0.1 Vanilla Transformer Neural Process

An Attention based encoder for the Neural Process was investigated in [Kim+19], though the results were impressive, the model fails to perform at larger scale and ‘tends to make over-confident predictions and have poor performance on sequential decision-making problems’ [NG23]. It is natural to think that the Transformer [Vas+17] 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. [NG23] 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 $\mathcal{C} = \{(\boldsymbol{x_i}, \boldsymbol{y_i})\}_{i=1}^{N_c}$ and target set $\mathcal{T} = \{(\boldsymbol{x_i})\}_{i=N_c+1}^{N}$, the model will encode a data point into a token $\boldsymbol{t_i}$ as follows:

$$\boldsymbol{t_i} = \begin{cases} \mathrm{MLP}(\texttt{cat}[\boldsymbol{x_i}, 0, \boldsymbol{y_i}]) & \text{if } i \leq N_c \\ \mathrm{MLP}(\texttt{cat}[\boldsymbol{x_i}, 1, \boldsymbol{0}]) & \text{if } i > N_c \end{cases}$$

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 $\boldsymbol{0}$ is a vector of zeros of the same dimension as $\boldsymbol{y_i}$.

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

TODO

ADD FIGURE OF MASKING

With the correct masking in place, 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.

TODO
ADD FIGURE OF TRANSFORMER NEURAL PROCESS

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.

TODO
Add a figure showing the effect of shifting the data on the predictions of the TNP

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.

0.2 Efficient Transformer Neural Process

TODO
Add a section on the Efficient Transformer Neural Process

0.3 Translation Equivariant Transformer Neural Process

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 $(\boldsymbol{x_i} - \boldsymbol{x_j})$ 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:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\boldsymbol{E}\right)\boldsymbol{V} \tag{0.3.1}$$
$$\boldsymbol{E_{ij}} = \frac{\boldsymbol{q_i} \cdot \boldsymbol{k_j}}{\sqrt{d_k}} \tag{0.3.2}$$

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:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{X}) = \text{softmax}\left(\boldsymbol{E} + F(\boldsymbol{\Delta})\right)\boldsymbol{V} \tag{0.3.3}$$
$$\boldsymbol{E_{ij}} = \frac{\boldsymbol{q_i} \cdot \boldsymbol{k_j}}{\sqrt{d_k}} \tag{0.3.4}$$
$$\boldsymbol{\Delta_{ij}} = \boldsymbol{x_i} - \boldsymbol{x_j} \tag{0.3.5}$$

Where F is some function that introduces non-linearity into the attention weights and is applied to each entry of the matrix independently ($F(\boldsymbol{\Delta})_{ij} = F(\boldsymbol{\Delta_{ij}})$), 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 $\boldsymbol{\tau}$, the relative term will remain the same $\boldsymbol{F(\Delta_{ij})} = F(\boldsymbol{x_i} + \boldsymbol{\tau} - \boldsymbol{x_j} - \boldsymbol{\tau}) = F(\boldsymbol{x_i} - \boldsymbol{x_j})$ 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 $\boldsymbol{x}$ values from the tokenization of the data points. The tokenization of the data points is now:

$$\boldsymbol{t_i} = \begin{cases} \text{MLP}(\texttt{cat}[0, \boldsymbol{y_i}]) & \text{if } i \leq N_c \\ \text{MLP}(\texttt{cat}[1, \boldsymbol{0}]) & \text{if } i > N_c \end{cases}$$

TODO

TETNP architecture figure

TODO

Highlight diffeerence in the encoding of the data points between TNP and TETNP

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

Bibliography

[Kim+19] Hyunjik Kim et al. *Attentive Neural Processes*. 2019. arXiv: 1901.05761 [cs.LG].

[NG23] Tung Nguyen and Aditya Grover. *Transformer Neural Processes: Uncertainty-Aware Meta Learning Via Sequence Modeling*. 2023. arXiv: 2207.04179 [cs.LG].

[Vas+17] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].