A Linguistic Phenomena in Word Formation

In the theory of linguistic morphology, morphemes are considered to be the smallest meaning-bearing or grammatical units of a language.

In the word formation, common linguistic phenomena include inflection, derivation, and compounding, as shown in Table 1 of our paper. **(1)** Inflection occurs when a word has different forms but essentially the same meaning. For example, the underlying meanings of '`cooks`', '`cooked`', and '`cooking`' are all '`cook`', and there are only certain grammatical differences. **(2)** Derivation makes a word with a different meaning, such as '`possible`' and '`impossible`'. **(3)** Compounding occurs when multiple words are combined to form a new word. For example, '`policeman`' is obtained by compounding '`police`' and '`man`'. Based on these common phenomena, there are common connections between words due to shared morphemes, and morphologically similar words with some of the same morphemes are often semantically related. Therefore, it makes sense to segment words into morphemes to learn word representations in a more fine-grained way.

B The Example of Tensor Products

Typically, there exist two types of definitions for the tensor product. Here, we give an example for the tensor product between two vectors, but it also holds for the tensor product between matrices.

Definition 1. a tensor product between an m-sized vector and an n-sized vector results in an $m \times n$ matrix.

$$\begin{aligned} \mathbf{a} \otimes^1 \mathbf{b} &= \begin{bmatrix} a_1 \\ \\ a_2 \end{bmatrix} \otimes \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} \\ &= \begin{bmatrix} a_1 b_1 & a_1 b_2 & a_1 b_3 \\ \\ a_2 b_1 & a_2 b_2 & a_2 b_3 \end{bmatrix} \end{aligned} \tag{9}$$

Definition 2. a tensor product between an m-sized vector and an n-sized vector results in an mn-sized vector.

$$\begin{aligned} \mathbf{a} \otimes^2 \mathbf{b} &= \begin{bmatrix} a_1 & a_2 \end{bmatrix} \otimes \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} \\ &= \begin{bmatrix} a_1 \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} & a_2 \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} \end{bmatrix} \\ &= \begin{bmatrix} a_1 b_1 & a_1 b_2 & a_1 b_3 & a_2 b_1 & a_2 b_2 & a_2 b_3 \end{bmatrix} \end{aligned} \tag{10}$$

Note that Definition 1 increases the dimensions of the tensors while Definition 2 does not. As formulated in Eq. 2 in our paper, we use Definition 2 without dimension increase if not specified. The difference between Definition 1 and Definition 2 is that Definition 2 further **reshapes** the RHS of Definition 1 (previously a two-dimensional tensor, i.e., a matrix) into a flattened vector. Since a word embedding is a vector in neural networks, we make use of Definition 2 so that any resulted embeddings for words are necessarily vectors.

As formulated in Eq. 2, a tensor product between a g-sized vector and h-sized vector results in a gh-sized vector. Similarly, a cumulative tensor product of n q-size vectors will be a q^n-sized vector. We show a concrete example of cumulative tensor product with $n = 3$ and $q = 2$ as follow:

$$\begin{aligned} &\mathbf{a} \otimes \mathbf{b} \otimes \mathbf{c} \\ &= \begin{bmatrix} a_1 & a_2 \end{bmatrix} \otimes \begin{bmatrix} b_1 & b_2 \end{bmatrix} \otimes \begin{bmatrix} c_1 & c_2 \end{bmatrix} \\ &= \begin{bmatrix} a_1 \begin{bmatrix} b_1 & b_2 \end{bmatrix} & a_2 \begin{bmatrix} b_1 & b_2 \end{bmatrix} \end{bmatrix} \otimes \begin{bmatrix} c_1 & c_2 \end{bmatrix} \\ &= \begin{bmatrix} a_1 b_1 & a_1 b_2 & a_2 b_1 & a_2 b_2 \end{bmatrix} \otimes \begin{bmatrix} c_1 & c_2 \end{bmatrix} \\ &= \begin{bmatrix} a_1 b_1 \begin{bmatrix} c_1 & c_2 \end{bmatrix} & a_1 b_2 \begin{bmatrix} c_1 & c_2 \end{bmatrix} & a_2 b_1 \begin{bmatrix} c_1 & c_2 \end{bmatrix} & a_2 b_2 \begin{bmatrix} c_1 & c_2 \end{bmatrix} \end{bmatrix} \\ &= \begin{bmatrix} a_1 b_1 c_1 & a_1 b_1 c_2 & a_1 b_2 c_1 & a_1 b_2 c_2 & a_2 b_1 c_1 & a_2 b_1 c_2 & a_2 b_2 c_1 & a_2 b_2 c_2 \end{bmatrix} \end{aligned} \tag{11}$$

Table 8: Parameter complexities for different word embedding methods. $|V|$ and d are the word vocabulary size and word embedding size, respectively. r and n are the rank and order. $|M|$ is the morpheme vocabulary size.

Method	Parameter	**Method**	Parameter	**Method**	Parameter
Original	$\|V\|d$	Word2ket	$rn\|V\|\sqrt[n]{d}$	Word2ketXs	$rn\sqrt[n]{\|V\|}\sqrt[n]{d}$
MorphTE	$\|M\|\sqrt[n]{d}r + \|V\|n$	Matrix Factor.	$r\left(\|V\| + d\right)$	Tensor Train	$\left((n-2)\,r^2 + 2r\right)\ \sqrt[n]{\|V\|}\sqrt[n]{d}$
MorphSum	$\left(\|V\| + \|M\|\right)d$	MorphLSTM	$\|M\|d + 8d^2$		

C Analysis of Parameter

In the experiments of the paper, we directly report the specific number of parameters for different word embedding methods. To illustrate how these parameter numbers are calculated, we report the parameter complexity and settings of different methods in Section C.1 and Section C.2, respectively.

C.1 Parameter Complexity of Embedding Methods

We summarize the parameter complexities of different embedding methods in Table 8. Suppose the **word vocabulary size** is $|V|$ and the **word embedding size** (dimensionality) is d. Suppose the **morpheme vocabulary size** is $|M|$, which involves embedding methods utilizing morphology. Suppose the **rank** is r and the **order** is n, which involves decomposition-based embedding compression methods. Note that the rank and order have different meanings in different decomposition methods. The value of the order is generally 2 to 4. The number of parameters of the decomposition method is generally controlled by adjusting the rank, and the value of rank may vary greatly in different methods. The detailed analysis of the parameter complexity is as follows:

- **Original**: The original word embedding layer has a word embedding matrix, each row of which is the embedding (vector) of a word. The parameter number of the embedding matrix is $|V|d$.
- **MorphSum**: MorphSum needs to specify a unique embedding for each morpheme and word, resulting in the parameter number of $\left(|V| + |M|\right)d$.
- **MorphLSTM**: MorphLSTM needs to specify a unique embedding for each morpheme, resulting in the parameter number of $|M|d$. In addition, the LSTM network in MorphLSTM requires $8d^2$ parameters. Therefore, the total parameter number is $|M|d + 8d^2$.
- **Matrix Factor.**: The low–rank matrix factorization method decomposes the original word embedding matrix into the product of two small matrices of size $|V| \times r$ and $r \times d$. r is called the rank of this method. Therefore, the number of parameters required by this method is $r\left(|V| + d\right)$.
- **Word2ket**: Referring to Eq. 4 in the paper, Word2ket represents the embedding of a word as an entangled tensor of rank r and n. The number of parameters required by this method is at least $rn|V|\sqrt[n]{d}$.
- **Word2ketXs**: Word2ketXs compresses both the vocabulary size ($|V|$) and the word embedding size (d), while Word2ket only compresses the word embedding size **(**d**)** . The number of parameters required by this method is at least $rn\sqrt[n]{|V|}\sqrt[n]{d}$.
- **Tensor Train**: Like word2ketXS, Tensor Train method compresses both the vocabulary size ($|V|$) and the word embedding size (d). This method compresses the embedding matrix into 2 tensors of sizes $|V|_1 \times d_1 \times r$ and $|V|_n \times d_n \times r$, and $n-2$ tensors with size of $|V|_i \times d_i \times r^2$ $(1 < i < n)$, where $|V| = \prod_{k=1}^{n} |V|_k$, and $d = \prod_{k=1}^{n} d_k$. To simplify the calculation, let $|V|_1 = \ldots = |V|_n = \sqrt[n]{|V|}$ and $d_1 = \ldots = d_n = \sqrt[n]{d}$, and then the number of parameters of this method is $\left((n-2)\,r^2 + 2r\right)\ \sqrt[n]{|V|}\sqrt[n]{d}$.
- **MorphTE**: Similar to Word2ket, the size of the morpheme vector is at least $\sqrt[n]{d}$. Referring to Section 4, there are r morpheme embedding matrices of size $|M|\sqrt[n]{d}$, resulting in $|M|\sqrt[n]{d}r$ trainable parameters. In addition, storing the morpheme index matrix also requires additional constant parameters of $|V|n$. Therefore, the total parameter number is $|M|\sqrt[n]{d}r + |V|n$.

C.2 Settings of Embedding Methods

We show detailed settings of different embedding methods on translation tasks in Tables 9 and 10, and these two tables correspond to Table 2 of the paper. We show detailed settings of different embedding methods on WikiQA of question answering tasks and SNLI of natural language inference tasks in Tables 11 and 12. These two tables correspond to Table 3 of the paper.

Table 9: Detailed settings of Original, Matrix Factor., and Tensor Train methods on translation tasks, corresponding to Table 2. **De**-En and De-**En** mean the settings of the source language (De) and target language (En), respectively. Since other datasets (e.g., En-It) use shared source-target dictionaries, only one set of settings is reported. $|V|$ is the size of the word vocabulary and d is the size (dimensionality) of a word embedding. #Emb is the parameter number of the word embedding layer. n and r are the order and rank of decomposition methods, respectively. $\Delta|V|$ and Δd are the decomposition of the word vocabulary size ($|V|$) and word embedding dimensionality (d) by the decomposition method, respectively. For example, in Wordket, Δd is $(8, 8, 8)$, because a word embedding is constructed from three ($n = 3$) 8-size vectors.

Dataset	Original			Matrix Factor.				Tensor Train						
	$\|V\|$	d	#Emb	20×		40×		n	$\Delta\|V\|$	Δd	20×		40×	
				r	#Emb	r	#Emb				r	#Emb	r	#Emb
De-En	8848	512	4.53M	25	0.23M	12	0.11M	3	(18,20,25)	(8,8,8)	34	0.20M	23	0.09M
De-**En**	6632	512	3.40M	25	0.19M	12	0.09M	3	(14,20,25)	(8,8,8)	34	0.19M	23	0.09M
En-It	41280	512	21.14M	25	1.04M	12	0.50M	3	(30,35,40)	(8,8,8)	59	1.01M	41	0.49M
En-Es	41336	512	21.16M	25	1.05M	12	0.50M	3	(30,35,40)	(8,8,8)	59	1.01M	41	0.49M
En-Ru	42000	512	21.50M	25	1.06M	12	0.51M	3	(30,35,40)	(8,8,8)	59	1.01M	43	0.54M

Table 10: Detailed settings of Word2ketXs, Word2ket, and MorphTE methods on translation tasks, corresponding to Table 2. $|M|$ is the size of the morpheme vocabulary. Refer to Table 9 for the meaning of other fields.

Dataset	Word2ketXs							Word2ket				MorphTE						
	n	$\Delta\|V\|$	Δd	20×		40×		n	Δd	20×		$\|M\|$	n	Δd	20×		40×	
				r	#Emb	r	#Emb			r	#Emb				r	#Emb	r	#Emb
De-En	2	(95,95)	(16,32)	44	0.20M	22	0.10M	3	(8,8,8)	1	0.21M	3013	3	(8,8,8)	7	0.20M	3	0.10M
De-**En**	2	(82,82)	(16,32)	44	0.17M	22	0.09M	3	(8,8,8)	1	0.16M	2744	3	(8,8,8)	7	0.17M	3	0.08M
En-It	2	(205,205)	(16,32)	104	1.02M	50	0.49M	3	(8,8,8)	1	0.99M	10818	3	(8,8,8)	10	0.99M	4	0.45M
En-Es	2	(205,205)	(16,32)	104	1.02M	50	0.49M	3	(8,8,8)	1	0.99M	11377	3	(8,8,8)	10	1.03M	4	0.49M
En-Ru	2	(205,205)	(16,32)	104	1.02M	53	0.52M	3	(8,8,8)	1	1.01M	12423	3	(8,8,8)	9	1.02M	4	0.52M

Table 11: Detailed settings of Original, Matrix Factor., and Tensor Train methods on WikiQA and SNLI tasks, corresponding to Table 3. Refer to Table 9 for the meaning of the fields in this table.

Dataset	Original			Matrix Factor.		Tensor Train				
	$\|V\|$	d	#Emb	r	#Emb	n	$\Delta\|V\|$	Δd	r	#Emb
WikiQA	12333	512	6.314M	6	0.077M	3	(20,25,26)	(8,8,8)	19	0.079M
SNLI	16936	512	8.671M	13	0.227M	3	(25,25,32)	(8,8,8)	33	0.233M

Table 12: Detailed settings of Word2ketXs, Word2ket, and MorphTE methods on WikiQA and SNLI tasks, corresponding to Table 3. $|M|$ is the size of the morpheme vocabulary. Refer to Table 9 for the meaning of the fields in this table.

Dataset	Word2ketXs					Word2ket				MorphTE				
	n	$\Delta\|V\|$	Δd	r	#Emb	n	Δd	r	#Emb	$\|M\|$	n	Δd	r	#Emb
WikiQA	3	(24,24,24)	(8,8,8)	137	0.079M	3	(8,8,8)	1	0.296M	5152	3	(8,8,8)	1	0.078M
SNLI	3	(8,29,73)	(8,8,8)	260	0.229M	3	(8,8,8)	1	0.406M	5572	3	(8,8,8)	4	0.229M

D Implementation and Training Details

In this section, we report the hyperparameter details of the experiments in our paper. The hyperparameter details of implementation and training on machine translation tasks are shown in Table 13. The hyperparameter details of implementation and training on question answering and natural language inference tasks are shown in Table 14.

Table 13: Hyperparameter details of implementation and training on translation tasks. IWSLT'14 means the De-En dataset, and OPUS-100 means the En-It, En-Es, and En-Ru datasets.

Hyperparameter	IWSLT'14	OPUS-100
num encoder layers	6	6
num decoder layers	6	6
embedding size	512	512
feed-forward size	1024	2048
attention heads	4	8
learning rate	5e-4	7e-4
label smoothing	0.1	0.1
max tokens per batch	4096	36864
dropout	0.3	0.25
Adam-betas	(0.9, 0.98)	(0.9, 0.98)
warmup steps	20000	20000
weight decay	0.0001	0.0
beam size	5	5

Table 14: Hyperparameter details of implementation and training on question answering and natural language inference tasks. QA means the question answering task of the WikiQA dataset, and NLI means the natural language inference task of the SNLI dataset.

Hyperparameter	QA	NLI
num encoder blocks	2	3
num convolutional encoders	3	2
kernel size	3	3
embedding size	512	512
hidden size	200	150
learning rate	1e-3	2e-3
batch size	400	512
dropout	0.1	0.1
Adam-betas	(0.9, 0.999)	(0.9, 0.999)
weight decay	0.0	0.0
lr decay rate	1.0	0.94

E Morpheme Statistics on IWSLT'14 De-En

We performed a statistical analysis of morphemes on the IWSLT'14 De-En dataset. The statistical results are shown in Table 15. The mor_∞ means the segmentation that does not limit the number of morphemes for a word. As can be seen from the row of Table 15 where mor_∞ is located, the number of words in German containing 1, 2, and 3 morphemes is 1728, 3623, and 2627, respectively. These words make up 91% of the total number of words (8848) in the corpus of German. In terms of English, the numbers of these three types of words are 1724, 2977, and 1632 respectively, accounting for a higher proportion (95%). Since most words contain no more than 3 morphemes, we tend to choose the segmentation that restricts the number of morphemes to a maximum of 3, and the order of MorphTE is set to 3.

Table 15: Statistics of morpheme segmentation results on IWSLT'14 De-En. The column of segmentation means different morphological segmentation schemes, mor_∞ means the segmentation that does not limit the number of morphemes for a word, and mor_i means the segmentation that restricts the number of morphemes to a maximum of i, referring to Eq. 6 in our paper. $N = n$ represents the number of words containing n morphemes. $|M|$ indicates the size of the morpheme vocabulary after the morpheme segmentation.

Language	segmentation	$\mathbf{N=1}$	$\mathbf{N=2}$	$\mathbf{N=3}$	$\mathbf{N=4}$	$\mathbf{N>4}$	$\mathbf{\lvert M\rvert}$
De (German)	mor_∞	1728	3623	2627	716	154	2531
	mor_4	1728	3623	2627	870	0	2643
	mor_3	1728	3623	3497	0	0	3013
	mor_2	1728	7120	0	0	0	4302
	mor_1	8848	0	0	0	0	8848
En (English)	mor_∞	1724	2977	1632	260	39	2584
	mor_4	1724	2977	1632	299	0	2614
	mor_3	1724	2977	1931	0	0	2744
	mor_2	1724	4908	0	0	0	3352
	mor_1	6632	0	0	0	0	6632

F Effect of Morpheme Segmenters on MorphTE

In previous experiments, our MorphTE utilized the morfessor [40] to segment words. It is a widely used morpheme segmentation tool [3, 31, 12]. In this section, we study how the morpheme segmenter affects the performance of MorphTE. We hypothesize that high-quality morpheme segmentation could improve the performance of MorphTE, while unreasonable segmentation could be counterproductive. To illustrate this, we add a comparative experiment on morpheme segmentation methods. Instead of using a morpheme segmenter, we design a random segmentation method called randomSeg. The randomseg does not segment words with a length of no more than three. For words with more than three characters, randomSeg randomly selects two gaps from the inside of the word to divide the word into 3 parts. Obviously, randomSeg makes little use of morpheme knowledge.

Table 16: Experimental results of different morpheme segmenters on translation tasks. Original dictates the original word embeddings without compression. MorphTE w/ morfessor dictates the MorphTE we use in the paper. MorphTE w/ randomSeg indicates the MorphTE using randomSeg for morpheme segmentation.

Method	De-En	En-It	En-Es	En-Ru
Original	34.5	32.9	39.1	31.6
MorphTE w/ morfessor	34.9	32.9	39.1	31.9
MorphTE w/ randomSeg	33.8	31.7	38.6	30.4

As shown in Table 16, the performance of MorphTE w/ randomSeg is inferior to the original word embedding method and MorphTE w/ morfessor on four translation datasets. This shows that morpheme segmentation can affect the effectiveness of MorphTE, and improving morpheme segmentation may further enhance MorphTE.

G Memory and Time Analysis

G.1 Memory Analysis

Table 17: Experimental results of the memory cost for different embedding methods. #Emb represents the memory/parameter cost of the word embedding layer, and #Stuc represents the memory/parameter cost of other parts of the model except the word embedding layer. P represents the proportion of the memory/parameter cost of the word embedding layer to the total cost of the model.

Method	Parameter			Memory		
	#Stuc/ M	#Emb / M	P / %	#Stuc / MiB	#Emb / MiB	P / %
Original	3.8	6.3	62	14.6	24.1	62
Matrix Factor.	3.8	0.08	2	14.6	0.3	2
Tensor Train	3.8	0.08	2	14.6	0.3	2
Word2ketXS	3.8	0.08	2	14.6	0.3	2
Word2ket	3.8	0.3	7	14.6	1.1	7
MorphTE	3.8	0.08	2	14.6	0.3	2

Through MorphTE or other compression methods, the parameters of the word embedding layer can be greatly reduced. In this section, We further experimentally test the memory cost of different word embedding methods on the WikiQA task. The experimental setup is consistent with the main experiment in our paper. As shown in Table 17, the memory reduction of different word embedding compression methods is consistent with their parameter reduction. MorphTE could reduce the proportion of the original word embedding parameters/memory to the total model parameters/memory from about 62% to 2%.

G.2 Time Analysis

Table 18: Experimental results of the time cost for different embedding methods. #Emb represents the time cost of the word embedding layer, and #Stuc represents the time cost of other parts of the model except the word embedding layer. P represents the proportion of the time cost of the word embedding layer to the total time cost of the model.

Method	GPU			CPU		
	#Stuc / ms	#Emb / ms	P / %	#Stuc / ms	#Emb / ms	P / %
Original	10.6	0.1	0.9	92.0	0.1	0.1
Matrix Factor.	10.6	0.2	1.9	92.0	1.0	1.1
Tensor Train	10.6	0.5	4.5	92.0	45.0	33
Word2ketXS	10.6	0.6	5.4	92.0	49.0	35
Word2ket	10.6	0.5	4.5	92.0	2.0	2.1
MorphTE	10.6	0.6	5.4	92.0	3.0	3.2

The original word embedding method can directly index the word vector from the word embedding matrix, and this process has almost no computational overhead. Almost all word embedding compression methods, including our method, require certain pre-operations to generate word embeddings, which introduces a certain computational overhead. To this end, we test the time cost of different word embedding methods on the WikiQA task. The experimental setup is also consistent with the main experiment in our paper. We test the time cost of different word embedding methods on Intel E5-2698 CPU and Tesla V100 GPU. Specifically, we repeat the test 100 times on an input sample, and average the time.

As shown in Table 18, although word embedding compression methods increase the time cost of word embeddings, most of them only account for a small fraction of the total time cost. The time cost of our MorphTE accounts for 5.4% and 3.2% of the model on GPU and CPU, respectively. Compared with the significant parameter/memory reduction, this slight computational overhead is tolerable. In addition, we also found that the time cost of word2ketxs and tensor train methods increased significantly on CPU devices. This may be related to the fact that these two methods have more computational operations under the condition of the same number of parameters.