TEMPORAL DIFFERENCE VARIATIONAL AUTO-ENCODER

Anonymous authors
Paper under double-blind review

ABSTRACT

To act and plan in complex environments, we posit that agents should have a mental simulator of the world with three characteristics: (a) it should build an abstract state representing the condition of the world; (b) it should form a belief which represents uncertainty on the world; (c) it should go beyond simple step-by-step simulation, and exhibit temporal abstraction. Motivated by the absence of a model satisfying all these requirements, we propose TD-VAE, a generative sequence model that learns representations containing explicit beliefs about states several steps into the future, and that can be rolled out directly without single-step transitions. TD-VAE is trained on pairs of temporally separated time points, using an analogue of temporal difference learning used in reinforcement learning.

1 INTRODUCTION

Generative models of sequential data have received a lot of attention, due to their wide applicability in domains such as speech synthesis (van den Oord et al., 2016a; 2017), neural translation (Bahdanau et al., 2014), image captioning (Xu et al., 2015), and many others. Different application domains will often have different requirements (e.g. long term coherence, sample quality, abstraction learning, etc.), which in turn will drive the choice of the architecture and training algorithm.

Of particular interest to this paper is the problem of reinforcement learning in partially observed environments, where, in order to act and explore optimally, agents need to build a representation of the uncertainty about the world, computed from the information they have gathered so far. Furthermore, in order to plan in a model-based fashion, an agent needs to be able to imagine distant futures which are consistent with the agent's past. In many situations however, planning step-by-step is not a cognitively or computationally realistic approach.

To successfully address an application such as the above, we argue that a model of the agent's experience should exhibit the following properties:

- The model should learn an abstract *state representation* of the data and be capable of making predictions at the state level, not just the observation level.
- The model should learn a *belief state*, i.e. a deterministic, coded representation of the filtering posterior of the state given all the observations up to a given time. A belief state contains all the information an agent has about the state of the world and thus about how to act optimally.
- The model should exhibit *temporal abstraction*, both by making 'jumpy' predictions (predictions several time steps into the future), and by being able to learn from temporally separated time points without backpropagating through the entire time interval.

To our knowledge, no model in the literature meets these requirements. In this paper, we develop a new model and associated training algorithm, called *Temporal Difference Variational Auto-Encoder* (TD-VAE), which meets all of the above requirements. We first develop TD-VAE in the sequential, non-jumpy case, by using a modified evidence lower bound (ELBO) for stochastic state space models (Krishnan et al., 2015; Fraccaro et al., 2016; Buesing et al., 2018) which relies on jointly training a filtering posterior and a local smoothing posterior. We demonstrate that on a simple task, this new inference network and associated lower bound lead to improved likelihood compared to methods classically used to train deep state-space models.

Following the intuition given by the sequential TD-VAE, we develop the full TD-VAE model, which learns from temporally extended data by making jumpy predictions into the future. We show it can be used to train consistent jumpy simulators of complex 3D environments. Finally, we illustrate how training a filtering a posterior leads to the computation of a neural belief state with good representation of the uncertainty on the state of the environment.

2 MODEL DESIDERATA

2.1 CONSTRUCTION OF A LATENT STATE-SPACE

Autoregressive models. One of the simplest way to model sequential data $(x_1, \ldots, x_T)$ is to use the chain rule to decompose the joint sequence likelihood as a product of conditional probabilities, i.e. $\log p(x_1, \ldots, x_T) = \sum_t \log p(x_t \,|\, x_1, \ldots, x_{t-1})$. This formula can be used to train an autoregressive model of data, by combining an RNN which aggregates information from the past (recursively computing an internal state $h_t = f(h_{t-1}, x_t)$) with a conditional generative model which can score the data x_t given the context h_t. This idea is used in handwriting synthesis (Graves, 2013), density estimation (Uria et al., 2016), image synthesis (van den Oord et al., 2016b), audio synthesis (van den Oord et al., 2017), video synthesis (Kalchbrenner et al., 2016), generative recall tasks (Gemici et al., 2017), and environment modeling (Oh et al., 2015; Chiappa et al., 2017).

While these models are conceptually simple and easy to train, one potential weakness is that they only make predictions in the original observation space, and don't learn a compressed representation of data. As a result, these models tend to be computationally heavy (for video prediction, they constantly decode and re-encode single video frames). Furthermore, the model can be computationally unstable at test time since it is trained as a next step model (the RNN encoding real data), but at test time it feeds back its prediction into the RNN. Various methods have been used to alleviate this issue (Bengio et al., 2015; Lamb et al., 2016; Goyal et al., 2017; Amos et al., 2018).

State-space models. An alternative to autoregressive models are models which operate on a higher level of abstraction, and use latent variables to model stochastic transitions between *states* (grounded by observation-level predictions). This enables to sample state-to-state transitions only, without needing to render the observations, which can be faster and more conceptually appealing. They generally consist of decoder or prior networks, which detail the generative process of states and observations, and encoder or posterior networks, which estimate the distribution of latents given the observed data. There is a large amount of recent work on these type of models, which differ in the precise wiring of model components (Bayer & Osendorfer, 2014; Chung et al., 2015; Krishnan et al., 2015; Archer et al., 2015; Fraccaro et al., 2016; Liu et al., 2017; Serban et al., 2017; Buesing et al., 2018; Lee et al., 2018; Ha & Schmidhuber, 2018).

Let $\mathbf{z} = (z_1, \ldots, z_T)$ be a state sequence and $\mathbf{x} = (x_1, \ldots, x_T)$ an observation sequence. We assume a general form of state-space model, where the joint state and observation likelihood can be written as $p(\mathbf{x}, \mathbf{z}) = \prod_t p(z_t \,|\, z_{t-1}) p(x_t \,|\, z_t)$.[1] These models are commonly trained with a VAE-inspired bound, by computing a posterior $q(\mathbf{z} \,|\, \mathbf{x})$ over the states given the observations. Often, the posterior is decomposed autoregressively: $q(\mathbf{z} \,|\, \mathbf{x}) = \prod_t q(z_t \,|\, z_{t-1}, \phi_t(\mathbf{x}))$, where ϕ_t is a function of $(x_1, \ldots, x_t)$ for filtering posteriors or the entire sequence $\mathbf{x}$ for smoothing posteriors. This leads to the following lower bound:

$$\log p(\mathbf{x}) \geq \mathbb{E}_{\mathbf{z} \sim q(\mathbf{z} \,|\, \mathbf{x})} \left[\sum\nolimits_t \log p(x_t \,|\, z_t) + \log p(z_t \,|\, z_{t-1}) - \log q(z_t \,|\, z_{t-1}, \phi_t(\mathbf{x})) \right]. \quad (1)$$

2.2 ONLINE CREATION OF BELIEF STATE.

A key feature of sequential models of data is that they allow to reason about the conditional distribution of the future given the past: $p(x_{t+1}, \ldots, x_T \,|\, x_1, \ldots, x_t)$. For reinforcement learning in partially observed environments, this distribution governs the distribution of returns given past observations, and as such, it is sufficient to derive the optimal policy. For generative sequence modeling, it enables conditional generation of data given a context sequence. For this reason, it is desirable

[1] For notational simplicity, $p(z_1 \,|\, z_0) = p(z_1)$. Also note the conditional distributions could be very complex, using additional latent variables, flow models, or implicit models (for instance, if a deterministic RNN with stochastic inputs is used in the decoder).

to compute sufficient statistics $b_t = b_t(x_1, \ldots, x_t)$ of the future given the past, which allow to rewrite the conditional distribution as $p(x_{t+1}, \ldots, x_T \,|\, x_1, \ldots, x_t) \approx p(x_{t+1}, \ldots, x_T \,|\, b_t)$. For an autoregressive model as described in section 2.1, the internal RNN state h_t can immediately be identified as the desired sufficient statistics b_t. However, for the reasons mentioned in the previous section, we would like to identify an equivalent quantity for a state-space model.

For a state-space model, the filtering distribution $p(z_t \,|\, x_1, \ldots, x_t)$, also known as the belief state in reinforcement learning, is sufficient to compute the conditional future distribution, due to the Markov assumption underlying the state-space model and the following derivation:

$$p(x_{t+1}, \ldots, x_T \mid x_1, \ldots, x_t) = \int p(z_t \,|\, x_1, \ldots, x_t) p(x_{t+1}, \ldots, x_T \mid z_t) \,\mathrm{d}z_t. \tag{2}$$

Thus, if we train a network that extracts a code b_t from $(x_1, \ldots, x_t)$ so that $p(z_t \,|\, x_1, \ldots, x_t) \approx p(z_t \,|\, b_t)$, b_t would contain all the information about the state of the world the agent has, and would effectively form a neural belief state, i.e. a code fully characterizing the filtering distribution.

Classical training of state-space model does not compute a belief state: by computing a joint, autoregressive posterior $q(\mathbf{z} \,|\, \mathbf{x}) = \prod_t q(z_t \,|\, z_{t-1}, \mathbf{x})$, some of the uncertainty about the marginal posterior of z_t may be 'leaked' in the sample z_{t-1}. Since that sample is stochastic, to obtain all information from $(x_1, \ldots, x_t)$ about z_t, we would need to re-sample z_{t-1}, which would in turn require re-sampling z_{t-2} all the way to z_1.

While the notion of a belief state itself and its connection to optimal policies in POMDPs is well known (Astrom, 1965; Kaelbling et al., 1998; Hauskrecht, 2000), it has often been restricted to the tabular case (Markov chain), and little work investigates computing belief states for learned deep models. A notable exception is (Igl et al., 2018), which uses a neural form of particle filtering, and represents the belief state more explicitly as a weighted collection of particles. Related to our definition of belief states as sufficient statistics is the notion of predictive state representations (PSRs) (Littman & Sutton, 2002); see also (Venkatraman et al., 2017) for a model that learns PSRs which, combined with a decoder, can predict future observations.

Our last requirement for the model is that of temporal abstraction. We postpone the discussion of this aspect until section 4.

3 Belief-state-based ELBO for sequential TD-VAE

In this section, we develop a sequential model that satisfies the requirements given in the previous section, namely (a) it constructs a *latent state-space*, and (b) it creates a online *belief state*. We consider an arbitrary state space model with joint latent and observable likelihood given by $p(\mathbf{x}, \mathbf{z}) = \prod_t p(z_t \,|\, z_{t-1}) p(x_t \,|\, z_t)$, and we aim to optimize the data likelihood $\log p(\mathbf{x})$. We begin by autoregressively decomposing the data likelihood as: $\log p(\mathbf{x}) = \sum_t \log p(x_t \,|\, x_{<t})$. For a given t, we evaluate the conditional likelihood $p(x_t \,|\, x_{<t})$ by inferring over two latent states only: z_{t-1} and z_t, as they will naturally make belief states appear for times $t-1$ and t:

$$\begin{aligned}\log p(x_t \,|\, x_{<t}) \geq \mathop{\mathbb{E}}_{(z_{t-1}, z_t) \sim q(z_{t-1}, z_t | x_{\leq t})} \Big[& \log p(x_t \,|\, z_{t-1}, z_t, x_{<t}) + \log p(z_{t-1}, z_t \,|\, x_{<t}) \\ & - \log q(z_{t-1}, z_t \,|\, x_{\leq t}) \Big].\end{aligned} \tag{3}$$

Because of the Markov assumptions underlying the state-space model, we can simplify $p(x_t \,|\, z_{t-1}, z_t, x_{<t}) = p(x_t \,|\, z_t)$ and decompose $p(z_{t-1}, z_t \,|\, x_{<t}) = p(z_{t-1} \,|\, x_{<t}) p(z_t \,|\, z_{t-1})$. Next, we choose to decompose $q(z_{t-1}, z_t \,|\, x_{\leq t})$ as a belief over z_t and a one-step smoothing distribution over z_{t-1}: $q(z_{t-1}, z_t \,|\, x_{\leq t}) = q(z_t \,|\, x_{\leq t}) q(z_{t-1} \,|\, z_t, x_{\leq t})$. We obtain the following belief-based ELBO for state-space models:

$$\begin{aligned}\log p(x_t \,|\, x_{<t}) \geq \mathop{\mathbb{E}}_{(z_{t-1}, z_t) \sim q(z_{t-1}, z_t \,|\, x_{\leq t})} \Big[& \log p(x_t \,|\, z_t) + \log p(z_{t-1} \,|\, x_{<t}) + \log p(z_t \,|\, z_{t-1}) \\ & - \log q(z_t \,|\, x_{\leq t}) - \log q(z_{t-1} \,|\, z_t, x_{\leq t}) \Big].\end{aligned} \tag{4}$$

Both quantities $p(z_{t-1} \,|\, x_{\leq t-1})$ and $q(z_t \,|\, x_{\leq t})$ represent the belief state of the model at different times, so at this stage we approximate them with the same distribution $p_B(z \,|\, b)$, with $b_t = f(b_{t-1}, x_t)$

representing the belief state code for z_t. Similarly, we represent the smoothing posterior over z_{t-1} as $q(z_{t-1} \mid z_t, b_{t-1}, b_t)$. We obtain the following loss:

$$-\mathcal{L} = \mathop{\mathbb{E}}_{\substack{z_t \sim p_B(z_t|b_t) \\ z_{t-1} \sim q(z_{t-1}|z_t, b_t, b_{t-1})}} \Big[\log p(x_t \mid z_t) + \log p_B(z_{t-1} \mid b_{t-1}) + \log p(z_t \mid z_{t-1}) - \log p_B(z_t \mid b_t) - \log q(z_{t-1} \mid z_t, b_{t-1}, b_t) \Big]. \tag{5}$$

We provide an intuition on the different terms of the ELBO in the next section.

4 TD-VAE AND JUMPY STATE MODELING

The model derived in the previous section expresses a state model $p(z_t \mid z_{t-1})$ that describes how the state of the world evolves from one time step to the next. However, in many applications, the relevant timescale for planning may not be the one at which we receive observations and execute simple actions. Imagine for example planning for a trip abroad; the different steps involved (discussing travel options, choosing a destination, buying a ticket, packing a suitcase, going to the airport, and so on), all occur at vastly different time scales (potentially months in the future at the beginning of the trip, and days during the trip). Certainly, making a plan for this situation does not involve making second-by-second decisions. This suggests that we should look for models that can imagine future states directly, without going through all intermediate states.

Beyond planning, there are several other reasons that motivate modeling the future directly. First, training signal coming from the future can be stronger than small changes happening between time steps. Second, the behavior of the model should ideally be independent from the underlying temporal sub-sampling of the data, if the latter is an arbitrary choice. Third, jumpy predictions can be computationally efficient; when predicting several steps into the future, there may be some intervals where the prediction is either easy (e.g. a ball moving straight), or the prediction is complex but does not affect later time steps — which Neitz et al. (2018) call inconsequential chaos.

There is a number of research directions that consider temporal jumps. Koutnik et al. (2014) and Chung et al. (2016) consider recurrent neural network with skip connections, making it easier to bridge distant timesteps. Buesing et al. (2018) temporally sub-sample the data and build a jumpy model (for fixed jump size) of this data; but by doing so they also drop the information contained in the skipped observations. Neitz et al. (2018) and Jayaraman et al. (2018) predict sequences with variable time-skips, by choosing as target the most predictable future frames. They predict the observations directly without learning appropriate states, and only focus on nearly fully observed problems (and therefore do not need to learn a notion of belief state). For more general problems, this is a fundamental limitation, as even if one could in principle learn a jumpy observation model $p(x_{t+\delta}|x_{\leq t})$, it cannot be used recursively (feeding $x_{t+\delta}$ back to the RNN and predicting $x_{t+\delta+\delta'}$). This is because $x_{t+\delta}$ does not capture the full state of the system and so we would be missing information from t to $t+\delta$ to fully characterize what happens after time $t+\delta$. In addition, $x_{t+\delta}$ might not be appropriate even as target, because some important information can only be extracted from a number of frames (potentially arbitrarily separated), such as a behavior of an agent.

4.1 THE TD-VAE MODEL

Motivated by the model derived in section 3, we extend sequential TD-VAE to exhibit time abstraction. We start from the same assumptions and architectural form: there exists a sequence of states $z_1, \ldots, z_T$ from which we can predict the observations $x_1, \ldots, x_T$. A forward RNN encodes a belief state b_t from past observations $x_{\leq t}$. The main difference is that, instead of relating information known at times t and $t+1$ through the states z_t and z_{t+1}, we relate two distant time steps t_1 and t_2 through their respective states z_{t_1} and z_{t_2}, and we learn a jumpy, state-to-state model $p(z_{t_2} \mid z_{t_1})$ between z_{t_1} and z_{t_2}. Following equation 5, the negative loss for the TD-VAE model is:

$$\mathcal{L}_{t_1,t_2} = \mathop{\mathbb{E}}_{(z_{t_1}, z_{t_2}) \sim q(z_{t_1}, z_{t_2}|b_{t_1}, b_{t_2})} \Big[\log p(x_{t_2} \mid z_{t_2}) + \log p_B(z_{t_1} \mid b_{t_1}) + \log p(z_{t_2} \mid z_{t_1}) - \log p_B(z_{t_2} \mid b_{t_2}) - \log q(z_{t_1} \mid z_{t_2}, b_{t_1}, b_{t_2}) \Big] \tag{6}$$

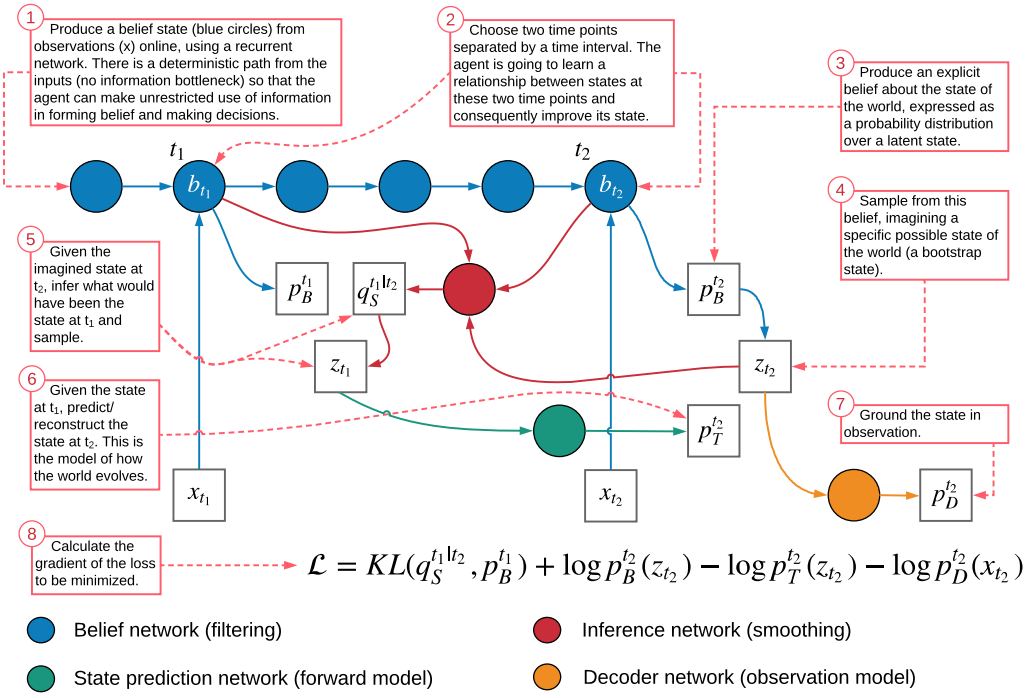


Figure 1: **Diagram of TD-VAE**. Follow the red panels for an explanation of the architecture. For succinctness, we use the notation p_D to denote the decoder $p(x|z)$, p_T to denote the transition distribution $p(s_{t_2}|s_{t_1})$, q_S for the smoothing distribution and p_B for the belief distribution.

To train this model, one should choose the distribution of times t_1, t_2; for instance, t_1 can be chosen uniformly from the sequence, and $t_2 - t_1$ uniformly over some finite range $[1, D]$; other approaches could be investigated. Figure 1 describes in detail the computation flow of the model.

Finally, it would be desirable to model the world with different hierarchies of state, the higher-level states predicting the same-level or lower-level states, and ideally representing more invariant or abstract information. For this reason, we also develop stacked (hierarchical) version of TD-VAE, which uses several layers of latent states. Hierarchical TD-VAE is detailed in the appendix.

4.2 Intuition behind TD-VAE

In this section, we provide a more intuitive explanation behind the computation and loss of the model. Assume we want to predict a future time step t_2 from all the information we have up until time t_1. All relevant information up until time t_1 (respectively t_2) has been compressed into a code b_{t_1} (respectively b_{t_2}). We make an observation x_t of the world[2] at every time step t, but posit the existence of a state z_t which fully captures the full condition of the world at time t.

Because b_t represents the state of knowledge of the agent at time t, it should not contain any information from the future, and therefore b_t should be computed from a forward RNN. Furthermore, b_t should have an unrestricted access to information; it should ideally not be disturbed by sampling (two identical agents with the same information should compute the same information; this will not be the case if the computation involves sampling), nor go through information bottlenecks.

At time t_2, the agent can make a guess about the state of the world by sampling from its belief model $p_B(z_{t_2} \mid b_{t_2})$. Because the state z_{t_2} should entail the corresponding observation x_{t_2}, the agent aims to maximize $p(x_{t_2} \mid z_{t_2})$, with a variational bottleneck penalty $-\log p(z_{t_2} \mid b_{t_2})$ to prevent too much information from the current observation x_{t_2} from being encoded into z_{t_2}. Then follows the question 'could the state of the world at time t_2 be predicted from the state of the world at time t_1?'. By now,

[2]In RL, this observation may include the reward and previous action.

the agent has aggregated observations between t_1 and t_2 that are informative about the state of the world at time t_1, but the agent is still uncertain about some elements of the world. However, the agent has a belief $p_B(z_{t_2} \mid b_{t_2})$ about the world and can sample an instance of this belief as a bootstrap, which, together with information acquired between t_1 and t_2, can be used to form the smoothing distribution $q(z_{t_1}|z_{t_2}, b_{t_1}, b_{t_2})$ and draw a sample z_{t_1}. Having guessed states of the world z_{t_1} and z_{t_2}, the agent aims to optimize its predictive jumpy model of the world state $p(z_{t_2} \mid z_{t_1})$. Further, it should attempt to see how predictable the revealed information was, or in other words, to assess whether the smoothing distribution $q(z_{t_1} \mid z_{t_2}, b_{t_2})$ could have been predicted from information only available at time t_1. The agent can do so by minimizing the KL between the smoothing distribution and the belief distribution at time t_1: $\mathbf{KL}(q(z_{t_1} \mid z_{t_2}, b_{t_1}, b_{t_2}) \,||\, p(z_{t_1} \mid b_{t_1}))$. Summing all the losses described so far, we obtain the TD-VAE loss.

4.3 Connection with temporal-difference learning

In reinforcement learning, the state of an agent represents a belief about the sum of discounted rewards $R_t = \sum_\tau r_{t+\tau}\gamma^\tau$. In the classic setting, the agent only models the mean of this distribution represented by the value function V_t or action dependent Q-function Q_t^a (Sutton & Barto, 1998). Recently in (Bellemare et al., 2017), a full distribution over R_t has been considered. To estimate V_{t_1} or $Q_{t_1}^a$ at time t_1, one does not usually wait to get all the rewards to compute R_{t_1}. Instead, one uses an estimate at some future time t_2 as a bootstrap to estimate V_{t_1} or $Q_{t_1}^a$ (temporal difference).

In our case, the model expresses a *belief* $p_B(z_t \mid b_t)$ about possible future *states* instead of the *sum* of discounted *rewards*. The model trains the belief $p_B(z_{t_1} \mid b_{t_1})$ at time t_1 using belief $p_B(z_{t_2} \mid b_{t_2})$ at some time t_2 in the future. It accomplishes this by (variationally) auto-encoding a sample z_{t_2} of the future state into a sample z_{t_1}, using the approximate posterior distribution $q(z_{t_1} \mid z_{t_2}, b_{t_1}, b_{t_2})$ and the decoding distribution $p(z_{t_2} \mid z_{t_1})$. This auto-encoding mapping translates between states at t_1 and t_2, forcing beliefs at the two time steps to be consistent. Sample z_{t_1} forms the target for training the belief $p_B(z_{t_1} \mid b_{t_1})$, which appears as a prior distribution over z_{t_1}.

5 Experiments.

The first experiment using sequential TD-VAE, which enables a direct comparison to related algorithms for training state-space models. Subsequent experiments use the full TD-VAE model.

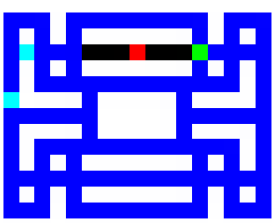
5.1 Partially observed MiniPacman

We use a partially observed version of the MiniPacman environment (Racanière et al., 2017), shown in Figure 2. The agent (Pacman) navigates a maze, and tries to eat all the food while avoiding being eaten by a ghost. Pacman sees only a 5×5 window around itself. To achieve a high score, the agent needs to form a belief state that captures memory of past experience (e.g. which parts of the maze have been visited) and uncertainty on the environment (e.g. where the ghost might be).

We evaluate the performance of sequential (non-jumpy) TD-VAE on the task of modeling a sequence of the agent's observations. We compare it with two state-space models trained using the standard ELBO of equation 1:

- A **filtering model** with encoder $q(\mathbf{z} \mid \mathbf{x}) = \prod_t q(z_t \mid z_{t-1}, b_t)$, where $b_t = \mathrm{RNN}(b_{t-1}, x_t)$.
- A **mean-field model** with encoder $q(\mathbf{z} \mid \mathbf{x}) = \prod_t q(z_t \mid b_t)$, where $b_t = \mathrm{RNN}(b_{t-1}, x_t)$.

Figure 2 shows the ELBO and estimated negative log probability on a test set of MiniPacman sequences for each model. TD-VAE outperforms both baselines, whereas the mean-field model is the least well-performing. We note that b_t is a belief state for the mean-field model, but not for the filtering model; the encoder of the latter explicitly depends on the previous latent state z_{t-1}, hence b_t is not its sufficient statistics. This comparison shows that naively restricting the encoder in order to obtain a belief state hurts the performance significantly; TD-VAE overcomes this difficulty.



	ELBO	$-\log p(\mathbf{x})$ (est.)
Filtering model	0.1169 ± 0.0003	0.0962 ± 0.0007
Mean-field model	0.1987 ± 0.0004	0.1678 ± 0.0010
TD-VAE	$\mathbf{0.0773} \pm \mathbf{0.0002}$	$\mathbf{0.0553} \pm \mathbf{0.0006}$

Figure 2: **MiniPacman**. **Left**: A full frame from the game (size 15×19). Pacman (green) is navigating the maze trying to eat all the food (blue) while being chased by a ghost (red). **Top right**: A sequence of observations, consisting of consecutive 5×5 windows around Pacman. **Bottom right**: ELBO and estimated negative log probability on a test set of MiniPacman sequences. Lower is better. Log probability is estimated using importance sampling with the encoder as proposal.

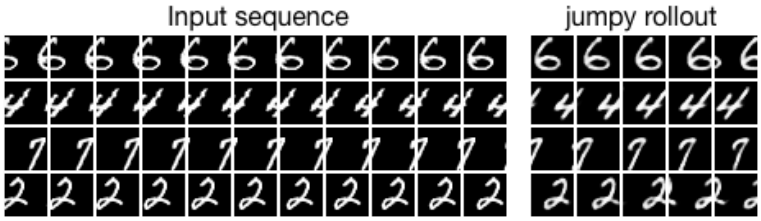


Figure 3: **Moving MNIST**. **Left**: Rows are example input sequences. **Right**: Jumpy rollouts from the model. We see that the model is able to roll forward by skipping frames, keeping the correct digit and the direction of motion.

5.2 Moving MNIST

In this experiment, we show that the model is able to learn the state and roll forward in jumps. We consider sequences of length 20 of images of MNIST digits. For each sequence, a random digit from the dataset is chosen, as well as the direction of movement (left or right). At each time step, the digit moves by one pixel in the chosen direction, as shown in Figure 3. We train the model with t_1 and t_2 separated by a random amount $t_2 - t_1$ from the interval $[1, 4]$. We would like to see whether the model at a given time can roll out a simulated experience in time steps $t_1 = t + \delta_1$, $t_2 = t_1 + \delta_2$, $\ldots$ with $\delta_1, \delta_2, \ldots > 1$, without considering the inputs in between these time points. Note that it is not sufficient to predict the future inputs $x_{t_1}, \ldots$ as they do not contain information about whether the digit moves left or right. We need to sample a state that contains this information.

We roll out a sequence from the model as follows: (a) b_t is computed by the aggregation recurrent network from observations up to time t; (b) a state z_t is sampled from $p_B(z_t \mid b_t)$; (c) a sequence of states is rolled out by repeatedly sampling $z \leftarrow z' \sim p(z' \mid z)$ starting with $z = z_t$; (d) each z is decoded by $p(x \mid z)$, producing a sequence of frames. The resulting sequences are shown in Figure 3. We see that indeed the model can roll forward the samples in steps of more than one elementary time step (the sampled digits move by more than one pixel) and that it preserves the direction of motion, demonstrating that it rolls forward a state.

5.3 Noisy harmonic oscillator

We would like to demonstrate that the model can build a state even when little information is present in each observation, and that it can sample states far into the future. For this we consider a 1D sequence obtained from a noisy harmonic oscillator, as shown in Figure 4 (first and fourth rows). The frequencies, initial positions and initial velocities are chosen at random from some range. At every update, noise is added to the position and the velocity of the oscillator, but the energy is approximately preserved. The model observes a noisy version of the current position. Attempting to predict the input, which consists of one value, 100 time steps in the future would be uninformative; such a prediction wouldn't reveal what the frequency or the magnitude of the signal is, and because the oscillator updates are noisy, the phase information would be nearly lost. Instead, we should try to

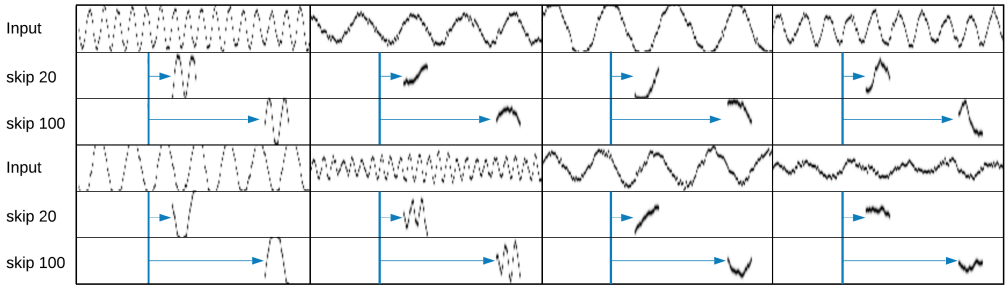
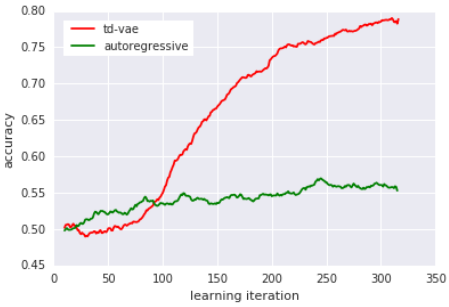


Figure 4: **Skip-state prediction for 1D signal**. The input is generated by a noisy harmonic oscillator. Rollouts consist of (a) a jumpy state transition with either $dt = 20$ or $dt = 100$, followed by 20 state transitions with $dt = 1$. The model is able to create a state and predict it into the future, correctly predicting frequency and magnitude of the signal.

predict as much as possible about the state, which consists of frequency, magnitude and position, and it is only the position that cannot be accurately predicted.

The aggregation RNN is an LSTM; we use a hierarchical TD-VAE with two layers, where the latent variables in the higher layer are sampled first, and their results are passed to the lower layer. The belief, smoothing and state-transition distributions are feed-forward networks, and the decoder simply extracts the first component from the z of the first layer. We also feed the time interval $t_2 - t_1$ into the smoothing and state-transition distributions. We train on sequences of length 200, with $t_2 - t_1$ taking values chosen at random from $[1, 10]$ with probability 0.8 and from $[1, 120]$ with probability 0.2.

We analyze what the model has learned as follows. We pick time $t_1 = 60$ and sample $z_{t_1} \sim p_B(z_{t_1} \,|\, b_{t_1})$. Then, we choose a time interval $\delta_t \in \{20, 100\}$ to skip, sample from the forward model $p(z_2 \,|\, z_1, \delta_t)$ to obtain z_{t_2} at $t_2 = t_1 + \delta_t$. To see the content of this state, we roll forward 20 times with time step $\delta = 1$ and plot the result, shown in Figure 4. We see that indeed the state z_{t_2} is predicted correctly, containing the correct frequency and magnitude of the signal. We also see that the position (phase) is predicted well for $dt = 20$ and less accurately for $dt = 100$ (at which point the noisiness of the system makes it unpredictable).



Finally, we show that TD-VAE training can improve the quality of the belief state. For this experiment, the harmonic oscillator has a different frequency in each interval $[0, 10), [10, 20), [20, 120), [120, 140)$. The first three frequencies f_1, f_2, f_3 are chosen at random. The final frequency f_4 is chosen to be one fixed value f_a if $f_1 > f_2$ and another fixed value f_b otherwise (f_a and f_b are constants). In order to correctly model the signal in the final time interval, the model needs to learn the relation between f_1 and f_2, store it over length of 100 steps, and apply it over a number of time steps (due to the noise) in the final interval. To test whether the belief state contains the information about this relationship, we train a binary classifier from the belief state to the final frequency f_4 at points just before the final interval. We compare two models with the same recurrent architecture (an LSTM), but trained with different objective: next-step prediction vs TD-VAE loss. The figure on the right shows the classification accuracy for the two methods, averaged over 20 runs. We found that the longer the separating time interval (containing frequency f_3) and the smaller the size of the LSTM, the better TD-VAE is compared to next-step predictor.

5.4 DeepMind Lab environment

In the final experiment, we analyze the model on a more visually complex domain. We use sequences of frames seen by an agent solving tasks in the DeepMind Lab environment (Beattie et al., 2016). We aim to demonstrate that the model holds explicit beliefs about various possible futures, and that it can roll out in jumps. We suggest functional forms inspired by convolutional DRAW: we use

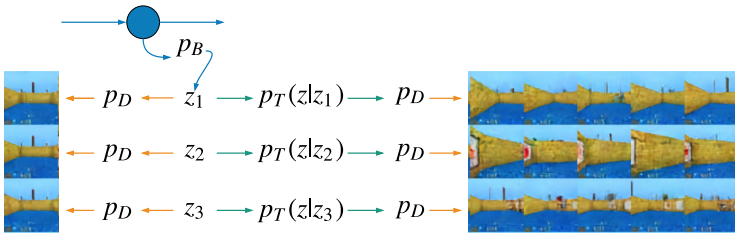


Figure 5: **Beliefs of the model**. **Left**: Independent samples z_1, z_2, z_3 from current belief; all 3 decode to roughly the same frame. **Right**: Multiple predicted futures for each sample. The frames are similar for each z_i, but different across z_i's.

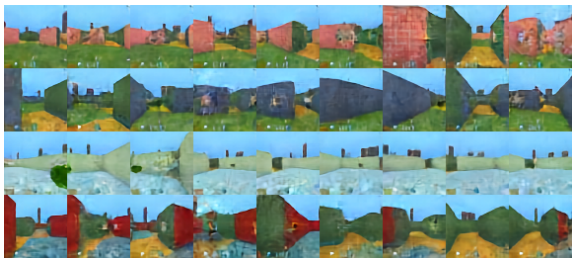


Figure 6: **Rollout from the model**. The model was trained on steps uniformly distributed in $[1, 5]$. The model is able to create forward motion that skips several time steps.

convolutional LSTMs for all the circles in Figure 8 and make the model 16 layers deep (except for the forward updating LSTMs which are fully connected with depth 4).

We use time skips $t_2 - t_1$ sampled uniformly from $[1, 40]$ and analyze the content of the belief state b. We take three samples z_1, z_2, z_3 from $p_B(z \mid b)$, which should represent three instances of possible futures. Figure 5 (left) shows that they decode to roughly the same frame. To see what they represent about the future, we draw 5 samples $z_i^k \sim p(\hat{z} \mid z)$, $k = 1, \ldots, 5$ and decode them, as shown in Figure 5 (right). We see that for a given i, the predicted samples decode to similar frames (images in the same row). However z's for different i's decode to different frames. This means b represented a belief about several different possible futures, while different z_i each represent a single possible future.

Finally, we show what rollouts look like. We train on time separations $t_2 - t_1$ chosen uniformly from $[1, 5]$ on a task where the agent tends to move forward and rotate. Figure 6 shows 4 rollouts from the model. We see that the motion appears to go forward and into corridors and that it skips several time steps (real single step motion is slower).

6 Conclusions

In this paper, we argued that an agent needs a model that is different from an accurate step-by-step environment simulator. We discussed the requirements for such a model, and presented TD-VAE, a sequence model that satisfies all requirements. TD-VAE builds states from observations by bridging time points separated by random intervals. This allows the states to relate to each other directly over longer time stretches and explicitly encode the future. Further, it allows rolling out in state-space and in time steps larger than, and potentially independent of, the underlying temporal environment/data step size. In the future, we aim to apply TD-VAE to more complex settings, and investigate a number of possible uses in reinforcement learning such are representation learning and planning.

REFERENCES

Brandon Amos, Laurent Dinh, Serkan Cabi, Thomas Rothörl, Sergio Gómez Colmenarejo, Alistair Muldal, Tom Erez, Yuval Tassa, Nando de Freitas, and Misha Denil. Learning awareness models. *arXiv preprint arXiv:1804.06318*, 2018.

Evan Archer, Il Memming Park, Lars Buesing, John Cunningham, and Liam Paninski. Black box variational inference for state space models. *arXiv preprint arXiv:1511.07367*, 2015.

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

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Justin Bayer and Christian Osendorfer. Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*, 2014.

Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. DeepMind Lab. *arXiv preprint arXiv:1612.03801*, 2016.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*, 2017.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 1171–1179, 2015.

Lars Buesing, Theophane Weber, Sebastien Racaniere, SM Eslami, Danilo Rezende, David P Reichert, Fabio Viola, Frederic Besse, Karol Gregor, Demis Hassabis, et al. Learning and querying fast generative models for reinforcement learning. *arXiv preprint arXiv:1802.03006*, 2018.

Silvia Chiappa, Sébastien Racaniere, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators. *arXiv preprint arXiv:1704.02254*, 2017.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pp. 2980–2988, 2015.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*, 2016.

Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther. Sequential neural models with stochastic layers. In *Advances in neural information processing systems*, pp. 2199–2207, 2016.

Mevlana Gemici, Chia-Chun Hung, Adam Santoro, Greg Wayne, Shakir Mohamed, Danilo J Rezende, David Amos, and Timothy Lillicrap. Generative temporal models with memory. *arXiv preprint arXiv:1702.04649*, 2017.

Anirudh Goyal, Alessandro Sordoni, Marc-Alexandre Côté, Nan Ke, and Yoshua Bengio. Z-forcing: Training stochastic recurrent networks. In *Advances in Neural Information Processing Systems*, pp. 6713–6723, 2017.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Karol Gregor, Frederic Besse, Danilo Jimenez Rezende, Ivo Danihelka, and Daan Wierstra. Towards conceptual compression. In *Advances In Neural Information Processing Systems*, pp. 3549–3557, 2016.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Milos Hauskrecht. Value-function approximations for partially observable Markov decision processes. *Journal of artificial intelligence research*, 13:33–94, 2000.

Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson. Deep variational reinforcement learning for POMDPs. *arXiv preprint arXiv:1806.02426*, 2018.

Dinesh Jayaraman, Frederik Ebert, Alexei A Efros, and Sergey Levine. Time-agnostic prediction: Predicting predictable video frames. *arXiv preprint arXiv:1808.07784*, 2018.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Nal Kalchbrenner, Aaron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Video pixel networks. *arXiv preprint arXiv:1610.00527*, 2016.

Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, pp. 4743–4751, 2016.

Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. A clockwork RNN. *arXiv preprint arXiv:1402.3511*, 2014.

Rahul G Krishnan, Uri Shalit, and David Sontag. Deep Kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

Alex Lamb, Anirudh Goyal, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. In *Advances In Neural Information Processing Systems*, pp. 4601–4609, 2016.

Alex X Lee, Richard Zhang, Frederik Ebert, Pieter Abbeel, Chelsea Finn, and Sergey Levine. Stochastic adversarial video prediction. *arXiv preprint arXiv:1804.01523*, 2018.

Michael L Littman and Richard S Sutton. Predictive representations of state. In *Advances in neural information processing systems*, pp. 1555–1561, 2002.

Hao Liu, Lirong He, Haoli Bai, and Zenglin Xu. Efficient structured inference for stochastic recurrent neural networks. 2017.

Alexander Neitz, Giambattista Parascandolo, Stefan Bauer, and Bernhard Schölkopf. Adaptive skip intervals: Temporal abstraction for recurrent dynamical models. *arXiv preprint arXiv:1808.04768*, 2018.

Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems*, pp. 2863–2871, 2015.

Sébastien Racanière, Théophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 5694–5705, 2017.

Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, pp. 3546–3554, 2015.

Iulian Vlad Serban, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron C Courville, and Yoshua Bengio. A hierarchical latent variable encoder-decoder model for generating dialogues. In *AAAI*, pp. 3295–3301, 2017.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *The Journal of Machine Learning Research*, 17(1):7184–7220, 2016.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016a.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016b.

Aaron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis C Cobo, Florian Stimberg, et al. Parallel waveNet: Fast high-fidelity speech synthesis. *arXiv preprint arXiv:1711.10433*, 2017.

Arun Venkatraman, Nicholas Rhinehart, Wen Sun, Lerrel Pinto, Martial Hebert, Byron Boots, Kris Kitani, and J Bagnell. Predictive-state decoders: Encoding the future into recurrent networks. In *Advances in Neural Information Processing Systems*, pp. 1172–1183, 2017.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pp. 2048–2057, 2015.

A INTUITIVE DERIVATION OF THE TD-VAE MODEL

In this section we start with a general recurrent variational auto-encoder and consider how the desired properties of requiring a belief state and sampling in latent space (without needing to sample inputs) constrain the architecture. We find that indeed, they impose significant constraints and lead naturally to TD-VAE.

Let us first construct a relatively general form of temporal variational auto-encoder. We only consider models where the same architecture is applied at every step and where the inputs are sampled forward (so that arbitrarily long sequences can be sampled). A very general form of such an architecture consist of forward-backward encoder RNNs and a forward decoder RNN (Figure 7) but otherwise allowing for all the connections. Several works (Chung et al., 2015; Lee et al., 2018; Archer et al., 2015; Fraccaro et al., 2016; Liu et al., 2017; Goyal et al., 2017; Buesing et al., 2018; Serban et al., 2017) fall into this framework.

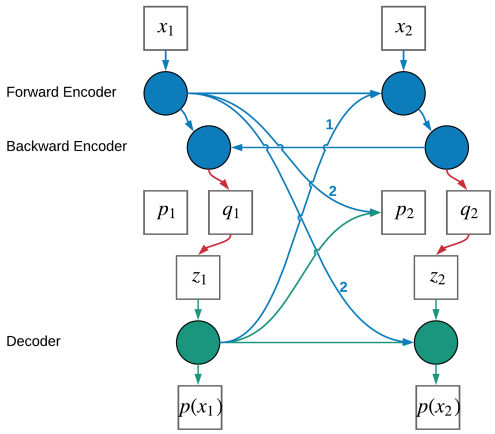


Figure 7: **Recurrent variational auto-encoder**. Quite general recurrent variational auto-encoder, obtained by imposing recurrent structure, forward sampling and allowing all the connections. Note that the encoder can have several alternating layers of forward and backward RNNs. Also note that the connection 1 has to be absent if backwards encoder is present (unless it is the first network). Possible skip connections are not shown as they can be implemented by the RNNs themselves. If connections 2 are absent, the model samples (rolls forward) in latent space only.

Now let us consider our desired properties. In order to sample forward in latent space, the encoder must not feed (deterministically) into the decoder or the prior of the latent variables (since we need the input to update the encoder and would therefore need the sampled input to roll forward). What is then the online belief state? Clearly, it must not receive an input from backwards RNN as that is not online. It also needs to have an unrestricted access to the input information — not pass only through information bottlenecks of latent variables — in order to form the best belief it can. This suggests the encoder RNN for the belief state. Now, if the decoder RNN is present, the information needed to predict the future could be stored there, and therefore it does not need to be stored in the encoder, causing the encoder to not be a state. This presents two options: include the decoder state in the belief state (together with the encoder state) or make the prior $p(z_{t+1} \mid z_t)$ and the reconstruction $p(x_t \mid z_t)$ depend only on z_t. This first case contradicts the next point.

We argue that smoothing, or the dependence of posterior on the future, is an important property that should be present. It is needed for interpretability of latent variables. As an example, imagine a box that can contain two items A and B and two time points: t_1 before opening the box, when we don't know the content of the box, and t_2 after opening it. We would want our latent variable to represent the content of the box. The perfect model of the content of the box is that the content doesn't change (the same object is in the box before and after opening it). Now imagine B is in the box. Our belief at t_2 is high for B but our belief at t_1 is uncertain. If we sample this belief at t_1 without considering t_2 we would sample A half of the time. However, then we would be learning a wrong model of the

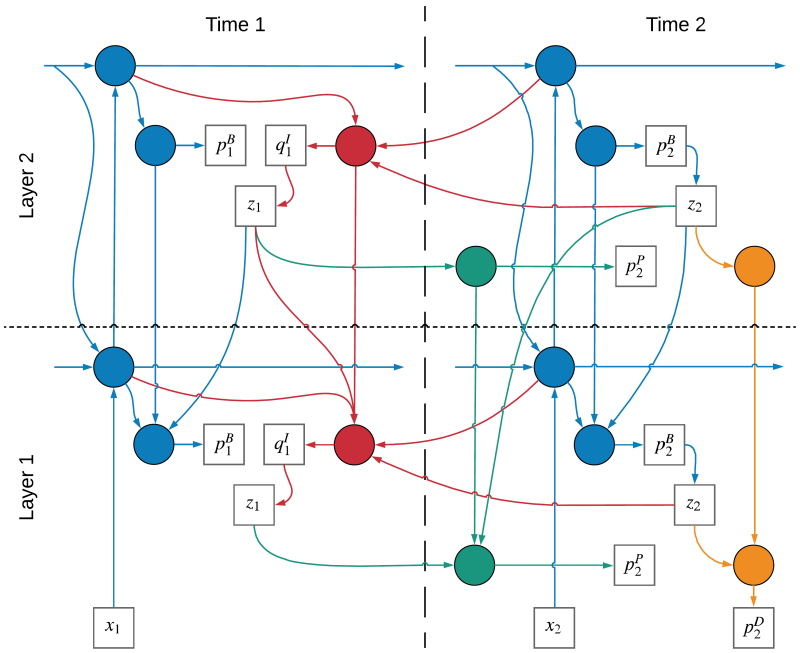


Figure 8: **Deep version of the model from Figure 1**. A deep version of the model is formed by creating a layer similar to the shallow model of Figure 1 and replicating it. Both sampling and inference proceed downwards through the layers. Circles have the same meaning as in Figure 1 and are implemented using neural networks, such as LSTMs.

world: that A goes to B. To solve this problem, we should sample t_2 first and then, given this value, sample t_1: the smoothing (backwards inference) operation.

If we want smoothing then (backwards encoder), we cannot let the decoder RNN be a part of our state, since it would depend on the future and not be online. Thus, we require a latent-to-latent model $p(z_{t+1} \mid z_t)$.

We are therefore left with the encoder RNN for state, some form of backwards function for smoothing and a latent-to-latent model. If the backwards function is in the inference, how do we make sure the encoder RNN contains a belief state? We can let the encoder RNN have a separate distribution expressing the (online) belief about the world $p_B(z_t \mid b_t)$. To make the encoder RNN represent a state, we can try to make $p_B(z_t \mid b_t)$ be close to $q(z_t \mid b_t)$ by imposing a KL term. However, normally there already is a KL term between p and q already. We would like to formulate a single, well-defined objective. That is precisely the TD-VAE algorithm presented in the paper.

B Hierarchical Model

In the main paper we detailed a framework for learning models by bridging two temporally separated time points. It would be desirable to model the world with different hierarchies of state, the higher-level states predicting the same-level or lower-level states, and ideally representing more invariant or abstract information. In this section we describe a stacked (hierarchical) version of the model.

The first part to extend to L layers is the RNN that aggregates observations to produce the belief state b. Here we simply use a deep LSTM, but with layer l receiving inputs also from layer $l+1$ from the previous time step. This is so that the higher layers can influence the lower ones (and vice versa). For $l = 1, \ldots, L$:

$$b_t^l = \mathrm{RNN}(b_t^l, b_t^{l-1}, b_{t-1}^{l+1}, x_t) \tag{7}$$

and setting $b_0 = b_L$ and $b_{L+1} = \emptyset$.

We create a deep version of the belief part of the model by stacking the shallow one, as shown in Figure 8. In the usual spirit of deep directed models, the model samples downwards, generating higher level representations before the lower level ones (closer to pixels). The model implements deep inference, that is, the posterior distribution of one layer depends on the samples from the posterior distribution in previously sampled layers. The order of inference is a choice, and we use the same direction as that of generation, from higher to lower layers, as done for example by Gregor et al. (2016); Kingma et al. (2016); Rasmus et al. (2015). We implement the dependence of various distributions on latent variables sampled so far using a recurrent neural network that summarizes all such variables (in a given group of distributions). We don't share the weights between different layers. Given these choices, we can allow all connections consistent with the model. Next we describe the functional forms used in our model.

B.1 Functional Forms

For the model trained on a 1D signal coming from the harmonic oscillator we use the following functional forms. The aggregation RNN is an LSTM. The belief network is a one-layer MLP with tanh nonlinearity:

$$q_{\text{param}}(x) = W \tanh(Wx + c) + c \tag{8}$$

where $q_{\text{param}}(x)$ is a vector that is the concatenation of the mean and log variance of a Gaussian distribution.

For the smoothing model and the forward model, we use the following function:

$$p_{\text{param}}(x, dt) = W(\tanh((Wdt)(Wx) + c) \cdot \sigma((Wdt)(Wx) + c)) + c \tag{9}$$

where dt is the time separation, and the multiplication between Wdt and Wx is point-wise.

For the decoder we simply extract the first component of z for the parameter of the mean of the observation and use a learned bias as the variance. This simple form stabilizes the training.

For the MNIST experiments we used standard MLPs for all these distributions and an MLP for the decoder. However, we recently found that the above functions forms for the posterior and model work even better.

For the DeepMind Lab experiments, all the circles in Figure 3 are LSTMs. For the rightward moving blue circles, we use a fully connected LSTM and for the remaining ones we use convolutional LSTMs. Convolutional LSTM updates are the same as fully connected ones, except (1) the states are three-dimensional (or four-dimensional if we include the batch dimension), having two spatial plus one feature dimensions, (2) the weight operators are convolutional from spatial layers to spatial layers, and fully connected followed by broadcasting from one-dimensional to three-dimensional layers.

We use a fully connected LSTM of size 512 and convolutional layers of size $4 \times 4 \times 256$. All kernel sizes are 3×3. The decoder layer has an extra canvas layer, similar to DRAW.