Isomorphism, Normalizing Flows, and Density Estimation:
Preserving Relationships Between Data

Steven Walton[1], [1]SHI Lab @ U of Oregon

Abstract

Normalizing Flows are a powerful type of generative model that transforms an intractable distribution of data into a more desirable one through the use of bijective functions. Their concept is simple to understand and create powerful models that allow one to work with much simpler distributions than that of the underlying data. In essence, our motivation in studying Normalizing Flows is to preserve relationships between data. Normalizing Flows have a wide variety of uses and applications, providing important statistical information about data as well as enabling more interpretable control over latent structures. They can be used for mathematical applications, such as: variational inference, density estimation, anomoly detection, manifold analysis; as well as more application focused works, such as: pose estimation, speach generation, image generation, and more. An important aspect of Normalizing Flows is that they preserve the latent structure of the data that they are trained upon, which makes gives them to the power to perform the aforementioned tasks. In this review we provide an introduction to Normalizing Flows, clarifying how they differ from other popular generative models, provide an updated overview of the current literature, discuss their applications, as well as the future of these models and how they can play a critical role in AI research. We also aim to clarify the distinction between different generative models in a more clear and precise way than many other works. We aim to make this work a sufficient introduction to Normalizing Flows and be self-contained with little to no background required.

1. Introduction

Defining probability distributions and having structured data is a common goal for many domains in science and application development. Unfortunately, many distributions are intractable in nature; distributions that are difficult to compute or express mathematically. This intractability is common across modern datasets and makes analyzing, interpreting, and controlling the data difficult. Most methods accept losses in interpretability and structure so that the data can be worked with, with many machine learning models being referred to as "black boxes." Normalizing Flows provide a framework to work with these distributions without loss of algebraic structure or dimensionality, making them important tools for many domains.

While many other models are substantially more popular, we believe that Normalizing Flows play an important role in the future of Artificial Intelligence and Machine Learning. Normalizing Flows are often considered to have a higher barrier to entry than other types of machine learning models due to the mathematics required for them. In this review we hope to demystify this, provide a complete overview of the literature, and motivate more research into the field.

There are three important prior surveys that complement this work. The thesis from Papamakarios [131] provides a detailed introduction and has a focus on density estimation and likelihood free inference. Kobyzev *et al*. [98] provides a more compact and up to date review that is valuable to those first being introduced to the topic as well as providing a curated quantitative comparison of works. The review from Papamakarios *et al*. [132] provides a more detailed review and is more tutorial in nature. In this work we aim to provide a sufficient introduction to the work, demystify the mathematical concepts, and bring the reader to an understanding of the current literature and motivation.

This work is intended to be read from start to finish, building upon prior sections, but readers may find it helpful to skip certain sections depending on their motivation and experience. We will begin with a description of Normalizing Flows Section 2 – providing simple definitions for many of the mathematical terms that are commonly used Section 2.2, a concise definition of of Normalizing Flows Section 2.3, and a brief overview of generative models and how they differ Section 2.4. We will then give an overview of the current methods Section 3 – providing a detailed description and how the works have built upon one another, giving the necessary background to understand the motivations and limitations. After that we will discuss some of the many applications of Normalizing Flows Section 4 – providing motivation and how the nature of flows have many advantages to certain applications. Finally, in Section 5 we will discuss the current state of Normalizing Flows and some future directions in research and their importance to the Machine Learning, Data Science, and Artificial Intelligence.

It is important to remember that mathematics, and as a consequence machine learning, is not the study of data or objects, but rather the relationships between the data [142]. With this in mind, **our primary motivation in studying Normalizing Flows is to preserve *all* relationships that are inherent to the data.** This distinguishes them from other architectures and it is important to keep this goal in mind. In essence, a Normalizing Flow provides a record of transformations wherein the original relationships between data can be (approximately) recovered without loss. In this respect, it also becomes important to remember that research can highly valuable even if results may seem worse or there are increases in computational costs, as the preservation of the relationships within the data remains the primary motivation.

2. Normalizing Flows

In this section we will provide our notation Section 2.1, a brief description of mathematical definitions useful to the discussion Section 2.2, we will then define Normalizing Flows in Section 2.3, and finally give a brief overview of generative models and how other methods compare Section 2.4.

2.1. Notation

We will use a fairly standard notation: where variables are in bold, such as $\mathbf{x}$, $\mathbf{x}_i$, $\vec{\mathbf{x}}_i$ (when we know explicitly a vector), or $\mathbf{M}$ where a capital explicitly denotes a matrix (tensor >rank 2); and constants are written as lowercase non-bold characters. Usually constants will take on letters from the early alphabet (abc...) or greek letters, indices from the middle starting with i ($ijk\cdots$), and variables take on the best letter to represent what it itself represents.Constant values will be not bold and take the form as a, $a \in \mathbb{R}$, $a_i \in \mathbb{R}$, or $a_i \in \mathbb{R}^2$. Generally, constants will be written with Greek letters to reduce ambiguity. Otherwise, we will use capital letters to represent tensors and arrays of arbitrary ranks: $\mathbf{M}$. Probabilities will be specified with $\Pr(\cdot)$ and $\Pr(\cdot|\cdot)$ representing conditional probabilities. $p(\cdot)$ or $q(\cdot)$ will represent a probability density, and we will use a subscript when it is helpful to refer to the variable they are being taken with respect to $p_x(\vec{\mathbf{x}})$. We will also use lower case letters to represent random variables and capital letters to represent distributions or sets from which they will be drawn from. In some cases we will still have ambiguous notation but we will make this as clear as possible. ∇f will represent the gradient, such as $\nabla_\theta f = \left[\frac{\partial f}{\partial \theta_0}, \cdots, \frac{\partial f}{\partial \theta_n}\right]$. In the case of the Jacobian we will say $J_f(\mathbf{x})$ for the Jacobian of function f with respect to $\mathbf{x}$. Finally, we will be borrowing notation from Einsteinian Notation and remove summations where they are implicit. Therefore we have $\sum_{i=0}^{N} \vec{\mathbf{x}}_i\vec{\mathbf{y}}_i \equiv \vec{\mathbf{x}}_i\vec{\mathbf{y}}_i$. We believe this will make the equations more compact and more readable, especially as packages like *einsum* are becoming more popular and is more natural to programming. Other notations will be introduced later.

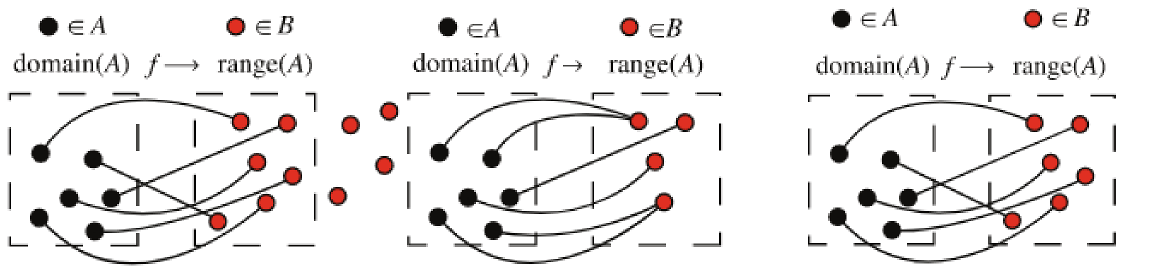
2.2. Math Definitions

In this section we will include important math definitions that will be necessary for understanding the subsequent sections. Specifically, when we introduce our definition and classification for generative models (Section 2.4) this section will become useful. This section may be skipped and it is okay if the novice reader does not understand every definition from the first reading. We will mix both more formal mathematical descriptions and simpler ones to help readers parse works, as it is common for more rigorous notation to be used. The importance of this section is so that we may have a common language and demystify seemingly complex concepts. Unfortunately, this mathematical language creates a barrier for many wishing to understand Normalizing Flows but we believe that the concepts can be easily understood by anyone with a basic understanding of Linear Algebra and Calculus.

Some of this confusion comes from the fact that researchers come from different backgrounds and may use different terminology that may be functionally identical. Additionally, some terminology is used in both studies but may contain slightly different terminology. In this respect this section will at least unify the terminology used within this document. It also means that there may be disagreement. In this regard we welcome others to help us create more precise terminology, as we will see its importance and the goals at hand later in Section 2.3 and Section 2.4.

We will begin with definitions from set theory, then category (and topology), and include some definitions from measure theory, doing our best to place these terms in the proper category. For a more in depth and detailed description of set theory, topology, and category theory we recommend nLab [127] while for measure theory and probability theory we recommend Shao [162], but note that both may not be the best resources for novices [27, 103, 117]. Such a complete understanding will not be necessary for this work and in general is not necessary for research in Normalizing Flows. Most of these definitions are deceptively simple, and we believe when put in simpler terms most readers will feel comfortable with them.

First we must understand three important terms from set theory: *injection*, *surjection*, and *bijection*. The definitions are below and a visual depiction is given in Figure 1.



(a) Injection: one-to-one (b) Surjection: onto (c) Bijection: one-to-one and onto

Figure 1. Visual representation of injections(Definition 1), surjections (Definition 2), and bijections (Definition 3). Source: *Wolfram Mathworld*

Definition 1. Injection (One-To-One): *An injective function is who's Domain has a unique representation within the Range.*

$$\forall \mathbf{x}_i \in \mathbf{A} \; \exists \, \mathbf{y}_i \in \mathbf{B} \;\text{ s.t. }\; f(\mathbf{x}_i) = f(\mathbf{x}_j) \text{ iff } i = j$$

An injective function is one where every point from set **A** has a unique representation in set **B**, but not every point in set B needs to have a corresponding point in **A**.

Definition 2. Surjection (Onto): *A surjective function who's Domain covers the Codomain.*

$$\forall \mathbf{x}_i \in \mathbf{A} \; \exists \, \mathbf{y}_j \in \mathbf{B} \; \forall \mathbf{y}_j$$

A surjective function is one where each point in set **B** has a corresponding point from set **A** but we do not need a unique representation.

Definition 3. Bijection (One-To-One & Onto): *A bijective function both injective and surjective.*

$$\forall \mathbf{x}_i \in \mathbf{A} \;, \; \forall \mathbf{y}_i \in \; \exists \, f(\mathbf{x}_i) = \mathbf{y}_i \;\text{ s.t. }\, f(\mathbf{x}_i) \neq f(\mathbf{x}_j)$$

Simply put, a bijection is a function where every point in set **A** has a unique representation in set **B**, and all points in **B** have a unique representation in **A**.

This gives us the following corollary, which is essential to the understanding of Normalizing Flows.

Corollary 1. *All bijective functions are invertible.*

In other words, if f is a bijector (a bijective function) then f has an inverse such that $f^{-1}(f(\mathbf{x})) = f(f^{-1}(\mathbf{x})) = \mathbf{x}$. Since we always have a unique map between the two sets we are always able to travel back and forth between **A** and **B**. This means that we can recover the original orientation of the data.

Theorem 1. Cantor-Schröder–Bernstein Theorem: *Let* $f : \mathbf{A} \mapsto \mathbf{B}$ *be an injection between sets* **A** *and* **B**, *and let* $g : \mathbf{B} \mapsto \mathbf{A}$ *be an injection between* **B** *and* **A**. *Then there must exist a bijection* $h : \mathbf{A} \mapsto \mathbf{B}$.

Understanding this simple idea is critical, and may not always be obvious. Such a

We will also introduce some commonly used definitions from topology and category theory that are frequently seen. The definitions are simple but if one is unfamiliar they may seem incomprehensible. Many of them are similar but there are some distinct differences. Note that some authors may confuse terms between set theory and category theory, which is a major motivation for providing these definitions. Note that set theory describes relationships between elements (objects) in a set while category theory describes the relationship between the objects.

Definition 4. Morphism: *A map between two categorical objects.*

A morphism is simply any mapping function. This can be between sets, like we have used above, manifolds, or any other object. Sometimes this is simply defined as an arrow between two objects, as in $f : A \mapsto B$. We can also refer to a category as a collection of arrows or morphism. Sometimes Category Theory is even referred to the study of "dots and arrows." A morphism is the more abstract version of a homomorphism and in general will be considered the same.

Definition 5. Homomorphism: *A morphism between two structured sets that preserve the algebraic structure.*

$$\begin{aligned} f(a+b) &= f(a) + f(b) \\ f(ab) &= f(a)f(b) \end{aligned} \tag{1}$$

This term is often used synonymously with morphism but we need to note that it implies that there is an underlying structure [19, 28]. Mathematical structure is also vaguely defined but refers to any extra properties or intrinsic relationships for the data. The term is general as mathematics is the study of relationships between objects [142], and thus we must simply think of these as algebras. Here we are more specifically discussing the algebra or operators relating set elements. We note that a *functor* is a homomorphism of categories, and we can think of these as functions. We may call a functor "forgetful" if some structure is lost.

Definition 6. **Isomorphism:** *An invertible morphism*

An isomorphism can generally be thought of as a bijection but note that an isomorphism is the relationship between categories. We may similarly say that an epimorphism (epic) is a generalization of surjection and monomorphism (monic) is a generalization of injection. An isomorphism is epic and monic. Specifically, an isomorphism specifies that two sets are equal to one another. While this is generally thought to be structure preserving, this is noth always true, though it may be recovered. A quick example is a function f that maps set $A = \{0,1,2\}$ to set $B = \{1,0,2\}$, in which we see that order is not preserved, and thus the distance between two elements is not equal: $d_A(a_0, a_1) \neq d_B(b_0, b_1)$. But we can see that applying f onto the distance preserves the relationships: $f(d_B(b_0, b_1)) = d_A(a_1, a_0)$. A simple example of a homomorphism that is not an isomorphism is the map $f : \mathbb{Z} \mapsto 2\mathbb{Z}$, where we can see that all the mathematical operators are preserved and exist but not all elements or their relationships are.[1]

Definition 7. **Homeomorphism:** *An isomorphism of topological spaces.*

A homeomorphism, not to be confused with homomorphism, specifically refers to the topological category. We can say that a homeomorphism $f : X \mapsto Y$ is a continuous topological map and its inverse $f : Y \mapsto X$ is also a continuous topological map. To clarify this, we note that a topological space is continuous collection of points, such as a line or the surface of a plane. That is, they have an infinite collection of points. This is distinguished from a set which may be finite. Specifically, an isomorphism is between algebraic structures while a homeomorphism is between topological spaces.

Definition 8. **Diffeomorphism:** *A map between two k-times differential manifolds that has a differentiable inverse.*

In simpler terms, this means that we have a topological space, such as a surface, that we can differentiate over a local (small) region. This is quite similar to homeomorphism, but we need to be careful to note that not all homeomorphisms are diffeomorphisms. For example, if we have the function $f(x) = x^3$, where $f : \mathbb{R} \mapsto \mathbb{R}$, we will note that the inverse is not differentiable at the origin. This point This function is homeomorphic but not diffeomorphic. The distinction here is important as many works refer to Normalizing Flows as specifically diffeomorphic.

[1]A careful reader will notice that the ring $\mathbb{Z}_2$ is isomorphic to the factor ring $\mathbb{Z}/2\mathbb{Z}$ [6].

Definition 9. **Hausdorff Space:** *A topological space where any two distinct points within that space have a disjoint open neighborhoods.*

$$x_i \in U \subset X, x_j \in V \subset X \text{ s.t. } x_i \neq x_j \text{ , } U \cap V = \{\emptyset\}$$

A Hausdorff space is deceptively trivial and in the simplest of terms means that we are able to distinguish points within this space. We can think of the local neighborhoods, U and V, around the points as small regions, and in a continuous space these converge to infinitesimal points. Most topological spaces that the reader is familiar with are Hausdorff.

We also have important definitions from measure theory that are commonly used. Notably, probability theory is a subset of measure theory which specifically studies the probability measure. The verbiage here can be easily confused, so special attention is needed. A *measure* is a mathematical extension of size, such as the length, area, or volume of a subset. We will use the notation of Shao [162], and first define a sample space (outcome space), Ω, as the set of all possible outcomes. Ω is in general what will be of interest.

Definition 10. **σ-algebra:** *A collection of subsets, $\mathcal{F}$, in sample space Ω is a σ-algebra (or σ-field) if (i)the empty set is in $\mathcal{F}$, (ii) the complement of any element in $\mathcal{F}$ is also in $\mathcal{F}$, and (iii) the union of any element in $\mathcal{F}$ is also in $\mathcal{F}$.*

$$\begin{aligned}(i)&\ \ \emptyset \in \mathcal{F}\\ (ii)&\ \ A \in \mathcal{F} \implies A^c \in \mathcal{F} \text{ s.t. } A \cap A^c = \emptyset\\ (iii)&\ \ A_i \in \mathcal{F}\ \forall i \implies \cup A_i \in \mathcal{F}\end{aligned}$$

A σ-algebra (also called σ-field) is a complex way of saying that things work as we'd expect them to. This is more easily understood through an example. If we trained a neural network classifier then Ω represents all possible valid outputs/outcomes of our network. An example of $\mathcal{F}$ could be all the animals that our network can classify, A can be all the cats, and A_i can be a specific cat breed. Then we read (ii) as saying all classifiable cats and all classifiable non-cat mammals are are classifiable, and that no classifiable cat is also a classifiable non-cat mammal (e.g. a cat is not a dog). (i) is then bookkeeping, and is necessary since cats aren't dogs. Then (iii) means that if we group every possible classifiable cat then the group is also classifiable. Note that order isn't specified. This should all sound obvious since we're only working with classifiable classes, but math can be tedious and such tediousness is unfortunately necessary. The example is a simplification, but should help readers trivialize this notation which is common.

We also say that a pair $(\Omega, \mathcal{F})$ is a **measurable** space, not to be confused with *measure* space. The definition here is now apparent as from our understanding of a σ-algebra we can *measure* everything, since we have an outcome.

Definition 11. <u>**Measure:**</u> *If $(\Omega, \mathcal{F})$ is a* ***measurable*** *space, then a measure ν is defined on $\mathcal{F}$ iff (i) the measure is non-negative, (ii) the measure of the empty set is 0, and (iii) the measure of the union of disjoint subsets A_i is equal to the sum of the measures of the disjoint sets.*

$$
\begin{aligned}
(i)&\ \ 0 \leq \nu(A) \leq \infty\ \ \forall A \in \mathcal{F}\\
(ii)&\ \ \nu(\emptyset) = 0\\
(iii)&\ \ \nu(\cup A_i) = \nu(A_i)\ \ \forall A_i\ \text{ s.t. }\ A_i \cap A_j = \emptyset \forall i \neq j
\end{aligned}
$$

We can think of a *measure* as a distance, and so (i) says that distances are never negative (they don't have direction), (ii) says that if we don't *measure* anything that the *measure* is 0, and (iii) says that if we *measure* the group of collection of all items in A that it is the same as the sum (implicit) of measuring each individual item. Again, notice that order is not specified in (iii), as other definitions may discuss symmetry. That says essentially that $\nu(A)$ is well defined.With this, we then call a *measure space* $(\Omega, \mathcal{F}, \nu)$. More abstractly a *measure* can be thought of as a function that assigns numbers.

If we then let the measure of all possible outcomes sum to 1 ($\nu(\Omega) = 1$), then we call the *measure* a probability *measure* and the space a <u>probability</u> space. Similarly, we could say that a <u>probability</u> space is $\left([0,1], \mathcal{B}_{[0,1]}\right)$, where $\mathcal{B}$ is the Borel *measure*. A Borel σ-algebra is the collection of all finite open intervals on the real numbers ($\mathbb{R}$), a Borel *measure* is a *measure* defined on all open sets, and $\mathcal{B}_{[0,1]}$ is defined for all open sets on the closed set $[0,1]$. You can now see how our probability rules all rise out of this simple definition.

Definition 12. <u>**Pushforward Measure:**</u> *Let $(\Omega, \mathcal{F})$ and $(\Lambda, \mathcal{G})$ be measurable spaces, ν be a measure on $(\Omega, \mathcal{F})$, and f be a measurable function $f : (\Omega, \mathcal{F}) \mapsto (\Lambda, \mathcal{G})$, then the pushforward, $f_*\nu$, of measure ν defines the subset of $(\Lambda, \mathcal{G})$ that is in the preimage under f.*

$$
\begin{aligned}
(f_*\nu)(B) &= \nu(f^*(B))\ \text{ where }\ B \subseteq (\Lambda, \mathcal{G})\\
\nu \circ f^*(B) &= \nu(f^*B)\ \text{ where }\ B \subseteq (\Lambda, \mathcal{G})
\end{aligned}
$$

The definitions are equivalent with $(f_*\nu)(\cdot)$ and $\nu \circ f^*(\cdot)$ both being called the pushforward. Essentially, the pushforward a subset of the map where the measure is valid in both measurable spaces. The preimage, f^*, is sometimes called the inverse and written as f^{-1}, and is simply all the elements from the domain into the codomain. If we say that $B \subseteq (\Lambda, \mathcal{G})$ then we would define the preimage as $f^*(B) = \{a \in (\Omega, \mathcal{F}) | f(a) \in B\}$. This does not imply bijection though! So here we can see that our pushforward measure can more accurately described as the a function that maps elements form one measurable space to another.

With this understanding we may find some authors refer to the preimage, $f_*(\cdot)$ as the pullback. In the context of differential geometry these can be clarified. If our function f is diffeomorphic then it is a pushforward and the pullback is identical to the inverse, viewing this as simply a change of coordinates. Similarly, for morphisms we can say that f_* is in the covariant direction ($\rightarrow$) while f^* is in the contrivariant direction ($\leftarrow$).

We should now have a common enough language to accurately describe our networks. This is an unfortunate necessity but will be extremely valuable and let us understand our networks at a far deeper level.

2.3. Normalizing Flow Definition

Unfortunately, Normalizing Flows have been described several different ways. For the most part, the definitions are quite similar and there is an understanding that authors are referring to the same thing. The first usage of the phrase that we are aware of that by Tabak *et al*. [169] and then more formally described in Agnelli *et al*. [2], noting that there are shared authors. [169] proposes performing density estimation of an intractable probability function by mapping data $\mathbf{x} \sim \mathbf{X}$ to variables $\mathbf{y} \sim \mathbf{Y}$ with a known probability distribution $q(\mathbf{y})$: $p(\mathbf{x}) = J_{\mathbf{y}}(\mathbf{x})q(\mathbf{y}(\mathbf{x}))$. Here they describe their map as "an infinite composition of infinitesimal transformations", dubbing the term "flow". They note that their process is similar to the Gaussanization process of Chen and Gopinath [26]. It is notable here that they set up the problem such that the estimated distribution is exactly the original distribution, $\tilde{p}(\mathbf{x}) = p(\mathbf{x})$, and restrict their maps to ensure this. Papamakarious *et al*. [132] notes the relationship of Flows, Gaussanization, and *Whitening Transforms* [44] but we believe is categorically different and closer to diffusion models [66].Agnelli *et al*. [2] changes the definition slightly, focusing on classification through density estimation. They say that a parametric family of distributions $p_k(\mathbf{x};\theta)$ of k classes characterized by a map $J_k(\mathbf{x})$ with a common target distribution $q(\mathbf{y})$. Specifically, $p_k(\mathbf{x}) = J_k(\mathbf{x})q(\mathbf{y}_k(\mathbf{x}))$, where again $J_k(\mathbf{x})$ is the Jacobian and $J_k : \mathbf{x} \mapsto \mathbf{y}_k$. Dinh *et al*. [33] produced what is commonly considered the first Normalizing Flow, but did not use this name. They described their framework as the composition of invertible functions, f, through change of variables.

$$
p(\mathbf{x}) = q(f(\mathbf{x}))|\det J_f(\mathbf{x})| \tag{2}
$$

$|\det J_{\mathbf{x}}|$ is the absolute value of the Jacobian determinant of $\mathbf{x}$. We will also frequently write this in terms of the "log density", as this is often easier to compute:

$$
\log p(\mathbf{x}) = \log\left(q_i(f_i(\mathbf{x}))\right) + \log\left|\det J_j(\mathbf{x})\right| \tag{3}
$$

Where $f(\mathbf{x}) = (f_i(\mathbf{x}))_{i \leq N}$. We can see here that the composition of flows is entirely dependent on the log absolute Jacobian determinant, and that the log density $q(\cdot)$ is seperable from the Jacobian.

We note here that this looks similar to the change of variables for Riemann integrals:

Theorem 2. **Change of Variables (Calculus):**
Let $f : \mathbb{R}^n \mapsto \mathbb{R}^n$ be $\mathbf{y} = f(\mathbf{x})$ where f is 1-differentiable, then

$$\int_Y g(\mathbf{y})d\mathbf{y} = \int_X g(f(\mathbf{x}))f'(\mathbf{x})d\mathbf{x} \tag{4}$$

$\int g(\mathbf{y})d\mathbf{y} = \int g(f(\mathbf{x}))f'(\mathbf{x})d\mathbf{x}$ where $\mathbf{y} = f(\mathbf{x})$ and $f'(\mathbf{x})$ is the derivative of f (the Jacobian). Theorem 1.16 of Shao [162] defines a change of variables on a **measurable** space as:

Theorem 3. **Change of Variables (metric):**[2]
Let f be a measurable from $(\Omega, \mathcal{F}, \nu)$ to $(\Lambda, \mathcal{G})$ and g be Borel on $(\Lambda, \mathcal{G})$, then

$$\int_\Omega g \circ f d\nu = \int_\Lambda gd\left(\nu \circ f^*\right) \tag{5}$$

This also means that if either integral exists then the other does, and the two are the same. Dinh *et al.* is the first to explicitly discuss change of variables and the use of invertible functions. Following, Rezende and Mohamad [152] follows, explicitly saing "A Normalizing Flow describes the transformation of a probability density through a sequence of invertible mappings."

Kobyzev *et al.* [98] better formalizes the definiton through the language of measure theory.

Theorem 4. *Let* $\mathbf{Y}$ *be a tractable distribution with known distribution $q : \mathbb{R}^n \mapsto \mathbb{R}$, let g be in invertable function such that $\mathbf{X} = g(\mathbf{Y})$, and $f = g^{-1}$. Then*

$$\begin{aligned} p(\mathbf{x}) &= q(f(\mathbf{x}))\left|detJ_f(\mathbf{x})\right| \\ &= q(f(\mathbf{x}))\left|detJ_g(\mathbf{x})\right|^{-1} \end{aligned} \tag{6}$$

$p(\mathbf{x})$ is the *pushforward* of q specified as g_*q. g was selected to note that this is the *generative direction*, pushing our tractable distribution towards the target intractable one. This makes f the *normalizing direction*. With a less restrictive function $g : \mathcal{Y} \mapsto \mathcal{X}$ they say that using a pushforward measure defines a generative model within the context of transportation theory [182]. They specifically restrict all σ-algebras to be Borel and all measures to be continuous with respect to the Lebesgue measure.[3] This gives us the following definition

Definition 13. **Normalizing Flow (Kobyzev *et al.*):**
Given a function between two standard Boreal spaces $g : (\Omega, \mathcal{B}_0) \mapsto (\Lambda, \mathcal{B}_1)$ such that g, g^{-1} are differentiable almost everywhere with respect to the Lebesgue measure, then the composition of pushforward functions describes a Normalizing Flow.

$$\begin{aligned} p(\mathbf{x}) &= (q \circ g_0(\mathbf{Y})) \circ \cdots \circ (q \circ g_n(\mathbf{Y}))) \\ &= g_*^0 q(\mathbf{y}) \circ \cdots \circ g_*^n q(\mathbf{y}) \end{aligned} \tag{7}$$

We commend Kobyzev *et al.* for this extension as this definition makes the understanding of Normalizing Flows more general for the study of non-Euclidean spaces, as they pointed out.

We would like to abstract this definition even further. We will do so in the context of category theory, and we believe this generalizes the concept of Normalizing Flows and even makes the concept easier to understand. In Section 2.4 we will also discuss other generative models under a similar fomulation.

Definition 14. **Normalizing Flow:** *The composition of isomorphisms constitutes a Normalizing Flow*

This should feel rather straight forward, but the distinction is critical. The goal of Normalizing Flows has been to transform distributions into ones that are more meaningful. But abstracting this, we need not think of just distributions, but objects in general. Kobyzev *et al.* made the extension as not to restrict the models to Euclidean spaces. This should remind us of our goal in preserving the relationships between data. While data transformer through an isomorphism my not itself preserve all such relationships, they provide a record through which all such relationships may be recovered. This is our main motivation with respect to Normalizing Flows, and through this lens we can see how these models provide a more interpretable framework within the context of machine learning.

We believe that this framing can help our understanding of different machine learning frameworks and help unify the field and language around it. With the language of Isomorphic Flows we better clarify that there not be a need to work strictly withing the probability metric, manifolds, not even topological spaces in general. This framing can help us understand the usage in networks such as Graph Neural Networks [106], Implicit Models [37, 109], and discrete networks [13, 65, 68, 175], which all do not require a topological space.[4] Most importantly, we believe that this framing will help in the understanding of multi-modality. Similarly, we believe that this framing will help for those wishing to understand these models within the context of frameworks like Symbolic Reasoning [177], Implicit Modeling [37, 109], and Neural ODEs (Section 3.7).

Additionally, we believe that this framing will be beneficial in the advancement towards Artificial General Intelligence. We observe in human and animal intelligences

[2] $\nu \circ f^{-1}$ is called an induced measure by f. Shao uses the notation f^{-1}, which we have changed, as he defines $f^{-1}(B) = \{f \in B\} = \{\omega \in \Omega : f(\omega) \in B\}$ which we identify as the definition of a Pushforward Measure from Definition 12. He carefully notes "The inverse function f^{-1} need not exist for $f^{-1}(B)$ to be defined." See Section 1.1.2 for more.
[3]

[4] Note the difference between "a topology" and "topological space." But we may also be interested in learning relationships between sets who's intersection is not empty.

that they can form relationships between more general objects and concepts. More specifically, humans are able to learn deep abstractions, thus necessitating the exploration of learning through these abstractions. With this in mind, it is still useful to understand the representations between objects in a manner that is able to preserve and maintain inherit structures.

For the purpose of this work, we will be focusing on Homeomorphic Flows, and more specifically Diffeomorphic Flows. With this respect, the definition of Kobyzev *et al*. [98] will be appropriate for the work described herein.

2.4. Generative Modeling

The definition of a generative model has often been quite vague and not distinctive. Classically we have used a framework wherein we have discriminative models, generative models, and reinforcement models. Definitions along the lines of *a function that specifies how to generate data* is commonly used [51, 123, 172], specifically from the probabilistic perspective. More specifically, this is generally considered learning a probability density function $p(\mathbf{x})$. Tomczak [172] specifically defines them as having the joint form $p(\mathbf{x}, y) = p(y|\mathbf{x})p(\mathbf{x})$, which can be reworked for the marinal $p(\mathbf{x})$. Murphy [123], agrees but specifies that a generative model can specifically run backwards: inferring the input data from $p(\mathbf{x}|y)$.

Additionally, many models that are called generative do not explicitly write a density function. GANs [52, 118, 122, 156, 183] do not directly model a probability distribution but rather develop a stochastic procedure to generate samples, such as images. Energy Based Models (EBMs) [37, 101] do not directly learn a distribution but free-energy, which is often called an "unnormalized distribution." Often these are learned through the gradient of the density function, which ignores the normalizing constant, called Stein Score Matching [76, 93].[5] This training procedure is even the backbone for many modern state of the work designs [66, 166, 167]. Furthermore, [55] showed that the logits of a classifier network can be reformulated as an EBM.

Goodfellow [51] suggested a taxonomy of Generative models, which we show a slight variation of in Figure 2. Originally it was described as maximum likelihood estimators, branching into explicit and implicit densities, following our explanation above which may more accurately be called Synthesis Learning [58]. [6] The approximate densities are models that do not have consistent estimators for the density or perform some reduction technique. [7]

Similar to our formulation for Normalizing Flows, we will formulate a different hierarchy. The concept of "data generating" is a weak distinction, since realistically all models can be seen as performing this task. Similarly, we do not want to restrict ourselves to assuming data is generated from a distribution as this may not always be true and obfuscates our understanding of such things as Symbolic Reasoning [177] Physics Informed Neural Networks [22, 147] since neural networks are often Universal Function Apprximators [70, 160] and can solve a wide variety of problems. We can more generally say that machine learning studies the relationship between data, just as in Category Theory we define dots and arrows. In general, we can think of all networks as Homomorphisms, as they can be abstracted to a function map. Homeomorphic models can be broken down into the class of Isomorphic (bijective), Epimorphic (surjective), and Monomorphic (injective) as defined by the relationship between the function map being described (in general we suggest the final representation). Isomorphic models are invertable, such as that of Normalizing Flows. We can further down classify our Isomorphic models into Homeomorphic models and Diffeomorphic models depending if they are operating on topological spaces (assuming the manifold hypothesis) or differentiable, respectively.

With this formulation we can more clearly distinguish certain models, such as between that of Diffusion Models and Normalizing Flows. For example, we would say that Diffusion Models are bi-Epic, since they formulate maps $f : \mathbb{R}^n \mapsto \mathbb{R}^n$ and $g : \mathbb{R}^n \mapsto \mathbb{R}^n$ through the use of surjections, and since the Gaussian Distribution doesn't have an inverse. Similarly, we can call Normalizing Flows Isomorphic, as they form maps $f : \mathbb{R}^n \mapsto \mathbb{R}^n$ and $f^{-1} : \mathbb{R}^n \mapsto \mathbb{R}^n$. Our typical classification network will be Monic, $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ where $m < n$. We can also see that works like GANs as inclusion maps, since they typically generate from a smaller latent representation.

We are not the first to introduce the concept of categories into machine learning and we note that we are far from the first to think of machine learning in the perspective of categories [23, 43, 47, 163], but we hope to have put a more simplistic and approachable view on this. In so, this formulation is not meant to be complete but rather a first step and will require further development and collaboration. We also believe that this formulation will help remind us of *how relationships between data is preserved or lost through our transformations.* This notion will help us to remember aspects of interpretability and our main motivation in researching Normalizing Flows. The rest of this work will focus specifically on Normalizing Flows and the their current status in Machine Learning. We will see concepts from cat-

[5] It should be noted that score matching is not a consistent estimator

[6] [58] breaks learning problems down into Synthesis –simulating patterns form the source – Restoration – recovering the original source pattern – Recognition – determining the pattern class – Extrapolation – inferring data that could not be observed – and Understanding – intrinsic and extrinsic understanding of the data knowledge.

[7] If both the data distribution generating hypothesis and the manifold hypothesis are true, then it is possible for a reduction technique to accurately calculate the density.

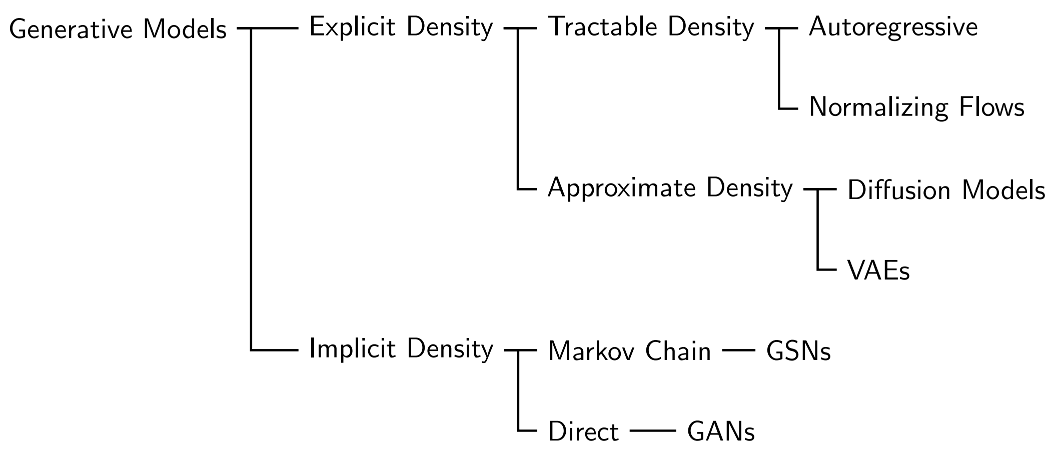


Figure 2. Slight variation on Goodfellow's Taxonomy of Generative Models [51]

egories, metric theory, and other domains and we help this work will help make these concepts clearer and the reader will be better suited to see the larger picture. From a more philosophical perspective, it is important to remember that many scientific advancements can be seen as recognizing the similarities between things that were thought to be different and we believe this abstraction will help encourage researchers to make these types of connections. We will see many examples of this throughout this work.

Unless otherwise stated, we will make the following assumptions:

- Data Distribution Assumption: Data is generated through the process of a data distribution.
- Manifold Assumption: Data lies on a topological manifold that is differentiable almost everywhere.
- i.i.d. Sample Assumption: Samples are independent and identically distributed.

3. Types of Flows

In this section we will discuss several types of flows and their benefits and limitations. Before we begin, we may wish to clarify some basic properties.

- To calculate the likelihood of a sample from the target distribution only the normalizing direction needs to be tractable.
- To generate samples from the base distribution, approximating the target distribution, the inverse needs to be tractable.

These points are important as we may have different computational constraints in the normalizing direction compared to the generating direction. Even though an inverse may exist, it is not always easy to calculate. Thus if we're only trying to calculate the likelihood of samples we may be able to develop a more expressive flow by using diffeomorphisms who's inverses are much more computationally cumbersome. The restricted nature of flows also means that we are imparting structural information onto the learning algorithms, or specific inductive biases. Flows are often difficult to construct as there needs to be a balance between computational complexity as well as expresstivity. In general, flows that are simple to compute are also not that expressive. In this section we will discuss these trade-offs and what structural information is being pushed onto the learning algorithm.

This section will start with the most basic flows and work its way to more complex formulations. This will not be a complete coverage of every work, but we believe that these works tell the necessary stories for the progression within these subclassifications. The intent is to build from the start so the motivation of the recent works can be accurately understood. This should, intentionally, make works feel incremental, but note that what seems obvious post hoc is not a priori. We intend our writing to complement these works, helping readers parse the complex topics and explain these concepts in simpler language. Section 3.1 introduces Element-wise Flows, which expand or contract the volume of the manifold, then Section 3.2 introduces Permutation, where permutations can be thought of as automorphisms, Section 3.3 introduces linear flows, a more expressive version of the elementwise ones, Section 3.4 flows have triangular Jacobian determinants, Section 3.5 introduces the concepts of Radial and Sylvester flows, which formulate structures as hyperplanes and hyperspheres, Section 3.6 introduces the use of splines residual networks which allow for free form Jacobian model, Section 3.7 introduces the other free form networks who are parameterized by differential equations.

3.1. Element-wise Flows

Performing operations element-wise is the simplest type of flow that one could generate. We often see this expressed in one of 3 equivalent ways:

$$\begin{aligned}
T(x) &= (h(\mathbf{x}_0), h(\mathbf{x}_1), \cdots, h(\mathbf{x}_n))^T \\
&= D\mathbf{x}^T \\
&= \mathbf{h}\mathbf{x}^T \\
T^{-1}(x) &= \mathbf{D}\mathbf{x}^T
\end{aligned} \tag{8}$$

In the first line we see that h ($\mathbf{h} \in \mathbb{R}^n$) is operating on each element in the $\mathbf{x}$ tensor. The second like D is a diagonal matrix ($D \in \mathbb{R}^{n\times n}$), and the third line is vector multiplication. In each of these cases the operation can be either learned or static. These simple types of flows have some desirable qualities because they both trivial to invert as well as calculate the Jacobian determinant. This is most obviously seen in the diagonal case as the determinant is the product of the diagonal elements $\det D = \prod d_{i,i}$. We can also see that the exprestivity is fairly low in these types of flows as they are only performing scalar operations and there is no correlations between dimensions. This lack of correlation means that we should not expect this type of flow to be able to solve complex problems on their own unless the features are already linearly independent. For a geometric interpretation it is best to conclude that these flows squeeze or expand elements. Due to this, the flow will fail to model sufficiently complex distributions without any form of mixing. Another way that we can think about this formulation is through the use of activation functions, as long as they are bijective. Works such as [3, 62, 116], which use learned activation functions, can be thought of as flows (in whole or part). Motivation in these papers included learning parameterized Leaky ReLUs (LReLU) to avoid the zero gradient (dead neuron) problems.

3.2. Permutation

Another extremely simple type of flow is one that simply permutes or rearranges elements. While these again aren't very expressive they are highly useful parts of more complicated networks as these permutations can make it easier to learn differing correlation between variables. For example, if a model has some inductive bias that assumes localized structure (often called "local inductive bias"), such as is seen in convolutions (CNNs) and is often assumed about images, then these permutations may help reduce the bias imposed on the model and allow for longer range communication between data points. Another useful feature of these types of flows is that they are volume preserving. Since we are just reordering elements, the absolute value of the Jacobian determinant will always be 1. Additionally, the inverses are almost always trivial to calculate. There are several common versions of these which we will discuss.

3.2.1 Simple Permutations

The random permutation is a fairly common type of permutation where we simply randomly permute the elements of the data. To invert this all we need to do is remember the ordering. Even simpler is simply inverting the data or reversing the order. Another common permutation is to swap dimensions, such as channels (we will see the usefulness of this later). One can think of various ways to perform similar types of operations but it is important to note that these types of operations are useful for adding additional complexity into other types of flows and will help us reduce the issues with our architectural constraints.

3.2.2 Squeeze Flows

Another common static permutation is the Squeeze, which shrinks one or more dimension into another. RealNVP [34] uses this operation to squeeze half the pixel dimensions, height and width, into channel dimensions. That is $T : \mathbb{R}^{c\times h\times w} \mapsto \mathbb{R}^{4c\times \frac{h}{2}\times\frac{w}{2}}$. This of course can be done with any factorization and does not require that it be 2. Mixing pixel space with channel space can give a few advantages, including: reducing reliance on local structure within pixel space and allow us to learn more complex channel-wise operations (more on this later).

3.2.3 Invertible 1×1 Convolutions

Glow [94] introduced a learnable permutation through the use of 1×1 convolutions.

$$\begin{aligned}
T(x) &= W\mathbf{x}^T + \mathbf{b} \\
T^{-1}(x) &= W^{-1}(\mathbf{x}^T - \mathbf{b})
\end{aligned} \tag{9}$$

This convolution has the same number of input and output channels thus the filter is shaped $W \in \mathbb{R}^{c\times c}$. Note that if W is not properly characterized then it may become a non-invertible matrix during the learning process. This type of convolution ends up being a generalization of a permutation matrix for channel permutations. The reasoning for this is that this matrix can be decomposed into a permutation matrix, lower, and upper triangular matrices. That is $W = PL(U + \text{diag}(\mathbf{s}))$, where P is a permutation matrix (a binary orthogonal matrix), and L and U are triangular matrices. Typical LU factorizations will resolve with either L or U having ones on the diagonals, unitriangular. Thus the triangular matrix, say U, can be further decomposed into $U = U' + \text{diag}(\mathbf{s})$, where U' is unitriangular. Differentiating W can be quite costly, $\mathcal{O}(n^3)$, although this is typically not that expensive as in practice this is operating on channels (e.g. for a 3 channel image $\mathcal{O}(e^3) = \mathcal{O}(9)$). The *LU* decomposition makes this even easier the determinant of permutation matricies and unitriangular matrices are 1, and thus we just need the sum of $\mathbf{s}$. If W is parameterized directly in terms of $\mathbf{s}$ then our computation drops to constant time, $\mathcal{O}(1)$.

In GLOW the authors initialized their matrix by first sampling a random rotation matrix, performed *PLU* decomposition, and allowed L, U, and $\mathbf{s}$ to be optimized. Note that P here is fixed and that this limits flexibility.[8] [67] recognized this limitation and instead decided to perform *QR* factorization, where Q is orthogonal and R is strictly triangular. They also imposed that R be unitriangular using a diagonal matrix like above: $R = R' + \text{diag}(s)$. The Jacobian determinant calculation is identical in this case since the determinant of orthogonal matricies is 1. *QR* decomposition has an advantage in that it is possible to perform for any real square matrix. They then exploit the fact that any $n \times n$ orthogonal matrix, Q, can be constructed from at most n Householder reflections: $\mathbf{Q} = \mathbf{Q}_0, \mathbf{Q}_1, \cdots, \mathbf{Q}_{n-1}$ where

$$\mathbf{Q}_i = \mathbf{I} - 2\frac{\mathbf{v}_i \mathbf{v}_i^T}{\mathbf{v}_i^T \mathbf{v}_i} \tag{10}$$

They then allow $\mathbf{v}$ to be optimized. This also gives an additional advantage in that they can trade off computational complexity with flexibility by allowing for a variable number of Householder reflections. In practice this only led to slightly increased flexibility.

3.3. Linear Flows

Linear flows are an important class of transformations as they are actually able to perform cross-correlations between different elements, and thus are much more expressive than the aforementioned Element-wise Flows. These transformations are expressed in the form

$$\begin{aligned} T(x) &= A\mathbf{x} + \mathbf{b} \\ T^{-1}(x) &= A^{-1}(\mathbf{x} - \mathbf{b}) \end{aligned} \tag{11}$$

where $A \in \mathbb{R}^{n \times n}$ and $\mathbf{b} \in \mathbb{R}^n$. We may note that this is quite similar to the formulation of a densely connected layer but we have a restriction that A must be invertible. While this flow has more expressivity, it is still limited. [98] gives a useful example by imagining this transform on a Gaussian distribution. We see that $T(\mathcal{N}(\mathbf{x}, \mu, \Sigma)) = \mathcal{N}(\mathbf{x}, A\mu + \mathbf{b}, A\Sigma A^{-1})$. In a 1D setting this would correspond to shifting the distribution, via the mean μ, and either flattening or squeezing it, via the variance σ. In this case we're just transforming one Gaussian to another. Calculating the Jacobian determinant and inverse both can be quite cumbersome as calculating them requires $\mathcal{O}(n^3)$ complexity. Though this can be reduced by applying certain restrictions to A; such as requiring it to be diagonal, as is the case in an Element-wise Flow. Similarly, the Permutation Flows above can be described as Linear Flows, where W is defined as a matrix with a single 1 per row and column and 0 elsewhere, which again trivializes both calculations. These variations may also have other useful properties and we note that the Linear Flow is the backbone of many other types of flows.

[8]Many implementations ignore this and simply update the matrix W. The authors of GLOW used both implementations.

3.3.1 Normalization

The commonly used Batch Normalization [80] can be viewed directly as a linear transform. This transformation performs a scale and shift operation based on the batch statistics, making them element-wise flows.

$$\begin{aligned} T(x) &= \gamma\hat{\mathbf{x}} + \beta \\ &= \gamma\frac{x - \mathbb{E}[\mathbf{x}]}{\sqrt{\text{Var}(\mathbf{x}) + \varepsilon}} + \beta \end{aligned} \tag{12}$$

In this case the Jacobian determinant is simply the inverse square root of the product of the variances (plus epsilon): $\det J = \left(\prod \sigma_i^2 + \varepsilon\right)^{-\frac{1}{2}}$ Similarly, this can be used in any type of normalization format, such as weight normalization [8], layer normalization [7], or any other. RealNVP used this batch normalization and weight normalization as a means to stabilize training, and used moving averages across mini-batches. Alternatively, Glow used a data dependent weight normalization [157] for the scale and bias terms ($\mathbf{s}$ and $\mathbf{t}$, respectively), calling this "ActNorm". These terms were initialized such that the the scale and bias terms had zero mean and unit-variance across each channel using the initial mini-batch. After this initialization the scale and bias terms were treated as normal learnable parameters.

3.3.2 Factorization and Decomposition

A simple way to reduce the computational overhead is to simply decompose A into something simpler. This reduction in computational costs does come with reduced expressive though, as we are not able to correlate all the features of the data. Fortunately we can use methods such as permutation to resolve some of these issues. [174] presented a volume preserving version of this by using Householder transform [71] as a substitute by using eigenvalue decomposition: $A = UDU^T$. Remembering that we can use a Householder transform to estimate the covariance matrix, Σ, for a variational autoencoder (VAE) [96].

$$T(x) = \left(I - 2\frac{\mathbf{v}(\mathbf{x})\mathbf{v}(\mathbf{x})^T}{||\mathbf{v}(\mathbf{x})||}\right)\mathbf{x} \tag{13}$$

A similar idea was also explored in [173] where a unitriangular matrix was learned instead of an orthogonal.

$$T(x) = \left(\sum y(\mathbf{x})L(\mathbf{x})\right)\mathbf{x} \tag{14}$$

Here y is softmaxed, creating a weight, and L is a lower unitriangular matrix. Both of these were conditioned on the encoder of the VAE, but the formulation holds for arbitrary networks.

3.4. Autoregressive and Coupling Flows

Coupling and autoregressive flows are the most familiar works and offer a balance between ease of understanding and expressivity. Both these formulations have triangular jacobians, making them often easy to invert as the determinant of a triangular matrix can be read from the diagonal. They work through the use of a conditioner, which need not be bijective, which can make them exceptionally expressive. In addition, they provide a good balance between computational complexity and representative power.

3.4.1 Coupling Flows

Coupling Flows, presented in NICE [33], introduced the concept where an input is split into two parts. The first part is scaled independently to the data and the second part was scaled, with a neural network, conditioned on the first part. This means that coupling flows are actually a special case of autoregressive models where the Jacobian is even simpler.

$$J = \begin{bmatrix} I & 0 \\ A & D \end{bmatrix} \tag{15}$$

Here $I \in \mathbb{R}^{n\times n}$ is an identity matrix and $D \in \mathbb{R}^{(N-n)\times(N-n)}$ is a diagonal matrix. Coupling flows are typically formulated in one of two ways: that their data is split into two equal part and operated upon independently, or a mask is applied for one operation and the inverse of the mask is applied to the other.

There have been several variants upon this methodology and we will briefly discuss a few. NICE introduced additive coupling layers where we see $T(\mathbf{x}) = [\mathbf{x}_0, \mathbf{x}_1 + f(\mathbf{x}_0)]$, where f is an arbitrary transformation. This has a unit Jacobian determinant and is thus volume preserving. RealNVP introduced an affine coupling layer, which itself is a linear transform, which introduced more exprestivity: $T(x) = \left[\mathbf{x}_0, e^{s(\mathbf{x}_0)}\mathbf{x}_1 + t(\mathbf{x}_0)\right]$. While NICE used channel-wise masks, RealNVP used both channel-wise and checkerboard masks, which have become increasingly common. Flow++ [65] extended this further, noting that the scaling term could be considered a cumulative distribution function for a mixture of logits:

$$\begin{aligned} T(x) &= \mathbf{x}_0, \\ \sigma^{-1}&(\text{MixLogCDF}\,(\mathbf{x}_1; \\ &\qquad \pi(\mathbf{x}_0), \mu(\mathbf{x}_0), \\ &\qquad s(\mathbf{x}_0))e^{a(\mathbf{x}_0)} + t(\mathbf{x}_0))] \end{aligned} \tag{16}$$

, where $\text{MixLogCDF}(\mathbf{x}_1; \pi, \mu, s) = \pi_i \sigma(\mathbf{x}_1 - \mu_i)e^{(-s_i)})$. Here π, μ, s represent the probabilities, means, and log scales, respectively, which are all learned parameters based on $\mathbf{x}_0$. Additionally, the scale and translation parameters, a and t, are also learned on $\mathbf{x}_0$. σ^{-1} is also the inverse sigmoid function, which is used to ensure that the inverse always exists. The inverse of the CDF is not simple to directly calculate, but since it is monotonically increasing the inverse always exists, and can be found by using root finding methods such as the bisection method.

3.4.2 Autoregressive Flows

Autoregressive models are a special type of flows where the Jacobian is a triangular matrix. An autoregressive model is typically defined as

$$T(x) = \sum A_i \mathbf{x}_{t-i} + \varepsilon_t \tag{17}$$

We can note where that we can treat A and $\mathbf{x}_i$ as equally sized but with zeroed terms where $i < n$. Masked Autoencoder for Density Estimation (MADE) [48] makes this clear by masking an autoencoder, to allow for density estimation. [9] The mask is what provides the *autoregressive* property, that each output $\mathbf{x}_n$ depends only on the previous points $\mathbf{x}_{<n}$ and not on any other parts. The key here is that an autoregressive model is being represented as a sequence of time, and viewed this way we say that our model's predictions of a point t only depends on the history and not the future. This same type of modeling is commonly used in Natural Language Processing (NLP) for sequence prediction, where words in a sequence are masked such that the predicted word depends solely on the previously typed words. Masked Transformers [181], such as BERT [31] or GPT [21, 145, 146], used these masked types of sequences and have shown incredible performance in sequence modeling. It is important to note that the masks applied here are strictly lower triangular.

For problems where there is not a clearly defined sequence, the order is agnostic [178] and thus permutations of the masks could be used and actually an ensemble of models can be trained simultaneously. This agnostic ordering and ensembling is valuable because order matters

[9]Note that autoregressive models often find that a direct connection between the input and output layers and can often make significant differences in results [12]

when calculating density but this order may not be known a priori. A similar approach can be seen in masked convolutions [130, 180] and causal convolutions [129], which enforce similar structuring. Inverse Autoregressive Flow (IAF) [95] noticed that the Jacobian of the inverse of an autoregressive model is lower triangular – where the diagonals are the derivatives of the transformation of each element – and thus constitutes a flow. Masked Autoregressive Flow (MAF) [133] then extended the idea, with concerns about the sequence ordering, noting that these flows can be stacked one upon another with different, or random, sequence ordering. This work generalized IAF and RealNVP (MAF and IAF use MADE as backbones). This was an important step as autoregressive models are universal approximators.

3.5. Radial and Sylvester Flows

Radial and Planar Flows [152] were some of the first flows used for variational inference and have a deeper connection to many other types of flows as well. These can sometimes be viewed as Linear Flows but we will have our own section for added clarity. The inverses of these flows are not easily computed and so they were not used for some time, with the original work focusing on variational inference rather than generation. More recent works revisited the ideas and generalized of these forms into substantially more powerful networks.

3.5.1 Planar Flows

A Planar Flow and named such because their operation expands and contracts the manifold with respect to a hyperplane.

$$T(\mathbf{x}) = \mathbf{x} + uf\left(w^T\mathbf{x} + b\right) \tag{18}$$

where $u, w \in \mathbb{R}^n$ and $b \in \mathbb{R}$ are free parameters. The function $f : \mathbb{R} \mapsto \mathbb{R}$ is a smooth element-wise nonlinearity. The Jacobian determinant can be computed easily in $\mathcal{O}(n)$

$$\det J = \mathbb{1} + u^T f'(w^t\mathbf{x} + b)w \tag{19}$$

Where $f'(\cdot)$ is the derivative of our nonlinearity. Thus or flow direction expands and contracts the manifold with respect to the hyperplane $w^t\mathbf{x} + b = 0$. Not all forms of $f(\cdot)$ are invertible and this can limit the ability to perform a generative process. Additionally, IAF [95] pointed out that a planar flow can be seen as a multi-layer perceptron (MLP) with a single node bottleneck, meaning that all information must pass through this singular node. Unfortunately, this means that long chains are needed to express high dimensional information. But the original authors did compare their work to others (NICE) and found that it scaled better and required fewer parameters.

3.5.2 Sylvester Flows

Sylvester flows have received a lot of attention recently, focusing on generalization and optimization. In this section we will discuss both of these improvements but a good understanding of Linear Algebra is needed and an understanding of optimization is highly beneficial.

Seeing this limitation [60] introduced Orthogonal Normalizing Flows. If we slightly change Eq. (18) to

$$T(\mathbf{x}) = \mathbf{x} + Uf(W\mathbf{x} + b) \tag{20}$$

Where $U \in \mathbb{R}^{n\times m}$, $W \in \mathbb{R}^{m\times n}$, and $b \in \mathbb{R}^n$ (each dimension extended) and restrict $m \leq n$. This equation actually represents a single-layer MLP with m hidden units. We can use Sylvester's Determinant Identity ($\det(\mathbb{I}_n + AB) = \det(\mathbb{I}_m + BA)$) to find a more expressive Jacobian Determinant with

$$\det J = \det\left(\mathbb{I}_m + \operatorname{diag}\left(f'(W\mathbf{x} + b)\right)WU\right) \tag{21}$$

This still isn't always invertible so they let $U = QD$ and $W = Q\tilde{D}$, where Q is orthonomral and D is diagonal. Using the algorithm form [15] we find $Q_{k+1} = Q_k(\mathbb{I} + \frac{1}{2}(\mathbb{I} - Q_k^T Q_k))$, which allows for greater expression.

[14] improved this work, calling the above flow O-SNF (Orthogonal Sylvester Normalizing Flow) and presented two others: H-SNF (Householder) and T-SNF (Triangular). In both of these cases they used QR decomposition instead, similarly with Q being identical. H-SNF uses the same Householder transformation for Q, and T-SNF reversed Q (identical to letting R and $\tilde{R}$ swap between upper and lower triangular).

[1] pointed out shortcomings of QR decomposition, specifically in the context of SNFs, and used exponential maps and Cayley maps (from Lie group theory) to address some of these issues. They noticed that other factorizations introduced spurious local minima, numerical instabilities, and that they resulted in poor gradient estimates.What's noted is that $Q \in \mathbb{R}^{n\times n}$ is in the orthogonal group, $O(n)$, and that if the absolute value of the determinant was 1 then it belonged to the special orthogonal group, $SO(n)$. When these groups are represented as manifolds they have dimension of $n(n-1)/2$, meaning that less than half, $< \frac{n}{2}$, parameters are needed to properly characterize Q, which would result in significant size reductions. C-SNF (Cayley) used the factorization $Q = (I + D)(I - D)^{-1}$ and E-SNF (exponential) used $Q = e^D = \frac{D_i}{i!}$. [10]

[69] generalize the Sylvester NF by developing new transforms, called the convolution exponential and graph

[10]The Cayley factorization fails to be invertible when $||D||$ tends to infinity, but is quick to compute. The exponential factorization is actually a surjective function but is a diffeomorphism between a bounded and connected subset of *Skew(n)* and all but a negligible part of $SO(n)$ (When D has measure 0 on *Skew(n)*). Authors note that the exponential factorization tends to be better than the Cayley.

convolution exponential. The transform and associated Jacobian determinant are

$$\begin{aligned} T(\mathbf{x}) &= \mathbf{x} + W^{-1} f_{AR}(W\mathbf{x}) \\ \det J &= \det\left(\mathbb{I} + J_{f_{AR}}(W\mathbf{x})\right) \end{aligned} \tag{22}$$

Where W is any invertible matrix, f_{AR} is a smooth autoregressive function ($f_{AR} : \mathbb{R}^n \mapsto \mathbb{R}^n$), and $J_{f_{AR}}(W\mathbf{x})$ represents the Jacobian determinant of $f_{AR}(W\mathbf{x})$. This represents a generalized case of Eq. (20). To correctly parameterize W they introduce an exponential convolution. Noting that for a cross-correlation (referred to as a convolution) with kernel m can be represented as a matrix multiplied with a vector, $m \star x = M\mathbf{x}$, they defined the exponential convolution $m \star_e x = e^M \mathbf{x}$ (as long as M can be expressed as an exponential). This gives the final version of their Convolutional Sylvester Flows (CSF) as

$$\begin{aligned} T(\mathbf{x}) &= \mathbf{x} + Q^T\left((-m) \star_e f_{AR}(m \star_e Q\mathbf{x})\right) \\ (det)J &= \det(\mathbb{I} + J_{f_{AR}}(m \star_e Q\mathbf{x})) \end{aligned} \tag{23}$$

noting that $J_{f_{AR}}$ is lower triangular and that Q is a 1×1 convolution parameterized by Househoulder reflections. For the matrix exponential they use a power-series expansion for approximation, noting that a matrix exponential can be used to construct a linear transform that is the solution to a linear ODE, $\mathbf{x}(t) = e^{Mt}\mathbf{x}_0$.

3.5.3 Radial Flows

[152] also proposed a family of transformations that would radially expand and contract the manifold around a reference point, $\mathbf{x}_0$ in the base density, called Radial Flows.

$$\begin{aligned} T(\mathbf{x}) &= \mathbf{x} + \beta f(\alpha, r)(\mathbf{x} - \mathbf{x}_0) \\ &= \mathbf{x} + \frac{\beta}{a + ||\mathbf{x} - \mathbf{x}_0||}(\mathbf{x} - \mathbf{x}_0) \end{aligned} \tag{24}$$

$$\begin{aligned} \det J = &\left(\mathbb{1} + \frac{\beta}{\alpha + ||\mathbf{x} - \mathbf{x}_0||}\right)\mathbb{I} \\ &- \frac{\beta(\mathbf{x} - \mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)^T}{(||\mathbf{x} - \mathbf{x}_0||)(\alpha + ||\mathbf{x} - \mathbf{x}_0||)^2)} \end{aligned} \tag{25}$$

where $\alpha \in \mathbb{R}^+$ and $\beta \in \mathbb{R}$. Again, not all forms of $f(\cdot)$ are invertible, and thus we're limited in our architectural choices.

3.6. Monotonic Flows

In this section we introduce the class of monotonic flows, who's focus is on using a class of monotonic functions. We'll begin with flows that are actually coupling flows, but the connection will be more apparent when we get to residual flows. A reminder of the following theorems will be helpful to the reader.

Theorem 5. Strong Monotone Operator: *A function $f : X \mapsto Y$ is k-strongly monotone if for all x, y and $k > 0$*

$$\begin{aligned} \langle f(x) - f(y), x - y\rangle &\geq k||x - y||_2^2 \\ x < y &\implies f(x) < f(y) \end{aligned} \tag{26}$$

The two lines are equivalent and say that the operator is *strictly* increasing. The actual theorem is even a bit stronger as it includes Hilbert spaces, but for us this works. Note that a k-strongly montonic function is always invertible. Note that f not need be absolutely continuous.

Theorem 6. Lipschitz Continuity: *A function $f : X \mapsto Y$ that maps two metric spaces is k-Lipschitz continuous, for a non-negative k, if*

$$\begin{aligned} \langle f(x) - f(y)\rangle &\leq k\langle x - y\rangle \\ d(f(x), f(y)) &\leq kd(x, y) \end{aligned} \tag{27}$$

The two lines are the same, but showing this works for any distance metric. If $k < 1$ we call this contrastive, $k = 1$ nonexpansive, and $k > 1$ expansive. This can be derived from the fundamental theorem of calculus and we can see that if a function is differentiable $\implies$ Lipschitz continuous $\implies$ continuous (but not the reverse).

Corollary 2. *When f is a contrastive Lipschitz function, its set of fixed points $x|f(x) = x$ are convex*

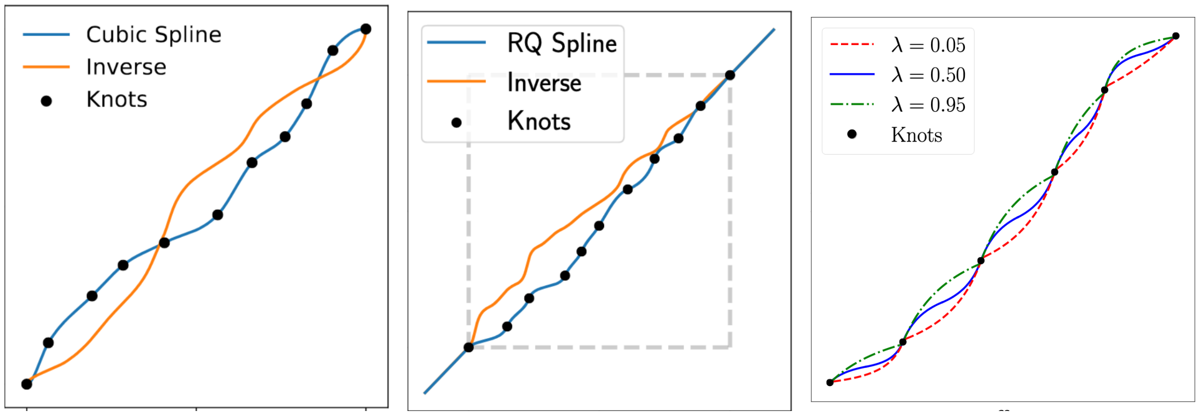
Theorem 7. Banach Fixed Point: *If a function f is a contraction then the iteration $x_{i+1} = f(x_i)$ converges to a fixed point on f*

3.6.1 Spline Flows

A simple way to ensure that our transformations is to impose that our transform is monotonic. A good choice for this is to use a monotonicly spline.[11] Splines are piecewise functions with k segments (parameterized by $k+1$ points), are simple to evaluate and invert (both can be done with $\mathcal{O}(\log k)$ via binary search), and can be arbitrarily complex (determined by k). Because of these properties, splines are an attractive choice for many types of problems but they may have a higher barrier to entry when understanding with respect to the math. We will do our best to explain these concepts and the motivations here. Most of our notation will be per-bin and we will drop the respective subscript, as will be explained in the first example.

[124] propose the usage of piecewise linear and piecewise quadratic coupling transforms. For the linear version the coupling function partitions the input into k segments with equal width of k^{-1} and this gives a transform for each

[11] Monotonic functions are guaranteed to have an inverse



(a) Cubic Spline (b) Rational Quadratic Spline (c) Linear Rational Spline

Figure 3. Comparison of different splines and their respective flexibilities. Images from [35, 39, 40]

bin.

$$f(\mathbf{x};\theta) = \alpha\theta_0 + \sum\theta_k$$
$$\det J = \prod\frac{\theta_i}{w} \tag{28}$$

Here θ was a softmaxed matrix, $\sigma(Q)$, and $\alpha = k\mathbf{x} - \lfloor k\mathbf{x}\rfloor$. [12] denotes the same equation as $f_i(\mathbf{x}_i^B;Q) = \alpha Q_{i,b} + \sum_{k=1}^{b-1} Q_{i,k}$. We drop the i which represents the i^{th} dimension and B representing the split from the coupling function, which they note as $\mathbf{x} = [\mathbf{x}^A, \mathbf{x}^B$. If $[\mathbf{x}^A, \mathbf{x}^B] = [\mathbf{x}_{1:n-1}, \mathbf{x}_{n:N}]$ then i represents each element in the set $n : N$. We take $\mathbf{x}^B$ to be a given since these are coupling flows. Additionally we convert Q to θ to make the flexibility more apparent, as this can often be an arbitrary network or matrix. Therefore we see that the coupling function is per dimension, operating on the proper row of $\mathbf{x}$ and θ, using the initial points and summing over the other bins (k) of the same dimension. We believe that our notation is less cumbersome and allows for the reader to focus on the transform and thus will be using similar notation throughout this section.

Unfortunately, with fixed bin widths the network is not as flexible. If the network could predict then bin widths then this could generate more complex functions, where smooth regions have large widths and complex regions have smaller widths. The quadratic version addresses this issue but modeling the $(k+1)$ vertices, storing vertical coordinates in a matrix (V) and the bin widths in another (W). This gives us the per-bin coupling function:

$$f(\mathbf{x};V,W) = \frac{\alpha^2}{2}(V_{b+1} - V_b)W + \alpha VW + \sum\frac{V_k + V_{k+1}}{2}W_k \tag{29}$$

[12]Note that [124]

Here b represents the associated bin and $\alpha = (\mathbf{x} - \sum W_k)/W$. Interestingly they use this to generalize one-hot encoding into one-blob encoding, which discritizes it into bins using a gaussian kernel ($\sigma = \frac{1}{k}$). One-blob encoding is lossless, as opposed to the lossy nature of one-hot encoding.

Both of the splines in [124], called "polynomial splines", restrict the coupling function's domain to $[0,1]$. [39] generalizes the prior work by using monotonic piecewise cubic splines [45] where each segment is a monotonically increasing polynomial. They then use Steffen's Method [168] to parameterize the knots, enforcing the boundary knots to be at $(0,0)$ and $(1,1)$, respectively. They use an arbitrary function (neural net) that maps $f(\cdot) : \mathbb{R}^{n-2} \mapsto \mathbb{R}^{2k+2}$ which then is chunked into segments with dimensions $\mathbb{R}^k, \mathbb{R}^k, \mathbb{R}^2$, representing the height and width coordinates of the knots and the boundary derivatives. The height and width segments are constrained with a softmax (needs to be positive and sum to 1). Finally, a cumulative sum is applied to the partitions, ensuring that the function is monotonically increasing. The inverse can be solved with a cubic solver [16, 17, 139] and the Jacobian determinant can be computed quickly as it is lower triangular (per i). With the knot coordinates (x_k, y_k), the derivative of the cubic function is

$$\begin{aligned} J_k(\xi) &= \alpha_{k1} + 2\alpha_{k2}\xi + 3\alpha_{k3}\xi^2 \\ &= d_k + 2\left(\frac{3s_k - 2d_k - d_{k+1}}{w_k}\right)\xi \\ &\quad + 3\left(\frac{d_k + d_{k+1} - 2s_k}{w_k^2}\right)\xi^2 \end{aligned} \tag{30}$$

$$\xi = \frac{x - x_k}{w_k}, \quad s_k = \frac{y_{k+1} - yk}{w_k}, \quad w_k = x_{k+1} - x_k \tag{31}$$

Here d represents the derivative of the k^{th} knot, which is

$$d_k = \begin{cases} 2\min s_{k-1}, s_k & p_k > 2\min s_{k-1}, s_k \\ p_k & \text{else} \end{cases} \tag{32}$$
$$p_k = \frac{s_{k-1}w_k + s_k w_{k-w}}{w_{k-1} + w_k}$$

[40], the same authors, extended the work to rational quadratic (RQ) splines. The general idea stayed the same but some minor changes were made to increase the flexibility which allowed coupling based models to match the power of autoregressive ones. First, they allowed the interval to be non-unitary and instead allowed for the knot coordinates to lie within an arbitrary compact space $[-B, B]$, but can be projected to a unit interval via the method used by [57]. This significantly increases the flexibility, as we can interpret this as placing knots with higher precision. In this version θ is projected into $\mathbb{R}^{3K-1}$ and is then partitioned into $\mathbb{R}^K, \mathbb{R}^K, \mathbb{R}^{K-1}$, similar to before. The height and width projections are then softmaxed, as before, and multiplied by 2B, ensuring they span the compact manifold, and the derivative projection is passed through a softplus activation. We can then write the rational quadratic polynomial for the k^{th} bin and its derivative as

$$\begin{aligned} \frac{\alpha_k(\xi)}{\beta_k(\xi)} &= y_k + \frac{(y_{k+1} - y_k)(s_k\xi^2 + \delta_k\xi(1-\xi))}{s_k + (\delta_{k+1} + \delta_k - 2s_k)\xi(\xi - 1)} \\ \frac{d}{dx}\left(\frac{\alpha_k(\xi)}{\beta_k(\xi)}\right) &= \frac{s_k^2\left(\delta_{k+1}\xi^2 + 2s_k\xi(1-\xi) + \delta_k(1-\xi)^2\right)}{s_k + (\delta_{k+1} + \delta_k - 2s_k)\xi(1-\xi)^2} \end{aligned} \tag{33}$$

While this result did not produce state of the art generation on Cifar-10 [99] or ImageNet [30], the work performed similarly to Glow but with nearly a quarter of the parameters.5.

The above methods have quite a difficult problem though, since they require solving n-degree polynomial fits. [35] resolves this by creating a "linear rational spline." Using a technique from [46], they show that each spline bin can be split into two, allowing for greater flexibility as these, as there are fewer constraints on the middle bin. Given the homographic function [13]

$$f(\mathbf{x}) = \frac{w_k y_k(1-\phi) + w_{k+1}y_{k+1}\phi}{w_k(1-\phi) + w_{k+1}\phi} \tag{34}$$

where $\phi = (x - x_k)/(x_{k+1} - x_k)$ we can define our middle bin through linear interpolation: $x_m = (1-\lambda)x_k + \lambda x_{k+1}$. This gives the spline more flexibility, as it can be located at any point between the two knots ($0 \leq \lambda \leq 1$). The advantage of this is that one needs to now only worry about the fit (between the bin) needs only be continuous and differentiable. This gives the following update to Eq. (34)

$$f(\phi) = \begin{cases} \frac{w_k y_k(\lambda_k - \phi) + w_m y_m \phi}{w_k(\lambda_k - \phi) + w_m\phi} & 0 \leq \phi \leq \lambda_k \\ \frac{w_m y_m(1-\phi) + w_{k+1}y_{k+1}(\phi - \lambda_k}{w_m(1-\phi) + w_{k+1}(\phi - \lambda_k)} & \lambda_k \leq \phi \leq 1 \end{cases} \tag{35}$$

Where λ, the ws, and y are learnable parameters. Also useful is that the inverse of this function does not require solving polynomial functions.

Splines are still an evolving research area and highly studied in graphics and mathematics. These works show that this research can also be leveraged to advance the state of normalizing flows. Splines offer flexible representations that require few parameters but do come at the cost of, usually, requiring a complex calculation when inverting is necessary.

3.6.2 Residual Flows

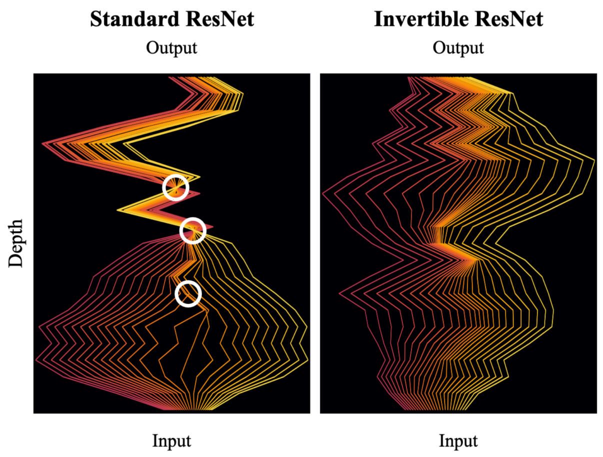


Figure 4. Invertible ResNets (i-ResNet) compared to standard ResNet. Standard residual connections create knots which prevent the process from being bijective. Image from [10]

Residual Flows are a unique type of architecture compared to what we've discussed previously. They can actually use any architecture type but are restricted in other ways. While other types of flows have restricted Jacobian determinants, Residual Flows are free-formed. This allows them to be more expressive than other types of flows, but they come with their own complications, namely computation.

ResNets [63] was one of the most influential papers in machine learning, allowing the extremely deep networks that we see today be possible. The advantage of these networks is that they have free-form Jacobians and thus can be extremely expressive. Unfortunately, this simple but effective process isn't invertible in an obvious manner. [10] sought to solve this, noticing that residual networks share

[13]The quotient of two first degree polynomial (affine) functions.

a simiar form to Euler's method to solve initial value problems of ODEs (forward and backward). If $F_\theta : \mathbb{R}^n \mapsto \mathbb{R}^n$ is a composable function that represents a ResNet, then the t^{th} composition can be written as $F_\theta^t = I + g_{\theta_t}$. The ResNet F_θ becomes invertible if the Lipschitz constant of $g_{\theta_t} < 1 \forall t$. [14] The Lipschitz constraint can easily be applied by using a spectral norm [121, 176, 190] for each of the convolutional layers and like [53] directly calculate through a power series. This allows the jacobian determinant to be calculated $\log|\det J_F(\mathbf{x})| = \mathrm{tr}(\log J_F)$, letting $F(\mathbf{x}) = (I+g)(\mathbf{x})$. Thus, we get $J_F = I + J_g(\mathbf{x})$, which can be computed using a power series using the Skilling-Hutchinson estimator [75, 165]. The reverse direction can also be solved through fixed-point iteration, noting that the Lipschitz bound for the reverse direction is also < 1. Their transform can be seen below in Eq. (36)

$$\log(p(\mathbf{x})) = \log(q(\mathbf{x};\theta)) + \mathrm{tr}\left(\sum \frac{-1^{i+1}}{k} J_g(\mathbf{x})^k\right) \quad (36)$$

Unfortunately [10]'s method requires approximation of the density, due to the infinite series calculation, and thus a bias. Worse, this bias grows with the data dimensionality and the Lipschitz constant of g. [25] argued that this means i-ResNet can't calculate MLE, instead optimizing the bias, and thus sought to resolve this, creating an unbiased estimator. Using the Russian Roulette Unbiased Estimator [84, 114], which is an unbiased estimator for the limit of a sequence, the trace of matrices J_g^k (Equation (36)) can be reformulated as

$$\begin{aligned}\log(p(\mathbf{x})) &= \log(q(f(\mathbf{x}))) \\ &+ \mathbb{E}_{n,v}\left[\sum \frac{-1^{k+1}}{k} \frac{v^T (J_g(\mathbf{x})^k v}{\mathbb{P}(N \geq k)}\right]\end{aligned} \quad (37)$$

Where $n \sim p(N)$, $v \sim (N)(0,I)$, and $\mathbb{P}$ represents polynomials. Unfortunately this is computationally inefficient for backpropagation as each gradient in the series has to be tracked, making a $\mathscr{O}(nm)$, representing the computed terms and residual blocks, respectively. To resolve this two methods are used, Neumann gradient series and backward-in-forward early computation, allowing for $\mathscr{O}(1)$ memory usage. Even with this change we note that the computation is still quite expensive, with several days of training even on a multi-GPU setup just for MNIST. i-ResNet also found that smooth and non-monotonic activation functions worked well and decided to use Swish [148], which is a gated sigmoid-weighted linear unit [41]. To ensure that Swish has a Lipschitz constant of less than 1 they divide by a bound, which is found by the max of the absolute value of the derivative.

$$\mathrm{LipSwish}(x) = x\frac{\sigma(\beta x)}{1.1} \quad (38)$$

Here β is learnable and strictly positive (enforced by SoftPlus).

[138] introduced invertible DenseNets (i-DenseNet). Dense networks [73, 74] are have connections to every previous layer, allowing for more efficient training and better performance than a traditional ResNet. While a traditional DenseNet outputs only $g(x)$, having concatenated residual inputs, i-DenseNet outputs the input as well, $x + g(x)$.

$$g(x) = W_n \circ h_{n-1} \circ \cdots \circ h_0(x) \quad (39)$$

Where W_n is a 1×1 convolution that reshapes the output back the the appropriate dimension. Each layer, h_i, contains both the input signal

$$\begin{aligned} h_0(x) &= \begin{bmatrix} x \\ \phi(T_0(x)) \end{bmatrix}, \\ h_i(\ldots h_0(x)) &= \begin{bmatrix} h_{i-1}(x) \\ \phi(T_i(h_{i-1}(x))) \end{bmatrix} \end{aligned} \quad (40)$$

Where T is a transform, typically a convolutional layer, and ϕ is an activation function. As the previous Residual Flows, we require that $\mathrm{Lip}(\phi) \leq 1$. Similar to [161] they use a concatenated activation function, in this case LipSwish 38.

If activation functions are though of as filters, concatenated activation functions can be though of as filtering from both sides by applying it twice. Because it is applied twice our channels double, and thus a smaller growth factor is applied.

$$\begin{aligned} \mathrm{CLipSwish}(x) &= \phi(x)/\mathrm{Lip}(\phi) \\ \phi(x) &= \begin{bmatrix} \mathrm{LipSwish}(+x) \\ \mathrm{LipSwish}(-x) \end{bmatrix} \\ \mathrm{Lip}(\phi) &= \sup_x ||J_\phi(x)||_2 \end{aligned} \quad (41)$$

$$\begin{aligned} \sup_x ||J_\phi(x)||_2 &= \sup_x \sigma_{max}(J_\phi(x)) \\ &= \sup_x \sqrt{\left(\frac{\partial \psi_1}{\partial x}\right)^2 + \left(\frac{\partial \psi_2}{\partial x}\right)^2} \\ &\approx 1.004 \quad \forall \beta \end{aligned} \quad (42)$$

Where J is the jacobian determinant of the concatenated LipSwish. The tighter lower bound (1.004) allows for better gradient norm attenuation [104]. [15] gives a clear explanation of this under ℓ_2 and [192] for the ℓ_p norm case.

[14]Lipschitz constant tells us the maximum rate of change a function can have, thus relating to its derivative and relating to continuity. The Lipschitz Constant, L is defined by $|f(\mathbf{x}_0 - f(\mathbf{x}_1)| \leq L|\mathbf{x}_0 - \mathbf{x}_1|$. The authors use contraction mapping via the Banarch Fixed-Point Theorem, which allows us to perform the invertiblity calculation needed later.

[15]A tighter Lipschitz bound is often desirable as it provides better robustness. [102]

Implicit Normalizing Flows [109] work to generalize residual flows by using implicit neural networks (INNs) [5, 9]. INNs work differently compared to general methodologies. INNs are specified by the conditions that are desired from the output rather than, for traditional networks, specifying how to compute the output. Often this is done through fixed point iteration, which is where the connection forms. In this work we can simply define a function, with mapping $F : \mathbb{R}^{2d} \mapsto \mathbb{R}^d$, with two variables, $x, z \in \mathbb{R}^d$, that defines a unique invertible map, $F(z, x) = 0$. We will also define z to be dependent on x, $z = f(x)$, where $f : \mathbb{R}^d \mapsto \mathbb{R}^d$. This can be then decomposed such that

$$F(z,x) = g_x(x) - g_z(z) + x - z \tag{43}$$

Where $g_i : \mathbb{R}^d \mapsto \mathbb{R}^d$ and $Lip(g_i) < 1$, for both z, x. This formulation is essentially identical to ResFlow [25] but we have both directions.

$$\begin{aligned}(g_x + id)(x) = g_x(x) + x = g_z(z) + z = (g_z + id)(z)\\ z = ((g_z + id)^{-1}(g_x + id))(x)\\ = (\text{Inverse ResFlow})(\text{ResFlow})\\ = (\text{implicit})(\text{explicit})\end{aligned} \tag{44}$$

z can be solved simply by solving for the fixed point, which can be done by quasi-Newton Methods such as Broyden's. The logdeterminants are similar to those as ResFlow's as well

$$\begin{aligned}\log(p(x)) = \log(p(z))\\ + \log(\det(I + J_{g_x}(x)))\\ - \log(\det(I + J_{g_z}(z)))\end{aligned} \tag{45}$$

This formulation increases the expressivity of ResFlows because the forward ResFlow handles small Lipschitz parts of the target function and the Inverse ResFlow handles the large Lipschitz parts. This work represents a significant improvement in generalizing ResFlows, but experiments were limited due to computational constraints. The work shows that implicit representations are a useful avenue for research in normalizing flows and much remains to be explored here.

More recent work by [4] proposed a monotone formulation as a means to escape the Lipschitz constraints. A function is monotone if every output increases for every increasing input: $(F(x) - F(y))^T(x - y) \geq 0$, $\forall x, y \in \mathbb{R}^n$. A strongly montone function, $(F(x) - F(y))^T(x - y) \geq m||x - y||_2^2$ when $m > 0$, also has a clear Liptschitz constant defined by m and is also invertible [64, 155]. [4] realized that Equation (44) is a split operator and that it looks quite like the Cayley operator, where $(id + g_z)^{-1}$ looks like a Resolvant, R, they decided to formalize this. This allowed them to parameterize the Cayley operator, $2R - id = 2(\lambda F + id)^{-1} - id$, which allows for a more flexible Lipschitz constraint.

$$\log\det J_f = \mathrm{tr}\left[\log(I - J_G) - \log(I + J_G)\right] \tag{46}$$

$$\begin{aligned}\log p_X(x) = \log p_Z(z)\\ + \mathbb{E}_{n,v}\left[\frac{(-1) - (-1)^{k+1}}{k} \frac{v^T J_G^k v}{P(N \geq k)}\right]\end{aligned} \tag{47}$$

We let $n \sim p_N(n)$ and $y \sim \mathcal{N}(0, I)$. [4] also noticed that the LipSwish activation still suffered from a vanishing gradient problem since the first derivative is still close to 0 when $x = 0$. Thus the authors introduced a new activation function, positive identity 1-Lipschitz activation (Pila) (Equation (48)). Pila is substantially smoother and has a velocity near 1 at $x = 0$, making it a better choice.

$$\mathrm{Pila}(x) = \begin{cases} x & x \geq 0 \\ \left(\frac{k^2}{2}x^3 - kx^2 + x\right)e^{kx} & x < 0 \end{cases} \tag{48}$$

$$\mathrm{CPila}(x) = \alpha_1\left[\mathrm{Pila}(x - \alpha_2),\ \mathrm{Pila}(-x - \alpha_2)\right]^T \tag{49}$$

Where $k > 0$ is either learned or fixed, usually 5, $\alpha_1 = 1/1.06$ and $\alpha_2 = 0.2$.

Residual Flows are a promising research area, but do require significantly more mathematical analysis than other forms. In addition to this, they are often quite difficult to compute given that all the above methods use fixed-point iterations. Despite that, they have free-form Jacobian Determinants and can thus be far more expressive when compared to other forms. If new algorithms for faster fixed point iterations, better ways to formulate the Lipschitz constraint, or that other methods can be found then these algorithms could be quite promising.

3.7. Continuous Flows and Neural ODEs

Another promising method for free-form Jacobian Determinants is through the use of Ordinary Differential Equations (ODEs). In this section we will discuss the major works in this area, but note that it is an active research area and things are changing quite quickly. Many of these methods may not be referred to as Normalizing Flows, despite explicit use of bijectors, but not all Neural ODEs (NODEs) are Normalizing Flows (or even isomorphic) either. In general these works can better be seen from the framework of homeomorphic functions, and thus are included within the isomorphic understanding of Normalizing Flows. For a complete description of NODEs we recommend [88]'s thesis, which also includes their relationship to implicit neural networks [9, 49]. These networks have a lot of similarities to the previous section but we feel that it is helpful to distinguish them, and this may help specifically in seeing how Normalizing Flows can be used in non-distributional settings.

The first major work was Neural Ordinary Differential Equations [24] where the concept of a Continuous Normalizing Flow is introduced. Previous works [110] had shown

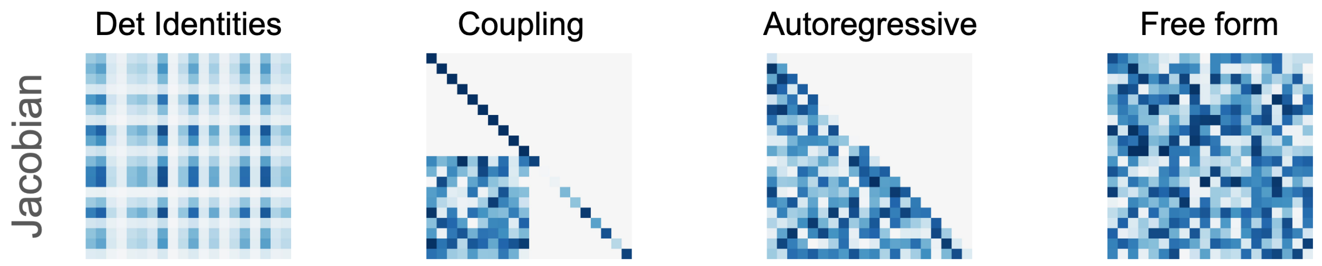


Figure 5. Expressive power of Normalizing Flows. Depiction of Jacobians for different Normalizing Flow methods. Sylvester flows have identity Jacobian determinants while Residual Flows and ODEs have free-form Jacobian determinants. Image from [25]

that ResNets, and others, could be interpreted as discritizations of differential equations, and thus sought to create continuous versions. The idea is simple and just recognizes that $h_{t+1} = h_t + f(h_t, \theta)$ is the Euler discretization of the continuous function $h'(t) = f(h(t), t, \theta)$, and one simply need to define the initial value problem (defining $h(0)$ and $h(T)$). Their formulation does not concern itself with the ODE solver, treating as a black box, and uses the adjoint method [143] to compute the gradients. This solves an augmented ODE in the reverse direction and we can see this as reverse-mode automatic differentiation through the computational graph. A scalar loss function L can then be formulated as

$$\begin{aligned} L(z(t_1)) &= L\left(z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta)dt\right) \\ &= L(\text{ODESolver}(z(t_0), f, t_0, t_1, \theta)) \end{aligned} \tag{50}$$

Where we can now solve the gradients by

$$\nabla_\theta L = -\int_{t_1}^{t_0} a(t)^T \nabla_\theta f(z(t), t, \theta)dt \tag{51}$$

$$\nabla_t a = -a(t)^T \nabla_z f(z(t), t, \theta) \tag{52}$$

Where $a(t) = \nabla_{z(t)} L$ is the *adjoint* state (sometimes called *sensitivity*). They then introduce the concept of continuous Normalizing Flows (CNF), recognizing how the adjoint state looks like our typical Jacobian. They give us the following theorm

Theorem 8. **Instantaneous Change of Variables:** *Let $\mathbf{z}(t)$ be a finite continuous time-dependent random variable with probability $p(\mathbf{z}(t))$. Let $\nabla_t \mathbf{z} = f(\mathbf{z}(t), t)$ be continuous in time function, where f is uniformly Lipschitz in $\mathbf{z}$ and continuous in t. The change in log probability follows the differential equation*

$$\nabla_t \log p(\mathbf{z}(t)) = -tr\left(\nabla_{\mathbf{z}(t)} f\right) \tag{53}$$

This may be called "free form" as f does not need to be bijective as uniqueness exists and the transformation is whole is bijective (similar to the formulations we had in coupling flows). We see that $\nabla_{\mathbf{z}} f$ is actually the Jacobian. An additional benefit is that this formulation reduces computation. Log density can be calculated as

$$\log p(\mathbf{z}(t_1)) = \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \text{Tr}\left(\nabla_{\mathbf{z}(t)} f\right) dt \tag{54}$$

FFJORD (Free-form Jacobian of Reversible Dynamics) [54] expanded this to reduce some of the constraints, introducing an unbiased stochastic estiator that has linear cost with respect to the dimension, $\mathcal{O}(D)$. They introduce Hutchinson's trace estimator to reduce computation, reformulating as

$$\begin{aligned} \log p(\mathbf{z}(t_1)) &= \log p(\mathbf{z}(t_0)) \\ &\quad - \mathbb{E}_{p(\varepsilon)}\left[\int_{t_0}^{t_1} \varepsilon^T \text{Tr}(\nabla_{\mathbf{z}} f)\, \varepsilon\right] dt \end{aligned} \tag{55}$$

With a fixed ε. An important aspect of this formulation is that it dramatically reduces the number of parameters needed, they were able to get the same performance as Glow [94] using less than 2% as many parameters. They did this with a single flow step and with a fixed Gaussian target distribution.

RNODE [42] recognized this as an optimal transport problem [182](pg 114) and that the dual of maximizing the likelihood of $p(\mathbf{x})$ is the minimization of $J(p(\mathbf{x}) = -\frac{1}{N}\log p(\mathbf{x}_i)$ (equivalent to the KL divergence). They additionally recognized that the transport function was poorly conditioned and that this could be regularized for significantly faster and increase stability, reducing the number of steps an ODE solver needs. They first use the Benamou-Brenier formulation [11], adding the kinetic energy to encourage particles to prefer straight lines with constant speed.

$$\begin{aligned} J_\lambda(f) &= \frac{\lambda}{N}\int_0^T ||f(\mathbf{z}_i, t)||^2 dt \\ &\quad - \frac{1}{N}\log p(\mathbf{x}_i) \end{aligned} \tag{56}$$

Then they added the Frobenius norm $\left(||A||_F = \sqrt{a_{i,j}^2} = \sqrt{\text{tr}(AA^T)}\right)$ to the trace estimator, which requires no extra computational cost. This gives the improved formulation

$$\begin{aligned}\min_f \frac{1}{Nd} & - \log p(\mathbf{z}(\mathbf{x}_i, T)) \\ & - \int_0^T \nabla f(\mathbf{z}(\mathbf{x}_i, s), s) ds \\ & + \alpha \int_0^T ||f(\mathbf{z}(\mathbf{x}_i, s), s)||^2 ds \\ & + \beta \int_0^T ||\nabla_z f(\mathbf{z}(\mathbf{x}_i, s), s)||_F^2 ds\end{aligned} \tag{57}$$

Where α, β are regularization weights for the kinetic energy and Jacobian Frobenius norm, respectively.

ANODE [38][16] recognized that there are many functions that NODEs cannot represent, as they can only perform smooth transformations of a manifold. That is, their deformations may not cut or tear, which prevents trajectories from crossing. They sought to instead create a transform which is *homeomorphic*, removing the constraint of differentiability, and to view data as a discrete collection of points rather than continuous. This formulation then assumes the function map $f : \mathbb{R}^n \mapsto \mathbb{R}^{n+p}$, and that there exists a point $\mathbf{a}(t) \in \mathbb{R}^p$ which represents a point in the augmented space. Their new formulation is

$$\begin{bmatrix} h(t) \\ a(t) \end{bmatrix} = f\left(\begin{bmatrix} f(t) \\ a(t) \end{bmatrix}, t\right) \tag{58}$$

$$\begin{bmatrix} h(0) \\ a(0) \end{bmatrix} = f\left(\begin{bmatrix} \mathbf{x} \\ \mathbf{0} \end{bmatrix}, t\right) \tag{59}$$

Where Eq. (59) are the initial values. They note that ANODE significantly increases stability as the timestep for a NODE may be smaller than that of machine precision to properly converge, even when such convergence is possible. While ANODEs increase the speed of NODEs, they are still slower than ResNets. Additionally, augmentations in the data changes the input dimension, which can add complexity.

Zhang *et al.* [193] further this formulation and proved that any homeomorphism on a n-dimensional Euclidean space can be approximated by a $2n$-dimensional NODE. In fact, they show that any $n+1$ dimensional NODE followed by a linear layer is a universal approximator for $f : \mathbb{R}^n \mapsto \mathbb{R}$. An important extension is that they show that any n-dimensional i-ResNet, which is zero padded into $2n$-dimensions, can approximate any n-homeomorphisms as long as the Lipschitz constant is finite and an upper bound is know. The upper bound of the Lipschitz constant is needed to determine the number of layers that are needed.

[16]It is worth noting that [38] does not call their work a Normalizing Flow but view formulation similarly to [24] where in this can be used to solve the Jacobian

While this work may seem complete, there are still important limitations within these works. They still remain comparatively computationally expensive and new methods need to be created to reduce these trade-offs. Additionally, Zhang *et al.* [193] still contains limitations. Continued research is still need to make these homeomorphisms universal approximators without an additional neural network as well as the work needs to be extended to non-Euclidean spaces and non-continuous functions.

4. Applications of Flows

Normalizing Flows have a wide variety of applications, as some have been discussed previously. Primary our discussion has focused on that of density estimation and generation, but we have many more applications. We start with a discussion of causality in Section 4.1, as this is one of the most fundamental areas of science. In general, science is the understanding of how our world works, and we wish to understand that through causal relationships. This is why we placed importance on the framing with respect to understanding relationships between data and their inherit structures. Then we will discuss applications in probability and statistics (Section 4.2), applications in audio (Section 4.3), and finally applications in vision (Section 4.4)

4.1. Causality

Flows have a few nice properties that make they quite useful for analysis. After all, our reasoning for creating these formulations was to either explicitly preserve all relationships between data, or provide a way to recover them. From this understanding, causality becomes a natural extension of these works. This is because causality itself is the understanding of those relationships and specifically the direction of influence. Judea Pearl [83, 134, 135] is one of the "fathers" of causality in the area of artificial intelligence and we will understand these relationships through the hierarchical ladder that he presents Table 1.

We assume that i.i.d. data is being sampled from a joint distribution $p(\mathbf{x}, \mathbf{y}, \mathbf{z}, \cdots)$ and that p is known and tractable. **Association** is simple to see, and in fact many models have this power, since they are conditioned on prior terms (what those terms are matters though!). Simply this is $p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{x},\mathbf{y})}{p(\mathbf{y})}$, our familiar form. **Intervention** is deceptively more difficult as we have to invoke the do-calculus [136]. An important part is that we can't sample from $p(\mathbf{x}, \mathbf{y}, \mathbf{z}, \cdots)$ but instead need to sample from the joint $p_{\text{do}(\mathbf{X}=\mathbf{x})}(\mathbf{x}, \mathbf{y}, \mathbf{z}, \cdots)$, noting that these are *NOT* the same. This is best seen with an examples.

- Suppose we're measuring the voltage across a resistor in a circuit, where $\mathbf{x}$ represents the voltage of the cir-

Association	$\Pr(x\|y)$	How does y change by belief about x?
Intervention	$\Pr(x\|\text{do}(y),z)$,	If we do y (independent of z) how does this change my belief about x
Counterfactual	$\Pr(x\|x',y')$	How would the outcome $\mathbf{x}$ have changed if $\mathbf{y}$ was $\mathbf{y}'$ instead?

Table 1. Judea Pearl's Ladder of Causal Relationships. These 3 forms of reasoning allow us to answer stronger causal questions and better understand the relationship between random events. They are in increasing order of causal understanding.

cuit and $\mathbf{y}$ represents the reading on an analogue voltmeter. If we intervene by holding the needle at the 0 voltage mark (setting $\mathbf{y} = 0$) we find $p(x|\text{do}(y = 0))$ is just the marganalized distribution $p(\mathbf{x})$.

In the example we note that the voltmeter measurement, $\mathbf{y}$, allows us to predict the voltage across the resistor, $\mathbf{x}$, but that the voltage *measurement* doesn't cause the voltage across the circuit: or $p(\mathbf{x},\mathbf{y}) \neq p_{\text{do}(\mathbf{x}=0)}(\mathbf{x},\mathbf{y}) = p(\mathbf{y})$. In the real world we may frequently do do-calculus with experiments such as randomized controlled medical trials. But we may also want to observe natural phenomena or may not even be able to intervene! **Counterfactuals** are a bit more difficult to understand as they are about a single datapoint. We will be using Pearl's definition: a probabilistic answer to a "what if" question. We will illustrate this with an example:

- Suppose that Jennifer is a female Assistant Professor who did make tenure. What would the probability of her getting tenure given that she was a man instead of a woman?

Here we have an already existing data point but want to perform a probability using do-calculus in the past. Unfortunately, we cannot make such a change given that we do not have a time-machine. But if we were wishing to understand how gender biases affected the decision process for Jennifer's promotion, we would have needed to do this *and* we would have had to ensure that all the other variables remained unchanged – which is unlikely to be possible even if we had our time machine. Through these examples, we can see how we've created a hierarchy in importance of causal measures, with counterfactual tests being the strongest indicator of causation.

With this understanding, we can observe that many models associative, but that we need a precise model of a probability density function to be able to gain strong causal understandings. The careful reader may have noticed that these types of relationships can be built into Normalizing flows! Flows can naturally a form of Structural Equation Model (SEM) [159] as they form relationships between variables in a controlled manner. In fact, [87] performed exactly this using autoregressive flows. Without knowing the causal direction they train multiple flows and compare the likelihood ratios [78]. Two candidate models allows them to test two potential directions – $x_1 \mapsto x_2$ vs $x_2 \mapsto x_1$ – where the sign dictates the causal direction. With this model framework the authors were able to perform experiments where both interventions and counterfactuals were able to be performed.

In an alternative approach [151] used a Spline Coupling Flow to model the noise in an additive noise model (ANM) [72]. The reasoning for this being that other ANMs assumed that the noise is Gaussian, which isn't realistic for natural processes. [194] demonstrated that a wrong parameterization of the noise resulted in biases estimate, but only extended the approach by using a mixture of gaussian model. Using a NF to model the noise, directly from the data, enables a much more accurate model and thus a reduction in bias. This approach even reduced the computational time by more than 60x. [189] uses a similar approach to model noise, but instead use it to create a causal latent space for a VAE [96]. Models such as [50,150,184] also use flows as part of their implementations, especially when invertibility is needed, but rely on other network structures to perform causal tasks. The invertible property of flows is specifically needed for counterfactual intervention, since we need to revert conditions given an outcome. [79,125] even create a causal model by chaining together different Spline Coupling Flows, demonstrating that the bijective mechanism allows for causal inference even in more complex structures.

Causation is an incredibly important part of scientific discovery as well as is an important aspect in developing highly intelligent machines. Given that causation is at the root of scientific discovery it reasons that Normalizing flows are a powerful tool for scientists, especially when working with data that is difficult to process or may even have intractable densities. But understanding causation is an important part of creating machines that can think and reason, which requires at least some form of interventional and counterfactual estimation. While humans are not perfect, nor even that good, at determining causal relationships, such a feature is still highly desirable and is something we often seek out. After all, causality is at the root of science.

4.2. Probabilistic Modeling and Inference

Probabilistic modeling remains one of the main motivations for studying Normalizing Flows. This has rather obvious implications many times, considering these models directly approximate the density function. We will briefly discuss these works and how flows are used for these.

Variational Inference (VI) is one of the most common tasks performed with Normalizing Flows [14,39,40,54,60, 65,95,153,158,174]. VI is mainly interested in approximating the posterior distribution in Bayesian Inference. Since this is typically intractable, Flows provide a natural way to perform this. In fact, VI is seen in most of the works discussed previously. This problem can be solved by estimating the *evidence lower bound* (ELBO) and more generally thought of as similar to the "reparameterization trick."

Simply being able to estimate the distribution leads to many different tasks. Flows have been used for Bayesian Experimental Design Optimization [86], wherein they allowed for higher flexibility and accuracy. These methods typically use the expectation of entropy, wherein the density needs to be estimated. They have also been used for Imputation [105, 154], which is the replacing of missing data (such as in-painting). These methods realistically rely upon drawing from an accurate distribution, often with conditional data.

4.3. Audio

A common application for Normalizing Flows is actually found in audio based ML. Some of the best Neural Audio Synthesis and Text-To-Speech (TTS) systems actually rely on Normalizing Flows [91]. These flow based networks are broadly broken down into two categories [170]: autoregressive [113, 128, 140] and bipartite (coupling) [89, 90, 92, 119, 141, 144, 164, 185]. Autoregressive models, either AF or IAF, often have the highest quality, but come at the cost of higher computation and typically using a teacher model. Bipartite models are much faster, but require more parameters and don't reach as high of performance. That said, many of these works can synthesize speech at a high rate, faster than real time (22kHz) [128, 140, 141, 144], even doing so without a GPU and on compact systems like a RaspberryPi [91].

It is important to note that these works are not Flows from end to end, but rather use these as part of their networks. In this way they can be thought of similar to VAEs such as in NAVE [179] or VAE+Flows [173, 174]. Often this can be used to model the duration of speech [90, 120, 164], or is used to control pitch and speech style – [90, 91] use flows to align the word order. Flows are also a useful tool simply because it allows the density to be modeled, giving a meaningful performance benchmark to be measured. It is worth noting that many of these works also use more complex loss functions. VITS [91] has a loss function composed of the form:

$$L_{\text{vae}} = L_{\text{recon}} + L_{\text{KL}} + L_{\text{dur}} + L_{\text{adv}}(G) + L_{\text{fm}}(G) \tag{60}$$

reconstruction (L_{recon}), KL-divergence (L_{KL}), a variational lower bound for duration (L_{dur}), generative adversarial loss ($L_{\text{adv}}(G)$), and feature matching loss ($L_{\text{fm}}(G)$) [100]. This should be interesting to many, as for traditional image generation Flow based works focus on only using the likelihood, which has severe limitations with respect to image quality [171], but it highly depends what task we are aligning with. In the case of audio speech, where we want realistic sounding audio, it may be beneficial to actually not perfectly estimate the density distribution.

4.4. Vision

There are also many applications used in the subfield of computer vision. Many of the previous works discussed actually use works like CIFAR [99] and ImageNet [30], but generally we saw unconditional generation or class conditional generation, and these performances were rather poor. But Flows that simply regenerate the images that were passed into them can regenerate those images with little to no loss, which also makes them good for imputation and compression.Because of this, Super-Resolution is a common application area for these networks [97, 191, 196]. A unique advantage that Flows have over GANs is that they create a more diverse distribution, since they learning is based on the entire distribution rather than just good sampling, and that they can be more easily edited without additional techniques. The latter being that density methods will cluster similar features near one another, making local perturbations in the latent space produce similar outputs in the generative space. This makes more sense from an imputation perspective as it is reasonable to assume that there are many possible ways to fill in missing data, and that by having the likelihood of these different possibilities enables for better generation.

Invertible Rescaling Net (IRN) [187] proposes that the up-scaling problem (not to be confused with super-resolution) can be seen as the inverse of the down-scaling problem and develops a bijective network to account for this. This means that they are both training the compression and the decompression of the data, where gaussian noise is drawn for the missing data. Using a Nice-like [33] network, they train both the down-scaling and up-scaling networks together then are able to conditionally sample from the low-resolution image. Its follow-up work [186], using a similar network but with the 1×1-convolution and squeeze from [94], improved the technique, and also showed how this could be used for other restoration techniques like color-restoration.

A concurrent work, SRFlow [111], takes this even further, by the more powerful Glow-like [94] model and an invertible injection layer and also training with low-resolution and high-resolution image pairs, applying to the super-resolution problem. This allowed them to outperform GANs not only in quality, but also be able to modify their upscaled images directly from the latent representations, which enables better generation for artists who may wish to tune or tweak the output. The follow-up work SRFlow-DA [82] further improves the performance and enables 4x and 8x scaling.

Local Implicit Normalizing Flow (LINF) [166,188] took these methods even further, showing that Flows could be used for arbitrary scale generation. This work recognized the imputation power but introduced additional data so that the model knew the coordinates of the image that it has been conditioned upon. In this way, a recursive style generation could be formed. Importantly, they also showed that these Flow based models, and especially LINF, could smoothly control the trade-off between PSNR and LPIPS metrics through sampling temperature. Importantly, this model uses two parts, where there is a local implicit model that generates the conditions for the coordinate conditional flow model. The flow model itself is simple and uses affine layers.

These works demonstrate many advantages that flows have in image generation tasks, where even human preference prefers these images [112]. Many of these works also do not use the modern and improved flows, and there is ample opportunities for researchers to improve upon them. At the same time, many of these works chose simple models due to the fact that inference speed is important for these tasks and while more complciated networks do perform better in unconditional generation, it should be noted that they tend to suffer from higher computational costs.

5. Outlook: Future of Flows

Mathematics, data science, and machine learning all study relationships between things. We have discussed in detail Normalizing Flows, their construction, and their many uses. We see that our main motivation is to recover the original structure of the data and how these allow us to preserve all such relationships. This makes these types of models highly beneficial to many domains. We particularly see how this can help us in such cases as causal structures, which are another way to view the ordering and interaction of things. There still remains open problems and limitations, which we will conclude with.

5.1. Universal Approximation and Expressiveness

One of the main issues in Normalizing Flows is how expressive these architectures are. Due to their restrictions, they may not be able to properly express all function forms. Many of the works we've seen towards the end of each section has relatively high flexibility, and have shown to be highly effective for many applications. There are still some open questions.

Work still needs to be done to make flows more compact and *expressive*, especially with respect to *cheaper computations*. Most flows still require a larger amount of compute than other types of models. Additionally, *scaling* remains an important research area and no research to date has studied flows at the scales that we see in modern machine learning models. Modern classification networks [36, 61, 107, 108] and often in hundreds of millions of parameters, or even tens of billions [29]! GANs [20,85,183,195] are often in the tens of millions of parameters to a few hundred of millions of parameters, and have the advantage of generating from a low resolution size (typically around $\mathbb{R}^{4\times4\times512}$), and can generate images with resolution 1024×1024. Furthermore, diffusion models [32, 66, 126] can have a hundred millon paramters for a "small" dataset (64×64 [126]) to nearly a billion parameters [137, 149] (before text modeling). On the other hand, the two largest autoregressive style Flow models that the authors are currently aware of are DenseFlow [56] and MaCow [115] with 130M and 177M, respectively. These pales in comparison to modern network architectures, but it is worth noting that DenseFlow is currently one of the, if not the, best density estimators for ImageNet on 32×32 and 64×64 (work was later than MaCow). We can see that there is still a lot of potential for these models to perform better, but may take large amounts of investment and will from the community.

In addition, there has not been as much work done for proving *universal approximation* conditions. For **autoregressive style flows**, the works of Hyvärien and Pajunen [77] and Borgachev *et al.* [18] create the foundations for universal approximation claims. We will discuss [72] in a bit more detail than the reviews of [98,132] as we feel it is important to understand their value and limitations. We believe that these limitations are important to our motivation of learning Normalizing Flows as well as have important implications for Large Language Models (LLMs) and AI in general. Hyvärien and Pajunen address the nonlinear independent component analysis (ICA) problem, which has the form

$$\mathbf{x} = f(\mathbf{x}) \tag{61}$$

where $\mathbf{x}$ is the observed random variables, $\mathbf{s}$ latent variables (independent components), and $f : \mathbb{R}^n \mapsto \mathbb{R}^n$. They show that it is possible to construct a function $g : \mathbb{R}^n \mapsto \mathbb{R}^n$ such that components of $\mathbf{y} = f(\mathbf{x})$ are independent and that it has at least one solution. They give us the following theorem (near verbatim):

Theorem 9. *Assuming that $y_1,\cdots,y_m$ are independent scalar random variables which follow a joint uniform distribution in the unit cube $[0,1]^m$. Let x be any scalar random variable (such that the joint distribution of $y_1,\cdots,y_m,x$ has a probability density with respect to the Lebesgue measure of $\mathbb{R}^{m+1}$). Define g as*

$$\begin{aligned} g(a_1,\cdots,a_m,b;p_{y,x}) &= Pr(x \le b | y_i = a_i)) \\ &= \frac{\int_{-\infty}^{b} p_{y,x}(a_1,\cdots,a_m,\xi)d\xi}{p_y(a_1,\cdots,a_m)} \end{aligned} \tag{62}$$

where $p_y(\cdot)$ and $p_{y,x}(\cdot)$ are the marginal densities of $(y_{\le m})$ and $(y_{\le m}, x)$. Set

$$y_{m+1}. = g(y_1,\cdots,y_m,x;p_{y,x}) \tag{63}$$

Then y_{m+1} is independent from $y_1, \cdots, y_m$. In particular, the variables $y_1, \cdots, y_{m+1}$ are jointly uniformly distributed in the unit cube $[0,1]^{m+1}$.

They relate this to Gram-Schmidt orthogonalization and we can see that this recursive process looks just like the autoregressive process (if we drop the absolute value of the Jacobian determinant and use its inverse). Their proof even uses the change of variables to create a bijection.

There is an important limitation to this formulation, as we have no guarantee that number of independent components is equal to the number of dimensions: n. If the number of independent components is more than n, the construction only gives n variables. If the number of independent components is less than n, then $p(\mathbf{x})$ is only supported on a smaller dimension (degenerate), and thus the construction would not be defined. This is a critical issue if one presumes the manifold hypothesis for the distribution of data that one is working with.

Hyvärien and Pajunen also show that the formulation does not create a unique solution, and that this means there are many indeterminacies to the problem. Due to this, just constraining the elements to be independent does not make the elements identifiable. In other words, we do not have an unambiguous isomorphic mapping between the source and target distributions. This has important consequences for comparing models, interpretability, and causality in general as it shows that we can generate different paths between our source and target distributions, with different interpretations of what variables interact and how.

The work of Borgachev *et al.* [18] further formalizes these ideas through the language of metric theory and optimal transport theory. We will leave this for the readers, as we do not expect the language to be as clear and apparent as [77], but encourage trying to understand this in detail. Their formulation more specifically uses the idea of increasing triangular mappings, which our autoregressive model follows, since our Jacobian determinite should fit this case. That is because for there to be a proper inverse the matrix needs to be full rank, which requires non-zero entries on the diagonal for a triangular matrix. Our transform is always positive definite because we use the absolute value of the Jacobian determinant.

If we recall that the formulations have the two following Jacobians in block matrix form:

$$\begin{matrix} \text{Autoregressive:} & \text{Coupling:} \\ \begin{bmatrix} \mathbf{L}_0 & \mathbf{0} \\ \mathbf{M} & \mathbf{L}_1 \end{bmatrix} & \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{M} & \mathbf{D} \end{bmatrix} \end{matrix} \tag{64}$$

We note here that as long as $\mathbf{L}_0$, $\mathbf{L}_1$, and $\mathbf{D}$ have non-zero entries, then the composition is increasingly triangular. From here we can see that *the autoregressive and coupling formulations allow for universal approximation of any density.* Papamakarios *et al.* [132] notes that we can extend this proof to any target density if we use the unit cube as an intermediate distribution ($F : S \mapsto [0,1]^n \mapsto T$). We must clarify that this does not mean every network or architecture is sufficient to approximate and an **open problem** remains in analyzing the necessary complexity, width, and depth necessary for universal approximation. Jaini *et al.* [81] did show that their network met the sufficient conditions and expressiveness to achieve universal approximation.

For **Sylvester** Flows the question is still open, but given that there is a block diagonal representation it reasons that it is possible to construct a similar triangular mapping representation as above. For **Monotonic Flows** it is obvious that for a function that is non-constant everywhere can be approximated to arbitrary precision if the correct bounds are selected via a piecewise combination of monotonic functions. This still remains open work available to anyone who may wish to take on these challenges.

5.2. Future Applications

There are a lot of future applications for Normalizing Flows that still need to be applied. We've seen how these networks are motivating to areas where you wish to control your latent variables and have a more interpretable network, even as just part of a network. We've also seen how they are highly useful for scientific domains where we want to maintain structure and interpretability, or perform density estimation with precision. As Flows advance more, we will see them used within more applications and they are now likely sufficient for wider use than they currently have.

Little work has been done with respect to discrete distributions, but this remains a powerful motivation, especially in the era of LLMs. Similarly, no one has tried to create a large IAF to compete with these works, but given the cost, this is unsurprising.

Additionally, while working with tabular datasets is not as flashy or popular, there is still work to be done in this region. A recent famous paper "Why do tree-based models still outperform deep learning on tabular data?" [59]shows that there is still motivation for working with tabular data in the data science domain, but works such as these have not compared to Flows, as simply Flows have focused on density estimation rather than classification or regression. These are not limitations of the networks, but simply relatively unexplored areas.

5.3. Where To Next?

Flows may play an important role in the future of ML. We specifically discussed using more abstract language to prevent us from narrowing our views too much. We recognize that mathematics will be an important part towards building future AGI systems, and without a doubt we know that isomorphisms play an important role within mathematics. We believe that flows will play an important role in this future, especially in regards to interpreting data and controlling model parameters.

References

[1] Adam Golinski, Mario Lezcano-Casado, and Tom Rainforth. Improving Normalizing Flows via Better Orthogonal Parameterizations. In *Workshop on Invertible Neural Nets and Normalizing Flows*, 2019. 12

[2] Juan P Agnelli, M Cadeiras, Esteban G Tabak, Cristina Vilma Turner, and Eric Vanden-Eijnden. Clustering and classification through normalizing flows in feature space. *Multiscale Modeling & Simulation*, 8(5):1784–1802, 2010. Publisher: SIAM. 5

[3] Forest Agostinelli, Matthew Hoffman, Peter Sadowski, and Pierre Baldi. Learning Activation Functions to Improve Deep Neural Networks, Apr. 2015. arXiv:1412.6830 [cs, stat]. 9

[4] Byeongkeun Ahn, Chiyoon Kim, Youngjoon Hong, and Hyunwoo J. Kim. Invertible Monotone Operators for Normalizing Flows, Oct. 2022. arXiv:2210.08176 [cs]. 17

[5] Brandon Amos and J. Zico Kolter. OptNet: Differentiable Optimization as a Layer in Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 136–145. PMLR, July 2017. ISSN: 2640-3498. 17

[6] Marlow Anderson and Todd Feil. *A first course in abstract algebra: rings, groups, and fields*. CRC Press, 2014. 4

[7] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization, July 2016. arXiv:1607.06450 [cs, stat]. 10

[8] Vijay Badrinarayanan, Bamdev Mishra, and Roberto Cipolla. Understanding symmetries in deep networks, Nov. 2015. arXiv:1511.01029 [cs]. 10

[9] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep Equilibrium Models, Oct. 2019. arXiv:1909.01377 [cs, stat]. 17

[10] Jens Behrmann, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible Residual Networks, May 2019. arXiv:1811.00995 [cs, stat]. 15, 16

[11] Jean-David Benamou and Yann Brenier. A computational fluid mechanics solution to the Monge-Kantorovich mass transfer problem. *Numerische Mathematik*, 84(3):375–393, Jan. 2000. 18

[12] Yoshua Bengio and Samy Bengio. Modeling High-Dimensional Discrete Data with Multi-Layer Neural Networks. In *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. 11

[13] Rianne van den Berg, Alexey A. Gritsenko, Mostafa Dehghani, Casper Kaae Sønderby, and Tim Salimans. IDF++: Analyzing and Improving Integer Discrete Flows for Lossless Compression. Oct. 2020. 6

[14] Rianne van den Berg, Leonard Hasenclever, Jakub M. Tomczak, and Max Welling. Sylvester Normalizing Flows for Variational Inference, Feb. 2019. arXiv:1803.05649 [cs, stat]. 12, 20

[15] Å. Björck and C. Bowie. An Iterative Algorithm for Computing the Best Estimate of an Orthogonal Matrix. *SIAM Journal on Numerical Analysis*, 8(2):358–364, 1971. Publisher: Society for Industrial and Applied Mathematics. 12

[16] James F Blinn. How to Solve a Cubic Equation Part 2 – The 1 1 Case. 14

[17] James F. Blinn. How to Solve a Cubic Equation, Part 5: Back to Numerics. *IEEE Computer Graphics and Applications*, 27(3):78–89, May 2007. 14

[18] V I Bogachev, A V Kolesnikov, and K V Medvedev. Triangular transformations of measures. *Sbornik: Mathematics*, 196(3):309–335, Apr. 2005. 22, 23

[19] Nicolas Bourbaki. Structures. In *Theory of Sets*, pages 259–346. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. 3

[20] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. Sept. 2018. 22

[21] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners, July 2020. arXiv:2005.14165 [cs]. 11

[22] Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis. Physics-informed neural networks (PINNs) for fluid mechanics: a review. *Acta Mechanica Sinica*, 37(12):1727–1738, Dec. 2021. 7

[23] Matteo Capucci, Bruno Gavranović, Jules Hedges, and Eigil Fjeldgren Rischel. Towards Foundations of Categorical Cybernetics. *Electronic Proceedings in Theoretical Computer Science*, 372:235–248, Nov. 2022. arXiv:2105.06332 [math]. 7

[24] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018. 17, 19

[25] Ricky T. Q. Chen, Jens Behrmann, David Duvenaud, and Jörn-Henrik Jacobsen. Residual Flows for Invertible Generative Modeling, July 2020. arXiv:1906.02735 [cs, stat]. 16, 17, 18

[26] Scott Chen and Ramesh Gopinath. Gaussianization. In *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000. 5

[27] Eugenia Cheng. *The Joy of Abstraction: An Exploration of Math, Category Theory, and Life*. Cambridge University Press, 2022. 2

[28] Leo Corry. Nicolas Bourbaki and the Concept of Mathematical Structure. *Synthese*, 92(3):315–348, 1992. Publisher: Springer. 3

[29] Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, Rodolphe Jenatton, Lucas Beyer, Michael Tschannen, Anurag Arnab, Xiao Wang, Carlos Riquelme, Matthias

Minderer, Joan Puigcerver, Utku Evci, and Manoj Kumar. Scaling Vision Transformers to 22 Billion Parameters. 22

[30] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009. ISSN: 1063-6919. 15, 21

[31] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. 11

[32] Prafulla Dhariwal and Alexander Quinn Nichol. Diffusion Models Beat GANs on Image Synthesis. Nov. 2021. 22

[33] Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-linear Independent Components Estimation, Apr. 2015. arXiv:1410.8516 [cs]. 5, 11, 21

[34] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP, Feb. 2017. arXiv:1605.08803 [cs, stat]. 9

[35] Hadi M. Dolatabadi, Sarah Erfani, and Christopher Leckie. Invertible Generative Modeling using Linear Rational Splines, Apr. 2020. arXiv:2001.05168 [cs, stat]. 14, 15

[36] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. Oct. 2020. 22

[37] Yilun Du and Igor Mordatch. Implicit Generation and Generalization in Energy-Based Models, June 2020. arXiv:1903.08689 [cs, stat]. 6, 7

[38] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented Neural ODEs. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. 19

[39] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Cubic-Spline Flows, June 2019. arXiv:1906.02145 [cs, stat]. 14, 20

[40] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural Spline Flows. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. 14, 15, 20

[41] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning, Nov. 2017. arXiv:1702.03118 [cs]. 16

[42] Chris Finlay, Joern-Henrik Jacobsen, Levon Nurbekyan, and Adam Oberman. How to Train Your Neural ODE: the World of Jacobian and Kinetic Regularization. In *Proceedings of the 37th International Conference on Machine Learning*, pages 3154–3164. PMLR, Nov. 2020. ISSN: 2640-3498. 18

[43] Brendan Fong, David I. Spivak, and Rémy Tuyéras. Backprop as Functor: A compositional perspective on supervised learning, May 2019. arXiv:1711.10455 [cs, math]. 7

[44] Jerome H. Friedman. Exploratory Projection Pursuit. *Journal of the American Statistical Association*, 82(397):249–266, 1987. Publisher: [American Statistical Association, Taylor & Francis, Ltd.]. 5

[45] F. N. Fritsch and R. E. Carlson. Monotone Piecewise Cubic Interpolation. *SIAM Journal on Numerical Analysis*, 17(2):238–246, Apr. 1980. 14

[46] Richard D. Fuhr and Michael Kallay. Monotone linear rational spline interpolation. *Computer Aided Geometric Design*, 9(4):313–319, Sept. 1992. 15

[47] Bruno Gavranović. Category Theory Machine Learning, July 2020. original-date: 2020-07-09T14:16:58Z. 7

[48] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. MADE: Masked Autoencoder for Distribution Estimation. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 881–889. PMLR, June 2015. ISSN: 1938-7228. 11

[49] Laurent El Ghaoui, Fangda Gu, Bertrand Travacca, Armin Askari, and Alicia Y. Tsai. Implicit Deep Learning, Aug. 2020. arXiv:1908.06315 [cs, math, stat]. 17

[50] Wenbo Gong, Joel Jennings, Cheng Zhang, and Nick Pawlowski. Rhino: Deep Causal Temporal Relationship Learning with History-dependent Noise. Feb. 2023. 20

[51] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. 7, 8

[52] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks, June 2014. arXiv:1406.2661 [cs, stat]. 7

[53] Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J. Cree. Regularisation of Neural Networks by Enforcing Lipschitz Continuity, Aug. 2020. arXiv:1804.04368 [cs, stat]. 16

[54] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. FFJORD: Free-Form Continuous Dynamics for Scalable Reversible Generative Models. Dec. 2018. 18, 20

[55] Will Grathwohl, Kuan-Chieh Wang, Joern-Henrik Jacobsen, David Duvenaud, Mohammad Norouzi, and Kevin Swersky. Your classifier is secretly an energy based model and you should treat it like one. Dec. 2019. 7

[56] Matej Grcić, Ivan Grubišić, and Siniša Šegvić. Densely connected normalizing flows. Nov. 2021. 22

[57] J. A. GREGORY and R. DELBOURGO. Piecewise Rational Quadratic Interpolation to Monotonic Data. *IMA Journal of Numerical Analysis*, 2(2):123–130, Apr. 1982. _eprint: https://academic.oup.com/imajna/article-pdf/2/2/123/2267745/2-2-123.pdf. 15

[58] Ulf Grenander and Michael I Miller. *Pattern theory: from representation to inference*. OUP Oxford, 2006. 7

[59] Leo Grinsztajn, Edouard Oyallon, and Gael Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? June 2022. 23

[60] Leonard Hasenclever, Jakub M Tomczak, and Max Welling. Variational Inference with Orthogonal Normalizing Flows. In *Bayesian Deep Learning*, 2017. 12, 20

[61] Ali Hassani, Steven Walton, Jiachen Li, Shen Li, and Humphrey Shi. Neighborhood Attention Transformer. 22

[62] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, Dec. 2015. ISSN: 2380-7504. 9

[63] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016. ISSN: 1063-6919. 15

[64] Heinz H. Bauschke and Patrick L. Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. CMS Books in Mathematics. Springer, 2011. 17

[65] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design, May 2019. arXiv:1902.00275 [cs, stat]. 6, 11, 20

[66] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models, Dec. 2020. arXiv:2006.11239 [cs, stat]. 5, 7, 22

[67] Emiel Hoogeboom, Rianne van den Berg, and Max Welling. Emerging Convolutions for Generative Normalizing Flows, May 2019. arXiv:1901.11137 [cs, stat]. 10

[68] Emiel Hoogeboom, Jorn Peters, Rianne Van Den Berg, and Max Welling. Integer discrete flows and lossless compression. *Advances in Neural Information Processing Systems*, 32, 2019. 6

[69] Emiel Hoogeboom, Jakub M Tomczak, Victor Garcia Satorras, and Max Welling. The Convolution Exponential and Generalized Sylvester Flows. volume 33. 12

[70] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, Jan. 1989. 7

[71] Alston S. Householder. Unitary Triangularization of a Nonsymmetric Matrix. *Journal of the ACM*, 5(4):339–342, Oct. 1958. 10

[72] Patrik Hoyer, Dominik Janzing, Joris M Mooij, Jonas Peters, and Bernhard Schölkopf. Nonlinear causal discovery with additive noise models. In *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc., 2008. 20, 22

[73] Gao Huang, Zhuang Liu, Geoff Pleiss, Laurens van der Maaten, and Kilian Q. Weinberger. Convolutional Networks with Dense Connectivity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(12):8704–8716, Dec. 2022. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence. 16

[74] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks, Jan. 2018. arXiv:1608.06993 [cs] version: 5. 16

[75] M.F. Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, 19(2):433–450, Jan. 1990. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/03610919008812866. 16

[76] Aapo Hyvärinen. Estimation of Non-Normalized Statistical Models by Score Matching. *Journal of Machine Learning Research*, 6(24):695–709, 2005. 7

[77] Aapo Hyvärinen and Petteri Pajunen. Nonlinear independent component analysis: Existence and uniqueness results. *Neural Networks*, 12(3):429–439, Apr. 1999. 22, 23

[78] Aapo Hyvärinen and Stephen M. Smith. Pairwise likelihood ratios for estimation of non-Gaussian structural equation models. *The Journal of Machine Learning Research*, 14(1):111–152, Jan. 2013. 20

[79] Maximilian Ilse, Patrick Forré, Max Welling, and Joris M. Mooij. Combining Interventional and Observational Data Using Causal Reductions, Feb. 2023. arXiv:2103.04786 [cs, stat]. 20

[80] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 448–456. PMLR, June 2015. ISSN: 1938-7228. 10

[81] Priyank Jaini, Kira A. Selby, and Yaoliang Yu. Sum-of-Squares Polynomial Flow. In *Proceedings of the 36th International Conference on Machine Learning*, pages 3009–3018. PMLR, May 2019. ISSN: 2640-3498. 23

[82] Younghyun Jo, Sejong Yang, and Seon Joo Kim. SRFlow-DA: Super-Resolution Using Normalizing Flow With Deep Convolutional Block. pages 364–372, 2021. 21

[83] Pearl Judea. An Introduction to Causal Inference. *The International Journal of Biostatistics*, 6(2):1–62, 2010. Publisher: De Gruyter. 19

[84] Herman Kahn. Use of Different Monte Carlo Sampling Techniques. Technical report, RAND Corporation, Jan. 1955. 16

[85] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-Free Generative Adversarial Networks. 22

[86] Noble Kennamer, Steven Walton, and Alexander Ihler. Design Amortization for Bayesian Optimal Experimental Design. Feb. 2023. arXiv:2210.03283 [cs, stat]. 21

[87] Ilyes Khemakhem, Ricardo Pio Monti, Robert Leech, and Aapo Hyvärinen. Causal Autoregressive Flows, Feb. 2021. arXiv:2011.02268 [cs, stat]. 20

[88] Patrick Kidger. On Neural Differential Equations, Feb. 2022. arXiv:2202.02435 [cs, math, stat]. 17

[89] Hyeongju Kim, Hyeonseung Lee, Woo Hyun Kang, Sung Jun Cheon, Byoung Jin Choi, and Nam Soo Kim. WaveNODE: A Continuous Normalizing Flow for Speech Synthesis. June 2020. 21

[90] Jaehyeon Kim, Sungwon Kim, Jungil Kong, and Sungroh Yoon. Glow-TTS: A Generative Flow for Text-to-Speech via Monotonic Alignment Search. In *Advances in Neural Information Processing Systems*, volume 33, pages 8067–8077. Curran Associates, Inc., 2020. 21

[91] Jaehyeon Kim, Jungil Kong, and Juhee Son. Conditional Variational Autoencoder with Adversarial Learning for End-to-End Text-to-Speech. In *Proceedings of the*

38th International Conference on Machine Learning, pages 5530–5540. PMLR, July 2021. ISSN: 2640-3498. 21

[92] Sungwon Kim, Sang-Gil Lee, Jongyoon Song, Jaehyeon Kim, and Sungroh Yoon. FloWaveNet : A Generative Flow for Raw Audio. In *Proceedings of the 36th International Conference on Machine Learning*, pages 3370–3378. PMLR, May 2019. ISSN: 2640-3498. 21

[93] Durk P Kingma and Yann Cun. Regularized estimation of image statistics by Score Matching. In *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010. 7

[94] Durk P Kingma and Prafulla Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. 9, 18, 21

[95] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved Variational Inference with Inverse Autoregressive Flow. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. 12, 20

[96] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes, Dec. 2022. arXiv:1312.6114 [cs, stat]. 10, 20

[97] Kyungdeuk Ko, Bokyeung Lee, Jonghwan Hong, Donghyeon Kim, and Hanseok Ko. MRIFlow: Magnetic resonance image super-resolution based on normalizing flow and frequency prior. *Journal of Magnetic Resonance*, 352:107477, July 2023. 21

[98] Ivan Kobyzev, Simon J. D. Prince, and Marcus A. Brubaker. Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3964–3979, Nov. 2021. arXiv:1908.09257 [cs, stat]. 1, 6, 7, 10, 22

[99] Alex Krizhevsky, Geoffrey Hinton, and others. Learning multiple layers of features from tiny images. 2009. Publisher: Toronto, ON, Canada. 15, 21

[100] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1558–1566. PMLR, June 2016. ISSN: 1938-7228. 21

[101] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and Fujie Huang. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006. 7

[102] Klas Leino, Zifan Wang, and Matt Fredrikson. Globally-Robust Neural Networks, June 2021. arXiv:2102.08452 [cs, stat]. 16

[103] Tom Leinster. Basic Category Theory, Dec. 2016. arXiv:1612.09375 [math]. 2

[104] Qiyang Li, Saminul Haque, Cem Anil, James Lucas, Roger Grosse, and Jörn-Henrik Jacobsen. Preventing Gradient Attenuation in Lipschitz Constrained Convolutional Networks, Nov. 2019. arXiv:1911.00937 [cs, stat]. 16

[105] Yang Li, Shoaib Akbar, and Junier Oliva. ACFlow: Flow Models for Arbitrary Conditional Likelihoods. In *Proceedings of the 37th International Conference on Machine Learning*, pages 5831–5841. PMLR, Nov. 2020. ISSN: 2640-3498. 21

[106] Jenny Liu and Aviral Kumar. Graph Normalizing Flows. 6

[107] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, Furu Wei, and Baining Guo. Swin Transformer V2: Scaling Up Capacity and Resolution. 22

[108] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A ConvNet for the 2020s. 22

[109] Cheng Lu, Jianfei Chen, Chongxuan Li, Qiuhao Wang, and Jun Zhu. Implicit Normalizing Flows, Mar. 2021. arXiv:2103.09527 [cs, stat]. 6, 17

[110] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond Finite Layer Neural Networks: Bridging Deep Architectures and Numerical Differential Equations, Mar. 2020. arXiv:1710.10121 [cs, stat]. 17

[111] Andreas Lugmayr, Martin Danelljan, Luc Van Gool, and Radu Timofte. SRFlow: Learning the Super-Resolution Space with Normalizing Flow. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, Lecture Notes in Computer Science, pages 715–732, Cham, 2020. Springer International Publishing. 21

[112] Andreas Lugmayr, Martin Danelljan, Fisher Yu, Luc Van Gool, and Radu Timofte. Normalizing Flow as a Flexible Fidelity Objective for Photo-Realistic Super-Resolution. pages 1756–1765, 2022. 22

[113] Manh Luong and Viet Anh Tran. FlowVocoder: A small Footprint Neural Vocoder based Normalizing Flow for Speech Synthesis. In *Interspeech 2022*, pages 1576–1580. ISCA, Sept. 2022. 21

[114] Anne-Marie Lyne, Mark Girolami, Yves Atchadé, Heiko Strathmann, and Daniel Simpson. On Russian Roulette Estimates for Bayesian Inference with Doubly-Intractable Likelihoods. *Statistical Science*, 30(4), Nov. 2015. arXiv:1306.4032 [stat]. 16

[115] Xuezhe Ma, Xiang Kong, Shanghang Zhang, and Eduard Hovy. MaCow: Masked Convolutional Generative Flow. 22

[116] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. 9

[117] Barry Mazur. When is one thing equal to some other thing? 2

[118] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled Generative Adversarial Networks, May 2017. arXiv:1611.02163 [cs, stat]. 7

[119] Chenfeng Miao, Shuang Liang, Minchuan Chen, Jun Ma, Shaojun Wang, and Jing Xiao. Flow-TTS: A Non-Autoregressive Network for Text to Speech Based on Flow. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7209–7213, May 2020. ISSN: 2379-190X. 21

[120] Chenfeng Miao, Liang Shuang, Zhengchen Liu, Chen Minchuan, Jun Ma, Shaojun Wang, and Jing Xiao. EfficientTTS: An Efficient and High-Quality Text-to-Speech Architecture. In *Proceedings of the 38th International Conference on Machine Learning*, pages 7700–7709. PMLR, July 2021. ISSN: 2640-3498. 21

[121] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks, Feb. 2018. arXiv:1802.05957 [cs, stat]. 16

[122] Shakir Mohamed and Balaji Lakshminarayanan. Learning in Implicit Generative Models, Feb. 2017. arXiv:1610.03483 [cs, stat]. 7

[123] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012. 7

[124] Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural Importance Sampling. *ACM Trans. Graph.*, 38(5):145:1–145:19, Oct. 2019. Place: New York, NY, USA Publisher: ACM. 13, 14

[125] Arash Nasr-Esfahany, Mohammad Alizadeh, and Devavrat Shah. Counterfactual Identifiability of Bijective Causal Models, June 2023. arXiv:2302.02228 [cs, stat]. 20

[126] Alex Nichol and Prafulla Dhariwal. Improved Denoising Diffusion Probabilistic Models. 22

[127] nLab authors. nLab, June 2023. 2

[128] Aaron Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George Driessche, Edward Lockhart, Luis Cobo, Florian Stimberg, Norman Casagrande, Dominik Grewe, Seb Noury, Sander Dieleman, Erich Elsen, Nal Kalchbrenner, Heiga Zen, Alex Graves, Helen King, Tom Walters, Dan Belov, and Demis Hassabis. Parallel WaveNet: Fast High-Fidelity Speech Synthesis. In *Proceedings of the 35th International Conference on Machine Learning*, pages 3918–3926. PMLR, July 2018. ISSN: 2640-3498. 21

[129] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A Generative Model for Raw Audio, Sept. 2016. arXiv:1609.03499 [cs]. 12

[130] Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel Recurrent Neural Networks. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1747–1756. PMLR, June 2016. ISSN: 1938-7228. 12

[131] George Papamakarios. Neural Density Estimation and Likelihood-free Inference, Oct. 2019. arXiv:1910.13233 [cs, stat]. 1

[132] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing Flows for Probabilistic Modeling and Inference. *J. Mach. Learn. Res.*, 22(1), Jan. 2021. Publisher: JMLR.org. 1, 5, 22, 23

[133] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked Autoregressive Flow for Density Estimation. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 12

[134] Judea Pearl. Causal inference in statistics: An overview. *Statistics Surveys*, 3(none):96–146, Jan. 2009. Publisher: Amer. Statist. Assoc., the Bernoulli Soc., the Inst. Math. Statist., and the Statist. Soc. Canada. 19

[135] Judea Pearl. The Mathematics of Causal Relations. In *Causality and Psychopathology*. Oxford University Press, Feb. 2011. 19

[136] Judea Pearl. The Do-Calculus Revisited, Oct. 2012. arXiv:1210.4852 [cs, stat]. 19

[137] William Peebles and Saining Xie. Scalable Diffusion Models with Transformers, Mar. 2023. arXiv:2212.09748 [cs]. 22

[138] Yura Perugachi-Diaz, Jakub M. Tomczak, and Sandjai Bhulai. Invertible DenseNets with Concatenated LipSwish, Oct. 2021. arXiv:2102.02694 [cs, stat]. 16

[139] Christoph Peters. How to solve a cubic equation, revisited. *Moments in Graphics*, 2016. 14

[140] Wei Ping, Kainan Peng, and Jitong Chen. ClariNet: Parallel Wave Generation in End-to-End Text-to-Speech. Sept. 2018. 21

[141] Wei Ping, Kainan Peng, Kexin Zhao, and Zhao Song. WaveFlow: A Compact Flow-based Model for Raw Audio. In *Proceedings of the 37th International Conference on Machine Learning*, pages 7706–7716. PMLR, Nov. 2020. ISSN: 2640-3498. 21

[142] Henri Poincaré and Francis Maitland. *Science and method*. Courier Corporation, 2003. 2, 3

[143] Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. CRC press, 1987. 18

[144] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A Flow-based Generative Network for Speech Synthesis. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3617–3621, May 2019. ISSN: 2379-190X. 21

[145] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training. 11

[146] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. 11

[147] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, Feb. 2019. 7

[148] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for Activation Functions, Oct. 2017. arXiv:1710.05941 [cs] version: 1. 16

[149] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-Shot Text-to-Image Generation. In *Proceedings of the 38th International Conference on Machine Learning*, pages 8821–8831. PMLR, July 2021. ISSN: 2640-3498. 22

[150] Rajat Rasal, Daniel C. Castro, Nick Pawlowski, and Ben Glocker. Deep Structural Causal Shape Models, Aug. 2022. arXiv:2208.10950 [cs]. 20

[151] Nico Reick, Felix Wiewel, Alexander Bartler, and Bin Yang. Estimation of Bivariate Structural Causal Models by Variational Gaussian Process Regression Under Likelihoods Parametrised by Normalising Flows, Sept. 2021. arXiv:2109.02521 [cs, stat]. 20

[152] Danilo Rezende and Shakir Mohamed. Variational Inference with Normalizing Flows. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1530–1538. PMLR, June 2015. ISSN: 1938-7228. 6, 12, 13

[153] Danilo Jimenez Rezende, George Papamakarios, Sebastien Racaniere, Michael Albergo, Gurtej Kanwar, Phiala Shanahan, and Kyle Cranmer. Normalizing Flows on Tori and Spheres. In *Proceedings of the 37th International Conference on Machine Learning*, pages 8083–8092. PMLR, Nov. 2020. ISSN: 2640-3498. 20

[154] Trevor W. Richardson, Wencheng Wu, Lei Lin, Beilei Xu, and Edgar A. Bernal. McFlow: Monte Carlo Flow Models for Data Imputation. pages 14205–14214, 2020. 21

[155] Ernest K Ryu and Stephen Boyd. A PRIMER ON MONOTONE OPERATOR METHODS. *APPL. COMPUT. MATH.*, 2016. 17

[156] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved Techniques for Training GANs. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. 7

[157] Tim Salimans and Diederik P. Kingma. Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks, June 2016. arXiv:1602.07868 [cs]. 10

[158] Hadi Salman, Payman Yadollahpour, Tom Fletcher, and Kayhan Batmanghelich. Deep Diffeomorphic Normalizing Flows, Nov. 2018. arXiv:1810.03256 [cs, stat]. 20

[159] Randall E Schumacker and Richard G Lomax. *A beginner's guide to structural equation modeling*. psychology press, 2004. 20

[160] Anton Maximilian Schäfer and Hans Georg Zimmermann. Recurrent Neural Networks Are Universal Approximators. In Stefanos D. Kollias, Andreas Stafylopatis, Włodzisław Duch, and Erkki Oja, editors, *Artificial Neural Networks – ICANN 2006*, Lecture Notes in Computer Science, pages 632–640, Berlin, Heidelberg, 2006. Springer. 7

[161] Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units, July 2016. arXiv:1603.05201 [cs]. 16

[162] Jun Shao. *Mathematical statistics*. Springer Science & Business Media, 2003. 2, 4, 6

[163] Dan Shiebler, Bruno Gavranović, and Paul Wilson. Category Theory in Machine Learning, June 2021. arXiv:2106.07032 [cs]. 7

[164] Kevin J. Shih, Rafael Valle, Rohan Badlani, Adrian Lancucki, Wei Ping, and Bryan Catanzaro. RAD-TTS: Parallel Flow-Based TTS with Robust Alignment Learning and Diverse Synthesis. June 2021. 21

[165] John Skilling. The eigenvalues of mega-dimensional matrices. *Maximum Entropy and Bayesian Methods: Cambridge, England, 1988*, pages 455–466, 1989. Publisher: Springer. 16

[166] Ki-Ung Song, Dongseok Shim, Kang-Wook Kim, Jae-Young Lee, and Younggeun Kim. FS-NCSR: Increasing Diversity of the Super-Resolution Space via Frequency Separation and Noise-Conditioned Normalizing Flow. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 967–976, June 2022. ISSN: 2160-7516. 7, 22

[167] Yang Song and Stefano Ermon. Improved Techniques for Training Score-Based Generative Models, Oct. 2020. arXiv:2006.09011 [cs, stat]. 7

[168] Matthias Steffen. A simple method for monotonic interpolation in one dimension. *Astronomy and Astrophysics*, 239:443, 1990. 14

[169] Esteban G Tabak and Eric Vanden-Eijnden. Density estimation by dual ascent of the log-likelihood. *Communications in Mathematical Sciences*, 8(1):217–233, 2010. Publisher: International Press of Boston. 5

[170] Xu Tan, Tao Qin, Frank Soong, and Tie-Yan Liu. A Survey on Neural Speech Synthesis, July 2021. arXiv:2106.15561 [cs, eess]. 21

[171] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models, Apr. 2016. arXiv:1511.01844 [cs, stat]. 21

[172] Jakub M Tomczak. *Deep generative modeling*. Springer, 2022. 7

[173] Jakub M. Tomczak and Max Welling. Improving Variational Auto-Encoders using convex combination linear Inverse Autoregressive Flow, June 2017. arXiv:1706.02326 [stat]. 11, 21

[174] Jakub M. Tomczak and Max Welling. Improving Variational Auto-Encoders using Householder Flow, Jan. 2017. arXiv:1611.09630 [cs, stat]. 10, 20, 21

[175] Dustin Tran, Keyon Vafa, Kumar Agrawal, Laurent Dinh, and Ben Poole. Discrete Flows: Invertible Generative Models of Discrete Data. Apr. 2019. 6

[176] Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama. Lipschitz-Margin Training: Scalable Certification of Perturbation Invariance for Deep Neural Networks. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. 16

[177] Silviu-Marian Udrescu, Andrew Tan, Jiahai Feng, Orisvaldo Neto, Tailin Wu, and Max Tegmark. AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. In *Advances in Neural Information Processing Systems*, volume 33, pages 4860–4871. Curran Associates, Inc., 2020. 6, 7

[178] Benigno Uria, Iain Murray, and Hugo Larochelle. A Deep and Tractable Density Estimator. In *Proceedings of the 31st International Conference on Machine Learning*, pages 467–475. PMLR, Jan. 2014. ISSN: 1938-7228. 11

[179] Arash Vahdat and Jan Kautz. NVAE: A Deep Hierarchical Variational Autoencoder. In *Advances in Neural Information Processing Systems*, volume 33, pages 19667–19679. Curran Associates, Inc., 2020. 21

[180] Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, koray kavukcuoglu, Oriol Vinyals, and Alex Graves. Conditional Image Generation with PixelCNN Decoders. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. 12

[181] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, Dec. 2017. arXiv:1706.03762 [cs]. 11

[182] Cédric Villani. *Topics in optimal transportation*, volume 58. American Mathematical Soc., 2021. 6, 18

[183] Steven Walton, Ali Hassani, Xingqian Xu, Zhangyang Wang, and Humphrey Shi. StyleNAT: Giving Each Head a New Perspective, Nov. 2022. arXiv:2211.05770 [cs]. 7, 22

[184] Rongguang Wang, Pratik Chaudhari, and Christos Davatzikos. Harmonization with Flow-based Causal Inference, July 2021. arXiv:2106.06845 [cs, eess, stat]. 20

[185] Ron J. Weiss, RJ Skerry-Ryan, Eric Battenberg, Soroosh Mariooryad, and Diederik P. Kingma. Wave-Tacotron: Spectrogram-Free End-to-End Text-to-Speech Synthesis. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5679–5683, June 2021. ISSN: 2379-190X. 21

[186] Mingqing Xiao, Shuxin Zheng, Chang Liu, Zhouchen Lin, and Tie-Yan Liu. Invertible Rescaling Network and Its Extensions. *International Journal of Computer Vision*, 131(1):134–159, Jan. 2023. 21

[187] Mingqing Xiao, Shuxin Zheng, Chang Liu, Yaolong Wang, Di He, Guolin Ke, Jiang Bian, Zhouchen Lin, and Tie-Yan Liu. Invertible Image Rescaling. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, Lecture Notes in Computer Science, pages 126–144, Cham, 2020. Springer International Publishing. 21

[188] Jie-En Yao, Li-Yuan Tsao, Yi-Chen Lo, Roy Tseng, Chia-Che Chang, and Chun-Yi Lee. Local Implicit Normalizing Flow for Arbitrary-Scale Image Super-Resolution. pages 1776–1785, 2023. 22

[189] Weiran Yao, Yuewen Sun, Alex Ho, Changyin Sun, and Kun Zhang. Learning Temporally Causal Latent Processes from General Temporal Data, Feb. 2022. arXiv:2110.05428 [cs, stat]. 20

[190] Yuichi Yoshida and Takeru Miyato. Spectral Norm Regularization for Improving the Generalizability of Deep Learning, May 2017. arXiv:1705.10941 [cs, stat]. 16

[191] Jason J. Yu, Konstantinos G. Derpanis, and Marcus A Brubaker. Wavelet Flow: Fast Training of High Resolution Normalizing Flows. In *Advances in Neural Information Processing Systems*, volume 33, pages 6184–6196. Curran Associates, Inc., 2020. 21

[192] Bohang Zhang, Du Jiang, Di He, and Liwei Wang. Rethinking Lipschitz Neural Networks and Certified Robustness: A Boolean Function Perspective, Oct. 2022. arXiv:2210.01787 [cs, stat]. 16

[193] Han Zhang, Xi Gao, Jacob Unterman, and Tom Arodz. Approximation Capabilities of Neural ODEs and Invertible Residual Networks. In *Proceedings of the 37th International Conference on Machine Learning*, pages 11086–11095. PMLR, Nov. 2020. ISSN: 2640-3498. 19

[194] Kun Zhang, Zhikun Wang, Jiji Zhang, and Bernhard Schölkopf. On Estimation of Functional Causal Models: General Results and Application to the Post-Nonlinear Causal Model. *ACM Trans. Intell. Syst. Technol.*, 7(2), Dec. 2015. Place: New York, NY, USA Publisher: Association for Computing Machinery. 20

[195] Long Zhao, Zizhao Zhang, Ting Chen, Dimitris N. Metaxas, and Han Zhang. Improved Transformer for High-Resolution GANs. Nov. 2021. 22

[196] Yiming Zhu, Cairong Wang, Chenyu Dong, Ke Zhang, Hongyang Gao, and Chun Yuan. High-frequency Normalizing Flow for Image Rescaling. *IEEE Transactions on Image Processing*, pages 1–1, 2022. Conference Name: IEEE Transactions on Image Processing. 21