4th Symposium on Advances in Approximate Bayesian Inference, 2022 1–15

Embedded-model flows: Combining the inductive biases of model-free deep learning and explicit probabilistic modeling

Anonymous Authors
Anonymous Institution

Abstract

Normalizing flows have shown great success as general-purpose density estimators. However, many real world applications require the use of domain-specific knowledge, which normalizing flows cannot readily incorporate. We propose *embedded-model flows* (EMF), which alternate general-purpose transformations with *structured* layers that embed domain-specific inductive biases. These layers are automatically constructed by converting user-specified differentiable probabilistic models into equivalent bijective transformations. We also introduce *gated structured layers*, which allow bypassing the parts of the models that fail to capture the statistics of the data. We show that EMFs enable a high performance form of variational inference where the structure of the prior model is embedded in the variational architecture. In our experiments, we show that this approach outperforms state-of-the-art methods in common structured inference problems.

1. Introduction

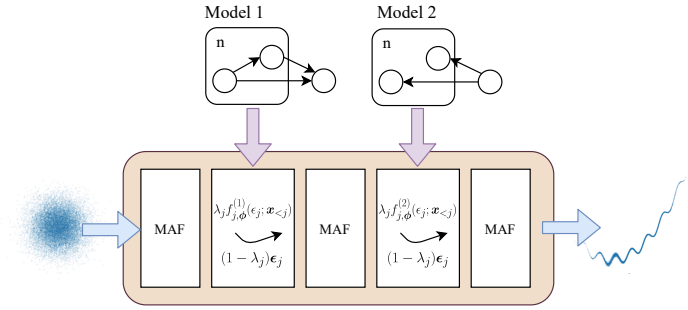

Figure 1: Didactic diagram of a hypothetical embedded-model architecture with model-free masked autoregressive flow (MAF) layers and two gated layers corresponding to two different probabilistic models.

Normalizing flows have emerged in recent years as a strong framework for high-dimensional density estimation, a core task in modern machine learning (Papamakarios et al., 2021; Kobyzev et al., 2020). As with other deep generative architectures such as GANs and VAEs, normalizing flows are commonly designed with the flexibility to model a very general class of distributions. This works well for large naturalistic datasets, such as those common in computer vision. However, general-purpose flows are less appealing when datasets are small or when we possess strong *a priori* knowledge regarding the relationships between variables.

On the other hand, differentiable (or 'deep') probabilistic programming is a powerful and general framework for explicit stochastic modeling that can express strong assumptions

© A. Authors.

about the data-generating process (Tran et al., 2017, 2016; Bingham et al., 2019; Dillon et al., 2017; Piponi et al., 2020; Kucukelbir et al., 2017; Ambrogioni et al., 2021a). For example, the user may know that the data represents a process evolving over time or in space, or is subject to known physical laws, or that there is an unobserved common cause or other shared factor behind a set of observations. Such structured prior information can enable the user to form effective inferences using much less data than would otherwise be required. For example, in the field of astrophysics differentiable probabilistic programs have been successfully used to study strong gravitational lensing (Chianese et al., 2020) and dark matter substructures (Coogan et al., 2020; Varma et al., 2020). All of these applications entail the translation of sophisticated physical theories into highly parameterized and structured probabilistic programs. However, it remains the case that "all models are wrong" (Box, 1976), and it is rare that a modeler can be fully confident in the strong assumptions (e.g., Gaussianity, statistical independence) expressed by typical parametric probability models.

In this paper we bridge these two worlds, providing an automatic technique to convert differentiable probabilistic programs into equivalent normalizing flow layers with domain-specific inductive biases. We call these architectures *embedded-model flow s* (EMF). EMFs can be use to integrate the structure of "wrong but useful" models within generic deep learning architectures capable of correcting the deviation of the model from the data. Furthermore, in the context of Bayesian inference, we show that the EMF architecture can be used to define a powerful class of approximate posterior distributions that embed the structure of the prior.

2. Converting probabilistic programs into normalizing flows

In this section, we will show that a large class of differentiable probabilistic programs can be converted into equivalent normalizing flow layers. More precisely, we will construct an invertible function $F_{\boldsymbol{\phi}}$ that maps spherical normal random inputs to the joint distribution of the probabilistic program:

$$\boldsymbol{x} = F_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}) \sim p(\boldsymbol{x}; \boldsymbol{\phi}). \qquad (\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})) \tag{1}$$

In practice the inputs may be arbitrary real-valued vectors, for example, the output of a previous flow layer, which likely will not be normally distributed. In such cases the transformed values will not be samples from the original probabilistic program, but we expect the transformation to capture some of its properties, such as multimodality and the dependence structure of its random variables. Appendix A.3 describes the single variable transformation, while the next section presents a simple approach to construct structured bijections (i.e., flows) from multivariate distributions expressed as probabilistic programs.

2.1. From differentiable probabilistic programs to structured layers

A probabilistic program samples a sequence of variables in turn, where (as in eq. (8)) the distribution of each variable is conditioned on the previously-sampled variables. At the sampling site of the j-th random variable $\boldsymbol{x}_j$, we therefore have all the information necessary to represent its conditional distribution as an invertible transformation $f_{j,\boldsymbol{\phi}}(\boldsymbol{\epsilon}_j; \boldsymbol{x}_{<j})$ of a standard normal input $\boldsymbol{\epsilon}_j$. Feeding the result back into the model defines a sequence of local

transformations, which together form a joint flow layer $F_\phi(\boldsymbol{\epsilon}_{1:n})$ (pseudocode in A.4). This layer transforms a spherical Gaussian variable $\boldsymbol{\epsilon}_{1:n}$ into a variable that follows the same distribution of the input program.

As with autoregressive flows, the forward transformation of the j-th input requires us to have computed the preceding outputs $\boldsymbol{x}_{<j} = F^j_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}_{<j})$. This may be expressed recursively as follows:

$$[F_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}_{1:n})]_j = f_{j,\boldsymbol{\phi}}(\boldsymbol{\epsilon}_j; F^j_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}_{<j})), \tag{2}$$

In the reverse direction, the variables $\boldsymbol{x}_{1:n}$ are given, so all dimensions may be evaluated in parallel:

$$\left[F^{-1}_{\boldsymbol{\phi}}(\boldsymbol{x}_{1:n})\right]_j = f^{-1}_{j,\boldsymbol{\phi}}(\boldsymbol{x}_j; \boldsymbol{x}_{<j}) \tag{3}$$

This is a generalization of the inversion formula used in autoregressive flows (Kingma et al., 2016; Papamakarios et al., 2017). The Jacobian of the transformation is block triangular (up to a permutation of the variables) and, consequently, the log determinant is just the sum of the log determinants of the block diagonal entries:

$$\log \left|\det J^{-1}_{\phi}(\boldsymbol{x}_{1:n})\right| = \sum_j \log \left|\det J^{-1}_{\phi,j}(\boldsymbol{x}_j; \boldsymbol{x}_{<j})\right| \tag{4}$$

where $J^{-1}_{\phi,j}(\boldsymbol{x}_j; \boldsymbol{x}_{<j})$ is the Jacobian of the local inverse transformation $f^{-1}_{\boldsymbol{\phi}}(\boldsymbol{x}_j; \boldsymbol{x}_{<j})$. When used in a normalizing flow, we denote the transformation $F_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}_{1:n})$ as a *structured layer*.

Gated structured layers Most user-specified models offer a simplified and imperfect description of the data. In order to allow the flow to skip the problematic parts of the model, we consider *gated* layers. In a gated layer, we take each local bijective transformation $g_{j,\boldsymbol{\phi}}$ to be a convex combination of the original transformation $f_{j,\boldsymbol{\phi}}$ and the identity mapping:

$$g_{j,\boldsymbol{\phi}}(\epsilon_j; \boldsymbol{x}_{<j}, \lambda_j) = \lambda_j f_{j,\boldsymbol{\phi}}(\epsilon_j; \boldsymbol{x}_{<j}) + (1 - \lambda_j)\epsilon_j \tag{5}$$

This gating technique is conceptually similar to the method used in highway networks (Srivastava et al., 2015), recurrent highway networks (Zilly et al., 2017) and, more recently, highway flows (Ambrogioni et al., 2021b). Here the gates $\lambda_j \in (0, 1)$ are capable of decoupling each node from its parents, bypassing part of the original structure of the program. As before, the local inverses $g^{-1}_{\boldsymbol{\phi}}(\boldsymbol{x}_j; \boldsymbol{x}_{<j}, \lambda_j)$ can in general be computed by numeric root search, but the Gaussian case admits a closed form in both directions:

$$g_{\mu,\sigma}(\epsilon, \lambda) = (1 + \lambda(\sigma - 1))\epsilon + \lambda\mu; \qquad g^{-1}_{\mu,\sigma}(x, \lambda) = \frac{x - \lambda\mu}{1 + \lambda(\sigma - 1)} \tag{6}$$

Analogues of eq. (2), eq. (3), and eq. (4) then define the gated joint transformation $G_{\boldsymbol{\lambda},\boldsymbol{\phi}}$, its inverse and log Jacobian determinant, respectively. Note that the gating affects only the diagonal of the Jacobian, so the overall block-triangular structure is unchanged.

2.2. Automation

All the techniques described in this section can be fully automated in modern deep probabilistic programming frameworks such as TensorFlow Probability. A user-specified probabilistic

program in the form given by Eq. 8 is recursively converted into a bijective transformation as shown in the code in Fig. 3. In TensorFlow Probability, the *bijector* abstraction uses pre-implemented analytic inverse and Jacobian formulas and can be adapted to revert to root finding methods otherwise. We will release the full TensorFlow Probability code for automatic structured layer construction prior to publication.

2.3. Embedded-model flows

A EMF is a normalizing flow architecture that contains one or more (gated) structured layers. EMFs can be used to combine the inductive biases of explicit models with those of regular normalizing flow architectures. Several probabilistic programs can be embedded in a EMF both in parallel or sequentially. In the parallel case, a single flow layer is comprised by two or more (gated) structured layers acting on non-overlapping subsets of variables. The use of several programs embedded in a single architecture allows the model to select the most appropriate structure during training by changing the gating parameters and the downstream/upstream transformations. This allows the user to start with a "wrong but useful model" and then learn the deviation from the distribution of the collected data using flexible generic normalizing flow transformations. A visualization of a hypothetical EMF architecture is given in Fig. 1.

3. Related work

Embedded-model flows may be seen as a generalization of autoregressive flows (Kingma et al., 2016; Papamakarios et al., 2017) in the special case where the probabilistic program consists of a Gaussian autoregression parameterized by deep link networks, although generic autoregressive architectures often permute the variable order after each layer so that the overall architecture does not reflect any specific graphical structure. Graphical normalizing flows (Wehenkel and Louppe, 2021) generalize autoregressive structure to arbitrary DAG structures; however, this approach only constrains the conditional independence structure and does not embed the forward pass of a user-designed probabilistic program.

In the last few years, the use of tailored model-based normalizing flow architectures has gained substantial popularity. Perhaps the most successful example comes from physics, where gauge-equivariant normalizing flows are used to sample from lattice field theories in a way that guarantees preservation of the intricate symmetries of fundamental fields (Albergo et al., 2019; Kanwar et al., 2020; Albergo et al., 2021; Nicoli et al., 2021). Similarly, structured flows have been used to model complex molecules (Satorras et al., 2021), turbulence in fluids (Bezgin and Adams, 2021), classical dynamical systems (Rezende et al., 2019; Li et al., 2020) and multivariate timeseries (Rasul et al., 2020). These approaches are tailored to specific applications and their design requires substantial machine learning and domain expertise.

Normalizing flows have been applied to variational inference since their introduction by Rezende and Mohamed (2015). Recognizing that structured posteriors can be challenging for generic architectures, a number of recent approaches attempt to exploit the model structure. Structured conditional continuous normalizing flows (Weilbach et al., 2020) use minimally faithful inversion to constrain the posterior conditional independence structure, rather than embedding the forward pass of the prior model. Our architecture can be seen as a flexible extension of previous variational models that also embed the prior structure,

such as stochastic structured variational inference (Hoffman and Blei, 2015), automatic structured variational inference (Ambrogioni et al., 2021a) and cascading flows (Ambrogioni et al., 2021b). It is also closely related to the non-centering transformations applied by Gorinova et al. (2020) to automatically reparameterize a probabilistic program, although that work focused on improving inference geometry and did not directly exploit the bijective aspect of these reparameterizations.

4. Applications

Model-embedding allows users of all levels of expertise to insert domain knowledge into normalizing flow architectures. We will discuss the application of EMF to automatic structured variational inference, where we use EMF architectures that embed the prior distribution. Details about the datasets used, the models' hyperparameters and the training procedures can be found in Appendix A.5, A.6 and A.7 respectively.

4.1. Variational inference

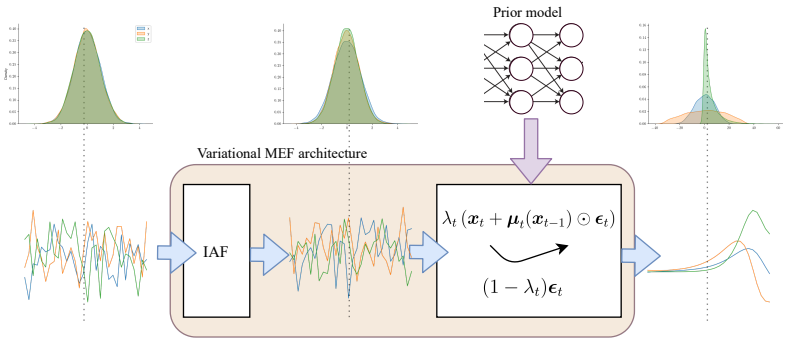

Figure 2: Diagram of a gated prior-embedding architecture with a sample of transformed variables for a Lorentz dynamics smoothing problem.

Normalizing flows are often used as surrogate posterior distributions for variational inference in Bayesian models, where the target posterior may contain dependencies arising both from the prior structure and from conditioning on observed data via collider (or "explaining away") phenomena. Generic flow architectures must learn the prior structure from scratch, which wastes parameters and computation and may lead the optimization into suboptimal local minima. Embedding the prior model into the flow itself as a gated structured layer avoids this difficulty and should allow the remaining layers to focus their capacity on modeling the posterior.

We deploy a gated EMF architecture with a two-layers of inverse autoregressive flow (IAF) and a final structured layer embedding the prior model. This is visualized in figure 2 for inference in a Lorentz dynamical system. In addition, we train plain IAF, automatic differentiation Mean Field (MF) and Multivariate Normal (MVN) surrogate posteriors from Kucukelbir et al. (2017) and ASVI from Ambrogioni et al. (2021a). We experiment on both time series and hierarchical models. The time series models are Brownian motion (BR) and a Lorenz system (LZ), with Bernoulli (classification) emissions. Furthermore, we either

observe all the emissions (smoothing setting) or we omit a central time window (bridge setting). As hierarchical models we use the Eight Schools (ES) dataset and a Gaussian binary tree model with different depths and link functions, in which only the last node is observed (Ambrogioni et al., 2021b). The results are shown in Table 1. Results on additional experiments can be found in Appendix A.9.1. GEMF-T outperforms the baselines on all the problems, showing the benefits of embedding the prior program for highly structured problems, and the capability of such architecture to model collider dependencies.

Table 1: Results in negative ELBO. We report mean and standard error of the mean over ten different runs. S and B correspond respectively to smoothing and bridge, while c stands classification. As an example, BRB-c is the Brownian motion with the bridge-classification settings. For the binary tree experiments, Lin and Tanh correspond to the link function (Lin is linear link), and the following number is the depth of the tree.

	GEMF-T	MF	MVN	ASVI	IAF
BRS-c	$\mathbf{15.882 \pm 1.405}$	23.764 ± 1.262	16.148 ± 1.394	15.894 ± 1.403	15.950 ± 1.393
BRB-c	$\mathbf{11.880 \pm 0.990}$	20.174 ± 0.883	12.098 ± 0.985	11.884 ± 0.987	11.947 ± 0.985
LZS-c	$\mathbf{8.317 \pm 1.033}$	60.458 ± 0.511	43.024 ± 0.621	26.296 ± 2.008	27.604 ± 0.717
LZB-c	$\mathbf{5.819 \pm 0.492}$	53.418 ± 0.394	36.063 ± 0.449	19.318 ± 1.722	20.778 ± 0.542
ES	$\mathbf{36.140 \pm 0.004}$	36.793 ± 0.039	36.494 ± 0.014	36.793 ± 0.039	36.169 ± 0.007
Lin-8	$\mathbf{2.596 \pm 0.213}$	108.869 ± 0.467	13.834 ± 0.271	26.232 ± 0.340	3.440 ± 0.246
Tanh-8	$\mathbf{1.626 \pm 0.159}$	144.668 ± 0.349	44.230 ± 0.258	14.241 ± 0.650	4.127 ± 0.220

5. Discussion

We introduced EMF, a technique to combine domain-specific inductive biases with normalizing flow architectures. Our method automatically converts differentiable probabilistic programs into bijective transformations, allowing users to easily embed their knowledge into their models. We showed how we achieved state-of-the-art results on a series of structured variational inference problems. **Limitations:** While we can map a very large family of probabilistic models to normalizing flow architectures, the approach is still limited by the topological constraints of bijective differentiable transformations (Cornish et al., 2020). This forces the support of the program to be homeomorphic to the range, making impossible to map models with non-connected support sets into normalizing flows. Another limitation is that, while the root finding methods used for inverting univariate densities are reliable and relatively fast, they do have higher computational demands than closed form inversion formulas. **Future work:** Embedded-model flows are a generic tool that can find application in several domains in which there is structure in the data. In theory, any number of structured layers can be added to EMF architectures, and combining different structures in the same model may help model complex dependencies in the data. Further empirical work is needed to asses the capacity of trained EMFs to select the correct (or the least wrong) model. While we only considered the use of structured layers in normalizing flows, they may also be used in any kind of deep net in order to embed domain knowledge in tasks such as classification, regression and reinforcement learning.

References

M. S. Albergo, G. Kanwar, and P.E. Shanahan. Flow-based generative models for markov chain monte carlo in lattice field theory. *Physical Review D*, 100(3):034515, 2019.

M. S. Albergo, D. Boyda, D. C. Hackett, G. Kanwar, K. Cranmer, S. Racanière, D. J. Rezende, and P. E. Shanahan. Introduction to normalizing flows for lattice field theory. *arXiv preprint arXiv:2101.08176*, 2021.

L. Ambrogioni, K. Lin, E. Fertig, S. Vikram, M. Hinne, D. Moore, and M. Gerven. Automatic structured variational inference. *International Conference on Artificial Intelligence and Statistics*, 2021a.

L. Ambrogioni, G. Silvestri, and M. van Gerven. Automatic variational inference with cascading flows. *International Conference on Machine Learning*, 2021b.

J. E. Angus. The probability integral transform and related results. *SIAM review*, 36(4): 652–654, 1994.

D. A. Bezgin and N. A. Adams. Normalizing flows as a novel pdf turbulence model. *arXiv preprint arXiv:2101.03590*, 2021.

E. Bingham, J. P Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman. Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research*, 20:973–978, 2019.

G. E. P. Box. Science and statistics. *Journal of the American Statistical Association*, 71 (356):791–799, 1976.

T. R. Chandrupatla. A new hybrid quadratic/bisection algorithm for finding the zero of a nonlinear function without using derivatives. *Advances in Engineering Software*, 28(3): 145–149, 1997.

M. Chianese, A. Coogan, P. Hofma, S. Otten, and C. Weniger. Differentiable strong lensing: Uniting gravity and neural nets through differentiable probabilistic programming. *Monthly Notices of the Royal Astronomical Society*, 496(1):381–393, 2020.

A. Coogan, K. Karchev, and C. Weniger. Targeted likelihood-free inference of dark matter substructure in strongly-lensed galaxies. *arXiv preprint arXiv:2010.07032*, 2020.

R. Cornish, A. Caterini, G. Deligiannidis, and A. Doucet. Relaxing bijectivity constraints with continuously indexed normalising flows. *International Conference on Machine Learning*, 2020.

J. V. Dillon, I. Langmore, D. Tran, E. Brevdo, S. Vasudevan, D. Moore, B. Patton, A. Alemi, M. Hoffman, and R. A. Saurous. Tensorflow distributions. *arXiv preprint arXiv:1711.10604*, 2017.

L. Dinh, D. Krueger, and Y. Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

M. Figurnov, S. Mohamed, and A. Mnih. Implicit reparameterization gradients. *Advances in Neural Information Processing Systems*, 2018.

M. Gorinova, D. Moore, and M. Hoffman. Automatic reparameterisation of probabilistic programs. *International Conference on Machine Learning*, 2020.

M. Hoffman and D. Blei. Stochastic structured variational inference. *International Conference on Artificial Intelligence and Statistics*, 2015.

G. Kanwar, M. S. Albergo, D. Boyda, K. Cranmer, D. C. Hackett, S. Racaniere, D. J. Rezende, and P. E. Shanahan. Equivariant flow-based sampling for lattice gauge theory. *Physical Review Letters*, 125(12):121601, 2020.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2014.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. *International Conference on Learning Representations*, 2013.

D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. Improved variational inference with inverse autoregressive flow. *Advances in Neural Information Processing Systems*, 2016.

I. Kobyzev, S. Prince, and M. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

A. Kucukelbir, D. Tran, R. Ranganath, A. Gelman, and D. M. Blei. Automatic differentiation variational inference. *The Journal of Machine Learning Research*, 18:430–474, 2017.

S. Li, C. Dong, L. Zhang, and L. Wang. Neural canonical transformation with symplectic flows. *Physical Review X*, 10(2):021020, 2020.

K. A. Nicoli, C. J. Anders, L. Funcke, T. Hartung, K. Jansen, P. Kessel, S. Nakajima, and P. Stornati. Estimation of thermodynamic observables in lattice field theories with deep generative models. *Physical Review Letters*, 126(3):032001, 2021.

G. Papamakarios, T. Pavlakou, and I. Murray. Masked autoregressive flow for density estimation. *Advances in Neural Information Processing Systems*, 2017.

G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *The Journal of Machine Learning Research*, 22:1–60, 2021.

D. Piponi, D. Moore, and J. V. Dillon. Joint distributions for tensorflow probability. *arXiv preprint arXiv:2001.11819*, 2020.

K. Rasul, A. Sheikh, I. Schuster, U. Bergmann, and R. Vollgraf. Multivariate probabilistic time series forecasting via conditioned normalizing flows. *arXiv preprint arXiv:2002.06103*, 2020.

D. Rezende and S. Mohamed. Variational inference with normalizing flows. *International Conference on Machine Learning*, 2015.

D. J. Rezende, S. Racanière, I. Higgins, and P. Toth. Equivariant hamiltonian flows. *arXiv preprint arXiv:1909.13739*, 2019.

John Salvatier, Thomas V Wiecki, and Christopher Fonnesbeck. Probabilistic programming in python using pymc3. *PeerJ Computer Science*, 2:e55, 2016.

V. Garcia Satorras, Emiel H., Fabian B. F., I. Posner, and M. Welling. E(n) equivariant normalizing flows for molecule generation in 3d. *arXiv preprint arXiv:2105.09016*, 2021.

P. O. J. Scherer. *Computational Physics*. Springer, 2010.

R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

D. Tran, A. Kucukelbir, A. B. Dieng, M. Rudolph, D. Liang, and D. M. Blei. Edward: A library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*, 2016.

D. Tran, M. D. Hoffman, R. A. Saurous, E. Brevdo, K. Murphy, and D. M Blei. Deep probabilistic programming. *International Conference on Learning Representations*, 2017.

Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An introduction to probabilistic programming. *arXiv preprint arXiv:1809.10756*, 2018.

S. Varma, M. Fairbairn, and J. Figueroa. Dark matter subhalos, strong lensing and machine learning. *arXiv preprint arXiv:2005.05353*, 2020.

A. Wehenkel and G. Louppe. Graphical normalizing flows. *International Conference on Artificial Intelligence and Statistics*, 2021.

C. Weilbach, B. Beronov, F. Wood, and W. Harvey. Structured conditional continuous normalizing flows for efficient amortized inference in graphical models. *International Conference on Artificial Intelligence and Statistics*, 2020.

J. Georg Zilly, Rupesh K. S., J. Koutnık, and J. Schmidhuber. Recurrent highway networks. *International Conference on Machine Learning*, 2017.

Appendix A. Appendix

A.1. Preliminaries

Normalizing flows define a random variable as an invertible differentiable transformation $\boldsymbol{x} = f_{\boldsymbol{\phi}}(\epsilon)$ of a base variable $\epsilon \sim p_0(\epsilon)$. In these expressions, $\boldsymbol{\phi}$ are parameters that control the form of the transformation and consequently the distribution of $\boldsymbol{x}$. The transformed (log-)density can be given explicitly using the change of variables formula from probability theory,

$$\log p(\boldsymbol{x}) = \log p_0(f_{\boldsymbol{\phi}}^{-1}(\boldsymbol{x})) - \log |\det J_{\boldsymbol{\phi}}(\boldsymbol{x})|, \tag{7}$$

where $J_{\boldsymbol{\phi}}(\boldsymbol{x})$ is the Jacobian matrix of $f_{\boldsymbol{\phi}}$. The base distribution p_0 is often taken to be standard normal ($\epsilon \sim N(\mathbf{0}, \boldsymbol{I})$). In typical applications, the functional form of $f_{\boldsymbol{\phi}}$ is specified by a deep architecture engineered to guarantee stable and tractable invertibility and tractable (log-)Jacobian computations (Dinh et al., 2014; Rezende and Mohamed, 2015). The parameters $\boldsymbol{\phi}$ are usually trained by stochastic gradient descent to maximize the log-likelihood (given observed data) or evidence lower bound (in the variational inference setting).

A.2. Probabilistic programs and graphical models

Probabilistic programming allows a joint probability distribution over a set of random variables to be specified intuitively as a program that includes random sampling steps (van de Meent et al., 2018). Computations on this distribution, including the joint log-density function, structured inference algorithms, and the bijective transformations described in this paper, may then be derived automatically as program transformations. Recently frameworks have emerged for 'deep' probabilistic programming, which exploit the automatic differentiation and hardware acceleration provided by deep learning frameworks; examples include Pyro (Bingham et al., 2019), PyMC (Salvatier et al., 2016), Edward (Tran et al., 2017), and TensorFlow Probability (Dillon et al., 2017; Piponi et al., 2020). These allow for straightforward implementation of gradient-based inference and parameter learning, and enabling methods such as ours that integrate probabilistic programs with standard 'deep learning' components such as flows.

Probabilistic programs that contain only deterministic control flow may equivalently be viewed as directed graphical models, in which each random variable $\boldsymbol{x}_i$ is a graph node connected by incoming edges to some subset of the previous variables $\boldsymbol{x}_{<i}$. This leads to the following general expression for the joint density of a set of vector-valued variables $\boldsymbol{x}_{1:n} = \{\boldsymbol{x}_j\}_{j=1}^n$:

$$p(\boldsymbol{x}_{1:n}; \boldsymbol{\phi}) = p_0(\boldsymbol{x}_0; \boldsymbol{\theta}_0(\boldsymbol{\phi})) \prod_{j=1}^{n} p_j(\boldsymbol{x}_j | \boldsymbol{\theta}_j(\boldsymbol{x}_{<j}; \boldsymbol{\phi})) \tag{8}$$

Here $\boldsymbol{\theta}_j(\boldsymbol{x}_{<j}; \boldsymbol{\phi})$ denotes the 'link function' associated with each node that takes as input the model parameters $\boldsymbol{\phi}$ and the values of all parent variables (excepting $\boldsymbol{\theta}_0$, where there are no parents), and outputs the parameters of its respective density function.

As a practical matter, flow architectures generally assume layers of fixed size, so the experiments in this paper focus on programs of fixed structure and we use graphical-model notation for simplicity. However, the methods we describe do not require access to an explicit graph structure and could be straightforwardly applied to programs with stochastic control flow.

A.3. Individual random variables as bijective transformations

We first consider the case of a univariate random variable $x \sim p_{\boldsymbol{\phi}}(x)$, which we can express as a continuous bijective transformation $f_{\boldsymbol{\phi}}$ of a standard normal variable ϵ via a slight extension of inverse transform sampling. The probability integral transform theorem states that the cumulative distribution function (CDF) of $p_{\boldsymbol{\phi}}$, $C_{\boldsymbol{\phi}}(x) = \int_a^y p_{\boldsymbol{\phi}}(y)\mathrm{d}y$, is uniformly distributed, $u = C_{\boldsymbol{\phi}}(x) \sim \mathcal{U}(0, 1)$ (Angus, 1994). If the CDF is invertible, i.e., strictly

increasing within its domain, it follows that we can sample $x = C_{\boldsymbol{\phi}}^{-1}(u) \sim p_{\boldsymbol{\phi}}$ given only a uniform random variable $u \sim \mathcal{U}(0,1)$. We obtain u through a second application of the probability integral transform theorem to the standard normal CDF $\Phi_{0,1}$. Our univariate 'flow' is therefore given by

$$x = f_{\boldsymbol{\phi}}(\epsilon) = C_{\boldsymbol{\phi}}^{-1}(\Phi_{0,1}(\epsilon)) \qquad (\epsilon \sim \mathcal{N}(0,1)) \tag{9}$$

Closed-form expressions for $f_{\boldsymbol{\phi}}$ may be available in some cases (see below). In the general univariate setting where the inverse distribution function $C_{\boldsymbol{\phi}}^{-1}(u)$ is not available in closed form, it may be evaluated efficiently using numerical root-finding methods such as the method of bisection (see Appendix A.8), with derivatives obtained via the inverse function theorem and the implicit reparameterization trick (Figurnov et al., 2018). However, this is not needed for maximum likelihood training of the parameters $\boldsymbol{\phi}$, since eq. (7) only involves $f_{\boldsymbol{\phi}}^{-1}$ and its derivatives. **Example: Gaussian Variables** The inverse CDF of a Gaussian variable $x \sim \mathcal{N}(\mu, \sigma^2)$ can be expressed as a scale-shift transformation of the inverse CDF of a standard normal variable: $\Phi_{\mu,\sigma}^{-1}(u) = \mu + \sigma\Phi_{0,1}^{-1}(u)$. We can obtain the forward flow transformation by composing this function with the CDF of a standard Gaussian: $f_{\mu,\sigma}(\epsilon) = \Phi_{\mu,\sigma}^{-1}(\Phi_{0,1}(\epsilon)) = \mu + \sigma\epsilon$. This is the famous reparameterization formula used to train VAEs and other variational methods (Kingma and Welling, 2013). The inverse function is $f_{\mu,\sigma}^{-1}(x) = (x-\mu)/\sigma$ while the log-Jacobian is equal to $\log\sigma$.

It is not generally straightforward to evaluate or invert the CDF of a multivariate distribution. However, the most popular multivariate distributions, such as the multivariate Gaussian, can be expressed as transformations of univariate standard normal variables, and so may be treated using the methods of this section.

A.4. Pseudocode

A.5. Datasets

Here we provide a detailed description of the datasets used in the experiments.

A.5.1. Stochastic differential equations

We generate data using two stochastic differential equations (SDEs) models, discretized with the Euler-Maruyama method. The SDEs are: Brownian motion and Lorenz system. In the following, W_t corresponds to a Wiener process, and the series length $T = 30$ unless specified.

Brownian motion: the dynamics evolve as:

$$\dot{x_t} = \mu x_t dt + \sigma x_t dW_t$$

where μ and σ are constants. The Brownian motion sequence is generated as:

$$\begin{aligned} &x_0 \sim \mathcal{N}(\mu, \sigma) \\ &x_t \sim \mathcal{N}(x_{t-1}, \sigma), \forall t \in [1, \ldots, T-1] \end{aligned}$$

where the mean $\mu = 0$ and the standard deviation $\sigma = 0.1$.

```python
def forward_and_log_det_jacobian(
    model_generator, epsilons, gated=True):
  x, xs = None, []
  forward_log_det_jacobian = 0
  gen = model_generator()
  d = gen.send(None)  # Dist. of first RV.
  for eps in epsilons:
    # Apply the j'th local transformation.
    x, fldj_term = (
      d.as_bijector(gated=gated).
        forward_and_log_det_jacobian(eps))
    xs.append(x)
    forward_log_det_jacobian += fldj_term
    # Run the model to the next sample site.
    d = gen.send(x)  # Needs output `x`.

  return xs, forward_log_det_jacobian
```

```python
def inverse_and_log_det_jacobian(
    model_generator, xs, gated=True):
  epsilons = []
  inverse_log_det_jacobian = 0
  gen = model_generator()
  d = gen.send(None)  # Dist. of first RV.
  for x in xs:
    # Invert the j'th local transformation.
    eps, ildj_term = (
      d.as_bijector(gated=gated).
          inverse_and_log_det_jacobian(x))
    epsilons.append(eps)
    inverse_log_det_jacobian += ildj_term
    # Run the model to the next sample site.
    d = gen.send(x)  # `eps` is unused.

  return epsilons, inverse_log_det_jacobian
```

Figure 3: Python code implementing both directions of the structured joint transformation $F_\phi(\boldsymbol{\epsilon}_{1:n})$. The probabilistic program is specified as a generator that `yield`s the conditional distribution of the current random variable after receiving the realized value of the previous variable. Each distribution instance implements the method `as_bijector` (not shown), which follows appendix A.3 to represent itself as a bijective transformation of a standard normal variable.

Lorenz system: the Lorenz dynamics evolve as follows:

$$\begin{aligned}\dot{x}_t &= \phi(y_t - x_t)\\ \dot{y}_t &= x_t(\rho - z_t) - y_t\\ \dot{z}_t &= x_t y_t - \beta z_t\end{aligned}$$

where $\phi = 10$, $\rho = 28$ and $\beta = \frac{8}{3}$. In the discrete case we have:

$$\begin{aligned}x_0, y_0, z_0 &\sim \mathcal{N}(0,1)\\ \forall t \in &[1,\ldots,T-1]\\ x_t &\sim \mathcal{N}(x_0 + s(\phi(y_{t-1} - x_{t-1})), \sqrt{s} * \sigma)\\ y_t &\sim \mathcal{N}(y_0 + s(x_{t-1}(\rho - z_{t-1}) - y_{t-1}), \sqrt{s} * \sigma)\\ z_t &\sim \mathcal{N}(z_0 + s(x_{t-1}y_{t-1} - \beta z_{t-1}), \sqrt{s} * \sigma)\end{aligned}$$

with standard deviation $\sigma = 0.1$ and step size $s = 0.02$.

A.5.2. Smoothing and Bridge data

The data is generated with either a Brownian mmotion or a Lorenz system. In both cases, the true time series remains unobserved, and we only have access to noisy emissions. In the regression case, the emissions follow a Gaussian distribution:

$$e_t \sim \mathcal{N}(x_t, \sigma_e)$$

where σ_e is the emission noise, and is $\sigma_e = 0.15$ for the Brownian motion and $\sigma_e = 1$ for the Lorenz system. In the classification case, the emissions follow a Bernoulli distribution:

$$e_t \sim Bernoulli(kx_t)$$

with k a gain parameter, $k = 5$ in the Brownian Motion and $k = 2$ in the Lorenz system. In the smoothing problem, we observe emission for all the time steps, while in the bridge problem we observe emissions only in the first and last 10 time points. For the Lorenz system, the emissions are observed only for the x variables in both smoothing and bridge.

A.5.3. Hierarchical Models

Eight Schools: The Eight Schools (ES) model considers the effectiveness of coaching programs on a standardized test score conducted in parallel at eight schools. It is specified as follows:

$$\begin{aligned} \mu &\sim \mathcal{N}(0, 100) \\ \log \tau &\sim \log \mathcal{N}(5, 1) \\ \theta_i &\sim \mathcal{N}(\mu, \tau^2) \\ y_i &\sim \mathcal{N}(\theta_i, \sigma_i^2) \end{aligned}$$

where μ represents the prior average treatment effect, τ controls how much variance there is between schools, $i = 1, \dots, 8$ is the school index, and y_i and σ_i are observed.

Gaussian Binary tree: The Gaussian Binary tree is a reverse tree of D layers, in which the variables at a given layer d, x_j^d, are sampled from a Gaussian distribution where the mean is function of two parent variables $\pi_{j,1}^{d-1}, \pi_{j,2}^{d-1}$ at the previous layer:

$$x_j^d \sim \mathcal{N}(link(\pi_{j,1}^{d-1}, \pi_{j,2}^{d-1}), \sigma^2)$$

All the variables at the 0-th layers are sampled from a Gaussian with mean 0, and variance $\sigma^2 = 1$. The last node of the tree is observed, and the inference problem consists in computing the posterior of the nodes at the previous layers. We use a linear coupling function $link(x, y) = x - y$ and a nonlinear one $link(x, y) = \tanh(x) - \tanh(y)$, with trees of depth $D = 4$ and $D = 8$.

A.6. Flow Models

In all the experiments, we use IAF with two autoregressive layers and a standard Gaussian as a base density. Each autoregressive layer has two hidden layers with 512 units each. We use ReLU nonlinearity for all the models but for IAF without EMF or GEMF, for which we empirically found Tanh activation to be more stable. The features permutation always happen between the two autoregressive layers. We found empirically that training the gates for GEMF is slow as the variables lie on an unbounded space before the Sigmoid activation. A possible solution is scaling the variables (by 100 in our experiments) before applying Sigmoid, which speeds-up the training for the gates, leading to better results.

A.7. Training details

In this section we describe the training procedure for the experiments. We train all the models with Adam optimizer (Kingma and Ba, 2014).

Variational inference: we fit all the surrogate posteriors for 100000 iterations with full-batch gradient descent, using a 50-samples Monte Carlo estimate of the ELBO, and learning rate 1e-3 for all the methods but IAF, EMF-T and GEMF-T, for which we use 5e-5. For GMEF, the gates are initialized to be close to the prior, whit a value of 0.999 after Sigmoid activation.

A.8. Numerical root finding methods

There are cases in which a closed form inversion for the transformation induced by EMF is not available, such as for the Mixture of Gaussian case. In those cases, we can still compute the inverse function and log-Jacobian determinants by using numerical root finding methods. In this work, we use either the secant method or the Chandrupatla's method (Chandrupatla, 1997; Scherer, 2010), as the implementation is available in the probabilistic programming framework Tensorflow Probability.

A.8.1. Secant method

The secant method is a root-finding algorithm that uses a succession of roots of secant lines to better approximate a root of a function. The secant method can be thought of as a finite-difference approximation of Newton's method. The secant method starts with two initial values x_0 and x_1 which should be chosen to lie close to the root, and then uses the following recurrent relation:

$$\begin{aligned} x_n &= x_{n-1} - f(x_{n-1})\frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})} \\ &= \frac{x_{n-2}f(x_{n-1}) - x_{n-1}f(x_{n-2})}{f(x_{n-1}) - f(x_{n-2})} \end{aligned}$$

A.8.2. Chandrupatla's method

This root-finding algorithm is guaranteed to converge if a root lies within the given bounds. At each step, it performs either bisection or inverse quadratic interpolation. The specific procedure can be found in (Chandrupatla, 1997; Scherer, 2010).

A.9. Additional results

A.9.1. VI experiments

As additional variational inference experiments, we show the results on the time series with Gaussian (regression) emissions and a shallow version of the Gaussian binary tree model with depth 4. The results are shown in tables 2. We also compute the forward KL divergence for each model. Such results are reported in table 3.

The EMF bijective transformation can be also used in combination with models which do not use normalizing flows. We combine EMF with the mean field and multivariate normal

approximations from (Kucukelbir et al., 2017), with results in table 4. We also report results of the non gated version of EMF.

Table 2: Results in negative ELBO. We report mean and standard error of the mean over ten different runs

	GEMF-T	MF	MVN	ASVI	IAF
BRS-r	-3.341 ± 0.977	0.147 ± 1.002	-3.126 ± 0.977	$\mathbf{-3.354 \pm 0.974}$	-3.304 ± 0.974
BRB-r	-3.122 ± 0.967	1.932 ± 0.951	-2.866 ± 0.967	$\mathbf{-3.142 \pm 0.966}$	-3.086 ± 0.961
LZS-r	$\mathbf{46.981 \pm 0.476}$	1492.572 ± 255.358	1485.622 ± 251.296	1257.942 ± 330.467	1337.364 ± 301.002
LZB-r	$\mathbf{110.500 \pm 40.958}$	756.338 ± 146.012	748.298 ± 146.063	490.696 ± 166.080	598.402 ± 153.412
Lin-4	$\mathbf{1.513 \pm 0.576}$	6.136 ± 0.573	1.542 ± 0.571	3.268 ± 0.561	1.526 ± 0.581
Tanh-4	$\mathbf{-0.035 \pm 0.119}$	6.640 ± 0.157	0.075 ± 0.124	2.866 ± 0.113	-0.026 ± 0.118

Table 3: Results in forward KL divergence. We report mean and standard error of the mean over ten different runs.

	GEMF-T	MF	MVN	ASVI	IAF
BRS-r	37.362 ± 1.049	32.332 ± 2.051	$\mathbf{37.608 \pm 1.122}$	37.346 ± 1.091	37.319 ± 1.101
BRS-c	28.539 ± 0.920	-32.940 ± 9.829	28.174 ± 0.958	$\mathbf{28.571 \pm 0.913}$	28.458 ± 0.896
BRB-r	32.403 ± 0.762	11.608 ± 6.311	31.800 ± 0.921	$\mathbf{32.449 \pm 0.748}$	32.267 ± 0.769
BRB-c	$\mathbf{26.928 \pm 1.940}$	-76.227 ± 48.668	26.582 ± 2.004	26.825 ± 1.946	26.816 ± 1.906
LZS-r	$\mathbf{245.458 \pm 2.404}$	$-1.4e^{+08} \pm 2.3e^{+07}$	$-7.6e^{+06} \pm 1.8e^{+06}$	$-3.6e^{+04} \pm 1.3e^{+04}$	$-1.0e^{+08} \pm 6.4e^{+07}$
LZS-c	$\mathbf{245.337 \pm 1.577}$	$-1.7e^{+08} \pm 1.9e^{+07}$	$-1.1e^{+07} \pm 1.4e^{+06}$	$-3.e^{+04} \pm 7.9e^{+03}$	$-1.2e^{+06} \pm 7.1e^{+05}$
LZB-r	$\mathbf{148.488 \pm 51.783}$	$-1.2e^{+08} \pm 2.9e^{+07}$	$-6.9e^{+06} \pm 1.9e^{+06}$	$-4.2e^{+04} \pm 2.e^{+04}$	$-2.1e^{+06} \pm 1.1e^{+06}$
LZB-c	$\mathbf{249.709 \pm 1.348}$	$-1.8e^{+08} \pm 2.1e^{+07}$	$-1.1e+^{07} \pm 1.5e^{+06}$	$-3.5e+^{04} \pm 6.9e^{+03}$	$-7.7e+^{06} \pm 3.3e^{+06}$
ES	-13.032 ± 0.910	-13.541 ± 1.204	-13.687 ± 1.294	-13.533 ± 1.201	$\mathbf{-12.876 \pm 0.937}$
Lin-4	$\mathbf{5.780 \pm 0.495}$	-26.195 ± 8.401	5.739 ± 0.518	-0.020 ± 2.165	5.763 ± 0.553
Lin-8	$\mathbf{89.049 \pm 2.772}$	-1387.313 ± 201.479	77.042 ± 3.843	-0.475 ± 25.415	88.506 ± 2.930
Tanh-4	17.386 ± 1.111	-24.730 ± 13.715	$\mathbf{17.404 \pm 1.081}$	12.408 ± 3.369	17.316 ± 1.150
Tanh-8	$\mathbf{311.528 \pm 3.827}$	-2941.246 ± 364.753	255.127 ± 9.701	239.432 ± 17.222	304.161 ± 5.220

A.9.2. Surrogate posteriors

We show and compare the obtained surrogate posteriors using IAF, EMF-T and GMEF-T, together with the ground truth and the observations (figures 4 and 5).

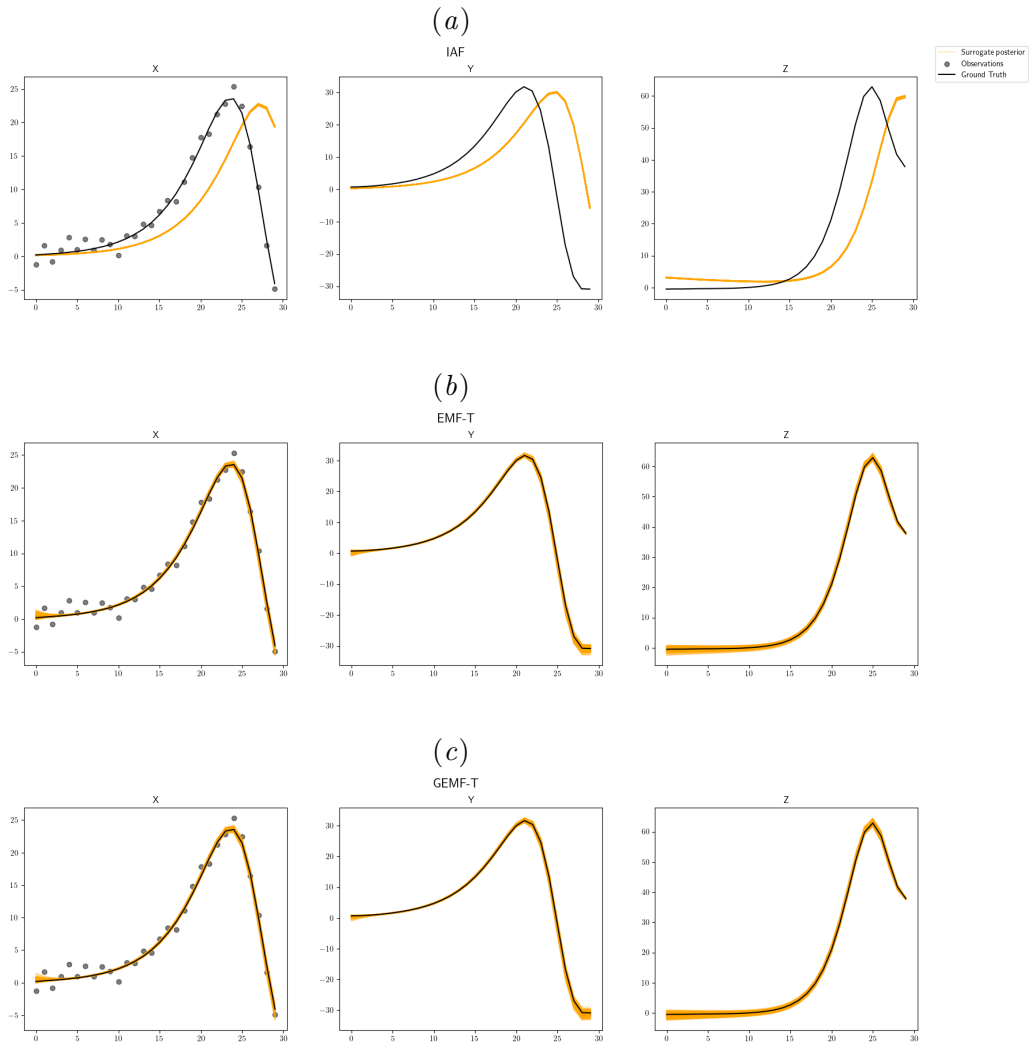

Figure 4: Surrogate posterior for Lorenz Smoothing.

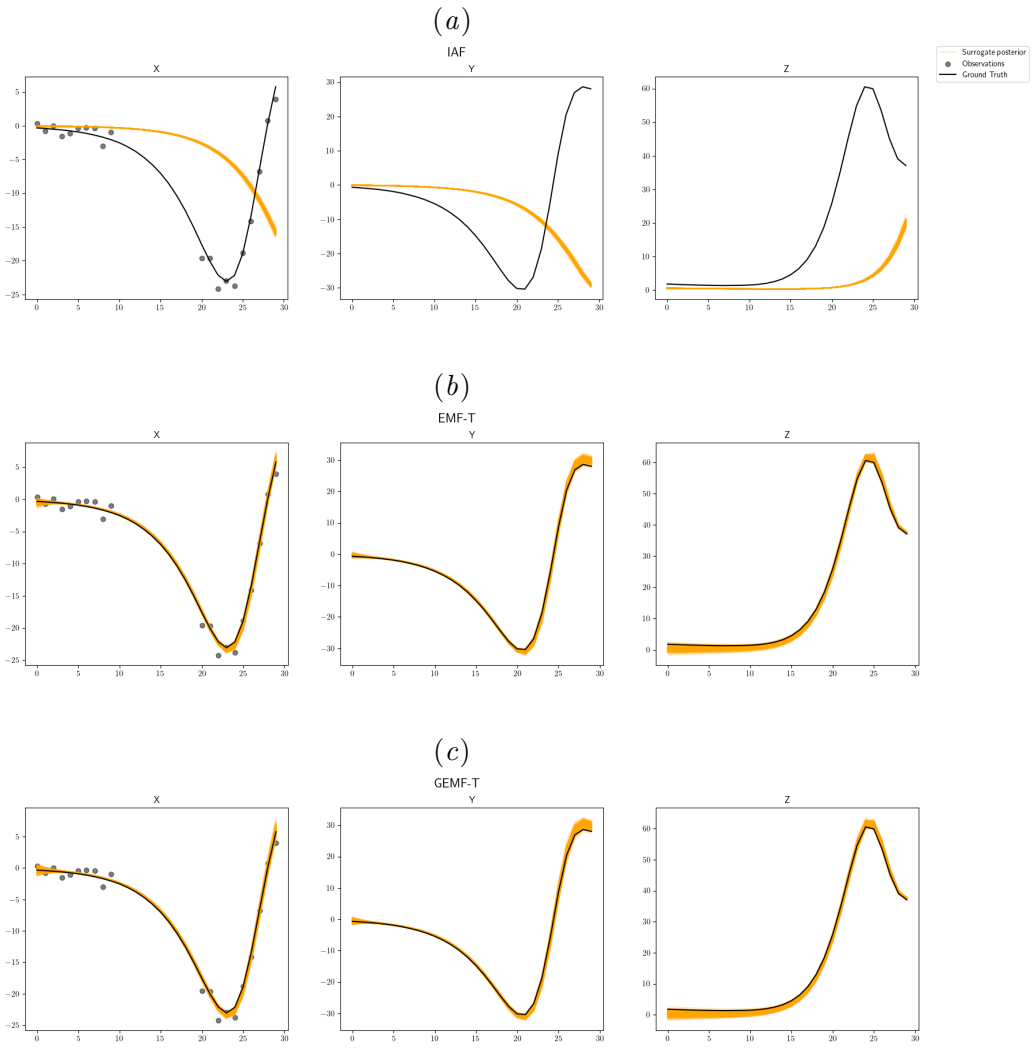

Figure 5: Surrogate posterior for Lorenz Bridge.

Table 4: Results in variational inference for additional models: a non gated EMF-T, and the combination of Mean Field and Multivariate Normal with EMF-T and GEMF-T.

		EMF-T	MF-EMF-T	MVN-EMF-T	MF-GEMF-T	MVN-GEMF-T
BRS-r	-ELBO	-3.293 ± 0.974	13.905 ± 1.224	-3.123 ± 0.968	-3.087 ± 0.967	-3.077 ± 0.966
	FKL	37.483 ± 1.067	-6.429 ± 5.058	32.458 ± 0.775	32.261 ± 0.779	32.351 ± 0.822
BRS-c	-ELBO	15.894 ± 1.406	18.602 ± 1.723	15.917 ± 1.397	15.960 ± 1.399	15.921 ± 1.403
	FKL	28.554 ± 0.907	27.117 ± 1.373	28.562 ± 0.906	28.500 ± 0.907	28.386 ± 0.938
BRB-r	-ELBO	-3.096 ± 0.971	13.905 ± 1.224	-3.123 ± 0.968	-3.150 ± 0.967	-3.077 ± 0.966
	FKL	32.291 ± 0.856	-6.429 ± 5.058	32.458 ± 0.775	32.426 ± 0.764	32.351 ± 0.822
BRB-c	-ELBO	11.878 ± 0.990	13.838 ± 1.251	11.912 ± 0.985	11.940 ± 0.988	11.930 ± 0.988
	FKL	27.002 ± 1.885	25.407 ± 2.120	26.880 ± 1.914	26.674 ± 2.041	26.845 ± 1.924
LZS-r	-ELBO	46.995 ± 0.450	527.145 ± 319.063	1255.087 ± 330.588	52.865 ± 0.694	1254.725 ± 330.517
	FKL	245.505 ± 2.401	-8660.092 ± 5474.940	$-4.8e+04 \pm 1.9e+04$	-1976.326 ± 2022.022	$-4.6e+04 \pm 1.8e+04$
LZS-c	-ELBO	8.290 ± 1.031	56.846 ± 45.443	23.895 ± 1.942	10.844 ± 1.104	23.540 ± 1.881
	FKL	245.214 ± 1.583	176.425 ± 18.073	$-3.4e+04 \pm 9.4e+03$	192.223 ± 16.425	$-3.7e+04 \pm 1.0e+04$
LZB-r	-ELBO	85.425 ± 36.684	187.988 ± 87.731	487.722 ± 166.150	148.388 ± 57.199	487.486 ± 166.134
	FKL	192.466 ± 41.929	$-1.8e+04 \pm 1.1e+04$	$-5.8e+04 \pm 2.8e+04$	$-1.5e+04 \pm 1.4e+04$	$-5.8e+04 \pm 2.8e+04$
LZB-c	-ELBO	5.800 ± 0.479	39.506 ± 15.506	17.014 ± 1.592	51.010 ± 18.540	16.732 ± 1.557
	FKL	249.858 ± 1.379	-2045.551 ± 2165.953	$-4.4e+04 \pm 9.2e+03$	114.907 ± 46.043	$-4.4e+04 \pm 9.2e+03$
ES	-ELBO	36.139 ± 0.004	36.794 ± 0.040	36.532 ± 0.016	36.798 ± 0.041	36.512 ± 0.019
	FKL	-13.043 ± 0.906	-13.525 ± 1.203	-13.666 ± 1.295	-13.582 ± 1.219	-13.732 ± 1.311
Lin-4	-ELBO	1.512 ± 0.573	5.981 ± 0.520	1.513 ± 0.574	3.371 ± 0.650	1.533 ± 0.572
	FKL	5.806 ± 0.518	1.085 ± 1.070	5.842 ± 0.499	0.002 ± 2.137	5.821 ± 0.528
Lin-8	-ELBO	2.609 ± 0.208	109.551 ± 3.608	5.600 ± 0.260	26.396 ± 0.329	4.173 ± 0.195
	FKL	89.005 ± 2.776	22.826 ± 7.270	87.689 ± 3.056	-2.279 ± 26.566	88.143 ± 2.818
Tanh-4	-ELBO	-0.037 ± 0.119	6.221 ± 0.161	-0.025 ± 0.125	2.876 ± 0.110	-0.002 ± 0.120
	FKL	17.436 ± 1.097	6.913 ± 4.124	17.472 ± 1.069	12.561 ± 3.256	17.244 ± 1.084
Tanh-8	-ELBO	1.866 ± 0.119	19.266 ± 1.790	5.338 ± 0.124	14.134 ± 0.784	4.237 ± 0.194
	FKL	311.594 ± 3.618	295.872 ± 4.505	285.647 ± 7.425	280.296 ± 9.743	303.473 ± 5.529