ROLE OF BIAS TERMS IN DOT-PRODUCT ATTENTION

Mahdi Namazifar, Devamanyu Hazarika, Dilek Hakkani-Tür

Amazon Alexa AI

ABSTRACT

Dot-product attention is a core module in the present generation of neural network models, particularly transformers, and is being leveraged across numerous areas such as natural language processing and computer vision. This attention module is comprised of three linear transformations, namely *query*, *key*, and *value* linear transformations, each of which has a bias term. In this work, we study the role of these bias terms, and mathematically show that the bias term of the *key* linear transformation is redundant and could be omitted without any impact on the attention module. Moreover, we argue that the bias term of the *value* linear transformation has a more prominent role than that of the bias term of the *query* linear transformation. We empirically verify these findings through multiple experiments on language modeling, natural language understanding, and natural language generation tasks.

Index Terms— Attention, Transformer, Softmax

1. INTRODUCTION

Attention mechanism has revolutionized the application of neural networks in numerous areas such as computer vision and natural language processing. Different mechanism of attention such as Additive Attention [**?**], Multiplicative Attention [**?**], and Key-Value Attention [**?**] have been introduced in the past. Among all different attention mechanisms, perhaps the one that is used most frequently is the Dot-Product Attention [**?**] that was introduced for transformers. Hereon, any mention of attention refers to this Dot-Product attention.

In the recent past, the attention module has been analyzed by various works, primarily in attempts to improve its squared complexity and aid its feasibility for long sequences [**?**]. Additionally, other works have analyzed potential redundancies in components like the multiple attention heads [**?**, **?**, **?**]. While these variants have shown promising evidence, the original transformer seems to be performing the best when compared in various scales of model sizes [**?**]. In this work, we analyze the attention module and attempt to dissect it to better understand its components.

Attention mechanism includes three linear transformations, namely *query*, *key*, and *value* transformations, which are affine transformations with respective bias terms. In this work, we study the role of these bias terms, and mathematically show that the bias term for the *key* linear transformation does not have any role in the attention function and could be omitted altogether. This result was also independently reported in [**?**]. We next verify this result numerically, and show that replacing these bias vectors with arbitrary and random vectors does not result in any significant difference[1] in the output of transformers. Another implication of this result is in BitFit [**?**] where only bias terms in a transformer-based language model are fine-tuned on downstream tasks for parameter-efficient training. We show that by freezing the *key* bias parameters in attention, we could reduce the number of trainable parameters in BitFit by over 11% with no impact on the performance of the final model on downstream tasks.

2. NOTATION

In attention mechanism, an input vector $\boldsymbol{h} \in \mathbb{R}^d$, attends to a set of n vectors (subject of attention) which are represented as columns of matrix $\boldsymbol{C} \in \mathbb{R}^{d\times n}$. Within the attention mechanism, first a query vector $\boldsymbol{q} \in \mathbb{R}^d$ is constructed based on $\boldsymbol{h}$ using a linear transformation, i.e., $\boldsymbol{q} = \boldsymbol{W}_q\boldsymbol{h} + \boldsymbol{b}_q$, where $\boldsymbol{W}_q \in \mathbb{R}^{d\times d}$ and $\boldsymbol{b}_q \in \mathbb{R}^d$ are the respective weight and bias parameters. Also, a set of key vectors that establish a key matrix $\boldsymbol{K} \in \mathbb{R}^{d\times n}$ are constructed by applying another linear transformation on $\boldsymbol{C}$, i.e., $\boldsymbol{K} = \boldsymbol{W}_k\boldsymbol{C} + \boldsymbol{b}_k\mathbb{1}^T$, where $\mathbb{1} \in \mathbb{R}^n$ is a vector of 1s. Next, score distribution between the query and the keys is created by applying softmax ($\sigma(.)$) on the product of these two linear transformations:

$$\sigma\left((\boldsymbol{W}_q\boldsymbol{h} + \boldsymbol{b}_q)^T\left(\boldsymbol{W}_k\boldsymbol{C} + \boldsymbol{b}_k\mathbb{1}^T\right)\right) = \sigma\left(\boldsymbol{q}^T\boldsymbol{K}\right),$$

which is referred to as attention distribution. On the other hand, similar to the process of creating the set of key vectors ($\boldsymbol{K}$), a set of value vectors, constituting matrix $\boldsymbol{V}$, are created by applying another linear transformation on $\boldsymbol{C}$, i.e., $\boldsymbol{V} = (\boldsymbol{W}_v\boldsymbol{C} + \boldsymbol{b}_v\mathbb{1}^T)$. Finally, attention is computed by multiplying the value vectors and the attention distribution:

$$\text{Attn}^{\boldsymbol{C}}(\boldsymbol{h}) = \boldsymbol{V}\sigma\left(\boldsymbol{q}^T\boldsymbol{K}\right)^T, \tag{1}$$

[1]The difference is non-zero due to numerical errors. See Section 4.1 for details.

which could be thought of as a convex combination of value vectors (columns of $\boldsymbol{V}$).[2]

3. DISSECTING ATTENTION

To better understand the inter-workings of attention, we take a deeper look into the interactions of the different elements that form the attention mechanism. First, we expand $\boldsymbol{V}$ in Equation (1):

$$\begin{aligned}\text{Attn}^{\boldsymbol{C}}(\boldsymbol{h}) &= \boldsymbol{V}\sigma\left(\boldsymbol{q}^T\boldsymbol{K}\right)^T \\ &= (\boldsymbol{W}_v\boldsymbol{h} + \boldsymbol{b}_v\mathbb{1}^T)\sigma\left(\boldsymbol{q}^T\boldsymbol{K}\right)^T \\ &= \boldsymbol{W}_v\boldsymbol{h}\sigma\left(\boldsymbol{q}^T\boldsymbol{K}\right)^T + \boldsymbol{b}_v\mathbb{1}^T\sigma\left(\boldsymbol{q}^T\boldsymbol{K}\right)^T. \qquad (2)\end{aligned}$$

Since $\boldsymbol{b}_v\mathbb{1}^T$ is a matrix with identical columns $\boldsymbol{b}_v$, any convex combination of its columns, including the one using $\sigma\left(\boldsymbol{q}^T\boldsymbol{K}\right)$ as weights, is essentially equal to $\boldsymbol{b}_v$, i.e.,

$$\boldsymbol{b}_v\mathbb{1}^T\sigma\left(\boldsymbol{q}^T\boldsymbol{K}\right)^T = \boldsymbol{b}_v.$$

Therefore, from Equation (2), we can write the attention function as,

$$\text{Attn}^{\boldsymbol{C}}(\boldsymbol{h}) = \boldsymbol{W}_v\boldsymbol{h}\sigma\left(\boldsymbol{q}^T\boldsymbol{K}\right)^T + \boldsymbol{b}_v. \qquad (3)$$

Next, we expand $\boldsymbol{K}$ in Equation (3):

$$\begin{aligned}\text{Attn}^{\boldsymbol{C}}(\boldsymbol{h}) &= \boldsymbol{W}_v\boldsymbol{h}\sigma\left(\boldsymbol{q}^T\boldsymbol{K}\right)^T + \boldsymbol{b}_v \\ &= \boldsymbol{W}_v\boldsymbol{h}\sigma\left(\boldsymbol{q}^T(\boldsymbol{W}_k\boldsymbol{C} + \boldsymbol{b}_k\mathbb{1}^T)\right)^T + \boldsymbol{b}_v \\ &= \boldsymbol{W}_v\boldsymbol{h}\sigma\left(\boldsymbol{q}^T\boldsymbol{W}_k\boldsymbol{C} + \boldsymbol{q}^T\boldsymbol{b}_k\mathbb{1}^T\right)^T + \boldsymbol{b}_v. \qquad (4)\end{aligned}$$

Using the fact that $\boldsymbol{b}_k\mathbb{1}^T$ is a matrix with identical columns, it follows that $\boldsymbol{q}^T\boldsymbol{b}_k\mathbb{1}^T$ is a vector with equal elements, where all elements are equal to $\boldsymbol{q}^T\boldsymbol{b}_k$. On the other hand, softmax is invariant under translation by the same value in each coordinate. In other words, $\forall\delta \in \mathbb{R}, \forall\boldsymbol{z} \in \mathbb{R}^n$, $\sigma(\boldsymbol{z} + \boldsymbol{\delta}) = \sigma(\boldsymbol{z})$, where $\boldsymbol{\delta} = \delta\mathbb{1}$. As a result, and from Equation (4) we can conclude that,

$$\text{Attn}^{\boldsymbol{C}}(\boldsymbol{h}) = \boldsymbol{W}_v\boldsymbol{h}\sigma\left(\boldsymbol{q}^T\boldsymbol{W}_k\boldsymbol{C}\right)^T + \boldsymbol{b}_v.$$

Next, in the equation above we replace $\boldsymbol{q}$ with its original linear transformation, resulting in,

$$\begin{aligned}&\text{Attn}^{\boldsymbol{C}}(\boldsymbol{h}) = \\ &\quad \boldsymbol{W}_v\boldsymbol{h}\sigma\left((\boldsymbol{W}_q\boldsymbol{h} + \boldsymbol{b}_q)^T\boldsymbol{W}_k\boldsymbol{C}\right)^T + \boldsymbol{b}_v. \qquad (5)\end{aligned}$$

This rewriting of attention highlights the different roles that $\boldsymbol{b}_q$, $\boldsymbol{b}_k$, and $\boldsymbol{b}_v$ play in the attention function. From Equation (5) it is clear that $\boldsymbol{b}_k$ plays **no role** in the attention function and it is in fact redundant. $\boldsymbol{b}_v$ on the other hand plays a very important role in attention since it is one of the two terms that are added to each other to constitute the attention function. Finally, $\boldsymbol{b}_q$ plays a role in creating the attention distribution along with other parameters of the attention namely, $\boldsymbol{W}_q$ and $\boldsymbol{W}_K$.

[2]Note that we omit writing the scaling factor $\frac{1}{\sqrt{d}}$ employed on the dot-product, for brevity.

4. NUMERICAL ANALYSIS

In Section 3, we mathematically show that the bias term of the *key* linear transformation within the attention mechanism, i.e., $\boldsymbol{b}_k$, is redundant in the attention function and can be removed. In this section, we verify these results numerically. We also discuss some computational gains that could be achieved due to this result.

4.1. Changing $\boldsymbol{b}_k$ in Pre-Trained Language Models

We examine the sensitivity of transformer-based pre-trained language models to changes to attention bias terms, i.e., $\boldsymbol{b}_k$, $\boldsymbol{b}_q$, and $\boldsymbol{b}_v$, for all attention modules within the model. The idea behind this analysis is that since we show that $\boldsymbol{b}_k$ is redundant in theory, changing its pre-trained values to arbitrary values should not impact the output of the models in practice. On the other hand, for $\boldsymbol{b}_q$ and $\boldsymbol{b}_v$, which are not redundant in the attention function, changing their values should result in significant changes in model outputs.

For this analysis, we take the first 100 sentences from the Wikipedia page for "Machine Learning"[3]. The length of these sentences ranges from 1 ("Overview") to 75 tokens (counted using spaCy[4]). Next, we feed these sentences to several pre-trained language models and get their last layer's hidden state for each sentence. We represent this hidden state matrix for sentence i as $\boldsymbol{H}_i \in \mathbb{R}^{d\times s_i}$, where s_i is the length of sentence i. We also feed these sentences to the same pre-trained language models but with changes applied to their $\boldsymbol{b}_k$, $\boldsymbol{b}_q$, or $\boldsymbol{b}_v$ (details are forthcoming) and represent the corresponding hidden state matrix as $\boldsymbol{H}'_i$, which is also in $\mathbb{R}^{d\times s_i}$. We then compare $\boldsymbol{H}_i$ and $\boldsymbol{H}'_i$ across all is for each of the models:

$$x^\star := \inf\left\{x \in \mathbb{Z} \mid \max_i \|\boldsymbol{H}_i - \boldsymbol{H}'_i\|_{\text{max}} \leq 10^x\right\},$$

where $\|.\|_{\text{max}}$ is matrix max norm [5], $\mathbb{Z}$ is the set of all integers, and $i \in \{1, \ldots, 100\}$. In other words, the *tolerance level* (i.e., $10^{x^\star}$) at which $\boldsymbol{H}_i$ and $\boldsymbol{H}'_i$ are equal across all sentences is calculated.

We run this analysis for base and large sizes of both RoBERTa [?] and BART [?] using Huggingface. We set the values of the elements of bias vectors $\boldsymbol{b}_k$, $\boldsymbol{b}_q$, or $\boldsymbol{b}_v$ to 0, 1, 10, and random numbers uniformly sampled from $[-5, 5]$. In other words, $\boldsymbol{b}_k$, $\boldsymbol{b}_q$, or $\boldsymbol{b}_v$ is set to a vector of zeros, ones,

[3]https://en.wikipedia.org/wiki/Machine˙learning
[4]`en_core_web_sm` package. https://spacy.io/
[5]https://en.wikipedia.org/wiki/Matrix˙norm#Max˙norm

	$\boldsymbol{b}_k$				$\boldsymbol{b}_q$				$\boldsymbol{b}_v$			
	0	**1**	**10**	**[-5, 5]**	**0**	**1**	**10**	**[-5, 5]**	**0**	**1**	**10**	**[-5, 5]**
RoBERTa-base	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10	10	10^2	10^2	10	10	10^2	10^2
RoBERTa-large	10^{-5}	10^{-5}	10^{-5}	10^{-5}	1	1	1	1	1	1	10	10
BART-base	10^{-5}	10^{-5}	10^{-5}	10^{-5}	10	1	10	10	1	10	10	10
BART-large	10^{-5}	10^{-5}	10^{-5}	10^{-5}	10	1	10	10	1	10	10	10

Table 1. Tolerance level at which the final hidden state of the models before and after changing values of attention bias terms $\boldsymbol{b}_k$, $\boldsymbol{b}_q$, or $\boldsymbol{b}_v$ for 100 sentences are equal. The elements of bias vectors are set to one of 0 (equivalent to removing), 1, 10, or a random value between -5 and 5. The models are not sensitive to even drastic changes to $\boldsymbol{b}_k$.

tens, or a random vector. This is done for all attention modules (e.g., both self- and cross-attentions in BART) within the models. A small python script for this is shown in the Appendix. The tolerance levels ($10^{x^\star}$) for this analysis are reported in Table 1. For both BART-base and -large, as well as for RoBERTa-large, we see the models are not sensitive to the values of $\boldsymbol{b}_k$ at 10^{-5} tolerance level. For RoBERTa-base this tolerance level is 10^{-4}. On the other hand, for the other two attention bias terms $\boldsymbol{b}_q$ and $\boldsymbol{b}_v$, the models are very sensitive to changes to these bias terms. For instance, for RoBERTa-base, changing the elements of $\boldsymbol{b}_q$ or $\boldsymbol{b}_v$ to random values in $[-5, 5]$, results in the elements of the final hidden states to change up to 100 in value. This study shows that $\boldsymbol{b}_k$ does not have any significant impact on the output of the models.

From the conclusion of Section 3 that $\boldsymbol{b}_k$ is redundant in the computations of attention, one might expect that the tolerance levels reported in Table 1 under $\boldsymbol{b}_k$ should be much lower. This discrepancy is simply due to numerical errors associated with calculating softmax within the attention function. For example, in theory, softmax of $[0.1, 0.2]$ is equal to the softmax of $[5.1, 5.2]$, since softmax is invariant under translation by the same value in each coordinate. However, numerically this equality only holds at 10^{-9} tolerance level[6]. These errors propagated across hundreds of dimensions (instead of 2 in this example), and numerous transformer layers would lead to tolerance levels of 10^{-4} and 10^{-5} that are reported in Table 1.

We conduct a similar experiment on text classification task and measure how changes in $\boldsymbol{b}_k$, $\boldsymbol{b}_q$, and $\boldsymbol{b}_v$ impact the accuracy of models. For this purpose for three GLUE [**?**] tasks, namely SST-2, MNLI, and QQP, we take pre-trained RoBERTa-large based models and change the values of $\boldsymbol{b}_k$, $\boldsymbol{b}_q$; and $\boldsymbol{b}_v$ to random values uniformly sampled from $[-5, 5]$; we report accuracy of models on the validation set before and after these changes in Table 2. From the numbers, it is immediately clear that applying these drastic changes to $\boldsymbol{b}_k$ results in no change in the accuracy of the models. On the other hand these changes to $\boldsymbol{b}_q$ and $\boldsymbol{b}_v$ result in very large degradation in the performance of the models. It is also evident that changes in $\boldsymbol{b}_v$ result in much larger degradation than changes in $\boldsymbol{b}_q$, which supports the evidence in Section 3 about role of $\boldsymbol{b}_v$.

[6]Calculated using the implementation of softmax in the nn module of PyTorch version 1.10.0

		[-5,5]		
	Original	$\boldsymbol{b}_k$	$\boldsymbol{b}_q$	$\boldsymbol{b}_v$
SST-2[7]	0.9644	0.9644	0.7007	0.4908
MNLI[8]	0.9060	0.9060	0.4101	0.3182
QQP[9]	0.9214	0.9214	0.6434	0.3682

Table 2. Accuracy of GLUE [**?**] tasks as the values of $\boldsymbol{b}_k$, $\boldsymbol{b}_q$, and $\boldsymbol{b}_v$ of trained models, which are based on RoBERTa-Large, are set to uniform random values between -5 and 5.

	GPT-2	**RoBERTa-base**
Original	2.9251	5.8890
No $\boldsymbol{b}_k$	2.9250	5.8909

Table 3. Average loss (on test set) of GPT-2 (small) and RoBERTa-base compared to their variants without $\boldsymbol{b}_k$, trained from scratch. Numbers are average over 3 runs with different random seed, and no statistically significant difference is observed.

4.2. Pre-Training of Language Models without $\boldsymbol{b}_k$

We train two transformer-based language models, GPT-2 [**?**] and RoBERTa, from scratch both with and without $\boldsymbol{b}_k$, and compare the language modeling performance of the model on a test set. We use Huggingface with the original hyper-parameters for training these models. Both of these models are trained on wikitext-103-v1 [**?**] dataset. We train each of the GPT-2 (small) and RoBERTa (base) models from scratch with three different random seeds for 50,000 steps, and we report the loss of final model on the test set averaged over three runs in Table 3. Note that there is no statistically significant difference between the two settings at a p-value ≤ 0.05 in an unpaired two-tailed T-test.

[7]https://huggingface.co/philschmid/roberta-large-sst2
[8]https://huggingface.co/roberta-large-mnli
[9]https://huggingface.co/howey/roberta-large-qqp

		Average Rouge			
		R-1	R-2	R-L	R-Sum
BART-large	Orig.	40.66	17.32	32.25	32.25
	No $\boldsymbol{b}_k$	40.67	17.33	32.26	32.26

Table 4. BitFit on BART-large with/without $\boldsymbol{b}_k$ on XSUM. Second row has 11.11% less trainable parameters than the first row. No statistically significant difference in the evaluation set accuracy according to a two-tailed T-test at p-value 0.05.

4.3. BitFit without $\boldsymbol{b}_k$

One place where removing the redundant $\boldsymbol{b}_k$ could result in significant savings in computation is in BitFit [?], where a pre-trained transformer based language model is fine-tuned for downstream tasks such as text classification, summarization, etc., by freezing all the trainable parameters of the model, except for bias terms within different modules of the model. This is further discussed in details in Section 4.3.

Next, we study the effect of freezing $\boldsymbol{b}_k$ vectors across all transformer layers in BitFit. Normally in BitFit, $\boldsymbol{b}_k$ vectors are among the fine-tuned parameters, but since we show that $\boldsymbol{b}_k$ is redundant in the attention function, we study what happens if these vectors are not fine-tuned in BitFit. In this section all models are fine-tuned using the exact hyper-parameters used in [?] for the corresponding models. Table 4 shows the results of BitFit for summarization task on the XSUM [?] dataset, using BART-large with and without fine-tuning $\boldsymbol{b}_k$. Freezing $\boldsymbol{b}_k$ in BitFit results in 11.1% decrease in the number of trainable parameters. Each model is fine-tuned three times with different random seeds. The reported numbers in Table 4 are different rouge metrics averaged over the three runs. According to a two-tailed T-test there is no statistically significant difference between the rouge metrics at p-value ≤ 0.05.

We conduct a similar experiment for a text classification problem, namely SST-2 from the GLUE benchmark using RoBERTa. The main difference between this and the summarization setting is the new classification layers that need to be added to RoBERTa for performing classification, whose parameters are fine-tuned along with the bias parameters in BitFit. As a result, the savings in the number of trainable parameters by freezing $\boldsymbol{b}_k$ in this setting is smaller than the summarization setting with BART. For RoBERTa-base and RoBERTa-large freezing $\boldsymbol{b}_k$ during fine-tuning using BitFit results in 1.3% and 1.9% savings in trainable parameters, respectively. Table 5 shows the average accuracy (over five runs with different random seeds) of the fine-tuned models on the evaluation set for SST-2. According to a two-tailed T-test, at p-value ≤ 0.05 there is no statistically significant difference between the BitFit results with and without $\boldsymbol{b}_k$ for both base and large sizes of RoBERTa.

		Eval. Accuracy
RoBERTa-base	Original	94.98
	No $\boldsymbol{b}_k$	94.92
RoBERTa-large	Original	95.83
	No $\boldsymbol{b}_k$	95.85

Table 5. Average accuracy over 5 runs of BitFit on RoBERTa-base and -large with and without $\boldsymbol{b}_k$ on SST-2. No statistically significant difference in accuracy is observed according to a two-tailed T-test at p-value 0.05.

5. IMPLICATIONS FOR TRANSFORMERS

As was shown in the previous sections, the bias term of the *key* linear transformation, i.e., $\boldsymbol{b}_k$ in the attention function is redundant. In the context of transformers, if we consider one transformer layer, $\boldsymbol{b}_k$ constitutes only less than 0.01% of the parameters of the layer. As a result, removing $\boldsymbol{b}_k$ from the transformer architecture both during training or even from a pre-trained model at inference time does not result in significant savings in computations. However, for the following reasons we argue that this finding is important. (1) The small size of these redundant parameters within one of the most widely used neural networks architectures does not change the fact that they are redundant, and we argue that this redundancy, however small, should be addressed. (2) It is important to note that this small redundancy appears in thousands of transformer-based models that are being invoked millions of times a day across academia and different industries within different products. From this aggregate perspective this redundancy results in significant redundant computational operations that could be avoided. (3) Recent works such as $(IA)^3$ [?] show how a small set of parameters (of the same size as b_k) could be used for efficiently adapting large language models to downstream tasks (often rivaling fully fine-tuned variants). From this angle, the redundant b_k parameters could be repurposed to improve the performance of models. (4) Finally, in some recent works the bias terms of "dense kernels" and "layer norms" are dropped from the architecture based on the observation of positive impact on stability of the training process. Our analysis reveals that from a theoretical standpoint some additional biases (b_k) could be dropped from these architectures as well.

6. CONCLUSIONS

In this work, we analyze the attention function predominant in present-day transformer architectures and find that the biases in the linear transformations of the attention function play different roles. Our analysis reveals that $\boldsymbol{b}_v$ is an important component in the attention computation, whereas the bias of the key linear transformation, $\boldsymbol{b}_k$, is completely redundant. We also numerically confirm that removing $\boldsymbol{b}_k$ does not significantly change the outcome of transformers.

While our analysis has been focused on the softmax-based (scaled) dot-product attention, recent works have demonstrated how attention can be generalized from the kernel lens. This has led to innovations in different kernel designs that perform equivalently or better and it would be interesting to explore the role of bias terms in the proposed kernels.