ENCODING WORD ORDER IN COMPLEX EMBEDDINGS

Wang, Benyou; Zhao, Donghao; Lioma, Christina; Li, Qiuchi; Peng, Zhang; Simonsen, Jakob Grue

Publication date:
2020

Document version
Peer reviewed version

Document license:
Other

Citation for published version (APA):
ENCODING WORD ORDER IN COMPLEX EMBEDDINGS

Benyou Wang *  Donghao Zhao *  Christina Lioma
University of Padua  Tianjin University  University of Copenhagen
wang@dei.unipd.it  zhaodh@tju.edu.cn  chrh@di.ku.dk

Qiuchi Li  Peng Zhang †  Jakob Grue Simonsen
University of Padua  Tianjin University  University of Copenhagen
qiuchili@dei.unipd.it  pzhang@tju.edu.cn  simonsen@di.ku.dk

ABSTRACT

Sequential word order is important when processing text. Currently, neural networks (NNs) address this by modeling word position using position embeddings. The problem is that position embeddings capture the position of individual words, but not the ordered relationship (e.g., adjacency or precedence) between individual word positions. We present a novel and principled solution for modeling both the global absolute positions of words and their order relationships. Our solution generalizes word embeddings, previously defined as independent vectors, to continuous word functions over a variable (position). The benefit of continuous functions over variable positions is that word representations shift smoothly with increasing positions. Hence, word representations in different positions can correlate with each other in a continuous function. The general solution of these functions is extended to complex-valued domain due to richer representations.

We extend CNN, RNN and Transformer NNs to complex-valued versions to incorporate our complex embedding (we make all code available). Experiments on text classification, machine translation and language modeling show gains over both classical word embeddings and position-enriched word embeddings. To our knowledge, this is the first work in NLP to link imaginary numbers in complex-valued representations to concrete meanings (i.e., word order).

1 INTRODUCTION

When processing text, the sequential structure of language is important, but can be computationally costly to model with neural networks (NNs) (Socher et al., 2011) due to the difficulty in parallelization. This has been alleviated by modeling word sequence not on the NN architecture level, but by adding position embeddings on the feature level. This has been done by the convolutional sequence model (ConvSeq) (Gehring et al., 2017) and the Transformer model (Vaswani et al., 2017) that replaces recurrent and convolution operations with purely attention mechanisms. More generally, vanilla position embeddings (Gehring et al., 2017) assume that individual word positions are independent and do not consider relations between neighbouring word positions. We posit that both the global absolute positions of words and their inner sequential and adjacent relationships are crucial in language. This is supported by recent empirical findings by Shaw et al. (2018) and Dai et al. (2019) who show the importance of modeling distance between sequential elements, and explicitly use extra relative position encodings to capture the relative-distance relationship of words.

We present a novel and principled approach to model both the global absolute positions of words and their inner sequential and adjacent relationships as follows: we extend each word embedding, previously defined as an independent vector, as a continuous function over an independent variable i.e., position. The benefit of continuous functions over variable positions is that word representations shift smoothly with increasing positions. Hence, word representations in different positions

* First two authors contributed equally.
† Corresponding author: pzhang@tju.edu.cn
‡ The code is on [https://github.com/iclr-complex-order/complex-order](https://github.com/iclr-complex-order/complex-order)
can correlate with each other in a continuous function. Fig. 1 illustrates this type of word representation with a three-dimensional complex-valued embedding, where the amplitudes \{r_1, r_2, r_3\} denote semantic aspects corresponding to classical word vectors, and periods \{p_1, p_2, p_3\} denote how sensitive the word is to positional information. We further discuss the necessary properties of these functions to model sequential information and obtain a general solution in the form of a complex-valued embedding. Interestingly, there is a direct connection between a specific case of our general embedding and the well-known positional encoding in Vaswani et al. (2017) (see App. A).

We contribute (i) a novel paradigm that extends word vectors as continuous functions over changing variables like word position, and (ii) a general word embedding that models word order in a mathematically-sound manner. We integrate our complex word embeddings in state-of-the-art (SOTA) NN architectures (CNN, RNN, Transformer and experimentally find that it yields gains over both classical word embeddings and position-enriched word embeddings in text classification, machine translation and language modeling. Note that this is the first work in NLP to link imaginary numbers in complex-valued representation to concrete meanings (i.e., word order).

2 MODELLING WORD ORDER IN EMBEDDING SPACE

A Word Embedding (WE) generally defines a map \( f_{we} : \mathbb{N} \rightarrow \mathbb{R}^D \) from a discrete word index to a \( D \)-dimensional real-valued vector and \( \mathbb{N} = \{0, 1, 2, \ldots\} \). Similarly, a Position Embedding (PE) (Gehring et al. [2017] Vaswani et al. [2017]) defines another map \( f_{pe} : \mathbb{N} \rightarrow \mathbb{R}^D \) from a discrete position index to a vector. The final embedding for word \( w_j (w_j \in \mathbb{W} \text{ with index } j \text{ in a given vocabulary } \mathbb{W}) \) in the \( pos \)-th position in a sentence is usually constructed by the sum

\[
 f(j, pos) = f_{we}(j) + f_{pe}(pos),
\]

and \( f(j, pos) \in \mathbb{R}^D \). Since both the word embedding map \( f_{we} \) and the position embedding map \( f_{pe} \) only take integer values as word indexes or position indexes, embedding vectors for individual words or positions are trained independently. The independent training for each word vector is reasonable, since a word index is based on the order of a given arbitrary vocabulary and does not capture any specific sequential relationship with its neighboring words. However, the position index captures an ordered relationship, for instance adjacency or precedence, leading to the problem that position embeddings in individual positions (Gehring et al. [2017]) are independent of each other; the ordered relationship between positions is not modelled. We refer to this as the position independence problem. This problem becomes more crucial when position embeddings are used in position-insensitive NNs, e.g., FastText (Mikolov et al. [2013]), ConvSeq (Gehring et al. [2017]) and Transformer (Vaswani et al. [2017]), because it is hard for such position-insensitive NNs with vanilla position embeddings (Gehring et al. [2017]) to infer that \( w_{j_1} \) in the \( pos \)-th position is close to \( w_{j_2} \) in the \( pos + 1 \)-th position, or that \( w_{j_1} \) precedes \( w_{j_2} \); instead, it is only inferred that \( w_{j_1} \) and \( w_{j_2} \) are in different positions, while the relative distance between them is almost unknown. Thus vanilla position embeddings (Gehring et al. [2017]) cannot fully capture the sequential aspect of language.

Next, we first introduce the necessary properties to model word order in embeddings, and then give a unique solution to meet such properties.
2.1 Extending Vectors to Functions

In the general definition in Eq. \([1]\) each dimension of the position embedding is obtained based on the discrete position indexes \(\{0, 1, 2, \ldots, \text{pos}, \ldots\}\). This makes it difficult to model the ordered relationship between the positions. One solution to this problem is to build continuous functions over a variable (i.e., position index) to represent a specific word in an individual dimension. Formally, we define a general embedding as

\[
    f(j, \text{pos}) = g_j(\text{pos}) \in \mathbb{R}^D, \tag{2}
\]

where \(g_j\) is short for \(g_{\text{we}}(j) \in (\mathcal{F})^D\), indicating \(D\) functions over position index \(\text{pos}\), and \(g_{\text{we}}(\cdot) : \mathbb{N} \rightarrow (\mathcal{F})^D\) is a mapping from a word index to \(D\) functions. By expanding the \(D\) dimension of \(g_j\), a word \(w_j\) in the \(\text{pos}\)-th position can be represented as a \(D\)-dimensional vector as shown in

\[
    [g_{j,1}(\text{pos}), g_{j,2}(\text{pos}), \ldots, g_{j,D}(\text{pos})] \in \mathbb{R}^D, \tag{3}
\]

in which \(\forall g_{j,d}(\cdot) \in \mathcal{F} : \mathbb{N} \rightarrow \mathbb{R}, d \in \{1, 2, \ldots, D\}\) is a function over the position index \(\text{pos}\). To move the word \(w_j\) from the current position \(\text{pos}\) to another one \(\text{pos}'\), it needs only replace the variable \(\text{pos}\) to \(\text{pos}'\) without changing \(g_j\).

Functions for words, especially continuous functions, are expected to capture smooth transformation from a position to its adjacent position therefore modeling word order. The position-independent position embedding (Gehring et al., 2017) can be considered as a special case of our definition when it only takes independent values for individual positions in the embedding function.

2.2 Properties for the Functions to Capture Word Order

Relative distance is hard to compute because position indices are not visible in NNs after vector embedding (discrete position indices are necessarily embedded as vectors like words to be back-propagated with the gradient). Hence, we claim that the modeling of relative distance in NNs should be position-free: absolute position indices cannot be directly accessed in intermediate layers. Instead of processing position-free operations in NNs to capture relative distance between words, prior work (Shaw et al., 2018; Dai et al., 2019) first calculates the relative distance between words, and then feeds the relative distance as an additional feature or as embeddings/weights to NNs, instead of directly feeding with the raw position indices.

Assume that words are embedded into \(\mathbb{R}^D\), and let, for \(1 \leq d \leq D\), the function \(g_{j,d} : \mathbb{N} \rightarrow \mathbb{R}\) be the embedding function giving the \(d\)-th coordinate of the representation of word \(w_j\) (i.e., \(g_{j,d}(\text{pos})\)) is the \(d\)-th coordinate of the embedding of \(w_j\) if it occurs at position \(\text{pos}\). In the following, we simply write \(g\) instead of \(g_{j,d}\) when there is no risk of confusion. Ideally, one would like there to exist a function \(\text{Transform}_n : \mathbb{R} \rightarrow \mathbb{R}\) that transforms the embedding of any word at some position \(\text{pos}\) to the embedding of a word at position \(\text{pos} + n\) such that \(\text{Transform}_n\) is only dependent on the embedded value itself, but independent of the position \(\text{pos}\), that is \(\forall \text{pos} : g(\text{pos} + n) = \text{Transform}_n(g(\text{pos}))\).

Prior work in NLP (Li et al., 2019), Information Retrieval (Van Rijsbergen, 2004) and Machine Learning (Trabelsi et al., 2017) has shown the usefulness of complex numbers as richer representations. Complex word embeddings (Wang et al., 2019; Li et al., 2019; Li et al., 2018) have been used to model language. To investigate the potential of complex-valued representation, we extend the target domains of \(g(\cdot)\) from \(\mathbb{R}^D\) to \(\mathbb{C}^D\) without losing generality, since real-valued numbers are specific complex numbers with their imaginary part being zero. This property regarding “position-free offset transformation” in complex-valued domains is formally defined in Property 1 below.

**Property 1. Position-free offset transformation:** An embedding function \(g : \mathbb{N} \rightarrow \mathbb{C}\) is said to be a position-free offset transformation if there exists a function \(\text{Transform} : \mathbb{N} \times \mathbb{C} \rightarrow \mathbb{C}\) (called the witness) such that for all \(n \geq 1\), the function \(\text{Transform}_n(\cdot) = \text{Transform}(n, \cdot)\) satisfies \(\forall \text{pos} \in \mathbb{N} : g(\text{pos} + n) = \text{Transform}_n(g(\text{pos}))\). A position-free offset transformation \(g\) is said to be linearly witnessed if there is a function \(w : \mathbb{N} \rightarrow \mathbb{C}\) such that \(g\) has a witness \(\text{Transform}\) satisfying, for all \(n\), \(\text{Transform}(n, \text{pos}) = \text{Transform}_n(\text{pos}) = w(n)\) (i.e., each \(\text{Transform}_n\) is a linear function).

Additionally, a boundedness property is necessary to ensure that the position embedding can deal with text of any length (\(\text{pos}\) could be large in a long document).

**Property 2. Boundedness:** The function over the variable position should be bounded, i.e. \(\exists \delta \in \mathbb{R}^+, \forall \text{pos} \in \mathbb{N}, |g(\text{pos})| \leq \delta\).
Formally, we prove the following claim that there is a unique solution that meets Properties [1] and [2] under the condition that the embedding function is linearly witnessed. We use linear functions because they are well-understood and simple with a single floating-point operation in NNs.

**Claim 1.** A function \( g : \mathbb{N} \rightarrow \mathbb{C} \) is a bounded and linearly witnessed position-free offset transformation iff it is on the form \( g(pos) = z_2z_1^{pos} \) for \( z_1, z_2 \in \mathbb{C} \) with \( |z_1| \leq 1 \).

**Proof.** Assume that \( g \) is a bounded and linearly witnessed position-free offset transformation. Then, by linear witnessing, we have for all pos, \( n_1, n_2 \in \mathbb{N} \):

\[
w(n_1)w(n_2)g(pos) = w(n_2)g(pos + n_1) = g(pos + n_1 + n_2)
\]

whence \( w(n_1 + n_2) = w(n_1)w(n_2) \). Write \( w(1) = z_1 \) and \( g(0) = z_2 \). As \( n_1, n_2 \in \mathbb{N} \) were arbitrary, we have \( w(n) = (w(1))^n = z_1^n \) for all \( n \in \mathbb{N} \). But then \( g(pos + n) = w(n)g(pos) = z_1^n g(pos) \).

Furthermore, observe that for \( pos \geq 1 \), we have \( g(pos) = g(1 + pos - 1) = w(pos)g(0) = z_1^{pos} z_2 = z_2z_1^{pos} \). For \( pos = 0 \), \( g(0) = z_2 = z_2z_1^0 \), whence \( g(pos) = z_2z_1^{pos} \), as desired. Observe that if \( |z_1| > 1 \), then \( g(pos) \) is unbounded, whence we have \( |z_1| \leq 1 \). Conversely, assume that \( g \) is on the form \( g(pos) = z_2z_1^{pos} \) with \( |z_1| \leq 1 \). Then, \( |g(pos)| \leq |z_2| z_1^{pos} \), hence \( g \) is bounded. Define, for each \( n \in \mathbb{N} \), \( w(n) = z^n_1 \) and \( \text{Transform}_n(pos) = w(n)pos \). Then, for all \( pos, n \in \mathbb{N} \),

\[
g(pos + n) = z_2z_1^{pos+n} = z_2z_1^{pos} z_1^n = g(pos)z_1^n = \text{Transform}_n(g(pos))
\]

showing that \( g \) is a linearly witnessed position-free offset transformation. \( \square \)

For any \( z \in \mathbb{C} \), we may write \( z = re^{i\theta} = r(\cos \theta + i \sin \theta) \). Thus, for the general form of the embedding \( g \) from Theorem [3] we have:

\[
g(pos) = z_2z_1^{pos} = re^{i\theta_1}e^{i\theta_1}e^{i\theta_2}e^{i(\theta_2 + \theta_1)pos} = r_2^pe^{i(\theta_2 + \theta_1)pos}
\]

subject to \( |r_1| \leq 1 \) (4)

In implementations, the above definition of \( g \) will lead to an optimization problem due to the constraint \( |r_1| \leq 1 \). A natural and simple way to avoid this is to fix \( r_1 = 1 \); note that \( e^{ix} \equiv 1 \), thus automatically satisfying the constraint, in contrast to a real-valued embedding where one would need to explicitly devise functions satisfying the constraint. Finally, Eq. [4] can be written in the simplified form: \( g(pos) = re^{i(pos+\theta)} \). Thus, one can think of \( g \) as embedding positions counterclockwise on a complex circle of radius \( r \) with a fixed period \( (r \text{ is the amplitude term, } \theta \text{ is the initial phase term, } \frac{\omega}{2\pi} \text{ is the frequency, and } \frac{\omega}{2} \text{ is the period term}) \)

### 2.3 Complex-Valued Word Embedding

We now define our complex-valued word embedding \( g \) as a map taking a word index, \( j \) and position word index \( pos \) to \( \mathbb{C}^d \). For a word \( w_j \) in position \( pos \), our general complex-valued embedding is defined as \( f(j, pos) = g_j(pos) = r_j e^{i(\omega_j pos + \theta_j)} \). Therefore, \( f(j, pos) \) is defined as:

\[
[r_{j,1} e^{i(\omega_{j,1} pos + \theta_{j,1})}, ..., r_{j,K} e^{i(\omega_{j,K} pos + \theta_{j,K})}, ..., r_{j,D} e^{i(\omega_{j,D} pos + \theta_{j,D})}]
\]

(5)

Note that each coordinate \( d \) (1 \( \leq d \leq D \)) has a separate amplitude \( r_{j,d} \), period \( p_{j,d} = \frac{2\pi}{\omega_{j,d}} \), and initial phase \( \theta_{j,d} \). In Fig. [4] each dimension is represented as a wave which is parameterized by an amplitude, a period/frequency, and an initial phase. The trainable parameters of the embedding are the amplitudes vector \( r_j = [r_{j,1}, ..., r_{j,D}] \), the period/frequency related weights \( \omega_j = [\omega_{j,1}, ..., \omega_{j,D}] \), and the initial phase vector \( \theta_j = [\theta_{j,1}, ..., \theta_{j,D}] \). Note that the mean values of \( f(j, \cdot) \) over all positions are linearly dependent on the amplitude. Observe that the period/frequency determines to what degree the word is sensitive to the position. With an extremely long period (i.e., \( \omega_j \) very small), the complex-valued embedding is approximately constant for all possible values of \( pos \), and hence approximates a standard word embedding. Conversely, if the period is short, the embedding will be highly sensitive to the position argument.

In our embedding, the mean vectors of \( f(j, \cdot) \) taken over all positions are linearly correlated to the amplitude embedding \( r_j = [r_{j,1}, ..., r_{j,K}] \) with a coefficient \( \frac{2}{K} \). The amplitude \( r_{j,d} \) of our embedding depends only on the word \( w_j \) (and coordinate \( d \)), not on the position of the word, whence one can
Table 1: Dataset Statistics. CV means 10-fold cross validation. The last 2 datasets come with train/dev/test splits.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>train</th>
<th>test</th>
<th>vocab.</th>
<th>task</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR (Hu &amp; Liu 2014)</td>
<td>4K</td>
<td>CV</td>
<td>6K</td>
<td>product reviews</td>
<td>2</td>
</tr>
<tr>
<td>MPQA (Wray et al. 2005)</td>
<td>11k</td>
<td>CV</td>
<td>6K</td>
<td>opinion polarity</td>
<td>2</td>
</tr>
<tr>
<td>SUBJ (Pang &amp; Lee 2005)</td>
<td>10k</td>
<td>CV</td>
<td>21k</td>
<td>subjectivity</td>
<td>2</td>
</tr>
<tr>
<td>MR (Pang &amp; Lee 2005)</td>
<td>11.9k</td>
<td>CV</td>
<td>20k</td>
<td>movie reviews</td>
<td>2</td>
</tr>
<tr>
<td>SST (Socher et al. 2013)</td>
<td>67k</td>
<td>2.2k</td>
<td>18k</td>
<td>movie reviews</td>
<td>2</td>
</tr>
<tr>
<td>TREC (Li &amp; Roth 2002)</td>
<td>5.4k</td>
<td>0.5k</td>
<td>10k</td>
<td>Question</td>
<td>6</td>
</tr>
</tbody>
</table>

3 EXPERIMENTAL EVALUATION

We evaluate our embeddings in text classification, machine translation and language modeling.

3.1 TEXT CLASSIFICATION

Experimental Setup. We use six popular text classification datasets: CR, MPQA, SUBJ, MR, SST, and TREC (see Tab. 1). We use accuracy as evaluation measure based on fixed train/dev/test splits or cross validation, as per prior work. We use Fasttext (Joulin et al. 2016), CNN (Kim 2014), LSTM and Transformer (Vaswani et al. 2017) as NN baselines. We use each of them: (1) without positional information; (2) with Vanilla Position Embeddings (PE) (randomly initialized and updated during training using the sum between word and position vectors (Gehring et al. 2017); (3) with Trigonometric Position Embeddings (TPE) (defined position embeddings as trigonometric functions as per Eq. 6); (4) with Complex-vanilla word embeddings (where the amplitude embedding is initialized by the pre-trained word vectors, and the phase embedding is randomly initialized in a range from $-\pi$ to $\pi$ without considering word order (Wang et al. 2019)); and (5) with our order-aware complex-valued word embeddings, Complex-order (which encode position in the phase parts, train the periods, and where the amplitude embedding is also initialized by pretrained word vectors). For more details on the complex-valued extensions of NNs, see App. B and App. C.

Our embedding generally has $3 \times D \times |W|$ parameters with D-dimensional word vectors and $|W|$ words, while previous work (Mikolov et al. 2013b, Pennington et al. 2014) usually employs only $D \times |W|$ parameters for embedding lookup tables. To increase efficiency and facilitate fair comparison with previous work we set initial phases $\theta_j = [\theta_{j,1}, ..., \theta_{j,D}]$ to a shared constant value (such as zero). Furthermore, the period vectors $\omega_{j,d}$ depend on word index $j$ with length $|W|$ and the coordinate index $d$ with length $D$. To decrease the number of parameters, one can either use a word-sharing scheme (i.e., $\omega_{j,d} = \omega_{.,d}$), or a dimension-sharing scheme ($\omega_{j,d} = \omega_{.,.}$), leading to $|W| + D + |W|$ and $|W| + D + D$ parameters in total for the embedding layer.

Published as a conference paper at ICLR 2020
Table 2: Text classification accuracy without position embeddings, with random position embeddings (PE), with trigonometric position embeddings (TPE), with complex-valued NNs without position embeddings (complex-vanilla), and with our complex-order embeddings. 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.

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

Table 3: Text classification accuracy. ⋆ means that scores are reported from other papers.

<table>
<thead>
<tr>
<th>Method</th>
<th>MR</th>
<th>SUBJ</th>
<th>CR</th>
<th>MPQA</th>
<th>SST</th>
<th>TREC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word2vec Bow (Conneau et al., 2017)</td>
<td>0.777</td>
<td>0.909</td>
<td>0.798</td>
<td>0.883</td>
<td>0.797</td>
<td>0.836</td>
</tr>
<tr>
<td>Sent2Vec (Pagliardini et al., 2017)</td>
<td>0.763</td>
<td>0.912</td>
<td>0.791</td>
<td>0.872</td>
<td>0.802</td>
<td>0.858</td>
</tr>
<tr>
<td>QuickThoughts (Logeswaran &amp; Lee, 2018)</td>
<td>0.824</td>
<td>0.948</td>
<td>0.860</td>
<td>0.902</td>
<td>-</td>
<td>0.928</td>
</tr>
<tr>
<td>InferSent (Conneau et al., 2017) ⋆</td>
<td>0.811</td>
<td>0.924</td>
<td><strong>0.863</strong></td>
<td>0.902</td>
<td>0.846</td>
<td>0.882</td>
</tr>
<tr>
<td>QPDN (Wang et al., 2019) ⋆</td>
<td>0.801</td>
<td>0.927</td>
<td>0.810</td>
<td>0.870</td>
<td>0.839</td>
<td>0.882</td>
</tr>
</tbody>
</table>

We search the hyper parameters from a parameter pool, with batch size in {32, 64, 128}, learning rate in {0.001, 0.0001, 0.00001}, L2-regularization rate in {0, 0.001, 0.0001}, and number of hidden layer units in {120, 128}. We use pre-trained 300-dimensional vectors from word2vec (Mikolov et al., 2013a) in all models except for Transformers. The models with trainable trigonometric position embedding produce nearly identical results compared to the non-trainable version, therefore we report the result of fixed position embeddings as per Vaswani et al. (2017). We adopt narrow convolution and max pooling in CNN, with number of filters in {64, 128}, and size of filters in {3, 4, 5}. In all Transformer models, we only use the encoder layer to extract feature information, where the layer is 1, dimension of word and inner hidden are 256 and 512 respectively, and head number is 8.

**Results.** The results are shown in Tab. 2. Our complex-order embeddings outperform all other variations at all times. This gain in effectiveness comes at a negligible (or non-existent) cost in efficiency (it varies per NN architecture – see Fig. 2). CNNS are the best performing NN as expected following Bai et al. (2018). Transformer NNs benefit the most from our complex-order embeddings, most likely because they are our weakest baseline. To contextualize these results, Tab. 3 shows classification accuracy of five typical approaches on the same datasets (as reported in the original papers). Our complex-order embeddings outperform all methods, except for the CR dataset, where InferSent is marginally better. Overall, our approach is on a par with the SOTA in embeddings.

We perform an ablation test (Tab. 4) on Transformer because it is the most common NN to be used with position embeddings. The two period-sharing schemas (dimension-sharing and word-sharing)
slightly drop performance, because fewer parameters limit the representative power. Adding initial phases also hurts performance, although we observed that the loss could decrease faster in early epochs compared to the setting without offset. The negative effect of initial phases may be due to periodicity, and $\omega$ cannot be directly regularized with L2-norm penalties. The sharing schemes slightly decrease the performance with less parameters. More details of the learned periods/frequencies (e.g. the distributions of periods/frequencies and case studies) are shown in App. D.

Note that the word-sharing schema outperform the Vanilla Transformer, (both have a comparable number of parameters). If we choose $\Re(W_{Q/K/V}) = \Im(W_{Q/K/V})$, the additional parameters in the embedding layers will affect much less the whole parameter scale in the multiple-layer Transformer, since a embedding layer is only used in the first layer instead of the following Transformer layers.

3.2 MACHINE TRANSLATION

Experimental Setup. We use the standard WMT 2016 English-German dataset (Sennrich et al., 2016), whose training set consists of 29,000 sentence pairs. We use four baselines: basic Attentional encoder-decoder (AED) (Bahdanau et al., 2014); AED with Byte-pair encoding (BPE) subword segmentation for open-vocabulary translation (Sennrich et al., 2016); AED with extra linguistic features (morphological, part-of-speech, and syntactic dependency labels) (Sennrich & Haddow, 2016); and a 6-layer Transformer. Our approach (Transformer Complex-order) uses a batch size of 64, a head of 8, 6 layers, the rate of dropout is 0.1, and the dimension of the word embedding is 512. The embedding layer does not use initial phases, i.e., following $f(j, \text{pos}) = r_j e^{i(\omega_j \text{pos})}$. We evaluate MT performance with the Bilingual Evaluation Understudy (BLEU) measure.
Table 5: Machine translation results. ⋆ marks scores reported from other papers.

<table>
<thead>
<tr>
<th>Method</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>AED (Bahdanau et al., 2014) ⋆</td>
<td>26.8</td>
</tr>
<tr>
<td>AED+Linguistic (Sennrich &amp; Haddow, 2016) ⋆</td>
<td>28.4</td>
</tr>
<tr>
<td>AED+BPE (Sennrich et al., 2016) ⋆</td>
<td>34.2</td>
</tr>
<tr>
<td>Transformer (Ma et al., 2019) ⋆</td>
<td>34.5</td>
</tr>
<tr>
<td>Transformer complex vanilla</td>
<td>34.7</td>
</tr>
<tr>
<td>Transformer Complex-order</td>
<td>35.8</td>
</tr>
</tbody>
</table>

Table 6: Language modeling results. ⋆ marks scores reported from other papers.

<table>
<thead>
<tr>
<th>Method</th>
<th>BPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>BN-LSTM (Cooijmans et al., 2016) ⋆</td>
<td>1.36</td>
</tr>
<tr>
<td>LN HM-LSTM (Chung et al., 2016) ⋆</td>
<td>1.29</td>
</tr>
<tr>
<td>RHN (Zilly et al., 2017) ⋆</td>
<td>1.27</td>
</tr>
<tr>
<td>Large mLSTM (Krause et al., 2016) ⋆</td>
<td>1.27</td>
</tr>
<tr>
<td>Transformer XL 6L (Dai et al., 2019)</td>
<td>1.29</td>
</tr>
<tr>
<td>Transformer XL Complex-order 6L</td>
<td>1.30</td>
</tr>
</tbody>
</table>

Results. Tab. 5 shows the MT results. Our approach outperforms all baselines. Two things are worth noting: (1) Both the vanilla Transformer and our Transformer Complex-order outperform the three Attentional encoder-decoder baselines which are based on an LSTM encoder and decoder, even when AED uses additional features. (2) Our Transformer Complex-Order outperforms the Vanilla Transformer and complex-vanilla Transformer by 1.3 and 1.1 in absolute BLEU score respectively.

3.3 Language Modeling

Experimental Setup. We use the text8 dataset, consisting of English Wikipedia articles. The text is lowercased from a to z, and space. The dataset contains 100M characters (90M for training, 5M for dev, and 5M for testing, as per Mikolov et al. (2012)). We use as baselines BN-LSTM, LN HM-LSTM RHN and Large mLSTM, which are typical recurrent NNs for language modeling in this dataset. We evaluate performance with the Bits Per Character (BPC) measure, (the lower, the better). We ran the coder in Dai et al. (2019) with 6 layers for Transformer XL 6L; our model, named Transformer XL complex-order, directly replaces the word embedding with our proposed embedding under the same setting. We choose 6 layers due to limitations in computing resources. For Transformer XL Complex-order, all other parameter settings are as for Transformer XL (Dai et al., 2019). Our complex-order model does not use initial phases.

Results. We see in Tab. 6 that our method outperforms all baselines. The first four baselines (BN-LSTM, LN HM-LSTM, RHN and Large mLSTM) are based on recurrent NNs and rely on different regulation methods to become applicable in multiple-layer recurrent architectures. Transformer-based architectures can easily be stacked with multiple layers due to their advantages in parallelization, however the vanilla Transformer does not outperform the multiple-layer recurrent baselines, most likely due to its limitation of 6 layers. Our Transformer XL Complex-order outperforms its vanilla counterpart under the 6-layer setting (and also strong recurrent network baselines), demonstrating that our embedding also generalizes well in tasks with long-term dependency. With limited resources, slightly increasing the parameters in the feature layer like our proposed embedding could be more beneficial than stacking more layers with linearly increasing parameters.

4 Related Work

Complex-valued NNs are not new (Georgiou & Koutsougeras, 1992; Kim & Adalı, 2003; Hirose, 2003). Complex-valued weights have been used in NNs, motivated by biology (Reichert & Serre, 2013), and also as signal processing in speech recognition (Shi et al., 2006). More recently, Arjovsky et al. (2016) shifted RNNs into the complex domain and Wolter & Yao (2018) proposed a novel complex gated recurrent cell. Trabelsi et al. (2017) also developed a complex-valued NN for computer vision and audio processing.

Complex numbers have also been applied to text processing like (Van Rijsbergen, 2004; Melucci, 2015; Blacoe et al., 2013). Trouillon et al. (2016) adopt complex embedding for entities in Knowledge Graph Completion to represent antisymmetric relations with Hermitian dot product. Li et al. (2019); Wang et al. (2019) extend word embeddings to complex-valued fashion in quantum probability driven NNs, seeing the overview in Wang et al. (2019). However, the physical meaning of both the complex-valued entity and word embeddings is unknown, since a complex number was considered as two real numbers in a black-box learning paradigm. Our work first links the phase in complex numbers to word position to define concrete physical meaning in document representations.
5 CONCLUSIONS

We extended word vectors to word functions with a variable i.e. position, to model the smooth shift among sequential word positions and therefore implicitly capture relative distances between words. These functions are well-defined to model the relative distances, therefore we derive a general solution in complex-valued fashion. Interestingly, the position embedding in Vaswani et al. (2017) can be considered a simplified version of our approach. We extend CNN, RNN and Transformer NNs to complex-valued versions to incorporate our complex embedding. Experiments on text classification, machine translation and language modeling show that our embeddings are more effective than vanilla position embeddings (Vaswani et al., 2017).

ACKNOWLEDGMENTS

We thank Massimo Melucci and Emanuele Di Buccio for their helpful comments, Xindian Ma for his detailed experimental suggestions.

This work is supported by the Quantum Access and Retrieval Theory (QUARTZ) project, which has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 721321. Peng Zhang and Donghao Zhang are supported in part by Natural Science Foundation of China (grant No. 61772363, U1636203)

REFERENCES


A Linking to the Position Embeddings in (Vaswani et al., 2017)

Vaswani et al. (2017) proposed a new initialization for position embedding, resulting in comparable performance with previous one (Gehring et al., 2017) even without fine-tuning. The position embedding is empirically selected as

\[
PE_{2k}(\cdot, pos) = \sin(pos/10000^{2k/d_{model}})
\]

\[
PE_{2k+1}(\cdot, pos) = \cos(pos/10000^{2k/d_{model}})
\]

(7)

Where pos is the position index, 2k and 2k + 1 is the dimension index and \(d_{model}\) is the dimension size of embedding. The reason for choosing this position embedding was not well-explained and its general extension is unknown, leading to some difficulties to improve it.

We claim that the proposed position embedding in (Vaswani et al., 2017) is a degraded version of one of our specific complex word embedding in word-sharing schema (i.e., \(\omega_{j,d} = \omega_{.d}\)), in which \(p_{j,k} = 2\pi \times 10000^{2k/d_{model}}\) and the initial phases are set as zero. In our complex-valued position embedding, let \(f_{pe,k}(\cdot, pos) = e^{j \times 10000^{2k/d_{model}}} = \cos(10000^{2k/d_{model}} pos) + j \sin(10000^{2k/d_{model}} pos)\). Note that there exists a bi-jection between \(PE(\cdot, pos)\) and \(f_{pe,k}(\cdot, pos)\):

\[
PE_{2k}(\cdot, pos) = \Re(f_{pe,k}(\cdot, pos)),
\]

\[
PE_{2k+1}(\cdot, pos) = \Im(f_{pe,k}(\cdot, pos))
\]

(8)

where \(\Re\) and \(\Im\) are the operations to take the real and imaginary part of a complex-valued number. Its inverse transformation is

\[
f_{pe,k}(\cdot, pos) = PE_{2k+1}(\cdot, pos) + jPE_{2k}(\cdot, pos)
\]

(9)

In our overall embedding, each dimension \(f_k(j, pos) = f_{pe,k}(j) \odot f_{pe,k}(\cdot, pos)\) in our approaches, while it is \(E_k(j, pos) = W E_k(j) + PE_k(\cdot, pos)\) in (Gehring et al., 2017; Vaswani et al., 2017). Hence the position embedding in (Vaswani et al., 2017) is equivalent, albeit not identical, to our complex-valued position embedding with \(p_{j,k} = 2\pi \times 10000^{2k/d_{model}}\). It is therefore a particular case of our complex-valued position embedding, the word-sharing schema in which all words share the same period at a certain dimension, i.e., \(p_{j,k} = p_{.,k}\) is irrelevant to the choice of \(j\).

B Integrating Complex-Valued Embedding to General Neural Networks

Neural networks are typically given real numbers as inputs and return real numbers as outputs. To accommodate complex numbers as in- and output, we devise a complex-valued version of various neural network layers i.e. complex-valued FastText with dense layer, CNN, and RNN. Unlike existing complex-valued neural networks (Trabelsi et al., 2017; Wolter & Yao, 2018), our feature layers are also converted into complex-valued layers.

Complex-valued FastText

FastTest (Joanin et al. 2016) is a simple and efficient neural network architecture using a dense layer over the sum of all word embeddings for general text classification. For a linear dense layer, i.e., \(z = \text{dense}(x + iy)\), where \(x + iy\) and \(z\) denote the complex-valued in- and output, respectively. Let \(\mathbf{W} = \mathbf{A} + i\mathbf{B}\) and \(\mathbf{b} = c + id\) be complex-valued linear weights and bias, respectively. Then, the complex-valued dense layer is given by:

\[
z = \sigma(\mathbf{A}x - \mathbf{B}y + c) + i\sigma(\mathbf{B}x + \mathbf{A}y + d)
\]

(10)

where \(\sigma\) is a real-valued activation function such as the sigmoid function. By rewriting (10) in matrix form (Trabelsi et al., 2017), we obtain:

\[
\begin{bmatrix}
\Re(z) \\
\Im(z)
\end{bmatrix} = 
\begin{bmatrix}
\sigma(\mathbf{A}x - \mathbf{B}y + c) \\
\sigma(\mathbf{B}x + \mathbf{A}y + d)
\end{bmatrix}
\]

(11)

where, for \(z = x + iy\), \(\Re(z) = x\) and \(\Im(z) = y\). To save parameters and fairly compare with our real-valued baselines, the weights for real-part and imaginary-part input can be shared, i.e., \(\mathbf{A} = \mathbf{B}, c = d\).
**Complex-valued CNN**  For the complex-valued version of the convolution operation [Trabelsi et al. (2017)], we similarly define a complex-valued convolution with separate real and imaginary kernels $A$ and $B$, to compute convolutions on the real and imaginary parts of the input in Eq. 10. A complex-valued CNN network is constructed by stacking the operations based on a complex-valued convolution kernel, adding a complex dense layer in [10], and taking their norm as the final prediction in the last layer.

**Complex-valued RNN**  The basic complex RNN formulation is:

$$h_t^C = f \left( W^h h_{t-1} + W^z z_t + b \right)$$

where $z_t$ and $h_t$ represent the complex-valued input and complex-value hidden state vectors at time $t$, $b$ is a complex-valued bias, $W^h$ and $W^z$ are complex-valued weight transitions for hidden state and input state, and $f(z) = \sigma(\Re(z)) + i\sigma(\Im(z))$ is the activation function. The multiplication $W^h h_{t-1}$ and $W^z z_t$ is computed as defined in [10] above. Similarly, the complex-valued gates are used in LSTM via operations as in (12). In the final layer, an l2-norm operation is adopted to obtain a real-valued loss for backpropagation.

**Complex-valued Transformer**  The main components in the Transformer are self-attention sublayers and position-wise feed-forward (FFN) sublayers. A self-attention sublayer employs $h$ attention heads and the concatenation of all heads is used as the output followed by a parameterized linear transformation. For a sequence embedded as complex-valued vector $\text{input} = \{w_1, w_2, ..., w_n\}$, the output of each head is computed as a weighted sum of a linear transformation of the input sequence itself, namely

$$output_i = \sum_j a_{i,j} w_j W^V,$$

where $w_j$ is a complex-valued vector and $W^V$ is a complex linear transformation; therefore $output_i$ is also complex. Hence, $output$ is a sequence of complex-valued vectors with the same shape as $\text{input}$. The weight coefficient, $a_{i,j}$, which is defined as in real-valued domain, is calculated as the softmax of the product between complex-valued query vectors and key vectors:

$$a_{i,j} = \text{softmax} \left( \sum_{n=1}^n e_{i,j} \right),$$

where $z$ is a complex number since $w_i$ and $w_j$ are complex-valued vectors and $W^Q, W^K$ are complex-valued transformation. To extend another variant of Transformer called Transformer XL, we keep its original relative position embedding and additionally replace its word embedding with our proposed embedding.

Correspondingly, the FFN sublayer can easily be extended to a complex-valued version by replacing the real-valued layers with complex-valued ones. We use batch-normalization separately for the real and imaginary parts.

### C IMPLEMENTATION OF THE PROPOSED EMBEDDING

Words functions are implemented in neural networks by storing the function parameters $\{r, \omega, \theta\}$ and then construct the values based on the arguments. Based on the definition, the implementation of the proposed embedding can easily be implemented with only modifying the embedding layer.

We list the basic code to construct our general embedding as below:

```python
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)
```
Published as a conference paper at ICLR 2020

Figure 3: The distribution of the $\delta_j$. Higher values mean that the word representations are more sensitive to the word positions.

```python
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.unsqize(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)
    ...
```

Note that both the frequency vectors $\omega$ and initial-phase vectors $\theta$ can be shared between words or dimensions, to save parameters. The proposed embedding can be also used in real-valued neural networks if one directly concatenates the real-part numbers and imaginary-part numbers as a double-size real-valued vector; therefore it could easily be extended in any existing networks without any complex-valued components. For instance, it could be a good extension for Transformer based pretrained models like (Devlin et al., 2018) by enriching the feature layer.

D Visualization of Frequencies/Periods

After training, we obtain the frequency vector $\omega$ for each word. For each word, the mean value of the absolute frequency values, i.e., $\delta_j = \frac{1}{|D|} \sum_{d=1}^{D} |\omega_{j,d}|$ is considered as a metric to test the positional sensitivities of the word, since a period value could be negative during training. The density of the $\delta_j$ is shown in Fig.4.

Words with the 50 greatest, and the 50 smallest, frequencies in the SST dataset are shown in Tab.7. For the words with greatest frequencies, most of them are strong sentiment words like “worst”
“,stupid” and “powerful”; a reason for this may be that such words appear in many positions in many documents during training, and thus they are more sensitive to the positions. Conversely, there are fewer words expressing strong sentiment among words with smaller frequencies, as shown in the second row.

| greatest frequencies in descending order | worst | solid stupid powerful | mess | wonderful remarkable | suffers | intoxicating thoughtful rare | captures | portrait gem frontal | terrific unique | wannabe witty lousy pointless | contrived none worse refreshingly charming inventive amazing junk incoherent refreshing mediocre unfunny thinks enjoyed heartbreaking delightfully | crisp brilliant | heart spirit perfectly nowhere mistake engrossing fashioned excellent unexpected wonderfully means |
| smallest frequencies in ascending order | slowly proposal schemes roiling juliette titles fabric superstar ah wow choreographed tastelessness beg fabulous mucino jacobi legendary jae rate example code sensation counter deaths hall eun drug mctiernan storylines cellophane wild motion ups trick comedy entertained mission frightening witnesses snoots liners african groan satisfaction calm saturday estranged holm refuses inquisitive |

Table 7: Words with greatest frequencies and frequencies periods (based on $\delta_j$) in SST (a sentiment classification task), all words are converted to lower-case. The strong sentiment words are bold based on manual labeling.