Block Neural Autoregressive Flow

Nicola De Cao [1 2] **Wilker Aziz** [1] **Ivan Titov** [1 2]

Abstract

Normalising flows (NFs) map two density functions via a differentiable bijection whose Jacobian determinant can be computed efficiently. Recently, as an alternative to hand-crafted bijections, Huang et al. (2018) proposed neural autoregressive flow (NAF) which is a universal approximator for density functions. Their flow is a neural network (NN) whose parameters are predicted by another NN. The latter grows quadratically with the size of the former and thus an efficient technique for parametrization is needed. We propose *block neural autoregressive flow* (B-NAF), a much more compact universal approximator of density functions, where we model a bijection directly using a single feed-forward network. Invertibility is ensured by carefully designing each affine transformation with block matrices that make the flow autoregressive and (strictly) monotone. We compare B-NAF to NAF and other established flows on density estimation and approximate inference for latent variable models. Our proposed flow is competitive across datasets while using orders of magnitude fewer parameters.

1. Introduction

Much of the research in Normalizing Flows (NFs) focuses on designing expressive transformations while satisfying practical constraints. In particular, autoregressive flows (AFs) decompose a joint distribution over $y \in \mathbb{R}^d$ into a product of d univariate conditionals. A transformation $y = f(x)$, that realizes such a decomposition, has a lower triangular Jacobian with the determinant (necessary for application of the change of variables theorem for densities) computable in $O(d)$-time. Kingma et al. (2016) proposed *inverse autoregressive flows* (IAFs), an AF based on transforming each conditional by a composition of a finite number of trivially invertible affine transformations.

[1]University of Amsterdam, The Netherlands [2]University of Edinburgh, United Kingdom. Correspondence to: Nicola De Cao <nicola.decao@uva.nl>.

First workshop on *Invertible Neural Networks and Normalizing Flows* (ICML 2019), Long Beach, CA, USA

Recently, Huang et al. (2018) introduced *neural autoregressive flows* (NAFs). They replace IAF's transformation by a learned bijection realized as a strictly monotonic neural network. Notably, they prove that their method is a universal approximator of real and continuous distributions. Though, whereas parametrizing affine transformations in an IAF requires predicting d pairs of scalars per step of the flow, parametrizing a NAF requires predicting all the parameters of a feed-forward *transformer* network. The *conditioner* network which parametrizes the transformer grows quadratically with the width of the transformer network, thus efficient parametrization techniques are necessary. A NAF is an instance of a hyper-network (Ha et al., 2017).

We propose *block neural autoregressive flows* (B-NAFs) [1], which are AFs based on a novel transformer network which transforms conditionals directly, i.e. without the need for a conditioner network. To do that we exploit the fact that invertibility only requires $\partial y_i / \partial x_i > 0$, and therefore, careful design of a feed-forward network can ensure that the transformation is both autoregressive (with unconstrained manipulation of $x_{<i}$) and strictly monotone (with positive $\partial y_i / \partial x_i$). We do so by organizing the weight matrices of dense layers in block matrices that independently transform subsets of the variables and constrain these blocks to guarantee that $\partial y_i / \partial x_j = 0$ for $j > i$ and that $\partial y_i / \partial x_i > 0$. Our B-NAFs are much more compact than NAFs while remaining universal approximators of density functions.

2. Background

In this section, we provide an introduction to normalizing flows and their applications (§ 2.1). Then, we motivate autoregressive flows in § 2.2 and present the necessary background for our contributions (§ 2.3).

2.1. Normalizing Flow

A (finite) normalizing flow is a bijective function $f : \mathcal{X} \to \mathcal{Y}$ between two continuous random variables $X \in \mathcal{X} \subseteq \mathbb{R}^d$ and $Y \in \mathcal{Y} \subseteq \mathbb{R}^d$ (Tabak et al., 2010). The change of variables theorem expresses a relation between the probability density functions $p_Y(y)$ and $p_X(x)$:

$$p_Y(y) = p_X(x) \left| \det \mathbf{J}_{f(x)} \right|^{-1}, \tag{1}$$

[1] https://github.com/nicola-decao/BNAF

where $y = f(x)$, and $\left|\det \mathbf{J}_{f(x)}\right|$ is the absolute value of the determinant of the Jacobian of f evaluated at x. The Jacobian matrix is defined as $\left(\mathbf{J}_{f(x)}\right)_{ij} = \partial f(x)_i / \partial x_j$. The determinant quantifies how f locally expands or contracts regions of $\mathcal{X}$. Note that a composition of invertible functions remain invertible, thus a composition of NFs is itself a normalizing flow.

2.2. Autoregressive Flows

We can construct $f(x)$ such that its Jacobian is lower triangular, and thus has determinant $\prod_{i=1}^{d} \partial f(x)_i / \partial x_i$, which is computed in time $\mathcal{O}(d)$. Flows based on autoregressive transformations meet precisely this requirement (Kingma et al., 2016; Oliva et al., 2018; Huang et al., 2018). For a multivariate random variable $X = \langle X_1, \ldots, X_d \rangle$ with $d > 1$, we can use the chain rule to express the joint probability of x as product of d univariate conditional densities:

$$p_X(x) = p_{X_1}(x_1) \prod_{i=2}^{d} p_{X_i|X_{<i}}(x_i|x_{<i}) \ . \tag{2}$$

When we then apply a normalizing flow to each univariate density, we have an autoregressive flow. Specifically, we can use a set of functions $f^{(i)}$ that can be decomposed via *conditioners* $c^{(i)}$, and invertible *transformers* $t^{(i)}$:

$$y_i = f_\theta^{(i)}(x_{\leq i}) = t_\theta^{(i)}(x_i, c_\theta^{(i)}(x_{<i})) \ , \tag{3}$$

where each transformer $t^{(i)}$ must be an invertible function with respect to x_i, and each $c^{(i)}$ is an unrestricted function. The resulting flow has a lower triangular Jacobian since each y_i depends only on $x_{\leq i}$. The flow is invertible when the Jacobian is constructed to have a non-zero diagonal.

2.3. Neural Autoregressive Flow

The invertibility of the flow as a whole depends on each $t^{(i)}$ being an invertible function of x_i. For example, Dinh et al. (2014) and Kingma et al. (2016) model each $t^{(i)}$ as an affine transformation whose parameters are predicted by $c^{(i)}$. As argued by Huang et al. (2018), these transformations were constructed to be trivially invertible, but their simplicity leads to a cap on expressiveness of f, thus requiring complex conditioners and a composition of multiple flows. They propose instead to learn a complex bijection using a neural network monotonic in x_i — this only requires constraining $t^{(i)}$ to having non-negative weights and using strictly increasing activation functions. Each conditioner $c^{(i)}$ is an unrestricted function of $x_{<i}$. To parametrize a monotonically increasing transformer network $t^{(i)}$, the outputs of each conditioner $c^{(i)}$ are mapped to the positive real coordinate space by application of an appropriate activation (e.g. $\exp$). The result is a flexible transformation with lower triangular Jacobian whose diagonal elements are positive.[2]

For efficient computation of all pseudo-parameters, as Huang et al. (2018) call the conditioners' outputs, they use a masked autoregressive network (Germain et al., 2015). The Jacobian of a NAF is computed using the chain rule on f_θ through all its hidden layers $\{h^{(\ell)}\}_{\ell=1}^{l}$:

$$\mathbf{J}_{f_\theta(x)} = \left[\nabla_{h^{(l)}} y\right] \left[\nabla_{h^{(l-1)}} h^{(l)}\right] \ldots \left[\nabla_x h^{(1)}\right] \ . \tag{4}$$

Since f_θ is autoregressive, $\mathbf{J}_{f_\theta(x)}$ is lower triangular and only the diagonal needs to be computed, i.e. $\partial y_i / \partial x_i$ for each i. Thus, this operation requires only computing the derivatives of each $t^{(i)}$, reducing the time complexity.

Because the universal approximation theorem for densities holds for NAFs (Huang et al., 2018), increasing the expressiveness of a NAF is only a matter of employing larger transformer networks. However, the conditioner grows quadratically with the size of the transformer network and a combination of restricting the size of the transformer and a technique similar to *conditional weight normalization* (Krueger et al., 2017) is necessary to reduce the number of parameters. We propose to parametrize the transformer network directly without a conditioner network by exploiting the fact that the monotonicity constraint only requires $\partial y_i / \partial x_i > 0$, and therefore, careful design of a single feed-forward network can directly realize a transformation that is both autoregressive (with unconstrained manipulation of $x_{<i}$) and strictly monotone (with positive $\partial y_i / \partial x_i$).

3. Block Neural Autoregressive Flow

In the spirit of NAFs, we model each $f_\theta^{(i)}(x_{\leq i})$ as a neural network with parameters θ, but differently from NAFs, we do not predict θ using a conditioner network, and instead, we learn θ directly. In dense layers of $f_\theta^{(i)}$, we employ affine transformations with strictly positive weights to process x_i. This ensures strict monotonicity and thus invertibility of each $f_\theta^{(i)}$ with respect to x_i. However, we do not impose this constraint on affine transformations of $x_{<i}$. Additionally, we need to always use invertible activation functions to ensure that the whole network is bijective (e.g., $\tanh$ or LeakyReLU). Each $f_\theta^{(i)}$ is then a univariate flow implemented as an arbitrarily wide and deep neural network which can approximate any invertible transformation. Much like other AFs, we can efficiently compute all $f_\theta^{(i)}$ in parallel by employing a single masked autoregressive network (Germain et al., 2015). In the next section, we show how to construct each affine transformation using block matrices. Additionally, we show that our flow is an universal approx-

[2] Note that the expressiveness of a NAF comes at the cost of analytic invertibility, i.e. though each $t^{(i)}$ is bijective, thus invertible in principle, inverting the network itself is non-trivial.

imator of density functions in Appendix F. From now on, we will refer to our novel family of flows as *block neural autoregressive flows* (B-NAFs).

3.1. Affine Transformations With Block Matrices

For each affine transformation of x, we parametrize the bias term freely and we construct the weight matrix $W \in \mathbb{R}^{ad \times bd}$ as a lower triangular *block* matrix for some $a, b \geq 1$. We use $d \times (d+1)/2$ blocks $B_{ij} \in \mathbb{R}^{a \times b}$ for $i \in \{1, .., d\}$ and $1 \leq j \leq i$. We let B_{ij} (with $i > j$) be freely parametrized and constrain diagonal blocks to be strictly positive applying an element-wise function $g : \mathbb{R} \to \mathbb{R}_{>0}$ to each of them. Thus:

$$W = \begin{bmatrix} g(B_{11}) & 0 & \dots & 0 \\ B_{21} & g(B_{22}) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ B_{d1} & B_{d2} & \dots & g(B_{dd}) \end{bmatrix}, \quad (5)$$

where we chose $g(\cdot) = \exp(\cdot)$. Since the flow has to preserve the input dimensionality, the first and the last affine transformations in the network must have $b = 1$ and $a = 1$, respectively. Inside the network, the size of a and b can grow arbitrarily.

The intuition behind the construction of W is that every row of blocks $B_{i1}, .., B_{ii}$ is a set of affine transformations (projections) that are processing $x_{\leq i}$. In particular, blocks in the upper triangular part of W are set to zero to make the flow autoregressive. Since the blocks B_{ii} are mapped to $\mathbb{R}_{>0}$ through g, each transformation in such set is strictly monotonic for x_i and unconstrained on $x_{<i}$.

B-NAF with masked networks In practice, a more convenient parameterization of W consists of using a full matrix $\hat{W} \in \mathbb{R}^{ad \times bd}$ which is then transformed applying two *masking* operations. One mask $M_d \in \{0, 1\}^{ad \times bd}$ selects only elements in the diagonal blocks, and a second one M_o selects only off-diagonal and lower diagonal blocks. Thus, for each layer ℓ we get

$$W^{(\ell)} = g\left(\hat{W}^{(\ell)}\right) \odot M_d^{(\ell)} + \hat{W}^{(\ell)} \odot M_o^{(\ell)}, \quad (6)$$

where $\odot$ is the element-wise product.

Since each weight matrix $W^{(\ell)}$ has some strictly positive and some zero entries, we need to take care of a proper initialization which should take that into account. Indeed, weights are usually initialized to have a zero centred normally distributed output with variance dependent on the output dimensionality (Glorot & Bengio, 2010). Instead of carefully designing a new initialization technique to take care of this, we choose to initialize all blocks with a simple distribution and to apply weight normalization (Salimans & Kingma, 2016) to better cope the effect of such initialization. See Appendix C for more details.

When constructing a stacked flow though a composition of n B-NAF transformations, we add gated residual connections for improving stability such that the composition is $\hat{f}_n \circ \cdots \circ \hat{f}_2 \circ \hat{f}_1$ where $\hat{f}_i(x) = \alpha f_i(x) + (1 - \alpha)x$ and $\alpha \in (0, 1)$ is a trainable scalar parameter.

3.2. Efficient Jacobian computation

Our flow is autoregressive and invertible (see Appendix D for proofs). Autoregressiveness is particularly useful for an efficient computation of $\det \mathbf{J}_{f_\theta(x)}$ since we only need the product of its diagonal elements $\partial y_i / \partial x_i$. Thus, we can avoid computing the other entries. Since the determinant is the result of a product of positive values, we also remove the absolute-value operation resulting in

$$\log |\det \mathbf{J}_{f_\theta(x)}| = \sum_{i=0}^{d} \log \left(\mathbf{J}_{f_\theta(x)}\right)_{ii}. \quad (7)$$

Additionally, when using matrix multiplication, elements in the diagonal blocks (or entries) depend only on diagonal blocks of the same row and column partition. Since all diagonal blocks are positive, we compute them directly in the log-domain to have more numerically stable operations:

$$\begin{aligned} \log \left(\mathbf{J}_{f_\theta(x)}\right)_{ii} = \log g(B_{ii}^{(\ell)}) \star \\ \log \mathbf{J}_{\sigma^{(\ell)}(h_\alpha^{(\ell-1)})} \star \cdots \star \log g(B_{ii}^{(1)}), \end{aligned} \quad (8)$$

where $\star$ denotes the log-matrix multiplication, $\sigma^{(\ell)}$ the strictly increasing non-linear activation function at layer ℓ, and α indicates the set of indices corresponding to diagonal elements that depend on x_i. Notice that, since we chose $g(\cdot) = \exp(\cdot)$ we can remove all redundant operations $\log g(\cdot)$. The log-matrix multiplication can be implemented with a stable *log-sum-exp* operation (see Appendix E).

4. Experiments

In this experiment, we use a B-NAF to perform density estimation on 5 real datasets (4 datasets (Dua & Karra Taniskidou, 2017) from UCI machine learning repository[3] and one dataset of patches of images (Martin et al., 2001). See Appendix H.1 for a description of those datasets. We train using Adam MLE maximizing $\mathbb{E}_{p_{\text{data}}}[\log p_{X|\theta}(x)]$ stacking 5 B-NAF flows (see Appendix H.2 for a complete description of the all other hyper-parameters). We also have two additional experiments on 2D toy task in Appendix G and an experiment on variational inference in Appendix I.

Table 1 shows the results reporting log-likelihood on test set. In all datasets, our B-NAF is better than Real NVP,

[3] http://archive.ics.uci.edu/ml

Model	POWER↑ d=6;N=2,049,280	GAS↑ d=8;N=1,052,065	HEPMASS↑ d=21;N=525,123	MINIBOONE↑ d=43;N=36,488	BSDS300↑ d=63;N=1,300,000
Real NVP (Dinh et al., 2017)	$0.17_{\pm.01}$	$8.33_{\pm.14}$	$-18.71_{\pm.02}$	$-13.55_{\pm.49}$	$153.28_{\pm1.78}$
Glow (Kingma & Dhariwal, 2018)	$0.17_{\pm.01}$	$8.15_{\pm.40}$	$-18.92_{\pm.08}$	$-11.35_{\pm.07}$	$155.07_{\pm.03}$
MADE MoG (Germain et al., 2015)	$0.40_{\pm.01}$	$8.47_{\pm.02}$	$-15.15_{\pm.02}$	$-12.27_{\pm.47}$	$153.71_{\pm.28}$
MAF (Papamakarios et al., 2017)	$0.30_{\pm.01}$	$10.08_{\pm.02}$	$-17.73_{\pm.02}$	$-12.24_{\pm.45}$	$156.36_{\pm.28}$
FFJORD (Grathwohl et al., 2019).	$0.46_{\pm.01}$	$8.59_{\pm.12}$	$-14.92_{\pm.08}$	$-10.43_{\pm.04}$	$157.40_{\pm.19}$
NAF-DDSF (Huang et al., 2018)	$0.62_{\pm.01}$	$11.96_{\pm.33}$	$-15.09_{\pm.40}$	$-8.86_{\pm.15}$	$157.43_{\pm.30}$
TAN (Oliva et al., 2018)	$0.60_{\pm.01}$	$12.06_{\pm.02}$	$-13.78_{\pm.02}$	$-11.01_{\pm.48}$	$159.80_{\pm.07}$
Ours	$0.61_{\pm.01}$	$12.06_{\pm.09}$	$-14.71_{\pm.38}$	$-8.95_{\pm.07}$	$157.36_{\pm.03}$
Parameters ratio NAF (5) / B-NAF	$2.29\times$	$1.30\times$	$17.94\times$	$43.97\times$	$8.24\times$
Parameters ratio NAF (10) / B-NAF	$4.57\times$	$2.60\times$	$35.88\times$	$87.91\times$	$16.48\times$

Table 1. Log-likelihood on the test set (higher is better) for 4 datasets (Dua & Karra Taniskidou, 2017) from UCI machine learning and BSDS300 (Martin et al., 2001). Here d is the dimensionality of datapoints and N the size of the dataset. We report average ($\pm$std) across 3 independently trained models. We also report the ratios between the number of parameters used by NAF-DDSF (with 5 or 10 flows) and our B-NAF. In highly dimensional datasets B-NAF uses orders of magnitude fewer parameters than NAF.

Glow, MADE, and MAF and it performs comparable or better to NAF. B-NAF also outperforms FFJORD in all dataset except on BSDS300 where there is a marginal difference ($< 0.02\%$) between the two methods. On GAS and HEPMASS, B-NAF performs better than most of the other models and even better than NAF. In other datasets, the gap in performance compared to NAF is marginal. We observed that in most cases, the best performing model was the largest one in the grid search ($L = 2$ and $H = 40d$). It is possible that we do a too narrow hyper-parameter search compared to what other methods do. For instance, FFJORD results come from a wider grid search than ours. Grathwohl et al. (2019), Huang et al. (2018), and Oliva et al. (2018) also varied the number of flows when tuning.

We compare NAF and our B-NAF in terms of the number of parameters employed and report the ratio between the two for each dataset. For datasets with low-dimensional datapoints (i.e, GAS and POWER) our model uses a comparable number of parameters to NAF. For high-dimensional datapoints the gap between the parameters used by NAF and B-NAF grows, with B-NAF much smaller, as we intended. For instance, on both HEPMASS and MINIBOONE, our models have marginal differences in performance with NAF while having respectively $\sim 18\times$ and $\sim 40\times$ fewer parameters than NAF. This evidence supports our argument that NAF models are over-parametrized and it is possible to achieve similar performance with an order of magnitude fewer parameters. Besides, when training models on GPUs, being memory efficiency allows to train more models in parallel on the same device. Additionally, in general, a normalizing flow can be a component of a larger architecture that might require more memory than the flow itself.

5. Related work

Current research on NFs focuses on constructing expressive parametrized invertible trasformations with tractable Jacobians. Rezende & Mohamed (2015) were the first to suggest the use of parameterized flows in the context of variational inference proposing two parametric families: the planar and the radial flow. More recently, van den Berg et al. (2018) generalized the use of planar flows showing improvements without increasing the number of transformations, and instead, by making each transformation more expressive.

In the context of density estimation, Germain et al. (2015) proposed MADE, a masked feed-forward network that efficiently computes an autoregressive transformation. MADEs are important building blocks in AFs, such as the inverse autoregressive flows (IAFs) introduced by Kingma et al. (2016). IAFs are based on trivially invertible affine transformations of the preceding coordinates of the input vector. The parameters of each transformation (a location and a positive scale) are predicted in parallel with a MADE, and therefore IAFs have a lower triangular Jacobian whose determinant is fast to evaluate.

Larochelle & Murray (2011) were among the first to employ neural networks for autoregressive density estimation (NADE) for high-dimensional binary data. Non-linear independent components estimation (NICE) explored the direction of learning a map from high-dimensional data to a latent space with a simpler factorized distribution (Dinh et al., 2014). Papamakarios et al. (2017) proposed masked autoregressive flows (MAFs) as a generalization of real non-volume-preserving flows (Real NVP) by Dinh et al. (2017) showing improvements on density estimation.

References

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018.

Daniels, H. and Velikova, M. Monotone and partially monotone neural networks. *IEEE Transactions on Neural Networks*, 21(6):906–917, 2010.

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

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real NVP. *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

Dua, D. and Karra Taniskidou, E. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Germain, M., Gregor, K., Murray, I., and Larochelle, H. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pp. 881–889, 2015.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

Grathwohl, W., Chen, R. T. Q., Bettencourt, J., Sutskever, I., and Duvenaud, D. FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models. *International Conference on Learning Representations*, 2019.

Ha, D., Dai, A., and Le, Q. V. Hypernetworks. *International Conference on Learning Representations*, 2017.

Huang, C.-W., Krueger, D., Lacoste, A., and Courville, A. Neural autoregressive flows. *International Conference on Learning Representations*, 2018.

Hyvärinen, A. and Pajunen, P. Nonlinear independent component analysis: Existence and uniqueness results. *Neural Networks*, 12(3):429–439, 1999.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2014.

Kingma, D. P. and Dhariwal, P. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pp. 10236–10245, 2018.

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

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. Improved variational inference with inverse autoregressive flow. In *Advances in neural information processing systems*, pp. 4743–4751, 2016.

Krueger, D., Huang, C.-W., Islam, R., Turner, R., Lacoste, A., and Courville, A. Bayesian hypernetworks. *arXiv preprint arXiv:1710.04759*, 2017.

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Larochelle, H. and Murray, I. The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 29–37, 2011.

Marlin, B., Swersky, K., Chen, B., and Freitas, N. Inductive principles for restricted boltzmann machine learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 509–516, 2010.

Martin, D., Fowlkes, C., Tal, D., and Malik, J. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference*, volume 2, pp. 416–423. IEEE, 2001.

Oliva, J., Dubey, A., Zaheer, M., Poczos, B., Salakhutdinov, R., Xing, E., and Schneider, J. Transformation autoregressive networks. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3898–3907, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/oliva18a.html.

Papamakarios, G., Pavlakou, T., and Murray, I. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pp. 2338–2347, 2017.

Park, K. I. and Park. *Fundamentals of Probability and Stochastic Processes with Applications to Communications*. Springer, 2018.

Polyak, B. T. and Juditsky, A. B. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

Rezende, D. J. and Mohamed, S. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*, pp. 1530–1538. JMLR. org, 2015.

Salimans, T. and Kingma, D. P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 901–909, 2016.

Tabak, E. G., Vanden-Eijnden, E., et al. Density estimation by dual ascent of the log-likelihood. *Communications in Mathematical Sciences*, 8(1):217–233, 2010.

Tieleman, T. and Hinton, G. Lecture 6.5-rmsprop, coursera: Neural networks for machine learning. *University of Toronto, Technical Report*, 2012.

van den Berg, R., Hasenclever, L., Tomczak, J. M., and Welling, M. Sylvester normalizing flows for variational inference. *34th Conference on Uncertainty in Artificial Intelligence (UAI18)*, 2018.

A. Objective for density estimation

When performing density estimation for a random variable X, we only have access to samples from the unknown target distribution $X \sim p_\star$ (i.e., the unknown data distribution) but we do not have access to $p_\star$ directly (Papamakarios et al., 2017). Using Equation 1, we can use a normalizing flow to transform a complex parametric model $p_{X|\theta}$ of the target distribution into a simpler distribution p_Y (i.e., a uniform or a Normal distribution), which can be easily evaluated. In this case, we will learn the parameters θ of the model by minimizing $\mathbb{KL}(p_\star \| p_{X|\theta})$:

$$\theta^* = \min_\theta \ \mathbb{KL}(p_\star \| p_{X|\theta}) \tag{9}$$

$$= \min_\theta \ \mathbb{E}_{p_\star(x)} \left[\log \frac{p_\star(x)}{p_{X|\theta}(x)} \right] \tag{10}$$

$$= \min_\theta \ \underbrace{\mathbb{E}_{p_\star(x)}[\log p_\star(x)]}_{\text{=constant}} - \mathbb{E}_{p_\star(x)}[\log p_{X|\theta}(x)] \tag{11}$$

$$= \max_\theta \ \mathbb{E}_{p_\star(x)} \left[\log p_Y(f_\theta(x)) + \log \left| \det \mathbf{J}_{f_\theta(x)} \right| \right] . \tag{12}$$

where $p_{X|\theta}(x) = p_Y(y) \left| \det \mathbf{J}_{f_\theta(x)} \right|$ and $y = f_\theta(t)$. Notice that minimizing the KL is equivalent of doing maximum likelihood estimation (MLE).

B. Objective for density matching

We can learn how to sample from a complex target distribution $p_\star$ (or, more generally, an energy function) for which we have access to its analytical form but we do not have an available sampling procedure. Using Equation 1, we can use a normalizing flow to transform samples from a simple distribution p_X, which we can easily evaluate and sample from, to a complex one (the target). In this case, we estimate θ by minimizing $\mathbb{KL}(p_{Y|\theta} \| p_\star)$:

$$\theta^* = \min_\theta \ \mathbb{KL}(p_{Y|\theta} \| p_\star) \tag{13}$$

$$= \min_\theta \ \mathbb{E}_{p_{Y|\theta}(y)} \left[\log \frac{p_{Y|\theta}(y)}{p_\star(y)} \right] \tag{14}$$

$$= \min_\theta \ \mathbb{E}_{p_{Y|\theta}(y)} [\log p_{Y|\theta}(y) - \log p_\star(y)] \tag{15}$$

$$= \min_\theta \ \mathbb{E}_{p_X(x)} \big[\log p_X(x) - \log \left| \det \mathbf{J}_{f_\theta(x)} \right| - \log p_\star(f_\theta(x)) \big] . \tag{16}$$

where $p_{Y|\theta}(y) = p_X(x) \left| \det \mathbf{J}_{f_\theta(x)} \right|^{-1}$ and $y = f_\theta(x)$. Notice that in general, with normalizing flows, it is possible to learn a flexible distribution from which we can sample and evaluate the density of its samples. These two proprieties are particularly useful in the context of variational inference (Rezende & Mohamed, 2015).

C. Weight initialization and normalization

Since the weight matrix W has some strictly positive and some zero entries, we need to take care of a proper initialization. Indeed, it is well known that principled parameters initialization benefits not only training but also the generalization of neural networks (Glorot & Bengio, 2010). For instance, Xavier initialization is commonly used and it takes into account the size of the input and output spaces in the affine transformations. However, since we have some zero entries, we cannot benefit from it. We choose instead to initialize all blocks with a simple distribution and to apply weight normalization (Salimans & Kingma, 2016) to better regulate the effect of such initialization. Weight normalization decomposes each row $w \in \mathbb{R}^{b \cdot d}$ of W in terms of the new parameters using $w = \exp(s) \cdot v / \|v\|$ where v has the same dimensionality of w and s is a scalar. We initialize v with a simple Normal distribution of zero mean and unit variance and $s = \log(u)$ with $u \sim \mathcal{U}(0, 1)$. Such reparametrization disentangles the direction and magnitude of w and it is known to improve and speed up optimization.

D. Autoregressiveness and Invertibility

In this section, we show that our flow $f_\theta : \mathbb{R}^d \to \mathbb{R}^d$ meets the following requirements: i) its Jacobian $\mathbf{J}_{f_\theta(x)}$ is lower triangular (needed for efficiency in computing its determinant), and ii) the diagonal entries of such Jacobian are positive (to ensure that f_θ is a bijection).

Proposition 1. *The final Jacobian $\mathbf{J}_{f_\theta(x)}$ of such transformation is lower triangular.*

Proof sketch. When applying the chain rule (Equation 4), the Jacobian of each affine transformation is W (Equation (6), a lower triangular block matrix), whereas the Jacobian of each element-wise activation function is a diagonal matrix. A matrix multiplication between a lower triangular block matrix and a diagonal matrix yields a lower triangular block matrix, and a multiplication between two lower triangular block matrices results in a lower triangular block matrix. Therefore, after multiplying all matrices in the chain, the overall Jacobian is lower triangular. □

Proposition 2. *When using strictly increasing activation functions (e.g.,* $\tanh$ *or LeakyReLU), the diagonal entries of $\mathbf{J}_{f_\theta(x)}$ are strictly positive.*

Proof sketch. When applying the chain rule (Equation 4), the Jacobian of each affine transformation has strictly positive values in its diagonal blocks where the Jacobian of each element-wise activation function is a diagonal matrix with strictly positive elements. When using matrix multiplication between two lower triangular block matrices (or one diagonal and one lower triangular block matrix) $C = AB$ the resulting blocks on the diagonal of C are the result of a multiplication between only diagonal blocks of A, B. Indeed, such resulting blocks depend only on

blocks of the same row and column partition. Using the notation of Equation 5, the resulting diagonal blocks of C are $B_{ii}^{(C)} = g(B_{ii}^{(A)})g(B_{ii}^{(B)})$. Therefore, they are always positive. Eventually, using the chain rule, the final Jacobian is a lower triangular matrix with strictly positive elements in its diagonal. □

E. Log-matrix multiplication

The log-matrix multiplication $C = A \star B$ of two matrices $A \in \mathbb{R}^{m \times n} = \log \hat{A}$ and $B \in \mathbb{R}^{n \times p} = \log \hat{B}$ can be implemented with a stable *log-sum-exp* operation since

$$C_{ij} = \log \sum_{k=1}^{n} \exp\left(\hat{A}_{ik} + \hat{B}_{kj}\right) . \tag{17}$$

F. Universal density approximator

In this section, we expose an intuitive proof sketch that our block neural autoregressive flow can approximate any real continuous probability density function (PDF).

Given a multivariate real and continuous random variable $X = \langle X_1, \dots, X_d \rangle$, its joint distribution can be factorized into a set of univariate conditional distributions (as in Equation 2), using an arbitrary ordering of the variables, and we can define a set of univariate conditional cumulative distribution functions (CDFs) $Y_i = F_{X_i|X_{<i}}(x_i|x_{<i}) = \mathbb{P}[X_i \leq x_i|X_{<i} = x_{<i}]$. According to Hyvärinen & Pajunen (1999), such decomposition exists and each individual Y_i is independent as well as uniformly distributed in $[0, 1]$. Therefore, we can see F_X as a particular *normalizing flow* that maps $X \in \mathbb{R}^n$ to $Y \in [0, 1]^n$ where the distribution p_Y is uniform in the hyper-cube $[0, 1]^n$. Note that F_X is an autoregressive function and its Jacobian has a positive diagonal since $\partial y_i / \partial x_i = p_{X_i|X_{<i}}(x_i|x_{<i})$.

If each univariate flow $f_\theta^{(i)}$ (see Equation 3) can approximate any invertible univariate conditional CDF, then f_θ can approximate any PDF (Huang et al., 2018). Note that in general, a CDF $F_{X_i|X_{<i}}$ is non-decreasing, thus not necessary invertible (Park & Park, 2018). Using B-NAF, each CDF is approximated with an arbitrarily large neural network and the output can be eventually mapped to $(0, 1)$ with a sigmoidal function. Recalling that we only use positive weights for processing x_i, a neural network with non-negative weights is an universal approximator of monotonic functions (Daniels & Velikova, 2010). We use *strictly* positive weights to approximate a *strictly* monotonic function for x_i and we use arbitrary weights for $x_{<i}$ (as there is no monotonicity constraint for them). Therefore, B-NAF can approximate any invertible CDF, and thus its corresponding PDF.

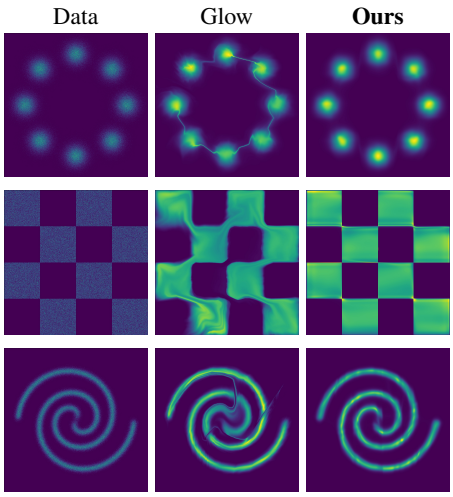

Figure 1. Comparison between Glow and B-NAF on density estimation for 2D toy data.

G. Toy experiments

G.1. Density estimation on toy 2D data

In this experiment, we use our B-NAF to perform density estimation on 2-dimensional data as this helps us visualizing the model capabilities to learn. We use the same toy data as Grathwohl et al. (2019) comparing the results with Glow (Kingma & Dhariwal, 2018), as they do. Given samples from a dataset with empirical distribution p_{data}, we parametrize a density $p_{X|\theta}$ with a normalizing flow $p_{X|\theta}(x) = p_Y(f_\theta(x))|\det \mathbf{J}_{f_\theta(x)}|$ using B-NAF with p_Y a standard Normal distribution. We train for 20k iterations a single flow of B-NAF with 3 hidden layers of 100 units each using maximum likelihood estimation (i.e., maximizing $\mathbb{E}_{p_{\text{data}}}[\log p_{X|\theta}(x)]$, see Appendix A for more details and derivation of the objective). We used Adam optimizer (Kingma & Ba, 2014) with an initial learning rate of $\alpha = 10^{-1}$ (and decay of 0.5 with patience of 2k steps), default β_1, β_2, and a batch size of 200. We took figures of Glow from (Grathwohl et al., 2019) who trained such models with 100 layers.

Results The learned distributions of both Glow and our method can be seen in Figure 1. Glow is capable of learning a multi-modal distribution, but it has issues assigning the correct density in areas of low probability between disconnected regions. Our model is instead able to perfectly capture both multi-modality and discontinuities.

G.2. Density matching on toy 2D data

In this experiment, we use B-NAF to perform density matching on 2-dimensional target energy functions to visualize the model capabilities of matching them. We use the same

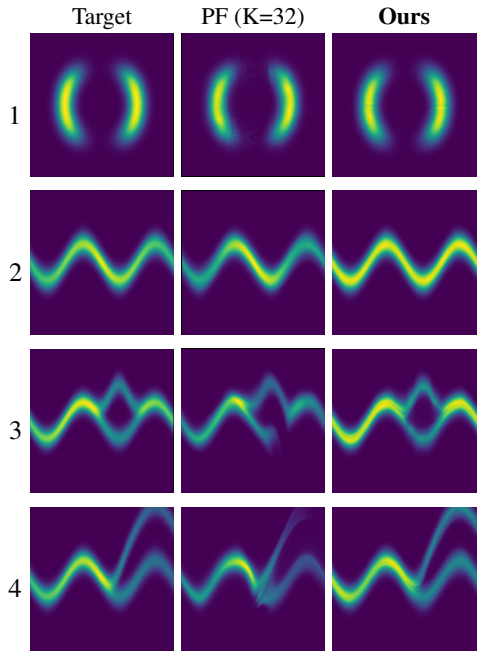

Figure 2. Comparison between planar flow (PF) and B-NAF on four 2D energy functions from Table 1 of (Rezende & Mohamed, 2015).

energy functions described by Rezende & Mohamed (2015) comparing the results with them (using planar flows). For this task, we train a parameterized flow minimizing the KL divergence between the learned $q_{Y|\theta}$ and the given target p_Y. We used a single flow using a B-NAF with 2 hidden layers of 100 units each. We train by minimizing $\mathbb{KL}(q_{Y|\theta}\|p_Y)$ (see Appendix B for a detailed derivation) using Monte Carlo sampling. We optimized using Adam for 20k iterations with an initial learning rate of $\alpha = 10^{-2}$ (and decay of 0.5 with patience of 2k steps), default β_1, β_2, and a batch size of 200. Planar flow figures were taken from Chen et al. (2018). Note that planar flows were trained for 500k iterations using RMSProp (Tieleman & Hinton, 2012).

Results Figure 2 shows that our model perfectly matches all target distributions. Indeed, on functions 3 and 4 it looks like B-NAF can better learn the density in certain areas. The model capacity of planar normalizing flows is determined by their depth (K) and Rezende & Mohamed (2015) had to stack 32 flows to match the energy function reasonably well. Deeper networks are harder to optimize, and our flow matches all the targets using a neural network with only 2 hidden layers.

H. Real data density estimation

H.1. Datasets

Following Papamakarios et al. (2017), we perform unconditional density estimation on four datasets (Dua & Karra Taniskidou, 2017) from UCI machine learning repository[4] as well as one dataset of patches of images (Martin et al., 2001): POWER containing electric power consumption in a household over a period of 4 years, GAS containing logs of 8 chemical sensors exposed to a mixture of gases, HEPMASS, a dataset from a Monte Carlo simulation for high energy physics experiments, MINIBOONE that contains examples of electron neutrino and muon neutrino, and BSDS300 which is obtained by extracting random patches from the homonym datasets of natural images.

H.2. Hyper-parameters

For our B-NAF, we stacked 5 flows and we employed a small grid search on the number of layers and the size of hidden units per flow ($L \in \{1, 2\}$ and $H \in \{10d, 20d, 40d\}$, respectively, where d is the input size of datapoints which is different for each dataset). When stacking B-NAF flows, the elements of each output vector are permuted so that a different set of elements is considered at each flow. This technique is not novel and it is also used by Dinh et al. (2017); Papamakarios et al. (2017); Kingma et al. (2016). We trained using Adam with Polyak averaging (with $\phi = 0.998$) as in NAF (Polyak & Juditsky, 1992). We also applied an exponentially decaying learning rate schedule (from $\alpha = 10^{-2}$ with rate $\lambda = 0.5$) based on no-improvement with patience of 20 epochs. We trained until convergence (but maximum 1k epochs), stopping after 100 epochs without improvement on validation set.

I. Variational Auto-Encoders

An interesting application of our framework is modelling more flexible posterior distributions in a variational auto-encoder (VAE) setting (Kingma & Welling, 2013). In this setting, we assume that an observation x (i.e., the data) is drawn from the marginal of a deep latent model, i.e. $X \sim p_{X|\theta}$, where $p_{X|\theta}(x) = \int p_Z(z) p_{X|Z,\theta}(x|z)\mathrm{d}z$ where $Z \sim \mathcal{N}(0, I)$ is unobserved. The goal is performing maximum likelihood estimation of the marginal. Since Z is not observed, maximizing the objective would require marginalization over the latent variables, which is generally intractable. Using variational inference (Jordan et al., 1999), we can maximize a lower bound on log-likelihood:

$$\log p_{X|\theta}(x) \geq \mathbb{E}_{q_{Z|X,\phi}(z)}\left[\log \frac{p_{XZ|\theta}(x,z)}{q_{Z|X,\phi}(z|x)}\right], \quad (18)$$

[4] http://archive.ics.uci.edu/ml

Model	MNIST		Freyfaces		Omniglot		Caltech 101	
	-ELBO↓	NLL↓	-ELBO↓	NLL↓	-ELBO↓	NLL↓	-ELBO↓	NLL↓
VAE	$86.55_{\pm.06}$	$82.14_{\pm.07}$	$4.53_{\pm.02}$	$4.40_{\pm.03}$	$104.28_{\pm.39}$	$97.25_{\pm.23}$	$110.80_{\pm.46}$	$99.62_{\pm.74}$
Planar	$86.06_{\pm.31}$	$81.91_{\pm.22}$	$4.40_{\pm.06}$	$4.31_{\pm.06}$	$102.65_{\pm.42}$	$96.04_{\pm.28}$	$109.66_{\pm.42}$	$98.53_{\pm.68}$
IAF	$84.20_{\pm.17}$	$80.79_{\pm.12}$	$4.47_{\pm.05}$	$4.38_{\pm.04}$	$102.41_{\pm.04}$	$96.08_{\pm.16}$	$111.58_{\pm.38}$	$99.92_{\pm.30}$
Sylvester	$83.32_{\pm.06}$	$80.22_{\pm.03}$	$4.45_{\pm.04}$	$4.35_{\pm.04}$	$99.00_{\pm.04}$	$93.77_{\pm.03}$	$104.62_{\pm.29}$	$93.82_{\pm.62}$
Ours	$83.59_{\pm.15}$	$80.71_{\pm.09}$	$4.42_{\pm.05}$	$4.33_{\pm.04}$	$100.08_{\pm.07}$	$94.83_{\pm.10}$	$105.42_{\pm.49}$	$94.91_{\pm.51}$

Table 2. Negative log-likelihood (NLL) and negative evidence lower bound (-ELBO) for static MNIST, Freyfaces, Omniglot and Caltech 101 Silhouettes datasets. For the Freyfaces dataset the results are reported in bits per dim. For the other datasets the results are reported in nats. For all datasets we report the mean and the standard deviations over 3 runs with different random initializations.

where $p_{X|Z,\theta}$ and $q_{Z|X,\phi}$ are parametrized via neural networks with learnable parameters θ and ϕ (Kingma & Welling, 2013), in particular, $q_{Z|X,\phi}$ is an approximation to the intractable posterior $p_{Z|X,\theta}$. This bound is called the evidence lower bound (ELBO), and maximizing the ELBO is equivalent to minimizing $\mathbb{KL}(q_{Z|X,\phi}\|p_{Z|X,\theta})$. The more expressive the approximating family is, the more likely we are to obtain a tight bound. Recent literature approaches tighter bounds by approximating the posterior with normalizing flows. Also note that NFs reparametrize $q_{Z|X,\phi}(z|x) = q_Y(f_\phi(z;x))\left|\det \mathbf{J}_{f_\phi(z;x)}\right|$ via a simpler fixed base distribution, e.g. a standard Gaussian, and therefore we can follow stochastic gradient estimates of the ELBO with respect to both sets of parameters. In this experiment, we use our flow for posterior approximation showing that B-NAF compares with recently proposed NFs for variational inference. We reproduce experiments by van den Berg et al. (2018) (Sylvester flows or SNF) while replacing their flow with ours. We keep the encoder and decoder networks exactly the same to fairly compare with all models trained with such procedure. We compare our B-NAF to their flows on the same 4 datasets as well as to a normal VAE (Kingma & Welling, 2013), planar flows (Rezende & Mohamed, 2015), and IAFs (Kingma et al., 2016).[5]

In this experiment, the input dimensionality of the flow is fixed to $d = 64$. We employed a small grid search on the MNIST dataset on the number of flows $K \in \{4, 8\}$, and on thee size of hidden units per flow $H \in \{2d, 4d, 8d\}$ while keeping the number of layers fixed at $L = 1$. The elements of each output vector are permuted after each B-NAF flow (as we do in § 4). We keep the best hyper-parameters of this search for the other datasets. We train using Adamax with $\alpha = 5 \cdot 10^{-4}$. We point to Appendix A of van den Berg et al. (2018) for details on the network architectures for the encoder and decoder.

[5] Although also Huang et al. (2018) proposed an experiment with VAEs for NAF, they used only one dataset (MNIST) employed a different encoder/decoder architecture than van den Berg et al. (2018). Therefore, results are not comparable.

Datasets Following van den Berg et al. (2018) we carried our experiments on 4 datasets: statically binarized MNIST (Larochelle & Murray, 2011), Freyfaces[6], Omniglot (Lake et al., 2015) and Caltech 101 Silhouettes (Marlin et al., 2010). All those datasets consist of black and white images of different sizes.

Amortizing flow parameters When using NFs in an amortized inference setting, the parameters of each flow are not learned directly but predicted with another function from each datapoint (Rezende & Mohamed, 2015). In our case, we do not amortize all parameters of B-NAF since that would require very large predictors and we want to keep our flow memory efficient. Alternatively, every affine matrix $W \in \mathbb{R}^{n\times m}$ is shared among all datapoints. Then, for each affine transformation, we achieve a degree of amortization by predicting 3 vectors, the bias $b \in \mathbb{R}^n$ and 2 vectors $v_1 \in \mathbb{R}^n$ and $v_2 \in \mathbb{R}^m$ that we multiply row- and column-wise respectively to W.

Results Table 2 shows the results of these experiments. From the grid search, it turned out that the best B-NAF model has $K = 8$ (flows) and $H = 4d$ (hidden units). Note that the best models reported by van den Berg et al. (2018) used 16 flows. Our model is quite flexible without being as deep as other models. Results show that B-NAF is better than normal VAE, planar flows, and IAFs in all four datasets. Although B-NAF performs slightly worse than Sylvester flows, van den Berg et al. (2018) applied a full amortization for the parameters of the flow, while we do not. They proposed two alternative parametrizations to construct Sylvester flows: orthogonal SNF and Houseolder SNF. For each datapoint, SNF has to predict from 50.7k to 76.8k values (depending on the parametrization) to fully amortize parameters of the flow, while we use only 7.7k (i.e., $6.64\times$ to $10.0\times$ fewer). Notably, recalling that these are not trainable parameters, we use $6.16\times$ (orthogonal SNF) and $9.35\times$ (Householder SNF) fewer trainable parameters

[6] http://www.cs.nyu.edu/~roweis/data/frey_rawface.mat

as well. Besides, we also use $14.45\times$ fewer parameters than IAF. This shows that IAF and SNF are over-parametrized too, and it is possible to achieve similar performance in the context of variational inference with an order of magnitude fewer parameters.