Bachelor’s Programme in Science and Technology

Normalizing Flows for Graph Generation

Literature review

Hau Phan

Aalto University
BACHELOR’S THESIS 2022

Normalizing Flows for Graph Generation

Literature review

Hau Phan

Thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Technology.
Otaniemi, August 5, 2022

Supervisor:	Maarit Käpylä
Advisor:	Anirudh Jain

Aalto University
School of Science
Bachelor’s Programme in Science and Technology

Author
Hau Phan

Title
Normalizing Flows for Graph Generation

School School of Science	
Degree programme Bachelor's Programme in Science and Technology	
Major Data Science	**Code** IL3011
Supervisor Maarit Käpylä	
Advisor Anirudh Jain	

Level Bachelor's thesis **Date** August 5, 2022 **Pages** 30 **Language** English

Abstract

Lorem ipsum dolor sit amet, consequat adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Keywords graph neural network, normalizing flows, continuous normalizing flows, graphs generation, machine learning, geometric machine learning

urn https://aaltodoc.aalto.fi

Contents

Abstract **ii**

Contents **iii**

0.1 Motivation . . . 1

0.2 Thesis outline . . . 3

1. Background **4**

1.1 Notations . . . 4

1.2 Related works . . . 4

2. Normalizing Flows **5**

2.1 Background . . . 6

2.1.1 Formal Definition . . . 7

2.1.2 Universality . . . 8

2.1.3 General Properties . . . 8

2.2 Basic Flows . . . 9

2.2.1 Elementwise Flows . . . 9

2.2.2 Linear flows . . . 9

2.3 Coupling flows and Autoregressive flows . . . 12

2.3.1 Coupling flows . . . 12

2.3.2 Autoregressive flows . . . 14

2.3.3 Coupling functions . . . 15

2.4 Continuous flows . . . 18

2.4.1 Background . . . 18

2.4.2 Formulation . . . 19

3. Graph Neural Network **22**

3.1 Spectral methods . . . 23

3.2 Spatial methods . . . 23

3.2.1 Basic approaches . . . 23

3.2.2 Attention mechanism 23
3.2.3 Framework 23

4. Normalizing Flows for Graph Generation **24**

4.1 Dequantization 24
4.2 GraphNVP 24
4.3 GraphAF 24
4.4 MoFlow 24
4.5 GraphDF 24

5. Discussion **25**

5.1 Future work 25

6. Conclusion **26**

Bibliography **27**

Introduction

0.1 Motivation

Non-Euclidean data and graph

Deep learning (DL) has attracted significant attention in recent years due to its effectiveness in capturing latent representations of Euclidean data such as images, audio and natural languages. However, not all data is Euclidean and there is an increasing number of application of data represented as discrete structures such as graphs, sets and mappings that expresses abstract relations between objects. Particularly, graph is among the most fundamental structures in nature and can be observed everywhere across multiple domains. For example, molecules are essentially graphs of atoms bounded to one another by chemical bonds; users-products graphs are bipartite graphs that imply preferences of a customer towards specific sets of products; citation networks represents citationships between academic papers and inter/cross connections between scientific domains; road networks and traffics flows can also be considered as attributed spatial temporal graphs bestowed with directions [Bronstein et al., 2017]. Graphs are everywhere since they model connections and relationships, which are high-order abstraction and thus ubiquitous in nature. However, graphs are also arbitrarily complex mathematical objects: they can be irregular and have arbitrary numbers of nodes and vertices, where each node and edge might be number, vector or even smaller "child" graph; they are also invariant with respect to rotation, translation and permutation as there are no markers of direction or origin with graphs [Bronstein et al., 2017]. This is in direct contrast to Euclidean data (e.g. images) where the notion of directions and rotations is straightforward to define. The complexity that accompanies the expressive power of graphs has posed significant challenges to existing deep learning methods that were once successfully applied on Euclidean data. For example, the convolution operation of convolution neural network (CNN), which is simple to define for images but is nontrivial for graphs due for their being permutation invariant [Wu et al., 2021]. The domain of performing machine learning on graphs were summarized by the encompassing term *graph representation learning* which consists of various deep learning and non-deep-learning approaches to learn latent representations of graphs. Overall, devising new approaches to bridge the gap between deep learning on Euclidean and non-Euclidean data has been a popular topic recently among machine learning (ML) researchers.

Generative modeling

An interesting subdomain of machine learning is generative modeling which aims to generate new data/observation that is similar (in some abstract representation) to other samples seen in a given dataset. Most generative methods usually consist of an *inference* phase that learns a latent distribution of the data and a *sampling* phase that samples from it using various methods. Generative modeling in particular has been popularized to various modalities since the introduction of Generative Adversarial Network (GANs) [Goodfellow et al., 2014] and Variational Autoencoder (VAEs) [Kingma and Welling, 2014], where both had shown remarkable success in generating new images of faces, scenery and even audio [Hershey et al., 2017]. For graphs generation in particular, the majority of research efforts are directed at solving the molecular graph generation problem, which have high practical value in drug discovery [Wu et al., 2021]. Generally speaking, there are two primary approaches to generate new molecular graphs that are currently explored: *sequential generation* and *global generation*. Sequential approaches generate graphs by proposing nodes and edges step by step, while global approaches output new graphs all at once, which is also referred as *one-shot generation*. Pioneers in generating graph sequentially include graph generation using SMILE representation [Dai et al., 2018, Kusner et al., 2017, Gómez-Bombarelli et al., 2018], DeepGMG [Li et al., 2018] and GraphRNN [You et al., 2018]. For global approaches, existing deep learning architectures based on GANs and VAEs are adapted to graph domain by replacing CNN layers with Graph Neural Network (GNN) layers, albeit, with certain differences in architectural choices. Notable works from this line of research include MolGAN [De Cao and Kipf, 2018], GraphVAE [Simonovsky and Komodakis, 2018] and NetGAN [Bojchevski et al., 2018].

An exciting path of research in generative modeling is to adapt *normalizing flows* to graph domain, which has experienced a surge in popularity due to various desirable properties of flow-based models such as exact and tractable likelihood, efficient sampling and usable latent space for downstream tasks. Broadly speaking, the normalizing flows framework aims to learn a continuous invertible and deterministic mapping between the data distribution and the latent distribution, which is usually a Gaussian distribution. Sampling can be conducted by sampling from the latent distribution and pass through the reverse mapping to retrieve a sample from the data distribution. The mappings here are simply continuous invertible functions or compositions of them. Designing functions that are computational efficient to compose these mappings is the main focus when improving the expressivity of these model [Kobyzev et al., 2021]. Notable works in applying normalizing flows to graph include GraphNVP [Madhawa et al., 2019], GraphAF [Shi et al., 2020] and MoFlow [Zang and Wang, 2020]. A recent novel development in normalizing flows model are *continuous flows*, where rather than a discrete sequence of transformations that are chained together, the invertible mapping is modeled as solution to an ordinary differential equation (ODE). Continuous time-dependent mappings allow the data distribution to smoothly "morphed" into the latent distribution, albeit in practice, demand slightly more computation due to additional queries of black-box ODE solvers. One of the most notable works in the domain of continuous flow is FFJORD [Grathwohl et al., 2018] which is a direct followup improvement of the popular Neural ODEs (NODE) [Chen et al., 2019] that first formulated residual networks and normalizing flows as continuous differential processes. Practical applications of continuous flows, however, is a sparsely explored domain.

0.2 Thesis outline

This thesis aim to provide a systematic depth-wise exploration of normalizing flows models for graph generation, with focus on GNNs as the primary computational building blocks for flows and molecule generation as the targeted task. The thesis is divided into seven sections, starting with Section 1, where an overview of notations and related works on graph generation is introduced. Section 2 considers a walkthrough of the normalizing flows framework, including motivation, formulation, categorization of different flows accompanied by some remarks. Section 3 primarily focuses on topics of GNNs, explored under the *Message Passing Framework* [Gilmer et al., 2017], however, compressed and filtered in terms of relevance to graph normalizing flows. Lastly, Section 4 consists of different applications of normalizing flows on graph data, with a particular emphasis on molecule generation due to its high practical values and being extensively studied. As for closing remarks, Section 5 gives an open discussion on graph generative models along with personal thoughts on future works and research direction. Section 6 presents the final conclusion and the thesis's closing remarks.

1. Background

1.1 Notations

1.2 Related works

2. Normalizing Flows

One of the most straightforward to define but difficult to solve problem in statistic is modeling a *probability distribution* given samples drawn from it. Broadly speaking, a probability distribution is a mathematical object that assigns the chance of some event occurring to some number. Formally, a probability distribution is a *measure* μ on a topological *sample space* $(\mathcal{X}, \sigma, \mu)$ mapping that space to the interval $[0, 1]$. Regardless of the choice of definition, modeling a probability distribution can be reduced to searching some mappings or patterns on the data space (albeit, with additional constraints in the context of probability theory). This can be seen as an example of *unsupervised learning* or *generative modeling*, which had accumulated great attention among academics and the public alike. Its importance can be attributed to the abundance of unlabeled data compared to labeled in nature, where it remains an untapped source of information in training machine learning models. Moreover, learning patterns in data is a high-level general capability that can be utilized in practically every domain of science, from humanities such as psychology to STEM fields such as physics. Frequent applications of generative modeling include density estimation, outlier detection, prior construction and dataset summarization.

There have been various approaches in tackling this distribution modeling problem. Direct analytic approaches based on the assumption that observations was drawn from a fixed regular family of named distributions with parameters that can be inferred. However, this is an extremely optimistic and restricting assumption that might not be accurate for more complex multidimensional domain in nature such as images and sound. Variational inference and expectation maximization rely on latent variables to explain the data, but also introduce further complexity during learning and inference. With the boom of parallel computing architecture such as GPUs, novel deep learning approaches was now viable for learning massive datasets in generative modeling. Notable mentions are generative adversarial networks (GANs) and variational autoencoders (VAEs). While they had shown impressive results in image generation tasks, there are notable downsides of these black-box architectures. First is the inaccessible probability density for new observations, which make it unsuitable for density estimation tasks. Second is their notoriously difficult training and convergence when implemented. Adversarial approaches are based on training competing models that are uncertain to either converge to the true optimal or even converge at all. Mode collapse/posterior collapses are frequent occurrences while training these models, which happens when the model learn a simplified distribution that covers only some signature modes rather than a diverse set of them. Furthermore, vanishing gradients are pervasive in deep learning models, since fundamentally, neural models are composite functions: the longer the computation chain, the more magnified errors and digression of gradients at output.

Normalizing flow models is a novel family of generative models that are capable of learning irregular complex distribution with exact density evaluation, while also allowing efficient sampling. There have been numerous applications of normalizing flows and approches to improve on it being explored in the last decade, spanning multiple scientific domains. Notably mentions are noise modeling, audio generation, reinforcement learning, computer graphics and physics simulation. For this thesis, application of normalizing flow for molecular and graph generations are explored. Furthermore, for conciseness, we will only investigate normalizing flows approaches that is relevant to these applications in particular.

2.1 Background

A normalizing flow, as seen from a generation perspective, is a transformation of a simple probability distribution (e.g. normal distribution) into a more complex data distribution by a sequence of invertible and differentiable mappings. During generation, a sample is drawn from the simple distribution and undergo the deterministic transformation to yield an output. For density estimation, however, a slightly more complex procedure is performed: the observation is first transformed by an inverse mapping associated with the original transformation. Then the probability density of this observation (i.e. the likelihood) can be computed as the product of 1) the probability density of the inverse-transformed sample under the simple distribution 2) the associated change in volume induced by the sequence of inverse transformation. The change in volume is the product of the absolute values of the determinants of the Jacobians for each transformation, as required by the change of variables formula.

Normalizing flows hence can be seen as a systematic approach to model complex distribution by first choosing an initial (simple) distribution and chaining together a sequence of parameterized, invertible, differentiable transformations. Under this framework, the simple distribution is used as an interface for sampling, likelihood estimation, interpolation, ... which was otherwise impossible to be done directly on the complex data distribution.

2.1.1 Formal Definition

Let $\mathbf{Z} \in \mathbb{R}^D$ be a random variable following a known and tractable multivariate distribution with probability density function $p_{\mathbf{Z}} : \mathbb{R}^D \to [0,1]$. Similarly, let $\mathbf{Y} \in \mathbb{R}^D$ with $p_{\mathbf{Y}} : \mathbb{R}^D \to [0,1]$. Let also denote g an invertible function such that $\mathbf{Y} = \mathrm{g}(\mathbf{Z})$. Denote its inverse as $\mathrm{f} = \mathrm{g}^{-1}$. Then by the change of variables formula, we have:

$$p_{\mathbf{Y}}(\mathbf{y}) = p_{\mathbf{Z}}(\mathbf{f}(\mathbf{y})) \left| det \frac{\partial \mathbf{f}}{\partial \mathbf{y}} \right| \qquad \textit{Forward (Generation)}$$

$$p_{\mathbf{Z}}(\mathbf{z}) = p_{\mathbf{Y}}(\mathbf{g}(\mathbf{z})) \left| det \frac{\partial \mathbf{g}}{\partial \mathbf{z}} \right| \qquad \textit{Backward (Inference)}$$

Here $\frac{\partial \mathbf{f}}{\partial \mathbf{y}}$ and $\frac{\partial \mathbf{g}}{\partial \mathbf{z}}$ denotes the Jacobian of f and g respectively.

The above equations represent two directions of a normalizing flow: the *generative direction* where the base distribution $p_{\mathbf{Z}}$ (also referred as the 'noise') is *push forward* to a more complex probability distribution $p_{\mathbf{Y}}$ as indicated by the first equation; and the *normalizing direction* where the complex distribution $p_{\mathbf{Y}}$ is *normalized* or *simplified* to the base distribution $p_{\mathbf{Z}}$. The function g is aptly named the *generator* while the function f is referred as the *normalizer*. The terms normalizing flows is doubly correct if the base distribution is chosen to be the normal distribution, which is often the case in practice.

More formally, one can explain normalizing flow using the language of measure theory, characterizing distributions as generic measurable spaces and the functions g and f as diffeomorphisms between such spaces. Such characterization is necessary for theoretical study of generative models on non-Euclidean/discrete spaces and manifolds. This abstract perspective on normalizing flow resonate with the works on optimal transportation theory [Villani, 2003].

It is worth mentioning that usually in the literature, diffeomorphism are simply referred as "bijections" even though this is formally incorrect. In general, it is not necessary that the pushforward function g is differentiable everywhere, but rather differentiable almost everywhere with respect to the Lebesgue measure on $\mathbb{R}^D$. This allows for *piecewise differentiable functions* such as splines to be used in the construction, which will be explored later on.

2.1.2 Universality

Intuitively, given a complex enough generator g, one can transform any base distribution to a complex distribution under some reasonable assumptions. In fact, the existence of such transformation has been proven formally. A broad summary of the proof is that by modeling g as an elementwise cumulative density function (CDF) of the autoregressive conditional probabilities (assuming these conditional probabilities are differentiable), it can be shown that arbitrary complex distribution $p_{\mathbf{Y}}(\mathbf{y})$ can be transformed into a uniform distribution on the open unit cube $(0,1)^D$ and such transform is diffeomorphic by the monotonicity of CDF(s). The remaining part is to construct a diffeomorphism from $(0,1)^D$ to the base distribution $p_{\mathbf{Z}}$, which is trivial. Note that while such theoretical guarantee is useful, it might not translate in practical settings since expressivity might be limited by other architectural decisions in favor of efficiency.

2.1.3 General Properties

In summary, normalizing flows should meet several necessary requirements in order to be practical. Specifically, they should be:

1. **invertible**: for sampling and density estimation, the function g and its inverse $\mathrm{f} = \mathrm{g}^{-1}$ must exist. Differentiability might not be necessary depending on the type of normalizing flow (i.e. infinitestimal normalizing flow).

2. **expressive**: for modeling the complex multimodal data distribution.

3. **computationally efficient**: both the transformation g and its inverse f should be efficient to compute. However, depending on the application, one can tolerate longer computing time in favor of better sampling and density accuracy or vice versa. In both case, however, the calculation of the determinant of the Jacobian is required to be fast for training.

The following sections give a short overview of different types of normalizing flows with increasing expressivity (and complexity) that had been explored in the literature, accompanied by some remarks.

2.2 Basic Flows

2.2.1 Elementwise Flows

A multivariate transformation is invertible if it acts independently on each dimension and each elementwise transformation is also invertible. Formally, let $h_i : \mathbb{R} \to \mathbb{R}$ for $i = 1, \ldots, D$ be scalar-valued bijections. Then for $\mathbf{x} = (x_1, x_2, \ldots, x_D)^T$, we have:

$$\mathbf{g}(\mathbf{x}) = (h_1(x_1), h_2(x_2), \ldots, h_D(x_D))^T.$$

is also a bijection. The inverse $\mathbf{f} = \mathbf{g}^{-1}$ is rather straightforward:

$$\mathbf{f}(\mathbf{x}) = (h_1^{-1}(x_1), h_2^{-1}(x_2), \ldots, h_D^{-1}(x_D))^T.$$

and the Jacobian is the product of individual derivatives:

$$\frac{\partial \mathbf{g}}{\partial \mathbf{x}} = \prod_{i=1}^{D} \frac{dh_i}{dx_i}.$$

From the perspective of deep learning, elementwise flows is practically identical to activation functions, which also acts (mostly) elementwise. Note that the common non-linear activation function ReLU is not bijective and cannot be directly used as a non-linear mapping for normalizing flows. However, parametric leaky ReLU among other spline-based activation function are monotonic and thus and can be used instead.

2.2.2 Linear flows

Elementwise flows, as indicated by the constructions, does not possess any capability for modeling correlation between dimensions and thus have poor expressivity. While they are useful for introducing non-linearity, additional computation steps must be taken to incorporate dependencies between dimensions. One such approach is to use linear mappings that can express correlation between dimensions:

$$\mathbf{g}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}.$$

where $\mathbf{A} \in \mathbb{R}^{DxD}$ and $\mathbf{b} \in \mathbb{R}^D$ are learnable parameters. For the above transformation to be invertible, its required that $\mathbf{A}$ is an invertible matrix. If such condition is satisfied, then $\mathbf{f} = \mathbf{g}^{-1}$ can be straightforward deduced as:

$$\mathbf{f}(\mathbf{x}) = \mathbf{A}^{-1}(\mathbf{x} - \mathbf{b}).$$

Despite accounting for correlation between dimensions, due to the linear nature of the transformation, these flows still have limited expressivity. For example, if the base distribution is a Gaussian distribution (i.e. $p_{\mathbf{Z}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}, \mu, \Sigma)$), the resulting output distribution of the flow is also necessary Gaussian:

$$p_{\mathbf{Y}}(\mathbf{y}) = \mathcal{N}(\mathbf{y}, \mathbf{A}\mu + b, \mathbf{A}^T\Sigma, \mathbf{A}).$$

More generally, if the base distribution belongs to the exponential family then this property is preserved by linear flows and the resulting distribution also belongs to the exponential family. These limitations of linear flows resulted in them not being used separately on their own but as building blocks for higher-order flows such as coupling flows and autoregressive flows.

A consideration while learning the optimized parameterization of $\mathbf{A}$ is that invertibility is not guaranteed. Furthermore, even if we can ensure invertibility of $\mathbf{A}$ in each step of the optimization, it was formally proved that such continuous way of learning $\mathbf{A}$ will leave out certain invertible matrices. A simplified argument starts as follows: Consider two invertible matrices $\mathbf{W}_a$ and $\mathbf{W}_b$ such that $det(\mathbf{W}_a) > 0$ and $det(\mathbf{W}_b) < 0$. If there exists a continuous parameterization of all invertible matrices, then there exists a continuous path of transforming $\mathbf{W}_a$ to $\mathbf{W}_b$. Since the determinant is a continuous function of the matrix entries, such path will inevitably cross an invertible matrix with zero determinant, which is contradictory. This leads to a characterization of the space of invertible matrix as having two "islands" of positive and negative determinant matrices, which means continuous parameterization are restricted in one of these two islands only.

In the general case of arbitrary $\mathbf{A}$, the determinant of the Jacobian of linear flows is simply $det(\mathbf{A})$, which can be computed in $O(D^3)$. Its inverse $\mathbf{A}^{-1}$ also has a strict upperbound of $O(D^3)$ time to compute. Therefore, linear flows computation can be expensive for large D, which is common as natural data is usually high dimensional. However, by restricting the form of $\mathbf{A}$, these practical problems can be avoided partially but at the expense of expressivity of the model. The following subsections shall discuss some common approaches in alleviating this problem by limiting the form of $\mathbf{A}$ or reparameterizing to make linear flows more practical.

Diagonal and Triangular

The triangular matrix is a less expressive form of the full linear transformation matrix but with the desirable property of $O(D)$ determinant and $O(D^2)$ inversion time complexity. The determinant can be computed in linear time since its the product of its diagonal entries. Inversion is also inexpensive as a single back-substitution is sufficient, which required $O(D^2)$ operations. In the degenerate case, one can force all non-diagonal entries to be 0 and have a diagonal matrix instead. The time complexity of both operations is now $O(D)$ but the resulting flow is an elementwise flow and hence incapable of modeling correlation between dimensions. Nonetheless, an elementwise flow can be useful for representing normalization transformations, which had become ubiquitous in modern neural network architectures.

A generalized approach to improve the expressivity of triangular matrix is to consider a linear combination of them. This line of methods was first explored by [Tomczak and Welling, 2017], where K triangular matrices $\mathbf{T}_i$, each with ones on the diagonal and weighted by a K-dimensional learnable vector ω, linearly combined to form the general linear flow:

$$\mathbf{y} = \left(\sum_{i=1}^{K} \omega_i \mathbf{T}_i \right) \mathbf{z}.$$

This construction has the advantage of the determinant of the Jacobian being 1, but the cost of inverse is $O(D^3)$ due to the use of heterogeneous set of upper- and lower-triangular matrices.

Permutation

While triangular matrix is efficient to compute and have sufficient expressivity, it is subjected to inductive bias induced by the fixed ordering of the dimensions. One can mitigate this problem by permuting the dimensions with a permutation matrix, but at the downside that such permutation cannot be directly optimized via gradients-based algorithms. This reordering of dimensions is computationally inexpensive since the determinant of permutation matrices is 1 and the inverse is trivial to compute. Various strategies have been applied including fixing a random permutation and computing the reverse in advance [Dinh et al., 2017], [Kingma and Dhariwal, 2018]. However, there is no guarantee that the randomly-initialized permutation matrix is optimal and thus might lead to poor generative power in practice.

A more generalized alternative to permutation is orthogonal transformation induced by orthogonal matrices. The determinant and inverse can be computed efficiently since the formulas for them consists of basic matrix multiplications. Constructing a parameterizable orthogonal matrix for optimization has been done using *Householder transform* [Tomczak and Welling, 2017]. The formulation is grounded in the following mathematical fact from linear algebra: all orthogonal matrices can be expressed as a product of reflections. Hence, in turn, we can parameterize reflection matrices instead, which can be done straightforwardly by introducing a nonzero vector $\mathbf{v} \in \mathbb{R}^D$ and defines the reflection matrix as $\mathbf{1} - \frac{2}{||\mathbf{v}||^2}\mathbf{v}\mathbf{v}^T \in \mathbb{R}^D$

Factorization
A less limiting approaches to constructing linear flows is to reparameterize the matrix $\mathbf{A}$ through factorization. This was first proposed in the Glow model by [Kingma and Dhariwal, 2018], which used LU factorization to decompose $\mathbf{A}$:

$$\mathbf{g}(\mathbf{x}) = \mathbf{PLUx} + \mathbf{b}.$$

where $\mathbf{L}$ is lower triangular with ones on the diagonal, $\mathbf{U}$ is upper triangular with non-zero diagonal entries, and $\mathbf{P}$ is a permutation matrix. The determinant $det(\mathbf{A}) = det(\mathbf{PLU}) = det(\mathbf{U}) = \prod_{i=1}^{D} U_{ii}$ can be computed in $O(D)$ while the inverse requires $O(D^2)$ to calculate by backward substitution, which is efficient. However, this approaches faces the same problems of discrete permutation matrix that cannot be directly optimized and thus must be randomly-initialized and fixed during optimization. A solution proposed by [Hoogeboom et al., 2019] is to used QR decomposition in favor of LU factorization, where Q is formulated from Householder transforms.

2.3 Coupling flows and Autoregressive flows

2.3.1 Coupling flows

Formulation
Coupling flows are among the most widely used flow architectures currently explored in literature since its introduction by [Dinh et al., 2015]. One can attribute the flow's popularity to its extensibility and simplicity, which is evident in its general formulation:

Consider the disjoint partition of the input $\mathbf{x} \in \mathbb{R}^D$ and output $\mathbf{y} \in \mathbb{R}^D$:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}^A \\ \mathbf{x}^B \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} \mathbf{y}^A \\ \mathbf{y}^B \end{bmatrix}.$$

where $(\mathbf{x}^A, \mathbf{x}^B), (\mathbf{y}^A, \mathbf{y}^B) \in \mathbb{R}^d \times \mathbb{R}^{D-d}$ for some $d \in (0, D) \cap \mathbb{N}$:

Denote $\mathbf{h}(\cdot\,; \theta) : \mathbb{R}^d \to \mathbb{R}^d$ as some bijection parameterized by θ. Then consider the following generator function g applied partition-wise:

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} \mathbf{h}(\mathbf{x}^A\,; \Theta(\mathbf{x}^B)) \\ \mathbf{x}^B \end{bmatrix}.$$

where $\Theta(\mathbf{x}^B)$ only depends on $\mathbf{x}^B$. The function $\mathbf{h}$ is referred as the *coupling function* while the resulting generator function g is called the *coupling flow*. The function Θ is referred as the *conditioner*. Given $\mathbf{h}$ is a bijection with respect to $\mathbf{x}^A$, we have the inverse of the coupling flow g as:

$$\mathbf{g}^{-1}(\mathbf{y}) = \begin{bmatrix} \mathbf{h}^{-1}(\mathbf{y}^A\,; \Theta(\mathbf{y}^B)) \\ \mathbf{y}^{\mathbf{B}} \end{bmatrix}.$$

The Jacobian of g is a triangular block matrix where the diagonal blocks are $\frac{\partial \mathbf{h}}{\partial \mathbf{x}^A} \in \mathbb{R}^{d\times d}$ and the identity matrix $I \in \mathbb{R}^{(D-d)\times(D-d)}$ respectively:

$$\frac{\partial \mathbf{g}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{h}}{\partial \mathbf{x}^A} & \\ \mathbf{L}(\mathbf{x}) & \mathbf{I} \end{bmatrix}.$$

Here $\mathbf{L}(\mathbf{x})$ is some dense complex matrix that is bypassed in computation. The determinant of the Jacobian of the coupling flow is therefore $det\left(\frac{\partial \mathbf{h}}{\partial \mathbf{x}^A}\right)$. In fact in practice, most coupling functions are elementwise functions:

$$\mathbf{h}(\mathbf{x}^A; \theta) = (h_1(x_1^A; \theta_1), \ldots, h_d(x_d^A); \theta_d)).$$

where individual $h_i(\cdot\,; \theta_i) : \mathbb{R} \to \mathbb{R}$ for $i = 1 \ldots d$ is a scalar bijection. This reduces $\frac{\partial \mathbf{h}}{\partial \mathbf{x}^A}$ to be a diagonal matrix and thus the determinant of the Jacobian of g is just the product of the derivatives of individual scalar bijections $\prod_{i=1}^{d} \frac{\mathrm{d}h_i}{\mathrm{d}x_i}$.

The expressive power of coupling flows resides in the choice of the conditioner $\Theta(\mathbf{x}^B)$, which can be arbitrary complex. In practice, $\Theta(\mathbf{x}^B)$ is realized by some neural network architecture depending on the domain. For instance, for graph generation tasks, Θ is chosen to be a graph neural networks [Zang and Wang, 2020] while for images generation, a shallow ResNet can be used [Kingma and Dhariwal, 2018].

Multi-scale flows

Conditioner Θ can also be independent of $\mathbf{x}^B$ and simplified to be constant. This is the premise of *multiscale flow* introduced by RealNVP [Dinh et al., 2017]. The idea is to construct the generator $\mathbf{g}$ as a composition of coupling flows that transform an increasingly large partition of the input while keep other dimensions unchanged. In the normalizing direction, the number of dimensions involved in computing the inverted flow decreases exponentially in each step, which greatly reduces computational costs of transforming the full high dimensional data distribution during inference. [Kruse et al., 2021] further improved upon the idea of multiscale flows by introducing recursive hierarchical coupling flows that keep the expressive form $\Theta(\mathbf{x_B})$ while preserve necessary parallelizability for fast sampling and inference. The multiscale composite nature of this flow also allow the capturing of high-order structure that exhibits certain types of natural data such as images.

2.3.2 Autoregressive flows

Autoregressive flows can be considered as a non-linear generalization of triangular linear flows with high capacity to model complex correlations between dimensions. First introduced by [Kingma et al., 2016], autogressive flow has become a popular choice due to its high expressivity and sampling quality but at the cost of computation time due to the sequential nature of autoregressive models. Nonetheless, it is an important basis for developing more expressive flows models with theoretical guarantee for universality (i.e. the induced transformation can represent arbitrary complex mapping between the distributions under some reasonable assumptions). In fact the proof referred in Section 2.1.2 stemmed from the choice of the transformation being autoregressive.

Formulation

Autoregressive flows can be considered as the extreme case of coupling flows where the dimensions are completely partitioned into individual terms $x_1, x_2, \ldots, x_d \in \mathbb{R}$. Consider the *coupling function* which is now a scalar bijection $h(\cdot\,;\theta) : \mathbb{R} \to \mathbb{R}$. Then the autoregressive generator $\mathbf{g} : \mathbb{R}^D \to \mathbb{R}^D$ models each entry of the output $\mathbf{y} = \mathbf{g}(x)$ as:

$$y_t = h(x_t, \Theta_t(\mathbf{x}_{<t})).$$

where Θ_t are also called *conditioners* which map $\mathbb{R}^{t-1}$ to parameter space of h. Note that in some literature, h is referred as the *transformer*, which is unrelated to the Transformer architecture that utilizes the attention mechanism.

The expressivity of autoregressive flow can again be attributed by the unboundedness in architectural complexity of Θ_t, which can be arbitrary nonlinear functions. In practice, Θ_t are parameterized by a single model and the evaluation of g can be efficiently computed in a single pass through given appropriate masking of learnable parameters. This is the main idea behind Masked Autoregressive Flow (MAF) proposed by [Papamakarios et al., 2017] in an attempt to make autoregressive flows parallelizable in modern computing hardware.

The Jacobian of **g** is a dense (lower) triangular matrix and hence can be efficiently computed as the product of its diagonal entries:

$$det\left(\frac{\partial \mathbf{g}}{\partial \mathbf{x}}\right) = \prod_{t=1}^{D} \frac{dy_t}{dx_t}.$$

However, while the generation direction can be computed in parallel with masked autoregressive flow, the inverse can only be computed sequentially due to the dependence of x_t on $\mathbf{x}_{<t}$:

$$x_t = h^{-1}(y_t, \Theta_t(\mathbf{x}_{<t})).$$

Alternatively, the autoregressive dependence can be modified such that Θ_t take $\mathbf{y}_{<t}$ as inputs which results in its inverse being a direct autoregressive flow:

$$y_t = h(x_t, \Theta_t(\mathbf{y}_{<t})).$$

$$x_t = h^{-1}(y_t, \Theta_t(\mathbf{y}_{<t})).$$

This formulation of autoregressive flows induce the opposite trade-off: relative efficient sampling at the expense of unparralelizable density estimation and is the basis of Inverse Autoregressive Flows (IAF) [Kingma et al., 2016]. Thus, depending on the intended applications, one might prefer the use of IAF for fast sampling while MAF for efficient density estimation. Regardless of the decision, both methods are based on the same principles and can be shown to be theoretical equivalent.

Universality

Autoregressive flows can be shown to be universal, i.e. capabable of modeling complex target distribution to arbitrary precision given sufficient data and capacity [Jaini et al., 2019] [Huang et al., 2018]. As a corollary, an autoregressive flow will possess such universality property if it can be demonstrated that its coupling function h is monotonic or the family of such coupling flows are dense on the space of monotonic functions in pointwise convergence topology.

2.3.3 Coupling functions

The coupling function $\mathbf{h}(\cdot\,;\theta)$ is the primary building blocks in constructing both coupling flows and autoregressive flows. While they are defined as multivariate functions in coupling flows, they are usually formulated as elementwise bijections for analytic tractability in practice. Furthermore, in autoregressive flows, coupling functions are also scalar functions and thus, we will explore different choice of coupling functions on the premise of them being simple scalar bijections $h : \mathbb{R} \rightarrow \mathbb{R}$.

Affine couplings

NICE (nonlinear independent component estimation) [Dinh et al., 2015] contained some of the earliest proposals of coupling functions, which are *additive coupling function*:

$$h(x\,;\theta) = x + \theta \quad \theta \in \mathbb{R}.$$

and *affine coupling function*:

$$h(x\,;\theta) = \theta_1 x + \theta_2 \quad \theta_1 \neq 0, \theta_2 \in \mathbb{R}.$$

The efficient computation and simplicity leads affine couplings to be widely used in flows literature. However, they have limited expressivity and thus many consecutive flows need to be composed to represent complicated distribution. When designing $\mathbf{h}(\cdot\,;\theta)$, as alluded in Section 2.1.2, one might ensure its monotonicity for theoretical guarantee of universality. Therefore, it is frequent for *affine coupling flows* to be formulated such that $\theta_1 > 0$, which results in the following alternative forms for coupling and autoregressive flows respectively:

$$\mathbf{h}(\mathbf{x}_B\,;\Theta, \mathbf{x}_A) = \mathbf{x}_B \odot e^{\mathrm{s}(\mathbf{x}_A\,;\Theta_1)} + \mathbf{t}(\mathbf{x}_A\,;\Theta_2).$$

$$h(x_t\,;\Theta, \mathbf{x}_{<t}) = x_t \cdot e^{s(\mathbf{x}_{<t}\,;\Theta_1)} + t(\mathbf{x}_{<t}\,;\Theta_2).$$

where $\Theta = (\Theta_1, \Theta_2)$ is a partitioned parameter of the coupling function and $\mathrm{s}, \mathbf{t} : \mathbb{R}^d \to \mathbb{R}^{D-d}$ (or $s, t : \mathbb{R}^d \to \mathbb{R}$ in elementwise form) are arbitrary functions which are usually parameterized by neural networks with weights Θ_1 and Θ_2. Note that s and t are sometimes called the **scale function** and **transformation function**. The inverse of h in coupling flows is efficient as $\mathbf{y}_A = \mathbf{x}_B$:

$$\begin{aligned}\mathbf{h}^{-1}(\mathbf{y}_B\,;\Theta, \mathbf{x}_A) &= (\mathbf{y}_B - \mathbf{t}(\mathbf{x}_A\,;\Theta_2)) \odot e^{-\mathrm{s}(\mathbf{x}_A\,;\Theta_1)} \\ &= (\mathbf{y}_B - \mathbf{t}(\mathbf{y}_A\,;\Theta_2)) \odot e^{-\mathrm{s}(\mathbf{y}_A\,;\Theta_1)}\end{aligned}$$

$$h^{-1}(y_t, \Theta, \mathbf{x}_{<t}) = (y_t - t(\mathbf{x}_{<t}\,;\Theta_2)) \cdot e^{-s(\mathbf{x}_t;\Theta_1)}.$$

However, for autoregressive flows, the sequential dependence hinder much parallel optimization. Representative works of affine coupling functions for coupling flows includes NICE [Dinh et al., 2015], RealNVP [Dinh et al., 2017], Glow [Kingma and Dhariwal, 2018] and for autoregressive flows IAF [Kingma et al., 2016] and MAF [Papamakarios et al., 2017].

Nonlinear couplings

There have been many research efforts in constructing non-linear complex couplings that capable of more expressivity and modeling power. For example, [Ziegler and Rush, 2019] proposed the ideas of nonlinear squared flow:

$$h(x;\theta) = ax + b + \frac{c}{1 + (dx + h)^2}.$$

parameterized by $\theta = (a, b, c, d, h) \in \mathbb{R}^5$. The invertibility of h is ensured as the resulting analytical computation contains solving a cubic polynomial with only one real root. Empirically, this type of flows tends to improve learning of multimodal data distributions.

Flow++ [Ho et al., 2019] further improve upon the idea of affine couplings by introducing an additional invertible transformation based on CDFs of K logistics mixture:

$$h(x;\theta) = \theta_1 F(x, \theta_3) + \theta_2.$$

$$F(x, \boldsymbol{\pi}, \boldsymbol{\mu}, \mathrm{s}) = \sigma^{-1}\left(\sum_{j=1}^{K} \pi_j \sigma\left(\frac{x - \mu_j}{s_j}\right)\right).$$

The inverse of such flows does not possess an analytical form but can be computed numerically via fixed-point iteration (e.g. bisection algorithm). The upside to using monotonic mixture of such CDFs is the derivatives are logistic PDFs with inexpensive evaluation. Experimental ablation study showed that such modification improve performance slightly.

Another approach to constructing monotonic nonlinear coupling functions is to use *splines* of various degree. A spline is a piecewise polynomial function defined by a set of points called *knots* where the spline must exactly cross. Denote this set of $K + 1$ knots as $(x_i, y_i)_{i=0}^{K}$. For a spline to be monotonic, which we will further assume to be increasing (without the lost of generality), the set of knots must have $x_i < x_{i+1}$, $y_i < y_{i+1}$ for $i = 0, \ldots, K - 1$. Interpolating between two or more consecutive knots with monotonic polynomials of degree P results in a valid monotonic coupling function. For $P = 1, 2$ we have linear and quadratic monotone splines [Müller et al., 2019]. For $P = 3$ we have cubic monotone splines, usually seeded with the derivatives at the two end points and constructed via the Steffen's method [Durkan et al., 2019]. Note that splines are usually defined on compact intervals such as $[0, 1]$ and thus it is common that $h(x\,;\theta) = \sigma(\hat{h}(x\,;\theta)), \hat{h} : [0, 1] \to [0, 1]$ where $\hat{h}$ is a surrogate spline and σ is the sigmoid function to ensure the right range of h.

Intuitively, it seems desirable to model h as a neural network due to the universal approximation theorem. [Huang et al., 2018] follows this approach to devise

Neural Autoregressive Flow (NAF) where $h(\cdot;\theta)$ is parameterized by a multi layer perceptron (MLP). Indeed, the resulting autoregressive flows possess universality property. In general, MLP are not invertible but under sufficient conditions on its weights, it can be shown that the learned MLP is monotonic and thus reversible:

Proposition 1 *Let $\mathcal{NN}(\cdot) : \mathbb{R} \to \mathbb{R}$ be a MLP such that all weights are positive, and all activation functions are strictly monotone, then $\mathcal{NN}(\cdot)$ is a strictly monotone function.*

Empirically, however, such restrictions greatly limit expressivity of full unconstrained MLP and thus require more complex conditioner for the same modeling performance. [Wehenkel and Louppe, 2019] proposed a solution to such problem by relax the requirements to strictly-positive-output MLPs and integrate numerically to produce monotonic increasing functions.

[Jaini et al., 2019] suspected h is overparameterized if modeled as MLPs and thus proposed a simplier coupling function computed by summing squared polynomials and then integrate:

$$h(x\,;\theta) = \int_0^x \sum_{k=1}^{K} \left(\sum_{l=0}^{L} a_{kl} u^l \right)^2 du.$$

The resulting sum-of-square polynomial flows are named SOS [Jaini et al., 2019] and experimentally are easier to train compared to NAF due to the unconstrained nature of parameters a_{kl}. Note that K and L are hyperparameters that can be chosen beforehand to tune expressivity at the risk of over/under-fitting. When $L = 0$, SOS reduces to *affine coupling functions* and can be considered as a generalization of affine coupling flow. It was also proven by [Jaini et al., 2019] that sum-of-squares polynomial flow is a universal flow and can transform the base distribution to arbitrarily complex distribution.

2.4 Continuous flows

2.4.1 Background

Continuous flows can be considered as a continuous-time generalization of residual flows, which in turn, inspired by the idea of residual networks [He et al., 2016]. Such networks are composed of functions of the form:

$$\mathbf{g}(\mathbf{x}) = \mathbf{x} + \mathbf{F}(\mathbf{x}) \tag{2.1}$$

Here $\mathrm{g}(\mathbf{x})$ are called a *residual connection* and $\mathbf{F}(\mathbf{x})$ is usually a neural network. Specifically, the original ResNet [He et al., 2016] parameterized $\mathbf{F}(\mathbf{x})$ as blocks of CNNs followed by a batch normalization layer. Flow networks that utilize composition of such residual connections are usually referred as *residual flows*.

Residual flow is essentially a discrete version of the more general so-called *continuous* or *infinitestimal* flows [Chen et al., 2019], which model the process of transformation as a continuous time dependent function rather than a discrete composition of residual connections g. Such function is the solution to the initial-value problem of an ordinary differential equation (ODE):

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{F}(\mathbf{x}(t), \theta(t)) \tag{2.2}$$

Propagating through such networks requires solving an ODE via a black-box solver to find the evaluation of the function at a predefined endpoint in time. This approach to model continuous infinite flows (or infinite depth neural networks) has spawn a novel branch of research in normalizing flows with potential higher modeling capacity and faster inference.

2.4.2 Formulation

Consider the initial value problem for the ODE given in Equation 2.2. In the generative direction, let $t \in [0, 1]$ and initial condition $\mathbf{x}(0) = \mathbf{z}$. Assuming Lipschitz continuity of $\mathbf{F}$ in $\mathbf{x}$ and continuity in t, then the solution exists and is unique by Picard-Lindelof-Lipschitz-Cauchy theorem [Arnold, 1992], which in turn, shown that the solution set consists of invertible functions. Now denote the solution at each time t as $\Phi^t(\mathbf{z})$. Define the solution at $t = 1$ as the generator (i.e. $\Phi^1(\mathbf{z}) = \mathrm{g}(\mathbf{z}) = \mathbf{y}$). Intuitively, this give ways to model $\mathrm{g}(\mathbf{z})$ as a dynamic process through time t with output as the process's state at $t = 1$. Modeling the generator as $\Phi^1(\mathbf{z})$ (also called the *time-one map*) and solving it via black-box solvers was the idea behind NODE (Neural ODE) [Chen et al., 2019].

Mathematically, the solutions $\Phi^t(\cdot) : \mathbb{R}^D \to \mathbb{R}^D$ form a one-parameter group of diffeomorphism $(\mathcal{P}, \circ)$, where $\circ$ is the composition operator (i.e. $\Phi^t \circ \Phi^s = \Phi^{t+s} \in \mathcal{P}, t + s \leq 1$). Such a group is referred as a smooth flow in dynamical systems theory and differential geometry [Katok and Hasselblatt, 1997].

From the context of neural network, this can be considered as an "infinite depth" (residual) neural network with input $\mathbf{z}$ and output $\mathbf{y}$, parameterized by continuous time-dependent weights $\theta(t)$.

The primary concerns regarding this method of continuous transformation is how to propagate the gradients needed for optimization on supervised tasks.

Since gradients can not be trivially propagated through the computation of ODE solvers, the *adjoin sensitivity method* is used as a replacement. Logically, the method can be considered as a continuous analog of the backpropagation: if the gradient at the n-th hidden layer in a neural network is $\frac{dL}{d\mathbf{h}_n} = \frac{dL}{d\mathbf{h}_{n+1}} \frac{d\mathbf{h}_{n+1}}{d\mathbf{h}_n}$ then the algebraic-continuation time-dependent gradient $\mathbf{a}(t)$ must satisfy the augmented ODE:

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)\frac{dF(\mathbf{x}(t), \theta(t))}{\mathbf{x}(t)}.$$

where $\mathbf{a}(t) = \frac{dL}{d\mathbf{x}(t)}$ is also called the *adjoint* or *sensitivity*.

For density estimation however, we do not have a loss but instead aim to maximize the log likelihood. The continuous log-likelihood at each time step t is now also given as solution to another ODE analogous to the change of variable formula:

$$\frac{d}{dt}\log(p(\mathbf{x}(t))) = -\mathbf{Tr}\left(\frac{dF(\mathbf{x}(t))}{\mathbf{x}(t)}\right).$$

The log likelihood of a data point $\mathbf{y}$ is computed by first solve the reverse ODE to yield the corresponding $\mathbf{z}$ and then solve the "change of variable" ODE above at $t = 1$ (i.e. $p_{\mathbf{Y}}(\mathbf{y}) = p(\mathbf{x}(1))$), given the initial-value $p(\mathbf{x}(0)) = p_{\mathbf{Z}}(\mathbf{z})$. Note that the determinant is no longer needed in the formula and thus can be theoretically less computationally expensive.

[Grathwohl et al., 2018] further compliment NODE with the following works of FFJORD, which improved upon the existing continuous change of variable formula by introducing Hutchinson estimator to approximate the trace term for faster evaluation. [Finlay et al., 2020] introduced two regularization terms to the loss function of FFJORD, which was the Frobenius norm of the Jacobian $det\left(\frac{dF(\mathbf{x}(t))}{\mathbf{x}(t)}\right)$ and a forcing term that limits the solution trajectories to follow straight paths with zero acceleration. A promising finding of using continuous flows in comparison to discrete compositive flows is the efficient use of parameters. A comparison experiment suggested that on the CIFAR10 dataset, for comparable performance, FFJORD requires less than 2% as many parameters as Glow.

However, there certain limitations with continuous flows that have been recognized, specifically focused on the corresponding Jacobian of such flows. Since the defined ODE traces a continuous path in the space of diffeomorphism (which consists of functions with nonzero Jacobian determinant) that starts from the identity solution at time-zero i.e. $\Phi^0(\mathbf{z}) = I\mathbf{z}$ to the time-one map i.e. $\Phi^1(\mathbf{z}) = \mathbf{y}$, it is necessary that $det\left(\frac{d\Phi^t(\mathbf{z})}{d\mathbf{z}}\right) \forall t \in [0, 1]$ share the same sign. Since $det\left(\frac{d\Phi^0(\mathbf{z})}{d\mathbf{z}}\right) = det(I) > 0$, it is necessary that at all time step t, the Jacobian of the solution $\Phi^t(\mathbf{z})$ is positive or in order words, *orientation preserving*. This is a subspace of the space of all possible transformations with positive and negative Jacobian and thus can be considered a limitation in expressivity of NODEs. A novel solution to this problem is to consider an *augmented* version of NODE (ANODE) that can represent a more inclusive class of diffeomorphisms by adding "slack" variables $\hat{\mathbf{x}}(t) \in \mathbb{R}^p$ and consider the *augmented* ODE [Dupont et al., 2019]:

$$\frac{d}{dt}\begin{bmatrix}\mathbf{x}(t)\\ \hat{\mathbf{x}}(t)\end{bmatrix} = \hat{\mathbf{F}}\left(\begin{bmatrix}\mathbf{x}(t)\\ \hat{\mathbf{x}}(t)\end{bmatrix}, \theta(t)\right).$$

with initial conditions $[\mathbf{x}(0), \hat{\mathbf{x}}(0)]^T = \mathbf{0}$. [Dupont et al., 2019] suspected that such addition of a slack variable allows for the Jacobian of the solutions to wander more freely to the negative domain, permitting the learning of distributions that previous NODE incapable of represent. This hypothesis seemed to be reflected in the papers' quantitative experiments with an additional surprise of shorter training time. In fact, it has been formally proven by [Zhang et al., 2020a] that the space of all possible transformations is covered by ANODE and thus the resulting continuous flows are indeed universal.

3. Graph Neural Network

Graph neural networks (GNNs) in their most general definition can be considered as composite deep-learning-inspired methods that operate on graphs. There have been multiple surveys published in effort of standardizing and categorizing graph neural networks. [Zhang et al., 2020b] [Zhang et al., 2020b] [Chami et al., 2022] and [Gilmer et al., 2017] are notable mentions. For convenience and consistency, this thesis began the journey of categorizing graph neural networks by first looking through the lens of deep learning architectural design. Under this view, GNNs are computing pipelines, composed of *computational modules* that fall into three main categories: propagation modules, sampling modules and pooling modules. Propagation modules can be considered as the central information aggregators that through the process of propagating information between nodes, are capable of capturing feature and topological information. This notion of information propagation between vertices is also referred as *message passing* [Gilmer et al., 2017].

#WIP

There are two main operators used in vertical (or depthwise) propagation: convolution operators and recurrent operators, which can be generalized to an encompassing superset of convolutional graph neural network (ConvGNNs) and recurrent graph neural networks (RecGNNs) [Wu et al., 2021]. The central idea of convolution operators is to generalize convolutions from other domains to graph domain. Convolutions abstractly can be represented as layers with different parameterized filters that convolve with the information passed in as inputs and output a latent representation. The main distinction between recurrent approaches and convolution approaches is the reusing of parameters during information propagation. While convolution use separate weights for each operating layers, recurrent approaches recycle the weights and autogressively pass the output as the next input. Consequently, the notion of layer depth are mostly applied to convolution methods and not recurrent ones.

In an effort to scope the evolving field of GNN, only convolutional graph neural networks are explored in detail in this chapter, specifically under the categorization laid by Message Passing Neural Network (MPNN) [Gilmer et al., 2017].

3.1 Spectral methods

3.2 Spatial methods

3.2.1 Basic approaches

3.2.2 Attention mechanism

3.2.3 Framework

4. Normalizing Flows for Graph Generation

4.1 Dequantization

4.2 GraphNVP

4.3 GraphAF

4.4 MoFlow

4.5 GraphDF

5. Discussion

5.1 Future work

6. Conclusion

Bibliography

[Arnold, 1992] Arnold, V. I. (1992). *Ordinary Differential Equations*. Springer Science & Business Media.

[Bojchevski et al., 2018] Bojchevski, A., Shchur, O., Zügner, D., and Günnemann, S. (2018). NetGAN: Generating Graphs via Random Walks.

[Bronstein et al., 2017] Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. (2017). Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42.

[Chami et al., 2022] Chami, I., Abu-El-Haija, S., Perozzi, B., Ré, C., and Murphy, K. (2022). Machine Learning on Graphs: A Model and Comprehensive Taxonomy.

[Chen et al., 2019] Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. (2019). Neural Ordinary Differential Equations.

[Dai et al., 2018] Dai, H., Tian, Y., Dai, B., Skiena, S., and Song, L. (2018). Syntax-directed variational autoencoder for molecule generation. In *Proceedings of the International Conference on Learning Representations*.

[De Cao and Kipf, 2018] De Cao, N. and Kipf, T. (2018). MolGAN: An implicit generative model for small molecular graphs.

[Dinh et al., 2015] Dinh, L., Krueger, D., and Bengio, Y. (2015). NICE: Non-linear Independent Components Estimation.

[Dinh et al., 2017] Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2017). Density estimation using Real NVP.

[Dupont et al., 2019] Dupont, E., Doucet, A., and Teh, Y. W. (2019). Augmented Neural ODEs.

[Durkan et al., 2019] Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. (2019). Cubic-Spline Flows.

[Finlay et al., 2020] Finlay, C., Jacobsen, J.-H., Nurbekyan, L., and Oberman, A. M. (2020). How to train your neural ODE: The world of Jacobian and kinetic regularization.

[Gilmer et al., 2017] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1263–1272. PMLR.

[Gómez-Bombarelli et al., 2018] Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. (2018). Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules. *ACS Central Science*, 4(2):268–276.

[Goodfellow et al., 2014] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Networks.

[Grathwohl et al., 2018] Grathwohl, W., Chen, R. T. Q., Bettencourt, J., Sutskever, I., and Duvenaud, D. (2018). FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

[Hershey et al., 2017] Hershey, S., Chaudhuri, S., Ellis, D. P. W., Gemmeke, J. F., Jansen, A., Moore, R. C., Plakal, M., Platt, D., Saurous, R. A., Seybold, B., Slaney, M., Weiss, R. J., and Wilson, K. (2017). CNN architectures for large-scale audio classification. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 131–135.

[Ho et al., 2019] Ho, J., Chen, X., Srinivas, A., Duan, Y., and Abbeel, P. (2019). Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design.

[Hoogeboom et al., 2019] Hoogeboom, E., Berg, R. V. D., and Welling, M. (2019). Emerging Convolutions for Generative Normalizing Flows. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2771–2780. PMLR.

[Huang et al., 2018] Huang, C.-W., Krueger, D., Lacoste, A., and Courville, A. (2018). Neural Autoregressive Flows. In *Proceedings of the 35th International Conference on Machine Learning*, pages 2078–2087. PMLR.

[Jaini et al., 2019] Jaini, P., Selby, K. A., and Yu, Y. (2019). Sum-of-Squares Polynomial Flow.

[Katok and Hasselblatt, 1997] Katok, A. and Hasselblatt, B. (1997). *Introduction to the Modern Theory of Dynamical Systems*. Cambridge University Press.

[Kingma and Dhariwal, 2018] Kingma, D. P. and Dhariwal, P. (2018). Glow: Generative Flow with Invertible 1x1 Convolutions. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

[Kingma et al., 2016] Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016). Improved Variational Inference with Inverse Autoregressive Flow. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

[Kingma and Welling, 2014] Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes.

[Kobyzev et al., 2021] Kobyzev, I., Prince, S. J. D., and Brubaker, M. A. (2021). Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3964–3979.

[Kruse et al., 2021] Kruse, J., Detommaso, G., Köthe, U., and Scheichl, R. (2021). HINT: Hierarchical Invertible Neural Transport for Density Estimation and Bayesian Inference.

[Kusner et al., 2017] Kusner, M. J., Paige, B., and Hernández-Lobato, J. M. (2017). Grammar Variational Autoencoder. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1945–1954. PMLR.

[Li et al., 2018] Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. (2018). Learning Deep Generative Models of Graphs.

[Madhawa et al., 2019] Madhawa, K., Ishiguro, K., Nakago, K., and Abe, M. (2019). GraphNVP: An Invertible Flow Model for Generating Molecular Graphs.

[Müller et al., 2019] Müller, T., McWilliams, B., Rousselle, F., Gross, M., and Novák, J. (2019). Neural Importance Sampling.

[Papamakarios et al., 2017] Papamakarios, G., Pavlakou, T., and Murray, I. (2017). Masked Autoregressive Flow for Density Estimation. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[Shi et al., 2020] Shi, C., Xu, M., Zhu, Z., Zhang, W., Zhang, M., and Tang, J. (2020). GraphAF: A Flow-based Autoregressive Model for Molecular Graph Generation.

[Simonovsky and Komodakis, 2018] Simonovsky, M. and Komodakis, N. (2018). GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders. In Kůrková, V., Manolopoulos, Y., Hammer, B., Iliadis, L., and Maglogiannis, I., editors, *Artificial Neural Networks and Machine Learning – ICANN 2018*, Lecture Notes in Computer Science, pages 412–422, Cham. Springer International Publishing.

[Tomczak and Welling, 2017] Tomczak, J. M. and Welling, M. (2017). Improving Variational Auto-Encoders using convex combination linear Inverse Autoregressive Flow.

[Villani, 2003] Villani, C. (2003). *Topics in Optimal Transportation.* American Mathematical Soc.

[Wehenkel and Louppe, 2019] Wehenkel, A. and Louppe, G. (2019). Unconstrained Monotonic Neural Networks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[Wu et al., 2021] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. (2021). A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24.

[You et al., 2018] You, J., Ying, R., Ren, X., Hamilton, W., and Leskovec, J. (2018). GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models. In *Proceedings of the 35th International Conference on Machine Learning*, pages 5708–5717. PMLR.

[Zang and Wang, 2020] Zang, C. and Wang, F. (2020). MoFlow: An Invertible Flow Model for Generating Molecular Graphs. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 617–626.

[Zhang et al., 2020a] Zhang, H., Gao, X., Unterman, J., and Arodz, T. (2020a). Approximation Capabilities of Neural ODEs and Invertible Residual Networks.

[Zhang et al., 2020b] Zhang, Z., Cui, P., and Zhu, W. (2020b). Deep Learning on Graphs: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 34(1):249–270.

[Ziegler and Rush, 2019] Ziegler, Z. and Rush, A. (2019). Latent Normalizing Flows for Discrete Sequences. In *Proceedings of the 36th International Conference on Machine Learning*, pages 7673–7682. PMLR.