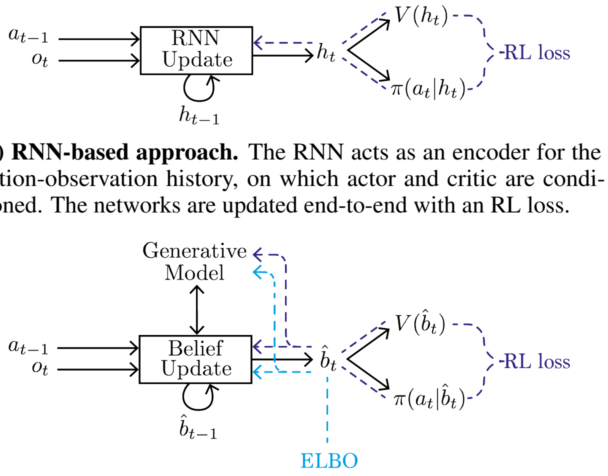
Deep Variational Reinforcement Learning for POMDPs

Maximilian Igl [1] Luisa Zintgraf [1] Tuan Anh Le [1] Frank Wood [2] Shimon Whiteson [1]

Abstract

Many real-world sequential decision making problems are partially observable by nature, and the environment model is typically unknown. Consequently, there is great need for reinforcement learning methods that can tackle such problems given only a stream of incomplete and noisy observations. In this paper, we propose deep variational reinforcement learning (DVRL), which introduces an inductive bias that allows an agent to learn a generative model of the environment and perform inference in that model to effectively aggregate the available information. We develop an n-step approximation to the evidence lower bound (ELBO), allowing the model to be trained jointly with the policy. This ensures that the latent state representation is suitable for the control task. In experiments on Mountain Hike and flickering Atari we show that our method outperforms previous approaches relying on recurrent neural networks to encode the past.



(a) RNN-based approach. The RNN acts as an encoder for the action-observation history, on which actor and critic are conditioned. The networks are updated end-to-end with an RL loss.

(b) DVRL. The agent learns a generative model which is used to update a belief distribution. Actor and critic now condition on the belief. The generative model is learned to optimise both the ELBO *and* the RL loss.

Figure 1: Comparison of RNN and DVRL encoders.

1. Introduction

Most deep reinforcement learning (RL) methods assume that the state of the environment is fully observable at every time step. However, this assumption often does not hold in reality, as occlusions and noisy sensors may limit the agent's perceptual abilities. Such problems can be formalised as partially observable Markov decision processes (POMDPs) (Astrom, 1965; Kaelbling et al., 1998). Because we usually do not have access to the true generative model of our environment, there is a need for reinforcement learning methods that can tackle POMDPs when only a stream of observations is given, without any prior knowledge of the latent state space or the transition and observation functions.

POMDPs are notoriously hard to solve: since the current observation does in general not carry all relevant information for choosing an action, information must be aggregated over time and in general the entire history must be taken into account.

This history can be encoded either by remembering features of the past (McCallum, 1993) or by performing inference to determine the distribution over possible latent states (Kaelbling et al., 1998). However, the computation of this *belief state* requires knowledge of the model.

Most previous work in deep learning relies on training a recurrent neural network (RNN) to summarize the past. Examples are the deep recurrent Q-network (DRQN) (Hausknecht & Stone, 2015) and the action-specific deep recurrent Q-network (ADRQN) (Zhu et al., 2017). Because these approaches are completely model-free, they place a heavy burden on the RNN. Since performing inference implicitly requires a known or learned model, they are likely to summarise the history either by only remembering features of the past or by computing simple heuristics instead of actual belief states. This is often suboptimal in complex tasks. Generalisation is also often easier over beliefs than over

[1]University of Oxford, United Kingdom [2]University of British Columbia, Canada. Correspondence to: Maximilian Igl <maximilian.igl@eng.ox.ac.uk>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

trajectories since distinct histories can lead to similar or identical beliefs.

The premise of this work is that deep policy learning for POMDPs can be improved by taking less of a black box approach than DRQN and ADRQN. While we do not want to assume prior knowledge of the transition and observation functions or the latent state representation, we want to allow the agent to learn models of them and infer the belief state using this learned model.

To this end, we propose DVRL, which implements this approach by providing a helpful inductive bias to the agent. In particular, we develop an algorithm that can learn an internal generative model and use it to perform approximate inference to update the belief state. Crucially, the generative model is not only learned based on an ELBO objective, but also by how well it enables maximisation of the expected return. This ensures that, unlike in an unsupervised application of variational autoencoders (VAEs), the latent state representation and the inference performed on it are suitable for the ultimate control task. Specifically, we develop an approximation to the ELBO based on autoencoding sequential Monte Carlo (AESMC) (Le et al., 2018), allowing joint optimisation with the n-step policy gradient update. Uncertainty in the belief state is captured by a particle ensemble. A high-level overview of our approach in comparison to previous RNN-based methods is shown in Figure 1.

We evaluate our approach on Mountain Hike and several *flickering* Atari games. On Mountain Hike, a low dimensional, continuous environment, we can show that DVRL is better than an RNN based approach at inferring the required information from past observations for optimal action selection in a simple setting. Our results on flickering Atari show that this advantage extends to complex environments with high dimensional observation spaces. Here, partial observability is introduced by (1) using only a single frame as input at each time step and (2) returning a blank screen instead of the true frame with probability 0.5.

2. Background

In this section, we formalise POMDPs and provide background on recent advances in VAEs that we use. Lastly, we describe the policy gradient loss based on n-step learning and A2C.

2.1. Partially Observable Markov Decision Processes

A partially observable Markov decision process (POMDP) is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, F, U, R, b_0)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, and $\mathcal{O}$ the observation space. We denote as $s_t \in \mathcal{S}$ the latent state at time t, and the distribution over initial states s_0 as b_0, the initial belief state. When an action $a_t \in \mathcal{A}$ is executed, the state changes according to the transition distribution, $s_{t+1} \sim F(s_{t+1}|s_t, a_t)$. Subsequently, the agent receives a noisy or partially occluded observation $o_{t+1} \in \mathcal{O}$ according to the distribution $o_{t+1} \sim U(o_{t+1}|s_{t+1}, a_t)$, and a reward $r_{t+1} \in \mathbb{R}$ according to the distribution $r_{t+1} \sim R(r_{t+1}|s_{t+1}, a_t)$.

An agent acts according to its policy $\pi(a_t|o_{\leq t}, a_{<t})$ which returns the probability of taking action a_t at time t, and where $o_{\leq t} = (o_1, \dots, o_t)$ and $a_{<t} = (a_0, \dots, a_{t-1})$ are the observation and action histories, respectively. The agent's goal is to learn a policy π that maximises the expected future return

$$J = \mathbb{E}_{p(\tau)}\left[\sum_{t=1}^{T} \gamma^{t-1} r_t\right], \tag{1}$$

over trajectories $\tau = (s_0, a_0, \dots, a_{T-1}, s_T)$ induced by its policy[1], where $0 \leq \gamma < 1$ is the discount factor. We follow the convention of setting a_0 to *no-op* (Zhu et al., 2017).

In general, a POMDP agent must condition its actions on the entire history $(o_{\leq t}, a_{<t}) \in \mathcal{H}_t$. The exponential growth in t of $\mathcal{H}_t$ can be addressed, e.g., with suffix trees (McCallum & Ballard, 1996; Shani et al., 2005; Bellemare et al., 2014; Bellemare, 2015; Messias & Whiteson, 2017). However, those approaches suffer from large memory requirements and are only suitable for small discrete observation spaces.

Alternatively, it is possible to infer the filtering distribution $p(s_t|o_{\leq t}, a_{<t}) =: b_t$, called the *belief state*. This is a sufficient statistic of the history that can be used as input to an optimal policy $\pi^\star(a_t|b_t)$. The belief space does not grow exponentially, but the belief update step requires knowledge of the model:

$$b_{t+1} = \frac{\int b_t U(o_{t+1}|s_{t+1}, a_t) F(s_{t+1}|s_t, a_t) \mathrm{d}s_t}{\int\int b_t U(o_{t+1}|s_{t+1}, a_t) F(s_{t+1}|s_t, a_t)\, \mathrm{d}s_t\, \mathrm{d}s_{t+1}}. \tag{2}$$

2.2. Variational Autoencoder

We define a family of priors $p_\theta(s)$ over some latent state s and decoders $p_\theta(o|s)$ over observations o, both parameterised by θ. A variational autoencoder (VAE) learns θ by maximising the sum of log marginal likelihood terms $\sum_{n=1}^{N} \log p_\theta(o^{(n)})$ for a dataset $(o^{(n)})_{n=1}^{N}$ where $p_\theta(o) = \int p_\theta(o|s) p_\theta(s)\, \mathrm{d}s$ (Rezende et al., 2014; Kingma & Welling, 2014)) . Since evaluating the log marginal likelihood is intractable, the VAE instead maximises a sum of ELBOs where each individual ELBO term is a lower bound on the log marginal likelihood,

$$\mathrm{ELBO}(\theta, \phi, o) = \mathbb{E}_{q_\phi(s|o)}\left[\log \frac{p_\theta(o|s) p_\theta(s)}{q_\phi(s|o)}\right], \tag{3}$$

[1] The trajectory length T is stochastic and depends on the time at which the agent-environment interaction ends.

for a family of encoders $q_\phi(s|o)$ parameterised by ϕ. This objective also forces $q_\phi(s|o)$ to approximate the posterior $p_\theta(s|o)$ under the learned model. Gradients of (3) are estimated by Monte Carlo sampling with the reparameterisation trick (Kingma & Welling, 2014; Rezende et al., 2014).

2.3. VAE for Time Series

For sequential data, we assume that a series of latent states $s_{\leq T}$ gives rise to a series of observations $o_{\leq T}$. We consider a family of generative models parameterised by θ that consists of the initial distribution $p_\theta(s_0)$, transition distribution $p_\theta(s_t|s_{t-1})$ and observation distribution $p_\theta(o_t|s_t)$. Given a family of encoder distributions $q_\phi(s_t|s_{t-1}, o_t)$, we can also estimate the gradient of the ELBO term in the same manner as in (3), noting that:

$$p_\theta(s_{\leq T}, o_{\leq T}) = p_\theta(s_0) \prod_{t=1}^{T} p_\theta(s_t|s_{t-1}) p_\theta(o_t|s_t), \quad (4)$$

$$q_\phi(s_{\leq T}|o_{\leq T}) = p_\theta(s_0) \prod_{t=1}^{T} q_\phi(s_t|s_{t-1}, o_t), \quad (5)$$

where we slightly abuse notation for q_ϕ by ignoring the fact that we sample from the model $p_\theta(s_0)$ for $t = 0$. Le et al. (2018), Maddison et al. (2017) and Naesseth et al. (2018) introduce a new ELBO objective based on sequential Monte Carlo (SMC) (Doucet & Johansen, 2009) that allows faster learning in time series:

$$\text{ELBO}_{\text{SMC}}(\theta, \phi, o_{\leq T}) = \mathbb{E}\left[\sum_{t=1}^{T} \log\left(\frac{1}{K}\sum_{k=1}^{K} w_t^k\right)\right], \quad (6)$$

where K is the number of particles and w_t^k is the weight of particle k at time t. Each particle is a tuple containing a weight w_t^k and a value s_t^k which is obtained as follows. Let s_0^k be samples from $p_\theta(s_0)$ for $k = 1, \ldots, K$. For $t = 1, \ldots, T$, the weights w_t^k are obtained by resampling the particle set $(s_{t-1}^k)_{k=1}^K$ proportionally to the previous weights and computing

$$w_t^k = \frac{p_\theta(s_t^k|s_{t-1}^{u_{t-1}^k}) p_\theta(o_t|s_t^k)}{q_\phi(s_t^k|s_{t-1}^{u_{t-1}^k}, o_t)}, \quad (7)$$

where s_t^k corresponds to a value sampled from $q_\phi(\cdot|s_{t-1}^{u_{t-1}^k}, o_t)$ and $s_{t-1}^{u_{t-1}^k}$ corresponds to the resampled particle with the ancestor index $u_0^k = k$ and $u_{t-1}^k \sim \text{Discrete}((w_{t-1}^k / \sum_{j=1}^K w_{t-1}^j)_{k=1}^K)$ for $t = 2, \ldots, T$.

2.4. A2C

One way to learn the parameters ρ of an agent's policy $\pi_\rho(a_t|s_t)$ is to use n-step learning with A2C (Dhariwal et al., 2017; Wu et al., 2017), the synchronous simplification of asynchronous advantage actor-critic (A3C) (Mnih et al., 2016). An actor-critic approach can cope with continuous actions and avoids the need to draw state-action sequences from a replay buffer. The method proposed in this paper is however equally applicable to other deep RL algorithms.

For n-step learning, starting at time t, the current policy performs n_s consecutive steps in n_e parallel environments. The gradient update is based on this mini-batch of size $n_e \times n_s$. The target for the value-function $V_\eta(s_{t+i}), i = 0, \ldots, n_s - 1$, parameterised by η, is the appropriately discounted sum of on-policy rewards up until time $t + n_s$ and the off-policy bootstrapped value $V_\eta^-(s_{t+n_s})$. The minus sign denotes that no gradients are propagated through this value. Defining the advantage function as

$$\begin{aligned} A_\eta^{t,i}(s_{t+i}, a_{t+i}) := &\left(\sum_{j=0}^{n_s-i-1} \gamma^j r_{t+i+j}\right) \\ &+ \gamma^{n_s-i} V_\eta^-(s_{t+n_s}) - V_\eta(s_{t+i}) \ , \end{aligned} \quad (8)$$

the A2C loss for the policy parameters ρ at time t is

$$\begin{aligned} \mathcal{L}_t^A(\rho) = &-\frac{1}{n_e n_s} \\ &\sum_{\text{envs}}^{n_e} \sum_{i=0}^{n_s-1} \log \pi_\rho(a_{t+i}|s_{t+i}) A_\eta^{t,i,-}(s_{t+i}, a_{t+i}). \end{aligned} \quad (9)$$

and the value function loss to learn η can be written as

$$\mathcal{L}_t^V(\eta) = \frac{1}{n_e n_s} \sum_{\text{envs}}^{n_e} \sum_{i=0}^{n_s-1} A_\eta^{t,i}(s_{t+i}, a_{t+i})^2. \quad (10)$$

Lastly, an entropy loss is added to encourage exploration:

$$\mathcal{L}_t^H(\rho) = -\frac{1}{n_e n_s} \sum_{\text{envs}}^{n_e} \sum_{i=0}^{n_s-1} H(\pi_\rho(\cdot|s_{t+i})), \quad (11)$$

where $H(\cdot)$ is the entropy of a distribution.

3. Deep Variational Reinforcement Learning

Fundamentally, there are two approaches to aggregating the history in the presence of partial observability: remembering features of the past or maintaining beliefs.

In most previous work, including ADRQN (Zhu et al., 2017), the current history $(a_{\leq t}, o_{<t})$ is encoded by an RNN, which leads to the recurrent update equation for the latent state h_t:

$$h_t = \text{RNNUpdate}_\phi(h_{t-1}, a_{t-1}, o_t) \ . \quad (12)$$

Since this approach is model-free, it is unlikely to approximate belief update steps, instead relying on memory or simple heuristics.

Inspired by the premise that a good way to solve many POMDPs involves (1) estimating the transition and observation model of the environment, (2) performing inference under this model, and (3) choosing an action based on the inferred belief state, we propose deep variational reinforcement learning (DVRL). It extends the RNN-based approach to explicitly support belief inference. Training everything end-to-end shapes the learned model to be useful for the RL task at hand, and not only for predicting observations.

We first explain our baseline architecture and training method in Section 3.1. For a fair comparison, we modify the original architecture of Zhu et al. (2017) in several ways. We find that our new baseline outperforms their reported results in the majority of cases.

In Sections 3.2 and 3.3, we explain our new latent belief state $\hat{b}_t$ and the recurrent update function

$$\hat{b}_t = \text{BeliefUpdate}_{\theta,\phi}(\hat{b}_{t-1}, a_{t-1}, o_t) \tag{13}$$

which replaces (12). Lastly, in Section 3.4, we describe our modified loss function, which allows learning the model jointly with the policy.

3.1. Improving the Baseline

While previous work often used Q-learning to train the policy (Hausknecht & Stone, 2015; Zhu et al., 2017; Foerster et al., 2016; Narasimhan et al., 2015), we use n-step A2C. This avoids drawing entire trajectories from a replay buffer and allows continuous actions.

Furthermore, since A2C interleaves unrolled trajectories and performs a parameter update only every n_s steps, it makes it feasible to maintain an approximately correct latent state. A small bias is introduced by not recomputing the latent state after each gradient update step.

We also modify the implementation of backpropagation-throught-time (BPTT) for n-step A2C in the case of policies with latent states. Instead of backpropagating gradients only through the computation graph of the current update involving n_s steps, we set the size of the computation graph independently to involve n_g steps. This leads to an average BPTT-length of $(n_g - 1)/2$.[2] This decouples the bias-variance tradeoff of choosing n_s from the bias-runtime tradeoff of choosing n_g. Our experiments show that choosing $n_g > n_s$ greatly improves the agent's performance.

3.2. Extending the Latent State

For DVRL, we extend the latent state to be a set of K particles, capturing the uncertainty in the belief state (Thrun, 2000; Silver & Veness, 2010). Each particle consists of the triplet (h_t^k, z_t^k, w_t^k) (Chung et al., 2015). The value h_t^k of particle k is the latent state of an RNN; z_t^k is an additional stochastic latent state that allows us to learn stochastic transition models; and w_t^k assigns each particle an importance weight.

Our latent state $\hat{b}_t$ is thus an approximation of the belief state in our learned model

$$p_\theta(h_{\leq T}, z_{\leq T}, o_{\leq T}|a_{<T}) = p_\theta(h_0)\prod_{t=1}^{T} p_\theta(z_t|h_{t-1}, a_{t-1})$$
$$p_\theta(o_t|h_{t-1}, z_t, a_{t-1})\delta_{\psi_\theta^{\text{RNN}}(h_{t-1},z_t,a_{t-1},o_t)}(h_t), \tag{14}$$

with stochastic transition model $p_\theta(z_t|h_{t-1}, a_{t-1})$, decoder $p_\theta(o_t|h_{t-1}, z_t, a_{t-1})$, and deterministic transition function $h_t = \psi_\theta^{\text{RNN}}(h_{t-1}, z_t, a_{t-1}, o_t)$ which is denoted using the delta-distribution δ and for which we use an RNN. The model is trained to jointly optimise the ELBO and the expected return.

3.3. Recurrent Latent State Update

To update the latent state, we proceed as follows:

$$u_{t-1}^k \sim \text{Discrete}\left(\frac{w_{t-1}^k}{\sum_{j=1}^K w_{t-1}^j}\right) \tag{15}$$

$$z_t^k \sim q_\phi(z_t^k|h_{t-1}^{u_{t-1}^k}, a_{t-1}, o_t) \tag{16}$$

$$h_t^k = \psi_\theta^{\text{RNN}}(h_{t-1}^{u_{t-1}^k}, z_t^k, a_{t-1}, o_t) \tag{17}$$

$$w_t^k = \frac{p_\theta(z_t^k|h_{t-1}^{u_{t-1}^k}, a_{t-1})p_\theta(o_t|h_{t-1}^{u_{t-1}^k}, z_t^k, a_{t-1})}{q_\phi(z_t^k|h_{t-1}^{u_{t-1}^k}, a_{t-1}, o_t)}\,. \tag{18}$$

First, we resample particles based on their weight by drawing ancestor indices u_{t-1}^k. This improves model learning (Le et al., 2018; Maddison et al., 2017) and allows us to train the model jointly with the n-step loss (see Section 3.4).

For $k = 1 \ldots K$, new values for z_t^k are sampled from the encoder $q_\phi(z_t^k|h_{t-1}^{u_{t-1}^k}, a_{t-1}, o_t)$ which conditions on the resampled ancestor values $h_{t-1}^{u_{t-1}^k}$ as well as the last actions a_{t-1} and current observation o_t. Latent variables z_t are sampled using the reparameterisation trick. The values z_t^k, together with $h_{t-1}^{u_{t-1}^k}, a_{t-1}$ and o_t, are then passed to the transition function ψ_θ^{RNN} to compute h_t^k.

The weights w_t^k measure how likely each new latent state value (z_t^k, h_t^k) is under the model and how well it explains the current observation.

To condition the policy on the belief $\hat{b}_t = (z_t^k, h_t^k, w_t^k)_{k=1}^K$, we need to encode the set of particles into a vector representation $\hat{h}_t$. We use a second RNN that sequentially takes in each tuple (z_t^k, h_t^k, w_t^k) and its last latent state is $\hat{h}_t$.

[2] This is implemented in PyTorch using the `retain_graph=True` flag in the `backward()` function.

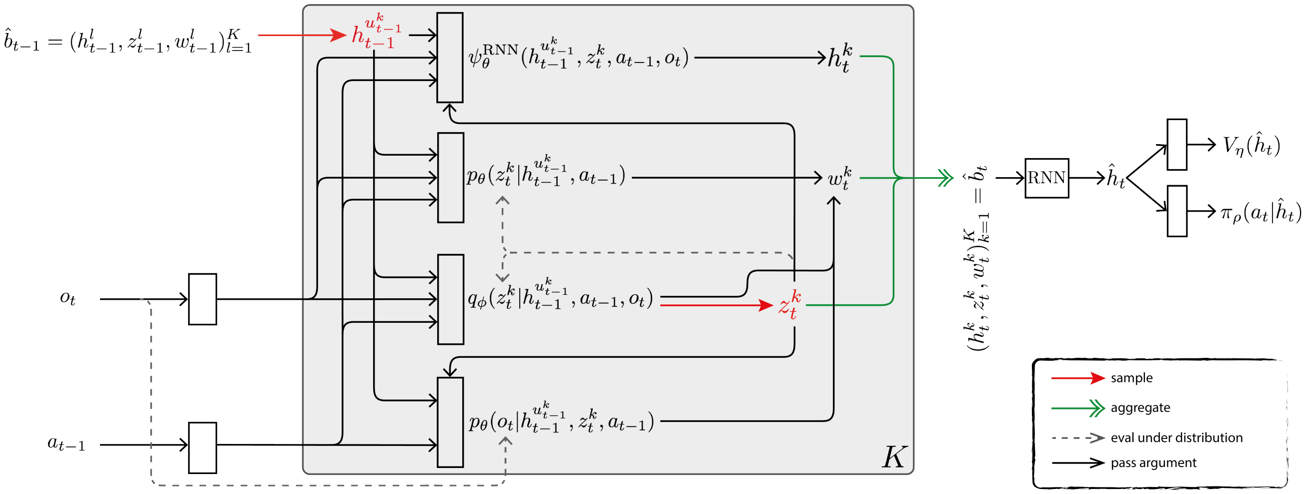

$\hat{b}_{t-1} = (h^l_{t-1}, z^l_{t-1}, w^l_{t-1})^K_{l=1}$
$h^{u^k_{t-1}}_{t-1}$
$\psi^{\text{RNN}}_\theta(h^{u^k_{t-1}}_{t-1}, z^k_t, a_{t-1}, o_t)$
h^k_t
$p_\theta(z^k_t|h^{u^k_{t-1}}_{t-1}, a_{t-1})$
w^k_t
o_t
$q_\phi(z^k_t|h^{u^k_{t-1}}_{t-1}, a_{t-1}, o_t)$
z^k_t
a_{t-1}
$p_\theta(o_t|h^{u^k_{t-1}}_{t-1}, z^k_t, a_{t-1})$
K
$(h^k_t, z^k_t, w^k_t)^K_{k=1} = \hat{b}_t$
RNN
$\hat{h}_t$
$V_\eta(\hat{h}_t)$
$\pi_\rho(a_t|\hat{h}_t)$
sample
aggregate
eval under distribution
pass argument

Figure 2: Overview of DVRL. We do the following K times to compute our new belief $\hat{b}_t$: Sample an ancestor index u^k_{t-1} based on the previous weights $w^{1:K}_{t-1}$ (Eq. 15). Pick the ancestor particle value $h^{u^k_{t-1}}_{t-1}$ and use it to sample a new stochastic latent state z^k_t from the encoder q_ϕ (Eq. 16). Compute h^k_t (Eq. 17) and w^k_t (Eq. 18). Aggregate all K values into the new belief $\hat{b}_t$ and summarise them into a vector representation $\hat{h}_t$ using a second RNN. Actor and critic can now condition on $\hat{h}_t$ and $\hat{b}_t$ is used as input for the next timestep. Red arrows denote random sampling, green arrows the aggregation of K values. Black solid arrows denote the passing of a value as argument to a function and black dashed ones the evaluation of a value under a distribution. Boxes indicate neural networks. Distributions are normal or Bernoulli distributions whose parameters are outputs of the neural network.

Additional encoders are used for a_t, o_t and z_t; see Appendix A for details. Figure 2 summarises the entire update step.

3.4. Loss Function

To encourage learning a model, we include the term

$$\mathcal{L}^{\text{ELBO}}_t(\theta, \phi) = -\frac{1}{n_e n_s} \sum_{\text{envs}}^{n_e} \sum_{i=0}^{n_s-1} \log\left(\frac{1}{K}\sum_{k=1}^{K} w^k_{t+i}\right) \tag{19}$$

in each gradient update every n_s steps. This leads to the overall loss:

$$\mathcal{L}^{\text{DVRL}}_t(\rho, \eta, \theta, \phi) = \mathcal{L}^A_t(\rho, \theta, \phi) + \lambda^H \mathcal{L}^H_t(\rho, \theta, \phi) + \lambda^V \mathcal{L}^V_t(\eta, \theta, \phi) + \lambda^E \mathcal{L}^{\text{ELBO}}_t(\theta, \phi)\ . \tag{20}$$

Compared to (9), (10) and (11), the losses now also depend on the encoder parameters ϕ and, for DVRL, model parameters θ, since the policy and value function now condition on the latent states instead of s_t. By introducing the n-step approximation $\mathcal{L}^{\text{ELBO}}_t$, we can learn θ and ϕ to jointly optimise the ELBO and the RL loss $\mathcal{L}^A_t + \lambda^H \mathcal{L}^H_t + \lambda^V \mathcal{L}^V_t$.

If we assume that observations and actions are drawn from the stationary state distribution induced by the policy π_ρ, then $\mathcal{L}^{\text{ELBO}}_t$ is a stochastic approximation to the action-conditioned ELBO:

$$\frac{1}{T}\mathbb{E}_{p(\tau)}\,\text{ELBO}_{\text{SMC}}(o_{\leq T}|a_{<T}) = \frac{1}{T}\mathbb{E}_{p(\tau)}\mathbb{E}\left[\sum_{t=1}^{T}\log\left(\frac{1}{K}\sum_{k=1}^{K} w^k_t\right)\middle|\, a_{\leq T}\right]\ , \tag{21}$$

which is a conditional extension of (6) similar to the extension of VAEs by Sohn et al. (2015). The expectation over $p(\tau)$ is approximated by sampling trajectories and the sum $\sum_{t=1}^{T}$ over the entire trajectory is approximated by a sum $\sum_{i=t}^{t+n_s-1}$ over only a part of it.

The importance of the resampling step (15) in allowing this approximation becomes clear if we compare (21) with the ELBO for the importance weighted autoencoder (IWAE) that does not include resampling (Doucet & Johansen, 2009; Burda et al., 2016):

$$\text{ELBO}_{\text{IWAE}}(o_{\leq T}|a_{<T}) = \mathbb{E}\left[\log\left(\frac{1}{K}\sum_{k=1}^{K}\prod_{t=1}^{T} w^k_t\right)\middle|\, a_{\leq T}\right]. \tag{22}$$

Because this loss is not additive over time, we cannot approximate it with shorter parts of the trajectory.

4. Related Work

Most existing POMDP literature focusses on *planning* algorithms, where the transition and observation functions, as well as a representation of the latent state space, are known

(Barto et al., 1995; McAllester & Singh, 1999; Pineau et al., 2003; Ross et al., 2008; Oliehoek et al., 2008; Roijers et al., 2015). In most realistic domains however, these are not known a priori.

There are several approaches that utilise RNNs in POMDPs (Bakker, 2002; Wierstra et al., 2007; Zhang et al., 2015; Heess et al., 2015), including multi-agent settings (Foerster et al., 2016), learning text-based fantasy games (Narasimhan et al., 2015) or, most recently, applied to Atari (Hausknecht & Stone, 2015; Zhu et al., 2017). As discussed in Section 3, our algorithm extends those approaches by enabling the policy to explicitly reason about a model and the belief state.

Another more specialised approach called QMDP-Net (Karkus et al., 2017) learns a value iteration network (VIN) (Tamar et al., 2016) end-to-end and uses it as a transition model for planning. However, a VIN makes strong assumptions about the transition function and in QMDP-Net the belief update must be performed analytically.

The idea to learn a particle filter based policy that is trained using policy gradients was previously proposed by Coquelin et al. (2009). However, they assume a known model and rely on finite differences for gradient estimation.

Instead of optimising an ELBO to learn a maximum-likelihood approximation for the latent representation and corresponding transition and observation model, previous work also tried to learn those dynamics using spectral methods (Azizzadenesheli et al., 2016), a Bayesian approach (Ross et al., 2011; Katt et al., 2017), or nonparametrically (Doshi-Velez et al., 2015). However, these approaches do not scale to large or continuous state and observation spaces. For continuous states, actions, and observations with Gaussian noise, a Gaussian process model can be learned (Deisenroth & Peters, 2012). An alternative to learning an (approximate) transition and observation model is to learn a model over trajectories (Willems et al., 1995). However, this is again only possible for small, discrete observation spaces.

Due to the complexity of the learning in POMDPs, previous work already found benefits to using auxiliary losses. Unlike the losses proposed by Lample & Chaplot (2017), we do not require additional information from the environment. The *UNREAL* agent (Jaderberg et al., 2016) is, similarly to our work, motivated by the idea to improve the latent representation by utilising all the information already obtained from the environment. While their work focuses on finding unsupervised auxiliary losses that provide good training signals, our goal is to use the auxiliary loss to better align the network computations with the task at hand by incorporating prior knowledge as an inductive bias.

There is some evidence from recent experiments on the dopamine system in mice (Babayan et al., 2018) showing that their response to ambiguous information is consistent with a theory operating on belief states.

5. Experiments

We evaluate DVRL on Mountain Hike and on flickering Atari. We show that DVRL deals better with noisy or partially occluded observations and that this scales to high dimensional and continuous observation spaces like images and complex tasks. We also perform a series of ablation studies, showing the importance of using many particles, including the ELBO training objective in the loss function, and jointly optimising the ELBO and RL losses.

More details about the environments and model architectures can be found in Appendix A together with additional results and visualisations. All plots and reported results are smoothed over time and parallel executed environments. We average over five random seeds, with shaded areas indicating the standard deviation.

5.1. Mountain Hike

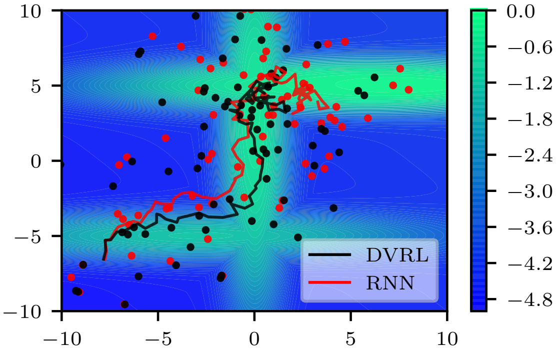


Figure 3: Mountain Hike is a continuous control task with observation noise $\sigma_o = 3$. Background colour indicates rewards. Red line: trajectory for RNN based encoder. Black line: trajectory for DVRL encoder. Dots: received observations. Both runs share the same noise values $\epsilon_{i,t}$. DVRL achieves higher returns (see Fig. 4) by better estimating its current location and remaining on the high reward mountain ridge.

In this task, the agent has to navigate along a mountain ridge, but only receives noisy measurements of its current location. Specifically, we have $\mathcal{S} = \mathcal{O} = \mathcal{A} = \mathbb{R}^2$ where $s_t = [x_t, y_t]^T \in \mathcal{S}$ and $o_t = [\hat{x}_t, \hat{y}_t]^T \in \mathcal{O}$ are true and observed coordinates respectively and $a_t = [\Delta x_t, \Delta y_t]^T \in \mathcal{A}$ is the desired step. Transitions are given by $s_{t+1} = s_t + \tilde{a}_t + \epsilon_{s,t}$ with $\epsilon_{s,t} \sim \mathcal{N}(\cdot|0, 0.25 \cdot I)$ and $\tilde{a}_t$ is the vector a_t with length capped to $\|\tilde{a}_t\| \leq 0.5$. Observations are noisy with $o_t = s_t + \epsilon_{o,t}$ with $\epsilon_{o,t} \sim \mathcal{N}(\cdot|0, \sigma_o \cdot I)$ and $\sigma_o \in \{0, 1.5, 3\}$. The reward at each timestep is $R_t = r(x_t, y_t) - 0.01\|a_t\|$ where $r(x_t, y_t)$ is shown in Figure 3. The starting position is sampled from $s_0 \sim \mathcal{N}(\cdot|[-8.5, -8.5]^T, I)$ and each episode ends after 75 steps.

DVRL used 30 particles and we set $n_g = 25$ for both RNN and DVRL. The latent state h for the RNN-encoder architecture was of dimension 256 and 128 for both z and h for DVRL. Lastly, $\lambda^E = 1$ and $n_s = 5$ were used, together with RMSProp with a learning rate of 10^{-4} for both approaches.

The main difficulty in Mountain Hike is to correctly estimate the current position. Consequently, the achieved return reflects the capability of the network to do so. DVRL outperforms RNN based policies, especially for higher levels of observation noise σ_o (Figure 4). In Figure 3 we compare the different trajectories for RNN and DVRL encoders for the same noise, i.e. $\epsilon^{\text{RNN}}_{s,t} = \epsilon^{\text{DVRL}}_{s,t}$ and $\epsilon^{\text{RNN}}_{o,t} = \epsilon^{\text{DVRL}}_{o,t}$ for all t and $\sigma_o = 3$. DVRL is better able to follow the mountain ridge, indicating that its inference based history aggregation is superior to a largely memory/heuristics based one.

The example in Figure 3 is representative but selected for clarity: The shown trajectories have $\Delta J(\sigma_o = 3) = 20.7$ compared to an average value of $\Delta \bar{J}(\sigma_o = 3) = 11.43$ (see Figure 4).

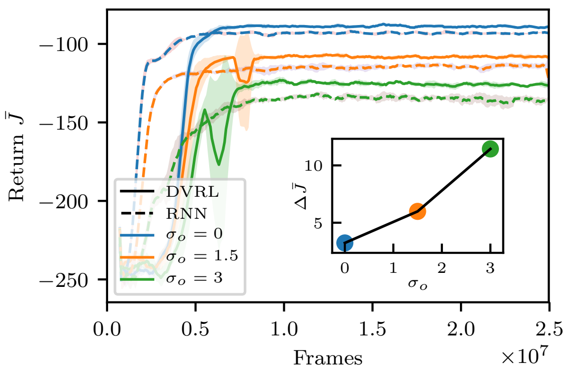


Figure 4: Returns achieved in Mountain Hike. Solid lines: DVRL. Dashed lines: RNN. Colour: Noise levels. *Inset:* Difference in performance between RNN and DVRL for same level of noise: $\Delta \bar{J}(\sigma_o) = \bar{J}(\text{DVRL}, \sigma_o) - \bar{J}(\text{RNN}, \sigma_o)$. DVRL achieves slighly higher returns for the fully observable case and, crucially, its performance deteriorates more slowly for increasing observation noise, showing the advantage of DVRL's inference computations in encoding the history in the presence of observation noise.

5.2. Atari

We chose flickering Atari as evaluation benchmark, since it was previously used to evaluate the performance of ADRQN (Zhu et al., 2017) and DRQN (Hausknecht & Stone, 2015). Atari environments (Bellemare et al., 2013) provide a wide set of challenging tasks with high dimensional observation spaces. We test our algorithm on the same subset of games on which DRQN and ADRQN were evaluated.

Partial observability is introduced by *flickering*, i.e., by a probability of 0.5 of returning a blank screen instead of the actual observation. Furthermore, only one frame is used as the observation. This is in line with previous work (Hausknecht & Stone, 2015). We use a frameskip of four[3] and for the stochastic Atari environments there is a 0.25 chance of repeating the current action for a second time at each transition.

DVRL used 15 particles and we set $n_g = 50$ for both agents. The dimension of h was 256 for both architectures, as was the dimension of z. Larger latent states decreased the performance for the RNN encoder. Lastly, $\lambda^E = 0.1$ and $n_s = 5$ was used with a learning rate of 10^{-4} for RNN and $2 \cdot 10^{-4}$ for DVRL, selected out of a set of 6 different rates based on the results on ChopperCommand.

Table 1 shows the results for the more challenging stochastic, flickering environments. Results for the deterministic environments, including returns reported for DRQN and ADRQN, can be found in Appendix A. DVRL significantly outperforms the RNN-based policy on five out of ten games and narrowly underperforms significantly on only one. This shows that DVRL is viable for high dimensional observation spaces with complex environmental models.

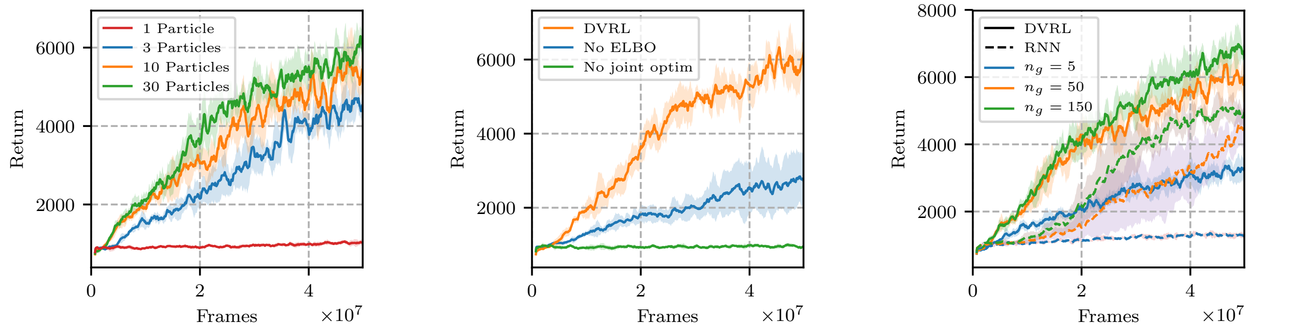
Table 1: Returns on stochastic and flickering Atari environments, averaged over 5 random seeds. Bold numbers indicate statistical significance at the 5% level. Out of ten games, DVRL significantly outperforms the baseline on five games and underperforms narrowly on only one game. Comparisons against DRQN and ADRQN on deterministic Atari environments are in Appendix A.

Env	DVRL($\pm std$)	RNN($\pm std$)
Pong	$\mathbf{18.17}(\pm 2.67)$	$6.33(\pm 3.03)$
Chopper	$\mathbf{6602}(\pm 449)$	$5150(\pm 488)$
MsPacman	$2221(\pm 199)$	$2312(\pm 358)$
Centipede	$4240(\pm 116)$	$4395(\pm 224)$
BeamRider	$1663(\pm 183)$	$1801(\pm 65)$
Frostbite	$\mathbf{297}(\pm 7.85)$	$254(\pm 0.45)$
Bowling	$29.53(\pm 0.23)$	$\mathbf{30.04}(\pm 0.18)$
IceHockey	$\mathbf{-4.88}(\pm 0.17)$	$-7.10(\pm 0.60)$
DDunk	$\mathbf{-5.95}(\pm 1.25)$	$-15.88(\pm 0.34)$
Asteroids	$1539(\pm 73)$	$1545(\pm 51)$

5.3. Ablation Studies

Using more than one particle is important to accurately approximate the belief distribution over the latent state (z, h). Consequently, we expect that higher particle numbers provide better information to the policy, leading to higher returns. Figure 5a shows that this is indeed the case. This is an important result for our architecture, as it also implies that the resampling step is necessary, as detailed in Section 3.4. Without resampling, we cannot approximate the ELBO

[3] A frameskip of one is used for Asteroids due to known rendering issues with this environment



(a) Influence of the particle number on performance for DVRL. Only using one particle is not sufficient to encode enough information in the latent state.

(b) Performance of the full DVRL algorithm compared to setting $\lambda^E = 0$ ("No ELBO") or not backpropagating the policy gradients through the encoder ("No joint optim").

(c) Influence of the maximum backpropagation length n_g on performance. Note that RNN suffers most from very short lengths. This is consistent with our conjecture that RNN relies mostly on memory, not inference.

Figure 5: Ablation studies on flickering ChopperCommand (Atari).

on only n_s observations.

Secondly, Figure 5b shows that the inclusion of $\mathcal{L}^{\text{ELBO}}$ to encourage model learning is required for good performance. Furthermore, not backpropagating the policy gradients through the encoder and only learning it based on the ELBO ("No joint optim") also deteriorates performance.

Lastly, we investigate the influence of the backpropagation length n_g on both the RNN and DVRL based policies. While increasing n_g universally helps, the key insight here is that a short length $n_g = 5$ (for an average BPTT-length of 2 timesteps) has a stronger negative impact on RNN than on DVRL. This is consistent with our notion that RNN is mainly performing memory based reasoning, for which longer backpropagation-through-time is required: The belief update (2) in DVRL is a one-step update from b_t to b_{t+1}, without the need to condition on past actions and observations. The proposal distribution can benefit from extended backpropagation lengths, but this is not necessary. Consequently, this result supports our notion that DVRL relies more on inference computations to update the latent state.

6. Conclusion

In this paper we proposed DVRL, a method for solving POMDPs given only a stream of observations, without knowledge of the latent state space or the transition and observation functions operating in that space. Our method leverages a new ELBO-based auxiliary loss and incorporates an inductive bias into the structure of the policy network, taking advantage of our prior knowledge that an inference step is required for an optimal solution.

We compared DVRL to an RNN-based architecture and found that we consistently outperform it on a diverse set of tasks, including a number of Atari games modified to have partial observability and stochastic transitions.

We also performed several ablation studies showing the necessity of using an ensemble of particles and of joint optimisation of the ELBO and RL objective. Furthermore, the results support our claim that the latent state in DVRL approximates a belief distribution in a learned model.

Access to a belief distribution opens up several interesting research directions. Investigating the role of better generalisation capabilities and the more powerful latent state representation on the policy performance of DVRL can give rise to further improvements. DVRL is also likely to benefit from more powerful model architectures and a disentangled latent state. Furthermore, uncertainty in the belief state and access to a learned model can be used for curiosity driven exploration in environments with sparse rewards.

Acknowledgements

We would like to thank Wendelin Boehmer and Greg Farquar for useful discussions and feedback. The NVIDIA DGX-1 used for this research was donated by the NVIDIA corporation. M. Igl is supported by the UK EPSRC CDT in Autonomous Intelligent Machines and Systems. L. Zintgraf is supported by the Microsoft Research PhD Scholarship Program. T. A. Le is supported by EPSRC DTA and Google (project code DF6700) studentships. F. Wood is supported by DARPA PPAML through the U.S. AFRL under Cooperative Agreement FA8750-14-2-0006; Intel and DARPA D3M, under Cooperative Agreement FA8750-17-2-0093. S. Whiteson is supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement number 637713).

References

Astrom, Karl J. Optimal control of markov decision processes with incomplete state estimation. *Journal of mathematical analysis and applications*, 10:174–205, 1965.

Azizzadenesheli, Kamyar, Lazaric, Alessandro, and Anandkumar, Animashree. Reinforcement learning of pomdps using spectral methods. *arXiv preprint 1602.07764*, 2016.

Babayan, Benedicte M, Uchida, Naoshige, and Gershman, Samuel J. Belief state representation in the dopamine system. *Nature communications*, 9(1):1891, 2018.

Bakker, Bram. Reinforcement learning with long short-term memory. In *Advances in neural information processing systems*, pp. 1475–1482, 2002.

Barto, Andrew G, Bradtke, Steven J, and Singh, Satinder P. Learning to act using real-time dynamic programming. *Artificial intelligence*, 72(1-2):81–138, 1995.

Bellemare, Marc, Veness, Joel, and Talvitie, Erik. Skip context tree switching. In *International Conference on Machine Learning*, pp. 1458–1466, 2014.

Bellemare, Marc G. Count-based frequency estimation with bounded memory. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

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

Burda, Yuri, Grosse, Roger, and Salakhutdinov, Ruslan. Importance weighted autoencoders. In *ICLR*, 2016.

Chung, Junyoung, Kastner, Kyle, Dinh, Laurent, Goel, Kratarth, Courville, Aaron C, and Bengio, Yoshua. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, 2015.

Coquelin, Pierre-Arnaud, Deguest, Romain, and Munos, Rémi. Particle filter-based policy gradient in pomdps. In *NIPS*, 2009.

Deisenroth, Marc Peter and Peters, Jan. Solving nonlinear continuous state-action-observation pomdps for mechanical systems with gaussian noise. 2012.

Dhariwal, Prafulla, Hesse, Christopher, Klimov, Oleg, Nichol, Alex, Plappert, Matthias, Radford, Alec, Schulman, John, Sidor, Szymon, and Wu, Yuhuai. Openai baselines, 2017.

Doshi-Velez, Finale, Pfau, David, Wood, Frank, and Roy, Nicholas. Bayesian nonparametric methods for partially-observable reinforcement learning. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):394–407, 2015.

Doucet, Arnaud and Johansen, Adam M. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of nonlinear filtering*, 12(656-704):3, 2009.

Foerster, Jakob N, Assael, Yannis M, de Freitas, Nando, and Whiteson, Shimon. Learning to communicate to solve riddles with deep distributed recurrent q-networks. *arXiv preprint 1602.02672*, 2016.

Hausknecht, Matthew and Stone, Peter. Deep recurrent q-learning for partially observable MDPs. In *2015 AAAI Fall Symposium Series*, 2015.

Heess, Nicolas, Hunt, Jonathan J, Lillicrap, Timothy P, and Silver, David. Memory-based control with recurrent neural networks. *arXiv preprint 1512.04455*, 2015.

Jaderberg, Max, Mnih, Volodymyr, Czarnecki, Wojciech Marian, Schaul, Tom, Leibo, Joel Z, Silver, David, and Kavukcuoglu, Koray. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint 1611.05397*, 2016.

Kaelbling, Leslie Pack, Littman, Michael L, and Cassandra, Anthony R. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1), 1998.

Karkus, Peter, Hsu, David, and Lee, Wee Sun. Qmdp-net: Deep learning for planning under partial observability. In *Advances in Neural Information Processing Systems*, pp. 4697–4707, 2017.

Katt, Sammie, Oliehoek, Frans A, and Amato, Christopher. Learning in pomdps with monte carlo tree search. In *International Conference on Machine Learning*, 2017.

Kingma, Diederik P and Welling, Max. Auto-encoding variational Bayes. In *ICLR*, 2014.

Lample, Guillaume and Chaplot, Devendra Singh. Playing fps games with deep reinforcement learning. In *AAAI*, pp. 2140–2146, 2017.

Le, Tuan Anh, Igl, Maximilian, Jin, Tom, Rainforth, Tom, and Wood, Frank. Auto-encoding sequential Monte Carlo. In *ICLR*, 2018.

Maddison, Chris J, Lawson, John, Tucker, George, Heess, Nicolas, Norouzi, Mohammad, Mnih, Andriy, Doucet, Arnaud, and Teh, Yee. Filtering variational objectives. In *Advances in Neural Information Processing Systems*, 2017.

McAllester, David A and Singh, Satinder. Approximate planning for factored pomdps using belief state simplification. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, 1999.

McCallum, Andrew Kachites and Ballard, Dana. *Reinforcement learning with selective perception and hidden state*. PhD thesis, University of Rochester. Dept. of Computer Science, 1996.

McCallum, R Andrew. Overcoming incomplete perception with utile distinction memory. In *Proceedings of the Tenth International Conference on Machine Learning*, pp. 190–196, 1993.

Messias, João V and Whiteson, Shimon. Dynamic-depth context tree weighting. In *Advances in Neural Information Processing Systems*, pp. 3330–3339, 2017.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Mnih, Volodymyr, Badia, Adria Puigdomenech, Mirza, Mehdi, Graves, Alex, Lillicrap, Timothy, Harley, Tim, Silver, David, and Kavukcuoglu, Koray. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016.

Naesseth, Christian A, Linderman, Scott W, Ranganath, Rajesh, and Blei, David M. Variational sequential monte carlo. In *AISTATS (To Appear)*, 2018.

Narasimhan, Karthik, Kulkarni, Tejas, and Barzilay, Regina. Language understanding for text-based games using deep reinforcement learning. *arXiv preprint 1506.08941*, 2015.

Oliehoek, Frans A, Spaan, Matthijs TJ, Whiteson, Shimon, and Vlassis, Nikos. Exploiting locality of interaction in factored dec-pomdps. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, 2008.

Pineau, Joelle, Gordon, Geoff, Thrun, Sebastian, et al. Point-based value iteration: An anytime algorithm for pomdps. In *IJCAI*, volume 3, 2003.

Rezende, Danilo Jimenez, Mohamed, Shakir, and Wierstra, Daan. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, 2014.

Roijers, Diederik Marijn, Whiteson, Shimon, and Oliehoek, Frans A. Point-based planning for multi-objective pomdps. In *IJCAI*, pp. 1666–1672, 2015.

Ross, Stéphane, Pineau, Joelle, Paquet, Sébastien, and Chaib-Draa, Brahim. Online planning algorithms for pomdps. *Journal of Artificial Intelligence Research*, 32: 663–704, 2008.

Ross, Stéphane, Pineau, Joelle, Chaib-draa, Brahim, and Kreitmann, Pierre. A bayesian approach for learning and planning in partially observable markov decision processes. *Journal of Machine Learning Research*, 2011.

Shani, Guy, Brafman, Ronen I, and Shimony, Solomon E. Model-based online learning of pomdps. In *European Conference on Machine Learning*, pp. 353–364. Springer, 2005.

Silver, David and Veness, Joel. Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, pp. 2164–2172, 2010.

Sohn, Kihyuk, Lee, Honglak, and Yan, Xinchen. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, pp. 3483–3491, 2015.

Tamar, Aviv, Wu, Yi, Thomas, Garrett, Levine, Sergey, and Abbeel, Pieter. Value iteration networks. In *Advances in Neural Information Processing Systems*, pp. 2154–2162, 2016.

Thrun, Sebastian. Monte carlo pomdps. In *Advances in neural information processing systems*, pp. 1064–1070, 2000.

Wierstra, Daan, Foerster, Alexander, Peters, Jan, and Schmidhuber, Juergen. Solving deep memory pomdps with recurrent policy gradients. In *International Conference on Artificial Neural Networks*, pp. 697–706. Springer, 2007.

Willems, Frans MJ, Shtarkov, Yuri M, and Tjalkens, Tjalling J. The context-tree weighting method: basic properties. *IEEE Transactions on Information Theory*, 41 (3):653–664, 1995.

Wu, Yuhuai, Mansimov, Elman, Grosse, Roger B, Liao, Shun, and Ba, Jimmy. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. In *Advances in neural information processing systems*, pp. 5285–5294, 2017.

Zhang, Marvin, Levine, Sergey, McCarthy, Zoe, Finn, Chelsea, and Abbeel, Pieter. Policy learning with continuous memory states for partially observed robotic control. *CoRR*, 2015.

Zhu, Pengfei, Li, Xin, and Poupart, Pascal. On improving deep reinforcement learning for POMDPs. *arXiv preprint 1704.07978*, 2017.

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 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 DVRL and action-specific deep recurrent A2C network (ADR-A2C) 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

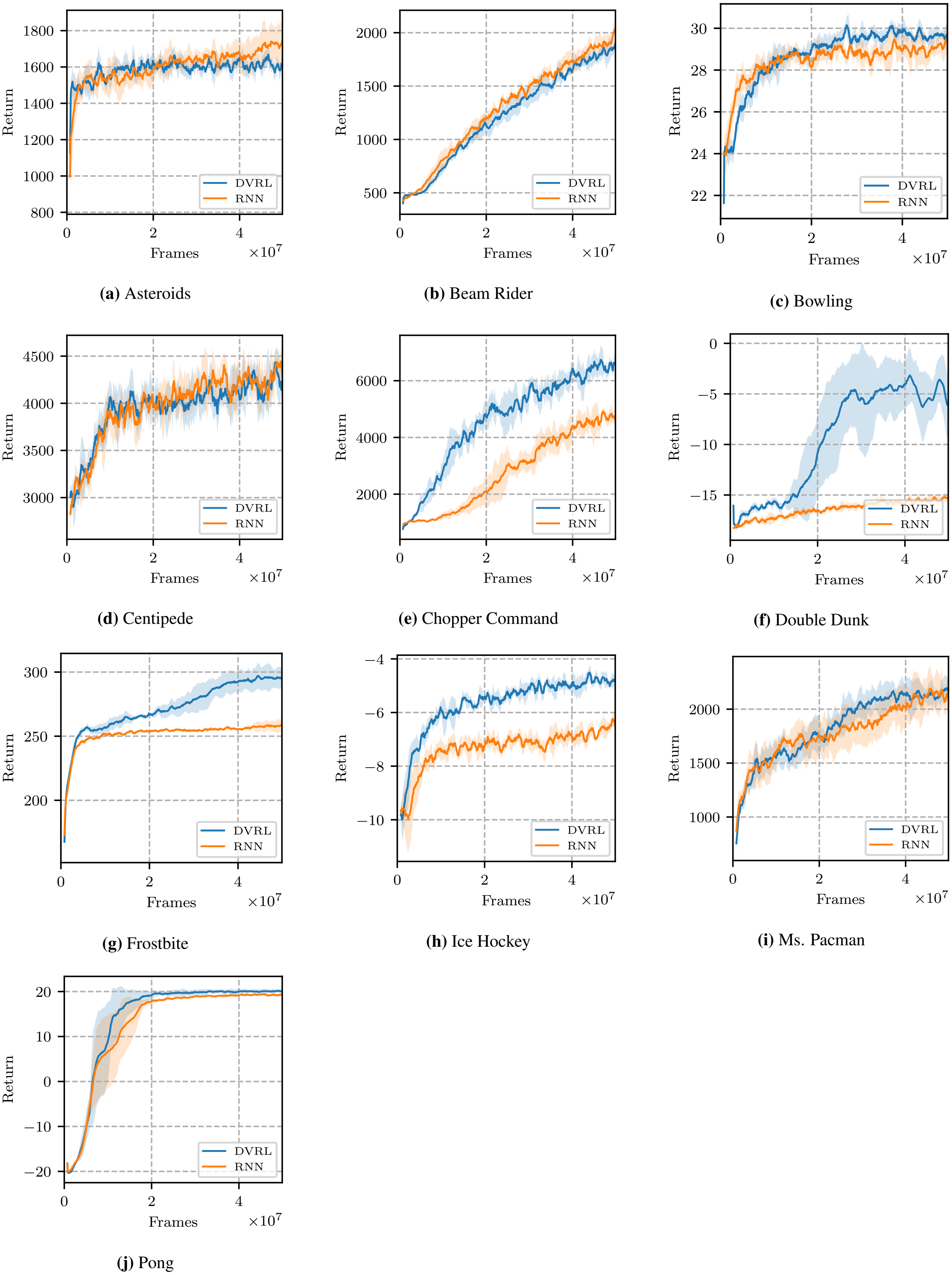
Table 2 shows the on *deterministic* and flickering Atari, averaged over 5 random seeds. The values for DRQN and ADRQN are taken from the respective papers. Note that DRQN and ADRQN rely on Q-learning instead of A2C, so the results are not directly comparable.

Figure 6 and 7 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

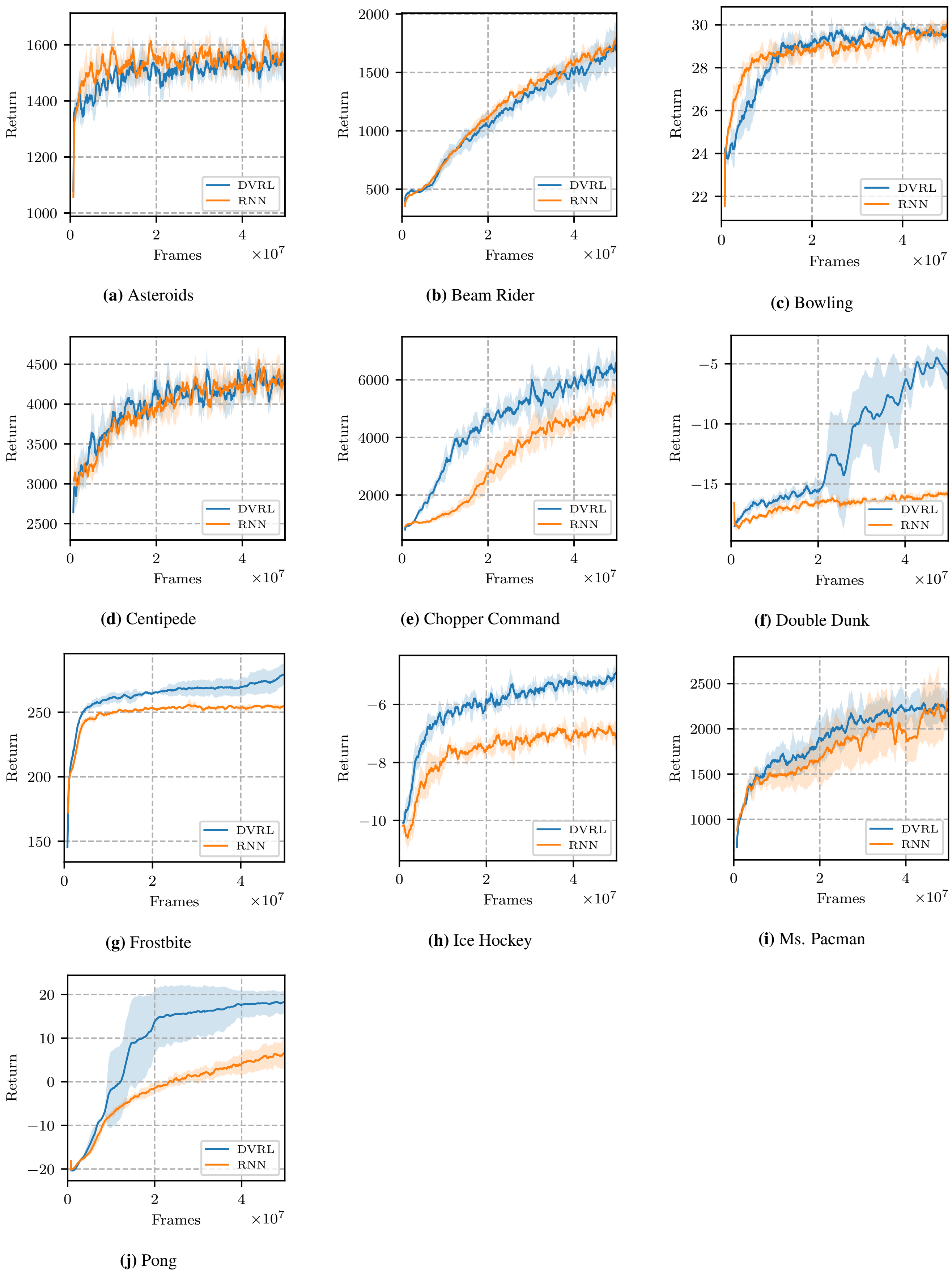
(g) Frostbite

(h) Ice Hockey

(i) Ms. Pacman

(j) Pong

Figure 6: 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 7: Training curves on the full set of evaluated Atari games, in the case of flickering and *stochastic* environments.

Table 2: 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	$\mathbf{20.07}(\pm0.39)$	$19.3(\pm0.26)$	$12.1(\pm2.2)$	$7(\pm4.6)$
Chopper	$\mathbf{6619}(\pm532)$	$4619(\pm306)$	$1330(\pm294)$	$1608(\pm707)$
MsPacman	$2156(\pm127)$	$2113(\pm135)$	$1739(\pm942)$	
Centipede	$4171(\pm127)$	$4283(\pm187)$	$4319(\pm4378)$	
BeamRider	$1901(\pm67)$	$\mathbf{2041}(\pm81)$	$618(\pm115)$	
Frostbite	$\mathbf{296}(\pm8.2)$	$259(\pm5.7)$	$414(\pm494)$	$2002(\pm734)$
Bowling	$29.74(\pm0.49)$	$29.38(\pm0.52)$	$65(\pm13)$	
IceHockey	$\mathbf{-4.87}(\pm0.24)$	$-6.49(\pm0.27)$	$-5.4(\pm2.7)$	
DDunk	$\mathbf{-6.08}(\pm3.08)$	$-15.25(\pm0.51)$	$-14(\pm2.5)$	$-13(\pm3.6)$
Asteroids	$1610(\pm63)$	$\mathbf{1750}(\pm97)$	$1032(\pm410)$	$1040(\pm431)$

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

A.4. Model Predictions

In Figure 8 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 Section 5.2. The reconstructed and predicted images are a weighted average over all 16 particles.

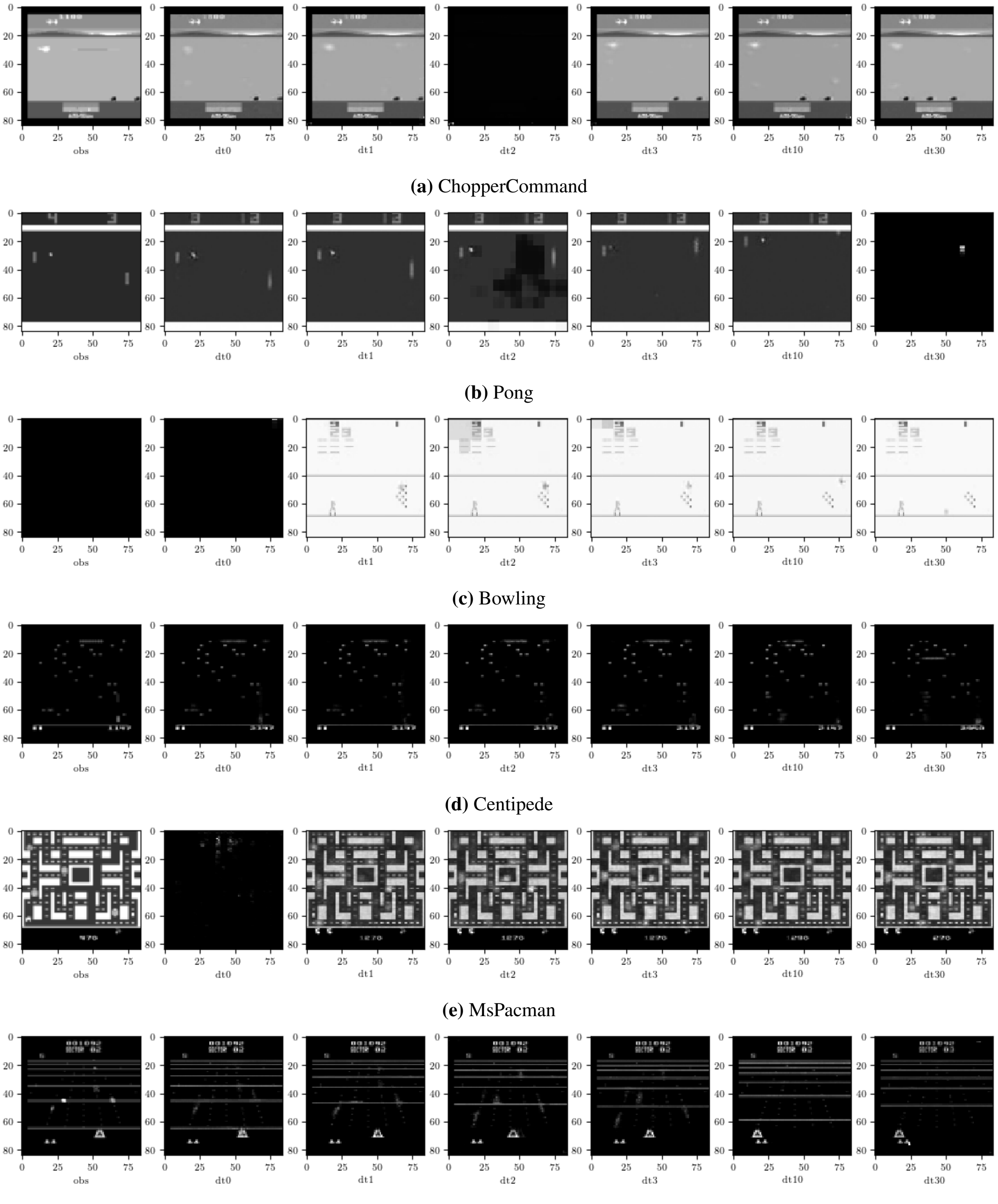
Note that the model is able to correctly predict features of future observations, for example the movement of the cars in ChopperCommand, the (approximate) ball position in Pong or the missing pins in Bowling. Furthermore, it is able to do so, even if the current observation is blank like in Bowling. 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 8: 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 $\mathrm{Env}$, Encoder $\mathrm{Enc}_{\theta,\phi}$ (either RNN or DVRL)
Initialize observation $o_1$ from $\mathrm{Env}$.
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
  $\mathcal{L}_j^{Enc}, a_j, a'_j, s_j, s'_j, o_{j+1}, r_{j+1}, done_{j+1} \leftarrow NULL \quad j = 1 \dots n$
  {Run $n$ steps forward:}
  for $j = 1$ to $n$ do
    $s_j, \mathcal{L}_j^{\mathrm{ELBO}} \leftarrow \mathrm{Enc}_{\theta,\phi}(o_j, a'_{j-1}, s'_{j-1})$
    Sample $a_j \sim \pi_\rho(a_j|s_j)$
    $o_{j+1}, r_{j+1}, done_{j+1} \leftarrow \mathrm{Env}(a_j)$
    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 $\mathrm{Env}()$
    else
      $s'_j, a'_j \leftarrow s_j, a_j$
    end if
  end for
  {Compute targets}
  $s_{n+1} \leftarrow \mathrm{Enc}_{\theta,\phi}(o_{n+1}, a'_n, s'_n)$
  $Q_{n+1}^{target} \leftarrow V_\eta(s_{n+1}).detach()$
  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
    $\mathcal{L}_j^V \leftarrow (Q_j^{target} - V_\eta(s_j))^2$
    $\mathcal{L}_j^A \leftarrow -\log \pi_\rho(a_j|s_j)(Q_j^{target} - V_\eta(s_j))$
    $\mathcal{L}_j^H \leftarrow -\mathrm{Entropy}(\pi_\rho(\cdot|s_j))$
  end for
  $\mathcal{J} \leftarrow \sum_j(\lambda^V \mathcal{L}_j^V + \mathcal{L}_j^A + \lambda^H \mathcal{L}_j^H + \lambda^E \mathcal{L}_j^{\mathrm{ELBO}})$
  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
```