Knowledge-Based Systems 240 (2022) 108075

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Mixhead: Breaking the low-rank bottleneck in multi-head attention language models

Zhong Zhang [a,b], Nian Shao [c], Chongming Gao [d], Rui Miao [e], Qinli Yang [a,b], Junming Shao [a,b,*]

[a] University of Electronic Science and Technology of China, Chengdu, 611731, PR China
[b] Yangtze Delta Region Institute (Huzhou) of UESTC, Huzhou, 313000, PR China
[c] The University of Edinburgh, Edinburgh, UK
[d] University of Science and Technology of China, Hefei, 230026, PR China
[e] Guizhou University, Guiyang, 550025, PR China

ARTICLE INFO

Article history:
Received 3 June 2021
Received in revised form 20 December 2021
Accepted 24 December 2021
Available online 5 January 2022

Keywords:
Language model
Multi-head attention
Low-rank bottleneck

ABSTRACT

The Transformer-based models have achieved significant advances in language modeling, while the multi-head attention mechanism in Transformers plays an indispensable part in their success. However, the too-small head size caused by the multi-head mechanism will lead to one problem called the *low-rank bottleneck*, which means that the rank of the attention weight matrix is too small to represent any desired attention. Naively increasing the head size is insufficient to solve the problem because it leads to severe parameter explosion and overfitting. To tackle this problem, we propose a mix-head attention (Mixhead) which mixes multiple attention heads by learnable mixing weights to improve the expressive power of the model. In contrast, Mixhead achieves a higher rank of the attention weight matrix while introducing a negligible number of parameters. Furthermore, Mixhead is quite general and can be easily adopted to most multi-head attention based models. We conduct extensive experiments including language modeling, machine translation, and finetuning BERT to demonstrate the effectiveness of our method.

1. Introduction

In recent years, the Transformer-based models have been rapidly emerging and achieved great success in a wide variety of natural language processing (NLP) tasks such as machine translation [1], language modeling [2], relation classification [3], text summarization [4], reading comprehension [5] and question answering [6]. Besides, the Transformer architecture also has been widely adopted to fields outside of NLP such as computer vision (CV) [7], recommender system [8], and graph mining [9]. Especially for CVs, there is a hot trend of replacing convolutional neural networks (CNNs) with Transformers on tasks like image classification [10], object detection [11] and segmentation [12]. Despite the success of Transformer, it is still a young network architecture that needs in-depth study.

The Transformer architecture is mainly built upon the multi-head attention module. Unlike the single-head attention, multi-head attention divides the full hidden space into multiple parallel subspaces for computing attention scores. It is found that the multi-head mechanism has a significant impact on improving the model performance [1]. However, there is a side effect of the multi-head mechanism that the head size becomes considerably small because a commonly adopted heuristic is to scale the head size by a ratio of 1/(number of heads) to control the parameter growth. Therefore, when the number of heads is too large, the head size will be decreased too small to persevere the expressive power of the model. We call it the *low-rank bottleneck* [13] problem.

In a nutshell, the low-rank bottleneck refers to the rank deficiency for the attention weight matrix when the head size is smaller than the sequence length, which hurts the expressive power of an individual head. Specifically, the $n \times n$ attention weight matrix is computed from the inner product of the $n \times d_q$ query matrix and the $n \times d_k$ key matrix, where n is the sequence length and $d_k = d_q$ is the head size. Suppose $d_k < n$, the rank of the inner product matrix is limited to at most d_k. Although it needs to go through a non-linear Softmax function to output the final desired attention weight matrix, as we will show in the later section, the low-rank bottleneck still exists. As natural language is complex in nature and sequence modeling is very context-dependent [14], we can reasonably argue that increasing the rank of the attention weight matrix is beneficial for the language modeling task. Prior work proposed the Fixhead [13] for

* Corresponding author at: University of Electronic Science and Technology of China, Chengdu, 611731, PR China.
E-mail addresses: zhongzhang@std.uestc.edu.cn (Z. Zhang), junmshao@uestc.edu.cn (J. Shao).

https://doi.org/10.1016/j.knosys.2021.108075
0950-7051/

this problem, which is to simply fix the head size to the length of the input sequence. However, such a setting suffers from severe parameter explosion and overfitting, which limits its use in actual language modeling tasks.

In this paper, we are going to tackle the low-rank bottleneck problem in multi-head attention models. We propose a new mix-head attention (Mixhead) which mixes multiple attention heads to improve the expressive power of the model. Our method has the following advantages.

1. Comparing with the vanilla multi-head attention and the previously proposed Fixhead method, our Mixhead successfully achieves a higher rank of the attention weight matrix and improves the expressive power of the model (See Fig. 1, Fig. 3).
2. Unlike the Fixhead, Mixhead only introduces a negligible number of parameters, which not only significantly alleviates the computational and the storage burden, but also reduces the risk of overfitting (See Table 10).
3. Our method is simple yet effective and quite general for most multi-head attention based models. People can easily adopt Mixhead to their models with a minimum effort of one-line-of-code change. Since Transformer models have been popularized in various fields, we believe that many studies could benefit from the Mixhead.

The rest of this paper is organized as follows. In Section 2, we briefly introduce preliminaries about the Transformer architecture and the low-rank bottleneck problem. In Section 3, we present the proposed Mixhead for the problem. Section 4 shows the experimental details and the results. Section 5 introduces related work. In Section 6, we draw our conclusion and discuss the future research direction.

2. Preliminaries

In this section, we introduce notations and give a basic overview of the Transformer architecture. Then we introduce the low-rank bottleneck in multi-head attention models to motivate the main idea of this paper.

2.1. Transformer architecture

A Transformer neural network is stacked by l layers of Transformer block, which consists of a multi-head attention layer followed by a position-wise feed-forward layer [1]. Both of the two sub-layers use residual connection [15] and layer normalization [16].

We focus on the multi-head attention layer for analysis and do not strictly distinguish attention and self-attention here. For a given input sequence $\boldsymbol{X} \in \mathbb{R}^{n\times d}$, where n is the sequence length and d is the embedding dimension, the output of a multi-head attention layer is given as follows.

$$\boldsymbol{Q}_i, \boldsymbol{K}_i, \boldsymbol{V}_i = \boldsymbol{X}\boldsymbol{W}_i^Q, \boldsymbol{X}\boldsymbol{W}_i^K, \boldsymbol{X}\boldsymbol{W}_i^V, \tag{1}$$

$$\text{head}_i = \text{Attention}(\boldsymbol{Q}_i, \boldsymbol{K}_i, \boldsymbol{V}_i) \tag{2}$$

$$= \text{Softmax}\Big[\frac{\boldsymbol{Q}_i\boldsymbol{K}_i^T}{\sqrt{d_k}}\Big]\boldsymbol{V}_i \tag{3}$$

$$= \boldsymbol{P}_i\boldsymbol{V}_i, \tag{4}$$

$$\text{MultiHead} = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\boldsymbol{W}^O, \tag{5}$$

where $\boldsymbol{W}_i^Q \in \mathbb{R}^{d\times d_q}$, $\boldsymbol{W}_i^K \in \mathbb{R}^{d\times d_k}$, $\boldsymbol{W}_i^V \in \mathbb{R}^{d\times d_v}$ and $\boldsymbol{W}^O \in \mathbb{R}^{hd_v\times d}$ are learnable projection matrices. For multi-head attention with h heads, the projection dimensions are set in a common heuristic manner as $d_q = d_k = d_v = d/h$ to control the parameter growth. $\boldsymbol{P}_i \in \mathbb{R}^{n\times n}$ is the attention weight matrix for the ith head. Concat$(\cdot)$ is the concatenation operation.

For the masked self-attention that is usually used in language modeling tasks, Eq. (3) becomes

$$\text{Softmax}\Big[\frac{\boldsymbol{Q}_i\boldsymbol{K}_i^T}{\sqrt{d_k}} \odot \boldsymbol{C}_i\Big]\boldsymbol{V}_i, \tag{6}$$

where $\odot$ denotes the Hadamard product, $\boldsymbol{C}_i$ is a lower triangular mask matrix whose lower triangle elements are all 1 and upper triangle elements are negative infinity. Since the masked self-attention is a special case of the self-attention, without loss of generality, we omit the mask in the following discussion.

2.2. Low-rank bottleneck in multi-head attention

For single-head attention, it is necessary for the projection dimension d to be no less than the sequence length n to be able to represent any desired attention weight matrix $\boldsymbol{P}$. This can be formulated as the following Representation Theorem [13].

Theorem 1 (*Representation Theorem*). *If $d_q = d_k = d \geq n$, then given any full row rank matrix $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ and any arbitrary $n \times n$ positive row stochastic matrix $\boldsymbol{P}$, there always exist $d \times d$ projection matrices $\boldsymbol{W}^Q$ and $\boldsymbol{W}^K$ such that*

$$\text{Softmax}\Big[\frac{(\boldsymbol{X}\boldsymbol{W}^Q)(\boldsymbol{X}\boldsymbol{W}^K)^T}{\sqrt{d_k}}\Big] = \boldsymbol{P}. \tag{7}$$

If $d_q = d_k = d < n$, there exist $\boldsymbol{X}$ and $\boldsymbol{P}$ such that Eq. (7) does not hold for all $\boldsymbol{W}^Q$ and $\boldsymbol{W}^K$.

In other words, if the projection dimension d is less than the sequence n, the produced attention weight matrix $\boldsymbol{P}$ could be low-rank and thus has limited expressive power. This is the so-called low-rank bottleneck. Interestingly, the Softmax is a non-linear function that should be able to output a matrix with arbitrary rank. However, empirical studies are against such an intuition, which suggests that the low-rank bottleneck still exists.

This problem is getting worse for the multi-head attention because the head size d_k is divided by the number of heads h, which is much smaller than the sequence length n. For example, a sequence with long length (e.g., $n = 512$) is usually preferred to capture rich contextual information and long-range dependency in language modeling [2]. However, by giving the number of attention heads $h = 8$ and projection dimension $d = 1024$, the head size d_k is only $1024/8 = 128$, which largely suffers from the low-rank bottleneck problem.

We empirically show that the low-rank bottleneck indeed exists in the multi-head attention models. We train a language model using a 16-layer Transformer [17] on the WikiText-103 dataset [18] and give a spectrum analysis of the attention weight matrix. In Fig. 1, we plot the normalized cumulative singular values of attention weight matrix for each layer with red curves. As expected, the curves are close to the upper left corner which means the corresponding matrices are low-rank. In contrast, the blue lines represent results from our method. It can be seen that our method results in more uniformly distributed singular values and successfully breaks the low-rank bottleneck.

3. Proposed method

In this section, we propose an alternative mix-head attention to replace the vanilla multi-head attention to tackle the low-rank bottleneck problem.

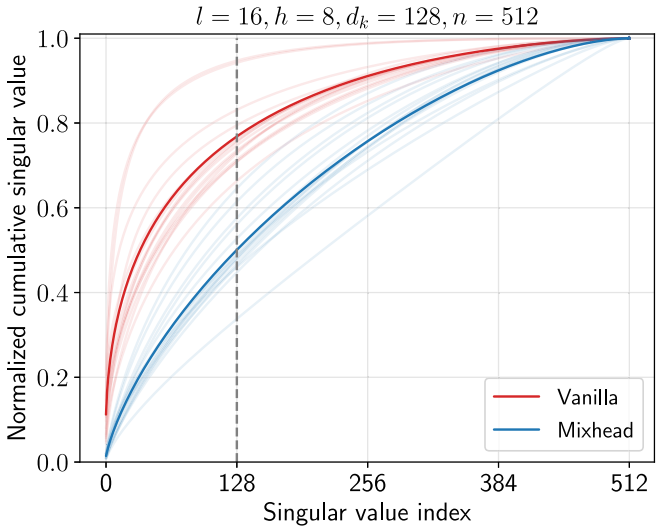


Fig. 1. The plot of the normalized cumulative singular values of the attention weight matrix per layer. Bold lines represent the average values. The closer the curve is to the upper left corner, the lower rank of the corresponding matrix.

3.1. Mix-head attention

For conciseness, we first reformulate the multi-head attention formula in a tensor manner as follows.

$$\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} = \boldsymbol{X} \cdot \boldsymbol{W}^Q, \boldsymbol{X} \cdot \boldsymbol{W}^K, \boldsymbol{X} \cdot \boldsymbol{W}^V, \tag{8}$$

$$\boldsymbol{P} = \text{Softmax}\Big[\frac{\boldsymbol{Q} \cdot \boldsymbol{K}'}{\sqrt{d_k}}\Big], \tag{9}$$

$$\boldsymbol{H} = \boldsymbol{P} \cdot \boldsymbol{V}, \tag{10}$$

$$\text{MultiHead} = \text{Flatten}(\boldsymbol{H})\boldsymbol{W}^O, \tag{11}$$

where $\boldsymbol{X} \in \mathbb{R}^{n \times d \times 1}$, $\boldsymbol{W}^Q \in \mathbb{R}^{d \times d_q \times h}$, $\boldsymbol{W}^K \in \mathbb{R}^{d \times d_k \times h}$, $\boldsymbol{W}^V \in \mathbb{R}^{d \times d_v \times h}$, $\boldsymbol{Q} \in \mathbb{R}^{n \times d_q \times h}$, $\boldsymbol{K}' \in \mathbb{R}^{d_k \times n \times h}$, $\boldsymbol{V} \in \mathbb{R}^{n \times d_v \times h}$, $\boldsymbol{P} \in \mathbb{R}^{n \times n \times h}$, $\boldsymbol{H} \in \mathbb{R}^{n \times d_v \times h}$ and $\boldsymbol{W}^O \in \mathbb{R}^{hd_v \times d}$. We use $\cdot$ to denote batch matrix product, where the last dimension is the batch (broadcast) dimension. Flatten$(\cdot)$ is the operation that flattens the last two dimensions. In addition, we will use the transpose symbol "T" to reverse all dimensions of a tensor in this paper.

Instead of directly setting the projection dimension d equal to the sequence length n [13] to achieve a high-rank attention weight matrix $\boldsymbol{P}_i$, we propose to mix multiple low-rank matrices to achieve the goal. The mixed attention is formulated as follows.

$$\bar{\boldsymbol{P}}^T = \boldsymbol{M}^T \cdot \boldsymbol{P}^T, \tag{12}$$

where $\boldsymbol{M}$ is the mixing weight matrix. We propose two forms of $\boldsymbol{M}$. The first is to let $\boldsymbol{M} \in \mathbb{R}^{1 \times h \times h}$ be a learnable mixing weight matrix, where the first dimension corresponds to the sequence position and is broadcasted when doing batch matrix product. We call it *position-independent mixing* since the mixing weights are shared across all positions of the input sequence. Each mixed head is a linear combination of other heads, i.e., $\bar{\boldsymbol{P}}_i = \sum_{j=1}^h m_{ji} \boldsymbol{P}_j$. This implies an inductive bias that the way of mixing attention heads is invariant to positions. The second is the *position-wise mixing*, which calculates exclusive mixing weights for each input token. The intuition is that different types of words might require different types of mixing of heads. For example, a noun token and a verb token can use different mixing weights as the attention heads are shown to specialize in different linguistic notions [19]. Similar to the position-wise feed-forward network in the Transformer [1], we model the position-wise mixing weights by a simple linear model. Let $\boldsymbol{W}^M \in \mathbb{R}^{1 \times d_q \times h}$ and $\boldsymbol{B} \in \mathbb{R}^{1 \times h \times h}$ be two learnable parameters, $\boldsymbol{Q}' \in \mathbb{R}^{n \times h \times d_q}$ be a transpose of the query matrix $\boldsymbol{Q} \in \mathbb{R}^{n \times d_q \times h}$ that has n input tokens. We compute the position-wise mixing weight matrix as $\boldsymbol{M}^T = \boldsymbol{W}^{M^T} \cdot \boldsymbol{Q}'^T + \boldsymbol{B}^T$, where $\boldsymbol{M} \in \mathbb{R}^{n \times h \times h}$ and its first dimension corresponds to the n positions of the input sequence. We can see that the position-wise mixing is a general form of the position-independent mixing by setting $\boldsymbol{W}^M = \boldsymbol{0}$. Therefore, we could expect that the position-wise mixing is more or at least equally expressive compared with the position-independent mixing.

3.2. Initialization strategy

We find that the initialization for the mixing weights has a significant impact on convergence and language modeling performance. Here we propose the initialization strategy that performs best in our experiments compared with other initialization strategies like random and uniform initialization. For the position-independent mixing, we initialize $\boldsymbol{M}$ to an identity matrix $\boldsymbol{I}$. In this way, the mix-head attention is identical to the vanilla multi-head attention at the beginning of training, i.e., $\bar{\boldsymbol{P}}_i = \sum_{j=1}^h m_{ji} \boldsymbol{P}_j = \boldsymbol{P}_i$. As the training goes on, the mixing weights are optimized for finding the best way of mixing multiple attention heads to minimize the training loss. Analogously, for the position-wise mixing, we initialize $\boldsymbol{W}^M$ to $\boldsymbol{0}$ and $\boldsymbol{B}$ to $\boldsymbol{I}$, which introduces zero prior knowledge about the data nor the model structure. From the perspective of initialization, we can view the mix-head attention as a general form of the vanilla multi-head attention because they are the same at the beginning of training. Therefore, we could reasonably expect the mix-head attention to be more expressive than the vanilla multi-head attention.

3.3. Orthogonal regularization

It is worth noting that we do not constrain the mixing weight matrix to be a column stochastic matrix in the previous formulation, i.e., the mixing weights are unnormalized ($\sum_{j=1}^h m_{ji} \neq 1$). The resulting side effect is that the mixed attention weights do not follow a valid probability distribution, i.e., $\bar{\boldsymbol{P}}_i \boldsymbol{1} \neq \boldsymbol{1}$. We elaborate on the reason for this choice and then propose an alternative orthogonal regularization for normalization.

Firstly, we allow the mixing weight matrix to have negative values because it can be more expressive than a non-negative matrix. This means that not all heads contribute positively to the new mixed head and some of them should be subtracted. If we want to normalize the mixing weights while keeping negative values, the commonly used Softmax function is not suitable because it only produces positive values. Therefore, we can only choose the following normalization function.

$$\text{norm}(\boldsymbol{M}_{\cdot i}) = \frac{\boldsymbol{M}_{\cdot i}}{\sum_{j=1}^h m_{ji}}, \tag{13}$$

where $\boldsymbol{M}_{\cdot i}$ is the ith column vector of $\boldsymbol{M}$. However, although Eq. (13) perseveres negative values, it is numerically unstable because the denominator could be zero, which is not acceptable in actual implementation. Therefore, we have to give up the property of being a stochastic matrix and turn to an alternative ℓ_2 normalization as follows.

$$\text{norm}_{\ell_2}(\boldsymbol{M}_{\cdot i}) = \frac{\boldsymbol{M}_{\cdot i}}{\sqrt{\sum_{j=1}^h m_{ji}^2}} = \frac{\boldsymbol{M}_{\cdot i}}{\|\boldsymbol{M}_{\cdot i}\|_2}. \tag{14}$$

The ℓ_2 normalization is more numerically stable because its denominator is strictly greater than zero. More importantly, an ℓ_2-normalized vector is a unit vector, i.e., $\boldsymbol{M}_{\cdot i}^T \boldsymbol{M}_{\cdot i} = 1$. This unit

property makes it easy to implement the constraint, as we will show later.

On the other hand, as previous research has shown that different attention heads specialize in specific aspects [1,19,20], we hope that the new learned mix-head attention will not lose this property. Our proposal is to make the mixed heads as diverse as possible. Note that a mixed head is a linear combination of other heads weighted by a column of the mixing weight matrix, i.e., $\bar{\boldsymbol{P}}_i = \sum_{j=1}^{h} m_{ji}\boldsymbol{P}_j$. Therefore, we constrain the columns of $\boldsymbol{M}$ to be orthogonal to guarantee the diversity of the mixed heads, which is formulated as

$$\boldsymbol{M}_{\cdot i}^T \boldsymbol{M}_{\cdot j} = 0, \ \forall i \neq j. \tag{15}$$

The unit property and the column orthogonality property together make $\boldsymbol{M}$ an orthogonal matrix. Therefore, we can easily implement all the above constraints by one regularization term as follows.

$$R_{orth} = \|\boldsymbol{M}^T\boldsymbol{M} - \boldsymbol{I}\|_F^2. \tag{16}$$

3.4. Theoretical justification

We justify that the mix-head attention is able to output a high-rank attention weight matrix under certain conditions. Without loss of generality, we consider the case of $\bar{\boldsymbol{P}} = \boldsymbol{P}_1 + \boldsymbol{P}_2$ with the mixing weights m absorbed, from which the general case of $\bar{\boldsymbol{P}} = \sum_{i=1}^{h} m_i\boldsymbol{P}_i$ can be easily generalized. Marsaglia [21] gives a lower bound for the rank of the sum of two matrices as follows.

Theorem 2. *Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be two matrices of the same size. Let their row spaces be $\mathscr{R}_1$ and $\mathscr{R}_2$; their column spaces be $\mathscr{C}_1$ and $\mathscr{C}_2$. If*

$$d = \dim(\mathscr{R}_1 \cap \mathscr{R}_2), \tag{17}$$

$$c = \dim(\mathscr{C}_1 \cap \mathscr{C}_2), \tag{18}$$

then

$$\operatorname{rank}(\boldsymbol{A}+\boldsymbol{B}) \geq \operatorname{rank}(\boldsymbol{A}) + \operatorname{rank}(\boldsymbol{B}) - c - d. \tag{19}$$

By the theorem, $\operatorname{rank}(\bar{\boldsymbol{P}})$ is no smaller than $\operatorname{rank}(\boldsymbol{P}_1)$ and $\operatorname{rank}(\boldsymbol{P}_2)$ if $\min\{\operatorname{rank}(\boldsymbol{P}_1), \operatorname{rank}(\boldsymbol{P}_2)\} \geq c + d$. In other words, we need c and d to be as small as possible to make $\bar{\boldsymbol{P}}$ high-rank. Fortunately, there are a lot of empirical evidence that different attention heads have their specific attending subspaces [1,19,20], which suggests that the row/column spaces of $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ are non-intersected and thus c and d are small. The mixed attention weight matrix could be high-rank in practice, and our empirical studies also confirm this observation.

Furthermore, we can explicitly impose orthogonal constraint on $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$, i.e., $\boldsymbol{P}_1\boldsymbol{P}_2^T = \boldsymbol{0} \Leftrightarrow d = 0$ and $\boldsymbol{P}_1^T\boldsymbol{P}_2 = \boldsymbol{0} \Leftrightarrow c = 0$. However, such a constraint cannot be strictly satisfied without degeneration. Hence, we relax the constraint by the following regularization terms.

$$R_d = \|\boldsymbol{P}_1\boldsymbol{P}_2^T\|_F^2, \quad R_c = \|\boldsymbol{P}_1^T\boldsymbol{P}_2\|_F^2. \tag{20}$$

By minimizing these regularization terms, a high-rank matrix $\bar{\boldsymbol{P}}$ can be achieved.[1]

4. Experiments

In this section, we present the experimental results of the proposed method on typical NLP tasks including language modeling, machine translation, and finetuning BERT. Then we take language modeling as an example to further investigate the proposed method.

[1] Though we did not apply this regularization in our experiments as the mix-head attention already produces promising results (see Fig. 1), we still provide it as a tool for readers' use.

4.1. Language modeling

4.1.1. Datasets

The language modeling experiments are performed on three widely used public datasets including the WikiText-2 & WikiText-103 datasets [18] and the One Billion Word dataset (LM1B) [22]. The WikiText-103 dataset comprises consecutive sentences and has about 103M training tokens and a vocabulary of 267K word types. The WikiText-2 dataset is a small version of the WikiText-103 dataset which shares the same validation set and test set, but has a smaller training set of 2M tokens. The One Billion Word dataset is the largest language modeling benchmark with shuffled sentences, which contains approximately 768M training tokens and a vocabulary of 793K word types.

4.1.2. Model configuration and experimental setting

We implement our mix-head attention upon four Transformer baselines including the Vanilla Transformer [1], Transformer-Baevski [17], Transformer-XL [2], and Synthesizer [23]. Vanilla Transformer is a network simply stacking multiple multi-head attention layers mentioned before. Transformer-Baevski is a Transformer variant which adopts an adaptive input embedding technique to lower the computational complexity. Transformer-XL introduces a recurrent mechanism into the Transformer. Synthesizer replaces the dot-product attention with the synthetic attention which is parameterized by a shallow neural network. The Transformer-XL model is implemented using the official source code,[2] and the others are implemented using the fairseq toolkit[3] [24]. We refer to the position-independent and the position-wise mixing as the Mixhead A and the Mixhead B, respectively. If not specified, we generally follow the default parameter configuration given in their original source code. Note that for the Transformer-Baevski model, the default parameter configuration is too computationally intensive for us to bear. Therefore, we have to half the effective batch size for saving the experiment budget. We ensure that the Mixhead and the baseline methods use identical hyper-parameters for comparability. Please refer to the appendix for whole detailed model configurations.[4]

In addition, a prior work proposes an alternative non-linear activation function Sigsoftmax [25] to replace the Softmax function to address the low-rank bottleneck problem. We also compare the Mixhead with this Sigsoftmax function, which is defined as follows.

$$\operatorname{Sigsoftmax}(\boldsymbol{A}_{i\cdot}) = \frac{\exp(\boldsymbol{A}_{i\cdot})\sigma(\boldsymbol{A}_{i\cdot})}{\sum_{j=1}^{n}\exp(A_{ij})\sigma(A_{ij})}, \tag{21}$$

where $\sigma(\cdot)$ is the sigmoid function.[5] Note that we will compare the prior work Fixhead [13] in the later ablation section since it simply increases the head size.

We evaluate all four Transformers on the medium-sized dataset WikiText-103. However, for the small dataset WikiText-2, there is a known issue that Transformers are easy to overfit on small datasets. Therefore, we only evaluate two small models Vanilla Transformer and Transformer-XL on this small dataset. For the large dataset LM1B, we evaluate two models Transformer-XL and Transformer-Baevski on it. Since LM1B comprises shuffled sentences and there is no contextual dependency between consecutive sentences, it is mainly used for testing the ability of modeling short-term dependency. Therefore, the default configuration of the Transformer-XL model only takes a short sequence of length

[2] https://github.com/kimiyoung/transformer-xl.

[3] https://github.com/pytorch/fairseq.

[4] https://github.com/zhongzhang1/mixhead

[5] We subtract the max value to prevent overflow in our implementation.

Table 1
Language modeling results on the WikiText-103 dataset, in terms of perplexity. The parameters in parentheses after the model names represent the number of layers l, the number of heads h, the head size d_k and the sequence length n, respectively.

Models	# Para.	Valid.	Test
Vanilla Transformer ($l = 6, h = 8, d_k = 64, n = 512$)	156M	30.75	29.78
Vanilla Transformer + Sigsoftmax	–	30.66	29.61
Ours - Vanilla Transformer + Mixhead A	+384	30.00	28.86
Ours - Vanilla Transformer + Mixhead A + Sigsoftmax	+384	30.08	28.93
Ours - Vanilla Transformer + Mixhead B	+3456	**29.65**	**28.67**
Ours - Vanilla Transformer + Mixhead B + Sigsoftmax	+3456	29.68	28.87
Transformer-Baevski ($l = 16, h = 8, d_k = 128, n = 512$)	247M	21.45	20.54
Transformer-Baevski + Sigsoftmax	–	21.10	20.18
Ours - Transformer-Baevski + Mixhead A	+1024	20.49	19.58
Ours - Transformer-Baevski + Mixhead A + Sigsoftmax	+1024	20.98	19.95
Ours - Transformer-Baevski + Mixhead B	+17408	**20.37**	**19.47**
Ours - Transformer-Baevski + Mixhead B + Sigsoftmax	+17408	20.88	20.06
Transformer-XL ($l = 16, h = 10, d_k = 41, n = 150$)	151M	23.41	23.69
Transformer-XL + Sigsoftmax	–	23.29	23.56
Ours - Transformer-XL + Mixhead A	+1600	22.99	23.56
Ours - Transformer-XL + Mixhead A + Sigsoftmax	+1600	22.96	23.47
Ours - Transformer-XL + Mixhead B	+8160	22.66	22.92
Ours - Transformer-XL + Mixhead B + Sigsoftmax	+8160	**22.57**	**22.89**
Synthesizer ($l = 16, h = 8, d_k = 128, n = 512$)	248M	24.32	23.79
Synthesizer + Sigsoftmax	–	24.02	23.53
Ours - Synthesizer + Mixhead A	+1024	23.98	23.51
Ours - Synthesizer + Mixhead A + Sigsoftmax	+1024	24.03	23.57
Ours - Synthesizer + Mixhead B	+17408	**23.40**	**22.77**
Ours - Synthesizer + Mixhead B + Sigsoftmax	+17408	23.47	22.92

Table 2
Language modeling results on the WikiText-2 dataset, in terms of perplexity. The parameters in parentheses after the model names represent the number of layers l, the number of heads h, the head size d_k and the sequence length n, respectively.

Models	# Para.	Valid.	Test
Vanilla Transformer ($l = 16, h = 8, d_k = 64, n = 256$)	67M	92.16	83.97
Vanilla Transformer + Sigsoftmax	–	91.62	83.34
Ours - Vanilla Transformer + Mixhead A	+1024	91.56	83.39
Ours - Vanilla Transformer + Mixhead A + Sigsoftmax	+1024	91.09	82.04
Ours - Vanilla Transformer + Mixhead B	+9216	92.57	82.93
Ours - Vanilla Transformer + Mixhead B + Sigsoftmax	+9216	**90.97**	**81.77**
Transformer-XL ($l = 16, h = 10, d_k = 41, n = 150$)	40M	67.52	64.47
Transformer-XL + Sigsoftmax	–	66.90	63.93
Ours - Transformer-XL + Mixhead A	+1600	65.91	63.25
Ours - Transformer-XL + Mixhead A + Sigsoftmax	+1600	65.87	63.02
Ours - Transformer-XL + Mixhead B	+8160	65.53	62.46
Ours - Transformer-XL + Mixhead B + Sigsoftmax	+8160	**64.58**	**61.88**

32 as input. We adopt this default configuration for evaluation and find that the low-rank bottleneck still exists when processing short sequences, we will discuss this finding in the later section. To demonstrate the low-rank bottleneck effect when processing long sequences, we partition the training data into blocks of 512 contiguous tokens instead of individual sentences for batching, then we use the Transformer-Baevski model for evaluation. We show that language modeling on this setting can still benefit from the Mixhead even if the sentences in a sequence have no long-term dependency.

We train vanilla baseline models and the Mixhead enhanced models on the training set, and evaluate them on the validation set. The final test results are obtained from the checkpoint that performs the best on the validation set. We use perplexity as the performance metric for the language modeling task.

4.1.3. Results

Table 1 shows the language modeling results on the WikiText-103 dataset in terms of perplexity. As we can see, Mixhead outperforms the baseline methods on all settings. The position-wise mixing (Mixhead B) is better than the position-independent mixing (Mixhead A) since it has more parameters. Still, the increased numbers of parameters are quite small. The alternative activation function Sigsoftmax outperforms the baseline method, though it still underperforms our Mixhead method. However, combining both the Mixhead and the Sigsoftmax methods does not bring a significant improvement on this dataset.

The WikiText-2 dataset is a subset of the WikiText-103 dataset. Because of its small size, we encountered a severe overfitting problem. Therefore, we have to apply stronger regularization techniques like dropout to obtain reasonable results. Nevertheless, we did not finetune the model hyper-parameters because our purpose is only to demonstrate the effectiveness of the proposed method, not to achieve the state-of-the-art performance on the small dataset. Table 2 reports the language modeling results on this dataset. Mixhead generally outperforms baseline methods except for the setting of Vanilla Transformer + Mixhead B. This may be because stronger regularization makes the training process more volatile and a bad local optimum is obtained. Different from the WikiText-103 dataset, combining both the Mixhead and the Sigsoftmax methods achieves significant improvement on this

Table 3
Language modeling results on the One Billion Word dataset, in terms of perplexity. The parameters in parentheses after the model names represent the number of layers l, the number of heads h, the head size d_k and the sequence length n, respectively.

Models	# Para.	Valid.	Test
Transformer-Baevski ($l = 16, h = 8, d_k = 128, n = 2048$)	330M	38.03	37.66
Ours - Transformer-Baevski + Mixhead A	+1024	37.05	36.90
Ours - Transformer-Baevski + Mixhead B	+17408	**34.21**	**34.38**
Transformer-XL ($l = 18, h = 8, d_k = 128, n = 32$)	294M	32.03	32.16
Ours - Transformer-XL + Mixhead A	+1152	29.88	29.95
Ours - Transformer-XL + Mixhead B	+19584	**29.44**	**29.51**

small dataset. However, we have to be cautious to draw any conclusion based on the small dataset because it might not be generalized to other settings.

Table 3 reports the language modeling results on the large-sized LM1B dataset. Our Mixhead method outperforms the baseline methods by significant margins, and the position-wise mixing is better than the position-independent mixing as expected. However, it is worth noting that the Transformer-Baevski baseline method can get a much better perplexity according to the paper's results, and the performance gap between the baseline method and the Mixhead method can be further narrowed if training longer for full convergence. This is because the effective batch size of our runs is halved and the way of batching is different. The original setting requires 64 GPUs for training, which is beyond our ability. Despite the compromise of experimental settings, we can see that Mixhead is effective in improving the perplexity and convergence. Interestingly, Mixhead can outperform the Transformer-XL baseline despite the fact that it should not be affected by the low-rank bottleneck problem as $d_k > n$. We will discuss this phenomenon in the later ablation study section.

4.2. Machine translation

4.2.1. Remark

Before we start presenting the machine translation experiment, we must justify that machine translation is not a suitable task for demonstrating the low-rank bottleneck problem. We include the experiment only for a comprehensive study. The reason has two folds: Firstly, the common machine translation corpora are built for short sentence translation, whose average sequence lengths are short (e.g., $n \approx 24 < d_k$). Thus, it is not affected by the low-rank bottleneck problem. Secondly, the machine translation tasks adopt the encoder–decoder architecture and is sensitive to source–target alignment [1,26,27]. To translate a word correctly, it generally requires a precise attention of certain specific words. In other words, the attention weight matrix for the machine translation task is inherently low-rank, which is against the basic motivation of the Mixhead. However, we still show that the Mixhead can outperform the baseline model when intentionally setting $n > d_k$.

4.2.2. Dataset

Since the machine translation task is not the target task of this paper, we chose a small dataset only for a primary demonstration. We use the IWSLT 2014 German–English (De-En) dataset [28] for demonstration, which contains approximately 160K sentence pairs for training and 7K sentence pairs for validation. We concatenate chunks *tst2010*, *tst2011*, *tst2012*, *dev2010* and *dev2012* to form the test set. The source/target language has an average sequence length of 24.2/23.6 and a vocabulary of 8848/6632 word types, respectively.

Table 4
Machine translation results on the IWSLT 2014 DE-EN dataset, in terms of BLEU.

Models	h	d_k	n	Validation	Test
Baseline	4	128	≈24	**35.28**	**34.76**
+ Mixhead A				34.71	33.84
+ Mixhead B				35.06	34.25
Baseline	32	16	≈24	**33.31**	**32.60**
+ Mixhead A				32.94	32.58
+ Mixhead B				33.15	32.35
Baseline	64	8	≈24	28.69	27.88
+ Mixhead A				31.74	31.32
+ Mixhead B				**31.75**	**31.44**

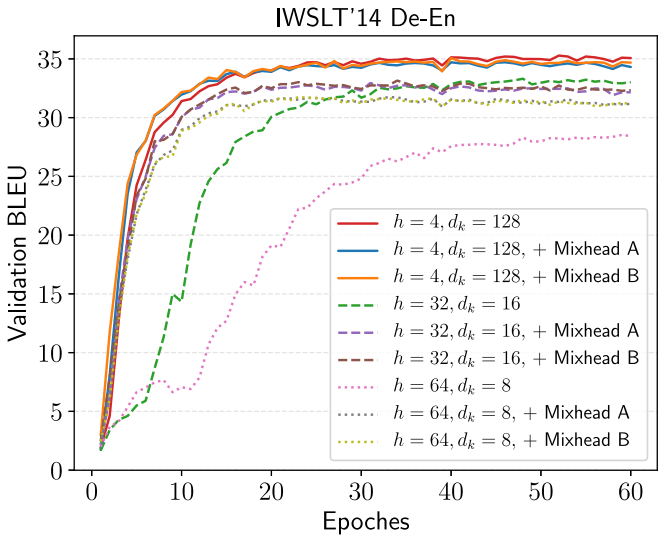


Fig. 2. Learning curve of machine translation on the IWSLT 2014 DE-EN dataset.

4.2.3. Model configuration

We build the baseline Transformer model [1] using the fairseq toolkit. Specifically, it has a 6-layer encoder and a 6-layer decoder with 512-dimensional hidden units and embeddings. The inner feed-forward layer has 1024-dimensional hidden units. For the normal setting, we set the number of heads $h = 4$ that yields $d_k = 128$. To demonstrate the low-rank bottleneck effect ($n > d_k$), we intentionally set the number of heads h to be 32 and 64, and the resulting head size d_k is 16 and 8, respectively. The three settings have the same number of parameters, thus the performance discrepancy is mainly from the mix-head attention mechanism. We give the detailed model configuration in the appendix.

4.2.4. Results

We summarize the machine translation results in terms of BLEU score in Table 4 and plot the corresponding learning curves in Fig. 2.

Table 5
GLUE test results scored by the online evaluation server.

Tasks	Mixhead	BERT - Our run	BERT - Official
MNLI-m	**83.6**	83.4	84.6
MNLI-mm	82.8	**82.9**	83.4
QQP	**70.2**	70.1	71.2
QNLI	**90.9**	90.7	90.5
SST-2	**93.7**	93.5	93.5
CoLA	**55.4**	52.6	52.1
STS-B	**85.2**	84.7	85.8
MPRC	**87.7**	87.4	88.9
RTE	**68.4**	68.0	66.4

As mentioned earlier, when the sequence length is much smaller than the head size, i.e., $n \approx 24 \ll d_k = 128$, the model is not affected by the low-rank bottleneck problem, but the introduced additional mixing weight parameters could be of no use to machine translation and increase the learning burden, which accounts for why Mixhead underperforms the baseline to an extent. When $n \approx 24 > d_k = 16$, the model starts to be affected by the low-rank bottleneck problem. Note that 24 is the average length of the input sequence, the medium length is only 19 which is marginally larger than 16. Therefore, the low-rank bottleneck effect is not obvious, and the benefits of Mixhead just start to appear. Although the Mixhead still underperforms the baseline, the gap of the test BLEU scores between the Mixhead A/B and the baseline narrows from 0.92/0.51 to 0.02/0.25. When $n \approx 24 \gg d_k = 8$, the model largely suffers from the low-rank bottleneck problem. Therefore, the model can benefit a lot from the Mixhead. We can see that the Mixhead outperforms the baseline by a large margin.

From the learning curves shown in Fig. 2, we can see that the Mixhead always converges faster than the baseline, especially for the settings of $n > d_k$. We conjecture that mixing heads can make the learning parameters less discrete and yield smoother gradient flow, which eases the difficulty of optimization. In addition, we also experiment with fixing the number of heads and head size while increasing sequence length by concatenating sentences (e.g., $n \approx 240$) to synthesize the low-rank bottleneck effect. We observe interesting results that the Mixhead can converge well and output reasonable translation (BLEU $= 34.5$), while the baseline does not learn any translation at all (BLEU < 1.0). This further suggests that the Mixhead has advantages on convergence. However, the reason behind remains unclear, and we will leave this issue to future study.

4.3. Finetuning BERT

4.3.1. Settings

It would be ideal to train a BERT model from scratch to evaluate the effectiveness of Mixhead for pre-trained language models. However, such a setting is too expensive for us to bear. Therefore, we decide to only conduct finetuning experiments using the pre-trained BERT. Since the parameters of Mixhead are not pre-trained, and the pre-trained BERT parameters are not trained to fit the mixing paradigm, we have to modify the standard finetuning procedure with more training steps and a larger learning rate for Mixhead parameters. We implement the position-wise mixing upon the Transformers toolkit[6] [29], and adopt the pre-trained BERT-base model for finetuning. We finetune the model on the GLUE [30] and the SQuAD v1.1 benchmark [31]. The whole detailed parameters settings are listed in the appendix.

[6] https://github.com/huggingface/transformers.

Table 6
SQuAD v1.1 validation results.

Models	EM	F1
Official - BERT	80.8	88.5
Our run - BERT	80.6	88.0
+ Mixhead	**81.0**	**88.4**

4.3.2. Results

The results of finetuning BERT on the GLUE and the SQuAD benchmark are presented in Tables 5 and 6, respectively. On the GLUE benchmark, Mixhead outperforms the baseline over 8 tasks by a small margin except for the CoLA task which gains a significant +2.8 improvement. On the SQuAD benchmark, Mixhead achieves a 0.4/0.4 improvement over the baseline in terms of EM/F1. It is not a surprise that Mixhead only gains a small improvement on finetuning tasks since the BERT weights are not designed and pre-trained for the mixing. We have to admit that finetuning BERT is still insufficient to validate the effectiveness of Mixhead. However, we do not have enough computational resources to support a full training-from-scratch experiment at present, we will leave it to future research.

4.4. Ablation study

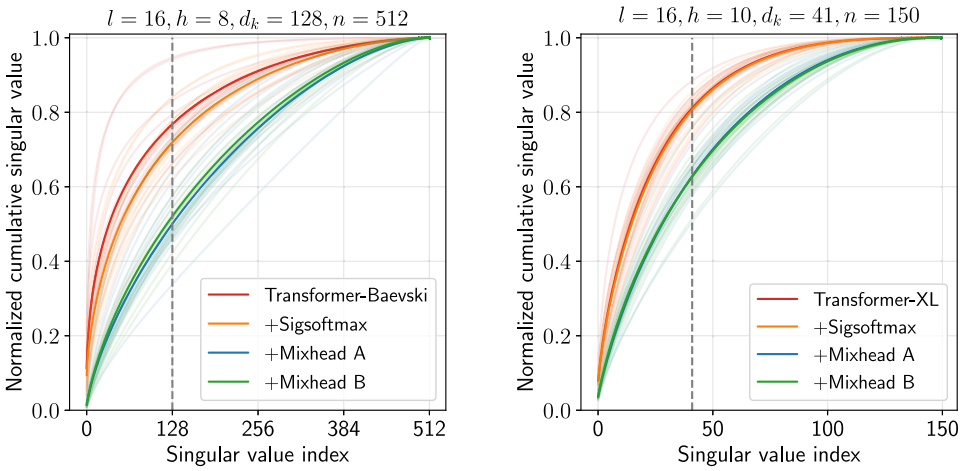
In this section, we perform ablation experiments over a number of facets of Mixhead. We take the Transformer-Baevski model [17] as the baseline and perform language modeling on the WikiText-103 dataset. It is important to note that ablation results from different tables are not comparable even though they use the same network because they use different data batching and evaluation protocols. We have to make such a compromise because of the hardware limit. However, we strictly ensure that the results within each table use the identical setting and thus are comparable. We give a detailed configuration of the ablation study in the appendix.

4.4.1. Spectrum analysis of the attention weight matrix

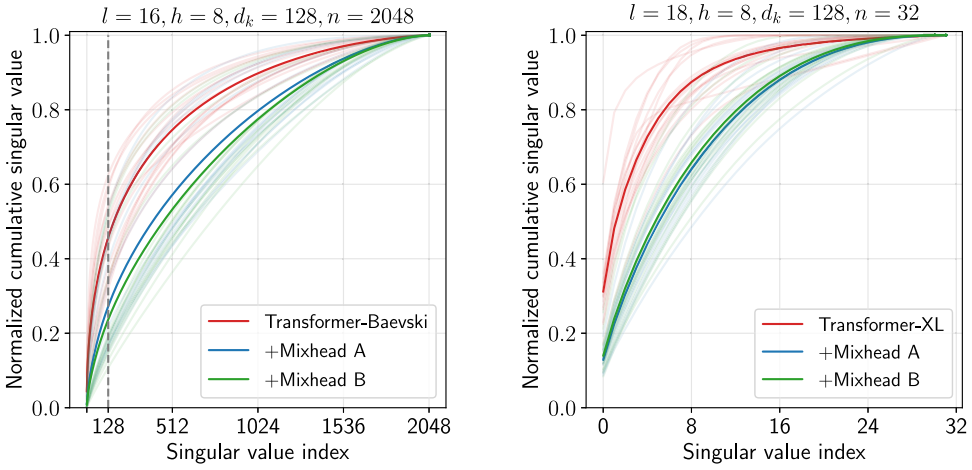
We verify that Mixhead achieves the goal of improving the rank of the attention weight matrix. It is worth noting that when we say that a matrix is of low rank, we actually mean that the singular values of the matrix follow a long-tailed distribution, which means a few large singular values take the most mass of the distribution. The reason is that there are no strictly zero-valued singular values of the matrix for us to calculate the rank because of the non-linear function Softmax. Therefore, we give a spectrum analysis of the attention weight matrix.

We report the results on the WikiText-103 dataset and the LM1B dataset. Fig. 3 shows the normalized cumulative singular values of the attention weight matrices. The closer the curve is to the upper left corner, the lower rank of the corresponding matrix. As we can see, the vanilla baseline does show a low-rank effect since the singular values follow long-tailed distributions, especially for certain layers. For the Sigsoftmax method, it has only a marginal improvement on the spectrum. Compared with the baseline, the two Mixhead methods achieve more uniformly distributed singular values. However, there is no significant difference between the two mixing methods.

Interestingly, the low-rank bottleneck effect also appears in the setting of $n < d_k$ (the lower right figure). This is because the input sequence matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ is not full row rank. The reason lies in that English sentences follow similar syntactic structure patterns, and sentences are similar in frequently used function words (e.g., articles, pronouns, prepositions, conjunctions, et al.). Therefore, the input sequences are linearly dependent. This phenomenon further exacerbates the low-rank bottleneck problem, and it also explains the performance improvement of language modeling when $n < d_k$.



(a) WikiText-103



(b) LM1B

Fig. 3. The spectrum analysis of the attention weight matrix on the WikiText-103 and the LM1B datasets. The dash lines represent different layers and the bold lines represent the average values over layers.

Table 7
Ablation study over random mixing and average mixing.

Models	h	d_k	n	Validation	Test
Baseline	8	64	512	21.82	20.89
+Mixhead A				21.20	20.48
+Mixhead B				**21.10**	**20.18**
+Mixhead$_{\text{rand}}$				22.26	21.39
+Avghead				22.50	21.57

4.4.2. Random mixing and average mixing

The random attention is shown to work in recent papers [23, 32,33]. Analogously, we study whether randomly mixing the attention heads can have improvement. We use Mixhead$_{\text{rand}}$ to denote the random mixing method, whose mixing weight matrix $\boldsymbol{M}$ is initialized from the normal distribution and fixed during the training. We also set a comparison baseline denoted by Avghead whose $\boldsymbol{M}$ is initialized from the uniform distribution, which essentially averages all heads. As shown in Table 7, it is no surprise that Avghead is the worst since it degenerates to the single-head attention with redundant parameters. Mixhead$_{\text{rand}}$ outperforms Avghead but underperforms Mixhead and baseline by a large margin. This suggests that learning an effective combination of multiple heads is essential to the model performance.

4.4.3. Ablation over number of heads, sequence length, and head size

We explore the impact of number of heads h, sequence length n, and head size d_k on language modeling performance, respectively.

Number of Heads. We fix the head size $d_k = 64$, sequence length $n = 512$, and then vary the number of heads h to observe its impact on language modeling performance. Note that the head size and the number of heads are decoupled, thus the number of model parameters increases with the number of heads. As we can see from Table 8, a general trend is that the performance improves as the number of heads increases. However, we observe that the performance of $h = 10$ is worse than that of $h = 8$. We consider this is because the model starts to overfit since the number of parameters increases and the training perplexity of $h = 10$ is smaller than that of $h = 8$ ($19.51 < 19.90$ for baseline, and $18.08 < 18.61$&$18.00 < 18.33$ for Mixhead A/B).

Sequence Length. Analogously, we vary the sequence length n while fixing the other parameters and give the ablation results in Table 9. Since language modeling is highly context-dependent, models can always benefit from longer sequences. Compared with the baseline, the Mixhead has lower perplexities for all lengths. Note that the numbers of model parameters are the same for all length settings, thus the improvement comes from the increase of sequence length rather than the increase of parameters. However, the quadratic complexity of self-attention

Table 8
Ablation study over number of heads h, in terms of validation/test perplexity.

h	d_k	n	Baseline	Mixhead	
				A	B
2	64	512	23.62/22.87	22.90/22.11	22.66/21.89
4			22.55/21.68	22.02/21.19	21.95/21.02
6			22.04/21.09	22.09/21.13	22.02/21.15
8			21.82/20.89	21.20/20.48	21.10/20.18
10			22.11/21.33	21.29/20.49	21.65/20.67

Table 9
Ablation study over sequence length n, in terms of validation/test perplexity.

n	h	d_k	Baseline	Mixhead	
				A	B
32	8	64	37.05/27.36	36.11/26.59	36.19/26.36
64			29.59/24.66	28.87/23.93	28.82/23.83
128			25.47/23.07	24.66/22.38	24.99/22.73
256			23.05/22.13	22.27/21.45	22.40/21.57
512			21.79/21.69	21.34/21.29	21.26/21.16

Table 10
Ablation study over head size d_k, in terms of validation/test perplexity.

d_k	n	h	Baseline	Mixhead	
				A	B
32	512	8	22.66/21.77	22.40/21.74	22.35/21.46
64			21.85/21.02	21.39/20.60	21.49/20.75
128			21.03/20.15	20.23/19.30	21.08/20.09
256			20.72/19.74	20.15/19.29	20.15/19.22
512[a]			21.21/20.17	20.37/19.46	20.36/19.51

[a]Indicates the Fixhead [13] which sets the head size equal to the sequence length.

prevents increasing the sequence length without limit. Therefore, we recommend setting $n = 512$ for the language modeling task.

Head Size. Table 10 shows the ablation results over the head size d_k. Note that this ablation study can be seen as a comparison between the Mixhead and the Fixhead [13] because the Fixhead simply increases the head size. Although we indeed observe that the performance improves with the increase of head size, the model starts to overfit when $d_k = n = 512$. Besides, a large head size considerably increases the computing and memory burden. This suggests that naively increasing the head size is not the best solution to the problem. By comparison, Mixhead can achieve a better result while having significantly fewer parameters. Take the Baseline $d_k = 256$ and the Mixhead A $d_k = 128$ as an example, Mixhead achieves a lower perplexity $19.30 < 19.74$ but only uses $247\text{M}/314\text{M} = 0.79$ times the number of parameters, and note that the Fixhead that has 448M parameters encounters a heavy overfitting problem and starts to fail.

5. Related work

5.1. Transformer model

The Transformer is initially developed as an encoder–decoder model for the machine translation task [1]. Now, when talking about Transformers, it usually refers to the encoder that is used as a feature extractor. The Transformer has been largely popularized since the success of the pre-trained language models like BERT [34] and GPT [35], which are essentially stacking of Transformer blocks. Since then, Transformer variants have been rapidly emerging from various aspects. For example, Dai et al. [2] equip the Transformer with a recurrence mechanism for modeling long sequence; Baevski and Auli [17] introduce the adaptive input representation for neural language modeling, which substantially reduces model parameters; Wang et al. [36] propose a constituent attention module to integrate the tree structure; Child et al. [37], Beltagy et al. [38] and Zaheer et al. [32] explore the sparsity structure of the attention weight matrix; Kitaev et al. [39] use a locality-sensitive hashing to replace the dot-product attention for improving efficiency; Tay et al. [23] propose to replace the dot-product attention with the synthetic attention which is parameterized by a shallow neural network. Note our method is orthogonal to these methods, and the low-rank bottleneck problem exists universally in Transformers, thus our solution can be quite general for most models.

5.2. Low-rank bottleneck

Yang et al. [14] first identify the Softmax bottleneck in the last Softmax layer of recurrent neural networks (RNNs). They prove that when the output projection dimension is smaller than the vocabulary size, the Softmax function is incapable of producing the desired true data distribution. Subsequent work further investigates this problem for RNNs [25,40–43]. However, the situation of Transformers is slightly different from that of RNNs. The Softmax bottleneck in RNNs occurs because it is the Log-softmax function that the logits actually go through for computing the negative log-likelihood (NLL) loss. And the output of the Log-softmax function is the linear combination of the input and an all-one vector $\mathbf{1}$, which increases the rank by at most one [25]. In other words, the Log-softmax function is essentially linear and thus the expressive power of the RNN model is limited. For Transformers, they do not compute NLL loss when calculating the attention weights, i.e., it is the Softmax function that the logits go through, not the Log-softmax function. Therefore, the attention weight matrix should be arbitrarily high-rank because of the non-linear Softmax function. However, Bhojanapalli et al. [13] prove that the Softmax function is not non-linear enough to increase the rank of the attention weight matrix so that the Transformers still suffer from the low-rank bottleneck problem. They propose to set the head size equal to the sequence length to tackle the problem, which however, encounters severe parameter explosion and overfitting problems. Besides, another line of contrasting work leverages the low-rank structure in the attention weight matrix and uses the low-rank matrix factorization technique to reduce the computational complexity of Transformers [44], which also confirms the existence of the low-rank bottleneck in multi-head attention models.

5.3. Mixture model

The mixture models have been developed for a long history in the field of machine learning. The most popular paradigm is the Mixture of Experts (MoE) [45,46], which sets multiple independent sub-models or layers to learn from the data and then aggregates them to output the final result. The Mixture of Softmax (MoS) proposed by Yang et al. [14] can be seen as a particular MoE model, which introduces multiple Softmax layers and aggregates their outputs to break the Softmax bottleneck. Our work is partially inspired by the MoS model. However, we are different in two folds. Firstly, MoS needs to introduce additional Softmax layers and then mix their outputs, while Mixhead can naturally leverage the existing attention heads for mixing, which only introduces a small number of parameters. Secondly, MoS targets the last Softmax layer of the RNN model, and Mixhead is for all multi-head attention layers of the Transformer model. For Transformer models, the most closely related models are the Weighted Transformer [47] and the Multi-branch Attentive Transformer [48]. However, these methods cannot tackle the low-rank bottleneck problem, and we are different not only in motivation but also in formula details.

6. Conclusion

In this paper, we study the low-rank bottleneck problem in multi-head attention models. We propose a simple yet effective method to tackle this problem which is to replace the vanilla attention head with a mix-head attention parameterized by a learnable mixing weight. We show that our method is effective in breaking the low-rank bottleneck and is more efficient than the previously proposed method. Besides, our method is easy to implement and can be adopted to any multi-head attention model. For future study, we are planning to investigate how the Mixhead will behave when training large-scale pre-trained language models from scratch.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by the Fundamental Research Funds for the Central Universities, China (ZYGX2019Z014), National Natural Science Foundation of China (61976044, 52079026), Fok Ying-Tong Education Foundation (161062), and Sichuan Science and Technology Program (2020YFH0037).

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.knosys.2021.108075.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: Advances In Neural Information Processing Systems, Vol. 30, 2017.
[2] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. Le, R. Salakhutdinov, Transformer-XL: Attentive language models beyond a fixed-length context, in: Proceedings Of The 57th Annual Meeting Of The Association For Computational Linguistics, 2019, pp. 2978–2988.
[3] W. Huang, Y. Mao, Z. Yang, L. Zhu, J. Long, Relation classification via knowledge graph enhanced transformer encoder, Knowl.-Based Syst. 206 (2020) 106321.
[4] Y. Du, Q. Li, L. Wang, Y. He, Biomedical-domain pre-trained language model for extractive summarization, Knowl.-Based Syst. 199 (2020) 105964.
[5] R. Li, Z. Jiang, L. Wang, X. Lu, M. Zhao, D. Chen, Enhancing transformer-based language models with commonsense representations for knowledge-driven machine comprehension, Knowl.-Based Syst. 220 (2021) 106936.
[6] Y. Chen, H. Li, DAM: Transformer-based relation detection for question answering over knowledge base, Knowl.-Based Syst. 201–202 (2020) 106077.
[7] M. Chen, A. Radford, R. Child, J. Wu, H. Jun, P. Dhariwal, D. Luan, I. Sutskever, Generative pretraining from pixels, in: Proceedings Of The 37th International Conference On Machine Learning, 119, 2020, pp. 1691–1703.
[8] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, P. Jiang, BERT4rec: Sequential recommendation with bidirectional encoder representations from transformer, in: Proceedings Of The 28th ACM International Conference On Information And Knowledge Management, 2019, pp. 1441–1450.
[9] S. Yun, M. Jeong, R. Kim, J. Kang, H.J. Kim, Graph transformer networks, in: Advances In Neural Information Processing Systems, Vol. 32, 2019, pp. 11983–11993.
[10] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, N. Houlsby, An image is worth 16x16 words: Transformers for image recognition at scale, in: International Conference On Learning Representations, 2021.
[11] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, S. Zagoruyko, End-to-end object detection with transformers, in: European Conference On Computer Vision, 2020, pp. 213–229.
[12] H. Wang, Y. Zhu, B. Green, H. Adam, A. Yuille, L.-C. Chen, Axial-deeplab: Stand-alone axial-attention for panoptic segmentation, in: European Conference On Computer Vision, 2020, pp. 108–126.
[13] S. Bhojanapalli, C. Yun, A.S. Rawat, S.J. Reddi, S. Kumar, Low-rank bottleneck in multi-head attention models, in: Proceedings Of The 37th International Conference On Machine Learning, 119, 2020, pp. 864–873.
[14] Z. Yang, Z. Dai, R. Salakhutdinov, W.W. Cohen, Breaking the softmax bottleneck: A high-rank RNN language model, in: International Conference On Learning Representations, 2018.
[15] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings Of The IEEE Conference On Computer Vision And Pattern Recognition, 2016, pp. 770–778.
[16] J.L. Ba, J.R. Kiros, G.E. Hinton, Layer normalization, 2016, arXiv preprint arXiv:1607.06450.
[17] A. Baevski, M. Auli, Adaptive input representations for neural language modeling, in: International Conference On Learning Representations, 2019.
[18] S. Merity, C. Xiong, J. Bradbury, R. Socher, Pointer sentinel mixture models, in: Proceedings Of The 5th International Conference On Learning Representations, 2017.
[19] K. Clark, U. Khandelwal, O. Levy, C.D. Manning, What does BERT look at? An analysis of BERT's attention, in: Proceedings Of The 2019 ACL Workshop BlackboxNLP: Analyzing And Interpreting Neural Networks For NLP, 2019, pp. 276–286.
[20] J. Li, Z. Tu, B. Yang, M.R. Lyu, T. Zhang, Multi-head attention with disagreement regularization, in: Proceedings Of The 2018 Conference On Empirical Methods In Natural Language Processing, 2018, pp. 2897–2903.
[21] G. Marsaglia, Bounds for the Rank of the Sum of Two Matrices, Technical Report, Boeing Scientific Research Laboratories, 1964.
[22] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, T. Robinson, One billion word benchmark for measuring progress in statistical language modeling, in: Proceedings Of The 15th Annual Conference Of The International Speech Communication Association, 2014, pp. 2635–2639.
[23] Y. Tay, D. Bahri, D. Metzler, D.-C. Juan, Z. Zhao, C. Zheng, Synthesizer: Rethinking self-attention in transformer models, 2020, arXiv preprint arXiv:2005.00743.
[24] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, M. Auli, Fairseq: A fast, extensible toolkit for sequence modeling, in: Proceedings Of The 2019 Conference Of The North American Chapter Of The Association For Computational Linguistics, 2019, pp. 48–53.
[25] S. Kanai, Y. Fujiwara, Y. Yamanaka, S. Adachi, Sigsoftmax: Reanalysis of the softmax bottleneck, in: Advances In Neural Information Processing Systems, Vol. 31, 2018, pp. 286–296.
[26] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, in: International Conference On Learning Representations, 2015.
[27] Y. Cheng, S. Shen, Z. He, W. He, H. Wu, M. Sun, Y. Liu, Agreement-based joint training for bidirectional attention-based neural machine translation, in: Proceedings Of The 25th International Joint Conference On Artificial Intelligence, 2016, pp. 2761–2767.
[28] M. Cettolo, J. Niehues, S. Stüker, L. Bentivogli, M. Federico, Report on the 11th IWSLT evaluation campaign, IWSLT 2014, in: Proceedings Of The International Workshop On Spoken Language Translation, 57, 2014.
[29] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T.L. Scao, S. Gugger, M. Drame, Q. Lhoest, A.M. Rush, Transformers: State-of-the-art natural language processing, in: Proceedings Of The 2020 Conference On Empirical Methods In Natural Language Processing, 2020, pp. 38–45.
[30] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, S. Bowman, GLUE: A multi-task benchmark and analysis platform for natural language understanding, in: Proceedings Of The 2018 EMNLP Workshop BlackboxNLP: Analyzing And Interpreting Neural Networks For NLP, 2018, pp. 353–355.
[31] P. Rajpurkar, J. Zhang, K. Lopyrev, P. Liang, SQuAD: 100,000+ questions for machine comprehension of text, in: Proceedings Of The 2016 Conference On Empirical Methods In Natural Language Processing, 2016, pp. 2383–2392.
[32] M. Zaheer, G. Guruganesh, K.A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, A. Ahmed, Big bird: Transformers for longer sequences, in: Advances In Neural Information Processing Systems, Vol. 33, 2020, pp. 17283–17297.
[33] C. Yun, Y.-W. Chang, S. Bhojanapalli, A.S. Rawat, S. Reddi, S. Kumar, O(n) connections are expressive enough: Universal approximability of sparse transformers, in: Advances In Neural Information Processing Systems, Vol. 33, 2020, pp. 13783–13794.
[34] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings Of The 2019 Conference Of The North American Chapter Of The Association For Computational Linguistics: Human Language Technologies, 2019, pp. 4171–4186.
[35] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, Improving Language Understanding by Generative Pre-Training, Technical Report, OpenAI, 2018.

[36] Y. Wang, H.-Y. Lee, Y.-N. Chen, Tree transformer: Integrating tree structures into self-attention, in: Proceedings Of The 2019 Conference On Empirical Methods In Natural Language Processing And The 9th International Joint Conference On Natural Language Processing (EMNLP-IJCNLP), 2019, pp. 1060–1070.
[37] R. Child, S. Gray, A. Radford, I. Sutskever, Generating long sequences with sparse transformers, 2019, arXiv preprint arXiv:1904.10509.
[38] I. Beltagy, M.E. Peters, A. Cohan, Longformer: The long-document transformer, 2020, arXiv preprint arXiv:2004.05150.
[39] N. Kitaev, L. Kaiser, A. Levskaya, Reformer: The efficient transformer, in: International Conference On Learning Representations, 2020.
[40] S. Takase, J. Suzuki, M. Nagata, Direct output connection for a high-rank language model, in: Proceedings Of The 2018 Conference On Empirical Methods In Natural Language Processing, 2018, pp. 4599–4609.
[41] S. Flennerhag, H. Yin, J. Keane, M. Elliot, Breaking the activation function bottleneck through adaptive parameterization, in: Advances In Neural Information Processing Systems, Vol. 31, 2018, pp. 7739–7750.
[42] Z. Yang, T. Luong, R.R. Salakhutdinov, Q.V. Le, Mixtape: Breaking the softmax bottleneck efficiently, in: Advances In Neural Information Processing Systems, Vol. 32, 2019, pp. 5775–5783.
[43] O. Ganea, S. Gelly, G. Bécigneul, A. Severyn, Breaking the softmax bottleneck via learnable monotonic pointwise non-linearities, in: Proceedings Of The 36th International Conference On Machine Learning, 97, 2019, pp. 2073–2082.
[44] S. Wang, B. Li, M. Khabsa, H. Fang, H. Ma, Linformer: Self-attention with linear complexity, 2020, arXiv preprint arXiv:2006.04768.
[45] R.A. Jacobs, M.I. Jordan, S.J. Nowlan, G.E. Hinton, Adaptive mixtures of local experts, Neural Comput. 3 (1) (1991) 79–87.
[46] S.E. Yuksel, J.N. Wilson, P.D. Gader, Twenty years of mixture of experts, IEEE Trans. Neural Netw. Learn. Syst. 23 (8) (2012) 1177–1193.
[47] K. Ahmed, N.S. Keskar, R. Socher, Weighted transformer network for machine translation, 2017, arXiv preprint arXiv:1711.02132.
[48] Y. Fan, S. Xie, Y. Xia, L. Wu, T. Qin, X.-Y. Li, T.-Y. Liu, Multi-branch attentive transformer, 2020, arXiv preprint arXiv:2006.10270.