Circuits, Systems, and Signal Processing (2023) 42:7433–7466
https://doi.org/10.1007/s00034-023-02454-8

Transformers in Time-Series Analysis: A Tutorial

Sabeen Ahmed[1] · Ian E. Nielsen[2] · Aakash Tripathi[1] · Shamoon Siddiqui[2] · Ravi P. Ramachandran[2] · Ghulam Rasool[1]

Received: 23 December 2022 / Revised: 4 July 2023 / Accepted: 10 July 2023 /
Published online: 25 July 2023

Abstract

Transformer architectures have 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 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.

Keywords Transformer · Time-series · Self-attention · Positional encoding

Sabeen Ahmed, Ian E. Nielsen and Aakash Tripathi have contributed equally to this work.

✉ Sabeen Ahmed
sabeen.ahmed@moffitt.org

Ian E. Nielsen
nielseni6@rowan.edu

Aakash Tripathi
aakash.tripathi@moffitt.org

Shamoon Siddiqui
siddiq76@rowan.edu

Ravi P. Ramachandran
ravi@rowan.edu

Ghulam Rasool
ghulam.rasool@moffitt.org

[1] Department of Machine Learning, Moffitt Cancer Center, 12902 USF Magnolia Drive, Tampa, FL 33612, USA

[2] Department of Electrical and Computer Engineering, Rowan University, 201 Mullica Hill Rd, Glassboro, NJ 08028, USA

1 Introduction

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 [88]. Recently, Transformers have been successfully employed to tackle various problems in machine learning and achieve state-of-the-art performance [50]. Apart from classical NLP tasks, examples from other areas include image classification [25], object detection and segmentation [43], image and language generation [56], sequential decision-making in reinforcement learning [16], multi-modal (text, speech, and image) data processing [96], and analysis of tabular and time-series data [61]. This tutorial paper focuses on time-series analysis using Transformers.

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. Time-series analysis may include processing the numeric data for multiple tasks, including forecasting, prediction, and 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) [19, 21, 35, 51]. 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) [14]. LSTMs and GRUs have been successfully used in many applications [5, 13, 23, 24, 78]. 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 [28, 69]. 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 accelerators [27]. 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 data sequences with the limited use of parallelization offered by modern hardware impact the effectiveness and efficiency of RNN-based models [77].

The Transformer architecture allows for parallel computation of sequential data [84] without a substantial increase in the complexity of the network [44]. Transformers, owing to their architecture, can use parallel processing capabilities of Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs) [41]. Given the attention-based operations, Transformers can correlate information from all elements in the sequence to each other in parallel without suffering from vanishing gradients as is the case with RNNs and their variants [9, 62, 92].

Transformers have substantially improved long-term and multi-variate time-series forecasting [59, 108]. However, the self-attention mechanism has high computational complexity and memory requirements hampering long sequence modeling. Various modifications have been proposed in the literature to optimize Transformer performance for time-series tasks [55, 111], [17]. Training large Transformer models is challenging, especially for massive datasets. Many techniques have been proposed in the literature to efficiently train large Transformer models. These techniques include layer-wise adaptive large batch optimization [103], distributed training [37], knowledge inheritance [68], progressive training [46], and mapping parameters of smaller models to initialize larger models [91].

The contribution of this tutorial includes,

1. Illustrative explanation of the intuition behind the Transformer operations. These intuitions help us understand how Transformers revolutionized NLP tasks.
2. Discuss various aspects and techniques introduced in the Transformer architecture and internal operations for efficient time-series analyses.
3. Compilation of some use cases of Transformers for time-series analysis, including comparative performance.
4. Guidelines on techniques and tricks for efficiently training Transformers.

Transformers-based models have revolutionized applications in NLP, computer vision, time-series analysis, and many other areas. However, we have limited our study to time-series applications. Another limitation of the study is related to the variations in the architectures of Transformer models. In recent years, hundreds of variations have been proposed for performance improvement of Transformers. However, we focused on the basic architecture proposed by Vaswani et al. [88].

We start by providing an overview of the Transformer architecture based upon self-attention, scaled dot-product, multi-head, and positional encoding in Sect. 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 Sect. 4. Finally, we provide "best practices" for training Transformers in Sect. 5 before concluding the paper.

2 Transformers: Nuts and Bolts

We begin by explaining the inner workings of the Transformer as proposed by Vaswani et al. [88] to solve the challenging problem of neural machine translation. We then proceed with a deep dive into the operations performed inside each component of

Transformers and the intuition behind those operations. Several variations of the Transformer architecture have been developed. However, the intuition behind the basic operations used remains the same and is covered in this section [18, 34, 70].

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 [88]. 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 [11]. 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 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 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 [42, 52]. Transformers address these challenges by using *self-attention* and *positional encoding* techniques to jointly attend to and encode the ordered information as current data samples in the sequence are analyzed. These techniques keep the sequential information intact for learning while eliminating the classical notion of *recurrence* [33]. 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 [94].

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 LSTM or GRU) and a Transformer as illustrated in Fig. 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 [58, 64, 65, 73]. 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 data processing. The sequence order is tracked using *positional encoding*.

2.2 Self-attention Operation

The Transformer architecture is based on finding associations or relationships between various input segments (after adding the position information in these segments) using the dot product [30]. Let $\{\mathbf{x}_i\}_{i=1}^{n}, \mathbf{x} \in \mathcal{R}^d$ be a set of n words (or data points) in a

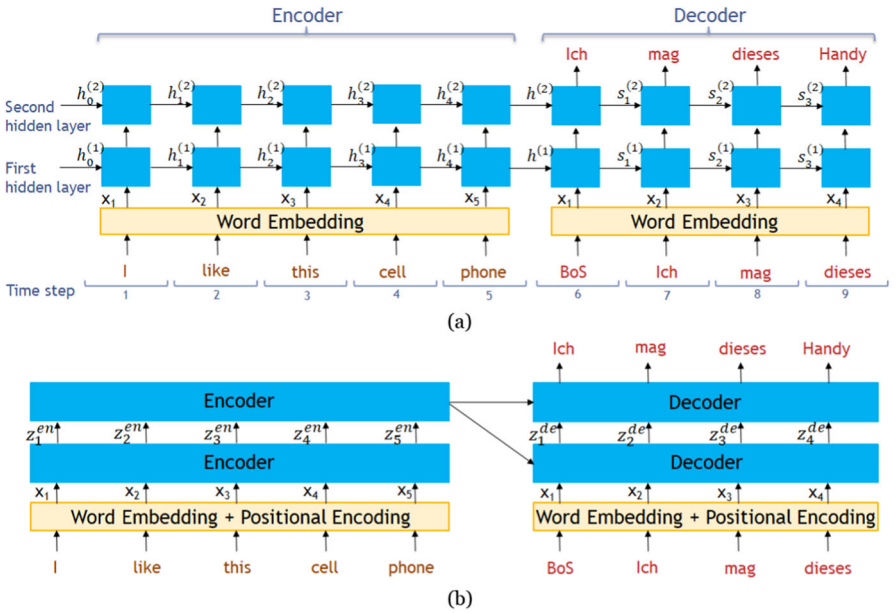

Fig. 1 An example of a simple language translation task performed using **a** a classical model, such as an RNN, LSTM, or GRU, and **b** a Transformer

single sequence. The subscript i represents the position of the vector $\mathbf{x}_i$, 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 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 $j = 1, \ldots, n$:

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

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 $\{\mathbf{x}_i\}_{j=1}^{n}$:

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

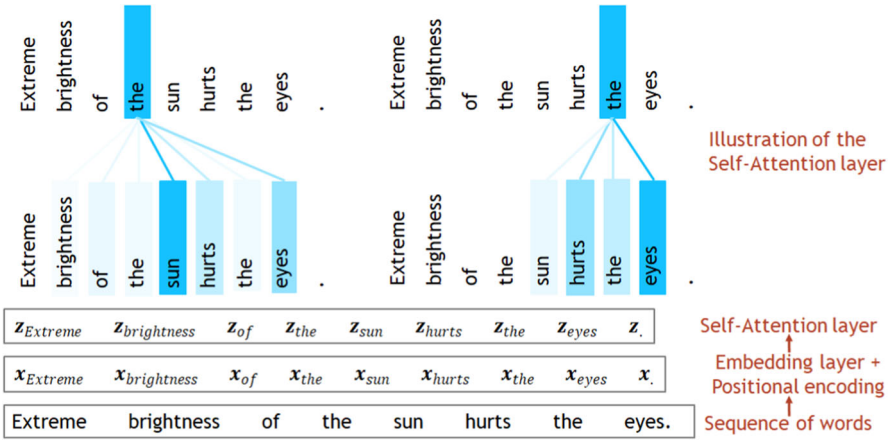

Fig. 2 The role of self-attention for the example sentence "Extreme brightness of the sun hurts the eyes". The self-attention operation determines the relative correlation of each word with all the words in the sequence. In this example, the first occurrence of the word *the* is most correlated to the 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 [89]

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 correlation value, as measured by the normalized dot product between $\mathbf{x}_i$ and $\mathbf{x}_j$. Note that $\mathbf{z}_i$ in Eq. 2 retains 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 $\mathbf{z}_{\text{sun}}$ would be weighted more by the first occurrence of 'the' as compared to the second occurrence of 'the'. Similarly, $\mathbf{z}_{\text{eyes}}$ would be more heavily weighted by $\mathbf{x}_{\text{hurts}}$ and $\mathbf{x}_{\text{the}}$ (second occurrence of 'the') than $\mathbf{x}_{\text{of}}$ and the first occurrence of 'the' (Fig. 3).

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 value $\mathbf{v} \in \mathbb{R}^{s}$. Intuitively, a **query** is a question that 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'.

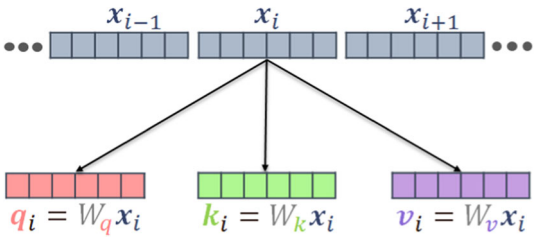

Fig. 3 A set of weighted linear transformations applied to the input vector of length $d = 6$ are presented. The resulting vectors are referred to as **query**, **key**, and **value**, each of size $s_1 = s = d$. There are a total of n such inputs in each sequence processed by the Transformer, which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

For an input $\mathbf{x}_i$, the 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 the 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

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

←d_q is the dimension of each of the query/key vectors

In matrix form, we have:

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

←d_k is the dimension of each of the query/key vectors

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 different ways. Multiple self-attention heads are introduced that operate on the same input in parallel and use distinct

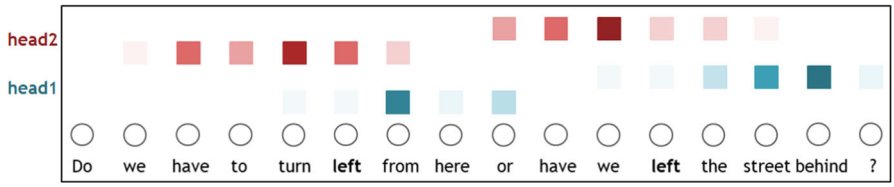

Fig. 4 Multi-head self-attention example. Two heads can learn different attention weights for two different uses of the words "left" based on their meaning in the sentence. The relative attention weights shown are generated using the Transformer attention visualization tool [89]

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. As shown in Fig. 4, Transformers can capture such information using multiple heads. 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. Using multiple heads in a Transformer is analogous to using multiple kernels at each layer in a convolutional neural network, where each kernel is responsible for learning distinct features or representations [3].

The following three steps can describe the operations involved in multi-head self-attention.

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 Fig. 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 Fig. 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 a 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).

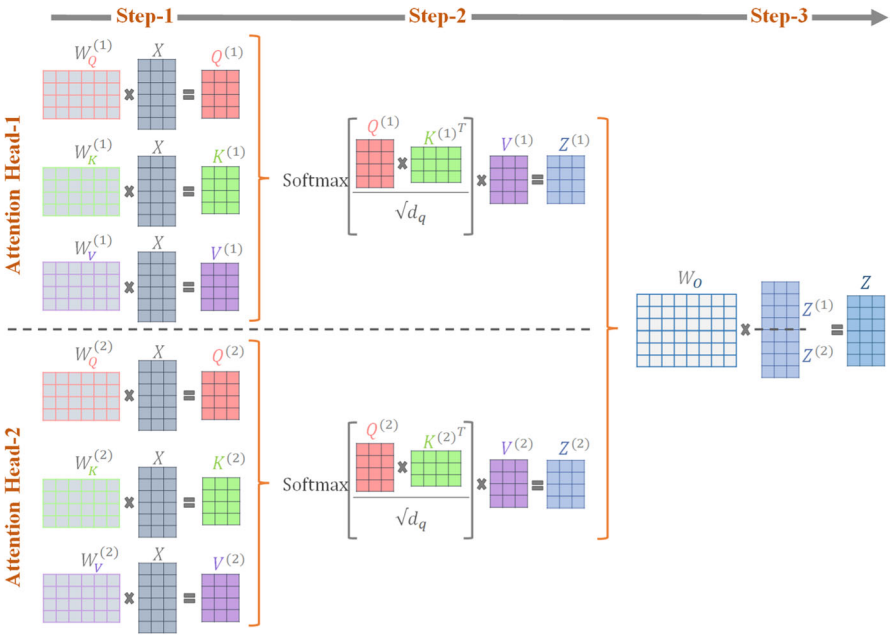

Fig. 5 Multi-head attention steps. The example consists of two heads, and the input sequence comprises of three words. The input $\{\mathbf{x}_i\}_{i=1}^{n}$ is mapped to word embedding, and positional encoding vectors are added, resulting in the input matrix $X \in \mathbb{R}^{6\times 3}$. Two sets 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

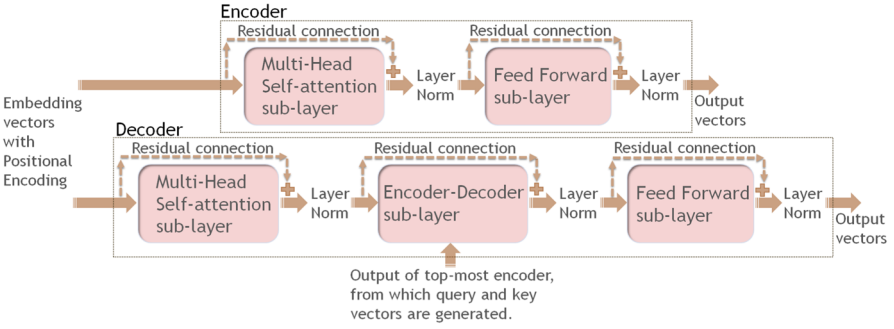

Fig. 6 An encoder and decoder block is presented. Each encoder block is built using multi-head self-attention and feed-forward layers with residual connections around each layer followed by the layer normalization operation. The decoder block is similar to the encoder block with an additional encoder–decoder attention layer. This encoder–decoder attention layer receives the output of the topmost encoder block and uses it to generate the key and value vectors. The query vectors are produced from the output of the multi-head self-attention layer preceding the encoder–decoder layer

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 Fig. 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 [82]. The layer normalization operation is also commonly used in neural networks for processing sequential data. It helps with the faster convergence of the model training [6]. The feed-forward layer comprises two linear layers with a ReLU activation function [1]. The output of an encoder block is used as an input to 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, three layers 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 Fig. 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 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 them through a word embedding layer and the addition of PE, as shown in Fig. 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 Fig. 7 [87]. The stacked

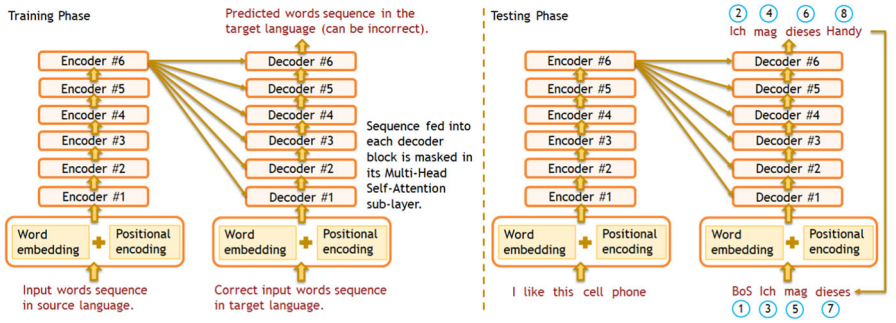

Fig. 7 Stacked encoder and decoder blocks used in Transformers are presented for the machine translation task. The number of stacked blocks may vary depending on the task, analogous to multiple layers used in traditional neural networks. Input to the stack of encoders is word embeddings enriched with positional information. The output of the topmost 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

encoder/decoder blocks resemble 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 the 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. Depending on the desired task, the operations on the output from the last decoder block can be selected for classification or regression.

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 the input data in a sequence. Transformers (1) use the concept of positional encoding to add the positional information to the input and (2) process the modified input in parallel, thereby avoiding the challenges of processing data sequentially. The technique consists of calculating n PE vectors (denoted as $p \in \mathbb{R}^d$) and adding these to the input $\{\mathbf{x}_i\}_{i=1}^n$.

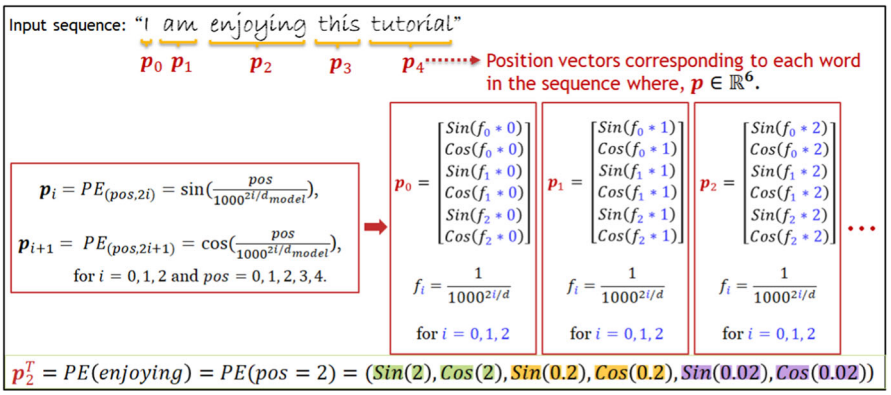

Fig. 8 A set of positional encoding vectors are presented for an example input sequence. In this example, the formulation of PE vectors uses a value of 1000 instead of 10, 000 (as in the original Transformer implementation)

2.5.1 Sinusoidal PE

In the original work, the authors proposed sinusoidal functions for pre-calculating PE vectors for the input dataset [88]. 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,

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

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

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 differ depending on the input sequence's length.

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 Fig. 9 [83, 88]. 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.

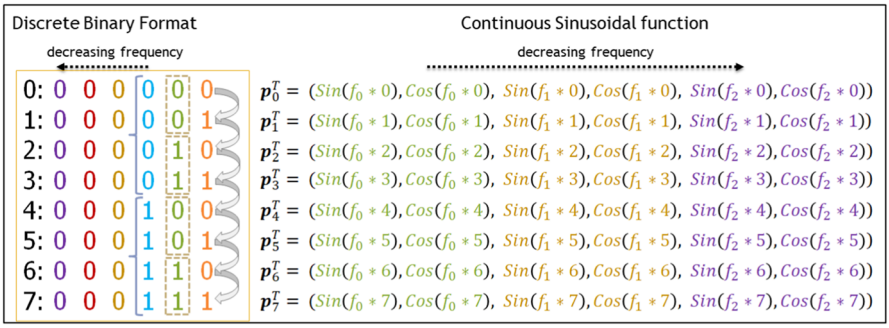

Fig. 9 An analogy between binary format and sinusoidal function for positional encoding is presented. In the binary format, the frequency of alternating bits decreases as we move from orange to purple bits. Sinusoidal PE shows a similar behavior of changing frequency as we move along the positional encoding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

2.5.3 Positional Encoding and Rotation Matrix

The 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, the PE vector $\mathbf{p}_{i+k}$ 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

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

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

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

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.

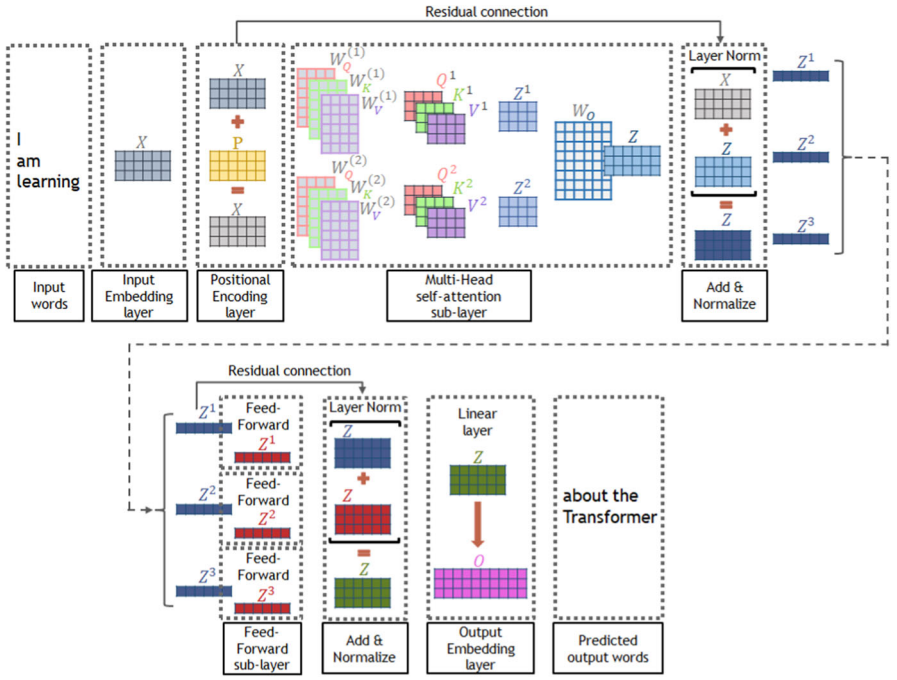

Fig. 10 An example of a Transformer model with a single encoder is presented. The end-to-end sequence of operations is shown for an input sequence of three words and an output prediction of the expected three following words

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 types of information. The PE vectors contain information about the location of a word in the input sequence, while the word embeddings 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 the two interfering with each other.

Now that we have discussed each operation individually as implemented in the Transformer architecture, Fig. 10 depicts the end-to-end flow of the internal operations in the Transformer architecture with a single encoder and three input words to predict the next three words.

3 Road Map of Transformers for Time-Series Analysis

Since the inception of Transformers in 2017 [88], there have been many advances in these networks for time-series analysis. The original model mainly focuses on NLP tasks, but now the architecture has been expanded for classification [105], time-series analysis [80], semantic segmentation [109], and more. Time-series data were not part of the initial conception of Transformers. Many researchers have customized and improved the architecture's performance for time-series analysis. 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 accomplishes two main tasks. Within each task, 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 citations) 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 Fig. 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 allowing them to learn, categorize, and forecast data that has not been labeled. For example, there is a considerable amount of unlabeled satellite imaging data. In [106], a proposed model, SITS-BERT, learns from unlabeled data to apply region classification from satellite imagery. Other self-supervised or unsupervised learning models include anomaly Transformer [100] and self-supervised Transformer for Time-Series (STraTS) [86].

3.1.2 Data Preprocessing

Preprocessing is routinely performed to prepare the data to be fed into a machine learning model. Preprocessing operations are known to impact the performance of machine learning models. Some researchers either add noise to input data features [106], perform masking [75], or variable selection [49]. The operation of masking removes features in the input thereby improving performance by making the model better at predicting missing features. A similar concept applies when adding noise to the input for training, except that the features become relatively noisier but are not

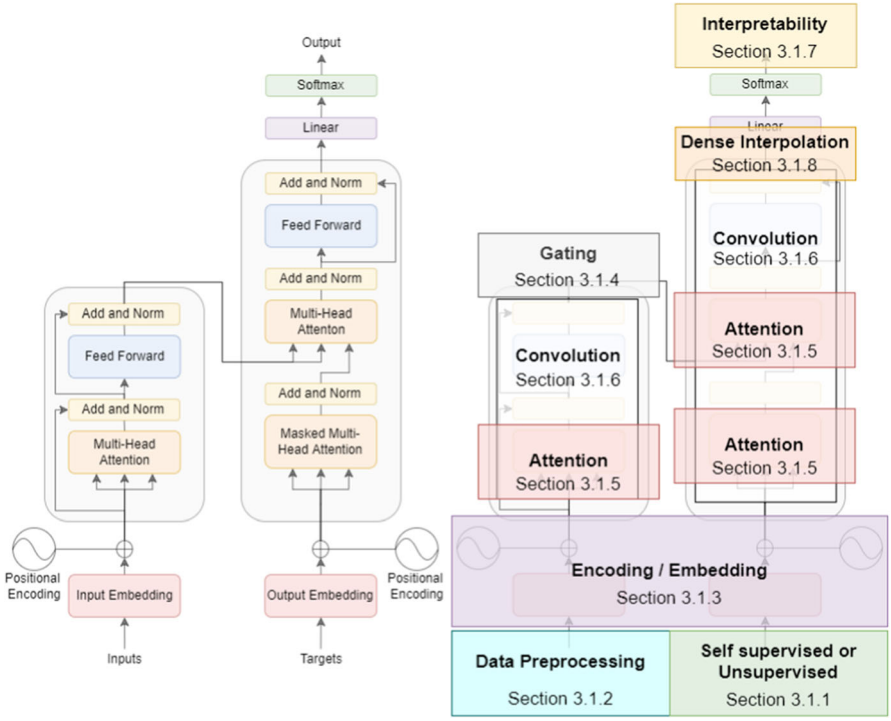

Fig. 11 Overview of avenues of improvement for time-series Transformers. On the left is a recreation of the original Transformer architecture [88]. The right side shows locations in the original architecture for each of the avenues of improvement in Sect. 3.1 with respect to the original Transformer model

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 in [88]. 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 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 [75], temporal encoding [12], and other methods have been proposed for creating Transformers which can be trained more efficiently [49].

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 [54]. This operation combines and selects

multiple outputs from an encoder or decoder block. Gating also benefits from applying non-linear data processing when 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) [54] and Temporal Fusion Transformers (TFT) [49]. 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. The TFT proposes Gated Linear Units (GLUs) [20] 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.

The Tightly-Coupled Convolutional Transformer (TCCT) [76] proposes three architectures that improve upon attention. The first is called Cross Stage Partial Attention (CSPAttention). This approach combines Cross Stage Partial Network (CSPNet) [90] 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 pass-through 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) [15] and Informer [110]. 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 the 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 [47], TFT [49], and YFormer [57]. 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 down-sampling 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 that improve upon Transformers via convolution include TCCT [76], LogSparse Transformers [47], TabAConvBERT [75], and Traffic Transformers [12]. The TCCT uses a mechanism called dilated causal convolution from the Informer [110] 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, called convolutional self-attention. TabAConvBERT employs one-dimensional convolutions considering that it is naturally effective for time-series data. Traffic Transformers [12] 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. The black-box phenomenon is a prevalent problem in artificial intelligence. This refers to the situation where only the inputs and outputs of a learning model can be observed. It is not precisely known how the parameters of the model interact to arrive at the final output.

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 [60]. One possible approach is 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 [49, 54, 86]. These models can produce explanations that 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 [80]. This approach does not utilize output embedding, thereby 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, the work in [80] 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 Transformer paper is Bidirectional Encoder Representations from Transformers (BERT) [22]. 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.

The BERT architecture has inspired many new Transformer models for time-series data [67, 75, 100, 106]. 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 [26], ambient temperature [110], traffic volume [26], satellite imagery [106], various forms of healthcare data [40], 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 networks learn 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 [98]. 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 that 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 [12, 15, 47, 49, 57, 67, 76, 86, 97, 98, 110]. 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 [54, 75, 80, 100, 106]. 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 [110]. 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 $\mathcal{O}(J L \log L)$. Furthermore, the self-attention distilling operation removes redundant combinations of value vectors. This operation is inspired by dilated convolution as proposed in [104] and [32]. The output of the multi-head self-attention is fed into 1-D convolution filters with kernel size equal to 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 Fig. 12. Effectively, the total space complexity reduces considerably. Stacked replicas are also built, with an input length of 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 Fig. 12) thereby considerably reducing inference time.

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

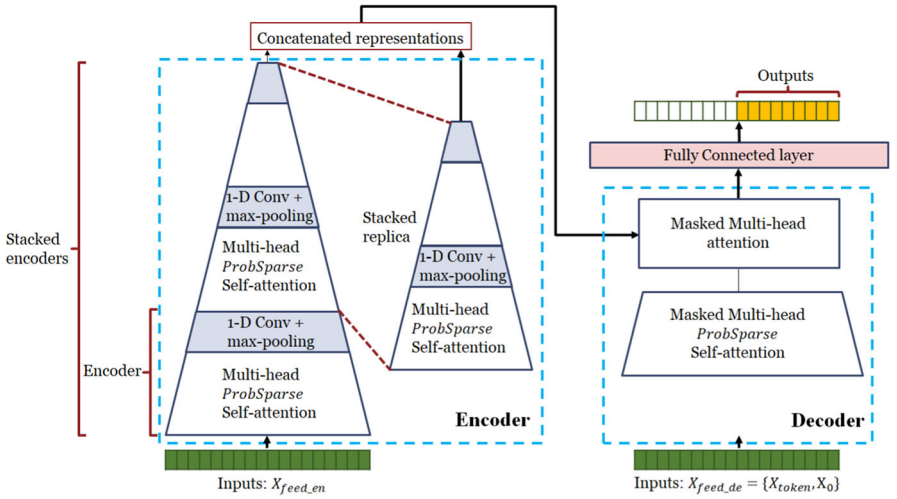

Fig. 12 Informer architecture as proposed in [110] is presented. Informer consists of stacked encoders and decoders. The encoder comprises multi-head ProbSparse self-attention and self-attention distilling operations. Stacked replicas are built with an input length of half of the previous layer. The 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

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) [4], Prophet [85], LSTMa [8], LSTnet [45], and DeepAR [72].

4.2 LogSparse Transformer Architecture

Li et al. proposed LogSparse Transformers to overcome memory challenges, thus making Transformers more feasible for time-series data with long-term dependencies [47]. LogSparse Transformers allow each time step to attend to previous time steps that are selected using an exponential step size. This reduces the memory utilization from $\mathcal{O}(L^2)$ to $\mathcal{O}(L \log_2 L)$ in each self-attention layer. Figure 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 the LogSparse step size after a particular range of time steps.

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 Fig. 14.

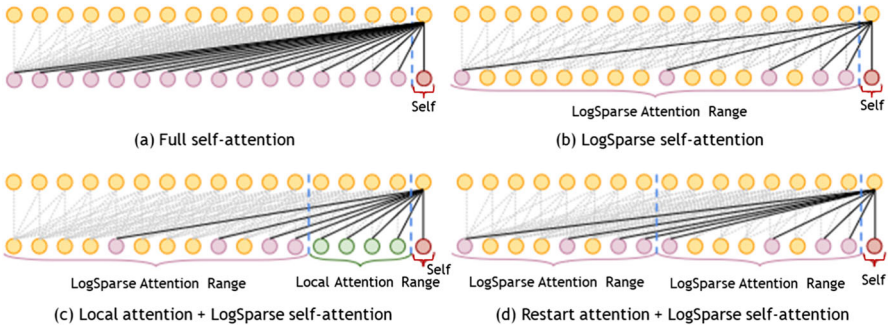

Fig. 13 Various ways to apply LogSparse self-attention as proposed by Li et al. [47]

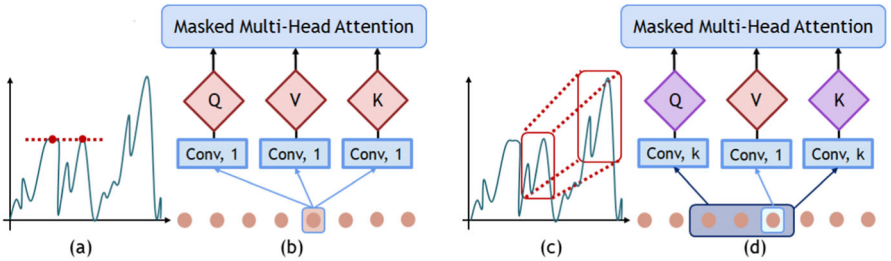

Fig. 14 The canonical self-attention mechanism is similar to using convolution with kernel size equal to one, as shown in (**b**) [47]. It is only able to capture point-wise similarity, as shown in (**a**). Local context is captured using convolution with kernel size greater than one proposed by Li et 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 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 multivariate, 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 [40], named Simply Attend and Diagnose (SAnD) [80]. 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

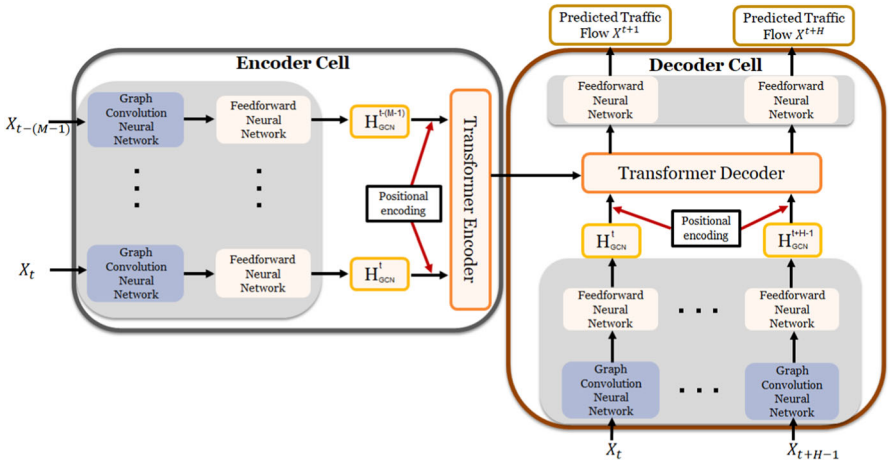

Fig. 15 The Traffic Transformer introduced in [12] is presented. The encoder comprises of Graph Neural Network for capturing the spatial dependencies in the traffic data, followed by a feed-forward layer. The output is fed into the Transformer encoder to capture the temporal dependencies. The output 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

layer'. This layer, together with positional encoding, is used to capture the temporal structure of clinical data. A linear layer and softmax/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 spatiotemporal dependencies in traffic forecasting. The Traffic Transformer [12] has been built based on two existing networks, the first being a Graph Neural Network [74, 102] followed by the Transformer as shown in Fig. 15. The Transformer models the temporal dependencies, and the Graph Neural Network is used to model spatial dependencies.

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

(a) Continuity of time-series data:

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

(b) Periodicity of time-series data:

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 $\left(\frac{24\times 60}{5}\right)$ 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 [39] and the California Transportation Agencies Performance Measurement System https://pems.dot.ca.gov/([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 [71] from raw optical satellite images is done using Transformers. The performance of the Transformer architecture for vegetation classification, compared to LSTM-RNN [35], MS-ResNet [93], DuPLO [38], TempCNN [63], 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 [97] using the data of weekly count of flu cases in a particular area. A single week's prediction is made using ten previous weeks' 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, the 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 data 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.

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

Time-delay embedding of different dimensions d, from 2 till 32, are formed with $\tau = 1$. Out of all the experiments, the RMSE value reached a minimum with the dimension of 8, which conforms to similar results of an independent study on clinical data [81].

5 Best Practices for Training Time-Series Transformers

The Transformer architecture is becoming increasingly popular and leading many researchers to seek ways to optimize these networks for various applications. The approaches include adapting the architecture for use in a specific problem domain, adapting 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 [88] utilized many different strategies to stabilize the gradients during training for deeper networks. The use of residual connections allows 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 a 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 Implementing Transformers in Popular Frameworks

Here, we provide an overview of popular frameworks for implementing and training Transformer models. These frameworks offer a user-friendly interface and support for custom model architectures, enabling researchers and developers to experiment with and adapt Transformers for time-series analysis and other tasks.

Hugging Face Transformers is an open-source library that provides pre-trained Transformer models and user-friendly APIs for a wide range of natural language processing tasks. Compatible with both PyTorch and TensorFlow, the library offers an extensive collection of pre-trained models and community-contributed models,

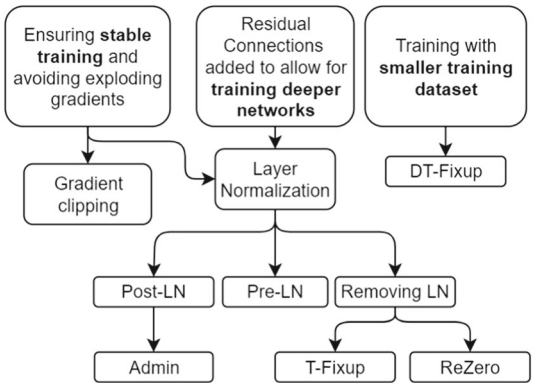

Fig. 16 An overview of best practices for training Transformers

making it an excellent starting point for researchers and developers. More details about the library can be found at the Hugging Face Transformers GitHub repository.

PyTorch Lightning is a lightweight wrapper for PyTorch designed to simplify deep learning research and development. It offers a structured approach to training deep learning models, including Transformer models, with minimal boilerplate code. PyTorch Lightning is compatible with the Hugging Face Transformers library, enabling users to leverage pre-trained models and fine-tune them for specific tasks. More information is available at the PyTorch Lightning GitHub repository. Other libraries in PyTorch for efficient large Transformer model training include Microsoft DeepSpeed and MosaicML Composer.

TensorFlow is an open-source machine learning library developed by Google that supports building and training deep learning models, including Transformers. The library provides implementations of the original Transformer architecture and various pre-trained models that can be fine-tuned for specific tasks. TensorFlow also offers extensive documentation and community support for implementing custom Transformer models. The TensorFlow website offers more information.

5.3 Improvements in the Transformer Architecture for Better Training

Many advancements to the Transformer architecture have been proposed with the aim of resolving 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 (Fig. 17).

The original Transformer architecture can be referred to as post-layer normalization (post-LN), where the layer normalization is located outside the residual block [99]. Post-LN converges much slower and requires a learning rate warm-up strategy [99]. 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.

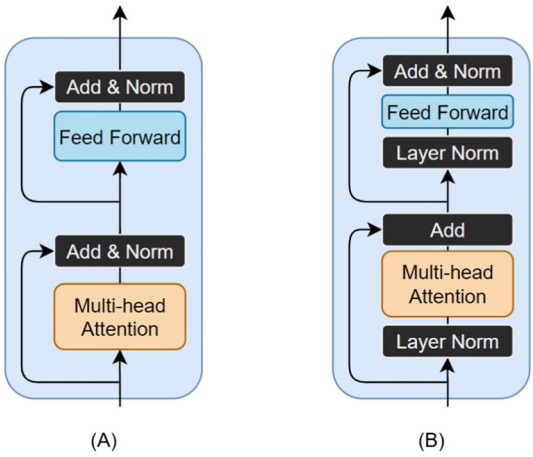

Fig. 17 post-LN transformer (**a**) vs. pre-LN transformer (**b**)

The Pre-LN Transformer can achieve this by controlling the gradient magnitudes and balancing the residual dependencies [99]. Although the Pre-LN Transformer architecture does not require learning rate warm-up, it has inferior empirical performance as compared to the 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 [53]. *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 [53]. 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. The *ReZero* approach replaces the layer normalization operation with a trainable parameter, α, which is initialized to 0 in each residual layer [7]. Consequently, the entire network is initialized to compute the identity function, with a gradual and adaptive introduction of the 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.

5.4 Practical Issues for Training Transformers

5.4.1 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. [48] demonstrated that in some instances, training large models and then compressing the results leads to better performance. The *lottery ticket hypothesis* [29] reveals that *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.

5.4.2 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 requires large datasets, making training with a small dataset challenging. Small datasets require pre-trained models and small batch sizes to perform well. However, these two requirements make training additional Transformer layers more difficult. Using a small batch size causes the variance of the updates to be larger. Even with large batch size, the model may typically generalize poorly. In some cases, better initialization techniques can optimize the model making it perform well on the smaller datasets. The use of Xavier initialization is the most common scheme for Transformers [31]. Recently, T-Fixup has been shown to outperform Xavier [36]. The motivation for the T-Fixup scheme, introduced by Huang et al., is as follows. It was found that when training a Transformer without learning rate warm-up, the variance in the Adam optimizer was amplified by a large initial learning rate thereby leading to large updates at the beginning of training [36]. 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 T-Fixup method was proposed in which a new weight initialization scheme provides theoretical guarantees that keep model updates bounded and removes both warm-up and layer normalization [36]. Following the work of T-Fixup, a data-dependent initialization technique (known as DT-Fixup) was developed [101]. DT-Fixup allows for training deeper Transformer models with small datasets, given that the correct optimization procedures are followed.

5.4.3 Other Strategies to Consider

Batch Size. Popel et al. found that the optimal batch size depends on the complexity of the model [66]. They considered two classes of models, one a "base" and one a "big." For the base model, a larger batch size (up to 4500) performed well, whereas a different set of parameters yielded superior results for the big model. The big model needed a minimum batch size (1450 in their experiments) before it 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 [10, 79, 107] 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 [66].

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 [66].

6 Conclusion and Future Trends

In conclusion, the Transformer architecture has proven to be a powerful tool for tackling time-series tasks, offering an efficient alternative to RNNs, LSTMs, and GRUs, while also overcoming their limitations. To effectively handle time-series data, modifications to the original Transformer architecture have been proposed. Various best practices for training Transformers have been developed and many open-source frameworks are available for efficiently training large Transformer models. Robustness [95], failure detection [2], and multimodal learning [96] are some of the future trends in deep learning. Moving forward, the development of robust, self-aware Transformer architectures using uncertainty estimation in time-series prediction is an open challenge currently being pursued by the research community. The availability of large datasets having multiple modalities such as images, videos, and text, combined with time-series data can lead to the development of Transformer-based foundation models capable of learning indiscernible and subtle features that may lead to unprecedented discoveries for the time-series tasks.

Acknowledgements This work was partly supported by the National Science Foundation Awards ECCS-1903466, OAC-2008690, and OAC-2234836.

Data Availability The manuscript has no associated data.

Declarations

Conflict of interest There are no conflicts of interest or competing interests.

References

1. A.F. Agarap, Deep learning using rectified linear units (relu) (2018). arXiv:1803.08375
2. S. Ahmed, D. Dera, S.U. Hassan, N. Bouaynaya, G. Rasool, Failure detection in deep neural networks for medical imaging. Front. Med. Technol. (2022). https://doi.org/10.3389/fmedt.2022.919046
3. S. Albawi, T.A. Mohammed, S. Al-Zawi. Understanding of a convolutional neural network. in *2017 International Conference on Engineering and Technology (ICET)* (IEEE, 2017), pp. 1–6. https://doi.org/10.1109/ICEngTechnol.2017.8308186
4. A.A. Ariyo, A.O. Adewumi, C.K. Ayo, Stock price prediction using the arima model. in *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation* (IEEE, 2014), pp. 106–112. https://doi.org/10.1109/UKSim.2014.67
5. K. ArunKumar, D.V. Kalaga, C.M.S. Kumar, M. Kawaji, T.M. Brenza, Forecasting of COVID-19 using deep layer recurrent neural networks (RNNs) with gated recurrent units (GRUs) and long short-term memory (LSTM) cells. Chaos Solitons Fractals **146**, 110861 (2021). https://doi.org/10.1016/j.chaos.2021.110861

6. J.L. Ba, J.R. Kiros, G.E. Hinton, Layer normalization (2016). arXiv:1607.06450
7. T. Bachlechner, B.P. Majumder, H. Mao, G. Cottrell, J. McAuley, C. de Campos, M.H. Maathuis, (eds) ReZero is all you need: fast convergence at large depth. in *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence Vol. 161 of Proceedings of Machine Learning Research*, ed by C. de Campos, M. H. Maathuis (PMLR, 2021), pp. 1352–1361
8. D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate (2014). arXiv:1409.0473
9. A. Bapna, M. Chen, O. Firat, Y. Cao, Y. Wu, Training deeper neural machine translation models with transparent attention. in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (Association for Computational Linguistics, Brussels, Belgium, 2018), pp. 3028–3033. https://doi.org/10.18653/v1/D18-1338
10. L. Behera, S. Kumar, A. Patnaik, On adaptive learning rate that guarantees convergence in feedforward networks. IEEE Trans. Neural Netw. **17**(5), 1116–1125 (2006). https://doi.org/10.1109/TNN.2006.878121
11. C. Bergmeir, R.J. Hyndman, B. Koo, A note on the validity of cross-validation for evaluating autoregressive time series prediction. Comput. Stat. Data Anal. **120**, 70–83 (2018)
12. L. Cai, K. Janowicz, G. Mai, B. Yan, R. Zhu, Traffic transformer: capturing the continuity and periodicity of time series for traffic forecasting. Trans. GIS **24**(3), 736–755 (2020). https://doi.org/10.1111/tgis.12644
13. Z. Che, S. Purushotham, K. Cho, D. Sontag, Y. Liu, Recurrent neural networks for multivariate time series with missing values. Sci. Rep. **8**(1), 6085 (2018). https://doi.org/10.1038/s41598-018-24271-9
14. G. Chen, A gentle tutorial of recurrent neural network with error backpropagation (2016). arXiv:1610.02583
15. K. Chen, et al. NAST: non-autoregressive spatial-temporal transformer for time series forecasting (2021). arXiv:2102.05624
16. L. Chen et al., Decision transformer: reinforcement learning via sequence modeling. Adv. Neural. Inf. Process. Syst. **34**, 15084–15097 (2021)
17. W. Chen, et al. Learning to rotate: quaternion transformer for complicated periodical time series forecasting. in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD '22* (Association for Computing Machinery, New York, NY, USA, 2022), pp. 146–156. https://doi.org/10.1145/3534678.3539234
18. K. Choromanski, et al. Rethinking attention with performers (2020). arXiv:2009.14794
19. J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling (2014). arXiv:1412.3555
20. Y.N. Dauphin, A. Fan, M. Auli, D. Grangier, D. Precup, Y.W. Teh, Y.W. (eds) Language modeling with gated convolutional networks. in *Proceedings of the 34th International Conference on Machine Learning Vol. 70 of Proceedings of Machine Learning Research*, ed. by D. Precup, Y.W. Teh (PMLR, 2017), pp. 933–941
21. D. Dera, S. Ahmed, N.C. Bouaynaya, G. Rasool, Trustworthy uncertainty propagation for sequential time-series analysis in rnns. IEEE Trans. Knowl. Data Eng. (2023). https://doi.org/10.1109/TKDE.2023.3288628
22. J. Devlin, M.W. Chang, K. Lee, K. 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), pp. 4171–4186 (2019)
23. L. Di Persio, O. Honchar, Recurrent neural networks approach to the financial forecast of google assets. Int. J. Math. Comput. Simul. **11**, 7–13 (2017)
24. M. Dixon, J. London, Financial forecasting with α-rnns: a time series modeling approach. Front. Appl. Math. Stat. **6**, 551138 (2021). https://doi.org/10.3389/fams.2020.551138
25. A. Dosovitskiy, et al. An image is worth 16x16 words: transformers for image recognition at scale (2020). arXiv:2010.11929
26. D. Dua, C. Graff, UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences (2017). http://archive.ics.uci.edu/ml
27. J. El Zini, Y. Rizk, M. Awad, An optimized parallel implementation of non-iteratively trained recurrent neural networks. J. Artif. Intell. Soft Comput. Res. **11**(1), 33–50 (2021). https://doi.org/10.2478/jaiscr-2021-0003

28. H. Fei, F. Tan, Bidirectional grid long short-term memory (bigridlstm): a method to address context-sensitivity and vanishing gradient. Algorithms **11**(11), 172 (2018). https://doi.org/10.3390/a11110172
29. J. Frankle, M. Carbin, The lottery ticket hypothesis: finding sparse, trainable neural networks (2018). arXiv:1803.03635
30. J. Gehring, M. Auli, D. Grangier, D. Yarats, Y.N. Dauphin, Convolutional sequence to sequence learning. in *34th International Conference on Machine Learning* ICML **2017**(3), pp. 2029–2042 (2017)
31. X. Glorot, Y. Bengio, Y.W. Teh, M. Titterington, (eds) Understanding the difficulty of training deep feedforward neural networks. in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics Vol. 9 of Proceedings of Machine Learning Research*, ed. by Y.W. Teh, M. Titterington. (PMLR, Chia Laguna Resort, Sardinia, Italy, 2010), pp. 249–256
32. A. Gupta, A.M. Rush, Dilated convolutions for modeling long-distance genomic dependencies (2017). arXiv:1710.01278
33. J. Hao, et al. Modeling recurrence for transformer (2019). arXiv:1904.03092
34. J. Ho, N. Kalchbrenner, D. Weissenborn, T. Salimans, Axial attention in multidimensional transformers (2019). arXiv:1912.12180
35. S. Hochreiter, J. Schmidhuber, Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997). https://doi.org/10.1162/neco.1997.9.8.1735
36. X.S. Huang, F. Perez, J. Ba, M. Volkovs, H.D. III, A. Singh, (eds) Improving transformer optimization through better initialization. in *Proceedings of the 37th International Conference on Machine Learning Vol. 119 of Proceedings of Machine Learning Research*, ed. by H.D. III, A. Singh (PMLR, 2020), pp. 4475–4483
37. Y. Huang, H. Wallach, et al. (eds) GPipe: efficient training of giant neural networks using pipeline parallelism. in *Advances in Neural Information Processing Systems*, ed. by H. Wallach, et al. Vol. 32. (Curran Associates, Inc., 019)
38. R. Interdonato, D. Ienco, R. Gaetano, K. Ose, DuPLO: a DUal view Point deep Learning architecture for time series classificatiOn. ISPRS J. Photogramm. Remote. Sens. **149**, 91–104 (2019)
39. H.V. Jagadish et al., Big data and its technical challenges. Commun. ACM **57**(7), 86–94 (2014). https://doi.org/10.1145/2611567
40. A.E. Johnson et al., MIMIC-III, a freely accessible critical care database. Sci. data **3**(1), 1–9 (2016)
41. N. Jouppi, C. Young, N. Patil, D. Patterson, Motivation for and evaluation of the first tensor processing unit. IEEE Micro **38**(3), 10–19 (2018). https://doi.org/10.1109/MM.2018.032271057
42. A. Katharopoulos, A. Vyas, N. Pappas, F. Fleuret, III, H.D., A. Singh, (eds) Transformers are RNNs: fast autoregressive transformers with linear attention. in *Proceedings of the 37th International Conference on Machine Learning Vol. 119 of Proceedings of Machine Learning Research*, ed. by III, H. D., A. Singh (PMLR, 2020), pp. 5156–5165
43. A. Kirillov, et al. Segment anything (2023). arXiv:2304.02643
44. N. Kitaev, Ł. Kaiser, A. Levskaya, Reformer: the efficient transformer (2020). arXiv:2001.04451
45. G. Lai, W.-C. Chang, Y. Yang, H. 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, SIGIR '18* (Association for Computing Machinery, New York, NY, USA, 2018), pp. 95–104. https://doi.org/10.1145/3209978.3210006
46. C. Li, et al. Automated progressive learning for efficient training of vision transformers. in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE Computer Society, Los Alamitos, CA, USA, 2022), pp. 12476–12486. https://doi.org/10.1109/CVPR52688.2022.01216
47. S. Li, et al. H. Wallach, et al. (eds) Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. in *Advances in Neural Information Processing Systems*, ed by. H. Wallach, et al. Vol. 32 (Curran Associates, Inc., 2019)
48. Z. Li, et al. III, H. Daumé, S. Aarti (ed.) Train big, then compress: rethinking model size for efficient training and inference of transformers. in *Proceedings of the 37th International Conference on Machine Learning Vol. 119 of Proceedings of Machine Learning Research*, ed by. III, H. Daumé, S. Aarti (PMLR, 2020), pp. 5958–5968
49. B. Lim, S. Arık, N. Loeff, T. Pfister, Temporal fusion transformers for interpretable multi-horizon time series forecasting. Int. J. Forecast. **37**(4), 1748–1764 (2021). https://doi.org/10.1016/j.ijforecast.2021.03.012

50. T. Lin, Y. Wang, X. Liu, X. Qiu, A survey of transformers. AI Open **3**, 111–132 (2022). https://doi.org/10.1016/j.aiopen.2022.10.001
51. Z.C. Lipton, J. Berkowitz, C. Elkan, A critical review of recurrent neural networks for sequence learning. (2015). arXiv:1506.00019
52. A. Liška, G. Kruszewski, M. Baroni, Memorize or generalize? searching for a compositional rnn in a haystack (2018). arXiv:1802.06467
53. L. Liu, X. Liu, J. Gao, W. Chen, J. Han, Understanding the difficulty of training transformers. in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (Association for Computational Linguistics, Online, 2020), pp. 5747–5763. https://doi.org/10.18653/v1/2020.emnlp-main.463
54. M. Liu, et al. Gated transformer networks for multivariate time series classification. (2021). arXiv:2103.14438
55. S. Liu, et al. Pyraformer: low-complexity pyramidal attention for long-range time series modeling and forecasting. In: International conference on learning representations (2021)
56. J. Lu, C. Clark, R. Zellers, R. Mottaghi, A. Kembhavi, Unified-IO: a unified model for vision, language, and multi-modal tasks (2022). arXiv:2206.08916
57. K. Madhusudhanan, J. Burchert, N. Duong-Trung, S. Born, L. Schmidt-Thieme, Yformer: U-net inspired transformer architecture for far horizon time series forecasting (2021). arXiv:2110.08255
58. T. Mikolov, K. Chen, G. Corrado, J. Dean, efficient estimation of word representations in vector space (2013). arXiv:1301.3781
59. Y. Nie, N.H. Nguyen, P. Sinthong, J. Kalagnanam, A time series is worth 64 words: long-term forecasting with transformers (2022). arXiv:2211.14730
60. I.E. Nielsen, D. Dera, G. Rasool, R.P. Ramachandran, N.C. Bouaynaya, Robust explainability: a tutorial on gradient-based attribution methods for deep neural networks. IEEE Signal Process. Mag. **39**(4), 73–84 (2022). https://doi.org/10.1109/MSP.2022.3142719
61. I. Padhi, et al. Tabular transformers for modeling multivariate time series. in *2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (IEEE, Toronto, 2021), pp. 3565–3569. https://doi.org/10.1109/ICASSP39728.2021.9414142
62. R. Pascanu, T. Mikolov, Y. Bengio, S. Dasgupta, D. McAllester, (eds) On the difficulty of training recurrent neural networks. in *Proceedings of the 30th International Conference on Machine Learning Vol. 28 of Proceedings of Machine Learning Research* ed. by S. Dasgupta, D. McAllester (PMLR, Atlanta, Georgia, USA, 2013), pp. 1310–1318
63. C. Pelletier, G.I. Webb, F. Petitjean, Temporal convolutional neural network for the classification of satellite image time series. Remote Sens (2019). https://doi.org/10.3390/rs11050523
64. J. Pennington, R. Socher, C. Manning, GloVe: global vectors for word representation. in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* Vol. 31 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2014), pp. 1532–1543. https://doi.org/10.3115/v1/D14-1162
65. M.E. Peters, et al. 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). https://doi.org/10.18653/v1/n18-1202
66. M. Popel, O. Bojar, Training tips for the transformer model. Prague Bull. Math. Linguist. **110**(1), 43–70 (2018). https://doi.org/10.2478/pralin-2018-0002
67. X. Qi, et al. From known to unknown: knowledge-guided transformer for time-series sales forecasting in Alibaba (2021). arXiv:2109.08381
68. Y. Qin, et al. Knowledge inheritance for pre-trained language models (2021). arXiv:2105.13880
69. A.H. Ribeiro, K. Tiels, L.A. Aguirre, T. & Schön, S. Chiappa, R. Calandra, (eds) Beyond exploding and vanishing gradients: analysing rnn training using attractors and smoothness. in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics Vol. 108 of Proceedings of Machine Learning Research* ed. by S. Chiappa, R. Calandra (PMLR, 2020), pp. 2370–2380
70. A. Roy, M. Saffar, A. Vaswani, D. Grangier, Efficient content-based sparse attention with routing transformers. Trans. Assoc. Comput. Linguist. **9**, 53–68 (2021). https://doi.org/10.1162/tacl_a_00353
71. M. Rußwurm, M. Körner, Self-attention for raw optical satellite time series classification. ISPRS J. Photogramm. Remote. Sens. **169**, 421–435 (2020). https://doi.org/10.1016/j.isprsjprs.2020.06.006
72. D. Salinas, V. Flunkert, J. Gasthaus, T. Januschowski, DeepAR: probabilistic forecasting with autoregressive recurrent networks. Int. J. Forecast. **36**(3), 1181–1191 (2020)

73. G. 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)* (Association for Computing Machinery, New York, NY, USA, 1962), pp. 234–250. https://doi.org/10.1145/1461518.1461544
74. F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model. IEEE Trans. Neural Netw. **20**(1), 61–80 (2008)
75. S.M. Shankaranarayana, D. Runje, Attention augmented convolutional transformer for tabular time-series (IEEE Computer Society, 2021), pp. 537–541. https://doi.org/10.1109/ICDMW53433.2021.00071
76. L. Shen, Y. Wang, TCCT: tightly-coupled convolutional transformer on time series forecasting. Neurocomputing **480**, 131–145 (2022). https://doi.org/10.1016/j.neucom.2022.01.039
77. A. Sherstinsky, Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. Physica D **404**, 132306 (2020). https://doi.org/10.1016/j.physd.2019.132306
78. A. Shewalkar, D. Nyavanandi, S.A. Ludwig, Performance evaluation of deep neural networks applied to speech recognition: RNN, LSTM and GRU. J. Artif. Intell. Soft Comput. Res. **9**(4), 235–245 (2019). https://doi.org/10.2478/jaiscr-2019-0006
79. L.N. Smith, A disciplined approach to neural network hyper-parameters: part 1–learning rate, batch size, momentum, and weight decay (2018). arXiv:1212.5701
80. H. Song, D. Rajan, J. Thiagarajan, A. Spanias, Attend and diagnose: clinical time series analysis using attention models. Proc. AAAI Conf. Artif. Intell. (2018). https://doi.org/10.1609/aaai.v32i1.11635
81. G. Sugihara, R.M. May, Nonlinear forecasting as a way of distinguishing chaos from measurement error in time series. Nature **344**(6268), 734–741 (1990)
82. C. Szegedy, et al. Going deeper with convolutions. in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2015)
83. S. Takase, N. Okazaki, Positional encoding to control output sequence length (2019). arXiv:1904.07418
84. Y. Tay, M. Dehghani, D. Bahri, D. Metzler, Efficient transformers: a survey. ACM Comput. Surv. **55**(6), 1–28 (2022). https://doi.org/10.1145/3530811
85. S.J. Taylor, B. Letham, Forecasting at scale. Am. Stat. **72**(1), 37–45 (2018). https://doi.org/10.1080/00031305.2017.1380080
86. S. Tipirneni, C.K. Reddy, Self-supervised transformer for multivariate clinical time-series with missing values. arXiv:2107.14293 (2021)
87. M. Tschannen, O. Bachem, M. Lucic, Recent advances in autoencoder-based representation learning (2018). arXiv:1812.05069
88. A. Vaswani et al., Attention is all you need, in *Advances in Neural Information Processing Systems 30* (2017)
89. J. Vig, (2022). BertViz. https://github.com/jessevig/bertviz Accessed 5 May 2022
90. C.-Y. Wang, et al. CSPNet: a new backbone that can enhance learning capability of CNN. in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (IEEE, 2020), pp. 1571–1580
91. P. Wang, et al. Learning to grow pretrained models for efficient transformer training (2023). arXiv:2303.00980
92. Q. Wang, et al. Learning deep transformer models for machine translation (2019). arXiv:1906.01787
93. Z. Wang, W. Yan, T. Oates, Time series classification from scratch with deep neural networks: a strong baseline. in *2017 International joint conference on neural networks (IJCNN)* (IEEE, 2017), pp. 1578–1585. https://doi.org/10.1109/IJCNN.2017.7966039
94. Z. Wang, Y. Ma, Z. Liu, J. Tang, R-Transformer: recurrent neural network enhanced transformer (2019). arXiv:1907.05572
95. A. Waqas, H. Farooq, N.C. Bouaynaya, G. Rasool, Exploring robust architectures for deep artificial neural networks. Commun. Eng. **1**(1), 46 (2022). https://doi.org/10.1038/s44172-022-00043-2
96. A. Waqas, A. Tripathi, R.P. Ramachandran, P. Stewart, G. Rasool, multimodal data integration for oncology in the era of deep neural networks: a review (2023). arXiv:2303.06471
97. N. Wu, B. Green, X. Ben, S. O'Banion, Deep transformer models for time series forecasting: the influenza prevalence case (2020). arXiv:2001.08317
98. S. Wu, et al. H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin, (eds) Adversarial sparse transformer for time series forecasting. in *Advances in Neural Information Processing Systems*, ed.

by Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. & Lin, H., Vol. 33 (Curran Associates, Inc., 2020), pp. 17105–17115
99. R. Xiong, et al. III, H. D. & Singh, A. (eds) On layer normalization in the transformer architecture. in *Proceedings of the 37th International Conference on Machine Learning Vol. 119 of Proceedings of Machine Learning Research*, ed. by III, H. D. & Singh, A. (PMLR, 2020), pp. 10524–10533
100. J. Xu, H. Wu, J. Wang, M. Long, anomaly transformer: time series anomaly detection with association discrepancy (2021). arXiv:2110.02642
101. P. Xu, et al. 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)* (Association for Computational Linguistics, Online, 2021), pp. 2089–2102. https://doi.org/10.18653/v1/2021.acl-long.163
102. H. Yang, AliGraph: a comprehensive graph neural network platform. in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '19* (Association for Computing Machinery, New York, NY, USA, 2019), pp. 3165–3166. https://doi.org/10.1145/3292500.3340404
103. Y. You, et al. Large batch optimization for deep learning: training bert in 76 minutes. (2019). arXiv:1904.00962
104. F. Yu, V. Koltun, T. Funkhouser, Dilated residual networks. in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE Computer Society, Los Alamitos, CA, USA, 2017), pp. 636–644. https://doi.org/10.1109/CVPR.2017.75
105. L. Yuan, et al. Tokens-to-Token ViT: Training vision transformers from scratch on ImageNet. in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 538–547 (2021)
106. Y. Yuan, L. Lin, Self-supervised pretraining of transformers for satellite image time series classification. IEEE J. Sel. Topics Appl. Earth Obs. Remote Sens. **14**, 474–487 (2020)
107. M.D. Zeiler, Adadelta: an adaptive learning rate method arXiv:1212.5701 (2012)
108. Y. Zhang, J. Yan, Crossformer: transformer utilizing cross-dimension dependency for multivariate time series forecasting. in *The Eleventh International Conference on Learning Representations* (2023)
109. J. Zheng, S. Ramasinghe, S. Lucey, Rethinking positional encoding (2021). arXiv:2107.02561
110. H. Zhou, et al. Informer: beyond efficient transformer for long sequence time-series forecasting Vol. 35, 11106–11115 (2021). https://doi.org/10.1609/aaai.v35i12.17325
111. T. Zhou, et al. K. Chaudhuri, et al. (eds) FEDformer: fequency enhanced decomposed transformer for long-term series forecasting. in *Proceedings of the 39th International Conference on Machine Learning Vol. 162 of Proceedings of Machine Learning Research*, ed. by K. Chaudhuri, et al. (PMLR, 2022), pp. 27268–27286