Masked Autoregressive Flow for Density Estimation

George Papamakarios
University of Edinburgh
g.papamakarios@ed.ac.uk

Theo Pavlakou
University of Edinburgh
theo.pavlakou@ed.ac.uk

Iain Murray
University of Edinburgh
i.murray@ed.ac.uk

Abstract

Autoregressive models are among the best performing neural density estimators. We describe an approach for increasing the flexibility of an autoregressive model, based on modelling the random numbers that the model uses internally when generating data. By constructing a stack of autoregressive models, each modelling the random numbers of the next model in the stack, we obtain a type of normalizing flow suitable for density estimation, which we call Masked Autoregressive Flow. This type of flow is closely related to Inverse Autoregressive Flow and is a generalization of Real NVP. Masked Autoregressive Flow achieves state-of-the-art performance in a range of general-purpose density estimation tasks.

1 Introduction

The joint density $p(\mathbf{x})$ of a set of variables $\mathbf{x}$ is a central object of interest in machine learning. Being able to access and manipulate $p(\mathbf{x})$ enables a wide range of tasks to be performed, such as inference, prediction, data completion and data generation. As such, the problem of estimating $p(\mathbf{x})$ from a set of examples $\{\mathbf{x}_n\}$ is at the core of probabilistic unsupervised learning and generative modelling.

In recent years, using neural networks for density estimation has been particularly successful. Combining the flexibility and learning capacity of neural networks with prior knowledge about the structure of data to be modelled has led to impressive results in modelling natural images [6, 31, 34, 41, 42] and audio data [38, 40]. State-of-the-art neural density estimators have also been used for likelihood-free inference from simulated data [24, 26], variational inference [16, 27], and as surrogates for maximum entropy models [22].

Neural density estimators differ from other approaches to generative modelling—such as variational autoencoders [15, 28] and generative adversarial networks [10]—in that they readily provide exact density evaluations. As such, they are more suitable in applications where the focus is on explicitly evaluating densities, rather than generating synthetic data. For instance, density estimators can learn suitable priors for data from large unlabelled datasets, for use in standard Bayesian inference [43]. In simulation-based likelihood-free inference, conditional density estimators can learn models for the likelihood [7] or the posterior [26] from simulated data. Density estimators can learn effective proposals for importance sampling [25] or sequential Monte Carlo [11, 24]; such proposals can be used in probabilistic programming environments to speed up inference [18, 19]. Finally, conditional density estimators can be used as flexible inference networks for amortized variational inference and as part of variational autoencoders [15, 28].

A challenge in neural density estimation is to construct models that are flexible enough to represent complex densities, but have tractable density functions and learning algorithms. There are mainly two families of neural density estimators that are both flexible and tractable: *autoregressive models* [39] and *normalizing flows* [27]. Autoregressive models decompose the joint density as a product of conditionals, and model each conditional in turn. Normalizing flows transform a base density (e.g. a standard Gaussian) into the target density by an invertible transformation with tractable Jacobian.

Our starting point is the realization (as pointed out by Kingma et al. [16]) that autoregressive models, when used to generate data, correspond to a differentiable transformation of an external source of

randomness (typically obtained by random number generators). This transformation has a tractable Jacobian by design, and for certain autoregressive models it is also invertible, hence it precisely corresponds to a normalizing flow. Viewing an autoregressive model as a normalizing flow opens the possibility of increasing its flexibility by stacking multiple models of the same type, by having each model provide the source of randomness for the next model in the stack. The resulting stack of models is a normalizing flow that is more flexible than the original model, and that remains tractable.

In this paper we present *Masked Autoregressive Flow* (MAF), which is a particular implementation of the above normalizing flow that uses the Masked Autoencoder for Distribution Estimation (MADE) [9] as a building block. The use of MADE enables density evaluations without the sequential loop that is typical of autoregressive models, and thus makes MAF fast to evaluate and train on parallel computing architectures such as Graphics Processing Units (GPUs). We show a close theoretical connection between MAF and Inverse Autoregressive Flow (IAF) [16], which has been designed for variational inference instead of density estimation, and show that both correspond to generalizations of the successful Real NVP [6]. We experimentally evaluate MAF on a wide range of datasets, and we demonstrate that MAF outperforms RealNVP and achieves state-of-the-art performance on a variety of general-purpose density estimation tasks.

2 Background

2.1 Autoregressive density estimation

Using the chain rule of probability, any joint density $p(\mathbf{x})$ can be decomposed into a product of one-dimensional conditionals as $p(\mathbf{x}) = \prod_i p(x_i \mid \mathbf{x}_{1:i-1})$. Autoregressive density estimators [39] model each conditional $p(x_i \mid \mathbf{x}_{1:i-1})$ as a parametric density, whose parameters are a function of a hidden state $\mathbf{h}_i$. In recurrent architectures, $\mathbf{h}_i$ is a function of the previous hidden state $\mathbf{h}_{i-1}$ and the ith input variable x_i. The Real-valued Neural Autoregressive Density Estimator (RNADE) [36] uses mixtures of Gaussian or Laplace densities for modelling the conditionals, and a simple linear rule for updating the hidden state. More flexible approaches for updating the hidden state are based on Long Short-Term Memory recurrent neural networks [34, 42].

A drawback of autoregressive models is that they are sensitive to the order of the variables. For example, the order of the variables matters when learning the density of Figure 1a if we assume a model with Gaussian conditionals. As Figure 1b shows, a model with order (x_1, x_2) cannot learn this density, even though the same model with order (x_2, x_1) can represent it perfectly. In practice is it hard to know which of the factorially many orders is the most suitable for the task at hand. Autoregressive models that are trained to work with an order chosen at random have been developed, and the predictions from different orders can then be combined in an ensemble [9, 37]. Our approach (Section 3) can use a different order in each layer, and using random orders would also be possible.

Straightforward recurrent autoregressive models would update a hidden state sequentially for every variable, requiring D sequential computations to compute the probability $p(\mathbf{x})$ of a D-dimensional vector, which is not well-suited for computation on parallel architectures such as GPUs. One way to enable parallel computation is to start with a fully-connected model with D inputs and D outputs, and drop out connections in order to ensure that output i will only be connected to inputs $1, 2, \ldots, i-1$. Output i can then be interpreted as computing the parameters of the ith conditional $p(x_i \mid \mathbf{x}_{1:i-1})$. By construction, the resulting model will satisfy the autoregressive property, and at the same time it will be able to calculate $p(\mathbf{x})$ efficiently on a GPU. An example of this approach is the Masked Autoencoder for Distribution Estimation (MADE) [9], which drops out connections by multiplying the weight matrices of a fully-connected autoencoder with binary masks. Other mechanisms for dropping out connections include masked convolutions [42] and causal convolutions [40].

2.2 Normalizing flows

A normalizing flow [27] represents $p(\mathbf{x})$ as an invertible differentiable transformation f of a base density $\pi_u(\mathbf{u})$. That is, $\mathbf{x} = f(\mathbf{u})$ where $\mathbf{u} \sim \pi_u(\mathbf{u})$. The base density $\pi_u(\mathbf{u})$ is chosen such that it can be easily evaluated for any input $\mathbf{u}$ (a common choice for $\pi_u(\mathbf{u})$ is a standard Gaussian). Under the invertibility assumption for f, the density $p(\mathbf{x})$ can be calculated as

$$p(\mathbf{x}) = \pi_u\big(f^{-1}(\mathbf{x})\big) \left| \det\left(\frac{\partial f^{-1}}{\partial \mathbf{x}} \right) \right|. \tag{1}$$
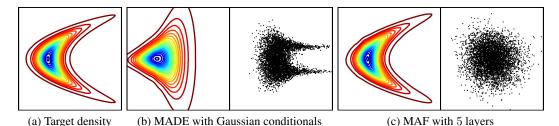
| (a) Target density | (b) MADE with Gaussian conditionals | (c) MAF with 5 layers |

Figure 1: **(a)** The density to be learnt, defined as $p(x_1, x_2) = \mathcal{N}(x_2 \,|\, 0, 4)\mathcal{N}\left(x_1 \,|\, \frac{1}{4}x_2^2, 1\right)$. **(b)** The density learnt by a MADE with order (x_1, x_2) and Gaussian conditionals. Scatter plot shows the train data transformed into random numbers $\mathbf{u}$; the non-Gaussian distribution indicates that the model is a poor fit. **(c)** Learnt density and transformed train data of a 5 layer MAF with the same order (x_1, x_2).

In order for Equation (1) to be tractable, the transformation f must be constructed such that (a) it is easy to invert, and (b) the determinant of its Jacobian is easy to compute. An important point is that if transformations f_1 and f_2 have the above properties, then their composition $f_1 \circ f_2$ also has these properties. In other words, the transformation f can be made deeper by composing multiple instances of it, and the result will still be a valid normalizing flow.

There have been various approaches in developing normalizing flows. An early example is Gaussianization [4], which is based on successive application of independent component analysis. Enforcing invertibility with nonsingular weight matrices has been proposed [3, 29], however in such approaches calculating the determinant of the Jacobian scales cubically with data dimensionality in general. Planar/radial flows [27] and Inverse Autoregressive Flow (IAF) [16] are models whose Jacobian is tractable by design. However, they were developed primarily for variational inference and are not well-suited for density estimation, as they can only efficiently calculate the density of their own samples and not of externally provided datapoints. The Non-linear Independent Components Estimator (NICE) [5] and its successor Real NVP [6] have a tractable Jacobian and are also suitable for density estimation. IAF, NICE and Real NVP are discussed in more detail in Section 3.

3 Masked Autoregressive Flow

3.1 Autoregressive models as normalizing flows

Consider an autoregressive model whose conditionals are parameterized as single Gaussians. That is, the ith conditional is given by

$$p(x_i \,|\, \mathbf{x}_{1:i-1}) = \mathcal{N}\left(x_i \,|\, \mu_i, (\exp \alpha_i)^2\right) \quad \text{where} \quad \mu_i = f_{\mu_i}(\mathbf{x}_{1:i-1}) \ \text{and} \ \alpha_i = f_{\alpha_i}(\mathbf{x}_{1:i-1}). \quad (2)$$

In the above, f_{μ_i} and f_{α_i} are unconstrained scalar functions that compute the mean and log standard deviation of the ith conditional given all previous variables. We can generate data from the above model using the following recursion:

$$x_i = u_i \exp \alpha_i + \mu_i \quad \text{where} \quad \mu_i = f_{\mu_i}(\mathbf{x}_{1:i-1}), \ \alpha_i = f_{\alpha_i}(\mathbf{x}_{1:i-1}) \ \text{and} \ u_i \sim \mathcal{N}(0, 1). \quad (3)$$

In the above, $\mathbf{u} = (u_1, u_2, \ldots, u_I)$ is the vector of random numbers the model uses internally to generate data, typically by making calls to a random number generator often called `randn()`.

Equation (3) provides an alternative characterization of the autoregressive model as a transformation f from the space of random numbers $\mathbf{u}$ to the space of data $\mathbf{x}$. That is, we can express the model as $\mathbf{x} = f(\mathbf{u})$ where $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. By construction, f is easily invertible. Given a datapoint $\mathbf{x}$, the random numbers $\mathbf{u}$ that were used to generate it are obtained by the following recursion:

$$u_i = (x_i - \mu_i) \exp(-\alpha_i) \quad \text{where} \quad \mu_i = f_{\mu_i}(\mathbf{x}_{1:i-1}) \ \text{and} \ \alpha_i = f_{\alpha_i}(\mathbf{x}_{1:i-1}). \quad (4)$$

Due to the autoregressive structure, the Jacobian of f^{-1} is triangular by design, hence its absolute determinant can be easily obtained as follows:

$$\left| \det\left(\frac{\partial f^{-1}}{\partial \mathbf{x}}\right) \right| = \exp\left(-\sum_i \alpha_i\right) \quad \text{where} \quad \alpha_i = f_{\alpha_i}(\mathbf{x}_{1:i-1}). \quad (5)$$

It follows that the autoregressive model can be equivalently interpreted as a normalizing flow, whose density $p(\mathbf{x})$ can be obtained by substituting Equations (4) and (5) into Equation (1). This observation was first pointed out by Kingma et al. [16].

A useful diagnostic for assessing whether an autoregressive model of the above type fits the target density well is to transform the train data $\{\mathbf{x}_n\}$ into corresponding random numbers $\{\mathbf{u}_n\}$ using Equation (4), and assess whether the u_i's come from independent standard normals. If the u_i's do not seem to come from independent standard normals, this is evidence that the model is a bad fit. For instance, Figure 1b shows that the scatter plot of the random numbers associated with the train data can look significantly non-Gaussian if the model fits the target density poorly.

Here we interpret autoregressive models as a flow, and improve the model fit by stacking multiple instances of the model into a deeper flow. Given autoregressive models $M_1, M_2, \ldots, M_K$, we model the density of the random numbers $\mathbf{u}_1$ of M_1 with M_2, model the random numbers $\mathbf{u}_2$ of M_2 with M_3 and so on, finally modelling the random numbers $\mathbf{u}_K$ of M_K with a standard Gaussian. This stacking adds flexibility: for example, Figure 1c demonstrates that a flow of 5 autoregressive models is able to learn multimodal conditionals, even though each model has unimodal conditionals. Stacking has previously been used in a similar way to improve model fit of deep belief nets [12] and deep mixtures of factor analyzers [32].

We choose to implement the set of functions $\{f_{\mu_i}, f_{\alpha_i}\}$ with *masking*, following the approach used by MADE [9]. MADE is a feedforward network that takes $\mathbf{x}$ as input and outputs μ_i and α_i for all i with a single forward pass. The autoregressive property is enforced by multiplying the weight matrices of MADE with suitably constructed binary masks. In other words, we use MADE with Gaussian conditionals as the building layer of our flow. The benefit of using masking is that it enables transforming from data $\mathbf{x}$ to random numbers $\mathbf{u}$ and thus calculating $p(\mathbf{x})$ in one forward pass through the flow, thus eliminating the need for sequential recursion as in Equation (4). We call this implementation of stacking MADEs into a flow *Masked Autoregressive Flow* (MAF).

3.2 Relationship with Inverse Autoregressive Flow

Like MAF, Inverse Autoregressive Flow (IAF) [16] is a normalizing flow which uses MADE as its component layer. Each layer of IAF is defined by the following recursion:

$$x_i = u_i \exp \alpha_i + \mu_i \quad \text{where} \quad \mu_i = f_{\mu_i}(\mathbf{u}_{1:i-1}) \quad \text{and} \quad \alpha_i = f_{\alpha_i}(\mathbf{u}_{1:i-1}). \tag{6}$$

Similarly to MAF, functions $\{f_{\mu_i}, f_{\alpha_i}\}$ are computed using a MADE with Gaussian conditionals. The difference is architectural: in MAF μ_i and α_i are directly computed from previous *data variables* $\mathbf{x}_{1:i-1}$, whereas in IAF μ_i and α_i are directly computed from previous *random numbers* $\mathbf{u}_{1:i-1}$.

The consequence of the above is that MAF and IAF are different models with different computational trade-offs. MAF is capable of calculating the density $p(\mathbf{x})$ of *any* datapoint $\mathbf{x}$ in one pass through the model, however sampling from it requires performing D sequential passes (where D is the dimensionality of $\mathbf{x}$). In contrast, IAF can generate samples and calculate their density with one pass, however calculating the density $p(\mathbf{x})$ of an externally provided datapoint $\mathbf{x}$ requires D passes to find the random numbers $\mathbf{u}$ associated with $\mathbf{x}$. Hence, the design choice of whether to connect μ_i and α_i directly to $\mathbf{x}_{1:i-1}$ (obtaining MAF) or to $\mathbf{u}_{1:i-1}$ (obtaining IAF) depends on the intended usage. IAF is suitable as a recognition model for stochastic variational inference [15, 28], where it only ever needs to calculate the density of its own samples. In contrast, MAF is more suitable for density estimation, because each example requires only one pass through the model whereas IAF requires D.

A theoretical equivalence between MAF and IAF is that training a MAF with maximum likelihood corresponds to fitting an implicit IAF to the base density with stochastic variational inference. Let $\pi_x(\mathbf{x})$ be the data density we wish to learn, $\pi_u(\mathbf{u})$ be the base density, and f be the transformation from $\mathbf{u}$ to $\mathbf{x}$ as implemented by MAF. The density defined by MAF (with added subscript x for disambiguation) is

$$p_x(\mathbf{x}) = \pi_u\big(f^{-1}(\mathbf{x})\big) \left| \det\left(\frac{\partial f^{-1}}{\partial \mathbf{x}} \right) \right|. \tag{7}$$

The inverse transformation f^{-1} from $\mathbf{x}$ to $\mathbf{u}$ can be seen as describing an implicit IAF with base density $\pi_x(\mathbf{x})$, which defines the following implicit density over the $\mathbf{u}$ space:

$$p_u(\mathbf{u}) = \pi_x(f(\mathbf{u})) \left| \det\left(\frac{\partial f}{\partial \mathbf{u}} \right) \right|. \tag{8}$$

Training MAF by maximizing the total log likelihood $\sum_n \log p(\mathbf{x}_n)$ on train data $\{\mathbf{x}_n\}$ corresponds to fitting $p_x(\mathbf{x})$ to $\pi_x(\mathbf{x})$ by stochastically minimizing $D_{\mathrm{KL}}(\pi_x(\mathbf{x}) \,\|\, p_x(\mathbf{x}))$. In Section A of the appendix, we show that

$$D_{\mathrm{KL}}(\pi_x(\mathbf{x}) \,\|\, p_x(\mathbf{x})) = D_{\mathrm{KL}}(p_u(\mathbf{u}) \,\|\, \pi_u(\mathbf{u})). \tag{9}$$

Hence, stochastically minimizing $D_{\mathrm{KL}}(\pi_x(\mathbf{x}) \,\|\, p_x(\mathbf{x}))$ is equivalent to fitting $p_u(\mathbf{u})$ to $\pi_u(\mathbf{u})$ by minimizing $D_{\mathrm{KL}}(p_u(\mathbf{u}) \,\|\, \pi_u(\mathbf{u}))$. Since the latter is the loss function used in variational inference, and $p_u(\mathbf{u})$ can be seen as an IAF with base density $\pi_x(\mathbf{x})$ and transformation f^{-1}, it follows that training MAF as a density estimator of $\pi_x(\mathbf{x})$ is equivalent to performing stochastic variational inference with an implicit IAF, where the posterior is taken to be the base density $\pi_u(\mathbf{u})$ and the transformation f^{-1} implements the reparameterization trick [15, 28]. This argument is presented in more detail in Section A of the appendix.

3.3 Relationship with Real NVP

Real NVP [6] (NVP stands for Non Volume Preserving) is a normalizing flow obtained by stacking *coupling layers*. A coupling layer is an invertible transformation f from random numbers $\mathbf{u}$ to data $\mathbf{x}$ with a tractable Jacobian, defined by

$$
\begin{aligned}
\mathbf{x}_{1:d} &= \mathbf{u}_{1:d} \\
\mathbf{x}_{d+1:D} &= \mathbf{u}_{d+1:D} \odot \exp\boldsymbol{\alpha} + \boldsymbol{\mu}
\end{aligned}
\qquad \text{where} \qquad
\begin{aligned}
\boldsymbol{\mu} &= f_\mu(\mathbf{u}_{1:d}) \\
\boldsymbol{\alpha} &= f_\alpha(\mathbf{u}_{1:d}).
\end{aligned}
\tag{10}
$$

In the above, $\odot$ denotes elementwise multiplication, and the $\exp$ is applied to each element of $\boldsymbol{\alpha}$. The transformation copies the first d elements, and scales and shifts the remaining $D-d$ elements, with the amount of scaling and shifting being a function of the first d elements. When stacking coupling layers into a flow, the elements are permuted across layers so that a different set of elements is copied each time. A special case of the coupling layer where $\boldsymbol{\alpha}=\mathbf{0}$ is used by NICE [5].

We can see that the coupling layer is a special case of both the autoregressive transformation used by MAF in Equation (3), and the autoregressive transformation used by IAF in Equation (6). Indeed, we can recover the coupling layer from the autoregressive transformation of MAF by setting $\mu_i = \alpha_i = 0$ for $i \leq d$ and making μ_i and α_i functions of only $\mathbf{x}_{1:d}$ for $i > d$ (for IAF we need to make μ_i and α_i functions of $\mathbf{u}_{1:d}$ instead for $i > d$). In other words, both MAF and IAF can be seen as more flexible (but different) generalizations of Real NVP, where each element is individually scaled and shifted as a function of all previous elements. The advantage of Real NVP compared to MAF and IAF is that it can both generate data and estimate densities with one forward pass only, whereas MAF would need D passes to generate data and IAF would need D passes to estimate densities.

3.4 Conditional MAF

Given a set of example pairs $\{(\mathbf{x}_n, \mathbf{y}_n)\}$, conditional density estimation is the task of estimating the conditional density $p(\mathbf{x} \,|\, \mathbf{y})$. Autoregressive modelling extends naturally to conditional density estimation. Each term in the chain rule of probability can be conditioned on side-information $\mathbf{y}$, decomposing any conditional density as $p(\mathbf{x} \,|\, \mathbf{y}) = \prod_i p(x_i \,|\, \mathbf{x}_{1:i-1}, \mathbf{y})$. Therefore, we can turn any unconditional autoregressive model into a conditional one by augmenting its set of input variables with $\mathbf{y}$ and only modelling the conditionals that correspond to $\mathbf{x}$. Any order of the variables can be chosen, as long as $\mathbf{y}$ comes before $\mathbf{x}$. In masked autoregressive models, no connections need to be dropped from the $\mathbf{y}$ inputs to the rest of the network.

We can implement a conditional version of MAF by stacking MADEs that were made conditional using the above strategy. That is, in a conditional MAF, the vector $\mathbf{y}$ becomes an additional input for every layer. As a special case of MAF, Real NVP can be made conditional in the same way.

4 Experiments

4.1 Implementation and setup

We systematically evaluate three types of density estimator (MADE, Real NVP and MAF) in terms of density estimation performance on a variety of datasets. Code for reproducing our experiments (which uses Theano [33]) can be found at `https://github.com/gpapamak/maf`.

MADE. We consider two versions: (a) a MADE with Gaussian conditionals, denoted simply by MADE, and (b) a MADE whose conditionals are each parameterized as a mixture of C Gaussians, denoted by MADE MoG. We used $C = 10$ in all our experiments. MADE can be seen either as a MADE MoG with $C = 1$, or as a MAF with only one autoregressive layer. Adding more Gaussian components per conditional or stacking MADEs to form a MAF are two alternative ways of increasing the flexibility of MADE, which we are interested in comparing.

Real NVP. We consider a general-purpose implementation of the coupling layer, which uses two feedforward neural networks, implementing the scaling function f_α and the shifting function f_μ respectively. Both networks have the same architecture, except that f_α has hyperbolic tangent hidden units, whereas f_μ has rectified linear hidden units (we found this combination to perform best). Both networks have a linear output. We consider Real NVPs with either 5 or 10 coupling layers, denoted by Real NVP (5) and Real NVP (10) respectively, and in both cases the base density is a standard Gaussian. Successive coupling layers alternate between (a) copying the odd-indexed variables and transforming the even-indexed variables, and (b) copying the even-indexed variables and transforming the odd-indexed variables.

It is important to clarify that this is a general-purpose implementation of Real NVP which is different and thus not comparable to its original version [6], which was designed specifically for image data. Here we are interested in comparing coupling layers with autoregressive layers as building blocks of normalizing flows for general-purpose density estimation tasks, and our design of Real NVP is such that a fair comparison between the two can be made.

MAF. We consider three versions: (a) a MAF with 5 autoregressive layers and a standard Gaussian as a base density $\pi_u(\mathbf{u})$, denoted by MAF (5), (b) a MAF with 10 autoregressive layers and a standard Gaussian as a base density, denoted by MAF (10), and (c) a MAF with 5 autoregressive layers and a MADE MoG with $C = 10$ Gaussian components in each conditional as a base density, denoted by MAF MoG (5). MAF MoG (5) can be thought of as a MAF (5) stacked on top of a MADE MoG and trained jointly with it.

In all experiments, MADE and MADE MoG order the inputs using the order that comes with the dataset by default; no alternative orders were considered. MAF uses the default order for the first autoregressive layer (i.e. the layer that directly models the data) and reverses the order for each successive layer (the same was done for IAF by Kingma et al. [16]).

MADE, MADE MoG and each layer in MAF is a feedforward neural network with masked weight matrices, such that the autoregressive property holds. The procedure for designing the masks (due to Germain et al. [9]) is as follows. Each input or hidden unit is assigned a degree, which is an integer ranging from 1 to D, where D is the data dimensionality. The degree of an input is taken to be its index in the order. The D outputs have degrees that sequentially range from 0 to $D - 1$. A unit is allowed to receive input only from units with lower or equal degree, which enforces the autoregressive property. In order for output i to be connected to all inputs with degree less than i, and thus make sure that no conditional independences are introduced, it is both necessary and sufficient that every hidden layer contains every degree. In all experiments except for CIFAR-10, we sequentially assign degrees within each hidden layer and use enough hidden units to make sure that all degrees appear. Because CIFAR-10 is high-dimensional, we used fewer hidden units than inputs and assigned degrees to hidden units uniformly at random (as was done by Germain et al. [9]).

We added batch normalization [13] after each coupling layer in Real NVP and after each autoregressive layer in MAF. Batch normalization is an elementwise scaling and shifting operation, which is easily invertible and has a tractable Jacobian, and thus it is suitable for use in a normalizing flow. We found that batch normalization in Real NVP and MAF reduces training time, increases stability during training and improves performance (a fact that was also observed by Dinh et al. [6] for Real NVP). Section B of the appendix discusses our implementation of batch normalization and its use in normalizing flows.

All models were trained with the Adam optimizer [14], using a minibatch size of 100, and a step size of 10^{-3} for MADE and MADE MoG, and of 10^{-4} for Real NVP and MAF. A small amount of ℓ_2 regularization was added, with coefficient 10^{-6}. Each model was trained with early stopping until no improvement occurred for 30 consecutive epochs on the validation set. For each model, we selected the number of hidden layers and number of hidden units based on validation performance (we gave the same options to all models), as described in Section D of the appendix.

Table 1: Average test log likelihood (in nats) for unconditional density estimation. The best performing model for each dataset is shown in bold (multiple models are highlighted if the difference is not statistically significant according to a paired t-test). Error bars correspond to 2 standard deviations.

	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
Gaussian	-7.74 ± 0.02	-3.58 ± 0.75	-27.93 ± 0.02	-37.24 ± 1.07	96.67 ± 0.25
MADE	-3.08 ± 0.03	3.56 ± 0.04	-20.98 ± 0.02	-15.59 ± 0.50	148.85 ± 0.28
MADE MoG	$\mathbf{0.40 \pm 0.01}$	8.47 ± 0.02	$\mathbf{-15.15 \pm 0.02}$	-12.27 ± 0.47	153.71 ± 0.28
Real NVP (5)	-0.02 ± 0.01	4.78 ± 1.80	-19.62 ± 0.02	-13.55 ± 0.49	152.97 ± 0.28
Real NVP (10)	0.17 ± 0.01	8.33 ± 0.14	-18.71 ± 0.02	-13.84 ± 0.52	153.28 ± 1.78
MAF (5)	0.14 ± 0.01	9.07 ± 0.02	-17.70 ± 0.02	$\mathbf{-11.75 \pm 0.44}$	155.69 ± 0.28
MAF (10)	0.24 ± 0.01	$\mathbf{10.08 \pm 0.02}$	-17.73 ± 0.02	-12.24 ± 0.45	154.93 ± 0.28
MAF MoG (5)	0.30 ± 0.01	9.59 ± 0.02	-17.39 ± 0.02	$\mathbf{-11.68 \pm 0.44}$	$\mathbf{156.36 \pm 0.28}$

4.2 Unconditional density estimation

Following Uria et al. [36], we perform unconditional density estimation on four UCI datasets (POWER, GAS, HEPMASS, MINIBOONE) and on a dataset of natural image patches (BSDS300).

UCI datasets. These datasets were taken from the UCI machine learning repository [21]. We selected different datasets than Uria et al. [36], because the ones they used were much smaller, resulting in an expensive cross-validation procedure involving a separate hyperparameter search for each fold. However, our data preprocessing follows Uria et al. [36]. The sample mean was subtracted from the data and each feature was divided by its sample standard deviation. Discrete-valued attributes were eliminated, as well as every attribute with a Pearson correlation coefficient greater than 0.98. These procedures are meant to avoid trivial high densities, which would make the comparison between approaches hard to interpret. Section D of the appendix gives more details about the UCI datasets and the individual preprocessing done on each of them.

Image patches. This dataset was obtained by extracting random 8×8 monochrome patches from the BSDS300 dataset of natural images [23]. We used the same preprocessing as by Uria et al. [36]. Uniform noise was added to dequantize pixel values, which was then rescaled to be in the range $[0, 1]$. The mean pixel value was subtracted from each patch, and the bottom-right pixel was discarded.

Table 1 shows the performance of each model on each dataset. A Gaussian fitted to the train data is reported as a baseline. We can see that on 3 out of 5 datasets MAF is the best performing model, with MADE MoG being the best performing model on the other 2. On all datasets, MAF outperforms Real NVP. For the MINIBOONE dataset, due to overlapping error bars, a pairwise comparison was done to determine which model performs the best, the results of which are reported in Section E of the appendix. MAF MoG (5) achieves the best reported result on BSDS300 for a single model with 156.36 nats, followed by Deep RNADE [37] with 155.2. An ensemble of 32 Deep RNADEs was reported to achieve 157.0 nats [37]. The UCI datasets were used for the first time in the literature for density estimation, so no comparison with existing work can be made yet.

4.3 Conditional density estimation

For conditional density estimation, we used the MNIST dataset of handwritten digits [20] and the CIFAR-10 dataset of natural images [17]. In both datasets, each datapoint comes from one of 10 distinct classes. We represent the class label as a 10-dimensional, one-hot encoded vector $\mathbf{y}$, and we model the density $p(\mathbf{x} \,|\, \mathbf{y})$, where $\mathbf{x}$ represents an image. At test time, we evaluate the probability of a test image $\mathbf{x}$ by $p(\mathbf{x}) = \sum_{\mathbf{y}} p(\mathbf{x} \,|\, \mathbf{y}) p(\mathbf{y})$, where $p(\mathbf{y}) = \frac{1}{10}$ is a uniform prior over the labels. For comparison, we also train every model as an unconditional density estimator and report both results.

For both MNIST and CIFAR-10, we use the same preprocessing as by Dinh et al. [6]. We dequantize pixel values by adding uniform noise, and then rescale them to $[0, 1]$. We transform the rescaled pixel values into logit space by $\mathbf{x} \mapsto \text{logit}(\lambda + (1 - 2\lambda)\mathbf{x})$, where $\lambda = 10^{-6}$ for MNIST and $\lambda = 0.05$ for CIFAR-10, and perform density estimation in that space. In the case of CIFAR-10, we also augment the train set with horizontal flips of all train examples (as also done by Dinh et al. [6]).

Table 2: Average test log likelihood (in nats) for conditional density estimation. The best performing model for each dataset is shown in bold. Error bars correspond to 2 standard deviations.

	MNIST		CIFAR-10	
	unconditional	conditional	unconditional	conditional
Gaussian	-1366.9 ± 1.4	-1344.7 ± 1.8	2367 ± 29	2030 ± 41
MADE	-1380.8 ± 4.8	-1361.9 ± 1.9	147 ± 20	187 ± 20
MADE MoG	$\mathbf{-1038.5 \pm 1.8}$	$\mathbf{-1030.3 \pm 1.7}$	-397 ± 21	-119 ± 20
Real NVP (5)	-1323.2 ± 6.6	-1326.3 ± 5.8	2576 ± 27	2642 ± 26
Real NVP (10)	-1370.7 ± 10.1	-1371.3 ± 43.9	2568 ± 26	2475 ± 25
MAF (5)	-1300.5 ± 1.7	$-1302.9 \pm 1.7*$	2936 ± 27	$2983 \pm 26*$
MAF (10)	-1313.1 ± 2.0	$-1316.8 \pm 1.8*$	$\mathbf{3049 \pm 26}$	$\mathbf{3058 \pm 26}*$
MAF MoG (5)	-1100.3 ± 1.6	-1092.3 ± 1.7	2911 ± 26	2936 ± 26

Table 2 shows the results on MNIST and CIFAR-10. The performance of a class-conditional Gaussian is reported as a baseline for the conditional case. Log likelihoods are calculated in logit space. MADE MoG is the best performing model on MNIST, whereas MAF is the best performing model on CIFAR-10. On CIFAR-10, both MADE and MADE MoG performed significantly worse than the Gaussian baseline. MAF outperforms Real NVP in all cases. To facilitate comparison with the literature, Section E of the appendix reports results in bits/pixel.[1]

5   Discussion

We showed that we can improve MADE by modelling the density of its internal random numbers. Alternatively, MADE can be improved by increasing the flexibility of its conditionals. The comparison between MAF and MADE MoG showed that the best approach is dataset specific; in our experiments MAF outperformed MADE MoG in 5 out of 9 cases, which is strong evidence of its competitiveness. MADE MoG is a universal density approximator; with sufficiently many hidden units and Gaussian components, it can approximate any continuous density arbitrarily well. It is an open question whether MAF with a Gaussian base density has a similar property (MAF MoG clearly does).

We also showed that the coupling layer used in Real NVP is a special case of the autoregressive layer used in MAF. In fact, MAF outperformed Real NVP in all our experiments. Real NVP has achieved impressive performance in image modelling by incorporating knowledge about image structure. Our results suggest that replacing coupling layers with autoregressive layers in the original version of Real NVP is a promising direction for further improving its performance. Real NVP maintains however the advantage over MAF (and autoregressive models in general) that samples from the model can be generated efficiently in parallel.

Density estimation is one of several types of generative modelling, with the focus on obtaining accurate densities. However, we know that accurate densities do not necessarily imply good performance in other tasks, such as in data generation [35]. Alternative approaches to generative modelling include variational autoencoders [15, 28], which are capable of efficient inference of their (potentially interpretable) latent space, and generative adversarial networks [10], which are capable of high quality data generation. Choice of method should be informed by whether the application at hand calls for accurate densities, latent space inference or high quality samples. Masked Autoregressive Flow is a contribution towards the first of these goals.

---

[1]In earlier versions of the paper, results marked with * were reported incorrectly, due to an error in the calculation of the test log likelihood of conditional MAF. Thus error has been corrected in the current version.

A Equivalence between MAF and IAF

In this section, we present the equivalence between MAF and IAF in full mathematical detail. Let $\pi_x(\mathbf{x})$ be the true density the train data $\{\mathbf{x}_n\}$ is sampled from. Suppose we have a MAF whose base density is $\pi_u(\mathbf{u})$, and whose transformation from $\mathbf{u}$ to $\mathbf{x}$ is f. The MAF defines the following density over the $\mathbf{x}$ space:

$$p_x(\mathbf{x}) = \pi_u\big(f^{-1}(\mathbf{x})\big) \left| \det\left(\frac{\partial f^{-1}}{\partial \mathbf{x}} \right) \right|. \tag{11}$$

Using the definition of $p_x(\mathbf{x})$ in Equation (11), we can write the Kullback–Leibler divergence from $\pi_x(\mathbf{x})$ to $p_x(\mathbf{x})$ as follows:

$$D_{\mathrm{KL}}(\pi_x(\mathbf{x}) \,\|\, p_x(\mathbf{x})) = \mathbb{E}_{\pi_x(\mathbf{x})}(\log \pi_x(\mathbf{x}) - \log p_x(\mathbf{x})) \tag{12}$$

$$= \mathbb{E}_{\pi_x(\mathbf{x})}\left(\log \pi_x(\mathbf{x}) - \log \pi_u\big(f^{-1}(\mathbf{x})\big) - \log \left| \det\left(\frac{\partial f^{-1}}{\partial \mathbf{x}} \right) \right| \right). \tag{13}$$

The inverse transformation f^{-1} from $\mathbf{x}$ to $\mathbf{u}$ can be seen as describing an implicit IAF with base density $\pi_x(\mathbf{x})$, which would define the following density over the $\mathbf{u}$ space:

$$p_u(\mathbf{u}) = \pi_x(f(\mathbf{u})) \left| \det\left(\frac{\partial f}{\partial \mathbf{u}} \right) \right|. \tag{14}$$

By making the change of variables $\mathbf{x} \mapsto \mathbf{u}$ in Equation (13) and using the definition of $p_u(\mathbf{u})$ in Equation (14) we obtain

$$D_{\mathrm{KL}}(\pi_x(\mathbf{x}) \,\|\, p_x(\mathbf{x})) = \mathbb{E}_{p_u(\mathbf{u})}\left(\log \pi_x(f(\mathbf{u})) - \log \pi_u(\mathbf{u}) + \log \left| \det\left(\frac{\partial f}{\partial \mathbf{u}} \right) \right| \right) \tag{15}$$

$$= \mathbb{E}_{p_u(\mathbf{u})}(\log p_u(\mathbf{u}) - \log \pi_u(\mathbf{u})). \tag{16}$$

Equation (16) is the definition of the KL divergence from $p_u(\mathbf{u})$ to $\pi_u(\mathbf{u})$, hence

$$D_{\mathrm{KL}}(\pi_x(\mathbf{x}) \,\|\, p_x(\mathbf{x})) = D_{\mathrm{KL}}(p_u(\mathbf{u}) \,\|\, \pi_u(\mathbf{u})). \tag{17}$$

Suppose now that we wish to fit the implicit density $p_u(\mathbf{u})$ to the base density $\pi_u(\mathbf{u})$ by minimizing the above KL. This corresponds exactly to the objective minimized when employing IAF as a recognition network in stochastic variational inference [16], where $\pi_u(\mathbf{u})$ would be the (typically intractable) posterior. The first step in stochastic variational inference would be to rewrite the expectation in Equation (16) with respect to the base distribution $\pi_x(\mathbf{x})$ used by IAF, which corresponds exactly to Equation (13). This is often referred to as the reparameterization trick [15, 28]. The second step would be to approximate Equation (13) with Monte Carlo, using samples $\{\mathbf{x}_n\}$ drawn from $\pi_x(\mathbf{x})$, as follows:

$$D_{\mathrm{KL}}(p_u(\mathbf{u}) \,\|\, \pi_u(\mathbf{u})) = \mathbb{E}_{\pi_x(\mathbf{x})}\left(\log \pi_x(\mathbf{x}) - \log \pi_u\big(f^{-1}(\mathbf{x})\big) - \log \left| \det\left(\frac{\partial f^{-1}}{\partial \mathbf{x}} \right) \right| \right) \tag{18}$$

$$\approx \frac{1}{N} \sum_n \left(\log \pi_x(\mathbf{x}_n) - \log \pi_u\big(f^{-1}(\mathbf{x}_n)\big) - \log \left| \det\left(\frac{\partial f^{-1}}{\partial \mathbf{x}} \right) \right| \right). \tag{19}$$

Using the definition of $p_x(\mathbf{x})$ in Equation (11), we can rewrite Equation (19) as

$$\frac{1}{N} \sum_n (\log \pi_x(\mathbf{x}_n) - \log p_x(\mathbf{x}_n)) = -\frac{1}{N} \sum_n \log p_x(\mathbf{x}_n) + \mathrm{const}. \tag{20}$$

Since samples $\{\mathbf{x}_n\}$ drawn from $\pi_x(\mathbf{x})$ correspond precisely to the train data for MAF, we can recognize in Equation (20) the training objective for MAF. In conclusion, training a MAF by maximizing its total log likelihood $\sum_n \log p_x(\mathbf{x}_n)$ on train data $\{\mathbf{x}_n\}$ is equivalent to variationally training an implicit IAF with MAF's base distribution $\pi_u(\mathbf{u})$ as its target.

B Batch normalization

In our implementation of MAF, we inserted a batch normalization layer [13] between every two autoregressive layers, and between the last autoregressive layer and the base distribution. We did the

same for Real NVP (the original implementation of Real NVP also uses batch normalization layers between coupling layers [6]). The purpose of a batch normalization layer is to normalize its inputs $\mathbf{x}$ to have approximately zero mean and unit variance. In this section, we describe in full detail our implementation of batch normalization and its use as a layer in normalizing flows.

A batch normalization layer can be thought of as a transformation between two vectors of the same dimensionality. For consistency with our notation for autoregressive and coupling layers, let $\mathbf{x}$ be the vector closer to the data, and $\mathbf{u}$ be the vector closer to the base distribution. Batch normalization implements the transformation $\mathbf{x} = f(\mathbf{u})$ defined by

$$\mathbf{x} = (\mathbf{u} - \boldsymbol{\beta}) \odot \exp(-\boldsymbol{\gamma}) \odot (\mathbf{v} + \epsilon)^{\frac{1}{2}} + \mathbf{m}. \tag{21}$$

In the above, $\odot$ denotes elementwise multiplication. All other operations are to be understood elementwise. The inverse transformation f^{-1} is given by

$$\mathbf{u} = (\mathbf{x} - \mathbf{m}) \odot (\mathbf{v} + \epsilon)^{-\frac{1}{2}} \odot \exp \boldsymbol{\gamma} + \boldsymbol{\beta}, \tag{22}$$

and the absolute determinant of its Jacobian is

$$\left| \det\left(\frac{\partial f^{-1}}{\partial \mathbf{x}} \right) \right| = \exp\left(\sum_i \left(\gamma_i - \frac{1}{2} \log(v_i + \epsilon) \right) \right). \tag{23}$$

Vectors $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ are parameters of the transformation that are learnt during training. In typical implementations of batch normalization, parameter $\boldsymbol{\gamma}$ is not exponentiated. In our implementation, we chose to exponentiate $\boldsymbol{\gamma}$ in order to ensure its positivity and simplify the expression of the log absolute determinant. Parameters $\mathbf{m}$ and $\mathbf{v}$ correspond to the mean and variance of $\mathbf{x}$ respectively. During training, we set $\mathbf{m}$ and $\mathbf{v}$ equal to the sample mean and variance of the current minibatch (we used minibatches of 100 examples). At validation and test time, we set them equal to the sample mean and variance of the entire train set. Other implementations use averages over minibatches [13] or maintain running averages during training [6]. Finally, ϵ is a hyperparameter that ensures numerical stability if any of the elements of $\mathbf{v}$ is near zero. In our experiments, we used $\epsilon = 10^{-5}$.

C   Number of parameters

To get a better idea of the computational trade-offs between different model choices versus the performance gains they achieve, we compare the number of parameters for each model. We only count connection weights, as they contribute the most, and ignore biases and batch normalization parameters. We assume that masking reduces the number of connections by approximately half.

For all models, let D be the number of inputs, H be the number of units in a hidden layer and L be the number of hidden layers. We assume that all hidden layers have the same number of units (as we did in our experiments). For MAF MoG, let C be the number of components per conditional. For Real NVP and MAF, let K be the number of coupling layers/autoregressive layers respectively. Table 3 lists the number of parameters for each model.

For each extra component we add to MADE MoG, we increase the number of parameters by DH. For each extra autoregressive layer we add to MAF, we increase the number of parameters by $\frac{3}{2}DH + \frac{1}{2}(L-1)H^2$. If we have one or two hidden layers L (as we did in our experiments) and assume that D is comparable to H, the number of extra parameters in both cases is about the same. In other words, increasing flexibility by stacking has a parameter cost that is similar to adding more components to the conditionals, as long as the number of hidden layers is small.

Comparing Real NVP with MAF, we can see that Real NVP has about 1.3 to 2 times more parameters than a MAF of comparable size. Given that our experiments show that Real NVP is less flexible than a MAF of comparable size, we can conclude that MAF makes better use of its available capacity. The number of parameters of Real NVP could be reduced by tying weights between the scaling and shifting networks.

D   Additional experimental details

D.1   Models

MADE, MADE MoG and each autoregressive layer in MAF is a feedforward neural network (with masked weight matrices), with L hidden layers of H hidden units each. Similarly, each coupling

Table 3: Approximate number of parameters for each model, as measured by number of connection weights. Biases and batch normalization parameters are ignored.

	# of parameters
MADE	$\frac{3}{2}DH + \frac{1}{2}(L-1)H^2$
MADE MoG	$\left(C + \frac{1}{2}\right)DH + \frac{1}{2}(L-1)H^2$
Real NVP	$2KDH + 2K(L-1)H^2$
MAF	$\frac{3}{2}KDH + \frac{1}{2}K(L-1)H^2$

layer in Real NVP contains two feedforward neural networks, one for scaling and one for shifting, each of which also has L hidden layers of H hidden units each. For each dataset, we gave a number of options for L and H (the same options where given to all models) and for each model we selected the option that performed best on the validation set. Table 4 lists the combinations of L and H that were given as options for each dataset.

In terms of nonlinearity for the hidden units, MADE, MADE MoG and MAF used rectified linear units, except for the GAS datasets where we used hyperbolic tangent units. In the coupling layer of Real NVP, we used hyberbolic tangent hidden units for the scaling network and rectified linear hidden units for the shifting network.

Table 4: Number of hidden layers L and number of hidden units H given as options for each dataset. Each combination is reported in the format $L \times H$.

POWER	GAS	HEPMASS	MINIBOONE	BSDS300	MNIST	CIFAR-10
1×100	1×100	1×512	1×512	1×512	1×1024	1×1024
2×100	2×100	2×512	2×512	2×512		2×1024
				1×1024		2×2048
				2×1024		

D.2 Datasets

In the following paragraphs, we give a brief description of the four UCI datasets (POWER, GAS, HEPMASS, MINIBOONE) and of the way they were preprocessed.

POWER. The POWER dataset [1] contains measurements of electric power consumption in a household over a period of 47 months. It is actually a time series but was treated as if each example were an i.i.d. sample from the marginal distribution. The time feature was turned into an integer for the number of minutes in the day and then uniform random noise was added to it. The date was discarded, along with the global reactive power parameter, which seemed to have many values at exactly zero, which could have caused arbitrarily large spikes in the learnt distribution. Uniform random noise was added to each feature in the interval $[0, \epsilon_i]$, where ϵ_i is large enough to ensure that with high probability there are no identical values for the ith feature but small enough to not change the data values significantly.

GAS. Created by Fonollosa et al. [8], this dataset represents the readings of an array of 16 chemical sensors exposed to gas mixtures over a 12 hour period. Similarly to POWER, it is a time series but was treated as if each example were an i.i.d. sample from the marginal distribution. Only the data from the file `ethylene_CO.txt` was used, which corresponds to a mixture of ethylene and carbon monoxide. After removing strongly correlated attributes, the dimensionality was reduced to 8.

HEPMASS. Used by Baldi et al. [2], this dataset describes particle collisions in high energy physics. Half of the data are examples of particle-producing collisions (positive), whereas the rest come from a background source (negative). Here we used the positive examples from the "1000" dataset, where

the particle mass is 1000. Five features were removed because they had too many reoccurring values; values that repeat too often can result in spikes in the density and misleading results.

MINIBOONE. Used by Roe et al. [30], this dataset comes from the MiniBooNE experiment at Fermilab. Similarly to HEPMASS, it contains a number of positive examples (electron neutrinos) and a number of negative examples (muon neutrinos). Here we use the positive examples. These had some obvious outliers (11) which had values at exactly -1000 for every column and were removed. Also, seven of the features had far too high a count for a particular value, e.g. 0.0, so these were removed as well.

Table 5 lists the dimensionality and the number of train, validation and test examples for all seven datasets. The first three datasets in Table 5 were subsampled so that the product of the dimensionality and number of examples would be approximately 10M. For the four UCI datasets, 10% of the data was held out and used as test data and 10% of the remaining data was used as validation data. From the BSDS300 dataset we randomly extracted 1M patches for training, 50K patches for validation and 250K patches for testing. For MNIST and CIFAR-10 we held out 10% of the train data for validation. We augmented the CIFAR-10 train set with the horizontal flips of all remaining 45K train examples.

Table 5: Dimensionality D and number of examples N for each dataset.

	D	N train	validation	test
POWER	6	1,659,917	184,435	204,928
GAS	8	852,174	94,685	105,206
HEPMASS	21	315,123	35,013	174,987
MINIBOONE	43	29,556	3,284	3,648
BSDS300	63	1,000,000	50,000	250,000
MNIST	784	50,000	10,000	10,000
CIFAR-10	3072	90,000	5,000	10,000

E Additional results

E.1 Pairwise comparison

On the MINIBOONE dataset, the model with highest average test log likelihood is MAF MoG (5). However, due to the relatively small size of this dataset, the average test log likelihoods of some other models have overlapping error bars with that of MAF MoG (5). To assess whether the differences are statistically significant, we performed a pairwise comparison, which is a more powerful statistical test. In particular, we calculated the difference in test log probability between every other model and MAF MoG (5) on each test example, and assessed whether this difference is significantly positive, which would indicate that MAF MoG (5) performs significantly better. The results of this comparison are shown in Table 6. We can see that MAF MoG (5) is significantly better than all other models except for MAF (5).

E.2 Bits per pixel

In the main text, the results for MNIST and CIFAR-10 were reported in log likelihoods in logit space, since this is the objective that the models were trained to optimize. For comparison with other results in the literature, in Table 7 we report the same results in bits per pixel. For CIFAR-10, different colour components count as different pixels (i.e. an image is thought of as having $32 \times 32 \times 3$ pixels).

In order to calculate bits per pixel, we need to transform the densities returned by a model (which refer to logit space) back to image space in the range $[0, 256]$. Let $\mathbf{x}$ be an image of D pixels in logit space and $\mathbf{z}$ be the corresponding image in $[0, 256]$ image space. The transformation from $\mathbf{z}$ to $\mathbf{x}$ is

$$\mathbf{x} = \text{logit}\left(\lambda + (1 - 2\lambda)\frac{\mathbf{z}}{256}\right), \tag{24}$$

Table 6: Pairwise comparison results for MINIBOONE. Values correspond to average difference in log probability (in nats) from the best performing model, i.e. MAF MoG (5). Error bars correspond to 2 standard deviations. Significantly positive values indicate that MAF MoG (5) performs better.

	MINIBOONE
Gaussian	25.55 ± 0.88
MADE	3.91 ± 0.20
MADE MoG	0.59 ± 0.16
Real NVP (5)	1.87 ± 0.16
Real NVP (10)	2.15 ± 0.21
MAF (5)	$\mathbf{0.07 \pm 0.11}$
MAF (10)	0.55 ± 0.12
MAF MoG (5)	$\mathbf{0.00 \pm 0.00}$

Table 7: Bits per pixel for conditional density estimation (lower is better). The best performing model for each dataset is shown in bold. Error bars correspond to 2 standard deviations.

	MNIST		CIFAR-10	
	unconditional	conditional	unconditional	conditional
Gaussian	2.01 ± 0.01	1.97 ± 0.01	4.63 ± 0.01	4.79 ± 0.02
MADE	2.04 ± 0.01	2.00 ± 0.01	5.67 ± 0.01	5.65 ± 0.01
MADE MoG	$\mathbf{1.41 \pm 0.01}$	$\mathbf{1.39 \pm 0.01}$	5.93 ± 0.01	5.80 ± 0.01
Real NVP (5)	1.93 ± 0.01	1.94 ± 0.01	4.53 ± 0.01	4.50 ± 0.01
Real NVP (10)	2.02 ± 0.02	2.02 ± 0.08	4.54 ± 0.01	4.58 ± 0.01
MAF (5)	1.89 ± 0.01	$1.89 \pm 0.01*$	4.36 ± 0.01	$4.34 \pm 0.01*$
MAF (10)	1.91 ± 0.01	$1.92 \pm 0.01*$	$\mathbf{4.31 \pm 0.01}$	$\mathbf{4.30 \pm 0.01}*$
MAF MoG (5)	1.52 ± 0.01	1.51 ± 0.01	4.37 ± 0.01	4.36 ± 0.01

where $\lambda = 10^{-6}$ for MNIST and $\lambda = 0.05$ for CIFAR-10. If $p(\mathbf{x})$ is the density in logit space as returned by the model, using the above transformation the density of $\mathbf{z}$ can be calculated as

$$p_z(\mathbf{z}) = p(\mathbf{x}) \left(\frac{1 - 2\lambda}{256} \right)^D \left(\prod_i \sigma(x_i)(1 - \sigma(x_i)) \right)^{-1}, \tag{25}$$

where $\sigma(\cdot)$ is the logistic sigmoid function. From that, we can calculate the bits per pixel $b(\mathbf{x})$ of image $\mathbf{x}$ as follows:

$$b(\mathbf{x}) = -\frac{\log_2 p_z(\mathbf{z})}{D} \tag{26}$$

$$= -\frac{\log p(\mathbf{x})}{D \log 2} - \log_2(1 - 2\lambda) + 8 + \frac{1}{D} \sum_i (\log_2 \sigma(x_i) + \log_2(1 - \sigma(x_i))). \tag{27}$$

The above equation was used to convert between the average log likelihoods reported in the main text and the results of Table 7.

E.3 Generated images

Figures 2, 3 and 4 show generated images and real examples for BSDS300, MNIST and CIFAR-10 respectively. Images were generated by MAF MoG (5) for BSDS300, conditional MAF (5) for MNIST, and conditional MAF (10) for CIFAR-10.

The BSDS300 generated images are visually indistinguishable from the real ones. For MNIST and CIFAR-10, generated images lack the fidelity produced by modern image-based generative approaches, such as RealNVP [6] or PixelCNN++ [31]. This is because our version of MAF has

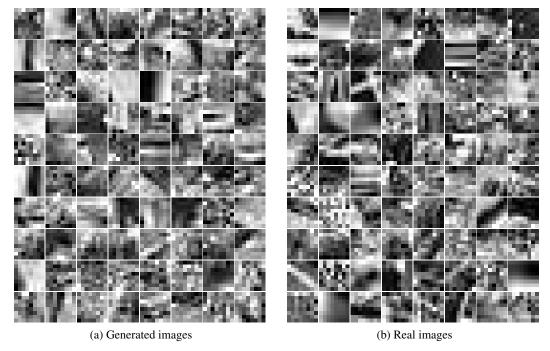(a) Generated images   (b) Real images

Figure 2: Generated and real images from BSDS300.

no knowledge about image structure, as it was designed for general-purpose density estimation and not for realistic-looking image synthesis. However, if the latter is desired, it would be possible to incorporate image modelling techniques in the design of MAF (such as convolutions or a multi-scale architecture as used by Real NVP [6]) in order to improve quality of generated images.

References

[1] Individual household electric power consumption data set. `http://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption`. Accessed on 15 May 2017.

[2] P. Baldi, K. Cranmer, T. Faucett, P. Sadowski, and D. Whiteson. Parameterized machine learning for high-energy physics. arXiv:1601.07913, 2016.

[3] J. Ballé, V. Laparra, and E. P. Simoncelli. Density modeling of images using a generalized normalization transformation. *Proceedings of the 4nd International Conference on Learning Representations*, 2016.

[4] S. S. Chen and R. A. Gopinath. Gaussianization. *Advances in Neural Information Processing Systems 13*, pages 423–429, 2001.

[5] L. Dinh, D. Krueger, and Y. Bengio. NICE: Non-linear Independent Components Estimation. arXiv:1410.8516, 2014.

[6] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using Real NVP. *Proceedings of the 5th International Conference on Learning Representations*, 2017.

[7] Y. Fan, D. J. Nott, and S. A. Sisson. Approximate Bayesian computation via regression density estimation. *Stat*, 2(1):34–48, 2013.

[8] J. Fonollosa, S. Sheik, R. Huerta, and S. Marco. Reservoir computing compensates slow response of chemosensor arrays exposed to fast varying gas concentrations in continuous monitoring. *Sensors and Actuators B: Chemical*, 215:618–629, 2015.

[9] M. Germain, K. Gregor, I. Murray, and H. Larochelle. MADE: Masked Autoencoder for Distribution Estimation. *Proceedings of the 32nd International Conference on Machine Learning*, pages 881–889, 2015.

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems 27*, pages 2672–2680, 2014.

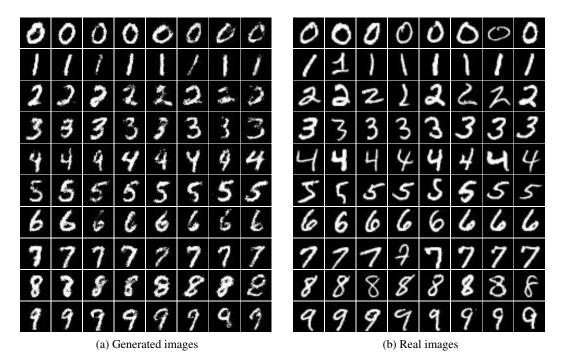(a) Generated images   (b) Real images

Figure 3: Class-conditional generated and real images from MNIST. Rows are different classes. Generated images are sorted by decreasing log likelihood from left to right.

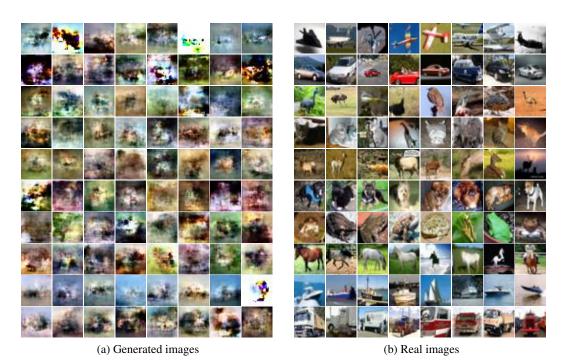(a) Generated images   (b) Real images

Figure 4: Class-conditional generated and real images from CIFAR-10. Rows are different classes. Generated images are sorted by decreasing log likelihood from left to right.

[11] S. Gu, Z. Ghahramani, and R. E. Turner. Neural adaptive sequential Monte Carlo. *Advances in Neural Information Processing Systems 28*, pages 2629–2637, 2015.

[12] G. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[13] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proceedings of the 32nd International Conference on Machine Learning*, pages 448–456, 2015.

[14] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

[15] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. *Proceedings of the 2nd International Conference on Learning Representations*, 2014.

[16] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. Improved variational inference with Inverse Autoregressive Flow. *Advances in Neural Information Processing Systems 29*, pages 4743–4751, 2016.

[17] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[18] T. D. Kulkarni, P. Kohli, J. B. Tenenbaum, and V. Mansinghka. Picture: A probabilistic programming language for scene perception. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4390–4399, 2015.

[19] T. A. Le, A. G. Baydin, and F. Wood. Inference compilation and universal probabilistic programming. *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 2017.

[20] Y. LeCun, C. Cortes, and C. J. C. Burges. The MNIST database of handwritten digits. URL `http://yann.lecun.com/exdb/mnist/`.

[21] M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

[22] G. Loaiza-Ganem, Y. Gao, and J. P. Cunningham. Maximum entropy flow networks. *Proceedings of the 5th International Conference on Learning Representations*, 2017.

[23] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. pages 416–423, 2001.

[24] B. Paige and F. Wood. Inference networks for sequential Monte Carlo in graphical models. *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

[25] G. Papamakarios and I. Murray. Distilling intractable generative models, 2015. Probabilistic Integration Workshop at Neural Information Processing Systems 28.

[26] G. Papamakarios and I. Murray. Fast ϵ-free inference of simulation models with Bayesian conditional density estimation. *Advances in Neural Information Processing Systems 29*, 2016.

[27] D. J. Rezende and S. Mohamed. Variational inference with normalizing flows. *Proceedings of the 32nd International Conference on Machine Learning*, pages 1530–1538, 2015.

[28] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *Proceedings of the 31st International Conference on Machine Learning*, pages 1278–1286, 2014.

[29] O. Rippel and R. P. Adams. High-dimensional probability estimation with deep density models. arXiv:1302.5125, 2013.

[30] B. P. Roe, H.-J. Yang, J. Zhu, Y. Liu, I. Stancu, and G. McGregor. Boosted decision trees as an alternative to artificial neural networks for particle identification. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 543(2–3):577–584, 2005.

[31] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. PixelCNN++: Improving the PixelCNN with discretized logistic mixture likelihood and other modifications. arXiv:1701.05517, 2017.

[32] Y. Tang, R. Salakhutdinov, and G. Hinton. Deep mixtures of factor analysers. *Proceedings of the 29th International Conference on Machine Learning*, pages 505–512, 2012.

[33] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. arXiv:1605.02688, 2016.

[34] L. Theis and M. Bethge. Generative image modeling using spatial LSTMs. *Advances in Neural Information Processing Systems 28*, pages 1927–1935, 2015.

[35] L. Theis, A. van den Oord, and M. Bethge. A note on the evaluation of generative models. *Proceedings of the 4nd International Conference on Learning Representations*, 2016.

[36] B. Uria, I. Murray, and H. Larochelle. RNADE: The real-valued neural autoregressive density-estimator. *Advances in Neural Information Processing Systems 26*, pages 2175–2183, 2013.

[37] B. Uria, I. Murray, and H. Larochelle. A deep and tractable density estimator. *Proceedings of the 31st International Conference on Machine Learning*, pages 467–475, 2014.

[38] B. Uria, I. Murray, S. Renals, C. Valentini-Botinhao, and J. Bridle. Modelling acoustic feature dependencies with artificial neural networks: Trajectory-RNADE. *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4465–4469, 2015.

[39] B. Uria, M.-A. Côté, K. Gregor, I. Murray, and H. Larochelle. Neural autoregressive distribution estimation. *Journal of Machine Learning Research*, 17(205):1–37, 2016.

[40] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu. WaveNet: A generative model for raw audio. arXiv:1609.03499, 2016.

[41] A. van den Oord, N. Kalchbrenner, L. Espeholt, K. Kavukcuoglu, O. Vinyals, and A. Graves. Conditional image generation with PixelCNN decoders. *Advances in Neural Information Processing Systems 29*, pages 4790–4798, 2016.

[42] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. *Proceedings of the 33rd International Conference on Machine Learning*, pages 1747–1756, 2016.

[43] D. Zoran and Y. Weiss. From learning models of natural image patches to whole image restoration. *Proceedings of the 13rd International Conference on Computer Vision*, pages 479–486, 2011.