# Temporal Difference Variational Auto-Encoder

**Karol Gregor**
DeepMind
karolg@google.com

**Frederic Besse**
DeepMind
fbesse@google.com

## Abstract

One motivation for learning generative models of environments is to use them as simulators for model-based reinforcement learning. Yet, it is intuitively clear that when time horizons are long, rolling out single step transitions is inefficient and often prohibitive. In this paper, we propose a generative model that learns state representations containing explicit beliefs about *states* several time steps in the future and that can be rolled out directly in these states without executing single step transitions. The model is trained on pairs of temporally separated time points, using an analogue of temporal difference learning used in reinforcement learning, taking the belief about possible futures at one time point as a bootstrap for training the belief at an earlier time. While we focus purely on the study of the model rather than its use in reinforcement learning, the model architecture we design respects agents' constraints as it builds the representation online.

## 1 Introduction

Having a model of the environment allows us to consider consequences of different actions without acting in the real environment and to formulate a plan (Browne et al., 2012; Silver et al., 2016; Betts, 1998; Bertsekas et al., 1995) or simply learn from simulated experience (Sutton, 1991). In a general agent setting, when a model is not available, the usual approach is to learn one and use one of the model-based methods, such as ones referenced above. This has proven difficult but has had some success on relatively simple domains (Deisenroth & Rasmussen, 2011; Racanière et al., 2017; Buesing et al., 2018; Lee et al., 2018; Ha & Schmidhuber, 2018).

Planning can be most successful if the typical time separation of the events over which one plans is not much smaller than the time scale of the task. For example in the classical planning of air transport, one plans over events such as loading cargo and flying to another city, not in terms of one-second time intervals. Yet, most environment model research has focused on building accurate simulators of environments (reviewed below). It seems that we need a different kind of model: a model that can bridge large time scales in one, or a small number of computational steps (see some approaches below). What are the properties that such models should satisfy? This question won't be settled until these models are successfully used in planning, which is a large research topic in itself. To make progress and to be able to focus on model building, we propose a set of properties that are desirable for models used by agents. Then, we build a model that satisfies them and test it in experiments.

We first review some approaches to sequence modelling which include environment simulators. One of the most straightforward methods is to use a recurrent network to make a deterministic prediction of the next frame (Oh et al., 2015; Chiappa et al., 2017). However, most environments are complex and/or stochastic and require stochastic modelling. The most direct method is an extension of the previous ones, conditioning various types of generative models on the past frames (Graves, 2013; Kalchbrenner et al., 2016; Van Den Oord et al., 2016). Finally, another approach is to build a latent variable model that can make decisions in latent space, processing them using a neural network to generate observations (Chung et al., 2015; Lee et al., 2018; Archer et al., 2015; Fraccaro et al., 2016; Liu et al., 2017; Buesing et al., 2018; Serban et al., 2017; Oord et al., 2017). Depending on the

Preprint. Work in progress.

architecture, some of these models are able to roll forward in latent space, and only generate pixels if needed for some purpose such as visualization. In these models the recurrent computations are carried out at a rate of one time step per input time step. Some works consider skipping a part of the computation for a number of time steps or various forms of directly predicting future more than one step ahead (Koutnik et al., 2014; Chung et al., 2016; Zheng et al., 2016; Venkatraman et al., 2017). Our model differs from these in significant aspects. First, it does not consider fixed time skips but has a belief about future states at flexible time separations. Second, it can roll forward in these states. Finally, it is able to learn from two time points without needing to back-propagate between them. The model satisfies the list of proposed desired properties, listed below:

**1. The model should be able to make predictions about *states* many time steps into the future and roll forward samples of these states, without considering single step transitions.** Consider a task that requires planning about a set of events $s_1, \ldots, s_n$. If these events are separated by large time delays, a single step planner is significantly more expensive and inefficient than a planner that plans directly in terms of these events. Crucially, the model should predict states, not observations. States contain information not present in a given input such as information about motion, a set of objects an agent currently has, or the order in which past events happened.

**2. The model should be able to learn from temporally separated time points without the need to back-propagate through all the time points between them.** This could have several advantages: The separation could be large and back-propagating would be expensive in terms of computational speed and memory and difficult due to the vanishing gradient problem. Bridging different time points directly could also allow many such pairs to be processed in a short amount of time. Finally, such training can allow for independence of prediction time scale from the underlying environment/data temporal step size.

**3. The model should be deep and layered.** The basic principle behind deep learning is not only to execute several computations in sequence (as opposed to shallow computation of one or two steps), but also to create an operation - a layer - that is replicated many times in depth. That is, rather then designing these operations by hand, we create one operation and replicate it, and let the system learn to make this operation represent different things in different layers. For example in convolutional networks, such operation is the triplet of convolution, batch normalization and the rectified-linear non-linearity.

**4. The model needs to build state representations from observations online.** This is required because an agent should use the representations to take actions.

**5. The state representation should contain explicit beliefs about various parts of the environment and in a way that is easy to access.** For example when moving around a room, the model should have beliefs about parts of the room it has not seen, or how what it has seen has changed. Extracting these beliefs should not require long computation such as long forward rolls from the model.

**6. To build a state and to act the model should be able to make the best use of the available information. Therefore there should be a deterministic pathway from observations to states.** A simple example of this is the Hidden Markov Model, which models a stochastic process, but the parameters of which, representing probabilities of different states, are obtained deterministically from observations.

## 2 Temporal difference variational auto-encoder (td-VAE).

Here we explain the single layer version of the model. It consists of two parts: an aggregation network and a belief network. The aggregation network is the computation that builds the state that the agent can use to act. The belief network represents explicit beliefs about the state of the world and provides the training signal for itself and for the aggregation network.

The aggregation network we use is simply a recurrent neural network (RNN) that consumes observations (including actions and rewards if present) and the state that the agent uses to act is the internal state of the RNN. Mathematically, the state $s_t$ at time $t$ is computed from the previous state $s_{t-1}$ and the input $x_t$ at $t$ and previous action $a_{t-1}$ and reward $r_{t-1}$ as $s_t = \text{RNN}(s_{t-1}, x_t, a_{t-1}, r_{t-1})$. In practice, if the observations are images, they are often processed using a convolutional network followed by an LSTM. From now on we will omit actions and rewards unless otherwise stated.

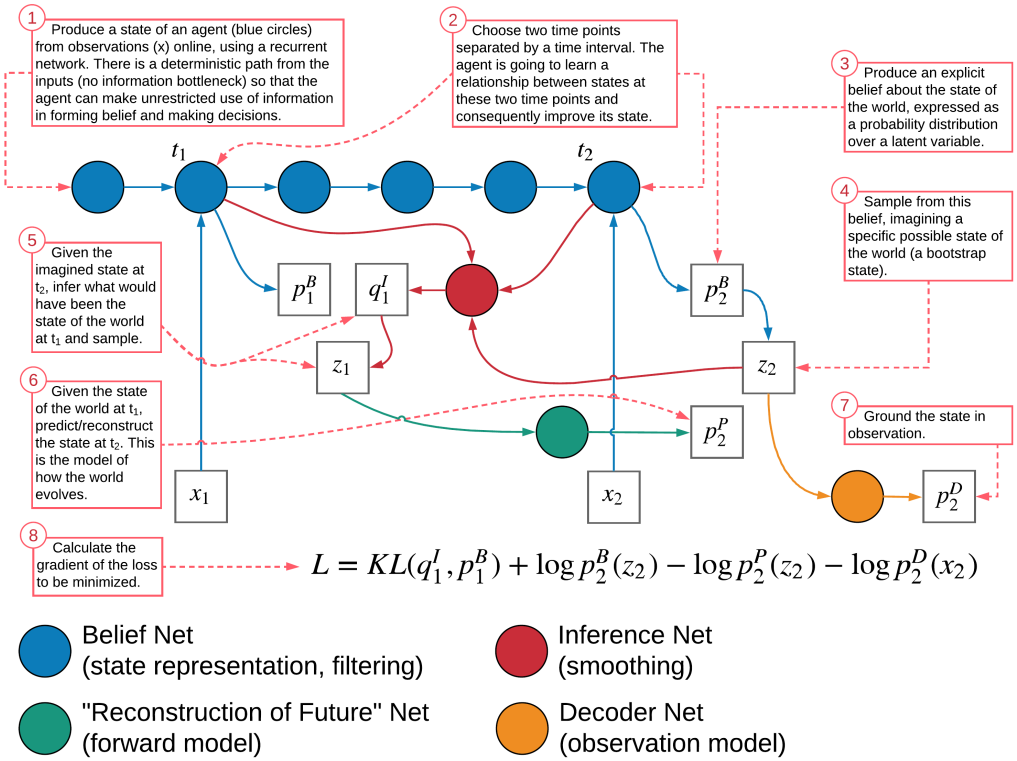


Figure 1: **A diagram of td-VAE.** Follow the numbers in the red circles for an explanation of the architecture.

Next, we build the belief network in a way that forces the state $s_t$ to contain beliefs about the world. We express these beliefs by a probability distribution $p^B(z|s)$ over a latent variable $z$. As an example, imagine that an agent enters a room and has seen only a part of it. The variable $z$ could represent the possible contents of the room. The elements of vector $z$ that represent what has been observed, or what is predicted from observations should have a low uncertainty under $p(z|s)$ and the remaining elements should have high uncertainty.

We are going to train the model by considering two time points $t_1$ and $t_2$ that are separated by a random amount from some range, for example $t_2 - t_1 \sim \text{Uniform}(1, 50)$. Except for the recurrent aggregation network above, we want all the updates to depend only on the variables at these two time points. The model is displayed and explained in Figure 1. We write its specifications here and derive them in the next paragraph. We let $s_1, z_1, s_2, z_2$ denote the state and latent variables at times $t_1, t_2$. The model is defined by the following equations.

$$s_t = \text{RNN}(s_{t-1}, x_t) \quad \text{Update online as the agent acts} \tag{1}$$

$$z_2 \sim p^B(z_2|s_2) \quad \text{Sample from belief at } t_2 \tag{2}$$

$$z_1 \sim q^I(z_1|s_1, s_2, z_2) \quad \text{Sample from approximate posterior at } t_1 \tag{3}$$

$$L = KL(q^I(z_1|s_1, s_2, z_2)|p^B(z_1|s_1)) + \log p^B(z_2|s_2) - \log p^P(z_2|z_1) - \log p^D(x_2|z_2) \tag{4}$$

Here $p^B$ denotes the agent's belief distribution, $p^P(z_2|z_1)$ is a reconstructive distribution of $z_2$ from $z_1$, $q^I$ is an inference distribution and $L$ is the loss that we minimize. $p^P(z_2|z_1)$ is at the same time a forward model of the environment that we can use to roll samples forward, skipping steps. Equations (1-4) form a type of a variational auto-encoder (**???**) as discussed below, which is trained using back-propagation as usual. Next we derive this model.

We aim for $p^B(z_t|s_t)$ to represent the model's belief about the world, having seen observations up to time $t$ (online). First, $p^B$ should represent the current frame, thus aiming to maximize $\log p^D(x_t|z_t)$ where $p^D$ is a 'decoder' distribution. Next, it should contain beliefs about the future *states*. Consider time $t_1$. We could theoretically get the most precise state of the world if we consider all the past

and the future frames that will be encountered and encode this information into $z_1$. There is still uncertainty left in our knowledge and therefore we can only have a belief over $z_1$, which we denote by $q^I(z_1|\text{all observations})$. If we had this distribution, we could train $p^B(z_1|s_1)$ to predict it (note that $p^B$ is computed online only from the observations up to time $t_1$). Now, in practice we can't consider all future inputs to compute this (and in addition it might be difficult to aggregate all the inputs that we do consider). Therefore we use our belief $p^B(z_2|s_2)$ at time $t_2$ to estimate the $q^I$. The belief $p^B(z_2|s_2)$ contains information about frames observed between $t_1$ and $t_2$ and also expresses the beliefs about what will be the future after $t_2$. We don't know which future will happen, and therefore we take a sample $z_2$ from this belief as a possible future. Now assuming this sampled future, we can infer what $z_1$ would have been using a function $q^I(z_1|s_1, s_2, z_2)$ and train the belief $p^B(z_1|s_1)$ to predict the belief $q^I$, that is, trying to minimize $KL(q^I(z_1|s_1, s_2, z_2), p^B(z_1|s_1))$. Now, in order for $z_1$ to represent a sample of the belief in the future, we train it to reconstruct this future. That is, assuming that $p^B(z_2|s_2)$ contains the belief about the state of the world at $t_2$, with $z_2$ being a specific instance of this belief, we can train $z_1$ to represent the corresponding state at time $t_1$ by trying to reconstruct $z_2$ from $z_1$. The reconstruction is expressed as distribution $p^P(z_2|z_1)$ with loss $\log p^B(z_2|s_2) - \log p^P(z_2|z_1)$ to be minimized. Adding these losses and the input reconstruction term gives us the loss (4).

### 2.1 Relationship to temporal difference learning of reinforcement 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 the belief of $R_t$ has been considered. To train the beliefs $V$ or $Q$, one does not usually wait to get all the rewards to compute $R_{t_1}$. Instead one uses a belief at some future time $t_2$ as a bootstrap to train $V$ or $Q$ at $t_1$ (temporal difference). In policy gradients one trains $V_{t_1}$ to be close to $\sum_{t=t_1}^{t_2-1} r_t\gamma^{t-t_1} + \gamma^{t_2-t_1}V_{t_2}$ and in Q learning one trains $Q_{t_1}(a)$ to be close to $r_{t_1} + \max_{a'} Q_{t_2=t_1+1}(a')$.

In our case, the model expresses belief $p^B(z_t|s_t)$ about possible future *states* instead of the *sum* of discounted *rewards*. The model trains the belief $p^B(z_{t_1}|s_{t_1}) = p^B(z_1|s_1)$ at time $t_1$ using beliefs $p^B(z_2|s_2)$ at various time points $t_2$ in the future. It accomplishes this by (variationally) auto-encoding a sample $z_2$ of the future state into a sample $z_1$, using the approximate posterior inference distribution $q^I$ and the decoding distribution $p^P$. The auto-encoding mapping translates between states at $t_1$ and $t_2$, forcing beliefs at the two time steps to be consistent. Sample $z_1$ forms the target for training the belief $p^B(z_1|s_1)$ (which appears as a prior distribution over $z_1$ in the language of VAE's).

### 2.2 Relationship to common forms of VAE and to filtering and smoothing of the field of state estimation.

Mathematically, td-VAE forms a two layer variational auto-encoder with the following structure (see Figure 1 of the supplement for a diagram). Variable $z_2$ is in the first layer and $z_1$ in the second. First, $z_2$ is sampled from the approximate posterior inference distribution $p^B(z_2|s_2)$, which depends on the observation $x_2$ through $s_2$. Next, $z_1$ is sampled from the second layer approximate posterior inference distribution $q^I(z_1|s_1, s_2, z_2)$. The prior for $z_1$ is $p^B(z_1|s_1)$. Variable $z_1$ is decoded into the space of $z_2$ using $p^P(z_2|z_1)$, which forms the prior over $z_2$. Variable $z_2$ is decoded into the input space using $p^D(x_2|z_2)$. The critical element that allows us to learn the state model from two temporally separated time points, is the use of $p^B$ as both the posterior at time $t_2$ and as a prior at time $t_1$. At $t_2$ it forms the (bootstrap) information about the world that is to be predicted from the information at time $t_1$.

In the language of state space estimation, $p^B$ forms a filtering distribution that depends only on the variables observed so far. The $q^I$ forms the smoothing distribution, correcting the belief at earlier times based on the future information. Finally $p^P$ is the forward model.

### 2.3 Experiment

We show that the model is able to learn the state and roll forward in jumps on the following simple example. We consider sequences of length 20 of images of MNIST digits. For a given sequence, a random digit from the dataset is chosen, as well as direction of movement: left or right. Then, at each time step in the sequence, the digit moves by one pixel in the chosen direction, as shown in Figure

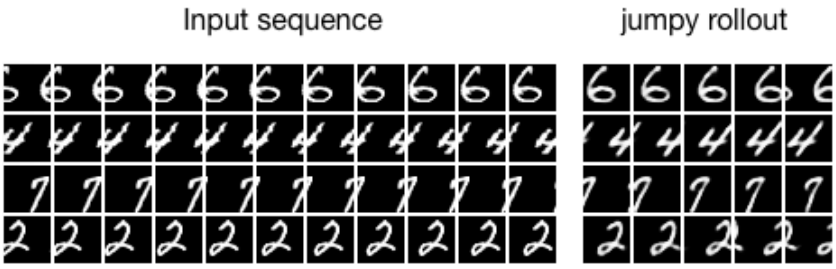


Figure 2: **Moving MNIST.** Left: Each row shows an example input sequence, consisting of a random MNIST digit, that is moving either left or right by one pixel. Right: The model was trained on pairs of points separated uniformly randomly between one and four time steps. Figure shows rollouts (sequential sampling) from the model. We see that the model is able to roll forward by skipping frames (the digits move by more then one pixel), keeping the correct digit and the direction of motion. The information about the motion is not present in the input, showing that the model is rolling forward states rather then observations.

2, left. 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 describe the functional forms we use below. We would like to see whether the model at a given time point can rollout a simulated experience in time steps $t_1 = t + \delta t_1$, $t_2 = t_1 + \delta t_2$, ... with $\delta t_1, \delta t_2, \ldots > 1$, without considering the inputs in between these time points. Note that it is not sufficient to predict the future inputs $x_{t_1}$, ... 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 sample a sequence from the model (do a rollout) as follows. State $s_t$ at the current time step $t$ is obtained online using the aggregation recurrent network from observations up to time $t$. Next, the belief $p^B(z_t|s_t)$ is computed and a sample $z_t$ from this belief is produced. This represents the current frame as well as beliefs about future states. The belief about the current input can be decoded from $z_t$ using the decoding distribution $p^D(x_t|z_t)$ and is generally very close to the input $x_t$. To produce the sequence, we sequentially sample $z \leftarrow z' \sim p^P(z'|z)$ starting with $z = z_t$. We decode each such $z$ using the decoder distribution $p^D(x|z)$. The resulting sequences are shown in the Figure 2, right. 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.

Why does the model do it? We train the model on two time points $t_1$ and $t_2$ ($t_1 < t_2$). The sample $z_1$ at $t_1$ tries to reconstruct the frame $x_1$ at $t_1$ and also reconstruct a future sample $z_2$. Therefore, it puts the information about the future into $z_1$ and thus $z_1$ effectively contains the motion information. The belief $p_1^B$ tries to predict this information based on its inputs $x_t$ up to including time $t_1$, which is rather easy for a recurrent network, and therefore $p_1^B$ will contain motion information. This is of course happening at every time step including $t_2$. That means that $z_2$ contains not only information about $x_2$ but also the motion information. Thus $z_1$ tried to reconstruct not only information about frame $x_2$ but also the motion information stored in $z_2$. Thus rolling forward from $z_1$ gives us a sample of the state at $t_2$, which can subsequently be rolled forward.

## 3 Deep Model

In the previous section we provided the framework for learning models by bridging two temporally separated time points. Experience and intuitive considerations suggest that for these models to be powerful, they need to be deep. We can of course use deep networks for various distributions used in the model. However, it would be more appealing to have layers of beliefs, ideally with higher layers representing more invariant or abstract information. In fact one of the principles of deep learning is not only to have several steps of computation, but to have a layer that is replicated several times to form a deep network. Here we form such deep network. In addition we suggest specific functional forms, derived from those that generally give rise to strong versions of variational auto-encoders.

The first part to extend to $L$ layers is the RNN that aggregates observations to produce the state $s$. Here we simply use a deep LSTM, but with a given 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

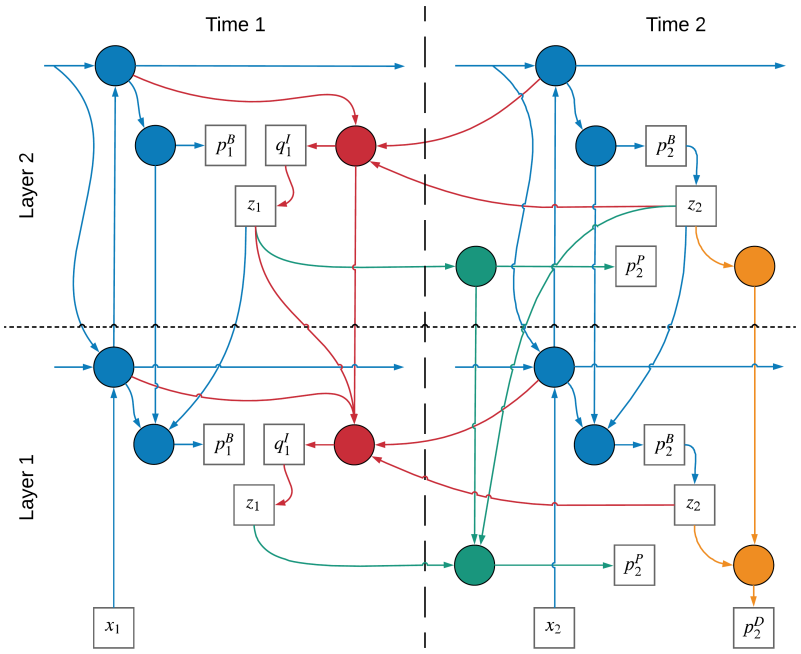


Figure 3: **Deep version of the model from the Figure 1.** A deep version of the model is formed by creating a layer similar to shallow model of Figure 1 and replicating it. Both sampling and inference proceed downwards through the layers. Circles have the same meaning as in the Figure 1 and are implemented using neural networks, such as LSTM.

versa). For $l = 1, \ldots, L$:

$$s_t^l = RNN(s_t^l, s_t^{l-1}, s_{t-1}^{l+1}, x_t) \tag{5}$$

and setting $s_0 = s_L$ and $s_{L+1} = \emptyset$.

We create a deep version of the belief part of the model as a stacking of the shallow one, as shown in Figure 3. 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 at 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 in 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). Note that we don't share the weights between different layers. Given these choices, we can allow all connections consistent with the model. The functional forms we used are described in the next subsection.

### 3.1 Experiments

In the first experiment we would like to demonstrate that the model can build a state even when very little information is present in a given frame and that it can sample states far into the future. For this we consider a one dimensional sequence obtained from a noisy harmonic oscillator, as shown in Figure 4 (the first and the 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. First, we would not know what is the frequency or the magnitude of the signal from this prediction. Furthermore, because the oscillator updates are noisy, the phase information is lost over such long period and it is even impossible to make an accurate prediction of the input. Instead we should try to predict as much as possible about the state, which consists of

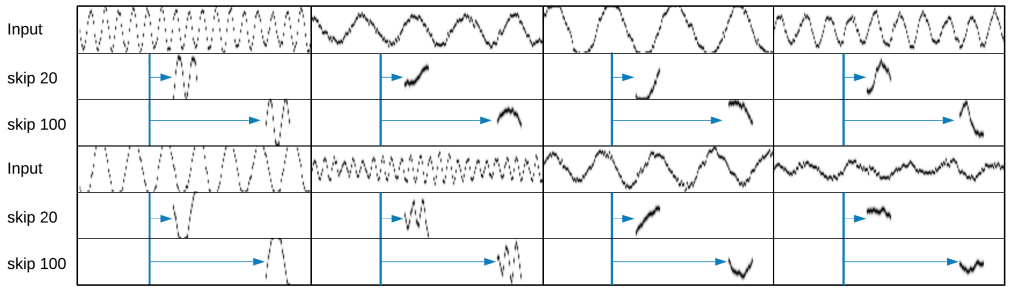


Figure 4: **Skip state prediction for one dimensional signal.** The input is one dimensional signal obtained from a noisy harmonic oscillator. We train the model on time separations from interval $[1, 120]$. The figure shows rollouts of first one step with time separation of 20 or 100, followed by 20 steps of separation one. We see that the model is able to create a state and predict it into the future, correctly predicting frequency and magnitude of the signal.

the frequency, the magnitude and the position, and it is only the position that cannot be accurately predicted.

The single step true model of the environment is a frequency dependent linear mapping between (position, velocity) vectors at consecutive time steps with added noise. For single frequency the single step update would be well modeled by the Kalman Filter. However, we are interested in a model that can skip time steps and work for all frequencies. We would also like to use the standard forms of variational auto-encoders which use diagonal variance Gaussian distributions. The correct belief about position and velocity has full covariance. To implement such belief, we need at least two layers of latent variables. The specific functional forms we use are in the supplement, but we summarize them briefly here. The aggregation RNN is an LSTM. There are two layers, the $z$'s at the second layer are sampled first, and their results are passed to the lower layer networks. The $p^B$, $q^I$ and $p^P$ 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 $q^I$ and $p^P$. We train on sequences of length 200, with $t_2 - t_1$ taking values chosen at random from interval $[1, 120]$.

We would like to see if the model can correctly predict future states directly. We analyze what the model has learned as follows. We pick time $t_1 = 60$ and sample $z_1 \sim p_1^B = p^B(z_1|s_1)$ from the belief distribution at $t_1$. Then we choose a time interval $dt$ to skip - we choose two intervals for demonstration, of length 20 and 100. We sample from the forward model $p^P(z_2|z_1, dt)$ to obtain sample $z_2$ at $t_2 = t_1 + dt$. To see the content of this state, we then roll forward repeatedly 20 times with with time step $dt_2 = 1$ and plot the result. This is shown in Figure 4. We see that indeed the state $z_2$ is predicted correctly, containing 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).

In the final experiment we would like to analyze the model on a more visually complex domain. We use sequences of frames seen by an agent solving tasks in DeepMind Lab environment (Beattie et al., 2016). We would like to demonstrate that the model holds explicit beliefs about various possible future states (desired property number 5 of introduction) and that it is able to make a rollout in jumps. We suggest functional forms inspired by a strong version of VAE, namely convolutional DRAW: we use convolutional LSTM's for all the circles in Figure 3 and make the model 16 layers deep (except for the forward updating LSTM's which are fully connected of depth 4. Figure 3 shows two layers.).

We consider time skips $t_2 - t_1$ sampled uniformly from interval $(1, 40)$ and analyze the content of the belief $p^B$, Figure 5. We take three samples $z_1, z_2, z_3$ from $p^B$ and look for similarities and differences. These samples should represent three instances of possible futures. First, to see what they represent about the current time, we decode them using $p^D$, and display the result in the Figure 5, left. We see that they decode to approximately the same frame. To see what they represent about the future, we make fives samples from $z_i^k \sim p^P(\hat{z}|z)$, $k = 1, \ldots, 5$ and decode them, as shown in Figure 5, right. We see that for a given $i$, the decoded samples (images in a given row) decode to a similar frames. However $z's$ for different $i$'s decode to different frames. That means the $p^B$ represents beliefs about different possible futures directly.

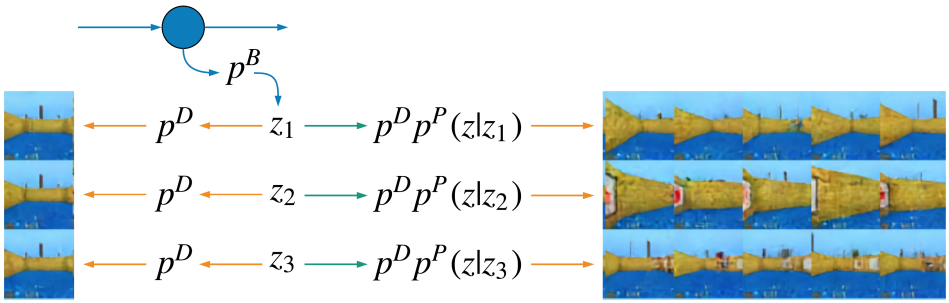


Figure 5: **Beliefs of the model.** We obtain three samples $z_1, z_2, z_3$ from the belief distribution $p^B$ at time $t$ to examine its content. The images on the left are decodings from $z_1, z_2, z_3$, showing that all three samples decode to roughly the same frame. Next, for each of the three samples, we take 5 samples $\hat{z}_{k_i} \sim p^P(\hat{z}|z_i)$, $k = 1, \ldots, 5$ and decode each, obtaining samples on the right. For a given $i$, corresponding to a row, the decodings are similar, but are different for different $i$. This implies that the $z_i$'s, despite decoding to the same frame at $t$, represent different instances of future states. This in turn implies that $p^B$ contains beliefs about different possible futures states.

Finally, we would like to see what a rollout looks like. To get a visual sense of it, we train on time separations $t_2 - t_1$ chosen uniformly from interval $[1, 5]$ on a task where the agent tends to move forward and rotate. Figure 6 shows four 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).

Figure 6: **Rollout from the model.** The model was trained on steps uniformly distributed between one and five. We see that the model is able to create forward motion that skips several real time steps.

## 4 Conclusions

In this paper we argued that an agent likely needs a model that is different from an accurate step by step environment simulator. We discussed the nature of such model and presented a variational auto-encoder based version that satisfies a set of useful properties. Our model builds states from observations by bridging time points separated by a random intervals. This allows the model to build states that relate to each other directly over longer time stretches, that contain explicit beliefs about future states and that can consequently allow the model to roll forward these states in time steps larger than, and potentially independent of, the underlying temporal environment/data step size. We proposed a deep layered version of the model and the corresponding functional forms. We demonstrated a number of desired properties of the model in proof of principle experiments, from simple one dimensional signal, to complex visual input obtained from a three dimensional simulated environment. In the future we hope to improve the model and apply it in more difficult settings, and investigate a number of possible uses in reinforcement learning such are representation learning and planning.

## References


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.

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.

Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 1995.

John T Betts. Survey of numerical methods for trajectory optimization. *Journal of guidance, control, and dynamics*, 21(2):193–207, 1998.

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

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.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472, 2011.

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.

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.

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.

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

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

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.

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.

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.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991.

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

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*, 2016.

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.

Stephan Zheng, Yisong Yue, and Jennifer Hobbs. Generating long-term trajectories using deep hierarchical networks. In *Advances in Neural Information Processing Systems*, pp. 1543–1551, 2016.

# Supplement

## 5 td-VAE as a two layer VAE

Figure 7 shows td-VAE in a format of two layer VAE. The explanation is in the caption.

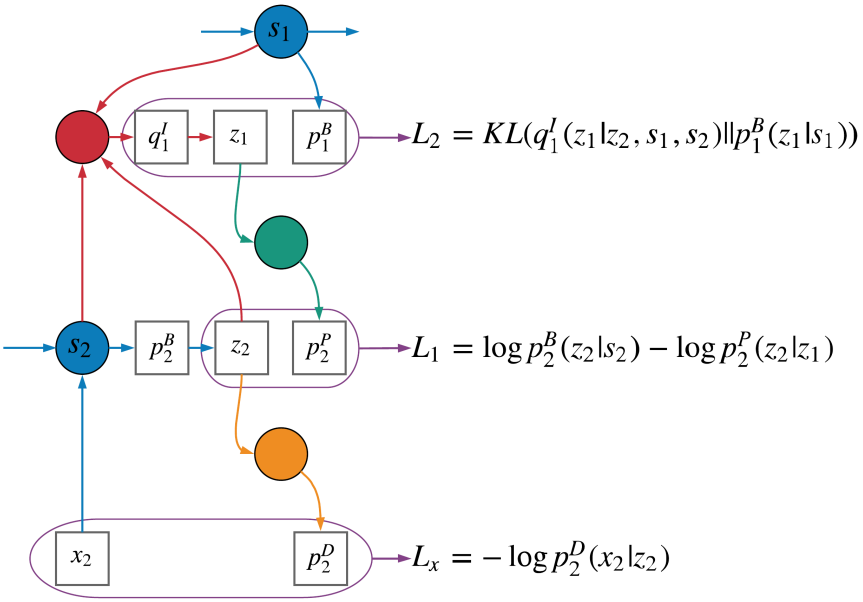


Figure 7: **A diagram of td-VAE of Figure 1 of the main text, drawn in a format more standard for (2 layer) VAE's.** The td-VAE forms a two layer VAE, with $z_2$ in what is commonly referred to as the first layer and $z_1$ in the second layer. The inputs $x_{<t_2}$, and the states $s_t$ at points other then $t_1$ and $t_2$ are not shown for simplicity, but the blue circles form a recurrent network, each receiving input $x_t$ and time $t$. During inference $x_2$ is passed to $s_2$ which computes the approximate posterior $p_2^B$ over $z_2$. The $z_2$ along with other variables is then used to compute the approximate posterior $q_1^I$ over $z_1$. Variable $z_1$ is then decoded to compute the prior $p_2^B$ over $z_2$. Variable $z_2$ is decoded to compute reconstruction distribution $p_2^D$. The losses are standard as follows. For the latent variables they are the differences between logarithms of posterior and prior and for the input the loss is the logarithm of $p^D$. The loss of layer containing $z_1$ is on average the $KL$ divergence displayed, which is a lower variance version of $\log q_1^I - \log p_1^B$. We cannot use the same approach for layer containing $z_2$ because the prior of $z_2$ depends on $z_2$ itself (which can be seen by writing down the math).

## 6 td-VAE functional forms used for MNIST experiment.

Here we explain the computations for the shallow model of td-VAE. The aggregation RNN is an LSTM that consumes observations $x_t$ and updates state $s_t$. The updates are standard

$$
\begin{aligned}
s_t &= \text{LSTM}(x_{t-1}, s_{t-1}): && (6)\\
h_{t-1}, c_{t-1} &= s_{t-1} && (7)\\
v &= [x_{t-1}, h_{t-1}] && (8)\\
b_i &= \sigma(Wv + b) && (9)\\
b_f &= \sigma(Wv + b) && (10)\\
b_o &= \sigma(Wv + b) && (11)\\
u &= \tanh(Wv + b) && (12)\\
c_t &= g_f c_{t-1} + g_i u && (13)\\
h_t &= g_o \tanh(c_t) && (14)\\
s_t &= (h_t, c_t) && (15)
\end{aligned}
$$

Where square bracket denotes concatenation and $W$'s and $b$'s are weights and biases. We use the same letter for all for simplicity, but they all have different values. We will use the same notation for the rest of the supplement.

All the remaining distributions $p^B$, $q^I$, $p^P$ and $p^D$ are the same functions (with different weights): One layer MLP with rectified linear units:

$$q_{param}(x) = W\text{Relu}(Wx + b) + b \tag{16}$$

where $q_{\text{param}}$ is a vector that is a concatenation of the mean and logarithm of variance of Gaussian distribution. If the there are several inputs, they are concatenated.

## 7 Deep model mathematical definition.

We write the computations defining the deep model, Figure 3 of the text. The computations at a given layer $l$ ($l = N, \ldots, 1$) are as follows:

$$\begin{aligned}
u_2^{B,l} &= f(u_2^{B,l+1}, z_2^{l+1}, s_2^l, x_2) && (17)\\
p_2^{B,l} &= g(u_2^{B,l}) && (18)\\
z_2^l &\sim p_2^{B,l} && (19)\\
u_1^{B,l} &= f(u_1^{B,l+1}, z_1^{l+1}, s_1^l, x_1) && (20)\\
p_1^{B,l} &= g(u_1^{B,l}) && (21)\\
u^{I,l} &= f(u^{I,l+1}, z_2^{l+1}, s_1^l, s_2^l, x_1, u_1^{B,l}) && (22)\\
q^{I,l} &= g(u^{I,l}) && (23)\\
z_1^l &\sim q^{I,l} && (24)\\
u^{P,l} &= f(u^{P,l+1}, z_1^l, z_2^{l+1}) && (25)\\
p^P &= g(u^{P,l}) && (26)\\
u^{D,l} &= f(u^{D,l+1}, z_2^{l+1}) && (27)\\
L_1^l &= \text{KL}(q^I, p_1^B) && (28)\\
L_2^l &= \log p_2^B(z_2^l) - \log p^P(z_2^l) && (29)\\
L^l &= L_1^l + L_2^l && (30)
\end{aligned}$$

Here $f, g$ are functions with internal weights, that are different for each instance and for each layer and where $g$ outputs probability distribution. The top states $u^{L+1}$ are initialized as learned biases and $z^{L+1}$ are absent.

After all the layers are executed, the reconstruction loss is computed and all the losses are added together

$$\begin{aligned}
L^D &= -\log p^D(x_2|u^{D,0}) && (31)\\
L &= L^D + \sum_l L^l && (32)
\end{aligned}$$

which is the loss that we minimize.

These are the computations which we use for the DeepMind Lab experiments, and can be done sequentially executing each layer before proceeding to the next one. However more generally, we first compute the $p_2^B$, $z_2$ in all the layers first, then, proceed to all the computations at time $t_1$ and making them dependent on $z_2$'s in all the layers, and then proceed to compute $p_2^P$, having it depend on all $z_1$'s. This is the setting we use for the harmonic oscillator.

## 8 Deep model functional forms.

For the model trained on one dimensional coming from the harmonic oscillator we use the following function forms. The aggregation RNN is an LSTM. The belief net is one layer MLP with tanh nonlinearity

$$q_{param}(x) = W \tanh(Wx + b) + b \tag{33}$$

where $q_{param}$ is a vector that is a concatenation of the mean and logarithm of variance of Gaussian distribution.

For the inference distribution $q^I$ and the forward model distribution $p^P$, we use the following function

$$p_{param}(x, dt) = W(\tanh((Wdt)(Wx) + b).\sigma((Wdt)(Wx) + b)) + b \tag{34}$$

where $dt$ is the time separation between the points, 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 learned bias as the variance. This simple form stabilizes the training.

For the DeepMind Lab experiments, all the circles in Figure 3 are LSTM's. For the rightward moving blue circles, we use fully connected LSTM and for the remaining one we use convolutional ones. Convolutional LSTM updates are the same as fully connected ones above, except: 1. The states are three dimensional (or four if we include the batch dimension): spatial plus 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 fully connected LSTM of size $512$ and convolutional layers of size $4 \times 4 \times 256$. All kernel sizes are $3 \times 3$. The decoder layer has an extra canvas layer, similar to DRAW.