0.1 Introduction

Neural Process (NP) is a meta-learning framework introduced in [Garnelo, Rosenbaum, et al. 2018; Garnelo, Schwarz, et al. 2018] that can be used for few-shot uncertainty aware meta learning. There exists two variants of the Neural Process, the Conditional Neural Process (CNP) and the Latent Neural Process (LNP), whilst we will discuss the differences between the two in this chapter, we will focus on the CNP for the majority of the project and hence we will implicitly refer to CNP as NP.

The main idea behind Neural Processes is to learn a distribution over the input locations we want to predict conditioned on the training data. In the NP literature we refer the training data as the context set and the input locations we want to predict the output for as the target set. The model is trained on a set of context-target pairs and then tested on a new set of context-target pairs to see how well it can generalize to new target sets.

0.2 Architecture

0.2.1 Conditional Neural Processes

Conditional Neural Processes (CNPs) [Garnelo, Rosenbaum, et al. 2018] was one of the two original Neural Processes introduced by Garnelo et al. in 2018. The general framework for a CNP requires us to take a context set $\mathcal{C} = \{\mathcal{C}_i\}_{i=1}^{N_c}$ where each context point $\mathcal{C}_i$ is a input-output pair $\mathcal{C}_i = (\boldsymbol{x}_i^{(c)}, \boldsymbol{y}_i^{(c)})$ and a target set $\mathcal{T} = \{\boldsymbol{x}_i^{(t)}\}_{i=1}^{N_t}$ where each target point $\boldsymbol{x}_i^{(t)}$ is an input point we want to predict the output for.

We firstly encode each data point in the context set $\mathcal{C}_i$ into an embedding using an encoder network.

$$\boldsymbol{r}(\mathcal{C}_i) = \text{Enc}_\theta(\mathcal{C}_i) = \text{Enc}_\theta([\boldsymbol{x}_i^{(c)}, \boldsymbol{y}_i^{(c)}]) \tag{0.2.1}$$

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

$$\boldsymbol{R}(\mathcal{C}) = \text{Process}(\{\boldsymbol{r}(\mathcal{C}_i)\}_{i=1}^{D}) \tag{0.2.2}$$

We require this 'processing' to be **permutation invariant**, so typically it 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 to give us a posterior distribution over the outputs $\boldsymbol{y}_i^{(t)}$.

$$p(\boldsymbol{y}_i^{(t)} | \boldsymbol{x}_i^{(t)}, \mathcal{C}) = \text{Dec}_\theta(\boldsymbol{x}_i^{(t)}, \boldsymbol{R}(\mathcal{C})) \tag{0.2.3}$$

The overall architecture is shown in Figure 0.2.1.

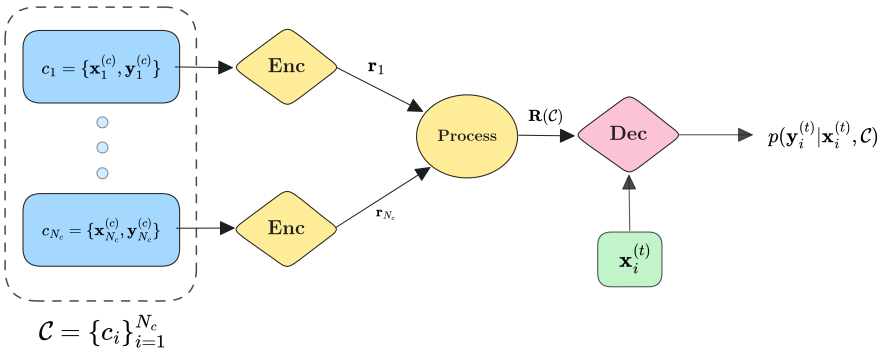

Figure 0.2.1: CNP Architecture: The model vectorizes each individual data point $\mathcal{C}_i$ in the context set $\mathcal{C}$ and then processes/aggregates them to obtain a global representation $\boldsymbol{R}(\mathcal{C})$ which is then used to condition the decoder to predict a distribution over the target points $\boldsymbol{y}^{(t)}$.

In the original CNP paper, the encoder and decoder are implemented as simple Multi-Layer Perceptrons (MLPs) and the processing is implemented as a mean operation, this happens to be an implementation off the 'DeepSet' architecture [Zaheer et al. 2018].

Importantly, 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.

$$p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, \mathcal{C}) \overset{(a)}{=} \prod_{i=1}^{N_t} p(y_i^{(t)}|x_i^{(t)}, \boldsymbol{R}(\mathcal{C})) \overset{(b)}{=} \prod_{i=1}^{N_t} \mathcal{N}(y_i^{(t)}|\mu_i, \sigma_i^2) \tag{0.2.4}$$

$$\tag{0.2.5}$$

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) [Garnelo, Rosenbaum, et al. 2018]. This adds an assumption as we have to select a likelihood for the distribution which may not be appropriate for the data we are modeling.

As the likelihood is a Gaussian, the model can be trained using simple maximum likelihood estimation (MLE) by maximizing the negative log-likelihood ($\mathcal{L}$) of the target points.

$$\mathcal{L} = \mathbb{E}_{(\mathcal{C},\mathcal{T})}\left[-\sum_{i=1}^{|\mathcal{T}|} \log p(y_i^{(t)}|x_i^{(t)}, \mathcal{C})\right] \tag{0.2.6}$$

0.3 Performance of Vanilla NP

Whilst the Vanilla CNP using DeepSets is flexible and scalable, in reality it is unable to perform well on more complicated and higher dimensional data since the model is unable to learn a good representation of the data using a simple MLP and summation operation.

Could we replace the encoder and decoder with more powerful networks? And if so, what would be the best architecture to use? We aim to answer in this project by exploring the use of a Convolutional Neural Network (CNN) and a Transformer as encoders of our NP. CNNs and Transformers have been shown to perform well on a variety of tasks and at scale, thus we hypothesize that they will be able to learn a better representation of the context set and thus improve the performance of the NP. Both are bound to have their unique advantages and disadvantages which we will explore in the following chapters by firstly exploring the Convolutional Neural Process (ConvCNP) and then the Transformer Neural Process (TNP).