What Do Position Embeddings Learn? An Empirical Study of Pre-Trained Language Model Positional Encoding

Yu-An Wang **Yun-Nung Chen**
Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan
r08922019@csie.ntu.edu.tw y.v.chen@ieee.org

Abstract

In recent years, pre-trained Transformers have dominated the majority of NLP benchmark tasks. Many variants of pre-trained Transformers have kept breaking out, and most focus on designing different pre-training objectives or variants of self-attention. Embedding the position information in the self-attention mechanism is also an indispensable factor in Transformers however is often discussed at will. Therefore, this paper carries out an empirical study on position embeddings of mainstream pre-trained Transformers, which mainly focuses on two questions: 1) Do position embeddings really learn the meaning of positions? 2) How do these different learned position embeddings affect Transformers for NLP tasks? This paper focuses on providing a new insight of pre-trained position embeddings through feature-level analysis and empirical experiments on most of iconic NLP tasks. It is believed that our experimental results can guide the future work to choose the suitable positional encoding function for specific tasks given the application property.[1]

1 Introduction

Word ordering often determines the meaning of a sentence; therefore how to utilize the position information of a word sequence has been an important topic in NLP and widely investigated recently. A common approach for modeling word ordering is to use recurrent neural networks (RNN), such as long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) or gated recurrent unit (GRU) (Chung et al., 2014), which use a hidden state to represent the information of an ordered sequence and update model weights by backpropagation through time (BPTT) (Werbos, 1990); thus the ordering information can be modeled by this structure. However, RNN and BPTT are very inefficient in modern GPU computation due to the difficulty of parallelization with the time dependency. To solve this problem, recent work, such as convolutional seq2seq (Gehring et al., 2017) and Transformers (Vaswani et al., 2017) which apply convolutional neural network (CNN) (LeCun et al., 1995) and self-attention respectively, succeed to eliminate the time dependency to take the computational advantage of GPU. Instead of storing the information of ordered sequences, these models utilize the position information by using a feature-level positional encoding. For example, convolutional seq2seq proposed learnable position embeddings to represent the positions in a sequence.

Recently, various pre-trained Transformer language models keep breaking state-of-the-art results in numerous NLP tasks. There are many different ways to pre-train a Transformer language model. For example, using an encoder, decoder, or the whole part of the Transformer, adapting the self-attention masks, or training with different objectives (Devlin et al., 2018; Liu et al., 2019; Radford et al., 2018, 2019; Lewis et al., 2019; Raffel et al., 2019; Yang et al., 2019). However, in terms of positional encoding, most work only used a learned position embedding which is originally proposed in convolutional seq2seq (Gehring et al., 2017) without any analysis, even different objectives may learn completely different position information.

Motivated by the above observations, our goal is to investigate what position information the pre-trained Transformers could learn under different settings. We conduct a deep analysis of the learned position embeddings among three iconic pre-trained Transformer language models: BERT (Devlin et al., 2018), RoBERTa (Liu et al., 2019) and GPT-2 (Radford et al., 2019). To examine the performance of different NLP types, we conduct

[1]The source code is available at: https://github.com/MiuLab/PE-Study

the experiments on text classification, language modeling, and machine translation, and empirically analyze and explain the meaning and influence of position embeddings from different aspects.

The contributions of this paper are 3-fold:

- This paper is among the first study that provides a complete analysis about what learned position embeddings capture in different pre-trained models.
- This paper empirically examines the performance of different position embeddings for many NLP tasks.
- This paper connects the empirical performance with the task property based on the analysis, providing the guidance of the future work for choosing the suitable positional encoding method in the target task.

2 Related Work

The concept of using position embedding on position-insensitive models was first proposed by convolutional seq2seq (Gehring et al., 2017), which built an encoder-decoder architecture on convolutional neural networks. Vaswani et al. (2017) proposed Transformers that used the self-attention mechanism in the basic blocks. Because the attention mechanism is *position-insensitive*, it proposed a pre-defined sinusoidal function as positional encoding. Pre-trained language models became a trend among many NLP tasks after (Peters et al., 2018) introduced ELMo. Affected by ELMo, OpenAI GPT (Radford et al., 2018) is the first pre-trained language model using a Transformer architecture, then many different variant of pre-trained Transformer including BERT (Devlin et al., 2018), RoBERTa (Roberts, 2005) and GPT-2 (Radford et al., 2019) started evolving the researches of NLP tremendously. In Transformers, the attention values are the same in each input position. Thus, Shaw et al. (2018) proposed a relative position representation in the attention level to address this issue. Dai et al. (2019) used a segment-level recurrence mechanism on Transformers and also utilized an adaptive version of relative position embeddings inspired by Shaw et al. (2018). Furthermore, Wang et al. (2019) extended the embedding space from real numbers to complex values , and also proposed a new learnable positional encoding function instead of a simple position embedding mapping.

3 Transformer

Transformer is an encoder-decoder sequence-to-sequence model proposed by Vaswani et al. (2017). In the architecture, Transformer is composed of self-attention blocks that are position-insensitive modules. Therefore, a positional embedding should be considered together with the NLP tasks. To elaborate on the experiments we conduct, this section briefly introduces Transformers.

Input Representation Due to the property of position-insensitive in the attention module, the input representations should also contain the position information. In Transformers (Vaswani et al., 2017), a word embedding is directly added with the positional encoding as the final representation:

$$z_i = WE(x_i) + PE(i),$$

where x_i is the token at the i-th position, WE is the word embedding, and PE is the positional encoding, which can be either a learnable embedding or a pre-defined function.

Multi-Head Self-Attention The attention mechanism is often used in an encoder-decoder architecture, and there are many variants of attention implementations (Bahdanau et al., 2014; Britz et al., 2017). In Transformers, the *scaled dot-product attention* is applied:

$$\text{attention}(Q, K, V) = \text{softmax}(\frac{QWK^TW}{\sqrt{d_k}})VW,$$

where W is a linear projection and Q, K, V represent query, key and value matrices respectively.

Transformer blocks are composed of multi-head self-attention. Literally, the inputs Q, K, V are the same and the attention is performed multiple times, and then the output heads are concatenated as the final output hidden state h. This process can be formulated as

$$\text{head}_i = \text{attention}(Q, K, V)$$

$$h = \text{concat}([\text{head}_1, ..., \text{head}_n])W.$$

Transformer Encoder A Transformer encoder layer is composed of multi-head self-attention following a position-wise feed-forward network (FFN) with the residual connection (He et al., 2016) and layer normalization (Ba et al., 2016):

$$\text{output} = \text{layernorm}(h + \text{FFN}(h)),$$

and then stacked the layers sequentially to form a Transformer encoder.

Transformer Decoder The Transformer decoder is also stacked by self-attention blocks, and it only has two major differences from the encoder:

1. Each Transformer decoder layer has an additional sub-layer to perform attention on the encoder output.

2. To ensure the decoder can only decode tokens depending on the tokens in the past, it uses an attention mask to mask the attention values of the subsequent tokens.

Therefore, the Transformer decoder can decode tokens autoregressively like other conventional language models such as RNN.

4 Position Embedding Analysis

In this section, we conduct feature-level analyses of the pre-trained position embeddings of two Transformer encoders: BERT (Devlin et al., 2018) and RoBERTa (Liu et al., 2019), one Transformer decoder: GPT-2 (Radford et al., 2019), and also the sinusoidal function proposed by Vaswani et al. (2017) is defined as

$$PE_{(i,2j)} = \sin(i/10000^{2j/d_{model}}),$$
$$PE_{(i,2j+1)} = \cos(i/10000^{2j/d_{model}}),$$

where i is the position index and j is the dimension index.

4.1 Do Embeddings Learn the Meaning of Positions?

Given the position space $\mathcal{P}$ and the embedding space $\mathcal{X}$, the goal of the position embedding function is to learn a mapping $f : \mathcal{P} \rightarrow \mathcal{X}$. In the following experiments, we focus on answering two questions for better understanding what the embeddings capture:

1. Can the learned embedding space $\mathcal{X}$ represent the absolute positions of the words?

2. Are $\mathcal{P}$ and $\mathcal{X}$ isomorphic?

4.1.1 Absolute Position Regression

If a position embedding can actually capture its *absolute position*, it should be easy to reconstruct a reversed mapping function $g : \mathcal{X} \rightarrow \mathcal{P}$. Thus, we use linear regression to learn a function g that transfers the embeddings to the original positions. The feature dimension is 768, and the maximum position in GPT-2 is trimmed from 1024 to 512 for comparison which BERT and RoBERTa. Because we only have 512 data points for each learned embedding, a 5-fold cross-validation is applied to avoid overfitting. The reversed mapping functions are evaluated by **Mean Absolute Error (MAE)**, and the result is shown in Table 1.

Type	PE	MAE
Learned	BERT	34.14
	RoBERTa	6.06
	GPT-2	1.03
Pre-Defined	sinusoid	0.0

Table 1: Mean absolute error of the reversed mapping function learned by linear regression.

Type	PE	Error Rate
Learned	BERT	19.72%
	RoBERTa	7.23%
	GPT-2	1.56%
Pre-Defined	sinusoid	5.08%

Table 2: Error rate of the relative position regression.

From the results, the reversed mapping function of sinusoid can perfectly represent the absolute positions, and GPT-2 only has a small error. In contrast, the embeddings learned by Transformer encoders do not learn the information about the absolute positions, especially BERT which has an extremely high mean absolute error.

Additionally, we have also tried some more complicated non-linear models such as SVM or MLP to map the embeddings back. However, they easily overfit and the testing results are even worse than linear models. This implies that the position information in Transformer can actually be modeled by a linear model.

4.1.2 Relative Position Regression

In addition to absolute positions, the relation between positions is also informative (*relative positions*). If $\mathcal{P}$ and $\mathcal{X}$ are isomorphic, there should exist a bijection of distance operation between two spaces. Thus we define a mapping function of distances from $\mathcal{X}$ to $\mathcal{P}$: $h(x_i, x_j) = \|i - j\|$, where i, j are two position indices, $\|i - j\|$ is the distance between i and j in the space $\mathcal{P}$, and x_k is the position embedding at the k-th position. In this scenario, we can also build a mapping func-

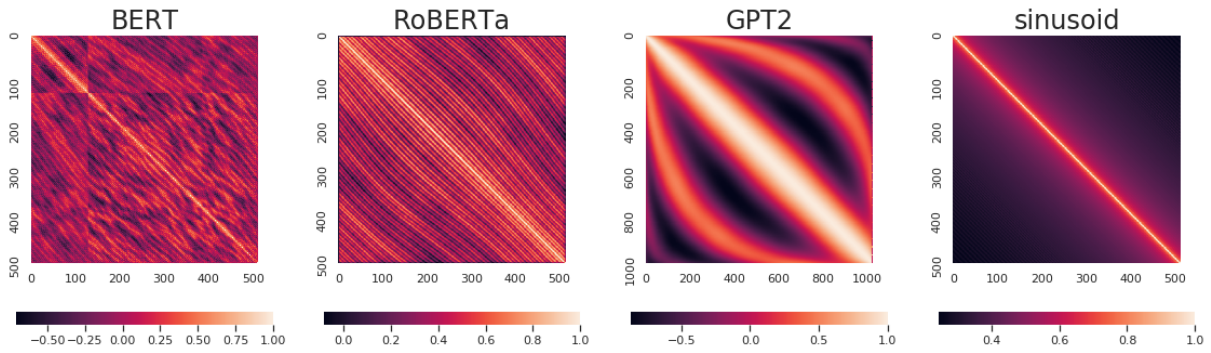

Figure 1: Visualization of position-wise cosine similarity of different position embeddings. Lighter in the figures denotes the higher similarity.

tion to check whether the embeddings can capture the relation between positions. However, in our preliminary experiments, we find that using linear regression to predict the distance of positions is too hard, since the relation between positions in the space $\mathcal{X}$ may not be completely linear. Hence, we simplify this problem to *whether the embeddings capture the order of every two positions*:

$$h(x_i, x_j) = \left\{ \begin{array}{ll} 1 & \text{if } i \geq j \\ 0 & \text{if } i < j \end{array} \right\},$$

and then use logistic regression to learn this binary classification problem.

The results in Table 2 show that the position embeddings of Transformer encoders still learn less information about their position relations, especially for BERT. Moreover, the sinusoid function, which can represent absolute positions perfectly, has a higher error rate than GPT-2 in relative positions but better than Transformer encoders, indicating the surprising capability of capturing such relations in GPT-2.

4.2 What do Transformer Encoders Capture about Positions?

According to the previous analyses, Transformer encoders (BERT and RoBERTa) may not well capture the meaning of positions (absolute and relative positions). Therefore, the interested question becomes "*what do Transformer encoders capture about positions?*".

4.2.1 Position-Wise Cosine Similarity

Figure 1 shows the visualization of position-wise cosine similarity of each position embedding. The point at (i, j) indicates the similarity between the i-th position and the j-th position. First, we observe that the embedding space of sinusoid and GPT-2

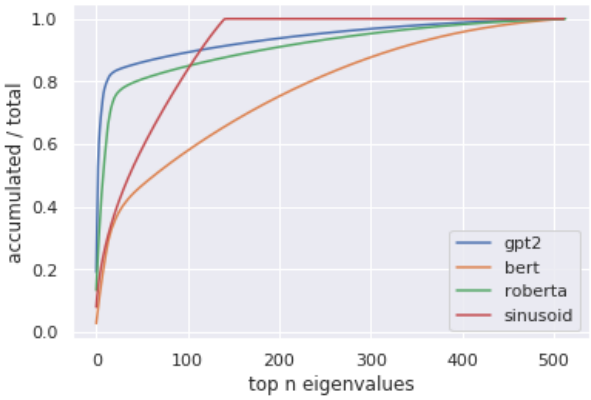

Figure 2: Accumulated top eigenvalues of position embeddings.

have obvious periodic patterns along with position orders, which aligns the findings in the section 4.1, where these two embeddings can actually capture the meanings of positions. With regard to BERT, we can only observe that embedding vectors are similar to the positions nearby but have no explainable patterns in long-term relations. Also, another observation is that BERT position embeddings have an obvious gap at the position 128, because the pre-trained procedure of BERT trains on the sentences with the length of 128 in the first stage, and then extends to the length of 512 in the second stage. The figure illustrates that the learned position information in the first stage can not be completely generalized to the second stage. Last but not least, the visualization of RoBERTa is similar to BERT, but have some limited non-periodic visible patterns at the positions nearby.

4.2.2 Informativeness of Position Embeddings

In order to examine the informativeness of the learned position embeddings, we apply singular

value decomposition (SVD) on position embeddings and analyze their eigenvectors. Figure 2 shows the curves of accumulated top n eigenvalues versus the proportion of total eigenvalues. Mathematically, the summation of the top n eigenvalues indicates how informative can a matrix be if the matrix is transformed into a n-dim space. For a position space $\mathcal{P}$, which is a 1-dim space, we may not need a high-dimension embedding space $\mathcal{X}$ to represent the positions. Thus, the summation of the top n eigenvalues in a position embedding should account for the most proportion of total eigenvalues with only a very small n. However, in Figure 2, we find that the position embeddings of BERT take a very large n to achieve a high proportion of total eigenvalues, and RoBERTa also takes a larger n than GPT-2 and sinusoid. This implies that the position embeddings of Transformer encoders may learn more complex information rather than only about positions, and this rich information may only be useful in Transformer encoders. This assumption will be further investigated in the experiments.

4.3 What Make the Differences?

Thus far, it can be found that the learned position embeddings between Transformer encoders and decoders are completely different. In this section, we will illustrate what makes these embeddings different.

4.3.1 Pre-Training Objectives

One of the main reason for the difference is the pre-training objectives. Pre-trained Transformer encoders minimize the masked language modeling loss, which can be formulated as

$$\mathcal{L}(\mathcal{U}) = \sum_i \log P(u_i|u_1, ...u_{i-1}, u_{i+1}, ...u_l; \theta),$$

where $\mathcal{U} = \{u_1, ...u_l\}$ is the pre-training corpus with the length l, and θ is the model parameters. For Transformer decoders, the objective is the traditional autoregressive language modeling loss:

$$\mathcal{L}(\mathcal{U}) = \sum_i \log P(u_i|u_1, ...u_{i-1}; \theta).$$

Transformer encoders can predict tokens depending on the tokens in both directions, while decoder can only predict depending on the token in the past. With enough context information, it is believed that Transformer encoders can succeed to predict tokens by only performing attention on the tokens nearby. That is why position embeddings learned by Transformer encoders do not need to involve the precise position information, aligning with the previous experiments in section 4.1.

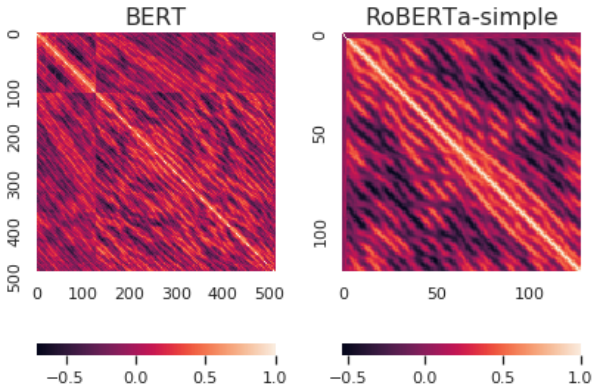

Figure 3: Visualized position-wise cosine similarity of the simplified RoBERTa position embeddings.

We infer that encoder position embeddings may capture the local position information, which can force the output capturing the positions nearby, especially BERT almost involving nothing about absolute positions. The inference makes the previous observations in sections 4.2.2 and 4.2.1 sensible and explainable, and we will verify this inference through empirical experiments in section 5.

4.3.2 Differences Between BERT and RoBERTa

Both BERT and RoBERTa use the same Transformer encoder architecture, but there are still some obvious differences in the previous discussion. Hence, we want to know what makes the position embeddings of BERT and RoBERTa different. The main improvements from BERT to RoBERTa are (Liu et al., 2019):

1. Sequentially increasing the batch size from 256 to 8192 during training.

2. Dynamically changing the token masks.

3. Eliminating the additional next sentence prediction (NSP) loss.

Due to the limitation of computing resources for experiments with a large batch size, we instead train a simplified version of RoBERTa that remains the 256 batch size and the shorter block length for faster convergence. The visualized position-wise cosine similarity of the simplified RoBERTa position embeddings showing in Figure 3 is very similar to BERT but without a gap at the position

128. As a result, we infer that a large batch pre-training can make the position embedding more robust and involve more clear position information in Transformer encoders.

5 Performance Effect

In addition to the behavior analysis, we are interested in the performance difference of positional embeddings for different NLP tasks, where we conduct text classification (encoding), language modeling (decoding) and machine translation (encoding and decoding). Note that each chosen task has its own important property where position information may cause different effects in Transformers.

5.1 Text Classification

Generally, for a text segment $s = \{x_1, x_2, ...x_n\}$ containing n tokens, a Transformer for classification can be formulated as

$$\begin{aligned} h^0 &= [z_1, ..., z_n], \\ h^i &= \text{transformer\_block}(h^{i-1}), \\ P(y \mid s) &= \text{softmax}(h_n^l), \end{aligned}$$

where z_i is the representation for the token x_i, y is the output class and h^i is the i-th layer output hidden state in Transformers.

Conventionally, a special token, usually `[eos]` or `[CLS]` would be appended to the end of input tokens, so that the output hidden state can perform attention on all other input tokens. In other words, no matter an encoder or a decoder is applied, the attention mask of the output hidden state and the objective can be identical. Therefore, we conduct a fair comparison with pre-trained position embeddings of both encoders and decoders in order to check whether all settings achieve similar performance.

Experimental Setup We experiment on six common text classification datasets: SST2, TREC, SUBJ, CR, MR, and MPQA. Since the last four datasets have no train/dev/test splits, we evaluate them with 5-fold cross-validation. We use the same model architecture as Wang et al. (2019), building a 1 layer Transformer encoder with 256 and 512 hidden size for self-attention and feed-forward respectively and 8 attention heads. Then five settings of the initialized position embeddings are performed: random, BERT, RoBERTa, GPT-2, and sinusoid, and other weights are initialized randomly.

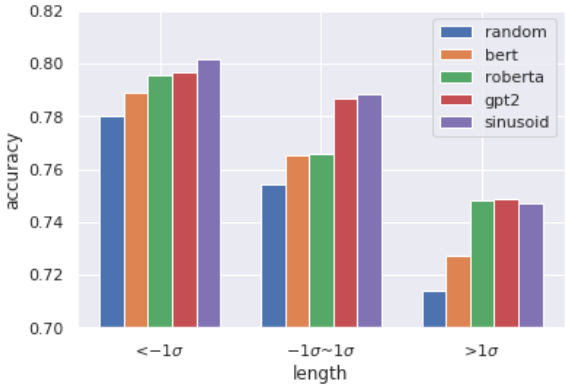

Figure 4: Length versus accuracy in text classification.

Discussions Table 3 shows the results of text classification accuracy. BERT and RoBERTa position embeddings perform much worse than GPT-2 and sinusoid in most cases. Because the output hidden state can utilize the information of all input tokens, the importance of absolute positions is certainly greater than local position information. However, in TREC and MPQA, the difference between 5 settings is insignificant, and we notice that the average lengths of these two sets are much shorter than others shown in the bottom of Table 3. Therefore, the position information is not very important in these tasks (TREC and MPQA), considering that the local positions or even random initialization can result in the performance as well as one with absolute positions. The experiments imply that even though text classification allows the model to utilize all tokens when making the prediction, the absolute positions, which GPT-2 can capture, may be still salient for longer inputs.

Length Sensitivity To further analyze how the position embeddings affect text classification with different sentence lengths, we plot different ranges of lengths versus accuracy in Figure 4. Here we only calculate the average accuracy of SUBJ, SST, and CR since the average lengths of TREC and MPQA are too short. MR dataset is also excluded, because we find the distribution of length and accuracy in MR is too different from other three datasets and it may cause a huge bias in the figure. Note that the results of MR roughly agrees with others.

In Figure 4, sinusoid and GPT-2 still have higher accuracy with the length shorter than one standard deviation of the whole dataset, but the difference is very subtle. In contrast, there is a significant gap between Transformer encoders and GPT-2 in longer sentences. In terms of extremely long sen-

PE	Average	Δ	SUBJ	SST2	CR	MR	TREC	MPQA
Random	0.7797	-	0.8638	0.7166	0.7313	0.7039	0.8520	0.8104
BERT	0.7868	(+0.0071)	0.8753	0.7221	0.7388	0.7142	0.8540	0.8125
RoBERTa	0.7886	(+0.0089)	0.8820	0.7353	0.7491	0.7266	0.8380	0.8004
GPT-2	0.7969	(+0.0172)	**0.8845**	0.7446	**0.7581**	0.7314	0.8540	0.8087
sinusoid	**0.7983**	**(+0.0186)**	0.8801	**0.7474**	0.7549	**0.7369**	0.8580	0.8125
Average Length			23	19	19	20	10†	3†

Table 3: Testing accuracy of text classification. † indicates the much shorter average length in TREC and MPQA, so position embedding can not significantly affect the result.

LM	PE	Wikitext-2		Wikitext-103	
		Perplexity	Δ	Perplexity	Δ
MLM	BERT	147.93	-	12.45	-
	+skip position	198.61	(+50.68)	323.12	(+310.67)
	RoBERTa	157.98	-	12.61	-
	+skip position	199.13	(+41.14)	14.44	(+1.83)
Autoregressive	GPT-2	172.97	-	25.83	-
	+skip position	171.20	(−1.77)	25.74	(−0.09)

Table 4: Testing perplexity in Wikitext-2 and Wikitext-103.

tences (longer than one standard deviation), we can only observe that BERT and random initialization perform much worse than others. We consider that the data distributions in this range have a too large bias so the results may not be robust. Therefore, the analysis provides a hint that GPT-2 may be better to tackle the longer inputs for classification.

5.2 Language Modeling

In section 4.3.1, we have introduced the objectives of the masked language model and autoregressive language model for Transformer encoders and decoders respectively. Also, in the previous discussions, it is believed that the masked language model only learns the local position information to make the output tokens capture the positions nearby. To further verify this inference, we propose the **skip position attack** on position embeddings.

Skip Position Attack We propose a *skip position attack* that skips the position index of input tokens. Originally, the input embedding can be represented as

$$z_i = WE(x_i) + PE(i)$$

. However, in this attack, we multiply the input position index by a constant k, then the input embedding of token x_i becomes

$$z_i = WE(x_i) + PE(i * k).$$

If the embedding only learns the local position information, skip position attack will skip the position indices nearby and lose local information. On the other hand, the absolute positions will not be influenced so much, because the order of the skipped positions is still the same. Based on the design, we conduct experiments to validate our inference.

Experimental Setup We conduct the experiments on the Wikitext-2 and Wikitext-103 datasets, which have 2 million and 103 million tokens respectively. For model architecture, we take `BERT-Base` for a masked language model and `GPT-2-Base` as an autoregressive language model, both models have 12 Transformer layers and 768 hidden size. Similar to text classification, all weights are randomly initialized except position embeddings. The constant k in the skip position attack is set to 4, and we slice the corpus into blocks of 128 length to fit the maximum length of pretrained position embeddings, which is 512.

Discussions Table 4 shows the results. On average, the masked language models (BERT and RoBERTa) have slightly lower perplexity than the autoregressive language model (GPT-2) due to their bidirectional token dependency. However, the skip position attack significantly harms the performance of the masked language models, while it affects

nothing on the autoregressive language model.

Another observation is that, in Wikitext-2, the distribution of position information is not robust enough so the difference between position embeddings of BERT and RoBERTa is not significant. However, in the larger dataset: Wikitext-103, skip position attack leads BERT position embeddings to extremely awful performance. The observation here is consistent with the inferences mentioned in section 4, and we can conclude that position embeddings of Transformer encoders focus on capturing the information nearby, especially BERT, which involves even less position information than RoBERTa.

5.3 Machine Translation

Neural machine translation is often trained by a sequence-to-sequence model (Sutskever et al., 2014), which includes both encoder and decoder in the model. Thus, there are two position embeddings in a Transformer for machine translation, and the position embeddings in the encoder and in the decoder may cause different effects in this task.

Experimental Setup We experiment on the Multi30k English-German dataset from WMT2016 shared tasks. The properties of the dataset are shown in Table 5. We use the scripts implemented by Fairseq (Ott et al., 2019) for a faster training process. The encoder and decoder have both 6 layers where each layer has 4 heads, 512, and 1024 hidden size for attention head and feed-forward respectively. Also, byte-pair encoding (BPE) (Sennrich et al., 2015; Gage, 1994) is applied to the corpus and the vocabulary size is reduced to $10,000$.

	Train	Valid	Test
Sentence Pairs	$29,000$	$1,015$	$1,000$
Average Length	12	12	12

Table 5: Statistics of Multi30k dataset.

To respectively investigate the effectiveness on the encoder and the decoder, there are total four different initialization settings of pre-trained position embeddings:

1. Position embeddings only for the encoder
2. Position embeddings only for the decoder
3. Different types of position embeddings for the encoder and decoder
4. Same position embeddings for both encoder and decoder

PE		BLEU	
Encoder	**Decoder**	**Full Set**	**Length $> 2\sigma$**
Random	Random	32.19	18.04
BERT	-	35.54	22.98
GPT-2	-	34.36	22.05
-	BERT	32.08	17.77
-	GPT-2	32.81	18.05
BERT	GPT-2	34.11	21.29
GPT-2	BERT	32.80	21.96
BERT	BERT	**35.94**	23.60
RoBERTa	RoBERTa	35.47	24.50
GPT-2	GPT-2	35.80	**25.12**

Table 6: BLEU scores on full set and long sentences ($> 2\sigma$) of Multi30k translation data. The hyphen (-) in the table means the same as the baseline (random).

For the first three settings, only BERT and GPT-2 are performed for conciseness.

The results are shown in Table 6, where we evaluate the BLEU scores on the sentences longer than 2 standard deviation (for both source and target) to analyze the effectiveness of longer sentences with consideration that the average length of Multi30k is relatively short.

Encoder Both BERT and GPT-2 position embedding can be effective in the encoder, especially BERT. The reason is that the decoded tokens can perform attention on all encoder outputs, thus the objective would be similar to the masked language modeling.

Decoder The effectiveness of position embeddings in the decoder is not as significant as one in the encoder, because the decoder cannot capture the order of the source language. We also observe that applying BERT position embeddings on the decoder even slightly harms the performance, since it may make the decoder tend to focus on the tokens nearby only.

Different for Encoder/Decoder According to the previous results, we hypothesize that using BERT in the encoder and GPT-2 in the decoder could perform best. However, in our experiments, using different pre-trained position embeddings is even worse, probably because the divergence of position embeddings trained by different models is quite huge and mixing them in the same model

may not suitable. Also, we swap the position embeddings in the encoder and decoder to see the impact. The BLEU score of the full set drops a lot, but in terms of long sentences, using GPT-2 in the encoder may not lose too much performance.

Same for Encoder/Decoder The results show that the performance between three pre-trained position embeddings are very close in the full set. However, in terms of longer sentences, GPT-2 is much better than BERT and RoBERTa. This observation aligns well with the previous analysis that the absolute position information is more important for longer sentences.

To sum up, there main observations are found: 1) The effectiveness of position embeddings in the encoder is more significant than one in the decoder. 2) Mixing different position embeddings in a model is not suitable. 3) GPT-2 position embeddings outperform others when modeling longer sentences.

6 Conclusion

This paper investigates the implicit meaning of pre-trained Transformer position embeddings. Transformer encoders learn the local position information that can only be effective in masked language modeling. On the other hand, the Transformer decoders for autoregressive language modeling actually learn about absolute positions. The empirical experiments on the pre-trained position embeddings validate our hypothesis. We also show that different NLP tasks with different model architectures and different training objectives may utilize the position information in different ways. As a result, it is believed that this study will benefit future work about choosing suitable positional encoding functions or designing other modeling methods for position information in the target NLP tasks based on their properties.

Acknowledgements

We thank reviewers for their insightful comments. This work was financially supported from the Young Scholar Fellowship Program by Ministry of Science and Technology (MOST) in Taiwan, under Grant 109-2636-E-002-026.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. 2017. Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906*.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

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 preprint 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 preprint arXiv:1810.04805*.

Philip Gage. 1994. A new algorithm for data compression. *C Users Journal*, 12(2):23–38.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252. JMLR. org.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Yann LeCun, Yoshua Bengio, et al. 1995. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

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 preprint arXiv:1907.11692*.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and

Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*.

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

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 preprint arXiv:1910.10683*.

Angus Roberts. 2005. Learning meronyms from biomedical text. In *Proceedings of the ACL Student Research Workshop*, pages 49–54, Ann Arbor, Michigan. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

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

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

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

Benyou Wang, Donghao Zhao, Christina Lioma, Qiuchi Li, Peng Zhang, and Jakob Grue Simonsen. 2019. Encoding word order in complex embeddings. *arXiv preprint arXiv:1912.12333*.

Paul J Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

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

A Reproducibility

A.1 Datasets

The datasets we used can be downloaded from the following linked pages, and the details of datasets are also described in the pages.

- Text Classification: link
- Language Modeling: link
- Machine Translation: link

A.2 Training Details

Text Classification

tokenizer	spacy
optimizer	Adam
lr	1^{-4}
batch_size	32
max_epoch	40

Language Modeling

	Wikitext02	Wikitext-103
lr	-	2.5^{-4}
batch_size	32	32
max_epoch	20	3
warup_steps	-	4000

Machine Translation

optimizer	Adam
weight_decay	0.0001
lr	1^{-4}
max_tokens	2048
max_epoch	40
lr_scheduler	inverse_sqrt
warup_steps	4000
label_smooth	0.1

Since the goal of this paper is to compare position embedding, we do not try too many hyperparameters on Transformers, and most setting are default as the implementation of hugginface and Fairseq.

A.3 Running Time

All our experiments are trained on 1 GTX 2080 TI GPU. Except language modeling on Wikitext-103 takes about 10 hours, all other trainings can be done within 2 hours.