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 $\mathcal{Q}^-$

for each episode **do**
  Set $t \leftarrow 0$
  Get initial observations o_0
  Process initial state $\bar{s}_0 \leftarrow \arg\max_s q_\phi(s|o_0)$
  Store memory $(o_0, \bar{s}_0)$ in $\mathcal{M}$
  while not terminal **do**
    Set $t \leftarrow t + 1$
    Take action a_t with ϵ-greedy strategy based on $\tilde{Q}(s_{t-1}, a)$ from $\mathcal{B}$
    Receive r_t, o_t
    Process new state $\bar{s}_t \leftarrow \arg\max_s q_\phi(s|o_{t-k:t})$
    Store memory $(o_t, \bar{s}_t, a_t, r_t)$ in $\mathcal{M}$
    Put transition $(\bar{s}_{t-1}, a_t, r_t, \bar{s}_t)$ on $\mathcal{Q}^+$
    if training step **then**
      Set gradient list $\mathcal{G} \leftarrow \{\}$
      for sample in minibatch **do**
        Get $(o_{j-k-1:j}, a_j)$ from random episode and step j in $\mathcal{M}$
        Process $q_\phi(s_{j-1}|o_{j-k-1:j-1})$, $q_\phi(s_j|o_{j-k:j})$ with encoder
        Sample $\hat{s}_{j-1}$, $\hat{s}_j \sim \hat{q}_\phi$ with temperature λ
        Process $p_\theta(o_j|\hat{s}_j)$, $p_\theta(\hat{s}_j|a_j, \hat{s}_{j-1})$ with decoder and transition network
        Append $\nabla_{\theta,\phi}\mathcal{F}(\theta, \phi; o_{j-k-1:j})$ to $\mathcal{G}$
        for i in $\{j-1, j\}$ **do**
          Process $\bar{s}_i^{new} \leftarrow \arg\max_s q_\phi(s|o_{i-k:i})$
          Get $(\bar{s}_{i-1}, a_i, r_i, \bar{s}_i, a_{i+1}, r_{i+1}, \bar{s}_{i+1})$ from $\mathcal{M}$
          if $\bar{s}_i \neq \bar{s}_i^{new}$ **then**
            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 $\mathcal{Q}^-$
            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 $\mathcal{Q}^+$
            Update $\bar{s}_i \leftarrow \bar{s}_i^{new}$ in $\mathcal{M}$
          end if
        end for
      end for
      Perform a gradient descent step according to $\mathcal{G}$ with given optimizer
    end if
  end while
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, discretized 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 $\mathcal{Q}^{-}$

while True **do**
  while $\mathcal{Q}^{+}$, $\mathcal{Q}^{-}$ empty **do**
    Remove top state s' from $\mathcal{P}$
    $\Delta U \leftarrow V(s') - U(s')$
    $U(s') \leftarrow V(s')$
    for all (s,a) pairs with $N_{sa}^{s'} > 0$ **do**
      $Q(s,a) \leftarrow Q(s,a) + \gamma N_{sa}^{s'}/N_{sa} \cdot \Delta U$
      $V(s) \leftarrow \max_b \{Q(s,b) | N_{sb} > 0\}$
      add/update s in $\mathcal{P}$ with priority $|U(s) - V(s)|$ if $|U(s) - V(s)| > p_{min}$
    end for
  end while
  for (s,a,r,s') in $\mathcal{Q}^{+}$ **do**
    $N_{sa} \leftarrow N_{sa} + 1$; $N_{sa}^{s'} \leftarrow N_{sa}^{s'} + 1$
    $Q(s,a) \leftarrow [Q(s,a)(N_{sa} - 1) + r + \gamma U(s')]/N_{sa}$
    $V(s) \leftarrow \max_b \{Q(s,b) | N_{sb} > 0\}$
    add/update s in $\mathcal{P}$ with priority $|U(s) - V(s)|$ if $|U(s) - V(s)| > p_{min}$
  end for
  for (s,a,r,s') in $\mathcal{Q}^{-}$ **do**
    $N_{sa} \leftarrow N_{sa} - 1$; $N_{sa}^{s'} \leftarrow N_{sa}^{s'} - 1$
    if $N_{sa} > 0$ **then**
      $Q(s,a) \leftarrow [Q(s,a)(N_{sa} + 1) - (r + \gamma U(s'))]/N_{sa}$
    else
      $Q(s,a) \leftarrow 0$
    end if
    if $\sum_b N_{sb} > 0$ **then**
      $V(s) \leftarrow \max_b \{Q(s,b) | N_{sb} > 0\}$
    else
      $V(s) \leftarrow 0$
    end if
    add/update s in $\mathcal{P}$ with priority $|U(s) - V(s)|$ if $|U(s) - V(s)| > p_{min}$
  end for
end while

3 Details to Q-value estimation

Here, we simplify the discretized 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 $\mathcal{E}_{sa} = \{(\tau,\mu)|s^\mu_\tau = s, a^\mu_\tau = a\}$ 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

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

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

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

We now consider an aggregate state s_{tm} by treating all states $s \in \mathcal{S}_{tm}$ as equivalent, i.e. $\mathcal{E}_{s_{tm}a} = \{(\tau,\mu)|s^\mu_\tau \in \mathcal{S}_{tm}, a^\mu_\tau = a\}$. 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

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

where we used Equation 4 to obtain the second equality.

4 Extended latent dimensionality analysis

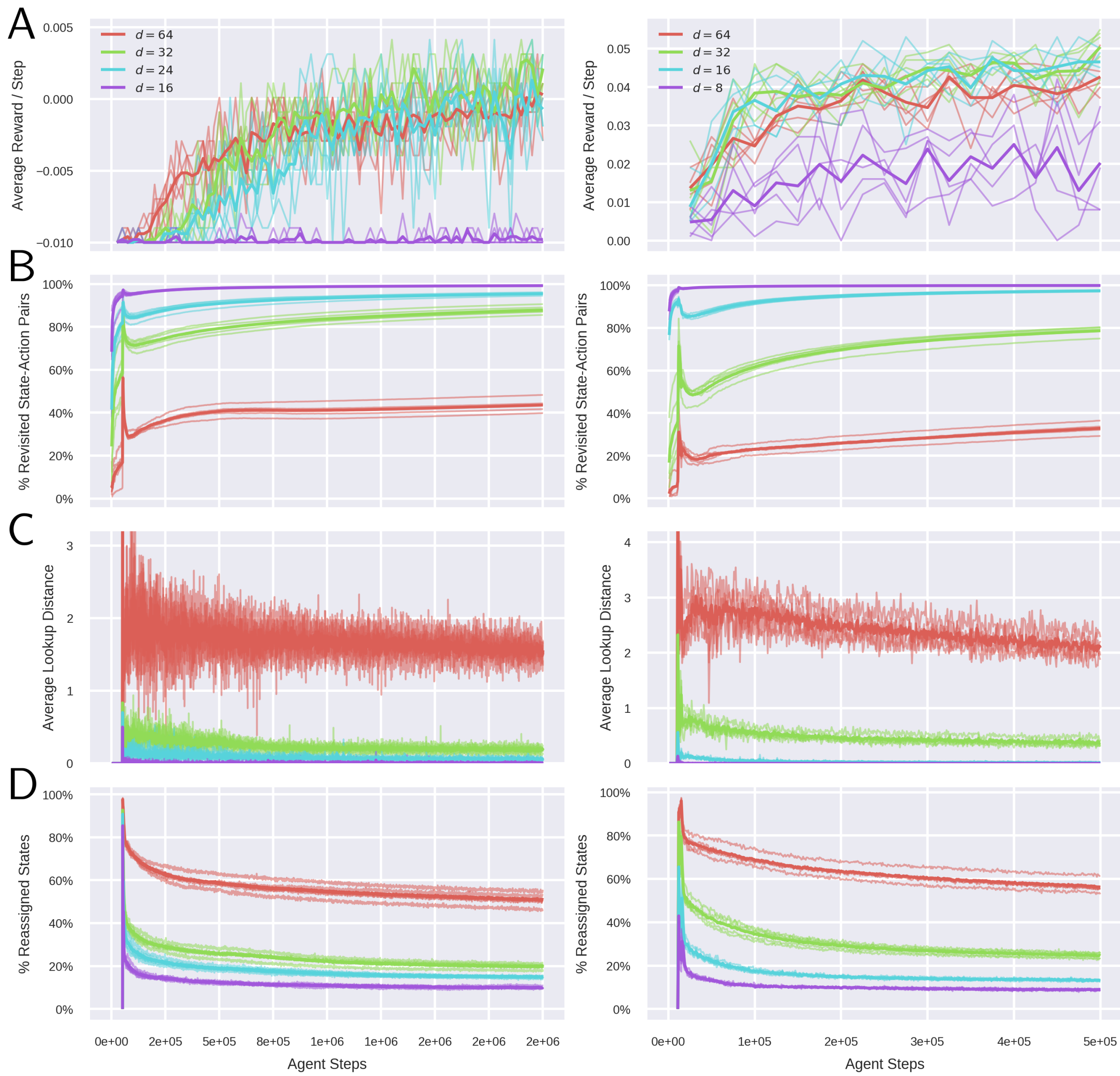

Figure 1: Effect of latent dimensionality in a large maze (left column, Figure 3B in main text) and a small maze (right column, Figure 6 in main text). $[A]$ Average reward. $[B]$ Cumulative percentage of revisited state–action pairs over the course of training. The sharp transition at 50 000 steps corresponds 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

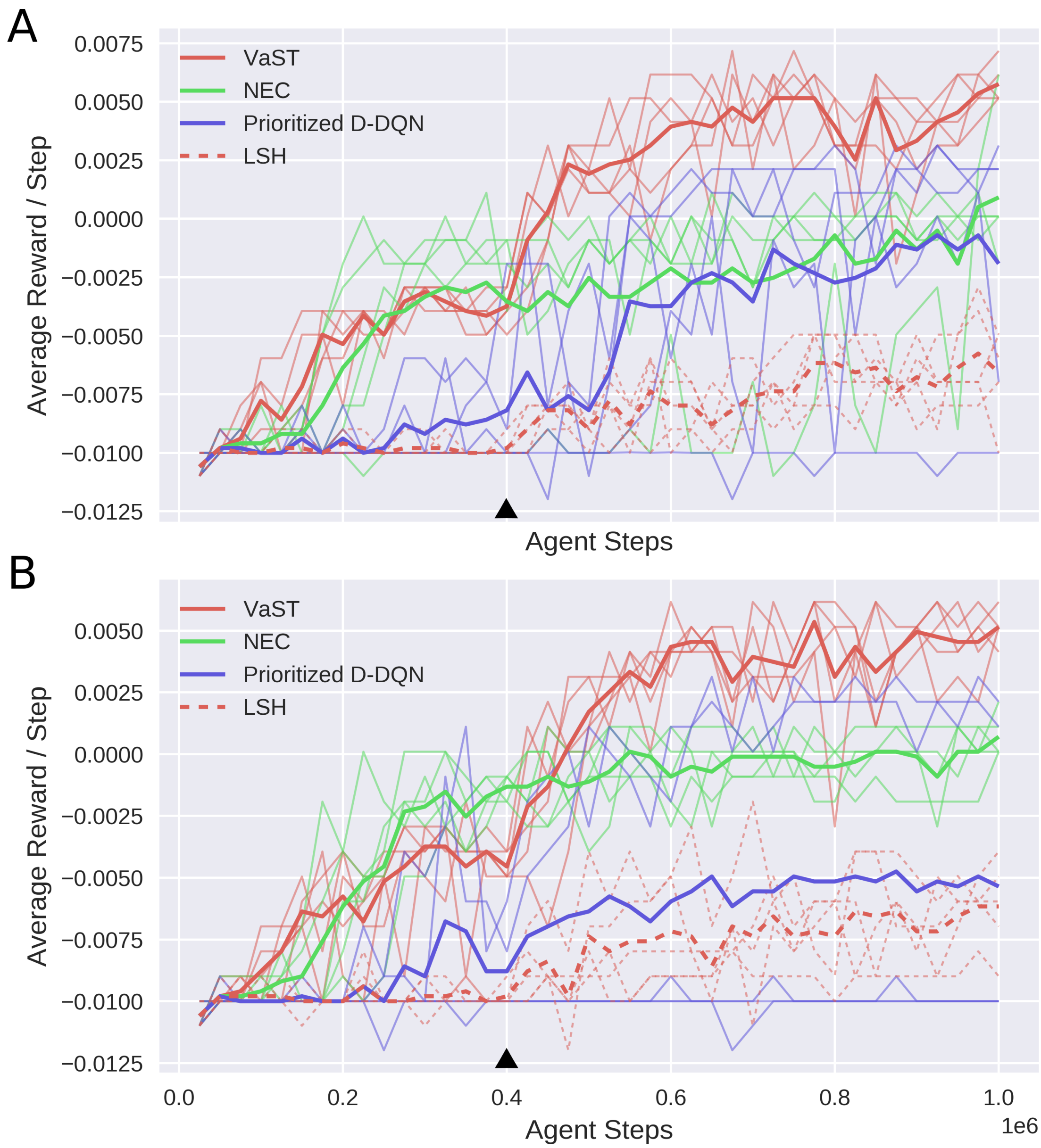

Figure 2: Performance comparison between models for $[A]$ rewarded forced runs (identical to Figure 5B in main text) and $[B]$ penalized forced runs. Black arrows indicate addition of teleporter and forced runs.

6 Effect of training on frame histories

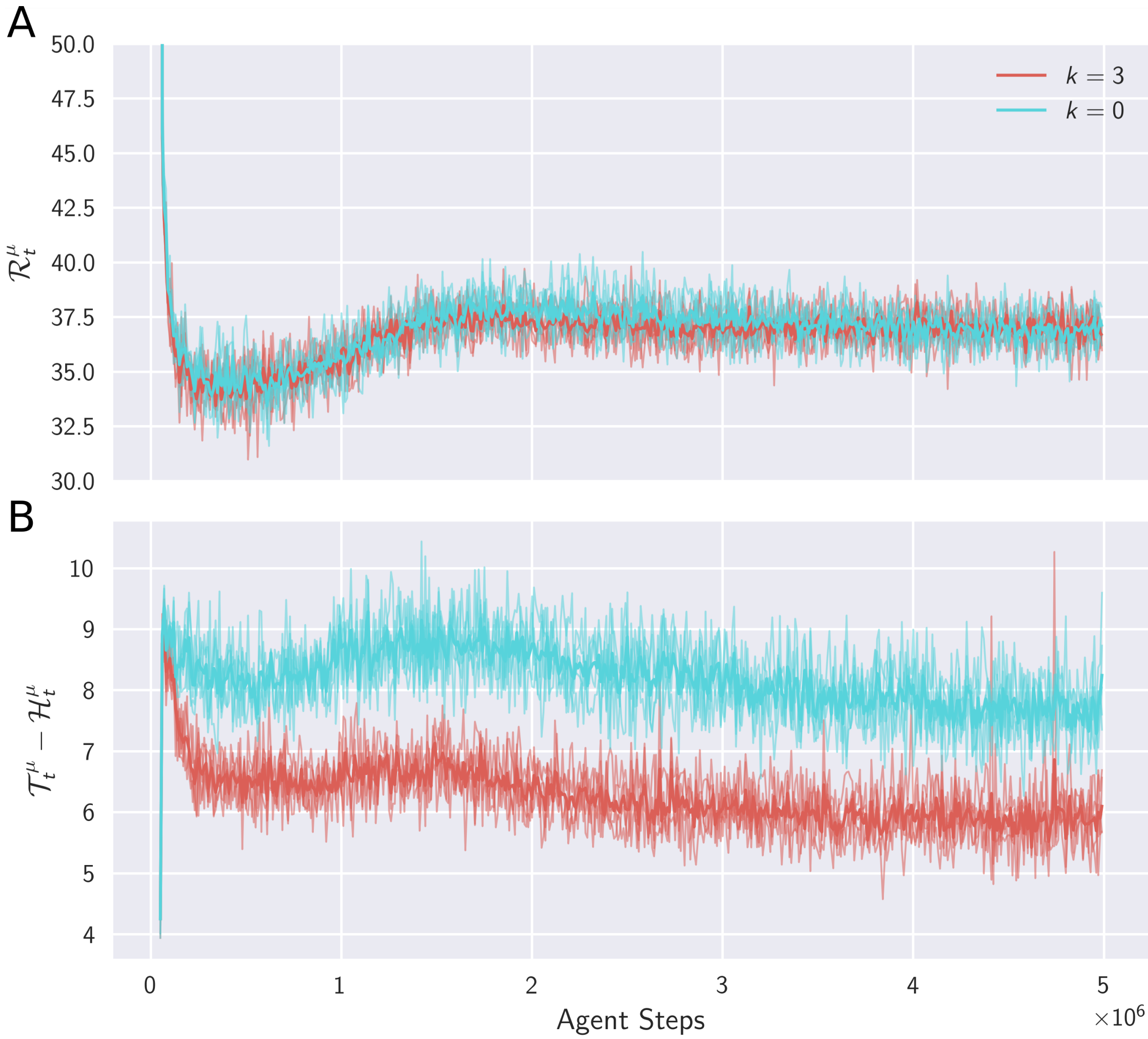

Figure 3: The free energy cost function over the course of training on Pong, broken into $[A]$ the reconstruction terms and $[B]$ the transition and entropy terms, conditioning on three additional past frames of observations ($k = 3$) and no additional frames ($k = 0$). Training with past frames as input 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 $\epsilon = 1\text{e}{-8}$, 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 $2 \times 1\text{e}{-4}$. For sweeping, we used $p_{min} = 5 \times 1\text{e}{-5}$. 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 $5 \times 1\text{e}{-5}$.

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 $1\text{e}{-4}$ 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 $p_{min} = 5 \times 1\text{e}{-5}$.

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.