

0.1 Vanilla Transformer Neural Process

An Attention based encoder for the Neural Process was investigated in [Kim et al. 2019], though the results were impressive, the model fails to perform at larger scale and ‘tends to make overconfident predictions and have poor performance on sequential decision-making problems’ [Nguyen and Grover 2023]. It is natural to think that the Transformer [Vaswani et al. 2017] could be used to improve the performance of the Neural Process, as Transformers have been proven to effectively model large scale data using attention mechanisms. [Nguyen and Grover 2023] introduced the Transformer Neural Process (TNP) which uses an *encoder-only* Transformer to learn the relationships between the context and target points via self-attention with appropriate masking.

0.1.1 Model Architecture

Similar to the standard Neural Process architecture we are required to encode data points within the context set into a vector representation, in language modelling literature we refer to this as tokenization. The tokenization is done by using a simple Multi-Layer Perceptron (MLP) to encode the data points into vector tokens with a configurable token dimension, D_{em} .

Where the TNP differs is that it also encodes the target points which is padded with zeros to represent the lack of value for the target data points, then both context and target tokens are passed into the transformer at the same time instead of computing a context representation and then passing the target points as in the standard NP.

A flag bit is introduced into the tokens to indicate whether the token is a context or target token. Consider the context dataset $\mathcal{C} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N_c}$ and target set $\mathcal{T} = \{(\mathbf{x}_i)\}_{i=N_c+1}^N$, the model will encode a data point into a token \mathbf{t}_i as follows:

$$\mathbf{t}_i = \begin{cases} \text{MLP}(\text{cat}[\mathbf{x}_i, 0, \mathbf{y}_i]) & \text{if } i \leq N_c \\ \text{MLP}(\text{cat}[\mathbf{x}_i, 1, \mathbf{0}]) & \text{if } i > N_c \end{cases}$$

Where we use a flag bit of 0 to indicate a context token and 1 to indicate a target token, cat is the concatenation operation and $\mathbf{0}$ is a vector of zeros of the same dimension as \mathbf{y}_i .

Now that the data points are tokenized we can pass them through the Transformer encoder to learn the relationships between the context and target points. Importantly the Transformer encoder is masked such that the target tokens can only attend to the context tokens and previous target tokens. Alternatively one perform self attention on the context tokens then cross attention between the context and target tokens giving a more efficient implementation [Feng et al. 2022], we use this implementation in our experiments.

We can now pass the tokens through the Transformer encoder layer several times to learn the relationships between the context and target points and generate a vector output. The output of the Transformer encoder is then passed through a Multi-Layer Perceptron (MLP) to generate the mean and variance of the predictive distribution of the target points.

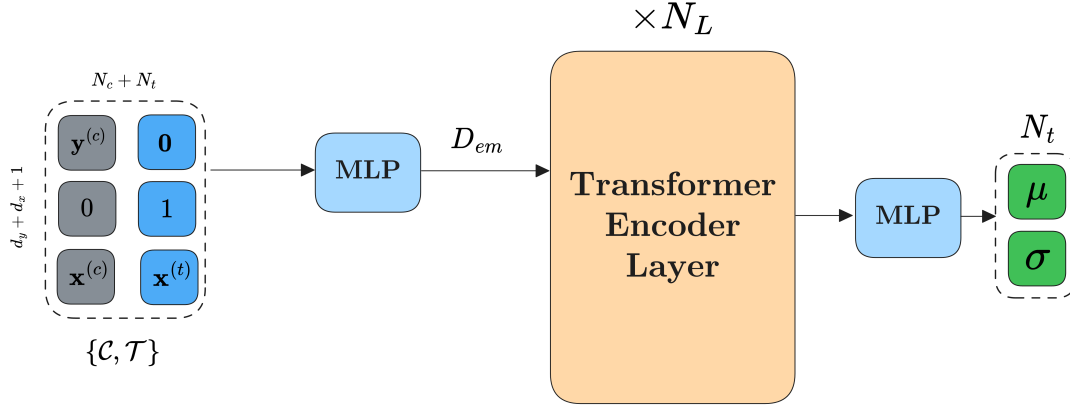


Figure 0.1.1: Vanilla Transformer Neural Process Architecture. All the context and target sets are tokenized and passed through the Transformer Layer. The output of the Transformer Layer is passed through an MLP to generate the mean and variance of the predictive distribution.

0.1.2 Performance

One of the key benefits of Transformers (and attention) is the ability to have a global view of the data, analogous to an ‘infinite receptive field’ in Convolutional Neural Networks. This allows the model to learn very complicated relationships across many length scales in the data. However, this can be a double-edged sword as the model can overfit to the data within its training region and fail to generalize to unseen data.

Translation Equivariance is a key property behind CNNs which allow them to generalize excellently to unseen data by learning features irrespective of their position in the data. Such feature learning is critical to modelling real world data where the positions of the data do not matter as much as the relative relationships between the data points, e.g. in image classification the position of the object in the image does not matter as much as the features of the object itself. Transformers lack this property, causing them to generate random predictions when the data is shifted out of context as shown in Figure 0.1.2.

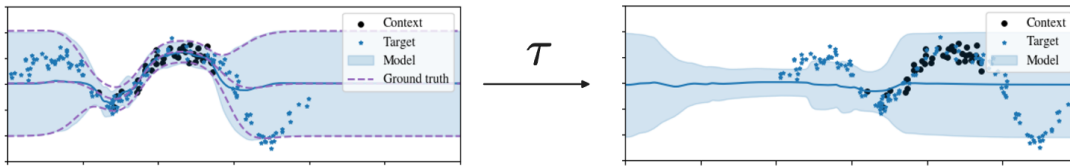


Figure 0.1.2: Vanilla Transformer Neural Process failing to generalize in out of context data. The TNP successfully models the data in the context region (left) but fails to make predictions when the data is shifted (right).

What if we could combine the best of both worlds? Could we create a Translation Equivariant Transformer Neural Process (TETNP) which possesses the global view of the Transformer and the generalization properties of the CNN? This is the question we will investigate in the next section and is the main contribution of this work and [Ashman et al. 2024].

0.2 Translation Equivariant TNP

Recall, Translation Equivariance requires the model to yield the same output when the input is shifted. Such property must imply that the model only learns the relative distances between the data points $(\mathbf{x}_i - \mathbf{x}_j)$ and *not* the absolute positions of the data points. How could one enforce such a property in the Transformer Neural Process? The solution used is very simple, we add a term to the attention mechanisms in the Transformer to enforce the Translation Equivariance property!

Consider the standard attention mechanism in the Transformer, the attention weights are computed as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{E}) \mathbf{V} \quad (0.2.1)$$

$$\mathbf{E}_{ij} = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}} \quad (0.2.2)$$

To create a Translation Equivariant Attention mechanism we add a term to the attention weights which enforces the Translation Equivariance property. The new attention weights are computed with the following equation:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{X}) = \text{softmax}(\mathbf{E} + F(\Delta)) \mathbf{V} \quad (0.2.3)$$

$$\mathbf{E}_{ij} = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}} \quad (0.2.4)$$

$$\Delta_{ij} = \mathbf{x}_i - \mathbf{x}_j \quad (0.2.5)$$

Where F is some function that introduces non-linearity into the attention weights and is applied to each entry of the matrix independently ($F(\Delta)_{ij} = F(\Delta_{ij})$), we will investigate the effect of different functions in the experiments later on. It is clear to see that if all the input locations are shifted by a constant τ , the relative term will remain the same $F(\Delta_{ij}) = F(\mathbf{x}_i + \tau - \mathbf{x}_j - \tau) = F(\mathbf{x}_i - \mathbf{x}_j)$ and the attention weights will remain the same.

Importantly since we are enforcing the Transformer to learn the relative distances between the data points, we must remove the \mathbf{x} values from the tokenization of the data points. The tokenization of the data points is now:

$$\mathbf{t}_i = \begin{cases} \text{MLP}(\text{cat}[0, \mathbf{y}_i]) & \text{if } i \leq N_c \\ \text{MLP}(\text{cat}[1, \mathbf{0}]) & \text{if } i > N_c \end{cases}$$

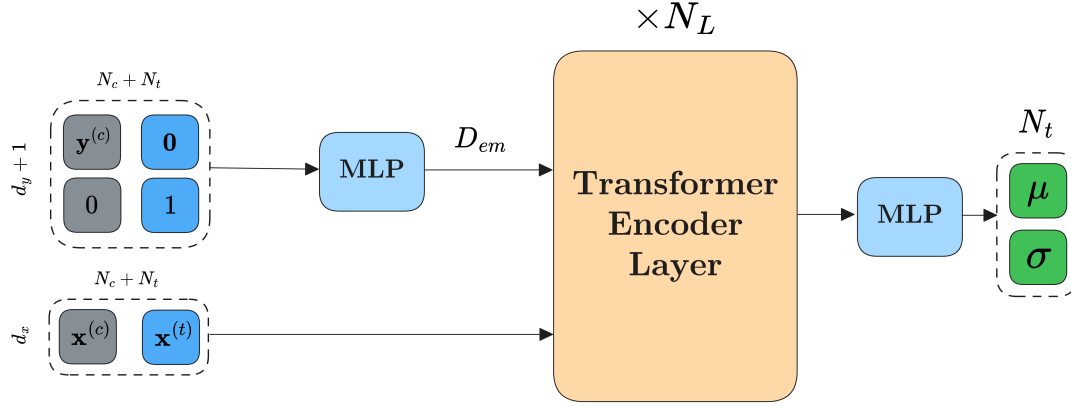


Figure 0.2.1: Translation Equivariant Transformer Neural Process Architecture. All the outputs of the context and target sets are tokenized and passed through the Transformer Layer, whilst the input locations are passed in raw to the attention mechanism to enforce the Translation Equivariance property. The output of the Transformer Layer is passed through an MLP to generate the mean and variance of the predictive distribution.

A benefit of removing the \mathbf{x} values from the tokenization is that the tokens only need to encode the \mathbf{y} values, reducing the dimensionality of the tokens whilst also separating the encoding for \mathbf{x} and \mathbf{y} values which could be beneficial for the model as in the vanilla TNP the model loses distinction between the \mathbf{x} and \mathbf{y} values since they are embedded together.

Bibliography

- Ashman, Matthew et al. (2024). “Translation-Equivariant Transformer Neural Processes”. In: *Forty-first International Conference on Machine Learning*. URL: <https://openreview.net/forum?id=pftXzp6Yn3>.
- Feng, Leo et al. (2022). “Efficient Queries Transformer Neural Processes”. In: *Sixth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems*. URL: https://openreview.net/forum?id=_3FyT_W1DW.
- Kim, Hyunjik et al. (2019). *Attentive Neural Processes*. arXiv: [1901.05761](#) [cs.LG].
- Nguyen, Tung and Aditya Grover (2023). *Transformer Neural Processes: Uncertainty-Aware Meta Learning Via Sequence Modeling*. arXiv: [2207.04179](#) [cs.LG].
- Vaswani, Ashish et al. (2017). *Attention Is All You Need*. arXiv: [1706.03762](#) [cs.CL].