Transformers for modelling Physical systems

Anuleela ketthe
am1200640@am.iitd.ac.in

Abstract

Transformers are widely used in the Natural language processing due to their ability for long term dependencies. These models are a type of deep learning model mostly used in natural language processing (NLP) tasks. These have little applicability outside of the natural language processing. In this work, we use transformers for predicting the dynamical representation of the physical phenomenon. Transformers rely on the self-Attention that allows the transformer to learn temporal dependencies by using Koopman embedding technique which is used for all dynamical systems to project any dynamical system into a vector representation.

1 Introduction

Transformer models built on self-Attention for large tasks related to natural language processing. Transformers are becoming state of art approach for many data sets. In the deep learning models there are many models such as Auto-regressive, Recurrent Neural Network, Long short Term Method (LSTM) that are also used to model the physical systems. In the recent times, transformers are entirely relying on the Self-Attention to model dynamics. In this work, a physics inspired embedding model is used to model the physical systems.

2 Methods

Mainly we focus on the dynamical systems which are in the form of differential equations or partial differential equations:

$$\phi_t = F\left(x, \phi(t,x,\eta), \nabla_x \phi, \nabla_x^2 \phi, \phi \cdot \nabla_x \phi, \ldots\right), \quad F : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n,$$
$$t \in \mathcal{T} \subset \mathbb{R}^+, \quad x \in \Omega \subset \mathbb{R}^m,$$

in which $\phi \in \mathbb{R}^n$ is the solution of this differential equation with parameters η, in the time interval $\mathcal{T}$ and spatial domain Ω with a boundary $\Gamma \subset \Omega$. In the problem, the continuous solution is discretized into both spatial and temporal domains. The discretized solution $\phi = \{\phi_0, \phi_1, \ldots, \phi_T\}$, $\phi_i \in \mathbb{R}^{n \times d}$ for which ϕ_i has been discretized by d points. The dynamical problem is posed as a time-series problem.

There are two core components:

1. Embedding for projecting physical state of dynamical problem into the vector representation
2. Training the transformer input as embedding

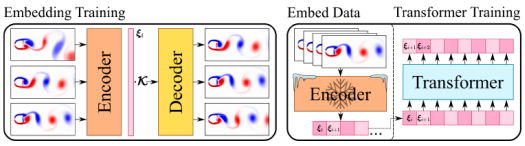

Figure 1: The two training stages for modelling the physical systems using transformers.

In the above figure 1 , we can see the two stages, first we train the embedding model with Koopman operator and then embedded training data is trained using the transformer followed by conversion of the embedding space into the physical state using the embedding decoder.

2.1 Transformer

Originally Transformers are designed with the NLP with word2vec embedding . Embedding model is used for physical systems whereas Self-Attention is used for the time-series problem. Input to the Transformer is embedding space $\mathbb{E} = \{\xi_0, \xi_1, \ldots, \xi_T\}$, where the embedded state at time-step i is denoted as $\xi_i \in \mathbb{R}^2$.

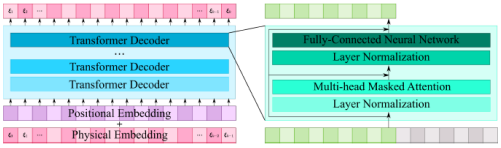

Figure 2: Transformer decoder used for predicting the dynamical system

We use the Transformer model 2 which is also used in the Generative pretrained models (GPT) models

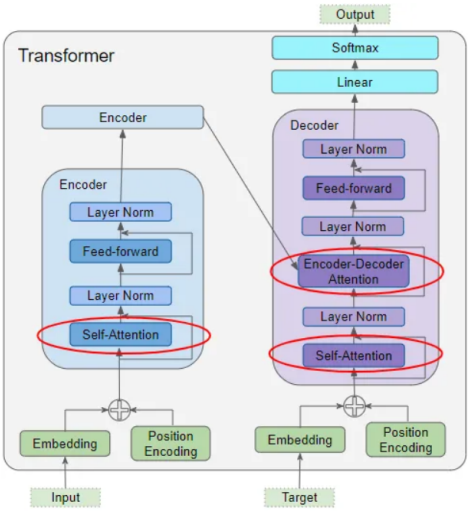

Figure 3: Transformer model used in the NLP

Main components of Transformer :

2.1.1 Word Embedding

The transformer has two embeddings, one in the encoder and another in decorder. We generally pass the test sequence IDs 4 which is mapped as text sequence to IDS according to the word vocabulary to embeddings.

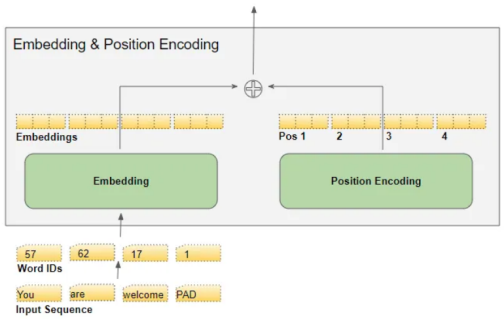

Figure 4: Embedding

2.1.2 Positional Encoding

RNN implements the for loop where it knows the position of the input explicitly. Since transformers take input inparallel, information regarding position is lost. Therefore we incorporate the positional ecoding to the achieve the positional information both at the encoder and decoder stacks which have the same dimensionality d_{model} as the embeddings.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

where pos is postion of the word in the sequence, d_{model} is the length of the encoding vector, i is the index of the vector

2.1.3 Self-attention

In context of NLP, in this component we find how the word is related to other words using Query (Q), Key (K) and value (V) matrix. The output is the weighted sum of the values of the input where weight is the self-Attention function, and Q, K and V are calculated by multiplying embedded word vector with W_q, W_k and W_v matries which are of size (d_{model}, d_{model}). Whereas in the context of physics, it is used to learn the temporal dependencies. However, there are many attention models, commonly used is scaled-dot product Attention, which are more faster and more space-efficient in practice.

Scaled-dot product :

$$\boldsymbol{k}_i = \mathcal{F}_k(\boldsymbol{x}_i), \quad \boldsymbol{q}_i = \mathcal{F}_q(\boldsymbol{x}_i), \quad \boldsymbol{v}_i = \mathcal{F}_v(\boldsymbol{x}_i)$$

$$\boldsymbol{c}_n = \sum_{i=1}^{k} \alpha_{n,i} \boldsymbol{v}_i, \qquad \alpha_{n,i} = \frac{exp(\boldsymbol{q}_n^T \boldsymbol{k}_i / \sqrt{d_k})}{\sum_{j=1}^{k} exp(\boldsymbol{q}_n^T \boldsymbol{k}_j / \sqrt{d_k})},$$

where k, q, v are the keys,queries,values which are learned through the neural networks, c_n is output, $\alpha_{n,i}$ is the Attention score. The attention scores always sum to one,$\sum_{i=1}^{k} \alpha_{n,i} = 1$, for every input.

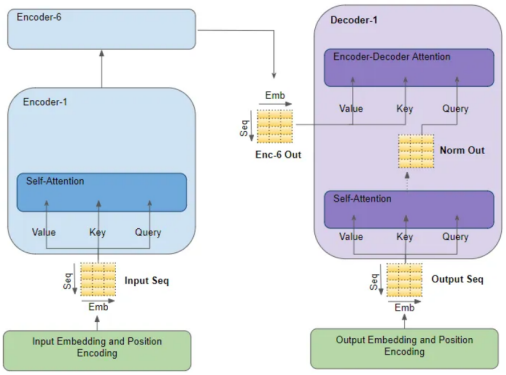

Figure 5: Three different self-Attention

Attention used at Three positions in Transformer:

1. **Attention used in encoder:** The input embedding and the positional embedding which has positional information for each word. Through the self-Attention, the Attention score is calculated for each word, then it passes to next stack
2. **Attention used in decoder:** The target is passed to output embedding and positional embedding, it passed to the self-Attention, attention score is calculated here then passed to the norm layer where it is fed to query parameter.
3. **Encoder-Decoder-attention in the Decoder:** At the encoder-Decoder Attention, the output from the encoder stack is passed to the key and value parameters and the attention score is added at each self-attention.

2.1.4 Multi-Head attention

All the Attention modules run in parallel, outputs from each attention heads are concatenated to produce the final Attention score. This gives greater power for transformer to encode multiple relationships for each word.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

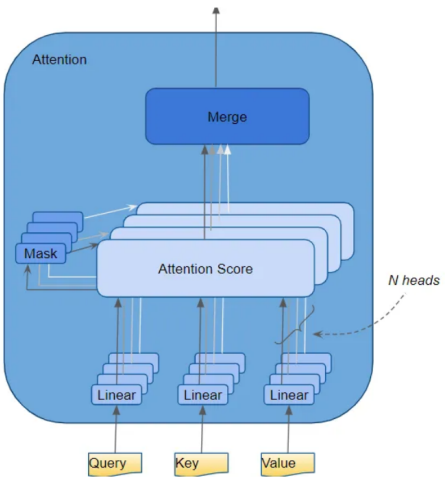

Figure 6: mutli-head attention