A. Experiments

A.1. Implementation Details

In our implementation, the transition and proposal distributions $p_\theta(z_t|h_{t-1}, a_{t-1})$ and $q_\theta(z_t|h_{t-1}, a_{t-1}, o_t)$ are multivariate normal distributions over z_t whose mean and diagonal variance are determined by neural networks. For image data, the decoder $p_\theta(o_t|z_t, a_{t-1})$ is a multivariate independent Bernoulli distribution whose parameters are again determined by a neural network. For real-valued vectors we use a normal distribution.

When several inputs are passed to a neural network, they are concatenated to one vector. ReLUs are used as nonlinearities between all layers. Hidden layers are, if not otherwise stated, all of the same dimension as h. Batch normalization was used between layers for experiments on Atari but not on Mountain Hike as they significantly hurt performance. All recurrent neural networks (RNNs) are GRUs.

Encoding functions φ^o, φ^a and φ^z are used to encode single observations, actions and latent states z before they are passed into other networks.

To encode visual observations, we use the the same convolutional network as proposed by Mnih et al. (2015), but with only 32 instead of 64 channels in the final layer. The transposed convolutional network of the decoder has the reversed structure. The decoder is preceeded by an additional fully connected layer which outputs the required dimension (1568 for Atari's 84×84 observations).

For observations in $\mathbb{R}^2$ we used two fully connected layers of size 64 as encoder. As decoder we used the same structure as for $p_\theta(z|\dots)$ and $q_\phi(z|\dots)$ which are all three normal distributions: One joint fully connected layer and two separated fully connected heads, one for the mean, one for the variance. The output of the variance layer is passed through a softplus layer to force positivity.

Actions are encoded using one fully connected layer of size 128 for Atari and size 64 for Mountain Hike. Lastly, z is encoded before being passed into networks by one fully connected layer of the same size as h.

The policy is one fully connected layer whose size is determined by the actions space, i.e. up to 18 outputs with softmax for Atari and only 2 outputs for the learned mean for Mountain Hike, together with a learned variance. The value function is one fully connected layer of size 1.

A2C used $n_e = 16$ parallel environments and $n_s = 5$-step learning for a total batch size of 80. Hyperparameters were tuned on *Chopper Command*. The learning rate of both deep variational reinforcement learning (DVRL) and RNN was independently tuned on the set of values $\{3 \times 10^{-5}, 1 \times 10^{-4}, 2 \times 10^{-4}, 3 \times 10^{-4}, 6 \times 10^{-4}, 9 \times 10^{-4}\}$ with 2×10^{-4} being chosen for DVRL on Atari and 1×10^{-4} for DVRL on MountainHike and RNN on both environments. Without further tuning, we set $\lambda^H = 0.01$ and $\lambda^V = 0.5$ as is commonly used.

As optimizer we use RMSProp with $\alpha = 0.99$. We clip gradients at a value of 0.5. The discount factor of the control problem is set to $\gamma = 0.99$ and lastly, we use 'orthogonal' initialization for the network weights.

The source code will be release in the future.

A.2. Additional Experiments and Visualisations

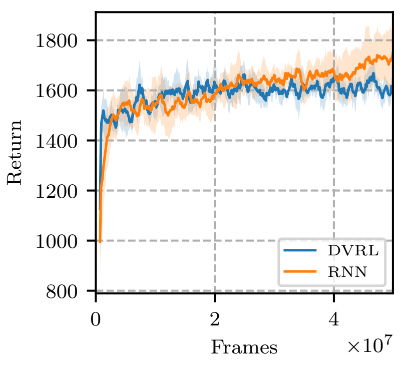
Table 1 shows the results on *deterministic* and flickering Atari, averaged over 5 random seeds. The values for deep recurrent Q-network (DRQN) and action-specific deep recurrent Q-network (ADRQN) are taken from the respective papers. Note that DRQN and ADRQN rely on Q-learning instead of A2C, so the results are not directly comparable.

Figure 1 and 2 show individual learning curves for all 10 Atari games, either for the deterministic or the stochastic version of the games.

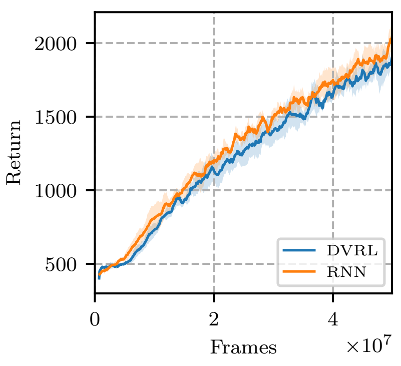
A.3. Computational Speed

The approximate training speed in frames per second (FPS) is on one GPU on a dgx1 for Atari:

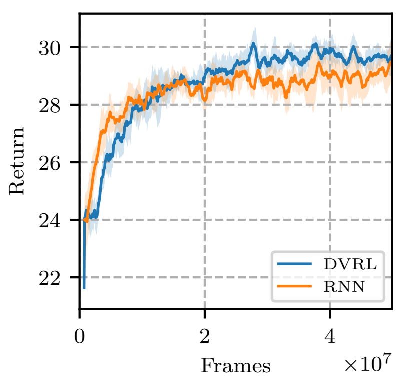
- RNN: 124k FPS
- DVRL (1 Particle): 64k FPS



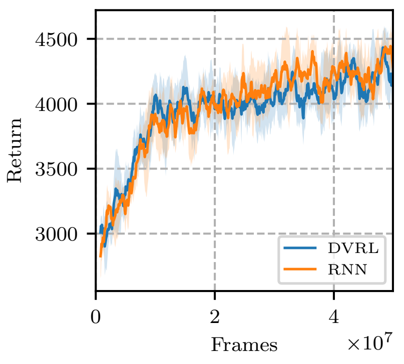
(a) Asteroids



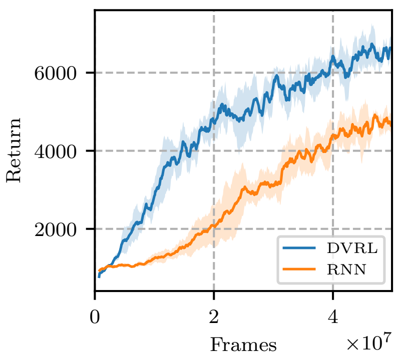
(b) Beam Rider



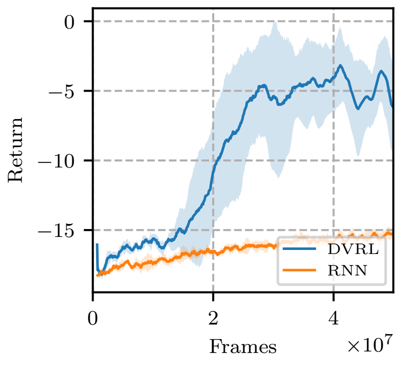
(c) Bowling



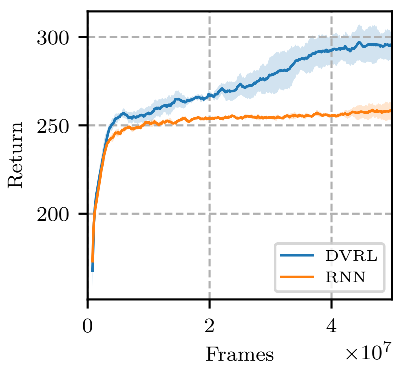
(d) Centipede



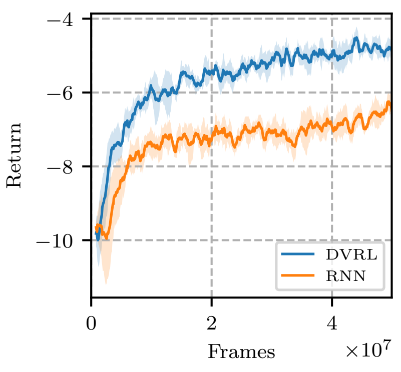
(e) Chopper Command



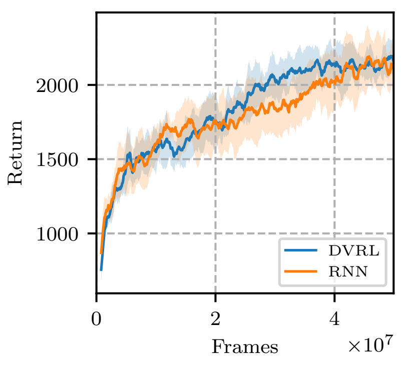
(f) Double Dunk



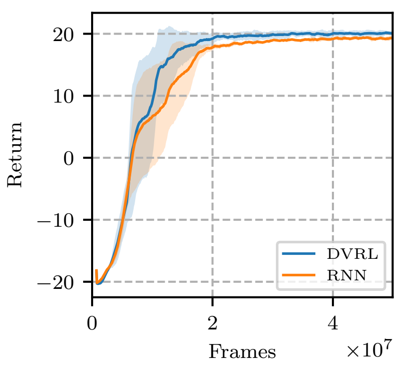
(g) Frostbite



(h) Ice Hockey

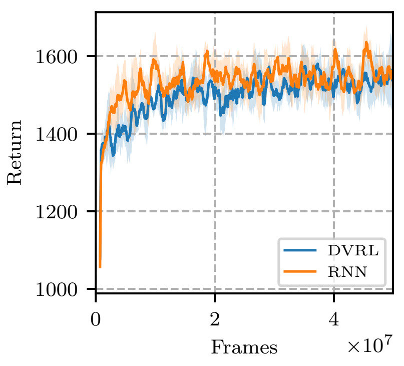


(i) Ms. Pacman

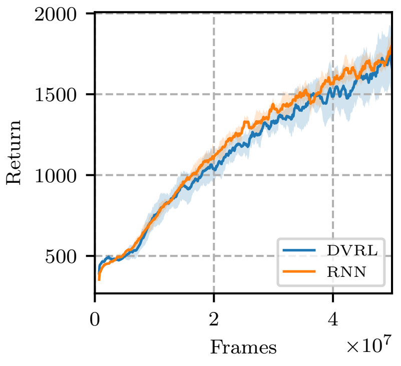


(j) Pong

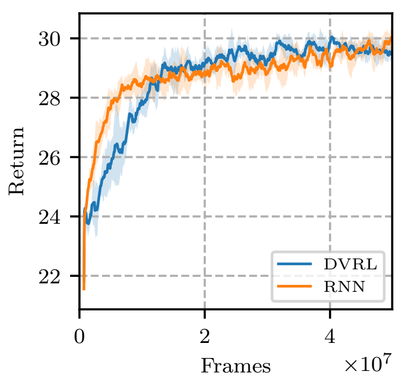
Figure 1: Training curves on the full set of evaluated Atari games, in the case of flickering and *deterministic* environments.



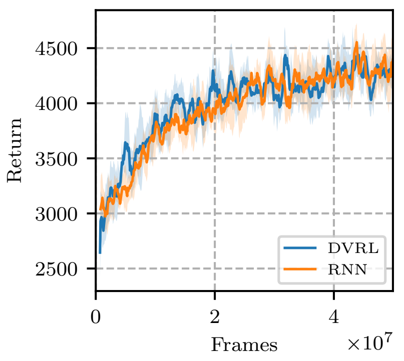
(a) Asteroids



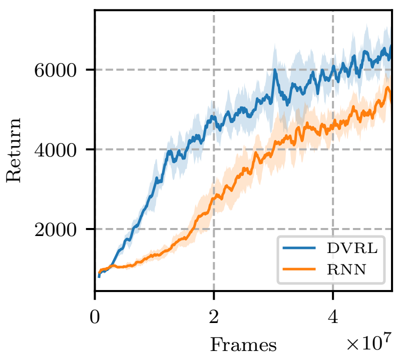
(b) Beam Rider



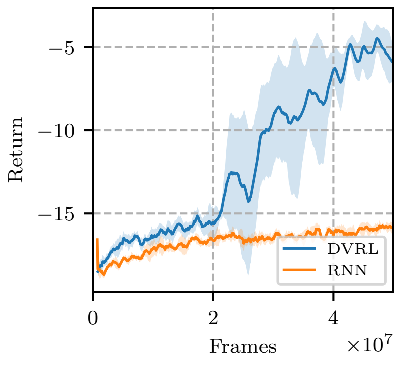
(c) Bowling



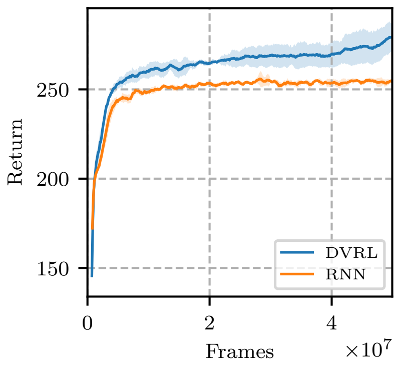
(d) Centipede



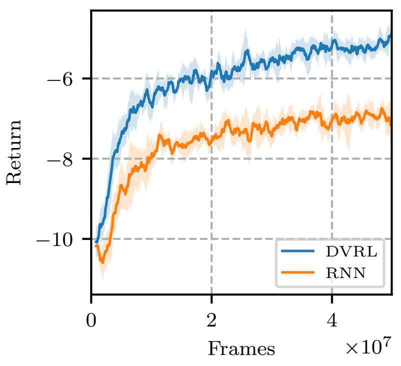
(e) Chopper Command



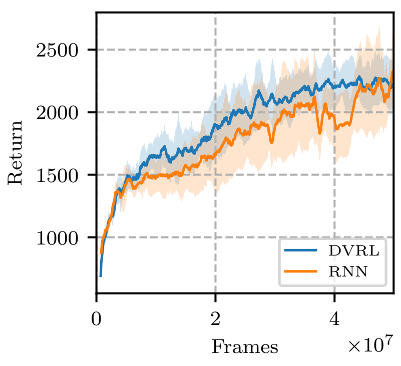
(f) Double Dunk



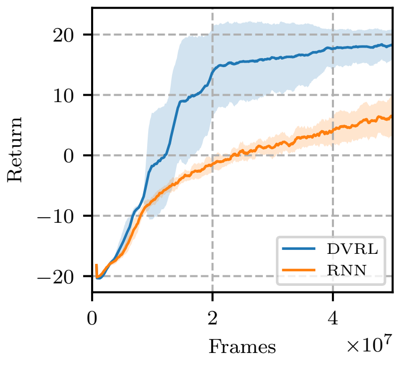
(g) Frostbite



(h) Ice Hockey



(i) Ms. Pacman



(j) Pong

Figure 2: Training curves on the full set of evaluated Atari games, in the case of flickering and *stochastic* environments.

Table 1: Final results on deterministic and flickering Atari environments, averaged over 5 random seeds. Bold numbers indicate statistical significance at the 5% level when comparing DVRL and RNN. The values for DRQN and ADRQN are taken from the respective papers.

Env	DVRL$(\pm std)$	RNN	DRQN	ADRQN
Pong	$\mathbf{20.07}(\pm 0.39)$	$19.3(\pm 0.26)$	$12.1(\pm 2.2)$	$7(\pm 4.6)$
Chopper	$\mathbf{6619}(\pm 532)$	$4619(\pm 306)$	$1330(\pm 294)$	$1608(\pm 707)$
MsPacman	$2156(\pm 127)$	$2113(\pm 135)$	$1739(\pm 942)$	
Centipede	$4171(\pm 127)$	$4283(\pm 187)$	$4319(\pm 4378)$	
BeamRider	$1901(\pm 67)$	$\mathbf{2041}(\pm 81)$	$618(\pm 115)$	
Frostbite	$\mathbf{296}(\pm 8.2)$	$259(\pm 5.7)$	$414(\pm 494)$	$2002(\pm 734)$
Bowling	$29.74(\pm 0.49)$	$29.38(\pm 0.52)$	$65(\pm 13)$	
IceHockey	$\mathbf{-4.87}(\pm 0.24)$	$-6.49(\pm 0.27)$	$-5.4(\pm 2.7)$	
DDunk	$\mathbf{-6.08}(\pm 3.08)$	$-15.25(\pm 0.51)$	$-14(\pm 2.5)$	$-13(\pm 3.6)$
Asteroids	$1610(\pm 63)$	$\mathbf{1750}(\pm 97)$	$1032(\pm 410)$	$1040(\pm 431)$

- DVRL (10 Particles): 48k FPS
- DVRL (30 Particle): 32k FPS

A.4. Model Predictions

In Figure 3 we show reconstructed and predicted images from the DVRL model for several Atari games. The current observation is in the leftmost column. The second column ('dt0') shows the reconstruction after encoding and decoding the current observation. For the further columns, we make use of the learned generative model to predict future observations. For simplicity we repeat the last action. Columns 2 to 7 show predicted observations for $dt \in \{1, 2, 3, 10, 30\}$ unrolled timesteps. The model was trained as explained in the main paper. The reconstructed and predicted images are a weighted average over all 16 particles.

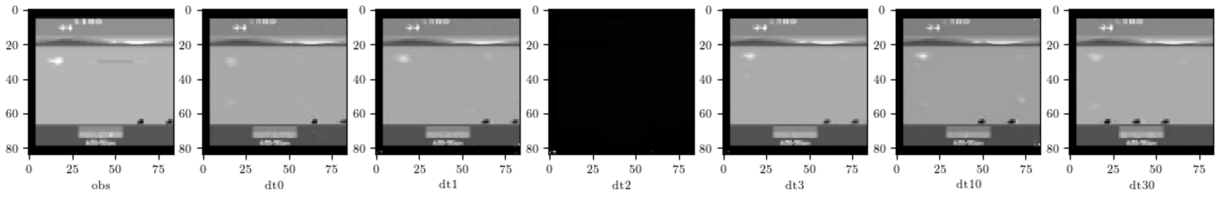
Note that the model is able to correctly predict features of future observations, for example the movement of the cars in ChopperCommand, the (approximate) ball position in Pong or the missing pins in Bowling. Furthermore, it is able to do so, even if the current observation is blank like in Bowling. The model has also correctly learned to randomly predict blank observations.

It can remember feature of the current state fairly well, like the positions of barriers (white dots) in Centipede. On the other hand, it clearly struggles with the amount of information present in MsPacman like the positions of all previously eaten fruits or the location of the ghosts.

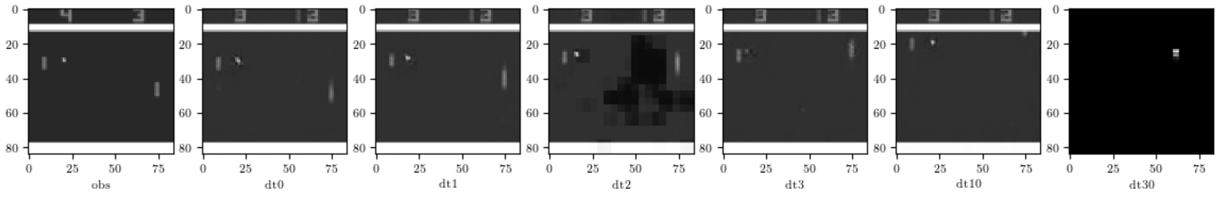
B. Algorithms

Algorithm 1 details the recurrent (belief) state computation (i.e. history encoder) for DVRL. Algorithm 2 details the recurrent state computation for RNN. Algorithm 3 describes the overall training algorithm that either uses one or the other to aggregate the history. Despite looking complicated, it is just a very detailed implementation of n-step A2C with the additional changes: Inclusion of $\mathcal{L}^{\text{ELBO}}$ and inclusing of the option to not delete the computation graph to allow longer backprop in n-step A2C.

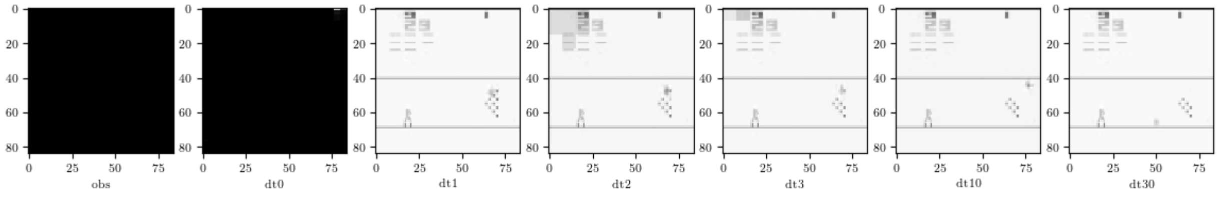
Results for also using the reconstruction loss $\mathcal{L}^{\text{ENC}}$ for the RNN based encoder aren't shown in the paper as they reliably performed worse than RNN without reconstruction loss.



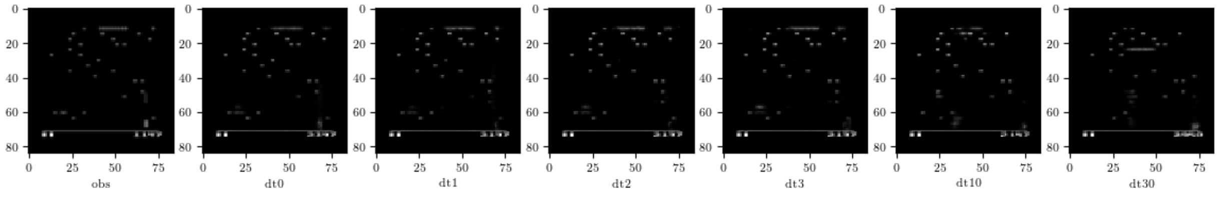
(a) ChopperCommand



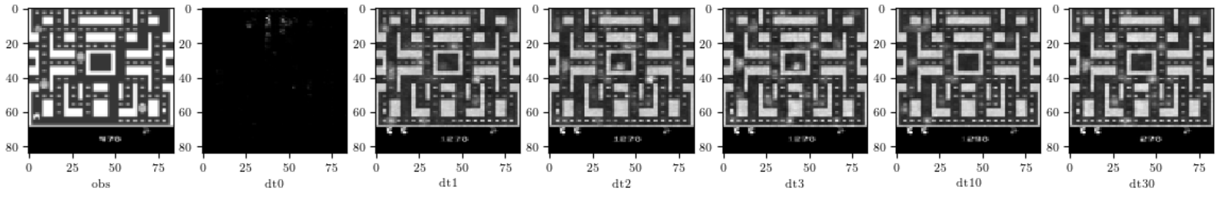
(b) Pong



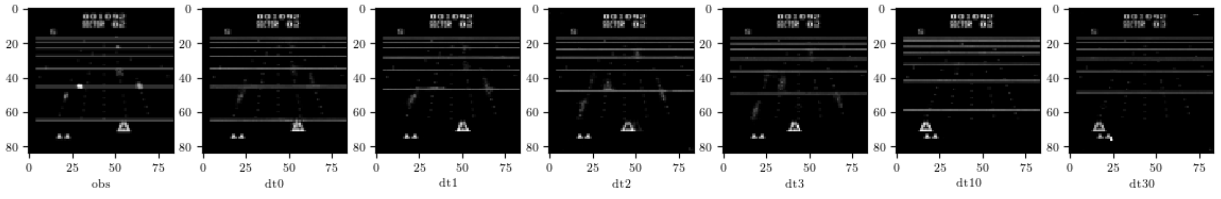
(c) Bowling



(d) Centipede



(e) MsPacman



(f) BeamRider

Figure 3: Reconstructions and predictions using the learned generative model for several Atari games. First column: Current obseration (potentially blank). Second column: Encoded and decoded reconstruction of the current observation. Columns 3 to 7: Predicted observations using the learned generative model for timesteps $dt \in \{0, 1, 2, 3, 10, 30\}$ into the future.

Algorithm 1 DVRL encoder

```
**Input:** Previous state $\hat{b}_{t-1}$, observation $o_t$, action $a_{t-1}$
Unpack $w_{t-1}^{1:K}, z_{t-1}^{1:K}, h_{t-1}^{1:K}, \hat{h}_{t-1} \leftarrow \hat{b}_{t-1}$
$x^o \leftarrow \varphi_\theta^o(o_t)$
$x^a \leftarrow \varphi_\theta^a(a_{t-1})$
**for** $k = 1$ **to** $K$ **do**
    Sample $h_{t-1}^k \sim h_{t-1}^{1:K}$ based on weights
    Sample $z_t^k \sim q_\theta(z_t^k | h_{t-1}^k, x^o, x^a)$
    $x^z \leftarrow \varphi_\theta^z(z_t)$
    $w_j^k \leftarrow p_\theta(z_t^k | h_{t-1}^k, x^a) p_\theta(o_t | h_t^k, x^z, x^a) / q_\theta(z_t^k | h_{t-1}^k, x^o, x^a)$
    $h_t^k \leftarrow \text{GRU}(h_{t-1}^k, x^z, x^o, x^a)$
**end for**
$\mathcal{L}_t^{\text{ELBO}} \leftarrow -\log \sum_k w_t^k - \log(K)$
$\hat{h}_t \leftarrow \text{GRU}(\text{Concat}(w_t^k, x^z, h_t^k)_{k=1}^K \text{passed sequentially})$
Pack $\hat{b}_t \leftarrow w_t^{1:K}, z_t^{1:K}, h_t^{1:K}, \hat{h}_t$
{When $V$ or $\pi$ is conditioned on $\hat{b}_t$, the summary $\hat{h}_t$ is used.}
**Output:** $\hat{b}_t, \mathcal{L}_t^{\text{ELBO}}$
```

Algorithm 2 RNN encoder

```
**Input:** Previous state $h_{j-1}$, observation $o_j$, action $a_{j-1}$
$x^o \leftarrow \varphi_\theta^o(o_t)$
$x^a \leftarrow \varphi_\theta^a(a_{t-1})$
$\hat{b}_j \leftarrow \text{GRU}_\theta(\hat{b}_{j-1}, x^o, x^a)$
$\mathcal{L}_j^{\text{ENC}} \leftarrow -\log p_\theta(o_j | \hat{b}_{j-1})$
**Output:** $h_j, \mathcal{L}_j^{\text{ENC}}$
```

Algorithm 3 Training Algorithm

```
Input: Environment Env, Encoder $\text{Enc}_{\theta,\phi}$ (either RNN or DVRL)
Initialize observation $o_1$ from $\text{Env}$.
Initialize encoder latent state $s_0 \leftarrow s_0^{init}$ as either $h_0$ (for RNN) or $\hat{b}_{0,\theta}$ (for DVRL)
Initialize action $a_0 = 0$ to no-op
Set $s'_0, a'_0 \leftarrow s_0, a_0$.
{The distinction between $s'_t$ and $s_t$ is necessary when the environment resets at time $t$.}
repeat
    $\mathcal{L}_j^{Enc}, a_j, a'_j, s_j, s'_j, o_{j+1}, r_{j+1}, done_{j+1} \leftarrow NULL \quad j = 1 \ldots n$
    {Run $n$ steps forward:}
    for $j = 1$ to $n$ do
        $s_j, \mathcal{L}_j^{\text{ELBO}} \leftarrow \text{Enc}_{\theta,\phi}(o_j, a'_{j-1}, s'_{j-1})$
        Sample $a_j \sim \pi_\rho(a_j|s_j)$
        $o_{j+1}, r_{j+1}, done_{j+1} \leftarrow \text{Env}(a_j)$
        if $done_{j+1}$ then
            $s'_j, a'_j \leftarrow s_0^{init}, 0$
            {$s_j$ is still available to compute $V_\eta(s_j)$}
            $o_{j+1} \leftarrow$ Reset $\text{Env}()$
        else
            $s'_j, a'_j \leftarrow s_j, a_j$
        end if
    end for
    {Compute targets}
    $s_{n+1} \leftarrow \text{Enc}_{\theta,\phi}(o_{n+1}, a'_n, s'_n)$
    $Q_{n+1}^{target} \leftarrow V_\eta(s_{n+1}).detach()$
    for $j = n$ to $1$ do
        $Q_j^{target} \leftarrow \gamma \cdot Q_{j+1}^{target}$
        if $done_{j+1}$ then
            $Q_j^{target} \leftarrow 0$
        end if
        $Q_j^{target} \leftarrow Q_j^{target} + r_{j+1}$
    end for
    {Compute losses}
    for $j = n$ to $1$ do
        $\mathcal{L}_j^V \leftarrow (Q_j^{target} - V_\eta(s_j))^2$
        $\mathcal{L}_j^A \leftarrow -\log \pi_\rho(a_j|s_j)(Q_j^{target} - V_\eta(s_j))$
        $\mathcal{L}_j^H \leftarrow -\text{Entropy}(\pi_\rho(\cdot|s_j))$
    end for
    $\mathcal{J} \leftarrow \sum_j(\lambda^V \mathcal{L}_j^V + \mathcal{L}_j^A + \lambda^H \mathcal{L}_j^H + \lambda^E \mathcal{L}_j^{\text{ELBO}})$
    TakeGradientStep($\nabla\mathcal{J}$)
    Delete or save computation graph of $s_n$ to determine backpropagation length
    $a'_0, s'_0 \leftarrow a_n, s_n$
    $o_1 \leftarrow o_{n+1}$
until converged
```