0.1 Background

When training AI models we want them to be able to generalize to unseen data and learn patterns in the data, for example when analysing 2D climate data, we would want the model to learn relations between the data in the spatio-temporal domain. In other words, we want the model to be able to learn the underlying function of the data without depending on the specific location. This property is called *translation equivariance* and is a property that is present in many real world problems. Translation Equivariance (TE) states that if our inputs are shifted in the input space, the output should be shifted in the output space in the same way. CNPs do not have this property, the ConvCNP aims to add this property to the CNP by utilizing the TE property of CNNs.

0.2 Model

As previously stated, ConvCNPs [Gor+20] utilize CNNs, more specifically the UNet architecture. However, a few hurdles are preventing us from directly plugging data into a CNN.

0.2.1 Encoder

CNNs operate on **on-grid data**, meaning that the data is on a regular grid, for example, an image. However, the data we are working with is sometimes **off-grid data**, meaning that the data is not on a regular grid, for example time series dataset with irregular timestamps. Furthermore to 'bake-in' the TE property, we need to be able to shift the data in the input space and the output space in the same way. This is not trivial when using standard vector representations of the data, as the data is not on a regular grid. [Gor+20] propose a solution to this problem by using **functional embeddings** to model the data. Functional embeddings are a way to represent data as a function that has a trivial translation equivariance property.

These functional embeddings are created using the **SetConv** operation. The SetConv operation is a generalization of the convolution operation that operates on sets of data, it takes a set of input-output pairs and outputs a continuous function. The SetConv operation is defined as follows:

$$\text{SetConv}(\{x_i, y_i\}_{i=1}^N)(x) = \sum_{i=1}^{N} [1, y_i]^T \mathcal{K}(x - x_i) \tag{0.2.1}$$

Where $\mathcal{K}$ is a kernel function that measures the distance between the queryable x and the datapoint x_i.

This operation has some key properties:

- We append 1 to output y_i when computing the SetConv, this acts as a flag to the model so that it knows which data points are observed and which are not. Say we have a datapoint of $y_i = 0$, then if we did not append the 1, the model would not be able to distinguish between an observed datapoint and an unobserved datapoint (as both would be 0).

- The 'weight' of the Kernal depends only on the relative distance between points on the input space which means that the model is translation equivariant.
- The summation over the datapoints naturally introduces **Permutation Invariance** to the model, meaning that the order of the datapoints does not matter.

In [Gor+20], they refer to the addition of the 1 as the **denisty channel**, this channel is used to distinguish between observed and unobserved data points.

Now that we have a function representation of the context set, we need to sample it/discretize it at **evenly** spaced points. Thus converting the data into a **on-grid** format. Now it can be fed into a CNN. Now that we have a CNN that operates on the data, we need to convert it back to a continuous function. This is done again by using the **SetConv** operation. The final output of the encoder is a continuous function that represents the context set which can be queried at any point in the input space using the target set.

$$R(x_t) = \text{SetConv}(\text{CNN}(\{\text{SetConv}(\mathcal{C})(x_d)\}_{d=1}^{D}))(x_t) \tag{0.2.2}$$

0.3 Decoder

The decoder is a simple MLP that takes the output of the encoder and the target set as input and outputs the mean and variance of the predictive distribution. The decoder is defined as follows:

$$\mu(x_t), \sigma^2(x_t) = \text{MLP}(R(x_t)) \tag{0.3.1}$$

Bibliography

[Gor+20] Jonathan Gordon et al. *Convolutional Conditional Neural Processes*. 2020. arXiv: 1910.13556 [stat.ML].