PhD dissertation

CONTRIBUTIONS TO DEEP GENERATIVE MODELING

FROM BLACK-BOX TO HYBRID APPROACHES

by ANTOINE WEHENKEL

LIÈGE université

JURY MEMBERS

???, Professor at the Université de Liège (President);

GILLES LOUPPE, Professor at the Université de Liège (Advisor);

???, Professor at the Université de Liège;

???, ???;

???, ???;

???, ???;

ABSTRACT

To be completed.

ACKNOWLEDGMENTS

To be completed.

CONTENTS

INTRODUCTION

If you erase this you are almost there! :D

1.1 OUTLINE AND CONTRIBUTIONS

If you erase this you are almost there! :D

1.2 PUBLICATIONS

To be completed.

2

BACKGROUND

OUTLINE

This chapter concerns the definition of unsupervised learning with a brief review of classical methods. Graphical models (in particular B-net are introduced here.) It continues with a review of deep generative modeling, with a discussion between explicit and implicit generative modeling. We introduce the concepts of GANs, VAEs, Normalizing Flows and diffusion models. With a note that VAEs and diffusions models are discussed in more details in further chapters.

2.1 UNSUPERVISED LEARNING

Part I

BLACK-BOX GENERATIVE MODELING

DEEP LATENT VARIABLES GENERATIVE MODELS

OUTLINE

Among likelihood-based approaches for deep generative modelling, variational autoencoders (VAEs) offer scalable amortized posterior inference and fast sampling. However, VAEs are also more and more outperformed by competing models such as normalizing flows (NFs), deep-energy models, or the new denoising diffusion probabilistic models (DDPMs). In this preliminary work, we improve VAEs by demonstrating how DDPMs can be used for modelling the prior distribution of the latent variables. The diffusion prior model improves upon Gaussian priors of classical VAEs and is competitive with NF-based priors. Finally, we hypothesize that hierarchical VAEs could similarly benefit from the enhanced capacity of diffusion priors.

3.1 INTRODUCTION

Over the last few years, the interest of the deep learning community for generative modelling has increased steadily. Among the likelihood-based approaches for deep generative modelling, variational autoencoders [?, VAEs] stand as one of the most popular, although competing approaches now demonstrate better performance. In particular, **???** recently showed that denoising diffusion probabilistic models (DDPMs) are competitive deep generative models, obtaining samples quality similar to those of the best implicit deep generative models such as ProgressiveGAN [?] and StyleGAN [?]. Similarly to VAEs, DDPMs train on a variational bound and may be interpreted under the encoding-decoding framework.

In the original formulation of VAEs, the prior and the posterior distributions over the latent variables are assumed to be both Gaussian. However, these assumptions are often incompatible and are thus limiting the performance of VAEs for complex modelling tasks [**??**]. A natural solution to this problem is to parameterize the prior, sometimes also the posterior, with more expressive distributions. In this preliminary work, we improve VAEs by demonstrating how DDPMs can be used for modelling the prior distribution of the latent variables. In addition to boosting DDPM with the compression properties of VAEs, combining the two models should eventually lead to greater generative performance by enabling complex generative modelling even with simple decoder architecture. Finally, working in the latent

space should eventually reduce the computational burden associated with diffusion generative models.

3.2 VARIATIONAL AUTO ENCODERS

We want to learn a generative model of an unknown distribution $p(x)$ given a dataset $X \in \mathbb{R}^{N \times d}$ of N i.i.d observations x sampled from this unknown distribution. The original VAE postulates a two-step generative process in which some unobserved variables $z \in \mathbb{R}^h$ are first sampled from a prior distribution $p(z)$ and then observations x are generated from a conditional distribution $p_\theta(x|z)$. The generative process can be expressed mathematically as

$$z \sim p(z) \quad \text{and} \quad x \sim p_\theta(x|z). \tag{3.1}$$

The prior $p(z)$ is chosen Gaussian while the likelihood $p_\theta(x|z)$ is modeled with a neural network. The likelihood model decodes latent variables into observations and is thus usually refereed as the decoder in the literature. In its original formulation, the likelihood is parameterized with a multivariate Gaussian $\mathcal{N}(\mu_\theta(z), (\sigma_\theta^2(z)))$ when the observations are continuous, and a categorical distribution when they are discrete.

Training the generative model is achieved by finding the parameters θ of the decoder that maximize the sum of the marginal likelihoods of individual points

$$p_\theta(X) = \sum_{x \in X} \log \int p_\theta(x|z)p(z)dz.$$

These integrals are intractable but the introduction of an encoder network that approximates the posterior distribution $q_\phi(z|x)$ allows maximizing the associated evidence lower bound

$$\text{ELBO} := \mathbb{E}_q \left[\log \frac{p_\theta(x|z)p(z)}{q_\psi(z|x)} \right] \tag{3.2}$$

$$= \log p_\theta(x) - \mathbb{KL} \left[q_\psi(z|x) \| p_\theta(z|x) \right] \tag{3.3}$$

$$\leqslant \log p_\theta(x). \tag{3.4}$$

The ELBO becomes tighter as the approximate posterior $q_\psi(z|x)$ gets closer to the true posterior. Learning the generative model is finally performed by jointly optimizing the parameters θ of the decoder and ϕ of the approximate posterior via stochastic gradient ascent. In the original VAE, the encoder models the approximate posterior as a conditional multivariate Gaussian distribution $\mathcal{N}(\mu_\phi(x), (\sigma_\phi^2(x)))$.

The ELBO loss presents two antagonistic goals to the encoder. It should be able to both encodes the data accurately while being as close as possible to the prior distribution. Consequently, the Gaussian assumptions made on both the prior and the posterior distributions

are often incompatible and limit the generative performance. A possible solution consists in learning a prior distribution that is compatible with the learned posteriors. For example, **?** and **?** respectively showed that autoregressive models and normalizing flows [**?**, NFs] greatly improve the performance of VAEs when used as prior distributions. In the following we present how denoising diffusion probabilistic models can be used to improve the performance of classical VAEs.

3.3 DIFFUSION MODELS

Inspired by non-equilibrium statistical physics, **?** originally introduced DDPMs while **?** demonstrated only more recently how to train these models for image synthesis, achieving results close to the state-of-the-art on this task. DDPMs formulate generative modelling as the reverse operation of diffusion, a physical process which progressively destroys information. Formally, the reverse process is a latent variable model of the form

$$p_\phi(x_0) := \int p_\phi(x_{0:T}) dx_{1:T},$$

where $x_0 := x$ denotes the observations and $x_1, \ldots, x_T$ denote latent variables of the same dimensionality as x_0. The joint distribution $p_\phi(x_{0:T})$ is modelled as a first order Markov chain with Gaussian transitions, that is

$$p_\phi(x_{0:T}) := p_\phi(x_T) \prod_{t=1}^{T} p_\phi(x_{t-1}|x_t), \tag{3.5}$$

$$p_\phi(x_T) := \mathcal{N}(0, I), \tag{3.6}$$

$$p_\phi(x_{t-1}|x_t) := \mathcal{N}(\mu_\phi(x_t, t), \sigma_t^2 I). \tag{3.7}$$

Similar to VAEs, the reverse Markov chain is trained on an ELBO. However, the approximate posterior $q(x_{1:T}|x_0)$ is fixed to a diffusion process that is also a first order Markov chain with Gaussian transitions,

$$q(x_{1:T}|x_0) := \prod_{t=1}^{T} q(x_t|x_{t-1}), \tag{3.8}$$

$$q(x_t|x_{t-1}) := \mathcal{N}(\sqrt{1 - \beta_t} x_{t-1}, \beta_t I), \tag{3.9}$$

where $\beta_1, \ldots, \beta_T$ are the variance schedule that is either fixed as training hyper-parameters or learned. The ELBO is then given by

$$\text{ELBO} := \mathbb{E}_q \left[\log \frac{p_\phi(x_{0:T})}{q(x_{1:T}|x_0)} \right] \leqslant \log p_\phi(x_0). \tag{3.10}$$

Provided that the variance schedule β_t is small and that the number of timesteps T is large enough, the Gaussian assumptions on the

generative process p_ϕ are reasonable. **?** take advantage of the Gaussian transitions to express the ELBO as

$$\mathbb{E}_q \left[\mathbb{KL} \left[q(\mathbf{x}_T|\mathbf{x}_0)\|p(\mathbf{x}_T) \right] - \log p_\phi(\mathbf{x}_0|\mathbf{x}_1) \right.$$

$$\left. + \sum_{t=2}^T \mathbb{KL} \left[q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)\|p_\phi(\mathbf{x}_{t-1}|\mathbf{x}_t) \right] \right]. \tag{3.11}$$

The inner sum in Equation (3.11) is made of comparisons between the Gaussian generative transitions $p_\phi(\mathbf{x}_{t-1}|\mathbf{x}_t)$ and the conditional forward posterior $q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)$ which can also be expressed in closed form as Gaussians $\mathcal{N}(\tilde{\mu}_t(\mathbf{x}_0,\mathbf{x}_t),\tilde{\beta}_t I)$, where $\tilde{\beta}_t$ are functions of the variance schedule. The KL can thus be calculated with closed form expressions which reduces the variance of the final expression. In addition, **?** empirically demonstrate that it is sufficient to take optimization steps on uniformly sampled terms of the sum instead of computing it completely. The final objective closely resembles denoising score matching over multiple noise levels [**?**]. These observations combined with additional simplifications leads to a simplified loss

$$L_{\text{DDPM}}(\mathbf{x}_0;\phi) := \mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_t} \left[\frac{1}{2\sigma_t^2} \|\mu_\phi(\mathbf{x}_t,t) - \tilde{\mu}_t(\mathbf{x}_0,\mathbf{x}_t)\|^2 \right], \tag{3.12}$$

where $\tilde{\mu}_t(\mathbf{x}_0,\mathbf{x}_t)$ is the mean of $q(\mathbf{x}_{t-1}|\mathbf{x}_0,\mathbf{x}_t)$, the forward diffusion posterior conditioned on the observation $\mathbf{x}_0$.

We now introduce our contribution which consists in using a DDPM for modelling the prior distribution in VAEs. We formulate the generative model as

$$\mathbf{z}_T \sim \mathcal{N}(0,I) \tag{3.13}$$

$$\mathbf{z}_{t-1|t} \sim p_\phi(\mathbf{z}_{t-1}|\mathbf{z}_t) \quad \forall t \in [T,\dots,1] \tag{3.14}$$

$$\mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{z}_0), \tag{3.15}$$

where ϕ denotes the parameters of the reverse diffusion model encoding the prior distribution. Equations (3.13) and (3.14) implicitly define a prior distribution over the usual latent variables $\mathbf{z}_0$ which is modelled with a reverse diffusion process.

Unfortunately, we cannot train a VAE with a diffusion prior directly on the ELBO as expressed in Equation (3.2) as $p_\phi(\mathbf{z}_0)$ cannot be evaluated. However, Equation (3.2) can be further developed as

$$\mathbb{E}_{q_\psi} \left[\log p_\theta(\mathbf{x}|\mathbf{z}_0) \right] - \mathbb{E}_{q_\psi} \left[\log q(\mathbf{z}_0|\mathbf{x}) \right] + \mathbb{E}_{q_\psi} \left[\log p_\phi(\mathbf{z}_0) \right] \tag{3.16}$$

in which a lower bound on the last term can be expressed by Equation (3.10). This finally leads to the following expression

$$\mathbb{E}_{q_\psi} \left[\log p_\theta(\mathbf{x}|\mathbf{z}_0) - \log q(\mathbf{z}_0|\mathbf{x}) + \mathbb{E}_q \left[\log \frac{p_\phi(\mathbf{z}_{0:T})}{q(\mathbf{z}_{1:T}|\mathbf{z}_0)} \right] \right] \tag{3.17}$$

$$\leqslant \mathbb{E}_{q_\psi} \left[\log p_\theta(x|z_0) - \log q(z_0|x) + \log p_\phi(z_0) \right] \tag{3.18}$$

$$\leqslant \log p_\theta(x), \tag{3.19}$$

which is a valid ELBO. Finally, the diffusion prior p_ϕ is trained jointly with the approximate posterior q_ψ and the likelihood models p_θ which are optimized as in a classical VAE. This leads to the following loss function:

$$\mathcal{L}(x; \phi, \theta, \psi) := \mathbb{E}_{q_\psi} \left[\log \frac{p_\theta(x|z)}{q_\psi(z|x)} \right] + \mathbb{E}_{q_\psi} \left[L_{\text{DDPM}}(z_0; \phi) \right]. \tag{3.20}$$

3.4 RELATED WORK

Various approaches have been proposed to improve the modelling capacity and the training of VAEs. As a first example, some state-of-the-art deep generative models based on VAEs model the posterior with normalizing flows or autoregressive models [**??**]. Autoregressive models are also often used as a replacement of the original likelihood parameterization, which assumes conditional independencies that are often unrealistic [**?**]. Another popular improvement made to the original VAE is the embedding of structure in the latent variables. In particular, hierarchical VAEs [**??**] combined with careful training demonstrate impressive results on generative modelling for images [**?**].

? concurrently proposed to use diffusion for modelling the prior distributions of VAEs. They obtain state-of-the-art results on image synthesis by combining continuous diffusion models and VAEs. Not as close to our work but related, **?** proposed to learn the prior as a solution to the mismatch between the approximate and the true posteriors. They model the prior with an autoregressive flow, which also closely relates to modelling the posterior distribution with an inverse autoregressive flow [**?**]. **?** takes inspiration from the aggregated posterior $\frac{1}{N} \sum_{i=1}^{N} q_\psi(z|x)$ [**??**] to introduce the VampPrior defined as a mixture of learned pseudo-inputs. An orthogonal line of work suggests that the mismatch between the approximate posterior and the exact posterior can be reduced by over-weighting the terms related to the prior and to the approximate posterior in the ELBO [**??**].

3.5 EXPERIMENTS

We now compare VAEs for different choices of priors, including the original Gaussian prior, an NF prior, and the proposed diffusion prior. All models share a same backbone encoder-decoder architecture inspired from DCGAN [**?**]. Optimization is performed with Adam for 250 epochs with a learning rate set to 0.0005. After each epoch, the models are evaluated on a validation set used to select the best one for each training setting. We compare the models on the CIFAR10

Table 3.1: FID scores for different models for prior modelling in VAEs and for different latent size. *Diffusion priors outperform classical VAE on CelebA but are slightly worse than NFs. FID scores do not reveal the superiority of any method on CIFAR10.*

Dataset	CelebA			CIFAR10		
Latent Size	40	100	200	40	100	200
Gaussian	154.3	149.4	139.1	176.0	126.2	123.9
NF	72.9	59.49	54.7	167.6	129.1	129.6
Diffusion	114.8	67.95	88.3	177.9	160.5	153.1

and CelebA datasets for 3 different latent variables dimensionality (40, 100, 200). The NF used in our experiments is a 3-step autoregressive affine flow with simple MLP backbones similar to the one used to model the transition function of DDPM.

Table 3.1 presents the FID scores for the different models. We first notice the large scores reached by all models on the CIFAR10 dataset. This can be explained by the simplicity of the models trained in our experiments. We believe these scores could be greatly improved by using a more sophisticated likelihood model such as a PixelCNN [**?**]. Although FID scores suggest that the Gaussian prior outperforms the diffusion prior in terms of generative performance, the visual inspection of Figure 3.1 shows that the diffusion prior results in samples slightly more realistic than those of the classical VAE. The best FID score is achieved by the NF prior, although its samples do not seem to reflect this superiority. In this case, we believe the FID scores are not entirely informative about the quality of the images synthesized by the models and should be interpreted with a grain of salt. Although learned priors seem to improve generative performance on CIFAR10, additional work is needed to reach results that would justify using a diffusion prior for this dataset.

On CelebA however, we observe in Table 3.1 that diffusion priors outperform the Gaussian prior. This is in line with the visual inspection of Section 3.5 and Section 3.5. As for CIFAR10, the NF prior outperforms the Gaussian and diffusion priors in terms of FID scores, although the visual inspection of the corresponding samples in Section 3.5 does not reveal a much better quality of images when compared to those resulting from the diffusion prior. We conclude from these observations that diffusion priors offer an interesting alternative to NFs for modelling the prior in a VAE.

3.6   CONCLUSION AND FUTURE WORK

This preliminary work presents how denoising diffusion probabilistic models can be used as a new class of learnable priors for VAEs. As

(a) Gaussian prior   (b) Gaussian prior   (c) Gaussian prior

Figure 3.1: Samples generated by a VAE trained on CIFAR10 for three differ-
ent prior models. *The diffusion prior leads to slightly better sampling
quality than the Gaussian distribution and similar to the NF prior.*

(a) Gaussian prior   (b) Gaussian prior   (c) Gaussian prior
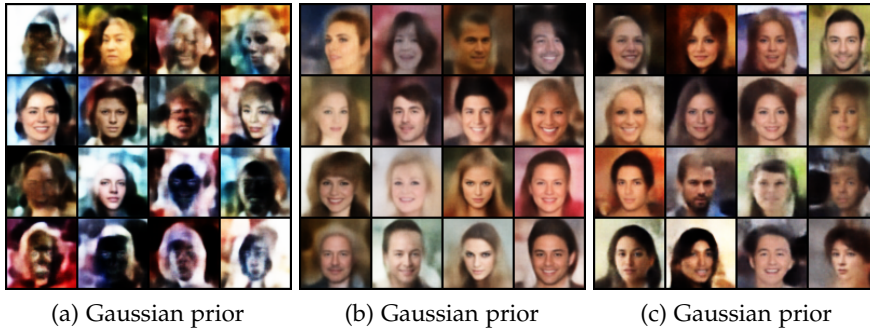
Figure 3.2: Samples generated by a VAE trained on CelebA for three differ-
ent prior models. *The diffusion prior leads to better sampling quality
than the Gaussian distribution and similar to the NF prior.*

a notable contribution, we empirically demonstrate that optimizing implicitly a prior on an ELBO can be performed jointly to training the encoder and the decoder of the VAE. In addition, our results suggest DDPM performs on par with NFs for modelling prior distribution.

A large spectrum of future research directions could benefit from the basic idea expressed in this preliminary work. As an example, recent advances in diffusion models such as the continuous formulation [?] or improvement to the training procedure of DDPM [?] could be implemented in the prior model. Similarly, many improvements could be made to the architectures used for the VAE and to the training procedure. In particular, image synthesis with hierarchical VAEs which organizes the latent variables into multiple scales images could reveal the full potential of diffusion priors. This would indeed combine the structural knowledge embed by such type of VAEs with the impressive performance of DDPM for modelling distributions over images. Finally, diffusion does not constrain the neural networks architectures and so enables the embedding of a larger choice of inductive biases in the prior distribution compared to autoregressive models and NFs.

LIMITATIONS OF AFFINE NORMALIZING FLOWS

OUTLINE

Normalizing flows have emerged as an important family of deep neural networks for modelling complex probability distributions. In this note, we revisit their coupling and autoregressive transformation layers as probabilistic graphical models and show that they reduce to Bayesian networks with a pre-defined topology and a learnable density at each node. From this new perspective, we provide three results. First, we show that stacking multiple transformations in a normalizing flow relaxes independence assumptions and entangles the model distribution. Second, we show that a fundamental leap of capacity emerges when the depth of affine flows exceeds 3 transformation layers. Third, we prove the non-universality of the affine normalizing flow, regardless of its depth.

4.1 INTRODUCTION

Normalizing flows [NF, **?**] have gained popularity in the recent years because of their unique ability to model complex data distributions while allowing both for sampling and exact density computation. This family of deep neural networks combines a base distribution with a series of invertible transformations while keeping track of the change of density that is caused by each transformation.

Probabilistic graphical models (PGMs) are well-established mathematical tools that combine graph and probability theory to ease the manipulation of joint distributions. They are commonly used to visualize and reason about the set of independencies in probabilistic models. Among PGMs, Bayesian networks [BN, **?**] offer a nice balance between readability and modeling capacity. Reading independencies stated by a BN is simple and can be performed graphically with the d-separation algorithm [**?**].

In this note, we revisit NFs as Bayesian networks. We first briefly review the mathematical grounds of these two worlds. Then, for the first time in the literature, we show that the modeling assumptions behind coupling and autoregressive transformations can be perfectly expressed by distinct classes of BNs. From this insight, we show that stacking multiple transformation layers relaxes independencies and entangles the model distribution. Then, we show that a fundamental change of regime emerges when the NF architecture includes 3

transformation steps or more. Finally, we prove the non-universality of affine normalizing flows.

4.2 BACKGROUND

4.2.1 *Normalizing flows*

A normalizing flow is defined as a sequence of invertible transformation steps $g_k : \mathbb{R}^d \to \mathbb{R}^d$ ($k = 1, ..., K$) that are composed together to create an expressive invertible mapping $g = g_1 \circ \cdots \circ g_K : \mathbb{R}^d \to \mathbb{R}^d$. This mapping can be used to perform density estimation, using $g(\cdot; \theta) : \mathsf{R}^d \to \mathsf{R}^d$ to map a sample $x \in \mathbb{R}^d$ to a latent vector $z \in \mathsf{R}^d$ equipped with a density $p_z(z)$. The transformation g implicitly defines a density $p(x; \theta)$ as given by the change of variables formula,

$$p(x; \theta) = p_z(g(x; \theta)) \left| \det J_{g(x; \theta)} \right| ,$$

where $J_{g(x; \theta)}$ is the Jacobian of $g(x; \theta)$ with respect to x. The resulting model is trained by maximizing the likelihood of the data $\{x^1, ..., x^N\}$. NFs can also be used for data generation tasks while keeping track of the density of the generated samples such as to improve the latent distribution in variational auto-encoders [?]. In the rest of this paper, we will not distinguish between g and g_k when the discussion will be focused on only one of these steps g_k.

In general, steps g can take any form as long as they define a bijective map. Here, we focus on a sub-class of normalizing flows for which these steps can be mathematically described as

$$g(x) = \left[g^1(x_1; c^1(x)) \quad \ldots \quad g^d(x_d; c^d(x)) \right] ,$$

where the c^i are denoted as the conditioners and constrain the structure of the Jacobian of g. The functions g^i, partially parameterized by their conditioner, must be invertible with respect to their input variable x_i. These are usually defined as affine or strictly monotonic functions, with the latter being the most general class of invertible scalar continuous functions. In this note, we mainly discuss affine normalizers that can be expressed as $g(x; m, s) = x \exp(s) + m$ where $m \in \mathbb{R}$ and $s \in \mathbb{R}$ are computed by the conditioner.

4.2.2 *Bayesian networks*

Bayesian networks allow for a compact and natural representation of probability distributions by exploiting conditional independence. More precisely, a BN is a directed acyclic graph (DAG) which structure encodes for the conditional independencies through the concept of d-separation [?]. Equivalently, its skeleton supports an efficient factorization of the joint distribution.

[node distance=.5cm and .35cm, mynode/.style=draw,circle,text width=.35cm,align=center, var$_x$/.sty

Figure 4.1: Bayesian networks for single-step normalizing flows on a vector $x = [x_1, x_2, x_3, x_4]^\top$. (a) BN for an autoregressive conditioner. (b) BN for a coupling conditioner. (c) Pseudo BN for a coupling conditioner, with the latent variables shown explicitly. Double circles stand for deterministic functions of the parents and non-directed edges stand for bijective relationships.

[node distance=.5cm and .35cm, mynode/.style=draw,circle,text width=.35cm,align=center, var$_x$/.style $=$ $draw, circle, double, textwidth = .35cm, align = center$] [var$_x$](u1)u$_1$; [var$_x$, right $= $ ofu1](u2)u$_2$; [var$_x$, below $= $ ofu1](u3)u$_3$; [var$_x$, right $= $ ofu3](u4)u$_4$; [mynode, left=of u1] (z1) z$_1$; [mynode, right=of u2] (z2) z$_2$; [mynode, left=of u3] (z3) z$_3$; [mynode, right=of u4] (z4) z$_4$; [var$_x$, right $= $ ofz2](x1)x$_1$; [var$_x$, right $= $ ofx1](x2)x$_2$; [var$_x$, right $= $ ofz4](x3)x$_3$; [var$_x$, right $= $ ofx3](x4)x$_4$; x1)edge[-latex]x3); x1)edge[-latex]x4); x2)edge[-latex]x3); x2)edge[-latex]x4); u\x)edge[bendleft=40]x1);u\x)edge[bendleft=40]x2); u\x)edge[bendleft=-40]x3);u\x)edge[bendleft=-40]x4); u4)edge[-latex]u2); u4)edge[-latex]u1); u3)edge[-latex]u2); u3)edge[-latex]u1); u4)edge[-latex]u1); z\x)edgeu1);z\x)edgeu2);z\x)edgeu3);z\x)edgeu4);

Figure 4.2: A Bayesian network equivalent to a 2-step normalizing flow based on coupling layers. Independence statements are relaxed by the second step.

A BN is able to model a distribution p if and only if it is an I-map with respect to p. That is, iff the set of independencies stated by the BN structure is a subset of the independencies that holds for p. Equivalently, a BN is a valid representation of a random vector x iff its density $p_x(x)$ can be factorized by the BN structure as

$$p_x(x) = \prod_{i=1}^{d} p(x_i | \mathcal{P}_i), \qquad (4.1)$$

where $\mathcal{P}_i = \{j : A_{i,j} = 1\}$ denotes the set of parents of the vertex i and $A \in \{0,1\}^{d \times d}$ is the adjacency matrix of the BN. As an example, fig:mono-step-flows-a is a valid BN for any distribution over x because it does not state any independence, leading to a factorization that results in the chain rule.

4.3   NORMALIZING FLOWS AS BAYESIAN NETWORKS

4.3.1   *Autoregressive conditioners*

Autoregressive conditioners can be expressed as

$$c^i(x) = h^i\left(\begin{bmatrix} x_1 & \dots & x_{i-1} \end{bmatrix}^\top\right),$$

where $h^i : \mathbb{R}^{i-1} \to \mathbb{R}^l$ are functions of the first $i-1$ components of x and whose output size depends on architectural choices. These conditioners constrain the Jacobian of g to be lower triangular, making the computation of its determinant $O(d)$. The multivariate density $p(x;\theta)$ induced by $g(x;\theta)$ and $p_z(z)$ can be expressed as a product of d univariate conditional densities,

$$p(x;\theta) = p(x_1;\theta) \prod_{i=2}^{d} p(x_i|x_{1:i-1};\theta). \tag{4.2}$$

When $p_z(z)$ is a factored distribution $p_z(z) = \prod_{i=1}^{d} p(z_i)$, we identify that each component z_i coupled with the corresponding function g^i encodes for the conditional $p(x_i|x_{1:i-1};\theta)$. An explicit connection between BNs and autoregressive conditioners can be made if we define $\mathcal{P}_i = \{x_1, \dots, x_{i-1}\}$ and compare (7.4) with (7.1). Therefore, and as illustrated in fig:mono-step-flows-a, autoregressive conditioners can be seen as a way to model the conditional factors of a BN that does not state any independence.

4.3.2   *Coupling conditioners*

Coupling conditioners **?** are another popular type of conditioners used in normalizing flows. The conditioners c^i made from coupling layers are defined as

$$c^i(x) = \begin{cases} \underline{h}^i & \text{if } i < k \\ h^i(x_{<k}) & \text{if } i \geqslant k \end{cases}$$

where the $\underline{h}^i$ symbol define constant values. As for autoregressive conditioners, the Jacobian of g made of coupling layers is lower triangular. Assuming a factored latent distribution, the density associated with these conditioners can be written as follows:

$$p(x;\theta) = \prod_{i=1}^{k-1} p(x_i) \prod_{i=k}^{d} p(x_i|x_{<k}),$$

$$\text{where} \quad p(x_i) = p(g^i(x_i;\underline{h}^i)) \frac{\partial g^i(x_i;\underline{h}^i)}{\partial x_i}$$

$$\text{and}\quad p(x_i|x_{<k}) = p(g^i(x_i; h^i(x_{<k})))\frac{\partial g^i(x_i; h^i(x_{<k}))}{\partial x_i}.$$

The factors define valid 1D conditional probability distributions because they can be seen as 1D changes of variables between z_i and x_i. This factorization can be graphically expressed by a BN as shown in fig:mono-step-flows-b. In addition, we can see fig:mono-step-flows-b as the marginal BN of fig:mono-step-flows-c which fully describes the stochastic process modeled by a NF that is made of a single transformation step and a coupling conditioner. In contrast to autoregressive conditioners, coupling layers are not by themselves universal density approximators, even when associated with very expressive normalizers g^i. Indeed, d-separation reveals independencies stated by this class of BN, such as the conditional independence between each pair in $x_{\geqslant k}$ knowing $x_{<k}$. These independence statements do not hold in general.

4.3.3 *Stacking transformation steps*

In practice, the invertible transformations discussed above are often stacked together in order to increase the representation capacity of the flow, with the popular good practice of permuting the vector components between two transformation steps. The structural benefits of this stacking strategy can be explained from the perspective of the underlying BN.

First, a BN that explicitly includes latent variables is faithful as long as the sub-graph made only of those latent nodes is an I-map with respect to their distribution. Normalizing flows composed of multiple transformation layers can therefore be viewed as single transformation flows whose latent distribution is itself recursively modeled by a normalizing flow. As an example, fig:two-steps-flows illustrates a NF made of two transformation steps with coupling conditioners. It can be observed that the latent vector u is itself a normalizing flow whose distribution can be factored out by a class of BN.

Second, from the BN associated to a NF, we observe that additional layers relax the independence assumptions defined by its conditioners. The distribution modeled by the flow gets more entangled at each additional layer. For example, fig:two-steps-flows shows that for coupling layers, the additional steps relax the strong conditional independencies between x_1 and x_2 of the single transformation NF of fig:mono-step-flows-c. Indeed, we can observe from the figure that x_1 and x_2 have common ancestors (z_3 and z_4) whereas they are clearly assumed independent in fig:mono-step-flows-b.

In general, we note that edges between two nodes in a BN do not model dependence, only the absence of edges does model independence. However, because some of the relationship between nodes are bijective, this implies that these nodes are strictly dependent on each

[node distance=.7cm and .7cm, mynode/.style=draw,circle,text
width=.35cm,align=center, minimum width=.9cm, var_x/.style $=$
$draw, circle, double, textwidth = .35cm, align = center, minimum width = .9cm$]
[mynode] (z1) z_1; [mynode, below=of z1] (z2) z_2;
[var_x, right $=$ ofz1](u11)u_1^1; [var_x, right $=$ ofz2](u12)u_2^1;
[var_x, right $=$ ofu11](u21)u_1^2; [var_x, right $=$ ofu12](u22)u_2^2;
[var_x, right $=$ ofu21](x1)x_1; [var_x, right $=$ ofu22](x2)x_2;
z1)edgeu11); z2)edgeu12); z1)edge[-latex]u12);
u11)edgeu21); u12)edgeu22); u12)edge[-latex]u21);
u21)edgex1); u22)edgex2); u21)edge[-latex]x2);

Figure 4.3: The Bayesian network of a three-steps normalizing flow on vector $x = [x_1, x_2]^\top \in \mathbb{R}^4$. It can be observed that the distribution of the intermediate latent variables, and at the end of the vector x, becomes more entangled at each additional transformation step. Considering the undirected edges as affine relationships, we see that while u_1^1 and u_2^1 are affine transformations of the latent z, the vector x cannot be expressed as a linear function of the latent z.

other. We represent these relationships with undirected edges in the BN, as it can be seen in fig:two-steps-flows.

4.4 AFFINE NORMALIZING FLOWS UNLOCK THEIR CAPACITY WITH 3 TRANSFORMATION STEPS

We now show how some of the limitations of affine normalizers can be relaxed by stacking multiple transformation steps. We also discuss why some limitations cannot be relaxed even with a large number of transformation steps. We intentionally put aside monotonic normalizers because they have already been proven to lead to universal density approximators when the conditioner is autoregressive **?**. We focus our discussion on a multivariate normal with an identity covariance matrix as base distribution $p_z(z)$.

We first observe from fig:mono-step-flows that in a NF with a single transformation step at least one component of x is a function of only one latent variable. If the normalizer is affine and the base distribution is normal, then this necessarily implies that the marginal distribution of this component is normal as well, which will very likely not lead to a good fit. We easily see that adding steps relaxes this constraint. A more interesting question to ask is what exactly the modeling capacity gain for each additional step of affine normalizer is. Shall we add steps to increase capacity or shall we increase the capacity of each step instead? We first discuss a simple 2-dimensional case, which has the advantage of unifying the discussion for autoregressive and coupling conditioners, and then extend it to a more general setting.

Affine NFs made of a single transformation step induce strong constraints on the form of the density. In particular, these models implicitly assume that the data distribution can be factorized as a product of conditional normal distributions. These assumptions are rexaled when accumulating steps in the NF. As an example, fig:2d-three-steps-flows shows the equivalent BN of a 2D NF composed of 3 steps. This flow is mathematically described with the following set of equations:

$$u_1^1 := z_1 \qquad u_2^1 := \exp(s_2^1(z_1))z_2 + m_2^1(z_1)$$
$$u_2^2 := u_2^1 \qquad u_1^2 := \exp(s_1^2(u_2^1))z_1 + m_1^2(u_2^1)$$
$$x_1 := u_1^2 \qquad x_2 := \exp(s_2^3(u_1^2))u_2^2 + m_2^3(u_1^2)$$

From these equations, we see that after one step the latent variables u_1^1 and u_2^1 are respectively normal and conditionally normal. This is relaxed with the second step, where the latent variable u_1^2 is a nonlinear function of two random variables distributed normally (by assumption on the distribution of z_1 and z_2). However, u_2^2 is a stochastic affine transformation of a normal random variable. In addition, we observe that the expression of u_1^2 is strictly more expressive than the expression of u_2^2. Finally, x_1 and x_2 are non-linear functions of both latent variables z_1 and z_2. Assuming that the functions s_j^i and m_j^i are universal approximators, we argue that the stochastic process that generates x_1 and the one that generates x_2 are as expressive as each other. Indeed, by making the functions arbitrarily complex the transformation for x_1 could be made arbitrarily close to the transformations for x_2 and vice versa. This is true because both transformations can be seen as an affine transformation of a normal random variables whose scaling and offset factors are non-linear arbitrarily expressive transformations of all the latent variables. Because of this equilibrium between the two expressions, additional steps do not improve the modeling capacity of the flow. The same observations can be made empirically as illustrated in fig:2d-three-steps-flows for 2-dimensional toy problems. A clear leap of capacity occurs from 2-step to 3-step NFs, while having 4 steps or more does not result in any noticeable improvement when s_j^i and m_j^i already have enough capacity.

For $d > 2$, autoregressive and coupling conditioners do not correspond to the same set of equations or BN. However, if the order of the vector is reversed between two transformation steps, the discussion generalizes to any value of d for both conditioners. Indeed, in both cases each component of the intermediate latent vectors can be seen as having a set of conditioning variables and a set of independent variables. At each successive step the indices of the non-conditioning variables are exchanged with the conditioning ones and thus any output vector's component can be expressed either as a component of the vector form of x_1 or of x_2.

[shorten >=1pt,->,draw=black!50, node distance=1.25cm, scale=0.45]
every pin edge=[<-,shorten <=1pt]
neuron=[circle,fill=black!25,minimum size=7pt,inner sep=0pt] input
neuron=[neuron, fill=black!50]; output neuron=[neuron,
fill=black!50]; hidden neuron=[neuron, fill=black!50]; annot = [text
width=4em, text centered] edge/.style = ->,-latex

all_flows.png

[inner sep=0pt] (graphical) at (0,0)   ;
[] () at (-7.2, 5) 1 step; [] () at (-4.4, 5) 2 steps; [] () at (-1.5, 5) 3 steps; []
() at (1.45, 5) 4 steps; [] () at (4.3, 5) 5 steps; [] () at (7.3, 5.08) Universal;

Figure 4.4: Evolution of an affine normalizing flow's capacity as the num-
ber of steps increases. For comparison, the density learned by a
universal density approximator is shown on the last column.

[node distance=.6cm and .6cm, mynode/.style=draw,circle,text
width=.69cm,align=center, minimum width=1.1cm, var_x/.style =
$draw, circle, double, textwidth = .69cm, align = center, minimumwidth = 1.1cm$][my
[var_x, right $= ofzI](u1I)u_I^1$; [var_x, right $= ofu1I](uiI)u_I^i$;
[var_x, right $= ofuiI](uKI)u_I^{K-1}$; [var_x, right $= ofuKI](xI)x_I$;
$zI)edgeu1I); u1I)edge[dashed]uiI); uiI)edge[dashed]uKI)$;
$uKI)edgexI)$;

Figure 4.5: The equivalent BN of a component with a unique latent variable
as ancestor.

4.5 AFFINE NORMALIZING FLOWS ARE NOT UNIVERSAL DENSITY APPROXIMATORS

We argue that affine normalizers do not lead to universal density approximators in general, even for an infinite number of steps. In the following, we assume again that the latent variables are distributed according to a normal distribution with a unit covariance matrix.

To prove the non-universality of affine normalizing flows, one only needs to provide a counter-example. Let us consider the simple setup in which one component x_I of the random vector x is independent from all the other components. Let us also assume that x_I is distributed under a non-normal distribution. We can then consider two cases. First, x_I has only one component of the latent vector z as an ancestor. This implies that the equivalent BN would be as in fig:BN-x-I, hence that x_I is a linear function of this ancestor and is therefore normally distributed. Else, x_I has n components of the latent vector as ancesto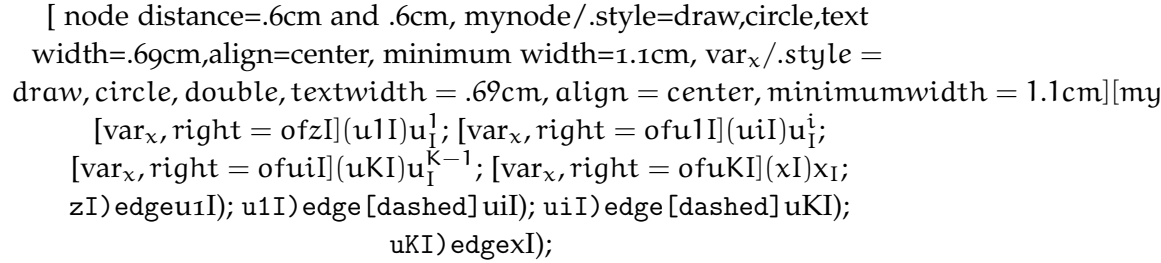rs. However, this second case would imply that at least one undirected edge is removed from the original BN considered in Section 4.3.3. This cannot happen since it would deadly hurt the bijectivity of the flow.

Besides proving the non-universality of affine NFs, this discussion provides the important insight that when affine normalizers must transform non-linearly some latent variables they introduce dependence in the model of the distribution. In some sense, this means that the additional disorder required to model this non-normal component is performed at the cost of some loss in entropy caused by mutual information between the random vector components.

4.6 SUMMARY

In this preliminary work, we have revisited normalizing flows from the perspective of Bayesian networks. We have shown that stacking multiple transformations in a normalizing flow relaxes independence assumptions and entangles the model distribution. Then, we have shown that affine normalizing flows benefit from having at least 3 transformation layers. Finally, we demonstrated that they remain non-universal density approximators regardless of their depths.

We hope these results will give practitioners more intuition in the design of normalizing flows. We also believe that this work may lead to further research. First, unifying Bayesian networks and normalizing flows could be pushed one step further with conditioners that are specifically designed to model Bayesian networks. Second, the study could be extended for other type of normalizing flows such as non-autoregressive monotonic flows. Finally, we believe this study may spark research at the intersection of structural equation modeling, causal networks and normalizing flows.

MONOTONIC NORMALIZING FLOWS

OUTLINE

Monotonic neural networks have recently been proposed as a way to define invertible transformations. These transformations can be combined into powerful autoregressive flows that have been shown to be universal approximators of continuous probability distributions. Architectures that ensure monotonicity typically enforce constraints on weights and activation functions, which enables invertibility but leads to a cap on the expressiveness of the resulting transformations. In this work, we propose the Unconstrained Monotonic Neural Network (UMNN) architecture based on the insight that a function is monotonic as long as its derivative is strictly positive. In particular, this latter condition can be enforced with a free-form neural network whose only constraint is the positiveness of its output. We evaluate our new invertible building block within a new autoregressive flow (UMNN-MAF) and demonstrate its effectiveness on density estimation experiments. We also illustrate the ability of UMNNs to improve variational inference.

5.1 INTRODUCTION

Monotonic neural networks have been known as powerful tools to build monotone models of a response variable with respect to individual explanatory variables [**?????**]. Recently, strictly monotonic neural networks have also been proposed as a way to define invertible transformations. These transformations can be combined into effective autoregressive flows that can be shown to be universal approximators of continuous probability distributions. Examples include Neural Autoregressive Flows [NAF, **?**] and Block Neural Autoregressive Flows [B-NAF, **?**]. Architectures that ensure monotonicity typically enforce constraints on weight and activation functions, which enables invertibility but leads to a cap on the expressiveness of the resulting transformations. For neural autoregressive flows, this does not impede universal approximation but typically requires either complex conditioners or a composition of multiple flows.

Nevertheless, autoregressive flows defined as stacks of reversible transformations have proven to be quite efficient for density estimation of empirical distributions [**???**], as well as to improve posterior modeling in Variational Auto-Encoders (VAE) [**???**]. Practical

successes of these models include speech synthesis [**??**], likelihood-free inference [**?**], probabilistic programming [**?**] and image generation [**?**]. While stacking multiple reversible transformations improves the capacity of the full transformation to represent complex probability distributions, it remains unclear which class of reversible transformations should be used.

In this work, we propose a class of reversible transformations based on a new Unconstrained Monotonic Neural Network (UMNN) architecture. We base our contribution on the insight that a function is monotonic as long as its derivative is strictly positive. This latter condition can be enforced with a free-form neural network whose only constraint is for its output to remain strictly positive.

We summarize our contributions as follows:

- We introduce the Unconstrained Monotonic Neural Network (UMNN) architecture, a new reversible scalar transformation defined via a free-form neural network.

- We combine UMNN transformations into an autoregressive flow (UMNN-MAF) and we demonstrate competitive or state-of-the-art results on benchmarks for normalizing flows.

- We empirically illustrate the scalability of our approach by applying UMNN on high dimensional density estimation problems.

5.2 UNCONSTRAINED MONOTONIC NEURAL NETWORKS

Our primary contribution consists in a neural network architecture that enables learning arbitrary monotonic functions. More specifically, we want to learn a strictly monotonic scalar function $F(x; \psi) : \mathbb{R} \to \mathbb{R}$ without imposing strong constraints on the expressiveness of the hypothesis class. In UMNNs, we achieve this by only imposing the derivative $f(x; \psi) = \frac{\partial F(x; \psi)}{\partial x}$ to remain of constant sign or, without loss of generality, to be strictly positive. As a result, we can parameterize the bijective mapping $F(x; \psi)$ via its strictly positive derivative $f(x; \psi)$ as

$$F(x; \psi) = \int_0^x f(t; \psi) \, dt + \underbrace{F(0; \psi)}_{\beta}, \tag{5.1}$$

where $f(t; \psi) : \mathbb{R} \to \mathbb{R}_+$ is a strictly positive parametric function and $\beta \in \mathbb{R}$ is a scalar. We make f arbitrarily complex using an unconstrained neural network whose output is forced to be strictly positive through an ELU activation unit increased by 1. ψ denotes the parameters of this neural network.

FORWARD INTEGRATION   The forward evaluation of $F(x; \psi)$ requires solving the integral in Equation (5.1). While this might appear daunting, such integrals can often be efficiently approximated numerically using Clenshaw-Curtis quadrature. The better known trapezoidal rule, which corresponds to the two-point Newton-Cotes quadrature rule, has an exponential convergence when the integrand is periodic and the range of integration corresponds to its period. Clenshaw-Curtis quadrature takes advantage of this property by using a change of variables followed by a cosine transform. This extends the exponential convergence of the trapezoidal rule for periodic functions to any Lipschitz continuous function. As a result, the number of evaluation points required to reach convergence grows with the Lipschitz constant of the function.

BACKWARD INTEGRATION   Training the integrand neural network f requires evaluating the gradient of F with respect to its parameters. While this gradient could be obtained by backpropagating directly through the integral solver, this would also result in a memory footprint that grows linearly with the number of integration steps. Instead, the derivative of an integral with respect to a parameter ω can be expressed with the Leibniz integral rule:

$$\frac{d}{d\omega} \left(\int_{a(\omega)}^{b(\omega)} f(t; \omega) \, dt \right) = f(b(\omega); \omega) \frac{d}{d\omega} b(\omega) - f(a(\omega); \omega) \frac{d}{d\omega} a(\omega) + \int_{a(\omega)}^{b(\omega)} \frac{\partial}{\partial\omega} f(t; \omega) \, dt.$$

(5.2)

Applying Equation (5.2) to evaluate the derivative of Equation (5.1) with respect to the parameters ψ, we find

$$\nabla_\psi F(x; \psi) = f(x; \psi) \nabla_\psi (x) - f(0; \psi) \nabla_\psi (0) + \int_0^x \nabla_\psi f(t; \psi) \, dt + \nabla_\psi \beta$$

$$= \int_0^x \nabla_\psi f(t; \psi) \, dt + \nabla_\psi \beta.$$

(5.3)

When using a UMNN block in a neural architecture, it is also important to be able to compute its derivative with respect to its input x. In this case, applying Equation (5.2) leads to

$$\frac{d}{dx} F(x; \psi) = f(x; \psi).$$

(5.4)

Equations (5.3) and (5.4) make the memory footprint for the backward pass independent from the number of integration steps, and therefore also from the desired accuracy. Indeed, instead of computing the gradient of the integral (which requires keeping track of all the integration steps), we integrate the gradient (which is memory efficient, as this corresponds to summing gradients at different evaluation points). We provide the pseudo-code of the forward and backward passes using Clenshaw-Curtis quadrature in Appendix **??**.

NUMERICAL INVERSION   In UMMNs, the modeled monotonic function F is arbitrary. As a result, computing its inverse cannot be done analytically. However, since F is strictly monotonic, it admits a unique inverse x for any point $y = F(x; \psi)$ in its image, therefore inversion can be computed efficiently with common root-finding algorithms. In our experiments, search algorithms such as the bisection method proved to be fast enough.

5.3   UMNN AUTOREGRESSIVE MODELS

5.3.1   *Normalizing flows*

A Normalizing Flow [NF, **?**] is defined as a sequence of invertible transformations $u_i : \mathbb{R}^d \to \mathbb{R}^d$ ($i = 1, ..., k$) composed together to create an expressive invertible mapping $u = u_1 \circ \cdots \circ u_k : \mathbb{R}^d \to \mathbb{R}^d$. It is common for normalizing flows to stack the same parametric function u_i (with different parameters values) and to reverse variables ordering after each transformation. For this reason, we will focus on how to build one of these repeated transformations, which we further refer to as $g : \mathbb{R}^d \to \mathbb{R}^d$.

DENSITY ESTIMATION   NFs are most commonly used for density estimation, that map empirical samples to unstructured noise. Using normalizing flows, we define a bijective mapping $u(\cdot; \theta) : \mathbb{R}^d \to \mathbb{R}^d$ from a sample $x \in \mathbb{R}^d$ to a latent vector $z \in \mathbb{R}^d$ equipped with a density $p_Z(z)$. The transformation u implicitly defines a density $p(x; \theta)$ as given by the change of variables formula,

$$p(x; \theta) = p_Z(u(x; \theta)) \left| \det J_{u(x; \theta)} \right|, \tag{5.5}$$

where $J_{u(x; \theta)}$ is the Jacobian of $u(x; \theta)$ with respect to x. The resulting model is trained by maximizing the likelihood of the data $\{x^1, ..., x^N\}$.

VARIATIONAL AUTO-ENCODERS   NFs are also used in VAE to improve posterior modeling. In this case, a normalizing flow transforms a distribution p_Z into a complex distribution q which can better model the variational posterior. The change of variables formula yields

$$q(u(z; \theta)) = p_Z(z) \left| \det J_{u(z; \theta)} \right|^{-1}. \tag{5.6}$$

5.3.2   *Autoregressive transformations*

To be of practical use, NFs must be composed of transformations for which the determinant of the Jacobian can be computed efficiently, otherwise its evaluation would be running in $O(d^3)$. A common solu-

tion consists in making the transformation g autoregressive, i.e., such that $g(x; \theta)$ can be rewritten as a vector of d scalar functions,

$$g(x; \theta) = \begin{bmatrix} g^1(x_1; \theta) & \dots & g^i(x_{1:i}; \theta) & \dots & g^d(x_{1:d}; \theta) \end{bmatrix},$$

where $x_{1:i} = \begin{bmatrix} x_1 & \dots & x_i \end{bmatrix}^T$ is the vector including the i first elements of the full vector x. The Jacobian of this function is lower triangular, which makes the computation of its determinant $O(d)$. Enforcing the bijectivity of each component g^i is then sufficient to make g bijective as well.

For the multivariate density $p(x; \theta)$ induced by $g(x; \theta)$ and $p_Z(z)$, we can use the chain rule to express the joint probability of x as a product of d univariate conditional densities,

$$p(x; \theta) = p(x_1; \theta) \prod_{i=1}^{d-1} p(x_{i+1} | x_{1:i}; \theta). \tag{5.7}$$

When $p_Z(z)$ is a factored distribution $p_Z(z) = \prod_{i=1}^{d} p(z_i)$, we identify that each component z_i coupled with the corresponding function g^i encodes for the conditional $p(x_i | x_{1:i-1}; \theta)$. Autoregressive transformations strongly rely on the expressiveness of the scalar functions g^i. In this work, we propose to use UMNNs to create powerful bijective scalar transformations.

5.3.3 *UMNN autoregressive transformations (UMNN-MAF)*

We now combine UMNNs with an embedding of the conditioning variables to build invertible autoregressive functions g^i. Specifically, we define

$$\begin{aligned} g^i(x_{1:i}; \theta) &= F^i(x_i, h^i(x_{1:i-1}; \phi^i); \psi^i) \\ &= \int_0^{x_i} f^i(t, h^i(x_{1:i-1}; \phi^i); \psi^i) \, dt + \beta^i(h^i(x_{1:i-1}; \phi^i)), \end{aligned}$$
$$\tag{5.8}$$

where $h^i(\cdot; \phi^i) : R^{i-1} \to R^q$ is a q-dimensional neural embedding of the conditioning variables $x_{1:i-1}$ and $\beta(\cdot)^i : R^{i-1} \to R$. Both degenerate into constants for $g^1(x_1)$. The parameters θ of the whole transformation $g(\cdot; \theta)$ is the union of all parameters ϕ^i and ψ^i. For simplicity we remove the parameters of the networks by rewriting $f^i(\cdot; \psi^i)$ as $f^i(\cdot)$ and $h^i(\cdot; \phi^i)$ as $h^i(\cdot)$.

In our implementation, we use a Masked Autoregressive Network [???] to simultaneously parameterize the d embeddings. In what follows we refer to the resulting UMNN autoregressive transformation as UMNN-MAF. Figure 5.1 summarizes the complete architecture.

every node=[font=]

[->, scale=0.9] neuron = [circle, draw=black!60, line width=0.3mm, fill=white];
at (0,0) (a) Normalizing flow;
(nf-x) at (0,-0.5) x; [rounded corners=4, minimum width=7em, draw] (nf-g1) at (0,
-1.25) g; [rounded corners=4, minimum width=7em, draw] (nf-g2) at (0, -2.0) g;
(nf-dots) at (0, -2.75) ...; [rounded corners=4, minimum width=7em, draw] (nf-gk) at
(0, -3.5) g; (nf-z) at (0,-4.25) z; black] (nf-x) edge (nf-g1); \pathblack] (nf-g1) edge
(nf-g2); black] (nf-g2) edge (nf-dots); \pathblack] (nf-dots) edge (nf-gk);
black] (nf-gk) edge (nf-z); \nodeinner sep=4, rounded corners=4, draw=black!60,
dashed, fit=(nf-g1) (nf-gk)] (nf) ;
at (4,0) (b) UMNN-MAF; at (4,-0.35) transformation;
[circle, fill=white, draw] (h-node1) at (3.5,-1.25-0.4) ; [circle, fill=white, draw]
(h-node2) at (4,-1.25-0.4) ; [circle, fill=white, draw] (h-node3) at (4.5,-1.25-0.4) ;
[circle, fill=white, draw, scale=0.5] (h-node4) at (3.5,-1.75-0.4) h^1; [circle, fill=white,
draw, scale=0.5] (h-node5) at (4,-1.75-0.4) h^2; [circle, fill=white, draw, scale=0.5]
(h-node6) at (4.5,-1.75-0.4) h^3; black] (h-node1) edge (h-node5); \pathblack]
(h-node1) edge (h-node6); black] (h-node2) edge (h-node6); \nodeinner sep=4,
rounded corners=4, draw, fit=(h-node1) (h-node6)] (man) ;
[circle, fill=white, draw, scale=0.5] (g-node1) at (3.5,-2.75-0.4) g^1; [circle, fill=white,
draw, scale=0.5] (g-node2) at (4,-2.75-0.4) g^2; [circle, fill=white, draw, scale=0.5]
(g-node3) at (4.5,-2.75-0.4) g^3; [inner sep=4, rounded corners=4, draw, fit=(g-node1)
(g-node3)] (g) ; black] (man) edge (g); \nodeinner sep=4, rounded corners=4,
draw=black!60, dashed, fit=(man) (g)] (umnn-maf) ;
(umnn-maf-x) at (2.25,-1.5-0.4) x; black] (umnn-maf-x) edge (man); \pathblack,-]
(2.75,-1.5-0.4) edge (2.75,-2.75-0.4);
black] (2.75,-2.75-\umnnmafoffset) edge (g); \pathblack!60, dotted, -]
(nf-g2.north east) edge (umnn-maf.north west);
black!60,dotted,-] (nf-g2.southeast) edge (umnn-maf.southwest); %UMNN=====
===
========\nodeat (7.85,0){(c)UMNN}; \nodecircle, fill=white, draw] (f-node1) at
(7.5,-1.25-0.6) ; [circle, fill=white, draw] (f-node2) at (7.5,-1.75-0.6) ; [circle, fill=white,
draw] (f-node3) at (7.5,-2.25-0.6) ; [circle, fill=white, draw] (f-node4) at
(8.25,-1.25-0.6) ; [circle, fill=white, draw] (f-node5) at (8.25,-1.75-0.6) ; [circle,
fill=white, draw] (f-node6) at (8.25,-2.25-0.6) ; [circle, fill=white, draw] (f-node7) at
(9,-1.75-0.6) ; black] (f-node1) edge (f-node4); \pathblack] (f-node1) edge
(f-node5); black] (f-node1) edge (f-node6); \pathblack] (f-node2) edge (f-node4);
black] (f-node2) edge (f-node5); \pathblack] (f-node2) edge (f-node6);
black] (f-node3) edge (f-node4); \pathblack] (f-node3) edge (f-node5);
black] (f-node3) edge (f-node6); \pathblack] (f-node4) edge (f-node7);
black] (f-node5) edge (f-node7); \pathblack] (f-node6) edge (f-node7); [inner
sep=4, rounded corners=4, draw, fit=(f-node1) (f-node3) (f-node7)] (f) ;
(umnn-t) at (6.75,-1.25-0.6) t; black] (umnn-t) edge (f-node1); \node (umnn-h3)
at (6.75,-2-\umnnoffset){$\mathbf{h}^3$}; \nodeinner sep=1, rounded
corners=4, draw=black!60, fit=(f-node2) (f-node3)] (umnn-h) ;
black] (umnn-h3) edge (umnn-h); \nodeminimum size=0, inner sep=0] (dt) at

(9.625,-1.75-0.6) dt; [scale=1.4, inner sep=0] (integral) at (6.25,-1.75-0.6) $\int$;

[inner sep=4, rounded corners=4, draw=black!60, dashed, fit=(integral) (f) (dt)]
(umnn) ;
(x3) at (7.85,-0.125-0.6) x_3; (z3) at (7.85,-3.4-0.6) z_3;
black] (x3) edge (umnn); \pathblack] (umnn) edge (z3);
black!60,dotted,-] (g-node3.north) edge (umnn.northwest); \pathblack!60,
dotted, -] (g-node3.south) edge (umnn.south west);

Figure 5.1: (a) A normalizing flow made of repeated UMNN-MAF transfor-
mations g with identical architectures. (b) A UMNN-MAF which
transforms a vector $x \in \mathbb{R}^3$. (c) The UMNN network used to map
x_3 to z_3 conditioned on the embedding $h^3(x_{1:2})$.

LOG-DENSITY   The change of variables formula applied to the UMMN autoregressive transformation results in the log-density

$$\log p(x; \theta) = \log p_Z(g(x; \theta)) \left| \det J_{g(x;\theta)} \right|$$

$$= \log p_Z(g(x; \theta)) + \log \left| \prod_{i=1}^{d} \frac{\partial F^i(x_i, h^i(x_{1:i-1}))}{\partial x_i} \right|$$

$$= \log p_Z(g(x; \theta)) + \sum_{i=1}^{d} \log f^i(x_i, h^i(x_{1:i-1})). \qquad (5.9)$$

Therefore, the transformation leads to a simple expression of (the determinant of) its Jacobian, which can be computed efficiently with a single forward pass. This is different from FFJORD [**?**] which relies on numerical methods to compute both the Jacobian and the transformation between the data and the latent space. Therefore our proposed method makes the computation of the Jacobian exact and efficient at the same time.

SAMPLING   Generating samples require evaluating the inverse transformation $g^{-1}(z; \theta)$. The components of the inverse vector $x^{\text{inv}} = g^{-1}(z; \theta)$ can be computed recursively by inverting each component of $g(x; \theta)$:

$$x_1^{\text{inv}} = \left(g^1\right)^{-1} \left(z_1; h^1\right) \qquad \qquad \text{if} \quad i = 1 \qquad (5.10)$$

$$x_i^{\text{inv}} = \left(g^i\right)^{-1} \left(z_i; h^i \left(x_{1:i-1}^{\text{inv}}\right)\right) \qquad \text{if} \quad i > 1 \qquad (5.11)$$

where $\left(g^i\right)^{-1}$ is the inverse of g^i. Another approach to invert an autoregressive model would be to approximate its inverse with another autoregressive network [**?**]. In this case, the evaluation of the approximated inverse model is as fast as the forward model.

UNIVERSALITY   Since the proof is straightforward, we only sketch that UMNN-MAF is a universal density approximator of continuous random variables. We rely on the inverse sampling theorem to prove that UMNNs are universal approximators of continuously derivable ($\mathbb{C}^1$) monotonic functions. Indeed, if UMNNs can represent any $\mathbb{C}^1$ monotonic function, then they can also represent the (inverse) cumulative distribution function of any continuous random variable. Any continuously derivable function $f : \mathcal{D} \to \mathcal{I}$ can be expressed as the following integral: $f(x) = \int_a^x \frac{df}{dx} dx + f(a), \quad \forall x, a \in \mathcal{D}$. The derivative $\frac{df}{dx}$ is a continuous positive function and the universal approximation theorem of NNs ensures it can be successfully approximated with a NN of sufficient capacity (such as those used in UMNNs).

5.4   RELATED WORK

The most similar work to UMNN-MAF are certainly Neural Autoregressive Flow [NAF, **?**] and Block Neural Autoregressive Flow [B-

NAF, **?**], which both rely on strictly monotonic transformations for building bijective mappings. In NAF, transformations are defined as neural networks which activation functions are all constrained to be strictly monotonic and which weights are the output of a strictly positive and autoregressive HyperNetwork [**?**]. **?** shows that NAFs are universal density approximators. In B-NAF, the authors improve on the scalability of the NAF architecture by making use of masking operations instead of HyperNetworks. They also present a proof of the universality of B-NAF, which extends to UMNN-MAF. Our work differs from both NAF and B-NAF in the sense that the UMNN monotonic transformation is based on free-form neural networks for which no constraint, beyond positiveness of the output, is enforced on the hypothesis class. This leads to multiple advantages: it enables the use of any state-of-the-art neural architecture, simplifies weight initialization, and leads to a more lightweight evaluation of the Jacobian.

More generally, UMNN-MAF relates to works on normalizing flows built upon autoregressive networks and affine transformations. **?** first introduced masking as an efficient way to build autoregressive networks, and proposed autoregressive networks for density estimation of high dimensional binary data. Masked Autoregressive Flows [**?**] and Inverse Autoregressive Flows [**?**] have generalized this approach to real data, respectively for density estimation and for latent posterior representation in variational auto-encoders. More recently, **?** proposed to stack various autoregressive architectures to create powerful reversible transformations. Meanwhile, **?** proposed a new Sum-of-Squares flow that is defined as the integral of a second order polynomial parametrized by an autoregressive NN.

With NICE, **?** introduced coupling layers, which correspond to bijective transformations splitting the input vector into two parts. They are defined as

$$z_{1:k} = x_{1:k} \quad \text{and} \quad z_{k+1:d} = e^{\sigma(x_{1:k})} \odot x_{k+1:d} + \mu(x_{1:k}), \quad (5.12)$$

where σ and μ are two unconstrained functions $R^{d-k} \to R^{d-k}$. The same authors introduced RealNVP [**?**], which combines coupling layers with normalizing flows and multi-scale architectures for image generation. Glow [**?**] extends RealNVP by introducing invertible 1x1 convolutions between each step of the flow. In this work we have used UMNNs in the context of autoregressive architectures, however UMNNs could also be applied to replace the linear transformation in coupling layers.

Finally, our architecture also shares a connection with Neural Ordinary Differential Equations [NODE, **?**]. The core idea of this architecture is to learn an ordinary differential equation which dynamic is parameterized by a neural network. Training can be carried out by backpropagating efficiently through the ODE solver, with constant memory requirements. Among other applications, NODE can be used

to model a continuous normalizing flow with a free-form Jacobian as in FFJORD [**?**]. Similarly, a UMNN transformation can be seen as a structured neural ordinary differential equation in which the dynamic of the vector field is separable and can be solved efficiently by direct integration.

5.5 EXPERIMENTS

In this section, we evaluate the expressiveness of UMNN-MAF on a variety of density estimation benchmarks, as well as for approximate inference in variational auto-encoders. The source code is accessible at https://github.com/AWehenkel/UMNN.

Experiments were carried out using the same integrand neural network in the UMNN component – i.e., in Equation 5.8, $f^i = f$ with shared weights $\psi^i = \psi$ for $i \in \{1, \ldots, d\}$. The functions β^i are taken to be equal to one of the outputs of the embedding network. We observed in our experiments that sharing the same integrand function does not impact performance. Therefore, the neural embedding function h^i must produce a fixed size output for $i \in \{1, \ldots, d\}$.

5.5.1 *2D toy problems*

We first train a UMNN-MAF on 2-dimensional toy distributions, as defined by **?**. To train the model, we minimize the negative log-likelihood of observed data

$$L(\theta) = -\sum_{n=1}^{N} \left[\log p_Z(g(x^n; \theta)) + \sum_{i=1}^{d} \log f(x_i^n, h^i(x_{1:i-1}^n)) \right].$$
(5.13)

The flow used to solve these tasks is the same for all distributions and is composed of a single transformation. More details can be found in Appendix **??**.

Figure 5.2 demonstrates that our model is able to learn a change of variables that warps a simple isotropic Gaussian into multimodal and/or discontinuous distributions. We observe from the figure that our model precisely captures the density of the data. We also observe that numerical inversion for generating samples yields good results.

5.5.2 *Density estimation*

We further validate UMNN-MAF by comparing it to state-of-the-art normalizing flows. We carry out experiments on tabular datasets (POWER, GAS, HEPMASS, MINIBOONE, BSDS300) as well as on MNIST. We follow the experimental protocol of **?**. All training hyper-parameters and architectural details are given in Appendix **??**. For each dataset,

figures/toy/all_flow.png

Figure 5.2: Density estimation and sampling with a UMNN-MAF network on 2D toy problems. Top: Samples from the empirical distribution $p(x)$. Middle: Learned density $p(x; \theta)$. Bottom: Samples drawn by numerical inversion. UMNN-MAF manages to precisely capture multi-modal and/or discontinuous distributions. Sampling is possible even if the model is not invertible analytically.

Table 5.1: Average negative log-likelihood on test data over 3 runs, error bars are equal to the standard deviation. Results are reported in nats for tabular data and bits/dim for MNIST; lower is better. The best performing architecture for each dataset is written in bold and the best performing architecture per category is underlined. (a) Non-autoregressive models, (b) Autoregressive models, (c) Monotonic and autoregressive models. UMNN outperforms other monotonic transformations on 4 tasks over 6 and is the overall best performing model on 2 tasks over 6.

Dataset	POWER	GAS	HEPMASS	MINIBOONE	BSDS300	MNIST
3*(a) RealNVP - ?	$-0.17_{\pm.01}$	$-8.33_{\pm.14}$	$18.71_{\pm.02}$	$13.55_{\pm.49}$	$-153.28_{\pm1.78}$	-
Glow - ?	$-0.17_{\pm.01}$	$-8.15_{\pm.40}$	$19.92_{\pm.08}$	$11.35_{\pm.07}$	$-155.07_{\pm.03}$	-
FFJORD - ?	$\underline{-0.46}_{\pm.01}$	$\underline{-8.59}_{\pm.12}$	$\underline{14.92}_{\pm.08}$	$\underline{10.43}_{\pm.04}$	$\underline{-157.40}_{\pm.19}$	-
3*(b) MADE - ?	$3.08_{\pm.03}$	$-3.56_{\pm.04}$	$20.98_{\pm.02}$	$15.59_{\pm.50}$	$-148.85_{\pm.28}$	$2.04_{\pm.01}$
MAF - ?	$-0.24_{\pm.01}$	$-10.08_{\pm.02}$	$17.70_{\pm.02}$	$11.75_{\pm.44}$	$-155.69_{\pm.28}$	$1.89_{\pm.01}$
TAN - ?	$\underline{-0.60}_{\pm.01}$	$\underline{-12.06}_{\pm.02}$	$\underline{13.78}_{\pm.02}$	$\underline{11.01}_{\pm.48}$	$\underline{-159.80}_{\pm.07}$	$\underline{1.19}$
3*(c) NAF - ?	$-0.62_{\pm.01}$	$-11.96_{\pm.33}$	$15.09_{\pm.40}$	$\underline{8.86}_{\pm.15}$	$-157.73_{\pm.30}$	-
B-NAF - ?	$-0.61_{\pm.01}$	$\underline{-12.06}_{\pm.09}$	$14.71_{\pm.38}$	$8.95_{\pm.07}$	$-157.36_{\pm.03}$	-
SOS - ?	$-0.60_{\pm.01}$	$-11.99_{\pm.41}$	$15.15_{\pm.1}$	$8.90_{\pm.11}$	$-157.48_{\pm.41}$	1.81
UMNN-MAF (ours)	$\underline{-0.63}_{\pm.01}$	$-10.89_{\pm.7}$	$\underline{13.99}_{\pm.21}$	$9.67_{\pm.13}$	$\underline{-157.98}_{\pm.01}$	$\underline{1.13}_{\pm.02}$

we report results on test data for our best performing model (selected on the validation data). At testing time we use a large number of integration steps (100) to compute the integral, this ensures its correctness and avoids misestimating the performance of UMNN-MAF.

Table 5.1 summarizes our results, where we can see that on tabular datasets, our method is competitive with other normalizing flows. For POWER, our architecture slightly outperforms all others. It is also better than other monotonic networks (category (c)) on 3 tabular datasets over 5. From these results, we could conclude that Transformation Autoregressive Networks [TAN, **?**] is overall the best method for density estimation. It is however important to note that TAN is a flow composed of many heterogeneous transformations (both autoregressive and non-autoregressive). For this reason, it should not be directly compared to the other models which respective results are specific to a single architecture. However, TAN provides the interesting insight that combining heterogeneous components into a flow leads to better results than an homogeneous flow.

Notably, we do not make use of a multi-scale architecture to train our model on MNIST. On this task, UMNN-MAF slightly outperforms all other models by a reasonable margin. Samples generated by a conditional model are shown on Figure 5.3, for which it is worth noting that UMNN-MAF is the first monotonic architecture that has been inverted to generate samples. Indeed, MNIST can be considered as a high dimensional dataset (d = 784) for standard feed forward neural networks which autoregressive networks are part of. NAF and B-NAF do not report any result for this benchmark, presumably because of memory explosion. In comparison, BSDS300, which data dimension is one order of magnitude smaller than MNIST ($63 \ll 784$), are the largest data they have tested on. Table 5.2 shows the number of parameters used by UMNN-MAF in comparison to B-NAF and NAF. For bigger datasets, UMNN-MAF requires less parameters than NAF to reach similar or better performance. This could explain why NAF has never been used for density estimation on MNIST.
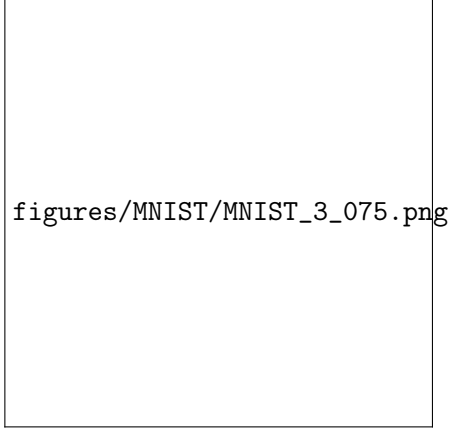
Figure 5.3: Samples generated by numerical inversion of a conditional UMNN-MAF trained on MNIST. Samples z are drawn from an isotropic Gaussian with $\sigma = .75$. See Appendix **??** for more details.

Table 5.2: Comparison of the number of parameters between NAF, B-NAF and UMNN-MAF. In high dimensional datasets, UMNN-MAF requires fewer parameters than NAF and a similar number to B-NAF.

Dataset	NAF	B-NAF	UMNN-MAF
POWER ($d = 6$)	(1)	(1)	(1)
GAS ($d = 8$)	(2)	(2)	(2)
HEPMASS ($d = 21$)	(3)	(3)	(3)
MINIBOONE ($d = 43$)	(4)	(4)	(4)
BSDS300 ($d = 63$)	(5)	(5)	(5)

5.5.3 *Variational auto-encoders*

To assess the performance of our model, we follow the experimental setting of **?** for VAE. The encoder and the decoder architectures can be found in the appendix of their paper. In VAE it is usual to let the encoder output the parameters of the flow. For UMNN-MAF, this would cause the encoder output's dimension to be too large. Instead, the encoder output is passed as additional entries of the UMNN-MAF. Like other architectures, the UMNN-MAF also takes as input a vector of noise drawn from an isotropic Gaussian of dimension 64.

Table 5.3 presents our results. It shows that on MNIST and Omniglot, UMNN-MAF slightly outperforms the classical VAE as well as planar flows. Moreover, on these datasets and Freyfaces, IAF, B-NAF and UMNN-MAF achieve similar results. FFJORD is the best among all, however it is worth noting that the roles of encoder outputs in FFJORD, B-NAF, IAF and Sylvester are all different. We believe that the heterogeneity of the results could be, at least in part, due to the different amortizations.

5.6 DISCUSSION AND SUMMARY

STATIC INTEGRAL QUADRATURE CAN BE INACCURATE. Computing the integral with static Clenshaw-Curtis quadrature only requires the evaluation of the integrand at predefined points. As such, batches of points can be processed all at once, which makes static

Table 5.3: Average negative evidence lower bound of VAEs over 3 runs, error bars are equal to the standard deviation. Results are reported in bits per dim for Freyfaces and in nats for the other datasets; lower is better. UMNN-NAF is performing slightly better than IAF but is outperformed by B-NAF. We believe that the gap in performance between B-NAF and UMNN is due to the way the NF is conditioned by the encoder's output.

Dataset	MNIST	Freyfaces	Omniglot	Caltech 101
5*(a) VAE - ?	$86.65_{\pm.06}$	$4.53_{\pm.02}$	$104.28_{\pm.39}$	$110.80_{\pm.46}$
Planar - ?	$86.06_{\pm.32}$	$4.40_{\pm.06}$	$102.65_{\pm.42}$	$109.66_{\pm.42}$
IAF - ?	$84.20_{\pm.17}$	$4.47_{\pm.05}$	$102.41_{\pm.04}$	$111.58_{\pm.38}$
Sylvester - ?	$83.32_{\pm.06}$	$4.45_{\pm.04}$	$99.00_{\pm.04}$	$104.62_{\pm.29}$
FFJORD - ?	$82.82_{\pm.01}$	$4.39_{\pm.01}$	$98.33_{\pm.09}$	$104.03_{\pm.43}$
2*(b) B-NAF - ?	$83.59_{\pm.15}$	$4.42_{\pm.05}$	$100.08_{\pm.07}$	$105.42_{\pm.49}$
UMNN-MAF (ours)	$84.11_{\pm.05}$	$4.51_{\pm.01}$	$100.98_{\pm.13}$	$110.45_{\pm.69}$

Clenshaw-Curtis quadrature well suited for neural networks. However, static quadratures do not account for the error made during the integration. As a consequence, the quadrature is inaccurate when the integrand is not smooth enough and the number of integration steps is too small. In this work, we have reduced the integration error by applying the normalization described by **?** in order to control the Lipschitz constant of the integrand and appropriately set the number of integration steps. We observed that as long as the Lipschitz constant of the network does not increase dramatically (< 1000), a reasonable number of integration steps (< 100) is sufficient to ensure the convergence of the quadrature. An alternative solution would be to use dynamic quadrature such as dynamic Clenshaw-Curtis.

EFFICIENCY OF NUMERICAL INVERSION. Architectures relying on linear transformations [**????**] are trivially exactly and efficiently invertible. In contrast, the UMNN transformation has no analytic inverse. Nevertheless, it can be inverted numerically using root-finding algorithms. Since most such algorithms rely on multiple nested evaluations of the function to be inverted, applying them naively to a numerical integral would quickly become very inefficient. However, the Clenshaw-Curtis quadrature is part of the nested quadrature family, meaning that the evaluation of the integral at multiple nested points can take advantage of previous evaluations and thus be implemented efficiently. As an alternative, **?** have shown that an invertible model can always be distilled to learn its inverse, and thus make the inversion efficient whatever the cost of inversion of the original model.

SCALABILITY AND COMPLEXITY ANALYSIS.   UMNN-MAF is particularly well suited for density estimation because the computation of the Jacobian only requires a single forward evaluation of a NN. Together with the Leibniz integral rule, they make the evaluation of the log-likelihood derivative as memory efficient as usual supervised learning, which is equivalent to a single backward pass on the computation graph. By contrast, density estimation with previous monotonic transformations typically requires a backward evaluation of the computation graph of the transformer NN to obtain the Jacobian. Then, this pass must be evaluated backward again in order to obtain the log-likelihood derivative. Both NAF and B-NAF provide a method to make this computation numerically stable, however both fail at not increasing the size of the computation graph of the log-likelihood derivative, hence leading to a memory overhead. The memory saved by the Leibniz rule may serve to speed up the quadrature computation. In the case of static Clenshaw-Curtis, the function values at each evaluation point can be computed in parallel using batch of points. In consequence, when the GPU memory is large enough to store "meta-batches" of size $d \times N \times B$ (with d the dimension of the data, N the number of integration steps and B the batch size) the computation is approximately as fast as a forward evaluation of the integrand network.

SUMMARY   We have introduced Unconstrained Monotonic Neural Networks, a new invertible transformation built upon free-form neural networks allowing the use of any state-of-the-art architecture. Monotonicity is guaranteed without imposing constraints on the expressiveness of the hypothesis class, contrary to classical approaches. We have shown that the resulting integrated neural network can be evaluated efficiently using standard quadrature rule while its inverse can be computed using numerical algorithms. We have shown that our transformation can be composed into an autoregressive flow, with competitive or state-of-the-art results on density estimation and variational inference benchmarks. Moreover, UMNN is the first monotonic transformation that has been successfully applied for density estimation on high dimensional data distributions (MNIST), showing better results than the classical approaches.

We identify several avenues for improvement and further research. First, we believe that numerical integration could be fasten up during training, by leveraging the fact that controlled numerical errors can actually help generalization. Moreover, the UMNN transformation would certainly profit from using a dynamic integration scheme, both in terms of accuracy and efficiency. Second, it would be worth comparing the newly introduced monotonic transformation with common approaches for modelling monotonic functions in machine learning. On a similar track, these common approaches could be combined

into an autoregressive flow as shown in Section 5.3.3. Finally, our monotonic transformation could be used within other neural architectures than generative autoregressive networks, such as multi-scale architectures [**?**] and learnable 1D convolutions [**?**].

6

DIGRESSION ON MONOTONIC FUNCTIONS IN MACHINE LEARNING

OUTLINE
???

Part II

HYBRID GENERATIVE MODELING

STRUCTURED DENSITY ESTIMATION WITH NORMALIZING FLOWS

OUTLINE
???

7.1 INTRODUCTION

Normalizing flows [NFs, **????**] have proven to be an effective way to model complex data distributions with neural networks. These models map data points to latent variables through an invertible function while keeping track of the change of density caused by the transformation. In contrast to variational auto-encoders (VAEs) and generative adversarial networks (GANs), NFs provide access to the exact likelihood of the model's parameters, hence offering a sound and direct way to optimize the network parameters. Normalizing flows have proven to be of practical interest as demonstrated by **??** and **?** for speech synthesis, by **??** and **?** for variational inference or by **?** and **?** for simulation-based inference. Yet, their usage as a base component of the machine learning toolbox is still limited in comparison to GANs or VAEs. Recent efforts have been made by **?** and **?** to define the fundamental principles of flow design and by **?** to provide coding tools for modular implementations. We argue that normalizing flows would gain in popularity by offering stronger inductive bias as well as more interpretability.

Sometimes forgotten in favor of more data oriented methods, probabilistic graphical models (PGMs) have been popular for modeling complex data distributions while being relatively simple to build and read [**??**]. Among PGMs, Bayesian networks [BNs, **?**] offer an appealing balance between modeling capacity and simplicity. Most notably, these models have been at the basis of expert systems before the big data era (e.g. [**???**]) and were commonly used to merge qualitative expert knowledge and quantitative information together. On the one hand, experts stated independence assumptions that should be encoded by the structure of the network. On the other hand, data were used to estimate the parameters of the conditional probabilities/densities encoding the quantitative aspects of the data distribution. These models have progressively received less attention from the machine learning community in favor of other methods that scale better with the dimensionality of the data.

Driven by the objective of integrating intuition into normalizing flows and the proven relevance of BNs for combining qualitative and

quantitative reasoning, we summarize our contributions as follows: **(i)** From the insight that coupling and autoregressive transformations can be reduced to Bayesian networks with a fixed topology, we introduce the more general graphical conditioner for normalizing flows, featuring either a prescribed or a learnable BN topology; **(ii)** We show that using a correct prescribed topology leads to improvements in the modeled density compared to autoregressive methods. When the topology is not known we observe that, with the right amount of ℓ_1-penalization, graphical conditioners discover relevant relationships; **(iii)** In addition, we show that graphical normalizing flows perform well in a large variety of density estimation tasks compared to classical black-box flow architectures.

7.2   BACKGROUND

BAYESIAN NETWORKS   A Bayesian network is a directed acyclic graph (DAG) that represents independence assumptions between the components of a random vector. Formally, let $x = [x_1, \ldots, x_d]^\top \in \mathbb{R}^d$ be a random vector distributed under p_x. A BN associated to x is a directed acyclic graph made of d vertices representing the components x_i of x. In this kind of network, the absence of edges models conditional independence between groups of components through the concept of d-separation [**?**]. A BN is a valid representation of a random vector x iff its density can be factorized as

$$p_x(x) = \prod_{i=1}^{d} p(x_i | \mathcal{P}_i), \tag{7.1}$$

where $\mathcal{P}_i = \{j : A_{i,j} = 1\}$ denotes the set of parents of the vertex i and $A \in \{0,1\}^{d \times d}$ is the adjacency matrix of the BN. As an example, fig:mono-step-flows-a is a valid BN for any distribution over x because it does not state any independence and leads to a factorization that corresponds to the chain rule. However, in practice we seek for a sparse and valid BN which models most of the independence between the components of x, leading to an efficient factorization of the modeled probability distribution. It is worth noting that making hypotheses on the graph structure is equivalent to assuming certain conditional independence between some of the vector's components.

NORMALIZING FLOWS   A normalizing flow is defined as a sequence of invertible transformation steps $g_k : \mathbb{R}^d \to \mathbb{R}^d$ $(k = 1, \ldots, K)$ composed together to create an expressive invertible mapping $g := g_1 \circ \cdots \circ g_K : \mathbb{R}^d \to \mathbb{R}^d$. This mapping can be used to perform density estimation, using $g(\cdot; \theta) : \mathbb{R}^d \to \mathbb{R}^d$ to map a sample $x \in \mathbb{R}^d$ onto a latent vector $z \in \mathbb{R}^d$ equipped with a prescribed density $p_z(z)$ such as

.29 [node distance=.7cm and 1.cm, var_x/.style $=$ $draw, circle, textwidth = .4cm, align = center][var_x](x1)x_1;$ $[var_x, right = ofx1](x2)x_2; [var_x, below = ofx1](x3)x_3;$ $[var_x, right = ofx3](x4)x_4;$ x1)edge[-latex]x2); x1)edge[-latex]x3); x1)edge[-latex]x4); x2)edge[-latex]x3); x2)edge[-latex]x4); x3)edge[-latex]x4);

Figure 7.1

.3 [node distance=.7cm and 1.cm, var_x/.style $=$ $draw, circle, textwidth = .4cm, align = center][var_x](x1)x_1;$ $[var_x, right = ofx1](x2)x_2; [var_x, below = ofx1](x3)x_3;$ $[var_x, right = ofx3](x4)x_4;$ x1)edge[-latex]x3); x1)edge[-latex]x4); x2)edge[-latex]x3); x2)edge[-latex]x4);

Figure 7.2

.5 [node distance=.7cm and 1.cm, var_z/.style $=$ $draw, circle, textwidth = .4cm, align = center, var_x$/.style $=$ $draw, circle, double, textwidth = .35cm, align = center][var_x](x1)x_1;$ $[var_x, right = ofx1](x2)x_2; [var_x, below = ofx1](x3)x_3;$ $[var_x, right = ofx3](x4)x_4; [var_z, left = ofx1](z1)z_1;$ $[var_z, right = ofx2](z2)z_2; [var_z, left = ofx3](z3)z_3;$ $[var_z, right = ofx4](z4)z_4;$ x1)edge[-latex]x3); x1)edge[-latex]x4); x2)edge[-latex]x3); x2)edge[-latex]x4); z\x)edge[-latex]x1);z\x)edge[-latex]x2);z\x)edge[-latex]x3);z\ x)edge[-latex]x4);

Figure 7.3

Figure 7.4: Bayesian networks equivalent to normalizing flows made of a single transformation step. (a) Autoregressive conditioner. (b) Coupling conditioner. (c) Coupling conditioner, with latent variables shown explicitly. Double circles stand for deterministic functions of the parents.

an isotropic Normal. The transformation g implicitly defines a density $p(x; \theta)$ as given by the change of variables formula,

$$p(x; \theta) = p_z(g(x; \theta)) \left| \det J_{g(x;\theta)} \right|, \tag{7.2}$$

where $J_{g(x;\theta)}$ is the Jacobian of $g(x; \theta)$ with respect to x. The resulting model is trained by maximizing the likelihood of the model's parameters θ given the training dataset $X = \{x^1, ..., x^N\}$. Unless needed, we will not distinguish between g and g_k for the rest of our discussion.

In general the steps g can take any form as long as they define a bijective map. Here, we focus on a sub-class of normalizing flows for which the steps can be mathematically described as

$$g(x) = \left[g^1(x_1; c^1(x)) \quad \dots \quad g^d(x_d; c^d(x)) \right]^T, \tag{7.3}$$

where the c^i are the conditioners which role is to constrain the structure of the Jacobian of g. The functions g^i, partially parameterized by their conditioner, must be invertible with respect to their input variable x_i. They are often referred to as transformers, however in this work we will use the term normalizers to avoid any confusion with attention-based transformer architectures.

The conditioners examined in this work can be combined with any normalizer. In particular, we consider affine and monotonic normalizers. An affine normalizer $g : \mathbb{R} \times \mathbb{R}^2 \to \mathbb{R}$ can be expressed as $g(x; m, s) = x \exp(s) + m$, where $m \in \mathbb{R}$ and $s \in \mathbb{R}$ are computed by the conditioner. There exist multiple methods to parameterize monotonic normalizers [????], but in this work we rely on Unconstrained Monotonic Neural Networks [UMNNs, ?] which can be expressed as $g(x; c) = \int_0^x f(t, c)dt + \beta(c)$, where $c \in \mathbb{R}^{|c|}$ is an embedding made by the conditioner and $f : \mathbb{R}^{|c|+1} \to \mathbb{R}^+$ and $\beta : \mathbb{R}^d \to \mathbb{R}$ are two neural networks respectively with a strictly positive scalar output and a real scalar output. **?** proved NFs built with autoregressive conditioners and monotonic normalizers are universal density approximators of continuous random variables.

7.3 NORMALIZING FLOWS AS BAYESIAN NETWORKS

AUTOREGRESSIVE CONDITIONERS   Due to computing speed considerations, NFs are usually composed of transformations for which the determinant of the Jacobian can be computed efficiently, as otherwise its evaluation would scale cubically with the input dimension. A common solution is to use autoregressive conditioners, i.e., such that

$$c^i(x) = h^i \left(\left[x_1 \quad \dots \quad x_{i-1} \right]^T \right)$$

are functions h^i of the first $i - 1$ components of x. This particular form constrains the Jacobian of g to be lower triangular, which makes the computation of its determinant $O(d)$.

For the multivariate density $p(x; \theta)$ induced by $g(x; \theta)$ and $p_z(z)$, we can use the chain rule to express the joint probability of x as a product of d univariate conditional densities,

$$p(x; \theta) = p(x_1; \theta) \prod_{i=2}^{d} p(x_i | x_{1:i-1}; \theta). \tag{7.4}$$

When $p_z(z)$ is a factored distribution $p_z(z) = \prod_{i=1}^{d} p(z_i)$, we identify that each component z_i coupled with the corresponding functions g^i and embedding vectors c^i encode for the conditional $p(x_i | x_{1:i-1}; \theta)$. Therefore, and as illustrated in fig:mono-step-flows-a, autoregressive transformations can be seen as a way to model the conditional factors of a BN that does not state any independence but relies on a predefined node ordering. This becomes clear if we define $\mathcal{P}_i = \{x_1, \ldots, x_{i-1}\}$ and compare (7.4) with (7.1).

The complexity of the conditional factors strongly depends on the ordering of the vector components. While not hurting the universal representation capacity of normalizing flows, the arbitrary ordering used in autoregressive transformations leads to poor inductive bias and to factors that are most of the time difficult to learn. In practice, one often alleviates the arbitrariness of the ordering by stacking multiple autoregressive transformations combined with random permutations on top of each other.

COUPLING CONDITIONERS   Coupling layers [?] are another popular type of conditioners that lead to a bipartite structure. The conditioners c^i made from coupling layers are defined as

$$c^i(x) = \begin{cases} \underline{h}^i & \text{if} \quad i \leqslant k \\ h^i \left(\begin{bmatrix} x_1 & \ldots & x_k \end{bmatrix}^T \right) & \text{if} \quad i > k \end{cases}$$

where the underlined $\underline{h}^i \in \mathbb{R}^{|c|}$ denote constant values and $k \in \{1, \ldots, d\}$ is a hyper-parameter usually set to $* \frac{d}{2}$. As for autoregressive conditioners, the Jacobian of g made of coupling layers is lower triangular. Again, and as shown in fig:mono-step-flows-b and 7.3, these transformations can be seen as a specific class of BN where $\mathcal{P}_i = \{\}$ for $i \leqslant k$ and $\mathcal{P}_i = \{1, \ldots, k\}$ for $i > k$. D-separation can be used to read off the independencies stated by this class of BNs such as the conditional independence between each pair in $x_{k+1:d}$ knowing $x_{1:k}$. For this reason, and in contrast to autoregressive transformations, coupling layers are not by themselves universal density approximators even when associated with very expressive normalizers g^i [?]. In practice, these bipartite structural independencies can be relaxed by stacking multiple layers, and may even recover an autoregressive structure. They also lead to useful inductive bias, such as in the multi-scale architecture with checkerboard masking [??].

Figure 7.5: Sampling

Algorithm 7.1 (H).
1: $z \sim \mathcal{N}(0, I)$
2: $x \leftarrow z$
3: **repeat**
4:     $c^i \leftarrow h^i(x \odot A_{i,:})$ $\forall i \in \{1, \ldots, d\}$
5:     $x_i \leftarrow (g^i)^{-1}(z_i; c^i, \theta)$ $\forall i \in \{1, \ldots, d\}$
6: **until** x *converged*

7.4 GRAPHICAL NORMALIZING FLOW

7.4.1 *Graphical conditioners*

Following up on the previous discussion, we introduce the graphical conditioner architecture. We motivate our approach by observing that the topological ordering (a.k.a. ancestral ordering) of any BN leads to a lower triangular adjacency matrix whose determinant is equal to the product of its diagonal terms (proof in Appendix **??**). Therefore, conditioning factors $c^i(x)$ selected by following a BN adjacency matrix necessarily lead to a transformation g whose Jacobian determinant remains efficient to compute.

Formally, given a BN with adjacency matrix $A \in \{0, 1\}^{d \times d}$, we define the graphical conditioner as being

$$c^i(x) = h^i(x \odot A_{i,:}), \tag{7.5}$$

where $x \odot A_{i,:}$ is the element-wise product between the vector x and the ith row of A – i.e., the binary vector $A_{i,:}$ is used to mask on x. NFs built with this new conditioner architecture can be inverted by sequentially inverting each component in the topological ordering. In our implementation the neural networks modeling the h^i functions are shared to save memory and they take an additional input that one-hot encodes the value i. An alternative approach would be to use a masking scheme similar to what is done by **?** in MADE as suggested by **?**.

The graphical conditioner architecture can be used to learn the conditional factors in a continuous BN while elegantly setting structural independencies prescribed from domain knowledge. In addition, the inverse of NFs built with graphical conditioners is a simple as it is for autoregressive and coupling conditioners, Algorithm 7.5 describes an inversion procedure. We also now note how these two conditioners are just special cases in which the adjacency matrix reflects the classes of BNs discussed in Section 7.3.

7.4.2 *Learning the topology*

In many cases, defining the whole structure of a BN is not possible due to a lack of knowledge about the problem at hand. Fortunately,

not only is the density at each node learnable, but also the DAG structure itself: defining an arbitrary topology and ordering, as it is implicitly the case for autoregressive and coupling conditioners, is not necessary.

Building upon *Non-combinatorial Optimization via Trace Exponential and Augmented lagRangian for Structure learning* [NO TEARS, **?**], we convert the combinatorial optimization of score-based learning of a DAG into a continuous optimization by relaxing the domain of A to real numbers instead of binary values. That is,

$$
\begin{array}{ccc}
\max\limits_{A \in \mathbb{R}^{d \times d}} F(A) & & \max\limits_{A \in \mathbb{R}^{d \times d}} F(A) \\
\text{s.t. } \mathcal{G}(A) \in \text{DAGs} & \Longleftrightarrow & \text{s.t. } w(A) = 0,
\end{array} \tag{7.6}
$$

where $\mathcal{G}(A)$ is the graph induced by the weighted adjacency matrix A and $F : \mathbb{R}^{d \times d} \to \mathbb{R}$ is the log-likelihood of the graphical NF g plus a regularization term, i.e.,

$$
F(A) = \sum_{j=1}^{N} \log \left(p(x^j; \theta) \right) + \lambda_{\ell_1} \|A\|_1, \tag{7.7}
$$

where λ_{ℓ_1} is an ℓ_1-regularization coefficient and N is the number of training samples x^i. The likelihood is computed as

$$
p(x; \theta) = p_z(g(x; \theta)) \prod_{i=1}^{d} \left| \frac{\partial g^i(x_i; h^i(x \odot A_{i,:}), \theta)}{\partial x_i} \right|.
$$

The function $w(A)$ that enforces the acyclicity is expressed as suggested by **?** as

$$
w(A) := \operatorname{tr} \left((I + \alpha A)^d \right) - d \propto \operatorname{tr} \left(\sum_{k=1}^{d} \alpha^k A^k \right),
$$

where $\alpha \in \mathbb{R}_+$ is a hyper-parameter that avoids exploding values for $w(A)$. In the case of positively valued A, an element (i, j) of $A^k = \underbrace{AA...A}_{k \text{ terms}}$ is non-null if and only if there exists a path going from node j to node i that is made of exactly k edges. Intuitively $w(A)$ expresses to which extent the graph is cyclic. Indeed, the diagonal elements (i, i) of A^k will be as large as there are many paths made of edges that correspond to large values in A from a node to itself in k steps.

In comparison to our work, **?** use a quadratic loss on the corresponding linear structural equation model (SEM) as the score function $F(A)$. By attaching normalizing flows to topology learning, our method has a continuously adjustable level of complexity and does not make any strong assumptions on the form of the conditional factors.

7.4.3 *Stochastic adjacency matrix*

In order to learn the BN topology from the data, the adjacency matrix must be relaxed to contain reals instead of booleans. It also implies that the graph induced by A does not formally define a DAG during training. Work using NO TEARS to perform topology learning directly plug the real matrix A in (7.6) [????] however this is inadequate because the quantity of information going from node j to node i does not continuously relate to the value of $A_{i,j}$. Either the information is null if $A_{i,j} = 0$ or it passes completely if not. Instead, we propose to build the stochastic pseudo binary valued matrix A' from A, defined as

$$A'_{i,j} = \frac{e^{\frac{\log(\sigma(A_{i,j}^2)) + \gamma_1}{T}}}{e^{\frac{\log(\sigma(A_{i,j}^2)) + \gamma_1}{T}} + e^{\frac{\log(1 - \sigma(A_{i,j}^2)) + \gamma_2}{T}}},$$

where $\gamma_1, \gamma_2 \sim \text{Gumbel}(0, 1)$ and $\sigma(a) = 2(\text{sigmoid}(2a^2) - \frac{1}{2})$ normalizes the values of A between 0 and 1, being close to 1 for large values and close to zero for values close to 0. The hyper-parameter T controls the sampling temperature and is fixed to 0.5 in all our experiments. In contrast to directly using the matrix A, this stochastic transformation referred to as the Gumbel-Softmax trick in the literature [??] allows to create a direct and continuously differentiable relationship between the weights of the edges and the quantity of information that can transit between two nodes. Indeed, the probability mass of the random variables $A'_{i,j}$ is mainly located around 0 and 1, and its expected value converges to 1 when $A_{i,j}$ increases.

7.4.4 *Optimization*

We rely on the augmented Lagrangian approach to solve the constrained optimization problem (7.6) as initially proposed by **?**. This optimization procedure requires solving iteratively the following sub-problems:

$$\max_A \mathbb{E}_{\gamma_1, \gamma_2} [F(A)] - \lambda_t w(A) - \frac{\mu_t}{2} w(A)^2, \qquad (7.8)$$

where λ_t and μ_t respectively denote the Lagrangian multiplier and penalty coefficients of the sub-problem t.

We solve these optimization problems with mini-batch stochastic gradient ascent. We update the values of γ_t and μ_t as suggested by **?** when the validation loss does not improve for 10 consecutive epochs. Once $w(A)$ equals 0, the adjacency matrix is acyclic up to numerical errors. We recover an exact DAG by thresholding the elements of A while checking for acyclicity with a path finding algorithm. We provide additional details about the optimization procedure used in our experiments in Appendix **??**.

Table 7.1: Datasets description. d=Dimension of the data. V=Number of edges in the ground truth Bayesian Network.

Dataset	d	V	Train	Test
Arithmetic Circuit	8	8	10,000	5,000
8 Pairs	16	8	10,000	5,000
Tree	7	8	10,000	5,000
Protein	11	20	6,000	1,466
POWER	6	$\leqslant 15$	1,659,917	204,928
GAS	8	$\leqslant 28$	852,174	105,206
HEPMASS	21	$\leqslant 210$	315,123	174,987
MINIBOONE	43	$\leqslant 903$	29,556	3,648
BSDS300	63	$\leqslant 1,953$	1,000,000	250,000

7.5 EXPERIMENTS

In this section, we demonstrate some applications of graphical NFs in addition to unifying NFs and BN under a common framework. We first demonstrate how pre-loading a known or hypothesized DAG structure can help finding an accurate distribution of the data. Then, we show that learning the graph topology leads to relevant BNs that support generalization well when combined with ℓ_1-penalization. Finally, we demonstrate that mono-step normalizing flows made of graphical conditioners are competitive density estimators.

7.5.1 *On the importance of graph topology*

The following experiments are performed on four distinct datasets, three of which are synthetic, such that we can define a ground-truth minimal Bayesian network, and the fourth is a causal protein-signaling network derived from single-cell data [**?**]. Additional information about the datasets are provided in tab:ds$_d$escandinAppendix**??**.Foreachexperimentalrunwefirstrandomly

PRESCRIBED TOPOLOGY   Rarely do real data come with their associated Bayesian network however oftentimes experts want to hypothesize a network topology and to rely on it for the downstream tasks. Sometimes the topology is known a priori, as an example the sequence of instructions in stochastic simulators can usually be translated into a graph topology (e.g. in probabilistic programming [**??**]). In both cases, graphical conditioners allow to explicitly take advantage of this to build density estimators while keeping the assumptions made about the network topology valid.

Table 7.2: Graphical vs autoregressive conditioners combined with monotonic normalizers. Average log-likelihood on test data over 5 runs, under-scripted error bars are equal to the standard deviation. Results are reported in nats; higher is better. The best performing architecture for each dataset is written in bold. *Graphical conditioners clearly lead to improved density estimation when given a relevant prescribed topology in 3 out of the 4 datasets.*

Conditioner	Graphical	Autoreg.
Arithmetic Circuit	3.986817$_{0.155741}$	3.059781$_{0.381778}$
8 Pairs	-9.398858$_{0.061945}$	-11.503608$_{0.271145}$
Tree	-6.845759$_{0.016316}$	-6.960843$_{0.054900}$
Protein	6.460093$_{0.075050}$	7.516195$_{0.099126}$

tab:known$_t$opopresentsthetestlikelihoodofautoregressiveandgraphicalnormalizir

HYBRID GENERATIVE MODELS

OUTLINE

Hybrid modelling reduces the misspecification of expert models by combining them with machine learning (ML) components learned from data. Like for many ML algorithms, hybrid model performance guarantees are limited to the training distribution. Leveraging the insight that the expert model is usually valid even outside the training domain, we overcome this limitation by introducing a hybrid data augmentation strategy termed *expert augmentation*. Based on a probabilistic formalization of hybrid modelling, we show why expert augmentation improves generalization. Finally, we validate the practical benefits of augmented hybrid models on a set of controlled experiments, modelling dynamical systems described by ordinary and partial differential equations.

8.1 INTRODUCTION

Generalization to unseen data is a key property of a useful model. When training and test data are independently and identically distributed (IID), one way to check generalization is by evaluating the model on a held out subset of the training data or with k-fold cross validation. Unfortunately, this setting is often unrealistic because the training scenario is rarely fully representative of the test scenario. This has motivated lot of recent research efforts to focus on the robustness of ML models [**???**]. Common strategies can be broadly grouped in two categories: The first class of methods aims at aligning specific properties of the model (e.g., invariance, equivariance, monotonicity, etc.) with expertise on the problem of interest [**????**]. The second category is data focused [**????**] and leverages variations present in the training data, e.g. by minimizing worst case sub-group performance, to achieve robustness.

The data oriented methods, which include Group-DRO [**?**] and Invariant Risk Minimization [**?**, IRM], can be very appealing because they only require implicit specification of invariances via domains or environments. However, these methods' performance is limited to variations present in the training data and the inductive bias of the ML algorithm. This may be insufficient when the modelling problem is too complex or the variations of interest are not present in the training data. On the other hand, methods based on domain-specific expertise do not suffer from such limitations. Embedding expertise
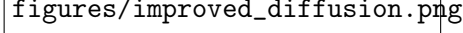
figures/improved_diffusion.png

Figure 8.1: APHYNITY, an existing hybrid modelling strategy, is unable to predict accurately the dynamic of a 2D diffusion reaction for a shifted test distribution although it predicts well configuration that follows the training distribution. On the opposite, APHYNITY+, the same model fine-tuned with our data augmentation, generalizes to shifted distributions as expected from the validity of the underlying physics.

into a model can be done via architectural inductive biases [**??**], data augmentation [**?**], or a learning objective [**?**] that enforces established symmetries of the problem. As an example, simple data augmentation techniques combined with convolutions lead to excellent performance on natural image problems [**?**]. Another natural approach to embed expertise in ML models, and the one studied in this paper, is called hybrid learning (HyL). HyL combines an expert model (e.g., physics-motivated equations) with a learned component that improves the expert model so that the combination better fits real-world data. A particularity of HyL is the central role played by the expertise, which is supposed to provide a simple and well-grounded parametric description of the process considered. HyL usually considers the expert model as an analytical function, or as a set of equations, that relates the expert parameters to the quantity of interest. The expert model is often motivated by the underlying physics of the system considered. Hence, we will use the terms *expert* model and *physical* model interchangeably.

In recent work [**??????**], HyL demonstrated success in complementing partial physical models and improving the inference of the corresponding parameters. However, contrarily to the common belief that HyL achieves better generalization than black box ML models, we argue that hybrid models do not meet their promise regarding robustness. Although HyL achieves strong performance on IID test distributions by exploiting the inductive bias of the expert models, we show that their performance collapses when the test domain is not included in the training domain. This is unsatisfactory as the expert model is typically well-defined for a range of parameters that can correspond to realistic data far outside of the training distribution. A test distribution not covered by the training data, but for which an

[node distance=.5cm and .5cm, mynode/.style=draw,circle,text
width=.6cm,align=center, $phantom_node/.style =$
$circle, textwidth = .6cm, align = center, halo/.style =$
$linejoin = round, double, linecap = round, double distance = 1, shorten >= -1/2, shorten <= -1/$
7mm][mynode]$(zp)Z_e$; [mynode, right=of zp] (x) X; [mynode,
right=of x] (za) Z_a; [mynode, below=of zp] (yp) Y_e; [mynode,
below=of za] (y) Y; zp) edge[-latex]yp); x) edge[-latex]yp);
x) edge[-latex]y); yp) edge[-latex]y); za) edge[-latex]y);
za) edge[-latex]y);
(A) at ([yshift=.1cm,xshift=-.1cm]zp.north west); (B) at
([yshift=.1cm,xshift=.15cm]x.north east); (C) at
([xshift=-.1cm,yshift=-.15cm]yp.south west); (D) at
([yshift=-1.1cm,xshift=.1cm]x.south);
$(A_right)at([yshift = .5cm, xshift =$
$0cm]A); at(A_right)[right]Expertmodel;$
[dashed] plot[smooth cycle] coordinates (A) (B) (D) (C);

Figure 8.2: A hybrid probabilistic model which describes the relationship be-
tween the input X and the output Y for a configuration of the sys-
tem as defined by the latent variables Z_e and Z_a. The prescribed
expert model defines the conditional density $p(y_e|z_e, x)$, where
Y_e is an approximation of Y. Hybrid learning aims at learning
the conditional distribution $p(y|z_a, y_e, x)$.

expert model exists, happens often in the real world. As an example,
? apply HyL to a pharmacological model describing the effect of a
COVID-19 treatment for which only a limited quantity of real-world
data is available. In this context, although the underlying biochemical
dynamic of treatments is well modelled, data is often scarce and bi-
ased. Therefore, the hybrid model does not necessarily generalize to
configurations that are well modelled by the pharmacological model
but unseen during training.

We introduce *expert augmentations* for training augmented hybrid
models (AHMs), a procedure that extends the range of validity of hy-
brid models and improves generalization as pictured by $fig:diffusion_shift.Ourcontributionistofirstfor$
$1)Learningaprobabilisticmodelpartiallydefinedbytheexpertmodel; 2)Performinginferenceovertl$
$tunethehybridmodelonanexpert-augmenteddatasetthatincludesdistributionshifts(seeresultso$
$definedoutsideofthetrainingdistribution.OurexperimentsonvariouscontrolledHyLproblemsdem$
$of-the-artHyLalgorithm.$

8.2 HYBRID LEARNING

In order to show that our proposed expert augmentations lead to ro-
bust models, we first formalize hybrid learning with the probabilistic
model depicted in $fig:gen_bnet.InthisBayesiannetwork, capitallettersdenoterandomvariables(e.g., \lambda$
$\in \mathcal{Y})$. In our formalism, the expert model is a conditional density
$p(y_e|x, z_e)$ that describes the distribution of the *expert response* Y_e
to an input x together with a parametric description of the system

z_e, denoting expert or physical parameters. We augment the expert model with the *interaction model* which is a conditional distribution $p(y|x, y_e, z_a)$ that describes the distribution of the observation Y given the input x, the expert model response y_e, and a parametric description of the interaction model z_a.

Our final goal is to create a robust predictive model $p(y|x, (x_o, y_o))$ of the random variable Y, given the input x together with independent observations (x_o, y_o) of the same system, where the subscript o denotes an observed quantity. As a concrete example, we consider predicting the evolution of a damped pendulum (described in sec:problems$_d$escription)g = $[\theta, \dot{\theta}]$) and a sequence of observations of the same pendulum. The expert model we assume is able to describe a frictionless pendulum whose dynamic is only characterized by one parameter $z_e := \omega_0$, denoting its fundamental frequency. A perfect description of the system should model the friction with a second parameter $z_a := \alpha$, the damping factor. In this problem, (x_o, y_o) and (x, y) are IID realization of the same pendulum which corresponds, in general terms, to samples from $p(x, y|z_a, z_e)$ for some fixed but unknown values of z_a and z_e. The expert variables z_e (e.g., ω_0) together with z_a (e.g., α) should accurately describe the system that produces Y (e.g., the evolution of the pendulum's angle and speed along time) from X (e.g., the initial pendulum's state). In our setting we assume that we are given a pair (x_o, y_o) (e.g., past observations) from which we can accurately infer the state of the system (z_a, z_e) as described by the interaction and expert models, and then predict the distribution of Y for a given input x (e.g., forecasting future observations) to the same system. Because the interaction between z_e and y is essentially defined by the expert model, it should be possible, and preferable, to learn an accurate predictive model of Y whose accuracy is independent from the training distribution of the expert variables z_e. Provided all probability distributions in fig:gen$_b$netareknown, theBayesoptimalhybridpredictorp_B can be written as

$$p_B(y|x, (x_o, y_o)) = \mathbb{E}_{p(z_a, z_e|(x_o, y_o))} [p(y|x, z_a, z_e)]. \qquad (8.1)$$

We observe that the Bayes optimal predictor explicitly depends on the posterior $p(z_a, z_e|(x_o, y_o))$ which is itself a function of the marginal distribution over z_e. This may preclude the existence of a good predictor that is invariant to shift of $p(z_e)$. However, in the following we will consider that the pair (x_o, y_o) contains enough information about the parameters z_a, z_e. As a consequence, the posterior distribution shrinks around the correct parameters value and the effect of the prior becomes negligible.

8.2.1 *Hybrid generative modelling*

We consider expert models that are deterministic; that is, for which $p_\theta(y_e|x, z_e)$ is a Dirac distribution. The expert model describes the system as a function $f_e : \mathcal{X} \times \mathcal{Z}_e \rightarrow \mathcal{Y}_e$ that computes the response y_e to an input x, parameterized by expert variables z_e. The goal of hybrid modelling is to augment the expert model with a learned component from data as depicted in fig:gen$_b$net. Formally, given a dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$ of N IID samples, we aim to learn the interaction model $p_\theta(y|x, y_e, z_a)$ that fits the data well but is close to the expert model. For example, we could define closeness via a small L2-distance between expert and hybrid outputs or via a small Kullback-Leibler (KL) divergence between the marginal distributions of Y and Y_e.

Learning a model that is close to the expert model and fits the training data well is a hard problem. However, the APHYNITY algorithm [?] and the Hybrid-VAE [?, HVAE] are two recent approaches that offer promising solutions to this problem. We now briefly describe these two methods and how they can be used to approximate the Bayes optimal predictor of (8.1). Our augmentation strategy is compatible (and effective) with both approaches.

APHYNITY. ? formulate hybrid learning in a context where the expert model is an ordinary differential equation (ODE). They consider an additive hybrid model that should perfectly fit the data, which is equivalent to assuming the conditional distribution $p_\theta(y|x, y_e, z_a)$ is a Dirac distribution. Formally, they solve the optimization problem

$$\min_{z_e, F_a} \|F_a\| \quad \text{s.t.} \quad \forall(x, y) \in \mathcal{D}, \forall t, \frac{dy_t}{dt} = (F_e + F_a)(y_t)$$
$$\text{with} \quad y_0 := x, \tag{8.2}$$

where $\|\cdot\|$ is a norm operator on the function space, $F_a : \mathcal{Y}_t \times \mathcal{Z}_a \rightarrow \mathcal{Y}_t$ is a learned function, $F_e : \mathcal{Y}_t \times \mathcal{Z}_e \rightarrow \mathcal{Y}_t$ defines the expert model and $\mathcal{D}$ is a dataset of initial states $x := y_0$ and sequences $y \in \mathcal{Y} := (\mathcal{Y}_t)^k$, where k is the number of observed timesteps. APHYNITY solves this problem with Lagrangian optimization and Neural ODEs [?] to compute derivatives. In the context of ODEs, the random variable X is the initial state of the system at t_0 and Y is the observed sequence of k states between t_0 and t_1.

This formulation only considers learning a missing dynamic for one realization of the system described by fig:gen$_b$net, for a single z_a and z_e. However, we are interested in learning a hybrid model that works for the full set of systems described by fig:gen$_b$net. As suggested in ?, we use an encoder network $g_\psi : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{Z}_a \times \mathcal{Z}_e$ that corresponds to a Dirac distribution located at g_ψ as the approximate posterior $q_\psi(z_a, z_e|x, y)$. The interaction

model is a product of Dirac distributions whose locations correspond to the solution of the ODE

$$\frac{dy_t}{dt} = F_e(y_t, z_e) + F_a(y_t, z_a; \theta), \quad y_0 := x. \tag{8.3}$$

Hence the corresponding approximate Bayes predictor replaces the parameters (z_a, z_e) in (8.3) with the prediction of g_ψ and predicts a product of Dirac distributions.

HYBRID-VAE (HVAE).   In contrast to APHYNITY, the model proposed by **?** is not limited to additive interactions between the expert model and the ML model, nor to ODEs. Instead, their goal is to learn the generative model described by fig:gen$_b$net. They achieve this with a variational auto-encoder (VAE) where the decoder specifically follows fig : gen$_b$net. Similarly to the amor predicts a posterior distribution over z_a and z_e, and the model is trained with the classical Evidence Lower Bound on the likelihood (ELBO). **?** observe that relying only on an architectural inductive bias and maximum likelihood training is not enough to ground the generative model to the expert equations. They propose to add three regularizers $R_{PPC}, R_{DA,1}$, and $R_{DA,2}$ that encourage the generative model to rely on the expert model. The final objective is

$$\max_{\theta, \psi} \mathbb{E}_{\mathcal{D}} \left[\text{ELBO}((x, y); \psi, \theta) \right] + \alpha R_{PPC} + \beta R_{DA,1}$$
$$+ \gamma R_{DA,2}. \tag{8.4}$$

The first regularizer, R_{PPC}, encourages the marginal distribution of samples generated by the complete model to be close to the marginal distribution that would be only generated by the physical model. The two other regularizers specifically require the encoder network for z_e to be made of two sub-networks. The first network filters the observations to keep only what can be generated by the expert model alone, and the second should map the filtered observations to the posterior distribution over z_e. $R_{DA,1}$ penalizes the objective if the observations generated by the expert model are not close to the filtered observations. Finally, $R_{DA,2}$ relies on data augmentation with the expert model to enforce that the second sub-network correctly identifies the expert variables z_e when the observations are correctly filtered. We refer the reader to **?** for more details on HVAE. For HVAE, the approximate predictor takes the form described by (8.1) where $p(z_a, z_e|(x_o, y_o))$ is approximated by the encoder $q_\psi(z_a, z_e|x, y)$ and $p(y|x, z_e, z_a)$ by the learned hybrid generative model.

8.3   ROBUST HYBRID LEARNING

We now formalize our definition of out of distribution (OOD) and robustness. In general, a test scenario is OOD if the joint test distribution $\tilde{p}(x, y)$ is different from the training distribution $p(x, y)$, that

is $d(\tilde{p}, p) > 0$ for any properly defined divergence or distance d. In the following, we reduce our discussion to a sub-class of distribution shifts for which the marginal train and test distributions over z_e may be different, $d(p(z_e), \tilde{p}(z_e)) > 0$, but the marginals of z_a and x are constant. As a consequence, the joint distribution of (x, y) pairs is also shifted. Formally, the training and test distributions are respectively defined as

$$p(x, y) := \mathbb{E}_{p(z_e)p(z_a)p(y_e|z_e, x)} \left[p(x)p(y|z_a, x, y_e) \right],$$
$$\tilde{p}(x, y) := \mathbb{E}_{\tilde{p}(z_e)p(z_a)p(y_e|z_e, x)} \left[p(x)p(y|z_a, x, y_e) \right].$$

In this context, we demonstrate, theoretically and empirically, that classical hybrid models fail. To address this failure, we introduce *augmented hybrid models* and show that, under some assumptions, they achieve optimal performance on both the train and test distributions.

Our goal is to learn a predictive model

$$p_{\theta,\psi}(y|x, (x_o, y_o)) = \mathbb{E}_{\substack{q_\psi(z_a, z_e|x_o, y_o) \\ p(y_e|z_e, x)}} \left[p_\theta(y|y_e, x, z_a) \right]$$

that is *exact* on both the train and test domains when they follow the aforementioned training and testing distribution shifts. We say that a learned predictive model $\hat{p}(a|b)$ is $\mathcal{E}$-*exact*, or *exact* on the sample space $\mathcal{E}$, if $\hat{p}(a|b) = p(a|b) \quad \forall (a, b) \in \mathcal{E}$. Here we qualify a predictive model as *robust* to a test scenario if its *exactness* on the training domain is sufficient to ensure exactness on the test domain.

We now define an augmented distribution $+p(z_e)$ over the expert variables whose support $+\mathcal{Z}_e$ includes the joint support $\mathcal{Z}_e \cup \tilde{\mathcal{Z}}_e$ between the train and test distribution of the physical parameters. As depicted in fig:ood$_v$izu, wedenotethecorrespondingsupportovertheobservationspace$X \times \mathcal{Y}$ as $+\Omega, \Omega$, and $\tilde{\Omega}$, respectively. In this context, and with **hyp:first**, we may demonstrate that even under perfect learning, classical hybrid learning algorithms do not produce an $\tilde{\Omega}$-*exact* predictor while our augmentation strategy does. [A8.3] Hybrid modelling learns an interaction model $p_\theta(y|y_e, x, z_a)$ that is $+\Omega$-*exact*. Although strong, **hyp:first** is consistent with the recent literature on hybrid modelling, which assumes that $p(y_e|x, z_e)$ is an accurate description of the system, thereby $p_\theta(y|y_e, x, z_a)$ should not be overly complex. As an example, we consider an additive interaction model in our experiments for which extrapolation to unseen y_e holds if this assumption is correct. That said, we still notice that the exactness of the interaction model p_θ on $+\Omega$ is insufficient to prove that the predictive model $p_{\theta,\psi}$ is $+\Omega$-*exact*. Indeed, the encoder q_ψ is only trained on the training data and cannot rely on a strong inductive bias in contrast to p_θ. Thus, even if the encoder is exact on the training distribution, the corresponding predictive model does not achieve exactness outside Ω.

8.3.1 *Expert augmentation*

We propose a data augmentation strategy to improve the robustness of hybrid models to unseen test scenarios. Once trained, the hybrid model is composed of an encoder q_ψ and an interaction model p_θ that are respectively Ω- and $+\Omega$-*exact*. We may create a new training distribution with a support over $+\Omega$ by sampling physical parameters z_e from a distribution that covers $+\mathcal{Z}_e$. We can then train the encoder q_ψ on $+\Omega$, under perfect training the corresponding predictive model $p_{\theta,\psi}(y|x, (x_o, y_o))$ is $+\Omega$-*exact*, hence exact on both train and test domains.

Our learning strategy is grounded in existing hybrid modelling algorithms. Here, we focus on APHYNITY and HVAE, but our approach is applicable to other HyL algorithms. We first train an encoder q_ψ and a decoder p_θ with a HyL algorithm. Together with experts we then decide on a realistic distribution $+p(z_e)$ and create a new dataset $+\mathcal{D}$ by sampling from the hybrid generative model defined by fig:gen$_b$net and the interaction model p_θ. A notable difference between the augmented training set $+\mathcal{D}$ and the original training set $\mathcal{D}$ is that the former contains ground truth values for the expert's variables z_e. As we assume that the interaction model is $+\Omega$-*exact*, we freeze it and only fine-tune the encoder q_ψ on $+\mathcal{D}$. We use a combination of the loss function ℓ of the original HyL algorithm (e.g., (8.4) for HVAE, and the Lagrangian of (8.2) for APHYNITY) and a supervision on the latent variable objective to learn a decoder that solves

$$\psi = \arg\min_\psi \mathbb{E}_{+\mathcal{D}} \left[\ell(x, y; \theta, \psi) - \log q_\psi(z_e|x, y) \right].$$

In our experiments we chose a Gaussian model for the posterior, which is equivalent to a mean square error (MSE) loss on the physical parameters. We provide a detailed description of the expert augmentation scheme in app:ahm$_d$esc.

As a side note, we would like to emphasize the difference between the data augmentation proposed in this paper and the one from **?**. While HVAE also requires to sample new physical parameters z_e, it is only to ensure that a sub-part of the encoder is able to infer correctly z_e given y_e. This augmentation does not contribute to robustness distribution shifts on y in contrast to ours.

8.4   EXPERIMENTS

8.4.1 *Problem description*

We assess the benefits of expert augmentation on three controlled problems described and simulated by the ODE

$$\frac{dy_t}{dt} = F_e(y_t; z_e) + F_a(y_t; z_a), \tag{8.5}$$

where $F_e : \mathcal{Y}_t \times \mathcal{Z}_e \to \mathcal{Y}_t$ is the expert model and $F_a : \mathcal{Y}_t \times \mathcal{Z}_a \to \mathcal{Y}_t$ complements it. In our notation X is the initial state y_0 and the response Y is the sequence of states $y_{1:t_1} := [y_{i\Delta t}]_{i=1}^{t_1/\Delta t}$. For all experiments we train the models to maximize $p_{\theta,\psi}(y = y_{1:t_1} | x = y_0)$ on the training data. We validate and test the models on the predictive distribution $p(y = y_{1:t_2} | x = y_0, x_o = y_0, y_o = y_{1:t_1})$, where $t_2 > t_1$ assesses the generalization over time. A brief description of the different problems is provided below.

The damped pendulum is often used as an example in the hybrid modelling literature [**??**]. The system's state at time t is $y_t = \begin{bmatrix} \theta_t & \dot{\theta}_t \end{bmatrix}^T$, where θ_t is the angle of the pendulum at time t and $\dot{\theta}_t$ its angular speed. The evolution of the state over time is described by (8.5), where $z_e := \omega$, $z_a = \alpha$ and

$$F_e := \begin{bmatrix} \dot{\theta} & -\omega_0^2 \sin\theta \end{bmatrix}^T \quad \text{and} \quad F_a := \begin{bmatrix} 0 & -\alpha\dot{\theta} \end{bmatrix}^T. \tag{8.6}$$

The corresponding systems are defined by the damping factor α and ω_0, the fundamental frequency of the pendulum.

The RLC series circuits are electrical circuits made of 3 electrical components that may model a large range of transfer functions. These models are often used in biology (e.g., the Hodgkin-Huxley class of models [**?**], in photoplethysmography [**?**]) and in electrical engineering to model the dynamics of various systems. The system's state at time t is $y_t = \begin{bmatrix} U_t & I_t \end{bmatrix}^T$, where U_t is the voltage around the capacitance and I_t the current in the circuit. The evolution of the state over time is described by (8.5), where $z_e := \{L, C\}$, $z_a = \{R\}$ and

$$F_e := \begin{bmatrix} \frac{I_t}{C} \\ \frac{1}{L}(V(t) - U_t) \end{bmatrix} \quad \text{and} \quad F_a := \begin{bmatrix} 0 \\ -\frac{R}{C}I_t \end{bmatrix}. \tag{8.7}$$

The dynamics described by the RLC circuit is more diverse than for the pendulum and the system can be hard to identify. This system is characterised by the resistance R, capacitance C, and inductance L, provided $V(t)$ is known.

The 2D reaction diffusion was used by **?** to assess the quality of APHYNITY. It is a 2D FitzHugh-Nagumo on a 32×32 grid. The system's state at time t is a $2 \times 32 \times 32$ tensor $y_t = \begin{bmatrix} u_t & v_t \end{bmatrix}^T$. The evolution of the state over time is described by (8.5), where $z_e := \{a, b\}$, $z_a = \{k\}$ and

$$F_e := \begin{bmatrix} a\Delta u_t \\ b\Delta v_t \end{bmatrix} \quad \text{and} \quad F_a := \begin{bmatrix} R_u(u_t, v_t; k) \\ R_v(u_t, v_t) \end{bmatrix}, \tag{8.8}$$

where Δ is the Laplace operator, the local reaction terms are $R_u(u, v; k) = u - u^3 - k - v$ and $R_v(u, v) = u - v$. This model is interesting to study

Dataset		APH.	HVAE	APH.+	HVAE+
2*Pendulum	Valid.	$6_{\pm2}$	$3_{\pm1}$	$6_{\pm2}$	$2_{\pm1}$
	Test	$66_{\pm9}$	$117_{\pm10}$	$10_{\pm4}$	$11_{\pm2}$
2*RLC	Valid.	$6_{\pm3}$	$38_{\pm2}$	$7_{\pm5}$	$28_{\pm1}$
	Test	$17_{\pm3}$	$25_{\pm2}$	$5_{\pm2}$	$12_{\pm1}$
2*Diffusion	Valid.	$2_{\pm0}$	$2_{\pm0}$	$2_{\pm0}$	$2_{\pm0}$
	Test	$27_{\pm2}$	$32_{\pm10}$	$3_{\pm1}$	$2_{\pm0}$

Table 8.1: Comparison of mean relative precision (in %, $\pm$ indicates one standard deviation) over 10 runs of predicted physical parameters of different hybrid modelling strategies in validation and OOD test settings. Augmented versions are denoted with a $+$. *While the accuracy of APHYNITY and HVAE is good on the validation set, it collapses on the OOD test set. On the opposite, the augmented versions perform well on both validation and test sets.*

as it considers a state space for which neural architectures may have a real advantage compared to other ML models.

In the following experiments we analyze the effect of our data augmentation strategy on APHYNITY and HVAE. All models explicitly use the assumption that the interaction model follows the structure of (8.5). For each problem the validation and test sets are respectively IID and OOD with respect to the training distribution. The best models are always selected based on validation performance, that is with samples from Ω. We provide additional details on the different expert models, dataset creation, and neural networks architectures in app:exp$_d$etails.

8.4.2   *Results*

PERFORMANCE GAIN FROM AUGMENTATION.   *This experiment demonstrates that HVAE and APHYNITY are not robust to OOD test scenarios in opposition to the corresponding AHMs, as shown in fig:diffusion$_s$hiftforthe2Ddiffusionproble 100 in some cases.*

STABILITY FOR NON-EXACT MODELS.   *The empirical results from fig:log$_m$seareveryimportantastheyshowthatevenwhenthedecoderisnotΩ-exact (and hence not $+\Omega$-exact), augmentation is still useful. In particular, tab:param$_a$ccshowsthatthe and neither should be the decoder. This table shows the relative error on the physical parameters computed as $\sum_{i=1}^{k} \frac{1}{k} \left| \frac{z_e^i - \mu_\theta^i}{z_e^i} \right|$, where μ_θ^i is the estimated most likely value of the ith component of the physical parameters. We first notice that APHYNITY and HVAE perform differently and their performance depends on the specific problem. While APHYNITY accurately estimates the physical parameters on the IID validation set for the 3 problems, HVAE's performance are mixed on the RLC problem as it makes prediction that are 38% away from the nominal parameter value on average whereas APHYNITY reduces this error to 6%. Interestingly, we observe that the pro-*

posed augmentation strategies improve the encoder such that it accurately estimates the physical parameters also on the OOD test set even for HVAE on the RLC problem. This confirms that the augmentation strategy is helpful even when the hybrid model is not Ω-exact. As a conclusion, augmented hybrid learning outperforms classical hybrid learning both on the predictive accuracy and at inferring the expert variables.

EFFECT OF OUT OF EXPERTISE SHIFT.   *This experiment shows that our augmentation strategy may remain beneficial even when the train and test supports of z_a are not identical. This scenario corresponds to samples (x, y) generated by $(z_a, z_e) \in (\ast \mathcal{Z}_a \setminus \mathcal{Z}_a) \times \tilde{\mathcal{Z}}_e$ depicted by the violet domains in fig:ood$_v$izu.Infig: za$_s$hiftedweobservethelog $-$ MSEofaugmentedandnon$-$ augmentedhybridmodelstrainedfor$(z_a, z_e) \in \mathcal{Z}_a \times \mathcal{Z}_e$ on test data that are generated with $(z_a, z_e) \in \tilde{\mathcal{Z}}_a \times \tilde{\mathcal{Z}}_e$. For the pendulum, the support over $z_a = \alpha$ is $[0, 0.3]$ in train and $[0.3, 0.6]$ in test; For the 2D reaction diffusion, $z_a = k$ is $[0.003, 0.005]$ in train and $[0.005, 0.008]$ in test. We observe that augmented models outperform the original models by a large margin. These results suggest that augmentation could be very valuable in practice, even when the distribution shift is also caused by non expert variables. However, if the shift on z_a becomes the dominant effect, augmented models also eventually becomes vulnerable to shifts on z_e as demonstrated by supplementary experiments in app:exp$_d$etails.*

8.5   RELATED WORK

8.5.1   *Hybrid modelling*

Hybrid Learning (HyL), or gray box modelling as called in its early days in the 90's [?????], has been an appropriate method to learn models that are both expressive and interpretable, while also allowing them to be learnt on fewer data. The interest for HyL [???????] has greatly renewed since the outbreak of recent neural network architectures that simplify the combination of physical equations within ML models. As an example, Neural ODE [?] and convolutional neural networks [?, CNN] are privileged architectures to work with dynamical systems described by ODEs or PDEs. While most of the HyL's literature focus on the predictive performance of hybrid models, recent work have also showed that HyL may help to infer the physical parameter accurately [??]. This is aligned with ? (see Section 40.2.2.2) which observe that inference on incomplete models results in a systematic bias. Similar to HyL, they extend the model with nuisance parameters in order to improve its fidelity, and to reduce the systematic bias.

In this work, we decided to study ? and ? for two reasons that distinguish them from the rest of the HyL literature. First, these are notable examples of HyL algorithms that can be applied to a broad class of problems in contrast to papers that focus on specific applications [??]. Second, those methods also learn a reliable inference model for the physical parameters, suggesting that

the expert model is used properly in the generative model, which is a key assumption for our augmentation. While **?** *claim to achieve robustness with HyL, we argue that this statement is incomplete as HVAE fails in OOD settings. In particular, their approach is only able to generalize with respect to unseen time or initial state if the model correctly identifies the latent variables* z_a, z_e.

8.5.2   *Combining hybrid modelling and data augmentation*

Close to our idea is the one proposed in **?** *where they train a GAN model that improves the realism of a simulated image while conserving its semantic content (e.g. eyes colour) as modeled by the simulation parameters. The generated data with their annotations may then be used for a downstream task, such as inferring the properties of real images that corresponds to simulation parameters. The GAN objective from* **?** *requires that the two distributions induced by the semantic content of real and simulated data are identical. On the opposite, we consider training data that corresponds to expert parameters with limited diversity, and overcome this scarcity with expert augmentation. Another line of work similar to ours is Sim2Real, which considers the task of transferring a model trained on simulated data to real world [***]. Robust HyL, as a way to enhance simulations, could be used for Sim2Real.*

8.5.3   *Robust ML and Invariant Learning*

Various statistical methods have been introduced to ensure models generalize under distribution shift. Domain-adversarial objectives aimed at learning (conditionally) invariant predictors [***], GroupDRO [**] optimizing for worst-case loss over multiple domains and IRM [**] as well as sub-group calibration [**] aiming to satisfy calibration or sufficiency constraints to learn features invariant across domains. Extensions, able to infer domain labels from training data have been proposed as well [**], partially inspired by fairness objectives [**]. In contrast to AHM, all of these methods rely on the variation of interest being present in the training data.

8.6   DISCUSSION

We now examine the assumptions we made to derive our augmentation strategy and discuss potential limitations.

ERRONEOUS INTERACTION MODEL.   *The exactness of the hybrid component* $p_\theta(y|x, y_e, z_a)$ *is a critical assumption underlying our expert-based augmentation strategy. Unfortunately, this component is learned from training data only, hence we cannot prove its exactness on the test domain, which corresponds to a different domain* $\mathcal{Y}_e$. *However, we argue that soft assumptions on the class of interaction model may alleviate this problem. As an example, when we consider an additive hybrid model, as in APHYNITY [**],*

and embed this hypothesis into the interaction model, generalization to unseen y_e follows. When this assumption is too strong, we could still expect generalization of $p_\theta(y|x, y_e, z_a)$ because HyL drives y samples from p_θ to be close to y_e. It implies that the corresponding function approximator is smooth, which helps generalization to unseen scenarios. This contrasts with the encoder q_ψ for which a a good inductive bias usually is not available.

DIAGNOSTIC.   *While crucial, we cannot guarantee the exactness of the decoder p_θ in general because we only evaluate the encoder and the decoder jointly on data points (x, y, x_o, y_o). However, in some cases we can detect model misspecification by observing that the predictive model $p_{\theta,\psi}(y|x, x_o, y_o)$ is imperfect. Making this observation is not always simple as it requires prior knowledge on the expected accuracy of an exact model. However, when the system is deterministically identifiable, we may argue that the accuracy should be only limited by the intrinsic noise between x and y given z_a and z_e.*

RELAXING EXACTNESS.   *Even with a strong inductive bias on the decoder, achieving exactness is hopeless in practical settings. However, our experiments demonstrate that expert-augmentation works in practice. We can explain this by taking a look at fig:ood$_v$izu.Ifthegenerativemodelthatmapsxand(z_a, z_e) is incorrect, the mapping from $\mathcal{Z}_a$ and $\mathcal{Z}_e$ could be slightly off from $+\Omega$. However, this does not preclude the set of augmented samples to be closer to $+\Omega$ than Ω and to induce a better predictive model on $+\Omega$ than the original model trained only on Ω.*

LIMITATIONS   *We considered expert models that are parameterized by a small number of parameters, which can be covered densely via sampling. Covering densely a higher dimensional parameter space with the augmentation strategy becomes quickly impossible, hence a smarter sampling strategy would be required, such as worst-case sampling. Another difficulty is to choose a plausible range of parameters that contains both the train and the test support, this will often require a human expert in the loop. Finally, we assume that the train distribution of z_a should be representative of the test distribution, we empirically observed that a softer version of this assumption could be enough. However, performance will eventually decline as the support of the test distribution for z_a is far from the training domain.*

8.7   CONCLUSION

In this work, we describe HyL with a probabilistic model in which one component of the latent process, denoted the expert model, is known. In this context, we establish that state-of-the-art HyL algorithms are vulnerable to distribution shifts even when the expert model is well defined for such configurations. Grounded in this formalisation, we derive that expert augmentations induce robustness to OOD settings. We discuss how our assumptions can transfer

to real-world settings and describe how to diagnose potential shortcomings. Finally, empirical evidence asserts that expert augmentations may be beneficial even when one of our assumptions on the class of distribution shift is violated.

Our augmentation is applicable to a large class of hybrid models, hence it should benefit from future progress in HyL. Thus, we believe research in HyL and formally defining its targeted objectives is an important direction for further improving the robustness of hybrid models. As an example, the minimal description length principle [?] could be a great resource to investigate the balance between the model's capacity and robustness. Finally, robust ML models must eventually translate to real-world applications, hence a next step would be to apply AHMs to real-wold data. Paving the way to future research combining AHM with robust ML methods.

every picture/.style=line width=0.5pt
[x=0.75pt,y=0.75pt,yscale=-1,xscale=1,scale=0.8]
[color=rgb, 255:red, 0; green, 0; blue, 0 ,draw opacity=1][fill=rgb,
255:red, 219; green, 204; blue, 226 ,fill opacity=1.] (247.5,65) ..
controls (247.5,51.75) and (258.25,41) .. (271.5,41) − (407.5,41) ..
controls (420.75,41) and (431.5,51.75) .. (431.5,65) − (431.5,137) ..
controls (431.5,150.25) and (420.75,161) .. (407.5,161) − (271.5,161) ..
controls (258.25,161) and (247.5,150.25) .. (247.5,137) − cycle ;
[fill=rgb, 255:red, 210; green, 233; blue, 200 ,fill opacity=1.] (2,79.2) ..
controls (2,68.6) and (10.6,60) .. (21.2,60) − (133.8,60) .. controls
(144.4,60) and (153,68.6) .. (153,79.2) − (153,136.8) .. controls
(153,147.4) and (144.4,156) .. (133.8,156) − (21.2,156) .. controls
(10.6,156) and (2,147.4) .. (2,136.8) − cycle ; [fill=rgb, 255:red, 210;
green, 233; blue, 200 ,fill opacity=1.] (255,78.7) .. controls (255,68.1)
and (263.6,59.5) .. (274.2,59.5) − (386.8,59.5) .. controls (397.4,59.5) and
(406,68.1) .. (406,78.7) − (406,136.3) .. controls (406,146.9) and
(397.4,155.5) .. (386.8,155.5) − (274.2,155.5) .. controls (263.6,155.5) and
(255,146.9) .. (255,136.3) − cycle ;
[fill=rgb, 255:red, 241; green, 183; blue, 176 ,fill opacity=1.] (11,83.7)
.. controls (11,79.23) and (14.63,75.6) .. (19.1,75.6) − (69.4,75.6) ..
controls (73.87,75.6) and (77.5,79.23) .. (77.5,83.7) − (77.5,108) ..
controls (77.5,112.47) and (73.87,116.1) .. (69.4,116.1) − (19.1,116.1) ..
controls (14.63,116.1) and (11,112.47) .. (11,108) − cycle ; [fill=rgb,
255:red, 184; green, 205; blue, 225 ,fill opacity=1.] (81.5,111.6) ..
controls (81.5,107.13) and (85.13,103.5) .. (89.6,103.5) − (139.9,103.5) ..
controls (144.37,103.5) and (148,107.13) .. (148,111.6) − (148,135.9) ..
controls (148,140.37) and (144.37,144) .. (139.9,144) − (89.6,144) ..
controls (85.13,144) and (81.5,140.37) .. (81.5,135.9) − cycle ;
[fill=rgb, 255:red, 241; green, 183; blue, 176 ,fill opacity=1.]
(263,83.7) .. controls (263,79.23) and (266.63,75.6) .. (271.1,75.6) −
(321.4,75.6) .. controls (325.87,75.6) and (329.5,79.23) .. (329.5,83.7) −
(329.5,108) .. controls (329.5,112.47) and (325.87,116.1) .. (321.4,116.1)
− (271.1,116.1) .. controls (266.63,116.1) and (263,112.47) .. (263,108) −
cycle ; [fill=rgb, 255:red, 184; green, 205; blue, 225 ,fill opacity=1.]
(333.5,111.6) .. controls (333.5,107.13) and (337.13,103.5) ..
(341.6,103.5) − (391.9,103.5) .. controls (396.37,103.5) and (400,107.13) ..
(400,111.6) − (400,135.9) .. controls (400,140.37) and (396.37,144) ..
(391.9,144) − (341.6,144) .. controls (337.13,144) and (333.5,140.37) ..
(333.5,135.9) − cycle ;
[color=rgb, 255:red, 0; green, 0; blue, 0 ,draw opacity=1][fill=rgb,
255:red, 184; green, 233; blue, 134 ,fill opacity=0.3][fill=rgb, 255:red,
219; green, 204; blue, 226 ,fill opacity=1.] (495.5,89.2) .. controls
(495.5,81.91) and (501.41,76) .. (508.7,76) − (577.8,76) .. controls
(585.09,76) and (591,81.91) .. (591,89.2) − (591,128.8) .. controls
(591,136.09) and (585.09,142) .. (577.8,142) − (508.7,142) .. controls
(501.41,142) and (495.5,136.09) .. (495.5,128.8) − cycle ; [color=rgb,
255:red, 0; green, 0; blue, 0 ,draw opacity=1] [fill=rgb, 255:red, 210;
green, 233; blue, 200 ,fill opacity=1.](520.5,105.1) .. controls
(520.5,100.63) and (524.13,97) .. (528.6,97) − (578.9,97) .. controls
(583.37,97) and (587,100.63) .. (587,105.1) − (587,129.4) .. controls
(587,133.87) and (583.37,137.5) .. (578.9,137.5) − (528.6,137.5) ..
controls (524.13,137.5) and (520.5,133.87) .. (520.5,129.4) − cycle ;
(149.5,122) − (333.5,122) ; [shift=(148,122), rotate = 0] [fill=rgb,
255:red, 0; green, 0; blue, 0][line width=0.28] [draw opacity=0]
(8.93,-4.29) − (0,0) − (8.93,4.29) − cycle ; (153,135.9) − (353,135.9) ;
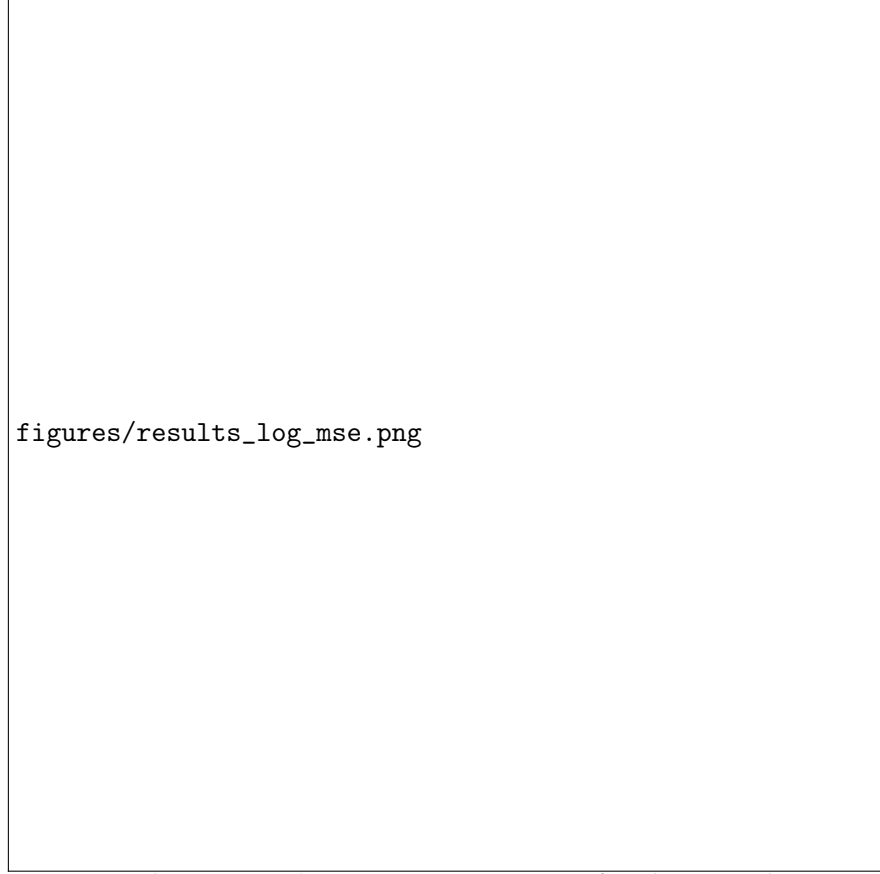
figures/results_log_mse.png

Figure 8.4: The average log-MSEs over 10 runs for three synthetic prob-
lems on the validation and test sets. We compare HVAE (in red)
and APHYNITY (in green), in light colours, to their expert aug-
mented versions HVAE+ and APHYNITY+, in darker colours.
*On the test sets, AHMs outperform the original models, and by a large
margin on the pendulum and diffusion problems. Moreover, augmen-
tation conserves the relatively good performance on the validation set
(IID w.r.t. the training set).*

figures/results_log_mse_za_OOD-4.png
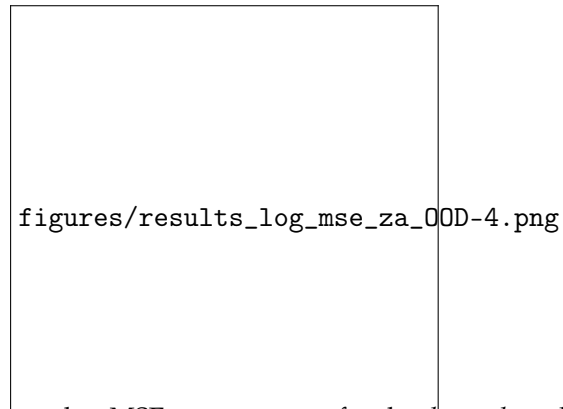
Figure 8.5: The average log-MSEs over 10 runs for the *damped pendulum* and
2D reaction diffusion problems on a test distribution for which z_a,
in addition to z_e, is also shifted. *AHM achieves better peformance
than stand HyL algorithms even when the test distribution support z_a
differs from the training.*

9

CONCLUSION

OUTLINE
???

Part III

APPENDIX

A

$\mathcal{A}$ *A supervised learning algorithm* .... **??**

$\mathcal{A}(\theta, \mathcal{L})$ *The model $\varphi_{\mathcal{L}}$ produced by algorithm $\mathcal{A}$ over $\mathcal{L}$ and hyperparameters θ* .... **??**

α_s *The proportion of samples in a random patch* .... **??**

α_f *The proportion of features in a random patch* .... **??**

b_l *The l-th value of a categorical variable* .... **??**

B *A subset $B \subseteq V$ of variables* .... **??**

c_k *The k-th class* .... **??**

C_p^k *The number of k-combinations from a set of p elements* **??**

$C(N)$ *The time complexity for splitting N samples* .... **??**

$\mathbb{E}$ *Expectation* ....

$\overline{E}(\varphi_{\mathcal{L}}, \mathcal{L}')$ *The average prediction error of $\varphi_{\mathcal{L}}$ over $\mathcal{L}'$* .... **??**

$Err(\varphi_{\mathcal{L}})$ *The generalization error of $\varphi_{\mathcal{L}}$* .... **??, ??**

$H(X)$ *The Shannon entropy of X* .... **??**

$H(X|Y)$ *The Shannon entropy of X conditional to Y* .... **??**

$\mathcal{H}$ *The space of candidate models* .... **??**

$i(t)$ *The impurity of node t* .... **??, ??**

$i_R(t)$ *The impurity of node t based on the local resubstitution estimate* .... **??, ??**

$i_H(t)$ *The entropy impurity of node t* .... **??**

$i_G(t)$ *The Gini impurity of node t* .... **??**

$\Delta i(s, t)$ *The impurity decrease of the split s at node t* .... **??**

$I(X; Y)$ *The mutual information between X and Y* .... **??**

$Imp(X_j)$ *The variable importance of X_j* .... **??, ??**

J *The number of classes* .... **??**

K *The number of folds in cross-validation* .... **??**
     The number of input variables drawn at each node for finding a split .... **??**

$K(x_i, x_j)$ *The kernel of x_i and x_j* .... **??, ??**

L *A loss function* .... **??**
     The number of values of a categorical variable .... **??**

$\mathcal{L}$ *A learning set (X, y)* .... **??**

$\mathcal{L}^m$ *The m-th bootstrap replicate of $\mathcal{L}$* .... **??**

$\mathcal{L}_t$ *The subset of node samples falling into node t* .... **??**

$\sigma^2_{\mathcal{L},\theta_m}(\boldsymbol{x})$	*The prediction variance at $X = \boldsymbol{x}$ of $\varphi_{\mathcal{L},\theta_m}$*	??
t	*A node in a decision tree*	??
t_L	*The left child of node t*	??, ??
t_R	*The right child of node t*	??, ??
θ	*A vector of hyper-parameter values*	??
	A random seed	??
θ^*	*The optimal hyper-parameters*	??
$\widehat{\theta}^*$	*The approximately optimal hyper-parameters*	??
θ_m	*The seed of the m-th model in an ensemble*	??
v	*A discretization threshold in a binary split*	??
v_k	*The k-th value of an ordered variable, when node samples are in sorted order*	??
v'_k	*The mid-cut point between v_k and v_{k+1}*	??
V	*The set $\{X_1,\dots,X_p\}$ of input variables*	??
V^{-j}	*$V \setminus \{X_j\}$*	??
$\mathbb{V}$	*Variance*	
$\boldsymbol{x}$	*A case, sample or input vector $(x_1,\dots,x_p)$*	??
$\boldsymbol{x}_i$	*The i-th input sample in $\mathcal{L}$*	??
x_j	*The value of variable X_j for the sample $\boldsymbol{x}$*	??
$\boldsymbol{X}$	*The $N \times p$ matrix representing the values of all N samples for all p input variables*	??
X_j	*The j-th input variable or feature*	??, ??
X	*The random vector $(X_1,\dots,X_p)$*	??
$\mathcal{X}_j$	*The domain or space of variable X_j*	??
$\mathcal{X}$	*The input space $\mathcal{X}_1 \times \cdots \times \mathcal{X}_p$*	??
$\mathcal{X}_t$	*The subspace $\mathcal{X}_t \subseteq \mathcal{X}$ represented by node t*	??
y	*A value of the output variable Y*	??
$\widehat{y}_t$	*The value labelling node t*	??
$\widehat{y}_t^*$	*The optimal value labelling node t*	??
$\mathbf{y}$	*The output values $(y_1,\dots,y_N)$*	??
Y	*The output or response variable Y*	??
$\mathcal{Y}$	*The domain or space of variable Y*	??

REFERENCES