Efficient Model–Based Deep Reinforcement Learning with Variational State Tabulation

Dane Corneil [1] **Wulfram Gerstner** [1] **Johanni Brea** [1]

Abstract

Modern reinforcement learning algorithms reach super–human performance on many board and video games, but they are sample inefficient, i.e. they typically require significantly more playing experience than humans to reach an equal performance level. To improve sample efficiency, an agent may build a model of the environment and use planning methods to update its policy. In this article we introduce Variational State Tabulation (VaST), which maps an environment with a high–dimensional state space (e.g. the space of visual inputs) to an abstract tabular model. Prioritized sweeping with small backups, a highly efficient planning method, can then be used to update state–action values. We show how VaST can rapidly learn to maximize reward in tasks like 3D navigation and efficiently adapt to sudden changes in rewards or transition probabilities.

1. Introduction

Classical Reinforcement Learning (RL) techniques generally assume a tabular representation of the state space (Sutton & Barto, 2018). While methods like prioritized sweeping (Sutton & Barto, 2018; Moore & Atkeson, 1993; Peng & Williams, 1993; Van Seijen & Sutton, 2013) have proven to be very sample–efficient in tabular environments, there is no canonical way to carry them over to very large (or continuous) state spaces, where the agent seldom or never encounters the same state more than once. Recent approaches to reinforcement learning have shown tremendous success by using deep neural networks as function approximators in such environments, allowing for generalization between similar

[1]Laboratory of Computational Neuroscience (LCN), School of Computer and Communication Sciences and Brain Mind Institute, School of Life Sciences, École Polytechnique Fédérale de Lausanne, Switzerland. Correspondence to: Dane Corneil <dane.corneil@epfl.ch>.

states (e.g. Mnih et al. (2015; 2016)) and learning approximate dynamics to perform planning at decision time (e.g. Silver et al. (2017); Oh et al. (2017); Farquhar et al. (2017); Racanière et al. (2017); Nagabandi et al. (2017)). However, methods like prioritized sweeping that use a model for offline updates of Q-values (i.e. background planning (Sutton & Barto, 2018)), have not yet been investigated in conjunction with function approximation by neural networks.

Adjusting the weights in a deep network is a slow procedure relative to learning in tabular environments. In particular, agents using deep architectures typically fail to take advantage of single experiences that significantly alter the policy. This was illustrated by recent work on Model–Free Episodic Control (MFEC) (Blundell et al., 2016), where a very simple agent using a semi–tabular approach significantly outperformed existing deep network approaches in the early stages of learning. The basic MFEC agent uses a random projection from the observation space to a low–dimensional space, and stores the discounted returns associated with observations in a lookup table. The Q-values of states observed for the first time are determined by a k–Nearest–Neighbour average over discounted returns associated with similar existing states in the lookup table.

As an example with an MFEC agent, we can consider the T–maze task shown in Figure 1A. The observations are continuous (x, y) coordinates; the agent can take a fixed–sized step in one of four cardinal directions, with a rebound on hitting a wall, and a terminal reward zone (green). The first episode (red) is spontaneously terminated without reward; the discounted returns along the red trajectory are therefore set to zero. On the second episode (blue), the agent reaches the reward zone, and the discounted reward is immediately associated with the states visited along the blue trajectory.

To improve sample efficiency, we consider how an agent could apply the experience of the rewarded trajectory to Q-value estimates in the top–right arm. In particular, by learning a model of the environment, the agent could learn that both trajectories pass through the center of the T–maze, and that discovering a reward at the bottom of the maze should therefore change Q-value estimates in both of the arms at the top. This idea is exploited by the model–based RL technique of prioritized sweeping (Moore & Atkeson,
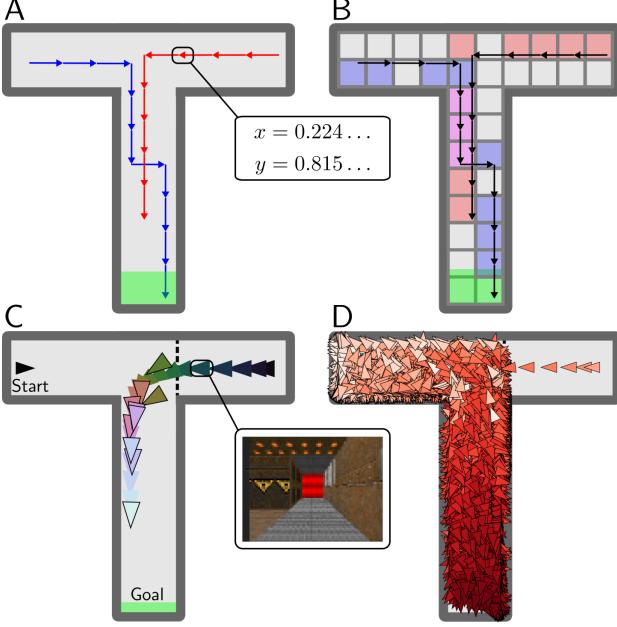
Figure 1. **Using state tabulation for efficient planning**. [A] Two episodes in a time discrete MDP with a continuous state space, given by (x, y) coordinates. [B] The same episodes after discretising the state space by rounding. States visited on each trajectory are shaded; magenta states were shared by both trajectories, and can be leveraged by prioritized sweeping. [C] 3D navigation with VaST. The agent was trained to run from the start position to the goal, with one arm blocked (dotted line). After training, the agent experienced a trajectory from the blocked arm to the stem of the maze (arrows, no outline). If the observations in this trajectory mapped to existing states, the average coordinates and orientation where those states were previously observed are shown (matching colour arrows, outlined). The observations for one state are illustrated. [D] A scatter plot of the values of all states after training according to the average position and orientation where they were observed; darker red corresponds to higher value.

1993; Peng & Williams, 1993; Van Seijen & Sutton, 2013).

However, assuming random restarts, the agent in this task never encounters the same state more than once. In this case, given a deterministic task, prioritized sweeping as implemented by Van Seijen & Sutton (2013) collapses to the MFEC learning algorithm (Brea, 2017). We are therefore motivated to consider mapping the observation space to a tabular representation by some form of discretisation. For example, with discretisation based on rounding the (x, y) coordinates (a simple form of state aggregation (Li et al., 2006; Sutton & Barto, 2018)), the two trajectories in Figure 1B now pass through several of the same states. A model–based prioritized sweeping algorithm would allow us to update the Q-values in the top–right arm of the maze to nonzero values after experiencing both episodes, despite the fact that the red trajectory did not result in reward.

If observations are given by high-dimensional visual inputs instead of (x, y) coordinates, the simple form of state aggregation by rounding (Figure 1B) is impractical. Instead, we propose and describe in this article the new method of Variational State Tabulation (VaST)[1] for learning discrete, tabular representations from high–dimensional and/or continuous observations. VaST can be seen as an action conditional hybrid ANN–HMM (artificial neural network hidden Markov model, see e.g. (Bengio et al., 1992; Tucker et al., 2017; Ng et al., 2016; Maddison et al., 2016)) with a d-dimensional binary representation of the latent variables, useful for generalization in RL. VaST is trained in an unsupervised fashion by maximizing the evidence lower bound. We exploit a parallelizable implementation of prioritized sweeping by small backups (Van Seijen & Sutton, 2013) to constantly update the value landscape in response to new observations. By creating a tabular representation with a dense transition graph (i.e. where the same state is revisited multiple times), the agent can rapidly update state–action values in distant areas of the environment in response to single observations.

In Figure 1C&D, we show how VaST can use the generalization of the tabular representation to learn from single experiences. We consider a 3D version of the example T–maze, implemented in the *VizDoom* environment (Kempka et al., 2016). Starting from the top–left arm, the agent was trained to run to a reward in the bottom of the T–maze stem. During training, the top–right arm of the T–maze was blocked by an invisible wall. After training, the agent observed a single, 20–step fixed trajectory (or "forced run") beginning in the top–right arm and ending in the stem, without reaching the reward zone (Figure 1C). The agent's early observations in the unexplored right arm were mapped to new states, while the observations after entering the stem were mapped to existing states (corresponding to observations at similar positions and orientations). The agent was able to update the values of the new states by prioritized sweeping from the values of familiar states (Figure 1D), without needing to change the neural network parameters, as would be necessary with model-free deep reinforcement learners like DQN (Mnih et al., 2013).

2. Learning the Model

In order to compute a policy using the model–based prioritized sweeping algorithm described by Van Seijen & Sutton (2013), we seek a posterior distribution $q(s_t|o_{t-k:t})$ over latent *discrete* states s_t given a causal filter over recent observations $o_{t-k:t}$. We use a variational approach to approximate this posterior distribution.

---

[1]The full code for VaST can be found at https://github.com/danecor/VaST/.

2.1. The variational cost function

For a sequence of states $s_{0:T} = (s_0, \ldots s_T)$ and observations $o_{0:T} = (o_0, \ldots o_T)$, we consider a family of approximate posterior distributions $q_\phi(s_{0:T}|o_{0:T})$ with parameters ϕ, which we assume to factorise given the current observation and a memory of the past k observations, i.e.

$$q_\phi(s_{0:T}|o_{0:T}) = \prod_{t=0}^{T} q_\phi(s_t|o_{t-k:t}), \qquad (1)$$

where observations before $t = 0$ consist of blank frames. To learn q_ϕ, we also introduce an auxiliary distribution p_θ parameterized by θ. Given a collection of M observation sequences $\mathcal{O} = \{o_{0:T^\mu}^\mu\}_{\mu=1}^M$ and hidden state sequences $\mathcal{S} = \{s_{0:T^\mu}^\mu\}_{\mu=1}^M$, we maximize the log–likelihood $\log \mathcal{L}(\theta; \mathcal{O}) = \sum_{\mu=1}^M \log p_\theta(o_{0:T^\mu}^\mu)$ of the weight parameters θ, while minimizing $\mathcal{D}_{KL}(q_\phi(\mathcal{S}|\mathcal{O})||p_\theta(\mathcal{S}|\mathcal{O}))$. Together, these terms form the evidence lower bound (ELBO) or negative variational free energy

$$-\mathcal{F}(\theta, \phi; \mathcal{O}) = \log \mathcal{L}(\theta; \mathcal{O}) - \mathcal{D}_{KL}(q_\phi(\mathcal{S}|\mathcal{O})||p_\theta(\mathcal{S}|\mathcal{O}))$$
$$= \mathbb{E}_{q_\phi}[\log p_\theta(\mathcal{S}, \mathcal{O})] + \mathcal{H}(q_\phi(\mathcal{S}|\mathcal{O})), \qquad (2)$$

where $\mathcal{H}$ denotes the entropy of the distribution. The term inside the expectation evaluates to

$$\log p_\theta(\mathcal{S}, \mathcal{O}) = \sum_{\mu=1}^M \log \pi_{\theta_0}(s_0^\mu) + \sum_{\mu=1}^M \sum_{t=0}^{T^\mu} \log p_{\theta_\mathcal{R}}(o_t^\mu|s_t^\mu)$$
$$+ \sum_{\mu=1}^M \sum_{t=1}^{T^\mu} \log p_{\theta_\mathcal{T}}(s_t^\mu|a_t^\mu, s_{t-1}^\mu), \qquad (3)$$

where a_t^μ denotes the action taken by the agent on step t of sequence μ, π_{θ_0} is the distribution over initial states, and $\theta_0 \cup \theta_\mathcal{R} \cup \theta_\mathcal{T} = \theta$.

We aim to learn the appropriate posterior distribution q_ϕ by minimizing the variational free energy (maximizing the ELBO). Our cost function from Eq. 2 can be written as

$$\mathcal{F}(\theta, \phi; \mathcal{O}) = \sum_{\mu=1}^M \sum_{t=0}^{T^\mu} [\mathcal{R}_t^\mu + \mathcal{T}_t^\mu - \mathcal{H}_t^\mu], \qquad (4)$$

with reconstruction cost terms

$$\mathcal{R}_t^\mu = -\sum_{s_t^\mu} q_\phi(s_t^\mu|o_{t-k:t}^\mu) \log p_{\theta_\mathcal{R}}(o_t^\mu|s_t^\mu), \qquad (5)$$

transition cost terms

$$\mathcal{T}_t^\mu = -\sum_{s_t^\mu, s_{t-1}^\mu} q_\phi(s_t^\mu, s_{t-1}^\mu|o_{t-k-1:t}^\mu) \log p_{\theta_\mathcal{T}}(s_t^\mu|a_t^\mu, s_{t-1}^\mu)$$
$$(6)$$

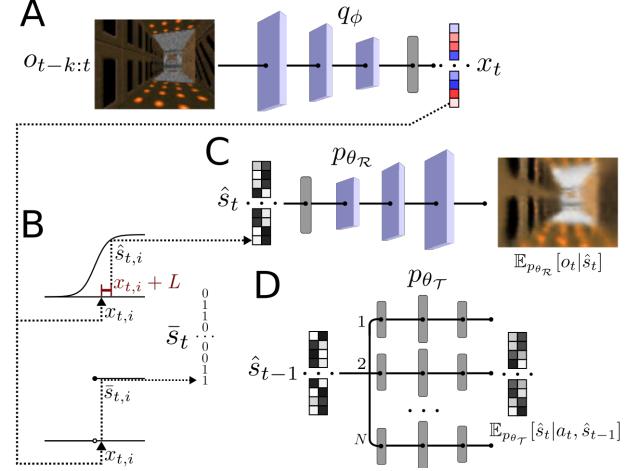*Figure 2.* **The network model**. [A] CNN encoder q_ϕ. [B] Encoder outputs can be used to sample each dimension from a Con–crete distribution for training ($\hat{s}_t$), or discretised to the Bernoulli mode $\bar{s}_t$ to update the table. The Con–crete distribution corresponds to a logistic activation with added noise L. [C] DCNN decoder $p_{\theta_\mathcal{R}}$ and [D] Transition network $p_{\theta_\mathcal{T}}$, with N possible actions. For illustration, $\mathbb{E}_{p_{\theta_\mathcal{R}}}[o_t|\hat{s}_t]$ and $\mathbb{E}_{p_{\theta_\mathcal{T}}}[\hat{s}_t|a_t, \hat{s}_{t-1}]$ are shown.

for $t > 0$ and $\mathcal{T}_0^\mu = -\sum_{s_0^\mu} q_\phi(s_0^\mu|o_0^\mu) \log \pi_{\theta_0}(s_0^\mu)$, and entropy terms

$$\mathcal{H}_t^\mu = -\sum_{s_t^\mu} q_\phi(s_t^\mu|o_{t-k:t}^\mu) \log q_\phi(s_t^\mu|o_{t-k:t}^\mu). \qquad (7)$$

We parameterize the posterior distribution q_ϕ (or "encoder") using a deep Convolutional Neural Network (CNN) (Krizhevsky et al., 2012), and the observation model $p_{\theta_\mathcal{R}}$ using a deep Deconvolutional Neural Network (DCNN) (Goodfellow et al., 2014), as shown in Figure 2. We use a multilayer perceptron (3 layers for each possible action) for the transition model $p_{\theta_\mathcal{T}}$, and learned parameters θ_0 for the initial state distribution π_{θ_0}. The architecture is similar to that of a Variational Autoencoder (VAE) (Kingma & Welling, 2013; Rezende et al., 2014), with the fixed priors replaced by learned transition probabilities conditioned on previous state–action pairs.

To allow for a similarity metric between discrete states, we model the state space as all possible combinations of d binary variables, resulting in $N = 2^d$ possible states. Each of the d outputs of the encoder defines the expectation of a Bernoulli random variable, with each variable sampled independently. The sampled states are used as input to the observation and transition networks, and as targets for the transition network.

The reconstruction and transition cost terms can now be used in stochastic gradient descent on $\mathcal{F}$ in θ, by estimating the gradient $\nabla_\theta \mathcal{F}$ with Monte Carlo samples from the vari-

ational posterior q_ϕ. To minimize $\mathcal{F}$ also in ϕ, we need to perform backpropagation over discrete, stochastic variables (i.e. over s_t^μ sampled from q_ϕ). There are several methods for doing this (see Discussion). We use the reparameterization trick together with a relaxation of the Bernoulli distribution: the binary Con–crete (or Gumbel–Softmax) distribution (Maddison et al. (2016); Jang et al. (2016)).

2.2. The reparameterization trick and the Con–crete distribution

Denoting the i*th* dimension of state s_t as $s_{t,i}$, we consider the i*th* output of the encoder at time t to correspond to $x_{t,i} = \text{logit}(q_\phi(s_{t,i} = 1|o_{t-k:t}))$. Following Maddison et al. (2016), we note that we can achieve a Bernoulli distribution by sampling according to $s_{t,i} = H(x_{t,i} + L)$, where H is the Heaviside step function and L is a logistic random variable. In this form, the stochastic component L is fully independent of ϕ, and we can simply backpropagate through the deterministic nodes (Kingma & Welling, 2013). However, the derivative of H is 0 almost everywhere. To address this, the Bernoulli distribution can be relaxed into a continuous Con–crete (*continuous* relaxation of *discrete*) distribution (Maddison et al., 2016). This corresponds to replacing the Heaviside non–linearity with a logistic non–linearity parameterized by the temperature λ:

$$\hat{s}_{t,i} = \frac{1}{1 + \exp(-(x_{t,i} + L)/\lambda)}, \tag{8}$$

with $\hat{s}_{t,i} \in [0,1]$. We use Con–crete samples from the encoder output for the input to both the reconstruction and transition networks and for the targets of the transition network, with temperatures taken from those suggested in (Maddison et al., 2016): $\lambda_1 = 2/3$ for the posterior distribution and $\lambda_2 = 0.5$ for evaluating the transition log–probabilities. The Con–crete relaxation corresponds to replacing the discrete joint Bernoulli samples s_t in the previous loss functions with their corresponding joint Con–crete samples $\hat{s}_t$. We train the network by sampling minibatches of observations and actions $(o_{t-k-1:t}^\mu, a_t^\mu)$ from a replay memory (Riedmiller, 2005; Mnih et al., 2015) of transitions observed by the agent.

2.3. Learning a tabular transition model

The model as described learns a joint Con–crete posterior distribution $\hat{q}_\phi(\hat{s}_t|o_{t-k:t})$. We can recover a discrete joint Bernoulli distribution $q_\phi(s_t|o_{t-k:t})$ by replacing the logistic non–linearity with a Heaviside non–linearity (i.e. as $\lambda \to 0$ in Eq. 8).

For prioritized sweeping, we need to build a tabular model of the transition probabilities in the environment (i.e. $p(s_t|a_t, s_{t-1})$). We could consider extracting such a model from $p_{\theta_\mathcal{T}}(\hat{s}_t|a_t, \hat{s}_{t-1})$, the transition network used to train the encoder. However, this is problematic for several reasons. The transition network corresponds to Con–crete states, and is of a particularly simple form, where each dimension is sampled independently conditioned on the previous state and action. Moreover, the transition network is trained through stochastic gradient descent and therefore learns slowly; we want the agent to rapidly exploit new transition observations.

We therefore build a state transition table based purely on the encoder distribution $q_\phi(s_t|o_{t-k:t})$, by treating the most probable sequence of states under this distribution as observed data. Since each dimension of s_t is independent conditioned on the observations, the mode $\bar{s}_t$ at time t corresponds to a d–length binary string, where $\bar{s}_{t,i} = H(x_{t,i})$. Likewise, since states within an episode are assumed to be independent conditioned on the causal observation filter, the most probable state sequence for an episode is $\mathcal{S}^\mu = \{\bar{s}_0^\mu, \bar{s}_1^\mu \ldots, \bar{s}_{T^\mu}^\mu\}$. We therefore record a transition between $\bar{s}_{t-1}$ and $\bar{s}_t$ under action a_t for every step taken by the agent, and update the expected reward $\mathbb{E}[r|a_t, \bar{s}_{t-1}]$ in the table with the observed reward. Each binary string $\bar{s}$ is represented as a d–bit unsigned integer in memory.

This process corresponds to empirically estimating the transition probabilities and rewards by counting, with counts that are revised during training. For instance, assume the agent encounters states A, B and C successively in the environment. We record transitions $A \to B$ and $B \to C$ in the table, and store the raw observations along with the corresponding state assignments A, B and C in the replay memory. If the observations associated with B are later sampled from the replay memory and instead assigned to state D, we delete $A \to B$ and $B \to C$ from the table and add $A \to D$ and $D \to C$. Both the deletion and addition of transitions through training can change the Q-values.

2.4. Using the model for reinforcement learning

The Q-values in the table are updated continuously using the learned transition model $p(\bar{s}_t|a_t, \bar{s}_{t-1})$, expected rewards $\mathbb{E}[r|a_t, \bar{s}_{t-1}]$ and prioritized sweeping with small backups (Van Seijen & Sutton, 2013). Prioritized sweeping converges to the same solution as value iteration, but can be much more computationally efficient by focusing updates on states where the Q-values change most significantly.

Given an observation history $o_{t-k:t}$, the agent follows an ϵ–greedy policy using the Q-values $Q(\bar{s}_t, a)$ in the lookup table for all possible actions a. For any pair $(\bar{s}_t, a)$ that has not yet been observed, we estimate the Q-value using an experience–weighted average over the nearest neighbours to $\bar{s}_t$ in Hamming distance (see Supplementary Materials for details). This Hamming neighbour estimate is parameter–less, and generally much faster than searching for nearest neighbours in continuous space.

2.5. Implementation details

The prioritized backups described by Van Seijen & Sutton (2013) are performed serially with environment exploration. To decrease training time and improve performance, we performed backups independently, and in parallel, to environment exploration and training the deep network.

We implemented state tabulation and prioritized sweeping as two separate processes (running on different CPU cores). The tabulation process acts in the environment and trains the neural networks by sampling the replay memory. The sweeping process maintains the transition table and continuously updates the Q-values using prioritized sweeping.

To perform greedy actions, the tabulation process requests Q-values from the sweeping process. To update the transition table, the tabulation process sends transition updates (additions and deletions) to the sweeping process. Our implementation of the sweeping process performed $\sim$6000 backups/second, allowing the agent to rapidly propagate Q-value changes with little effect on the simulation time.

The pseudocode of VaST, and of our implementation of prioritized sweeping, are in the Supplementary Material.

3. Results

We evaluated the VaST agent on a series of navigation tasks implemented in the *VizDoom* environment (see Figure 3A, Kempka et al. (2016)). Each input frame consists of a 3–channel $[60 \times 80]$ pixel image of the 3D environment, collected by the agent at a position (x, y) and orientation θ. The agent rarely observes the exact same frame from a previous episode ($0.05\% - 0.3\%$ of the time in the mazes used here), making it ill–suited for a traditional tabular approach; yet the discovery of new transitions (particularly shortcuts) can have a significant effect on the global policy if leveraged by a model–based agent. We considered the relatively low–data regime (up to 2 million steps). Three actions were available to the agent: move forward, turn left and turn right; due to momentum in the game engine, these give rise to visually smooth trajectories. We also trained the agent on the Atari game Pong (Figure 7). For 3D navigation, we used only the current frame as input to the network, while we tested both 1– and 4–frame inputs for Pong.

We compared the performance of VaST against two recently published sample–efficient model–free approaches: Neural Episodic Control (NEC) (Pritzel et al., 2017) and Prioritized Double–DQN (Schaul et al., 2015). We used the structure of the DQN network in (Mnih et al., 2015) for both NEC and Prioritized D–DQN as well as the encoder of VaST (excluding the output layers). Full hyperparameters are given in the Supplementary Material.
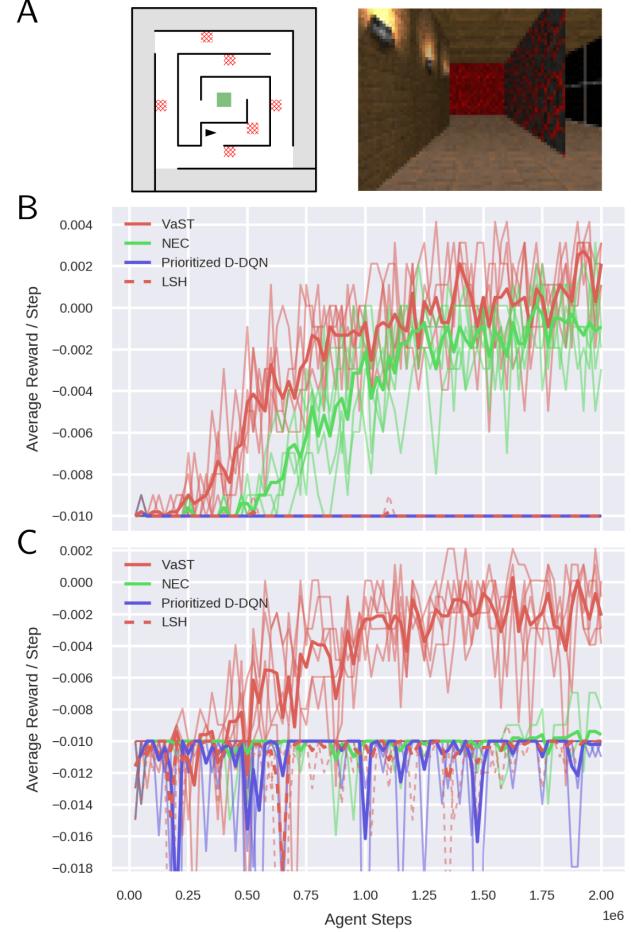
We also compared against prioritized sweeping using

Figure 3. **VaST learns quickly in complex mazes.** [*A*] The agent started at a random position and orientation in the outer rim of the 3D maze (highlighted in grey), and received a reward of +1 on reaching the center of the maze (highlighted in green), with a step penalty of -0.01. Red hatched areas correspond to the hazard regions in the second version of the task, where the agent received a penalty of -1 with a probability of 25%. We used a different texture for each wall in the maze, ending at a corner. An example observation is shown for an agent positioned at the black arrow. [*B*] Performance comparison between models for 5 individual runs with different random seeds (mean in bold). Rewards are very sparse ($\approx$ every 20 000 steps with a random policy); with longer training we expect DQN to improve. [*C*] Results for the second version of the task (including hazards).

Locality Sensitive Hashing (LSH) with random projections (Charikar, 2002), where each bit $\bar{s}_{t,i} = H(v_i \cdot o_t)$, and each fixed projection vector v_i had elements sampled from $\mathcal{N}(0, 1)$ at the beginning of training. The environment model and Q-values were determined as with VaST.

In the first task (Figure 3), the agents were trained to reach a reward of +1 in the center of a complex maze, starting from a random position and orientation in the outer region. In a
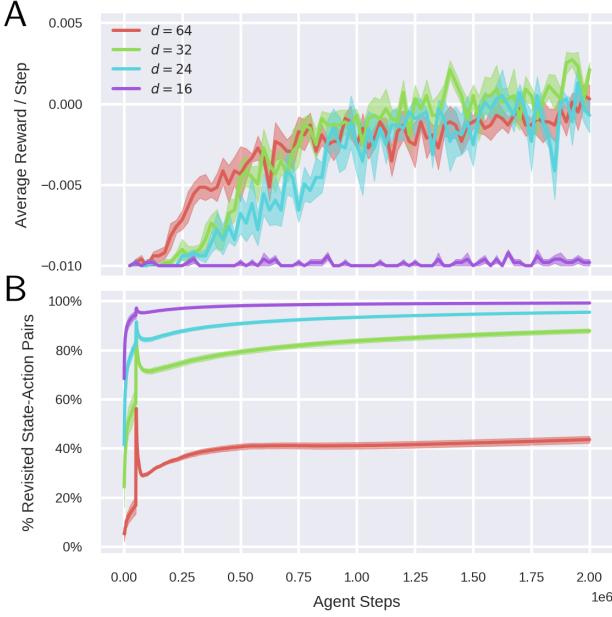
Figure 4. **Effect of latent dimensionality.** [*A*] Average test reward ± SEM during training for $d = 64$, $d = 32$, $d = 24$ and $d = 16$ for the task in Figure 3B (without hazards). [*B*] Cumulative percentage of revisited state–action pairs during learning. The sharp transition at 50 000 steps corresponds to the beginning of training the network.

second version of the task, we added six "hazard" regions which gave a penalty of -1 with a probability of 25% for each step. The agents were evaluated over a 1000–step test epoch, with $\epsilon = 0.05$, every 25 000 steps. VaST slightly outperformed NEC on the first version of the task and significantly outperformed all of the other models on the more difficult version (Figure 3C).

3.1. Dimensionality of the latent representation

We used $d = 32$ latent dimensions for the VaST agent in the navigation tasks, corresponding to a 32–bit representation of the environment. We examine the effect of d in Figure 4 and Supplementary Figure 1. High–dimensional representations ($d = 64$) tended to plateau at lower performance than representations with $d = 32$, but also resulted in faster initial learning in the more complex maze. The agent frequently revisited state–action pairs even using the high dimensional representation (Figure 4B). In general, we found that we could achieve similar performance with a wide range of dimensionalities; smaller mazes could be learned with as few as 8–16 bits (Supplementary Figure 1).

3.2. Sample efficiency

We hypothesized that the VaST agent would be particularly adept at rapidly modifying its policy in response to one new
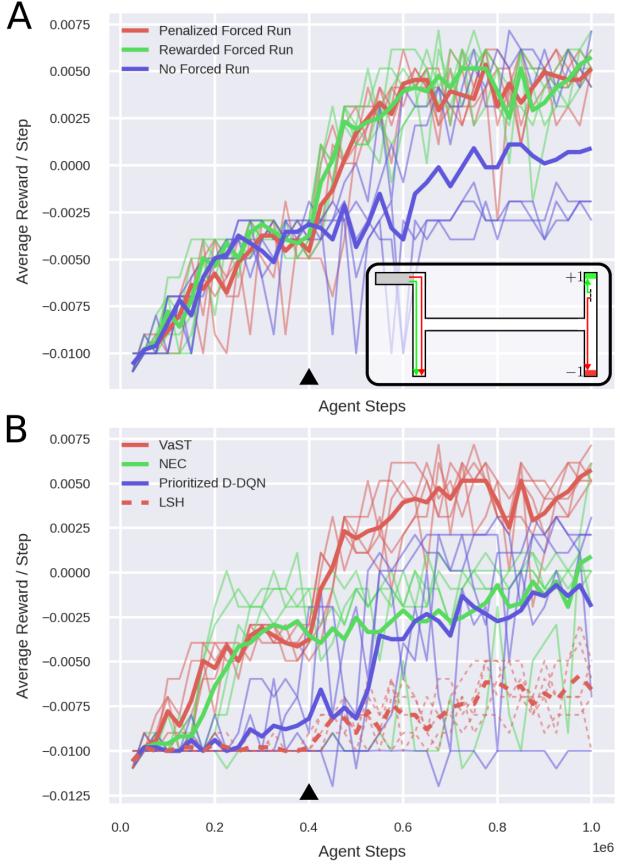

Figure 5. **VaST allows for rapid policy changes in response to single experiences.** [*A, Inset*] The agent learned to run from the starting area (grey) to a reward zone (green). After training, a new shortcut (teleporter) was introduced at the bottom of the left arm. The agent either observed no forced run, or a single forced run through the teleporter ending either in the rewarding (green) or the penalizing (red) terminal zone. The forced runs were 58 and 72 steps in length, respectively. [*A*] The teleporter was introduced after 400 000 steps (black triangle). The VaST agent's performance is shown for the three conditions: no forced run, rewarded forced run and penalized forced run. [*B*] Model performance comparison for rewarded forced runs.

experience. To test this, we designed an experiment in a 3D H–maze (Figure 5) that requires the agent to leverage a single experience of a new shortcut. The agent learned to run towards a terminal reward zone (+1) while avoiding a dead end and a terminal penalty zone (-1), with a step penalty of -0.01. After 400 000 steps of training (when the policy had nearly converged) we introduced a small change to the environment: running into the dead end would cause the agent to teleport to a position close to the reward zone, allowing it to reach the reward much faster. We informed the agent of the teleporter using a single forced run episode, in which the agent collected observations while running from the start box, through the teleporter, to either the reward
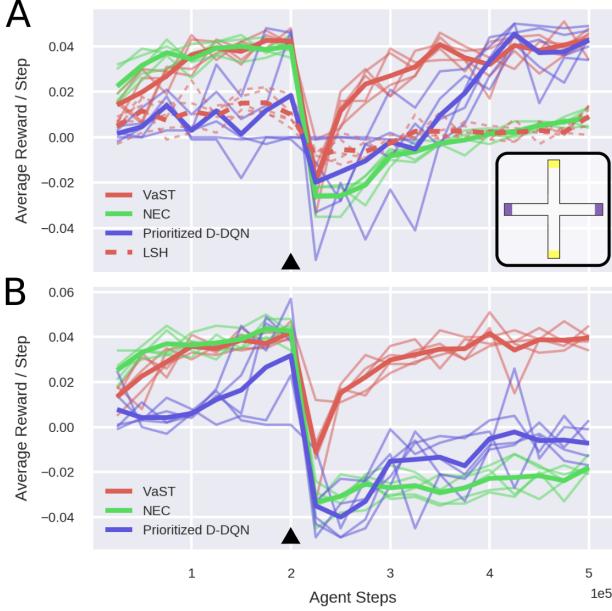
Figure 6. **VaST can adapt to changing rewards.** [*A, Inset*] The maze environment. Horizontal arms (purple) initially yielded a reward of +1 while vertical arms (yellow) yielded a penalty of -1. [*A*] After training for 200 000 steps (black triangle), the rewards and penalties in the maze were reversed. All agents used a replay memory size of $\mathcal{N} = 100\,000$ transitions. [*B*] The same task with a replay memory size of $\mathcal{N} = 500\,000$.

Figure 7. **Learning to play Pong.** Test epoch episode rewards for VaST trained over 5 million steps. We tested performance with no frame history ($k = 0$) and with 3 frames of history ($k = 3$) as input to the encoder q_ϕ. [*Inset*] Actual observations o_t (left) and reconstructed observations $\tilde{o}_t$ (right) for a trained agent.

zone or penalty zone under a fixed, predetermined policy. For the VaST agent, this corresponds to a single experience indicating a new shortcut: the transition between the states before and after the teleporter. After observing either the rewarded or penalized episode, performance rapidly improved as the agent adapted its policy to using the teleporter; in contrast, the agent discovered the teleporter on only 2/5 random seeds without the forced run. The agents switched to using the teleporter regularly approximately 20 000 steps after the forced run, on average (about 160 episodes). The VaST agent adapted to the teleporter more effectively than any of the other models (Figure 5B and Supplementary Figure 2).

3.3. Transfer learning: non–stationary rewards

VaST keeps separate statistics on immediate rewards and transition probabilities in the environment. If the rewards were suddenly modified, we hypothesized that the existing transition model could allow the agent to rapidly adjust its policy (after collecting enough data to determine that the expected immediate rewards had changed).

We tested this in the maze shown in Figure 6A (inset). Starting at a random position, the episode terminated at the end of any arm of the maze; the agent received a reward of +1 at the end of horizontal arms, and a penalty of -1 at the end of verti-
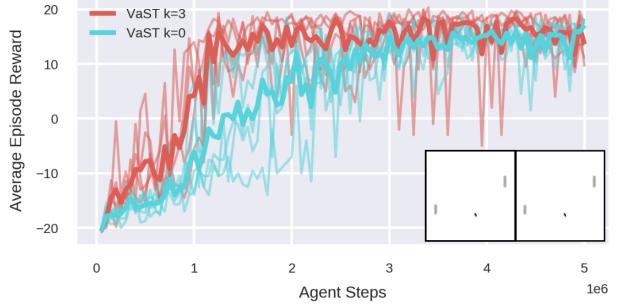
cal arms. The reward positions were reversed after 200 000 steps. We used two replay memory sizes ($\mathcal{N} = 100\,000$ and $\mathcal{N} = 500\,000$). Compared to NEC and Prioritized D–DQN, VaST both learned quickly in the initial phase and recovered quickly when the rewards were reversed. While both NEC and Prioritized D–DQN adapted faster with a smaller replay memory, VaST performed similarly in both conditions.

3.4. Training on Atari: Pong

In addition to 3D navigation, we trained the VaST agent to play the Atari game Pong using the Arcade Learning Environment (Bellemare et al., 2013), with preprocessing steps taken from Mnih et al. (2013). In Pong, a table tennis–like game played against the computer, the direction of the ball's movement is typically unclear given only the current frame as input. We therefore tried conditioning the posterior distribution q_ϕ on either the current frame ($k = 0$) or the current frame along with the last 3 frames of input ($k = 3$, following Mnih et al. (2013)). Using $k = 3$, the performance converged significantly faster on average (Figure 7). While the reconstruction cost was the same for $k = 0$ and $k = 3$, the transition and entropy cost terms decreased with additional frame history (Supplementary Figure 3).

4. Related Work

Model-based reinforcement learning Prioritized sweeping with small backups (Van Seijen & Sutton, 2013) is usually more efficient but similar to Dyna-Q (Sutton & Barto, 2018), where a model is learned and leveraged to update Q-values. Prioritized sweeping and Dyna-Q are background planning methods (Sutton & Barto, 2018), in that the action selection policy depends on Q-values that are updated in the background. In contrast, methods that rely on planning at decision time (like Monte Carlo Tree Search) estimate Q-values by expanding the decision tree from the current

state up to a certain depth and using the values of the leaf nodes. Both background and decision time planning methods for model-based reinforcement learning are well studied in tabular environments (Sutton & Barto, 2018). Together with function approximation, usually used to deal with high–dimensional raw (pixel) input, many recent works have focused on planning at decision time. Oh et al. (2017) and Farquhar et al. (2017), extending the predictron (Silver et al., 2017), train both an encoder neural network and an action-dependent transition network on the abstract states used to run rollouts up to a certain depth. Racanière et al. (2017) and Nagabandi et al. (2017) train a transition network on the observations directly. Racanière et al. (2017) additionally train a rollout policy, the rollout encoding and an output policy that aggregates different rollouts and a model-free policy. Planning at decision time is advantageous in situations like playing board games (Silver et al., 2017), where the transition model is perfectly known, many states are visited only once and a full tabulation puts high demands on memory. Conversely, background planning has the advantage of little computational cost at decision time, almost no planning cost in well–explored stationary environments and efficient policy updates after minor environment changes.

Successor representations for transfer learning The hybrid model–based/model–free approach of successor representations has recently been transferred from the tabular domain to deep function approximation (Dayan, 1993; Kulkarni et al., 2016). Under this approach, the agent learns a model of the immediate reward from each state and a model of the expected multi–step *future occupancy* of each successor state under the current policy. As in a model–based approach, the immediate rewards can be updated independently of the environment dynamics. However, the expected multi–step future occupancy is learned under a given policy, and the optimal policy will generally change with new rewards. The ability to generalize between tasks in an environment (as VaST does in Figure 6) then depends on the similarity between the existing and new policy. Recent work has proposed updating successor representations offline in a Dyna–like fashion using a transition model (Russek et al., 2017; Peng & Williams, 1993); we expect that prioritized sweeping with small backups could also be adapted to efficiently update tabular successor representations.

Navigation tasks Even though we demonstrate and evaluate our method mostly on navigation tasks, VaST does not contain any inductive bias tailored to navigation problems. Using auxiliary tasks (Mirowski et al., 2016; Jaderberg et al., 2016), we expect further improvement in navigation.

State aggregation in reinforcement learning State aggregation has a long history in reinforcement learning (Li et al., 2006; Sutton & Barto, 2018). To our knowledge,

VaST is the first approach that uses modern deep learning methods to learn useful and non-linear state discretisation. In earlier versions of our model we tried discretising with VAEs as used by (Blundell et al., 2016), with mixed success. The state aggregator $q_\phi(s_t|o_{t-k:t})$ of VaST can be seen as a byproduct of training a hybrid ANN–HMM. Different methods to train ANN-HMMs have been studied (Bengio et al., 1992; Tucker et al., 2017; Ng et al., 2016; Maddison et al., 2016). While none of these works study the binary representation of the latent states used by VaST for the generalization of Q-values, we believe it is worthwhile to explore other training procedures and potentially draw inspiration from the ANN-HMM literature.

5. Discussion

We found that the VaST agent could rapidly transform its policy based on limited new information and generalize between tasks in the same environment. In stationary problems, VaST performed better than competing models in complex 3D tasks where shortcut discovery played a significant role. Notably, VaST performs *latent learning*; it builds a model of the structure of the environment even when not experiencing rewards (Tolman & Honzik, 1930).

We also trained VaST to play the Atari game Pong. In general, we had less initial success training the agent on other Atari games. We suspect that many Atari games resemble deterministic tree Markov Decision Processes, where each state has exactly one predecessor state. In these tasks, prioritized sweeping conveys no benefit beyond MFEC (Brea, 2017). In contrast, intrinsically continuous tasks like 3D navigation can be well–characterized by a non–treelike tabular representation (e.g. by using a discretisation of (x,y,θ), where θ denotes the agent's orientation).

VaST differs from many deep reinforcement learning models in that the neural network is entirely reward–agnostic, where training corresponds to an unsupervised learning task. Many other possible architectures exist for the unsupervised tabulator; for instance, a Score Function Estimator such as NVIL (Mnih & Gregor, 2014; Mnih & Rezende, 2016; Tucker et al., 2017) could be used in place of the Con–crete relaxation for discrete stochastic sampling. In addition, while we chose here to show the strengths of a purely model–based approach, one could also consider alternative models that use value information for tabulation, resulting in hybrid model–based/model–free architectures.

The past decade has seen considerable efforts towards using deep networks to adapt tabular RL techniques to high–dimensional and continuous environments. Here, we show how the opposite approach – using deep networks to instead transform the environment into a tabular one – can enable the use of powerful model–based techniques.

Acknowledgements

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, Jun 2013.

Bengio, Y., De Mori, R., Flammia, G., and Kompe, R. Global optimization of a neural network-hidden markov model hybrid. *IEEE Transactions on Neural Networks*, 3 (2):252259, Mar 1992. ISSN 1045-9227. doi: 10.1109/72.125866.

Blundell, C., Uria, B., Pritzel, A., Li, Y., Ruderman, A., Leibo, J. Z., Rae, J., Wierstra, D., and Hassabis, D. Model-free episodic control. *arXiv preprint arXiv:1606.04460*, 2016.

Brea, J. Is prioritized sweeping the better episodic control? *ArXiv e-prints arXiv:1711.06677*, 2017.

Charikar, M. S. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pp. 380–388. ACM, 2002.

Dayan, P. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993.

Farquhar, G., Rocktäschel, T., Igl, M., and Whiteson, S. TreeQN and ATreeC: Differentiable Tree Planning for Deep Reinforcement Learning. *ArXiv e-prints*, October 2017.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

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.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaśkowski, W. ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In *IEEE Conference on Computational Intelligence and Games*, pp. 341–348, Santorini, Greece, Sep 2016. IEEE.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Kulkarni, T. D., Saeedi, A., Gautam, S., and Gershman, S. J. Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*, 2016.

Li, L., Walsh, T. J., and Littman, M. L. Towards a unified theory of state abstraction for MDPs. In *ISAIM*, 2006.

Maddison, C. J., Mnih, A., and Whye Teh, Y. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. *ArXiv e-prints arXiv:1611.00712*, November 2016.

Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A. J., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., Kumaran, D., and Hadsell, R. Learning to Navigate in Complex Environments. *ArXiv e-prints*, November 2016.

Mnih, A. and Gregor, K. Neural variational inference and learning in belief networks. In Xing, E. P. and Jebara, T. (eds.), *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pp. 1791–1799, Bejing, China, 22–24 Jun 2014. PMLR.

Mnih, A. and Rezende, D. Variational inference for monte carlo objectives. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 2188–2196, New York, New York, USA, 20–22 Jun 2016. PMLR.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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), 2015.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1928–1937, 2016.

Moore, A. W. and Atkeson, C. G. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning*, 13(1):103–130, 1993.

Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning. *ArXiv e-prints*, August 2017.

Ng, Y. C., Chilinski, P. M., and Silva, R. Scaling factorial hidden markov models: Stochastic variational inference without messages. In *Advances in Neural Information Processing Systems 29*, pp. 4044–4052. 2016.

Oh, J., Singh, S., and Lee, H. Value prediction network. In *Advances in Neural Information Processing Systems*, pp. 6120–6130, 2017.

Peng, J. and Williams, R. J. Efficient learning and planning within the dyna framework. *Adaptive Behavior*, 1(4): 437–454, 1993.

Pritzel, A., Uria, B., Srinivasan, S., Badia, A. P., Vinyals, O., Hassabis, D., Wierstra, D., and Blundell, C. Neural episodic control. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, 2017.

Racanière, S., Weber, T., Reichert, D., Buesing, L., Guez, A., Rezende, D. J., Badia, A. P., Vinyals, O., Heess, N., Li, Y., et al. Imagination-augmented agents for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 5694–5705, 2017.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Riedmiller, M. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pp. 317–328. Springer, 2005.

Russek, E. M., Momennejad, I., Botvinick, M. M., Gershman, S. J., and Daw, N. D. Predictive representations can link model-based reinforcement learning to model-free mechanisms. *PLOS Computational Biology*, 13(9): e1005768, 2017.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *ArXiv e-prints*, December 2017.

Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., and Degris, T. The predictron: End-to-end learning and planning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, 2017.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, (in progress) second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Tolman, E. C. and Honzik, C. H. Introduction and removal of reward, and maze performance in rats. *University of California publications in psychology*, 1930.

Tucker, G., Mnih, A., Maddison, C. J., Lawson, D., and Sohl-Dickstein, J. REBAR: Low-variance, unbiased gradient estimates for discrete latent variable models. *ArXiv e-prints*, March 2017.

Van Seijen, H. and Sutton, R. S. Efficient planning in MDPs by small backups. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, 2013.

Supplementary Material:
Efficient Model–Based Deep Reinforcement Learning with Variational State Tabulation

1 VaST pseudocode

Algorithm 1 Variational State Tabulation.

Initialize replay memory $\mathcal{M}$ with capacity $\mathcal{N}$

Initialize sweeping table process $\mathcal{B}$ with transition add queue $\mathcal{Q}^+$ and delete queue $\mathcal{Q}^-$

 1: **for** each episode **do**
 2:    Set $t \leftarrow 0$
 3:    Get initial observations o_0
 4:    Process initial state $\bar{s}_0 \leftarrow \arg\max_s q_\phi(s|o_0)$
 5:    Store memory $(o_0, \bar{s}_0)$ in $\mathcal{M}$
 6:    **while** not terminal **do**
 7:       Set $t \leftarrow t + 1$
 8:       Take action a_t with ϵ-greedy strategy based on $\tilde{Q}(s_{t-1}, a)$ from $\mathcal{B}$
 9:       Receive r_t, o_t
10:       Process new state $\bar{s}_t \leftarrow \arg\max_s q_\phi(s|o_{t-k:t})$
11:       Store memory $(o_t, \bar{s}_t, a_t, r_t)$ in $\mathcal{M}$
12:       Put transition $(\bar{s}_{t-1}, a_t, r_t, \bar{s}_t)$ on $\mathcal{Q}^+$
13:       **if** training step **then**
14:          Set gradient list $\mathcal{G} \leftarrow \{\}$
15:          **for** sample in minibatch **do**
16:             Get $(o_{j-k-1:j}, a_j)$ from random episode and step j in $\mathcal{M}$
17:             Process $q_\phi(s_{j-1}|o_{j-k-1:j-1})$, $q_\phi(s_j|o_{j-k:j})$ with encoder
18:             Sample $\hat{s}_{j-1}$, $\hat{s}_j \sim \hat{q}_\phi$ with temperature λ
19:             Process $p_\theta(o_j|\hat{s}_j)$, $p_\theta(\hat{s}_j|a_j, \hat{s}_{j-1})$ with decoder and transition network
20:             Append $\nabla_{\theta,\phi} \mathcal{F}(\theta, \phi; o_{j-k-1:j})$ to $\mathcal{G}$
21:             **for** i in $\{j-1, j\}$ **do**
22:                Process $\bar{s}_i^{new} \leftarrow \arg\max_s q_\phi(s|o_{i-k:i})$
23:                Get $(\bar{s}_{i-1}, a_i, r_i, \bar{s}_i, a_{i+1}, r_{i+1}, \bar{s}_{i+1})$ from $\mathcal{M}$
24:                **if** $\bar{s}_i \neq \bar{s}_i^{new}$ **then**
25:                   Put $(\bar{s}_{i-1}, a_i, r_i, \bar{s}_i)$, $(\bar{s}_i, a_{i+1}, r_{i+1}, \bar{s}_{i+1})$ on $\mathcal{Q}^-$
26:                   Put $(\bar{s}_{i-1}, a_i, r_i, \bar{s}_i^{new})$, $(\bar{s}_i^{new}, a_{i+1}, r_{i+1}, \bar{s}_{i+1})$ on $\mathcal{Q}^+$
27:                   Update $\bar{s}_i \leftarrow \bar{s}_i^{new}$ in $\mathcal{M}$
28:                **end if**
29:             **end for**
30:          **end for**
31:          Perform a gradient descent step according to $\mathcal{G}$ with given optimizer
32:       **end if**
33:    **end while**
34: **end for**

---

2 Details to prioritized sweeping algorithm

We follow the "Prioritized Sweeping with reversed full backups" algorithm (Van Seijen and Sutton, 2013) with some adjustments: a subroutine is added for transition deletions, and priority sweeps are performed continuously except when new transition updates are received. The Q-values of unobserved state–action pairs are never used, so we simply initialize them to 0. Finally, we kept a model of the expected immediate rewards $\mathbb{E}[r|s,a]$ explicitly, although this is not necessary and was not used in any of the experiments presented; we omit it here for clarity.

In the algorithm, discretised states $\bar{s}$ are simplified to s.

Algorithm 2 Prioritized Sweeping Process.

Initialize $V(s) = U(s) = 0$ for all s
Initialize $Q(s,a) = 0$ for all s, a
Initialize $N_{sa}, N_{sa}^{s'} = 0$ for all s, a, s'
Initialize priority queue $\mathcal{P}$ with minimum priority cutoff p_{min}
Initialize add queue $\mathcal{Q}^+$ and delete queue $\mathcal{Q}^-$

1: **while** True **do**
2: **while** $\mathcal{Q}^+$, $\mathcal{Q}^-$ empty **do**
3:   Remove top state s' from $\mathcal{P}$
4:   $\Delta U \leftarrow V(s') - U(s')$
5:   $U(s') \leftarrow V(s')$
6:   **for all** (s,a) pairs with $N_{sa}^{s'} > 0$ **do**
7:    $Q(s,a) \leftarrow Q(s,a) + \gamma N_{sa}^{s'}/N_{sa} \cdot \Delta U$
8:    $V(s) \leftarrow \max_b\{Q(s,b)|N_{sb} > 0\}$
9:    add/update s in $\mathcal{P}$ with priority $|U(s) - V(s)|$ if $|U(s) - V(s)| > p_{min}$
10:   **end for**
11: **end while**
12: **for** (s,a,r,s') in $\mathcal{Q}^+$ **do**
13:   $N_{sa} \leftarrow N_{sa} + 1$; $N_{sa}^{s'} \leftarrow N_{sa}^{s'} + 1$
14:   $Q(s,a) \leftarrow [Q(s,a)(N_{sa} - 1) + r + \gamma U(s')]/N_{sa}$
15:   $V(s) \leftarrow \max_b\{Q(s,b)|N_{sb} > 0\}$
16:   add/update s in $\mathcal{P}$ with priority $|U(s) - V(s)|$ if $|U(s) - V(s)| > p_{min}$
17: **end for**
18: **for** (s,a,r,s') in $\mathcal{Q}^-$ **do**
19:   $N_{sa} \leftarrow N_{sa} - 1$; $N_{sa}^{s'} \leftarrow N_{sa}^{s'} - 1$
20:   **if** $N_{sa} > 0$ **then**
21:    $Q(s,a) \leftarrow [Q(s,a)(N_{sa} + 1) - (r + \gamma U(s'))]/N_{sa}$
22:   **else**
23:    $Q(s,a) \leftarrow 0$
24:   **end if**
25:   **if** $\sum_b N_{sb} > 0$ **then**
26:    $V(s) \leftarrow \max_b\{Q(s,b)|N_{sb} > 0\}$
27:   **else**
28:    $V(s) \leftarrow 0$
29:   **end if**
30:   add/update s in $\mathcal{P}$ with priority $|U(s) - V(s)|$ if $|U(s) - V(s)| > p_{min}$
31: **end for**
32: **end while**

---

3   Details to Q-value estimation

Here, we simplify the discretised states $\bar{s}$ to s for clarity. We denote $\mathcal{S}$ as the set of all states corresponding to d–length binary strings, $\tilde{Q}(s, a)$ as the Q-value estimate used for action selection, and $Q(s, a)$ as the Q-value for a state–action pair in the lookup table as determined by prioritized sweeping (which is only used if (s, a) has been observed at least once).

In order to calculate $\tilde{Q}(s_t, a)$ for a particular state–action pair, we first determine the Hamming distance m to the nearest neighbour(s) $s \in \mathcal{S}$ for which the action a has already been observed, i.e.

$$m = \min_{s \in \mathcal{S}} \{D(s_t, s) | N_{sa} > 0\}, \tag{1}$$

where $D(s_t, s)$ is the Hamming distance between s_t and s and N_{sa} denotes the number of times that action a has been taken from state s. We then define the set $\mathcal{S}_{tm}$ of all m–nearest neighbours to state s_t,

$$\mathcal{S}_{tm} = \{s \in \mathcal{S} | D(s_t, s) = m\}, \tag{2}$$

and the Q-value estimate used for action selection is then given by

$$\tilde{Q}(s_t, a) := \frac{\sum_{s \in \mathcal{S}_{tm}} N_{sa} Q(s, a)}{\sum_{s \in \mathcal{S}_{tm}} N_{sa}}. \tag{3}$$

If (s_t, a) has already been observed, then $m = 0$, $\mathcal{S}_{tm} = \{s_t\}$ and $\tilde{Q}(s_t, a) = Q(s_t, a)$. If $m = 1$, $\tilde{Q}(s_t, a)$ corresponds to an experience–weighted average over all states s with a Hamming distance of 1 from s_t, $m = 2$ to the average over neighbours with a Hamming distance of 2 etc.

$\tilde{Q}(s_t, a)$ can be seen as the Q-value of an abstract aggregate state s_{tm} consisting of the m–nearest neighbours to s_t. To show this, we introduce the index set of past experiences $\mathcal{E}_{sa} = \{(\tau, \mu) | s_\tau^\mu = s, a_\tau^\mu = a\}$ that contains all the time indices τ for all episodes μ where action a was chosen in state s (taking into account all reassignments as described in section 2.3 of the main text and in Algorithm 1). With the above definition of N_{sa} we see that $N_{sa} = |\mathcal{E}_{sa}|$, i.e. there are N_{sa} elements in the set $\mathcal{E}_{sa}$. With this and the update mechanism of prioritized sweeping (Algorithm 2) we can write

$$Q(s, a) = \frac{1}{N_{sa}} \sum_{\tau, \mu \in \mathcal{E}_{sa}} r_\tau^\mu + \gamma \frac{1}{N_{sa}} \sum_{\tau, \mu \in \mathcal{E}_{sa}} V(s_{\tau+1}^\mu), \tag{4}$$

where $V(s) = \max_b \{Q(s, b) | N_{sb} > 0\}$. Substituting this into Equation 3, we obtain

$$\tilde{Q}(s_t, a) = \frac{\sum_{s \in \mathcal{S}_{tm}} \left[\sum_{\tau, \mu \in \mathcal{E}_{sa}} r_\tau^\mu + \gamma \sum_{\tau, \mu \in \mathcal{E}_{sa}} V(s_{\tau+1}^\mu) \right]}{\sum_{s \in \mathcal{S}_{tm}} N_{sa}}. \tag{5}$$

We now consider an aggregate state s_{tm} by treating all states $s \in \mathcal{S}_{tm}$ as equivalent, i.e. $\mathcal{E}_{s_{tm}a} = \{(\tau, \mu) | s_\tau^\mu \in \mathcal{S}_{tm}, a_\tau^\mu = a\}$. With this definition we get $\sum_{s \in \mathcal{S}_{tm}} \sum_{\tau, \mu \in \mathcal{E}_{sa}} = \sum_{\tau, \mu \in \mathcal{E}_{s_{tm}a}}$ and we obtain

$$\tilde{Q}(s_t, a) = \frac{\left[\sum_{\tau, \mu \in \mathcal{E}_{s_{tm}a}} r_\tau^\mu + \gamma \sum_{\tau, \mu \in \mathcal{E}_{s_{tm}a}} V(s_{\tau+1}^\mu) \right]}{N_{s_{tm}a}} \tag{6}$$

$$= Q(s_{tm}, a),$$

where we used Equation 4 to obtain the second equality.
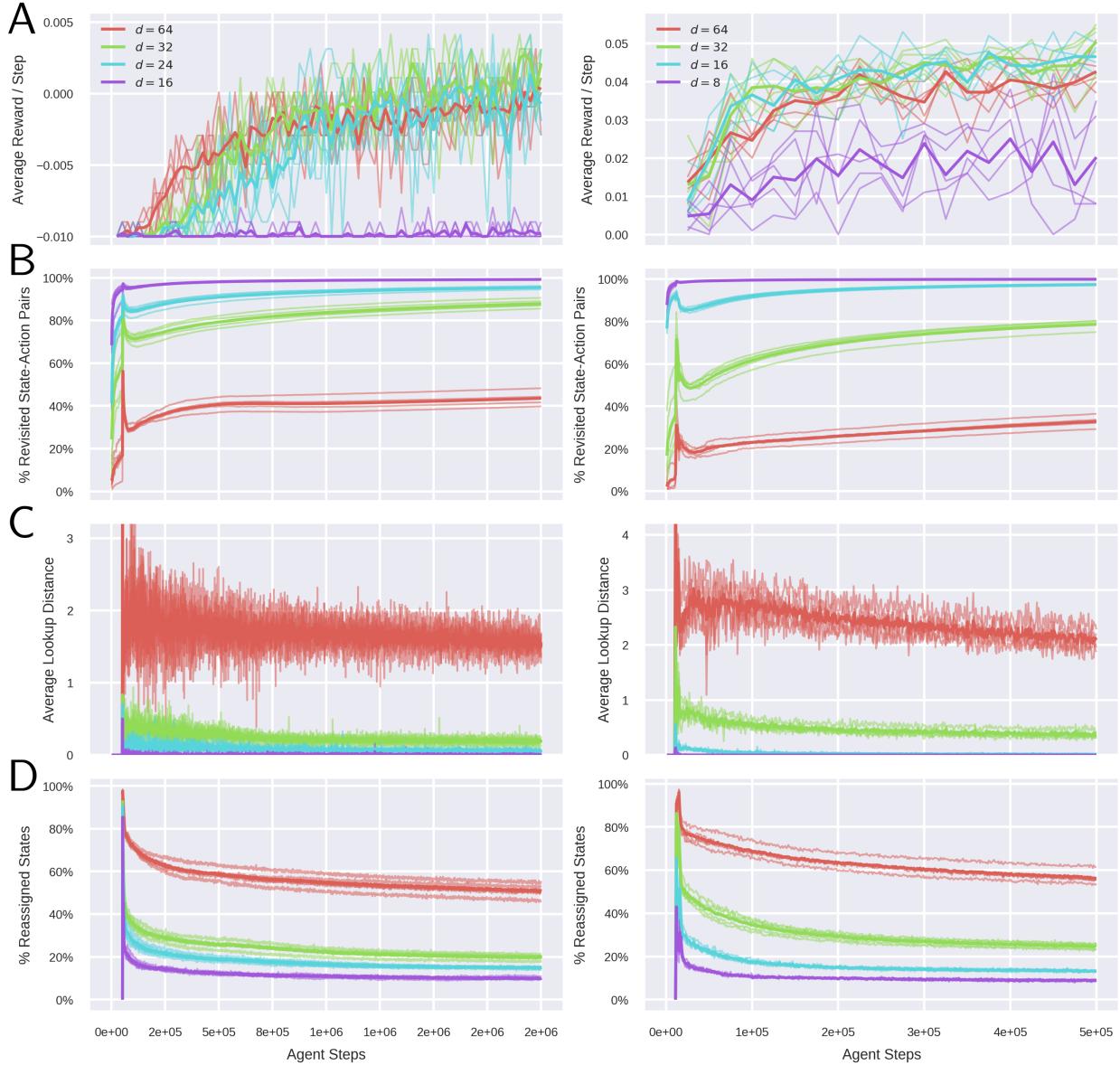
4   Extended latent dimensionality analysis

Figure 1: Effect of latent dimensionality in a large maze (left column, Figure 3B in main text) and a small maze (right column, Figure 6 in main text). [A] Average reward. [B] Cumulative percentage of revisited state–action pairs over the course of training. The sharp transition at 50 000 steps corresponds to the beginning of training. [C] The average lookup distance m as a function of time. [D] The average percentage of observations from a minibatch that were reassigned to a different state during training.
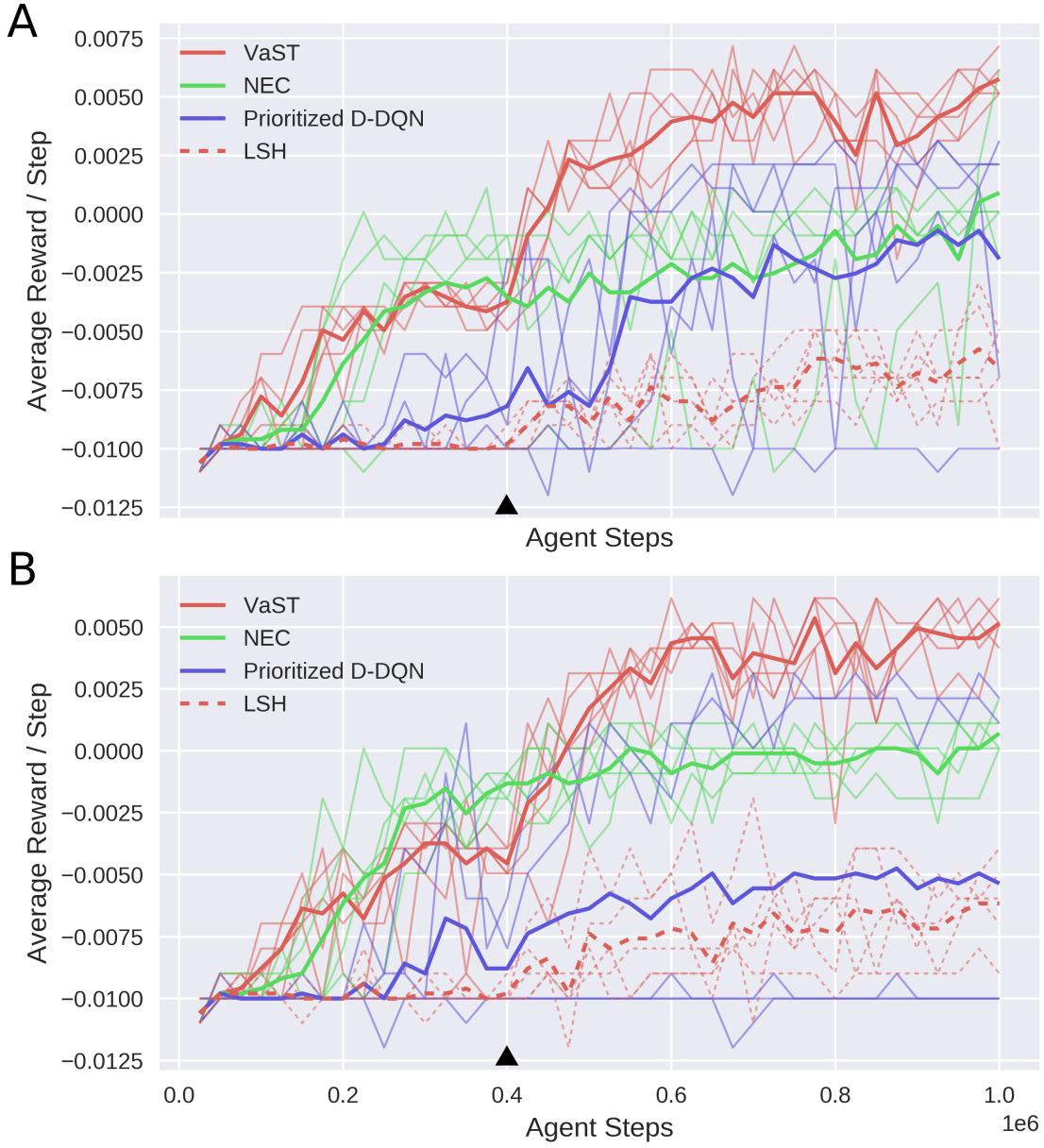
5 Extended sample efficiency results

Figure 2: Performance comparison between models for [A] rewarded forced runs (identical to Figure 5B in main text) and [B] penalized forced runs. Black arrows indicate addition of teleporter and forced runs.
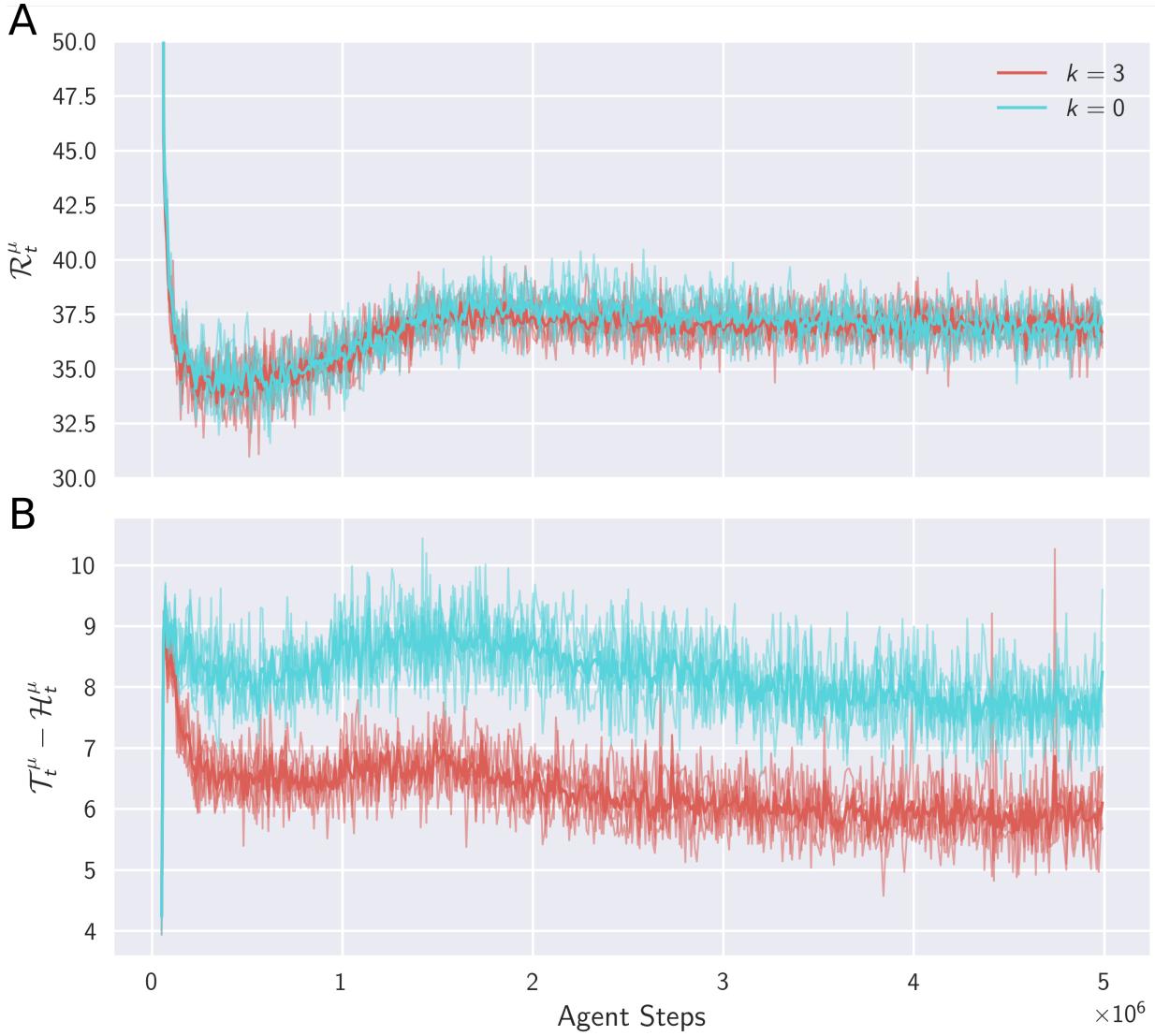
6 Effect of training on frame histories

Figure 3: The free energy cost function over the course of training on Pong, broken into [A] the reconstruction terms and [B] the transition and entropy terms, conditioning on three additional past frames of observations ($k = 3$) and no additional frames ($k = 0$). Training with past frames as input resulted in faster learning on Pong (main text, Figure 7). As shown here, training on past frames conveys no added benefit in reconstructing the current frame, but instead decreases the additional cost terms.

7 Hyperparameters

7.1 3D Navigation

For the three network–based models, hyperparameters were chosen based on a coarse parameter search in two mazes (Figure 3 excluding the hazards and Figure 5 excluding the teleporter), using the previously published hyperparameters as a starting point for the baselines (Pritzel et al., 2017; Schaul et al., 2015; Mnih et al., 2015). In all mazes except the smaller Plus–Maze, the agents explored randomly for 50 000 steps to initialize the replay memory before training; ϵ was then annealed from 1 to 0.1 over 200 000 steps. In the Plus–Maze, the agents explored randomly for 10 000 steps and ϵ was annealed over 40 000 steps. We used $\epsilon = 0.05$ for evaluation during test epochs, which lasted for 1000 steps. In all tasks we used a discount factor of 0.99.

The encoder of VaST and the networks for NEC and Prioritized D–DQN all shared the same architecture, as published in (Mnih et al., 2015), with ReLU activations. For all three networks, we used the Adam optimizer (Kingma and Ba, 2014) with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1e-8$, and trained on every 4th step. Unless otherwise stated, we used a replay memory size of $\mathcal{N} = 500\,000$ transitions.

7.1.1 VaST

We used a latent dimensionality of $d = 32$ unless otherwise stated. For training, we used a minibatch size of 128 and a learning rate of $2 \times 1e-4$. For sweeping, we used $p_{min} = 5 \times 1e-5$. For the Con–crete relaxation, we used the temperatures suggested by Maddison et al. (2016): $\lambda_1 = 2/3$ for sampling from the posterior and evaluating the posterior log–probability and $\lambda_2 = 0.5$ for evaluating the transition and initial state log–probabilities.

For the decoder architecture, we used a fully–connected layer with 256 units, followed by 4 deconvolutional layers with 4×4 filters and stride 2, and intermediate channel depths of 64, 64 and 32 respectively. We used an MLP with 3 hidden layers (with 512, 256 and 512 units respectively) for each action in the transition network.

7.1.2 NEC

We used a latent embedding of size 64, $n_s = 50$ for the n–step Q-value backups, and $\alpha = 0.1$ for the tabular learning rate. We performed a 50 approximate nearest–neighbour lookup using the ANNoy library (pypi.python.org/pypi/annoy) on Differentiable Neural Dictionaries of size 500 000 for each action. For training, we used a minibatch size of 32 and a learning rate of $5 \times 1e-5$.

7.1.3 Prioritized D–DQN

We used the rank–based version of Prioritized DQN with $\alpha = 0.7$ and $\beta = 0.5$ (annealed to 1 over the course of training). We used a minibatch size of 32 and a learning rate of $1e-4$ and updated the target network every 2000 steps.

7.1.4 LSH

The LSH–based algorithm does not use a neural network or replay memory, since the embedding is based on fixed random projections. We achieved the best results with $d = 64$ for the latent dimensionality. For prioritized sweeping, we used $p_{min} = 5 \times 1e-5$.

7.2 Atari: Pong

We used a latent dimensionality of $d = 64$, a replay memory size of $\mathcal{N} = 1\,000\,000$ transitions, and annealed ϵ over 1 000 000 steps. All other hyperparameters were the same as for navigation.

References

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

C. J. Maddison, A. Mnih, and Y. Whye Teh. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. *ArXiv e-prints arXiv:1611.00712*, November 2016.

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

Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adrià Puigdomènech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, 2017.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Harm Van Seijen and Richard S Sutton. Efficient planning in MDPs by small backups. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, 2013.