University Library



A gateway to Melbourne's research publications

Minerva Access is the Institutional Repository of The University of Melbourne

Author/s:
Dolatabadi, HM;Erfani, S;Leckie, C

Title:
Invertible Generative Modeling using Linear Rational Splines

Date:
2020

Citation:
Dolatabadi, H. M., Erfani, S. & Leckie, C. (2020). Invertible Generative Modeling using Linear Rational Splines. Chiappa, S (Ed.) Calandra, R (Ed.) International Conference on Artificial Intelligence and Statistics, Vol.108, 108, pp.4236-4245. AISTATS.

Persistent Link:
https://hdl.handle.net/11343/267645

Invertible Generative Modeling using Linear Rational Splines

Hadi M. Dolatabadi **Sarah Erfani** **Christopher Leckie**
School of Computing and Information Systems
The University of Melbourne

Abstract

Normalizing flows attempt to model an arbitrary probability distribution through a set of invertible mappings. These transformations are required to achieve a tractable Jacobian determinant that can be used in high-dimensional scenarios. The first normalizing flow designs used coupling layer mappings built upon affine transformations. The significant advantage of such models is their easy-to-compute inverse. Nevertheless, making use of affine transformations may limit the expressiveness of such models. Recently, invertible piecewise polynomial functions as a replacement for affine transformations have attracted attention. However, these methods require solving a polynomial equation to calculate their inverse. In this paper, we explore using linear rational splines as a replacement for affine transformations used in coupling layers. Besides having a straightforward inverse, inference and generation have similar cost and architecture in this method. Moreover, simulation results demonstrate the competitiveness of this approach's performance compared to existing methods.

1 INTRODUCTION

Flow-based modeling, widely known as normalizing flows (Tabak and Turner, 2013; Rezende and Mohamed, 2015), is a novel approach used for density estimation problems. The main idea behind this method is to model the distribution of any arbitrary set of data as the mapping of a simple base random variable using a set of invertible transformations. In doing so, they make use of the well-known change of variables formula from probability theory (Papoulis and Pillai, 2002). In particular, let $\mathbf{Z}$ denote a random vector with a simple distribution such as a standard normal. Furthermore, let $\mathbf{X}$ be the result of applying an invertible transformation $\mathbf{f}(\cdot)$ on $\mathbf{Z}$. Then, it can be shown that:

$$p_{\mathbf{X}}(\mathbf{x}) = p_{\mathbf{Z}}(\mathbf{z}) \left| \det\left(\frac{\partial \mathbf{f}}{\partial \mathbf{z}}\right) \right|^{-1}, \tag{1}$$

where $p_{\mathbf{X}}(\mathbf{x})$ and $p_{\mathbf{Z}}(\mathbf{z})$ denote the probability distributions of $\mathbf{X}$ and $\mathbf{Z}$, respectively. Here, $\left|\det\left(\frac{\partial \mathbf{f}}{\partial \mathbf{z}}\right)\right|$ indicates the absolute value of the *Jacobian determinant* for the transformation $\mathbf{f}(\cdot)$. This quantity captures the amount of change in $\mathbf{z}$ caused by the transformation (*flow*) $\mathbf{f}(\cdot)$ through the Jacobian operation. Then, this change in variable $\mathbf{z}$ can be used to *normalize* the probability distribution of the data through the determinant calculation.

Eq. (1) can be used for maximum likelihood problems. This can be done by parameterizing the distribution of the data through an invertible mapping $\mathbf{f}(\cdot)$ and then finding the parameters of this transformation using an appropriate optimization algorithm.

However, a major bottleneck to using normalizing flows in high-dimensional scenarios is the complexity of computing the Jacobian determinant. In general, determinant calculation has an $\mathcal{O}(D^3)$ complexity for an arbitrary $D \times D$ matrix. Hence, to adapt flow-based models to high-dimensional cases, this issue must be addressed. As a result, the main objective in designing normalizing flow algorithms is to come up with an invertible transformation whose Jacobian determinant is tractable.

This work presents a new transformation for use in normalizing flow algorithms. We show that the transformation used has an analytical inverse. Moreover, experiments done using this transformation indicate its competitive performance with existing state-of-the-art algorithms.

Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108. Copyright 2020 by the author(s).

2 RELATED WORK

In this section, we review the existing methods for flow-based modeling.

2.1 Coupling Layer Transformations

The main idea behind this category of transformations is as follows. They first split the data into two parts. The first part is output without any change. Then, a transformation is constructed based on the first partition of the data. This transformation is then applied to the second part of the data to generate the output.

Specifically, let $\mathbf{z} \in \mathbb{R}^{\mathrm{D}}$ show the data to be transformed. First, it is split into two parts $\mathbf{z}_1 \in \mathbb{R}^{\mathrm{d}}$ and $\mathbf{z}_2 \in \mathbb{R}^{\mathrm{D-d}}$ where $d < D$. Then, each part is transformed separately using

$$\begin{aligned} \mathbf{x}_1 &= \mathbf{z}_1 \\ \mathbf{x}_2 &= \mathbf{g}_{\boldsymbol{\theta}(\mathbf{x}_1)}(\mathbf{z}_2), \end{aligned} \tag{2}$$

where $\mathbf{g}_{\boldsymbol{\theta}(\mathbf{x}_1)}(\cdot)$ is an invertible, element-wise transformation with parameters $\boldsymbol{\theta}(\mathbf{x}_1)$ which have been computed based on $\mathbf{x}_1$. The final output of the transformation is then $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2]$. It can be shown that such transformations have a lower triangular Jacobian whose determinant is the multiplication of the diagonal elements. Note that coupling layers do not change the first split of their inputs. Thus, to prevent that part of the data from remaining unchanged, it is necessary to use an alternating mask and switch the splits at two consecutive transformations to make sure that every dimension of the data has a chance to be changed.

The NICE algorithm (Dinh et al., 2015) uses a simple translation function as its transformation $\mathbf{g}_{\boldsymbol{\theta}(\mathbf{x}_1)}(\cdot)$. It sets $\mathbf{g}_{\boldsymbol{\theta}(\mathbf{x}_1)}(\mathbf{z}_2) = \mathbf{z}_2 + \boldsymbol{\theta}(\mathbf{x}_1)$ where $\boldsymbol{\theta}(\cdot)$ is a multi-layer perceptron (MLP) with Rectified Linear Units (ReLU) as the activation function.

Real NVP (Dinh et al., 2017) generalizes NICE by using an affine transformation as $\mathbf{g}_{\boldsymbol{\theta}(\mathbf{x}_1)}(\cdot)$. Here, the authors use two Residual Networks (ResNet) (He et al., 2016) to come up with the translation and scaling operations required for an affine mapping. As in NICE, Real NVP uses a simple alternating mask to involve every dimension of the data in the transformation. In Kingma and Dhariwal (2018), it is suggested to linearly combine dimensions of the data before feeding it to each layer of transformation using an invertible 1×1 convolution operation, resulting in a new algorithm called Glow. This permutation is simply a matrix multiplication. Thus, to make the Glow algorithm fast, the authors propose using an LU-decomposition to calculate the matrix mentioned above. It is shown that Glow (Kingma and Dhariwal, 2018) improves Real NVP's performance (Dinh et al., 2017) for generative image modeling.

Despite its better performance compared to Real NVP, Glow may struggle to learn synthetic probability distributions with multiple separated modes (Grathwohl et al., 2019). Since all the previously mentioned algorithms use an affine transformation, this may be the cause of the degradation in their general expressiveness.

2.2 Autoregressive Transformations

The chain rule in probability theory states that if $\mathbf{Z} = (Z_1, Z_2, \ldots, Z_D)$ is a D-dimensional random variable, then the distribution of $\mathbf{Z}$ can be written as (Papoulis and Pillai, 2002)

$$p_{\mathbf{Z}}(\mathbf{z}) = p_{Z_1}(z_1) \prod_{i=2}^{D} p_{Z_i|\mathbf{Z}_{<i}}(z_i|\mathbf{z}_{<i}), \tag{3}$$

where $\mathbf{z}_{<i}$ is shorthand notation denoting all dimensions of vector $\mathbf{z}$ with an index less than i.

Autoregressive flows exploit this rule to build their transformations. In particular, they transform each one-dimensional variable z_i conditioned on its previous dimensions $\mathbf{z}_{<i}$ using an invertible transformation. It can be shown that the Jacobian determinant of such transformations is again lower triangular and can be computed efficiently.

Inverse Autoregressive Flows (IAF) (Kingma et al., 2016) and Masked Autoregressive Flows (MAF) (Papamakarios et al., 2017) are among the first designs in this category. They use affine functions to build their autoregressive transformation. Later, it was argued that the use of affine functions might limit the expressiveness of such models (Huang et al., 2018). Hence, Neural Autoregressive Flows (NAF) (Huang et al., 2018) were introduced. NAFs build their transformation using a neural network with positive weights and monotonic activations to ensure invertibility. A hyper neural network, called the conditioner network, is trained to capture the dependency of the neural network parameters to the previously seen data. Later, Block Neural Autoregressive Flows (B-NAF) (Cao et al., 2019) suggest a more straightforward structure omitting the conditioner network. It was shown that B-NAFs reach the same performance as NAFs using orders of magnitude fewer parameters. Moreover, universal density approximation has been proved for both NAFs and B-NAFs. This theorem states that a structure of these models always exists that could accurately represent the data.

Despite their success, there are two important observations regarding NAFs and B-NAFs. *First, univer-*

sal density approximation only proves the existence of such models without any convergence guarantee. Moreover, these two methods are not analytically invertible, questioning their usage for generative probabilistic modeling tasks where one might want to compute the inverse.

In general, unlike coupling layer transformations, inverting an autoregressive flow cannot be done in a single pass, making their inverse computationally expensive. Thus, it would be better if one can come up with a coupling layer transformation whose performance can compete with autoregressive flows. There are also other autoregressive models to which we refer the interested reader (Germain et al., 2015; Chen et al., 2017; Oliva et al., 2018).

2.3 Other Methods

In addition to the coupling layer and autoregressive transformations, there are other methods that do not fall into any of the previous categories. We summarize the most well-known ones here.

Continuous Normalizing Flows (CNF) are flow-based models that are constructed upon Neural Ordinary Differential Equations (ODE) (Chen et al., 2018). FFJORD (Grathwohl et al., 2019) improves CNF's computational complexity using the Hutchinson's trace estimator (Hutchinson, 1990). Both of these methods involve a system of first-order ODEs that replaces the change of variables formula in Eq. (1). Then, this ODE system is solved by a proper integrator resulting in a flow-based model. Here, the sampling process requires solving a system of ODEs, which may slow down sample generation.

A closely related method to FFJORD is invertible residual networks, or i-ResNets for short (Behrmann et al., 2019). In this paper, the authors first state a sufficient condition for making residual networks invertible. Then, a flow-based generative model is built using this invertible transformation. Realization of such models depends on calculating the Jacobian determinant of an entire residual network. In Behrmann et al. (2019), it is shown that an infinite power series can replace this Jacobian determinant. To compute this infinite series feasibly, the authors suggest truncating it after a finite number of terms n, which causes the Jacobian determinant estimator to become biased (Behrmann et al., 2019).

Residual Flows (Chen et al., 2019) address the bias issue of i-ResNets by using a so-called *Russian roulette* estimator (Kahn, 1955). In short, instead of a deterministic n used for truncation of the infinite series as in i-ResNets, a *Russian roulette* estimator models n as a discrete random variable on natural numbers. Then, the infinite power series is replaced with the first n terms divided by appropriate weights. Unlike i-ResNets, here n is a realization of an arbitrary distribution on natural numbers. Since Residual Flows use an unbiased estimator for the Jacobian determinant, it is shown that they can achieve a better performance compared to i-ResNets (Behrmann et al., 2019).

A major drawback of models such as i-ResNets and Residual Flows, which make use of invertible residual networks as their building blocks, is their inversion. Although i-ResNets are guaranteed to be invertible, they do not have an analytical inverse. I-ResNets, as well as Residual Flows, need a fixed-point iterative algorithm to compute the inverse of each layer. Hence, the inversion process takes much more time (around 5-20x) than inference (Behrmann et al., 2019).

3 PROPOSED APPROACH

3.1 Neural Spline Flows

As we saw in Eq. (2), the only requirements that $\mathbf{g}_{\boldsymbol{\theta}(\mathbf{x}_1)}(\cdot)$ must satisfy is being invertible and differentiable. Also, we saw that Real NVP and Glow use a simple affine transformation, to build their algorithm and maintain analytical invertibility. In contrast, autoregressive methods either use a simple affine transformation like IAFs and MAFs, or use non-analytically invertible neural networks with positive weights as in NAFs and B-NAFs. However, there are a plethora of differentiable functions that are invertible and lie in-between: they can be more expressive than a simple affine mapping while having an easy-to-compute inverse.

A family of functions with such properties is splines. Splines are piecewise functions where each piece is expressed as a closed-form standard function. The most popular form of splines is those defined by polynomials.

In the context of flow-based modeling, the idea of replacing the affine transformation used in methods such as NICE with piecewise polynomial functions was first introduced by Müller et al. (2019). They use a piecewise linear or quadratic function as a replacement for the affine transformation in coupling layers. It is shown in that this change improves the expressiveness of methods such as Real NVP.

Later, this work was extended to the cases of cubic and rational quadratic splines (Durkan et al., 2019a,b). It is shown that the previous coupling layer methods such as Real NVP and Glow could benefit from this change. Also, simulation results demonstrate their competi-

tive performance against the most expressive methods, such as NAF and B-NAF, without sacrificing analytical invertibility.

In this work, we explore the usage of another family of piecewise functions, namely linear rational splines, in the context of normalizing flows. We aim to seek more straightforward piecewise functions whose inversion can be done efficiently. Previous methods in this area use quadratic, cubic, and rational quadratic functions whose inversion is done after solving degree 2 or 3 polynomial equations. However, piecewise linear rational splines can perform competitively with these methods without requiring a polynomial equation to be solved in the inversion.

Next, we review linear rational splines and their application in constructing monotonically increasing functions. Then, based on this algorithm, we propose our flow-based model.

3.2 Monotonic Data Interpolation using Linear Rational Splines

Let $\{(x^{(k)}, y^{(k)})\}_{k=0}^{K}$ be a set of monotonically increasing points called knots. Furthermore, let $\{d^{(k)} > 0\}_{k=0}^{K}$ be a set of positive numbers representing the derivative of each point. Consider that we wish to find linear rational functions[1] of the form $y = \frac{ax+b}{cx+d}$ that fit the given points and their respective derivatives in each interval (also called a bin) $[x^{(k)}, x^{(k+1)}]$. Also, consider that we require the function to be monotone.

In each bin, after satisfying function value constraints at the start and end points, we can write down the desired function as:

$$g(x) = \frac{w^{(k)} y^{(k)} (1-\phi) + w^{(k+1)} y^{(k+1)} \phi}{w^{(k)}(1-\phi) + w^{(k+1)}\phi}, \qquad (4)$$

in which $w^{(k)}$ and $w^{(k+1)}$ are two arbitrary weights and $\phi = (x - x^{(k)})/(x^{(k+1)} - x^{(k)})$, which belongs to the interval $[0, 1]$. As can be seen in Eq. (4), we only have one degree of freedom left while still needing to satisfy two derivative constraints at the extreme points of the bin. Thus, we cannot use a single linear rational function and satisfy all the constraints of each bin.

Fuhr and Kallay (1992) suggest to solve this issue by considering an intermediate point in each interval $(x^{(k)}, x^{(k+1)})$. This way, we can treat each bin as it was two. Since there are no constraints on this particular intermediate point in terms of the value and derivative, we can use its associated parameters to add more degrees of freedom to the existing ones. Then, we can fit one linear rational function to each one of these two intervals and satisfy the end point values and derivative constraints.

In particular, let $x^{(m)} = (1-\lambda)x^{(k)} + \lambda x^{(k+1)}$. When $0 < \lambda < 1$, this equation denotes a point in the interval $(x^{(k)}, x^{(k+1)})$. As in Fuhr and Kallay (1992), we aim to fit two linear rational functions like Eq. (4) to the intervals $[x^{(k)}, x^{(m)}]$ and $[x^{(m)}, x^{(k+1)}]$. Here, the value, weight and derivative of the intermediate point are treated as parameters to satisfy the value and derivative constraints that we have at each bin.

Our desired function is required to be continuous and monotonic. Since the derivative of a linear rational function does not change its sign, when we interpolate it using positive derivatives, this constraint is always satisfied. Hence, we only need to take care of the continuity of the function and its derivative at $x^{(m)}$.

After satisfying all those constraints, the final algorithm for monotonic data interpolation using linear rational functions for each interval $[x^{(k)}, x^{(k+1)}]$ can be shown as in Algorithm 1 (Fuhr and Kallay, 1992). We can then use this algorithm for all the bins, and end up having a piecewise monotonically increasing function, also known as a linear rational spline. Specifically, for each bin $[x^{(k)}, x^{(k+1)}]$ we have:

$$g(\phi) = \begin{cases} \dfrac{w^{(k)} y^{(k)} (\lambda^{(k)} - \phi) + w^{(m)} y^{(m)} \phi}{w^{(k)}(\lambda^{(k)} - \phi) + w^{(m)}\phi} & 0 \le \phi \le \lambda^{(k)} \\ \dfrac{w^{(m)} y^{(m)} (1-\phi) + w^{(k+1)} y^{(k+1)} (\phi - \lambda^{(k)})}{w^{(m)}(1-\phi) + w^{(k+1)}(\phi - \lambda^{(k)})} & \lambda^{(k)} \le \phi \le 1 \end{cases} \qquad (5)$$

By having a monotonically increasing function whose derivatives exist at all points, we can use it as an alternative for the function used in Eq. (2) and construct a flow-based model. Unlike previously used piecewise functions, linear rational splines have the advantage of having a straightforward inverse that does not require solving a degree 2 or 3 polynomial equation. Moreover, the inverse of linear rational functions has the same format as its forward form, but with different parameters. Thus, the regular and inverse function evaluations cost the same. More interestingly, having one extra degree of freedom (namely $\lambda^{(k)}$[2]) per bin provides the opportunity to manipulate the curve that fits through a set of possible knots and derivatives. This flexibility is shown in Figure 1. Here, a set of knot points with fixed derivatives are interpolated with different $\lambda^{(k)}$s. In this figure, the assumption is that a single function shares the same value for all $\lambda^{(k)}$s.

[1] In the context of splines, they are also known as linear/linear rational functions. Also, they are sometimes referred to as homographic functions.

[2] Note that although $w^{(k)}$ is chosen freely in Algorithm 1, since $w^{(m)}$ and $w^{(k+1)}$ are a multiplication of $w^{(k)}$, this value does not provide any degree of freedom.

Algorithm 1: (Fuhr and Kallay, 1992) Linear Rational Spline Interpolation of Monotonic Data in the Interval $\left[x^{(k)}, x^{(k+1)}\right]$

Input: $x^{(k)} < x^{(k+1)}$, $y^{(k)} < y^{(k+1)}$, $d^{(k)} > 0$ and $d^{(k+1)} > 0$

Output: $\lambda^{(k)}$, $w^{(k)}$, $w^{(m)}$, $w^{(k+1)}$ and $y^{(m)}$ (Parameters of two homographic functions of Eq. 4)

Set $w^{(k)} > 0$ and $0 < \lambda^{(k)} < 1$;

Calculate $w^{(k+1)} = \sqrt{d^{(k)}/d^{(k+1)}}w^{(k)}$;

Compute $y^{(m)} = \dfrac{\left(1-\lambda^{(k)}\right)w^{(k)}y^{(k)} + \lambda^{(k)}w^{(k+1)}y^{(k+1)}}{\left(1-\lambda^{(k)}\right)w^{(k)} + \lambda^{(k)}w^{(k+1)}}$;

Calculate $w^{(m)} = \left(\lambda^{(k)}w^{(k)}d^{(k)} + \left(1-\lambda^{(k)}\right)w^{(k+1)}d^{(k+1)}\right)\dfrac{x^{(k+1)}-x^{(k)}}{y^{(k+1)}-y^{(k)}}$

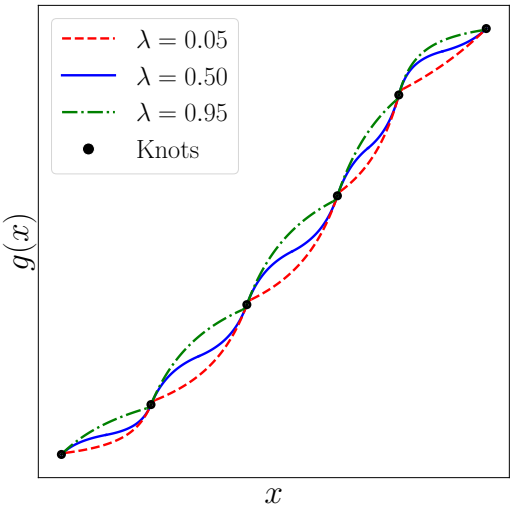


Figure 1: Monotonic data interpolation using linear rational splines. As it can be seen, different settings of λ can result in different curves.

However, this does not need to be the case, and one could manipulate $\lambda^{(k)}$ of each bin separately, resulting in greater flexibility.

Note that spline transformations are defined within a finite interval. To deal with unbounded data, there are two possible solutions. First, a transformation can be used to map the unbounded data to the desired range. For instance, Müller et al. (2019) and Durkan et al. (2019a) suggest mapping the data into a $[0, 1]$ interval to deal with this issue. A downside to this method is the numerical errors caused by this extra transformation (Durkan et al., 2019a). As an alternative approach, Durkan et al. (2019b) propose to use linear tails outside of the interval defined by piecewise functions. This way, the transformation range covers all the real line, circumventing the need to force the data itself to be in the interval $[0, 1]$. We also prefer this way to deal with this problem.

3.3 Linear Rational Spline (LRS) Flows

Having an algorithm for monotone data interpolation, we can exploit and adapt it to coupling layers, and thus, come up with a normalizing flow. We follow the steps of neural spline flows (Durkan et al., 2019b).

First, K and B, the number of bins and the boundary of the spline calculation are set. Then, in order to compute the transformation as in Eq. (2), a neural network such as a ResNet should be trained to determine the parameters of the transformation $\mathbf{g}_{\boldsymbol{\theta}(\mathbf{x}_1)}(\cdot)$. As we have seen in Algorithm 1, $4K-1$ parameters are required for each dimension of this transformation: a width, height, and λ for each bin; and $K-1$ derivatives at all points except the start and end points. At these two points the derivative is set to be 1 for consistency with linear tails. After determining the transformation parameters for each dimension of the data, we apply the linear rational spline algorithm to come up with a set of functions that later construct different dimensions of $\mathbf{g}_{\boldsymbol{\theta}(\mathbf{x}_1)}(\cdot)$. As in Glow, a 1×1 convolution constructed by LU-decomposition is also used to linearly combine the data that is going to be fed into the coupling layer. Furthermore, a multi-scale architecture is used for image generation scenarios as in Real NVP and Glow.

Note that all spline transformations can also be used in an autoregressive fashion. However, coupling layers are preferred as they can perform the inversion in a single pass.

Moreover, as Durkan et al. (2019b) suggest, we perform a transformation on the first partition of the data using trainable parameters. In particular, let $\boldsymbol{\phi}$ be a set of trainable parameters that does not depend on the data. Then, Eq. (2) can be re-written as

$$\begin{aligned} \mathbf{x}_1 &= \mathbf{g}_{\boldsymbol{\phi}}(\mathbf{z}_1) \\ \mathbf{x}_2 &= \mathbf{g}_{\boldsymbol{\theta}(\mathbf{x}_1)}(\mathbf{z}_2), \end{aligned} \tag{6}$$

where $\mathbf{g}_{\boldsymbol{\psi}}(\cdot)$ is an invertible, element-wise linear ratio-

nal spline with parameters ψ. This way all the variables are transformed while the Jacobian determinant still remains lower triangular.

Before considering the simulation results, it is worthwhile to highlight the differences between this work and previous approaches. First, here the inverse has a straightforward relationship and does not require solving degree 2 or 3 polynomial equations. Second, as in Real NVP and Glow, the inverse of this transformation is given with a similar format to its forward form: both of them are linear rational splines. This property can be useful in theoretical analysis of this flow-based model. For instance, it is sufficient to investigate mathematical properties (such as the bi-Lipschitz property that can be used for stability guarantees as in i-ResNets (Behrmann et al., 2019)) for a rational linear spline. Then, this property can hold for both the forward and the inverse transformations as both of them are linear rational splines. In contrast, rational quadratic splines need slightly fewer parameters. This issue can be avoided by fixing λ in linear rational splines.

4 SIMULATION RESULTS

In this section, we review our simulation results. We see that the proposed method can perform competitively despite using a lower order polynomial. The code is available online at: https://github.com/hmdolatabadi/LRS_NF.

4.1 Synthetic Density Estimation

As a first experiment, we studied a density estimation scenario on synthetic 2-d distributions. This task involves the reconstruction of a continuous probability distribution given a set of its samples.

Figure 2 compares the performance of Glow (Kingma and Dhariwal, 2018), i-ResNets (Behrmann et al., 2019) and our proposed method under this scenario. Qualitatively, linear rational spline (LRS) flows can reconstruct the underlying distribution precisely, and outperform the other two models.

In fact, linear rational splines can perform density estimation tasks on more complicated distributions. Figure 3 shows the result of a density estimation task, which involves a highly sophisticated distribution. In both of these experiments, our model consists of two coupling layers constructed by linear rational splines. For detailed information on the configuration used in these simulations, refer to Appendix B.1.

4.2 Density Estimation of Real-world Data

For the next experiment, we apply density estimation using maximum likelihood on standard benchmark datasets. Four of these datasets (Power, Gas, HEPMASS, and MiniBooNE) are tabular data from the UCI machine learning repository[3]. Furthermore, BSDS300 is a dataset containing patches of natural images (Martin et al., 2001). We used the preprocessed data of Masked Autoregressive Flows (Papamakarios et al., 2017), which is available online[4].

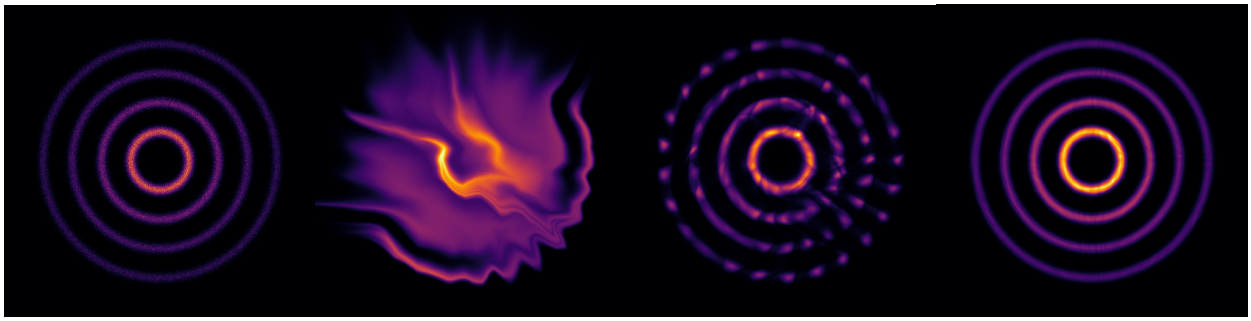
Table 1 shows the test set log-likelihood comparison of our proposed method and FFJORD (Grathwohl et al., 2019), Glow (Kingma and Dhariwal, 2018), Masked Autoregressive Flow (MAF) (Papamakarios et al., 2017), Neural Autoregressive Flow (NAF) (Huang et al., 2018), Block Neural Autoregressive Flow (B-NAF) (Cao et al., 2019), and Neural Spline Flows (NSF). Note that NSF models either use quadratic (Q) piecewise polynomials as in Müller et al. (2019) or rational quadratic (RQ) splines as in Durkan et al. (2019b). Moreover, recall that piecewise polynomials can be used in either coupling or autoregressive modes. These two modes are indicated in this table and the following ones with (C) and (AR), respectively. As suggested in Durkan et al. (2019b), ResMADE architecture (Durkan and Nash, 2019) was used for autoregressive transformations.

It can be seen in Table 1 that linear rational spline flows perform competitively with RQ-NSF despite using lower degree polynomials. Furthermore, although here we set the $\lambda^{(k)}$ of each bin to be a single parameter for itself, one could consider a unified parameter to be set for the $\lambda^{(k)}$ of all intervals. While this action can slightly degrade the performance, the results are still comparable to the other methods.

4.3 Generative Modeling of Image Datasets

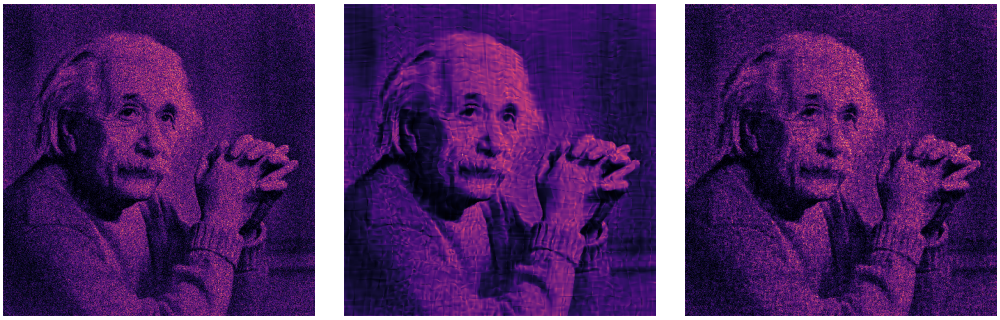
Next, we perform invertible generative modeling on benchmark image datasets including MNIST (LeCun, 1998), CIFAR-10 (Krizhevsky and Hinton, 2009), ImageNet 32×32 and 64×64 (Deng et al., 2009; Chrabaszcz et al., 2017). We measure the performance of our proposed method in bits per dimension, and then compare it with other existing methods. These include Real NVP (Dinh et al., 2017), Glow (Kingma and Dhariwal, 2018), FFJORD (Grathwohl et al., 2019), i-ResNets (Behrmann et al., 2019), residual flow (Chen et al., 2019) and rational quadratic spline flows (Durkan et al., 2019b). The results are

[3] http://archive.ics.uci.edu/ml
[4] https://doi.org/10.5281/zenodo.1161203



Data Samples Glow i-ResNet Ours

Figure 2: Density estimation on synthetic 2-d data samples. The first three images were taken from Behrmann et al. (2019), with permission.



Data Samples Estimated Density Model Samples

Figure 3: Density estimation using data samples of a complicated synthetic distribution derived from a picture. As it can be seen, LRS flows can reconstruct the underlying distribution accurately.

Table 1: Log-likelihood of the test set in nats (higher is better) for four UCI datasets plus BSDS300. All the results except for our method are taken from the existing literature. In rows with † superscript, the error bar is calculated across independently trained models, while in others it has been calculated on the test set.

MODEL	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
FFJORD	0.46	8.59	−14.92	−10.43	157.40
Glow	0.42 ± 0.01	12.24 ± 0.03	-16.99 ± 0.02	-10.55 ± 0.45	156.95 ± 0.28
Q-NSF (C)	0.64 ± 0.01	12.80 ± 0.02	-15.35 ± 0.02	-9.35 ± 0.44	157.65 ± 0.28
RQ-NSF (C)	0.64 ± 0.01	13.09 ± 0.02	-14.75 ± 0.03	-9.67 ± 0.47	157.54 ± 0.28
Ours (C)	0.65 ± 0.01	12.99 ± 0.02	-14.64 ± 0.03	-9.65 ± 0.48	157.70 ± 0.28
MAF	0.45 ± 0.01	12.35 ± 0.02	-17.03 ± 0.02	-10.92 ± 0.46	156.95 ± 0.28
NAF†	0.62 ± 0.01	11.96 ± 0.33	-15.09 ± 0.40	-8.86 ± 0.15	157.73 ± 0.04
B-NAF†	0.61 ± 0.01	12.06 ± 0.09	-14.71 ± 0.38	-8.95 ± 0.07	157.36 ± 0.03
Q-NSF (AR)	0.66 ± 0.01	12.91 ± 0.02	-14.67 ± 0.03	-9.72 ± 0.47	157.42 ± 0.28
RQ-NSF (AR)	0.66 ± 0.01	13.09 ± 0.02	-14.01 ± 0.03	-9.22 ± 0.48	157.31 ± 0.28
Ours (AR)	0.66 ± 0.01	13.07 ± 0.02	-13.80 ± 0.03	-9.77 ± 0.55	158.39 ± 0.28

given in Table 2. The results in Table 2 demonstrate the competitive performance of linear rational splines with respect to other methods despite their simplicity. Note that Glow uses twice as many parameters as used in neural spline flows including linear rational functions. Furthermore, although residual flows perform better than our proposed method, they require much more computation time in the sampling process where they have to compute the inverse of each invertible ResNet using a fixed point iterative method. In contrast, linear rational splines have an analytic inverse which is also a linear rational spline.

Table 2: Bits per dimension (BPD) (lower is better) of the test set for four standard benchmark datasets. All the results except for our method are taken from the existing literature.

MODEL	MNIST	CIFAR-10	IMAGENET 32	IMAGENET 64
Real NVP	1.06	3.49	4.28	3.98
Glow	1.05	3.35	4.09	3.81
FFJORD	0.99	3.40	—	—
i-ResNet	1.05	3.45	—	—
Residual Flows	0.97	3.28	4.01	3.75
RQ-NSF (C)	—	3.38	—	3.82
Ours (C)	0.91	3.38	4.09	3.82

Table 3: Test set evidence lower bound (ELBO) (higher is better) and negative log-likelihood (NLL) (lower is better) in nats for MNIST (LeCun, 1998) and EMNIST (Cohen et al., 2017). All the results except for our method are taken from Durkan et al. (2019b). As in Durkan et al. (2019b), the NLL is estimated using an importance-weighted approach (Burda et al., 2016).

	MNIST		EMNIST	
MODEL	ELBO ↑	NLL ↓	ELBO ↑	NLL ↓
Glow	-82.25 ± 0.46	79.72 ± 0.42	-120.04 ± 0.40	117.54 ± 0.38
RQ-NSF (C)	-82.08 ± 0.46	79.63 ± 0.42	-119.74 ± 0.40	117.35 ± 0.38
Ours (C)	-82.23 ± 0.46	79.66 ± 0.42	-120.46 ± 0.40	117.74 ± 0.38
IAF/MAF	-82.56 ± 0.48	79.95 ± 0.43	-119.85 ± 0.40	117.47 ± 0.38
RQ-NSF (AR)	-82.14 ± 0.47	79.71 ± 0.43	-119.49 ± 0.40	117.28 ± 0.38
Ours (AR)	-82.02 ± 0.46	79.42 ± 0.42	-119.50 ± 0.39	117.23 ± 0.38

This means both the forward and inverse have the same cost. In particular, our experiments indicate that the sampling process in linear rational splines is faster than residual flows by an order of magnitude.

For a detailed information on the experiments and randomly generated sample images of the model, see Appendices B.3 and C.1, respectively. Also, more simulation results can be found in Appendix D.

4.4 Variational Auto-encoders

Finally, we test the performance of our proposed model in a variational auto-encoder (VAE) (Kingma and Welling, 2014) setting. In short, in a VAE data points are modeled as realizations of a random variable whose distribution is assumed to be the result of marginalization over a lower-dimensional latent variable. To make this process tractable, the type of prior and approximate posterior random variables need to be determined carefully. Like other flow-based models, linear rational spline flows can be used as effective models for priors and approximate posteriors in a VAE setting. For a detailed explanation on normalizing flows in the context of VAEs, we refer the interested reader to (Rezende and Mohamed, 2015).

Table 3 shows the quantitative results of VAE simulation using linear rational splines. As can be seen, our proposed model performs almost as well as other models such as Glow (Kingma and Dhariwal, 2018) and rational quadratic splines (Durkan et al., 2019b). For more details on the experimental configuration and model samples see Appendices B.4 and C.2, respectively.

5 CONCLUSION

In this paper, we investigated the use of monotonic linear rational splines in the context of invertible generative modeling. We saw that this family of piecewise functions have the advantage that their inverse is straightforward, and does not require solving degree 2 or 3 polynomial equations. Furthermore, since the same family of functions defines both the forward and inverse, investigation of the mathematical properties of these models is more straightforward. Also, we showed that despite their simplicity, they could perform competitively with more complicated methods in a suite of experiments on synthetic and real datasets.

Acknowledgements

We would like to thank the reviewers for their valuable feedback on our work, helping us to improve the final manuscript.

This research was undertaken using the LIEF HPC-GPGPU Facility hosted at the University of Melbourne. This Facility was established with the assistance of LIEF Grant LE170100200.

References

Behrmann, J., Grathwohl, W., Chen, R. T. Q., Duvenaud, D., and Jacobsen, J. (2019). Invertible residual networks. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 573–582.

Burda, Y., Grosse, R. B., and Salakhutdinov, R. (2016). Importance weighted autoencoders. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*.

Cao, N. D., Aziz, W., and Titov, I. (2019). Block neural autoregressive flow. In *Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence (UAI)*, page 511.

Chen, R. T. Q., Behrmann, J., Duvenaud, D., and Jacobsen, J. (2019). Residual flows for invertible generative modeling. In *Proceedings of the Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 9916–9926.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. (2018). Neural ordinary differential equations. In *Proceedings of the Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 6572–6583.

Chen, X., Kingma, D. P., Salimans, T., Duan, Y., Dhariwal, P., Schulman, J., Sutskever, I., and Abbeel, P. (2017). Variational lossy autoencoder. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*.

Chrabaszcz, P., Loshchilov, I., and Hutter, F. (2017). A downsampled variant of ImageNet as an alternative to the CIFAR datasets. *CoRR*, abs/1707.08819.

Cohen, G., Afshar, S., Tapson, J., and van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. *CoRR*, abs/1702.05373.

Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Li, F. (2009). ImageNet: A large-scale hierarchical image database. In *Proceedings of the 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255.

Dinh, L., Krueger, D., and Bengio, Y. (2015). NICE: non-linear independent components estimation. In *Workshop Track Proceedings of the 3rd International Conference on Learning Representations (ICLR)*.

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

Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. (2019a). Cubic-spline flows. In *Workshop on Invertible Neural Nets and Normalizing Flows of 36th International Conference on Machine Learning (ICML)*.

Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. (2019b). Neural spline flows. In *Proceedings of the Advances in Neural Information Processing Systems 3232: Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 7511–7522.

Durkan, C. and Nash, C. (2019). Autoregressive energy machines. In *Proceedings of the 36th International Conference on Machine Learning, (ICML)*, pages 1735–1744.

Fuhr, R. D. and Kallay, M. (1992). Monotone linear rational spline interpolation. *Computer Aided Geometric Design*, 9(4):313–319.

Germain, M., Gregor, K., Murray, I., and Larochelle, H. (2015). MADE: masked autoencoder for distribution estimation. In *Proceedings of the 32nd International Conference on Machine Learning, (ICML)*, pages 881–889.

Grathwohl, W., Chen, R. T. Q., Bettencourt, J., Sutskever, I., and Duvenaud, D. (2019). FFJORD: free-form continuous dynamics for scalable reversible generative models. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Huang, C., Krueger, D., Lacoste, A., and Courville, A. C. (2018). Neural autoregressive flows. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 2083–2092.

Hutchinson, M. F. (1990). A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 19(2):433–450.

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

Kahn, H. (1955). Use of different monte carlo sampling techniques. *Santa Monica, CA: RAND Corporation*.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*.

Kingma, D. P. and Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. In *Proceedings of the Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 10236–10245.

Kingma, D. P., Salimans, T., Józefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016). Improving variational autoencoders with inverse autoregressive flow. In *Proceedings of the Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 4736–4744.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*.

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto.

LeCun, Y. (1998). The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., and Han, J. (2019). On the variance of the adaptive learning rate and beyond. *CoRR*, abs/1908.03265.

Loshchilov, I. and Hutter, F. (2017). SGDR: stochastic gradient descent with warm restarts. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*.

Martin, D. R., Fowlkes, C. C., Tal, D., and Malik, J. (2001). A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings of the 8th International Conference On Computer Vision (ICCV-01)*, pages 416–425.

Müller, T., McWilliams, B., Rousselle, F., Gross, M., and Novák, J. (2019). Neural importance sampling. *Accepted for publication in ACM Transactions on Graphics*.

Oliva, J. B., Dubey, A., Zaheer, M., Póczos, B., Salakhutdinov, R., Xing, E. P., and Schneider, J. (2018). Transformation autoregressive networks. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 3895–3904.

Papamakarios, G., Murray, I., and Pavlakou, T. (2017). Masked autoregressive flow for density estimation. In *Proceedings of the Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 2338–2347.

Papoulis, A. and Pillai, S. U. (2002). *Probability, random variables, and stochastic processes*. Tata McGraw-Hill Education.

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

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 15(1):1929–1958.

Tabak, E. G. and Turner, C. V. (2013). A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164.