1 VaST pseudocode

Algorithm 1 Variational State Tabulation.

Initialize replay memory $\mathcal{M}$ with capacity $N$
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 $\bar{s}_0 \leftarrow \text{arg max}_s q_\phi(s|o_0)$
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 $\epsilon$-greedy strategy based on $\tilde{Q}(s_{t-1}, a)$ from $\mathcal{B}$
9:   Receive $r_t, o_t$
10:  Process new state $\bar{s}_t \leftarrow \text{arg max}_s q_\phi(s|o_{t-k:t})$
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 $\mathcal{G} \leftarrow \emptyset$
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 $q_\phi(s_{j-1}|a_{j-k-1:j-1}), q_\phi(s_j|a_{j-k:j})$ with encoder
18:     Sample $\hat{s}_j \sim q_\phi$ with temperature $\lambda$
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 $\nabla_{\theta,\phi} F(\theta, \phi; o_{j-k-1:j})$ to $\mathcal{G}$
21:     for $i$ in $\{j-1, j\}$ do
22:       Process $\tilde{s}_{i}^{\text{new}} \leftarrow \text{arg max}_s q_\phi(s|o_{i-k:i})$
23:       Get $(\tilde{s}_{i-1}, a_i, r_i, \tilde{s}_{i}, a_{i+1}, r_{i+1}, \tilde{s}_{i+1})$ from $\mathcal{M}$
24:       if $\tilde{s}_{i} \neq \tilde{s}_{i}^{\text{new}}$ then
25:         Put $(\tilde{s}_{i-1}, a_i, r_i, \tilde{s}_{i}), (\tilde{s}_{i}, a_{i+1}, r_{i+1}, \tilde{s}_{i+1})$ on $\mathcal{Q}^-$
26:         Put $(\tilde{s}_{i-1}, a_i, r_i, \tilde{s}_{i}^{\text{new}}), (\tilde{s}_{i}^{\text{new}}, a_{i+1}, r_{i+1}, \tilde{s}_{i+1})$ on $\mathcal{Q}^+$
27:         Update $\tilde{s}_i \leftarrow \tilde{s}_{i}^{\text{new}}$ 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 $E[r | s, a]$ 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.

1. Initialize $V(s) = U(s) = 0$ for all $s$
2. Initialize $Q(s, a) = 0$ for all $s, a$
3. Initialize $N_{sa}, N_{s'}_{sa} = 0$ for all $s, a, s'$
4. Initialize priority queue $P$ with minimum priority cutoff $p_{\text{min}}$
5. Initialize add queue $Q^+$ and delete queue $Q^-$
6: while True do
   2: while $Q^+, Q^-$ empty do
      3: Remove top state $s'$ from $P$
      4: $\Delta U \leftarrow V(s') - U(s')$
      5: $U(s') \leftarrow V(s')$
      6: for all $(s, a)$ pairs with $N_{sa} > 0$ do
         7: $Q(s, a) \leftarrow Q(s, a) + \gamma N_{s'}_{sa} / N_{sa} \cdot \Delta U$
         8: $V(s) \leftarrow \max_a \{Q(s, a) | N_{sb} > 0\}$
         9: add/update $s$ in $P$ with priority $|U(s) - V(s)|$ if $|U(s) - V(s)| > p_{\text{min}}$
      10: end for
    11: end while
    12: for $(s, a, r, s')$ in $Q^+$ do
       13: $N_{sa} \leftarrow N_{sa} + 1; N_{s'}_{sa} \leftarrow N_{s'}_{sa} + 1$
       14: $Q(s, a) \leftarrow [Q(s, a) | N_{sa} - 1] + r + \gamma U(s') / N_{sa}$
       15: $V(s) \leftarrow \max_a \{Q(s, b) | N_{sb} > 0\}$
       16: add/update $s$ in $P$ with priority $|U(s) - V(s)|$ if $|U(s) - V(s)| > p_{\text{min}}$
    17: end for
    18: for $(s, a, r, s')$ in $Q^-$ do
       19: $N_{sa} \leftarrow N_{sa} - 1; N_{s'}_{sa} \leftarrow N_{s'}_{sa} - 1$
       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 $P$ with priority $|U(s) - V(s)|$ if $|U(s) - V(s)| > p_{\text{min}}$
    31: end for
32: end while
3 Details to \(Q\)-value estimation

Here, we simplify the discretized states \(\bar{s}\) to \(s\) for clarity. We denote \(S\) as the set of all states corresponding to \(d\)-length binary strings, \(\bar{Q}(s, a)\) 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 \(\bar{Q}(s_t, a)\) for a particular state–action pair, we first determine the Hamming distance \(m\) to the nearest neighbour(s) \(s \in S\) for which the action \(a\) has already been observed, i.e.

\[
m = \min_{s \in S} \{D(s_t, s)|N_{sa} > 0\},
\]

(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 \(S_{tm}\) of all \(m\)-nearest neighbours to state \(s_t\),

\[
S_{tm} = \{s \in S|D(s_t, s) = m\},
\]

(2)

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

\[
\bar{Q}(s_t, a) := \frac{\sum_{s \in S_{tm}} N_{sa} Q(s, a)}{\sum_{s \in S_{tm}} N_{sa}}.
\]

(3)

If \((s_t, a)\) has already been observed, then \(m = 0\), \(S_{tm} = \{s_t\}\) and \(\bar{Q}(s_t, a) = Q(s_t, a)\). If \(m = 1\), \(\bar{Q}(s_t, a)\) 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.

\(\bar{Q}(s_t, a)\) 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 \(E_{sa} = \{(\tau, \mu)|s_\tau^\mu = s, a_\tau^\mu = a\}\) that contains all the time indices \(\tau\) for all episodes \(\mu\) 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 \(N_{sa} = |E_{sa}|\), i.e. there are \(N_{sa}\) elements in the set \(E_{sa}\). With this and the update mechanism of prioritized sweeping (Algorithm 2) we can write

\[
Q(s, a) = \frac{1}{N_{sa}} \sum_{\tau, \mu \in E_{sa}} r_\tau^\mu + \frac{1}{N_{sa}} \sum_{\tau, \mu \in E_{sa}} V(s_{\tau+1}^\mu),
\]

(4)

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

\[
\tilde{Q}(s_t, a) = \frac{\sum_{s \in S_{tm}} \sum_{\tau, \mu \in E_{sa}} r_\tau^\mu + \gamma \sum_{\tau, \mu \in E_{sa}} V(s_{\tau+1}^\mu)}{\sum_{s \in S_{tm}} N_{sa}}.
\]

(5)

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

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

(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.
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 \cite{Pritzel2017, Schaul2015, Mnih2015}. In all mazes except the smaller Plus–Maze, the agents explored randomly for 50 000 steps to initialize the replay memory before training; \( \epsilon \) 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 \( \epsilon \) 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 \cite{Mnih2015}, with ReLU activations. For all three networks, we used the Adam optimizer \cite{Kingma2014} 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 \( N = 500 000 \) 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 \( p_{\text{min}} = 5 \times 1e^{-5} \). For the Concrete relaxation, we used the temperatures suggested by \cite{Maddison2016}: \( \lambda_1 = 2/3 \) 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 \( 4 \times 4 \) 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 \( p_{\text{min}} = 5 \times 1e^{-5} \).

7.2 Atari: Pong

We used a latent dimensionality of \( d = 64 \), a replay memory size of \( N = 1 000 000 \) transitions, and annealed \( \epsilon \) over 1 000 000 steps. All other hyperparameters were the same as for navigation.
References


