Multiplicative Position-aware Transformer Models for Language Understanding

Anonymous ACL submission

Abstract

In order to utilize positional ordering information in transformer models, various flavors of absolute and relative position embeddings have been proposed. However, there is no comprehensive comparison of position embedding methods in the literature. In this paper, we review existing position embedding methods and compare their accuracy on downstream NLP tasks, using our own implementations. We also propose a novel multiplicative embedding method which leads to superior accuracy when compared to existing methods. Finally, we show that our proposed embedding method, served as a drop-in replacement of the default absolute position embedding, can improve the RoBERTa-base and RoBERTa-large models on SQuAD1.1 and SQuAD2.0 datasets.

1 Introduction

The BERT (Devlin et al., 2018) and its variants RoBERTa (Liu et al., 2019), XLNet (Yang et al., 2019), ALBERT (Lan et al., 2019) and T5 (Raffel et al., 2019) have led to new state-of-the-art results for various NLP tasks. The backbone of BERT variants is a transformer model Vaswani et al. (2017), which uses absolute position embedding to capture word position information. The relative position embedding was initially proposed in Shaw et al. (2018); Huang et al. (2018). The work of Shaw et al. (2018) was modified and adopted in (Dai et al., 2019; Yang et al., 2019; Raffel et al., 2019; Song et al., 2020). More recent work on improving position embeddings can be found at Huang et al. (2020); He et al. (2020); Ke et al. (2020); Dufter et al. (2021).

Despite the proposal of various position embedding methods, a systematic comparison of these methods is missing in literature. Additionally, it is uncertain if simply replacing the default absolute position embedding with a relative position embedding method can improve an already strong model (e.g., RoBERTa). We attempt to answer these two questions in this paper. Our major contributions are three fold. 1) We implement[1] existing major position embedding methods including the absolute position embedding (Devlin et al., 2018) and existing relative position embeddings (Shaw et al., 2018; Raffel et al., 2019; Huang et al., 2020; He et al., 2020; Ke et al., 2020). We compare these methods in terms of their computational complexity and accuracy on GLUE and SQuAD datasets. 2) Inspired by Huang et al. (2020), we propose a novel multiplicative embedding method which leads to superior accuracy. 3) We show that our novel position embedding, when compared to the default absolute position embedding, can improve RoBERTa-base and RoBERTa-large models performance on the SQuAD1.1 and SQuAD2.0 datasets.

2 Position Embedding Methods

The attention weight from position j to i, e_{ij}, is defined in (Vaswani et al., 2017) as follows:

$$e_{ij} = \frac{(x_i W^Q)(x_j W^K)^T}{\sqrt{d_z}}, \quad (1)$$

where i, j represent the positions which are in the range of maximum sequence length, d_x and d_z are model input and output dimensions (identical in our case). $x_i, x_j \in \mathbb{R}^{d_x}$. The scaling factor, $\sqrt{d_z}$, is necessary to make the training stable. $W^Q, W^K \in \mathbb{R}^{d_x \times d_z}$ are parameter matrices. In this section, we review the absolute position embedding used in the original BERT paper and various relative position embedding methods. Due to space limit, we include all methods' complexity analysis at Appendix A.

2.1 Absolute position embedding in BERT

In the self-attention scheme, the absolute position embedding is defined as follows.

$$x_i = t_i + s_i + w_i, \quad (2)$$

[1]We will release the code soon.

where x_i, $i \in \{0, \ldots, n-1\}$ is the input embedding to the first transformer layer and t_i, s_i and $w_i \in \mathbb{R}^{d_x}$ are the token embeddings, segment embeddings and absolute position embeddings, respectively. Segment embedding indicates if a token is sentence A or sentence B, which was originally introduced in BERT (Devlin et al., 2018) to compute the next sentence prediction (NSP) loss. We drop the segment embedding in this paper as recent work (Yang et al., 2019; Liu et al., 2019; Raffel et al., 2019) suggested that the NSP loss does not help improve accuracy.

2.2 Shaw's relative position embedding

Edge representations, $a_{ij} \in \mathbb{R}^{d_z}$, are proposed in Shaw et al. (2018) to model how token t_i attends to token t_j. Equation (1) can be revised as follows to consider the distance between token i and token j when computing their attention.

$$e_{ij} = \frac{(x_i W^Q)(x_j W^K + a_{ij})^T}{\sqrt{d_z}}. \quad (3)$$

The authors also introduced a clipped value k which is the maximum relative position distance allowed. The authors hypothesized that the precise relative position information is not useful beyond a certain distance.

2.3 Raffel's relative position embedding

In Text-to-Text Transfer Transformer (T5), Raffel et al. (2019) uses a simplified position embedding where each embedding is a scalar that is added to the corresponding logit used for computing the attention weights. Specifically, the attention weight is defined as

$$e_{ij} = \frac{(x_i W^Q)(x_j W^K)^T + a_{ij}}{\sqrt{d_z}}. \quad (4)$$

where $a_{ij} \in \mathbb{R}$ are scalars used to represent how token i attends to j.

2.4 Huang's relative position embedding

Four relative embedding methods have been proposed in (Huang et al., 2020). Two methods among those four, **M2** and **M4**, are selected in comparison. Similar to Raffel's method, M2 uses scalars to model relative distance and thus have an identical number of parameters. However, M2 performs multiplication instead of the addition, as in Raffel's method, when computing the attention weights. Specifically, the attention weight defined in M2 can be written as follows.

$$e_{ij} = \frac{(x_i W^Q)(x_j W^K)^T \times a_{ij}}{\sqrt{d_z}}. \quad (5)$$

Huang's M4 method extends Shaw's method to include the dot product of all possible pairs of query, key, and relative position embeddings.

$$e_{ij} = \frac{(x_i W^Q) \cdot (x_j W^K) + (x_i W^Q) \cdot a_{ij} + (x_j W^K) \cdot a_{ij}}{\sqrt{d_z}}. \quad (6)$$

Three factors in the numerator model the interaction of query and key, query and relative position embedding, and key and relative position embedding, respectively.

2.5 DeBERTa relative position embedding

Recently, DeBERTa (He et al., 2020) proposed disentangled attention mechanism, where each word is represented using two vectors that encode its content and position, respectively.

$$e_{ij} = \frac{(x_i W^Q) \cdot (x_j W^K) + (x_i W^Q) \cdot (a_{ij} W^R) + (x_j W^K) \cdot (a_{ij} W^T)}{\sqrt{3d_z}}, \quad (7)$$

where $W^R, W^T \in \mathbb{R}^{d_x \times d_z}$ are additional linear projection matrices. DeBERTa's embedding method is similar to Huang's M4, with the following differences: 1) DeBERTa introduced projection matrices W^R and W^T. 2) DeBERTa has *different* relative embeddings across different heads, and 3) DeBERTa used a different scaling factor $\sqrt{3d_z}$, instead of the default $\sqrt{d_z}$, in the denominator.

2.6 Transformer with untied positional encoding (TUPE)

The attention weight definition in TUPE (Ke et al., 2020) , which consists of both absolute and relative embeddings, is listed as follows.

$$e_{ij} = \frac{(x_i W^Q) \cdot (x_j W^K) + (p_i U^Q) \cdot (p_j U^K)}{\sqrt{2d_z}} + a_{ij}, \quad (8)$$

where p_i and p_j are the absolute position embeddings, and $U^Q, U^K \in \mathbb{R}^{d_z \times d_z}$ are the linear projection matrices for absolute position embeddings. a_{ij} is used to model relative position embeddings, which is identical to Raffel's relative embeddings. Furthermore, the authors argued that the [CLS] token is a virtual token and it should not be applied in the same way to the normal tokens:

$$\begin{aligned} e_{ij} = &\frac{(x_i W^Q) \cdot (x_j W^K)}{\sqrt{2d_z}} \\ &+ \text{reset}_\theta\left(\frac{(p_i U^Q) \cdot (p_j U^K)}{\sqrt{2d_z}} + a_{ij}, i, j\right), \end{aligned} \quad (9)$$

where reset_θ is defined as the following, in which θ_1 is used as the attention weight from [CLS] token to others, and θ_2 is used from others to [CLS].

$$\text{reset}_\theta(v_{ij}, i, j) = \begin{cases} v_{ij} & i \neq 1, j \neq 1 \\ \theta_1 & i = 1 \\ \theta_2 & i \neq 1, j = 1 \end{cases} \quad (10)$$

2.7 Proposed M4 multiplicative (M4M) method

The M2 method proposed in (Huang et al., 2020) essentially replaces the additive operator with the multiplicative operator in Raffel's method, which improves downstream application accuracy as we will see in the experiment section 3.1. It is intuitive to apply the multiplicative operator to scale the content attention with respect to the relative positional attention. Inspired by this intuition, we propose to replace the additive operator with multiplicative operator in M4 method (equation 6) to the following.

$$e_{ij} = \frac{(x_i W^Q) \cdot (x_j W^K) \times (x_i W^Q) \cdot a_{ij} \times (x_j W^K) \cdot a_{ij}}{\sqrt{d_z}}. \quad (11)$$

We denote this method as M4 modified, M4M, which has exactly the same amount of parameters and computational complexity as M4.

3 Experiments

We start with a small-scale pretraining setup so we can afford to compare different embedding methods. We use training data of BooksCorpus (Zhu et al., 2015) and English Wikipedia (Wikipedia contributors, 2004; Devlin et al., 2018) (15G in total), which is the dataset used to pretrain the original BERT model (Devlin et al., 2018). We use the RoBERTa-base setting and set the maximum input length to 512. The model updates use a batch size of 96 and Adam optimizer with learning rate starting at 1e-4. This batch size was chosen to fit as many samples as possible. We train each model on one AWS P4DN instance (each containing 8 A100-SXM4-40GB GPUs) with maximum steps of 500000 (around 5.85 epochs). Each pretraining run takes approximately 30 hours. We evaluate on GLUE (Wang et al., 2018), SQuAD1.1 and SQuAD2.0) (Rajpurkar et al., 2016). We show the main experiment results in this section, with additional experiments listed in Appendices.

3.1 Position embedding methods comparison

We train RoBERTa-base models with our implementation of different position embedding methods and list the results on MNLI, SST-2 and SQuAD 1.1 datasets in Table 1. Absolute is the default absolute position embedding which results in the F1 score of 87.08 on the SQuAD1.1 dataset. The lower F1 score when compared to the BERT (Devlin et al., 2018) baseline of 88.5 can be attributed to under-training (6 epochs here vs 40 epochs in BERT). Shaw's method offers accuracy boost over the absolute embedding on all three datasets. For example, it improves the F1 from 87.08 to 88.62 on SQuAD1.1 dev dataset. M2 uses fewer parameters (12K) compared to Shaw's method (785K) and it results in a lower accuracy. M4 has the same number of parameters as Shaw's method. It outperforms Shaw's method on MNLI and SQuAD1.1 but underperforms on the SST-2 dataset. DeBERTa has mixed result when compared to M4. The proposed M4M method leads to accuracy improvement for all three datasets when compared to M4. Specifically, it achieves the highest accuracy (91.62) on the SST-2 dataset and F1 score (89.45) on the SQuAD1.1 among all methods.

Model	MNLI-m	SST-2	SQuAD1.1
Absolute	81.25	90.25	87.08
Shaw	82.88	91.28	88.62
M2	82.41	90.36	87.26
M4	83.05	91.05	89.36
DeBERTa	**83.67**	90.71	88.84
M4M	83.58	**91.62**	**89.45**
M4+Reset	82.30	91.39	88.43
ABS+M4M	83.19	91.16	88.68

Table 1: Accuracy of different position embeddings for RoBERTa-base.

We also implemented Raffel's method and observed that the training loss of Raffel is similar to other methods. However, the fine-tuning diverges on GLUE and SQuAD1.1 datasets if the same learning rate is used. TUPE (Ke et al., 2020) also uses Raffel's relative position embedding and it suffers from the same issue. We thus exclude Raffel and TUPE results in Table 1. In addition, we implemented two hybrid methods: M4+Reset and ABS+M4M. M4+Reset applies the reset equation (10) to M4 method. We do not find the reset help improve the accuracy. Similarly, the combination of absolute and relative position (ABS+M4M) does not lead to accuracy boost.

3.2 RoBERTa-base model with M4M method

We apply the best method, M4M, in a larger scale training setup. We now use the training data of

Model	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k
RoBERTa	87.32/87.31	**88.33**	92.84	93.92	**55.33**	**90.33**	91.42	71.84
RoBERTa-ABS	87.35/86.99	88.29	92.78	93.92	52.77	89.07	89.67	**72.92**
RoBERTa-M4M	**87.82/87.59**	88.28	**92.98**	**94.26**	47.02	89.45	**91.63**	68.59

Table 2: GLUE accuracy for RoBERTa, RoBERTa-ABS, and RoBERTa-M4M models. The number below each task denotes the number of training examples. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for other tasks.

BooksCorpus, English Wikipedia (16G) and OpenWeb Text (Gokaslan and Cohen., 2019) (38G). The CommonCrawl News dataset (Nagel., 2016) (76G) and STORIES (Trinh and Le., 2018) (31G) datasets, which were used in RoBERTa pre-train, are not publicly available so were not included. We compare three models in our experiments, *RoBERTa*, *RoBERTa-ABS*, and *RoBERTa-M4M*. RoBERTa is the official RoBERTa model[2]. RoBERTa-ABS and RoBERTa-M4M are the models pre-trained with absolute position embedding and M4M respectively. Both are initialized from RoBERTa. We use a batch size of 480 and learning rate starting from 1e-4. We train each model with maximum steps of 500000 (approximately 8.8 epochs). Each pretraining takes around 5 days.

Following Devlin et al. (2018), we use a batch size of 32 and 3-epochs of fine-tuning for each dataset in GLUE. We report the best accuracy on the development dataset with learning rates 2e-5, 3e-5 and 4e-5. Table 2 shows the results of GLUE datasets. As we use less data in pretraining (54G vs 161G), RoBERT-ABS underperforms the official RoBERTa model on 6 out 8 datasets. RoBERTa-M4M, however, is able to match the performance of the official RoBERTa model. It outperforms RoBERTa on the MNLI, QNLI, SST-2 and MRPC datasets but underperforms on rest four datasets.

For SQuAD1.1 and SQuAD2.0 datasets, Table 3 shows that RoBERTa-ABS boosts RoBERTa performance slightly. RoBERTa-M4M results in an even greater accuracy boost. It leads to the highest accuracies among three models on both the SQuAD1.1 and SQuAD2.0 datasets. Finally, we train a RoBERTa-M4M (denoted as RoBERTa-M4M (C4 en) in Table 3) on 800G C4-en dataset[3]. RoBERTa-M4M (C4 en) leads to additional accuracy gain when compared to RoBERTa-M4M. It reaches the F1 scores of 92.64 and 84.51 on SQuAD1.1 and SQuAD2.0 datasets respectively.

Model	SQuAD1.1 EM	SQuAD1.1 F1	SQuAD2.0 EM	SQuAD2.0 F1
RoBERTa	85.95	92.12	79.99	83.10
RoBERTa-ABS	86.10	92.31	80.67	83.69
RoBERTa-M4M	**86.44**	**92.52**	**80.88**	**84.00**
RoBERTa-M4M (C4 en)	**86.54**	**92.64**	**81.65**	**84.51**

Table 3: SQuAD1.1 and SQuAD2.0 accuracy for various base models.

3.3 RoBERTa-large model with M4M method

Similar to Section 3.2, we now apply M4M method to RoBERTa-large models. We use the training data of BooksCorpus plus English Wikipedia (16G) and OpenWeb Text (38G) to train a model denoted as *RoBERTa-M4M Large*, which is initialized from RoBERTa Large[4]. We use the batch size of 192, learning rate starting from 1e-4, and maximum steps of 500000. Table 4 shows that RoBERTa-M4M Large can improve RoBERTa Large model. Specifically, it improves from F1 score of 94.63 to 94.78 on SQuAD1.1 dataset, and from 87.62 to 88.34 on SQuAD2.0 dataset.

Model	SQuAD1.1 EM	SQuAD1.1 F1	SQuAD2.0 EM	SQuAD2.0 F1
RoBERTa Large	89.13	94.63	84.66	87.62
RoBERTa-M4M Large	**89.21**	**94.78**	**85.47**	**88.34**

Table 4: SQuAD1.1 and SQuAD2.0 accuracy for RoBERTa Large and RoBERTa-M4M Large models.

4 Conclusion

We compared existing position embedding methods. We proposed a novel multiplicative position embedding method which can improve both RoBERTa-base and RoBERTa-large models on SQuAD1.1 and SQuAD2.0 datasets.

[2] *roberta-base* downloaded from https://github.com/huggingface/transformers.

[3] https://huggingface.co/datasets/allenai/c4/tree/main.

[4] *roberta-large* downloaded from https://github.com/huggingface/transformers.

References

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv:1901.02860*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*.

Philipp Dufter, Martin Schmitt, and Hinrich Schütze. 2021. Position information in transformers: An overview. *arXiv:2102.11090*.

Aaron Gokaslan and Vanya Cohen. 2019. Openwebtext corpus.. *http://Skylion007.github.io/OpenWebTextCorpus/*.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv:2006.03654*.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Curtis Hawthorne, AndrewM-Dai, Matthew D Hoffman, and Douglas Eck. 2018. An improved relative self-attention mechanism for transformer with application to music generation. *arXiv:1809.04281*.

Zhiheng Huang, Davis Liang, Peng Xu, and Bing Xiang. 2020. Improve transformer models with better relative position embeddings. *findings of EMNLP 2020, arXiv:2009.13658*.

Guolin Ke, Di He, and Tie-Yan Liu. 2020. Rethinking positional encoding in language pre-training. *arXiv:2006.15595*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv:1909.11942*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv:1907.11692*.

Sebastian Nagel. 2016. Cc-news. *https://commoncrawl.org/the-data/get-started/*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv:1910.10683*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswanii. 2018. Self-attention with relative position representations. *arXiv:1803.02155*.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. Mpnet: Masked and permuted pre-training for language understanding. *arXiv:2004.09297*.

Trieu H Trinh and Quoc V Le. 2018. A simple method for commonsense reasoning. *arXiv:1806.02847*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv:1706.03762*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. A multi-task benchmark and analysis platform for natural language understanding. *2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*.

Wikipedia contributors. 2004. Plagiarism — Wikipedia, the free encyclopedia. [Online; accessed 22-July-2004].

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv:1906.08237*.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *IEEE international conference on computer vision*.

A Appendix: Complexity Analysis

We analyze the storage complexity of various position embedding methods in this section. For a transformer model with m layers, h attention heads per layer, and maximum sequence length of n, table 5 lists the parameter size for various position embeddings and the runtime storage complexity. In order to have sufficient expressive power, we allow different embedding parameters at different layers for all methods except absolute position embedding[5]. For example, Shaw's method introduces the following parameters size: $m(2n-1)d/h = 12*(2*512-1)*768/12 = 785K$. It has runtime storage complexity of $\mathcal{O}(mn^2d/h)$.

All position embedding methods introduce a small number of additional parameters to the BERT model. The maximum parameter size is 834K from DeBERTa. It is negligible when compared to the

[5]To be compatible to the original BERT implementation.

Method	Parameters	Parameter (K)	Complexity
Absolute	nd	393	$\mathcal{O}(nd)$
Shaw	$m(2n-1)d/h$	785	$\mathcal{O}(mn^2d/h)$
Raffel	$m(2n-1)$	12	$\mathcal{O}(mn^2)$
M2	$m(2n-1)$	12	$\mathcal{O}(mn^2)$
M4	$m(2n-1)d/h$	785	$\mathcal{O}(mn^2d/h)$
DeBERTa	$m(2n-1)d/h + m(d/h)^2$	834	$\mathcal{O}(mn^2d/h)$
TUPE	$mnd/h + m(d/h)^2 + m(2n-1)$	454	$\mathcal{O}(mn^2d/h)$
M4M	$m(2n-1)d/h$	785	$\mathcal{O}(mn^2d/h)$

Table 5: Parameter sizes and runtime storage complexities of various position embedding methods.

number of parameters in RoBERTa-base (110M parameters). Other methods introduce even fewer parameters. For example, Raffel and M2 only have 12K additional parameters. In terms of training and inference speed, all methods are similar to the absolute position embedding baseline. Note that since TUPE requires different treatment to [CLS] positions, the self-attention along batch dimension cannot be efficiently computed with broadcasting, thus slowing down the training speed.

B Appendix: Unsuccessful baselines

We first train a RoBERTa-base model without any position embedding information. Specifically, the input is the sum of the token id embedding and token type id embedding with no absolute position embedding. We find that the model pre-training does not converge. The training loss ended up being around 5.8[6] which does not result in good accuracy on downstream applications.

We use the full-sentences setting in Liu et al. (2019). Each input is packed with full sentences sampled contiguously from one or more documents, such that the total length is at most 512 tokens. With the absolute position embedding, each token has a position in the range of 1 to 512. Due to sentence packing, the beginnings of sentences can be associated with any positions. While this absolute position embedding leads to a reasonable baseline as we will see in Section 3.1, we train a RoBERTa model with the *real* absolute positions. That is, each token is associated with its real position in the sentence which it belongs to. With this setup, the first token of a sentence always has the position of 1. We hypothesize that the real absolute position may be useful to capture certain syntactic information in model training. Surprisingly, we find that the model pre-training does not converge; the training loss remains as around 4.3 after the pre-training.

[6]Normally a RoBERTa-base model reaches the training loss of 1.4.

C Appendix: The effect of scaling factor

In original transformer models, the scaling factor of $\sqrt{d_z}$ was used in equation (1) to make training stable. Different scaling factors have been introduced in the self-attention equations, for example, $\sqrt{3d_z}$ is used in equation (7) for DeBERTa and $\sqrt{2d_z}$ is used in equation (9) for TUPE. The effect of scaling has not been evaluated in any previous study. In this section, We vary the scaling factors to the M4 methods and report the downstream task accuracy in Table 6. The results show that different scaling factors do not make a significant accuracy difference on the MNLI, SST-2 and SQuAD1.1 datasets.

Scaling Factor	MNLI-m	SST-2	SQuAD1.1
1	83.05	91.05	**89.36**
2	82.80	**91.97**	88.96
3	**82.86**	91.51	89.20
4	82.58	91.28	**89.36**
6	82.71	90.71	88.95
9	82.34	90.36	88.65

Table 6: Accuracy of M4 method with different scaling factors.

D Appendix: The effect of sharing parameters

In our implementation of various relative position embedding methods, we share the parameters across different heads for each layer. That is why we see the term of d/h instead of d in the *parameters* column in Table 5 for Shaw, M4, DeBERTa, TUPE and M4M methods. Some methods, for example DeBERTa (He et al., 2020), allow different relative embeddings across different heads. In this

section, we train two RoBERTa base models with the M4 method, one with parameter sharing and one without. Table 7 shows their accuracy on the MNLI, SST-2 and SQuAD1.1 datasets. As we can see, the parameter sharing performs better on two out three datasets.

Parameter Sharing	MNLI-m	SST-2	SQuAD1.1
Yes	**83.05**	91.05	**89.36**
No	82.88	**91.17**	88.50

Table 7: Accuracy of M4 method with and without parameter sharing.