DeBERTa: Decoding-enhanced BERT with Disentangled Attention

Anonymous authors
Paper under double-blind review

Abstract

Recent progress in pre-trained neural language models has significantly improved the performance of many natural language processing (NLP) tasks. In this paper we propose a new model architecture **DeBERTa** (**D**ecoding-**e**nhanced **BERT** with disentangled **a**ttention) that improves the BERT and RoBERTa models using two novel techniques. The first is the disentangled attention mechanism, where each word is represented using two vectors that encode its content and position, respectively, and the attention weights among words are computed using disentangled matrices on their contents and relative positions. Second, an enhanced mask decoder is used to incorporate absolute positions in the decoding layer to predict the masked tokens in model pre-training. We show that these two techniques significantly improve the efficiency of model pre-training and the performance of both natural language understand(NLU) and natural langauge generation(NLG) tasks. Compared to RoBERTa-Large, a DeBERTa model trained on half of the training data performs consistently better on a wide range of NLP tasks, achieving improvements on MNLI by +0.9% (90.2% vs. 91.1%), on SQuAD v2.0 by +2.3% (88.4% vs. 90.7%) and RACE by +3.6% (83.2% vs. 86.8%).

1 Introduction

The Transformer has become the most effective neural network architecture for neural language modeling. Unlike recurrent neural networks (RNNs) that process text in sequence, Transformers apply self-attention to compute in parallel every word from the input text an attention weight that gauges the influence each word has on another, thus allowing for much more parallelization than RNNs for large-scale model training (Vaswani et al., 2017). Since 2018, we have seen the rise of a set of large-scale Transformer-based Pre-trained Language Models (PLMs), such as GPT (Radford et al., 2019; Brown et al., 2020), BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019c), XLNet (Yang et al., 2019), UniLM (Dong et al., 2019a), ELECTRA (Clark et al., 2020), T5 (Raffel et al., 2019), ALUM (Liu et al., 2020), StructBERT (Wang et al., 2019) and ERINE (Sun et al., 2019) . These PLMs have been fine-tuned using task-specific labels and created new state-of-the-art in many downstream natural language processing (NLP) tasks (Liu et al., 2019b; Minaee et al., 2020; Jiang et al., 2019; He et al., 2019a;b; Shen et al., 2020).

In this paper, we propose a new Transformer-based neural language model **DeBERTa** (**D**ecoding-**e**nhanced **BERT** with disentangled **a**ttention) which has been proven to be more effective than RoBERTa and BERT and after fine-tuning leads to better results on a wide range of NLP tasks.

DEBERTa proposes a disentangled self-attention mechanism. Unlike BERT where each word in the input layer is represented using a vector which is the sum of its word (content) embedding and position embedding, each word in DeBERTa is represented using two vectors that encode its content and position, respectively, and the attention weights among words are computed using disentangled matrices based on their contents and relative positions, respectively. This is motivated by the observation that the attention weight of a word pair depends on not only their contents but their relative positions. For example, the dependency between the words “deep” and “learning” is much stronger when they occur next to each other than when they occur in different sentences.

As an extension to disentangled attention, we enhance the output layer of BERT for pre-training to address a limitation of relative positions. We observe in some situations, it is challenging for the relative positions only mechanism to accurately predict masking tokens. For example, considering a sentence "*A new **store** opened near the new **mall***" with the words *store* and *mall* masked for prediction, only using the relative positions is not sufficient for the model to accurately predict *store* and *mall* in this sentence, since they are exchangeable in syntax although with a different meaning, also right after the word *new* with an exact relative position to it. To address this limitation, we propose to introduce absolute positions back in the output layer of BERT, as a complement to the relative positions.

We show through a comprehensive empirical study that the disentangle attentions with its extensions substantially improve the efficiency of pre-training and the performance of downstream tasks. In the NLU tasks, compared to RoBERTa-Large, a DeBERTa model trained on half the training data performs consistently better on a wide range of NLP tasks, achieving improvements on MNLI by +0.9% (90.2% vs. 91.1%), on SQuAD v2.0 by +2.3%(88.4% vs. 90.7%), and RACE by +3.6% (83.2% vs. 86.8%). In the NLG tasks, DeBERTa improves the perplexity from 21.6 to 19.5 on the Wikitext-103 dataset.

2 Background

2.1 Transformer structure

A Transformer-based language model is composed of stacked Transformer blocks (Vaswani et al., 2017). Each block contains a multi-head self-attention layer followed by a fully connected positional feed-forward network. The standard self-attention mechanism lacks a natural way to encode word position information. Thus, existing approaches add a positional bias to each input word embedding so that each input word is represented by a vector whose value depends on its content and position. The positional bias can be implemented using absolute position embedding (Vaswani et al., 2017; Radford et al., 2019; Devlin et al., 2019) or relative position embedding (Huang et al., 2018; Yang et al., 2019). It has been shown that relative position representations are more effective for natural language understanding and generation tasks (Dai et al., 2019; Shaw et al., 2018). The proposed Disentangled Attention mechanism differs from all existing approaches in that we represent each input word using two separate vectors that encode a word's content and position respectively, and attention weights among words are computed using disentangled matrices on their contents and relative positions.

2.2 Masked Language Model

Large-scale Transformer-based PLMs (Devlin et al., 2019; Liu et al., 2019c; Lan et al., 2019) are typically pre-trained on large amounts of text to learn contextual word representations using a self-supervision objective, known as Masked Language Model (MLM). Specifically, given a sequence $\boldsymbol{X} = \{x_i\}$, we corrupt it into $\tilde{\boldsymbol{X}}$ by masking 15% of its tokens at random and then train a language model parameterized by θ to reconstruct $\boldsymbol{X}$ by predicting the masked tokens $\tilde{x}$ conditioned on $\tilde{\boldsymbol{X}}$:

$$\max_\theta \log p_\theta(\boldsymbol{X}|\tilde{\boldsymbol{X}}) = \max_\theta \sum_{i \in \mathcal{C}} \log p_\theta(\tilde{x}_i = x_i|\tilde{\boldsymbol{X}}) \tag{1}$$

where $\mathcal{C}$ is the index set of the masked tokens in the sequence. The authors of BERT propose to keep 10% of the masked tokens unchanged, another 10% replaced with randomly picked tokens and the rest replaced with the `[MASK]` token.

3 Approach

3.1 Disentangled Attention

For a token at position i in a sequence, we represent it using two vectors, $\{\boldsymbol{H_i}\}$ and $\{\boldsymbol{P_{i|j}}\}$, which represent its content and relative position with the token at position j, respectively. The calculation of the cross attention score between tokens i and j can be decomposed into four components as

$$\begin{aligned} A_{i,j} &= \{\boldsymbol{H_i}, \boldsymbol{P_{i|j}}\} \times \{\boldsymbol{H_j}, \boldsymbol{P_{j|i}}\}^\intercal \\ &= \boldsymbol{H_i}\boldsymbol{H_j}^\intercal + \boldsymbol{H_i}\boldsymbol{P_{j|i}}^\intercal + \boldsymbol{P_{i|j}}\boldsymbol{H_j}^\intercal + \boldsymbol{P_{i|j}}\boldsymbol{P_{j|i}}^\intercal \end{aligned} \tag{2}$$

That is, the attention weight of a word pair can be computed as a sum of four attention scores using disentangled matrices on their contents and positions as *content-to-content*, *content-to-position*, *position-to-content*, and *position-to-position* [1].

Existing approaches (Shaw et al., 2018; Huang et al., 2018) to relative position encoding use a separate embedding matrix to compute the relative position bias in computing attention weights. This is equivalent to computing the attention weights using only the content-to-content and content-to-position terms in equation 2. We argue that the position-to-content term is also important since the attention weight of a word pair depends not only on their contents but on their relative positions, which can only be fully modeled using both the content-to-position and position-to-content terms. Since we use *relative* position embedding, the position-to-position term does not provide much additional information and is removed from equation 2 in our implementation.

Taking single-head attention as an example, the standard self-attention (Vaswani et al., 2017) can be formulated as:

$$\boldsymbol{Q} = \boldsymbol{H}\boldsymbol{W_q}, \boldsymbol{K} = \boldsymbol{H}\boldsymbol{W_k}, \boldsymbol{V} = \boldsymbol{H}\boldsymbol{W_v}, \boldsymbol{A} = \frac{\boldsymbol{Q}\boldsymbol{K}^\intercal}{\sqrt{d}}$$
$$\boldsymbol{H_o} = \text{softmax}(\boldsymbol{A})\boldsymbol{V}$$

where $\boldsymbol{H} \in R^{N\times d}$ represents the input hidden vectors, $\boldsymbol{H_o} \in R^{N\times d}$ the output of self-attention, $\boldsymbol{W_q}, \boldsymbol{W_k}, \boldsymbol{W_v} \in R^{d\times d}$ the projection matrices, $\boldsymbol{A} \in R^{N\times N}$ the attention matrix, N the length of input sequence, and d the dimension of hidden state.

Denote k as the maximum relative distance, $\delta(i,j) \in [0, 2k)$ as the relative distance from token i to token j, which is defined as:

$$\delta(i,j) = \begin{cases} 0 & \text{for} \quad i-j \leqslant -k \\ 2k-1 & \text{for} \quad i-j \geqslant k \\ i-j+k & \text{others} \end{cases} \tag{3}$$

We can represent the disentangled self-attention with relative position bias as equation 4, where $\boldsymbol{Q_c}, \boldsymbol{K_c}$ and $\boldsymbol{V_c}$ are the projected content vectors generated using projection matrices $\boldsymbol{W_{q,c}}, \boldsymbol{W_{k,c}}, \boldsymbol{W_{v,c}} \in R^{d\times d}$ respectively, $\boldsymbol{P} \in R^{2k\times d}$ represents the relative position embedding vectors shared across all layers (i.e., staying fixed during forward propagation), and $\boldsymbol{Q_r}$ and $\boldsymbol{K_r}$ are projected relative position vectors generated using projection matrices $\boldsymbol{W_{q,r}}, \boldsymbol{W_{k,r}} \in R^{d\times d}$, respectively.

$$\begin{aligned} \boldsymbol{Q_c} = \boldsymbol{H}\boldsymbol{W_{q,c}}, \boldsymbol{K_c} = \boldsymbol{H}\boldsymbol{W_{k,c}}, \boldsymbol{V_c} = \boldsymbol{H}\boldsymbol{W_{v,c}}, \boldsymbol{Q_r} = \boldsymbol{P}\boldsymbol{W_{q,r}}, \boldsymbol{K_r} = \boldsymbol{P}\boldsymbol{W_{k,r}} \\ \tilde{A}_{i,j} = \underbrace{\boldsymbol{Q^c_i}\boldsymbol{K^c_j}^\intercal}_{\text{(a) content-to-content}} + \underbrace{\boldsymbol{Q^c_i}\boldsymbol{K^r_{\delta(i,j)}}^\intercal}_{\text{(b) content-to-position}} + \underbrace{\boldsymbol{K^c_j}\boldsymbol{Q^r_{\delta(j,i)}}^\intercal}_{\text{(c) position-to-content}} \\ \boldsymbol{H_o} = \text{softmax}(\frac{\tilde{\boldsymbol{A}}}{\sqrt{3d}})\boldsymbol{V_c} \end{aligned} \tag{4}$$

$\tilde{A}_{i,j}$ is the element of attention matrix $\tilde{\boldsymbol{A}}$, representing the attention score from token i to token j. $\boldsymbol{Q^c_i}$ is the i-th row of $\boldsymbol{Q_c}$. $\boldsymbol{K^c_j}$ is the j-th row of $\boldsymbol{K_c}$. $\boldsymbol{K^r_{\delta(i,j)}}$ is the $\delta(i,j)$-th row of $\boldsymbol{K_r}$ with regarding to relative distance $\delta(i,j)$. $\boldsymbol{Q^r_{\delta(j,i)}}$ is the $\delta(j,i)$-th row of $\boldsymbol{Q_r}$ with regarding to relative distance $\delta(j,i)$. Note that we use $\delta(j,i)$ rather than $\delta(i,j)$ here. This is because for a given position i, position-to-content computes the attention weight of the key content at j with respect to the query position at i, thus the relative distance is $\delta(j,i)$. The position-to-content term is calculated as $\boldsymbol{K^c_j}\boldsymbol{Q^r_{\delta(j,i)}}^\intercal$. The content-to-position term is calculated in a similar way.

Finally, we apply a scaling factor of $\frac{1}{\sqrt{3d}}$ on $\tilde{\boldsymbol{A}}$ which is important for stabilizing model training Vaswani et al. (2017), especially for large-scale PLMs.

[1] In this sense, our model shares some similarity to Tensor Product Representation (Smolensky, 1990; Schlag et al., 2019; Chen et al., 2019) where a word is represented using a tensor product of its filler (content) vector and its role (position) vector.

3.1.1 EFFICIENT IMPLEMENTATION

For an input sequence of length N, it requires a space complexity of $O(N^2d)$ (Shaw et al., 2018; Huang et al., 2018; Dai et al., 2019) to store the relative position embedding for each token. However, taking content-to-position as an example, we note that since $\delta(i,j) \in [0, 2k)$ and thus the embedding of all possible relative positions are always a subset of $\boldsymbol{K_r} \in R^{2k\times d}$, then we can reuse $\boldsymbol{K_r}$ in the attention calculation for all the queries. In experiments, we set the maximum relative distance k to 512 for pre-training. The disentangled attention weights can be computed efficiently using Algorithm 1. Let $\boldsymbol{\delta}$ be the relative position matrix according to equation 3, i.e., $\boldsymbol{\delta}[i,j] = \delta(i,j)$. Instead of allocating a different relative position embedding matrix for each query, we multiply each $query$ vector $\boldsymbol{Q_c}[i,:]$ by $\boldsymbol{K_r}^\intercal \in R^{d\times 2k}$, as in line $3-5$. Then, we extract the attention weight using the relative position matrix $\boldsymbol{\delta}$ as the index, as in line $6-10$. To compute the position-to-content attention score, we calculate $\tilde{\boldsymbol{A}}_{\boldsymbol{p}\rightarrow\boldsymbol{c}}[:,j]$, i.e., the column vector of the attention matrix $\tilde{\boldsymbol{A}}_{\boldsymbol{p}\rightarrow\boldsymbol{c}}$, by multiplying each key vector $\boldsymbol{K_c}[j,:]$ by $\boldsymbol{Q_r}^\intercal$, as in line $11-13$. Finally, we extract the corresponding attention score via the relative position matrix $\boldsymbol{\delta}$ as the index, as in line $14-18$. In this way, we do not need to allocate memory to store a relative position embedding for each query and thus reduce the space complexity to $O(kd)$ (for storing $\boldsymbol{K_r}$ and $\boldsymbol{Q_r}$).

Algorithm 1 Disentangled Attention

Input: Hidden state $\boldsymbol{H}$, relative distance embedding $\boldsymbol{P}$, relative distance matrix $\boldsymbol{\delta}$. Content projection matrix $\boldsymbol{W_{k,c}}$, $\boldsymbol{W_{q,c}}$, $\boldsymbol{W_{v,c}}$, position projection matrix $\boldsymbol{W_{k,r}}$, $\boldsymbol{W_{q,r}}$.

1: $\boldsymbol{K_c} = \boldsymbol{HW_{k,c}}, \boldsymbol{Q_c} = \boldsymbol{HW_{q,c}}, \boldsymbol{V_c} = \boldsymbol{HW_{v,c}}, \boldsymbol{K_r} = \boldsymbol{PW_{k,r}}, \boldsymbol{Q_r} = \boldsymbol{PW_{q,r}}$

2: $\boldsymbol{A_{c\rightarrow c}} = \boldsymbol{Q_c K_c^\intercal}$

3: **for** $i = 0, ..., N-1$ **do**

4:     $\tilde{\boldsymbol{A}}_{\boldsymbol{c}\rightarrow\boldsymbol{p}}[i,:] = \boldsymbol{Q_c}[i,:]\boldsymbol{K_r^\intercal}$

5: **end for**

6: **for** $i = 0, ..., N-1$ **do**

7:     **for** $j = 0, ..., N-1$ **do**

8:         $\boldsymbol{A_{c\rightarrow p}}[i,j] = \tilde{\boldsymbol{A}}_{\boldsymbol{c}\rightarrow\boldsymbol{p}}[i, \boldsymbol{\delta}[i,j]]$

9:     **end for**

10: **end for**

11: **for** $j = 0, ..., N-1$ **do**

12:     $\tilde{\boldsymbol{A}}_{\boldsymbol{p}\rightarrow\boldsymbol{c}}[:,j] = \boldsymbol{K_c}[j,:]\boldsymbol{Q_r^\intercal}$

13: **end for**

14: **for** $j = 0, ..., N-1$ **do**

15:     **for** $i = 0, ..., N-1$ **do**

16:         $\boldsymbol{A_{p\rightarrow c}}[i,j] = \tilde{\boldsymbol{A}}_{\boldsymbol{p}\rightarrow\boldsymbol{c}}[\boldsymbol{\delta}[j,i], j]$

17:     **end for**

18: **end for**

19: $\tilde{\boldsymbol{A}} = \boldsymbol{A_{c\rightarrow c}} + \boldsymbol{A_{c\rightarrow p}} + \boldsymbol{A_{p\rightarrow c}}$

20: $\boldsymbol{H_o} = \text{softmax}(\frac{\tilde{\boldsymbol{A}}}{\sqrt{3d}})\boldsymbol{V_c}$

Output: $\boldsymbol{H_o}$

3.2 TWO EXTENSIONS OF THE DISENTANGLED ATTENTION

The DeBERTa model has two additional extensions. One is to address a limitation of the relative positions which have been fully captured by the disentangled attentions. The other is to enable generation tasks and a multi-task learning objective.

Given a sentence "*A new **store** opened near the new **mall***" with the words **store** and **mall** masked for prediction, only using the relative positions is hard for the model to distinguish *store* and *mall* in this sentence, since both of them are right after the word *new* with the exact relative positions. To address this limitation, we propose to reconsider the introduction of absolute positions in the model, as a complement to the relative positions. There are at least two ways to introduce the absolution positions. The BERT model incorporates the absolute positions in the input layer. In DeBERTa, we propose an alternative to consider it right after all the Transformer layers but right before the *softmax* for masked token decoding, as shown in Figure 2. In this way, DeBERTa captures the relative positions in all the Transformer layers and only character the absolute position as a complementary in the *softmax* decoding layer. We call this new approach as Enhanced Mask Decoder(EMD). In our empirical studies, we compare these two approaches to incorporate the absolute positions and observe that the new approach in DeBERTa is much better. We conjecture the early introduction of the absolute position will undesirably hamper the model from learning accurate relative positions information. In addition, this new design will enable us to introduce additional information besides positions to the pre-training, which is out of the scope of this paper and will be explored in future.

Besides natural language understanding(NLU), we further extend DeBERTa for natural language generation (NLG) to verify the impacts of the disentangled attention thoroughly in both settings. To enable the autoregressive generation, we follow (Dong et al., 2019b) by using a triangular matrix for self-attention and set the upper triangular part of the self-attention mask to $-\infty$.

4 Experiment

This section evaluates DeBERTa on various NLP tasks for both NLU and NLG.

4.1 Main Results on NLU tasks

Following previous papers on BERT, RoBERTa and XLNet, we report results using large and base models.

4.1.1 Performance on Large Models

Model	CoLA Mcc	QQP F1/Acc	MNLI-m/mm Acc	SST-2 Acc	STS-B Corr	QNLI Acc	RTE Acc	MRPC Acc	Avg.
BERT$_{large}$	60.6	91.3	86.6/-	93.2	90.0	92.3	70.4	88.0	84.05
RoBERTa$_{large}$	68.0	92.2	90.2/90.2	96.4	92.4	93.9	86.6	90.9	88.82
XLNet$_{large}$	69.0	92.3	90.8/90.8	**97.0**	92.5	94.9	85.9	90.8	89.15
ALBERT$_{xxlarge}$	**71.4**	92.2	90.8/-	96.9	**93.0**	95.3	**89.2**	90.9	89.96
ELECTRA$_{\text{large}}$	69.1	**92.4**	90.9/-	96.9	92.6	95.0	88.0	90.8	89.46
DeBERTa$_{large}$	70.5	92.3	**91.1/91.1**	96.8	92.8	**95.3**	88.3	**91.9**	**90.00**

Table 1: Comparison results on the GLUE development set.

We pre-train our large models following the setting of BERT (Devlin et al., 2019), except that we use the BPE vocabulary as (Radford et al., 2019; Liu et al., 2019c). For training data, we use Wikipedia (English Wikipedia dump[2]; 12GB), BookCorpus (Zhu et al., 2015) (6GB), OPENWEBTEXT (public Reddit content (Gokaslan & Cohen, 2019); 38GB), and STORIES (a subset of CommonCrawl (Trinh & Le, 2018); 31GB). The total data size after data deduplication(Shoeybi et al., 2019) is about 78GB. We report the details of pre-trained dataset in Appendix A.2. We use 6 DGX-2 machines with 96 V100 GPUs to train the model. A single model trained with 2K batch size and 1M steps takes about 20 days. Refer to Appendix A for the detailed hyperparamters.

We summarize the results on eight GLUE (Wang et al., 2018) tasks in Table 1, which compares DeBERTa with previous models with around 350M parameters: BERT, RoBERTa, XLNet, ALBERT and ELECTRA. Note that RoBERTa, XLNet, ALBERT [3] and ELECTRA use 160G training data while DeBERTa uses 78G training data. RoBERTa and XLNet are trained for 500K steps with 8K samples in a step, which amounts to four billion passes over training samples. We train DeBERTa for one million steps with 2K samples in each step. This amounts to two billion passes of its training samples, approximately half of either RoBERTa or XLNet. Table 1 shows that compared to BERT and RoBERTa, DeBERTa is consistently better across all the tasks. Meanwhile, DeBERTa outperforms XLNet in six out of eight tasks. Particularly, the improvements on MRPC (1.1% over XLNet and 1.0% over RoBERTa), RTE (2.4% over XLNet and 1.7% over RoBERTa) and CoLA (1.5% over XLNet and 2.5% over RoBERTa) are significant. Even compared to the SOTA pre-trained models,

[2]https://dumps.wikimedia.org/enwiki/

[3]ALBERT includes an additional sentence-order prediction task.

$\text{ALBERT}_{\text{xxlarge}}$ [4] and $\text{ELECTRA}_{\text{large}}$, DeBERTa still outperforms them in term of the average "GLUE" score. Note that MNLI is often used as an indicative task to monitor the progress of pre-training. DeBERTa significantly outperforms all existing models of similar size on MNLI and creates a new state-of-the-art (SOTA).

Model	MNLI-m/mm Acc	SQuAD v1.1 F1/EM	SQuAD v2.0 F1/EM	RACE Acc	ReCoRD F1/EM	SWAG Acc	NER F1
BERT_{large}	86.6/-	90.9/84.1	81.8/79.0	72.0	-	86.6	92.8
RoBERTa_{large}	90.2/90.2	94.6/88.9	89.4/86.5	83.2	90.6/90.0	89.9	93.4
XLNet_{large}	90.8/90.8	95.1/89.7	90.6/87.9	85.4	-	-	-
$\text{Megatron}_{336\text{M}}$	89.7/90.0	94.2/88.0	88.1/84.8	83.0	-	-	-
DeBERTa_{large}	**91.1/91.1**	**95.5/90.1**	**90.7/88.0**	**86.8**	**91.4/91.0**	**90.8**	**93.8**
$\text{Megatron}_{1.3\text{B}}$	90.9/91.0	94.9/89.1	90.2/87.1	87.3	-	-	-
$\text{Megatron}_{3.9\text{B}}$	91.4/91.4	95.5/90.0	91.2/88.5	89.5	-	-	-

Table 2: Results on MNLI in/out-domain, SQuAD v1.1, SQuAD v2.0, RACE, ReCoRD, SWAG, CoNLL 2003 NER development set. Note that missing results in literature are signified by "-".

We evaluate DeBERTa on additional benchmarks: (1) Question Answering: SQuAD v1.1 (Rajpurkar et al., 2016), SQuAD v2.0 (Rajpurkar et al., 2018), RACE (Lai et al., 2017), ReCoRD (Zhang et al., 2018) and SWAG (Zellers et al., 2018); (2) Natural Language Inference: MNLI (Williams et al., 2018); and (3) NER: CoNLL-2003. For comparison, we also include Megatron (Shoeybi et al., 2019) with three different model sizes: $\text{Megatron}_{336\text{M}}$, $\text{Megatron}_{1.3\text{B}}$ and $\text{Megatron}_{3.9\text{B}}$, which are trained using the same dataset as RoBERTa. Note that $\text{Megatron}_{336\text{M}}$ has a similar model size as other models mentioned above[5].

We summarize the results in Table 2. Compared to the previous SOTA models with similar sizes, including BERT, RoBERTa, XLNet and $\text{Megatron}_{336\text{M}}$, DeBERTa consistently outperforms them in all the 7 tasks. Taking RACE as an example, DeBERTa is significantly better than previous SOTA XLNet with an improvement of 1.4% (86.8% vs. 85.4%). Although $\text{Megatron}_{1.3\text{B}}$ is 3 times larger than DeBERTa, we observe that DeBERTa can still outperform $\text{Megatron}_{1.3\text{B}}$ in three of the four benchmarks. All the results show the superior performance of DeBERTa in various downstream tasks. We are confident that DeBERTa can perform even better with a larger model size – we leave it to future work.

4.1.2 Performance on Base Models

The setting for base model pre-training is similar to that for large models. The base model structure follows that of the BERT base model, i.e., $L = 12, H = 768, A = 12$. We use 4 DGX-2 with 64 V100 GPUs to train the base model and it takes about 10 days to finish a single pre-training of 1M training steps with batch size 2048. We train DeBERTa with the same 78G text data, and compare it with RoBERTa and XLNet trained using their 160G text data. For detailed comparison of datasets for pre-training, please refer the Appendix A.2.

We summarize the results in Table 3. Across all three tasks, DeBERTa consistently surpasses RoBERTa and XLNet, with more improvements than that in large models. For example, on the MNLI in-domain setting (MNLI-m), $\text{DeBERTa}_{\text{base}}$ obtains 1.2% (88.8% vs. 87.6%) over $\text{RoBERTa}_{\text{base}}$, and 2% (88.8% vs. 86.8%) over $\text{XLNet}_{\text{base}}$.

4.2 Main Results on Generation Tasks

We further evaluate the DeBERTa model with auto-regressive language model (ARLM) using Wikitext-103 (Merity et al., 2016). DeBERTa-MT denotes our model trained jointly with MLM

[4] The hidden dimension of $\text{ALBERT}_{\text{xxlarge}}$ is 4 times of DeBERTa and the computation cost is about 4 times of DeBERTa.

[5] Although T5 (Raffel et al., 2019) has more parameters (11B), it only reports the test results and it is not comparable with other models.

Model	MNLI-m/mm (Acc)	SQuAD v1.1 (F1/EM)	SQuAD v2.0 (F1/EM)
RoBERTa$_{base}$	87.6/-	91.5/84.6	83.7/80.5
XLNet$_{base}$	86.8/-	-/-	-/80.2
DeBERTa$_{base}$	**88.8/88.5**	**93.1/87.2**	**86.2/83.1**

Table 3: Results on MNLI in/out-domain (m/mm), SQuAD v1.1 and v2.0 development set.

and ARLM as in UniLM (Dong et al., 2019b). The training hyper-parameters are the same as DeBERTa$_{\text{base}}$ except we use less training steps (200k). For fair comparison, we use RoBERTa as our baseline in the same setting. At last, we include GPT-2 and Transformer-XL for references. All of those models use the base model settings. DeBERTa+AP denotes the DeBERTa model trained without EMD but adding the absolute position embedding into the input layer as RoBERTa.

Model	RoBERTa	DeBERTa+AP	DeBERTa	DeBERTa-MT	GPT-2	Transformer-XL
Dev PPL	21.6	20.7	20.5	**19.5**	-	23.1
Test PPL	21.6	20.0	19.9	**19.5**	37.50	24

Table 4: Language model results on Wikitext-103 .

Table 4 reports the results on Wikitext-103. Note that DeBERTa$_{\text{base}}$ obtains a better PPL on both dev and test, and the joint training of MLM and ARLM reduces PPL further, showing the effectiveness of DeBERTa. Moreover, as a comparison between different places to incorporate the absolute positions, we show that the DeBERTa approach via injecting the absolute positions in the decoder layer is better than the RoBERTa approach (i.e., DeBERTa + *AP*) on the absolute positions.

4.3 Model Analysis

In this section, we first present an ablation study to quantify the relative contributions of different components introduced in DeBERTa. Then, we study the convergence property to characterize the model training efficiency. Due to space limit, we illustrate the difference in attention patterns between DeBERTa and its counterpart RoBERTa in Appendix A.7. We run experiments for analysis using the base model setting where the Wikipedia + Bookcorpus data is used for model pre-training and a model can be pre-trained for 1M steps with batch size 256 in 7 days on a DGX-2 machine with 16 V-100 GPUs.

4.3.1 Ablation study

To verify our experimental setting, we pre-train the RoBERTa base model from scratch. We call the re-pre-trained RoBERTa RoBERTa-ReImp$_{\text{base}}$. To investigate the relative contributions of different components in DeBERTa, we design three variations:

- -EMD is the DeBERTa base model without EMD.
- -C2P is the DeBERTa base model without the content-to-position term ((c) in Eq. 4).
- -P2C is the DeBERTa base model without the position-to-content term ((b) in Eq. 4). As XLNet also used relative position bias, this model is close to XLNet plus EMD.

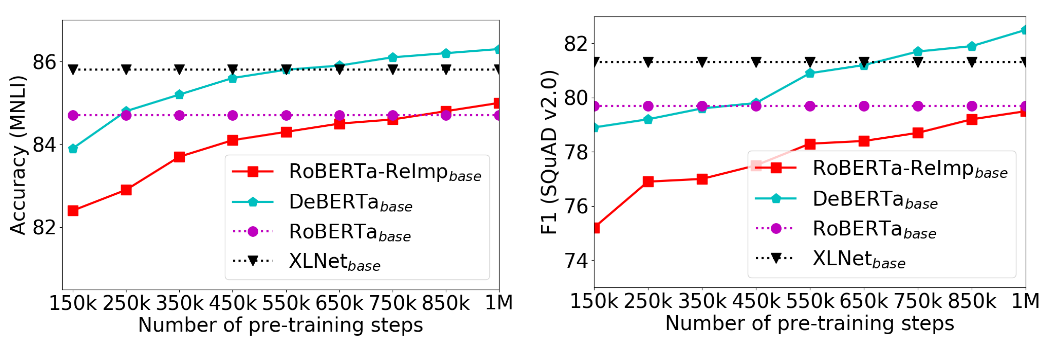
Table 5 summarizes the results on four benchmark datasets. First, comparing RoBERTa with RoBERTa-ReImp, we observe that they perform similarly across all the four benchmark datasets. Thus, we can confidently treat RoBERTa-ReImp as a solid baseline for comparison. Second, we see that removing any one component in DeBERTa results in a sheer performance drop in all the benchmarks. For instance, removing EMD (-EMD) results in a loss of 1.4% (71.7% vs. 70.3%) on RACE, 0.3% (92.1% vs. 91.8%) on SQuAD v1.1, 1.2% (82.5% vs. 81.3%) on SQuAD v2.0, 0.2% (86.3% vs. 86.1%) and 0.1% (86.2% vs. 86.1%) on MNLI-m/mm, respectively. Similarly, removing either *content-to-position* or *position-to-content* leads to consistent performance drops in all the benchmarks. As expected, removing two components results in even more significant deterioration in performance.

Model	MNLI-m/mm Acc	SQuAD v1.1 F1/EM	SQuAD v2.0 F1/EM	RACE Acc
BERT$_{base}$ Devlin et al. (2019)	84.3/84.7	88.5/81.0	76.3/73.7	65.0
RoBERTa$_{base}$ Liu et al. (2019c)	84.7/-	90.6/-	79.7/-	65.6
XLNet$_{base}$ Yang et al. (2019)	85.8/85.4	-/-	81.3/78.5	66.7
RoBERTa-ReImp$_{base}$	84.9/85.1	91.1/84.8	79.5/76.0	66.8
DeBERTa$_{base}$	**86.3/86.2**	**92.1/86.1**	**82.5/79.3**	**71.7**
-EMD	86.1/86.1	91.8/85.8	81.3/78.0	70.3
-C2P	85.9/85.7	91.6/85.8	81.3/78.3	69.3
-P2C	86.0/85.8	91.7/85.7	80.8/77.6	69.6
-(EMD+C2P)	85.8/85.9	91.5/85.3	80.3/77.2	68.1
-(EMD+P2C)	85.8/85.8	91.3/85.1	80.2/77.1	68.5

Table 5: Ablation study of the DeBERTa base model.

4.3.2 Pre-training Efficiency

To investigate the convergence of model pre-training, we plot the performance of fine-tuned downstream tasks as a function of the number of pre-training steps. As shown in Figure 1, for the RoBERTa-ReImp base model and the DeBERTa base model, we dump a checkpoint every 150K pre-training steps, and then fine-tune the checkpoint on two representative downstream tasks (MNLI and SQuAD v2.0) and then report the accuracy and F1 score, respectively. As a reference, we copy the final model performance of both the original RoBERTa base models (Liu et al., 2019c) and XLNet base models (Yang et al., 2019) and plot them as flat dot lines. The results show that DeBERTa consistently outperforms RoBERTa-ReImp during the course of pre-training, and converges faster to the performance of RoBERTa.



(a) Results on MNLI development

(b) Results on SQuAD v2.0 development

Figure 1: Pre-training performance curve between DeBERTa and its counterparts on the MNLI and SQuAD v2.0 development set.

5 Conclusions

This paper introduces a new model called DeBERTa for large-scale language model pre-training. DeBERTa first proposes the disentangled attention mechanism that represents each word using two vectors that encode its content and position, respectively, to thoroughly capture both contents and relative positions. As an extension to the disentangled attention, DeBERTa incorporates the absolute positions in the decoding layer as a complement to the relative positions. Compare to the strong RoBERTa and XLNet models, the DeBERTa model shows both better pre-training efficient and downstream NLU and NLG task accuracy consistently. For future work, we will explore alternative approaches to combine both relative and absolute position information in pre-training.

REFERENCES

Roy Bar-Haim, Ido Dagan, Bill Dolan, Lisa Ferro, and Danilo Giampiccolo. The second PASCAL recognising textual entailment challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*, 01 2006.

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

Luisa Bentivogli, Ido Dagan, Hoa Trang Dang, Danilo Giampiccolo, and Bernardo Magnini. The fifth pascal recognizing textual entailment challenge. In *In Proc Text Analysis Conference (TAC'09*, 2009.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*, 2017.

Kezhen Chen, Qiuyuan Huang, Hamid Palangi, Paul Smolensky, Kenneth D Forbus, and Jianfeng Gao. Natural-to formal-language generation using tensor product representations. *arXiv preprint arXiv:1910.02339*, 2019.

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

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In *Proceedings of the First International Conference on Machine Learning Challenges: Evaluating Predictive Uncertainty Visual Object Classification, and Recognizing Textual Entailment*, MLCW'05, pp. 177–190, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-33427-0, 978-3-540-33427-9. doi: 10.1007/11736790_9. URL `http://dx.doi.org/10.1007/11736790_9`.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2978–2988, 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina 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, Volume 1 (Long and Short Papers)*, pp. 4171–4186, 2019.

William B Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. Unified language model pre-training for natural language understanding and generation. In *Advances in Neural Information Processing Systems*, pp. 13042–13054, 2019a.

Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. Unified language model pre-training for natural language understanding and generation. *arXiv preprint arXiv:1905.03197*, 2019b.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pp. 1–9, Prague, June 2007. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/W07-1401`.

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

Pengcheng He, Xiaodong Liu, Weizhu Chen, and Jianfeng Gao. A hybrid neural network model for commonsense reasoning. *arXiv preprint arXiv:1907.11983*, 2019a.

Pengcheng He, Yi Mao, Kaushik Chakrabarti, and Weizhu Chen. X-sql: reinforce schema representation with context. *arXiv preprint arXiv:1908.08113*, 2019b.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Ian Simon, Curtis Hawthorne, Noam Shazeer, Andrew M Dai, Matthew D Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer: Generating music with long-term structure. 2018.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. *arXiv preprint arXiv:1911.03437*, 2019.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. Spanbert: Improving pre-training by representing and predicting spans. *arXiv preprint arXiv:1907.10529*, 2019.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 785–794, 2017.

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

Hector Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2012.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019a.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-task deep neural networks for natural language understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4487–4496, Florence, Italy, July 2019b. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P19-1441.

Xiaodong Liu, Hao Cheng, Pengcheng He, Weizhu Chen, Yu Wang, Hoifung Poon, and Jianfeng Gao. Adversarial training for large neural language models. *arXiv preprint arXiv:2004.08994*, 2020.

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

Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. 2018.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv*, pp. arXiv–1609, 2016.

Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao. Deep learning based text classification: A comprehensive review. *arXiv preprint arXiv:2004.03705*, 2020.

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

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

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL `https://www.aclweb.org/anthology/D16-1264`.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018.

Imanol Schlag, Paul Smolensky, Roland Fernandez, Nebojsa Jojic, Jürgen Schmidhuber, and Jianfeng Gao. Enhancing the transformer with explicit relational encoding for math problem solving. *arXiv preprint arXiv:1910.06611*, 2019.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 464–468, 2018.

Tao Shen, Yi Mao, Pengcheng He, Guodong Long, Adam Trischler, and Weizhu Chen. Exploiting structured knowledge in text via graph-guided representation learning. *arXiv preprint arXiv:2004.14224*, 2020.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Paul Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial intelligence*, 46(1-2):159–216, 1990.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Xuyi Chen, Han Zhang, Xin Tian, Danxiang Zhu, Hao Tian, and Hua Wu. Ernie: Enhanced representation through knowledge integration. *arXiv preprint arXiv:1904.09223*, 2019.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Wei Wang, Bin Bi, Ming Yan, Chen Wu, Zuyi Bao, Liwei Peng, and Luo Si. Structbert: Incorporating language structures into pre-training for deep language understanding. *arXiv preprint arXiv:1908.04577*, 2019.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*, 2018.

Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 1112–1122. Association for Computational Linguistics, 2018. URL http://aclweb.org/anthology/N18-1101.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pp. 5754–5764, 2019.

Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. Swag: A large-scale adversarial dataset for grounded commonsense inference. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 93–104, 2018.

Sheng Zhang, Xiaodong Liu, Jingjing Liu, Jianfeng Gao, Kevin Duh, and Benjamin Van Durme. Record: Bridging the gap between human and machine commonsense reading comprehension. *arXiv preprint arXiv:1810.12885*, 2018.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pp. 19–27, 2015.

A APPENDIX

A.1 DATASET

Corpus	Task	#Train	#Dev	#Test	#Label	Metrics
General Language Understanding Evaluation (GLUE)						
CoLA	Acceptability	8.5k	1k	1k	2	Matthews corr
SST	Sentiment	67k	872	1.8k	2	Accuracy
MNLI	NLI	393k	20k	20k	3	Accuracy
RTE	NLI	2.5k	276	3k	2	Accuracy
WNLI	NLI	634	71	146	2	Accuracy
QQP	Paraphrase	364k	40k	391k	2	Accuracy/F1
MRPC	Paraphrase	3.7k	408	1.7k	2	Accuracy/F1
QNLI	QA/NLI	108k	5.7k	5.7k	2	Accuracy
STS-B	Similarity	7k	1.5k	1.4k	1	Pearson/Spearman corr
Question Answering						
SQuAD v1.1	MRC	87.6k	10.5k	9.5k	-	Exact Match (EM)/F1
SQuAD v2.0	MRC	130.3k	11.9k	8.9k	-	Exact Match (EM)/F1
ReCoRD	MRC	101k	10k	10k	-	Exact Match (EM)/F1
RACE	MRC	87,866	4,887	4,934	4	Accuracy
SWAG	Multiple choice	73.5k	20k	20k	4	Accuracy
Token Classification						
CoNLL 2003	NER	14,987	3,466	3,684	8	F1

Table 6: Summary information of the NLP application benchmarks.

• **GLUE**. The General Language Understanding Evaluation (GLUE) benchmark is a collection of nine natural language understanding (NLU) tasks. As shown in Table 6, it includes question answering (Rajpurkar et al., 2016), linguistic acceptability (Warstadt et al., 2018), sentiment analysis (Socher et al., 2013), text similarity (Cer et al., 2017), paraphrase detection (Dolan & Brockett, 2005), and natural language inference (NLI) (Dagan et al., 2006; Bar-Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009; Levesque et al., 2012; Williams et al., 2018). The diversity of the tasks makes GLUE very suitable for evaluating the generalization and robustness of NLU models.

• **ReCoRD** is a commonsense Question Answering dataset. Each example consists of a news article, drawn from CNN and DailyMail, and a Cloze-style question about the article in which one entity is masked out (Zhang et al., 2018).

• **RACE** is a large-scale machine reading comprehension dataset, collected from English examinations in China, which are designed for middle school and high school students (Lai et al., 2017).

• **SQuAD v1.1/v2.0** is the Stanford Question Answering Dataset (SQuAD) v1.1 and v2.0 (Rajpurkar et al., 2016; 2018) are popular machine reading comprehension benchmarks. Their passages come from approximately 500 Wikipedia articles and the questions and answers are obtained by crowdsourcing. The SQuAD v2.0 dataset includes unanswerable questions about the same paragraphs.

• **SWAG** is a large-scale adversarial dataset for the task of grounded commonsense inference, which unifies natural language inference and physically grounded reasoning (Zellers et al., 2018). SWAG consists of 113k multiple choice questions about grounded situations.

• **CoNLL 2003** is an English dataset consisting of text from a wide variety of sources. It has 4 types of named entity.

A.2 PRE-TRAINING DATASET

For DeBERTa pre-training, we use Wikipedia (English Wikipedia dump[6]; 12GB), BookCorpus (Zhu et al., 2015) [7] (6GB), OPENWEBTEXT (public Reddit content (Gokaslan & Cohen, 2019); 38GB) and STORIES[8] (a subset of CommonCrawl (Trinh & Le, 2018); 31GB). The total data size after data deduplication(Shoeybi et al., 2019) is about 78GB. For pre-training, we also sample 5% training data as the validation set to monitor the training process. Table 7 compares datasets used in different pre-trained models.

Model	Wiki+Book 16GB	OpenWebText 38GB	Stories 31GB	CC-News 76GB	Giga5 16GB	ClueWeb 19GB	Common Crawl 110GB
BERT	✓						
XLNet	✓				✓	✓	✓
RoBERTa	✓	✓	✓	✓			
DeBERTa	✓	✓	✓				

Table 7: Comparison of the pre-training data.

A.3 IMPLEMENTATION DETAILS

Following RoBERTa (Liu et al., 2019c), we adopted dynamic data batching. We also include span masking(Joshi et al., 2019) as the additional masking strategy with the span size up to three. We list the detailed hyperparameters of pre-training in Table 8. For pre-training, we all use Adam (Kingma & Ba, 2014) as the optimizer with weight decay (Loshchilov & Hutter, 2018). For fine-tuning, even though we can get better and robust results with RAdam(Liu et al., 2019a) on some tasks, e.g. CoLA, RTE and RACE, we all use Adam(Kingma & Ba, 2014) as the optimizer for a fair comparison. For fine-tuning, we trained each task with a hyper-parameter search procedure, each run will take about 1-2 hours on a DGX-2 node. All the hyperparameters are presented in Table 9. The model selection is based on the performance on the task-specific development sets.

Our code is implemented based on Huggingface Transformers[9], FairSeq[10] and Megatron shoeybi2019megatron[11].

A.4 HANDLING LONG SEQUENCE INPUT

With relative position bias, we choose to truncate the maximum relative distance to k as in equation 3. Thus in each layer, each token can attend directly to at most $2(k-1)$ tokens and itself. By stacking Transformer layers, each token in the $l-$th layer can attend to at most $(2k-1)l$ tokens implicitly. Taking DeBERTa$_{\text{large}}$ as an example, where $k = 512, L = 24$, in theory, the maximum sequence length that can be handled is 24528. This is a byproduct benefit of our design choice and we found it is beneficial for the RACE task. A comparison of long sequence effect on the RACE task is shown in table 10.

Long sequence handling is an active research area as of late, there are a lot of works built on the Transformer to optimize its performance on long sequence handling(Beltagy et al., 2020; Kitaev et al., 2020; Child et al., 2019; Dai et al., 2019). One of the potential future works is to extend DeBERTa to deal with extremely long sequences and compare it with existing approaches.

A.5 MODEL COMPLEXITY

With Disentangled Attention, we introduced three additional parameters $\boldsymbol{W_{q,r}}, \boldsymbol{W_{k,r}} \in R^{d\times d}$ and $\boldsymbol{P} \in R^{2k\times d}$. The total increase in parameter is $2L \times d^2 + 2k \times d$. For the large model

[6]https://dumps.wikimedia.org/enwiki/
[7]https://github.com/butsugiri/homemade_bookcorpus
[8]https://github.com/tensorflow/models/tree/master/research/lm_commonsense
[9]https://github.com/huggingface/transformers
[10]https://github.com/pytorch/fairseq
[11]https://github.com/NVIDIA/Megatron-LM

Hyper-parameter	DeBERTa$_{large}$	DeBERTa$_{base}$	DeBERTa$_{base-ablation}$
Number of Layers	24	12	12
Hidden size	1024	768	768
FNN inner hidden size	4096	3072	3072
Attention Heads	16	12	12
Attention Head size	64	64	64
Dropout	0.1	0.1	0.1
Warmup Steps	10k	10k	10k
Learning Rates	2e-4	2e-4	1e-4
Batch Size	2k	2k	256
Weight Decay	0.01	0.01	0.01
Max Steps	1M	1M	1M
Learning Rate Decay	Linear	Linear	Linear
Adam ϵ	1e-6	1e-6	1e-6
Adam β_1	0.9	0.9	0.9
Adam β_2	0.999	0.999	0.999
Gradient Clipping	1.0	1.0	1.0
Gradient Clipping	1.0	1.0	1.0
Number of DGX-2 nodes	6	4	1
Training Time	20 days	10 days	7 days

Table 8: Hyper-parameters for pre-training DeBERTa.

Hyper-parameter	DeBERTa$_{large}$	DeBERTa$_{base}$
Dropout of task layer	{0,0.1,0.15}	{0,0.1,0.15}
Warmup Steps	{50,100,500,1000}	{50,100,500,1000}
Learning Rates	{5e-6, 8e-6, 9e-6, 1e-5}	{1.5e-5,2e-5, 3e-5, 4e-5}
Batch Size	{16,32,48,64}	{16,32,48,64}
Weight Decay	0.01	0.01
Maximun Training Epochs	10	10
Learning Rate Decay	Linear	Linear
Adam ϵ	1e-6	1e-6
Adam β_1	0.9	0.9
Adam β_2	0.999	0.999
Gradient Clipping	1.0	1.0

Table 9: Hyper-parameters for fine-tuning DeBERTa on down-streaming tasks.

Sequence length	Middle	High	Accuracy
512	88.8	85.0	86.3
768	88.7	86.3	86.8

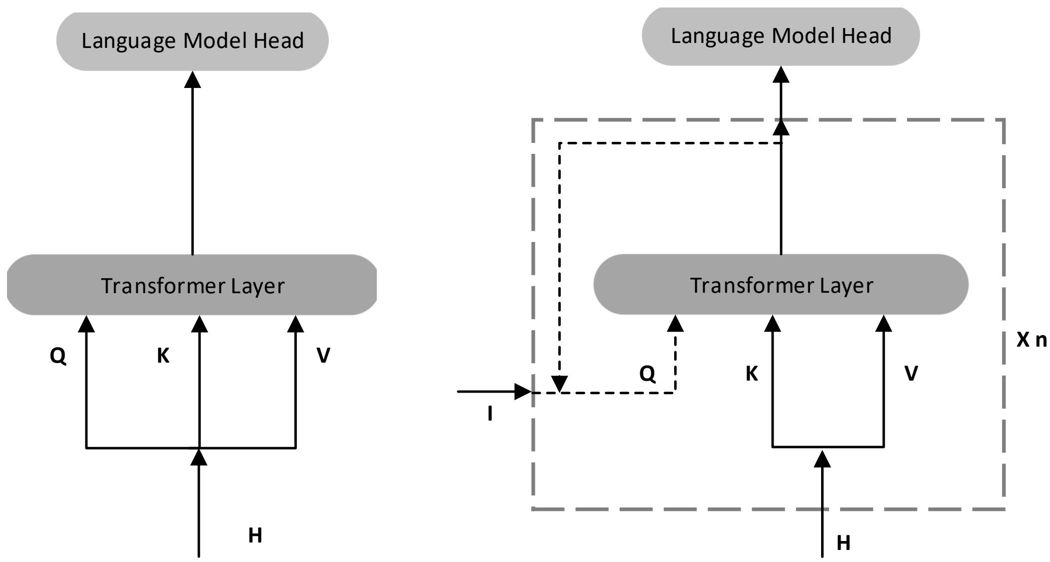
Table 10: The effect of handling long sequence input for RACE task with DeBERTa

($d = 1024, L = 24, k = 512$), this introduces about $49M$ additional parameters, which is an increment of 13%. For the base model($d = 768, L = 12, k = 512$), this introduces about $14M$ additional parameters, which is an increment of 12%. However, by sharing the projection matrix between content and position embedding, i.e. $\boldsymbol{W_{q,r}} = \boldsymbol{W_{q,c}}, \boldsymbol{W_{k,r}} = \boldsymbol{W_{k,c}}$, the number of parameters of DeBERTa will be the same as RoBERTa. Our experiment on base model shows that the results are almost the same. The results are shown in table 11. Due to computation resource limitation, we didn't run this setting with large model and we plan to re-run it in the future with this setting.

The additional computational complexity is $O(Nkd)$ due to the calculation of the additional *position-to-content* and *content-to-position* attention scores. Compared with BERT or RoBERTa, this intro-

Model	Parameters	MNLI-m/mm Acc	SQuAD v1.1 F1/EM	SQuAD v2.0 F1/EM
RoBERTa-ReImp$_{base}$	120M	84.9/85.1	91.1/84.8	79.5/76.0
DeBERTa$_{base}$	134M	86.3/86.2	92.1/86.1	82.5/79.3
DeBERTa$_{base}$+ShareProjection	120M	86.3/86.3	92.2/86.2	82.3/79.5

Table 11: Abluation study of the additional parameters in the DeBERTa base model.



(a) BERT decoding layer (b) Enhanced Mask Decoder

Figure 2: Comparison of the decoding layer.

duces about a 30% increase in computation. Compared with XLNet which also uses relative position embedding, the actual computation cost is about 15%. A further optimization by fusing the attention computation kernel could significantly reduce this additional cost. For EMD, since the decoder in pre-training only reconstructs the masked tokens, it does not introduce additional computation for unmasked tokens. In the situation where 15% tokens are masked and we use only two decoder layers, the additional cost is $0.15 \times 2/L$ which results in an additional computational cost of only 3% for base model($L = 12$) and 2% for large model($L = 24$) in EMD.

A.6 DETAIL OF ENHANCED MASK DECODER

The structure of EMD is shown in figure 2b. There are two inputs for EMD, i.e. I, H. H denotes the hidden states from the previous transformer layer, and I indicates the input for decoding which can be any necessary information for decoding, e.g., H, absolute position embedding or output from previous EMD layer. n denotes n stacked layers of EMD where the output of each EMD layer will be the input I for next EMD layer and the output of last EMD layer will be feed to language model head directly. The n layers can share the same weight. In our experiment we share the same weight for $n = 2$ layers to save parameters and use absolute position embedding as I of the first EMD layer. When $I = H$ and $n = 1$, EMD is the same as BERT decoder layer. However, EMD is more general and flexible as it can take more input information for the decoding task.

A.7 ATTENTION PATTERNS

To understand why DeBERTa performs differently from RoBERTa, we present their attention patterns in the last self-attention layer in Figure 3, where we also depict the attention patterns of the three DeBERTa variants for comparison. Comparing RoBERTa with DeBERTa, we observe two obvious

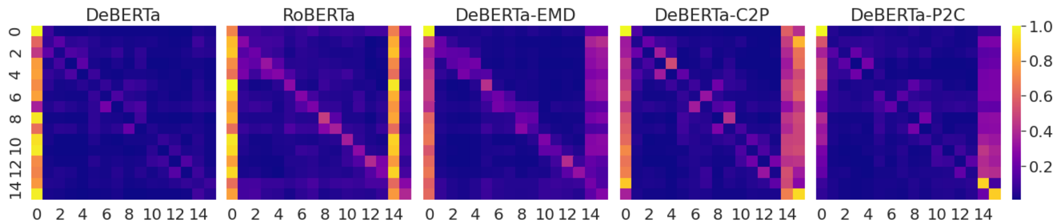
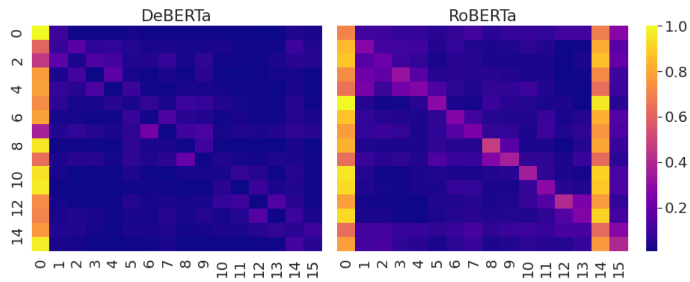


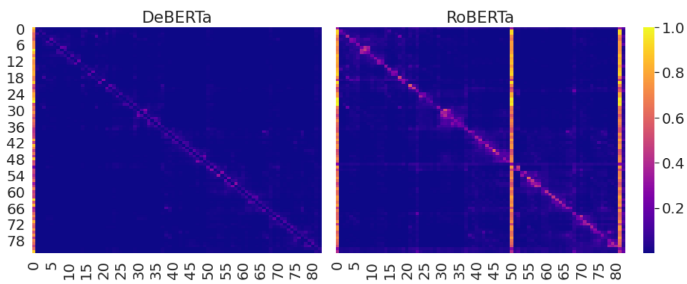
Figure 3: Comparison of attention patterns of the last layer among DeBERTa, RoBERTa and DeBERTa variants (i.e., DeBERTa without EMD, C2P and P2C respectively).

differences. First, RoBERTa has a clear diagonal line effect for a token to attend to itself, which is not observed in DeBERTa. This could be attributed to the use of EMD, in which the vectors of the masked but unchanged tokens are replaced with their position embeddings. This seems to be verified by examining the attention pattern of DeBERTa-EMD, where the diagonal line effect is brighter than the original DeBERTa. Second, there are vertical strips in the attention patterns of RoBERTa, which are mainly caused by high-frequent functional tokens (e.g., "a", "the", or punctuation). For DeBERTa, the strip appears in the first column, which represents the `[CLS]` token. We conjecture that a dominant emphasis on the `[CLS]` token is desirable for a good pre-trained model since the vector of this token is often used as a contextual representation of the entire input sequence in downstream tasks. We also observe that the vertical strip effect is quite obvious in the patterns of the three DeBERTa variants.

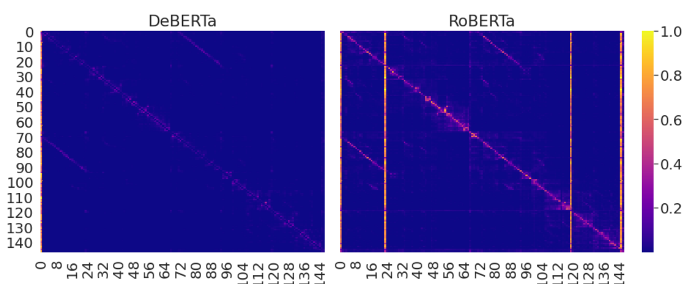
We provide three more examples to illustrate the difference in attention patterns between DeBERTa and RoBERTa as shown Figure 4, 5.



(a)

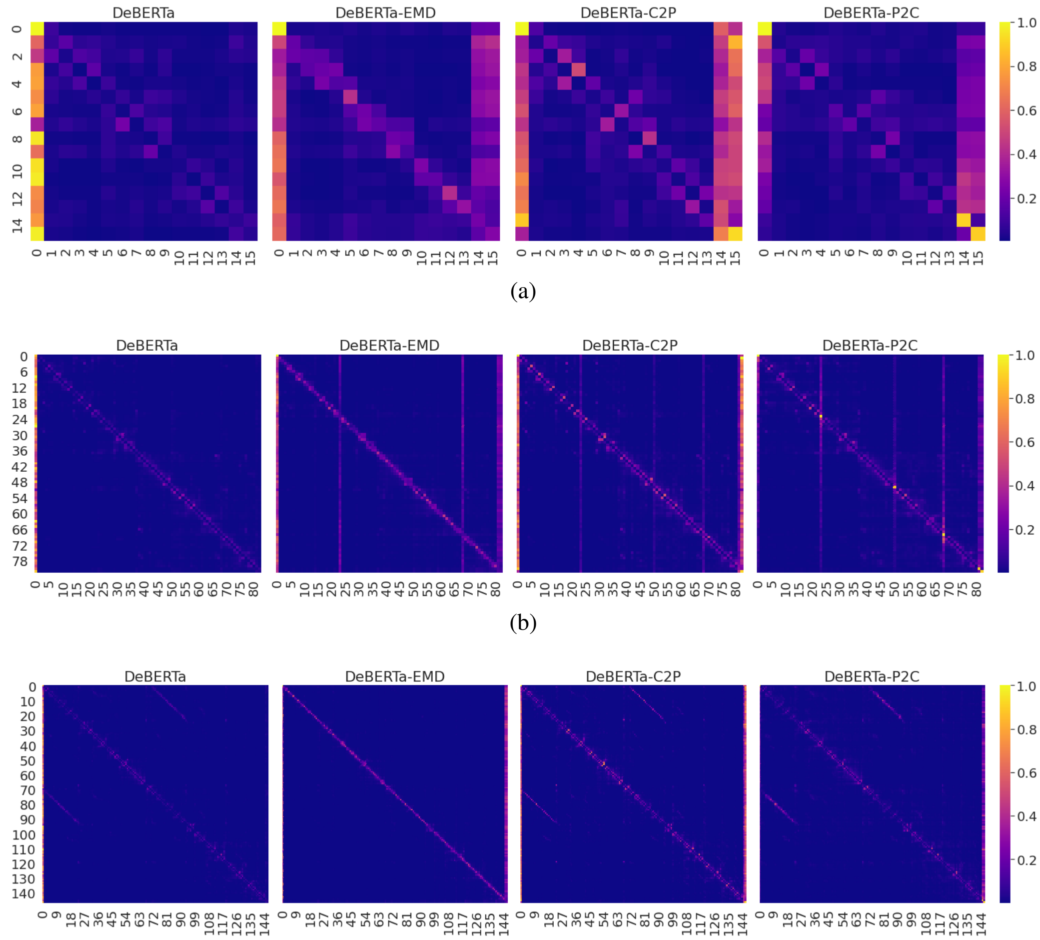


(b)



(c)

Figure 4: Comparison on attention patterns of the last layer between DeBERTa and RoBERTa.



(c)

Figure 5: Comparison on attention patterns of last layer between DeBERTa and its variants (i.e. DeBERTa without EMD, C2P and P2C respectively).