Unifying Transformers and Convolutional Neural Processes

Lakee Sivaraya (ls914)

May 9, 2024

Contents

Chapter 1

Introduction

1.1 Motivation

Machine learning models have been immensely successful in variety of applications to generate predictions in data-driven driven domains such as computer vision, robotics, weather forecasting. While the success of these models is undeniable, we lack the ability to understand the uncertaincy in the predictions made by most of the State-of-the-Art models. This is a major drawback in the deployment of these models in real-world applications, for example, in weather forecasting, it is important to know the uncertaincy in the prediction of the weather as this information is arguably as valuable as the prediction itself. In this work, we aim to implement a model is **Uncertainty-Aware** whilst also posessing further desirable properties.

1.2 Desirable Properties

Ontop of being uncertaincy aware, we would like to insert some desirable inductive biases that help the model to generalize better and be more interpretable. We would like the model to be able to:

Flexible: *The model should be able to work on a variety of data types.* As long as a data point can be represented as a vector, the model should be able to work on it. This allows the model to be used in a variety of applications and domains.

Scalable: *The model should be able to work on large datasets.* The model can be trained on large datasets and should be able to make predictions on large datasets. There should be no limit on the size of the dataset that the model can work on which is not the case with many traditional models such as LLMs. Another aspect of scalability is the ability to work on high-dimensional data and large data sets with good computational efficiency.

Permutation Invariant: *The prediction of the model should not change if the order of the input data is changed.* When each datapoint contains the information about input and output pairs, the model should not care about the order in which they are fed into the model.

For example, in the case of a weather forecasting model using data from multiple weather stations, the model should not care about the order in which the data from the weather stations is fed into the model, thus making the model permutation invariant.

Translation Equivariant: *Shifting the input data by a constant amount should result in a constant shift in the predictions.* For example, in the case of a weather forecasting model, if the data from the weather stations is shifted by a constant amount, the model should be able to predict the same shift in the weather forecast.

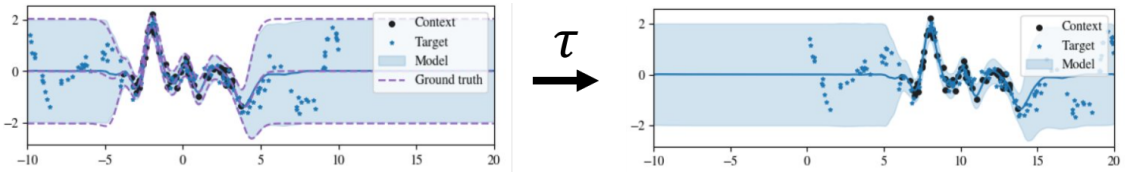

Figure 1.2.1: The Translation Equivariant property. Consider the given prediction on the black datapoints on the left. If the input is shifted by a constant amount, the prediction should also shift by the same amount (right).

TODO

Clearer figure

Figure 1.2.1 illustrates the Translation Equivariant (TE) property on a simple 1D dataset. If the model, f is TE then the following holds:

$$f : \boldsymbol{x} \to (\boldsymbol{x}, \hat{\boldsymbol{y}}) \tag{1.2.1}$$
$$f : \boldsymbol{x} + \boldsymbol{c} \to (\boldsymbol{x} + \boldsymbol{c}, \hat{\boldsymbol{y}}) \tag{1.2.2}$$

Where $\boldsymbol{x}$ is the input and $\hat{\boldsymbol{y}}$ is the output and $\boldsymbol{c}$ is a constant shift in the input.

Off-the-Grid Generalization: *The model should be able to work on off-the-grid data points.* Off-the-grid data points are the data points that are not in a regular gridded structure, for example, images that are missing pixel values are off-the-grid. Traditional models like Convolutional Neural Networks (CNNs) are not able to work on off-the-grid data points since they require a regular structure to apply the convolution operation. By making the model off-the-grid generalizable, we can create models that can work on many different types of data and easily handle missing data points. It also allows the model to generalize to regions of the input space that it has not seen during training. Tasks like image inpainting, where the model is required to fill in missing pixels in an image, can benefit from this property.

TODO

Add a figure to illustrate off-the-grid generalization

1.3 Aims and Objectives

Neural Processes (NPs) [Gar+18b] are a class of models that satisfy the above properties. The framework undermining NPs is general purpose and thus it can be modified with a variety of neural network architectures.

In this work, we aim to implement and compare two different neural network architectures for Neural Processes, the first being based on a Convolutional Neural Network (CNN) called Convolutional Neural Processes (ConvNP) and the second being based on a Transformer architecture called Transformer Neural Processes (TNP). We aim to compare the two models on a variety of tasks and datasets to see how they perform in terms of generalization, scalability, and uncertainty estimation.

Furthermore we investiage how the TNP can be optmized to achieve better performance. We then investigate new Transformer architectures that have better computational efficiency compared to the original Transformer architecture.

Our end goal is to better understand the properties of these models and how they can be used in real-world applications and figure out the best practices for using these models in different scenarios.

Chapter 2

Neural Processes

2.1 Neural Processes

Equation 1.1 Neural Processes are an expansion of Gaussian Processes using Neural Networks introduced in [Gar+18b] they are trained on a context dataset which is. Neural processes are a meta-learning algorithm that can be used for few-shot learning. They are trained on a context dataset which is a set of input-output pairs $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$.

2.1.1 Meta Learning

Meta-Learning is a method of ML where the model is trained on a set of tasks and then tested on a new task. The goal is to learn a model that can learn new tasks quickly'', i.e. 'learning to learn'. These come under the terminology of few-shot learning where the model is trained on a small number of examples (more specifically, we can say N-way-K-shot learning where N is the number of classes and K is the number of examples per class).

Consider a set of datasets $\mathcal{D} = \{\mathcal{D}_i\}_{i=1}^{D}$ where each dataset $\mathcal{D}_i$ is a set of input-output pairs $\mathcal{D}_i = \{(x_{ij}, y_{ij})\}_{j=1}^{N_i}$. The goal of meta-learning is to learn a model f that can learn a new task $\mathcal{D}_{N+1}$ with a small number of examples, i.e. few-shot learning.

2.1.2 Architecture

A meta-dataset is split into two sets, the context set and the target set (also called the query set) where the sets are disjoint $\mathcal{D} = \mathcal{C} \cup \mathcal{T}$ and $\mathcal{C} \cap \mathcal{T} = \emptyset$. The model is trained on the context set and then tested on the target set to see how well it can generalize to new tasks. In essence, our task is to predict the outputs for the target conditioned on the training of the context set.

To achieve this we used a neural network (usually MLP) to encode the context set into embeddings.

$$\boldsymbol{r}(\mathcal{C}_i) = \text{Enc}_\theta(\mathcal{C}_i) = \text{Enc}_\theta(\{\boldsymbol{x}_i^{(c)}, \boldsymbol{y}_i^{(c)}\}) \tag{2.1.1}$$

Where $\boldsymbol{r}$ is the embedding of the context set $\mathcal{C}_i$ and θ are the parameters of the encoder. Then we aggregate the embeddings of the context sets to get a global representation of the dataset.

$$\boldsymbol{R}(\mathcal{C}) = \text{Agg}(\{\boldsymbol{r}(\mathcal{C}_i)\}_{i=1}^{D}) \tag{2.1.2}$$

Typically the aggregation function is a simple summation of the embeddings. The global representation $\boldsymbol{R}$ is then used to condition the decoder to predict the outputs of the target set $\mathcal{T} = \{\boldsymbol{x}_i^{(t)}\}$ to give us a posterior distribution over the outputs $\boldsymbol{y_i^{(t)}}$.

$$p(\boldsymbol{y}_i^{(t)}|\boldsymbol{x}_i^{(t)}, \mathcal{C}) = \text{Dec}_\theta(\boldsymbol{x}_i^{(t)}, \boldsymbol{R}(\mathcal{C})) \tag{2.1.3}$$

The overall architecture is shown in Figure 2.1.1.

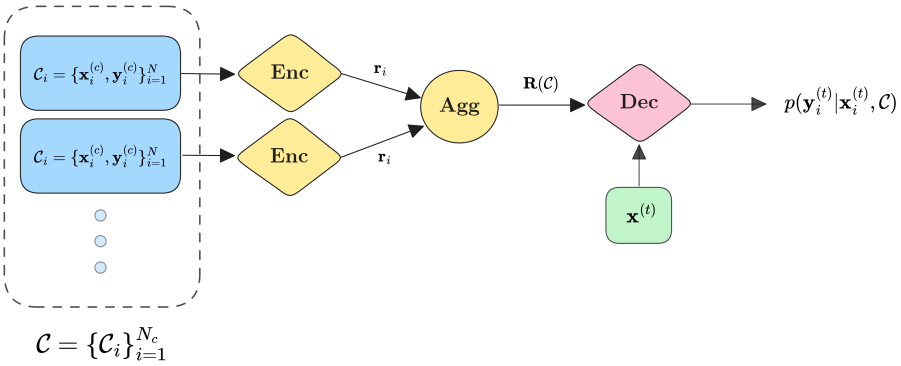

Figure 2.1.1: CNP Architecture

2.1.3 Neural Processes vs Gaussian Processes

From this, it is clear that NPs achieve a *prediction* map, π which maps the context set to the target set into a posterior distribution over the outputs.

$$\pi : \mathcal{C} \times \mathcal{X}^{(t)} \to \mathcal{P}(\mathcal{Y}) \tag{2.1.4}$$

This illustrates the similarity to Gaussian Processes (GP) [RW06] which perform the same underlying task of predicting the outputs of a target set conditioned on a context set. The difference is that GPs use a kernel function to compute the covariance between the context and target sets whereas NPs use a neural network to compute the covariance.

$$\text{GP} : \pi \to \mathcal{N}(m(\mathcal{C}, \boldsymbol{x}^{(t)})), k(\boldsymbol{x}^{(c)}, \boldsymbol{x}^{(t)})) \tag{2.1.5}$$

$$\text{NP} : \pi \to \text{Dec}_\theta(\boldsymbol{x}^{(t)}, \text{Agg}(\{\text{Enc}_\theta(\mathcal{C}_i)\}_{i=1}^{D})) \tag{2.1.6}$$

2.1.4 Conditonal Neural Processes

Conditional Neural Processes (CNPs) [Gar+18a] was one of the two original Neural Processes introduced by Garnelo et al. in 2018. Using the architecture framework described above, CNPs can be described as follows.

$$\begin{aligned} \text{Enc}_{\theta_e} &= \text{MLP}_{\theta_e} \\ \text{Agg} &= \text{Sum} \\ \text{Dec}_{\theta_d} &= \text{MLP}_{\theta_d} \end{aligned}$$

The encoding and summation is an implementation of the 'DeepSet' architecture [Zah+18] which is a neural network that is **Permutation Invariant** (PI) due to the summation operation. This means that the order of the input does not matter.

CNPs make the strong assumption that the posterior distribution *factorizes* over the target points. This means that the posterior distribution over the target points is independent of each other.

$$p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, \mathcal{C}) \overset{(a)}{=} \prod_{i=1}^{N_t} p(y_i^{(t)}|x_i^{(t)}, \boldsymbol{R}(\mathcal{C})) \tag{2.1.7}$$

$$\overset{(b)}{=} \prod_{i=1}^{N_t} \mathcal{N}(y_i^{(t)}|\mu_i, \sigma_i^2) \tag{2.1.8}$$

$$\overset{(c)}{=} \mathcal{N}(\boldsymbol{y}^{(t)}|\boldsymbol{\mu}(\boldsymbol{x}^{(t)}, \mathcal{C}), \boldsymbol{\Sigma}(\boldsymbol{x}^{(t)}, \mathcal{C})) \tag{2.1.9}$$

The benefit of this factorization assumption (a) is that the model can scale linearly with the number of target points with a tractable likelihood. However, this assumption means **CNPs are unable to generate coherent sample paths, they are only able to produce distributions over the target points.** Furthermore, we need to select a marginal likelihood for the distribution (b) which is usually a Heteroscedastic Gaussian Likelihood (Gaussian with a variance that varies with the input) [Gar+18a]. This also adds a further assumption as we have to select a likelihood for the distribution which may not be appropriate for the data we are modeling.

Since the product of Gaussians is a Gaussian (c) the model learns a mean and variance for each target point (though typically we learn the log variance to ensure it is positive), in this way, the model decoder can be interpreted as outputting a function of mean and variance for each target point.

As the likelihood is a Gaussian, the model can be trained using simple maximum likelihood estimation (MLE) by minimizing the negative log-likelihood (NLL) of the target points.

2.1.5 Latent Neural Processes

Neural Processes better named 'Latent Neural Processes' are an extension of CNPs that can generate coherent sample paths and are not restricted to a specific likelihood. They can do this by learning a latent representation of the context set $\mathcal{C}$ which is then used to condition the decoder. We will call this latent representation $\mathbf{z}$ (instead of $\boldsymbol{R}$) to avoid confusion with the non-latent global representation $\boldsymbol{R}$ used in CNPs).

The encoder learns a *distribution* over the latent representation $\mathbf{z}$ of the context set $\mathcal{C}$. Then the decoder learns a distribution over the target outputs $\boldsymbol{y}^{(t)}$ conditioned on the latent representation $\mathbf{z}$ and the target inputs $\boldsymbol{x}^{(t)}$.

$$
\begin{aligned}
p(\mathbf{z}|\mathcal{C}) &= \text{Enc}_\theta(\mathcal{C}) \\
p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, \mathbf{z}) &= \text{Dec}_\theta(\boldsymbol{x}^{(t)}, \mathbf{z})
\end{aligned}
$$

The full model is shown in Figure 2.1.2.

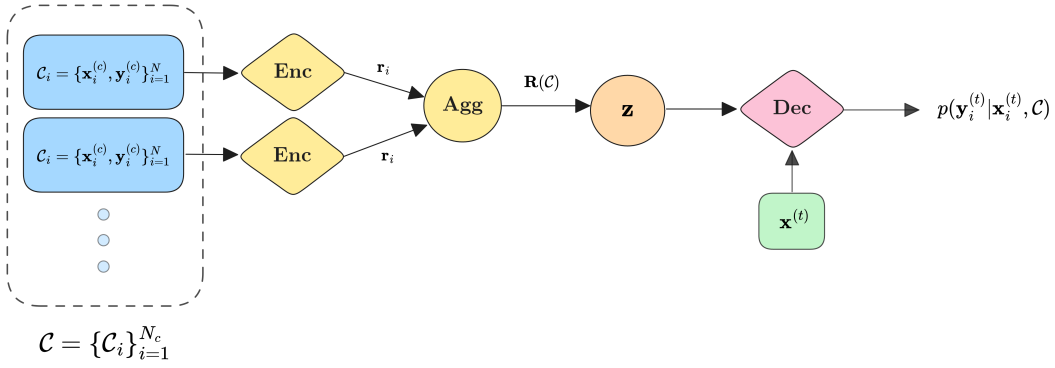

Figure 2.1.2: LNP Architecture

The likelihood of the target points is then given by the marginal likelihood of the latent representation $\mathbf{z}$.

$$
\begin{aligned}
p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, \mathcal{C}) &= \int p(\mathbf{z}|\mathcal{C}) p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, \mathbf{z}) d\mathbf{z} && (2.1.10) \\
&\overset{(a)}{=} \int p(\mathbf{z}|\mathcal{C}) \left(\prod_{i=1}^{N_t} p(y_i^{(t)}|x_i^{(t)}, \mathbf{z}) \right) d\mathbf{z} && (2.1.11) \\
&\overset{(b)}{=} \int p(\mathbf{z}|\mathcal{C}) \left(\prod_{i=1}^{N_t} \mathcal{N}(y_i^{(t)}|\mu_i, \sigma_i^2) \right) d\mathbf{z} && (2.1.12)
\end{aligned}
$$

We can see that we express the conditional distribution over the target points conditioned on the latent representation $\mathbf{z}$ as a factorized (a) product of Gaussians (b), this may seem

like we are making the same factorization assumption as CNPs. However, the difference is that we are not making this assumption conditioned on the latent variable instead of the context set. This integral models an *infinite mixture* of Gaussians which allows us to model *any* distribution over the target points. The downside is that the likelihood is intractable and we need to use approximate inference methods such as Variational Inference (VI) or Sample Estimation using Monte Carlo (MC) methods to train the model which typically leads to biased estimates of the likelihood with high variance, thus we require more samples to train the model.

Bibliography

[Gar+18a] Marta Garnelo et al. *Conditional Neural Processes*. 2018. arXiv: 1807.01613 [cs.LG].

[Gar+18b] Marta Garnelo et al. *Neural Processes*. 2018. arXiv: 1807.01622 [cs.LG].

[RW06] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, 2006, pp. I–XVIII, 1–248. ISBN: 026218253X.

[Zah+18] Manzil Zaheer et al. *Deep Sets*. 2018. arXiv: 1703.06114 [cs.LG].