Transformers in Time-series Analysis: A Tutorial

Sabeen Ahmed[1], Ian E. Nielsen[1], Aakash Tripathi[1]
Shamoon Siddiqui[1], Ghulam Rasool[1], Ravi P. Ramachandran[1]
[1] Rowan University
{ahmedsa, nielseni6, tripat67, siddiq76, rasool, ravi}@rowan.edu

Abstract

Transformer architecture has widespread applications, particularly in Natural Language Processing and computer vision. Recently Transformers have been employed in various aspects of time-series analysis. This tutorial provides an overview of the Transformer architecture, its applications, and a collection of examples from recent research papers in time-series analysis. We delve into an explanation of the core components of the Transformer, including the self-attention mechanism, positional encoding, multi-head, and encoder/decoder. Several enhancements to the initial, Transformer architecture are highlighted to tackle time-series tasks. The tutorial also provides best practices and techniques to overcome the challenge of effectively training Transformers for time-series analysis.

1 Introduction and Background

Transformers belong to a class of machine learning models that use self-attention or the scaled dot-product operation as their primary learning mechanism. Transformers were initially proposed for neural machine translation - one of the most challenging natural language processing (NLP) tasks [64]. Recently, transformers have been successfully employed to tackle various problems in machine learning and achieve state-of-the-art performance. Apart from classical NLP tasks, examples from other areas include image classification, object detection & segmentation, image & language generation, sequential decision making in reinforcement learning, multi-modal (text, speech, and image) data processing, and analysis of tabular and time-series data. This tutorial paper focuses on time-series analysis using Transformers.

The time-series data consist of ordered samples, observations, or features recorded sequentially over time. Time-series datasets often arise naturally in many real-world applications where data is recorded over a fixed sampling interval. Examples include stock prices, digitized speech signals, traffic measurements, sensor data for weather patterns, biomedical measurements, and various kinds of population data recorded over time. The time-series analysis may include processing the numeric data for multiple tasks, including forecasting, prediction, or classification. Statistical approaches involve using various types of models, such as autoregressive (AR), moving average (MA), auto-regressive moving average (ARMA), AR Integrated MA (ARIMA) and spectral analysis techniques.

Machine learning models with specialized components and architectures for handling the sequential nature of data have been extensively proposed in the literature and used by the community. The most notable of these machine learning models are Recurrent Neural Networks (RNNs) and their popular variants, including Long-Short Term Memory (LSTM) and Gated Recurrent Units (GRU) [35], [23], [13]. These models process batches of data sequentially, one sample at a time, and optimize unknown model parameters using the well-known gradient descent algorithm. The gradient information for updating model parameters is calculated using back-propagation through time (BPTT) [10]. LSTMs and GRUs have been successfully used in many applications. However, they suffer from several limitations due to the sequential processing of input data and the challenges associated with BPTT, especially while processing datasets with long dependencies. The training process of LSTM and GRU models also suffers from vanishing and exploding gradient problems. While processing long sequences, the gradient descent algorithm (using BPTT) may not update the model parameters as the gradient information is lost (either approaches zero or infinity). Additionally, these models generally do not benefit from the parallel computations offered by graphical processing units (GPUs), tensor processing units (TPUs), and other hardware. Certain architectural modifications and training tricks may help LSTMs and GRUs mitigate gradient related problems to a certain extent. However, challenges in learning from long sequences of data with the limited use of parallelization offered by modern hardware impact the effectiveness and efficiency of RNN based models [55].

In this tutorial, we start by providing an overview of the transformer architecture based upon self-attention, scaled dot-product, multi-head, and positional encoding in Section 2. Section 3 describes the

advancements of transformers for time-series applications. We then discuss some of the most popular recent time-series Transformer architectures in Section 4. Finally, we provide "best practices" for training transformers in Section 5 before concluding the tutorial paper.

2 Transformers: Nuts and Bolts

We begin by explaining the inner workings of the Transformer as proposed by Vaswani *et al.* in 2017 to solve the challenging problem of neural machine translation [64]. We then proceed with a deep dive into the operations performed inside each component of Transformers and the intuition behind those operations. While several variations of the Transformer architecture have been developed, we restrict our discussion to the original architecture [22, 48, 12].

2.1 The Transformer Architecture

The original Transformer is a sequence-to-sequence model designed in an encoder-decoder type of configuration that takes as input a sequence of words from the source language and then generates the translation in the target language [58, 64]. Given that the length of the two sequences and the vocabulary size are not necessarily the same, the model has to learn to encode the source sequence into a fixed-length representation that it can then decode to generate the target sequence in an auto-regressive fashion [8]. This auto-regressive property comes with a constraint of requiring information to propagate back to the beginning of the sequence during the generation of the translated sequences. The same constraint holds for the time-series analysis.

The machine learning models have been limited by how far back the impact of a specific data sample can be considered during the learning. In some cases, the auto-regressive nature of the training of machine learning models leads to memorization of past observations rather than the generalization of the training examples to new data [36, 30]. Transformers address these challenges by using *self-attention* and *positional encoding* techniques to jointly attend to and encode the order information as they analyze current data samples in the sequence. These techniques keep the sequential information intact for learning while getting rid of the classical notion of *recurrence* [21]. These techniques further allow Transformers to exploit parallelism offered by GPUs and TPUs. Recently, there have been some research attempts to incorporate recurrent components in Transformers [68].

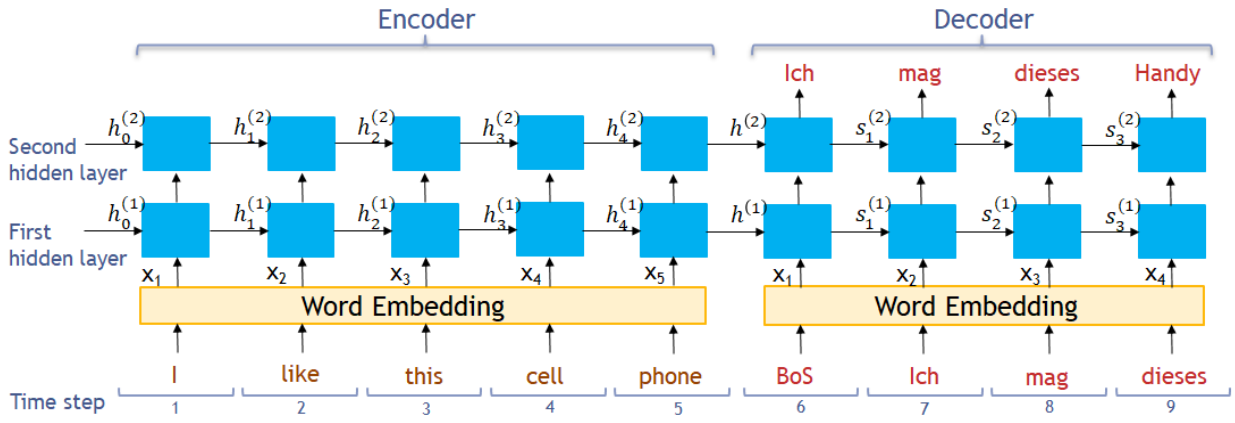
A Simple Translation Example: Consider an example of translating "I like this cell phone" to German "Ich mag dieses Handy" using a classical machine translation model (an RNN, LSTM, or GRU) and Transformer as illustrated in Figure 1. The input words must be first processed using an embedding layer to convert raw words into vectors of size d. The concept of embedding a discrete word into a continuous space of real numbers is a common practice in NLP [40, 51, 43, 44]. In classical language translation models, each embedded word in a sentence corresponds to a specific RNN/LSTM/GRU cell. The operations in the subsequent cells depend on the output from the previous cell. Therefore, each embedded word in the input is processed sequentially after processing the preceding word. In a model based on the Transformer architecture, the entire input sequence "I like this cell phone" is fed into the model simultaneously, eliminating the need for sequential processing of the data. The order of the sequence is tracked using *positional encoding*.

2.2 Self-Attention Operation

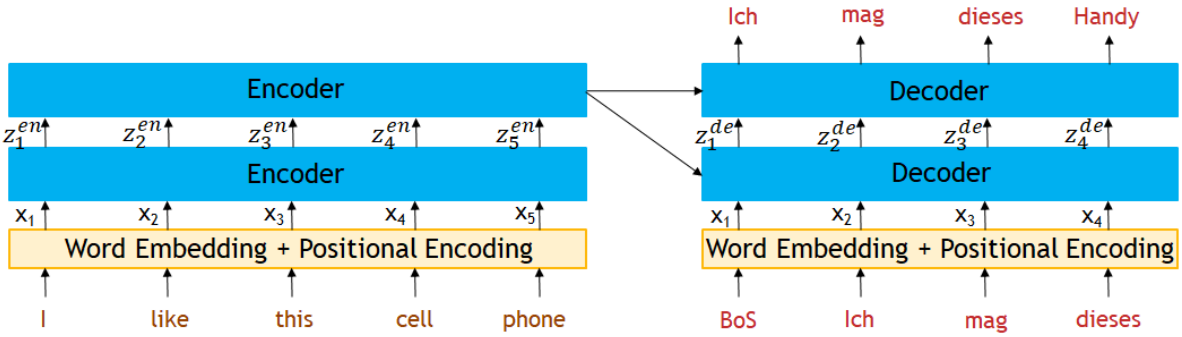
The Transformer architecture is based on finding associations or correlations between various input segments (after adding the position information in these segments) using the dot product [18]. Let $\{\mathbf{x}_i\}_{i=1}^{n}, \mathbf{x} \in \mathcal{R}^d$ be a set of n words (or data points) in a single sequence. The subscript i represents the position of the vector $\mathbf{x}_i$, which is equivalently the position of the word in the original sentence or the word sequence. The self-attention operation is the weighted dot product of these input vectors $\mathbf{x}_i$ with each other.

2.2.1 The Intuition Behind Self-Attention:

We can think about the self-attention operation as a two step process. The first step calculates a normalized dot product between all pairs of input vectors in a given input sequence. The normalization is performed



(a)



(b)

Figure 1: An example of a simple language translation task performed using (a) a classical model, such as an RNN, LSTM, or GRU, and (b) a Transformer.

using the softmax operator, which scales a given set of numbers such that the output numbers sum to unity. The normalized correlations are calculated between an input segment $\mathbf{x}_i$ and all others $j = 1, \ldots, n$:

$$w_{ij} = \text{softmax}\left(\mathbf{x}_i^T \mathbf{x}_j\right) = \frac{e^{\mathbf{x}_i^T \mathbf{x}_j}}{\sum_k e^{\mathbf{x}_i^T \mathbf{x}_k}}, \tag{1}$$

where $\sum_{j=1}^{n} w_{ij} = 1$ and $1 \leq i, j \leq n$.

In the second step, for a given input segment $\mathbf{x}_i$, we find a new representation $\mathbf{z}_i$, which is a weighted sum of all input segments $\{\mathbf{x}_i\}_{j=1}^{n}$:

$$\mathbf{z}_i = \sum_{j=1}^{n} w_{ij} \mathbf{x}_j, \quad \forall \;\; 1 \leq i \leq n. \tag{2}$$

We note that in eq. 2, for any input segment $\mathbf{x}_i$, the weights w_{ij} add up to 1. Thus, the resulting representation vector $\mathbf{z}_i$ will be similar to the input vector $\mathbf{x}_j$ having the largest attention weight w_{ij}. The largest attention weight has, in turn, resulted from the greatest value of correlation, as measured by the normalized dot product between $\mathbf{x}_i$ and $\mathbf{x}_j$. Note that $\mathbf{z}_i$ in eq. 2 retain the same position in the sequence as $\mathbf{x}_i$. Proceeding further with the next output vector $\mathbf{z}_{i+1}$, the new set of weights corresponding to $\mathbf{x}_{i+1}$ are calculated and used.

Consider the sentence, "Extreme brightness of the sun hurts the eyes". Figure 2 shows the sequence of vector mappings through the self-attention process and provides insight into the role of the self-attention operation. The vector $\mathbf{z}_{sun}$ would be weighted more by the first occurrence of 'the' as compared to the second

occurrence of 'the'. Similarly, $\mathbf{z}_{eyes}$ would be more heavily weighted by $\mathbf{x}_{hurts}$ and $\mathbf{x}_{the}$ (second occurrence of 'the') than $\mathbf{x}_{of}$ and first occurrence of 'the'.

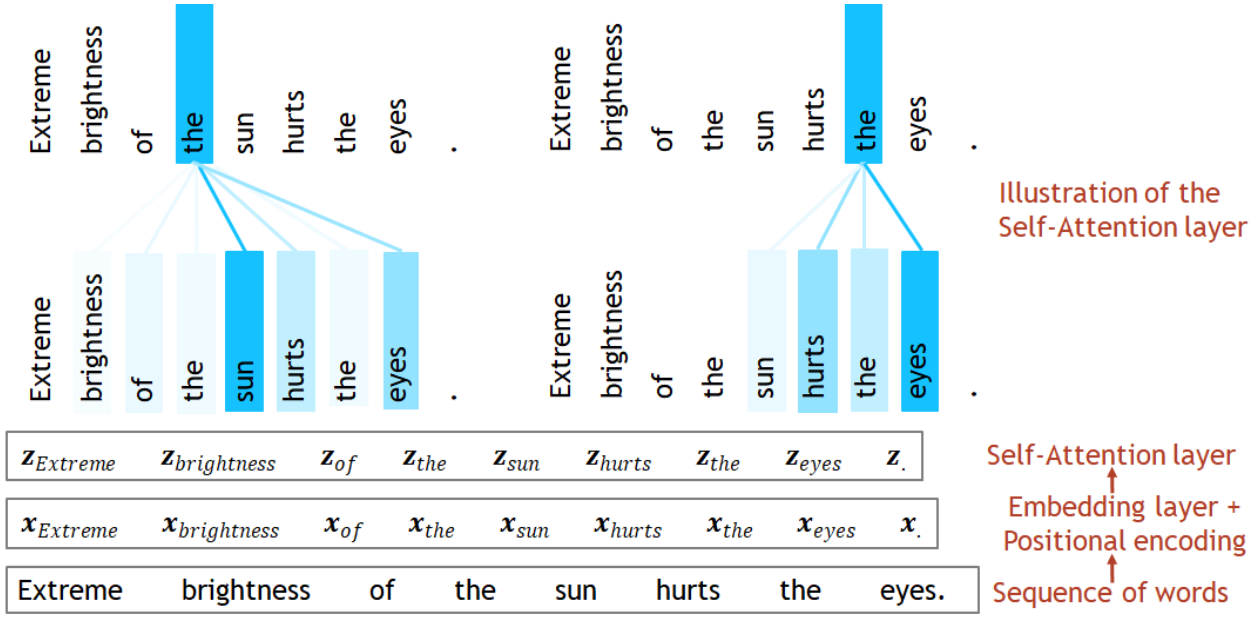


Figure 2: The role of self-attention for the example sentence "Extreme brightness of the sun hurts the eyes". The self-attention operation determines the relative correlation of each word with all the words in the sequence. In this example, the first occurrence of the word *the* is most correlated to the word *sun*, whereas the second occurrence of the word *the* has the highest correlation with the word *eyes*. The relative attention weights shown are generated using the Transformer attention visualization tool [BertViz].

2.2.2 Linearly Weighting Input Using Query, Key and Value:

The self-attention operation in Transformers starts with building three different linearly-weighted vectors from the input $\{\mathbf{x}_i\}_{i=1}^{n}$, referred to as query $\mathbf{q} \in \mathbb{R}^{s_1}$, key $\mathbf{k} \in \mathbb{R}^{s_1}$ and $\mathbf{v} \in \mathbb{R}^{s}$. Intuitively, a **query** is a question which can be one word or a set of words. An example is when one searches for the word 'network' on the internet to find more information about it. The search engine maps the query into a set of **keys**, e.g., neural, social, circuit, deep learning, communication, computer, and protocol. The **values** are the candidate websites that have information about the word 'network'.

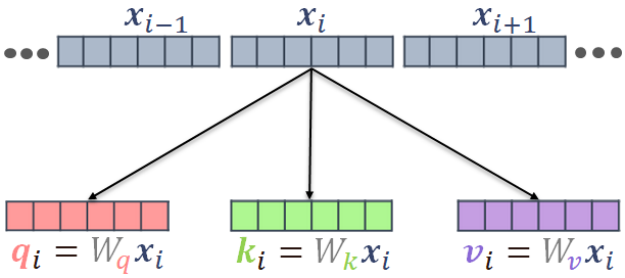


Figure 3: A set of weighted linear transformations applied to the input vector of length $d = 6$ are presented. The resulting vectors are referred to as **query**, **key**, and **value**, each of size $s_1 = s = d$. There are a total of n such inputs in each sequence processed by the Transformer, which result in n query, n key, and n value vectors.

For an input $\mathbf{x}_i$, query $\mathbf{q}_i$, key $\mathbf{k}_i$ and value $\mathbf{v}_i$ vectors can be found using:

$$\mathbf{q}_i = W_q \mathbf{x}_i, \quad \mathbf{k}_i = W_k \mathbf{x}_i, \text{ and } \mathbf{v}_i = W_v \mathbf{x}_i, \tag{3}$$

where W_q and $W_k \in \mathbb{R}^{s_1 \times d}$, $W_v \in \mathbb{R}^{s \times d}$, represent learnable weight matrices. The output vectors $\{\mathbf{z}_i\}_{i=1}^{n}$ are given by,

$$\mathbf{z}_i = \sum_j \text{softmax}\left(\mathbf{q}_i^T \mathbf{k}_j\right) \mathbf{v}_j. \tag{4}$$

We note that the weighting of the value vector $\mathbf{v}_i$ depends on the mapped correlation between the query vector $\mathbf{q}_i$ at position i and the key vector $\mathbf{k}_j$ at position j. The value of the dot product tends to grow with the increasing size of query and key vectors. As the softmax function is sensitive to large values, the attention weights are scaled by the square-root of the size of the query and key vectors d_q as given by

$$\mathbf{z}_i = \sum_j \text{softmax}\left(\frac{\mathbf{q}_i^T \mathbf{k}_j}{\sqrt{d_q}}\right) \mathbf{v}_j. \tag{5}$$

In the matrix form, we have:

$$Z = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V, \tag{6}$$

where Q and $K \in \mathbb{R}^{s_1 \times n}$, and $V \in \mathbb{R}^{s \times n}$, $Z \in \mathbb{R}^{s \times n}$ and ^{T} represents the transpose operation.

2.3 Multi-Head Self-Attention

The input data X may contain several levels of correlation information and the learning process may benefit from processing the input data in multiple different ways. Multiple self-attention heads are introduced that operate on the same input in parallel and use distinct weight matrices W_q, W_k, and W_v, to extract various levels of correlation between the input data. For example, consider the sentence "Do we have to turn left from here or have we left the street behind?". There are two occurrences of the word "left" in the sentence. Each occurrence has a different meaning, and consequently a different relationship with the rest of the words in the sentence. Transformers can capture such information by using multiple heads as shown in Figure 4. Each head is built using a separate set of query, key and value weight matrices and computes self-attention over the input sequence in parallel with other heads. The use of multiple heads in Transformer is analogous to using multiple kernels at each layer in convolutional neural network, where each kernel is responsible for learning distinct features or representations [2].

Figure 4: Multi-head self-attention example. Two heads are able to learn different attention weights for two different uses of the words "left" based on their meaning in the sentence. The relative attention weights shown are generated using the Transformer attention visualization tool [BertViz].

The operations involved in multi-head self-attention can be described by the following three steps.

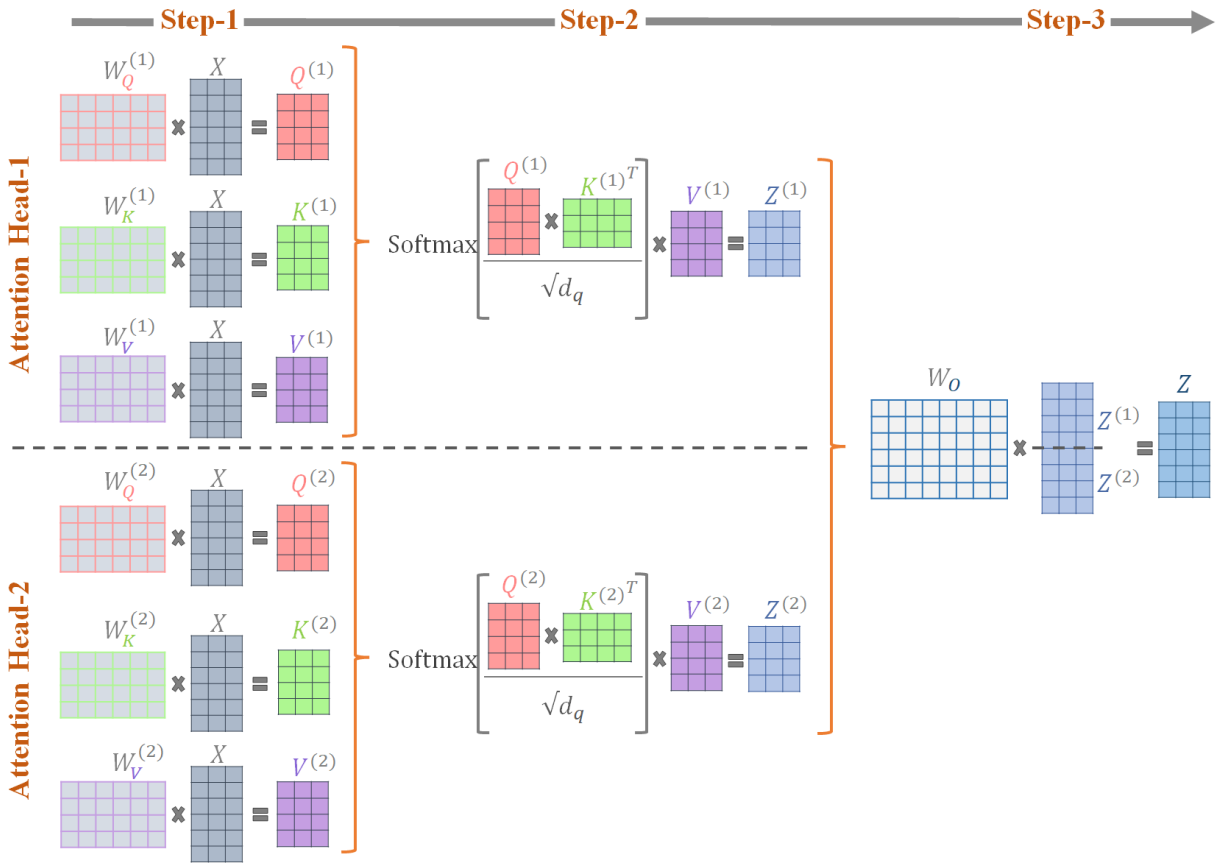


Figure 5: Multi-head attention steps. The example consists of two heads, and the input sequence comprises of three words. The input $\{\mathbf{x}_i\}_{i=1}^{n}$ is mapped to word embedding, and positional encoding vectors are added, resulting in the input matrix $X \in \mathbb{R}^{6\times 3}$. Two sets of query, key, and value matrices, $Q^{(l)}, K^{(l)}$, and $V^{(l)} \in \mathbb{R}^{4\times 3}$ $(l = 2)$, are constructed corresponding to each attention head. The self-attention operation applied on both sets of query, key, and value matrices produces output matrices $Z^{(1)}$ and $Z^{(2)} \in \mathbb{R}^{4\times 3}$. A linear operation, involving a learnable parameter $W_o \in \mathbb{R}^{d\times rs}$, follows the concatenation of $Z^{(1)}$ and $Z^{(2)}$ to map it to the same dimension as input matrix X. Output matrix $Z \in \mathbb{R}^{6\times 3}$ accumulates information from all the attention heads.

2.3.1 Step 1 - Generation of Multiple Sets of Distinct Query, Key and Value Vectors

Assuming we have a total of r heads, a total of r sets of weight matrices $\{W_q^{(l)}, W_k^{(l)}, W_v^{(l)}\}_{l=1}^{r}$ will generate r sets of distinct query, key and value matrices for the input X. The process is illustrated in Figure 5 for the case of an input with three vectors $(n = 3)$ of dimension six each $(d = 6)$ and $s_1 = s = 4$. This leads to the input matrix $X \in \mathbb{R}^{6\times 3}$, $W_q^{(l)}, W_k^{(l)}, W_v^{(l)} \in \mathbb{R}^{4\times 6}$ and $Q^{(l)}, K^{(l)}$, and $V^{(l)} \in \mathbb{R}^{4\times 3}$.

2.3.2 Step-2 - Scaled Dot Product Operations in Parallel

This step consists of implementing the following relationship as shown in Figure 5:

$$Z = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{7}$$

2.3.3 Step-3 - Concatenating and Linearly Combining Outputs

Finally, we concatenate outputs $Z^{(l)}$ from all r heads and linearly combine using a learnable weight matrix $W_o \in \mathbb{R}^{d\times rs}$. The output is matrix $Z \in \mathbb{R}^{d\times n}$. It is important to note that the input and output of the

multi-head self-attention are of the same dimension, that is, dimension (X) = dimension (Z).

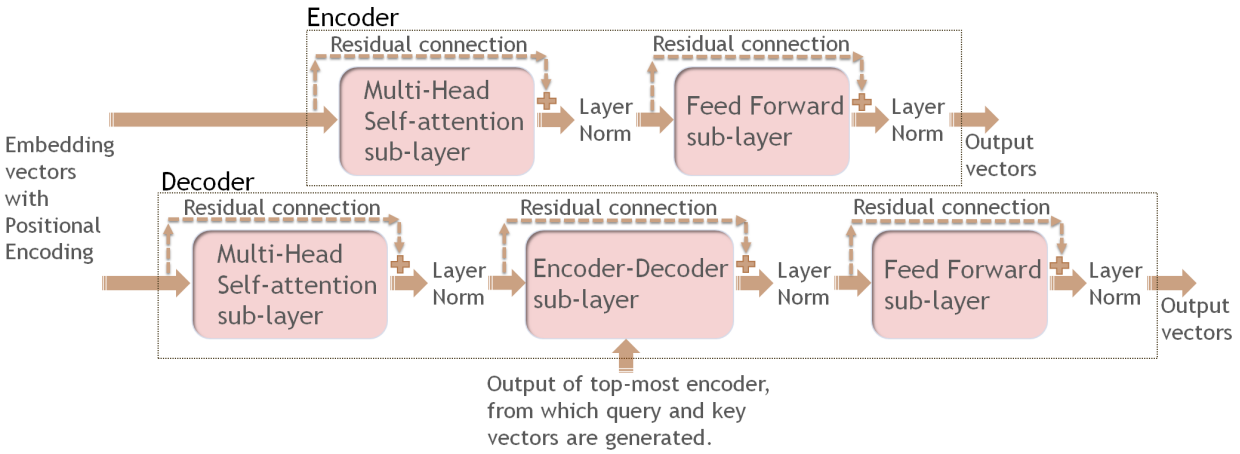


Figure 6: An encoder block is presented. Each encoder block is built using multi-head self-attention and feed forward layers with residual connections around each layer followed by the layer normalization operation.

2.4 Building Transformers Using Encoders and Decoders

The Transformer architecture is generally composed of multiple instances of two types of components referred to as encoders and decoders.

2.4.1 The Encoder Block

As shown in Figure 6, an encoder block consists of a multi-head self-attention layer and a feed forward layer connected back-to-back with residual connections and normalization layers. Residual connections are a commonly used technique for training deep neural networks and help in training stabilization and learning [59]. The layer normalization is also commonly used in neural networks for processing sequential data and helps in faster convergence of the training [4]. The feed forward layer comprises of two linear layers with a ReLU activation function [1]. The output of an encoder block is used as an input for the next encoder block. The input to the first encoder block consists of the sum of word embeddings and positional encoding (PE) vectors.

2.4.2 The Decoder Block

Each decoder block consists of similar layers and operations as the encoder block. However, a decoder receives two inputs, one from the previous decoder and the second from the last encoder. Inside a decoder, there are three layers that include (1) multi-head self-attention, (2) an encoder-decoder attention layer, and (3) a feed forward layer. There are residual connections and layer normalization operations, as shown in Figure 6. Inside the encoder-decoder attention layer, a set of key and value vectors are generated from the output of the last encoder. The query vectors are produced from the output of the multi-head self-attention layer preceding the encoder-decoder layer.

2.4.3 Masking in Self-Attention

Inside the decoder block, the multi-head self-attention layer masks parts of the target input during the training phase. This operation ensures that the self-attention operations do not involve future data points, i.e., the values that the decoder is expected to predict. During the training phase, the model's predicted output is not fed back into the decoder. Instead, the ground truth target (word embedding) is used to aid the learning. During the testing phase, the predicted words in the sequence are fed back to the decoder after passing through a word embedding layer and the addition of PE, as show in Figure 7.

2.4.4 Stacking Encoders and Decoders

A Transformer model may contain stacks of multiple encoder and decoder blocks depending on the problem being solved, as shown in Figure 7 [63]. The stacked encoder/decoder blocks are similar to multiple hidden layers used in traditional neural networks. However, it is important to note that generally, there is no reduction in the representation dimensions after processing by the encoder or the decoder. The input to the first encoder block is the word sequence mapped into word embeddings with PEs added.

2.4.5 The Output

The output from the last encoder is fed into each decoder along with input from the previous decoder. Optionally, the output of the last decoder block is passed through a linear layer to match the output dimension to the desired size, e.g., target language vocabulary size. The mapped output vectors are then passed through a softmax layer to find the probability of the next word in the output sequence. The operations on the output from the last decoder block can be selected for classification or regression depending upon the desired task.

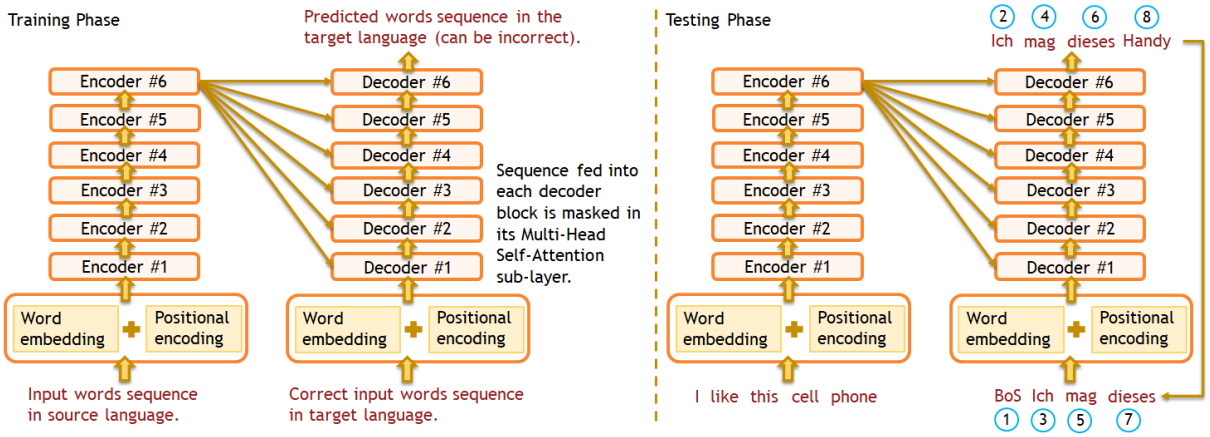


Figure 7: Stacked encoder and decoder blocks used in Transformers are presented for the machine translation task. The number of stacked blocks may vary depending on the task, analogous to multiple layers used in traditional neural networks. Input to the stack of encoders is word embeddings enriched with positional information. The output of the top most encoder is fed into each decoder, which helps develop the relationship between the source and target language. The output of the final decoder, mapped to the desired target language vocabulary, predicts the next word in the sequence. During training, unlike the testing phase, the correct sequence is fed back into the decoder stack irrespective of the predicted word.

2.5 Positional Encoding (PE)

The most important aspect of processing sequential data is incorporating the order of the sequence. The self-attention operations do not include any information about the order of input data in a sequence. Transformers use the concept of positional encoding to add the positional information to the input and process the modified input in parallel, avoiding the challenges of processing data sequentially. The technique consists of calculating n PE vectors ($p \in \mathbb{R}^d$) and adding these to the input $\{\mathbf{x}_i\}_{i=1}^n$.

2.5.1 Sinusoidal PE

In the original work, the authors proposed sinusoidal functions for pre-calculating PE vectors for the input dataset [64]. The PE vectors do not include any learnable parameters and are added directly to the word

embedding input vectors. Figure 8 shows the formulation of PE vectors with eqs. 8 and 9,

$$\text{PE}_{(pos,2i)} = \mathbf{p}_i \quad = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right), \tag{8}$$

$$\text{PE}_{(pos,2i+1)} = \mathbf{p}_{i+1} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right), \tag{9}$$

where, pos is the position (time-step) in the sequence of input words, i is the position along the embedding vector dimension ranging from 0 to $\frac{d}{2} - 1$, and d represents the dimension of the embedding vectors. The number 10,000 used in eqs. 8 and 9 can be different depending on the length of the input sequence.

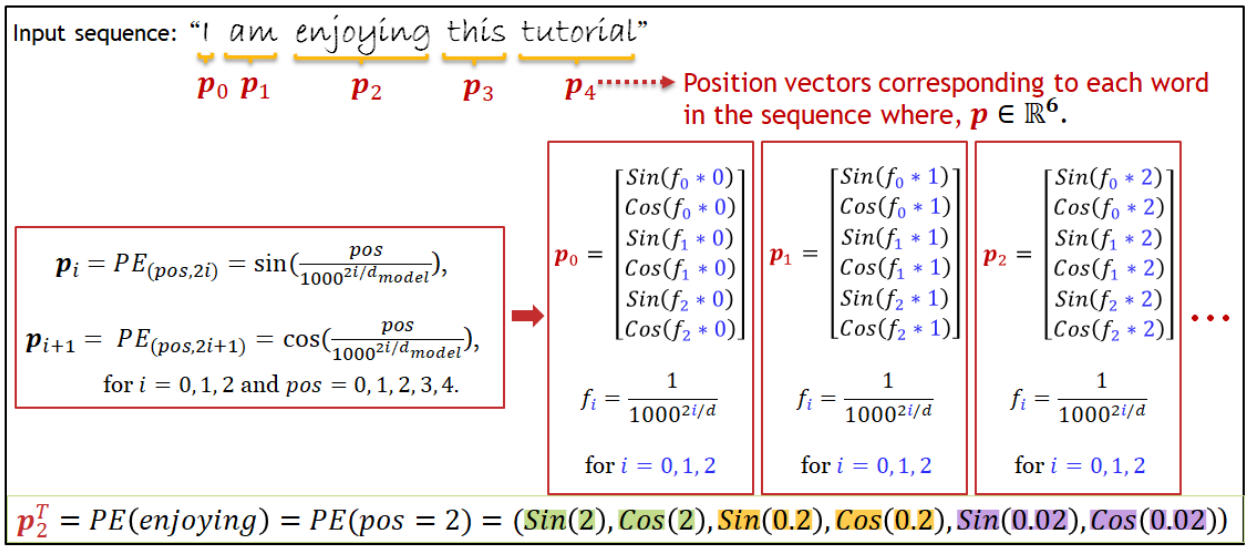


Figure 8: A set of positional encoding vectors are presented for an example input sequence. In this example the formulation of PE vectors uses a value of $1,000$ instead of $10,000$ (as in the original Transformer implementation).

2.5.2 Relationship of the Sinusoidal PE with Binary Encoding

We can draw an analogy between the proposed sinusoidal functions for PE and alternating bits in a six bit long binary number, as shown in Figure 9 [64, 60]. In the binary format, the least significant bit (shown in the orange color) alternates with the highest frequency. Moving to the left (indigo), the frequency of bit oscillation between 0 and 1, decreases. Similarly, in the sinusoidal PE, as we move along the positional encoding vector, the frequency of the sinusoidal function changes.

2.5.3 Positional Encoding and Rotation Matrix

PE vectors calculated using sinusoidal functions allow the model to learn about the relative position of words rather than their absolute position. For example, in the sentence "I am enjoying this tutorial", the absolute position of the word 'this' is 4. However, relative to the word 'am', the word 'this' is at position 3. Hence, for any fixed offset k, $\mathbf{p}_{i+k}$ can be represented as a linear transformation of $\mathbf{p}_i$. Consider $\mathbf{p}_i \in \mathbb{R}^2$ and let $T = \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \end{pmatrix}$ be a linear transformation with a representing the absolute position of a word in an input sequence. We can write

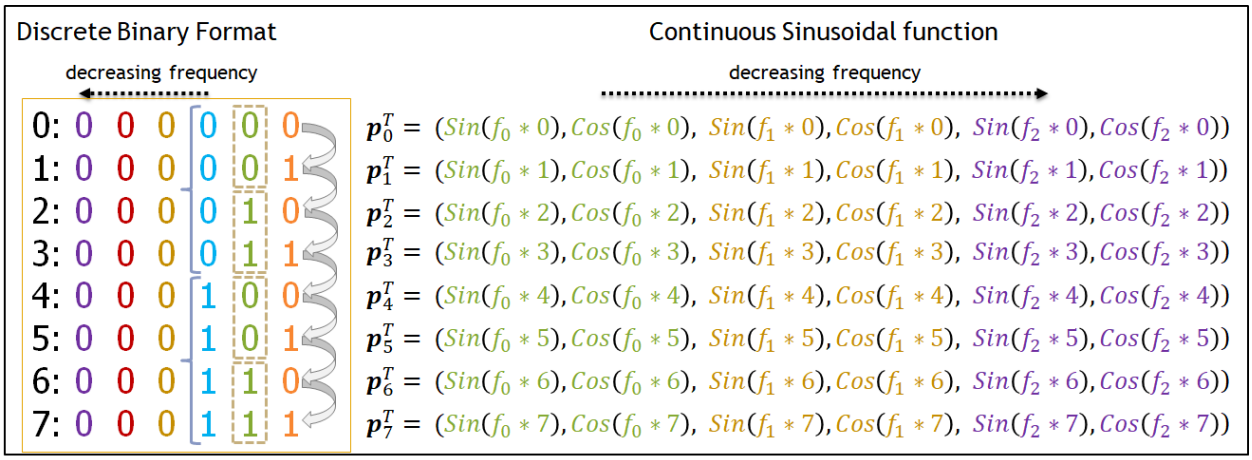


Figure 9: An analogy between binary format and sinusoidal function for positional encoding is presented. In the binary format, the frequency of alternating bits decreases as we move from orange to purple bits. Sinusoidal PE shows a similar behavior of changing frequency as we move along the positional encoding vector. We consider an input sequence of length $n = 8$ and a word embedding vector of size $d = 6$.

$$T \begin{bmatrix} \sin(f_i a) \\ \cos(f_i a) \end{bmatrix} = \begin{bmatrix} \sin(f_i (a+k)) \\ \cos(f_i (a+k)) \end{bmatrix}, \tag{10}$$

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix} \begin{bmatrix} \sin(f_i a) \\ \cos(f_i a) \end{bmatrix} = \begin{bmatrix} \sin(f_i(a+k)) \\ \cos(f_i(a+k)) \end{bmatrix}, \tag{11}$$

$$\begin{bmatrix} x_1 \sin(f_i a) + y_1 \cos(f_i a) \\ x_2 \sin(f_i a) + y_2 \cos(f_i a) \end{bmatrix} = \begin{bmatrix} \sin(f_i a)\cos(f_i k) + \sin(f_i k)\cos(f_i a) \\ \cos(f_i a)\cos(f_i k) - \sin(f_i a)\sin(f_i k) \end{bmatrix}. \tag{12}$$

Comparing both sides of eq. 12, we get $x_1 = \cos(f_i k)$, $y_1 = \sin(f_i k)$, $x_2 = -\sin(f_i k)$, and $y_2 = \cos(f_i k)$. The transformation T can now be written as:

$$T = \begin{bmatrix} \cos(f_i k) & \sin(f_i k) \\ -\sin(f_i k) & \cos(f_i k) \end{bmatrix}. \tag{13}$$

We note that the transformation T is a rotation matrix and depends on the relative position of words, not the absolute position. The sinusoidal PE function works for an input sequence of any length that need not be specified.

2.5.4 Combining Positional Encoding with Word Embeddings

The PE vectors are added to the word embeddings for each word in the input sequence. We may argue that the addition operation may result in the loss of some information from both sources, i.e., PE and word embeddings. However, that may not be the case, given that both PE and word embeddings encode different type of information. PE vectors contain information about the location of a word in the input sequence, while the word embedding encode semantic and contextual information about the word. The two encoding schemes belong to different sub-spaces, which may be orthogonal to each other. In this case, the addition of two such vectors may not result in the loss of information. We can consider an analogy with the frequency modulation operation as done in digital communication — a signal of lower frequency rides over a high frequency signal without interfering with each other.

2.5.5 Parameterized Learnable PEs

The computation of PE using sinusoidal functions does not include any learnable parameters and can be performed during or before training a neural network. Recently, various other methods for PE have been

proposed, including methods that have learnable parameters [79, 66]. However, no considerable difference between various PE methods is reported in the literature [64].

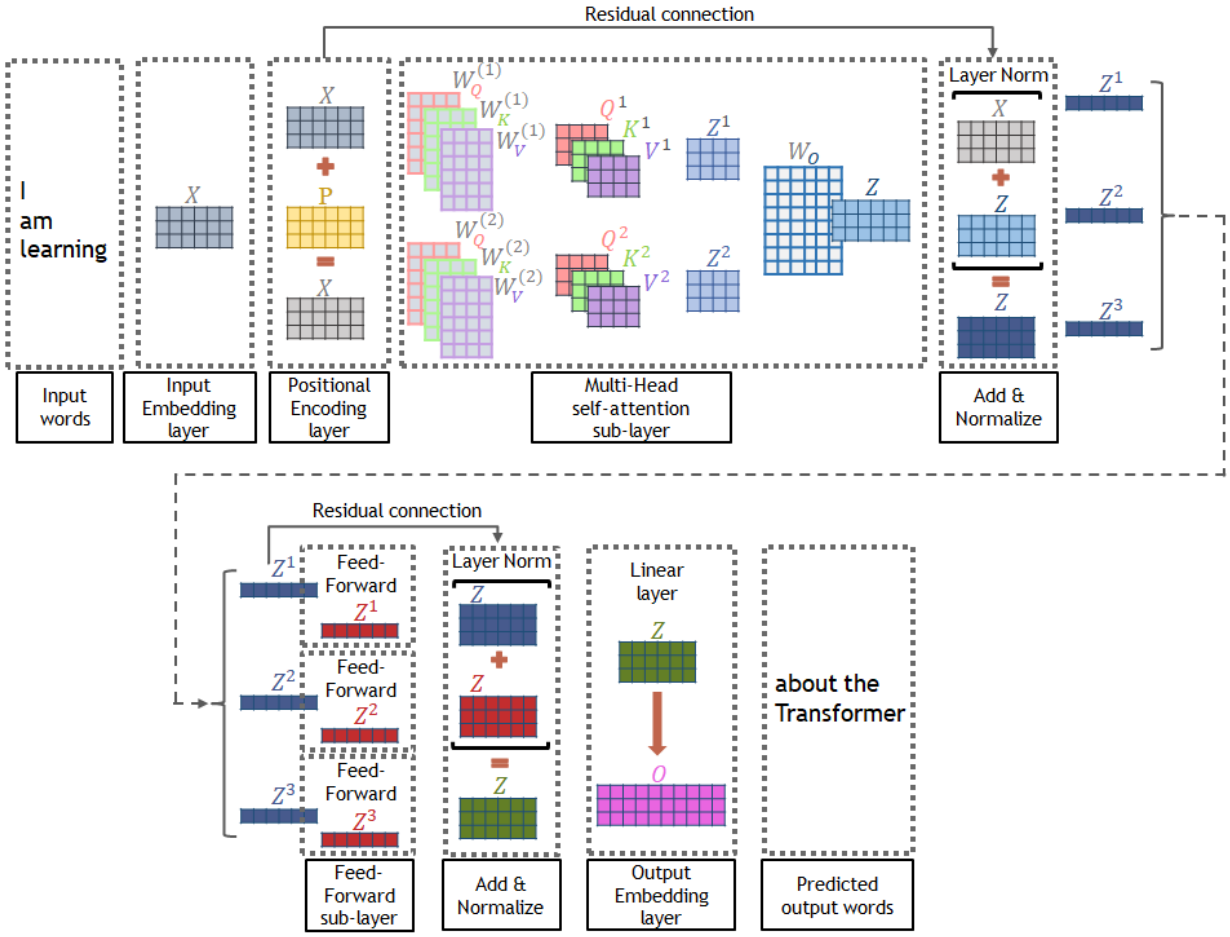


Figure 10: An example of a Transformer model with a single encoder is presented. End to end sequence of operations is shown for an input sequence of three words and an output prediction of the expected three following words.

3 Road Map of Transformers for Time-Series Analysis

Since the inception of Transformers in 2017 [64], there have been many advancements in these networks for time-series analysis. The original model mainly focuses on NLP tasks, but now the architecture has been expanded for classification [76], time-series analysis [57], semantic segmentation, [79] and more. Time-series data were not part of the initial conception of Transformers, but now many researchers have customized and improved the architecture's performance on this data type. In order to illustrate how this research has unfolded, we will provide a road map of time-series tasks and how the technology has advanced.

A commonality amongst the improvements is that the input layer is modified to accommodate time-series data. The use of Transformers for time-series analysis, forecasting, and classification accomplish two main tasks. Within each of these tasks we provide useful information and links to the datasets used in the state-of-the-art methods. The road map in this section provides a comprehensive breakdown of the advancements made in the past few years, and their relation to each other. Towards the end of each subsection is a list of the models (and citation) which are included in that category.

3.1 Avenues of Improvement for Time-Series Transformers

Each part of this subsection outlines major contributions and research articles that have improved specific mechanisms within the Transformer architecture. A schematic road map for studying Transformers in time-series analysis is shown in Figure 11.

3.1.1 Learning Type - Supervised, Self-supervised, or Unsupervised

Much of the mainstream Transformer applications rely on the training data, which is hand labeled. Labeling data can take a considerable time, rendering the massive amount of unlabeled data unusable. Self-supervised and unsupervised methods seek to improve Transformers by giving them the ability to learn, categorize, and forecast for data that has not been labeled. For example, there is a considerable amount of unlabeled satellite imaging data. Authors in [77], proposed a model, SITS-BERT, that learns from unlabeled data to apply region classification from satellite imagery. Other self-supervised or unsupervised learning models include anomaly Transformer [72] and self-supervised Transformer for Time-Series (STraTS) [62].

3.1.2 Data Preprocessing

Preprocessing is routinely performed to prepare the data for feeding into machine learning models. Preprocessing operations are known to impact the performance of the machine learning models. Some researchers either add noise to input data features [77], perform masking [53] or variable selection [34]. The operation of masking removes features in the input, improving performance by making the model better at predicting missing features. A similar concept applies when adding noise to the input for training, except the features become relatively noisier but are not replaced entirely. Both these operations can improve the robustness of the model so that it attains good accuracy despite the noise in the training or test data.

3.1.3 Positional Encoding (PEs)

Some recent work has focused on improving upon the original PEs proposed by [64]. Transformers use PEs to embed sequence/time information in the model input so that Transformers can process sequential data all at once, rather than one at a time like an RNN, LSTM, or a GRU. Embedding time (seconds, minutes, hours, weeks, years, etc.) into the input allows the model to better analyze time-series data and leverage computational benefits offered by modern hardware including, GPUs, TPUs, and others. Recently, timestamp embedding [53], temporal encoding [9] and other methods have been proposed for creating Transformers which can be trained more efficiently [34].

3.1.4 Gating Operation

A gating operation merges the output of two encoder towers of a standard Transformer model for a single set of output predictions, as shown by Liu et al. [38]. This operation combines and selects between multiple outputs from an encoder or decoder block. Gating also benefits from applying non-linear data processing when

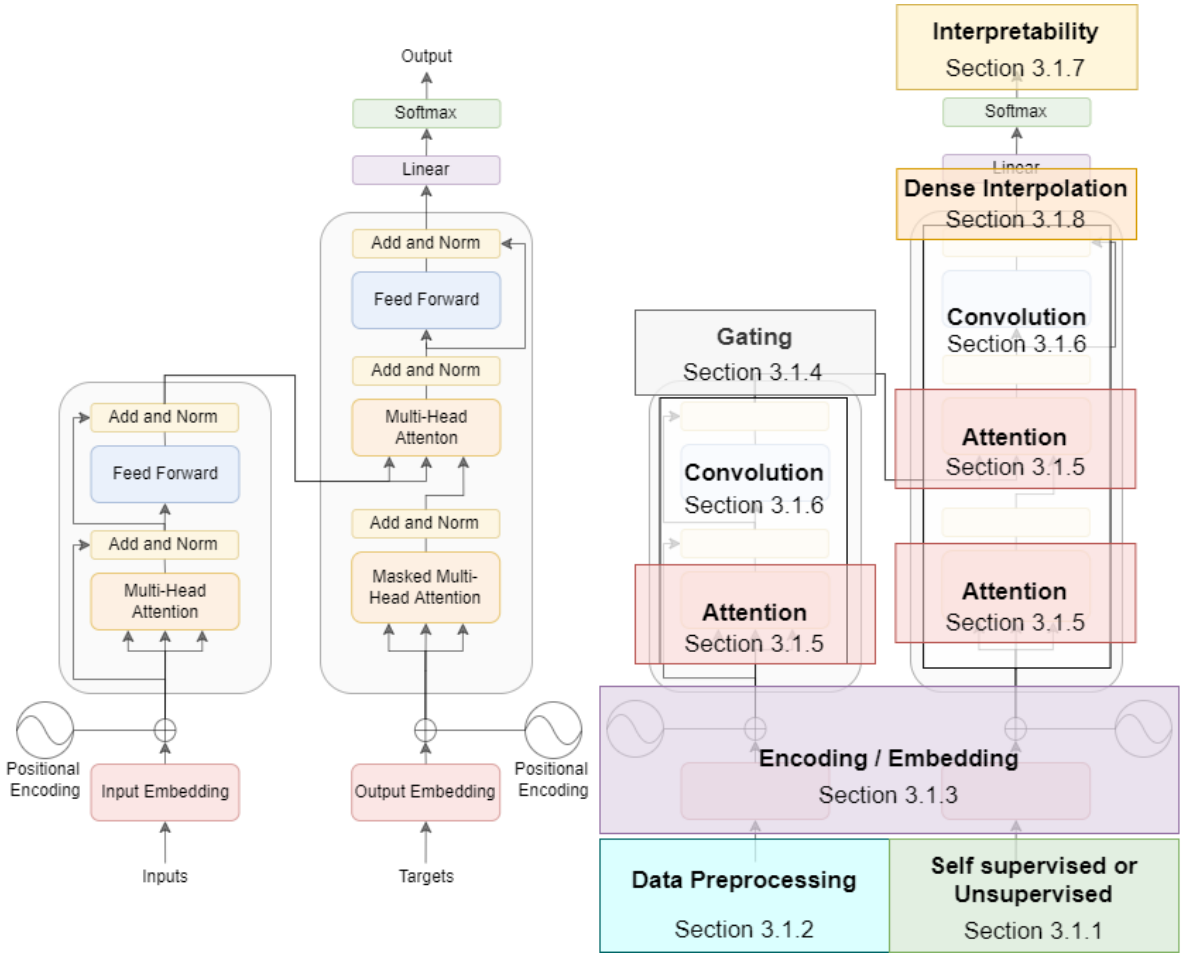


Figure 11: Overview of avenues of improvement for time-series Transformers. On the left is a recreation of the original Transformer architecture [64]. The right side shows locations in the original architecture for each of the avenues of improvement in subsection 3.1 with respect to the original Transformer model.

appropriate. Application of gating in various ways is a likely avenue for future innovations in Transformers, not only for time-series data but also for any other type of data. Several architectures use gating, including Gated Transformer Networks (GTN) [38] and Temporal Fusion Transformers (TFT) [34]. The proposed architecture of GTN uses gating techniques to incorporate information from two Transformer towers into the output. It does this by concatenating the output of each tower. TFT proposes Gated Linear Units (GLUs) [14] to allow different parts of the network to be emphasized or suppressed based on the dataset.

3.1.5 Attention

The models discussed in this section improve upon Transformers for time-series data by modifying and improving the attention mechanisms of the model.

Tightly-Coupled Convolutional Transformer (TCCT) [54] proposes three architectures that improve upon attention. The first is called Cross Stage Partial Attention (CSPAttention). This approach combines Cross Stage Partial Network (CSPNet) [65] with a self-attention mechanism to reduce required resources. CSPNet reduces computations by incorporating the feature map at the input into the output stage. CSPAttention applies this concept to just the attention layers, considerably reducing the time complexity and memory required. The second approach changes the self-attention distilling operation and how the self-attention blocks connect. Instead of canonical convolution, they use dilated causal convolution. Dilated causal convolution enhances the locality and allows the Transformer to attain exponentially receptive field growth. The

third architecture is a passthrough mechanism that allows multiple self-attention blocks to be stacked. This stacking is done by concatenating feature maps from multiple scales within the self-attention mechanisms. This mechanism improves the ability of the Transformer to pick up on fine-scale features.

More approaches to improving attention include Non-Autoregressive Spatial-Temporal Transformer (NAST) [11] and Informer [80]. The architecture proposed in NAST is referred to as the Spatial-Temporal Attention Block. This approach combines a spatial attention block, which makes predictions across space, with a temporal attention block which makes predictions across time. The result improves learning in both spatial and temporal domains. The Informer architecture replaces the canonical self-attention with the ProbSparse self-attention mechanism, which is more efficient in time complexity and memory usage. This Transformer also uses a self-attention distilling function to decrease space complexity. These two mechanisms make the Informer highly efficient at processing exceedingly large input data sequences.

Lastly we will discuss three architectures that improve upon attention, LogSparse Transformers [32], TFT [34], and YFormer [39]. LogSparse Transformers introduce convolutional self-attention blocks, consisting of a convolutional layer prior to the attention mechanism for creating the queries and keys. A temporal self-attention decoder is used in TFT for learning the long-term dependencies in the data. The YFormer architecture proposes a sparse attention mechanism combined with a downsampling decoder. This architecture allows the model to better detect long-range effects in the data.

3.1.6 Convolution

The original Transformer architecture does not make use of convolutional layers. However, this does not mean that Transformers would not benefit from the addition of convolutional layers. In fact, many Transformers designed for time-series data have benefited from adding convolutional layers or incorporating convolution into existing mechanisms. Most approaches incorporate convolutions either prior to or alongside the attention mechanism within the Transformer.

Approaches which improve upon Transformers via convolution include TCCT [54], LogSparse Transformers [32], TabAConvBERT [53], and Traffic Transformers [9]. TCCT uses a mechanism called dilated causal convolution from the Informer [80] architecture, replacing the canonical convolutional layers. LogSparse Transformers also use causal convolutional layers. As mentioned in the previous section, this layer generates queries and keys for the self-attention layers, referred to as convolutional self-attention. TabAConvBERT employs one-dimensional convolutions with the idea that this type of convolution is naturally effective for time-series data. Traffic Transformers [9] incorporate concepts from graph neural networks into the convolutional layers to produce Graph Convolutional Filters.

3.1.7 Interpretability/Explainability

Transformers are a relatively new class of machine learning models compared to CNNs or LSTMs. For their reliable and trustworthy use, we must understand the black-box nature of these models and explain their decisions. Many approaches to interpreting and explaining model predictions are post hoc, that is, the explanation is made after the fact. Post hoc approaches apply to almost any model. Many of these approaches provide visually appealing results, but might not accurately explain what is occurring inside the model [41]. One possible way to incorporate explanations and interpretability into the model itself, rather than being approximated after the fact. There now exist multiple time-series Transformers that are inherently interpretable [62, 38, 34]. These models can produce explanations which allow for better interpretations of results and greater user trust.

3.1.8 Dense Interpolation

The Transformer model generally consists of encoder and decoder blocks followed by linear and softmax layers for decision making. One approach referred to as Simply Attend and Diagnose (SAnD) replaces the decoder block with a dense interpolation layer to incorporate temporal order into the model's processing [57]. This approach does not utilize output embedding, cutting down the number of total layers in the model. Without the decoder, the model needs a way to process the outputs of the encoder block to be input into the linear layers. Simple concatenation leads to poor prediction accuracy; therefore [57] developed a dense interpolation algorithm with hyperparameters that can be tuned to improve performance.

3.2 Architectural Modifications

3.2.1 BERT-Inspired

A famous architecture that builds on the original Transformers paper is Bidirectional Encoder Representations from Transformers (BERT) [15]. The model is built by stacking the Transformer encoder blocks and introducing a new training scheme. The encoder block is pre-trained independently of the task. The decoder block can be added later and fine-tuned for the task at hand. This scheme allows training BERT models on large amounts of unlabeled data.

BERT architecture has inspired many new Transformer models for time-series data [77, 53, 47, 72]. Creating a BERT style model for time-series data has some challenges compared to NLP tasks. Language data is a standardized type of data that can be used for various tasks, including translation, text summarization, question answering, sentiment analysis, etc. All of these tasks can use the same data for pre-training. However, this is not the case for the time-series tasks. Examples of time-series data include electricity usage [16], ambient temperature [80], traffic volume [16], satellite imagery [77], various forms of healthcare data [28], and more. With this variety of data types, the pre-training process will have to be different for each task. This task dependent pre-training contrasts with the NLP tasks which can start with the same pre-trained models assuming all tasks are based on the same language semantics and structure.

3.2.2 GAN-Inspired

Generative adversarial networks (GANs) consist of two deep neural networks, the generator, and the discriminator, both learning adversarially from each other. GANs are commonly used in image processing for generating realistic images. The generator's task is to create images that will trick the discriminator. The discriminator is given the actual and generated (fake) images and must predict whether the input image was real or fake. A well-trained GAN can generate images that look very realistic to a human.

The same generator-discriminator learning principle has been applied to the time-series forecasting task [70]. The authors use a Transformer as the generator and a discriminator and train the model for accurate forecasting predictions. The generator's task is to create a forecast which the discriminator will classify as real or fake. As the training continues, the generator network will create more realistic data. At the end of the training, the model will be highly accurate at making predictions.

3.3 Time-Series Tasks

The two main tasks performed on time-series data are forecasting and classification. Forecasting seeks to predict real-valued numbers from given time-series data, referred to as regression. Many forecasting Transformers for time-series data have been developed in the recent literature [54, 11, 80, 32, 69, 34, 9, 70, 47, 62, 39]. The classification task involves categorizing the given time-series data into one or more target classes. There have been many recent advancements for time-series Transformers for classification tasks [53, 72, 77, 57, 38]. All of the time-series Transformer-based models discussed in this tutorial focus on one of these two tasks. Some of these models can accomplish both tasks after small modifications in the last layer and the loss function.

4 Time-Series Analysis - Architectures and Use Cases

4.1 The Informer Architecture

Recently, Zhou et al. proposed *Informer*, which uses a *ProbSparse* self-attention mechanism to optimize the computational complexity and memory usage of the standard Transformer architecture [80]. The authors also introduced the self-attention distilling operation, which considerably reduces the total space complexity of the model.

ProbSparse self-attention uses the dominant dot-product pairs by randomly selecting $\log L$ top query vectors and setting the rest of the query vector values to zero. The computational complexity and memory usage reduce with L value vectors to $\mathcal{O}(L \log L)$. With a stack of J encoders, the total memory usage reduces to $\mathcal{O}(JL \log L)$. Furthermore, the self-attention distilling operation removes redundant combinations of value

vectors. This operation is inspired by dilated convolution as proposed in [75] and [20]. The output of the multi-head self-attention is fed into 1-D convolution filters with kernel size = 3. Later, an exponential linear unit (ELU) activation function is applied, followed by a max-pooling operation with a stride of 2. These operations reduce the size in half and thus, form a pyramid, as shown in Figure 12. Effectively, the total space complexity reduces considerably. Stacked replicas are also built, with an input length half of the previous stack. Figure 12 shows only one replica stack. The output of the main stack and the replica stacks have the same dimension and are concatenated to form the final output of the encoder. These replica stacks enhance the robustness of the distilling operation.

The decoder consists of two stacked multi-head attention layers. The input to the decoder is a start token concatenated with a placeholder for the predicted target sequence (with initial values set to zero). It predicts all outputs by one forward procedure, as shown in Figure 12, considerably reducing inference time.

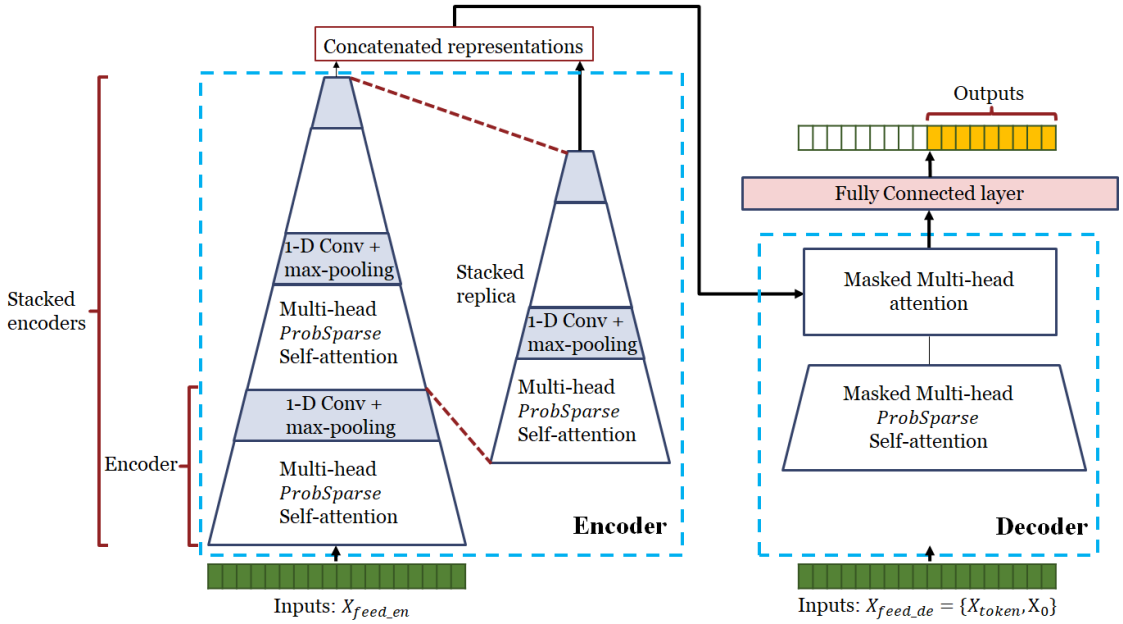


Figure 12: Informer architecture as proposed in [80] is presented. Informer consists of stacked encoders and decoders. The encoder comprises of multi-head ProbSparse self-attention and self-attention distilling operations. Stacked replicas are built with input length half of the previous layer. Encoder output is a concatenation of the outputs from the main and replica stacks. The decoder has two stacked multi-head attention layers and predicts the entire output sequence simultaneously in one forward pass.

In the Informer architecture, the scalar input is mapped to d dimensional vectors $\mathbf{u}_i$, using 1-D convolution filters. The local context is retained using the fixed positional encoding (PE) based on sinusoidal functions. A global timestamp, called stamp embedding (SE), is also included to capture hierarchical time information such as the week, month, or year as well as occasional events such as holidays. The input to the encoder is a sum of the scalar projection vectors ($\mathbf{u}_i$), PE_i, and SE.

The Informer architecture was tested on various datasets, including, electricity consumption load, and the weather dataset. The model performed better than state-of-the-art (SOTA), including Autoregressive Integrated Moving Average (ARIMA) [3], Prophet [61], LSTMa [6], LSTnet [31], and DeepAR [50].

4.2 LogSparse Transformer Architecture

Li et al. proposed LogSparse Transformers to overcome memory challenges, thus making Transformers more feasible for time-series with long-term dependencies [32]. LogSparse Transformers allow each time-step to attend to previous time-steps, selected using an exponential step size, reducing the memory utilization from $\mathcal{O}(L^2)$ to $\mathcal{O}(L \log_2 L)$ in each self-attention layer. Fig 13 shows the various ways in which LogSparse self-attention can be used for time-series analysis. The canonical self-attention mechanism used for neighboring

time-steps in a specific range allows for gathering more information. Beyond that range, the LogSparse self-attention mechanism is applied. Another way is to restart LogSparse step size after a particular range of time-steps.

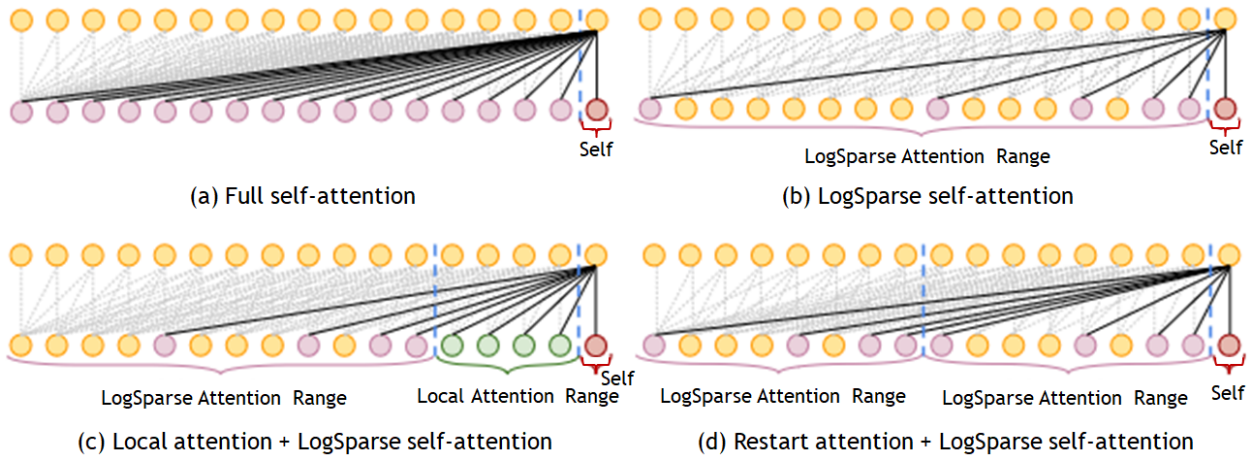


Figure 13: Various ways to apply LogSparse self-attention as proposed by Li et al.[32].

The patterns in time-series data may evolve significantly with time due to different events, like holidays or extreme weather. Therefore, it may be beneficial to capture information from the surrounding time points to determine whether the observed point is an anomaly, changing point, or a part of the pattern. Such behavior is captured using the causal convolutional self-attention mechanism, as shown in Figure 14.

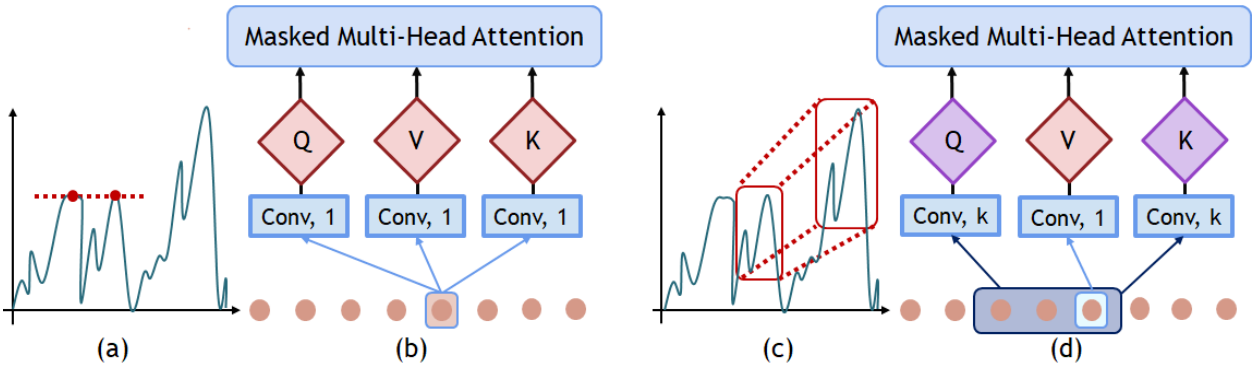


Figure 14: The canonical self-attention mechanism is similar to using convolution with kernel size equal to one, as shown in (b) [32]. It is only able to capture point-wise similarity, as shown in (a). Local context is captured using convolution with kernel size greater than one proposed by Li et al. and depicted in (d). Awareness of the local context helps capture feature similarities more accurately based on shape matching, as shown in (c).

The causal convolutional self-attention mechanism proposed ensures that the current position does not have access to future information. The convolution operation with a kernel size of more than one captures the local context information in the query and key vectors generated from the input. Value vectors are generated using a kernel size equal to one. With this mechanism, more accurate forecasting is performed.

4.3 Simply Attend and Diagnose (SAnD)

Clinical data such as Intensive Care Unit (ICU) measurements comprise of multi-variate, time-series observations coming from sensor measurements, test results, and subjective assessments. Song et al. introduced Transformers to predict various variables of clinical interest from the MIMIC-III benchmark dataset [29],

named Simply Attend and Diagnose or SAnD [57]. Preprocessed data is passed to an input embedding layer, which uses 1-D convolution for mapping inputs into d dimensional vectors ($d >$ number of variables in the multi-variate time-series data). After the addition of hard-coded positional encoding to the embedded data, it is passed through the attention module. The multi-head attention layer uses restricted self-attention to introduce causality, i.e., perform computations using information earlier than the current time. The output of the stacked attention modules passes on to the 'dense interpolation layer'. This layer, together with positional encoding, is used to capture the temporal structure of clinical data. A linear layer and soft-max/sigmoid are used as the final layers for classification. The proposed Transformer model was evaluated on all MIMIC-III benchmark tasks and was reported to perform better than RNNs.

4.4 Traffic Transformer

Traffic forecasting predicts future traffic, given a sequence of historical traffic observations like speed, density, and volume detected by sensors on a road network. In this case, M previous time-steps in a sequence are used to predict H future time-steps. It is important to encode the continuity and periodicity of time-series data and capture the spatio-temporal dependencies in traffic forecasting. The Traffic Transformer [9] has been built based on two existing networks, the first being a Graph Neural Network [52, 74] followed by the Transformer as shown in Figure 15. The Transformer models the temporal dependencies, and the Graph Neural Network is used to model spatial dependencies.

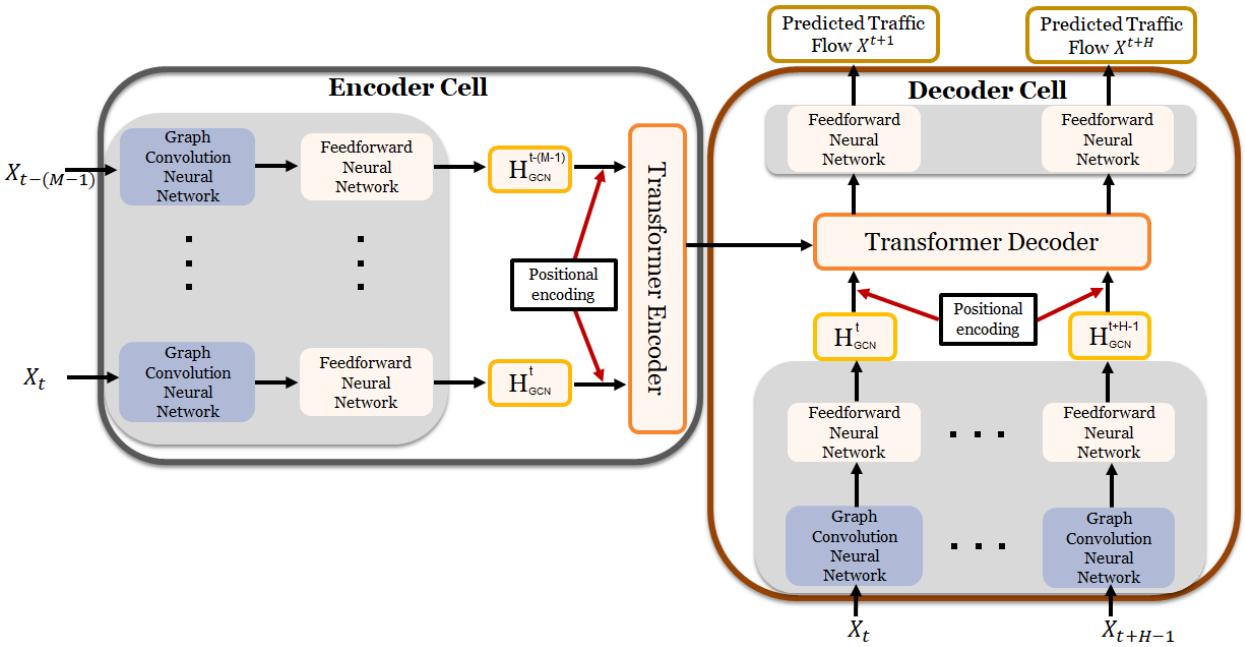


Figure 15: The Traffic Transformer introduced in [9] is presented. The encoder comprises of Graph Neural Network for capturing the spatial dependencies in the traffic data, followed by feed-forward layer. The output is fed into the Transformer encoder to capture the temporal dependencies. The output of the encoder is fed into the Transformer decoder, which has a similar composition to the encoder. Position information is included using either addition or similarity-based combination.

The output of the Graph Neural Network forms the input for the Transformer after the incorporation of positional encoding. There are two methods for adding the sequence information, (1) positional encoding vectors are added to the input vectors, and (2) attention weights for positional encoding vectors are calculated using the dot product. The positional attention weights are used to tweak the attention weights for the input vectors to the Transformer.

Four strategies for encoding temporal information of traffic data are introduced. Different combinations of these strategies can be used for various problems or types of the dataset.

a) Continuity of time-series:

   i. Relative position encoding: This strategy encodes the relative position of a time-step in the window of the source-target sequence regardless of the position of that time-step in the entire time-series. Hence, the same time-step might be assigned with a different position embedding depending on its position in a sequence pair. Position encoding is done using sine cosine functions.

   ii. Global position encoding: The entire sequence is encoded using sine cosine functions. In this way, both local and global positions of time-series are captured.

b) Periodicity of time-series:

   i. Periodic position encoding: A mechanism to encode the daily and weekly periodicity of data is introduced. Daily periodicity is captured by encoding two hundred and eighty-eight $\frac{24\times60}{5}$ positions. Weekly periodicity is captured by encoding seven positions (number of days in a week).

   ii. Time-series segments: This is done by concatenating the daily and weekly data segments to the 'M' recent time-steps.

The Traffic Transformer was tested with two real-world benchmark datasets, METR-LA [27] and the California Transportation Agencies Performance Measurement System ([PeMS]).

4.5 Self-Attention for Raw Optical Satellite Time-Series Classification

Vegetation life cycle events can be used as distinct temporal signals to identify types of vegetation. These temporal signals contain relevant features for differentiating various kinds of vegetation. The classification of vegetation type [49] from raw optical satellite images is done using Transformers. The performance of the Transformer architecture for vegetation classification, compared to LSTM-RNN [23], MS-ResNet [67], DuPLO [26], TempCNN [42], and random forest methods, is better for raw data. However, for pre-processed data, the performance of all methods is quite similar.

4.6 Deep Transformer Models for Time-Series Forecasting: The Influenza Prevalence Case

Transformers have also been used to predict cases of influenza [69] using the data of weekly count of flu cases in a particular area. A single week's prediction is made using ten previous week's data as the input in the first experiment. The Transformer architecture performed better in terms of 'root mean square error' (RMSE) compared to ARIMA, LSTMs, and sequence to sequence models with attention. In the second experiment, Transformer architecture was tested with multivariate time-series data by introducing 'week number' as a time-indexed feature and including the first and second-order differences of the time-series as two explicit numerical features. However, this did not show significant improvement in the results. In the third experiment, Time-delay embedding (TDE) is introduced and formed by embedding each scalar input x_t into a d-dimensional time-delay space as in Eq. 14.

$$\mathrm{TDE}_{d,\tau}x(t) = \left(x_t, x_{t-1}, \cdots, x_{t-(d-1)\tau}\right) \tag{14}$$

Time-delay embedding of different dimensions d, from 2 till 32, are formed with $\tau = 1$. In all cases, the RMSE value reached a minimum with the dimension of 8, which conforms to similar results of an independent study on clinical data.

5 Best Practices for Training Time-Series Transformers

The Transformer architecture is becoming increasingly popular, leading many researchers to seek ways to optimize these networks for various applications. Whether in adapting the architecture for use in a specific problem domain, training techniques, hyperparameter optimization, inference methods, hardware adaptation, and others. In this section we discuss best practices when training transformers for time-series analysis.

5.1 Training Transformers

The Transformers may not be easy to train from scratch for a beginner. The original transformer architecture [64] utilized many different strategies to stabilize the gradients during training for deeper networks. The authors' residual connections allow for training a deeper network. Later, layer normalization operations were added alongside an adaptive optimizer (Adam) to provide different learning rates for different parameters. Like most other deep learning models, Transformers are also sensitive to the learning rate. Given an optimal learning rate, Transformers can converge faster than traditional sequence models. During the first few epochs, it is common to observe performance drop. However, after a few epochs, the model will generally start to converge to better values. In the original implementation, the authors used a warm-up learning rate strategy that increases linearly for the first N training steps and then decreases proportionally to the inverse square root of the step number, $\frac{1}{\sqrt{N}}$.

5.2 Improvements in the Transformer Architecture for Better Training

Many advancements have been proposed in the Transformer architecture to resolve some of the problems related to achieving stable training with deeper architectures. Mainly optimizations have been made to balance residual dependencies by relocating layer normalization operations and finding better weight initialization techniques. These improvements have led to more stable training and, in some cases, eliminated the need for using some of the strategies proposed in the original architecture. Figure 16 provides an overview of some of the best practices for training Transformers and their respective reasons.

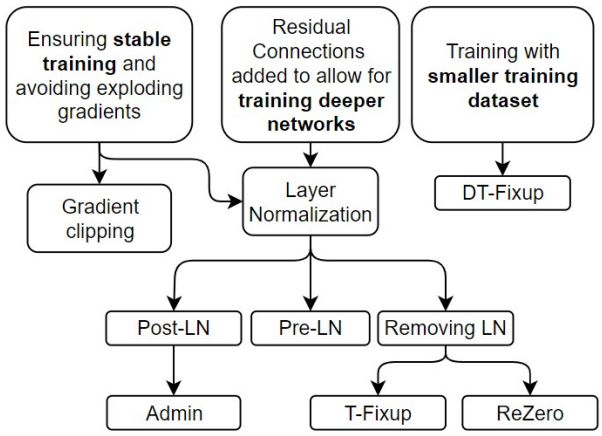


Figure 16: An overview of best practices for training Transformers.

The original transformer architecture can be referred to as post-layer normalization (post-LN), where the layer normalization is located outside the residual block [71]. Post-LN converges much slower and requires a learning rate warm-up strategy [71]. Xiong *et al.* proposed a pre-layer normalization (pre-LN) Transformer to address this issue, showing that it could help gradients converge faster while requiring no warm-up. Pre-LN Transformer can achieve this by helping control the gradient magnitudes and balancing the residual dependencies [71]. Although, Pre-LN Transformer architecture does not require learning rate warm-up, they have inferior empirical performance as compared to post-LN Transformer architecture.

Liu *et al.* proposed an adaptive model initialization (Admin) to benefit from the ease of training of the post-LN Transformer while achieving the performance of the pre-LN Transformer [37]. *Adaptive model initialization* is a technique used in other areas of machine learning to initialize the model such that the dependencies between the input and output variables are better captured [37]. This initialization technique helps the model learn the relationships between the variables more effectively and improves performance. Another option is to remove the layer normalization altogether. *ReZero* replaces the layer normalization operation with a trainable parameter, α, initialized at 0 in each residual layer [5]. Consequently, the entire network is initialized to compute the identity function, with a gradual and adaptive introduction of the

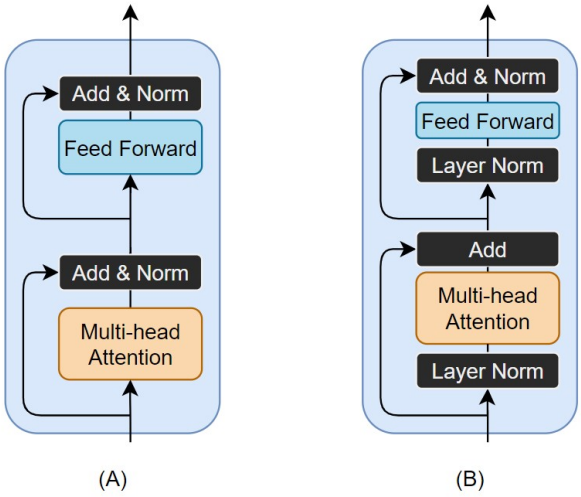


Figure 17: post-LN transformer (A) vs. pre-LN transformer (B)

contributions of the self-attention and MLP layers. The residual dependencies seem to balance well with Admin or ReZero for training deeper Transformer models with better generalization performance.

5.3 Best Practices for Training Transformers

5.3.1 Large Model Size

After selecting the Transformer model architecture, the challenge becomes dealing with the large model size. It may seem that smaller models would train faster than larger ones; however, this does not always hold true. For example, Li *et al.* [33] demonstrated that, in some instances, training large models and then compressing results leads to better results. The so called *lottery ticket hypothesis* [17], *pruning* can be applied to reduce the model size. Also, *quantization* to a lower precision allows for a smaller model. However, there are inevitable trade offs when using a larger model after pruning/quantization versus a smaller one. The most relevant is ensuring that the training dataset is large to avoid overfitting. Using a small dataset can lead to poor generalization. In general, when trying to train a Transformer model, we recommend using large models instead of the more conventional approach of starting with smaller models and then adding layers.

5.3.2 Training with Small Datasets

We have seen several solutions that allow the training of deeper Transformer models with improved performance as compared to the vanilla transformer architecture. Training these deep transformer models from scratch require large datasets, making training with a small dataset challenging. Small datasets require pre-trained models and small batch sizes to perform well, but these two requirements make training additional Transformer layers more difficult. Using a small batch size causes the variance of the updates to be larger, and even with a large batch size, the model may typically generalize poorly. In some cases, better initialization techniques can optimize the model performing well on the smaller datasets. The use of Xavier initialization is the most common scheme for Transformers [19], although recently, T-Fixup has been shown to outperform Xavier [24]. T-Fixup introduced by Huang *et al.*, found that when training Transformer without learning rate warm-up, the variance in the Adam optimizer was amplified by large initial learning rate leading to large updates at the beginning of training [25]. Hence, the learning rate warm-up requirement comes from the Adam optimizer's instability combined with gradient vanishing through layer normalization. To resolve this, the authors proposed a new weight initialization scheme with theoretical guarantees that keeps model updates bounded and allows to remove both warm-up and layer normalization [24]. Following the work of T-Fixup, a data-dependent initialization technique was developed by Borealis AI known as DT-Fixup [73].

DT-Fixup allows for training deeper transformer models with small datasets, given correct optimization procedures are followed.

5.3.3 Other Strategies to Consider

Batch Size: Popel *et al.* found that the optimal batch size depends on the complexity of the model [45]. They considered two classes of models, one a "base" and one a "big." For the base model, a larger batch size (up to 4,500) performed well, whereas a different set of parameters yielded superior results for the big model. Namely, there needed to be a minimum batch size (1,450 in their experiments) before the model started to converge. While batch size is almost entirely empirically selected, a large minimum should be used with Transformer models.

Learning Rate: Considerable research has been done in evaluating the effect of learning rate on model performance [78, 56, 7] and how they relate to each other. A study conducted by Popel *et al.* demonstrated that small learning rates tend to have slower convergence, whereas learning rates that are too high may lead to non-convergence [45].

Gradient Clipping: Many implementations of Transformers use gradient clipping during the training. This process tends to avoid exploding gradients and potential divergence if the number of steps is too large. Other methods of gradient clipping, such as choosing a step size proportional to the batch size, are not recommended for Transformers as these can lead to slower convergence [46].

6 Conclusion

Transformer architecture has found its application in solving various time-series tasks. This architecture combines the parallel computing capabilities in a convolutional network with capturing long-term dependencies in RNNs and its variants (LSTMs & GRUs). The Transformer architecture based on self-attention and positional encoding offer better or similar performance as RNNs (and variants LSTMs/GRUs). However, it is more efficient in computing time and overcomes other shortcomings of RNNs/LSTMs/GRUs. Various modifications have been made in the initially proposed architecture to enable Transformers to handle time-series tasks effectively. This architecture poses some challenges in training for which best practices are included to enable effective training of Transformers.

References

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[2] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. Ieee, 2017.

[3] Adebiyi A Ariyo, Adewumi O Adewumi, and Charles K Ayo. Stock price prediction using the arima model. In *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation*, pages 106–112. IEEE, 2014.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[5] Thomas Bachlechner, Bodhisattwa Prasad Majumder, Henry Mao, Gary Cottrell, and Julian McAuley. Rezero is all you need: fast convergence at large depth. In Cassio de Campos and Marloes H. Maathuis, editors, *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, volume 161 of *Proceedings of Machine Learning Research*, pages 1352–1361. PMLR, 27–30 Jul 2021.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[7] Laxmidhar Behera, Swagat Kumar, and Awhan Patnaik. On adaptive learning rate that guarantees convergence in feedforward networks. *IEEE transactions on neural networks*, 17(5):1116–1125, 2006.

[8] Christoph Bergmeir, Rob J Hyndman, and Bonsoo Koo. A note on the validity of cross-validation for evaluating autoregressive time series prediction. *Computational Statistics & Data Analysis*, 120:70–83, 2018.

[9] Ling Cai, Krzysztof Janowicz, Gengchen Mai, Bo Yan, and Rui Zhu. Traffic transformer: Capturing the continuity and periodicity of time series for traffic forecasting. *Transactions in GIS*, 24(3):736–755, 2020.

[10] Gang Chen. A gentle tutorial of recurrent neural network with error backpropagation. *arXiv preprint arXiv:1610.02583*, 2016.

[11] Kai Chen, Guang Chen, Dan Xu, Lijun Zhang, Yuyao Huang, and Alois Knoll. Nast: non-autoregressive spatial-temporal transformer for time series forecasting. *arXiv preprint arXiv:2102.05624*, 2021.

[12] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[13] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[14] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR, 2017.

[15] Jacob Devlin, Ming Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, 1(Mlm):4171–4186, 2019.

[16] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[17] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[18] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. *34th International Conference on Machine Learning, ICML 2017*, 3:2029–2042, 2017.

[19] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[20] Ankit Gupta and Alexander M Rush. Dilated convolutions for modeling long-distance genomic dependencies. *arXiv preprint arXiv:1710.01278*, 2017.

[21] Jie Hao, Xing Wang, Baosong Yang, Longyue Wang, Jinfeng Zhang, and Zhaopeng Tu. Modeling recurrence for transformer. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[22] Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers. *arXiv preprint arXiv:1912.12180*, 2019.

[23] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[24] Xiao Shi Huang, Felipe Perez, Jimmy Ba, and Maksims Volkovs. Improving transformer optimization through better initialization. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4475–4483. PMLR, 13–18 Jul 2020.

[25] Xiao Shi Huang, Felipe Perez, Jimmy Ba, and Maksims Volkovs. Improving transformer optimization through better initialization. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4475–4483. PMLR, 13–18 Jul 2020.

[26] Roberto Interdonato, Dino Ienco, Raffaele Gaetano, and Kenji Ose. Duplo: A dual view point deep learning architecture for time series classification. *ISPRS journal of photogrammetry and remote sensing*, 149:91–104, 2019.

[27] Hosagrahar V Jagadish, Johannes Gehrke, Alexandros Labrinidis, Yannis Papakonstantinou, Jignesh M Patel, Raghu Ramakrishnan, and Cyrus Shahabi. Big data and its technical challenges. *Communications of the ACM*, 57(7):86–94, 2014.

[28] Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.

[29] Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3:160035, 2016.

[30] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR, 2020.

[31] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 95–104, 2018.

[32] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *arXiv preprint arXiv:1907.00235*, 2019.

[33] Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joseph E. Gonzalez. Train large, then compress: Rethinking model size for efficient training and inference of transformers. *CoRR*, abs/2002.11794, 2020.

[34] Bryan Lim, Sercan O Arik, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *arXiv preprint arXiv:1912.09363*, 2019.

[35] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[36] Adam Liška, Germán Kruszewski, and Marco Baroni. Memorize or generalize? searching for a compositional rnn in a haystack. *arXiv preprint arXiv:1802.06467*, 2018.

[37] Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. Understanding the difficulty of training transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5747–5763, Online, November 2020. Association for Computational Linguistics.

[38] Minghao Liu, Shengqi Ren, Siyuan Ma, Jiahui Jiao, Yizhou Chen, Zhiguang Wang, and Wei Song. Gated transformer networks for multivariate time series classification. *arXiv preprint arXiv:2103.14438*, 2021.

[39] Kiran Madhusudhanan, Johannes Burchert, Nghia Duong-Trung, Stefan Born, and Lars Schmidt-Thieme. Yformer: U-net inspired transformer architecture for far horizon time series forecasting. *arXiv preprint arXiv:2110.08255*, 2021.

[40] Tomas Mikolov, Kai Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *ICLR*, 2013.

[41] Ian E Nielsen, Dimah Dera, Ghulam Rasool, Nidhal Bouaynaya, and Ravi P Ramachandran. Robust explainability: A tutorial on gradient-based attribution methods for deep neural networks. *arXiv preprint arXiv:2107.11400*, 2021.

[42] Charlotte Pelletier, Geoffrey I Webb, and François Petitjean. Temporal convolutional neural network for the classification of satellite image time series. *Remote Sensing*, 11(5):523, 2019.

[43] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[44] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *NAACL HLT 2018 - 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, 1:2227–2237, 2018.

[45] Martin Popel and Ondřej Bojar. Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110(1):43–70, Apr 2018.

[46] Martin Popel and Ondřej Bojar. Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110, 03 2018.

[47] Xinyuan Qi, Kai Hou, Tong Liu, Zhongzhong Yu, Sihao Hu, and Wenwu Ou. From known to unknown: Knowledge-guided transformer for time-series sales forecasting in alibaba. *arXiv preprint arXiv:2109.08381*, 2021.

[48] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.

[49] Marc Rußwurm and Marco Körner. Self-attention for raw optical satellite time series classification. *ISPRS Journal of Photogrammetry and Remote Sensing*, 169:421–435, 2020.

[50] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020.

[51] Gerard Salton. Some experiments in the generation of word and document associations. In *Proceedings of the December 4-6, 1962, Fall Joint Computer Conference*, AFIPS '62 (Fall), page 234–250, New York, NY, USA, 1962. Association for Computing Machinery.

[52] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[53] Sharath M Shankaranarayana and Davor Runje. Attention augmented convolutional transformer for tabular time-series. In *2021 International Conference on Data Mining Workshops (ICDMW)*, pages 537–541. IEEE, 2021.

[54] Li Shen and Yangzhu Wang. Tcct: Tightly-coupled convolutional transformer on time series forecasting. *Neurocomputing*, 2022.

[55] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

[56] Leslie N Smith. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.

[57] Huan Song, Deepta Rajan, Jayaraman J. Thiagarajan, and Andreas Spanias. Attend and diagnose: Clinical time series analysis using attention models, 2017.

[58] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[59] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[60] Sho Takase and Naoaki Okazaki. Positional encoding to control output sequence length. *arXiv preprint arXiv:1904.07418*, 2019.

[61] Sean J Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

[62] Sindhu Tipirneni and Chandan K Reddy. Self-supervised transformer for multivariate clinical time-series with missing values. *arXiv preprint arXiv:2107.14293*, 2021.

[63] Michael Tschannen, Olivier Bachem, and Mario Lucic. Recent advances in autoencoder-based representation learning. *arXiv preprint arXiv:1812.05069*, 2018.

[64] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[65] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

[66] Yu-An Wang and Yun-Nung Chen. What do position embeddings learn? an empirical study of pre-trained language model positional encoding. *arXiv preprint arXiv:2010.04903*, 2020.

[67] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*, pages 1578–1585. IEEE, 2017.

[68] Zhiwei Wang, Yao Ma, Zitao Liu, and Jiliang Tang. R-transformer: Recurrent neural network enhanced transformer. *arXiv preprint arXiv:1907.05572*, 2019.

[69] Neo Wu, Bradley Green, Xue Ben, and Shawn O'Banion. Deep transformer models for time series forecasting: The influenza prevalence case. *arXiv preprint arXiv:2001.08317*, 2020.

[70] Sifan Wu, Xi Xiao, Qianggang Ding, Peilin Zhao, Ying Wei, and Junzhou Huang. Adversarial sparse transformer for time series forecasting. *Advances in Neural Information Processing Systems*, 33:17105–17115, 2020.

[71] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 10524–10533. PMLR, 13–18 Jul 2020.

[72] Jiehui Xu, Haixu Wu, Jianmin Wang, and Mingsheng Long. Anomaly transformer: Time series anomaly detection with association discrepancy. *arXiv preprint arXiv:2110.02642*, 2021.

[73] Peng Xu, Dhruv Kumar, Wei Yang, Wenjie Zi, Keyi Tang, Chenyang Huang, Jackie Chi Kit Cheung, Simon J.D. Prince, and Yanshuai Cao. Optimizing deeper transformers on small datasets. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2089–2102, Online, August 2021. Association for Computational Linguistics.

[74] Hongxia Yang. Aligraph: A comprehensive graph neural network platform. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3165–3166, 2019.

[75] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. Dilated residual networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 636–644, 2017.

[76] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 558–567, 2021.

[77] Yuan Yuan and Lei Lin. Self-supervised pretraining of transformers for satellite image time series classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 14:474–487, 2020.

[78] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[79] Jianqiao Zheng, Sameera Ramasinghe, and Simon Lucey. Rethinking positional encoding. *arXiv preprint arXiv:2107.02561*, 2021.

[80] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *The Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Virtual Conference*, volume 35, pages 11106–11115. AAAI Press, 2021.