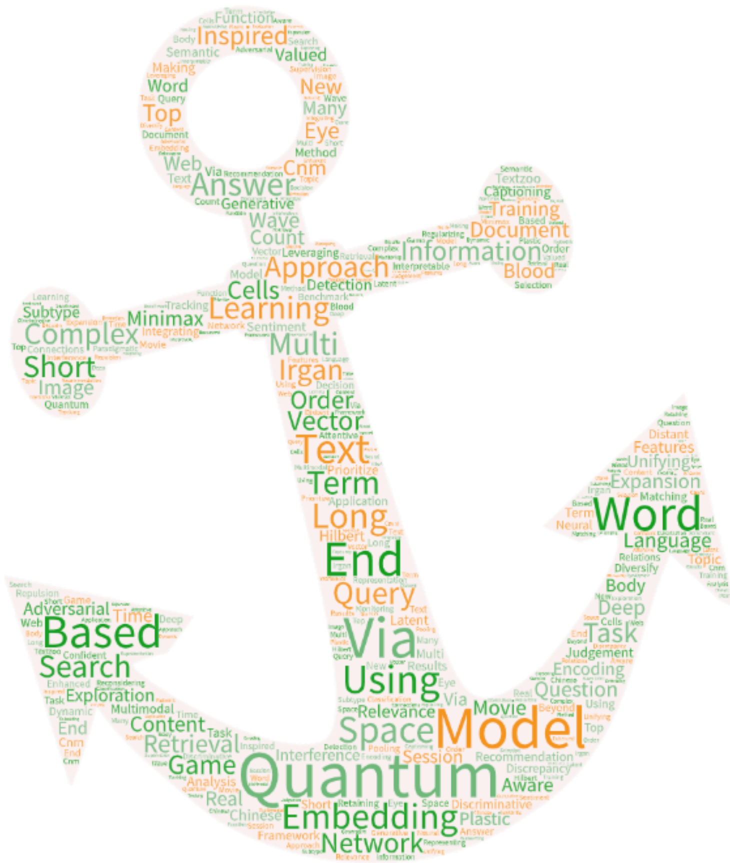
Beyond particles: modeling words as waves

Benyou Wang
Supervised by Massimo Melucci and Emanuele Di Buccio
University of Padua

Copenhagen, Denmark, 25/11/2019

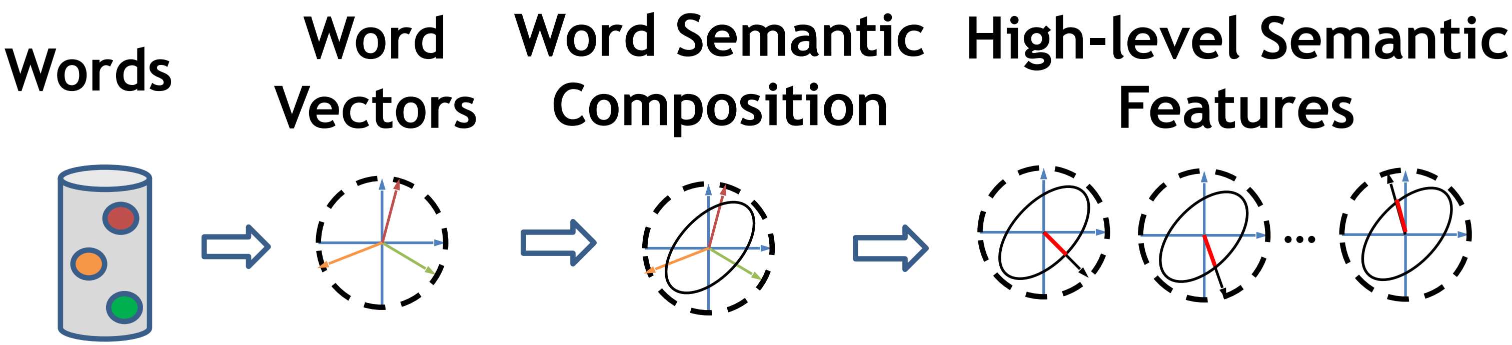
About me

Benyou Wang



- Research Experience
  - 2014-2017 Master student, Tianjin University, China
  - 2017-2018 Associate Researcher, Tencent
  - 2018-2021 Marie Curie Researcher and PhD student, University of Padua
- Awards
  - **SIGIR** 2017 Best paper Honourable mention
  - **NAACL** 2019 Best Explainable NLP Paper

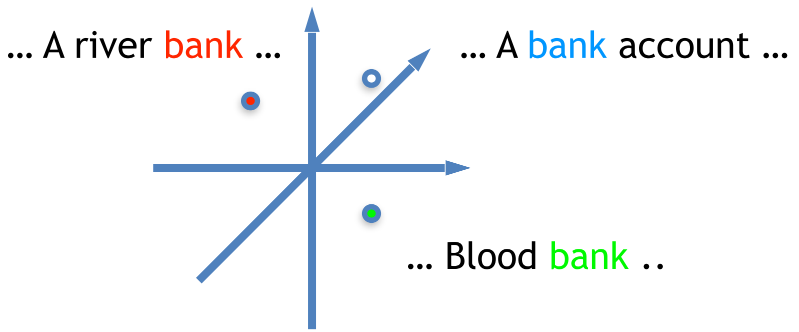
Understanding words: from particles to waves



Qiuchi Li*, **Benyou Wang***, Massimo Melucci. A Complex-valued Network for Matching. **NAACL 2019, Best Explainable NLP Paper**
Benyou Wang, Quantum formulations for language: understand words as **particles**, invited talk in Search Engines Amsterdam Meetup, University of Amsterdam, Amsterdam, Netherlands, Oct. 25th. 2019.

Some hints

- Contextualized word embedding [2]

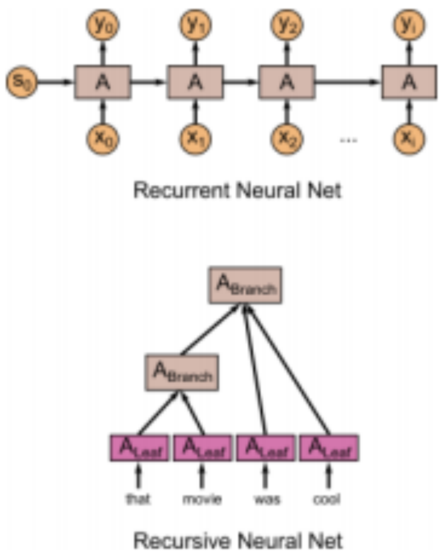


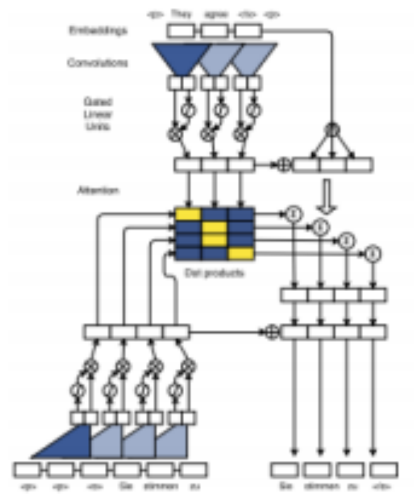
*For the same word, is there any possible **connections** between word vectors with individual context?*

Context: the neighboring words or a simpler case with only **considering word position**

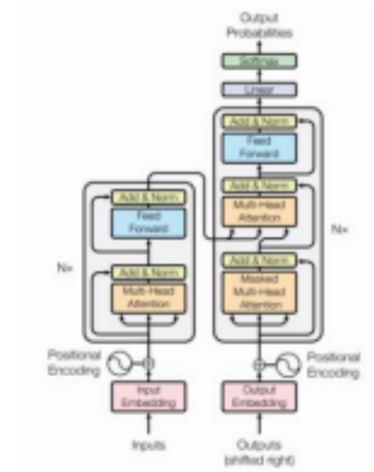
[1] Peters, Matthew E., et al. "Deep contextualized word representations." *NAACL* 2018 best paper.

Why word positions is important?





Conv seq2seq



Transformer

Especially If the network structures are insensitive to the word positions but more efficient !

Position embedding (PE)

Like Word Embedding (WE), PE also admits a map from a position index to a n-dimensional vector $\mathbb{N} \to \mathbb{R}^n$

A word w_j in pos-th position of a sentence is represented as

$$E(w_j, pos) = WE(w_j) + PE(pos) \in \mathbb{R}^n$$

where WE is map from word index to a n-dimensional vector $\mathbb{N} \to \mathbb{R}^n$.

PE

Solution 1 (PE vanilla):
Like word vectors, we just randomly initialize L independent vectors for each positio
and update them in a data-driven way.

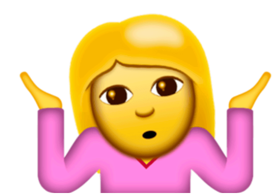
Problems:
In this case, position embeddings are independent without considering their order.

A B C
C B A
A C B
B C A
...

The order of word vocabulary does not matter !

1 2 3
3 2 1
1 3 2
...

The order of position does matter !



Gehring, Jonas, et al. "Convolutional sequence to sequence learning." ICML 2017.

TPE (Trigonometric PE)

Solution 2 (TPE):
Fixed embedding to capture relative distance:

$$PE'_{2k}(\cdot, pos) = \sin(pos/10000^{2k/d_{model}})$$
$$PE'_{2k+1}(\cdot, pos) = \cos(pos/10000^{2k/d_{model}})$$

Such that the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.

Problems:
Can not be trained ! Because it can not have potential to capture relative distance after training.

Vaswani, Ashish, et al. "Attention is all you need." *NIPS* 2017.

Problem

- The current position embedding **either** can capture relative relationship **or** be trainable!

- We want to propose a position embedding that can be **both** trainable **and** capture relative relationship

Word vectors to word functions

Extending embedding from a **vector** to a **continuous function** over variable the position (pos)

Technically, $f:(N, N) \rightarrow R^k$ To $\boldsymbol{f}:(\boldsymbol{N}) \rightarrow \boldsymbol{G}\{\boldsymbol{g}; \boldsymbol{g}: \boldsymbol{N} \rightarrow \boldsymbol{R}^k\}$

Word index

position index

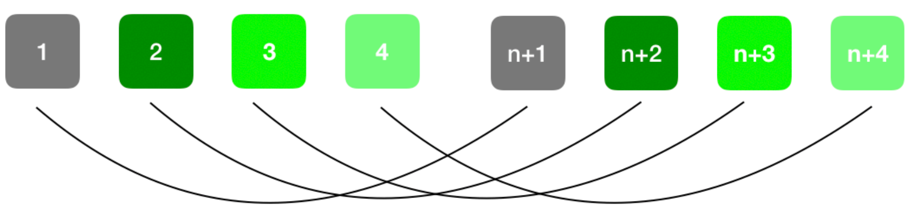
Now the question becomes *how to decide the function*

Desiderata for word functions

Now, for a specific word w, we have to get it embedding over all the positions, namely a function $\boldsymbol{g}_{\boldsymbol{w},d}: \boldsymbol{N} \rightarrow \boldsymbol{R}^{k}$

Property 1: Position-free relative-distance transformation

The word/position indexes are invisible in neural networks. It is easier if all the transformation pairs (move a word from one position to another one) $[g_{w,d}(1) \rightarrow g_{w,d}(n+1), g_{w,d}(2) \rightarrow g_{w,d}(n+2), \cdots, g_{w,d}(L) \rightarrow g_{w,d}(n+L)]$ correspond to a same n-offset-transformation without considering the start position.



Property 2: Boundedness

The function $\boldsymbol{g}_{\boldsymbol{w},d}$ should be bounded, in order to model long enough sentence

The first property make the problem much simper and can be feasible to solve

Property 1

Problem: We consider the simplest case when the n-offset transformation

$f(n): g(pos) \rightarrow g(n + pos)$

Which transform one from pos-th position to (pos+n) position to be **linear**.

$g_{w,d}(pos)f_{w,d}(n_1)f_{w,d}(n_2) = g_{w,d}(pos)f_{w,d}(n_1 + n_2)$

Solution: It is trivial to get the following solution (proof in the paper):

$f_{w,d}(n) = z_1^{\mathrm{n}}$

Result: z_1 is the parameters and $g_{w,d}(0) = z_2$ [1], such that

$g_{w,d}(pos) = z_2 z_1^{\mathbf{pos}}$;

[1] z_1, z_2 are related to the word index and position index, but superscripts are ignored for simplicity

Property 2

To make $g_{w,d}(pos)$ to be bounded:

$g_{w,d}(pos) = z_2 z_1^{\text{pos}}$; subject to $|z_1| \leq 1$

In real-domain, we necessary consider the extra constraint with some costs.

But if we extend z_1 in complex domain ($x = \alpha + \beta i = re^{i\theta}$), it is easier.

For example, $i = i; i^2 = -1; i^3 = -i; i^4 = 1; \cdots$

Property 2

To make $g_{w,d}(pos)$ to be bounded:

$g_{w,d}(pos) = z_2 z_1^{\text{pos}}$; subject to $|z_1| \leq 1$

In real-domain, we necessary consider the extra constraint with some costs.

But if we extend z_1 in complex domain ($x = \alpha + \beta i = re^{i\theta}$), it is easier.

For example, $i = 1; i^2 = -1; i^2 = -1; i^3 = -i; i^4 = 1; \cdots$

Let $z_1 = r_1 e^{i\theta_1}; z_2 = r_2 e^{i\theta_2}$

$g_{w,d}(pos) = z_2 z_1^{\text{pos}} = r_2 e^{i\theta_2}(r_1 e^{i\theta_1})^{\text{pos}} = r_2 r_1^{\text{pos}} e^{i(\theta_2 + \theta_1 \text{pos})}$ subject to $|r_1| \leq 1$

We directly make $r_1 = 1$, get

$g_w(pos) = r_2 e^{i(\theta_2 + \theta_1 \text{pos})}$

The proposed embedding

Our definition:

A word in *pos*-th position is represented as

$$[r_{j,1}e^{i(\omega_{j,1}\text{pos}+\theta_{j,1})}, \ldots, r_{j,2}e^{i(\omega_{j,2}\text{pos}+\theta_{j,2})}, \cdots, r_{j,D}e^{i(\omega_{j,D}\text{pos}+\theta_{j,D})}]$$

where each dimension like d has an amplitude $r_{j,d}$, and a unique period of $p_{j,d} = \dfrac{2\pi}{\omega_{j,d}}$.

i is the imaginary number.

Based on Euler's formula (i.e. $e^{ix} = \cos x + i \sin x$), each element can be rewritten as:

$g_{j,k} = r_{j,d}\cos(\omega_{j,d}\text{pos} + \theta_{j,d}) + r_{j,d}\sin(\omega_{j,d}\text{pos} + \theta_{j,d})i$

Link to TPE

TPE definition: $g'_{j,k} = WE'(j, \cdot\,) + PE'(\,\cdot\,, pos)$

$PE'_{2k}(\,\cdot\,, pos) = \sin(pos/10000^{2k/d_{model}});$
$PE'_{2k+1}(\,\cdot\,, pos) = \cos(pos/10000^{2k/d_{model}})$

It can be considered as a **specific case of ours** when $\omega_{\cdot,d} = \dfrac{1}{10000^{d/2d_{model}})}$

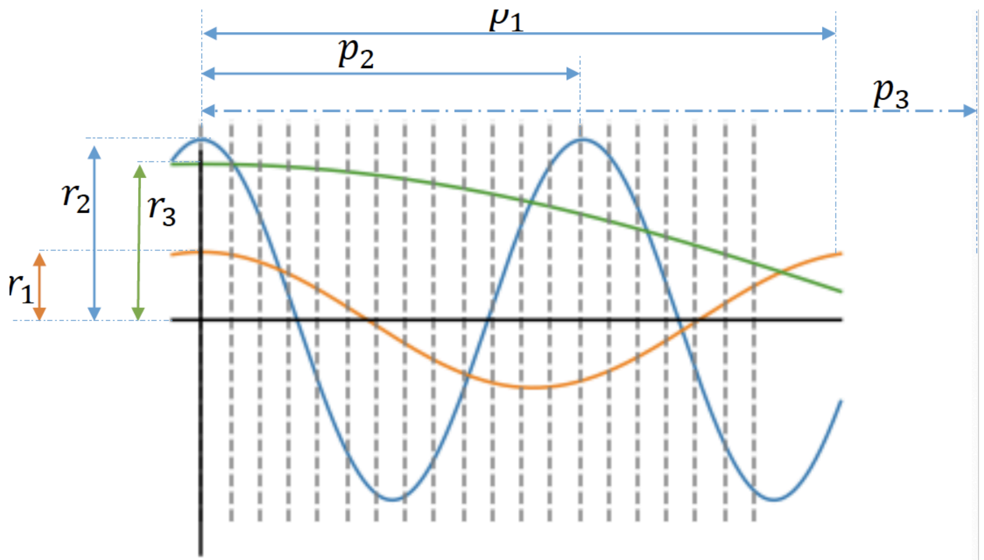
$$g_{j,k} = WE(j) \odot \left(\cos(\omega_{j,d}\text{pos}) + i\sin(\omega_{j,d}\text{pos})\right)$$

$$g_{j,k} = WE(j) \odot \left(PE'_{2k}(\,\cdot\,, pos) + iPE'_{2k}(\,\cdot\,, pos)\right)$$

$\odot$ is the element-wise multiplication

We argue that our proposed embedding is more general.

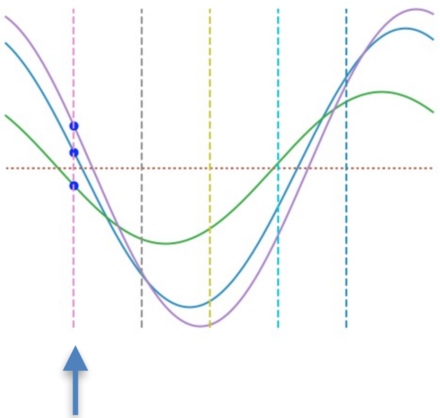
Example of proposed embedding



3-dimensional complex embedding for a single word in different positions. The three wave functions (setting the initial phases as zero) show the real part of the embedding. The x-axis denotes the absolute position of a word and the y-axis denotes the value of each element in its word vector. Colours mark different dimensions of the embedding. The three cross points between the functions and each vertical line (corresponding to a specific position pos) represent the embedding for this word in the pos-th position.

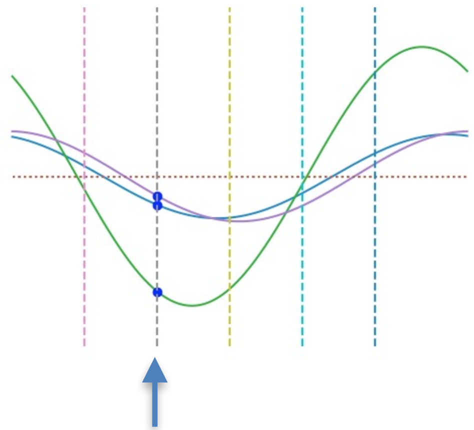
Words as waves

Word functions for ‘I'



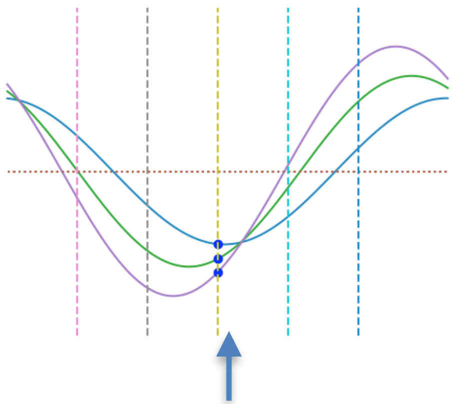
‘I’ is in the 1st position

Word functions for ‘love’



‘love’ is 2nd

Word functions for ‘Copenhagen’



‘Copenhagen’ is 3th

For the sentence ‘I love Copenhagen’

Talking is cheap !

```
import torch
import math
class ComplexNN(torch.nn.Module):
    def __init__(self, opt):
        super(ComplexNN, self).__init__()
        self.word_emb = torch.nn.Embedding(opt.n_token, opt.d_model)
        self.frequency_emb = torch.nn.Embedding(opt.n_token, opt.d_model)
        self.initial_phase_emb = torch.nn.Embedding(opt.n_token, opt.d_model)

    def get_embedding(self, x):

        amplitude = self.word_emb(x)
        frequency = self.frequency_emb(x)
        self.initial_phase_emb.weight = torch.nn.Parameter(self.initial_phase_emb.weight
            % (2 * math.pi))

        sent_len=x.size(-1)
        pos_seq = torch.arange(1, sent_len + 1, 1.0,device=amplitude.device)

        pos_seq = pos_seq.unsqueeze(0).unsqueeze(-1)
        pos_seq = pos_seq.repeat([x.size(0),1,amplitude.size(-1)])

        dimension_bais = self.initial_phase_emb (x)

        enc_output_phase = torch.mul(pos_seq,frequency)+ dimension_bais
        enc_output_real = amplitude * torch.cos(enc_output_phase)
        enc_output_image = amplitude * torch.sin(enc_output_phase)
        # return torch.cat([enc_output_real,enc_output_image],-1)
        return enc_output_real,enc_output_image

    def forward(self, x) :
        return self.get_embedding(x)
```

Applications

- For general neural networks
  - Complex valued neural networks [1,2]
  - Concat real and imaginal -part embedding
- For Transformer
  - Complex Transformer

Trabelsi, Chiheb, et al. "Deep complex networks." *arXiv preprint arXiv:1705.09792* (2017). ICLR 2018
Wolter, Moritz, and Angela Yao. "Complex gated recurrent neural networks." *NIPS* 2018.

Performance -1

In text classification

Method	MR	SUBJ	CR	MPQA	SST	TREC
Fasttext	0.765	0.916	0.789	0.874	0.788	0.874
Fasttext-PE	0.774	0.922	0.789	0.882	0.791	0.874
Fasttext-TPE	0.776	0.921	0.796	0.884	0.792	0.88
Fasttext-Complex-vanilla	0.773	0.918	0.79	0.867	0.803	0.872
Fasttext-Complex-order	**0.787**§†‡*	**0.929**§†‡*	**0.800**§†‡*	**0.889**§†‡*	**0.809**§†‡*	**0.892**§†‡*
LSTM	0.775	0.896	0.813	0.887	0.807	0.858
LSTM-PE	0.778	0.915	0.822	0.889	0.811	0.858
LSTM-TPE	0.776	0.912	0.814	0.888	0.813	0.865
LSTM-Complex-vanilla	0.765	0.907	0.810	0.823	0.784	0.784
LSTM-Complex-order	**0.790**§†‡*	**0.926**§†‡*	**0.828**§†‡*	**0.897**§†‡*	**0.819**§†‡*	**0.869**§†‡*
CNN	0.809	0.928	0.830	0.894	0.856	0.898
CNN-PE	0.816	0.938	0.831	0.897	0.856	0.890
CNN-TPE	0.815	0.938	0.836	0.896	0.838	0.918
CNN-Complex-vanilla	0.811	0.937	0.825	0.878	0.823	0.900
CNN-Complex-order	**0.825**§†‡*	**0.951**§†‡*	**0.852**§†‡*	**0.906**§†‡*	**0.864**§†‡*	**0.939**§†‡*
Transformer w/o position embedding	0.669	0.847	0.735	0.716	0.736	0.802
Transformer-PE	0.737	0.859	0.751	0.722	0.753	0.820
Transformer-TPE (Vaswani et al., 2017)	0.731	0.863	0.762	0.723	0.761	0.834
Transformer-Complex-vanilla	0.715	0.848	0.753	0.786	0.742	0.856
Transformer-Complex-order	**0.746**§†‡*	**0.895**§†‡*	**0.806**§†‡*	**0.863**§†‡*	**0.813**§†‡*	**0.896**§†‡*

Complex vanilla setting refers to the complex-valued word embedding as below:

Wang, Benyou, et al. "Semantic Hilbert Space for Text Representation Learning." *The World Wide Web Conference*. ACM, 2019.

Li, Qiuchi, et al. "Quantum-Inspired Complex Word Embedding." *Proceedings of The Third Workshop on Representation Learning for NLP*. 2018.

Superscripts §,†,‡and∗mean a significant improvement over a baseline without position embeddings§, PE†, TPE‡and Complex-vanilla∗using Wilcoxon’s signed-rank test $p<0.05$

Performance -2

In machine translation

Table 5.1: Machine translation results. ⋆marks scores reported from other papers.

Method	BLEU
AED (Bahdanau et al., 2014) ⋆	26.8
AED+Linguistic (Sennrich & Haddow, 2016) ⋆	28.4
AED+BPE (Sennrich et al., 2016) ⋆	34.2
Transformer (Ma et al., 2019) ⋆	34.5
Transformer complex vanilla	34.7
Transformer Complex-order	**35.8**

In language model

Table 5.2: Language modeling results. ⋆marks scores reported from other papers.

Method	BPC
BN-LSTM (Cooijmans et al., 2016) ⋆	1.36
LN HM-LSTM (Chung et al., 2016) ⋆	1.29
RHN (Zilly et al., 2017) ⋆	1.27
Large mLSTM (Krause et al., 2016) ⋆	1.27
Transformer XL 6L (Dai et al., 2019)	1.29
Transformer complex vanilla	1.30
Transformer XL Complex-order 6L	**1.26**

Take-away messages

- Extending word vectors to word functions
- First formal explanation for Trigonometric PE
- First embedding can trained to trade off word information and position information
- Complex-valued attention in Transformer
-

QA

Anything With BERT?

Just replace the word embedding layer with ours
or use our complex Transformer

Devlin, Jacob, et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." *NAACL best long paper* 2019.

Parameter scale

$[r_{j,1}e^{i(\omega_{j,1}\text{pos}+\theta_{j,1})}, \ldots, r_{j,2}e^{i(\omega_{j,2}\text{pos}+\theta_{j,2})}, \cdots, r_{j,D}e^{i(\omega_{j,D}\text{pos}+\theta_{j,D})}]$

For the proposed embedding, there are $3 \times |V| \times D$ parameters in total: $|V| \times D$ for each $r_{j,d}$, $\omega_{j,d}$, $\theta_{j,d}$.

Initialized phase can be ignored since it is empirically not working.

Two sharing schemas to share parameters:
word-sharing : $\omega_{j,d} = \omega_{\cdot,d}$
dimension-sharing : $\omega_{j,d} = \omega_{j,\cdot}$

Then we can get reasonable parameter scale.

Parameter scale

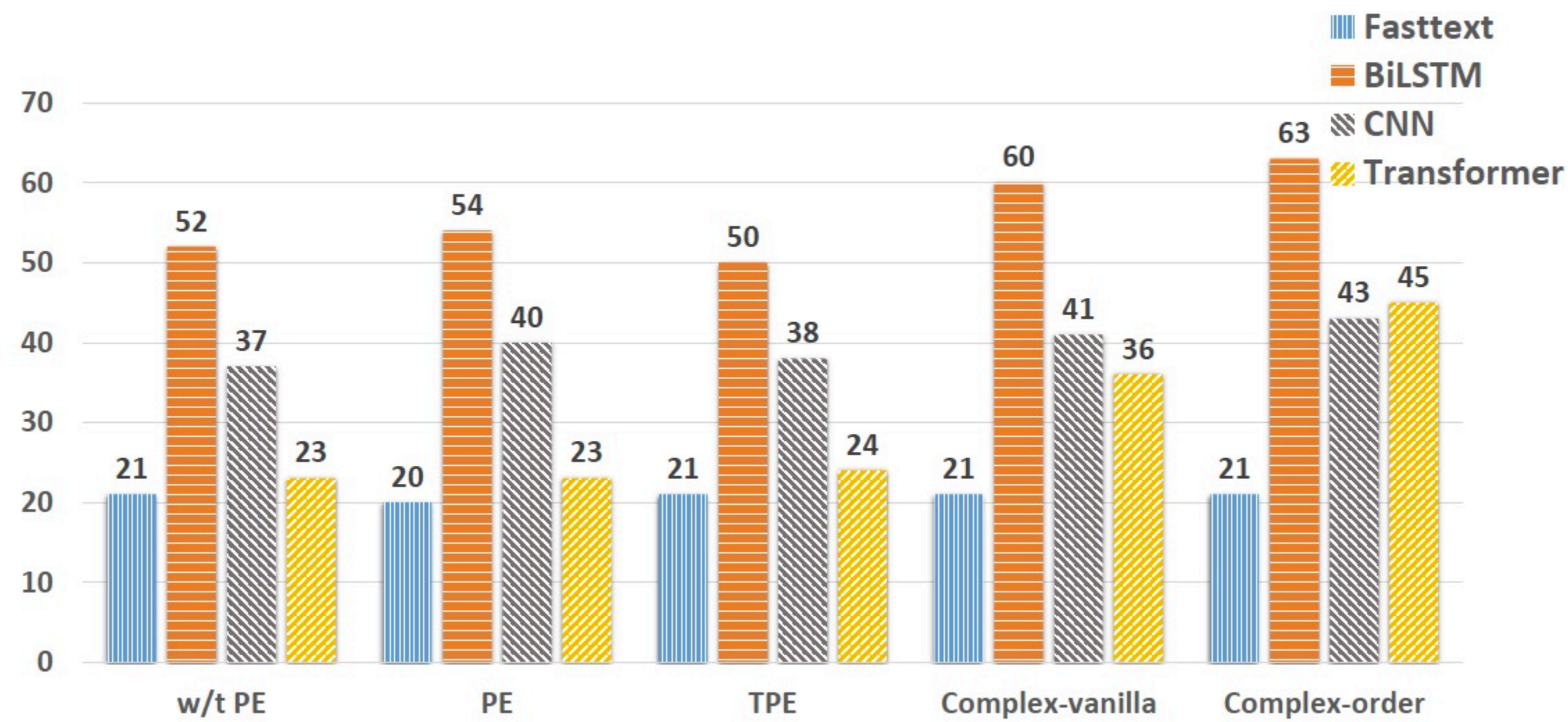
- In transformer

Table 4: Ablation test for Transformer, showing the effect of (i) the definition of embedding layer($f_d(j, \text{pos})$), and (ii) whether the real-part and imaginary transition share the weights, i.e., $\Re(W^{Q/K/V}) = \Im(W^{Q/K/V})$.

Method	Setting		Params	Accuracy	Δ
	$f_d(j, \text{pos})$	share in $W^{Q/K/V}$			
Transformer-complex-order	$r_{j,d}e^{i(\omega_{j,d}\text{pos})}$	×	8.33M	**0.813**	-
adding initial phases	$r_{j,d}e^{i(\omega_{j,d}\text{pos}+\theta_{j,d})}$	×	11.89M	0.785	-0.028
dimension-sharing period schema	$r_{j,d}e^{i\omega_{j,\cdot}\text{pos}}$	×	5.82M	0.797	-0.016
word-sharing period schema	$r_{j,d}e^{i\omega_{\cdot,d}\text{pos}}$	×	5.81M	0.805	-0.008
dimension-sharing amplitude schema	$r_{j,\cdot}e^{i\omega_{j,\cdot}\text{pos}}$	×	5.82M	0.798	-0.015
word-sharing amplitude schema	$r_{\cdot,d}e^{i\omega_{\cdot,d}\text{pos}}$	×	5.81M	0.804	-0.009
w/t encoding positions (complex-vanilla)	$r_{j,d}e^{i\omega_{j,d}}$	×	9.38M	0.764	-0.049
dimension-sharing period schema	$r_{j,d}e^{i\omega_{j,\cdot}\text{pos}}$	✓	4.77M	0.794	-0.019
word-sharing period schema	$r_{j,d}e^{i\omega_{\cdot,d}\text{pos}}$	✓	4.76M	0.797	-0.016
dimension-sharing amplitude schema	$r_{j,\cdot}e^{i\omega_{j,\cdot}\text{pos}}$	✓	4.77M	0.792	-0.021
word-sharing amplitude schema	$r_{\cdot,d}e^{i\omega_{\cdot,d}\text{pos}}$	✓	4.76M	0.801	-0.012
w/t encoding positions (complex-vanilla)	$r_{j,d}e^{i\omega_{j,d}}$	✓	8.33M	0.743	-0.07
vanilla Transformer (Vaswani et al., 2017)	$WE_{j,d} + PE_d$	-	4.1M	0.761	-0.052

Time Cost

Computing time (second per epoch) on TITAN X GPU



PE in GCN

setting	MR	SUBJ	CR	MPQA	SST	TREC
GCN	0.786	0.934	0.844	0.833	0.826	0.906
GCN-PE	0.781	0.931	0.810	0.830	0.822	0.884
GCN-TPE	0.548	0.928	0.656	0.828	0.818	0.886
GCN-Complex-vanilla	0.762	0.918	0.831	0.824	0.805	0.886
GCN-Complex-order	0.781	0.931	0.825	0.833	0.816	0.900

GCN also encode structural information (more advanced than positional level) inherently as part of the model, w
makes redundant any additional encoding of positional information at the embedding level.