Fastformer: Additive Attention Can Be All You Need

Chuhan Wu† Fangzhao Wu† Tao Qi† Yongfeng Huang† Xing Xie‡
†Department of Electronic Engineering & BNRist, Tsinghua University, Beijing 100084, China
‡Microsoft Research Asia, Beijing 100080, China
{wuchuhan15, wufangzhao, taoqi.qt}@gmail.com
yfhuang@tsinghua.edu.cn, xingx@microsoft.com

Abstract

Transformer is a powerful model for text understanding. However, it is inefficient due to its quadratic complexity to input sequence length. Although there are many methods on Transformer acceleration, they are still either inefficient on long sequences or not effective enough. In this paper, we propose Fastformer, which is an efficient Transformer model based on additive attention. In Fastformer, instead of modeling the pair-wise interactions between tokens, we first use additive attention mechanism to model global contexts, and then further transform each token representation based on its interaction with global context representations. In this way, Fastformer can achieve effective context modeling with linear complexity. Extensive experiments on five datasets show that Fastformer is much more efficient than many existing Transformer models and can meanwhile achieve comparable or even better long text modeling performance.

1 Introduction

Transformer (Vaswani et al., 2017) and their variants have achieved great success in many fields. For example, Transformer is the backbone architecture of many state-of-the-art pre-trained language models in NLP, such as BERT (Devlin et al., 2019) and GPT (Radford et al., 2019). Transformer also shows great promises in vision-related tasks (Dosovitskiy et al., 2021). The core of a Transformer model is self-attention mechanism, which allows the Transformer to model the contexts within an input sequence (Parikh et al., 2016). However, since self-attention computes the dot-product between the input representations at each pair of positions, its complexity is quadratic to the input sequence length (Vaswani et al., 2017). Thus, it is difficult for standard Transformer models to efficiently handle long input sequences (Tay et al., 2020).

There are many methods to accelerate the Transformer model (Beltagy et al., 2020; Zaheer et al., 2020; Wang et al., 2020b; Kitaev et al., 2020; Tay et al., 2021). For example, BigBird (Zaheer et al., 2020) computes sparse attention instead of a dense one. It uses a combination of local attention, global attention at certain positions and random attention between a certain number of tokens. However, sparse attention usually cannot fully model the global context (Wu et al., 2021b). Linformer (Wang et al., 2020b) exploits the low-rank characteristic of the self-attention matrix by computing approximated ones. It projects attention key and value into low-dimensional matrices that are independent of the sequence length. However, the approximation is in fact context-agnostic, which may weaken the context modeling ability of Transformer. In addition, these methods are not efficient enough when the input sequence length is very long.

In this paper we propose Fastformer†, which is an efficient Transformer variant based on additive attention that can achieve effective context modeling in linear complexity. In Fastformer, we first use additive attention mechanism to model global contexts, and then further transform each token representation based on its interaction with global context representations. It projects attention key and value into low-dimensional matrices that are independent of the sequence length. However, the approximation is in fact context-agnostic, which may weaken the context modeling ability of Transformer. In addition, these methods are not efficient enough when the input sequence length is very long.

In this paper we propose Fastformer†, which is an efficient Transformer variant based on additive attention that can achieve effective context modeling in linear complexity. In Fastformer, we first use additive attention mechanism to summarize the input attention query matrix into a global query vector. Next, we model the interaction between attention key and the global query vector via element-wise product to learn global context-aware key matrix, and we further summarize it into a global key vector via additive attention. Then we use element-wise product to aggregate the global key and attention value, which are further processed by a linear transformation to compute the global context-aware attention value. Finally, we add together the original attention query and the global context-aware attention value to form the final output. We conduct extensive experiments on five benchmark datasets in various tasks, including

†A pytorch version of Fastformer using the huggingface style is available at https://github.com/wuch15/Fastformer.
sentiment classification, topic prediction, news recommendation and text summarization. The results demonstrate that Fastformer is much more efficient than many Transformer models and can achieve quite competitive results in long text modeling.

The contributions of this paper are summarized as follows:

• We propose an additive attention based Transformer named Fastformer. To the best of our knowledge, Fastformer is the most efficient Transformer architecture.

• We propose to model the interaction between global contexts and token representations via element-wise product, which can help fully model context information in an efficient way.

• Extensive experiments on five datasets show that Fastformer is much more efficient than many Transformer models and can achieve competitive performance.

2 Related Work

2.1 Transformer and Self-Attention

The Transformer model is built upon multi-head self-attention, which can effectively model the contexts within a sequence by capturing the interactions between the inputs at each pair of positions (Vaswani et al., 2017). An $h$-head self-attention mechanism can be formulated as follows:

$$\begin{align*}
\text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O, \\
\text{head}_i &= \text{Attention}(QW^Q_i, KW^K_i, VW^V_i) = \text{softmax}\left(\frac{QW^Q_i(KW^K_i)^T}{\sqrt{d}}\right)VW^V_i,
\end{align*}$$

where $Q, K, V \in \mathbb{R}^{N \times d}$ are the input query, key and value matrices, $N$ is the sequence length, $d$ is the hidden dimension in each attention head, and $W^O \in \mathbb{R}^{hd \times d}$ is a linear transformation parameter matrix. The representation learned by each attention head is formulated as follows:

$$\begin{align*}
\text{Attention}(QW^Q_i, KW^K_i, VW^V_i) &= \text{softmax}\left(\frac{QW^Q_i(KW^K_i)^T}{\sqrt{d}}\right)VW^V_i,
\end{align*}$$

where $W^Q_i, W^K_i, W^V_i \in \mathbb{R}^{d \times d}$ are learnable parameters. From this formula, we can see that the computational complexity is quadratic to the sequence length $N$. It has become a bottleneck for Transformers to handle long sequences.

2.2 Efficient Transformer

In recent years, there are many approaches to improving the efficiency of Transformer architecture (Tay et al., 2020). Some methods use sparse attention mechanisms to reduce the complexity of self-attention (Child et al., 2019; Beltagy et al., 2020; Zaheer et al., 2020; Zhang et al., 2021). For example, Longformer (Beltagy et al., 2020) uses sliding window attention to attend local contexts and uses global attention on a few pre-selected input locations to capture global contexts. BigBird (Zaheer et al., 2020) combines local attention, global attention at certain positions and random attention on several randomly selected token pairs. However, these methods may need a relative larger number of attended tokens to reduce the performance degradation on longer sequences, which usually leads to a limited speed-up ratio.

Another way is using hashing technique to accelerate self-attention computation. For example, Reformer (Kitaev et al., 2020) uses a multi-round hashing scheme to put similar representations into same buckets when computing self-attention, which can theoretically reduce the complexity to $O(N \log(N))$. However, the computational complexity constant of Reformer is quite large, making it inefficient in processing common sequences with rather limited lengths.

There are also several methods that aim to reduce the computational cost by computing approximated self-attention (Choromanski et al., 2020; Wang et al., 2020; Wang et al., 2020b; Tay et al., 2021). For instance, Linformer (Wang et al., 2020b) projects the attention key and value into low-rank matrices to approximate the self-attention mechanism. Linear Transformer (Katharopoulos et al., 2020) uses a kernel-based formulation of self-attention and the associative property of matrix multiplication to approximate the dot-product attention. However, these methods approximate self-attention in a context-agnostic manner, which may not be optimal for text modeling. In addition, they still bring heavy computational cost when the sequence length is very long. Different from the aforementioned methods, Fastformer uses additive attention to model global contexts and uses element-wise product to model the interaction between each input representation and global contexts, which can greatly reduce the computational cost and meanwhile effectively capture contextual information.
3 Fastformer

In this section, we introduce our Fastformer approach based on additive attention. The architecture of Fastformer is shown in Fig. 1. It first uses additive attention mechanism to summarize the query sequence into a global query vector, next models the interaction between the global query vector and attention keys with element-wise product and summarize keys into a global key vector via additive attention, then models the interactions between global key and attention values via element-wise product and uses a linear transformation to learn global context-aware attention values, and finally adds them with the attention query to form the final output. In this way, the computational complexity can be reduced to linearity, and the contextual information in the input sequence can be effectively captured. Next, we introduce the details of Fastformer in the following section.

3.1 Architecture

The Fastformer model first transforms the input embedding matrix into the query, key and value sequences. The input matrix is denoted as \( E \in \mathbb{R}^{N \times d} \), where \( N \) is the sequence length and \( d \) is the hidden dimension. Its subordinate vectors are denoted as \( [e_1, e_2, \ldots, e_N] \). Following the standard Transformer, in each attention head\(^2\) we use three independent linear transformation layer to transform the input into the attention query, key and value matrices \( Q, K, V \in \mathbb{R}^{N \times d} \), which are written as \( Q = [q_1, q_2, \ldots, q_N] \), \( K = [k_1, k_2, \ldots, k_N] \) and \( V = [v_1, v_2, \ldots, v_N] \), respectively.

Next, modeling the contextual information of the input sequence based on the interactions among attention query, key and value is a critical problem for Transformer-like architectures. In the vanilla Transformer, dot-product attention mechanism is used to fully model the interaction between query and key. Unfortunately, its quadratic complexity makes it inefficient in long sequence modeling. A potential way to reduce the computational complexity is to summarize the attention matrices (e.g., query) before modeling their interactions. Additive attention is a form of attention mechanism that can efficiently summarize important information.

\(^2\)Different attention heads use the same formulation but different parameters.

Figure 1: The architecture of Fastformer.
within a sequence in linear complexity. Thus, we first use additive attention to summarize the query matrix into a global query vector $q \in \mathbb{R}^d$, which condenses the global contextual information in the attention query. More specifically, the attention weight $\alpha_i$ of the $i$-th query vector is computed as follows:

$$\alpha_i = \frac{\exp(w_q^T q_i / \sqrt{d})}{\sum_{j=1}^N \exp(w_q^T q_j / \sqrt{d})}, \quad (3)$$

where $w_q \in \mathbb{R}^d$ is a learnable parameter vector. The global attention query vector is computed as follows:

$$q = \sum_{i=1}^N \alpha_i q_i. \quad (4)$$

Then, a core problem in Fastformer is how to model the interaction between the summarized global query vector and the key matrix. There are several intuitive options, such as adding or concatenating the global query to each vector in the key matrix. However, they cannot differ the influence of the global query on different keys, which is not beneficial for context understanding. Element-wise product is an effective operation to model the nonlinear relations between two vectors (Wang et al., 2017). Thus, we use the element-wise product between the global query vector and each key vector to model their interactions and combine them into a global context-aware key matrix. We denote the $i$-th vector in this matrix as $p_i$, which is formulated as $p_i = q \ast k_i$ (the symbol $\ast$ means element-wise product). In a similar way, we use additive attention mechanism to summarize the global context-aware key matrix due to efficiency reasons. The additive attention weight of its $i$-th vector is computed as follows:

$$\beta_i = \frac{\exp(w_k^T p_i / \sqrt{d})}{\sum_{j=1}^N \exp(w_k^T p_j / \sqrt{d})}, \quad (5)$$

where $w_k \in \mathbb{R}^d$ is the attention parameter vector. The global key vector $k \in \mathbb{R}^d$ is further computed as follows:

$$k = \sum_{i=1}^N \beta_i p_i. \quad (6)$$

Finally, we model the interaction between attention value matrix and the global key vector for better context modeling. Similar with the query-key interaction modeling, we also perform element-wise product between the global key and each value vector to compute a key-value interaction vector $u_i$, which is formulated as $u_i = k \ast v_i$. Motivated by the vanilla Transformer, we apply a linear transformation layer to each key-value interaction vector to learn its hidden representation. The output matrix from this layer is denoted as $R = [r_1, r_2, \ldots, r_N] \in \mathbb{R}^{N \times d}$. This matrix is further added together with the query matrix to form the final output of Fastformer. Note that the output matrix from each attention head is concatenated along the hidden dimension axis.

We can see that in Fastformer, each key and value vector can interact with the global query or key vector to learn contextualized representations. By stacking multiple Fastformer layers, we can fully model contextual information. Motivated by the weight sharing techniques used in (Wang et al., 2020b), we share the value and query transformation parameters to reduce the memory cost. In addition, we share the parameters across different Fastformer layers to further reduce the parameter size and mitigate the risk of overfitting.

### 3.2 Complexity Analysis

In this section, we analyze the computational complexity of Fastformer. For the additive attention networks to learn global query and key vectors, their time and memory cost are both $O(N \cdot d)$, and their total number of additional parameters is $2hd$ ($h$ is the attention head number). In addition, the time cost and memory cost of element-wise product is also $O(N \cdot d)$, the total complexity is $O(N \cdot d)$, which is much more efficient than the standard Transformer with $O(N^2 \cdot d)$ complexity.\footnote{If the weight sharing technique is used, the total parameter size of Fastformer per layer is $3hd^2 + 2hd$. Compared with Transformer with at least $4hd^2$ parameters, Fastformer also uses fewer parameters. These analysis results demonstrate the theoretical efficiency of Fastformer.}

If the weight sharing technique is used, the total parameter size of Fastformer per layer is $3hd^2 + 2hd$. Compared with Transformer with at least $4hd^2$ parameters, Fastformer also uses fewer parameters. These analysis results demonstrate the theoretical efficiency of Fastformer.

### 4 Experiments

#### 4.1 Datasets and Experimental Settings

We conduct extensive experiments on five benchmark datasets for different tasks. Their details

\footnote{Note that the complexity of linear transformations is not taken into account by many prior works, and we also do not consider their effects on computational costs.\footnote{Including the query, key, value and output transformation matrices. The parameters in the bias term, feed-forward network and layer normalization are not counted.}}
| Dataset   | #Train | #Val | #Test | Avg. len. | #Class |
|-----------|--------|------|-------|-----------|--------|
| Amazon    | 40.0k  | 5.0k | 5.0k  | 133.4     | 5      |
| IMDB      | 108.5k | 13.6k| 13.6k | 385.7     | 10     |
| MIND      | 128.8k | 16.1k| 16.1k | 505.4     | 18     |

Table 1: Statistics of sentiment and news topic classification datasets.

|             | #News | #Users | #Train impression | #Val impression | #Test impression | Avg. click his. len. |
|-------------|-------|--------|-------------------|------------------|-------------------|----------------------|
|             | 161,013 | 1,000,000 | 2,232,748 | 376,471 | 2,370,727 | 37.1 |

Table 2: Statistics of MIND dataset for the news recommendation task.

| Dataset          | #Train | #Val | #Test | Avg. doc/summary len. |
|------------------|--------|------|-------|-----------------------|
| CNN/DailyMail    | 287.1k | 13.4k| 11.5k | 781/56                |
| PubMed           | 108.5k | 13.6k| 13.6k | 3016/203              |

Table 3: Statistics of the text summarization datasets.

are introduced as follows. The first one is Amazon (He and McAuley, 2016) (we use the Electronics domain), which is a widely used dataset for review rating prediction. The second one is IMDB (Diao et al., 2014). It is a benchmark dataset for movie review rating prediction. The third one is MIND (Wu et al., 2020), which is a large-scale English dataset for news recommendation and intelligence. We perform two tasks on this dataset, i.e., the news topic classification task based on news body and personalized news recommendation task based on the relevance between candidate news and user interests inferred from historical clicked news. The fourth one is CNN/DailyMail dataset (Hermann et al., 2015) (denoted as CNN/DM), which is a widely used benchmark dataset for text summarization. The fifth one is PubMed (Cohan et al., 2018), which is another benchmark text summarization dataset with much longer documents lengths. The detailed statistical information of the datasets introduced above are shown in Tables 1, 2 and 3.

In our experiments, we use Glove (Pennington et al., 2014) embeddings to initialize token embedding matrix. To obtain the embeddings in the classification and news recommendation tasks, we apply an additive attention network to convert the matrix output by Fastformer into an embedding. In addition, in the news recommendation task, following (Wu et al., 2019) we use Fastformer in a hierarchical way to first learn news embeddings from news titles and then learn user embeddings from the embeddings of historical clicked news. We use Adam (Bengio and LeCun, 2015) for model optimization. More detailed experimental settings are included in Appendix. We run our experiments on an Nvidia Tesla V100 GPU with 32GB memory. We repeat each experiment 5 times and report the average performance as well as the standard deviations. On the classification tasks, we use accuracy and macro-F scores as performance metrics. On the news recommendation task, following (Wu et al., 2020) we use AUC, MRR, nDCG@5 and nDCG@10 as the metrics. On the text summarization tasks, we use the ROUGE-1, ROUGE-2 and ROUGE-L metrics (denoted as R-1, R-2 and R-L) to evaluate the generated summaries.

4.2 Effectiveness Comparison

First, we compare the performance of Fastformer with many baseline methods, including: (1) Transformer (Vaswani et al., 2017), the vanilla Transformer; (2) Longformer (Beltagy et al., 2020), a Transformer variant with sparse attention. It combines sliding window attention and global attention to model local and global contexts; (3) BigBird (Zaheer et al., 2020), an extension of Longformer, which incorporates sparse random attention mechanism; (4) Linformer (Wang et al., 2020b), a Transformer variant with linear complexity, which use low-dimensional key and value matrices to compute approximated self-attention; (5) Pooling-former (Zhang et al., 2021), a hierarchical architecture that first uses sliding window self-attention to capture short-range contexts and then uses pooling self-attention to capture long-range contexts.

The performance of these methods on the three classification datasets are compared in Table 4. From the results, we find that efficient Transformer variants usually outperform the standard Transformer model. This is because the quadratic computational cost of vanilla Transformer limits the maximum sequence length can be handled, and many useful contexts are lost when truncating the input text sequence. Fastformer can achieve competitive or better performance than other efficient Transformer variants in both long and short text modeling. This is because Fastformer can effectively model global contexts and their relationship...
to different tokens, which can help understand context information accurately.

We also compare the performance of different methods in the news recommendation task. We add three recent news recommendation methods to the comparison, including: (1) NRMS (Wu et al., 2019), which uses multi-head self-attention networks to learn news and user representations; (2) FIM (Wang et al., 2020a), a fine-grained interest matching method for personalized news recommendation; (3) PLM-NR (Wu et al., 2021a), empowering news recommendation with pre-trained language models. In PLM-NR, we use the best performed UniLM (Bao et al., 2020) empowered model. In addition, we explore to replace the user encoder in PLM-NR with Fastformer. The results are shown in Table 5. We can see that among different Transformer architectures, Fastformer achieves the best performance, and it also outperforms its basic NRMS model. In addition, Fastformer can further improve the performance of PLM-NR, and the ensemble model achieves the best results on the MIND leaderboard. These results show that Fastformer is not only effective in text modeling, but also effective in understanding user interest.

We further conduct experiments on the text summarization tasks to verify the effectiveness of Fastformer in natural language generation. The results are shown in Table 6. We find that on the CNN/DM dataset, many efficient Transformer variants (except Poolingformer and Fastformer) are inferior to the vanilla Transformer. This is because sparse attention based method such as Longformer and BigBird cannot fully model the document contexts, and approximated self-attention based methods such as Linformer and Linear Transformer cannot effectively consider context information in the approximation. Since the summaries in the CNN/DM dataset have short lengths, they may be less effective than vanilla Transformer when the same sequence length is used. Fastformer can achieve the best performance in most metrics, which shows the advantage of Fastformer in natural language generation.

### Table 4: The results of different methods in the sentiment and topic classification tasks. Best average scores are highlighted.

| Methods     | Amazon/AUC | Amazon/Macro-F | IMDB/AUC | IMDB/Macro-F | MIND/AUC | MIND/Macro-F |
|-------------|-------------|----------------|----------|--------------|----------|--------------|
| Transformer | 65.32±0.35  | 42.31±0.33     | 52.04±0.50 | 42.69±0.47   | 80.90±0.20 | 60.02±0.21   |
| Longformer  | 65.45±0.39  | 42.48±0.44     | 52.21±0.36 | 43.36±0.38   | 81.36±0.21 | 62.59±0.23   |
| BigBird     | 66.14±0.42  | 42.96±0.40     | 53.23±0.46 | 44.03±0.44   | 81.93±0.24 | 63.58±0.26   |
| Linformer   | 66.20±0.49  | 43.13±0.48     | 53.17±0.59 | 44.34±0.57   | 82.16±0.28 | 63.77±0.30   |
| Linear Transformers | 66.12±0.42   | 43.04±0.44 | 53.09±0.47 | 44.30±0.49   | 82.25±0.23 | 63.81±0.22   |
| Poolingformer | 66.05±0.44   | 43.00±0.45 | 53.78±0.51 | 44.52±0.50   | 82.46±0.24 | 64.10±0.26   |
| Fastformer  | 66.13±0.29  | **43.23±0.30** | **54.10±0.42** | **44.65±0.44** | 82.34±0.19 | 63.89±0.20   |

### Table 5: Performance of different methods in the news recommendation task. *Ensemble of five independently trained models, which is the 1st ranked result on the MIND leaderboard.

| Methods     | AUC | MRR | nDCG@5 | nDCG@10 |
|-------------|-----|-----|--------|---------|
| NRMS        | 68.18 | 33.29 | 36.31 | 42.20   |
| FIM         | 68.31 | 33.42 | 36.47 | 42.35   |
| PLM-NR      | 70.64 | 35.39 | 38.71 | 44.38   |
| Transformer | 68.22 | 33.32 | 36.35 | 42.23   |
| Longformer  | 67.98 | 33.04 | 36.18 | 42.06   |
| BigBird     | 68.14 | 33.28 | 36.30 | 42.18   |
| Linformer   | 68.02 | 33.19 | 36.22 | 42.10   |
| Linear Transformers | 67.76                       | 32.94 | 36.16 | 41.97   |
| Poolingformer | 68.54                       | 33.60 | 36.69 | 42.60   |
| Fastformer  | 69.11 | 34.25 | 37.26 | 43.38   |
| Fastformer+PLM-NR | 71.04                   | 35.91 | 39.16 | 45.03   |
| Fastformer+PLM-NR* | 72.68                   | 37.45 | 41.51 | 46.84   |

### Table 6: Performance of different methods in the text summarization task. Best scores are highlighted.

| Methods     | CNN/DailyMail | PubMed |
|-------------|---------------|--------|
|              | R-1  | R-2  | R-L  | R-1  | R-2  | R-L  |
| Transformer  | 38.52 | 16.04 | 35.87 | 34.26 | 11.88 | 31.64 |
| Longformer   | 37.89 | 15.46 | 35.19 | 36.92 | 14.34 | 33.75 |
| BigBird      | 38.31 | 15.78 | 35.60 | 37.73 | 14.99 | 34.51 |
| Linformer    | 37.96 | 15.58 | 35.34 | 37.22 | 14.48 | 34.02 |
| Linear Transformer | 37.24 | 14.87 | 34.64 | 36.43 | 13.80 | 33.21 |
| Poolingformer | 38.58 | 16.16 | 36.17 | 37.82 | 15.15 | 34.63 |
| Fastformer   | 38.34 | 16.22 | 36.21 | 38.69 | 15.44 | 34.81 |

https://msnews.github.io/
Table 7: Asymptotic computational complexity of different methods. $N$ is the sequence length, $K$ is the average number of tokens to be attended by per token, $d$ is the hidden dimension, $\epsilon$ is the attention matrix approximation error in Linformer, and $w$ is the window size in Poolingformer.

| Method      | Complexity      |
|-------------|-----------------|
| Transformer | $O(N^2 \cdot d)$ |
| Longformer  | $O(N \cdot K \cdot d)$ |
| BigBird     | $O(N \cdot K \cdot d)$ |
| Linformer   | $O(N \cdot d/\epsilon^2)$ |
| Linear Transformer | $O(N \cdot d^2)$ |
| Poolingformer | $O(N \cdot d \cdot w)$ |
| Fastformer  | $O(N \cdot d)$ |

The complexity of vanilla Transformer is $O(N^2 \cdot d)$, while other compared methods have linear complexity with respect to the input sequence length. However, the complexity of Longformer and BigBird depends on the average number of tokens to be attended by each token, Linformer has a complexity depending on the square of the approximation error, the complexity of Linear Transformer has a quadratic term of hidden dimension, and the complexity of Poolingformer depends on its window size. Different from them, the complexity of Fastformer only depends on the sequence length and the hidden dimension, and it has the least complexity among compared methods. This result shows that Fastformer is efficient in theory.

Then, we conduct experiments to measure the real training and inference cost of different methods. 

Motivated by prior works (Katharopoulos et al., 2020; Wang et al., 2020b), we vary the sequence length from 128 to 65535 and scale the batch size inversely with the sequence length. We generate pseudo samples with random tokens and fix the token embeddings to better measure the computational cost of different methods. The results are shown in Fig. 2. We find that Transformer is inefficient when the sequence length is relatively long (e.g., 512). In addition, we find that although Poolingformer has linear complexity in theory, it is inefficient in practice. This is because it uses a large window size (e.g., 256) to compute pooling weight in a convolution-like way, which leads to a very large constant term of the computational cost. Besides, Fastformer is much more efficient than other linear complexity Transformer variants in terms of both training and inference time. These results verify the efficiency of Fastformer.

4.4 Influence of Interaction Function

Next, we study the influence of using different functions to model the interactions among query, key and value in Fastformer. We compare the performance of Fastformer and its variants using the add or concatenation functions to combine the global query/key vector with the vectors in the key/value matrix. The results are shown in Fig. 3. We find concatenating is not a good option for Fastformer. This is because simply concatenating two vectors cannot consider the interactions between them. In addition, we find adding is not optimal. This is

---

9 We exclude the complexity of the linear Transformation applied after the input and before the output.

10 We use the model configurations in the news topic classification task.

11 The maximum sequence length of Transformer is limited by the GPU memory.
Figure 3: Influence of different combination functions.

because the add function can only model the linear interactions between two vectors, which may also be insufficient to learn accurate context representations. Different from concatenation and add, element-wise product can model the non-linear interactions between two variables, which may help model the complex contexts in long sequences.

4.5 Influence of Parameter Sharing

Then, we study the influence of different parameter sharing techniques on the performance of Fastformer, including sharing query and value transformation matrices, sharing the parameters across different attention heads, and sharing parameters across different layers.12 The results are shown in Fig. 4. We find that using query-value parameter sharing can achieve similar or slightly better performance than the Fastformer model without any parameter sharing techniques. Thus, it is favorable to reduce the parameter size by sharing the query and value transformation matrix. In addition, we find head-wise parameter sharing will lead to notable performance drops. This is because different attention heads are expected to capture different patterns of contexts, and sharing their parameters is not beneficial for context modeling. Moreover, we find incorporating layer-wise sharing method can further improve the model performance. This is because parameter sharing among different layers can mitigate the risk of overfitting (Lan et al., 2020).

Thus, in Fastformer we incorporate both query-value sharing and layer-wise sharing strategies to
improve the model performance and meanwhile reduce the model parameter size.

4.6 Applications of Fastformer

We further apply Fastformer to a downstream Ad CVR prediction task. We conduct experiments on a large-scale Ad CVR prediction data collected from Bing Ads, which contains user behaviors such as search query, webpage browsing and the conversion of clicked Ads. The task is a binary classification task by predicting whether a clicked Ad leads to a conversion. The dataset contains 52.2m training samples, 387k validation samples and 611k test samples (logs in the last week are for test, and the rest are for training and validation). Similar to news recommendation, we first use a Transformer or Fastformer to learn ad/behavior embedding, and then use another Transformer or Fastformer to learn user embedding from behavior embeddings. The prediction AUC, training memory utilization and local inference latency of Transformer and Fastformer based methods are shown in Table 8. The results show the effectiveness and efficiency of Fastformer in Ad CVR prediction.

| Method    | AUC      | Memory  | Latency |
|-----------|----------|---------|---------|
| Transformer | 0.7299   | 30GB    | 176ms   |
| Fastformer | 0.7394(+1.3%) | 25GB(-16.7%) | 163ms(-7.2%) |

Table 8: Accuracy, memory cost and inference speed comparison.

5 Conclusion and Future Work

In this paper, we propose Fastformer, which is a Transformer variant based on additive attention that can handle long sequences efficiently with linear complexity. In Fastformer, we first use additive attention to summarize the query matrix into a global query vector. Next, we combine it with each key vector via element-wise product to learn global context-aware key matrix, and we further summarize it into a global key vector via additive attention. We then model the interactions between global context-aware key and the value to learn global context-aware attention value, which is further combined with the query to form the final output. Extensive experiments on five benchmark datasets show that Fastformer is much more efficient than many existing Transformer models and meanwhile can achieve competitive or even better performance in long text modeling.

In our future work, we plan to pre-train Fastformer-based language models to better empower NLP tasks with long document modeling. In addition, we will explore applying Fastformer to other scenarios such as e-commerce recommendation and Ads CTR prediction to improve user modeling based on long user behavior sequences.

References

Hangbo Bao, Li Dong, Furu Wei, Wenhui Wang, Nan Yang, Xiaodong Liu, Yu Wang, Jianfeng Gao, Song-hao Piao, Ming Zhou, et al. 2020. Unilmv2: Pseudo-masked language models for unified language model pre-training. In ICML, pages 642–652. PMLR.

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. arXiv preprint arXiv:2004.05150.

Yoshua Bengio and Yann LeCun. 2015. Adam: A method for stochastic optimization. In ICLR.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. arXiv preprint arXiv:1904.10509.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, David Belanger, Lucy Colwell, et al. 2020. Masked language modeling for proteins via linearly scalable long-context transformers. arXiv preprint arXiv:2006.03555.

Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwon Kim, Walter Chang, and Nazli Goharian. 2018. A discourse-aware attention model for abstractive summarization of long documents. In NAACL-HLT, pages 615–621.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In NAACL-HLT, pages 4171–4186.

Qiming Diao, Minghui Qiu, Chao-Yuan Wu, Alexander J Smola, Jing Jiang, and Chong Wang. 2014. Jointly modeling aspects, ratings and sentiments for movie recommendation (jmars). In KDD, pages 193–202.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In ICLR.
Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In WWW, pages 507–517.

Karl Moritz Hermann, Tomas Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. NIPS, 28:1693–1701.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. In ICML, pages 5156–5165.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. In ICLR.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In ICLR.

Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. In EMNLP, pages 2249–2255.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. OpenAI blog, 1(8):9.

A Appendix

A.1 Experimental Environment
Our experiments are conducted on a cloud Linux server with Ubuntu 16.04 operating system. The codes are written in Python 3.6 using the Keras library 2.2.4 with Tensorflow 1.12 backend. The GPU type is Nvidia Tesla V100 with 32GB GPU memory. Each experiment is run by a single thread.

A.2 Preprocessing
In our experiments, we use the NLTK tool to preprocess the texts. We use the word_tokenize function and function to convert the input texts into token sequences. The word embeddings of out-of-vocabulary words are filled with random vectors that have the same mean and co-variation values as other words.

A.3 Hyperparameter Settings
The detailed hyperparameter settings on each dataset used in this paper are listed in Table 9.
| Method                      | Amazon | IMDB  | MIND (classification) | MIND (recommendation) | CNN/DailyMail | PubMed |
|-----------------------------|--------|-------|-----------------------|-----------------------|---------------|--------|
| # Encoder Layer             | 2      | 2     | 2                     | 1                     | 4             | 2      |
| # Decoder Layer             | -      | -     | -                     | -                     | -             | -      |
| # Global Token              | 8      | 8     | 16                    | 8                     | 64            | 64     |
| # Random Token              | 8      | 8     | 16                    | 8                     | 64            | 64     |
| Window size (Longformer, BigBird) | 8      | 8     | 16                    | 8                     | 64            | 64     |
| Window size (Poolingformer) | 64     | 64    | 256                   | 64                    | 256           | 256    |
| Block length (BigBird)      | 4      | 4     | 8                     | 4                     | 32            | 32     |
| Projection dimension (Linformer) | 16    | 16    | 16                    | 16                    | 64            | 64     |
| # Attention Head            | 16     | 16    | 16                    | 16                    | 16            | 16     |
| Beam Size                   | -      | -     | -                     | -                     | 5             | 5      |
| Maximum Text Length (Transformer) | 512  | 512   | 512                   | 512                   | 512           | 512    |
| Maximum Text Length (others) | 512   | 1,024 | 2,048                 | 48                    | 2,048         | 4,096  |
| Maximum User Behaviors      | -      | -     | -                     | 50                    | -             | -      |
| Hidden dimension            | 256    | 256   | 256                   | 256                   | 256           | 256    |
| Loss                        | Crossentropy | Crossentropy | Crossentropy | Crossentropy | Crossentropy | Crossentropy |
| Batch Size                  | 64     | 64    | 64                    | 64                    | 32            | 32     |
| Optimizer                   | Adam   | Adam  | Adam                  | Adam                  | Adam          | Adam   |
| Learning Rate               | 1e-3   | 1e-3  | 1e-3                  | 3e-6 for PLM-NR       | 1e-4          | 1e-4   |
| Max Epochs                  | 3      | 3     | 3                     | 12                    | 15            |        |
| Dropout                     | 0.2    | 0.2   | 0.2                   | 0.2                   | 0.2           | 0.2    |

Table 9: Detailed hyperparameter settings on each dataset.