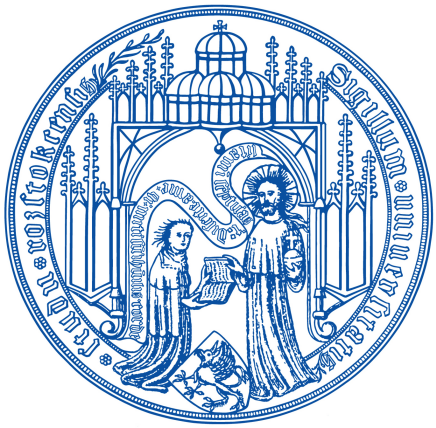
Density Estimation with Normalising Flows

Master Thesis

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik



submitted by:	Daniel Decker
matriculation number:	214207416
born on:	19.05.1988 in Potsdam
first supervisor:	Dr. Sebastian Bader
second supervisor:	Dr. Stefan Lüdke
submission date:	28.02.2022

Chapter 1

Introduction and Motivation

Calculating the real likelihood of a data point is only possible in rare ideal cases. When a data point is not from a parameterised distribution with known parameters, an estimate is needed. A medical doctor might want to seek the most likely disease given test results, patient data like age, race or sex. Choosing compound candidates for new drugs from large in vitro trials in pharmaceutics benefit from it. Whether forecasts or anomaly detection in security systems apply here too. A huge part of statistical sciences is about turning a set of samples, or dataset, into density functions.
Normalising Flows are a relatively new development in the domain of non parametric density estimation. And this is what this work is about. Tabak and Turner introduced the term in their 2013 paper [TT13]. Their idea is to chain a series of simple mappings together such that the chain transforms the data point to a point under a known distribution, like a standard normal. Some steps toward that direction have been done before [TVE10], [CG01]. Rippel and Adams first employed neural nets to calculate the parameters and since then NFs are generally understood as a framework that integrates neural nets for that reason. Rezende and Mohammed applied Normalising Flows to the domain of variational inference [RM16]. In 2018 Papamakarios wrote a paper, central to this work, which combines autoregression and Normalising flows [PPM18].
This work investigates the advantages and disadvantages of using Masked autoregressive Flows (MAF) to learn densities from different datasets. This includes finding a quality measure and setting up artificial distributions of various kinds and dimensions to experiment with as well as experiments with a real world dataset. Here the focus lies on creating new samples and whether they can be used to improve a classifier training process in some way.
Chapter 2 gives an introduction in density estimation, the general principles of Normalising Flows and Masked Autoregressive Flows in more detail. Chapter 3 is about how to apply MAFs for density estimation. The next

chapter 4 contains a variety of experiments that show the up and downsides of Masked Autoregressive Flows. Chapter 5 summarises the general findings.

Chapter 2

Preliminaries

This chapter gives an overview of common ways to realise density estimation, how to measure 'distance' of two distributions and introduces the concept of Normalising Flows in general and some specific variants. Afterwards it explains Masked Autoregressive Flows and its core principles in more detail, since it will be used in the next chapter. This work assumes the sample space is continuous and any distributions mentioned are D-dimensional or $\mathbb{R}^D$. You find the annotation used throughout this document in appendix 6.1.

2.1 Density Estimation

Density estimation is about resembling the probability function (PDF) of a random variable using samples only. The true PDF of the random variable is often referred to as $p(\boldsymbol{x})$. This is all about fitting some PDF function parameters in such a way that the likelihood of the training data is maximised. This function is referred to as $q(\boldsymbol{x})$. If $q(\boldsymbol{x})$ is known and fits 'well enough' to the the true distribution, $q(\boldsymbol{x})$ can substitute $p(\boldsymbol{x})$ in various tasks.

2.1.1 Methods

This section briefly discuss several approaches to solve density estimation. The key problem is that a continuous density function has to be constructed using a limited amount of samples only. But the density function must be valid everywhere in the feature space. So the space between the samples has to be 'filled' somehow.
Parametric models fit a distributions density function parameters to the data. You would estimate the mean and variance of a Gaussian distribution for example doing maximum likelihood optimisation. Parametric approaches, might not fit properly if the data distribution is not of the same kind. You can see how a single Gaussian is a bad fit for a multimodal

Gaussian distribution in figure 4.1. It has its mass in the centre where the multimodal one is near zero. Additionally, for real datasets, there often is insufficient knowledge about the distribution of the data which makes choosing a suitable prior density function even harder. A clue of how the data is roughly distributed is paramount here.

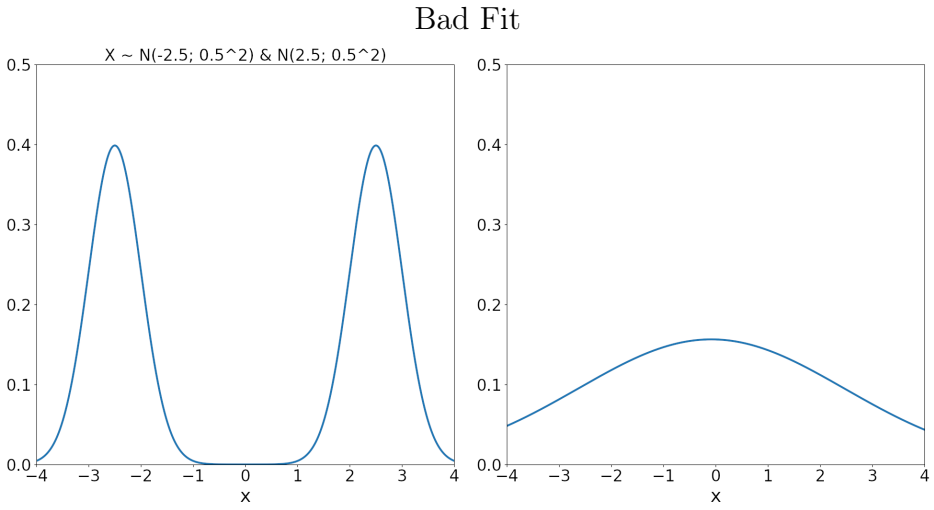


Figure 2.1: Left: original multimodal Gaussian distribution. Right: fitted unimodal Gaussian.

Non-parametric approaches, on the other hand, make no prior assumptions of the data. A straight forward and probably the simplest way of doing density estimation are histograms. The idea is to slice the feature space into equally large bins and simply count the amount of samples that fall into each bin. Let $b(\boldsymbol{x})$ be the function that returns the no of elements in the bin $\boldsymbol{x}$ falls in. The likelihood of $\boldsymbol{x}$, in case of N equally large bins is:

$$p(\boldsymbol{x}) = \frac{|b(\boldsymbol{x})|}{\sum_{i=0}^{N} |b_i|}$$

One issue with histograms is how to choose the amount of bins and how to choose a bins widths. There are numerous ways dealing with these problems. The quality of the density estimation greatly depends on the choice of these variables [Wik22b].
A more sophisticated approach is Kernel Density Estimation where a kernel function $k_i()$ is centred around each sample $\boldsymbol{x_i}$. The likelihood of a point $\boldsymbol{x}$ when there are N total samples:

$$p(\boldsymbol{x}) = \frac{1}{N} \sum_{i=0}^{N} k_i(\boldsymbol{x})$$

The quality of this estimator is governed by the bandwidth of the kernel. A too low or too high bandwidth leads over or under smoothing. Another important parameter is the choice of the kernel function. Popular kernels are: uniform, Gaussian and the Epanechnikov kernel. Both, histograms and KDE suffer from the curse of dimensionality, meaning that the numbers of samples required to obtain a reliable density estimator grows exponentially with the dimensionality of the data. This is because the room with a constant width grows with increasing dimensions, which lowers the sample density, which itself is just the sample over space ratio. To encounter this one could decrease the kernel band width with the increasing dimensionality. But this will eventually lead to under smoothing and eventually will reduce the probability mass towards the sample data points only [Wik22c].

2.1.2 Kullback-Leibler-Divergence

The KL-divergence $KL(P \parallel Q)$ is the statistical distance between the distributions P and Q. Its formula is this:

$$KL(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} \tag{2.1}$$

The KL-divergence returns a number greater or equal 0, with 0 meaning p and q being identical. However, it is not a distance in the usual sense, since a distance between two things is equal either direction. For divergences applies:

$$KL(P \parallel Q) \neq KL(Q \parallel P)$$

Yet, divergences tell whether distributions deviate from another. Since there is no way of calculating the integrals I employ a Monte Carlo estimation instead.

2.2 Normalising Flows

This section explains the framework of Normalising Flows in general and gives an overview of different variants and their properties.

2.2.1 Intuition

The core problem is learning is the unobservable distributions that created the actual dataset. Imagine a function that maps that intractable data distribution to a simple Gaussian or any other parameterised distribution that can easily be handled. Imagine that function could be inverted easily. That would enable us to calculate the likelihood of every data point and we could create new samples by sampling from the easy distribution and run the function in backward direction. The Normalising Flow framework

does exactly that. As mentioned, there are key elements that have to be understood to make this work. The successive sections provide an overview of the general principles of Normalising Flows, culprits and different ways to remedy these. You can make yourself familiar with the notation in the appendix 6.1.

2.2.2 Key Principles

Let's formalise the intuition. A Normalising Flow does two things: it transforms data points $\boldsymbol{x}$ to points sampled of a simple distribution $\boldsymbol{z} \sim q(\boldsymbol{z})$ and vice versa:

$$\boldsymbol{x} = g(\boldsymbol{z}) \text{ and } \boldsymbol{z} = f(\boldsymbol{x})$$

Since f is the inverse of g:

$$\boldsymbol{x} = g(f(\boldsymbol{x}))$$

There are some constraints though: f and g must be differentiable and $\boldsymbol{x}$ and $\boldsymbol{z}$ must be D-dimensional. So far, one point sampled from $p(\boldsymbol{x})$ was mapped onto a point under the distribution of $q(\boldsymbol{z})$. But what about the density at point $\boldsymbol{x}$? What cannot be used is $q(f(\boldsymbol{x}))$ as illustrated in figure 2.2 since the transformation has an effect on the likelihood. In the figure the solution is to multiply $q(f(\boldsymbol{x}))$ by $\frac{1}{2}$, a constant value. So a mechanism must exist that calculates this value instead.

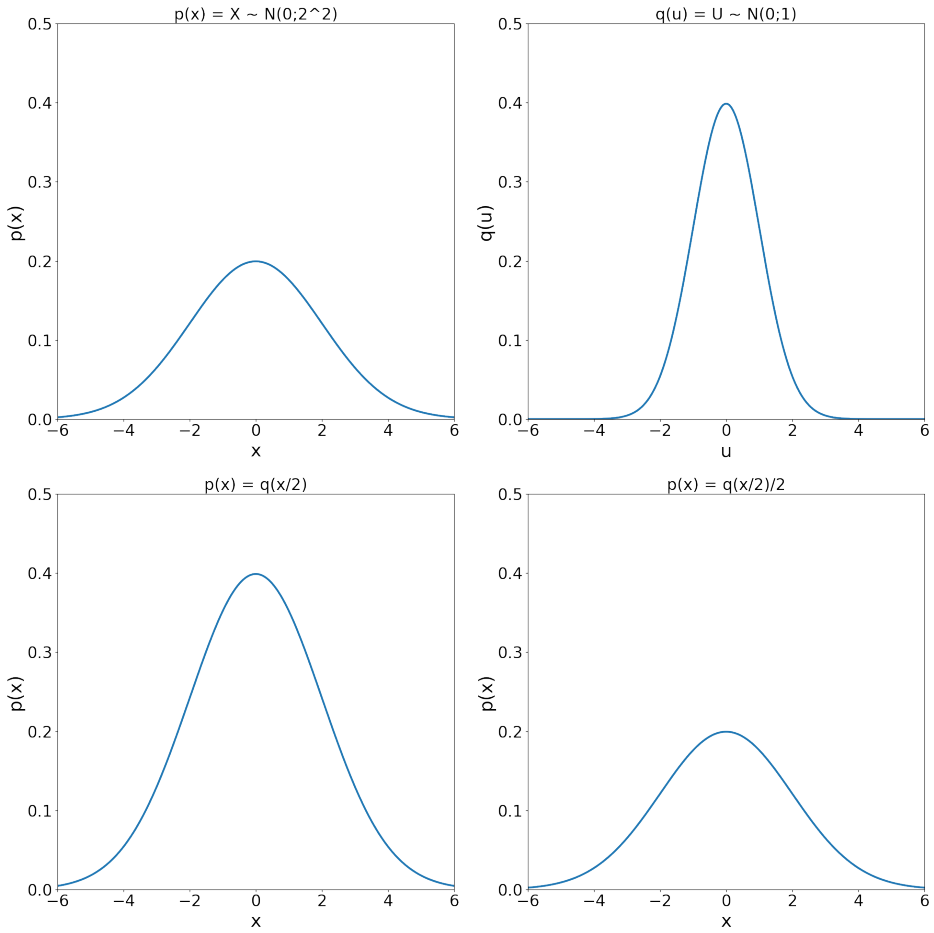


Figure 2.2: Top left: normal distribution $p(x)$, with variance of 4, that shall be transformed into the standard normal, $q(x)$ at the top right. Bottom left: using $q(\frac{x}{2})$ addresses the difference of the variance of p and q but it does not integrate to one anymore. Bottom right: fixing the higher volume by dividing by 2. Normalising Flows are a framework, that transforms data and undoes the change of volume for way more complex transformations than $\frac{x}{2}$ automatically.

Jacobian matrix and Determinant

'The Jacobian' as the Jacobian matrix is often called contains all first derivatives of a function. Let $f(\boldsymbol{x})$ be a continuously differentiable function that

can also be written:

$$f(\boldsymbol{x}) = \begin{bmatrix} f_1(x_1, x_2, ...x_n) \\ f_2(x_1, x_2, ...x_n) \\ ... \\ f_n(x_1, x_2, ...x_n) \end{bmatrix}$$

Then the Jacobian is defined as:

$$J_f(\boldsymbol{x}) = \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\boldsymbol{x}) & \dots & \frac{\partial f_1}{\partial x_n}(\boldsymbol{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(\boldsymbol{x}) & \dots & \frac{\partial f_n}{\partial x_n}(\boldsymbol{x}) \end{bmatrix}$$

The Jacobian can be thought of the the first derivative of f or to be more specific, the linear transformation that is closest to f in point $\boldsymbol{x}$. Additionally the determinant of J tells how the volume is changed by the linear transformation in $\boldsymbol{x}$ and therefore in $f(\boldsymbol{x})$. A Jacobian matrix is not necessarily an $n \times n$ matrix. But since Normalising Flows are bijective, both f and $\boldsymbol{x}$ are n-dimensional, the other cases can be put aside and looked at $n \times n$ matrices only. In the literature the determinant is sometimes called 'the Jacobian' too.
The normalising flow assumes that a data point springs into existence being sampled from a simple distribution $q(\boldsymbol{z})$, like a Gaussian, and then transformed into a point of the distribution the flow shall model: $p(\boldsymbol{x})$. Keep in mind that for any invertible matrix this applies:

$$p(\boldsymbol{x}) = q(f(\boldsymbol{x})) \left| \det \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \right|^{-1} = q(f(\boldsymbol{x})) \left| \det \frac{\partial g(\boldsymbol{x})}{\partial \boldsymbol{x}} \right| \tag{2.2}$$

and $p(\boldsymbol{x})$ can also be expressed in terms of the base distribution $q(\boldsymbol{z})$:

$$p(\boldsymbol{x}) = q(\boldsymbol{z}) |\det J_g(\boldsymbol{z})|^{-1} \quad \text{with } \boldsymbol{z} \sim Q(Z) \tag{2.3}$$

The equation transforms $\boldsymbol{z}$ and reverses its local volume change in $\boldsymbol{z}$ and hence the 'normalising' in Normalising Flow. Also note that this calculates the absolute value of the Jacobian since only volume preservation is important but not the change of direction of f and g.

Composition of multiple Layers

The expressiveness of Normalising Flows can be increased building deep models. Fortunately the composition of bijective functions yields another bijective function. Imagine a function f consisting of a composition of f_1 and f_2. Inversion and Jacobian determinant are straight forward:

$$(f_2 \circ f_1)^{-1} = f_2^{-1} \circ f_1^{-1}$$

$$\det J_{f_2 \circ f_2}(\boldsymbol{u}) = \det J_{f_2}(f_1(\boldsymbol{u})) \cdot \det J_{f_1}(\boldsymbol{u})$$

When extended to K layers the formula for the determinant is this:

$$\det J_f(\boldsymbol{x}) = \prod_{i=1}^{K} \det J_{f_i}(\boldsymbol{u}_i)$$

$\boldsymbol{u}_i$ results from applying the ith flow on its predecessors output starting with $\boldsymbol{u}_1 = \boldsymbol{x}$.
As you can see the transformation, inversion and the Jacobian determinant can be calculated for each layer individually. Hence the complexity regarding the amount of layers K is $\mathcal{O}(K)$.

KL divergence and maximum likelihood

Let $p(\boldsymbol{x})$ be the target or data distribution, $m(\boldsymbol{x};\boldsymbol{\theta})$ a Normalising Flow with parameters $\boldsymbol{\theta} = \{\boldsymbol{\phi}, \boldsymbol{\psi}\}$ and $\{\boldsymbol{x}_n\}_{n=1}^{N}$ be a dataset with N samples. Recall equation 2.1. It can be rewritten as:

$$\begin{aligned} KL(p(\boldsymbol{x}) \parallel Q) &= \mathbb{E}_p \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} \\ &= \mathbb{E}_p(\log p(\boldsymbol{x}) - \log q(\boldsymbol{x})) \\ &= -\mathbb{E}_p \log q(\boldsymbol{x}) + \mathbb{E}_p \log p(\boldsymbol{x}) \end{aligned} \tag{2.4}$$

In this case $\mathbb{E}_p \log p(\boldsymbol{x})$ is constant. Now Q is replaced with the model. Then the KL divergence between model and target distribution is:

$$\begin{aligned} KL[p(\boldsymbol{x}) \parallel m(\boldsymbol{x};\boldsymbol{\theta})] &= -\mathbb{E}_{p(\boldsymbol{x})}[\log m(\boldsymbol{x};\boldsymbol{\theta})] + \text{const} \\ &= -\mathbb{E}_{p(\boldsymbol{x})}[\log q(f(\boldsymbol{x};\boldsymbol{\phi});\boldsymbol{\psi}) + \log|\det J_f(\boldsymbol{x};\boldsymbol{\phi})|] + \text{const} \end{aligned} \tag{2.5}$$

Despite the target density $p(\boldsymbol{x})$ being inaccessible, an estimate can be made by Monte Carlo, because $\boldsymbol{x}$ is sampled from $p(\boldsymbol{x})$:

$$KL[p(\boldsymbol{x}) \parallel m(\boldsymbol{x};\boldsymbol{\theta})] \approx -\frac{1}{N}\sum_{n=1}^{N} \log q(f(\boldsymbol{x}_n;\boldsymbol{\phi});\boldsymbol{\psi}) + \log|\det J_f(\boldsymbol{x}_n;\boldsymbol{\phi})| + \text{const} \tag{2.6}$$

Which is very similar to the negative log likelihood used as the loss function to train the flow:

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{n=1}^{N} \log q(f(\boldsymbol{x}_n;\boldsymbol{\phi});\boldsymbol{\psi}) + \log|\det J_f(\boldsymbol{x}_n;\boldsymbol{\phi})| \tag{2.7}$$

In fact the only difference is the added constant in the loss function, meaning that maximising the likelihood is equivalent to minimising the KL divergence.

Interpretation

Equation 2.3 expresses a normalising flow as a transformation from the data distribution onto a base distribution, often a standard normal. Usually normalising refers to multiplying by some constant. In the Normalising Flow framework this is not a constant but the Jacobian determinant and therefore a function of the input. This transformation can, for example, translate, rotate and scale. It is not limited in any way other than it has to be bijective and differentiable. The absolute Jacobian expresses how much the transformation expands or contracts the space around a certain point. If the input is spread to a larger area, the Jacobian determinant is greater than 1. Accordingly it is smaller than one when the transformation contracts the space. The Jacobian is the key why a Normalising flow does not simply map all data points close to 0 in standard normal distribution space. In this case the Jacobian would decrease the likelihoods of all points according the the compression of the transformation. It punishes this approach.
You may also think of a NF as a distribution that can be trained and used like one. But it comes with some limitations such that it does not provide a cumulative density function.

Complexity

Regarding computation complexity the functions have to be easily reversible and constructed in such a way that its determinant can be computed quickly. Generally the complexity of inverting a $n \times n$ matrix as well as calculating the determinant is considered $\mathcal{O}(n^3)$. The Strassen algorithm, which is the fasted known practical one, reaches a complexity of $\mathcal{O}(n^{2.807})$ [Str69]. Currently, the algorithm with the lowest complexity reaches $O(n^{2.373})$, though it is a galactic algorithm, meaning it can only play out it's advantage when n is way too high for any real world problems ,[Wik22a].
Currently the lowest known complexity calculating the determinant of any reversible $n \times n$ matrix is $O(n^{2.3728596})$ using the fast matrix multiplication algorithm [AH74].
Since the complexity for both operations, inversion and determinant are greater than $\mathcal{O}(n^2)$ runtime will be seriously affected in higher dimensions. To mitigate this the transformations must be constructed in a way that allows faster computation.

Expressiveness

This shows that a normalising flow, with a simple base distribution $q(\boldsymbol{z})$, can represent any distribution $p(\boldsymbol{x})$. It follows the arguments of Papamakarios et al [PNR+21] who base their proof on the works of Hyvärinen [HP99] and more formal by Bogachev [BKM07]. The proof shows that there exists a diffeomorphism, meaning a smooth and invertible mapping, for each target

distribution $p(\boldsymbol{x})$ and base distribution $q(\boldsymbol{z})$ under certain circumstances. The first assumption is: $p(\boldsymbol{x}) > 0 \; \forall \boldsymbol{x} \in \mathbb{R}^D$. Secondly $P(v_d \leq x_d, |\boldsymbol{x}_{<d})$ is differentiable with respect to x_d and $\boldsymbol{x}_{<d}$. Remember that any distribution $p(\boldsymbol{x})$ can be rewritten as:

$$p(\boldsymbol{x}) = \prod_{d=1}^{D} p(x_d|\boldsymbol{x_{<d}}).$$

Next the *cumulative distribution function* comes into play:

$$u_d = F(x_d, \boldsymbol{x}_{<d}) = \int_{-\infty}^{x_d} p(v_d|\boldsymbol{x}_{<d}) dv_d = P(v_d \leq x_d|\boldsymbol{x_{<d}}).$$

This ensures the codomain of F is between 0 and 1:

$$F : \boldsymbol{x} \mapsto \boldsymbol{u} \in (0,1)^D$$

Alternatively you can think of F mapping $\boldsymbol{x}$ onto the D-dimensional unit cube. The Jacobian is a lower triangular matrix so its determinant is the product of the diagonal elements:

$$\det J_F(\boldsymbol{x}) = \prod_{d=1}^{D} \frac{\partial F_d}{\partial x_d} = \prod_{d=1}^{D} p(x_d|\boldsymbol{x}_{<d}) = p(\boldsymbol{x}) \tag{2.8}$$

Since the constrain of $p(\boldsymbol{x}) > 0 \quad \forall \boldsymbol{x} \in \mathbb{R}^D$ the Jacobian determinant is non-zero everywhere, hence F is invertible and a diffeomorphism. Rearranging equation 2.2 to $p(\boldsymbol{z})$ using change of variables results in:

$$q(\boldsymbol{u}) = p(\boldsymbol{x}) \left|\det J_F(\boldsymbol{x})\right|^{-1}$$

With equation 2.8 this leads to:

$$q(\boldsymbol{u}) = p(\boldsymbol{x}) \left|p(\boldsymbol{x})\right|^{-1} = 1$$

This means that $\boldsymbol{z}$ is distributed uniformly in the unit cube. Therefore a Normalising Flow, even when restricted to a uniform base distribution, can learn any other distribution. Furthermore $q(\boldsymbol{u})$ can be projected on another distribution that fulfil the same conditions as $p(\boldsymbol{x})$. The transformation from $q(\boldsymbol{z})$ to $\boldsymbol{u}$ is defined as followed:

$$u_d = H(z_d, \boldsymbol{z}_{<d}) = \int_{-\infty}^{u_d} q_z(v_d|\boldsymbol{z}_{<d}) dv_d = P(v_d \leq z_d|\boldsymbol{z}_{<d})$$

H is a diffeomorphism for the same reasons as F. Consequently a Normalising Flow with the transformation $H \circ F$ maps $p(\boldsymbol{x})$ onto $q(\boldsymbol{z})$.

2.2.3 Various Normalising Flows

This section introduces some Normalising Flow architectures that share one or more aspects with Masked Autoregressive Flows of the next chapter.

Linear Flows

Linear Flows are transformations of the following type:

$$\boldsymbol{u} = f(\boldsymbol{x}) = \boldsymbol{M}\boldsymbol{x} + \boldsymbol{b}$$

where b is a D-dimensional vector and M is an invertible $D \times D$-matrix. The Jacobian determinant of f is just the $\det M$, the translation of $\boldsymbol{b}$ does not inflict a change of volume. Both, the time complexity calculating the determinant as well as the inverse are considered $\mathcal{O}(D^3)$. This makes it unsuitable for larger D.
Diagonal M In case $\boldsymbol{M}$ is diagonal, meaning the only nonzero elements reside on the diagonal, its determinant is just the product of the diagonal and the inverse can be calculated in linear time. The major downside is the limited expressiveness because it cannot model any correlations between dimensions. Triangular M A triangular matrix has the advantage of having a quick to compute determinant, which is, again, the product of the diagonal. Inversion is $\mathcal{O}(D^2)$ using back-substitution. The expressiveness of a triangular transformation is affected by the ordering of the dimensions. Reordering can be done using a permutation matrix which is the D-dimensional unitary matrix with its columns shuffled. Hence its absolute determinant is 1.

Coupling Layers

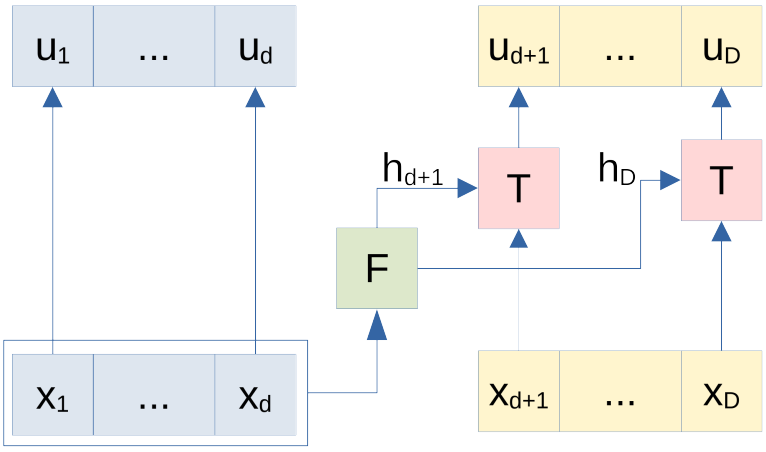


Figure 2.3: Schema of a coupling layer. T is a transformer that takes an x and an h and produces a u. The first d x stay unchanged.

The core idea is to transform the data partially with a transforming function T. The input $\boldsymbol{x}$ is split at d which commonly is $D/2$. Image 2.3 shows a common implementation of a coupling layer. The parameters of it are $\boldsymbol{h_1}...\boldsymbol{h_D}$ though here $\boldsymbol{h_1}...\boldsymbol{h_d}$ are left out and consider the transformation of $x_1 \dots x_d$ being the identity function. $x_1...x_d$ may be transformed in another way, but in this case $\boldsymbol{h_1}...\boldsymbol{h_d}$ must be constant. For more clarity and because this being uncommon, it is left out. F can be any approximator like a neural net. So the overall transformation of x is this:

$$\begin{aligned} u_1 \dot{u}_d &= x_1 \dots x_d \\ \boldsymbol{h_{d+1}}...\boldsymbol{h_D} &= F(x_1 \dots x_d) \\ u_{d+1} \dots u_D &= T(\boldsymbol{x_i}; \boldsymbol{h_i}) \text{ for } i > d \end{aligned} \tag{2.9}$$

As a result the Jacobian matrix looks like this:

$$J_f(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{A} & \boldsymbol{D} \end{bmatrix}$$

The upper left part $\boldsymbol{I}$, a $d \times d$-matrix, is a result of copying $x_1 \dots x_d$. $\boldsymbol{A}$ is a full $(D-d) \times d$-matrix resulting from the transformation of $x_{d+1} \dots x_D$ depending on $x_1 \dots x_d$. $\boldsymbol{D}$ is a diagonal $d \times d$-matrix since $x_{d+1} \dots x_D$ do not influence each other. All in all this leads to a lower triangular matrix and a simple determinant: the product of the diagonal.
Coupling layers are related to autoregression in the next chapter.

2.3 Masked Autoregressive Flow

This section explains the autoregressive property, autoregressive Normalising Flows and a specific implementation. In their 2015 paper 'MADE' Germain et all describe how to turn an autoencoder into an autoregressive neural net [GGML15]. Papamakarios et al integrated MADE into a Normalising Flow in their 2018 paper 'Masked Autoregressive Flow for Density Estimation' [PPM18].

2.3.1 Autoregression

Commonly autoregression is known from time series, where one might predict something at time step $t+1$ given data about time steps 1 to t or similar. Nevertheless, autoregression in Normalising Flows in general is not about time series. Consider $\boldsymbol{z} = f(\boldsymbol{x})$. f is autoregressive, sometimes referred to as having the autoregressive property, when every z_d can be calculated depending on $\boldsymbol{x}_{<d}$ only for all $d \in D$.

2.3.2 MADE: Masked Autoencoder for Distribution Estimation

MADE is a way to estimate the parameters of a distribution using neural nets. Like its name suggests it is autoencoder-like, though some modifications have been made. A joint distribtuion $p(\boldsymbol{x})$ can be decomposed into conditionals using the chain rule:

$$p(\boldsymbol{x}) = \prod_{d=1}^{D} p(x_d|x_{<d}) \tag{2.10}$$

Germain et al modify the autoencoder such that it outputs the parameters for the base distribution,e.g a Gaussian, for a certain $p(x_d|\boldsymbol{x_{<d}})$ but not the likelihood itself. Multiplying the neural nets weight matrices with binary matrices ensures that the dth output only depends on the inputs from 1 to $d-1$. This way the parameters can be estimated in one forward pass through the neural net instead of D passes of previous works [GGML15].

Masking

Consider

$$\begin{aligned} h(\boldsymbol{x}) &= act(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{x}) \\ \hat{\boldsymbol{x}} &= act(\boldsymbol{c} + \boldsymbol{V}h(\boldsymbol{x})) \end{aligned} \tag{2.11}$$

being a single hidden layer autoencoder. act is an activation function, $\boldsymbol{W}$ and $\boldsymbol{V}$ are weight matrices and $\boldsymbol{b}$ and $\boldsymbol{c}$ are biases. $h(\boldsymbol{x})$ is the hidden representation and $\hat{\boldsymbol{x}}$ the reconstructed data.
One can be eliminate computational paths from x_d to $\boldsymbol{x}_{>d}$ by adding an elementwise multiplication of the weight matrices $\boldsymbol{W}$ and $\boldsymbol{V}$ with masks $\boldsymbol{M}^W$ and $\boldsymbol{M}^V$. The mask matrices are set to 0 wherever a connection must be eliminated:

$$\begin{aligned} h(\boldsymbol{x}) &= act(\boldsymbol{b} + (\boldsymbol{W} \odot \boldsymbol{M^W})\boldsymbol{x}) \\ \hat{\boldsymbol{x}} &= act(\boldsymbol{c} + (\boldsymbol{V} \odot \boldsymbol{M^V})h(\boldsymbol{x})) \end{aligned} \tag{2.12}$$

Constructing the masks First a unique number is sampled from $[1..D]$ and assigned to every input node. Think of it as an input id. The outputs get the same IDs in the same order.
Each hidden unit is assigned a number m between 1 and $D-1$. m denotes the highest input id a node is connected to. Let $m^l(k)$ be the m assigned to the k-th node in the l-th layer and l_0 the input layer.
The connectivity matrix is set to 1 for each hidden layer where

$$m^l(k) >= m^{l-1}(k')$$

, otherwise 0.

The output layer is treated differently. Here the node with the lowest id has no incoming connection whatsoever, so it resembles a density of a single variable e.g. $p(x_2)$. The other output nodes have incoming connections with all nodes of the previous layer that have lower IDs. That leads to the input and output layer containing one node that is not connected to the rest of the network. In case of one dimensional data there will be no connection to the input at all and the network will return a constant value. A visual representation is given in figure 2.4.

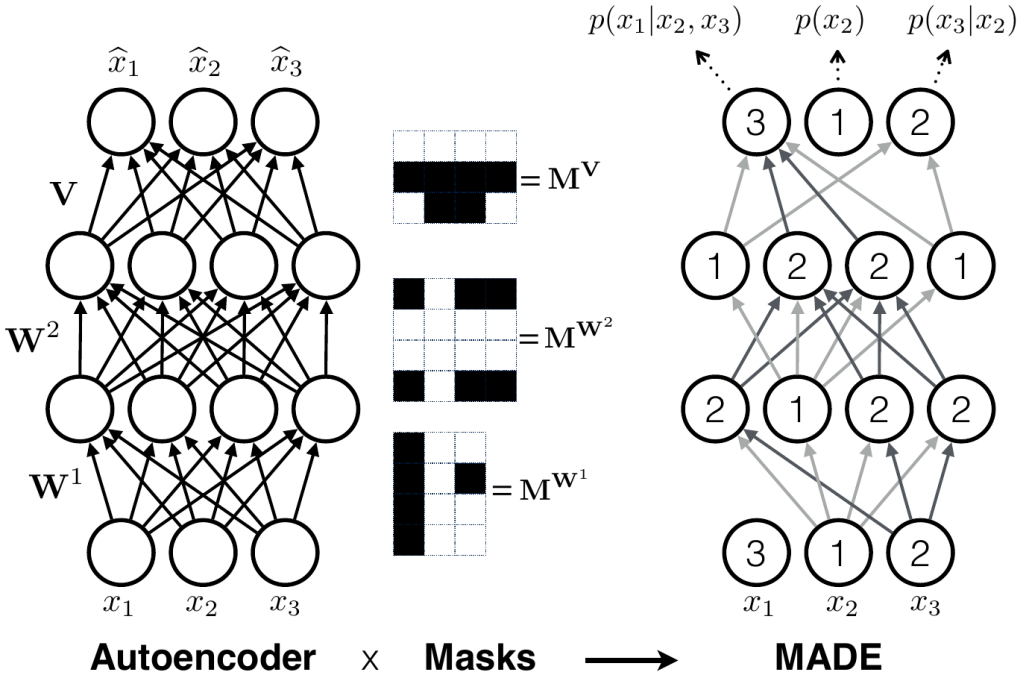


Figure 2.4: From [GGML15], Figure 1. On the left: autoencoder that reconstructs the input $\boldsymbol{x}$ as $\hat{\boldsymbol{x}}$. When the masks are applied, as depicted on the right side, one can interpret the output as the parameters of the decomposed conditional probability of $p(\boldsymbol{x})$. Note that both the input and output layer contain one node that is not connected. Thus, a one-dimensional MADE has no connections between input and output at all and always returns constant values for any input.

2.3.3 Autoregressive Normalising Flows

In the context of Normalising Flows autoregression in implemented as shown in 2.5. Here c_d is the conditioner that produces the parameters h_d for the transformer T. T then transforms x_d to u_d. The conditioner can be any function such as a neural net and must only depend on the inputs $x_1 \dots x_{d-1}$. It is therefore autoregressive. The transformer is a monotonous

function of x_d, parameterised by h_d. The main advantage of c being autoregressive is the resulting Jacobian matrix of T being triangular, meaning its determinant is the product of the diagonal which is quick to compute. On the other hand the sampling direction becomes more expensive.

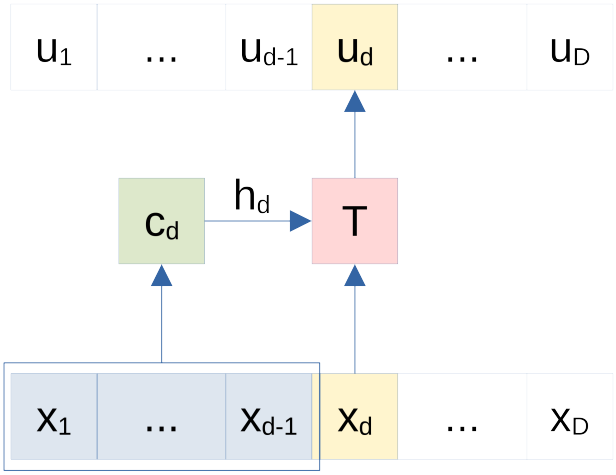


Figure 2.5: Forward pass of a d-dimensional, one layer, autoregressive flow. The conditioner c_d is autoregressive. There is no computational path from any u_d to any $x_{\geq d}$ which leads to a triangular Jacobian matrix of T.

2.3.4 Masked Autoregressive Flow

A Masked Autoregressive Flow is a Normalising Flow that applies an affine shift and scale transformation on a data point. To do so it employs a MADE, that calculates the parameters of the transformation based on the input data point. At the very end of the MAF is a base distribution that often is a multivariate standard normal distribution. The likelihood of the input data point is the likelihood of the transformed data point under the base distribution. Since the autoregressive MADE governs the scale and shift transformation the MAF inherits its autoregressive property.

2.3.5 Scheme of a Masked Autoregressive Flow

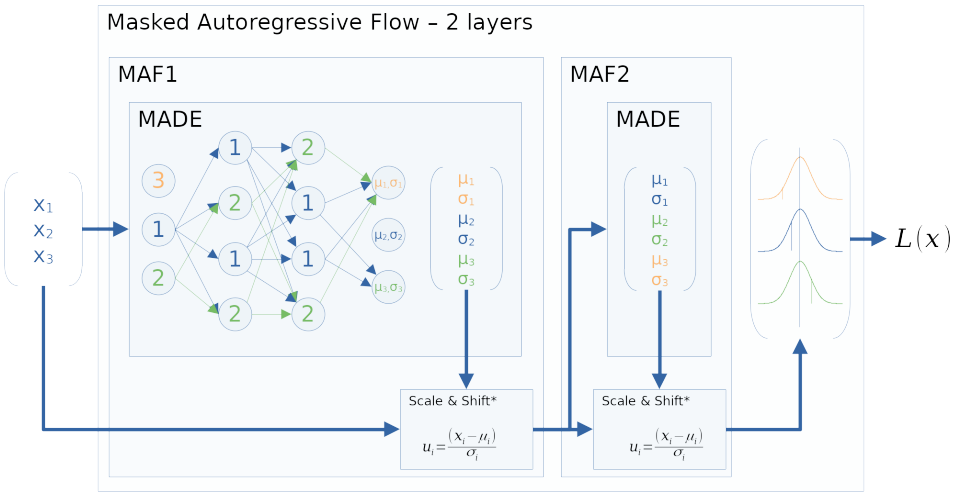


Figure 2.6: Schema of how the input data $\boldsymbol{x}$ is pushed through a 2-layer-MAF to calculate a likelihood $L(\boldsymbol{x})$ at the end. Note that neither normalisation nor permutation are displayed here.

2.6 shows a Masked Autoregressive Flow consisting of two MAF-layers. In the context of 2.5 the all conditioners c_d are realised by a single MADE. The scale and shift transformation equals T. A special case is $D = 1$ in which case no connection to the input is made and the scale and shift is fixed for all possible input values. This effectively reduces the whole Flow to a one dimensional MADE which returns constant values. Therefore the whole MAF fits a simple Gaussian bump when it is one dimensional. The MAF layers can be chained together and may possibly be augmented with batch normalisation layers as well.

2.3.6 Complexity

The likelihood of a sample can be computed in one pass or $\mathcal{O}(1)$. In contrast sampling is expensive. Sampling starts with a d-dimensional vector u drawn from the multivariate base distribution. Since sampling must not be connected with any input data the only thing left to do is sampling μ_b and σ_b from the last MADE layer where b is the index of the output node, thus constant. Now x_b can be calculated by reversing the shift and scale operation:

$$x_b = u_b * \sigma_b + \mu_b$$

This calculates a value for the bth dimension of $\boldsymbol{x}$. Now the next μ and σ can be sampled for the dimension that only depends on x_b. Therefore each

MADE layer requires D passes to obtain one sample compared to one in the density direction and the complexity is $\mathcal{O}(D)$.

2.3.7 Limitations

A MAF like all Normalising Flows can be regarded like any other parameterised distribution except there is no easy way to compute the cumulative distribution function. This is due to the calculation of the Jacobian determinant which 'corrects' the probability which is a non-linearity and point wise. So there is not way to integrate over a continuous range of values.
The expressivity of a single layer MAF is limited. When the base distribution is a Gaussian every z_i that is conditioned on $\boldsymbol{z_{<i}}$ will be Gaussian as well. A visual example is given in 4.1 . This limitation can be overcome by composing multiple MAF layers though it is currently unknown if affine autoregressive flows can represent any distribution [PNR+21].

Chapter 3

Applicability of MAFs to Density Estimation

This chapter explains what kind of data can be learned using MAFs and what steps have to be made. Keep in mind that this is all about a continuous sampling space. Normalising Flows learn the joint distribution of the input data. Masked Autoregressive Flows are universal approximators under the assumptions of section 2.2.2 [PNR+21]. Thus, a MAF is a good choice when there is no further knowledge about the target distribution. Furthermore MAFs are quick calculating densities but relatively slow generating new samples.
Before a MAF can be used for density estimation it must be trained. Thus, the data has to be set up the right way. The dataset of the target distribution should be normalised and split into training, validation and a test dataset.
A couple of choices, regarding the MAF, have to be made as well. As mentioned one of the most important parameters of a MAF is the amount of layers being used because it directly influences the expressiveness of the MAF [PNR+21]. Increasing the amount of layers also increases the amount of data needed to fit all the internal weights though. Each MAF layer contains a MADE that has its own parameters as well. The amount of dense layers, their activation function and how many nodes per per layer are some available parameters.
When training is done a MAF can calculate the densities of any valid input whether seen before or not. It is, however, not possible to calculate probabilities with MAFs since there is no way of calculating integrals over the density function a MAF learned.
Papamakarios et al compared the performance of various Normalising Flows, including MAFs on several datasets, with a varying amount of layers. They used high dimensional sensor data as well as images and recorded the average achieved log likelihoods. They found that MAFs mostly outperform MADEs and always outperform Real NVPs. They conclude that the optimal

choice, of the kind of model, as well as the amount of layers used, is dataset dependent [PPM18].

3.1 Generation

The next best thing one could do with a MAF, assuming runtime is not critical, is generating new data. Papamakarios et al [PPM18] use MAFs to learn MNIST and CIFAR-10, both image datasets, and generate new samples. According to their conclusion MAFs lack fidelity compared to other image based generative models like Real NVP.

Chapter 4

Evaluation

The goal is to make one or more MAFs learn a real world dataset and to measure how well it resembles it. A drawback of learning real world data is that divergences do not apply here because the lack of a density function. As a substitute to measure the MAFs performance, classifiers are trained for that real world dataset with varying amounts of original samples and those from MAFs. This will answer whether one can improve an existing classifier with additional synthetic samples. As Papamakarios et al state, the main factor that sets the expressivity of a MAF is the amount of layers it consists of. To narrow this parameter down several experiments, with artificial distributions, that aim finding out what number of layers to choose for the real world dataset are conducted. An important factor that must be determined is the amount of layers regarding the input dimensions. This is obviously the case since a one dimensional MAF can not be more expressive than a single Gaussian. A few one and two dimensional experiments are conducted anyway to build an intuition of their functionality. It may also be of interest, how the ratio of input dimensions to layers required to reach highest performances changes when adding more dimensions. Finally the real world experiments are run with the input dimension to layer ratio that worked best in the higher dimensional, artificial experiments. An overview of the higher dimensional experiments can be found in table 4.4.
This work uses the Tensorflow probability framework, which offers a huge toolbox for various tasks around probability theory. It offers basic distributions like multivariate Gaussians. However, multi modal distributions had to be implemented by wrapping those. It also provides a framework for Normalising Flows as well as autoregression both of which are used here.
The source code is available here:

- `https://git.informatik.uni-rostock.de/dd185/normalising_flows`
- `https://github.com/danielsGitStuff/normalising_flows`

4.1 Parameters

Some parameters of a Masked autoregressive Flow as worth mentioning though most are common in everyday neural networks. 'Hidden Shape' refers to the amount of layers and nodes for each MADE. '[200, 200]' means two layers with 200 nodes each. 'Activation' is the activation function used by those nodes. The polynomial decay function is provided by Keras.

Parameter	***Value***
Activation	ReLU
Batch Size	1024
Epochs	2000
Hidden Shape	[200, 200]
Learning Rate	PolyDecay($1e^{-3}$, $1e^{-4}$)
Optimiser	Adam
Patience	50
Samples	30720
ValSamples	2048

Table 4.1: Parameters for all MAFs used in the experiments if not mentioned otherwise. 'Layers' is not listed here, because it is varied.

4.2 Learning artificial distributions

To gain insight of the abilities of MAFs several multimodal, multivariate distributions are created. One kind is composed of uniform distributions, despite their appearance in the wild being very unlikely. The other kind is composed of Gaussians. Learning artificial distributions shall show practical issues in general, and those that depend on the amount of dimensions of the feature space.

4.2.1 One dimensional distributions

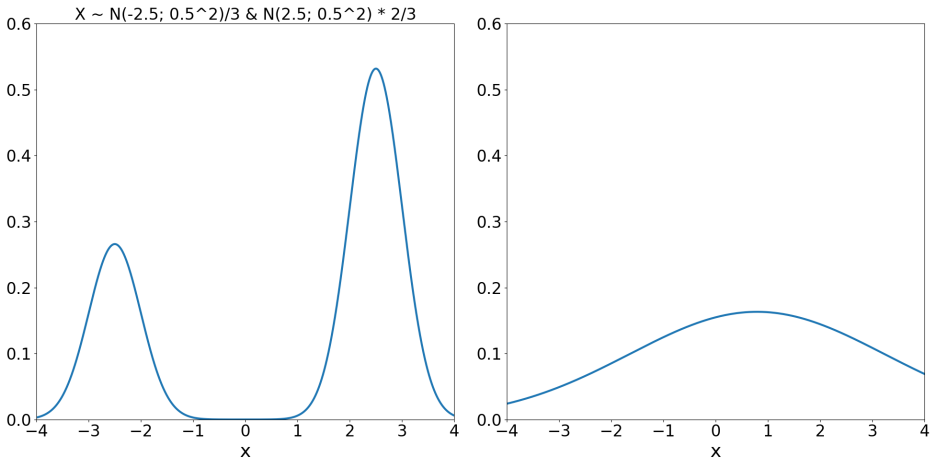


Figure 4.1: Likelihoods of the original, normal distribution on the left and a three layer MAF that learned the distribution.

In theory a MAF should only fits its base distribution to the data since the scale and shift parameters from the neural net are constant cause they are disconnected from the input. 4.1 shows two Gaussian bumps and the MAF did indeed fit a single Gaussian to the data independently of the amount of layers. There is no practical use employing a MAF for one dimension, since one could just fit a parameterised distribution directly.

4.2.2 Two dimensional distributions

Although Normalising Flows are capable of dealing with hundreds of dimensions, restricting it to two dimensions allows us to visualise the functionality of it. The original distributions are multimodal Gaussians or uniform. I arranged them in different ways, even randomly and visualised their density and those of the MAF with heat maps. Every MAF is fed with 30720 data points, 2048 validation points and can take up to 2000 epochs with a batch size of 1024. Training stops when the validation loss has not improved for 50 epochs. To evaluate whether progress in learning translates to an actual improvement in learning the original distribution, the KL-divergence between these two is calculated every 10 epochs. The first distribution I produced is that of figure 4.2. It shows a very basic example of a multimodal distribution of two Gaussian bumps with some distance between them. Intuitively the top right image shows how a 1D Gaussian parallel to the x-axis wanders from top to bottom, shifting its mean according to its y value. Both bumps stay connected by a ‘thread‘ though adding more layers remedies this as you can see in the latter two maps. This is interesting because the literature

about it does not mention this ‘gap issue‘ effect in particular. This made the author curious whether this might have serious effects on other scenarios. That’s why more distributions, with no or very low density between the modals, are created. This is one of the experiments with the lowest KL divergence and more layers seem to improve it further as you can see in table 4.2, which shows the performance, measured in KL divergence, for all conducted experiments. An overview of all two dimensional distributions used in experiments can be found in appendix 6.3. Additionally appendix 6.2 visualises the in-between steps of a transformation of a two layer MAF on it’s input.

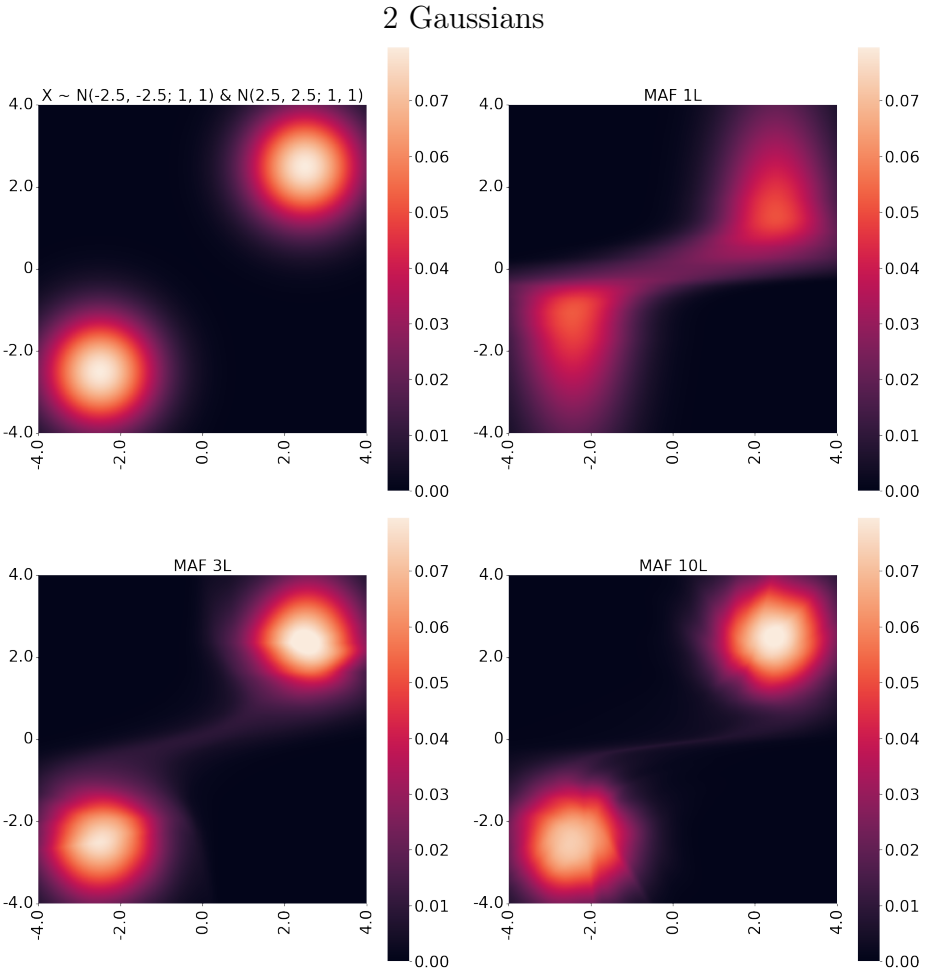


Figure 4.2: Clockwise: density of original distribution, 1-, 10-, 3-layer MAF.

As you can see in 4.3 a MAF is able to learn a uniform distribution albeit it requires 10 layers. Appendix 6.2 shows the in-between steps of trans-

formation the two layer MAF performs on its input. The divergence also decreases with each layer added. A uniform distribution does not satisfy $p(\boldsymbol{x}) > 0 \quad \forall \boldsymbol{x}$, mentioned in the section about universal expressiveness 2.2.2. However a MAF can still learn this distribution perfectly, because, practically a Normalising Flow does not learn all possible $\boldsymbol{x}$ but just the sampled 'region'. Within this region $p(\boldsymbol{x}) > 0$ is always satisfied for a single uniform distribution.

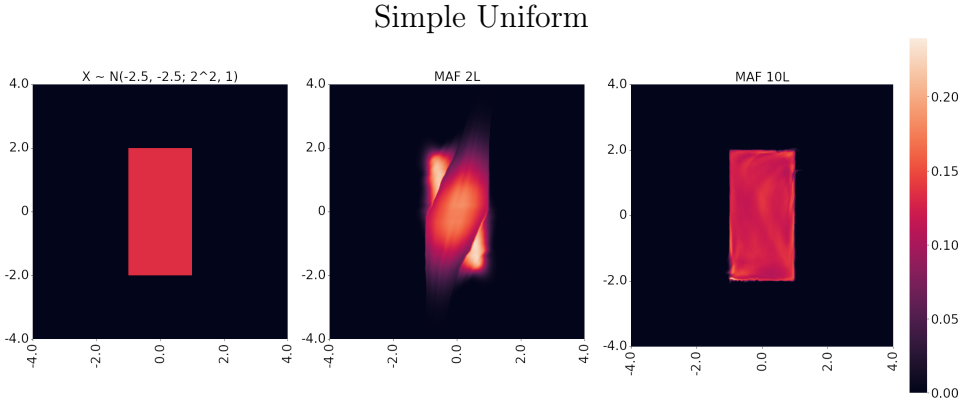


Figure 4.3: Likelihoods of the original, uniform distribution on the left. Middle: 2-layer MAF, right: 10-layer MAF.

The next experiment intends to escalate the 'gap issue' that appeared in '2 Gaussians' before. The target distribution is composed of 10 Gaussians that do not overlap and are spread a a larger area. Divergence is highest except for the one layer configuration throughout all experiments. Visual inspection of the density maps in figure 4.4 reveals the 'threads' got worse, often like smearing. The five and ten layer density maps, subjectively , seem to be very similar. And in fact they are, their KL divergence found to differ only by $1e-4$. The models with the lowest achieved divergences are displayed here.

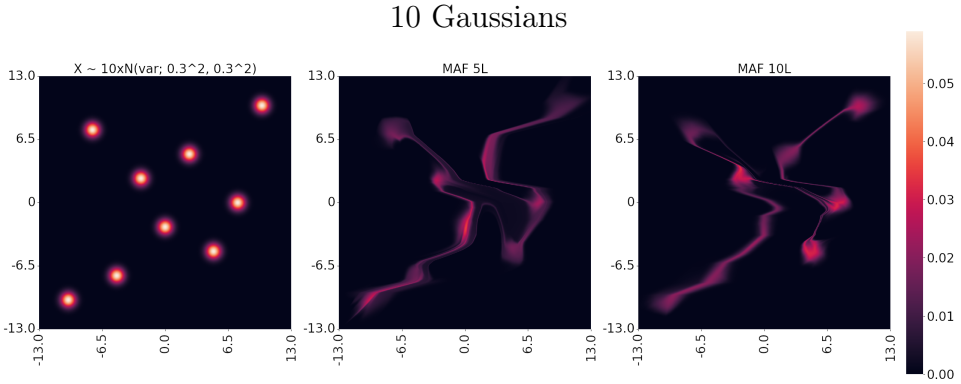


Figure 4.4: Visual comparison of 10 spatial separated Gaussian bumps and their learned representations with 5 and 10 layers.

But modals are not always far apart. That's why the experiment in figure 4.5 shows seven random Gaussians that are relatively close together and visually overlap. Just by looking at it there is not much of a difference between 3, 5 or 10 layers but the divergences decreases to one of the lowest levels throughout the experiments.

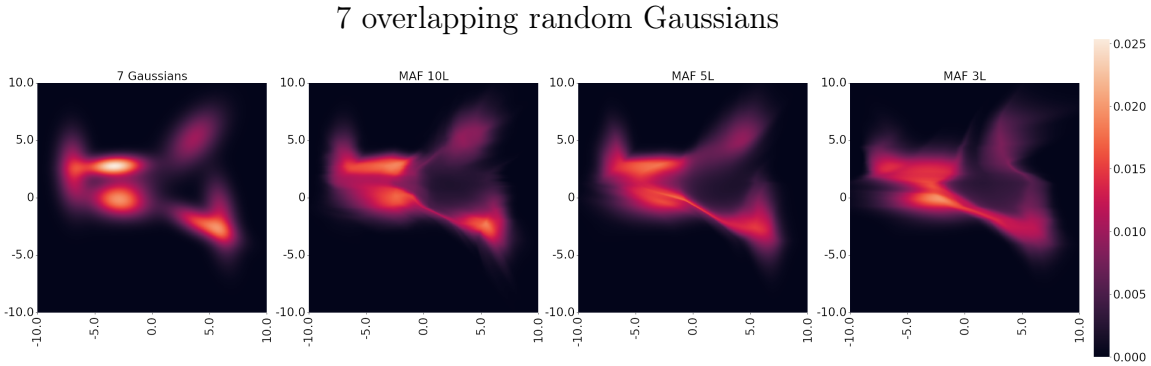


Figure 4.5: LTR: Original distribution, 10, 5 and 3 layer MAFs.

Average KL of 4 runs per experiment and no of layers, $2D$

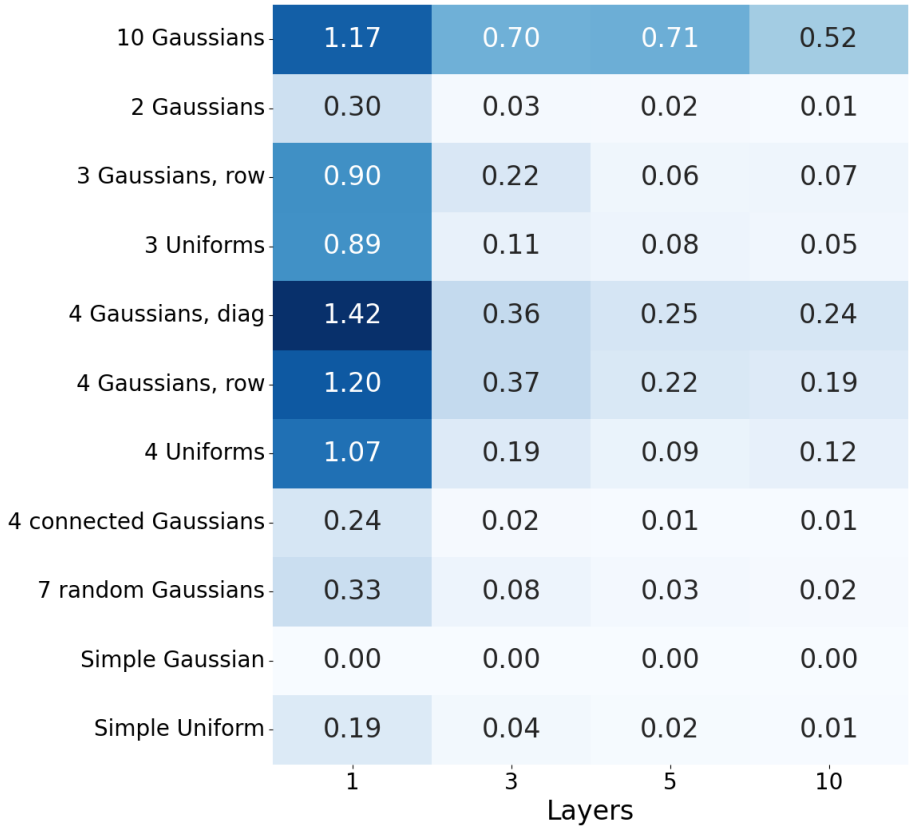


Table 4.2: The names of the experiments are to the very left. The two uni modal distributions start with 'Simple'. You can find all visual representations of the distributions in appendix 6.3

Findings

Given enough layers the divergence in case of the uni modal distributions almost drops to zero. No matter whether a uniform nor normal distribution had to be learned. In all cases the divergence decreases with more layers. There is likely a break even point but it was not intended to determine it. Multi modal distributions seem harder to learn. As you can see in table 4.2 their KL divergence does not drop as fast and far when adding more layers. In case of '4 Gaussians, diag' it even increases with ten layers. A visual inspection of the density map reveals that the modals are connected by thin 'threads' of density. In all experiments divergence drops to a small fraction of their one layer divergence when run with 10. Only '10 Gaussians' just halves its divergence from one to ten layers and then has the highest value by far.

This experiment has a lot of way to cover between the individual modals and the MAF often 'connects' them with what one could call smears. The multimodal experiments '7 random Gaussians' and '4 connected Gaussians' did not show this issue and reach far lower divergences. Finally, the amount of layers heavily influences the performance and it can easily reach multitudes of the input dimensions. Whether this applies for high dimensional input data is questionable since in a $2D$ setup only half of the input is actually connected to a neural net. In fact this is equivalent to a $2D$ coupling layer.

4.2.3 Higher Dimensions

Since there is concern that the amount of layers required will not grow with the input dimensions, several ten dimensional distributions are set up and learned several times with varying numbers of layers. And because there is no suitable way of displaying ten dimensions, KL divergence remains the only way to measure how close MAFs can get to the original distribution. Moreover, due to learning densities 'with gaps' like in figure 4.4 being a weakness of MAFs, three experiments are set up to address that. These experiments consist of seven Gaussians with the same covariance matrices. The means, sampled uniformly, are the same for all gap experiments except that they have been multiplied by different factors to variate the gap size. The 'Giant Gaps' range from -15 to 15, 'Large Gaps' from -10 to 10, 'Medium Gaps' from -4 to 4 and 'Small Gaps' from -2.2 to 2.2. An additional experiment with a simple zero centred multivariate Gaussian is created to show that this distribution can be learned by a single layer MAF. Each experiment will train three models for each amount of layers tested. For both kinds of experiments the only parameter that varies is the amount of layers used. The other parameters are set as in table 4.3.

Parameter	***Value***
Activation	ReLU
Batch Size	4096
Epochs	2000
Hidden Shape	[200, 200]
Learning Rate	PolyDecay($1e^{-3}$, $1e^{-4}$)
Optimiser	Adam
Patience	50
Samples	Simple Gaussian: 102400, other: 819200
ValSamples	Simple Gaussian: 10240, other: 12288

Table 4.3: Parameters for all MAFs used in $10D$ experiments.

Average KL of 6 runs per experiment and no of layers, $10D$

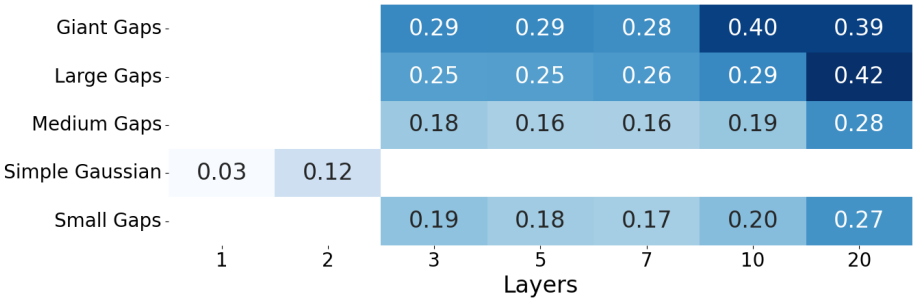


Table 4.4: Left column contains the names of the experiments.

The ‘Simple Gaussian‘ experiment only ran with one and two layers because a single layer MAF should be able to learn the distribution and adding more should make it worse. As you can see in table 4.5 the divergence quadruples from one to two layers. Figure 4.6 shows a box whisker plot of the KL divergences for the gap experiments only. The box shows the quartiles and whiskers the minimum and maximum divergence reached. Table 4.5 shows the divergences for each model and experiment and the averaged values for each model configuration.
In general the variance of the KL divergences shrinks with the width of the gaps. The lowest divergences were reached in the ‘Medium‘ experiment, followed by the ‘Small‘ one. The variance as well as the average values increase drastically in in the two larger gap scenarios. Regarding the median, the 20 layer MAF is the worst in all but one the ‘Giant‘ gap experiment. Five and seven layers have the lowest and second most lowest in ‘Medium‘ and ‘Small‘ and still compare relatively well in the other scenarios. On average seven layer perform best, except in the last gap scenario.

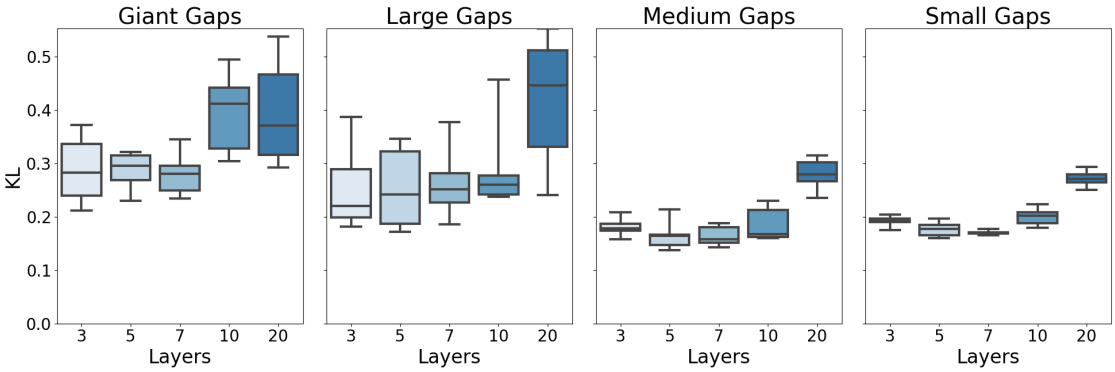


Figure 4.6: Box whisker of the KL-divergences of the ‘gaps‘ experiments, 6 runs each.

	Giant Gaps		**Large Gaps**		**Medium Gaps**		**Small Gaps**	
Layers	KL	Avg(KL)	KL	Avg(KL)	KL	Avg(KL)		
20	0.397	0.395	0.240	0.419	0.263	0.280	0.269	0.272
20	0.293		0.486		0.235		0.250	
20	0.306		0.406		0.284		0.263	
20	0.538		0.520		0.315		0.273	
20	0.345		0.307		0.275		0.282	
20	0.490		0.552		0.308		0.294	
10	0.494	0.396	0.278	0.289	0.230	0.185	0.197	0.200
10	0.426		0.237		0.169		0.185	
10	0.304		0.274		0.228		0.210	
10	0.399		0.457		0.165		0.206	
10	0.447		0.246		0.161		0.223	
10	0.304		0.241		0.160		0.180	
7	0.235	0.280	0.239	0.263	0.151	0.164	0.168	0.170
7	0.266		0.378		0.186		0.165	
7	0.345		0.223		0.152		0.169	
7	0.295		0.186		0.188		0.171	
7	0.296		0.287		0.164		0.170	
7	0.244		0.265		0.143		0.177	
5	0.264	0.287	0.172	0.253	0.165	0.165	0.180	0.177
5	0.309		0.212		0.167		0.162	
5	0.321		0.179		0.214		0.174	
5	0.317		0.339		0.164		0.196	
5	0.282		0.346		0.138		0.186	
5	0.229		0.272		0.141		0.160	
3	0.372	0.288	0.226	0.252	0.158	0.181	0.204	0.192
3	0.261		0.214		0.173		0.189	
3	0.212		0.310		0.208		0.175	
3	0.348		0.182		0.179		0.193	
3	0.304		0.193		0.178		0.192	
3	0.233		0.387		0.190		0.198	

Table 4.5: Divergences and epochs for every MAF trained on the small, medium, large and giant gap dataset.

4.2.4 Findings

While choosing more layers than dimensions decreased the divergence for two dimensions the opposite is true for 10 dimensions. Despite none of the configurations being unaffected by the size of the gaps, two configurations reached best divergences, on average in 3 out of 4 scenarios: five and seven layers. Based on these findings there is no point in using more layers than dimensions in a higher dimensional feature space. It seems that the favourable amount of layers should be somewhere near 50% to 70% of D.

4.3 MiniBooNE dataset

This shall show how well a MAF can learn a real dataset and whether adding synthetic samples from a MAF can improve a classifier. Due to the lack of a

MiniBooNE density function, that enables measuring the KL divergence, the accuracy of classifiers, trained on samples from a trained MAF and genuine samples from the MiniBooNE dataset, is measured instead. So the classier fulfil two purposes here: being a vehicle to measure the quality of the MAF and answering whether they might be improved learning new samples. 'MiniBooNE' is a data set taken from the 'MiniBooNE' experiment and provided by University of California Irvine. It contains roughly 130000 samples of two classes: signal and noise. Nearly 30% of the samples are signals. The dataset has 50 dimensions. The classifiers play a double role here.
There is one setup learning the original dataset and a second one that trains on a balanced subset. 10% of the dataset and the balanced dataset are kept out for testing. One MAF is trained for signal and noise each. Based on previous findings, MAFs with 30 layers are used. This equals 60% of the 50 input dimensions and is right in the middle of the 50% to 70% that worked best for the 10 dimensional experiment. Afterwards a whole new dataset of synthetic samples is generated from the MAF. The signal-to-noise ratio, for the MAF-generated datasets, is kept for each setup. Afterwards classifiers with varying amounts of genuine and synthetic samples are trained. David Lowe experimented with various neural net setups and found a configuration that reaches 95% accuracy on the genuine dataset [Low20]. His classifier is used throughout this experiment. For every combination of genuine and synthetic samples, ten classifiers are trained. You find the average test accuracy in figure 4.7.

4.3.1 Findings

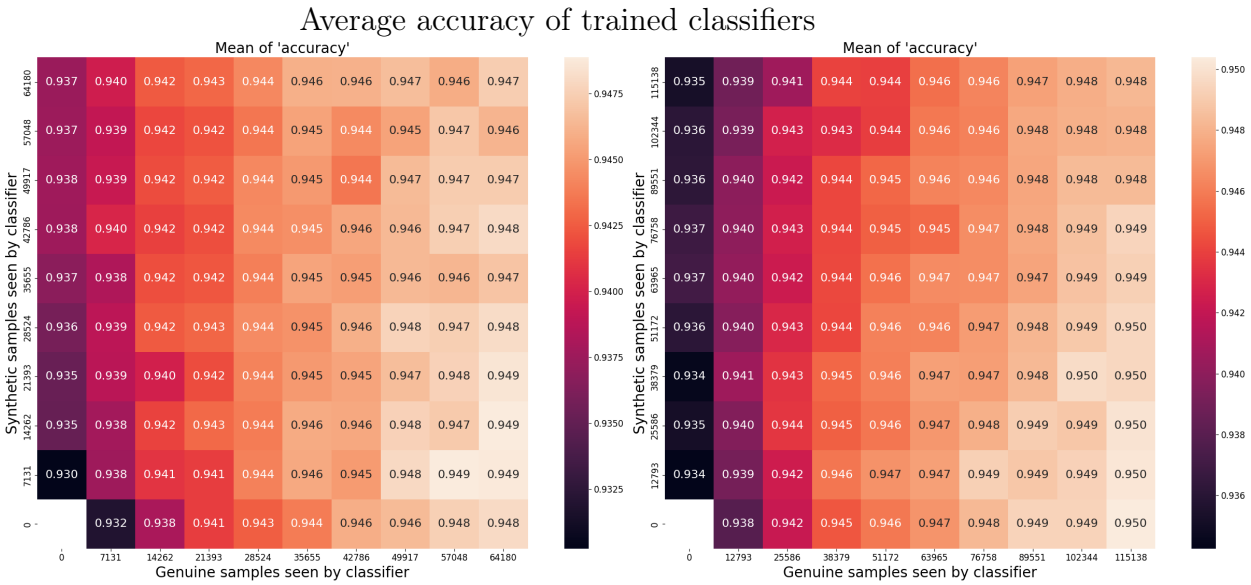


Figure 4.7: Mean accuracy of 10 classifiers trained with various amounts of synthetic and genuine samples. Left: balanced dataset. Right: unbalanced, original dataset.

Using more synthetic samples slightly decreases the accuracy for any amount of genuine samples. When using all synthetic samples, instead of all genuine, in the balanced setup, accuracy went down by 1.1% when using 64180 training samples at all and by 1.9% when using 7131 samples only. Increasing the synthetic samples, with no genuine samples, does not seem to improve beyond 57048 samples. The situation is similar for the original dataset. Again, accuracy drops about 1.3% when training on the maximum amount of samples and drops 1.6% when using 12793 samples. There is a gradual, slight decline in accuracy when synthetic samples are added to the genuine training set or replace genuine samples. Lowe's classifier reached 95% as promised.

Chapter 5

Conclusions

There is no practical use for Masked Autoregressive Flows in a one dimensional feature space. In two dimensions adding more layers can improve it's expressiveness drastically. Using up to ten layers improved the results in every but one case. This may also hold true for other low dimensional spaces but it is not subject of this work. For ten dimensions the most auspicious amount of layers ranges somewhere around 50% to 70% of D, in special cases with huge 'gaps', even less layer may be favourable. The classifiers for the real world dataset MiniBooNE lost around 1% to 2% accuracy when solely trained on synthetic samples. It showed no improvement over using genuine samples only, in any test case, whatsoever. This indicates that the MAF deviates slightly from the real dataset. Masked Autoregressive Flows seem to work best, when the target distribution has no 'gaps', meaning 0 or close to 0 density, between the spaces with higher densities. Nevertheless MAFs are a powerful density estimators that can adapt well to a lot of, especially higher dimensional distributions.The amount of layers must be chosen carefully as it has a significant impact on its quality.

5.1 Future Work and Open Points

Since there is a connection between KL divergence and maximum likelihood, as seen in 2.2.2, it might be of interest whether training a set of MAFs and picking the one with the lowest validation loss will improve the results on the MiniBooNE dataset. It also seems that some MAFs which do bad compared to those of their architectural identical siblings stop training much earlier. 'Easy to learn' distributions like the uni modals do not seem to show that effect, only the more complicated experiments do. This is of rather anecdotal nature since there was no intent to dive into this and there is no sophisticated amount of data to prove it yet. The author is also interested in how other Normalising Flow architectures and variational autoencoder compare to Masked Autoregressive Flows.

Chapter 6

Appendix

6.1 Notation

- D the number of dimensions of a data point
- bold lower letters are vectors: $\boldsymbol{x}$
- regular lower letters are scalars: x_d, z_3, b
- uppercase, bold letters are matrices: $\boldsymbol{M}$
- $p(\boldsymbol{x})$ is the data or target distribution
- $q(\boldsymbol{z})$ the base distribution
- $\boldsymbol{x} \sim p(\boldsymbol{x})$ is a sample from the target/data distribution
- $\boldsymbol{x_{<d}}$ refers to the vector of the first $d-1$ dimensions of $\boldsymbol{x}$
- $\boldsymbol{u} \sim q(\boldsymbol{u})$ is an intermediate distribution of the tranformation from $\boldsymbol{x}$ to $\boldsymbol{z}$
- $\boldsymbol{z} \sim q(\boldsymbol{z})$ and z_d refer to the base distribution side
- $f()$ maps data, $\boldsymbol{x}$ onto the base distribution
- $g()$ is the *inverse* of $f()$. Think of it as *generating*.

6.2 Visualising Transformations

This appendix may give you a better intuition of the MAF's internal functionality. 6.1 and 6.2 show the transformation a two layer MAF applies to the input. The top left shows the input points, center left is transformed input after the first MAF layer, the bottom left after the second MAF layer. So the bottom left points are used to calculate the density on a two dimensional standard normal. The points are colour coded so you can track them though the transformations. The top right shows the Jacbobian determinant for each point. Center right and bottom right show the same but after the first and second transformation respectively. The two layer MAF from Figure 4.3 on page 25 generated this transformation.

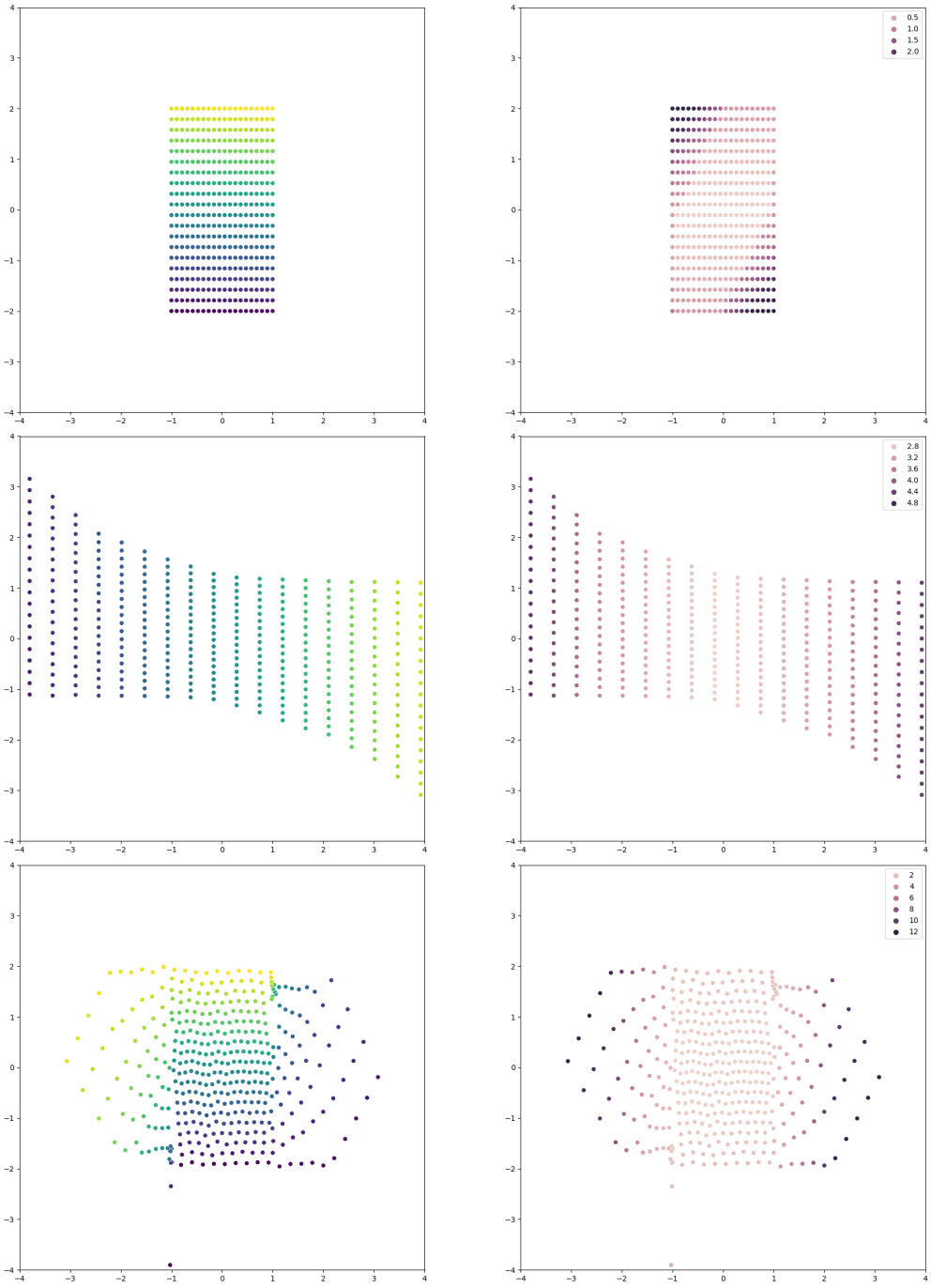


Figure 6.1: Left column: colours link a point to its original position in the upper row. Middle row: position after transformation of the first MAF layer. Bottom row: after two MAF layers. Right column: The absolute Jacobian determinant for each point.

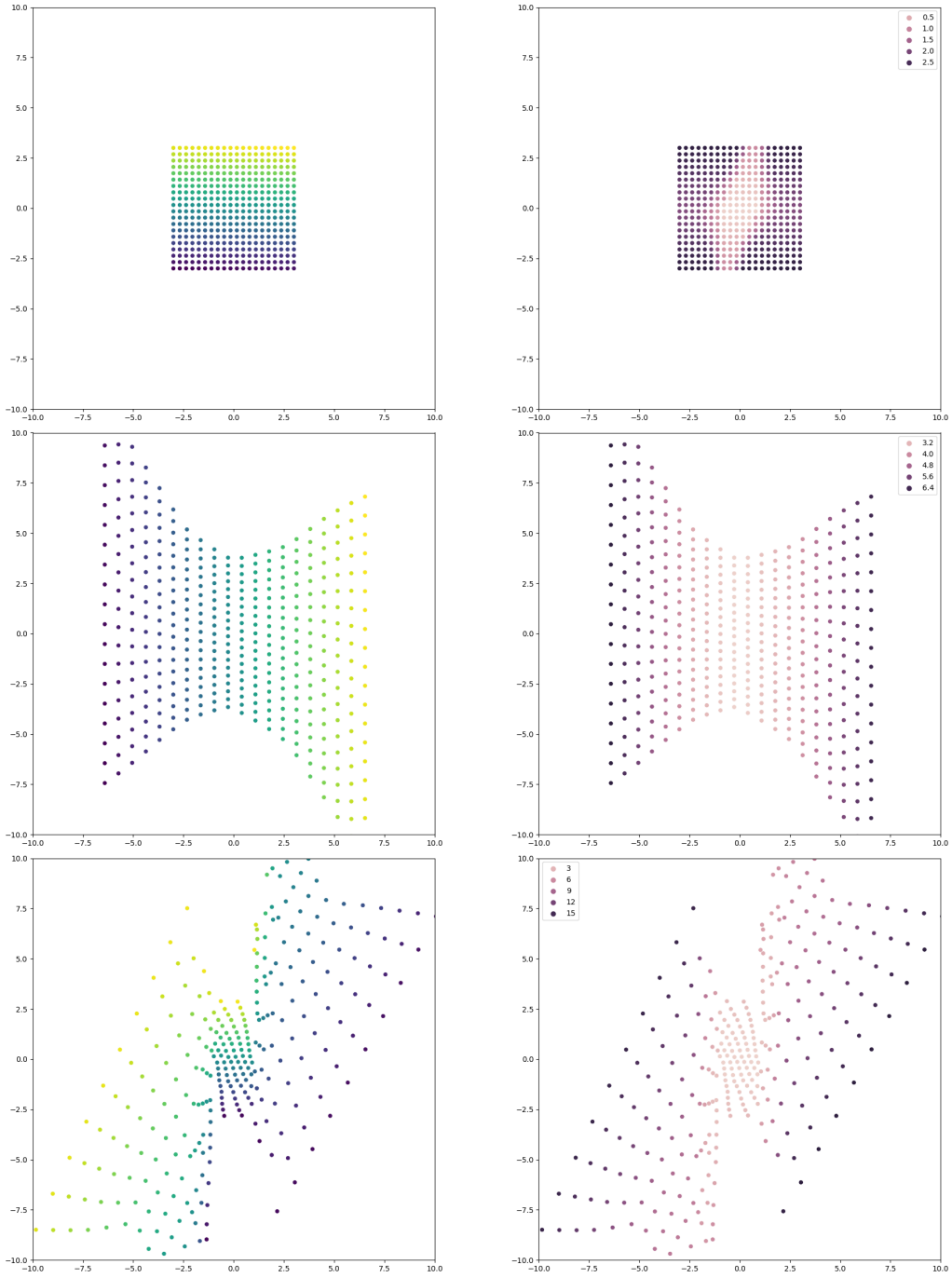
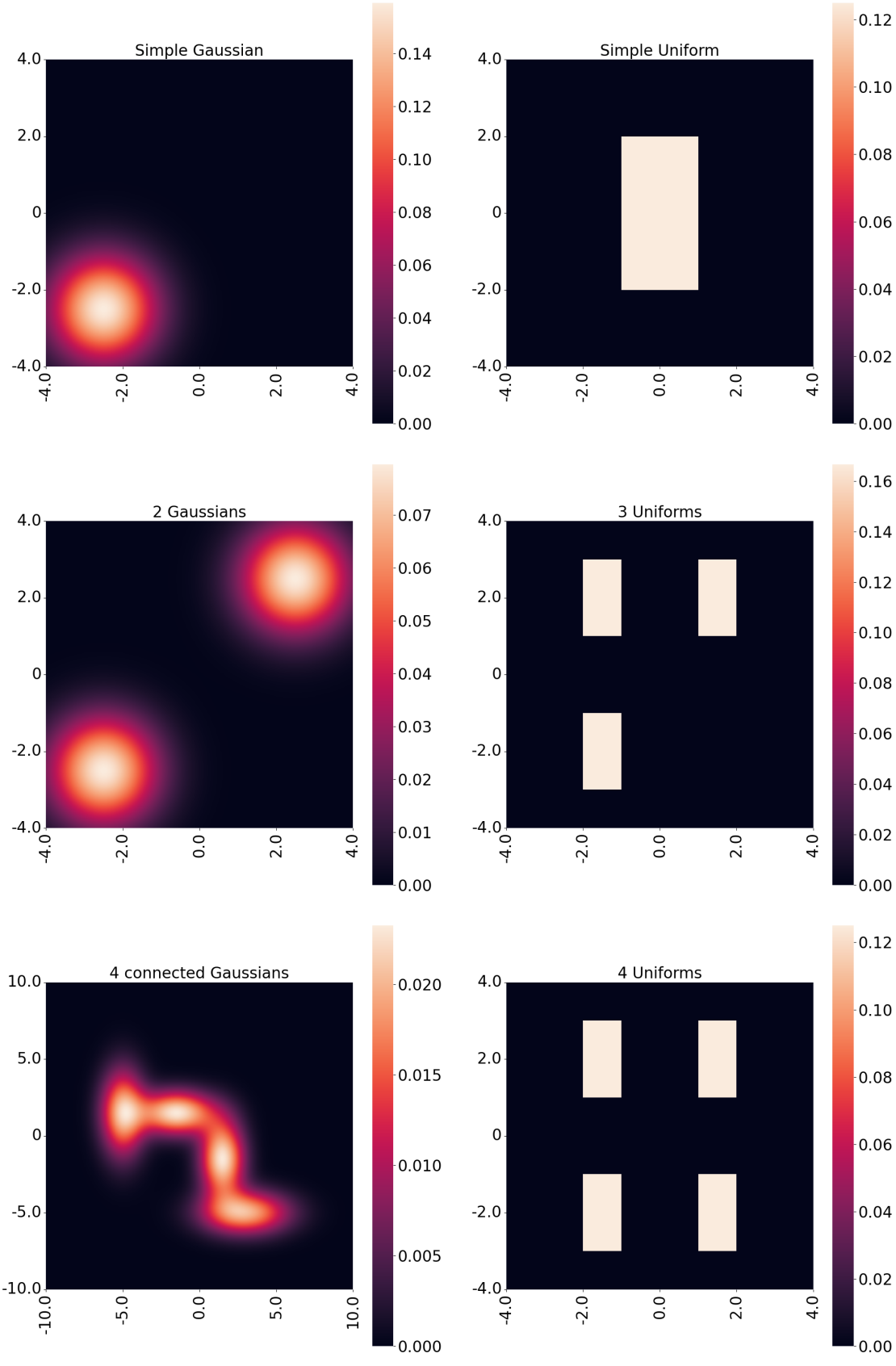
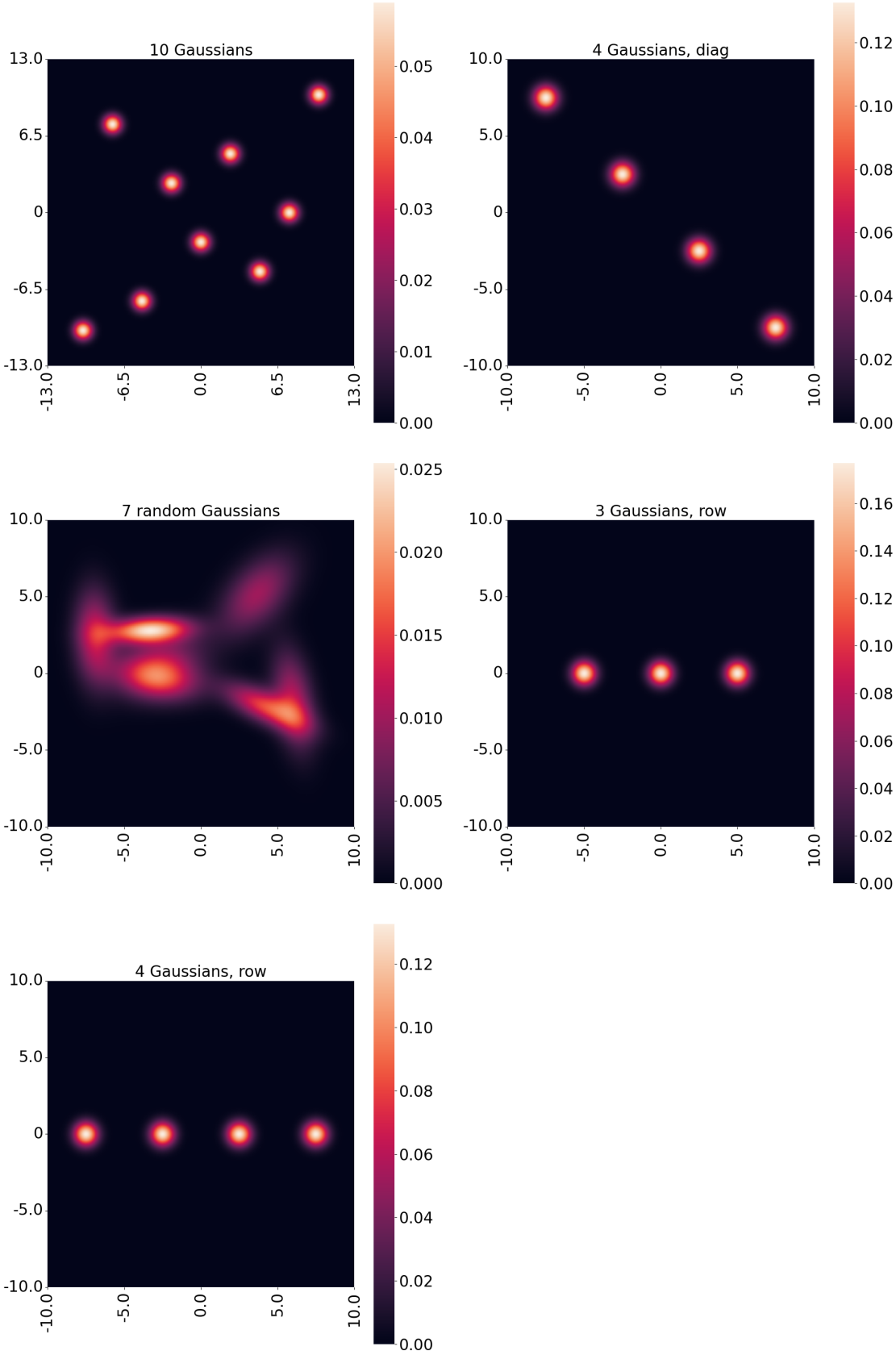


Figure 6.2: Same as 6.1 but with a larger input area.

6.3 Visualisation of experimental 2D distributions




10 Gaussians
13.0
6.5
0
-6.5
-13.0
-13.0
-6.5
0.0
6.5
13.0
0.05
0.04
0.03
0.02
0.01
0.00
4 Gaussians, diag
10.0
5.0
0
-5.0
-10.0
-10.0
-5.0
0.0
5.0
10.0
0.12
0.10
0.08
0.06
0.04
0.02
0.00
7 random Gaussians
10.0
5.0
0
-5.0
-10.0
-10.0
-5.0
0.0
5.0
10.0
0.025
0.020
0.015
0.010
0.005
0.000
3 Gaussians, row
10.0
5.0
0
-5.0
-10.0
-10.0
-5.0
0.0
5.0
10.0
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
4 Gaussians, row
10.0
5.0
0
-5.0
-10.0
-10.0
-5.0
0.0
5.0
10.0
0.12
0.10
0.08
0.06
0.04
0.02
0.00

Bibliography

[AH74] Aho, Alfred V. ; Hopcroft, John E.: *The Design and Analysis of Computer Algorithms*. 1st. USA : Addison-Wesley Longman Publishing Co., Inc., 1974. – ISBN 0201000296

[BKM07] Bogachev, Vladimir ; Kolesnikov, Alexander ; Medvedev, Kirill: Triangular transformations of measures. In: *Sbornik: Mathematics* 196 (2007), Oktober, S. 309. http://dx.doi.org/10.1070/SM2005v196n03ABEH000882. – DOI 10.1070/SM2005v196n03ABEH000882

[CG01] Chen, Scott ; Gopinath, Ramesh: Gaussianization. In: Leen, T. (Hrsg.) ; Dietterich, T. (Hrsg.) ; Tresp, V. (Hrsg.): *Advances in Neural Information Processing Systems* Bd. 13, MIT Press, 2001

[GGML15] Germain, Mathieu ; Gregor, Karol ; Murray, Iain ; Larochelle, Hugo: MADE: Masked Autoencoder for Distribution Estimation. In: *arXiv:1502.03509 [cs, stat]* (2015), Juni. http://arxiv.org/abs/1502.03509. – arXiv: 1502.03509

[HP99] Hyvärinen, Aapo ; Pajunen, Petteri: Nonlinear independent component analysis: existence and uniqueness results. In: *Neural Networks* 12 (1999), April, Nr. 3, 429–439. http://dx.doi.org/10.1016/S0893-6080(98)00140-3. – DOI 10.1016/S0893–6080(98)00140–3. – ISSN 0893–6080

[Low20] Lowe, David: Binary Classification Model for MiniBooNE Particle Identification Using TensorFlow. (2020), Oct. https://merelydoit.blog/2020/10/15/binary-classification-model-for-miniboone-particle-identification-using-te

[PNR+21] Papamakarios, George ; Nalisnick, Eric ; Rezende, Danilo J. ; Mohamed, Shakir ; Lakshminarayanan, Balaji: Normalizing Flows for Probabilistic Modeling and Inference. In: *arXiv:1912.02762 [cs, stat]* (2021), April. http://arxiv.org/abs/1912.02762. – arXiv: 1912.02762

[PPM18] Papamakarios, George ; Pavlakou, Theo ; Murray, Iain: Masked Autoregressive Flow for Density Estimation. In: *arXiv:1705.07057 [cs, stat]* (2018), Juni. http://arxiv.org/abs/1705.07057. – arXiv: 1705.07057

[RM16] Rezende, Danilo J. ; Mohamed, Shakir: Variational Inference with Normalizing Flows. In: *arXiv:1505.05770 [cs, stat]* (2016), Juni. http://arxiv.org/abs/1505.05770. – arXiv: 1505.05770

[Str69] Strassen, Volker: Gaussian Elimination is Not Optimal. In: *Numer. Math.* 13 (1969), aug, Nr. 4, 354–356. http://dx.doi.org/10.1007/BF02165411. – DOI 10.1007/BF02165411. – ISSN 0029–599X

[TT13] Tabak, E. G. ; Turner, Cristina V.: A Family of Nonparametric Density Estimation Algorithms. In: *Communications on Pure and Applied Mathematics* 66 (2013), Februar, Nr. 2, 145–164. http://dx.doi.org/10.1002/cpa.21423. – DOI 10.1002/cpa.21423. – ISSN 00103640

[TVE10] Tabak, Esteban G. ; Vanden-Eijnden, Eric: Density estimation by dual ascent of the log-likelihood. In: *Communications in Mathematical Sciences* 8 (2010), Nr. 1, 217–233. http://dx.doi.org/10.4310/CMS.2010.v8.n1.a11. – DOI 10.4310/CMS.2010.v8.n1.a11. – ISSN 15396746, 19450796

[Wik22a] Wikipedia: *Galactic algorithm — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Galactic%20algorithm&oldid=1068751300, 2022. – [Online; accessed 26-February-2022]

[Wik22b] Wikipedia: *Histogram — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Histogram&oldid=1070180124, 2022. – [Online; accessed 26-February-2022]

[Wik22c] Wikipedia: *Kernel density estimation — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Kernel%20density%20estimation&oldid=1068673208, 2022. – [Online; accessed 26-February-2022]