Geometric Extensions of Neural Processes

Andrew Newberry Carr
Brigham Young University

Geometric Extentions of Neural Processes

Andrew Newberry Carr

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

ABSTRACT

Geometric Extentions of Neural Processes

Andrew Newberry Carr
Department of Computer Science, BYU
Master of Science

Neural Processes (NPs) [19] are a class of regression models that learn a map from a set of input-output pairs to a distribution over functions. NPs are computationally tractable and provide a number of benefits over traditional nonlinear regression models. Despite these benefits, there are two main domains where NPs fail. This thesis is focused on presenting extensions of the Neural Process to these two areas.

The first of these is the extension of Neural Processes graph and network data which we call Graph Neural Processes (GNP). A Graph Neural Process is defined as a Neural Process that operates on graph data. It takes spectral information from the graph Laplacian as inputs and then outputs a distribution over values. We demonstrate Graph Neural Processes in edge value imputation and discuss benefits and draw backs of the method for other application areas.

The second extension of Neural Processes comes in the fundamental training mechanism. NPs are traditionally trained using maximum likelihood, a probabilistic technique. We show that there are desirable classes of problems where NPs fail to learn. We also show that this drawback is solved by using approximations of the Wasserstein distance. We give experimental justification for our method and demonstrate its performance. These Wasserstein Neural Processes (WNPs) maintain the benefits of traditional NPs while being able to approximate new classes of function mappings.

Keywords: optimal transport, machine learning, graph Laplacian, neural networks

ACKNOWLEDGMENTS

I would first like to acknowledge by darling wife AnnaLisa. She pushed me by showing me what a real Master's Thesis should look like. I am also grateful to my family for being willing to listen to me geek out about my research and believing in me, even if I didn't always believe in myself.

I also am immensely grateful for David Wingate. He helped me start my journey into ML in 2016. His guidance, mentorship, technical ability, and enthusiasm were pivotal.

I also want to acknowledge the Applied and Computational Math program at BYU. The creators, professors, students, and TAs pushed and stretched me in uncomfortable, but amazing ways. I am forever grateful for the skills and critical thinking abilities I gained in that program.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The main problems tackled in this thesis are regression problems where a model is trained to predict real valued outputs. In traditional linear regression, a set of parameters is fit to a series of input points, and these parameters can then be used to infer values for some desired set of target points. The regression in this work is similar, but typically higher dimensional with a more complex domain. The methods we utilize take some small subset of input points and predict a distribution over output values. This distribution over output values can be used to derive confidence intervals, error bars, and other descriptive statistics about model uncertainty.

A popular class of non-linear regression models, Gaussian Processes (GP), are used to learn an output distribution over functions by fitting a set of input data. By learning a distribution over functions, these well studied models are valuable when modeling uncertainty. However, GPs are $O(n^3)$ in the number of data points and therefore scale poorly to problems with large datasets.

In 2018 a new class of architectures was discovered [18, 19, 34] that generalized Gaussian Processes using neural methods. The new class of architectures, broadly referred to as Neural Processes (NP), overcome this scalability problem and are $O(n+m)$ where n represents the size of input points and m the size of the target points. Intuitively, NPs take a small set of input data (called context points) and build a compressed representation of that data. This representation is used, with a larger set of target points, to perform regression. NPs have been applied to regression problems on tabular data, image completion, and image generation. As mentioned, these models have a number of benefits over traditional GPs, including scalability and computational complexity. However, despite the success of NPs there are two cases where both NPs and GPs have poor performance.

First, NPs fail when the data is not Euclidean, for example graph and network data. These data are ubiquitous in modern data analysis, and are often encountered in biological systems, social networks, or natural language processing. The ability to perform regression on graphs is valuable for a number of problems. NPs cannot model this type of relational data because they were designed to work purely on data in R^d and there is a "type" error when trying to operate on non-Euclidean data.

Second, NPs can fail when the underlying data generating distribution has an unknown or intractable likelihood, or when the model's construction leads to a misspecified likelihood. This occurs in many image processing problems and other contexts where the exact likelihood of a distribution is not available, or where the data was not generated by the distribution specified by the model.

In this work, we present two variants of NPs that address these specific cases. First, we introduce Graph Neural Processes (GNP) that operate on graph structured data. We use local spectral information of the graphs, derived from the Laplacian, to modify the input of standard NPs. We use GNPs to predict weights, via regression, on edges in a graph, and show they out perform a number of competitive baselines.

Secondly, we introduce Wasserstein Neural Processes (WNP) which use the Wasserstein distance to overcome training problems associated with maximum likelihood. We analyze the situations were NPs break and show that WNPs perform well. We also show competitive performance on a higher dimensional image completion task.

Chapter 2 introduces the background of the Wasserstein distance that will help understand NPs and our proposed variants. Chapter 3 gives an introduction into NPs. Chapters 4 and 5 contain our main contributions. Chapter 4 covers GNPs with justification and experiments while Chapter 5 similarly covers WNPs.

Chapter 2

Background

In this chapter we give a formal and rigorous definition of the Wasserstein divergence and the associated probabilistic and mathematical theories. The goal of this chapter is to lay the theoretical ground work for the tools utilized in subsequent chapters. Readers can skip this chapter on first reading if a rigorous theory is not interesting.

Intuitively, the Wasserstein distance gives a measure between two probability distributions. If the distributions are identical, the Wasserstein distance is zero, and as the two distributions move apart, the distance increases. With some light assumptions about the embedding space's cost function, the Wasserstein distance is a true metric.

We leverage these facts and use the Wasserstein distance as a loss function for our regression tasks. By minimizing the Wasserstein distance while updating parameters of a model, a number of interesting distributional problems can be solved.

2.1 Metrics and Measures

In the world of mathematics, probability was used and informally defined for hundreds of years. However, in the 1950's the foundation of probability was rigorously built from first principles [23]. One of the main building blocks of probability is the *measure* and a number of mechanical tools (e.g., sigma algebra) to give formalism to randomness.

Informally, a measure is a function with certain properties familiar to us (e.g., integral of the entire probability space is 1) that operates on a probability space. A measure, typically written as $\mu : \Omega \mapsto \mathbb{R}$, is a real-valued function that we use as a rigorous way to work with probability distributions.

Formally, a measure is a generalization of length, area, and volume that defines the mass of some distribution. Let Ω be the sample space which is the set of all possible outcomes for a process. Then F is the set of events which consist of zero or more outcomes. A probability space is a triple (Ω, F, P) where P maps events to densities. In a measurable probability space P will be a measure μ. As one may expect, these three pieces have specific structure that makes them useful for analysis.

Firstly, the sample space Ω must be non-empty. The events object F must be a $\sigma-$algebra which means it is self inclusive, closed under complements, and closed under countable unions. Formally, this is summarized in the following:

$$\Omega \in F \tag{2.1}$$

$$\text{if } A \in F \text{ then we also know } (\Omega \backslash A) \in F \tag{2.2}$$

$$\text{if } A_i \in F \text{ for } i = 1, 2, \cdots \text{ then } (\cup_{i=1}^{\infty}) \in F \tag{2.3}$$

Finally, the measure μ must be a probability measure, this implies it behaves similarly to discrete probability distributions. In other words,

$$\text{if } \{A_i\}_{i=1}^{\infty} \subset F \text{ is pairwise disjoint and countable then} \tag{2.4}$$

$$\mu(\cup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} \mu(A_i) \tag{2.5}$$

$$\mu(\Omega) = 1 \tag{2.6}$$

This structure on the space gives us a probability space that we can use in a number of extremely interesting ways. We can now reason rigorously about the shape of distributions, the distances between these distributions, and properties of the distributions.

In general, spaces can be given additional structure through the identification of a metric. A metric is a distance. It is a function that maps a pair of points in the space to a single value on the real line. To be considered a metric, however, there are a number of axioms the function must fulfill.

$$d(x, y) \geq 0 \tag{2.7}$$

$$d(x, y) = 0 \text{ iff } x = y \tag{2.8}$$

$$d(x, y) = d(y, x) \tag{2.9}$$

$$d(x, z) \leq d(x, y) + d(y, z) \tag{2.10}$$

Looking forward, we will discuss a natural way to define a metric (distance) over measures in a probability space. This distance, called the Kantorovich, or Wasserstein, distance has many applications in applied math, computer vision, and machine learning.

2.2 Optimal Transport

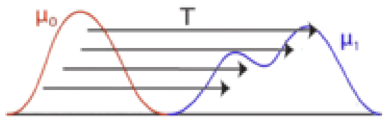

Figure 2.1: Continuous optimal transport theory [64]

As the founder of Optimal Transport theory, Gaspard Monge [47] was interested in the problem of a worker with a shovel moving a large pile of sand. The worker's goal is to erect a target pile of sand with a prescribed shape. The worker wishes to minimize their overall effort in carrying shovelfuls of sand. Monge formulated the problem as such in 1781 and worked to solve it.

Given two densities of mass $f, g \geq 0$ on $\mathbb{R}^d$ with unit mass $\int f(x)\, dx = \int g(y)\, dy = 1$, find an optimal (minimizing) map $T : \mathbb{R}^d \to \mathbb{R}^d$ which pushes one density onto another in such a way that

$$\int_A g(y)dy = \int_{T^{-1}(A)} f(x)\ dx \tag{2.11}$$

for any Borel subset $A \subset \mathbb{R}^d$ such that optimal is considered with respect to the quantity

$$M(T) = \int_{\mathbb{R}^d} |T(x) - x| f(x)\ dx \tag{2.12}$$

A physically intuitive interpretation is that of tiny sand particles that have a density described by f, and move to another configuration given by g in such a way as to minimize the average displacement across all particles. $T(x)$ represents the transport plan of particles which describes the motion or destination of one particle that started at x.

More generally, consider a measurable map $T : X \to Y$ where mass and density are preserved:

$$T_{\#}\mu(A) = \mu(T^{-1}(A)) \tag{2.13}$$

Then the optimal transport problem as formulated by Monge can be concisely written with two measurable functions μ, ν as

$$\inf_{T_{\#}\mu=\nu} \int_X c(x, T(x))\ d\mu(x) \tag{2.14}$$

for some more general cost function c. In many cases the cost function is taken to be the L_1 or L_2 norms which each have pros and cons. However, in general the non-linearity of the relation between the map, measures, and cost function make analysis of this formulation challenging. Additionally, this formulation requires all mass to go from source to target which implies that there is no mechanism for partial transportation of mass.

In general, progress wasn't really made on this problem until 1942 when Kantorovich introduce probabilistic couplings to the problem. Instead of giving the destination $T(x)$ for each particle, Kantorovich's formulation gives the number of particles moving between locations for all pairs (x, y). He did this by introducing a coupling that is called a "transport plan":

$$\Pi(\mu, \nu) = \{\pi \in P(X \times Y) : (\pi_x)_{\#}\pi = \mu, (\pi_y)_{\#}\pi = \nu\} \tag{2.15}$$

where π_x and π_y are projections from the product space. These are also often referred to as the marginals of the coupling.

The goal then in the Kantorovich formulation

$$W(\mu, \nu) = \inf_{\pi \in \Pi(\mu,\nu)} \int \int c(x, y)\ d\pi(x, y) \tag{2.16}$$

is to find the optimal coupling between to measures.

2.2.1 Wasserstein Distance

Over the years, optimal transport theory has grown into a mature and exciting field [61] with a number of awards going to those working on particular advances. In particular, the Wasserstein distance has seen much recent attention due to its nice theoretical properties and myriad applications [17]. These properties enable the study of distance between two probability distributions μ, ν for generative modeling.

The p-Wasserstein distance, $p \in [1, \infty]$, between μ, ν is defined to be the solution to the original Optimal Transport problem [61]:

$$W_p(\mu, \nu) = \left(\inf_{\pi \in \Pi(\mu,\nu)} \int_{X \times Y} d^p(x, y)\ d\pi(x, y) \right)^{\frac{1}{p}} \tag{2.17}$$

where $\Gamma(\mu, \nu)$ is the joint probability distribution with marginals μ, ν and $d^p(\cdot, \cdot)$ is the cost function.

An exciting result of the Wasserstein distance is that the quantity actually defines a distance (metric) on the space of probability measures. There are many geometric properties such as geodesics and gradient flow that can be calculated now using this distance.

In general, this distance is expensive to compute. There are, however, a number of methods used to approximate this distance. Firstly, we recall the Kantorovich-Rubinstein dual formulation [1, 61]:

$$W(\mu, \nu) = \sup_{||f||_L \leq 1} E_{x\sim\mu}[f(x)] - E_{x\sim\nu}[f(x)] \tag{2.18}$$

where we have let $p = 1$. It is important to note that the supremum is taken over all of the 1-Lipschitz functions. From this form emerges an adversarial algorithm for Wasserstein distance calculation. This is explored in a number of generative adversarial network (GAN) works, with [22] offering good performance with the addition of a gradient penalty. Additionally, by using a similar dual form a two-step computation that makes use of linear programming can be employed to exactly calculate the Wasserstein distance [44]. This method has favorable performance and is also used in generative modeling with the WGAN-TS[44].

As mentioned, exact computations of the Wasserstein distances are expensive. Therefore, many approximations have arisen that maintain many of the positive properties of optimal transport while being significantly more computationally efficient. We present a small subset of these methods here.

2.2.2 Sinkhorn

Another prominent method used to calculate Wasserstein distance that is much faster to compute is that of Sinkhorn distances [13]. This method makes use of Sinkhorn-Knopp's matrix scaling algorithm to compute the Wasserstein distance. These methods also introduce

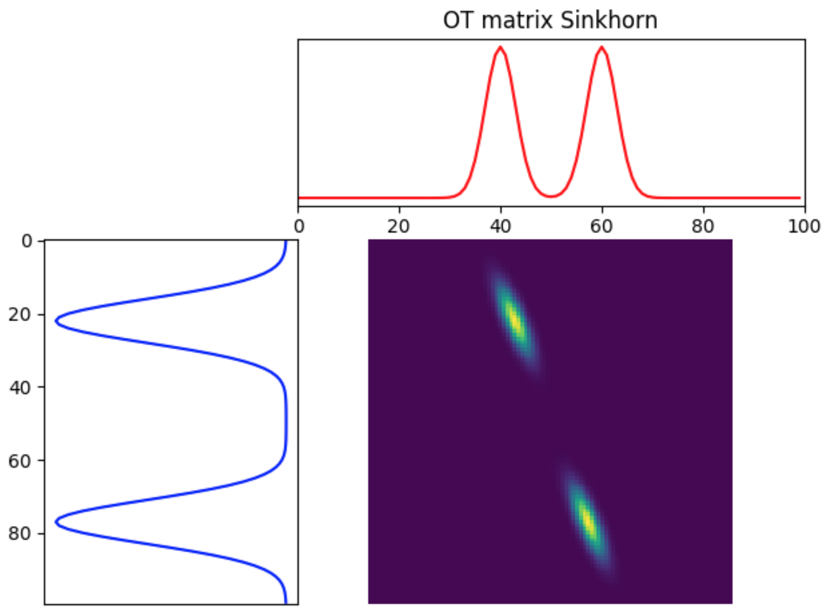

Figure 2.2: Regularized sinkhorn optimal transport plan between two empirical bimodal distributions

an entropic regularization term that drastically decreases the complexity of this problem offering speed ups of several orders of magnitude. Work has been successfully done to create a differentiable method [20] for using Sinkhorn loss during training of generative models.

In the case of empirical one-dimensional distributions, the p-Wasserstein distance is calculated by sorting the samples from both distributions and calculating the average distance between the sorted samples (according to the cost function). This is a fast operation requiring $\mathcal{O}(n)$ operations in the best case and $\mathcal{O}(n \log n)$ in the worst case with n the number of samples from each distribution [41]. This simple computation is taken advantage of in the computation of the sliced Wasserstein distance.

2.2.3 Sliced Wasserstein Distance

The sliced Wasserstein distance is qualitatively similar to Wasserstein distance, but is much easier to compute since it only depends on computations across one-dimensional distributions [39]. Intuitively, sliced Wasserstein distance is an aggregation of linear projects from high-dimensional distributions to one-dimensional distributions where regular Wasserstein distance is then calculated. The projection process is done via the Radon transform [40]. The one-dimensional projects are defined as

$$\mathcal{R}p_X(t;\theta) = \int_X p_X(x)\delta(t - \theta \cdot x)\ dx, \quad \forall \theta \in S^{d-1}, \quad \forall t \in \mathbb{R} \tag{2.19}$$

Then, by using these one-dimensional projections [7] the sliced Wasserstein distance is defined as

$$SW_p(\mu,\nu) = \left(\int_{S^{d-1}} W_p(\mathcal{R}_\mu(\cdot;\theta), \mathcal{R}_\nu(\cdot;\theta))\ d\theta\right)^{\frac{1}{p}} \tag{2.20}$$

It is important to note that this map SW_p inherits its being a distance from the fact that W_p is a distance. Similarly, the two distances induce the same topology on compact sets [53]. The integral in (eq. 2.20) is approximated by a Monte Carlo sampling variant of normal draws on S^{d-1} which are averaged across samples:

$$SW_p(\mu,\nu) \approx \left(\frac{1}{N}\sum_{i=1}^{N} W_p(\mathcal{R}_\mu(\cdot;\theta_i), \mathcal{R}_\nu(\cdot;\theta_i))\right)^{\frac{1}{p}} \tag{2.21}$$

Chapter 3

Neural Processes

We now turn our attention to Neural Processes. Recall that NPs map a set of context points and target points to a distribution over output values, non-linearly, for regression problems. To be more specific, NPs map an input $\vec{x_i} \in \mathbb{R}^{d_x}$ with a target $\vec{y_i} \in \mathbb{R}^{d_y}$ to an output distribution. This is done by defining a collection of distributions that are realized by encoding an arbitrary number of context points $X_C := (\vec{x_i})_{i \in C}$ and their associated outputs Y_C. Then an arbitrary number of target points $X_T := (\vec{x_i})_{i \in T}$, with their outputs Y_T, are used to recover $p(Y_T | X_T, Y_C, X_C)$. This entire process is accomplished using the samples from $P(X_C, Y_C)$.

3.1 Conditional Neural Processes

While NPs are the general model, to study the main mechanisms at work, we use a simpler version known as the Conditional Neural Process (CNP). First introduced in [18], Conditional Neural Processes are a simplified version of Neural Processes and are also used for regression tasks. CNPs still map inputs $\vec{x_i} \in \mathbb{R}^{d_x}$ and targets $\vec{y_i} \in \mathbb{R}^{d_y}$ to an output distribution. However, this is now done by defining a collection of conditional distributions by encoding an arbitrary number of context points $(X_C, Y_C) := ((\vec{x_i})_{i \in C}, (\vec{y_i})_{i \in C})$ and conditioning the target points $(X_T, Y_T) := ((\vec{x_i})_{i \in T}, (\vec{y_i})_{i \in T})$, with that encoding, to model the desired distribution. In other words, the model encodes a set of points X_C as input and uses this encoded representation to condition a set X_T for inference. Like GPs, CNPs are invariant to ordering of context points, and ordering of targets. It is important to note that, while the model is

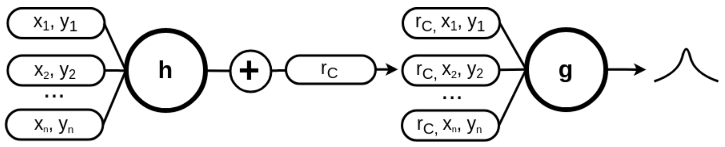

Figure 3.1: Conditional Neural Process Architecture. Traditionally, the data is maximized under the output distribution. In this work, we use the sliced Wasserstein distance. Also note that $\oplus$ is an arbitrary commutative aggregation function and h, g are parameterized neural networks. As an example, the context/target points could be (x, y) coordinates and pixel intensities if an image is being modeled.

defined for arbitrary context points C and target points T, it is common practice (which we follow) to use $C \subset T$.

The context point data is aggregated using a commutative operation $\oplus$ that takes elements in some $\mathbb{R}^d$ and maps them into a single element in the same space. In the literature, this is referred to as the r_C context vector. We see that r_C summarizes the information present in the observed context points. Formally, the CNP is learning the following conditional distribution:

$$P(Y_T|X_T, X_C, Y_C) \iff P(Y_T|X_T, r_C) \tag{3.1}$$

In practice, this is done by first passing the context points through a Neural Network h_θ to obtain a fixed length embedding r_i of each individual context point. These context point representation vectors are aggregated with $\oplus$ to form r_C. The target points X_T are then decoded, conditional to r_C, to obtain the desired output distribution z_i the models the target outputs y_i.

More formally, this process can be defined as:

$$r_i = h_\theta(\vec{x_i}) \qquad \forall \vec{x_i} \in X_C \tag{3.2}$$

$$r_C = r_1 \oplus r_2 \oplus r_3 \oplus \cdots \oplus r_n \tag{3.3}$$

$$z_i = g_\phi(\vec{y_i}|r_C) \qquad \forall \vec{y_i} \in X_T \tag{3.4}$$

3.2 Neural Process

Neural Processes are an extension of CNPs where a parameterized Normal $s_C := s(x_C, y_C)$ latent path is added to the model architecture. The purpose of this path is to model global dependencies between context and target predictions [19]. The computational complexity of the Neural Processes is $\mathcal{O}(n + m)$ for n contexts and m targets. Additionally, [19] show that in addition to being *scalable* these models are *flexible* and *invariant* to permutations in context points. Many of these benefits are shared in with CNPs, albeit with CNPs showing slightly worse experimental performance.

With the addition of the latent path, a new loss function is required. The encoder and decoder parameters are maximized using the ELBO loss:

$$\log p(y_T|x_T, x_C, y_C) \geq E_{q(z|s_T)}\left[\log p(y_T|x_T, r_C, z)\right] - D_{KL}(q(z|s_T)||q(z|s_C)) \tag{3.5}$$

A random subset of context points is used to predict a set of target points, with a KL regularization term that encourages the summarization of the context to be close to the summarization of the targets.

Work has been done [34] to explore other aggregation operations such as attention [3]. These extensions solve a fundamental underfitting problems that plague both CNPs and NPs. These Attentive Neural Processes (ANP) introduce several types of attention into the model architecture in place of the neural networks which greatly improves performance at the cost of computational complexity which was shown to be $\mathcal{O}(n(n + m))$. It is important to note, however, that the introduction of attention causes the models to train more quickly (i.e., fewer iterations) which is faster in wall-clock time even if inference is slower. The ANPs utilize a fundamentally different architecture while still maximizing the ELBO loss in (eq 3.5).

3.3 Extensions

The general NP framework is a useful extension and generalization of GPs. Researchers noticed, however, that a main bottle neck is the commutative aggregation operation. As such, [34] introduce the Attentive Neural Process which uses attention to combine the encoded r vectors. Other extensions [45, 49] have also been proposed to improve training and stability.

In this work, we extend the neural process in two distinct ways. Firstly, we design a mechanism to work on graph structured data. Secondly, we overcome weaknesses in maximum likelihood learning using Wasserstein distance to learn a larger variety of desirable functional distributions.

Chapter 4

Graph Neural Processes

4.1 Introduction

In this work, we consider the problem of imputing the value of an edge on a graph. This is a valuable problem when an edge is known to exist, but due to a noisy signal, or poor data acquisition process, the values on that edge are unknown. We solve this problem using a proposed method called Graph Neural Processes.

In recent years, deep learning techniques have been applied, with much success, to a variety of problems. However, many of these neural architectures (e.g., CNNs) rely on the underlying assumption that our data is Euclidean. However, since graphs do not lie on regular lattices, many of the concepts and underlying operations typically used in deep learning need to be extended to non-Euclidean valued data. Deep learning on graph-structured data has received much attention over the past decade [5, 21, 42, 56, 66] and has show significant promise in many applied fields [5]. These ideas were recently formalized as *geometric deep learning* [9] which categorizes and expounds on a variety of methods and techniques. These techniques are designed to work with graph-structured data and extend deep learning into new research areas.

Similarly, but somewhat orthogonally, progress has been made in applications of Bayesian methods to deep learning [2]. These extensions to typical deep learning give insight into what the model is learning and where it may encounter failure modes. In this work, we use some of this progress to impute the value distribution on an edge in graph-structured data.

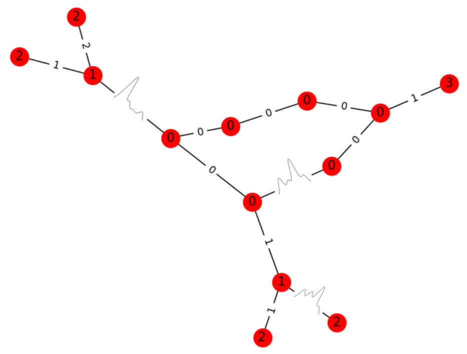

Figure 4.1: Graph with imputed edge distributions

In this work we propose a novel architecture and training mechanism which we call Graph Neural Processes (GNP). This architecture is based on the ideas first formulated in [18] in that it synthesizes global information to a fixed length representation that is then used for probability estimation. Our contribution is to extend those ideas to graph-structured data and show that the methods perform favorably.

Specifically, we use features typically used in Graph Neural Networks as a replacement to the convolution operation from traditional deep learning. These features, when used in conjunction with the traditional CNP architecture offer local representations of the graph around edges and assist in the learning of high level abstractions across classes of graphs. Graph Neural Processes learn a high level representation across a family of graphs, in part by utilizing these instructive features.

4.2 Background

4.2.1 Graph Structured Data

We define a graph $G = (a, V, E)$ where a is some global attribute[1], V is the set of nodes where $V = \{v_i\}_{i=1:N^v}$ with v_i being the node's attribute, and E is the set of edges where $E = \{e_k, v_k, u_k\}_{k=1:N^e}$ where e_k is the attribute on the edge, and v_k, u_k are the two nodes

[1]typically a member of the real numbers, $\mathbb{R}$. Our method does not utilize this global attribute.

connected by the edge. In this work, we focus on undirected graphs, but the principles could be extended to many other types of graphs.

4.2.2 Graph Neural Networks

Of all the inductive biases introduced by standard deep learning architectures, the most common is that we are working with Euclidean data. This fact is exploited by the use of convolutional neural networks (CNN), where spatially local features are learned for downstream tasks. If, however, our data is Non-Euclidean (e.g., graphs, manifolds) then many of the operations used in standard deep learning (e.g., convolutions) no longer produce the desired results. This is due to the fact that the intrinsic measures and mathematical structure on these surfaces violates assumptions made by traditional deep learning operations.

There has been much work done recently in Graph Neural Networks (GNN) that operate on graph-structured inputs [66]. There are two main building blocks of geometric methods in GNNs: *spectral* [9] and *spatial* [15] methods. These methods are unified into *Message Passing Neural Networks*, and *Non-local Neural Networks.* A more thorough discussion of these topics can be found in [66].

Typically, when using these GNN methods, the input graphs need to be of the same size. In other words have the same number of nodes and edges. This is because of the technicalities involved in defining a spectral convolution. However, in the case of Graph Neural Processes, local graph features are used, which allows one to learn conditional distributions over arbitrarily sized graph-structured data.

One important concept utilized in the spectral GNN methods is that of the Graph Laplacian. Typically, the graph Laplacian is defined as the adjacency matrix subtracted from the degree matrix $L = D - A$. This formulation, common in multi-agent systems and algebraic graph theory, hampers the flow of information and does not capture a number of local graph properties [11]. As such, the Normalized Symmetric Graph Laplacian is given as

$$L = I_n - D^{-1/2}AD^{-1/2} \tag{4.1}$$

where I is the nxn identity matrix and n is the number of nodes connected by an edge. This object can be thought of as the difference between the average node or edge value on a graph around a point and the actual node or edge value at that point. This, therefore, encodes local structural information about the graph itself.

In spectral GNN methods, the eigenvalues and eigenvectors of the graph Laplacian are used to define convolution operations on graphs. In this work, however, we use the local structural information encoded in the Laplacian as an input feature for the context points we wish to encode.

4.3 Related Work

4.3.1 Edge Imputation

In many applications, the existence of an edge is known, but the value of the edge is unknown (e.g., traffic prediction, social networks). Traditional edge imputation involves generating a point estimate for the value on an edge. This can be done through mean filling, regression, or classification techniques [28]. These traditional methods, especially mean filling, can fail to maintain the variance and other significant properties of the edge values. In [28] they show "Bias in variances and covariances can be greatly reduced by using a conditional distribution and replacing missing values with draws from this distribution." This fact, coupled with the neural nature of the conditional estimation, gives support to the hypothesis that Graph Neural Processes preserve important properties of edge values, and are effective in the imputation process.

4.3.2 Bayesian Deep Learning

In Bayesian neural networks, often the goal is to learn a function $y = f(x)$ given some training inputs $X = \{x_1, \cdots, x_n\}$ and corresponding outputs $Y = \{y_1, \cdots, y_n\}$. This function can be approximated using a fixed neural network that provides a likely correlational explanation for the relationship between x and y. There has been work done in this area [65] and there are two main types of Bayesian Deep Learning. This is not an exhaustive list of all the methods, but a broad overview of two. Firstly, instead of using point estimates for the parameters θ of the neural network, a distribution over parameter values is used. In other words, the weights are modeled as a random variable with an imposed prior distribution. This encodes, *a priori* uncertainty information about the neural transformation.

Since the weight values W are not deterministic, the output of the neural network can also be modeled as a random variable. In this first class a generative model can be learned based on the structure of the neural network and loss function being used. Predictions from these networks can be obtained by integrating with respect to the posterior distribution of W

$$p(y|x, X, Y) = \int p(y|x, W)\ p(W|X, Y)\ dW. \tag{4.2}$$

This integral is often intractable in practice. A number of techniques have been proposed in the literature to overcome [65] this problem.

A second type of Bayesian deep learning is when the output of the network is modeled as a distribution. This is more common in practice and allows a Bayesian neural network to encode information about the distribution over output values given certain inputs. This is a very valuable property of Bayesian deep learning that GNPs help capture.

In the case of Graph Neural Processes, we model the output of the process as a random variable and learn a conditional distribution over that variable. This fits into the second class of Bayesian neural networks since the weights W are not modeled as random variables in this work.

4.4 Model and Training

While the h_θ and g_ϕ encoder and decoder could be implemented arbitrarily, we use fully connected layers that operate on informative features from the graph. In GNPs, we use local spectral features derived from global spectral information of the graphs. Typical graph networks that use graph convolutions require a fixed size Laplacian to encode global spectral information; for GNPs we use an chosen fixed size neighborhood of the Laplacian around each edge.

To be precise, with the Laplacian L as defined in equation (4.1) one can compute the spectra $\sigma(L)$ of L and the corresponding eigenvector matrix Λ. In Λ, each column is an eigenvector of the graph Laplacian. To get the arbitrary fixed sized neighborhood, whose size is chosen through k, around each node/edge we define the restriction of Λ as

$$\Lambda|_k = (\Lambda_{kj})_{\substack{k \in r \\ 1 \leq j \leq m}} \tag{4.3}$$

where k represents a row (or rows) of the matrix and m represents the number of columns. We call this restriction the local structural eigenfeatures of an edge. These eigenfeatures are often used in conjunction with other, more standard, graph based features. For example, we found that the node values and node degrees for each node attached to an edge serve as informative structural features for GNPs. In other words, to describe each edge we use the local structural eigenfeatures concatenated with a 4-tuple: $(v_i, u_j, d(v_i), d(u_j))$ where $d : V \to \mathbb{R}$ is a function that returns the degree of a node and v_i, u_i represent the value of two nodes whose connecting edge value we wish to impute.

Formally, we define the **Encoder** h_θ, to be a 4-layer fully connected neural network with ReLU non-linearities. It takes the GNP features from each node pair v_i, u_i in the set of context points as input, and outputs a fixed length (e.g., 256) vector. This vector r_C is the aggregation of information across the arbitrarily sampled context points. The **Decoder** g_ϕ is also a 4-layer fully connected neural network with ReLU non-linearities. The decoder takes, as input, a concatenation of the r_C vector from the encoder and the GNP features for *every* node pair v_i, u_i which constitutes the set of target points. Then, for each edge, the concatenation of r_C and the associated GNP feature vector is passed through the layers of the decoder until an output distribution is reached. Where the output distribution size is defined over the number of unique values an edge could take on, which is dataset dependant.

Additionally, as a training choice, we obtain the **context points** by first defining a lower bound $p_0 \in [0,1]$ and an upper bound $p_1 \in [0,1]$ with $p_0 < p_1$ as parameters of a Uniform distribution. Then, the number of context points is $p \cdot n$ where $p \sim \text{unif}(p_0, p_1)$ and $n = |E|$. The context points all come from a single graph, and training is done over a family of graphs. The GNP is designed to learn a representation over a family of graphs and impute edge values on a new member of that family.

The loss function $\mathcal{L}$ is problem specific. In the CNP work, as mentioned above, Maximum Likelihood is used to encourage the output distribution of values to match the true target values. Additionally, one could minimize the Kullback–Leibler, Jensen–Shannon, or the Earth-Movers divergence between output distribution and true distribution, if that distribution is known. In this work, since the values found on the edges are categorical we use the multiclass Cross Entropy with j indexing all classes:

$$\begin{aligned}\text{loss}(x, class) &= -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) && (4.4)\\ &= -x[class] + \log\left(\sum_j \exp(x[j])\right) && (4.5)\end{aligned}$$

We train our method using the ADAM optimizer [35] with standard hyper-parameters. To give clarity, and elucidate the connection and differences between CNPs and GNPs, we present the training algorithm for Graph Neural Processes in Algorithm (1).

Algorithm 1 Graph Neural Processes. All experiments use $n_{\text{epochs}} = 10$ and default Adam optimizer parameters

Require: p_0, lower bound percentage of edges to sample as context points. p_1, corresponding upper bound. m, size of slice (neighborhood) of local structural eigenfeatures.
Require: θ_0, initial encoder parameters. ϕ_0 initial decoder parameters.

```
 1: Let $X$ input graphs
 2: for $t = 0, \cdots, n_{\text{epochs}}$ do
 3:     for $x_i$ in $X$ do
 4:         Sample $p \leftarrow \text{unif}(p_0, p_1)$
 5:         Assign $n_{\text{context points}} \leftarrow p \cdot |\text{Edges}(x_i)|$
 6:         Sparsely Sample $x_i^{cp} \leftarrow x_i|_{n_{\text{context points}}}$
 7:         Compute degree and adj matrix $D$, $A$ for graph $x_i^{cp}$
 8:         Compute $L \leftarrow I_{n_{\text{context points}}} - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$
 9:         Define $F^{cp}$ as empty feature matrix for $x_i^{cp}$
10:         Define $F$ as empty feature matrix for full graph $x_i$
11:         for edge $k$ in $x_i$ do
12:             Extract eigenfeatures $\Lambda|_k$ from $L$, see eq (4.3)
13:             Concatenate $[\Lambda|_k; v_k; u_k; d(v_k); d(u_k)]$ where $v_k, u_k$ are the attribute values at the
                node, and $d(v_k), d(u_k)$ the degree at the node.
14:             if edge $k \in x_i^{cp}$ then
15:                 Append features for context point to $F^{cp}$
16:             end if
17:             Append features for all edges to $F$
18:         end for
19:         Encode and aggregate $r_C \leftarrow h_\theta(F^{cp})$
20:         Decode $\tilde{x}_i \leftarrow g_\phi(F|r_C)$
21:         Calculate Loss $l \leftarrow \mathcal{L}(\tilde{x}_i, x_i)$
22:         Step Optimizer
23:     end for
24: end for
```

4.5 Applications

To show the efficacy of GNPs on real world examples, we performed a series of experiments on a collection of 16 graph benchmark datasets [33]. These datasets span a variety of application areas, and have a diversity of sizes; features of the explored datasets are summarized in Table 4.2. While the benchmark collection has more than 16 datasets, we selected only those that have both node and edge labels that are fully known; these were then artificially sparsified to create graphs with known edges but unknown edge labels.

We use the model described in Section (4.4) and the algorithm presented in Algorithm (1) and train a GNP to observe $p \cdot n$ context points where $p \in [.4, .9]$ and $n = |E|$. We compare GNPs with naive baselines and a strong random forest model for the task of edge imputation. The baselines are commonly used in edge imputation tasks [28]:

- **Random**: a random edge label is imputed for each unknown edge
- **Common**: the most common edge label is imputed for each unknown edge
- **Local Common**: the most common local edge value in a neighborhood is imputed for each unknown edge
- **Random forest**: from scikit-learn; default hyperparameters
- **Feed forward neural network**: constructed with the same number of parameters and same non-linearities as the GNP, but without the inductive bias

For each algorithm on each dataset, we calculate weighted precision, weighted recall, and weighted F1-score (with weights derived from class proportions), averaging each algorithm over multiple runs. Statistical significance was assessed with a two-tailed t-test with $\alpha = 0.05$.

4.5.1 Results

The results are pictured in Figures (4.2 – 4.4); all of the data is additionally summarized in Table 4.1. We first note several high-level results, and then look more deeply into the results on different subsets of the overall dataset.

First, we note that the GNP provides the best F1-score on 14 out of 16 datasets, and best recall on 14 out of datasets (in this case, recall is equivalent to classification accuracy). By learning a high-level abstract representation of the data, along with a conditional distribution over edge values, the Graph Neural Process is able to perform well at this edge imputation task, beating both naive and strong baselines. The GNP is able to do this on datasets with as few as about 300 graphs, or as many as about 9000. We also note that the GNP is able to overcome class imbalance.

AIDS. The AIDS Antiviral Screen dataset [51] is a dataset of screens checking tens of thousands of compounds for evidence of anti-HIV activity. The available screen results are chemical graph-structured data of these various compounds. A moderately sized dataset, the GNP beats the RF by a statistically significant ($p < 0.001$) 7% in average performance across the three measures.

bzr,cox2,dhfr,er. These are chemical compound datasets BZR, COX2, DHFR and ER which come with 3D coordinates, and were used by [46] to study the pharmacophore kernel. Results are mixed between algorithms on these datasets; these are the datasets that have an order of magnitude more edges (on average) than the other datasets. These datasets have a large class imbalance. Which means the Common baseline yields around 90% accuracy. The BZR dataset has 61,594 edges in class 1, and only 7,273 in the other 4 classes combined. Despite this imbalance, the GNP yields best F1 and recall on 2/4 of these datasets. while the random forest gives the best precision on 3/4. It may simply be that there is so much data to work with, and the classes are so imbalanced, that it is hard to find much additional signal.

mutagenicity, MUTAG. The MUTAG dataset [14] consists of 188 chemical compounds divided into two classes according to their mutagenic effect on a bacterium. While the mutagenicity dataset [31] is a collection of molecules and their interaction information with in vitro.

Here, GNP beats the random forest by several percent; both GNP and RF are vastly superior to naive baselines.

PTC_*. The various Predictive Toxicology Challenge [25] datasets consist of several hundred organic molecules marked according to their carcinogenicity on male and female mice and rats. [25] On the PTC family of graphs, GNP bests random forests by 10-15% precision, and 3-10% F1-score; both strongly beat naive baselines.

Tox21_*. This data consists of 10,000 chemical compounds run against human nuclear receptor signaling and stress pathway, it was originally designed to look for structure-activity relationships and improve overall human health. [16] On the Tox family of graphs, the GNP strongly outperforms all other models by about 20% precision; about 12% F1; and about 10% recall.

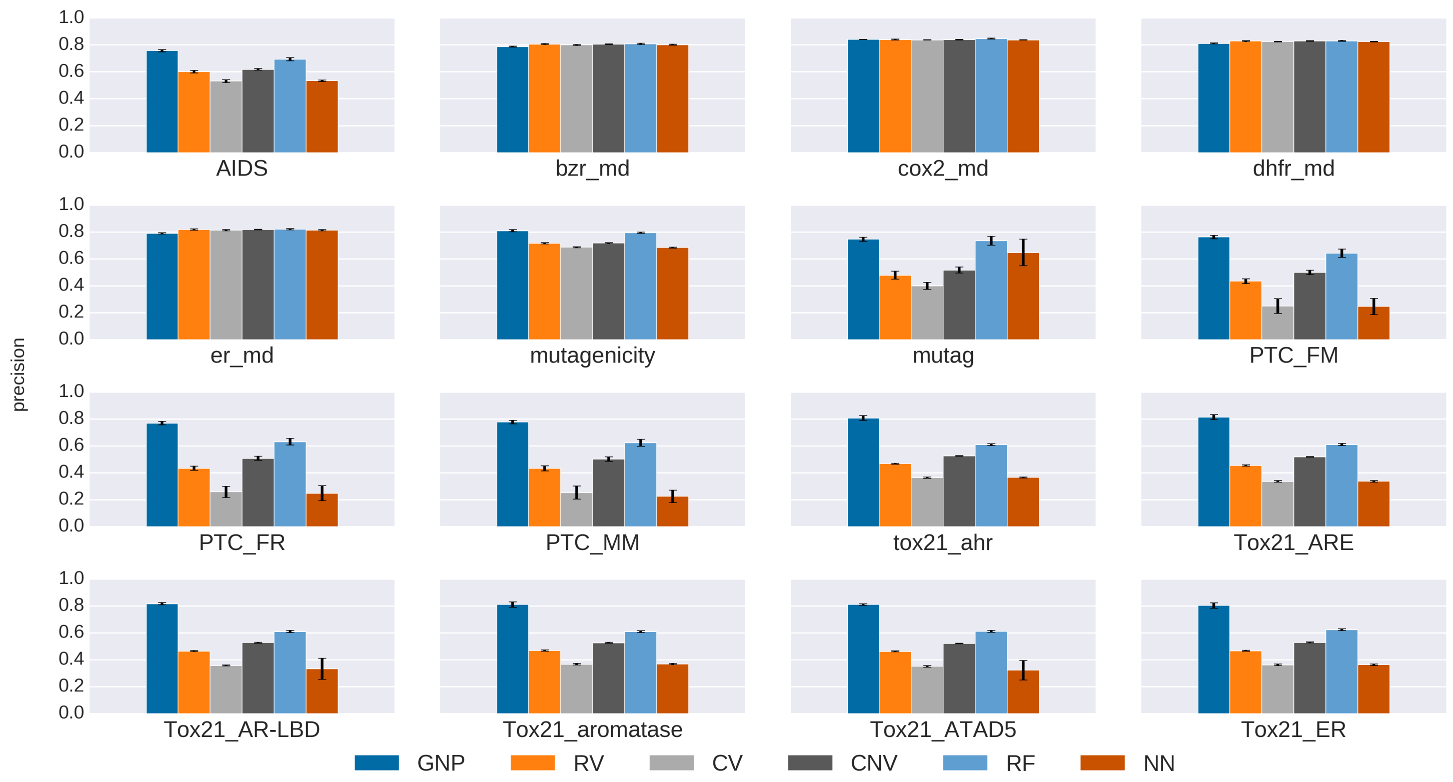

Figure 4.2: Experimental precision graph compared with baselines. We see our method performs achieves a $\sim .2$ higher precision on average.

4.6 Areas for Further Exploration

This introduction of GNPs is meant as a proof of concept to encourage further research into Bayesian graph methods. As part of this work, we list a number of problems where GNPs could be applied. **Visual scene understanding** [50] [54] where a graph is formed through some non-deterministic process where a collection of the inputs may be corrupted, or inaccurate. As such, a GNP could be applied to infer edge or node values in the scene to improve downstream accuracy. **Few-shot learning** [55] where there is hidden structural information. A method like [32] could be used to discover the form, and a GNP could then be leveraged to impute other graph attributes.

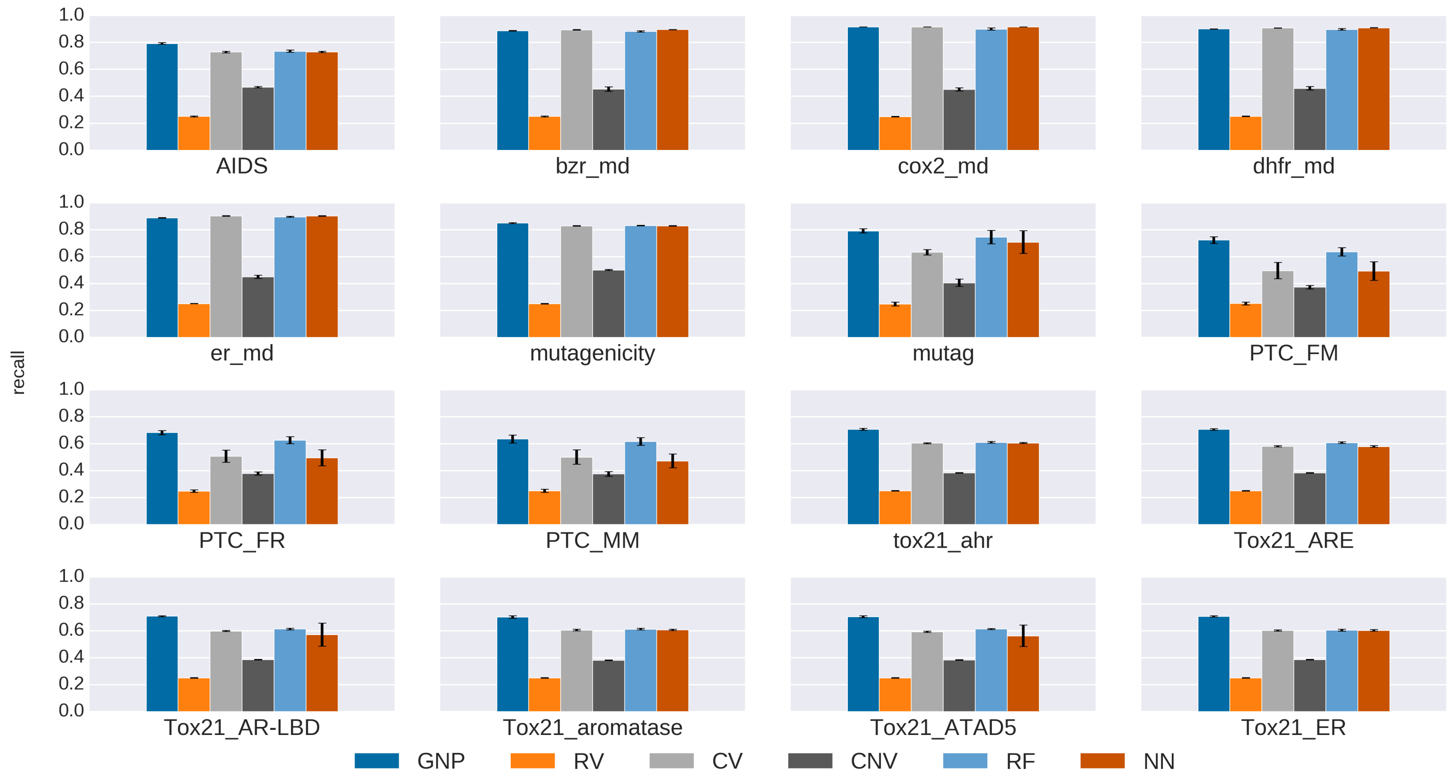

Figure 4.3: Experimental recall graph compared with baselines

Learning dynamics of physical systems [4] [10] [63] [60] [52] with gaps in observations over time, where the GNP could infer values involved in state transitions.

Traffic prediction on roads or waterways [43] [12] or throughput in cities. The GNP would learn a conditional distribution over traffic between cities

Multi-agent systems [58] [27][37] where you want to infer inter agent state of competing or cooperating agents. GNP inference could run in conjunction with other multi-agent methods and provide additional information from graph interactions.

Natural language processing in the construction of knowledge graphs [8] [48] [24] by relational infilling or reasoning about connections on a knowledge graph. Alternatively, they could be used to perform **semi-supervised text classification** [38] by imputing relations between words and sentences.

There are a number of computer vision applications where graphs and GNPs could be extremely valuable. For example, one could improve **3D meshes and point cloud** [62]

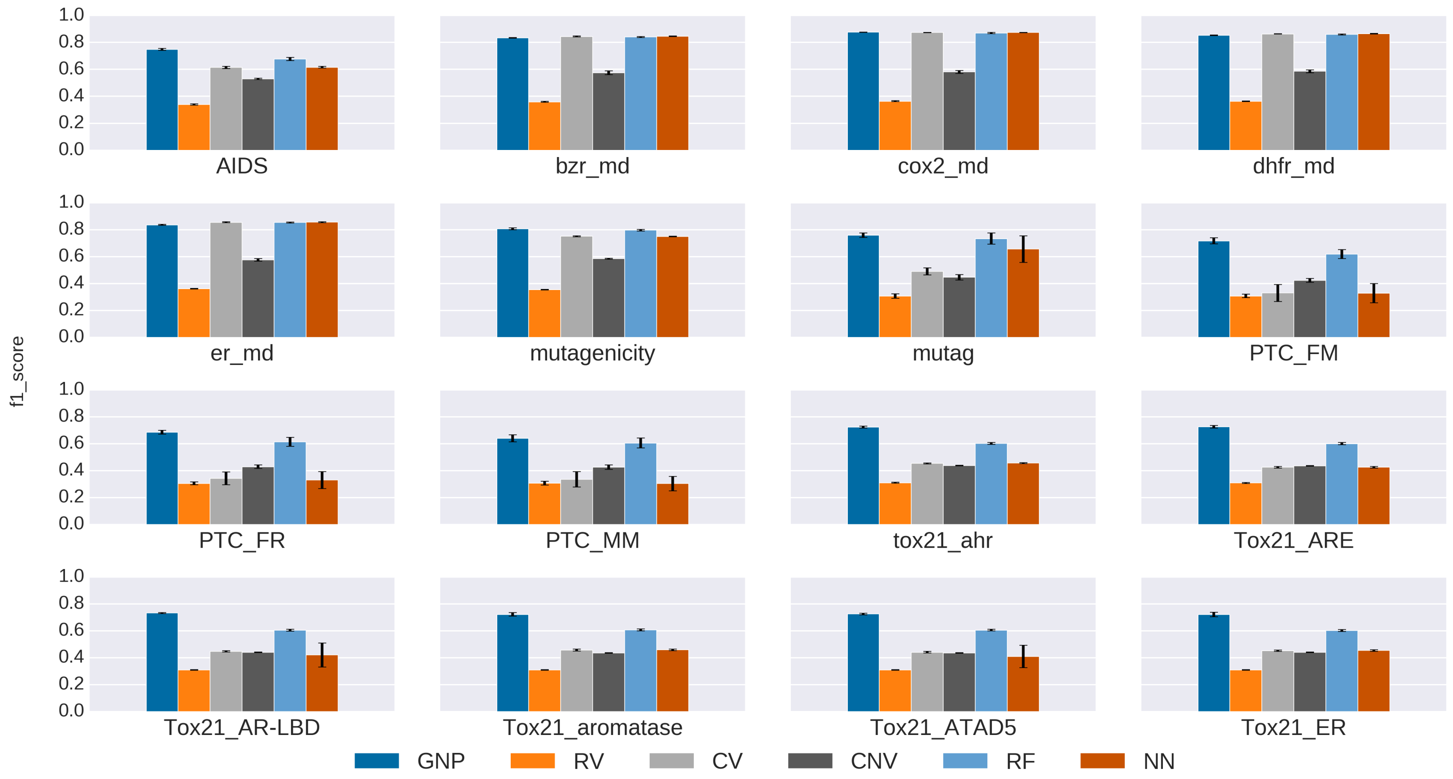

Figure 4.4: Experimental F1-score graph compared with baselines

construction using lidar or radar during tricky weather conditions. The values in the meshes or point clouds could be imputed directly from conditional draws from the distribution learned by the GNP.

Dataset	RF			RV			CV			GNP		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
AIDS	0.69	0.74	0.68	0.60	0.25	0.34	0.53	0.73	0.61	**0.76**	**0.79**	**0.75**
bzr_md	**0.81**	0.88	**0.84**	0.81	0.25	0.36	0.80	**0.89**	0.84	0.79	**0.89**	0.83
cox2_md	**0.85**	0.90	0.87	0.84	0.25	0.36	0.84	0.91	0.87	0.84	**0.92**	**0.88**
dhfr_md	**0.83**	0.90	0.86	**0.83**	0.25	0.36	0.82	**0.91**	0.86	0.81	0.90	**0.85**
er_md	0.82	**0.90**	0.85	0.82	0.25	0.36	0.81	**0.90**	**0.86**	0.79	0.89	0.84
mutagenicity	0.79	0.83	0.80	0.72	0.25	0.35	0.69	0.83	0.75	**0.81**	**0.85**	**0.81**
mutag	**0.73**	0.74	0.73	0.48	0.25	0.31	0.40	0.63	0.49	**0.75**	**0.79**	**0.76**
PTC_FM	0.64	0.64	0.62	0.43	0.25	0.31	0.25	0.50	0.33	**0.76**	**0.72**	**0.72**
PTC_FR	0.63	0.63	0.61	0.43	0.25	0.31	0.26	0.51	0.34	**0.77**	**0.68**	**0.69**
PTC_MM	0.62	0.62	0.61	0.43	0.25	0.31	0.25	0.50	0.34	**0.78**	**0.64**	**0.64**
tox21_ahr	0.61	0.61	0.60	0.47	0.25	0.31	0.36	0.60	0.45	**0.81**	**0.71**	**0.72**
Tox21_ARE	0.61	0.61	0.60	0.46	0.25	0.31	0.34	0.58	0.43	**0.81**	**0.71**	**0.73**
Tox21_AR-LBD	0.61	0.61	0.61	0.47	0.25	0.31	0.36	0.60	0.45	**0.82**	**0.71**	**0.73**
Tox21_aromatase	0.61	0.61	0.61	0.47	0.25	0.31	0.37	0.61	0.46	**0.81**	**0.70**	**0.72**
Tox21_ATAD5	0.61	0.61	0.61	0.46	0.25	0.31	0.35	0.59	0.44	**0.81**	**0.71**	**0.73**
Tox21_ER	0.62	0.60	0.60	0.47	0.25	0.31	0.36	0.60	0.45	**0.80**	**0.71**	**0.72**

Table 4.1: Experimental Results. RF=Random Forest; RV=Random edge label; CV=most common edge label; GNP=Graph Neural Process. P=precision; R=recall; F1=F1 score. Statistically significant bests are in bold with non-significant ties bolded across methods.

Dataset	$\|X\|$	$\|N\|$	$\|E\|$	$\|\cup\{e_k\}\|$
AIDS	2000	15.69	16.20	3
BZR_MD	306	21.30	225.06	5
COX2_MD	303	26.28	335.12	5
DHFR_MD	393	23.87	283.01	5
ER_MD	446	21.33	234.85	5
Mutagenicity	4337	30.32	30.77	3
MUTAG	188	17.93	19.79	4
PTC_FM	349	14.11	14.48	4
PTC_FR	351	14.56	15.00	4
PTC_MM	336	13.97	14.32	4
Tox21_AHR	8169	18.09	18.50	4
Tox21_ARE	7167	16.28	16.52	4
Tox21_aromatase	7226	17.50	17.79	4
Tox21_ARLBD	8753	18.06	18.47	4
Tox21_ATAD5	9091	17.89	18.30	4
Tox21_ER	7697	17.58	17.94	4

Table 4.2: Features of the explored data sets

Chapter 5

Wasserstein Neural Processes

5.1 Introduction

Gaussian Processes (GPs) are nonparametric probabilistic models that are widely used for nonlinear regression problems. They map an input $\vec{x_i} \in \mathbb{R}^{d_x}$ to an output $\vec{y_i} \in \mathbb{R}^{d_y}$ by defining a collection of jointly Gaussian conditional distributions. These distributions can be conditioned on an arbitrary number of context points $X_C := (\vec{x_i})_{i \in C}$ where C is the context points and their associated outputs Y_C. Given the context points, an arbitrary number of target points $X_T := (\vec{x_i})_{i \in T}$ for T target points, with their outputs Y_T, can be modeled using the conditional distribution $P(X_T | X_C, Y_C)$. This modeling is invariant to the ordering of context points and to the ordering of targets. Formally, they define a nonparametric distribution over smooth functions, and leverage the almost magical nature of Gaussian distributions to both solve regression tasks and provide confidence intervals about the solution. However, GPs are computationally expensive, and the strong Gaussian assumptions they make are not always appropriate.

Neural Processes (NPs) can be loosely thought of as the neural generalization of GPs. They blend the strengths of GPs and deep neural networks: like GPs, NPs are a class of models that learn a mapping from a context set of input-output pairs to a distribution over functions. Also like GPs, NPs can be conditioned on an arbitrary set of context points, but unlike GPs (which have quadratic complexity), NPs have linear complexity in the number of context points [19]. NPs inherit the high model capacity of deep neural networks, which gives them the ability to fit a wide range of distributions, and can be blended with the latest deep learning best practices (for example, NPs were recently extended using attention [34] to overcome a fundamental underfitting problem which lead to inaccurate function predictions).

Importantly, these NPs still make Gaussian assumptions; furthermore, the training algorithm of these Neural Processes uses a loss function that relies on maximum-likelihood and the KL divergence metric. Recently in the machine learning community there has been much discussion [26] around generative modeling, and the use of Wasserstein distances. The Wasserstein distance, a true metric, is a desirable tool because it measures distance between two probability distributions, even if the support of those distributions is disjoint. The KL divergence, however, is undefined in such cases (or infinity); this corresponds to situations where the data likelihood is exactly zero, which can happen in the case of misspecified models.

This raises the question: can we simultaneously improve the maximum likelihood training of NPs, and replace the Gaussian assumption in NPs with a more general model — ideally one *where we do not need to make any assumptions about the data likelihood*?

More formally, in the limit of data, maximizing the expected log probability of the data is equivalent to minimizing the KL divergence between the distribution implied by a model P_θ and the true data distribution Q:

$$\begin{aligned}
\theta_{\text{min KL}} &= \arg\min_\theta D_{KL}(Q||P_\theta) \\
&= \arg\min_\theta E_{\mathbf{x}\sim Q}[\log Q(\mathbf{x}) - \log P_\theta(\mathbf{x})] \\
&= \arg\min_\theta -E_{\mathbf{x}\sim Q}[\log p(\mathbf{x}|\theta)] \\
&= \arg\max_\theta \lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} \log(p(\mathbf{x_i}|\theta)) \\
&= \theta_{MLE}
\end{aligned}$$

However, if the two distributions Q and P_θ have disjoint support (i.e., if even one data point lies outside the support of the model), the KL divergence will be undefined (or infinite) due to the evaluation of $\log(0)$. Therefore, it has been shown recently [1] that substituting Wasserstein for KL gives stable performance for non-overlapping distributions in the same space. This leads to Wasserstein distance as a proxy for the KL divergence interpretation of maximum likelihood learning.

This new framing is valuable in the misspecified case where data likelihood is zero under the model and additionally for the case when likelihood is unknown (such as in high dimensional data supported on low-dimensional manifolds, such as natural images), or incalculable. However, in all cases, it is required that we can draw samples from the target distribution.

The central contribution of this chapter is the powerful non-linear class of Wasserstein Neural Processes that can learn a variety of distributions when the data are misspecified under the model, and when the likelihood is unknown or intractable. We do so by replacing the traditional maximum likelihood loss function with a Wasserstein divergence. We discuss computationally efficient approximations (including sliced Wasserstein distance [41]) and evaluate performance in several experimental settings.

5.2 Wasserstein Neural Processes

Our objective in combining Optimal Transport with Neural Processes is to model a desirable class of functions that are either misspecified, or with computationally intractable likelihood. We choose to use a CNP (fig 3.1) as our Neural Process architecture because CNPs are the simplest variant of the NP family and therefore we can illustrate the benefits of Wasserstein distance as a training regime without myriad ablation tests. Additionally, due to model simplicity, we can better ablate the performance and assign proper credit to the inclusion of Wasserstein distance. Extending WNPs to use a latent path, or attention, would be an interesting direction for future research.

To be precise, we use a parameterized Neural Network of shape and size determined according to task specifications. The input context x, y points are encoded and aggregated, as in traditional CNPs, using a commutative operation into an r_C of fixed length. The desired target x points (typically the entire set of points) are then conditioned (using concatenation) and decoded to produce a synthetic set of target y points. At this point, Wasserstein Neural Processes deviate from traditional NPs. Typically, as mentioned above, the target y likelihood would be calculated according to the output distribution (which we do as a comparison in the experiments). However, in our case, the decoded target y points are samples from an implicitly modeled distribution. As such, since the likelihood may be difficult to calculate, or non-existent, WNPs use sliced Wasserstein distance as a learning signal. This calculation is put forth in Algorithm (3).

The sliced Wasserstein distance [41] gives the benefit of scalable distance computation between two potentially disjoint distributions. We have included the algorithm for implementing the sliced Wasserstein distance in Algorithm (3). With this distance WNPs can model a larger variety of distributions than traditional NPs as discussed above.

Algorithm 2 Wasserstein Neural Processes.

Require: p_0, lower bound percentage of edges to sample as context points. p_1, corresponding upper bound.
Require: θ_0, initial encoder parameters. ϕ_0 initial decoder parameters.

```
1: Let $X$ input data in $\mathbb{R}^d$
2: for $t = 0, \cdots, n_{\text{epochs}}$ do
3:     for $x_i$ in $X$ do
4:         Sample $p \leftarrow \text{unif}(p_0, p_1)$
5:         Assign $n_{\text{context points}} \leftarrow p \cdot |\text{Target Points}|$
6:         Sparsely Sample $x_i^{cp} \leftarrow x_i|_{n_{\text{context points}}}$ and associated $y_i^{cp}$
7:         Obtain one-hot encoding of $x_i^{cp}$ points and concatenate with $y_i^{cp}$ as $F^{cp}$
8:         Encode and aggregate $r_C \leftarrow h_\theta(F^{cp})$
9:         Decode $\tilde{x}_i \leftarrow g_\phi(F|r_C)$
10:        Calculate Sliced Wasserstein Distance $l \leftarrow \mathcal{W}(\tilde{x}_i, x_i)$
11:        Step Optimizer
12:    end for
13: end for
```

Algorithm 3 Sliced Wasserstein Distance

```
Require: $n \in [5, 50]$, number of desired projections.
Require: $d$, dimension of embedding space.
Require: $p$, power of desired Wasserstein distance.

1: Let $X$ be the empirical synthetic distribution and $Y$ be samples from the true distribution
2: Sample projections $P$ from $S^d$ i.e., $\mathcal{N}$(size= $(n, d)$) and normalize using dimension-wise $L_2$
3: Project $\hat{X} = X \cdot P^T$
4: Project $\hat{Y} = Y \cdot P^T$
5: Sort $\hat{X}^T$ and $\hat{Y}^T$
6: Return $(\hat{X}^T - \hat{Y}^T)^p$
```

5.3 Experiments

As discussed in Section (5.1), traditional Neural Processes are limited by their use of the KL divergence, both explicitly and implicitly. In this section, we demonstrate that Wasserstein Neural Processes can learn effectively under conditions that cause a traditional NP to fail. The first two experiments illustrate NPs' fundamental failing of relying on the likelihood function for learning. The final experiment shows the ability of WNPs to work on larger scale tasks.

5.3.1 Misspecified Models - Linear Regression with Uniform Noise

In standard linear regression, parameters $\beta = (\beta_1, \beta_2, \cdots, \beta_n)^T$ are estimated in the model

$$\hat{Y} = \beta\vec{X} + b + \eta \tag{5.1}$$

where a stochastic noise model η is used to model disturbances and $\vec{X}$ is our row vector of data. Traditionally, $\eta \sim \mathcal{N}(\mu, \sigma)$ for some parameterized Normal distribution. In such a traditional case, Neural Processes can learn the a conditional distribution over the data x. This is because all the data has a calculable likelihood under any setting of the parameters.

However, consider the case where the noise model is Uniform (e.g., $Unif[-1, 1]$). There are now settings for β, b under which the data x would have exactly zero likelihood (i.e,, any time a single data point falls outside the "tube" of noise). In the worst case, there may be *no* setting of the parameters that provides non-zero likelihood. Furthermore, because of the uniform nature of the noise likelihoods, there are no useful gradients. As we see here,

$$L(\beta, b) = \Pi_{i \in I} p(y_i | x_i, \beta, b) \tag{5.2}$$

$$= \Pi_{i \in I} 0.5 \cdot \mathbb{1}(|y_i - (\beta x + b)| < 1) \tag{5.3}$$

with I an index into our training set, if even a single data point is outside the uniform density, the entire likelihood is zero. Therefore, any model based on likelihoods, including NPs, fails to learn. This is illustrated in fig. 5.1.

However, even if the data likelihood is degenerate, there is still a well-defined notion of the optimal transport cost between the observed data and the distribution implied by the model, no matter what setting of parameters is used. Since the Wasserstein distance is defined independent of likelihood, the WNP model still finds the proper fit of parameters given the data as seen in (fig. 5.1). This experiment could be analogous to the real world case where our prior knowledge of the data is fundamentally flawed. This flaw implies that our model is incorrect and misspecified. For this experiment, we generate synthetic data from $y = 1 \cdot x + 0 + \epsilon$ where $\epsilon \sim \mathcal{N}(0, 0.5)$. The sliced Wasserstein distance is calculated as described in Section 2.2.3 with $N = 50$. We let h and g be two-layer fully connected Neural Networks and our r_C vector is of size 32 and we use the Adam [35] optimizer with defaults.

5.3.2 Intractable Likelihoods - The Quantile "g-and-κ" Distribution

We now present a second example, with a distribution that is easy to sample from, but for which the likelihood cannot be tractably computed. The g-and-κ distribution [29, 59] is

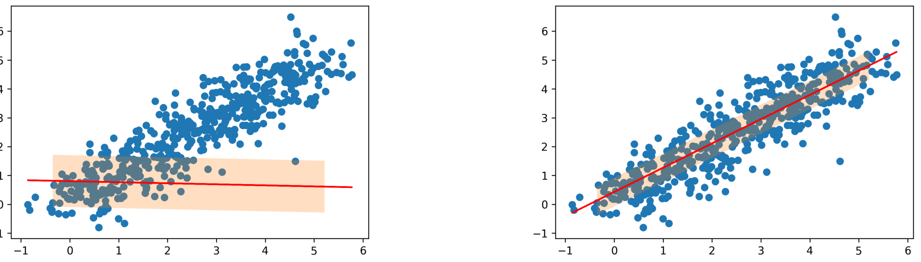

Figure 5.1: WNP regression results. Left: Failed NP regression with a uniform noise model $Unif[-1,1]$ and 500 synthetic data points. The line represents the estimated line of best fit, while the tube represents the undertainty of the model. Since the model is misspecified, the likelihood (eq 5.3) is zero and there is no learning signal for parameter updates. Right: Successful WNP regression with an identical experimental setup. In this case, the model does not rely on likelihood, instead it makes use of the signal from the sliced Wasserstein distance which persists even in the misspecified case.

defined for $r \in (0,1)$ and parameters $\theta = (a,b,g,\kappa) \in [0,10]^4$ as

$$a + b\left(1 + 0.8\frac{1-\exp(-gz(r))}{1+\exp(-gz(r))}\right)(1+z(r)^2)^{\kappa} z(r) \tag{5.4}$$

where $z(r)$ refers to the r-th quantile of a standard Normal distribution $\mathcal{N}(0,1)$. It is known that the density function and likelihood are analytically intractable, making it a perfect test for WNPs. This model is a standard benchmark for approximate Bayesian computation methods [57]. Despite the intractability, it is straightforward to sample i.i.d. variables from this distribution by simply plugging in standard Normals in place of $z(r)$ [6].

We follow the experimental set up in [6] and take $\theta = (3,1,2,0.5)$. Which produces a distribution seen in (fig. 5.2), each pane of the figure represents a time step of WNP training. Our model can learn this distribution even though the likelihood is analytically intractable. Additionally, we let h and g be two-layer fully connected neural networks which increases the capacity of the model. The r_C embedding vector is the same size (32 dimensional). In this case, we use a cyclic learning rate with the base learning rate $1e^{-3}$ and the max learning rate $1e^{-2}$, which are the default values in Pytorch. We found that in some cases, our model would output a linearly translated empirical distribution, and cycling the learning rate would

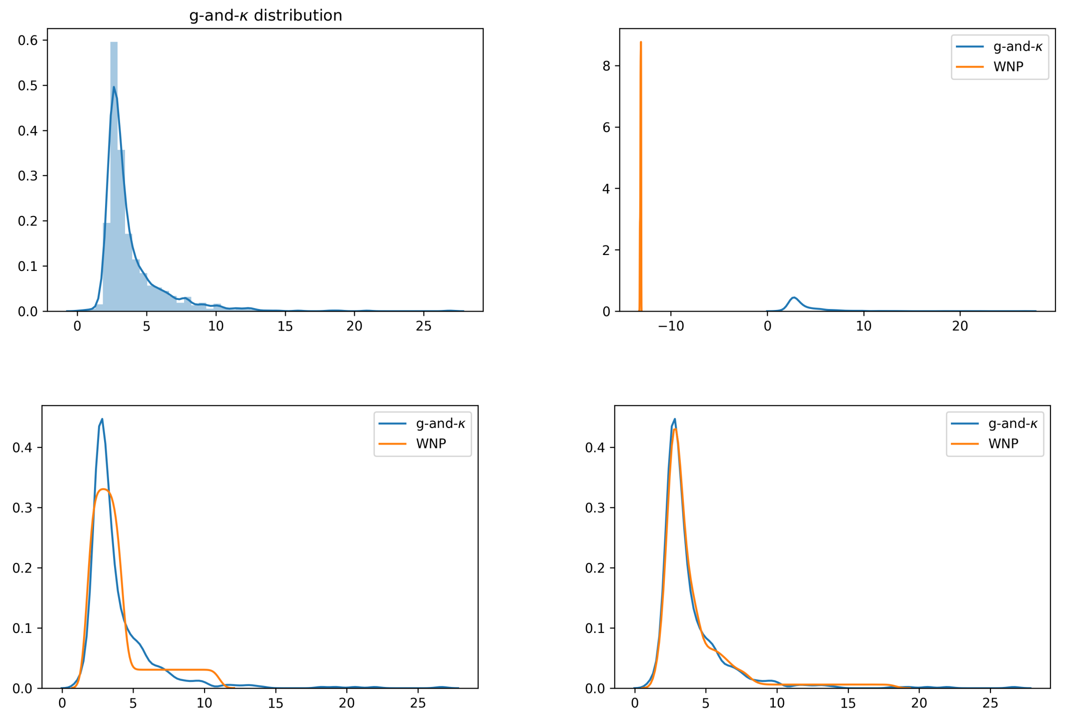

Figure 5.2: WNP g-and-κ distribution results. Top left: Initialization of learned distribution. It would be impossible to calculate the likelihood in this case for traditional NPs both because the model is misspecified and because the likelihood is computationally intractable. Top right: Beginning of the learning process. The output of the WNP is completely disjoint from the sampled g-and-κ distribution. Bottom left: Part way through the learning process (approximately 500 steps) we see the WNP is quickly able to achieve a good fit. Bottom right: Final fit of g-and-κ distribution where WNP learned to model the distribution without relying on the likelihood calculation

finish the optimization process by properly translating the model to achieve the desired fit. We hypothesize this as a function of the g-and-κ topology and leave it as an area for future exploration as it is beyond the scope of this work.

Due to the fact that the likelihood is intractable, if one wished to use this distribution as the model output of a Neural Process, it would be as impossible to train as in the uniform noise model regression task. This is due to the fact, as mentioned above, that standard Neural Processes rely on maximum likelihood as a learning signal. Since Wasserstein Neural Processes rely solely on the divergence between two empirical distributions, they can learn a conditional g-and-κ distribution.

5.3.3 CelebA tiles as super pixels

We now present our final experiment on high-dimensional image data. The purpose of this experiment is to illustrate the ability of WNP to model higher-dimensional distributions where the likelihood might be difficult to calculate or potentially meaningless. The experiment is run on the CelebA dataset.

To distinguish between a large class of prior work and our contribution, we highlight a distinction between common candidate objective functions and the true data distribution. Many frameworks for image prediction, including standard Neural Processes implicitly, involve *reconstruction-based* loss functions that are usually defined as $\|y - \hat{y}\|$. These loss functions compare a target y with the model's output $\hat{y}$ using pixel-wise MSE — but this is known to perform poorly in many cases [36], including translations and brightness variations. More generally, it completely ignores valuable higher-order information in images: pixel intensities are neither independent nor do they share the same variance. Additionally, since images are typically supported on low-dimensional manifolds, the Gaussian assumption of MSE is potentially problematic.

Put simply, pixel-wise losses imply false assumptions and results in a different objective than the one we would truly like to minimize. These false assumptions help explain why (for example) generative image models such as VAEs (based on reconstruction error) result in blurry reconstructions[18, 19, 34], while models such as Wasserstein GANs [1] (based on Wasserstein distances) result in much sharper reconstructions.

Neural Processes use Gaussian distributions over individual pixels. Predictably, this results in blurry predictions, seen in [18, 19, 34]. Analogously to the difference between GANs and VAEs, if we extend these experiments slightly, where instead of using pixel values as context points, we take large tiles of the image as context points, we might expect that we would see sharper reconstructions from WNPs as compared to NPs.

We test this by treating image completion as a high-dimensional regression task. 32×32 images are broken into 4×4 tiles; these tiles are then randomly subsampled. The

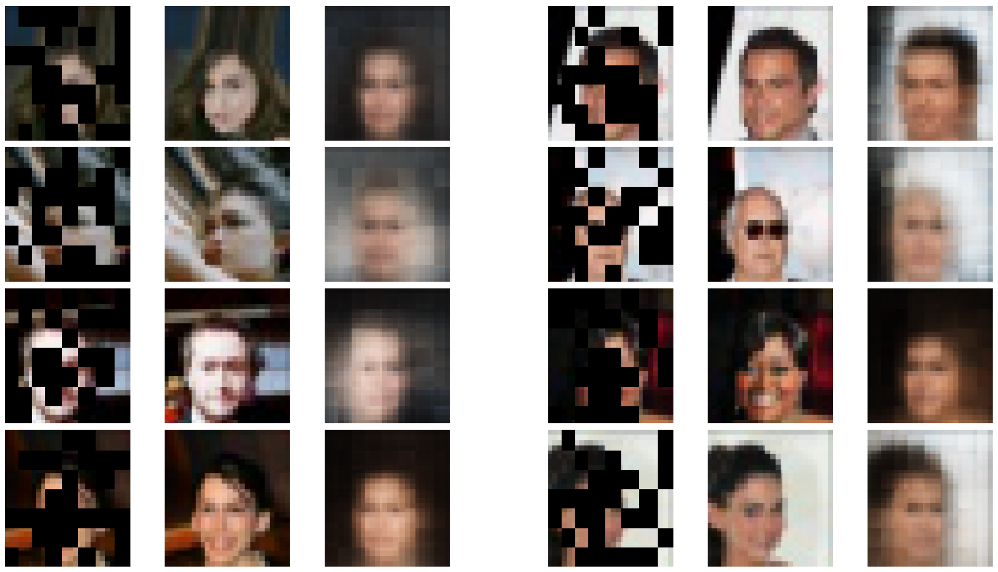

Figure 5.3: On the left: results from Neural Processes. The left-most column shows the sparse observations; the middle column shows ground truth; the right column shows the NP reconstruction. On the right: results for WNPs. See text for discussion.

X inputs are the tile coordinates; the Y outputs are the associated pixels. Given a set of between 4 – 16 tiles, the WNP must predict the remainder of the pixels by predicting entire tiles. The Neural Process is trained on the same data, using the same neural network; the only difference is the loss function.

5.3.4 Discussion

The results of our Celeb-A experiment are shown in (fig. 5.3). Here, the results are not what we expected: traditional Neural Processes and Wasserstein Neural Processes learn similarly blurry output when using tiles as context points instead of pixels. While neither of the images are as high quality as recent GAN work [30], this experiment shows that WNP still can capture background, facial structure, color, as can the traditional NP trained with maximum likelihood.

Chapter 6

Discussion and Future Work

This thesis has explored two specific extensions to the neural process framework. Neural processes allow the learning of a distribution of functions over a set of context and target points. Extending those to graph valued data and using optimal transport expands the class of functions that can be learned.

6.1 Graph Neural Processes

This thesis has introduced Graph Neural Processes, a model that learns a conditional distribution while operating on graph structured data. This model has the capacity to generate uncertainty estimates over outputs, and encode prior knowledge about input data distributions. Additionally, GNP's ability on edge imputation and given potential areas for future exploration was demonstrated.

While the encoder and decoder architectures can be extended significantly by including work from modern deep learning architectural design, this work is a step towards building Bayesian neural networks on arbitrarily graph structured inputs. Additionally, it encourages the learning of abstractions about these structures. In the future, we wish to explore the use of GNPs to inform high-level reasoning and abstraction about fundamentally relational data.

6.2 Wasserstein Neural Processes

This thesis has also explored a synthesis of Neural Processes and Optimal Transport. It was shown that there are desirable classes of functions which Neural Processes either struggle to learn, or cannot learn, but which can be learned by Wasserstein Neural Processes. These WNPs use Wasserstein distance in place of the intrinsic KL divergence used in maximum likelihood and can be tractably implemented using (for example) sliced approximations. Maximum likelihood fails to learn in the case of misspecified models and likelihoods that either don't exist, or are computationally intractable. As a concluding thought, we note that our technical strategy in developing this model was to replace a maximum-likelihood objective with a Wasserstein based objective. This raises the intriguing question: can *all* probabilistic models based on maximum-likelihood be reformulated in terms of Wasserstein divergence? We leave this exciting direction for future work.

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein Generative Adversarial Networks. In *International Conference on Machine Learning*, 2017.

[2] Tom Auld, Andrew W Moore, and Stephen F Gull. Bayesian neural networks for internet traffic classification. *IEEE Transactions on Neural Networks*, 18(1):223–239, 2007.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

[4] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Neural Information Processing Systems*, pages 4502–4510, 2016.

[5] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261*, 2018.

[6] Espen Bernton, Pierre E Jacob, Mathieu Gerber, and Christian P Robert. Approximate bayesian computation with the Wasserstein distance. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2019.

[7] Nicolas Bonnotte. Unidimensional and evolution methods for optimal transportation. *PhD Thesis*, 2013.

[8] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Neural Information Processing Systems*, pages 2787–2795, 2013.

[9] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE*, 34(4):18–42, 2017.

[10] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv:1612.00341*, 2016.

[11] Fan RK Chung and Fan Chung Graham. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

[12] Zhiyong Cui, Kristian Henrickson, Ruimin Ke, and Yinhai Wang. Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting. *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[13] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2292–2300. Curran Associates, Inc., 2013. URL `http://papers.nips.cc/paper/4927-sinkhorn-distances-lightspeed-computation-of-optimal-transport.pdf`.

[14] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, 1991.

[15] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Neural Information Processing Systems*, pages 2224–2232, 2015.

[16] National Center for Advancing Translation Services. Tox21 data challenge, 2014. URL `https://tripod.nih.gov/tox21/challenge/index.jsp`.

[17] Marco Cuturi Gabriel Peyré. *Computational Optimal Transport*. 2018. URL `https://arxiv.org/pdf/1803.00567.pdf`.

[18] M Garnelo, D Rosenbaum, CJ Maddison, T Ramalho, D Saxton, M Shanahan, YW Teh, DJ Rezende, and SM. Eslami. Conditional neural processes. *International Conference of Machine Learning*, 2018.

[19] Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *In International Conference of Machine Learning Workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018.

[20] Aude Genevay, Gabriel Peyre, and Marco Cuturi. Learning Generative Models with Sinkhorn Divergences. In *International Conference on Artificial Intelligence and Statistics*, 2018.

[21] M. Gori, G. Monfardini, and F Scarselli. A new model for learning in graph domains. *IJCNN*, 2:729–734, 2005.

[22] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of Wasserstein GANs. In *Advances in Neural Information Processing Systems*, 2017.

[23] Paul R Halmos. *Measure Theory*, volume 18. Springer, 2013.

[24] Takuo Hamaguchi, Hidekazu Oiwa, Masashi Shimbo, and Yuji Matsumoto. Knowledge transfer for out-of-knowledge-base entities: a graph neural network approach. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 2017.

[25] Christoph Helma, Ross D. King, Stefan Kramer, and Ashwin Srinivasan. The predictive toxicology challenge 2000–2001. *Bioinformatics*, 17(1):107–108, 2001.

[26] Avinash Hindupur. Gan zoo. URL `https://github.com/hindupuravinash/the-gan-zoo`.

[27] Yedid Hoshen. Vain: Attentional multi-agent predictive modeling. In *Neural Information Processing Systems*, pages 2701–2711, 2017.

[28] Mark Huisman. Imputation of missing network data: Some simple procedures. *Journal of Social Structure*, 10, 01 2009. doi: 10.1007/978-1-4614-6170-8_394.

[29] Martinez Jorge and Iglewicz Boris. Some properties of the Tukey g and h family of distributions. *Communications in Statistics-Theory and Methods*, 13(3):353–369, 1984.

[30] Tero Karras, Samuli Laine, and Timo Aila. A Style-Based Generator Architecture for Generative Adversarial Networks. In *Conference on Computer Vision and Pattern Recognition*, 2019.

[31] Jeroen Kazius, Ross McGuire, and Roberta Bursi. Derivation and validation of toxicophores for mutagenicity prediction. *Journal of Medicinal Chemistry*, 48(1):312–320, 2005.

[32] Charles Kemp and Joshua B Tenenbaum. The discovery of structural form. *PNAS*, 105 (31):10687–10692, 2008.

[33] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016. URL `http://graphkernels.cs.tu-dortmund.de`.

[34] Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. In *ICLR*, 2019. URL `https://openreview.net/forum?id=SkE6PjC9KX`.

[35] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

[36] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

[37] T Kipf, E Fetaya, K-C Wang, M Welling, R Zemel, et al. Neural relational inference for interacting systems. *Proceedings of Machine Learning Research*, 2018.

[38] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv:1609.02907*, 2016.

[39] Soheil Kolouri, Yang Zou, and Gustavo K Rohde. Sliced Wasserstein kernels for probability distributions. In *Conference on Computer Vision and Pattern Recognition*, 2016.

[40] Soheil Kolouri, Charles E. Martin, and Gustavo K. Rohde. Sliced-Wasserstein autoencoder: An embarrassingly simple generative model. *CoRR*, abs/1804.01947, 2018. URL `http://arxiv.org/abs/1804.01947`.

[41] Soheil Kolouri, Kimia Nadjahi, Umut Simsekli, Roland Badeau, and Gustavo Rohde. Generalized sliced Wasserstein distances. In *Advances in Neural Information Processing Systems*, 2019.

[42] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *ICLR*, 2016.

[43] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv:1707.01926*, 2017.

[44] Huidong Liu, Xianfeng Gu, and Dimitris Samaras. A two-step computation of the exact GAN Wasserstein distance. In *International Conference of Machine Learning*, 2018.

[45] Christos Louizos, Xiahan Shi, Klamer Schutte, and Max Welling. The functional neural process. In *Advances in Neural Information Processing Systems*, 2019.

[46] Pierre Mahé, Liva Ralaivola, Véronique Stoven, and Jean-Philippe Vert. The pharmacophore kernel for virtual screening with support vector machines. *Journal of Chemical Information and Modeling*, 46(5):2003–2014, 2006.

[47] Gaspard Monge. Mémoire sur la théorie des déblais et des remblais. *Histoire de l'Académie royale des sciences de Paris*, 1781.

[48] Daniel Oñoro-Rubio, Mathias Niepert, Alberto García-Durán, Roberto González, and Roberto J López-Sastre. Representation learning for visual-relational knowledge graphs. *arXiv:1709.02314*, 2017.

[49] Shenghao Qin, Jiacheng Zhu, Jimmy Qin, Wenshuo Wang, and Ding Zhao. Recurrent attentive neural process for sequential data. *arXiv preprint arXiv:1910.09323*, 2019.

[50] David Raposo, Adam Santoro, David Barrett, Razvan Pascanu, Timothy Lillicrap, and Peter Battaglia. Discovering objects and their relations from entangled scene representations. *arXiv:1702.05068*, 2017.

[51] Kaspar Riesen and Horst Bunke. Iam graph database repository for graph based pattern recognition and machine learning. In *Joint IAPR International Workshops on (SPR) and (SSPR)*, pages 287–297. Springer, 2008.

[52] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph Networks as Learnable Physics Engines for Inference and Control. In *International Conference on Machine Learning*, 2018.

[53] Filippo Santambrogio. Optimal transport for applied mathematicians, 2015.

[54] Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Neural Information Processing Systems*, pages 4967–4976. Curran Associates, Inc., 2017. URL `http://papers.nips.cc/paper/7082-a-simple-neural-network-module-for-relational-reasoning.pdf`.

[55] Victor Garcia Satorras and Joan Bruna Estrach. Few-shot learning with graph neural networks. In *ICLR*, 2018. URL `https://openreview.net/forum?id=BJj6qGbRW`.

[56] F. Scarselli, S. L. Yong, M. Gori, M. abd Hagenbuchner, A. C. Tsoi, and M. Maggini. Graph neural networks for ranking web pages. *IEEE/WIC/ACM International Conference on Web Intelligence*, pages 666–672, 2005.

[57] Scott A Sisson, Yanan Fan, and Mark Beaumont. *Handbook of approximate Bayesian computation*. Chapman and Hall/CRC, 2018.

[58] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In *Neural Information Processing Systems*, pages 2244–2252, 2016.

[59] J.W. Tukey. Modern techniques in data analysis. In *Proceedings of the NSF-Sponsored Regional Research Conference, volume 7, Dartmouth, Massachusetts. Southern Massachusetts University.*, 1977.

[60] Sjoerd van Steenkiste, Michael Chang, Klaus Greff, and Jürgen Schmidhuber. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. *arXiv:1802.10353*, 2018.

[61] C. Villani. *Optimal Transport: Old and New*. Grundlehren der mathematischen Wissenschaften. Springer Berlin Heidelberg, 2008. ISBN 9783540710509. URL `https://books.google.com/books?id=hV8o5R7_5tkC`.

[62] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph CNN for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5), 2019.

[63] Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. In *Neural Information Processing Systems*, pages 4539–4547, 2017.

[64] Wikipedia contributors. Transportation theory (mathematics) — Wikipedia, the free encyclopedia, 2019. URL `https://en.wikipedia.org/wiki/Transportation_theory_(mathematics)`. [Online; accessed 11-Nov-2019].

[65] Yingxue Zhang, Soumyasundar Pal, Mark Coates, and Deniz Ustebay. Bayesian graph convolutional neural networks for semi-supervised classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 2019.

[66] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *https://arxiv.org/abs/1812.08434*, 2019.