0.1 Introduction

Neural Process (NP) is a meta-learning framework introduced in [Gar+18a; Gar+18b] 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 tasks.

0.2 Architecture

0.2.1 Conditional Neural Processes

Conditional Neural Processes (CNPs) [Gar+18a] 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.

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

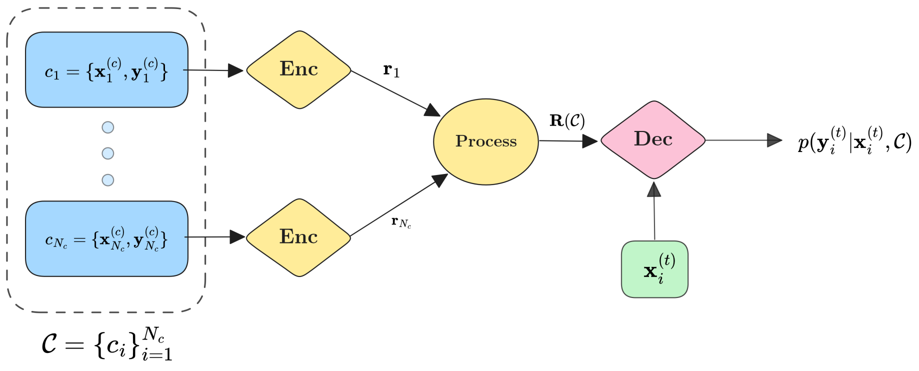
Where $\boldsymbol{r}$ is the embedding of the context set $\mathcal{C}_i$ and θ are the parameters of the encoder. Then we process the embeddings of the context sets to get a global representation of the dataset.

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

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.



$c_1 = \{\mathbf{x}_1^{(c)}, \mathbf{y}_1^{(c)}\}$
$c_{N_c} = \{\mathbf{x}_{N_c}^{(c)}, \mathbf{y}_{N_c}^{(c)}\}$
Enc
Enc
$\mathbf{r}_1$
$\mathbf{r}_{N_c}$
Process
$\mathbf{R}(\mathcal{C})$
Dec
$\mathbf{x}_i^{(t)}$
$p(\mathbf{y}_i^{(t)}|\mathbf{x}_i^{(t)}, \mathcal{C})$
$\mathcal{C} = \{c_i\}_{i=1}^{N_c}$

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 [Zah+18].

$$\begin{aligned} \mathrm{Enc}_{\theta_e} &= \mathrm{MLP}_{\theta_e} \\ \mathrm{Process} &= \mathrm{Mean} \\ \mathrm{Dec}_{\theta_d} &= \mathrm{MLP}_{\theta_d} \end{aligned}$$

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.

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

$$\overset{(b)}{=} \prod_{i=1}^{N_t} \mathcal{N}(y_i^{(t)}|\mu_i, \sigma_i^2) \tag{0.2.5}$$

$$\overset{(c)}{=} \mathcal{N}(\boldsymbol{y}^{(t)}|\boldsymbol{\mu}(\boldsymbol{x}^{(t)}, \mathcal{C}), \boldsymbol{\Sigma}(\boldsymbol{x}^{(t)}, \mathcal{C})) \tag{0.2.6}$$

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.

0.2.2 Latent Neural Processes

The ‘Latent Neural Processes’ is the 2nd variant of NP (introduced in [Gar+18b]) which 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 0.2.2.

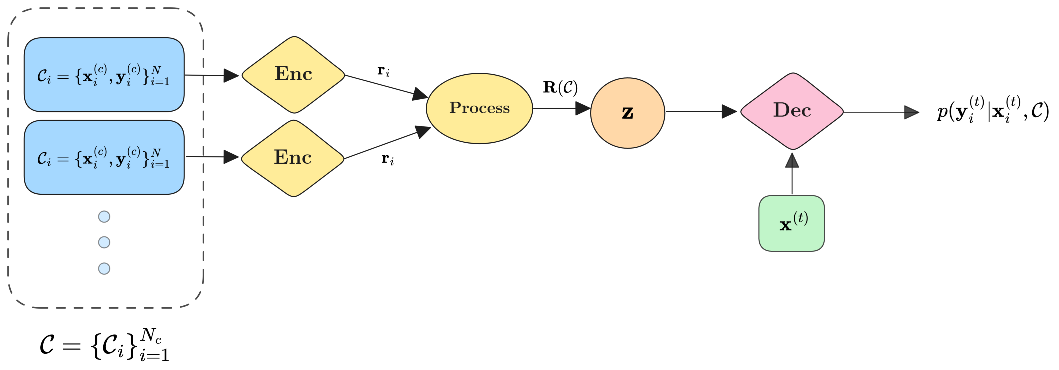

$\mathcal{C}_i = \{\mathbf{x}_i^{(c)}, \mathbf{y}_i^{(c)}\}_{i=1}^N$
$\mathcal{C}_i = \{\mathbf{x}_i^{(c)}, \mathbf{y}_i^{(c)}\}_{i=1}^N$
Enc
Enc
$\mathbf{r}_i$
$\mathbf{r}_i$
Process
$\mathbf{R}(\mathcal{C})$
$\mathbf{z}$
Dec
$p(\mathbf{y}_i^{(t)}|\mathbf{x}_i^{(t)}, \mathcal{C})$
$\mathbf{x}^{(t)}$
$\mathcal{C} = \{\mathcal{C}_i\}_{i=1}^{N_c}$

Figure 0.2.2: LNP Architecture

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

$$p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, \mathcal{C}) = \int p(\mathbf{z}|\mathcal{C}) p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, \mathbf{z}) d\mathbf{z} \tag{0.2.7}$$

$$\overset{(a)}{=} \int p(\mathbf{z}|\mathcal{C}) \left(\prod_{i=1}^{N_t} p(y_i^{(t)}|x_i^{(t)}, \mathbf{z}) \right) d\mathbf{z} \tag{0.2.8}$$

$$\overset{(b)}{=} \int p(\mathbf{z}|\mathcal{C}) \left(\prod_{i=1}^{N_t} \mathcal{N}(y_i^{(t)}|\mu_i, \sigma_i^2) \right) d\mathbf{z} \tag{0.2.9}$$

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.

Though the LNP has many uses, **in this report we will strictly focus on the CNP** as it is more tractable and easier to train. From this point on, when we refer to Neural Processes we will be referring to the Conditional Neural Process (CNP) unless otherwise stated.

0.3 Neural Processes vs Gaussian Processes

TODO

HERE

0.4 Performance of Vanilla NP

Whilst Neural Processes are very flexible and have the ability to scale, in reality due to the simple archiecture of the encoding and aggregation stage, they are unable to perform effectively in more complicated and higher dimensional data. This is because the model is unable to learn a good representation of the context set using a simple MLP and summation operation.

The benefit of the NP is we can replace the encoder and decoder with more powerful models such as Convolutional Neural Networks (CNNs) or Transformers to learn a better representation of the context set. This allows us to scale the model using well known architectures that have been shown to perform well on a variety of tasks and at scale. Both CNN and Transformer based NPs 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).