Smartformer: An Intelligent Model Compression Framework for Transformer

Xiaojian Wang
Harbin Institute of Technology
Yinan Wang
Rensselaer Polytechnic Institute
Jin Yang
Harbin Institute of Technology
Ying Chen (yingchen@hit.edu.cn)
Harbin Institute of Technology https://orcid.org/0000-0002-0366-131X

Article

Keywords: intelligent model compression, Transformer, CP-decomposition, reinforcement learning, deep learning

Posted Date: June 23rd, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1780688/v1

Smartformer: An Intelligent Model Compression Framework for Transformer

Xiaojian Wang[a], Yinan Wang[b], Jin Yang[a], Ying Chen[a]

[a] School of Management, Harbin Institute of Technology, Heilongjiang, 150000, China;

[b] Department of Industrial and Systems Engineering, Rensselaer Polytechnic Institute, NY, 12180, USA.

*Corresponding author: yingchen@hit.edu.cn (Y. Chen)

Abstract

Transformer, as one of the latest popular deep neural networks (DNNs), has achieved outstanding performance in various areas. However, this model usually requires massive amounts of parameters to fit. Over-parameterization not only brings storage challenges in a resource-limited setting but also inevitably results in the model over-fitting. Even though literature works introduced several ways to reduce the parameter size of Transformers, none of them have focused on addressing the over-parameterization issue by concurrently considering the following three objectives: preserving the model architecture, maintaining the model performance, and reducing the model complexity (number of parameters) to improve its convergence property. In this study, we propose an intelligent model compression framework, Smartformer, by incorporating reinforcement learning and CP-decomposition techniques. This framework has two advantages: 1) from the perspective of convenience, researchers can directly exploit this framework to train their designed Transformer models as usual without fine-tuning the original model training hyperparameters; 2) from the perspective of effectiveness, researchers can simultaneously fulfill the aforementioned three objectives by using this framework. We conduct two basic experiments and three practical experiments to demonstrate the proposed framework using various datasets and existing Transformer models. The results demonstrate that the Smartformer is able to improve the existing Transformers regarding the model performance and model complexity by intelligently incorporating the model decomposition and compression in the training process.

Keywords: intelligent model compression, Transformer, CP-decomposition, reinforcement learning, deep learning

Introduction

Recent years have witnessed a rapid development of Transformer, which is a variant of deep neural networks (DNN) (Han et al. 2022, Tay et al. 2022). Different from recurrent neural networks (RNN) and convolutional neural networks (CNN), Transformer has an encoder-decoder architecture that is solely based on the attention mechanism. Such a flexible structure helps Transformer not only achieve success in natural language processing (NLP) (Vaswani et al. 2017, Karita et al. 2019), but also perform well in computer vision (Parmar et al. 2018, Cornia et al. 2020), time-series forecasting (Wu et al. 2020, Li et al. 2019) and other fields. Despite its effective model architecture, similar to other popular variants of DNN, Transformer

usually has massive amounts of parameters (Vaswani et al. 2017, Parmar et al. 2018). Although over-parameterization is one key factor in the success of DNN (Neyshabur et al. 2017), deploying these parameters is a challenge in a resource-limited setting (Ma et al. 2019), and furthermore, over-parameterization inevitably introduces local optima (Ding et al. 2022) that hinders the convergence in training. Therefore, an instant question is how to intelligently select a subset of parameters in Transformer to simultaneously meet three objectives: (1) preserve its architecture, (2) maintain the model capacity (performance on a given task), (3) and reduce the model complexity (number of parameters) to improve its convergence property?

Most of the existing methods focus more on reducing the model complexity over other objectives. Evidence from the latest literature has shown that weight sharing, knowledge distillation, pruning, and quantization are likely to make the Transformer models smaller. As summarized in Xu and McAuley (2022), weight sharing is both a memory-efficient and a storage-efficient way to reduce computational cost and parameters by reusing the same parameters (see Sachan and Neubig 2018, Reid et al. 2021); however, it is not always clear how many weights should be shared before an unacceptable performance degradation of a given network architecture occurs (O'Neill 2020). Knowledge distillation is to transfer knowledge from a large model to a smaller one for easy training (see Liu et al. 2019, Wang et al. 2020), and pruning is to remove redundant parameters from a model while still maintaining its performance (see Michel et al. 2019, Voita et al. 2019). However, both of these two methods entail changing the model architectures, which sacrifices the effectiveness of current architecture and introduces extra efforts in the practical implementation. Quantization is to compress the Transformer with a reduced number of bits (see Prato et al. 2020, Kim et al. 2021). Nonetheless, this method requires an additional block to quantize the parameters, and the quantized parameters lack the interpretability. Moreover, the bit level in quantization is a hyperparameter, and so far, there is no existing algorithm to automatically optimize the bit level as the model training proceeds.

Another line of work tries to both reduce the model complexity and preserve its architecture by tensor decomposition. The idea is to decompose the weight matrices/tensors of each layer and propose low-rank alternatives for existing layers. For example, Ma et al. (2019) proposed a multi-linear attention module with block-term tensor decomposition (BTD), which is a combination of CANDECOMP/PARAFAC (CP)-decomposition (Carroll and Chang 1970, Kiers 2000) and Tucker decomposition (Tucker 1966); Li et al. (2021) incorporated tensor-train decomposition (TTD) with Transformer, and developed the compressed full tensor Transformer (CFTT); Chen et al. (2021) introduced a provably optimal low-rank decomposition method to decompose the fully-connected and self-attention layers efficiently; Panahi et al. (2021) used low-rank decomposed representation of a reshaped and rearranged original matrix to obtain space-efficient and expressive linear layers. The main advantage of these methods is to approximately preserve the original operation in the Transformer and break down the structured weight matrices/tensors for flexible selection. However, they still have two limitations:

(1) All existing works decomposed the model layers in a brute-force way, which failed to efficiently address the trade-off between reducing the parameters versus maintaining the model capacity.
(2) None of these studies completely derived the expressions of forward and backward

propagations for the decomposed layers. Thus, the theoretical foundation for independently tuning each decomposed weight matric/tensor is lacking in the literature.

In this study, we propose an intelligent model compression framework, Smartformer, to simultaneously address the three objectives mentioned above. Smartformer has two main sub-tasks: the first one is to propose a low-rank alternative of the Multi-Head Attention layer to both enable model compression and preserve its architecture, and the second one is to re-design the training process to both fit the model parameters and optimize model decomposition to preserve its capacity.

For the weight matric/tensor decomposition, we use the CP-decomposition method as the core technique for three reasons. First, compared with TTD and Tucker decomposition, the CP-decomposition method decomposes the tensor into multiple groups of vectors, which are unique and can uncover the actual latent components (Wang et al. 2021a). Second, researchers provided the theoretical guarantees in applying the CP-decomposition to DNN regarding compressibility and generalizability (Li et al. 2020). Third, the CP-decomposition method has achieved some success in decomposing various network architectures, namely, CNN (Lebedev et al. 2015 and Wang et al. 2021a), long-short term memory (LSTM) networks (Ma et al. 2021), and RNN (Wang et al. 2021b). For Transformer, to the best of our knowledge, the work that uses the CP-decomposition method to reduce the model parameters has not been explored. Ma et al. (2019) pointed out that the Multi-Head Attention layer, as the key component in Transformer for the contextual learning, required a large number of parameters, and Xu and McAuley (2022) stated that decomposing parameters in the token embedding layers were not efficient. Hence, to bridge the knowledge gap, our first focus is to use CP-decomposition to approximately compress the Multi-Head Attention layers in Transformer. We name this new layer after the CPAC-Attention layer. Different from the literature works (e.g., Ma et al. 2019 and Li et al. 2021), the CPAC-Attention layer can achieve a single-step training, which means it can be obtained directly from the original attention layer without additional steps to process the tensors. With this new layer, the expressions of forward and backward propagations in Transformer can be accordingly derived.

For the newly proposed training process, the main objective is to both fit the model parameters and optimize the decomposition and compression process of the model. More specifically, we focus on automatically adjusting the rank R in conducting the CP-decomposition for each layer during training. The existing works only intuitively set R with different levels to investigate how this hyperparameter affects the model complexity and model capacity (e.g., Wang et al. 2021a). As demonstrated, if the value of R is small, the model complexity will be significantly reduced, but the model capacity will also be largely decayed. However, how to optimize this hyperparameter is absent. Note that optimizing the value of R for each layer is a highly non-linear and non-convex optimization problem. To address it, we innovatively incorporate deep reinforcement learning (DRL) into the model training process to iteratively train a DRL agent to select the optimal R value for each decomposed attention layer.

In summary, our proposed framework provides an intelligent model compression framework for Transformer, which can both fit the model parameters and optimize the decomposition and compression process of the model by appropriately determining the value

of rank R. This framework does not require additional steps to process the tensors as the other methods (e.g., quantization methods, BTD and CFTT). Instead, researchers can just design their Transformer models as usual and use the proposed framework to train their models. The traditional training settings can be directly used in the newly proposed training platform. For instance, if the original Transformer is set to train 1000 epochs, the Smartformer can also be set to train 1000 epochs. With this framework, we can then obtain a more compact Transformer model with decomposed weight tensors and better convergence property. It is worth noting that although this framework is designed for Transformer, its core idea is transferable to other types of DNN. The contributions of this study can be listed below：

(1). The CPAC-Attention layer is proposed as a low-rank alternative to the Multi-Head Attention layer to enable compressing Transformer;
(2). The theoretical investigations of the newly proposed CPAC-Attention layer are thoroughly conducted, which include the expressions of the forward and backward propagations and the computational properties;
(3). A DRL-based end-to-end training process is proposed to simultaneously optimize the model parameters and decomposed model structures.

In the experiments, we first validate the effectiveness of the newly proposed CPAC-Attention layer on two standard datasets as a proof of concept. By manually setting various values of rank R, we demonstrate the proposed CPAC-Attention layer is able to not only compress the Multi-Head Attention layer in Transformer but also achieve comparable performance to the original model. Moreover, we further validate the strength of the proposed intelligent model compression framework, Smartformer, by conducting practical experiments to optimize the decomposition and compression process of the three latest models. For fair comparisons, the experiments are conducted on the datasets used in the benchmark methods. The results indicate that the Smartformer can always optimize the decomposition of existing Transformers to simultaneously preserve the model structure, reduce model complexity, and improve the model performance.

Results

In this section, we first conduct basic experiments to evaluate the proposed CPAC-Attention layer on two datasets: MNIST (LeCun and Cortes 2010) and electricity transformer temperature (ETT, Zhou et al. 2021). MNIST is a widely used dataset in the computer vision area, while ETT is a new dataset created in the time-series forecasting area. Using these two different datasets, we are able to examine whether a Transformer built upon the CPAC-Attention layers can achieve comparable performance to the original one with fewer parameters. After the basic experiments, we conduct three practical experiments. Specifically, we train three existing variants of Transformer, Spatial-Temporal Tiny Transformer (S3T, Song et al. 2021), Autoformer (Wu et al. 2021), and DEformer (Alcorn and Nguyen 2021), with our proposed framework, Smartformer, to optimize the model decomposition and compression process. For a fair comparison, we use the same datasets as the studies proposing these three models, which are the 2a dataset of brain-computer interface (BCI) competition IV (Brunner et al. 2008), Weather dataset, and binarized-MNIST dataset (Salakhutdinov and Murray 2008). All these three variants of Transformer are structurally different, and they are designed for diverse tasks.

With such a setting, we can explore whether the proposed Smartformer can automatically decompose and compress the original Transformer with the optimized value of *R*. For all the practical experiments, if the DRL agent generates the same model decomposition results in five consecutive epochs, we regard this as a sign of convergence. After convergence, the update of the model structure will be terminated. The experiments are implemented in PyTorch (Paszke et al. 2019) on a single NVIDIA Tesla V100 GPU. A rectified linear unit (ReLU) activation function is used in the basic experiments, and the code is available on https://github.com/XJWang0/Smartformer.

Basic experiments

MNIST dataset: The MNIST dataset is originally collected for handwritten digit recognition and has been widely used in the classification task for the model performance evaluation in the literature. This dataset consists of 60000 28×28 grayscale images for training and 10000 for testing. For this dataset, we use the Vision Transformer (Vi-Transformer) from Dosovitskiy et al. (2021) with three encoders to implement the classification task. In the experiment, we first run the Vi-Transformer, and then replace the Multi-Head Attention layer in each encoder with the CPAC-Attention layer. In order to explore how *R* affects the model performance, we run the CPAC-Attention-based Vi-Transformer (CPAC-Vi-Transformer) by manually changing the rank *R*. Eleven values of *R* are set according to the integer the set $\{1, 5 \times 1, \ldots, 5 \times 10\}$, and the corresponding compression ratio (CR) varies from 0.0205 to 1.0254. The model information and experiment results are summarized in Table 1, from which we can see, the classification accuracy for the original Vi-Transformer is 0.9853 (CR = 1). For the CPAC-Vi-Transformer, when *R* is set as 20, it receives the classification accuracy of 0.9815 with the CR as 0.4102, which indicates the CPAC-Vi-Transformer received a comparable performance to the original model with much fewer parameters. To better illustrate how *R* affects the classification accuracy and loss value when testing, we visualize the results from Table 3 in Fig. A1(a) (Appendix A1). Fig. A1(a) shows that with the increase of *R,* the classification accuracy of the CPAC-Vi-Transformer increases, and the testing loss value decreases. However, this pattern is not strictly monotonically. For example, when *R* increases from 10 to 50, the classification accuracy of the CPAC-Vi-Transformer fluctuates between 0.9754 and 0.9815, and the testing loss also has a small range of variations between 0.0677 and 0.079. This result indicates that the rank *R* affects the model complexity and model performance.

ETT dataset: ETT dataset is collected from an electricity transformer, which contains load and oil temperature that are recorded every 15 minutes between July 2016 and July 2018. Zhou et al. (2021) separated the ETT dataset into $ETTh_1$ and $ETTh_2$ for 1-hour-level and $ETTm_1$ for 15-minute-level. Besides this, Zhou et al. (2021) divided the dataset into the training set, validation set, and testing set, with the time horizons of 12 months, 4 months, and 4 months, respectively. In the experiment, we separate the dataset in the same way and forecast the ETT in the future 24 hours with the $ETTh_1$ dataset. We use the Informer model (Zhou et al. 2021) with two encoders and one decoder. Each encoder in Informer includes one ProbSparse self-attention layer, and the decoder includes a ProbSparse self-attention layer and a traditional self-attention layer. Note that the ProbSparse self-attention is a sparse version of the traditional self-attention layer, which is designed to reduce the time complexity and memory. We replace all the self-attention layers in the Informer model with the CPAC-Attention layers. Twenty-two values of

the rank R are set according to the integer set $\{20 \times 1, \ldots, 20 \times 22\}$, and the corresponding CR varies from 0.0446 and 0.9802. By observing Table 1, we find out that when R equals 100 and the corresponding CR is 0.2228, CPAC-Informer receives the smallest mean squared error (MSE) and the smallest mean absolute error (MAE) as of 0.0775 and 0.2182, respectively. As a comparison, the MSE and MAE for the original Informer model are 0.092 and 0.2385, respectively. We also plot the change of MSE and MAE with different values of R in Fig. A1(b) (see Appendix A1). We can conclude that both metrics have the highest values when R equals 20, and both of them tend to be steady when R is over 180. There are five selections of R that make the CPAC-Informer receive better performances than the original Informer. These results indicate the CPAC-Attention layer can not only reduce the model complexity but also improve its convergence property (model performance).

In summary, the results from the basic experiments denote that the 1) a larger number of parameters may not always lead to a better model performance; 2) when R reaches some level, it cannot significantly affect the model accuracy; 3) the proposed CPAC-Attention is able to use fewer parameters to achieve a comparable or even better performance compared with the original Transformer. The first two findings can be attributed to the model over-fitting issue and the existence of local optima in the over-parameterized model. The third finding indicates that the proposed CPAC-Attention layer can improve the convergence property by appropriately decomposing and compressing the model, which will be further discussed in the practical experiments.

Table 1. Basic experiment results

Model	Rank R	Tensor Size	# of Parameters	CR	Accuracy	Testing Loss
Vi-Transformer -3 × Encoder -1×self-attention layer -1 × FC Layer	——	(64,64)	4,096	1	0.9853	0.0492
CPAC-Vi-Transformer -3 × Encoder Layer -1×CPAC-Attention layer -1 × FC Layer	1	(64, R) (4, R) (16, R)	84R	0.0205	0.8933	0.3381
	5			0.1025	0.9719	0.0918
	10			0.2051	0.9762	0.0727
	15			0.3076	0.9774	0.0677
	20			0.4102	0.9815	0.0668
	25			0.5127	0.9765	0.0740
	30			0.6152	0.9777	0.0722
	35			0.7178	0.9767	0.0728
	40			0.8203	0.9754	0.0791
	45			0.9229	0.9778	0.0703
	50			1.0254	0.9792	0.0667
Model	Rank R	Tensor Size	# of Parameters	CR	MSE	MAE
Informer -2 × Encoder -1×ProbAttention -1 × Decoder -1×ProbAttention -1×self-attention layer	——	(512, 512)	262,144	1	0.0920	0.2385

	20			0.0446	0.1754	0.3476
	40			0.0891	0.1220	0.2804
	60			0.1337	0.0948	0.2389
	80			0.1782	0.1156	0.2726
	100			0.2228	0.0775	0.2182
	120			0.2673	0.0981	0.2477
	140			0.3119	0.1454	0.3131
	160			0.3565	0.1147	0.2710
CPAC-Informer	180			0.4010	0.0810	0.2209
-2 × Encoder	200	(512, R)		0.4456	0.0972	0.2447
-1×CPAC-ProbAttention	220	(8, R)	584R	0.4901	0.0824	0.2237
-1 × Decoder	240	(64, R)		0.5347	0.0970	0.2456
-1×CPAC-ProbAttention	260			0.5792	0.0881	0.2315
-1× CPAC-Attention	280			0.6238	0.1033	0.2544
	300			0.6683	0.0933	0.2391
	320			0.7129	0.1063	0.2566
	340			0.7575	0.0955	0.2425
	360			0.8020	0.0970	0.2431
	380			0.8466	0.1005	0.2496
	400			0.8911	0.0860	0.2301
	420			0.9357	0.0903	0.2340
	440			0.9802	0.0974	0.2430

(Note: Vi-Transformer model is from Dosovitskiy et al. (2021), and Informer model is from Wu et al. (2021))

Practical experiments

Given the insights from the basic experiments, we would like to further explore the potential of the proposed CPAC-Attention layer in improving the convergence property. In this section, we apply the proposed intelligent model compression framework, Smartformer, to the three latest variants of Transformer from the literature. For a fair comparison, we use the same datasets exploited in these studies to demonstrate the strength of the Smartformer, in terms of model accuracy and model complexity.

Case 1: In the first case, we work on the S3T model using the 2a dataset of BCI competition IV. As introduced in Song et al. (2021), this model contains four parts: preprocessing and spatial filter part, spatial transforming part, temporal transforming part, and classifier part. The middle two parts use the self-attention mechanism to extract the spatial and temporal features within the electroencephalogram (EEG) data into a highly distinguishable representation. In particular, spatial transforming exploits a feature-channel attention block to assign weights to different channels so that the model could pay more attention to the relevant channels, while temporal transforming first compresses the data to one dimension and then perceives the global temporal dependencies of EEG signals. In the S3T model, there are three encoders, and each has five multi-attention heads. To train S3T, Song et al. (2021) exploited the Adam optimizer (Kingma and Ba 2017) with a learning rate of 0.0002, the cross-entropy loss function, and the GeLU activation function (Hendrycks and Gimpel 2020). For other specific parameters, please refer to more details in Song et al. (2021). The 2a dataset of BCI competition IV contains the EEG data of nine subjects

with four different tasks, namely, the imagination of moving left hand, right hand, both feet, and tongue. For each subject, two sessions of data are collected, and each of them contains 288 trials (72 trials for each task). In the experiment, we apply the Smartformer to intelligently decompose and train the S3T model and repeat the process on nine subjects separately. We set the action space (rank R) of the DRL agent as {1, 2, 3, 4, 5, 6} and the loss tolerance threshold ϵ as 0.1 (see Eq. (27) about this hyperparameter). The DRL agent takes action every 50 epochs, and there are 2000 epochs in total for the experiment of each subject. Table 2 summarizes the results from both the proposed framework and other benchmark methods. We can conclude that among the benchmark methods, the original S3T model is the current state-of-the-art method with an average classification accuracy of 82.59. Our proposed Smartformer + S3T improves the average classification accuracy of the S3T model to 87.40, which, to the best of our knowledge, has been the best performance on this dataset so far. Moreover, the Smartformer improves the performance of S3T on all nine subjects, which further demonstrates that the proposed framework is able to intelligently decompose and compress the Transformer to both reduce its complexity and improve its performance. In addition, we put the parameter size of each decomposed model (Params) at the bottom of Table 2. It should be noted that the number of parameters in the original S3T is 8.68k, which is larger than all the decomposed models. From this experiment, we can demonstrate that training the Transformer model with Smartformer is able to not only improve the model capacity but also decrease the model parameters.

Table 2. The experiment results of Case 1 (the best one in each column is highlighted with boldface)

Model	**S01**	**S02**	**S03**	**S04**	**S05**	**S06**	**S07**	**S08**	**S09**	**Ave Acc**	**STD**
C2CM	87.50	65.28	90.28	66.67	62.5	45.49	89.58	83.33	79.51	74.46	14.15
CNN+LSTM	85.00	54.00	87.00	78.00	77.00	66.00	95.00	83.00	90.00	80.00	11.97
DFL	91.31	71.62	92.32	78.38	**80.10**	61.62	92.63	90.30	78.38	81. 85	**10.15**
S3T	91.67	71.67	95.00	78.33	61.67	66.67	96.67	93.33	88.33	82.59	12.52
Smartformer	**95.00**	**78.33**	**98.33**	**80.00**	70.00	**70.00**	**100.0**	**96.67**	**98.33**	**87.40**	12.67
Params	8.62k	8.62k	8.16k	8.47k	8.62k	8.62k	8.01k	8.16k	8.01k		

Note: C2CM is from Sakhavi et al. (2018), CNN+LSTM is from Zhang et al. (2019), DFL is from Yang et al. (2021) and S3T is from Song et al. (2021). Smartformer indicates the model of Smartformer + S3T in this case. Ave Acc is short for average accuracy, STD is short for standard deviation, and Params is short for parameter size. The parameter size of S3T is 8.68k.

Case 2: In the second case, we work on the Autoformer. Autoformer renovates Transformer into a decomposition architecture, including the inner series decomposition block, auto-correlation mechanism, and corresponding encoder and decoder for long-term time-series forecasting. As introduced in Wu et al. (2021), this model progressively divides the long-term trend information from predicted hidden variables. Thus, it can decompose and refine the intermediate results during the forecasting procedure. Wu et al. (2021) used five datasets to showcase the performance of the proposed model. For this experiment, we select the Weather dataset which records meteorological indicators (e.g., air temperature, humidity, etc.) every 10 minutes for the entire year 2020. As the same as Wu et al. (2021), we exploit four different prediction horizons, $O \in \{96,192,336,720\}$, to demonstrate the forecasting performance of the

proposed framework, and split the weather dataset into the training, validation, and testing set in chronological order by the ratio of 7:1:2. In addition, we make use of the L2 loss function, GeLU activation function (Hendrycks and Gimpel 2020), and Adam optimizer (Kingma and Ba) with an initial learning rate of 10^{-4}. It should be noted that in this case, Wu et al. (2021) set the training process to be early stopped within 10 epochs. Therefore, the DRL agent takes actions within each epoch, more specifically, every 100 batches. The architecture of Autoformer consists of two encoders and one decoder. Each of them contains eight multi-attention heads, and the parameter size is with the shape of 512×512. We set the action space (rank R) as {40, 50, 60, ..., 440, 450} and the loss tolerance threshold ϵ is 0.125. Similar to Wu et al. (2021), Informer (Zhou et al. 2021), LogTrans (Li et al. 2019), and Reformer et al. (Kitaev et al. 2020) are used as benchmark models, and MSE and MAE are used as evaluation metrics. We present the experiment results in Table 3, from which we can see that among all the benchmark methods, Autoformer receives the state-of-the-art performance. Our proposed Smartformer can further improve the performance of Autoformer in all four cases with fewer parameters. More specifically, when we train Autoformer using Smartformer (Smartformer + Autoformer), the average number of parameters on the four cases is 9.44 Million (M), which is 10.9% lower than that of the original Autoformer. Note that the number of parameters in the Autoformer is 10.60M. In summary, the results validate the effectiveness of the proposed Smartformer in improving the model performance and reducing the model complexity.

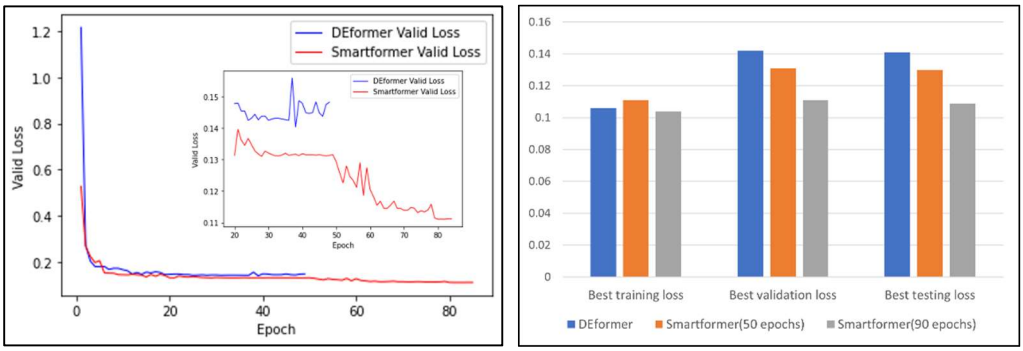
Table 3. The experiment results of Case 2 (the best one in each column is highlighted with boldface)

Model	**Metrics**	**96**	**192**	**336**	**720**
Informer	MSE	0.300	0.598	0.578	1.059
	MAE	0.384	0.544	0.523	0.741
LogTrans	MSE	0.458	0.658	0.797	0.869
	MAE	0.490	0.589	0.652	0.675
Reformer	MSE	0.689	0.752	0.639	1.130
	MAE	0.596	0.638	0.596	0.792
LSTNet	MSE	0.594	0.560	0.597	0.618
	MAE	0.587	0.565	0.587	0.599
Autoformer	MSE	0.266	0.307	0.359	0.419
	MAE	0.336	0.367	0.395	0.428
Smartformer	MSE	**0.250**	**0.305**	**0.345**	**0.410**
	MAE	**0.316**	**0.364**	**0.383**	**0.423**
Params		9.15M	10.23M	8.45M	9.92M

Note: Informer is from Zhou et al. (2021), LongTrans is from Li et al. (2019), Reformer is from Kitaev et al. (2020), Autoformer is from Wu et al. (2021). Smartformer indicates the

model of Smartformer + Autoformer. The parameter size of Autoformer is 10.60M. 96,192,336 and 720 represent the hourly forecasting horizon (e.g., 96 means the next 96 hours). Params is short for parameter size.

Case 3: In the third case, we work on the DEformer and the binarized-MNIST dataset. The DEformer is an entirely order-agnostic distribution estimating Transformer and is competitive with fixed-order distribution estimating algorithms (Alcorn and Nguyen 2021). The DEformer contains six encoders, and each of them has eight multi-attention heads. In DEformer, the dimensions of the input and output of each Multi-Head Attention layer are 512, and the dimension of each inner feed-forward layer is 2048. The binarized-MNIST dataset contains 70000 28×28 binary images representing digits (0-9). Among them, 60000 samples are split as the training dataset, and the rest 10000 samples are the testing dataset. For a fair comparison, we use the same experiment settings as Alcorn and Nguyen (2021), in which 1200 images from the training dataset are set aside for validation, and Adam optimizer (Kingma and Ba 2017) with an initial learning rate of 10^{-6} is selected to update the model parameters. For other parameter settings, please refer to Alcorn and Nguyen (2021) for more details. In the experiment, we train the DEformer using the proposed framework, Smartformer, and the number of epochs is firstly set as 50, which is the same as the experiment setting in Alcorn and Nguyen (2021). We set the action space (rank R) as {40, 50, 60, …, 440, 450}, and the corresponding CR is between 0.1 and 1. Similar to Case 2, the DRL agent takes action every 5000 batches, and the loss tolerance threshold ϵ is set as 0.25. In this case study, we compare the convergence performance of the DEformer with and without using Smartformer by visualizing the change of validation loss with respect to (w.r.t.) the training epoch in Fig. 1(a) and plotting the best training, validation, and testing losses in Fig. 1(b). We can summarize that the validation loss of DEformer without using Smartformer converges within 50 epochs, and the best testing loss is 0.141. By contrast, the validation loss of DEformer with using Smartformer has a better convergence performance starting from the 20th epoch to the 50th epoch (highlighted in the zoom-in plot in Fig. 1(a)), and the best testing loss is 0.130 at the 50th epoch. Furthermore, when we increase the training epoch for Smartformer, it continuously improves the decomposition and compression process of DEformer and converges to a smaller validation loss when approaching the 90th epoch. Correspondingly, as displayed in Fig. 1(b), the best testing loss is 0.109 at the 90th epoch when training the DEformer with Smartformer, which is much smaller than that (0.141) of the original DEformer. It is also worth noting that the parameter size of the compressed DEformer is 15.67M, which is almost 20% smaller than that (19.24M) of the original model. These results indicate that Smartformer can not only reduce the model parameters but also improves the convergence property compared with the original model.



(a) (b)

Figure 1. Experiment results of Case 3: (a) is the validation loss plot and (b) is the best training loss, validation loss and testing loss comparison results using DEformer and the decomposed. Note that the Smartformer in this case is the model of Smartformer + DEformer.

Discussion

Although Transformer has achieved success in wide application areas, it is also limited by the issues caused by the over-parameterization, such as trapping in local optima during training, redundancy in parameters, high computational cost, etc. Existing works have focused on addressing these issues in several ways, but none of them have taken the three objectives, i.e., preserving model architecture, maintaining model capacity, and reducing the model complexity to improve its convergence property, simultaneously into account. Considering these, we propose Smartformer, an intelligent model compression framework for Transformer by incorporating DRL and CP-decomposition.

The main theoretical contributions of this study are threefold: 1) we introduce the CPAC-Attention layer as a low-rank alternative to replace the Multi-Head Attention layer for compressing the Transformer model; 2) we theoretically derive the expressions of forward and backward propagations for the CPAC-Attention layer as well as its the computational properties; 3) we develop an end-to-end model compression framework using DRL to simultaneously fit the model parameters and optimize the model structures.

To demonstrate the effectiveness of the proposed Smartformer, we conduct two types of experiments, i.e., basic experiments and practical experiments. The basic experiments are designed to showcase the effectiveness of the CPAC-Attention layer and give an insight into how the value of rank R influences its performance. More specifically, we replace the Multi-Head Attention layer in two existing models, Vi-Transformer and Informer, with the newly proposed CPAC-Attention layer and conduct experiments on MNIST and ETT datasets by changing the value of R. The results indicate that when using the proposed CPAC-Attention layer in Transformer, it is likely to not only compress the model, but also achieve the comparable or even better performance when appropriately selecting the value of rank R. This finding motivates the necessity of our proposed Smartformer, which both fits the model parameters and optimizes the model decomposition and compression in the training process. We further conduct practical experiments to validate the effectiveness of the entire framework. More specifically, we re-train three existing variants of Transformer, i.e., S3T, Autoformer, and DEformer, with Smartformer to find the best model compression solution. The results clearly denote that following the same experimental settings as the original Transformer models, the Smartformer can improve the model performance by appropriately compressing the model with an optimized R. Especially, we achieve the state-of-the-art classification accuracy on the 2a of BCI competition IV dataset and the best forecasting accuracy on the Weather dataset. We can conclude that our proposed Smartformer provides a general training framework for the Transformer to simultaneously preserve the model architecture, maintain model capacity, and reduce the number of parameters to improve its convergence property. Its idea also has a broader impact on intelligently compressing other types of DNN.

Methods

In this section, we propose an intelligent model compression framework for Transformer, which is named after Smartformer, by incorporating DRL and CP-decomposition. Before proceeding to the Transformer model of decomposition, we first introduce the necessary theoretical backgrounds, i.e., CP-decomposition, Matrix calculus, Attention, and Transformer, in Appendix A2. The basic idea of our proposed Smartformer is:

(1) Since the CP-decomposition can approximate the high-order tensor by multiple groups of vectors, we propose to incorporate CP-decomposition to approximately compress the attention mechanism (CPAC-Attention) by replacing its weight matrices with multiple groups of rank-one vectors.

(2) When building the Transformer using CPAC-Attention, its performance is inevitably influenced by the hyperparameter rank R introduced in the CP-decomposition (see Eq. (A1)). Thus, we propose an intelligent framework, Smartformer, which is designed to leverage DRL to iteratively select a specific Multi-Head Attention layer, decompose its weight matrices, and optimize the hyperparameter R.

(3) To ensure the model performance when reducing the parameters, the layer importance is specifically defined to select the least important layer for approximation, such that its influence on the model performance will be mitigated.

In the following, we firstly adapt the CP-decomposition method to compress the attention mechanism. Then, we formulate the Smartformer by exploiting the CPAC-Attention mechanism and introducing the DRL algorithm.

Adapt CP-decomposition to approximately compress the attention mechanism

The attention mechanism is the key factor determining the outstanding performance of the Transformer, which also contributes to the majority of model parameters and computational cost. In this section, we propose the CPAC-Attention layer as a low-rank alternative to the Multi-Head Attention layer used in the Transformer. The expressions of forward and back propagations of the proposed CPAC-Attention layer are fully derived. The computational properties of the CPAC-Attention layer are also discussed.

Forward propagation of CPAC-Attention layer

In the Multi-Head Attention layer, the input is denoted as $I \in \mathbb{R}^{S\times hd}$, where S is the number of samples, h is the number of attention heads, and d is the input dimension. Then, the Q, K, V are generated by mapping the input via weight matrices $W^q \in \mathbb{R}^{hd\times hd}$, $W^k \in \mathbb{R}^{hd\times hd}$, and $W^v \in \mathbb{R}^{hd\times hd}$, respectively. We can find out that most of parameters in the Multi-Head Attention layer exist in these weight matrices. Without the loss of generality, the input and output of the mapping are assumed to have the same dimension d for the simplicity of notation. Because the expressions to calculate Q, K, V are the same, we only use Q as an example to generate the detailed derivations. In the original Multi-Head Attention layer, Q is generated by Eq. (A7).

The input for the Transformer can be reformulated into a three-way tensor $\mathcal{I} \in \mathbb{R}^{S\times h\times d}$. The reformulation process is visually demonstrated in Fig. A6 (see Appendix A3). Similarly,

the weight matrix W^q can be reformulated as $\mathcal{W}^q \in \mathbb{R}^{h\times d\times hd}$. However, it is not intuitive to directly reformulate Eq. (A7) into the tensor operation between $\mathcal{I}$ and $\mathcal{W}^q$. Instead, the CP-decomposition is firstly applied to weight tensor $\mathcal{W}^q$, which is given in Eq. (1):

$$\mathcal{W}^q = \sum_{r=1}^{R} q_r^d \circ q_r^h \circ q_r^{hd}. \tag{1}$$

In Eq. (1), q_r^d, q_r^h, and q_r^{hd} denote the decomposed vectors along each dimension of $\mathcal{W}^q$. Then, the tensor expression of Eq. (A7) can be formulated as follows:

$$Q = \sum_{r=1}^{R} \left(\left(\mathcal{I} \times_3 q_r^d \right) \times_2 q_r^h \right) \circ q_r^{hd}. \tag{2}$$

Comparing the calculations of Q between the original Multi-Head Attention layer and the proposed CPAC-Attention layer, we can conclude that our proposed CPAC-Attention layer provides a more detailed breakdown of model weights, thus enabling a more flexible model design by controlling the value of R to select the most informative subgroups of parameters.

Similarly, the tensor expressions to calculate K and V are formulated in Eqs. (3) and (4), respectively, where k_r^d, k_r^h, k_r^{hd} and v_r^d, v_r^h, v_r^{hd} are decomposed vectors of $\mathcal{W}^k$ and $\mathcal{W}^v$, separately. After generating the Q, K, and V, the rest operations in the forward propagation for our proposed CPAC-Attention layer remain the same as the original Multi-Head Attention layer.

$$K = \sum_{r=1}^{R} \left(\left(\mathcal{I} \times_3 k_r^d \right) \times_2 k_r^h \right) \circ k_r^{hd}. \tag{3}$$

$$V = \sum_{r=1}^{R} \left(\left(\mathcal{I} \times_3 v_r^d \right) \times_2 v_r^h \right) \circ v_r^{hd}. \tag{4}$$

Backward propagation of CPAC-Attention layer

To complete the theoretical investigation and provide insights into how the CPAC-Attention layer will influence the training process, we derive the backward propagation of our proposed CPAC-Attention layer as follows. Since the CPAC-Attention layer only modifies the mapping from the input I to Q, K, V, we can denote the operations afterward as a function $g(\cdot)$. Thus, the output of the CPAC-Attention layer is:

$$Y = g(Q, K, V). \tag{5}$$

Gradient-based optimization methods are commonly used in backward propagation to train the model parameters. Also, taking Q as an example, we need to derive the gradients of the layer output Y with respect to (w.r.t.) the newly introduced weight vectors $q_r^d, q_r^h, q_r^{hd}, r = 1, \dots, R$, which are $(\frac{\partial Y}{\partial q_r^d}, \frac{\partial Y}{\partial q_r^h}, \frac{\partial Y}{\partial q_r^{hd}})$.

Partial derivative w.r.t. q_r^{hd}: According to the chain rule, the expression of $\frac{\partial Y}{\partial q_r^{hd}}$ is shown in Eq. (6), where the first component is the same as the original Multi-Head Attention layer. The $\frac{\partial Q}{\partial q_r^{hd}}$ is then derived as follows.

$$\frac{\partial Y}{\partial q_r^{hd}} = \frac{\partial Y}{\partial Q}\frac{\partial Q}{\partial q_r^{hd}}. \tag{6}$$

First, the differential of Eq. (2) w.r.t. q_r^{hd} at both sides are calculated:

$$dQ = \left(\left(\mathcal{J} \times_3 q_r^d\right) \times_2 q_r^h\right) \circ dq_r^{hd}. \tag{7}$$

After vectorizing both sides, we have Eq. (8):

$$\mathrm{vec}(dQ) = \mathrm{vec}\left(\left(\left(\mathcal{J} \times_3 q_r^d\right) \times_2 q_r^h\right) \circ dq_r^{hd}\right). \tag{8}$$

We use $A_1 \in R^{S\times 1}$ to denote the constant part $\left(\left(\mathcal{J} \times_3 q_r^d\right) \times_2 q_r^h\right)$, and $dq_r^{hd} \in R^{hd\times 1}$. Substituting A_1 into Eq. (8) yields:

$$\begin{aligned}\mathrm{vec}(dQ) &= \mathrm{vec}\left(A_1 \circ dq_r^{hd}\right)\\ &= \mathrm{vec}\left(A_1[dq_r^{hd}]^\top\right)\\ &= (I_M \otimes A_1)\mathrm{vec}\left([dq_r^{hd}]^\top\right)\\ &= (I_M \otimes A_1)\mathrm{vec}\left(dq_r^{hd}\right)\end{aligned} \tag{9}$$

Moving $\mathrm{vec}\left(dq_r^{hd}\right)$ to the left side of Eq. (9), we have:

$$\frac{\mathrm{vec}(dQ)}{\mathrm{vec}\left(dq_r^{hd}\right)} = (I_M \otimes A_1) \tag{10}$$

$$\frac{\partial Q}{\partial q_r^{hd}} = I_M \otimes A_1^\top. \tag{11}$$

In Eq. (10), since $dq_r^{hd} \in R^{hd\times 1}$, we then have $\mathrm{vec}\left([dq_r^{hd}]^\top\right) = \mathrm{vec}\left(dq_r^{hd}\right)$.

Partial derivative w.r.t. q_r^h: The expression of $\frac{\partial Y}{\partial q_r^h}$ is shown in Eq. (12), and the $\frac{\partial Q}{\partial q_r^h}$ is derived as follows.

$$\frac{\partial Y}{\partial q_r^h} = \frac{\partial Y}{\partial Q}\frac{\partial Q}{\partial q_r^h} \tag{12}$$

First, we calculate the differential of Eq. (2) w.r.t. q_r^h:

$$dQ = \left(\left(\mathcal{J} \times_3 q_r^d\right) \times_2 dq_r^h\right) \circ q_r^{hd}. \tag{13}$$

After vectorizing both sides, we have Eq. (14):

$$\mathrm{vec}(dQ) = \mathrm{vec}\left(\left(\left(\mathcal{J} \times_3 q_r^d\right) \times_2 dq_r^h\right) \circ q_r^{hd}\right). \tag{14}$$

We use $A_2 \in R^{S\times h}$ to denote the constant part $\left(\mathcal{J} \times_3 q_r^d\right)$, and $B_1 \in R^{hd\times 1}$ to denote the constant part q_r^{hd}. After substituting A_2 and B_1 into Eq. (14), we can simplify it into Eq. (15):

$$\mathrm{vec}(dQ) = \mathrm{vec}\left(A_2 \times_2 dq_r^h \circ B_1\right)$$

$$= \mathrm{vec}\left(A_2 dq_r^h B_1^T\right)$$
$$= (B_1 \otimes A_2)\mathrm{vec}\left(dq_r^h\right) \quad (15)$$

Moving $\mathrm{vec}\left(dq_r^h\right)$ to the left side of Eq. (15), we have:

$$\frac{\mathrm{vec}(dQ)}{\mathrm{vec}\left(dq_r^h\right)} = (B_1 \otimes A_2), \quad (16)$$

$$\frac{\partial Q}{\partial q_r^h} = B_1^\top \otimes A_2^\top. \quad (17)$$

Partial derivative w.r.t. $\boldsymbol{q_r^d}$: The expression of $\frac{\partial Y}{\partial q_r^d}$ is shown in Eq. (18), and the $\frac{\partial Q}{\partial q_r^d}$ is derived as follows.

$$\frac{\partial Y}{\partial q_r^d} = \frac{\partial Y}{\partial Q}\frac{\partial Q}{\partial q_r^d} \quad (18)$$

First, we calculate the differential of Eq. (19) w.r.t. q_r^d:

$$dQ = \left(\left(\mathcal{J} \times_3 dq_r^d\right) \times_2 q_r^h\right) \circ q_r^{hd}. \quad (19)$$

We use $\mathcal{J}_{(3)} \in R^{d \times Sh}$ to denote the mode-3 unfolding of $\mathcal{J}$. Thus, we can rewrite the inner part $\left(\mathcal{J} \times_3 dq_r^d\right) \in R^{S \times h}$ of Eq. (19) into $P\left(\mathcal{J}_{(3)}^\top dq_r^d\right) \in R^{S \times h}$. $P(\cdot)$ represents a permutation operator, which means $(\mathcal{J} \times_3 dq_r^d)$ and $(\mathcal{J}_{(3)} dq_r^d)$ are matrices containing the same elements, but arranged differently. Hence, Eq. (19) can be transformed into Eq. (20):

$$dQ = \left[P\left(\mathcal{J}_{(3)}^\top dq_r^d\right) q_r^h\right]\left(q_r^{hd}\right)^\top. \quad (20)$$

Furthermore, we use $B_2 \in R^{h \times hd}$ to denote $q_r^h (q_r^{hd})^\top$. By substituting B_2 into Eq. (20) and vectorizing both sides, we obtain Eq. (21):

$$\mathrm{vec}(dQ) = \mathrm{vec}\left(P\left(\mathcal{J}_{(3)}^\top dq_r^d\right) B_2\right)$$
$$= (B_2^\top \otimes I_{Sh})\mathrm{vec}\left(P\left(\mathcal{J}_{(3)}^\top dq_r^d\right)\right). \quad (21)$$

As $P(\cdot)$ is a permutation operator, we can show that $\mathrm{vec}\left(P\left(\mathcal{J}_{(3)}^\top dq_r^d\right)\right)$ is the same as $P\left(\mathrm{vec}\left(\mathcal{J}_{(3)}^\top dq_r^d\right)\right)$:

$$\mathrm{vec}(dQ) = (B_2^\top \otimes I_{Sh})\mathrm{vec}\left(P\left(\mathcal{J}_{(3)}^\top dq_r^d\right)\right)$$
$$= (B_2^\top \otimes I_{Sh})P\left(\mathrm{vec}\left(\mathcal{J}_{(3)}^\top dq_d^d\right)\right)$$
$$= (B_2^\top \otimes I_{Sh})P\left(\left(I_1 \otimes \mathcal{J}_{(3)}^\top\right)\mathrm{vec}\left(dq_r^d\right)\right)$$
$$= (B_2^\top \otimes I_{Sh})P\left(\left(I_1 \otimes \mathcal{J}_{(3)}^\top\right)\right)\mathrm{vec}\left(dq_r^d\right). \quad (22)$$

$$\frac{\mathrm{vec}(dQ)}{\mathrm{vec}\left(dq_r^d\right)} = (B_2^\top \otimes I_{Sh})P\left(\left(I_1 \otimes \mathcal{J}_{(3)}^\top\right)\right). \quad (23)$$

$$\frac{\partial Q}{\partial q_r^d} = \left[(B_2^\top \otimes I_{Sh})P\left(\left(I_1 \otimes \mathcal{J}_{(3)}^\top\right)\right)\right]^\top$$
$$= P\left(\left(I_1 \otimes \mathcal{J}_{(3)}^\top\right)\right)^\top (B_2^\top \otimes I_{Sh})^\top$$
$$= P\left(\mathcal{J}_{(3)}^\top\right)(B_2 \otimes I_{Sh}). \quad (24)$$

Combining Eqs. (11), (17), and (24), we generate the detailed expressions of $(\frac{\partial Q}{\partial q_r^{hd}}, \frac{\partial Q}{\partial q_r^h}, \frac{\partial Q}{\partial q_r^d})$, which will be used to update decomposed kernels of CPAC-Attention layer in the backward propagation.

Up to now, we have derived the forward and backward propagations of a CPAC-Attention layer. In the following, we will discuss the properties of our proposed CPAC-Attention layer theoretically.

Properties of the CPAC-Attention

The theoretical properties of our proposed CPAC-Attention layer are investigated in this section, which includes the number of parameters, time complexity, and required memory.

Analysis of the number of parameters: Our proposed CPAC-Attention layer provides a detailed breakdown of the weight matrices used in the original Multi-Head Attention layer, which can reduce the number of parameters by controlling the hyperparameter R. Also, using the weight matrix for calculating Q as an example, the number of parameters is $N_1 = (hd)^2$ for the weight matrix $W^q \in R^{hd \times hd}$. In our proposed CPAC-Attention layer, we decompose the original weight matrices into multiple groups of vectors q_r^{hd}, q_r^h, and $q_r^d, (r = 1, \ldots, R)$. The number of parameters in decomposed vectors is $N_2 = R(hd + h + d)$. The CR is defined as the ratio of parameters in the CPAC-Attention layer and the Multi-Head Attention layer. By appropriately selecting the value of R, the newly proposed CPAC-Attention layer can use fewer parameters to approximate the original Multi-Head Attention layer.

Analysis of floating-point operations: To theoretically evaluate the time complexity of the newly proposed CPAC-Attention layer, we calculate the number of FLoating-point OPerations (FLOPs), which is to count the number of basic operations, such as addition, multiplication, etc., in neural networks. We also use Q as an example to present the detailed calculation of FLOPs. In the original Multi-Head Attention layer, the input is $I \in R^{S \times hd}$, the weight matrix is $W^q \in R^{hd \times hd}$, and the output is $Q \in R^{S \times hd}$. Thus, according to Eq. (A7), the number of FLOPs in calculating Q is $Shd(2hd - 1)$. Given the calculations of Q, K, V follow the same rule, the number of FLOPs in the Multi-Head Attention layer is written as:

$$\text{FLOPs}_{Multi-Hea\ \ Attention} = 3Shd(2hd - 1) \quad (25)$$

Our proposed CPAC-Attention layer can be regarded as a series of linear operations. Thus, the number of FLOPs is given in Eq. (26):

$$\text{FLOPs}_{CPAC-Attention} = 3RS[h(3d + 1) - 1] \quad (26)$$

By comparing Eqs. (25) and (26), we can conclude that the number of FLOPs of the CPAC-Attention layer can be much smaller than the original Multi-Head Attention layer by appropriately selecting the value of rank R in CP-decomposition.

Analysis of required memory: When considering the required memory of a model, it commonly consists of two parts: the memory taken by model parameters and the memory taken by intermediate variables. The intermediate variables exist in both the forward and backward propagations. In the forward propagation, the intermediate variables include the output of each layer, which is the same between the Multi-Head Attention layer and our proposed CPAC-Attention layer. In the backward propagation, the intermediate variables mainly include the gradients of model parameters. According to the above analysis, the decrease of model parameters in our proposed CPAC-Attention layer can accordingly reduce the memory required by model parameters and their gradients.

The parameters are usually stored in the single-precision float format, and each parameter can take 32 bits in the computer memory, which is 4 Bytes (B). Thus, for our proposed CPAC-Attention layer, the memory required by model parameters (consider all weight matrices for Q, K, V) is $4 \times 3R(hd + h + d) \times 10^{-6}$ Megabytes (MB). For the original Multi-Head Attention layer, the memory required by model parameters is $4 \times 3(hd)^2 \times 10^{-6}$ MB. Therefore, the parameters in the CPAC-Attention layer take less memory compared with the original Multi-Head Attention layer by appropriately selecting the value of R. Similarly, in the backward propagation, the gradients are calculated w.r.t. each parameter. Therefore, the memory taken by gradients will be reduced accordingly.

In summary, a comparison of the properties of the proposed CPAC-Attention layer and the original Multi-Head Attention layer is presented in Table 4.

Table 4. Comparison of properties between CPAC-Attention and Multi-Head Attention layers

	Multi-Head Attention	CPAC-Attention
Number of Parameters	$3(hd)^2$	$3R(hd + h + d)$
FLOPs	$3Shd(2hd - 1)$	$3RS[h(3d + 1) - 1]$
Required Memory (MB)	$12(hd)^2 \times 10^{-6}$	$12R(hd + h + d) \times 10^{-6}$

The development of Smartformer

The newly proposed CPAC-Attention layer provides a low-rank alternative to the Multi-Head Attention layer in the Transformer. Given the fact that a Transformer model usually consists of a stack of multiple Multi-Head Attention layers, the instant question is how to optimally select the hyperparameter R when decomposing each Multi-Head Attention layer to both reduce the parameters and maintain the model performance? To tackle this problem, we propose an intelligent model decomposition and compression framework, Smartformer, by incorporating the DRL algorithm (Sutton and Barto 2018) to sequentially update the value of R so as to appropriately replace the original Multi-Head Attention layers with the newly proposed CPAC-Attention layers.

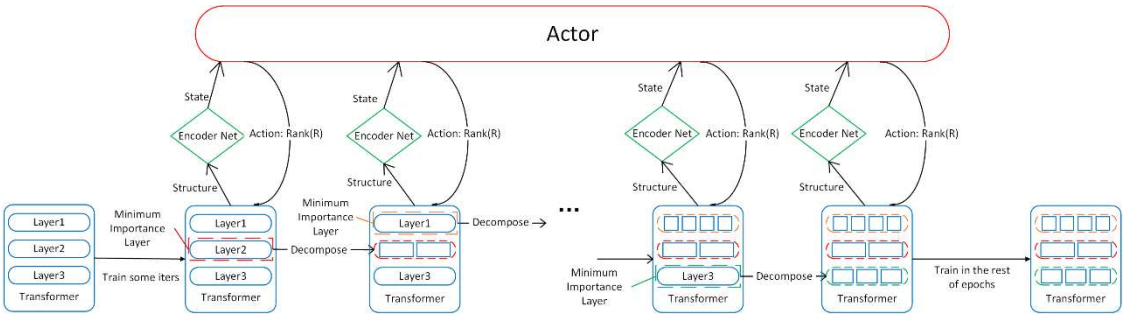


Figure 2. Overview of the architecture of Smartformer.

In Smartformer, the model structure re-design process is formulated as a Markov decision process (MDP), which consists of four key elements: state, action, reward, and transition probability. The transition probability is implicitly modeled by the policy network. A DRL agent (modeled by a neural network) is trained to maximize the cumulative reward by learning from repeated episodes. Each episode is a sequence of states, actions, and rewards. The basic steps to generate one episode are summarized in Fig. 2. Generally, there are five steps:

Step (1): Given the current structure of a Transformer, the training process is conducted to obtain the average importance score of each Multi-Head Attention layer and the reward obtained by the previous action. Please see the "Layer importance" in Appendix A4 for more details about the calculation of the layer importance score.

Step (2): The Multi-Head Attention layer with the minimum average importance score in the current Transformer architecture is selected to be replaced. The state in MDP is updated by encoding the current structure of the Transformer (introduced in the following section).

Step (3): The action (value of rank R in CP-decomposition) is accordingly updated by feeding the current state into the policy network. The value of the current state is estimated by the value network.

Step (4): Given the action, a CPAC-Attention layer is built to replace the selected Multi-Head Attention layer in the Transformer.

Step (5): The entire process is repeated from Step (1) till all layers are decomposed. The Advantage Actor-Critic (A2C) algorithm (Konda and Tsitsiklis 2000) are applied to update the parameters of the policy network and value network.

Reward

The reward function used in the DRL is defined as Eq. (27).

$$R(a,s) = \begin{cases} acc/acc_0\,, & \text{if } acc \geq acc_0\varepsilon \\ -\,acc_0/acc\,, & \text{else} \end{cases}, \tag{27}$$

where acc represents the average accuracy of training after decomposition, acc_0 represents the average accuracy of training before decomposition, and ϵ is the loss tolerance threshold that can be adjusted manually based on the preference of practitioners. If one prefers to compress the model much, ϵ can be set large; otherwise, ϵ can be set small to maintain the model performance. Intuitively, when a decomposition improves the model's performance over the given threshold, it will be encouraged by giving a positive reward. Otherwise, the reward is

negative. The goal of this framework is to maximize the cumulative reward.

State and action

State in MDP is to describe the current observation, which is fed into the policy network to generate the action. In our problem, the state is determined by the current structure of Transformer, and the action is to determine the value of rank R of the selected layer of interest given the current state. Following Cai et al. (2018), we exploit an encoder net to encode the current structure of Transformer into vectors with the following steps: (1) a dictionary containing all architectures of layers is established, and the given network structure is then represented as a string, which is fed into the dictionary to look up the corresponding sequence of indexes; (2) an embedding layer is followed to map each index into a vector; (3) the vector corresponding to the layer of interest is selected through positional encoding; (4) a long short-term memory (LSTM) network is further used to process the vector. The output of the encoder net is the state representation corresponding to the input layer, which will be further entered into the policy network and value network to update the action and estimate its value, respectively.

Advantage Actor-Critic

Advantage Actor-Critic (A2C) algorithm consists of a policy network and a value network. The policy network determines the action (value of rank R) given the current state, which is denoted as $\pi(a|s,\theta)$. The value network gives an estimation of the expected total reward of being in the current state and then following the current policy, which is denoted as $v^{\pi}(s|\psi)$. The policy function $\pi(a|s,\theta)$ and value function $v^{\pi}(s|\psi)$ are approximated by separate neural networks. They are all made up of two fully connected layers and ReLU activation. The loss function of the Actor is Negative Log-likelihood Loss, and the loss function of Critic is a linear function. In this study, we use the policy gradient and temporal-difference to update the parameters of the policy network and the value network, respectively.

Endnote

Weather dataset is from https://www.bgc-jena.mpg.de/wetter/.

Our code is available at https://github.com/XJWang0/Smartformer.

Reference

Alcorn, M. A., Nguyen, A. M. (2021) The DEformer: an order-agnostic distribution estimating Transformer, ICML Third Workshop on *Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models*.

Brunner, C., Leeb, R., Muller-Putz, G. R., Schlogl, A. (2008) BCI competition 2008-Graz data set A, p. 6.

Cai, H., Chen, T., Zhang, W., Yu, Y., Wang, J. (2018) Efficient architecture search by network transformation. *The Thirty-Second AAAI Conference on Artificial Intelligence*, 2787-2794.

Chen, P. H., Yu, H., Dhillon, I. S., Hsieh, C. (2021) DRONE: data-aware low-rank compression for large NLP models. *35th Conference on Neural Information Processing Systems*.

Cornia, M., Stefanini, M., Baraldi, L., Cucchiara, R. (2020) Meshed-memory Transformer for image captioning. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern*

Recognition (CVPR), 10578-10587.
Carroll, J. D., Chang, J-J. (1970) Analysis of individual differences in multidimentional scaling via an n-way generalization of "eckart-young" decomposition. *Psychometrika*, 35(3): 283-319.

Ding, T., Li, D., Sun, R. (2022) Suboptimal Local Minima Exist for Wide Neural Networks with Smooth Activations. *Mathematics of Operations Research*, DOI: https://doi.org/10.1287/moor.2021.1228

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T.,
Han, K., Wang, Y., Chen, H., Chen, X., Guo, J., Liu, Z., Tang, Y., Xiao, A., Xu, C., Xu, Y., Yang, Y., Tao, D. (2022) A survey on vision Transformer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, DOI: 10.1109/TPAMI.2022.3152247

Hendrycks, D., Gimpel, K. (2020) Gaussian error linear units (GELUs), Preprint *arXiv: 1606.08415*.

Kiers, H. A. L. (2000) Towards a standardized notation and terminology in multiway analysis. *Journal of Chemometrics*, 14(3), 105-122.
Kim, S., Gholami, A., Yao, Z., Mahoney, M. W., Keutzer, K. (2021) I-BERT: integer-only BERT quantization. *Proceedings of the 38th International Conference on Machine Learning*.

Kingma, D. P., Ba, J. (2017) Adam: a method for stochastic optimization. Preprint *arXiv: 1412.6980.*

Kitaev, N., Kaiser, L., Levskaya, A. (2020) Reformer: the efficient transformer. International Conference of Learning Representation.

Konda, V. R., Tsitsiklis, J. N. (2000) Actor-critic algorithms. *Advances in Neural Information Processing Systems*, 1008-1014.
Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I., Lempitsky, V. (2015) Speedingup convolutional neural networks using fine-tuned CP-decomposition. *International Conference on Learning Representation*.
LeCun, Y., Cortes, C. (2010) MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/. (accessed February 2022).

Li, J., Sun, Y., Su, J., Suzuki, T., Huang, F. (2020) Understanding generalization in deep learning via tensor methods. *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS)*, Palermo, Italy. Volume: 108.

Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y-X., Yan, X. (2019) Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *33rd Conference on Neural Information Processing Systems,* Vancouver, Canada.
Li, X., Shi, Q., Hu, G., Chen, L., Mao, H., Wang, Y., Yuan, M., Zeng, J., Cheng, Z. (2021) Block access pattern discovery via compressed full tensor Transformer. *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 957-966. November 1-5, Virtual Event, QLD, Australia.
Liu, Y., Xiong, H., He, Z., Zhang, J., Wu, H., Wang, H., Zong, C. (2019) End-to-end speed translation with knowledge distillation. *Proceedings of the 17th International Conference on Spoken Language Translation*, 80-88.
Ma, R., Wang, C., Li, X. (2021) CP decomposition for fast training of Bi-LSTM. *2021 IEEE International Conference on Dependable, Automatic and Secure Computing*, 25-28 Oct. 2021, AB, Canada.
Ma, X., Zhang, P., Zhang, S., Duan, N., Hou, Y., Song, D., Zhou, M. (2019) A tensorized Transformer for language modeling. *33rd Conference on Neural Information Processing*

Systems, Vancouver, Canada.

Magnus, J. R., Neudecker, H. (1985) Matrix differential calculus with applications to simple, hadamard, and kronecker products. *Journal of Mathematical Psychology*, 29(4), 474-492.

Molchanov, P., Mallya, A., Tyree, S, Frosio, I., Kautz, J. (2019) Importance estimation for neural network pruning. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 11256-11264.

Neyshabur, B., Bhojanapalli, S., McAllester, D. and Srebro, N. (2017) Exploring generalization in deep learning, *Advances in Neural Information Processing Systems*, 5949–5958.

O' Neill, J. T. (2020) A survey of neural network compression. Preprint *arXiv: 2006.03669*.

Panahi, A., Saeedi, S., Arodz, T. (2021) Shapeshifter: a parameterized-efficient Transformer using factorized reshaped matrices. *35th Conference on Neural Information Processing Systems*.

Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L, Shazeer, N., Ku, A., Tran, D. (2018) Image Transformer. *Proceedings of the 35th International Conference on Machine Learning*, Stockholm, Sweden.

Prato, G., Charlaix, E., Rezagholizadeh, M. (2020) Fully quantized transformer for machine translation. *Findings of the Association for Computational Linguistics: EMNLP*, 1-14.

Reid, M., Marrese-Taylor, E., Matsuo, Y. (2021) Subformer: exploring weight sharing for parameter efficiency in generative Transformers. Preprint *arXiv: 2101.00234*.

Sachan, D. S., Neubig, G. (2018) Parameter sharing methods for multilingual self-attentional models. Preprint *arXiv: 1809.00252*.

Sakhavi, S., Guan, C., Yan, S. (2018) Learning temporal information for brain-computer interface using convolutional neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11) 5619-5629.

Salakhutdinov, R., Murray, I. (2008) On the quantitative analysis of deep belief networks. *Proceedings of the 25th International Conference on Machine Learning*, 872-879.

Song, Y., Jia, X., Yang, L., Xie, L. (2021) Transformer-based spatial-temporal feature learning for EEG decoding. Preprint *arXiv: 2106.11170*.

Sutton, R. S., Barto, A. G. (2018) Reinforcement learning: an introduction. MIT Press, Cambridge, MA.

Tay, Y., Dehghani, M., Bahri, D., Metzler, D. (2022) Efficient Transformers: a survey. Preprint *arXiv:2009.06732*.

Tucker, L. R. (1966) Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3): 279-311.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. (2017) Attention is all you need. *31st Conference on Neural Information Processing Systems*.

Voita, E., Talbot, D., Moiseev, F., Sennrich, R., Titov, I. (2019) Analyzing multi-head self-attention: specialized heads do the heavy lifting, the rest can be pruned. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 5797-5808.

Wang, D., Wu, B., Zhao, G., Yao, M., Chen, H., Deng, L., Yan, T., Li, G. (2021b) Kronecker CP decomposition with fast multiplication for compressing RNNs. *IEEE Transactions on Neural Networks and Learning Systems*, DOI: 10.1109/TNNLS.2021.3105961

Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., Zhou, M. (2020) MiniLM: deep self-attention distillation for task-agnostic compression of pre-trained Transformers. Preprint *arXiv: 2002.10957*.

Wang, Y., Guo, W., Yue, X. (2021a) Tensor decomposition to compress convolutional layers in deep learning. *IISE Transactions*, DOI: https://doi.org/10.1080/24725854.2021.1894514
Wu, H., Xu, J., Wang, J., Long, M. (2021) Autoformer: decomposition transformers with auto-correlation for long-term series forecasting. *35th Conference on Neural Information Processing Systems.*
Wu, N., Green, B., Ben, X., Banion, S. (2020) Deep Transformer models for time series forecasting: the influenza prevalence case. Preprint *arXiv:2001.08317*.
Xu, C., McAuley, J. (2022) A survey on model compression for natural language processing. Preprint *arXiv: 2202. 07105*.
Yang, L., Song, Y., Ma, K., Xie, L. (2021) Motor imagery EEG decoding method based on a discriminative feature learning strategy. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 29, 368-379.
Zhang, R., Zong, Q., Dou, L., Zhao, X. (2019) A novel hybrid deep learning scheme for four-class motor imagery classification. *Journal of Neural Engineering*, 16(6) 066004.
Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., Zhang, W. (2021) Informer: beyond efficient transformer for long sequence time-series forecasting. *The Thirty-Fifth AAAI Conference on Artificial Intelligence*, 11106-11115.

Appendix A1

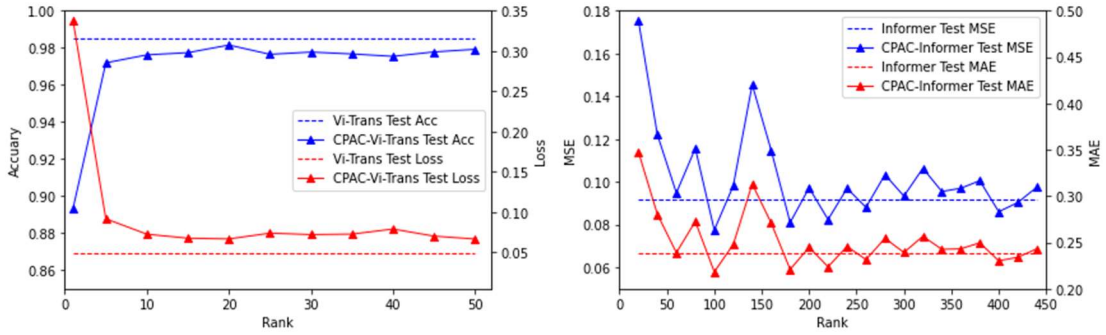


Figure A1. Performance comparison between Transformer and CPAC-Transformer: the left is on MNIST dataset and the right is on ETT dataset.

Appendix A2

CP-decomposition

The CP-decomposition decomposes a high-order tensor into multiple groups of rank-one tensors (vectors), which is demonstrated in Fig. A2 using the three-way tensor as an example. The expression of CP-decomposition on a three-way tensor is given in Eq. (1), where the $a_i \in \mathbb{R}^I, b_i \in \mathbb{R}^J, c_i \in \mathbb{R}^K, i = 1, \dots, R$ are decomposed rank-one tensors, and operator $\circ$ represents the outer product:

$$\mathcal{X} \approx \sum_{i=1}^{R} a_i \circ b_i \circ c_i \tag{A1}$$

For a general N^{th}-order tensor, $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, the expression of the CP decomposition can be further extended into Eq. (A2), where λ_r represents the coefficient corresponding to each group of rank-one tensors:

$$\mathcal{X} \approx \sum_{i=1}^{R} \lambda_r a_i^{(1)} \circ a_i^{(2)} \circ \cdots \circ a_i^{(N)} \tag{A2}$$

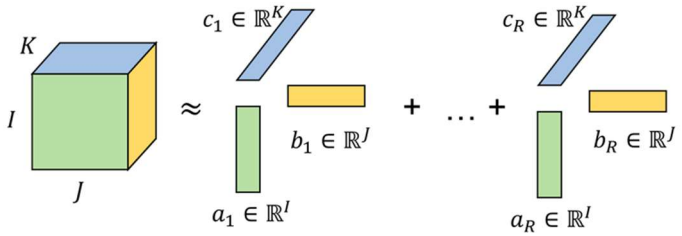


Figure A2. Demonstration of CP-decomposition on a three-way tensor.

Matrix calculus

The expressions of forward and backward propagations of our proposed method are derived from matrix calculus. The column stacking vectorization of a matrix $X \in \mathbb{R}^{m\times n}$ is given in Eq. (A3).

$$\mathrm{vec}(X) = [X_{11}, \dots, X_{m1}, X_{12}, \dots, X_{m2}, \dots, X_{1n}, \dots, X_{mn}]^{\mathrm{T}} \in \mathbb{R}^{mn\times 1} \tag{A3}$$

The vectorization of matrix product can be simplified by the following lemma.

Lemma 3.1 (Magnus and Neudecker 1985) *For any three matrices A, X, B such that the matrix product AXB is defined, the vectorization of AXB can be expressed as:*

$$\mathrm{vec}(AXB) = (B^T \otimes A)\mathrm{vec}(X). \tag{A4}$$

The matrix derivative is further defined as:

Definition 3.2 (Magnus and Neudecker 1985) *Let F be a differentiable $p \times q$ real matrix function of an $m \times n$ matrix of a real variable X. The derivative of F w.r.t. X is the $mn \times pq$ matrix:*

$$\frac{\partial F}{\partial X} = \frac{\partial \mathrm{vec}(F)}{\partial \mathrm{vec}(X)}. \tag{A5}$$

The relation between differential and derivative of the vectorized matrix is further extended in Eq. (A6) (Magnus and Neudecker 1985).

$$\mathrm{vec}(dF) = \frac{\partial F^T}{\partial X}\mathrm{vec}(dX) \tag{A6}$$

Attention

The main idea of the attention mechanism is to guide the model to focus on the most relevant information in the input when generating a certain output (Vaswani et al. 2017). It is formulated as mapping the query and a set of key-value pairs to the output, in which the similarity between the query and key will determine which part of the value will be focused. In practice, we calculate a set of attention functions at the same time and formulate all sets of query, key, and value into matrices Q, K, and V, respectively. As shown in Fig. A3, the inner product for the scaled dot-product attention will be conducted between query and key, and the results are used as the weights of values. The expressions of the scaled dot-product attention are summarized in Eqs. (A7-A10).

$$Q = IW^Q, \tag{A7}$$
$$K = IW^K, \tag{A8}$$
$$V = IW^V, \tag{A9}$$

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{A10}$$

where d_k is the dimension of the matrix Q and K.

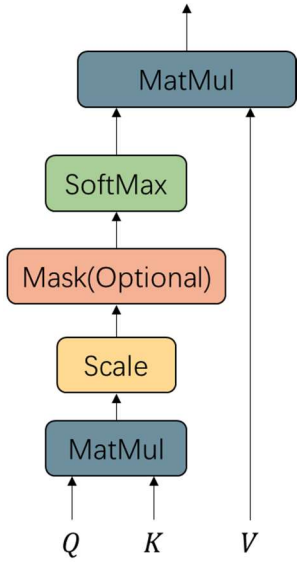


Figure A3.Scaled dot-product attention

In the Multi-Head Attention, the attention is calculated on h separated heads, as demonstrated in Fig. A4. For each head, the query, key, and value are firstly projected into lower-dimensional feature spaces with dimensions d_k, d_q, d_v using different weight matrices. Then, the attention is calculated separately on each head. Finally, all the outputs are concatenated and fed through a linear layer. The expressions of Multi-Head Attention are given in Eq. (A11).

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,\ldots,\text{head}_h)W^O,$$

where

$$\text{head}_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right), i = 1,2,\ldots,h \tag{A11}$$

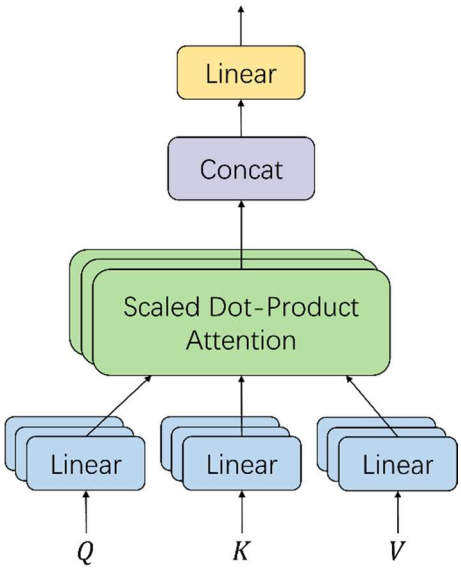


Figure A4：Multi-Head Attention

Transformer

The Transformer is based on the idea that the outputs can be calculated only based on the attention mechanism and consists of an encoder and a decoder (Vaswani et al. 2017). The encoder is a stack of N same modules (red box in Fig. A5), each of which consists of two layers. The first layer is the Multi-Head Attention layer, and the second layer is the feed-forward layer. The input of each layer is concatenated with its output using the skip connection, and then layer normalization is conducted. The final output after normalization can be expressed as $\mathrm{Norm}(x + f(x))$, where $f(x)$ is the output of the previous layer (either Multi-Head Attention or feed-forward) with x as the input. The decoder is also a stack of N same modules (blue box in Fig. A3). In addition to the same two layers used in the encoder, another Multi-Head Attention is introduced (highlighted in the blue dashed box in Fig. A5), which takes the output of the encoder as the key-value pairs and the output of the previous Multi-Head Attention as the query. Similar to the encoder, the input of each layer in the decoder is also concatenated with its output using the skip connection, and layer normalization is further conducted. In addition, the first Multi-Head Attention (highlighted in the green dashed box in Fig. A5) of the decoder is modified to mask (hide) the future information in the sequential prediction task. Such that when predicting the value of the i^{th} position, only the information in previous steps is preserved.

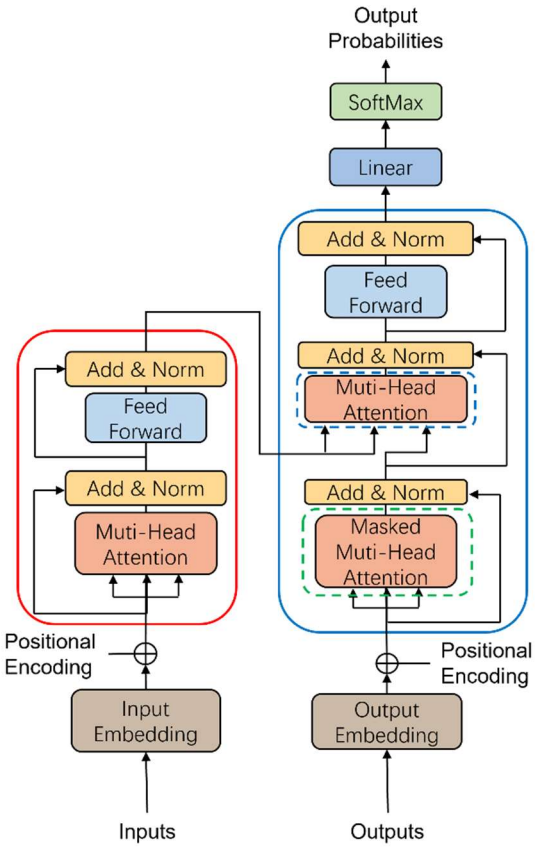


Figure A5. The architecture of Transformer (Vaswani et al. 2017)

Appendix A3

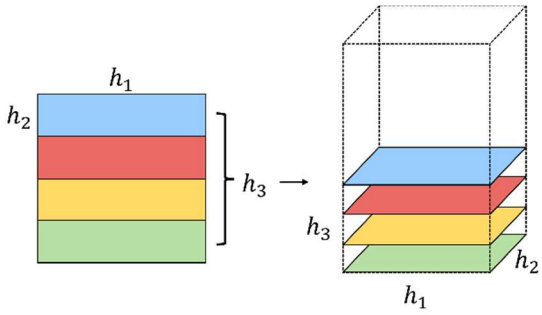


Figure A6. Visualization of reformulating a matrix into a tensor.

Appendix A4

Layer importance

The layer importance is defined to evaluate the influence of each layer on the model performance. By selecting the layer with the minimum importance score to conduct decomposition, we can ensure a smoother training process by avoiding significant fluctuation in model performance.

Suppose w_m denotes one single parameter, we have the set $\theta = \{w_0, w_1, \dots w_M\}$ that denotes all parameters in the neural network. Given the training dataset $\mathcal{D} = \{(x_0, y_0), (x_1, y_1), \dots, (x_K, y_K)\}$, the task of training is to generate the value of θ to receive the minimum value of loss function $E(\mathcal{D}|\theta)$.

The importance of a specific parameter can be quantified by the change of loss function value induced by setting its value as zero (Molchanov et al. 2019). Under the independent identically distributed (i.i.d.) assumption, this induced change can be measured as the squared difference of loss function values with and without the parameter w_m, which is given in Eq. (A12):

$$\mathcal{J}(w_m) = \left(E(\mathcal{D},\theta) - E(\mathcal{D},\theta|w_m = 0)\right)^2 \tag{A12}$$

In order to simplify the calculation, the first-order Taylor expansion is used to approximate $\mathcal{I}_m$, as shown in Eq. (A13):

$$\mathcal{J}^{(1)}(w_m) = \left(g_{w_m} w_m\right)^2, \tag{A13}$$

where g_{w_m} is the gradient of parameter w_m, which is already available from backward propagation. Then, the summation of the importance of every single parameter is used to approximate the joint importance of a structural set of parameters W_i (e.g., all parameters in the fully connected layer), which is given in Eq. (A14).

$$\mathcal{J}^{(1)}(W_i) \triangleq \sum_{w \in W_i} \mathcal{J}^{(1)}(w) = \sum_{w \in W_i} (g_w w)^2 \tag{A14}$$

In our method, the layer importance of each Multi-Head Attention layer is calculated to select the least significant layer to implement the decomposition.