Learning Multimodal Transition Dynamics for Model-Based Reinforcement Learning

Thomas M. Moerland, Joost Broekens, and Catholijn M. Jonker

Department of Computer Science
Delft University of Technology, The Netherlands
{T.M.Moerland,D.J.Broekens,C.M.Jonker}@tudelft.nl

Abstract. In this paper we study how to learn stochastic, multimodal transition dynamics in reinforcement learning (RL) tasks. We focus on evaluating transition function estimation, while we defer planning over this model to future work. Stochasticity is a fundamental property of many task environments. However, discriminative function approximators have difficulty estimating multimodal stochasticity. In contrast, deep generative models do capture complex high-dimensional outcome distributions. First we discuss why, amongst such models, conditional variational inference (VI) is theoretically most appealing for model-based RL. Subsequently, we compare different VI models on their ability to learn complex stochasticity on simulated functions, as well as on a typical RL gridworld with multimodal dynamics. Results show VI successfully predicts multimodal outcomes, but also robustly ignores these for deterministic parts of the transition dynamics. In summary, we show a robust method to learn multimodal transitions using function approximation, which is a key preliminary for model-based RL in stochastic domains.

1 Introduction

Reinforcement learning (RL) is a succesful learning paradigm for sequential decision making from data in agents and robots. A long standing debate in RL research has been whether to learn 'model-free' or 'model-based' (Figure 1) [29]. Model-based RL has shown some important benefits, most noteworthy increased data efficiency [6], the potential for targeted exploration [28], and natural transfer between tasks when only the reward function changes. Model-based RL consists of two steps: 1) transition function estimation through supervised learning, and 2) (sample-based) planning over the learned model (Figure 1, green arrows). Each step has a particular challenging aspect. For this work we focus on a key challenge of the first step, *stochasticity* in the transition dynamics, while we defer the second step, planning (under uncertainty), to future work (see Sec. 6 as well).

Stochasticity is an inherent property of many environments, and increases in real-world settings due to sensor noise. Transition dynamics usually combine both deterministic aspects (such as the falling trajectory of an object due to gravity) and stochastic elements (such as the behaviour of another car on the

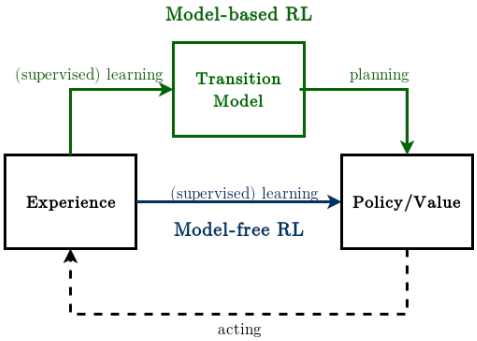

Fig. 1. Two types of reinforcement learning. *Model-free* RL (blue) directly learns a behavioural policy, while *model-based* RL (green) also attempts to learn the environment's transition dynamics. Learning this transition model allows the agent to predict future states and thereby allows the agent to *plan*.

road). Our goal is to learn to jointly predict these. Note that stochasticity has many forms, both homoscedastic versus heteroscedastic, and unimodal versus multimodal. In this work we specifically focus on multimodal stochasticity, as this should theoretically pose the largest challenge.

To learn such transition models, we require high-capacity function approximators that can predict next-state distributions of complex shape. This problem is not yet accurately solved by currently used methods in model-based RL, like tabular learning [2] (which does not scale to high-dimensions), linear function approximation [1] with Gaussian noise [18], random forests [12], or deep feed-forward networks trained on mean-squared error (MSE) [23] (See also Sec. 2).

A potential solution to this problem are Deep Generative Models (DGM) [11], as they are non-linear function approximators that can learn complex outcome distributions and scale to high-dimensions. In Section 2 we compare different DGM's on their theoretic appeal for stochastic model learning in the RL setting, and identify conditional Variational Inference (VI) as the most promising solution.

The remainder of the paper then continues as follows. In Section 3, we formally describe conditional variational inference with different types of discrete and continuous latent variables. In Section 4 we empirically compare the different approaches on a simulated function and on a typical RL tasks. Our results show that VI is accurately able to discriminate deterministic from stochastic aspects of the transition dynamics. We also show how the RL agent manages to learn an accurate transition model while solving a task. Finally, Sections 5 and 6 connect our work to related literature and identify opportunities for future work, respectively.

All code to reproduce the results in this paper is publicly available at `www.github.com/tmoer/multimodal_varinf`.

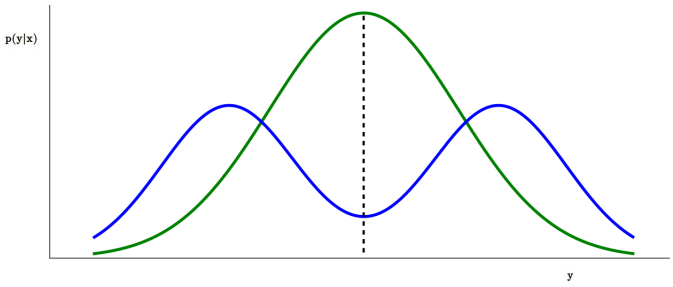

Fig. 2. Multimodal outcome distribution $p(y|x)$ (blue line) for a simple 1D observation space. Training on a mean-squared error will deterministically predict the conditional mean (dashed line), which implicitly assumes a uni-modal Gaussian outcome (green line).

2 Challenge of Multimodal Transitions

We will write $x \in \mathcal{X}$ for the current state and action, and $y \in \mathcal{Y}$ for the next state we want to predict. We are interested in models that can approximate distributions $p(y|x)$ with multiple local maxima ('modes'). A cardinal example of such a distribution is shown in Figure 2 (blue line).

Discriminative function approximators, for example trained at mean-squared error (MSE) loss, fail at this task. The point prediction of the function approximator will be the conditional expectation of the outcome distribution (Fig. 2, dashed line). Obviously, point estimate predictions will never be a good method to approximate a distribution, but they have actually been frequently applied in model-based RL work [23].[1] Clearly, a unimodal outcome distribution will neither solve the multimodality problem (Fig. 2).

A mixture of Gaussians per outcome dimension would neither solve the problem. Full covariance matrix Gaussians clearly do not scale to high-dimensional domains (such as [22]), while diagonal Gaussians would loose all covariance structure in the predictions. Moreover, mixture models are tedious to train. What we require are models that 1) flexibly approximate joint distributions of complex (multimodal) shape and 2) scale to high-dimensions.

We hypothesize the group of deep generative models (DGN) [11] are a promising candidate, as they fulfill both requirements. For model-based RL, where we will use the models to sample (a lot of) traces, we additionally require that the model is 1) easy to sample from, and 2) ideally allows for planning at an abstract level. Following the DGN taxonomy by Goodfellow [11] (Figure 3), we

[1] For example, Oh et al. [23] shows MSE training does work well in high-dimensional, deterministic domains. However, for example inspecting their Ms. Pacman (a stochastic game) predictions at `https://youtu.be/cy96rtUdBuE`, we see that the predictions for the stochastic elements (ghosts) fail. The ghosts disappear when they reach a corridor junction, where they stochastically choose in which direction to continue. The feed-forward network predicts the conditional mean of these choices, which completely blurs the ghosts in a few frames.

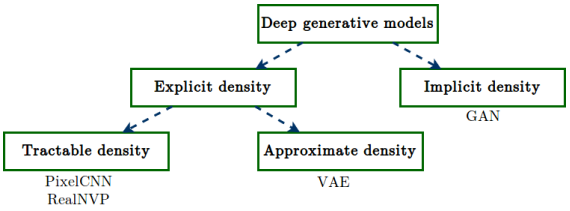

Fig. 3. Deep generative model taxonomy following [11].

now compare DGN models on their theoretical appeal for transition function estimation.

Implicit density models, like Generative Adverserial Networks (GAN) lack a clear probabilistic objective function, which was the focus of this work. Among the explicit density models, there are two categories. *Change of variable formula* models, like Real NVP [8], have the drawback that the latent space dimension must equal the observation space. Fully visible belief nets like pixelCNN [24], which factorize the likelihood in an auto-regressive fashion, hold state-of-the-art likelihood results. However, they have the drawback that sampling is a sequential operation (e.g. pixel-by-pixel, which is computationally expensive), and they do not allow for latent level planning either. Therefore, most suitable for model-based RL seem approximate density models, most noteworthy the variational auto-encoder (VAE) [17] framework. These models can estimate stochasticity at a latent level, allow for latent planning [31], are easy to sample from, and have a clear probabilistic interpretation. In the next section, we will formally introduce this methodology in the conditional setting, where the generative process of y is conditioned on other variables x.

3 Conditional Variational Inference

We will first introduce the conditional variational auto-encoder (CVAE) [26]. Our goal is to learn a generative model of a (possibly multimodal) distribution $p(y|x)$. We assume the generative process is actually conditioned on latent variables z:

$$p(y|x) = \int p(y|z,x)p(z|x)dz \tag{1}$$

Here $p(z|x)$ is the *prior* and $p(y|z,x)$ is the *generative model* or 'decoder'. The stochastic latent variables z provide the flexibility to predict more complex outcome distributions y. The posterior over z, $p(z|y,x)$ is intractable in most models of interest, for example deep non-linear neural networks. However, the parameters of this distribution can be efficiently approximated with Stochastic Gradient Variation Bayes (SGVB) [17], which uses a parametric recognition or *inference model* $q(z|y,x)$ to approximate the true posterior $p(z|y,x)$. The inference model learns a mapping from observations to latent space, providing generalization and thereby amortizing the cost of inference (compared to Markov chain Monte Carlo (MCMC) inference methods that needed computationally expensive iterative procedures to estimate the latent variables per datapoint).

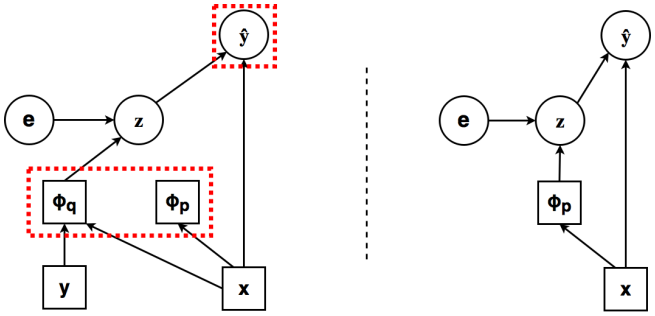

Fig. 4. Conditional Variational Auto-Encoder as a computational process. Squares are deterministic, circles are probabilistic nodes. **Left**: Training procedure. During training, we sample z according to $q(\cdot|x, y)$, where q is parametrized by ϕ_q. The training loss consists of two terms (indicated by the red dotted boxes): 1) the reconstruction loss $p(y|z, x)$, and 2) the KL-divergence between $q(z|x, y)$ and $p(z|x)$. The latter ensures that the posterior q puts probability mass at the same points as the prior p, effectively acting as a regularizer in latent space. We compute z with the reparametrization trick, where $\mathbf{e}$ can be any appropriate noise distribution. **Right**: Test procedure. At test time, we cut away the inference network q, and instead sample z according to the prior $p(z|x)$. This allows us to make stochastic predictions for y.

We can derive a variational lower bound $\mathcal{L}(y|x)$ on our data likelihood $p(y|x)$:

$$\begin{aligned}\log p(y|x) &\geq \mathbb{E}_{z\sim q(z|x,y)}\left[\log \frac{p(y,z|x)}{q(z|y,x)}\right] \\ &= \mathbb{E}_{z\sim q(z|x,y)}[\log p(y|z,x)] - D_{\mathrm{KL}}[q(z|x,y)\|p(z|x)] = \mathcal{L}(y|x;\theta,\phi) \quad (2)\end{aligned}$$

where θ denotes the parameters in the generative network, ϕ denotes the parameters in the inference network and prior, and D_{KL} denotes the Kullback-Leibler (KL) divergence. We can interpret the left-hand term of the last equation ($\log p(y|z, x)$) as the negative *reconstruction error*, which measures how well we reconstruct y after sampling z. The right-hand term (KL divergence) ensures q does not diverge too much from the prior p. This acts as a regularizer, and ensures that we can at test time (when we do not observe y) sample from $p(z|x)$ instead of $q(z|x, y)$.

In practice, we slightly modify the objective in Eq. 2, where we use importance sampling [3] to obtain a tighter bound, and minimize a different distance function instead of the KL-divergence (namely α-divergence with α=0.5 [7]). Details are provided in Appendix A.

3.1 Reparametrization

For this work we focus on variational methods that *reparametrize* the distribution of $q_\phi(z|y, x)$ to allow gradient-based training on a single computational graph. The trick works when we can write z as a function $z = f_\phi(\epsilon, y, x)$, with $f_\phi(\cdot)$

a deterministic, differentiable function, and $\epsilon \sim p(\epsilon)$ a noise distribution with independent marginal.

For a continuous variable z, the cardinal example is a location-scale transformation of a standard Gaussian distribution. If $q_\phi(z|y,x) = \mathcal{N}(z|\mu_\phi(y,x), \Sigma_\phi(y,x))$, then we can write

$$z = f_\phi(\epsilon, y, x) = \mu_\phi(y,x) + \Sigma_\phi(y,x) \cdot \epsilon, \quad \text{with} \quad \epsilon \sim \mathcal{N}(0,1) \tag{3}$$

The gain is that we can now backpropagate through expectations of the random variable z [17]:

$$\nabla_\phi \mathbb{E}_{z \sim q_\phi(z|x,y)}[\xi(z)] = \mathbb{E}_{\epsilon \sim \mathcal{N}(0,1)}\left[\nabla_\phi \xi(f_\phi(\epsilon, y, x))\right] \tag{4}$$

for some function $\xi(\cdot)$ of z. The right-hand term can then be approximated with a Monte-Carlo estimate. This allows us to backpropagate through z (see Fig. 4, left), giving the vanilla conditional variational auto-encoder (CVAE) with Gaussian latent variables. The overall train and test procedure is summarized in Figure 4. We now consider two methods to improve the capacity of the latent z distribution, that both could improve learning multimodal outcomes.

3.2 Discrete Latent Variables

As we want to model multimodal outcomes, it seems natural to consider discrete latent variables. However, for the reparametrization trick to be applicable we require the function $f_\phi(\cdot)$ to be differentiable, which is not possible for a discrete variable. It turns out we can get good estimates by making a smooth approximation to the discrete loss [15,20].

Let ω_i be an ordered set of class probabilities of a discrete variable z_i[2] with n_i categories. We can draw samples from this distribution through the Gumbel-Max trick:

$$z_i = \text{one-hot}\left(\underset{j \in [1..n_i]}{\arg\max}[g_j + \log \omega_{i,j}]\right) \tag{5}$$

with g_j i.i.d. draws from a Gumbel$(0,1)$ distribution[3]. Since arg max is not differentiable, we can make a softmax approximation to the above equation:

$$z_{i,j} = \frac{\exp((\log \omega_{i,j} + g_j)/\tau)}{\sum_{o=1}^{n_i} \exp((\log \omega_{i,o} + g_o)/\tau)} \qquad \text{for} \quad j = 1, .., n_i \tag{6}$$

which is known as the Gumbell-Softmax [15] or Concrete [20] distribution. The softmax temperature $\tau \in (0, \infty)$ regulates the discreteness of the approximation: for $\tau \to 0$, the samples effectively become one-hot, while for $\tau \to \infty$,

[2] We use subscripts z_i to index the elements of the vector random variable z, and double subscripts $z_{i,j}$ to index the categories within one discrete random variable.

[3] We can sample from a Gumbel(0,1) distribution by sampling $u \sim$ Uniform$(0,1)$ and computing $g = -\log(-\log(u))$.

the samples become uniform over the class categories. The above specification allows us to use the reparametrization trick for discrete latent variables, as the noise distribution g is now decoupled from the gradient path $\frac{\delta z}{\delta \omega}$. Note that Eq. 6 is a type of reparametrization $f_\phi(\cdot)$ (as introduced in Sec. 3.1, with g the noise distribution and $\omega_\phi(x, y)$ the distribution parameters. In practice, we anneal τ from > 1 to 0 over the course of training.

3.3 Transformations of Continuous Variables (Flow)

We already specified the reparametrization trick for spherical Gaussian latent variables (Eq. 3). As spherical Gaussians may be too restricting for multimodality, we can increase the capacity of the latent layer by using transformations of distributions for which we can track the density.

To obtain more expressive distributions for a continuous random variable $z \in \mathbb{R}^D$ with known density $q(z)$, we consider bijective smooth mappings $h : \mathbb{R}^D \to \mathbb{R}^D$ with inverse h^{-1}. We are interested in the distribution of the transformed variable $z' = h(z)$. As long as we are able to invert h, we can easily compute the density of the transformed variable z':

$$q(z') = q(z) \left| \det(\frac{\delta h^{-1}(z')}{\delta z'}) \right| = q(z) \left| \det(\frac{\delta h(z)}{\delta z}) \right|^{-1} \tag{7}$$

which is known as the *change-of-variable formula*. If we can specify our neural network to learn transformations which are easily invertible, we can construct more complicated distributions by repeatedly applying the above transformation (while being able to track the density). If we repeatedly apply a sequence of transformations $z^L = h^L \circ ... \circ h^1(z^0)$ for some random variable $z^0 \sim q^0(\cdot)$, then the density of the last variable z^L can be computed as:

$$\log q^L(z^L) = \log q^0(z^0) - \sum_{l=1}^{L} \log \left| \det \frac{\delta z^l}{\delta z^{l-1}} \right| \tag{8}$$

The problem with the above transformation is that, especially for high-dimensional domains, computing the determinant is computationally very expensive. An elegant solution appears from the observation that the determinant of a triangular matrix is simply the product of its diagonal terms [8] [16]. Therefore, given a random variable z of length D, we can specify the transformation $z' = h(z)$ as:

$$\begin{aligned} z'_{1:d} &= z_{1:d} \\ z'_{d+1:D} &= t(z_{1:d}) + z'_{d+1:D} \odot \exp(s(z_{1:d})) \end{aligned} \tag{9}$$

The Jacobian of the this transformation is:

$$\frac{\delta z'}{\delta z} = \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\delta z'_{d+1:D}}{\delta z_{1:d}} & \text{diag}(\exp(s(z_{1:d}))) \end{bmatrix} \tag{10}$$

The determinant of this matrix is easily computed as $\exp\left[\sum_i s(z_{1:d})_i)\right]$. Note that the $t(\cdot)$ (translation) and $s(\cdot)$ (scale) function can be arbitrarily complex functions, for example deep, non-linear neural networks. In these transformations, we do not need to compute the determinant of $s(\cdot)$ or $t(\cdot)$ to track the density of the random variable z'. Moreover, it is trivial to invert the above transformation:

$$\begin{aligned} z_{1:d} &= z'_{1:d} \\ z_{d+1:D} &= (z_{d+1:D} - t(z'_{1:d})) \odot \exp(-s(z'_{1:d})) \end{aligned} \tag{11}$$

This allows us to use the change-of-variable formula of the previous section. We effectively perform an auto-regressive transformation on the z variables. In practice, we repeatedly modify the order of the z variables to have a different part of z transformed in each layer. In Fig. 4, we would apply these transformations to a sample from $q(z|x, y)$ before calculating the KL-divergence with $p(z|x)$.

3.4 Enforcing Latent Variable Use

One of the challenges of training latent variable models is their tendency to overfit the prior early in training. Initially, the likelihood term $p(y|z, x)$ is relatively weak. Therefore, the learning signal is dominated by the KL-divergence, and stochastic optimization gets stuck in the undesirable equilibrium $q(z|y, x) \approx p(z|x)$.

To give a simple illustration, imagine y is strictly bimodal given a fixed x, taking value y_1 or y_2 with $p(y_1|x) = 0.3$ and $p(y_2|x) = 0.7$. We fit a latent model with a single binary variable z taking values z_1 or z_2. Clearly, we want our prior $p(z|x)$ to learn the distribution $\{p(z_1|x) = 0.3, p(z_2|x) = 0.7\}$ (assuming z_1 maps to y_1 and z_2 to y_2, which can of course be interchanged). However, the inference network $q(z|x, y)$ has access to additional information, as it knows which y we need to reconstruct. Therefore, if we present a datapair (x, y_1), then we want our latent distribution more like $\{q(z_1|x, y_1) = 0.999, q(z_2|x, y_1) = 0.001\}$, as this ensures we make a good draw and good reconstruction. However, for this datapoint this would incur a KL-penalty $D_{KL}(q(z|x, y_1)\|p(z|x)) \approx 1.20$. This illustrates how a good fitting VI model will necessarily incur some KL-cost.

A solution is to enforce each (set of) latent variables to encode a minimum amount of information [16], i.e. force $q(z|y, x)$ to at least have a KL-divergence of λ from the prior $p(z|x)$. The modified objective becomes:

$$\begin{aligned} \tilde{\mathcal{L}}(y|x) = \mathbb{E}_{(x,y)\sim\mathcal{M}}\Big[\mathbb{E}_{z\sim q(z|y,x)}\big[\log P(y|z, x)\big]\Big] - \\ \sum_{j=1}^{D_z} \max\Big(\lambda, \mathbb{E}_{(x,y)\sim\mathcal{M}}\big[D_{KL}[q(z_j|x, y)\|p(z_j|x)]\big]\Big) \end{aligned} \tag{12}$$

where D_z is the dimensionality of the latent space z, and $\mathcal{M}$ denotes a mini-batch. Different solutions have been proposed, like KL annealing [27], but we empirically found them to be less effective.

4 Results

We now test the different types of conditional variational inference, introduced in the previous section, on two tasks. Evaluating generative model performance is not straightforward, as standard metrics like mean-squared error (MSE) are non-valid for multimodal outcome distributions. In this work, we evaluate **i)** the log likelihood of a test set under the learned generative model (see Appendix B), and (if possible) **ii)** we draw new data from the learned model and compute KL divergences or Hellinger distances with respect to the true data generating distribution. Training details and hyperparameters are described in Appendix C.

4.1 Toy Problem

We generate a one-dimensional multimodal transition function by sampling $x \sim \text{Uniform}(-1, 1)$ and sampling y from a conditional Gaussian distribution $\mathcal{N}(\cdot|\mu = f(x), \sigma = 0.1)$ according to:

$$p(y|x) = \begin{cases} \mathcal{N}(2.5), & \text{if } x < -0.3 \\ \rho_1\mathcal{N}(4x) + \rho_2\mathcal{N}(-4x), & \text{if } -0.3 \leq x < 0.3 \\ \rho_3\mathcal{N}(5 + \log(x+1)) + \rho_4\mathcal{N}(-x + 0.2) + \rho_5\mathcal{N}(5x^2), & \text{if } x \geq 0.3 \end{cases}$$

where $\rho_1 = 0.2$, $\rho_2 = 0.8$, $\rho_3 = 0.3$, $\rho_4 = 0.5$ and $\rho_5 = 0.2$. This generates the multimodal function shown in Figure 5a. We study this toy problem to visualize how different architectures will fit this simple data structure with conditional unimodal (left), bimodal (middle) and trimodal (right) structure (see Figure 5a left, middle and right parts). Figure 5b-f show the samples generated by different models after training on 30,000 mini-batch steps (See Appendix C for details). Table 1 displays the variational lower bound (VLB) and negative log-likelihood (NLL) on a test set.

A feed-forward network trained on mean squared error deterministically predicts the conditional expectation (Figure 5b). For fair comparison, we also train a feed-forward network that does receive noise variables ϵ as input but without an inference network (Figure 5c). Theoretically this network could learn the same decoder distribution, but without the inference network the model does not converge.

Figure 5d-f show the samples generated by different variational methods (spherical Gaussian Sec. 3.1), Gaussian with flow (Sec. 3.3), and discrete latent variables (Sec. 3.2), respectively. We see how these models are much better at fitting the true data distribution. Importantly, notice that the variational approach consistently predicts the deterministic part correctly (left part of the function). This is important, as the network is able to ignore the input noise when needed. Table 1 indicates the discrete latent variable model fits this problem best.

4.2 Stochastic Gridworld

We now study a typical RL gridworld task with multimodal stochastic dynamics. The world is a 7x7 grid (see Figure 6) with some walls. The agent (green) starts in

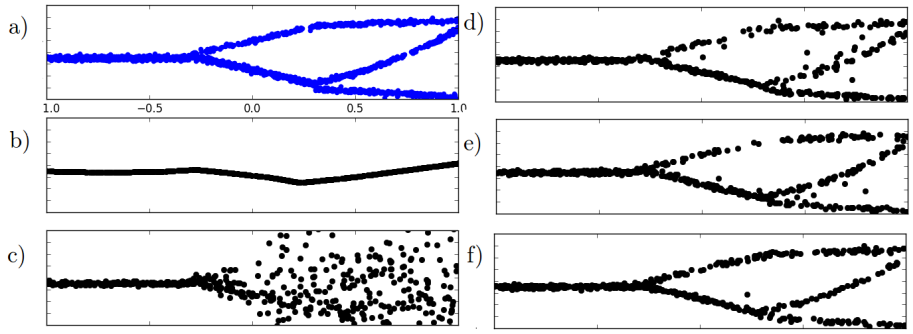

Fig. 5. Comparison of samples from the models produced by multi-layer perceptron (MLP) and variational inference (VI) networks after training for 30,000 mini-batches. a) Ground truth data. b) MLP (deterministic predictions). c) MLP with stochastic inputs. d) VI with spherical Gaussian. e) VI with spherical Gaussian and 5 layers of flow. f) VI with discrete latent variables. Numerical results are reported in Table 1.

Table 1. Performance on toy domain. All results are averaged over 10 runs. VLB = Variational Lower Bound, NLL = Negative Log Likelihood on test dataset, MLP = Multi-Layer Perceptron, VAE = Variational Auto-Encoder.

Method	**VLB**	**NLL**
MLP (deterministic)	NA	NA
MLP (with stochastic input)	NA	4.49
VAE continuous (n=3, no flow)	0.33	-0.29
VAE continuous (n=3, n_{flow}=5)	0.32	-0.33
VAE discrete (n=3, k=3)	**0.47**	**-0.48**

the bottom-left, can deterministically move in each cardinal direction, and needs to reach the top-right ($r = +10$). There are two ghosts, starting in locations as shown in Fig. 7, top-left. Ghost 1 (red) uniformly chooses one of the available directions. Ghost 2 (blue) has a bias to move to the left or right (40% each), and moves vertically with small probability (10%). Our interest here is to learn to predict this stochasticity from observed data. As state-space we use a vector of length 6 containing the 2D coordinates for the agent and both ghosts. Each element in the vector is treated as a categorical variable with 7 classes (for the 7x7 grid).[4]

Uncorrelated Data. On of the core challenges of RL is the exploration problem, which can make the data we observe strongly correlated. For example, if

[4] Note that, although this is a discrete MDP, it is not a trivial task to model multi-modality here. Indeed, the outcome distribution *per state dimension* is categorical, but the joint distribution (generally) does not factorize over the dimensions. Therefore, we would already need a categorical with $7^6 = 117649$ outcome categories to learn this problem without conditional variational inference, and this would exponentially aggravate in larger state-spaces (e.g. images).

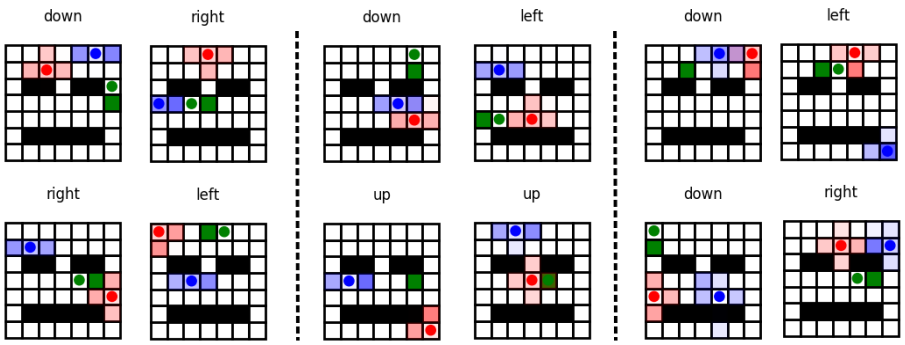

Fig. 6. Visual predictions on gridworld. Each sub-picture shows the agent (green), ghost 1 (red) and ghost 2 (blue) with current location as a circle, and predicted next location as a shaded box (color intensity corresponds to predicted probability). Black locations are walls, the text above each subplot indicates the action chosen by the agent. **Left:** Continuous latent variables (n=8, no flow), **Middle:** continuous latent variable (n=8, n_{flow} =6), **Right:** discrete latent variables (n=8,k=4). We observe stochastic predictions for the ghosts and deterministic predictions for the agent. Numerical comparison is provided in Table 2.

Table 2. Performance on gridworld predictions for different types of variational inference. For this table, $\hat{p}$ denotes the predicted distributions by the VAE model, while p denotes the ground truth (which is known for this scenario). VLB = Variational Lower Bound, NLL = Negative Log Likelihood.

Method	**VLB**	**NLL**	$D_{KL}(p\|\hat{p})$	$D_{Hel}(p\|\hat{p})$	$D_{KL}(\hat{p}\|p)$
VAE Continuous (n=8, no flow)	-2.53	2.52	**0.91**	**0.48**	**3.12**
VAE Continuous (n=8, n_{flow}=6)	-2.66	2.70	2.74	0.60	4.29
VAE Discrete (n=8, k=4)	**-2.17**	**2.20**	1.26	0.61	4.75

the agent never explores the top-left region of the domain, we can not expect it to learn an accurate model there. To overcome this problem, we first study an idealized setting in which our dataset consists of the transitions of state-action combinations randomly sampled across state-space.

On-policy Agent. The results on this task are shown in Table 2. Compared to Table 1 we do not show MLP results anymore for this task. We see the discrete latent variables again perform best on negative log-likelihood (NLL) evaluation. However, when we compare the learned distribution to the true distribution (which is available for this problem), we see the continuous latent variables without flow actually perform best. We see a conflict between both performance measures (the NLL indicates the discrete model performs best, while the distances with the true distribution point at the continuous latent model). In this case, visual comparison (Figure 6) does not show important differences across methods. We therefore conclude that the differences between methods are small

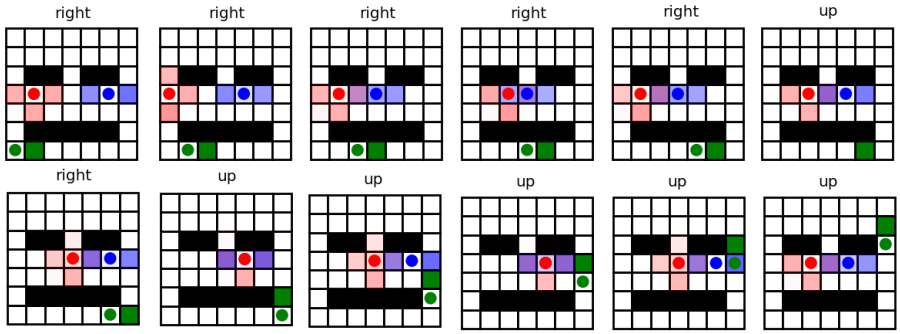

Fig. 7. *On-policy* predictions for RL agent (see Fig. 6 for color explanation). The subplots progress row-wise along a roll-out in the learned transition model. Note that this is a true 12-step roll-out, i.e. each next plot is based on sampling a single prediction from the model (we do not observe any true next state along the way).

for this problem, while the best performance measures for generative models remains an open questions in general (see [11]).

As a next step, we investigate to what extend an RL agent is capable of learning an accurate transition model *on-policy*, i.e. while observing correlated data. Note that the agent is still learning its policy in a model-free sense here (as a deep Q-network [22]), and we simply investigate to what extend the learned transition model is accurate after observing correlated data. Therefore, we evaluate the learned transition model while the agent is executing the policy.

Figure 7 shows the results of a roll-out in the learned model under a model-free policy. We see the agent first walking along the bottom corridor, and then moving up in the vertical corridor. Note that the agent consistently predicts its next state deterministically and correctly. In frame 6 it makes a wrong action decision, probably because we execute the behavioural policy with small ϵ-greedy noise. The ghosts have multimodal, stochastic behaviour. The first ghost (red) moves uniformly in one of the available directions, which is captured by the red shades around the current ghost location. Note that the model consistently predicts the ghost to move at each step. The second ghost (blue) has a bias to primarily step to the left or right. We also note the difference in the predicted next state between red and blue ghost, matching their true dynamics. Altogether, the agent has learned to predict both the deterministic effects of its own actions as well as its stochastic environment, from on-policy, correlated data.

5 Related Work

Variational inference in the conditional setting was previously studied by Sohn et al. [26] and Walker et al. [30]. Compared to our work, these papers only use spherical Gaussian priors, and do not focus on reinforcement learning tasks. Our work focussed on VI with reparametrization gradients. There is a second line of research on latent variable models that uses score function gradients

[21], which are also known as REINFORCE in the RL context. A benefit of reparametrization gradients is that they don't suffer from the high variance usually encountered with score function gradients (a problem also known in RL).

The idea to apply flow to the latent layer originates from Rezende and Mohamed [25]. The transformation in Eq. 9 are related to the *affine coupling layers* of Dinh et al. [8], but then applied to the latent layer of a CVAE, while Dnih et al. [8] use them directly from observation level without variational inference. Applying flow transformations at latent VI level was introduced by Kingma et al. [16], where the authors used fully autoregressive transformations (which are harder to implement compared to our transformations, but potentially have more representational capacity).

To increase the expressivity of the latent approximation, we focussed on different types of latent variables, as well as (normalizing) flow. A third way to increase latent capacity is to factorize the distribution into several layers [27]. However, activating deeper stochastic layers is not straightforward [27], requiring either batch normalization or weight normalization [16]. We defer factorized inference networks to future work, especially in higher-dimensional tasks.

The different deep generative models discussed in Section 3 are not mutually exclusive. For example, the variational lossy auto-encoder (LVAE) [4] combines variational inference with PixelCNN-based decoders [24]. Such architectures force high-level conceptual information into the latent level, while the decoder should capture fine-grained details. This could be beneficial to sparsify the latent layer and as such benefit RL planning as well.

There is relatively little work on Bayesian Neural Networks for RL. Closest to ours is the work by Depeweg et al. [7], who study VI to estimate both transition function stochasticity (as studied in this work) combined with uncertainty (due to limited data). Compared to their work, we use a parametric inference network which allows us to generalize in the inference part, while they perform VI per individual datapoint. Second, they only considered Gaussian latent variables, while we investigate discrete latent variables and normalizing flow as well. The results of Depeweg et al. [7] also show the ability to learn multimodal stochasticity, and additionally show the benefit of planning over the model. Watter et al. [31] also used variational auto-encoders in a control task, but only as a regularizer for learning representations, not to make stochastic predictions. Gal et al. [9] uses Bayesian neural networks, in the form of Bayesian dropout, to track uncertainty (due to limited data) in transition dynamics estimation.

Finally, there is also a line of RL research that uses the transition function target to speed-up model-free RL. This idea has been identified as RL with 'auxiliary tasks' [14]. The gradients of the transition function predictions are denser compared to the sparse RL training signal, and used to speed-up training of deeper network layers shared between policy and transition network. However, this approach does not learn stochastic transitions (but could benefit from it, as it improves the learning signal), nor is it used for sample-based planning as in model-based RL.

6 Future Work

One clear line of future work is to use these transition models to improve agent performance, by planning over the model with either a given or learned reward function. Depeweg et al. [7] already provided a study in this direction. Compared to their work, it would especially be interesting to apply more adaptive roll-outs in the model, like Monte Carlo Tree Search (MCTS). Moreover, it would be important to evaluate these methods in high-dimensional RL tasks, e.g. with convolutional neural networks on raw pixel data [23]. Another extension is to use these models to improve exploration in stochastic domains (e.g. [23,13]).

An important second challenge, briefly mentioned in the Introduction and Section 4.2, is planning under uncertainty. RL initially provides correlated data from a limited part of state-space. When planning over this model, we should not extrapolate too much, nor trust our model too early with limited data. Planning under uncertainty was for example studied by Gal et al. [9] and Houthooft et al. [13]. Note that 'uncertainty' (due to limited data) is fundamentally different from the 'stochasticity' (true probabilistic nature of the domain) discussed in this paper.

A third challenge for transition dynamics estimation is memory (partial observability), when the current state does not provide all available information to make an prediction. Proposed solutions are recurrent neural networks (RNN) or Neural Turing Machines (NTM), which have both been studied in the variational inference context (in [5] and [10], respectively).

Combining stochasticity, uncertainty and memory in one function approximator would be an important integrating step in model-based RL.

7 Conclusion

This paper studied multimodal transition function estimation for RL agents, with a focus on variational inference with different types of latent variables. Our experiments show variational inference is a robust method to discriminate deterministic and stochastic elements of the transition function using function approximation, clearly improving over discriminative training. We verified results on a typical RL domain where tabular learning would be infeasible, showing the ability of these models to learn the multimodal transition dynamics online. We did not observe important distinction in performance between the different types of latent variables studied. Therefore, for the domain size studied in this work, it seems safe to use the standard spherical Gaussian conditional VAE. Our results are generally applicable in RL, and help solve a fundamental problem of many domains: the complex stochastic behaviour of its transition dynamics. Code to reproduce the results in this paper is publicly available at `www.github.com/tmoer/multimodal_varinf`.

References

1. Atkeson, C.G., Moore, A.W., Schaal, S.: Locally weighted learning for control. In: Lazy learning, pp. 75–113. Springer (1997)

2. Brafman, R.I., Tennenholtz, M.: R-max-a general polynomial time algorithm for near-optimal reinforcement learning. Journal of Machine Learning Research 3(Oct), 213–231 (2002)
3. Burda, Y., Grosse, R., Salakhutdinov, R.: Importance weighted autoencoders. arXiv preprint arXiv:1509.00519 (2015)
4. Chen, X., Kingma, D.P., Salimans, T., Duan, Y., Dhariwal, P., Schulman, J., Sutskever, I., Abbeel, P.: Variational Lossy Autoencoder. arXiv preprint arXiv:1611.02731 (2016)
5. Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A.C., Bengio, Y.: A recurrent latent variable model for sequential data. In: Advances in neural information processing systems. pp. 2980–2988 (2015)
6. Deisenroth, M., Rasmussen, C.E.: 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)
7. Depeweg, S., Hernández-Lobato, J.M., Doshi-Velez, F., Udluft, S.: Learning and policy search in stochastic dynamical systems with bayesian neural networks. arXiv preprint arXiv:1605.07127 (2016)
8. Dinh, L., Sohl-Dickstein, J., Bengio, S.: Density estimation using Real NVP. arXiv preprint arXiv:1605.08803 (2016)
9. Gal, Y., McAllister, R.T., Rasmussen, C.E.: Improving PILCO with Bayesian Neural Network Dynamics Models. In: Data-Efficient Machine Learning workshop. vol. 951, p. 2016 (2016)
10. Gemici, M., Hung, C.C., Santoro, A., Wayne, G., Mohamed, S., Rezende, D.J., Amos, D., Lillicrap, T.: Generative Temporal Models with Memory. arXiv preprint arXiv:1702.04649 (2017)
11. Goodfellow, I.: NIPS 2016 Tutorial: Generative Adversarial Networks. arXiv preprint arXiv:1701.00160 (2016)
12. Hester, T., Stone, P.: Learning and using models. In: Reinforcement Learning, pp. 111–141. Springer (2012)
13. Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., Abbeel, P.: VIME: Variational information maximizing exploration. In: Advances in Neural Information Processing Systems. pp. 1109–1117 (2016)
14. Jaderberg, M., Mnih, V., Czarnecki, W.M., Schaul, T., Leibo, J.Z., Silver, D., Kavukcuoglu, K.: Reinforcement learning with unsupervised auxiliary tasks. arXiv preprint arXiv:1611.05397 (2016)
15. Jang, E., Gu, S., Poole, B.: Categorical Reparameterization with Gumbel-Softmax. arXiv preprint arXiv:1611.01144 (2016)
16. Kingma, D.P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., Welling, M.: Improved Variational Inference with Inverse Autoregressive Flow. In: Advances in Neural Information Processing Systems. pp. 4743–4751 (2016)
17. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114 (2013)
18. Li, L., Littman, M.L., Walsh, T.J., Strehl, A.L.: Knows what it knows: a framework for self-aware learning. Machine learning 82(3), 399–443 (2011)
19. Li, Y., Turner, R.E.: Rényi divergence variational inference. In: Advances in Neural Information Processing Systems. pp. 1073–1081 (2016)
20. Maddison, C.J., Mnih, A., Teh, Y.W.: The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. arXiv preprint arXiv:1611.00712 (2016)
21. Mnih, A., Gregor, K.: Neural variational inference and learning in belief networks. arXiv preprint arXiv:1402.0030 (2014)

22. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature 518(7540), 529–533 (2015)
23. Oh, J., Guo, X., Lee, H., Lewis, R.L., Singh, S.: Action-conditional video prediction using deep networks in atari games. In: Advances in Neural Information Processing Systems. pp. 2863–2871 (2015)
24. Oord, A.v.d., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., Kavukcuoglu, K.: Conditional image generation with PixelCNN decoders. arXiv preprint arXiv:1606.05328 (2016)
25. Rezende, D.J., Mohamed, S.: Variational inference with normalizing flows. arXiv preprint arXiv:1505.05770 (2015)
26. Sohn, K., Lee, H., Yan, X.: Learning structured output representation using deep conditional generative models. In: Advances in Neural Information Processing Systems. pp. 3483–3491 (2015)
27. Sønderby, C.K., Raiko, T., Maaløe, L., Sønderby, S.K., Winther, O.: Ladder variational autoencoders. In: Advances in Neural Information Processing Systems. pp. 3738–3746 (2016)
28. Stadie, B.C., Levine, S., Abbeel, P.: Incentivizing exploration in reinforcement learning with deep predictive models. arXiv preprint arXiv:1507.00814 (2015)
29. Sutton, R.S.: Dyna, an integrated architecture for learning, planning, and reacting. ACM SIGART Bulletin 2(4), 160–163 (1991)
30. Walker, J., Doersch, C., Gupta, A., Hebert, M.: An Uncertain Future: Forecasting from Static Images Using Variational Autoencoders. In: European Conference on Computer Vision. pp. 835–851. Springer (2016)
31. Watter, M., Springenberg, J., Boedecker, J., Riedmiller, M.: Embed to control: A locally linear latent dynamics model for control from raw images. In: Advances in Neural Information Processing Systems. pp. 2746–2754 (2015)

A Variational Auto-Encoder (VAE) Training Objective

We can obtain a tighter bound on Equation 2 by using importance sampling [3]. We sample M values of z per datapoint, and average over them inside the log. Otherwise, the model strongly penalizes for single samples that explain the objective poorly. Second, instead of the KL divergence we optimize Renyi α-divergences [19]). We use α=0.5 according to the results by Depeweg et al. [7], which makes the divergence term become a function of the Hellinger distance [19]. The combined objective, known as the variational Renyi (VR) bound [19] is:

$$\mathcal{L}_{VR}(y|x) = \frac{1}{1-\alpha} \log \frac{1}{M} \sum_{m=1}^{M} \left[\left(\frac{p(y, z^m|x)}{q(z^m|y, x)} \right)^{1-\alpha} \right] \tag{13}$$

with $z^m \sim q(\cdot|x, y)$.

B Test Set Negative Log-likelihood (NLL) for VAE

We are interested in the likelihood $p(y|x)$ of a set of test data $\{x_i, y_i\}_{i=1}^N$. We therefore need to marginalize over z:

$$p(y|x) = \mathbb{E}_{z\sim p(\cdot|x)} \Big[p(y|z, x) \Big] \tag{14}$$

One problem with this estimator is that we may need many empirical samples from z to get an accurate estimate. As an alternative, we estimate the quantity through importance sampling, by sampling from $q(\cdot|x_i, y_i)$ instead of $p(\cdot|x_i)$:

$$p(y|x) = \mathbb{E}_{z\sim q(\cdot|x,y)} \Big[p(y|z, x) \frac{p(z|x)}{q(z|x, y)} \Big] \tag{15}$$

The empirical estimate of the negative log likelihood (NLL), as reported in the results section, then becomes

$$-\log p(y|x) = -\frac{1}{N} \sum_{i=1}^{N} \log \Big[\frac{1}{M} \sum_{m=1}^{M} p(y_i|z_i^m, x_i) \frac{p(z_i^m|x_i)}{q(z_i^m|x_i, y_i)} \Big] \tag{16}$$

with $z_i^m \sim q(\cdot|x_i, y_i)$.

C Training Details

For all experiments we follow standard train, validation and test set set-up. For all domains, we train the VAE target on $k = 3$ importance samples with Renyi-α divergence for $\alpha = 0.5$ (see appendix A). This gave us slightly better results compared to the 'default' settings of $k = 1$ and $\alpha = 1.0$. All models are trained in Tensorflow using Adam optimizer.

Toy Domain: We draw a training set of size 2000, and independent validation and test sets of size 500 and 2000, respectively. The decoder distribution is

Gaussian, where we also learn its standard deviation. We train for 30000 batches with batch size 64, with a learning rate linearly annealed from 0.005 to 0.0005 over 90% of training steps. The minimal KL penalty per dimension λ (Eq. 12) is fixed at 0.07.

The generative network has three layers with 50 units per layer and Relu non-linearities. The inference network has two layers with 30 units per layer and Relu non-linearities. For the discrete latent variables, we anneal the temperature from 2.0 to 0.001 over 70% of training steps.

Gridworld: For the first task, we repeatedly draw training data by sampling a new state-action combination uniformly across state-space, and sampling a single transition. Optimal model performance is based on a VAE performance on a validation and test set of size 750 and 1500 respectively. The decoder distribution is discrete taking values in 7 categories. We train on mini-batches of size 32 for 75000 iterations, with a learning rate linearly annealed from 0.0005 to 0.0001 over 70% of training steps. The generative network has three layers with 250 units per layer and Relu non-linearities. The inference network has two layers with 100 units per layer and Relu non-linearities. The minimal KL penalty per dimension λ (Eq. 12) is fixed at 0.07.

For the on-policy evaluation, the RL policy is trained as a deep Q-network [22] with target network and no experience replay. The state-action value network has three layers of 50 units and Relu activations. Given a mini-batch $\mathcal{M}$ of roll-out data under the current policy, the network is trained on the 1 step Q-learning objective:

$$L_{RL}(\eta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{M}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \eta^-) - Q(s, a; \eta)\right)^2\right] \tag{17}$$

where s, a, r denote state, action and reward, $Q(s, a)$ is the expected discounted return (discount parameter $\gamma = 0.99$) from state s and action a under the current policy, η are the parameters in the value function network, and η^- the parameters in the target network (which are fixed in the above loss, and only updated every 500 steps). During learning, we follow an ϵ-greedy policy with ϵ linearly decayed from 1.0 to 0.10 over 60% of training steps.