Variationally Inferred Sampling
Through a Refined Bound for Probabilistic Programs

Víctor Gallego¹,², David Ríos Insua¹,²

¹Institute of Mathematical Sciences (ICMAT-CSIC), Madrid, Spain
²SAMSI, Duke University, Durham, NC, USA
{victor.gallego, david.rios}@icmat.es

Abstract
A framework to boost efficiency of Bayesian inference in probabilistic programs is introduced by embedding a sampler inside a variational posterior approximation, which we call the refined variational approximation. Its strength lies both in ease of implementation and in automatically tuning the sampler parameters to speed up mixing time. Several strategies to approximate the evidence lower bound (ELBO) computation are introduced, including a rewriting of the ELBO objective. A specialization towards state-space models is proposed. Experimental evidence of its efficient performance is shown by solving an influence diagram in a high-dimensional space using a conditional variational autoencoder (cVAE) as a deep Bayes classifier; an unconditional VAE on density estimation tasks; and state-space models for time-series data.

1 Introduction
Probabilistic programming offers powerful tools for Bayesian modelling, a framework for describing prior knowledge and reasoning about uncertainty. A probabilistic programming language (PPL) can be viewed as a programming language extended with random sampling and Bayesian conditioning capabilities, complemented with an inference engine that produces answers to inference, prediction and decision making queries. Some examples are WinBUGS (Lunn et al. 2000), Stan (Carpenter et al. 2017), or the recent Edward (Tran, Ranganath, and Blei 2015). The machine learning and artificial intelligence communities are pervaded by models that can be expressed naturally through a PPL. Variational autoencoders (VAE) (Kingma and Welling 2013) and hidden Markov models (HMM) (Rabiner 1989) are two relevant examples.

If we consider a probabilistic program to define a distribution \( p(x, z) \), where \( x \) are observations and \( z \) denote both latent variables and parameters, then we are interested in asking queries involving the posterior \( p(z|x) \). This distribution is typically intractable but, conveniently, PPLs provide inference engines to approximate this distribution using Monte Carlo methods (e.g. particle Markov Chain Monte Carlo (MCMC) (Andrieu, Doucet, and Holenstein 2010) or Hamiltonian Monte Carlo (HMC) (Neal and others 2011)) or variational approximations (e.g. Automatic Differentiation Variational Inference (ADVI) (Kucukelbir et al. 2017)). Whereas the latter are biased and underestimate uncertainty, the former methods may be exceedingly slow depending on the target distribution. For such reason, over the recent years, there has been an increasing interest in developing more efficient posterior approximations (Nalisnick, Hertel, and Smyth 2016, Salimans, Kingma, and Welling 2015, Tran, Ranganath, and Blei 2015).

It is well known that the performance of a sampling method depends on the parameters used (Papaspiliopoulos, Roberts, and Sköld 2007). In this work, we propose a framework to automatically adapt the shape of the posterior and also tune the parameters of a posterior sampler with the aim of boosting Bayesian inference efficiency in probabilistic programs. Our framework can be regarded as a principled way to enhance the flexibility of the variational posterior approximation, yet can be seen also as a procedure to tune the parameters of an MCMC sampler. Our contributions can be summarised as follows:

• A new flexible and unbiased variational approximation to the posterior, which consists of improving an initial variational approximation with a stochastic process.
• An alternative ELBO function objective formulation, which is a variant of the original one when this new variational approximation is adopted.
• A specialization to the case of Bayesian inference in state-space models.

1.1 Related work
The idea of preconditioning the posterior distribution to speed up the mixing time of an MCMC sampler has recently been explored in (Hoffman et al. 2018) and (Li and Wang 2018), where a reparameterization is learned before performing the sampling via HMC. Both papers extend seminal work of (Parno and Marzouk 2014) by learning an efficient and expressive deep, non-linear transformation instead of a polynomial regression. However, they do not account for tuning the parameters of the sampler as we introduce in Section 3.

The work of (Rezende and Mohamed 2015) introduced a general framework for constructing more flexible variational
distributions, called normalizing flows. These transformations are one of the main techniques to improve the flexibility of current VI approaches and have recently pervaded the literature of approximate Bayesian inference with current developments such as continuous-time normalizing flows (Chen et al. 2018) which extend an initial simple variational posterior with a discretization of Langevin dynamics. However, they require a generative adversarial network (GAN) (Goodfellow et al. 2014) to learn the posterior, which can be unstable in high-dimensional spaces. We overcome this issue with the novel formulation stated in Section 3. Our framework is also compatible with different optimizers, not only those derived from Langevin dynamics. Other recent proposals to create more flexible variational posteriors are based on implicit approaches, which typically require a GAN (Huszár 2017) or implicit schema such as UIVI (Titsias and Welling 2011) proposed a formulation of a continuous-time Markov process that converges to a target distribution \( p(z|x) \) with \( z \in \mathbb{R}^d \). It is based on the Euler-Maruyama discretization of Langevin dynamics:

\[
\begin{align*}
  z_{t+1} &\leftarrow z_t - \eta_t \nabla \log p(z_t, x) + \mathcal{N}(0, 2\eta_t I),
\end{align*}
\]

where \( \eta_t \) is the step size. The required gradient \( \nabla \log p(z_t, x) \) can be estimated using mini-batches of data. Several extensions of the original Langevin sampler have been proposed to increase the mixing speed, see for instance (Li et al. 2016b; Gallego and Insua 2018).

### 2.2 Inference as optimization

Variational inference, (Kucukelbir et al. 2017), tackles the problem of approximating the posterior \( p(z|x) \) with a tractable parameterized distribution \( q_\phi(z|x) \). The goal is to find parameters \( \phi \) so that the variational distribution (also referred to as the variational guide or variational approximation) \( q_\phi(z|x) \) is as close as possible to the actual posterior. Closeness is typically measured through Kullback-Leibler divergence \( KL(q||p) \), which is reformulated into the ELBO, the objective to be optimized using stochastic gradient descent techniques:

\[
\text{ELBO}(q) = \mathbb{E}_{q_\phi(z|x)} \left[ \log p(x, z) - \log q_\phi(z|x) \right].
\]

Typically, a deep, non-linear model conditioned on observation \( x \) defines the mean and covariance matrix of a Gaussian distribution \( q_\phi(z|x) \sim \mathcal{N}(\mu_\phi(x), \sigma_\phi(x)) \), to allow for greater flexibility.

### 3 The Variationally Inferred Sampling (VIS) framework

In standard VI, the variational approximation \( q_\phi(z|x) \) is analytically tractable. It is typically chosen as a factorized Gaussian distribution as described in Section 2.2. We propose to use a more flexible approximating posterior by embedding a sampler through:

\[
q_{\phi, \eta}(z|x) = \int Q_{\eta,T}(z|z_0)q_{0, \phi}(z_0|x)dz_0,
\]
where \(q_{0,\phi}(z|x)\) is the initial and tractable density (i.e., the starting state for the sampler). We will refer to \(q_{0,\phi}(z|x)\) as the refined variational approximation. The conditional distribution \(Q_{0,T}(z|y_0)\) refers to a stochastic process parameterized by \(\eta\) used to evolve the original density \(q_{0,\phi}(z|x)\) and achieve greater flexibility. In the following subsections we describe particular forms of \(Q_{0,T}(z|y_0)\). When \(T = 0\), no refinement steps are performed, so the refined variational approximation coincides with the original variational approximation, \(q_{0,\eta}(z|x) = q_{0,\phi}(z|x)\). As \(T\) increases, the variational approximation will be closer to the exact posterior, provided that \(Q_{0,T}\) is a valid MCMC sampler. Next, instead of optimizing the ELBO, a refined ELBO is maximized,

\[
\text{rELBO}(q) = \mathbb{E}_{q_{0,\eta}(z|x)}[\log p(x, z) - \log q_{\phi,\eta}(z|x)]
\]

(4)

to optimize the divergence \(KL(q_{\phi,\eta}(z|x)||p(z|x))\). The first term of the rELBO only requires sampling from \(q_{\phi,\eta}(z|x)\); however the second term, the entropy \(-\mathbb{E}_{q_{0,\eta}(z|x)}[\log q_{0,\eta}(z|x)]\) requires also evaluating the evolving, implicit density. Depending on the conditional distribution \(Q_{0,T}(z|y_0)\), the integral (3) may be analytically tractable or not. We propose a set of effective guidelines for the rELBO optimization:

1. Consider the evolved density \(q_{\phi,\eta}(z|x)\) as a finite mixture of Dirac Deltas (i.e. we approximate the density using a finite set of particles, so the previous entropy is zero. In more detail, we sample \(z^1,\ldots,z^K \sim q_{0,\phi}(z|x)\), and then \(q_{\phi,\eta}(z|x) = \frac{1}{K} \sum_{i=1}^K \delta(z-z^i)\) as the variational approximation.
2. An hybrid approach, in which some integrals can be analytically computed and the others approximated by the Delta approximation. See Section 3.3 for a discussion on this in the state-space model setting.

Regarding \(Q_{0,T}(z|y_0)\), we consider the following families of sampling algorithms.

### 3.1 Continuous latent variables

When the latent variables \(z\) are continuous (\(z \in \mathbb{R}^d\)), we propose to evolve the original variational density \(q_{0,\phi}(z|x)\) through a stochastic diffusion process. In order to make it tractable, we discretize the Langevin dynamics using the Euler-Maruyama scheme, arriving at the stochastic gradient Langevin dynamics (SGLD) sampler.

We now follow the process \(Q_{0,T}(z|y_0)\) (representing \(T\) iterations of an MCMC sampler). As an example, we make it explicit for the SGLD sampler through

\[
z_i = z_{i-1} + \eta \nabla \log p(x, z_{i-1}) + \xi_i,
\]

where \(i\) iterates from 1 to \(T\) and, in this case, the only parameter of the SGLD sampler is the learning rate \(\eta\). The noise for the SGLD is denoted \(\xi_i \sim \mathcal{N}(0, 2\eta I)\). Note that for some models, the previous gradient \(\nabla \log p(x, z_i)\) is a linear function of \(z_i\), so we can compute the exact distribution of \(q(z_{i+1})\) from the distribution of \(q(z_i)\). In other cases, we resort to approximate the non-analytical terms using the Delta approximation described before. Figure 1 provides a graphical representation of the variational approximation.

The initial variational distribution, \(q_{0,\phi}(z|x)\) is a Gaussian parameterized by a deep neural network (NN). Then, \(T\) iterations of a sampler \(Q\), parameterized by \(\eta\), are applied leading to the final distribution \(q_{\phi,\eta}(.|.)\).

Figure 1: Probabilistic graph for the refined variational approximation.

An alternative may be given by ignoring the noise vector \(\xi\) (Mandt, Hoffman, and Blei 2017), thus refining the initial variational approximation with just the stochastic gradient descent (SGD). For this particular case, one can instead define a slightly different variational approximation instead of the Dirac Delta approximation, by treating the gradient terms as points but keeping a Gaussian distribution as the variational distribution. Details are shown in Appendix A. Moreover, we can use Stein variational gradient descent (SVGD) (Liu and Wang 2016) or a stochastic version (Gallego and Insua 2018) to apply repulsion between particles and promote a more extensive exploration of the latent space.

The effect of using different samplers is left for future work.

### 3.2 Tuning sampler parameters

In standard VI, the variational approximation \(q(z|x; \phi)\) is parameterized by \(\phi\). The parameters are learned using SGD or variants such as Adam (Kingma and Ba 2014), using the gradient \(\nabla_\phi \text{ELBO}(q)\). Since we have shown how to embed a sampler inside the variational guide, it is also possible to compute a gradient of the objective with respect to the sampler parameters \(\eta\). For instance, we can compute a gradient with respect to the learning rate \(\eta\) from the SGLD or SGD process from Section 3.1 \(\nabla_\eta \text{rELBO}(q)\), to search for an optimal step size at every VI iteration. This is an additional step apart from using the gradient \(\nabla_\phi \text{ELBO}(q)\) which is used to learn a good initial sampling distribution.

### 3.3 State-space model specialization

The previous framework is particularly useful in large families of state-space models (and by extension, models that exhibit hierarchical and/or temporal structure), mainly through two complementary strategies: i) exact marginalization of some particular terms (i.e., Rao-Blackwellization (Murray et al. 2018) to reduce the variance); ii) exact computation in linear cases. Recall that a state-space model (Hamilton 1994) can be expressed with the following probabilistic model,
where the time-step $t$ iterates from 1 to $\tau$:
\[
  z_{t+1} \sim p(z_{t+1}|z_t, \theta_t), \\
  x_{t+1} \sim p(x_{t+1}|z_{t+1}, \theta_{em}).
\]
This formulation subsumes many models used in Machine Learning such as Hidden Markov Models (HMMs) or Dynamic Linear Models (DLMs). It is often required to perform inference on the $\theta := (\theta_{em}, \theta_t)$ parameters from the transition and emission equations, respectively. We propose to use a variational distribution $q(\theta)$, which will be refined by any sampling method (as described in Section 3.1):
\[
  \theta \leftarrow \theta + \nabla_\theta \log p(x_{1:T}, z_{1:T}, \theta) + \xi. \tag{5}
\]
Note that for a large class of models (including HMMs and DLMs) we can marginalize out $z_{1:T}$ and have reduced variance iterating with:
\[
  \theta \leftarrow \theta + \nabla_\theta \log p(x_{1:T}|\theta) + \xi, \tag{6}
\]
where the latent variables $z_{1:T}$ have been marginalized out using the sum-product algorithm. For linear-Gaussian models, we can also compute the exact form of the refined posterior, since all terms in Eq. 5 are linear with respect to the latent variables $\theta$. However, inference in these linear models is exact by using conjugate distributions, so the proposed framework is more suitable for the case of state-space models containing non-linear (or non-conjugate) components. For these families of models, we resort to use just a gradient estimator of the entropy or the Delta approximation in Section 5.1.

4 Analysis of VIS

In this Section we study in detail key properties of the proposed VIS framework.

4.1 Rewriting the ELBO

Performing variational inference with the refined variational approximation can be regarded as using the original variational guide while optimizing an alternative, tighter ELBO. Note that for a refined guide of the form $q(z|z_0)q(z_0|x)$, the objective function can be written as
\[
  \mathbb{E}_{q(z|z_0)q(z_0|x)} \left[ \log p(x, z) - \log q(z|z_0) - \log q(z_0|x) \right].
\]
However, using the Dirac Delta approximation for $q(z|z_0)$ and noting that $z = z_0 + \eta \nabla \log p(x, z_0)$ when using SGD with $T = 1$, we arrive at the modified objective:
\[
  \mathbb{E}_{q(z_0|x)} \left[ \log p(x, z_0 + \eta \nabla \log p(x, z_0)) - \log q(z_0|x) \right]
\]
which is equivalent to the refined ELBO introduced in 4.1. Since we are perturbing the latent variables in the steepest ascent direction, it is straightforward to show that, for moderate $\eta$,
\[
  \text{ELBO}(q) \leq \text{rELBO}(q),
\]
for the original variational guide $q(z_0|x)$. This reformulation of ELBO is also convenient since it provides a clear way of implementing our refined variational inference framework in any PPL supporting algorithmic differentiation.

4.2 Taylor expansion

From the result in subsection 4.1, we can further restrict to the case when the original variational approximation is also a Dirac point mass. Then, the original ELBO optimization resorts to use just a gradient estimator of the entropy or the Delta approximation. Within the VIS framework, we optimize instead $\max_{\tau} \log p(x, z + \Delta z)$, where $\Delta z$ is one iteration of the sampler, i.e., $\Delta z = \eta \nabla \log p(x, z)$ in the SGD case. For notational clarity we resort to the case $T = 1$, but a similar analysis can be straightforwardly done if more refinement steps are performed.

We may now perform a first-order Taylor expansion of the refined objective as
\[
  \log p(x, z + \Delta z) \approx \log p(x, z) + (\Delta z)^T \nabla \log p(x, z).
\]
Taking gradients of the first order approximation w.r.t. the latent variables $z$ we arrive at
\[
  \nabla_z \log p(x, z) + \eta \nabla \log p(x, z)^T \nabla_z \log p(x, z),
\]
where we have not computed the gradient through the $\Delta z$ term. That is, the refined gradient can be deemed as the original gradient plus a second order correction. Instead of being modulated by a constant learning rate, this correction is adapted by the chosen sampler. In the experiments in Section 5.4 we show that this is beneficial for the optimization as it can take less iterations to achieve lower losses. By further taking gradients through the $\Delta z$ term, we may tune the sampler parameters such as the learning rate as described in Section 3.2. Consequently, the next subsection describes both modes of differentiation.

4.3 Two modes of Automatic Differentiation for rELBO optimization

Here we describe how to implement two variants of the rELBO objective. First, we define a stop gradient operator that sets the gradient of its operand to zero, i.e., $\nabla_x \perp(x) = 0$ whereas in the forward pass it acts as the identity function, that is, $\perp(x) = x$. Then, the two variants of the rELBO objective are
\[
  \mathbb{E}_q \left[ \log p(x, z + \Delta z) - \log q(z + \Delta z|x) \right] \quad \text{(Full AD)}
\]
and
\[
  \mathbb{E}_q \left[ \log p(x, z + \perp(\Delta z)) - \log q(z + \perp(\Delta z)|x) \right]. \quad \text{(Fast AD)}
\]
The Full AD rELBO makes it possible to further compute a gradient w.r.t. sampler parameters inside $\Delta z$ at the cost of a slight increase in the computational burden. However, the Fast AD variant may be handy in multiple scenarios as we illustrate below.

5 Experiments

We first detail the experiments. We emphasize that our framework permits rapid iterations over a large class of models (i.e., it is more automatic than, e.g., manually setting up a Gibbs sampler). Through the following experiments, we aim to shed light on the following questions:

\footnote{corresponds to detach in Pytorch or stop\_gradient in tensorflow.}
Q1 Is the increased computational complexity of computing gradients through sampling steps worth the flexibility gains?
Q2 Is the proposed framework compatible with other structured inference techniques, such as the sum-product algorithm?
Q3 Does the more flexible posterior approximated by VIS help in auxiliary tasks, such as decision making or classification?

Within the spirit of reproducible research, the code will be released at https://github.com/vicgalle/vis. The VIS framework was implemented using Pytorch and Jax.

5.1 Funnel density
As a preliminary experiment, we test the VIS framework on a synthetic yet complex target distribution. The target, bi-dimensional density is defined through:
\[ z_1 \sim \mathcal{N}(0, 1.35) \]
\[ z_2 \sim \mathcal{N}(0, \exp(z_1)). \]
As a variational approximation we take the usual diagonal Gaussian distribution. For the VIS case, we consider to refine it for \( T = 1 \) steps using SGLD. Results are shown in Figure 2. In the top, we show the trajectories of the lower bound for up to 50 iterations of variational optimization with Adam. It is clear that our refined version achieves a tighter bound. The middle and bottom figures present the contour curves of the learned variational approximations. The VIS variant is placed nearer to the mean of the true distribution and is more disperse than the original variational approximation, confirming the fact that the refinement step helps in attaining more flexible posterior approximations.

5.2 State-space Markov models
We test our variational approximation on two state-space models, one for discrete data and the other for continuous observations. All the experiments in this subsection use the Fast AD version from Section 4.3 since it was not necessary to further tune the sampler parameters to have competitive results.

**Hidden Markov Model (HMM).** The model equations are given by
\[
p(z_{1:T}, x_{1:T}, \theta) = \prod_{t=1}^{T} p(x_t | z_t, \theta_{em})p(x_t | x_{t-1}, \theta_{tr})p(\theta),
\]
where each conditional is a Categorical distribution which takes 5 different classes and the prior \( p(\theta) = p(\theta_{em})p(\theta_{tr}) \) are two Dirichlet distributions that sample the emission and transition probabilities, respectively. We perform inference on the parameters \( \theta \).

**Dynamic Linear Model (DLM).** The model equations are the same as in the HMM case, though the conditional distributions are now Gaussian and the parameters \( \theta \) refer to the emission and transition variances. As before, we perform inference over \( \theta \).

The full model implementations can be checked in Appendix [B.1] based on funsor, a PPL on top of the Pytorch autodiff framework. For each model, we generate a synthetic dataset, and use the refined variational approximation with \( T = 0, 1, 2 \). As the original variational approximation to the parameters \( \theta \) we use a Dirac Delta. Performing VI with this approximation corresponds to MAP estimation using the Kalman filter in the DLM case (Zarchan and Musoff 2013) and the Baum-Welch algorithm in the HMM case (Rabiner 1989), so we marginalize out the latent variables \( z_{1:T} \). Model details are given in Appendix TODO. Figure 3 shows the results. The first row reports the experiments related to the HMM; the second one to the DLM. While in all graphs we report the evolution of the loglikelihood during inference, in the first column we report the number of ELBO iterations, whereas in the second column we measure wall-clock time as the optimization takes place. We confirm that VIS \( (T > 0) \) achieve better results than regular optimization with VI \( (T = 0) \) for a similar amount of time.

**Prediction tasks in a HMM** With the aim of assessing whether ELBO optimization helps in attaining better auxiliary scores, we also report results on a prediction task. We generate a synthetic time series of alternating 0 and 1 for \( \tau = 105 \) timesteps. We train the HMM model from before on the first 100 points, and report in Table 1 the accuracy of the predictive distribution \( p(y_t) \) averaged over the last 5 timesteps. We also report the predictive entropy since it helps in assessing the confidence of the model in its forecast and is a strictly proper scoring rule (Gneiting and Raftery 2007). To guarantee the same computational budget time and a fair

---

https://github.com/pyro-ppl/funsor/
The third batch of experiments aims to check whether the VIS framework is competitive with respect to other algorithms. Also, its predictive intervals are narrower than the non-refined counterpart. As can be seen, for similar wall-clock time per epoch using \( T = 5 \) is 10.46 s, whereas with no refinement \( T = 0 \) is 6.10 s (hence our decision to train the refined model without refining is run with 50 epochs, whereas the model with refinement is run for 20 epochs. We see that the refined model achieves higher accuracy than its counterpart; in addition it is correctly more confident in its predictions.

| Table 1: Prediction metrics for the HMM. |
|-----------------|-----------------|-----------------|
| \( T = 0 \)    | \( T = 1 \)    |
| accuracy        | 0.40            | 0.84            |
| predictive entropy | 1.414          | 1.056           |
| logarithmic score | −1.044         | −0.682          |

**Prediction task in a DLM**  We now test the VIS framework on the Mauna Loa monthly \( CO_2 \) time series data (Keeling 2005). As the training set, we take the first 10 years, and we evaluate over the next 2 years. We use a DLM composed of a local linear trend plus a seasonality block of periodicity 12. Full model specification can be checked in Appendix B.1. As a preprocessing step, we standardize the time series to zero mean and unitary deviation. To guarantee the same computational budget time, the model without refining is run for 10 epochs, whereas the model with refinement is run for 4 epochs. We report mean absolute error (MAE) and predictive entropy in Table 2. In addition, we compute the interval score as defined in (Gneiting and Raftery 2007), a strictly proper scoring rule. As can be seen, for similar wall-clock times, the refined model not only achieves lower MAE, but also its predictive intervals are narrower than the non-refined counterpart.

| Table 2: Prediction metrics for the DLM. |
|-----------------|-----------------|-----------------|
| \( T = 0 \)    | \( T = 1 \)    |
| MAE             | 0.270           | 0.239           |
| predictive entropy | 2.537          | 2.401           |
| interval score (\( \alpha = 0.05 \)) | 15.247          | 13.461           |

### 5.3 Variational Autoencoder

The third batch of experiments aims to check whether the VIS framework is competitive with respect to other algorithms from the recent literature. To this end, we test our approach in a Variational Autoencoder (VAE) model (Kingma and Welling 2013). Performing efficient and flexible inference in a VAE is useful since it is the building block of more complex models and tasks (Bouchacourt, Tomioka, and Nowozin 2018; Wang et al. 2018). The VAE defines a conditional distribution \( p_{\theta}(x|z) \), generating an observation \( x \) from a latent variable \( z \). For this task, we are interested in modelling two \( 28 \times 28 \) image distributions, MNIST and fashion-MNIST. To perform inference (learn parameters \( \theta \)) the VAE introduces a variational approximation \( q_{\phi}(z|x) \). In the standard setting, this distribution is Gaussian; we instead use the refined variational approximation comparing various values of \( T \). We also use the Full AD variant from Section 2.3.

As experimental setup, we reproduce the setting from (Titsias and Ruiz 2019). As model \( p_{\theta}(x|z) \), we use a factorized Bernoulli distribution parameterized with a two layer feed-forward network with 200 units in each layer and relu activation, except for the final sigmoid activation. As variational approximation \( q_{\phi}(z|x) \), we use a Gaussian whose mean and (diagonal) covariance matrix are parameterized by two separate neural networks with the same structure as the previous one, except the sigmoid activation for the mean and a softplus activation for the covariance matrix.

### Table 3: Test log-likelihood on binarized MNIST and fMNIST. VIS-X-Y denotes \( T = X \) refinement iterations during training and \( T = Y \) refinement iterations during testing.

| Method | MNIST | fMNIST |
|--------|-------|--------|
| UIVI   | \(-94.09\) | \(-110.72\) |
| SIVI   | \(-97.77\) | \(-121.53\) |
| VAE    | \(-98.29\) | \(-126.73\) |
| VCD    | \(-95.86\) | \(-117.65\) |
| HMC-DLGM | \(-96.23\) | \(-117.74\) |

Results are reported in Table 3. To guarantee a fair comparison, we trained the VIS-5-10 variant for 10 epochs, whereas all the other variants were trained for 15 epochs (fMNIST) or 20 epochs (MNIST), so that the VAE performance is comparable to the one reported in (Titsias and Ruiz 2019). Although VIS is trained for less epochs, by increasing the number of MCMC iterations \( T \), we dramatically improve on test log-likelihood. In terms of computational complexity, the average time per epoch using \( T = 5 \) is 10.46 s, whereas with no refinement \( T = 0 \) is 6.10 s (hence our decision to train the refined
variant for less epochs): a moderate increase in computing time may be worth the dramatic increase in log-likelihood while not introducing new parameters in the model, except for the learning rate $\eta$. We also show the results from the contrastive divergence approach from (Ruiz and Titsias 2019) and the HMC variant from (Hoffman 2017), showing that our framework can outperform those approaches in similar experimental settings. Finally, as a visual inspection of the quality of reconstruction from the VAE trained with the VIS framework, Figure 4 displays ten random samples of reconstructed digit images.

![Figure 4: Top row: original images. Bottom row: reconstructed images using VIS-5-10 at 10 epochs.](image)

### 5.4 Variational Autoencoder as a deep Bayes Classifier

With the final experiments we show that the VIS framework can deal with more general probabilistic graphical models. Influence diagrams (Howard and Matheson 2005) are one of the most familiar representations of a decision analysis problem. There is a long history on bridging the gap between influence diagrams and probabilistic graphical models (see Shachter 1988, for instance), so developing better tools for Bayesian inference can be automatically used to solve influence diagrams.

We showcase the flexibility of the proposed scheme to solve inference problems in an experiment with a classification task in a high-dimensional setting. As dataset, the MNIST (LeCun et al. 1998) handwritten digit classification task is chosen, in which grey-scale $28 \times 28$ images have to be classified in one of the ten classes $\mathcal{Y} = \{0, 1, \ldots, 9\}$. More concretely, we extend the VAE model to condition it on a discrete variable $y$, leading to the conditional VAE (cVAE). A cVAE defines a decoder distribution $p_{\theta}(x|z, y)$ on an input space $x \in \mathbb{R}^D$ given class label $y \in \mathcal{Y}$ and latent variable $z \in \mathbb{R}^d$. To perform inference, a variational posterior is learned as an encoder $q_{\phi}(z|x, y)$ from a prior $p(z) \sim \mathcal{N}(0, I)$. Leveraging the conditional structure on $y$, we use the generative model as a classifier using Bayes rule:

$$p(y|x) \propto p(y)p(x|y) = p(y) \int p_{\theta}(x|z, y)q_{\phi}(z|x, y)dz$$

$$\approx \frac{1}{K} \sum_{k=1}^{K} p_{\theta}(x|z^{(k)}, y)p(y)$$  \hspace{1cm} (7)

where we use $K$ Monte Carlo samples $z^{(k)} \sim q_{\phi}(z|x, y)$. In the experiments we set $K = 5$. Given a test sample $x$, the label $\hat{y}$ with highest probability $p(y|x)$ is predicted. Figure 5 in Appendix depicts the corresponding influence diagram. Additional details regarding the model architecture and hyperparameters can be found in Appendix B.

### 6 Conclusion

We have proposed a flexible and efficient framework to perform inference in probabilistic programs. We have shown that the scheme can be easily implemented under the probabilistic programming paradigm and used to efficiently perform inference in a wide class of models: state space time series, variational autoencoders and influence diagrams, defined with continuous, high-dimensional distributions.

Our framework can be seen as a general way of tuning MCMC sampler parameters, adapting the initial distributions and the learning rate. Section 5. Key to the success and applicability of the VIS framework is the Dirac Delta approximation of the refined variational approximation, which is computationally cheap but convenient. Better estimates of the refined density and its gradient may be a fruitful line of research, such as the spectral estimator from (Shi, Sun, and Zhu 2018).

### Acknowledgments

VG acknowledges support from grant FPU16-05034. DRI is grateful to the MINECO MTM2014-56949-C3-1-R project and the AXA-ICMAT Chair in Adversarial Risk Analysis. All authors acknowledge support from the Severo Ochoa Excellence Programme SEV-2015-0554.

We have proposed a flexible and efficient framework to perform inference in probabilistic programs. We have shown that the scheme can be easily implemented under the probabilistic programming paradigm and used to efficiently perform inference in a wide class of models: state space time series, variational autoencoders and influence diagrams, defined with continuous, high-dimensional distributions.

For comparison purposes, we perform various experiments changing $T$ for the transition distribution $Q_{\eta, T}$ in the refined variational approximation. Results are in Table 4. We report the test accuracy achieved at the end of training. Note we are comparing different values of $T$ depending on being on the training or testing phases (in the latter, where the model and variational parameters are kept frozen). The model with $T_{tr} = 5$ was trained for 10 epochs, whereas the other settings for 15 epochs, in order to give all settings similar training times. Results are averaged from 3 runs with different random seeds. From the results it is clear that the effect of using the refined variational approximation (the cases when $T > 0$) is crucially beneficial to achieve higher accuracy. The effect of learning a good initial distribution and inner learning rate by using the gradients $\nabla_{\phi} \text{ELBO}(q)$ and $\nabla_{\eta} \text{ELBO}(q)$ has a highly positive impact in the accuracy obtained.

![Table 4: Results on digit classification task using a deep Bayes classifier.](image)

| $T_{tr}$ | $T_{te}$ | Acc. (test)  |
|---------|---------|-------------|
| 0       | 0       | 96.5 ± 0.5% |
| 0       | 10      | 97.7 ± 0.7% |
| 5       | 10      | 99.8 ± 0.2% |

This material was based upon work partially supported by the Statistical and Applied Mathematical Sciences Institute.
References

[Andrieu, Doucet, and Holenstein 2010] Andrieu, C.; Doucet, A.; and Holenstein, R. 2010. Particle Markov chain monte carlo methods. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 72(3):269–342.

[Bingham et al. 2018] Bingham, E.; Chen, J. P.; Jankowiak, M.; Obermeyer, F.; Pradhan, N.; Karalekos, T.; Singh, R.; Szerlip, P.; Horsfall, P.; and Goodman, N. D. 2018. Pyro: Deep universal probabilistic programming. arXiv preprint arXiv:1810.09538.

[Bouchacourt, Tomioka, and Nowozin 2018] Bouchacourt, D.; Tomioka, R.; and Nowozin, S. 2018. Multi-level variational autoencoder: Learning disentangled representations from grouped observations. In Thirty-Second AAAI Conference on Artificial Intelligence.

[Carpenter et al. 2017] Carpenter, B.; Gelman, A.; Hoffman, M. D.; Lee, D.; Goodrich, B.; Betancourt, M.; Brubaker, M.; Guo, J.; Li, P.; and Riddell, A. 2017. Stan: A probabilistic programming language. Journal of statistical software 76(1).

[Chen et al. 2018] Chen, C.; Li, C.; Chen, L.; Wang, W.; Pu, Y.; and Carin, L. 2018. Continuous-time flows for efficient inference and density estimation.

[Cremer, Li, and Duvenaud 2018] Cremer, C.; Li, X.; and Duvenaud, D. 2018. Inference suboptimality in variational autoencoders. arXiv preprint arXiv:1801.03558.

[Feng, Wang, and Liu 2017] Feng, Y.; Wang, D.; and Liu, Q. 2017. Learning to draw samples with amortized stein variational gradient descent. arXiv preprint arXiv:1707.06626.

[Gallego and Insua 2018] Gallego, V., and Insua, D. R. 2018. Stochastic gradient mcmc with repulsive forces. arXiv preprint arXiv:1812.00071.

[Gneiting and Raftery 2007] Gneiting, T., and Raftery, A. E. 2007. Strictly proper scoring rules, prediction, and estimation. Journal of the American Statistical Association 102(477):359–378.

[Goodfellow et al. 2014] Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. In Advances in neural information processing systems, 2672–2680.

[Hamilton 1994] Hamilton, J. D. 1994. State-space models. Handbook of ecomometrics 4:3039–3080.

[Hoffman et al. 2018] Hoffman, M. D.; Sountsov, P.; Dillon, J.; Langmore, I.; Tran, D.; and Vasudevan, S. 2018. Neutralizing bad geometry in hamiltonian monte carlo using neural transport.

[Hoffman 2017] Hoffman, M. D. 2017. Learning deep latent Gaussian models with Markov chain Monte Carlo. In Precup, D., and Teh, Y. W., eds., Proceedings of the 34th International Conference on Machine Learning, volume 70 of Proceedings of Machine Learning Research, 1510–1519. International Convention Centre, Sydney, Australia: PMLR.

[Howard and Matheson 2005] Howard, R. A., and Matheson, J. E. 2005. Influence diagrams. Decision Analysis 2(3):127–143.

[Huszár 2017] Huszár, F. 2017. Variational inference using implicit distributions. arXiv preprint arXiv:1702.08235.

[Keeling 2005] Keeling, C. D. 2005. Atmospheric carbon dioxide record from mauna loa.

[Kingma and Ba 2014] Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[Kingma and Welling 2013] Kingma, D. P., and Welling, M. 2013. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114.

[Kucukelbir et al. 2017] Kucukelbir, A.; Tran, D.; Ranganath, R.; Gelman, A.; and Blei, D. M. 2017. Automatic differentiation variational inference. The Journal of Machine Learning Research 18(1):430–474.

[Li et al. 2016a] Li, C.; Chen, C.; Carlson, D.; and Carin, L. 2016a. Preconditioned stochastic gradient langevin dynamics for deep neural networks. In Thirtieth AAAI Conference on Artificial Intelligence.

[Li and Chen 2019] Li, C., and Chen, C. 2019. Communication efficient stochastic gradient mcmc for neural networks. In AAAI 2019.

[Li et al. 2016b] Li, C.; Chen, C.; Fan, K.; and Carin, L. 2016b. High-order stochastic gradient thermostats for bayesian learning of deep models. In Thirtieth AAAI Conference on Artificial Intelligence.

[Howard and Matheson 2005] Howard, R. A., and Matheson, J. E. 2005. Influence diagrams. Decision Analysis 2(3):127–143.

[Huszár 2017] Huszár, F. 2017. Variational inference using implicit distributions. arXiv preprint arXiv:1702.08235.

[Keeling 2005] Keeling, C. D. 2005. Atmospheric carbon dioxide record from mauna loa.

[Kingma and Ba 2014] Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[Kingma and Welling 2013] Kingma, D. P., and Welling, M. 2013. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114.

[Kucukelbir et al. 2017] Kucukelbir, A.; Tran, D.; Ranganath, R.; Gelman, A.; and Blei, D. M. 2017. Automatic differentiation variational inference. The Journal of Machine Learning Research 18(1):430–474.

[LeCun et al. 1998] LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P.; et al. 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11):2278–2324.

[Li and Chen 2019] Li, C., and Chen, C. 2019. Communication efficient stochastic gradient mcmc for neural networks. In AAAI 2019.

[Li et al. 2016a] Li, C.; Chen, C.; Carlson, D.; and Carin, L. 2016a. Preconditioned stochastic gradient langevin dynamics for deep neural networks. In Thirtieth AAAI Conference on Artificial Intelligence.

[Li et al. 2016b] Li, C.; Chen, C.; Fan, K.; and Carin, L. 2016b. High-order stochastic gradient thermostats for bayesian learning of deep models. In Thirtieth AAAI Conference on Artificial Intelligence.

[Liu and Wang 2016] Liu, Q., and Wang, D. 2016. Stein variational gradient descent: A general purpose bayesian inference algorithm. In Advances In Neural Information Processing Systems, 2378–2386.

[Lunn et al. 2000] Lunn, D. J.; Thomas, A.; Best, N.; and Spiegelhalter, D. 2000. Winbugs-a bayesian modelling framework: concepts, structure, and extensibility. Statistics and computing 10(4):325–337.

[Mandt, Hoffman, and Blei 2017] Mandt, S.; Hoffman, M. D.; and Blei, D. M. 2017. Stochastic gradient descent as approximate bayesian inference. The Journal of Machine Learning Research 18(1):4873–4907.

[Murray et al. 2018] Murray, L.; Lundén, D.; Kudlicka, J.; Bromann, D.; and Schön, T. B. 2018. Delayed sampling and automatic rao-blackwellization of probabilistic programs. In International Conference on Artificial Intelligence and Statistics (AISTATS).

[Nalisnick, Hertel, and Smyth 2016] Nalisnick, E.; Hertel,
L.; and Smyth, P. 2016. Approximate inference for deep latent gaussian mixtures.

[Neal and others 2011] Neal, R. M., et al. 2011. MCMC using Hamiltonian dynamics. Handbook of Markov Chain Monte Carlo 2(11):2.

[Papaspiliopoulos, Roberts, and Sköld 2007] Papaspiliopoulos, O.; Roberts, G. O.; and Sköld, M. 2007. A general framework for the parametrization of hierarchical models. Statistical Science 59–73.

[Parno and Marzouk 2014] Parno, M., and Marzouk, Y. 2014. Transport map accelerated markov chain monte carlo. arXiv preprint arXiv:1412.5492.

[Paszke et al. 2017] Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in pytorch.

[Rabiner 1989] Rabiner, L. R. 1989. A tutorial on hidden markov models and selected applications in speech recognition. Proceedings of the IEEE 77(2):257–286.

[Rezende and Mohamed 2015] Rezende, D., and Mohamed, S. 2015. Variational inference with normalizing flows. In International Conference on Machine Learning, 1530–1538.

[Ruiz and Titsias 2019] Ruiz, F., and Titsias, M. 2019. A contrastive divergence for combining variational inference and MCMC. In Chaudhuri, K., and Salakhutdinov, R., eds., Proceedings of the 36th International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning Research, 5537–5545. Long Beach, California, USA: PMLR.

[Salimans, Kingma, and Welling 2015] Salimans, T.; Kingma, D.; and Welling, M. 2015. Markov chain monte carlo and variational inference: Bridging the gap. In International Conference on Machine Learning, 1218–1226.

[Shachter 1988] Shachter, R. D. 1988. Probabilistic inference and influence diagrams. Operations Research 36(4):589–604.

[Shi, Sun, and Zhu 2018] Shi, J.; Sun, S.; and Zhu, J. 2018. A spectral approach to gradient estimation for implicit distributions. In International Conference on Machine Learning, 4651–4660.

[Titsias and Ruiz 2019] Titsias, M. K., and Ruiz, F. 2019. Unbiased implicit variational inference. In Chaudhuri, K., and Sugiyama, M., eds., Proceedings of Machine Learning Research, volume 89 of Proceedings of Machine Learning Research, 167–176. PMLR.

[Tran et al. 2018] Tran, D.; Hoffman, M. W.; Moore, D.; Suter, C.; Vasudevan, S.; and Radul, A. 2018. Simple, distributed, and accelerated probabilistic programming. In Advances in Neural Information Processing Systems, 7609–7620.

[Tran, Ranganath, and Blei 2015] Tran, D.; Ranganath, R.; and Blei, D. M. 2015. The variational gaussian process. arXiv preprint arXiv:1511.06499.

[Wang et al. 2018] Wang, W.; Pu, Y.; Verma, V. K.; Fan, K.; Zhang, Y.; Chen, C.; Rai, P.; and Carin, L. 2018. Zero-shot learning via class-conditioned deep generative models. In Thirty-Second AAAI Conference on Artificial Intelligence.

[Welling and Teh 2011] Welling, M., and Teh, Y. W. 2011. Bayesian learning via stochastic gradient langevin dynamics. In Proceedings of the 28th International Conference on Machine Learning (ICML-11), 681–688.

[Yin and Zhou 2018] Yin, M., and Zhou, M. 2018. Semi-implicit variational inference. arXiv preprint arXiv:1805.11183.

[Zarchan and Musoff 2013] Zarchan, P., and Musoff, H. 2013. Fundamentals of Kalman filtering: a practical approach. American Institute of Aeronautics and Astronautics, Inc.
A  Alternative variational approximation

We introduced the VIS posterior as
\[ q_{\phi,\eta}(z|x) = \int Q_{\eta,\tau}(z|z_0)q_{\phi,\eta}(z_0|x)dz_0. \]

In order to compute the ELBO, we sample \( z^1, \ldots, z^K \sim q_{\phi,\eta}(z|x) \), and then use \( \tilde{q}_{\phi,\eta}(z|x) = \frac{1}{K} \sum_{i=1}^{K} \delta(z - z^i) \) as the variational approximation. Though cheap to compute and competitive in our experiments, there are settings where it could be helpful to have a posterior approximation that places density over the whole latent space. For the particular case of using SGD as the inner kernel, we have
\[ z_0 \sim q_{0,\phi}(z_0|x) = N(z_0|\mu_{\phi}(x), \sigma_{\phi}(x)) \]
\[ z_i = z_{i-1} + \eta \nabla \log p(x, z_{i-1}), \quad i = 1, \ldots, T. \]

By treating the gradient terms as points, we have that the refined variational approximation can be computed as
\[ q_{\phi,\eta}(z|x) = N(z|z_T, \sigma_{\phi}(x)). \]

Note that there is an implicit dependence on \( \eta \) through \( z_T \).

B  Experiment details

B.1  State-space models

Initial experiments  For the HMM, both the emission and transition probabilities are Categorical distributions, taking values in the domain \( \{0, 1, 2, 3, 4\} \).

The equations of the DLM are given by
\[ z_{t+1} \sim \mathcal{N}(0.5z_t + 1.0, \sigma_{tr}) \]
\[ x_t \sim \mathcal{N}(3.0z_t + 0.5, \sigma_{em}) \]
with \( z_0 = 0.0 \).

Prediction task in a DLM  The DLM model is comprised of a linear trend component plus a seasonal block of period 12. The trend is specified as
\[ x_t = \mu_t + \epsilon_t \quad \epsilon_t \sim \mathcal{N}(0, \sigma_{obs}) \]
\[ \mu_t = \mu_{t-1} + \delta_{t-1} + \epsilon_t' \quad \epsilon_t' \sim \mathcal{N}(0, \sigma_{level}) \]
\[ \delta_t = \delta_{t-1} + \epsilon_t'' \quad \epsilon_t'' \sim \mathcal{N}(0, \sigma_{slope}). \]

With respect to the seasonal component, the main idea is to cycle the state: suppose \( \theta_i \in \mathbb{R}^p \), with \( p \) being the seasonal period. Then, at each timestep, the model focuses on the first component of the state vector:
\[ (\alpha_1, \alpha_2, \ldots, \alpha_p) \xrightarrow{\text{next period}} (\alpha_2, \alpha_3, \ldots, \alpha_p, \alpha_1). \]

Thus, we can specify the seasonal component via:
\[ x_t = F\theta_t + v_t \]
\[ \theta_t = G\theta_{t-1} + w_t \]
where \( F \) is a \( p \)-dimensional vector and \( G \) is a \( p \times p \) matrix such that
\[ G = \begin{bmatrix} 0 & 0 & \ldots & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \vdots \\ 0 & 0 & \ldots & 1 & 0 \end{bmatrix} \]
and \( F = (1, 0, \ldots, 0, 0) \).

B.2  VAE

class VAE(nn.Module):
    def __init__(self):
        super(VAE, self).__init__()
        self.x_d = 28*28
        self.h_d = 200
        self.z_d = 28*28

    self.fc1 = nn.Linear(self.x_d, self.h_d)
    self.fc1_mu = nn.Linear(self.h_d, self.z_d)
    self.fc1_cov = nn.Linear(self.h_d, self.z_d)
    self.fc2 = nn.Linear(self.z_d, self.h_d)
    self.fc2_mu = nn.Linear(self.h_d, self.z_d)
    self.fc2_cov = nn.Linear(self.h_d, self.z_d)
    self.fc3 = nn.Linear(self.z_d, self.x_d)
    self.fc4 = nn.Linear(self.x_d, self.h_d)

    def encode(self, x):
        h1_mu = F.relu(self.fc1_mu(x))
        h1_cov = F.relu(self.fc1_cov(x))
        h1_mu = F.relu(self.fc12_mu(h1_mu))
        h1_cov = F.relu(self.fc12_cov(h1_cov))
        return self.fc2_mu(h1_mu),
                torch.log(F.softplus(self.fc2_cov(h1_cov)))

    def decode(self, z):
        h3 = F.relu(self.fc3(z))
        return torch.sigmoid(self.fc4(h3))

Figure 5: Model architecture for the cVAE.

Model details  The VAE model is implemented with PyTorch (Paszke et al. 2017). The prior distribution \( p(z) \) for the latent variables \( z \in \mathbb{R}^p \) is a standard factorized Gaussian. The decoder distribution \( p_y(x|z) \) and the encoder distribution (initial variational approximation) \( q_{\phi,\eta}(z|x, y) \) are parameterized by two feed-forward neural networks whose details can be checked in Figure 5.

Hyperparameter settings  The optimizer Adam is used in all experiments, with a learning rate \( \lambda = 0.001 \). We also set \( \eta = 0.001 \). We train for 15 epochs (MNIST) and 20 epochs (MNIST), in order to achieve similar performance to the explicit VAE case in (Titsias and Ruiz 2019). For the VIS-5-10 setting, we train for only 10 epochs, to allow for a fair computational comparison (similar computing times).

B.3  CVAE

Model details  The cVAE model is implemented with PyTorch (Paszke et al. 2017). The prior distribution \( p(z) \) for the latent variables \( z \in \mathbb{R}^{10} \) is a standard factorized Gaussian. The decoder distribution \( p_y(x|y, z) \) and the encoder distribution (initial variational approximation) \( q_{\phi,\eta}(z|x, y) \) are parameterized by two feed-forward neural networks whose details can be checked in Figure 7. The integral (7) is approximated with 1 MC sample from the variational approximation in all experimental settings.
class cVAE(nn.Module):
    def __init__(self):
        super(cVAE, self).__init__()

        self.z_d = 10
        self.h_d = 200
        self.x_d = 28*28
        num_classes = 10

        self.fc1_mu = nn.Linear(self.x_d + num_classes, self.h_d)
        self.fc1_cov = nn.Linear(self.x_d + num_classes, self.h_d)
        self.fc12_mu = nn.Linear(self.h_d, self.h_d)
        self.fc12_cov = nn.Linear(self.h_d, self.h_d)
        self.fc2_mu = nn.Linear(self.h_d, self.z_d)
        self.fc2_cov = nn.Linear(self.h_d, self.z_d)
        self.fc3 = nn.Linear(self.z_d + num_classes, self.h_d)
        self.fc32 = nn.Linear(self.h_d, self.h_d)
        self.fc4 = nn.Linear(self.h_d, self.x_d)

    def encode(self, x, y):
        h1_mu = F.relu(self.fc1_mu(torch.cat([x, y], dim=-1)))
        h1_cov = F.relu(self.fc1_cov(torch.cat([x, y], dim=-1)))
        h1_mu = F.relu(self.fc12_mu(h1_mu))
        h1_cov = F.relu(self.fc12_cov(h1_cov))
        # we work in the logvar-domain
        return self.fc2_mu(h1_mu),
               torch.log(F.softplus(self.fc2_cov(h1_cov)))

    def decode(self, z, y):
        h3 = F.relu(self.fc3(torch.cat([z, y], dim=-1)))
        h3 = F.relu(self.fc32(h3))
        return torch.sigmoid(self.fc4(h3))

---

**Hyperparameter settings**  The optimizer Adam is used in all the experiments, with a learning rate $\lambda = 0.01$. We set the initial $\eta = 5e - 5$. 

---

Figure 6: Influence Diagram for the deep Bayes classifier.

Figure 7: Model architecture for the cVAE.