t + \gamma v_\theta(S_{t+1})$). For a given target definition Y, the value loss $\mathcal{L}_v$ to derive an update for θ is: $\mathcal{L}_v(\theta) = \frac{1}{2}\mathbb{E}_s[(v_\theta(s) - Y)^2]$. In constructing a target Y_t based on a trajectory of observations and rewards from time t, the observations are either unused (for a Monte Carlo return) or only indirectly exploited (when bootstrapping to obtain a value estimate, we come back to this point in Section 3.6). In all cases, the trajectory is distilled into a scalar signal that estimates the return of a policy, and other relevant aspects of future observations are discarded. In partially observed domains or domains with high-dimensional observation spaces, it can be particularly difficult to discover correlations with this noisy, possibly sparse, signal.

3.2. Model-based approach

An indirect way of estimating the value is to first learn a model of the dynamics. For example a 1-step observation model m_θ learns to predict the conditional distribution $s_{t+1}, r_t|s_t, a_t$. Then a value estimate $v(s)$ for state s can be obtained by autoregressively rolling out the model (until the end of the episode or to a fixed depth with a parametric value bootstrap).

The model is trained on potentially much richer data than the return signal since it exploits all information in the trajectory. Indeed, the observed transitions between states can reveal the structure behind a sparse reward signal. A drawback of classic model-based approaches is that they predict a high-dimensional signal, a task which may be costly and harder than directly predicting the scalar value. As a result, the approximation of the dynamics m_θ may contain errors where it matters most for predicting the value (Talvitie, 2014). Although the observations carry all the data from the environment, most of it is not essential to the task (Gelada et al., 2019). The concern that modelling all observations is expensive also applies when the model is not used for actual rollouts but merely for representation learning. So while classic model-based methods fully use this high-dimensional signal at some cost, model-free methods take the other extreme to focus only on the most relevant low-dimensional signal (the scalar return). Below we propose a method that strikes a balance between these paradigms.

3.3. Hindsight value and model

We first introduce a new value function that can only be computed at training time, the *hindsight* value function v^+. This value still represents the expected return from a state s_t but it is further conditioned on additional observations

$\tau_t^+ \in \mathrm{T}^+$ occuring after time t:

$$v^+(s_t, \tau_t^+) = \mathbb{E}[G|S_0 = s_t, \mathrm{T}^+ = \tau_t^+]. \tag{1}$$

where τ_t^+ can be defined to include any of the future observations, actions and rewards occurring in the trajectory following s_t. The conditioning on a policy π is implicit in this notation for simplicity. The hindsight value can be seen as an instance of a general value function in a stochastic computational graph as defined by Weber et al. (2019).

In this paper, we focus on the formulation where τ^+ is composed of k additional observations $\tau_t^+ = s_{t+1}, s_{t+2}, \dots s_{t+k}$ occurring after time t:

$$v^+(s_t, \tau_t^+) = \mathbb{E}[G|S_0 = s_t, \dots, S_k = s_{t+k}]. \tag{2}$$

Furthermore, we require estimates of v^+ to follow this particular parametric structure:

$$v^+(s_t, \tau_t^+; \theta) = \psi_{\theta_1}(f(s_t), \boldsymbol{\phi}_{\theta_2}(\tau_t^+)), \tag{3}$$

where $\theta = (\theta_1, \theta_2)$, which forces information about the future trajectory through some vector-valued function $\boldsymbol{\phi} \in \mathcal{R}^d$. Intuitively, v^+ is estimating the expected return from a past time point using privileged access to future observations. Note that if k is large enough, then v^+ may simply estimate the empirical return from time t given access to the state trajectory. However, if k is small and $\boldsymbol{\phi}$ is low-dimensional, then $\boldsymbol{\phi}$ becomes a bottleneck representation of the future trajectory τ_t^+, that is, by learning in hindsight, we identify features that are maximally useful to predict the return on the trajectory from time t.

The hindsight value function is not a useful quantity by itself, since – because of its use of privileged future observations – we cannot readily use it at test time. Furthermore, it cannot be used as a baseline either, as when computing the policy gradient it will yield a biased gradient estimator. Instead, the idea is to learn a model $\hat{\boldsymbol{\phi}}$ of $\boldsymbol{\phi}$, that can be used at test time. We conjecture that if privileged features $\boldsymbol{\phi}$ are useful for estimating the value, then the model of those features will also be useful for estimating the value function. We propose to learn the approximate expectation model $\hat{\boldsymbol{\phi}}_{\eta_2}(s)$ conditioned on the current state s and parametrized by η_2, by minimizing an appropriate loss $\mathcal{L}_{\text{model}}(\eta_2)$ between $\boldsymbol{\phi}$ and $\hat{\boldsymbol{\phi}}$ (e.g., a squared loss). The approximate model $\hat{\boldsymbol{\phi}}$ can then be leveraged to obtain a better model-based value estimate $v^m(s;\eta) = \psi_{\eta_1}(f(s), \hat{\boldsymbol{\phi}}_{\eta_2}(s))$. Although $\hat{\boldsymbol{\phi}}(s)$ cannot contain more information than included already in the state s, it can still benefit from having being trained using a richer signal before the value converges.

This formulation can be straightforwardly extended to a case where observations are provided as inputs instead of full states: a learned function then outputs a state representation h_t based on the current observation o_t and previous estimated state h_{t-1} (details in Section 4). Figure 2 summarizes the relation between the different quantities in this *hindsight modelling* (HiMo) approach.

3.4. Illustrative example

We consider the following example to illustrate how the approaches to estimating the value function can differ and how value-driven hindsight modelling might be beneficial. There are no actions in this example[2] and each episode consists of a single transition from initial state s to terminal state s', with a reward $r(s, s')$ on the way. The key aspect of this domain is that observing s' reveals structure that helps predicting the value function in the start state s. Namely, part of state s' provides a simplified view of some of the information present in s which is directly involved in predicting r (in the baseball narrative, this could correspond to observing the final configuration between the ball and bat as they collide). This is made visually obvious in the trajectories sampled in this domain shown in Figure 1, and the details of the dynamics are described in Appendix A.2

Let us consider how the different approaches to learning values presented above fare in this problem. For direct learning, the value from $v(s')$ is 0 since s' is terminal, so any n-step return is identical to the Monte-Carlo return, that is, the information present in observation s' is not leveraged. Results from learning v from s given the return is presented in Figure 1 (model-free, orange curve). A model-based approach first predicts s' from s, then attempts to predict the value given s and the estimated next state. When increasing the input dimension, given a fixed capacity, the model does not focus its attention on the reward-relevant structure in s' and makes errors where it matters most. As a result, it can struggle to learn v faster than a model-free estimate (cf. pink curve in Figure 1). When learning in hindsight, v^+ can directly exploit the revealed structure in the observation of τ^+ (this corresponds to terminal state s' in this case), and as a result the hindsight value learns faster than the regular causal model-free estimate (cf. dashed blue curve). This drives the learning of $\boldsymbol{\phi}$ and its model $\hat{\boldsymbol{\phi}}$, which directly gets trained to predict these useful features for the value. As a result, v^m also benefits and learns faster than the regular v estimate on this problem (cf. blue curve).

3.5. When is it advantageous to model in hindsight?

To understand the circumstances in which hindsight modelling provides a better value estimate, we first consider an analysis that relies on the following assumptions. Suppose that v_θ^m is sharing the same function ψ as v^+ (i.e., $\theta_1 = \eta_1$), and let ψ be linear. If we write $\psi_{\theta_1}(f, \boldsymbol{\phi}) = \binom{\omega_1}{\omega_2}^\top \binom{f}{\boldsymbol{\phi}} + b$,

[2]This can be understood as a Markov Reward Process or a policy evaluation setting.

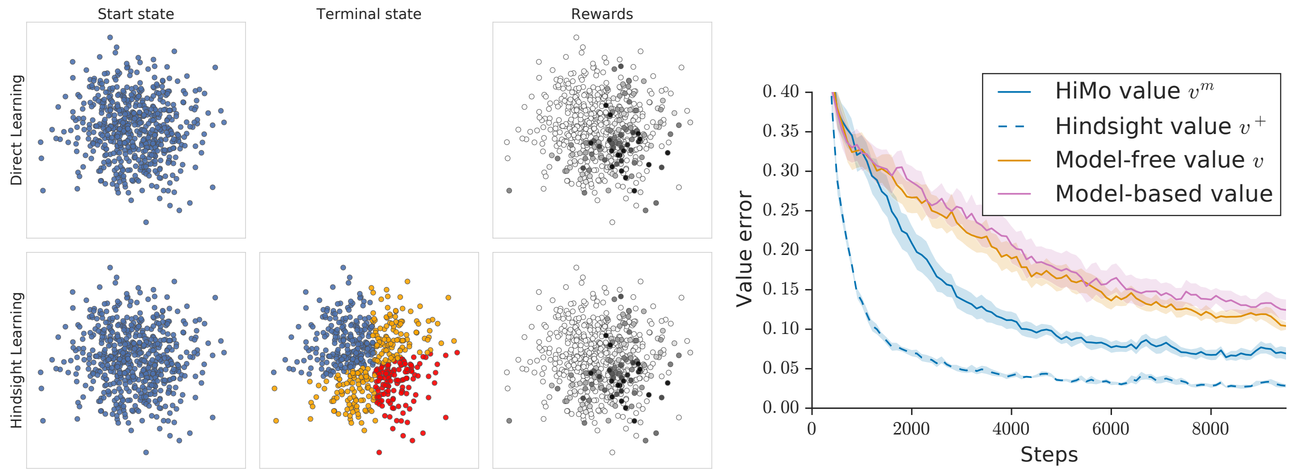


Figure 1. **Illustrative example of Section 3.4 (Left)** Visualization of trajectories. Model-free value prediction see the start state s on the left (which are normally distributed) and must predict the corresponding color-coded reward on the right (which is a complex function of s). Hindsight value prediction can leverage the observed structure in the terminal state s' to obtain a better value prediction. In more detail, this plot shows two dimensions of the initial state s on the left. In the middle, superimposed is the observed reward-relevant quantity from s' that has been color-coded. On the right is the reward for each trajectory (darker for larger rewards). The dimension of states is $D = 4$ in this example. **(Right)** Learning the value of the initial state in the example of Section 3.4. The dimension of the data is $D = 32$ for this experiment, with the dimension of the useful data in the next state is 4. The results are averaged over 4 different instances, each repeated twice. Note that v^+ (dotted line) is using privileged information (the next state).

where $\theta_1 = (\omega_1, \omega_2)$, then we have for fixed values of the parameters:

$$\begin{aligned} &\mathbb{E}[(v^m(s;\eta) - v^+(s,\tau^+;\theta)^2] && (4)\\ &\quad = \mathbb{E}[\|\omega_2^\top(\phi(\tau^+;\theta_2) - \hat{\phi}(s;\eta_2))\|^2] && (5)\\ &\quad \le \mathbb{E}[\|\omega_2\|^2\|\phi(\tau^+) - \hat{\phi}(s)\|^2] && (6)\\ &\quad = \|\omega_2\|^2\mathcal{L}_{\text{model}}(\eta_2), && (7)\end{aligned}$$

using the Cauchy-Schwarz inequality and assuming a squared loss for the model. Let $\mathcal{L}$ define the value error for a particular value function v: $\mathcal{L}(v) = \mathbb{E}[(v(s) - G)^2]$ and $\mathcal{L}(v^+) = \mathbb{E}[(v^+(s,\tau^+) - G)^2]$. Then we have:

$$\mathcal{L}(v^m) = \mathbb{E}[(v^m(s) - v^+(s,\tau^+) + v^+(s,\tau^+) - G)^2] \quad (8)$$

$$\le 2(\|\omega_2\|^2\mathcal{L}_{\text{model}}(\eta_2) + \mathcal{L}(v^+)), \quad (9)$$

using the fact that $\mathbb{E}[(X+Y)^2] \le 2(E[X^2] + E[Y^2])$ for random variables X and Y. If we assume $\mathcal{L}(v^+) = C\mathcal{L}(v)$ with $C < 0.5$ (i.e., estimating the value in hindsight with more information is an easier learning problem), then the following holds:

$$\mathcal{L}_{\text{model}}(\eta_2) < \frac{(1-2C)\mathcal{L}(v)}{2\|\omega_2\|^2} \implies \mathcal{L}(v^m) < \mathcal{L}(v). \quad (10)$$

In other words, this relates how small the modelling error needs to be to guarantee that the value error for v^m is smaller than the value error for the direct estimate v. The modeling error can be large for different reasons. If the environment or the policy is stochastic, then there is some irreducible modelling error for the deterministic model. Even in these cases, a small C can make hindsight modelling advantageous. The modeling error could also be high because predicting ϕ is hard. For example, it could be that ϕ essentially encodes the empirical return, which means predicting ϕ is at least as hard as predicting the value function ($\mathcal{L}_{\text{model}}(\eta_2) \ge \mathcal{L}(v)$). Or it could be that ϕ is high-dimensional, this could cause both a hard prediction problem but also would cause the acceptable threshold for $\mathcal{L}_{\text{model}}$ to decrease (since $\|\theta_2\|^2$ will grow). We address some of these concerns with specific architectural choices like v^+ having a limited view on future observations and having low dimensional ϕ (see architecture section). Note that the analysis above ignores any advantage that could be obtained from representation learning when training $\hat{\phi}$ (if the state encoding function f shares parameters with $\hat{\phi}$).

3.6. Relation to bootstrapping

We want to emphasize the difference between hindsight modelling and bootstrapping as done in temporal difference learning: both exploit a partial trajectory for value prediction but crucially have different objectives and mechanisms.

Bootstrapping helps to provide potentially better value targets from a trajectory (e.g., to reduce variance or deal with off-policyness), but it does not give a richer training signal in itself as it does not communicate more information

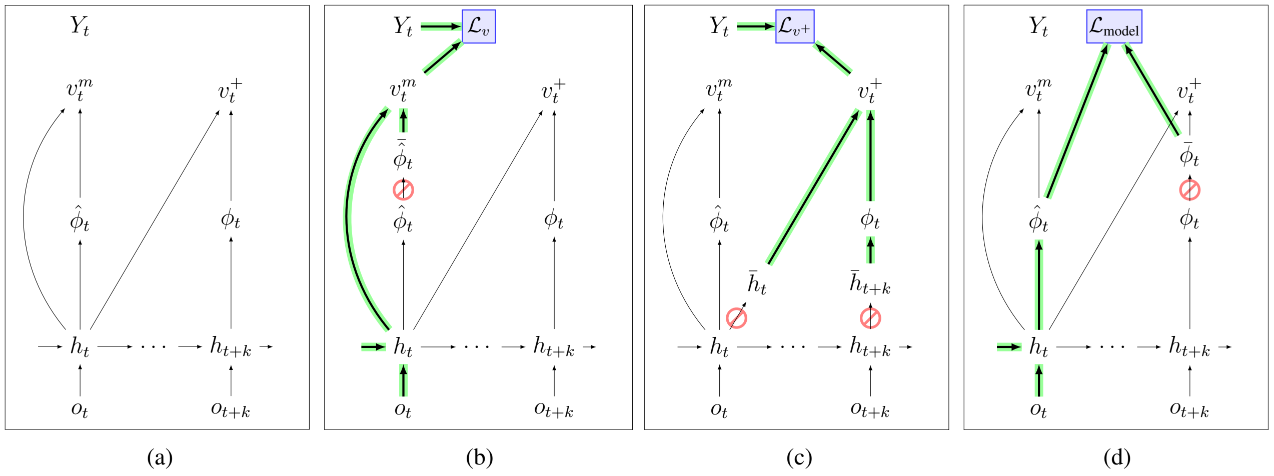


Figure 2. **Network configuration for HiMo.** Figure (a) depicts the overall HiMo architecture, nodes are tensors and edges are (learnable) tensor transformations, e.g. neural networks. In (b), (c) and (d) green identifies learned edges, and the red symbol $\emptyset$ denotes that gradients are stopped in backpropagation. Figure (b) shows the subset of the network used for the value function loss $\mathcal{L}_v$, (c) the hindsight value function loss $\mathcal{L}_{v^+}$ and (d) the model loss $\mathcal{L}_{\text{model}}$.

about the future than a return statistic. Consider for example a policy evaluation problem in a deterministic scenario where we want to estimate the initial value $v(s_0)$, and $v_\theta(s_t) = v(s_t)$ for $t > 0$, i.e., all subsequent value estimates are perfect. In this scenario the Monte-Carlo returns g and all the n-step returns from the start state are equal: $G = R_0 + \gamma v_\theta(s_1) = R_0 + \gamma R_1 + \gamma^2 v_\theta(s_2) = \dots$. It follows that in this particular scenario whether ones chooses to bootstrap or not has no consequence — the value target will be the same. Yet, there might still be some useful information in the trajectory, e.g. in s_2, which would accelerate the learning of $v_\theta(s_0)$, if predicted from s_0. Hindsight modeling has precisely the potential of leveraging this information, because such features of s_2 can be learned in by optimizing the hindsight value function.

4. Architecture

The architecture for HiMo we found to work at scale and tested in the experimental section of the paper is described here. To deal with partial observability, we employ a recurrent neural network, the state-RNN, which replaces the state s_t with a learned internal state h_t, a function of the current observation o_t and past observations through h_{t-1}: $h_t = f(o_t, h_{t-1}; \eta_3)$, where we have extended the parameter description of v^m as $\eta = (\eta_2, \eta_1, \eta_3)$. The model-based value function v^m and the hindsight value function v^+ share the same internal state representation h, but the learning of v^+ assumes h is fixed (we do not backpropagate through the state-RNN in hindsight). In addition, we force $\hat{\phi}$ to only be learned through $\mathcal{L}_{\text{model}}$, so that v^m uses it as an additional input. Denoting with the bar notation quantities treated as non-differentiable (i.e. where the gradient is stopped) this can be summarized as:

$$v^+(h_t, h_{t+k}; \theta) = \psi_{\theta_1}(\overline{h_t}, \boldsymbol{\phi}_{\theta_2}(\overline{h_{t+k}})), \tag{11}$$

$$v^m(h_t; \eta) = \psi_{\eta_1}(h_t, \overline{\hat{\boldsymbol{\phi}}_{\eta_2}(h_t)}), \tag{12}$$

The different losses in the HiMo architecture are combined in the following way:

$$\mathcal{L}(\theta, \eta) = \mathcal{L}_v(\eta) + \alpha \mathcal{L}_{v^+}(\theta) + \beta \mathcal{L}_{\text{model}}(\eta). \tag{13}$$

A diagram of the architecture is presented in Figure 2, and further implementation details can be found in the appendix. Computing v^+ and training $\hat{\phi}$ is done in an online fashion by simply delaying the updates by k steps (just like the computation of an n-step return).

This architecture can be straightforwardly generalized to cases where we also output a policy π_η for an actor-critic setup, providing h and $\hat{\boldsymbol{\phi}}$ as inputs to a policy network.[3] For a Q-value based algorithm like Q-learning, we predict a vector of values q^m and q^+ instead of v^m and v^+. We employ both an actor-critic and a value-based approach in the experimental section, which also provide details on the value target Y_t and update rules in each scenario.

5. Experiments

The illustrative example in Section 3.4 demonstrated the positive effect of hindsight modelling in a simple policy evaluation setting. In this section, we now explore these benefits in the context of policy optimization in challenging

[3]In this case, the total loss $\mathcal{L}(\theta, \eta)$ also contains an actor loss to update π_η and a negative entropy loss.

domains, a custom navigation task called Portal Choice, and Atari 2600. To demonstrate the generality and scalability of our approach we test hindsight value functions in the context of two high-performance RL algorithms, IMPALA (Espeholt et al., 2018) and R2D2 (Kapturowski et al., 2019).

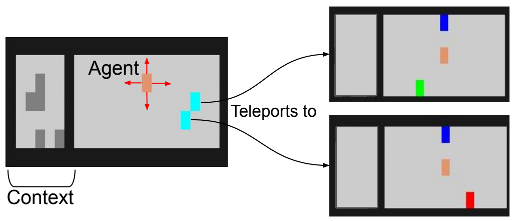


Figure 4. **Portal Choice task.** Left: an observation in the starting room of the Portal Choice task. Two portals (cyan squares) are available to the agent (orange), each of them leading to a different room deterministically based on their position. Right: The two possible goal rooms are identified by a green and red pixel. The reward upon reaching the goal (blue square) is a function of the room and the initial context.

5.1. Portal Choice task

The Portal Choice (Fig. 4) is a two-phase navigation task where, in phase one an agent is presented with a contextual choice between two portals, whose positions vary between episodes. The position of the portal determines its destination in phase two, one of two different goal rooms (green and red rooms). Critically, the reward when terminating the episode in the goal room depends on both the color of the goal room in phase two and a visually indicated combinatorial context shown in the first phase. If the context matches the goal room color, then a reward of 2 is given, otherwise the reward is 0 when terminating the episode (see appendix for the detailed description).

An easy suboptimal solution is to select the portal at random and finish the episode in the resulting goal room by reaching the goal pixel, which will result in a positive reward of 1 on average. A more difficult strategy is to be selective about which portal to take depending on the context, in order to get the reward of 2 on every episode. A model-free agent has to learn the joint mapping from contexts and portal positions to rewards. Even if the task is not visually complex, the context is combinatorial in nature (the agent needs to count randomly placed pixels) and the joint configuration space of context and portal is fairly large (around 250M). Since the mapping from portal position to rooms does not depend on context, learning the portal-room mapping independently is more efficient in this scenario.

For this domain, we implemented the HiMo architecture within a distributed actor-critic agent, named IMPALA proposed by Espeholt et al. (2018). In this case, the target Y_t to train v^m (used as a critic in this context) and v^+ is the V-trace target (Espeholt et al., 2018) to account for off-policy corrections between the behavior policy and the learner policy. The actor shares the same network as the critic and receives h and $\hat{\boldsymbol{\phi}}$ as inputs to output the policy π.

We found that HiMo+IMPALA learned reliably faster to reach the optimal behavior, compared to the vanilla IMPALA baseline that shared the same network capacity (see Figure 3a). The hindsight value v^+ rapidly learns to predict whether the portal-context association is rewarding based on seeing the goal room color in the future. Then $\boldsymbol{\phi}$ learns to predict the new information from the future that helps that prediction: the identity of the room (see analysis Fig 3c). The prediction of $\boldsymbol{\phi}$ becomes effectively a model of the mapping from portal to room identity (since the context does not correlate with the room identity). Having access to such mapping through $\hat{\boldsymbol{\phi}}$ helped the value prediction (Fig 3b), which led to better action selection. Note that if the two rooms were visually indistinguishable, for example with

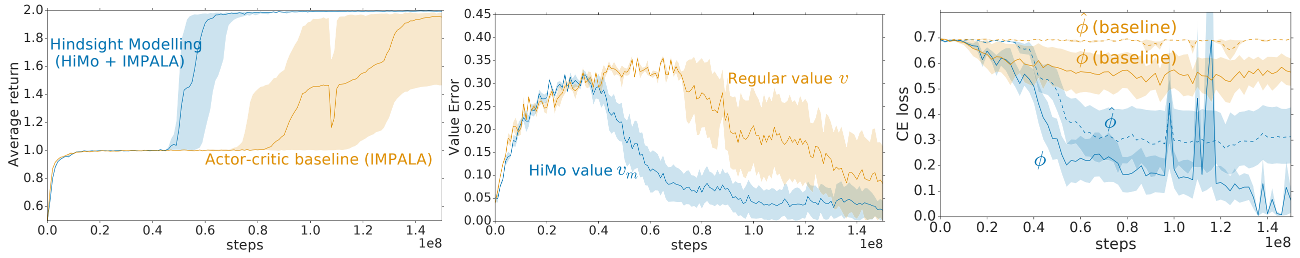


Figure 3. **Results in the Portal Choice task.** (a) shows the median performance as a function of environment steps out of 4 seeds. (b) shows the value error averaged across states on the same x-axis scale for different value function estimate. (c) is an analysis that shows the cross-entropy loss of a classifier that takes as input $\boldsymbol{\phi}$ (solid line) or $\hat{\boldsymbol{\phi}}$ (dotted line) and predicts the identity of the goal room (red or green) as a binary classification task. The HiMo curves (blue) show that information about the room identity becomes present first in $\boldsymbol{\phi}$ and then gets captured in its model $\hat{\boldsymbol{\phi}}$. For the baseline (where we set $\alpha = \beta = 0$), $\hat{\phi}$ is not trained based on ϕ and only achieves to classify the room identity at chance level.

no red/green rooms separation, HiMo would not be able to offer any advantage over its model-free counterpart.

5.2. Atari

We tested our approach in Atari 2600 videogames using the Arcade Learning Environment (Bellemare et al., 2013). We added HiMo on top of Recurrent Replay Distributed DQN (R2D2), a DQN-based distributed architecture introduced by Kapturowski et al. (2019) which previously achieved state-of-the-art scores in Atari games.

In this value-based setting, HiMo trains $q^m(\cdot,\cdot;\eta)$ and $q^+(\cdot,\cdot;\theta)$ based on n-step return targets:

$$Y_t = g\left(\sum_{m=0}^{n-1} \gamma^m R_{t+m} + \gamma^n g^{-1}\left(q^m(S_{t+n}, A^*; \eta^-)\right)\right), \tag{14}$$

where g is an invertible function, η^- are the periodically updated target network parameters (as in DQN by Mnih et al. (2015)), and $A^* = \arg\max_a q^m(S_{t+n}, a; \eta)$ (the Double DQN update proposed by Van Hasselt et al. (2016)). The details of the architecture and hyperparameters are described in the appendix.

We ran our approach on 57 Atari games for 200k gradient steps (around 1 day of training), with 3 seeds for each game. The evaluation averages the score between 200 episodes across seeds, each lasting a maximum of 30 minutes each and starting a random number (up to 30) of no-op actions. In order to compare scores between different games and aggregated results, we computed normalized scores for each game based on random and human performance so that 0% corresponds to random performance and 100% corresponds to human. We observed an increase of 132.5% in the median human normalized score compared to the R2D2 baseline with the same network capacity, aggregate results are reported in Table 1. Figure 5 details the difference in normalized score between HiMo and our R2D2 baseline for all games individually. We note that the original R2D2 results reported by Kapturowski et al. (2019), which used a similar hardware configuration but a different network architecture, were around 750% median human normalized score after a day of training.

In our experimental evaluation we observed that HiMo typically either offers improved data efficiency or has no overwhelming adverse effects in training performance. In Figure 5 we show training curves for a selection of representative Atari environments where at evaluation time HiMo both under-performed (left) and out-performed R2D2 (right); these seem to indicate that in the worst case scenario HiMo's training performance reduces to R2D2's.

	R2D2	R2D2 + HiMo
Median	832.5%	**965%**
Mean	2818.5%	**2980%**

Table 1. Median and mean human normalized scores across 57 Atari2600 games for HiMo versus the R2D2 baseline after a day of training.

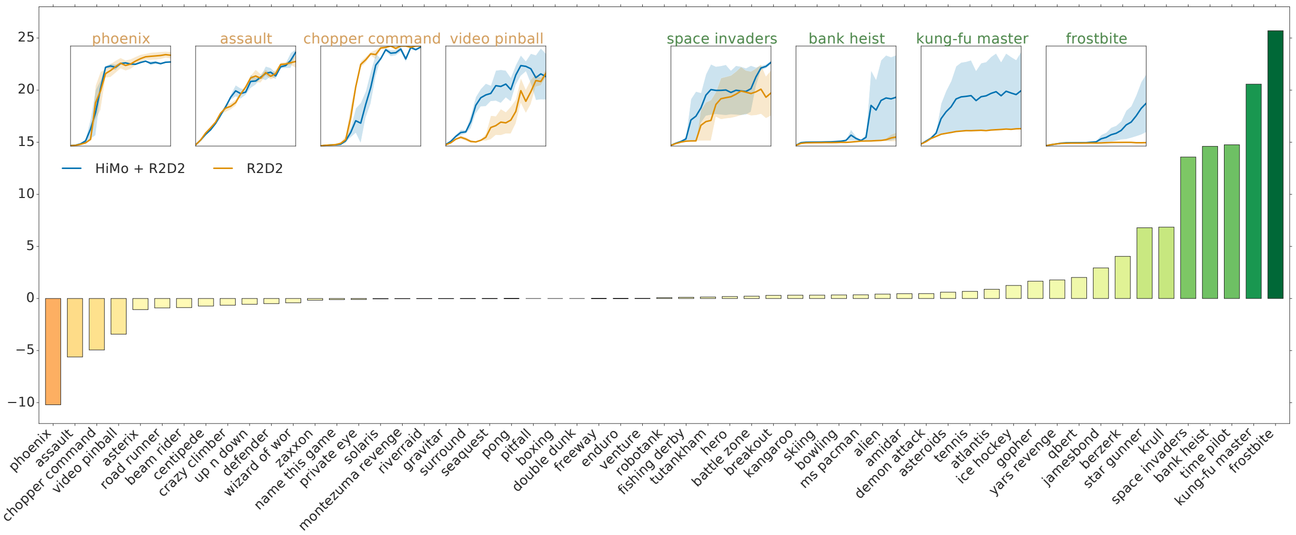
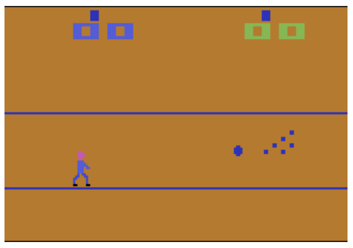
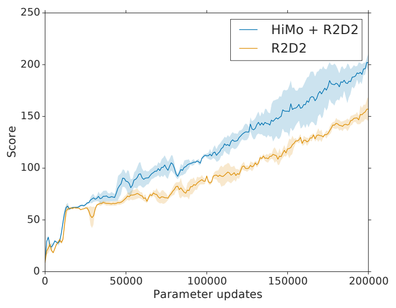
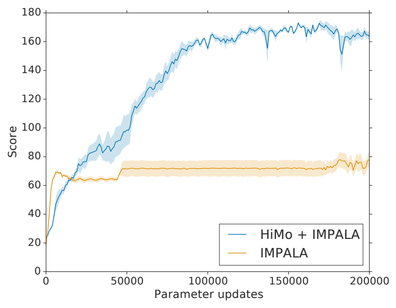


Figure 5. **Comparison of human normalized score in Atari.** Difference in human normalized score per game in Atari, HiMo versus the improved R2D2 after 200k learning steps, alongside learning curves for a selection of HiMo worst and top performing games. Note that the high variance of the curves in Atari between seeds can usually be explained by the variable timestep at which different seeds jump from one performance plateau to the next.


(a)



(b)



(c)

Figure 6. **Performance in the Atari bowling game.** (a) In the bowling game in Atari a delayed reward can be predicted by the intermediate event of the ball hitting the pins. (b-c) Learning curves for HiMo in the bowling game using two different RL methods: a value-based method (R2D2) in (b) and a policy-gradient method (IMPALA) in (c).

Bowling is one of the Atari games where rewards are delayed with relevant information being communicated through intermediate observations (the ball hitting the pins), just like the baseball example we have used in the introduction. We found HiMo to perform better than the R2D2 baseline in this particular game. We also ran HiMo with the actor-critic setup (IMPALA) described in the previous section, finding similar performance gain with respect to the model-free baseline. Learning curves for these experiments are presented in Figure 6.

6. Related Work

Recent work have used auxiliary predictions successfully in RL as a mean to obtain a richer signal for representation learning (Jaderberg et al., 2016; Sutton et al., 2011). However these additional prediction tasks are hard-coded and so they cannot adapt to the task demand when needed. We see them as a complementary approach to more efficient learning in RL. An exception is the recent work by Veeriah et al. (2019), done concurrently to our work, which studies an alternative formulation that employs meta-gradients to discover modelling targets (referred to as the questions).

Buesing et al. (2018) have considered using observations in an episode trajectory in hindsight to infer variables in a structural causal model of the dynamics, allowing to reason more efficiently in a model-based way about counterfactual actions. However this approach requires learning an accurate generative model of the environment.

Recent work in model-based RL has considered implicit models that directly predict the relevant quantities for planning (Silver et al., 2017; Oh et al., 2017; Schrittwieser et al., 2019), but these do not exploit the future observations. We see these approaches as being completementary to our work. In another recent model-based work by Farahmand (2018), the model loss minimizes some form of value consistency: the difference between the value at some next state and the expected value from starting in the previous state, applying the model and computing the value. While this makes the model sensitive to the value, it only exploits the future real state through V as a learning signal (just like in bootstrapping). A similar approach around reward consistency is proposed by Gelada et al. (2019).

In supervised learning, the learning using privileged information (LUPI) framework introduced by Vapnik & Izmailov (2015) considers ways of leveraging privileged information at train time. Although the techniques developed in that work do not apply directly in the RL setting, some of our approach can be understood in their framework as considering the future trajectory as the privileged information for a value prediction problem.

Finally, privileged information coming from full state observation has been leveraged in RL to learn better critic in asymmetric actor-critic architectures (Pinto et al., 2017; Zhu et al., 2018). However this does not use future information and only applies to settings where special side-information (full state) is available at train time.

7. Conclusion

High-dimensional observations in the intermediate future often contain task-relevant features that can facilitate the prediction of an RL agent's final return. We introduced a reinforcement learning algorithm, HiMo, that leverages this insight by the following two-stage approach. First, by reasoning in hindsight, the algorithm learns to extract relevant features of future observations that would be been most helpful for estimating the final value. Then, a forward model is learned to predict these features, which in turn is used as input to an improved value function, yielding better policy evaluation and training at test time. We demonstrated that this approach can help tame complexity in environments with rich dynamics at scale, yielding increased data efficiency and improving the performance of state-of-the-art model-free architectures.

References

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

Buesing, L., Weber, T., Zwols, Y., Racaniere, S., Guez, A., Lespiau, J.-B., and Heess, N. Woulda, coulda, shoulda: Counterfactually-guided policy search. *arXiv preprint arXiv:1811.06272*, 2018.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

Farahmand, A.-m. Iterative value-aware model learning. In *Advances in Neural Information Processing Systems*, pp. 9072–9083, 2018.

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

Guez, A., Mirza, M., Gregor, K., Kabra, R., Racaniere, S., Weber, T., Raposo, D., Santoro, A., Orseau, L., Eccles, T., et al. An investigation of model-free planning. In *International Conference on Machine Learning*, pp. 2464–2473, 2019.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Kapturowski, S., Ostrovski, G., Dabney, W., Quan, J., and Munos, R. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations*, 2019.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529, 2015.

Oh, J., Singh, S., and Lee, H. Value prediction network. In *Advances in Neural Information Processing Systems 30*, pp. 6118–6128. 2017.

Pinto, L., Andrychowicz, M., Welinder, P., Zaremba, W., and Abbeel, P. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*, 2017.

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

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

Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., et al. The predictron: End-to-end learning and planning. In *International Conference on Machine Learning*, pp. 3191–3199, 2017.

Sutton, R. S. and Barto, A. G. Reinforcement learning: An introduction. 2011.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 761–768, 2011.

Talvitie, E. Model regularization for stable sample rollouts. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, pp. 780–789. AUAI Press, 2014.

Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

Vapnik, V. and Izmailov, R. Learning using privileged information: similarity control and knowledge transfer. *Journal of machine learning research*, 16(2023-2049):2, 2015.

Veeriah, V., Hessel, M., Xu, Z., Rajendran, J., Lewis, R. L., Oh, J., van Hasselt, H. P., Silver, D., and Singh, S. Discovery of useful questions as auxiliary tasks. In *Advances in Neural Information Processing Systems*, pp. 9306–9317, 2019.

Weber, T., Heess, N., Buesing, L., and Silver, D. Credit assignment techniques in stochastic computation graphs. *arXiv preprint arXiv:1901.01761*, 2019.

Zhu, Y., Wang, Z., Merel, J., Rusu, A. A., Erez, T., Cabi, S., Tunyasuvunakool, S., Kramár, J., Hadsell, R., de Freitas, N., and Heess, N. Reinforcement and imitation learning for diverse visuomotor skills. *CoRR*, abs/1802.09564, 2018.

A. Appendix

A.1. Introduction Example

We describe a counting argument similar to the discrete probability factorization example in the introduction but that applies more directly to a value function scenario. Consider the following chain:

$$(X, X') \to (X, Y') \to Z, \tag{15}$$

where Z is to be interpreted as the expected return. Here the start state (X, X') is sampled randomly but the rest of the chain has deterministic transitions, and Y' is independent of X given X'. Let n be the number of possible values of X, and m the number of possible values of X', and suppose the number of possible values of Y' is 2. In a tabular setting, learning the start state's value function model-free (i.e. mapping (X, X') directly to Z) requires observing returns Z for all nm entries. In contrast, if we estimate the mappings $X' \to Y'$ and $(X, Y') \to Z$ separately, it requires $m+2n$ entries, which is better than nm (for $n, m > 4$). This shows that even in a policy evaluation scenario, the right model can more efficiently learn the value function. The illustrative task in Section 3.4 extends this to a function approximation setting but is similar in spirit.

A.2. Illustrative Example Details

We provide the precise definition of the illustrative tasks of Section 3.4. It consists of a value estimation problem in a 1-step Markov Reward Process (no actions), namely each episode consists of a single transition from initial state s to terminal state s', with a reward $r(s, s')$ on the way. The agent is trained on multiple episodes of each MRP instance (the x-axis in Figure 1-right). This learning process is repeated independently and averaged for multiple instances of the environment. Each instance of the task is parametrized by a square matrix W and a vector b sampled from a unit normal distribution, as well as randomly initialized MLP (see below), which together determine the uncontrolled MDP. Initial states s are of dimension D and sampled from a multivariate unit normal distribution ($s_i \sim N(0, 1)$ for all state dimension i). Given $s = \binom{s_1}{s_2}$, where s_1 and s_2 are of dimension D_1 and D_2 ($D = D_1 + D_2$), the next state $s' = \begin{pmatrix} s'_1 \\ s'_2 \end{pmatrix}$ is determined according to the transition function: $s'_1 = \text{MLP}(s) + \epsilon$ and $s'_2 = \sigma(W s_2 + b)$ where σ is the Heaviside function, and MLP is a randomly sampled Multi-Layer Perceptron. s'_1 acts as a distractor here, with additive noise $\epsilon \sim N(0, 1)$. The reward obtained is $r(s, s') = \sum_i s_1^{(i)} \sum_i s_2'^{(i)} / \sqrt{D}$. The true value in the start state is also $v(s) = r(s, s')$.[4]

Figure 1-left shows some trajectories for a low-dimensional version of the problem ($D = 4$) and the middle plot displays s'_2 by color-coding it according to $\sum_i s_2'^{(i)}$. The results in Figure 1-right use networks where each subnetwork is a small 1-hidden-layer MLP with 16 hidden units and ReLu activation functions, and ϕ has dimension 3. The dimension of the data is $D = 32$ for this experiment, with the dimension of the useful data in the next state $D_2 = 4$.

[4]Note that r only uses the part of s' which is obtained deterministically from s.

A.3. General Architecture Details

To compute v^+ and train $\hat{\phi}$ in an online fashion, we process fixed-length unrolls of the state-RNN and compute the hindsight value and corresponding updates at time t if $t + k$ is also within that same unroll. Also, we update v^+ at a slower rate (i.e., $\alpha < \beta$) to give enough time for the model $\hat{\boldsymbol{\phi}}$ to adapt to the changing hindsight features $\boldsymbol{\phi}$. In our experiments we found that even a low-dimensional $\boldsymbol{\phi}$ (in the order of $d = 3$) and a relatively short hindsight horizon k (in the order of 5) are sufficient to yield significant performance boosts, whilst keeping the extra model computational costs modest.

For most experiments described in the paper, the model loss $\mathcal{L}_{\text{model}}(\eta_2)$ is the following:

$$\mathcal{L}_{\text{model}}(\eta_2) = \mathbb{E}_{s,\tau^+}[\|\boldsymbol{\phi}_{\theta_2}(\tau^+) - \hat{\boldsymbol{\phi}}_{\eta_2}(s)\|_2^2] \tag{16}$$

where the expectation is taken over the distribution of states and partial trajectories τ^+ resulting from that state.

A.4. Portal Choice

Environment The observation is a 7×23 RGB frame (see Figure 4). There are 3 possible spawning points for the agent in the center and 42 possible portal positions (half of which lead to the green room, the other half leading to the red room). At the start of an episode, two portals, each leading to a different room, are chosen are random. They are both displayed as cyan pixels. Included in the observation in the first phase is the context, a random permutation in a 5×5 grid of N pixels, where is uniformly sampled at the start of each episode: $N \sim \mathcal{U}\{1, 10\}$. A fixed map $f : \{1, \ldots, 10\} \to \{0, 1\}$ determines which contexts are rewarding with the green room, the rest being rewarding with the red room. The reward when reaching the goal is determined according to:

$$R = 2(f(N)G + (1 - f(N))(1 - G)), \tag{17}$$

where $G \in \{0, 1\}$ is whether the reached room is green.

Note that the important decision in this task does not require any memory: everything is observed in the portal room to select the portal, yet there is a memory demand when in the reward room. We ran an extra control experiment where we gave h_{t-k} as an additional input to the policy and value for the baseline IMPALA agent and it did not perform better than what is reported in Figure 3a for the actor-critic baseline.

Network architecture The policy and value network takes in the observation and passes it to a ConvNet encoder (with filter channels [32, 32, 32], kernel shapes [4, 3, 3] applied with strides [2, 1, 1]) before being passed to a ConvLSTM network with 32 channels and 3x3 filters. The output of the ConvLSTM is the internal state h. The $\hat{\boldsymbol{\phi}}$ network is a ConvNet with [32, 32, 32, 1] filter channels with kernels of size 3 except for a final 1x1 filter, whose output is flatten and passed to an MLP with 256 hidden units with ReLu activation, before a linear layer with dimension $d = 3$. The $\boldsymbol{\phi}$ network is a similarly configured network with one less convolution layer and 128 hidden units in the MLP. The ψ_η network is an MLP with 256 hidden units followed by a linear layer that takes h and $\hat{\boldsymbol{\phi}}$ as input and outputs the policy π^m and the value v^m. v^+ is obtained similarly with a similar MLP that has a single scalar output. We used a future observation window of $k = 5$ steps in this domain and loss weights $\alpha = 0.25$, $\beta = 0.5$. Unroll length was 20, and $\gamma = 0.99$. Optimization was done with the Adam optimizer (learning rate of $5e^{-4}$), with batch size 32. The model-free baseline is obtained by using the same code and network architecture, and setting the modeling loss and hindsight value loss to 0 ($\alpha = \beta = 0$).

For the portal task, we found it better to employ a cross-entropy loss for the model loss:

$$\mathcal{L}_{\text{model}}(\eta_2) = \mathbb{E}_{s,\tau^+}[H(p(\boldsymbol{\phi}_{\theta_2}(\tau^+)), \hat{p}(\hat{\boldsymbol{\phi}}_{\eta_2}(s))] \tag{18}$$

where $p^{(i)} \propto e^{\boldsymbol{\phi}_{\theta_2}(\tau^+)^{(i)}}$ and $\hat{p}^{(i)} \propto e^{\hat{\boldsymbol{\phi}}_{\eta_2}(s)^{(i)}}$ are the softmax distributions when interpreting $\boldsymbol{\phi}$ and $\hat{\boldsymbol{\phi}}$ as the vector of logits.

A.5. Atari

Hyper-parameters and infrastructure are the same as reported in (Kapturowski et al., 2019), with deviations as listed in table 2. For our value target, we also average different n-step returns with exponential averaging as in $Q(\lambda)$ (with the return being truncated at the end of unrolls). The Q network is composed of a convolution network (cf. Vision ConvNet in table) which is followed by an LSTM with 512 hidden units. What we refer to in the main text as the internal state h is the output of the LSTM. The $\boldsymbol{\phi}$ and $\hat{\boldsymbol{\phi}}$ networks are MLPs with a single hidden layer of 256 units and ReLu activation function, followed by a linear which outputs a vector of dimension d. The ψ_{θ_1} function concatenates h and $\boldsymbol{\phi}$ as inputs to an MLP with 256 hidden units with ReLu activation function, followed by a linear which outputs q^+ (a vector of dimension 18, the size of the Atari action set). q^m is obtained by passing h and $\hat{\boldsymbol{\phi}}$ to a dueling network as described by (Kapturowski et al., 2019).

Other HiMo parameters are described in table 3. The R2D2 baseline with the same capacity is obtained by running the same architecture with $\alpha = \beta = 0$.

Table 2. Hyper-parameter values used for our R2D2 implementation.

Number of actors	320
Sequence length	80 (+ prefix of l = 20 in burn-in experiments)
Learning rate	$2e^{-4}$
Adam optimizer β_1	0.9
Adam optimizer β_2	0.999
λ	0.7
Target update interval	400
Value function rescaling	$g(x) = \text{sign}(x)\left(\sqrt{\|x\| + 1} - 1\right) + \epsilon x,\ \epsilon = 10^{-3}$
Frame pre-processing	None (full res. including no frame stacking)
Vision ConvNet filters sizes	[7, 5, 5, 3]
Vision ConvNet filters strides	[4, 2, 2, 1]
Vision ConvNet filters channels	[32, 64, 128, 128]

Table 3. Hindsight modelling parameters for Atari

α	0.01
β	1.0
k	5
d	3