Knowledge-Based Systems 194 (2020) 105553

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Kernel compositional embedding and its application in linguistic structured data classification[☆]

Hamed Ganji [a], Mohammad Mehdi Ebadzadeh [a,*], Shahram Khadivi [b,*]

[a] *Department of Computer Engineering and Information Technology, Amirkabir University of Technology, No. 424, Hafez Ave, Tehran, Iran*
[b] *eBay Inc., Kasernen Str. 25, Aachen, Germany*

A R T I C L E   I N F O

Article history:
Received 10 August 2019
Received in revised form 20 January 2020
Accepted 22 January 2020
Available online 23 January 2020

Keywords:
Structured data representation
Kernel methods
Compositional embedding
Structured object classification

A B S T R A C T

In many applications such as natural language processing, speech recognition, and computer vision, there are inputs with hierarchical compositional structure and long relation among their subcomponents. Introducing this information in the definition of a model can improve its performance in dealing with this type of data. On the other side, the high generalization power of kernel methods is proven in traditional machine learning problems. If we can employ some idea of these methods on handling structured objects, we can benefit from improving the performance and the generalization capability of compositional models. Accordingly, a new approach is introduced in this paper to realize the idea of simultaneously leveraging advantages of both kernel methods and compositional embedding to provide powerful representations and classifiers for structured data. Based on this approach, which is named Kernel Compositional Embedding (KCE), we propose two methods: Direct KCE (DKCE), and Indirect KCE (IKCE). In the DKCE method, we directly deal with a potentially infinite dimensional embedding space, i.e., the embeddings are used implicitly in the classification task. In IKCE method, instead of implicitly performing all operations inside embedding space, only some elements of one or more reproducing kernel Hilbert spaces are used to embed structured objects into low-dimensional Euclidean spaces. To evaluate the performance of the proposed methods, we apply them on two common computational linguistic tasks, i.e, sentiment analysis and natural language inference. The experimental results illustrate that the classification performance of the proposed methods is higher than or competitive to some well-known competing methods.

1. Introduction

Data representation is one of the fundamental tasks in machine learning. In a long history, feature engineering has been a common way for this purpose, and researchers aim at designing better features for specific tasks. Recently, the rapid development of representation learning and deep learning has brought new inspiration to various domains. Many of the proposed methods in this area are known by "embedding" term. It refers to providing a representing vector for a data object that preserves its properties from different aspects [1–5]. Particularly, in Natural Language Processing (NLP), word embedding methods map semantic and syntactic properties of words into a low-dimensional feature space. These methods have recently been attracted a tremendous amount of interest and have been shown to be beneficial to many tasks, including machine translation [6], syntactic parsing [7,8], text classification [9], sentiment analysis [10], and question answering [11,12].

A challenging problem in this domain is embedding of non-singleton data objects that are made up of multiple components (sub-objects) with a specific communication among them. In some situations, each of these constituent components also consists of another set of smaller components itself. Due to this recursive constitutional relation, this kind of data is called recursive structured data. Examples of this kind of data can be seen in different modalities. Especially, phrases and sentences with some syntactic or semantic structures are common samples of this type of data in linguistic tasks. As an example, a sentence with its parse tree is shown in Fig. 1. Recently, some compositional models have been proposed that provide embeddings for the recursive structured objects while capturing compositional syntactic and semantic information of their constituents [13–15]. These methods are used for embedding linguistic data in various NLP tasks, particularly by involving the syntactic structure of sentences. For example, the vectors of colored circles in Fig. 1 indicate the

[☆] No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.knosys.2020.105553.

[*] Corresponding authors.
E-mail addresses: hamed_ganji@aut.ac.ir (H. Ganji), ebadzadeh@aut.ac.ir (M.M. Ebadzadeh), skhadivi@ebay.com (S. Khadivi).

https://doi.org/10.1016/j.knosys.2020.105553
0950-7051/© 2020 Elsevier B.V. All rights reserved.

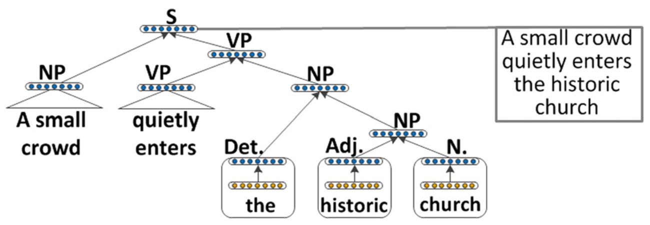

Fig. 1. An example of a natural language sentence with a corresponding parse tree and embeddings of its constituents. All the nodes are labeled using syntactic categories of the grammar. The vectors of orange circles show the prior representation vectors for the words. The vectors of blue circles indicate the embeddings of structured objects, i.e., phrases. These embeddings can be achieved using a compositional embedding method. In this fashion, the embedding of each node is obtained by composing the embeddings of its child nodes. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

embeddings of the sentence's constituents, i.e., the words and phrases. These embeddings can be obtained by a compositional model.

On the other side, kernel methods are well-known in machine learning communities and have been successfully applied in various real applications. Particularly, kernel SVM whose leaning optimization problem focuses on minimizing the expected risk is one of the most prominent of them due to its excellent generalization performance [16]. Also in data representation area, many popular classic techniques have been extended to their kernel form, such as kernel principal component analysis [17] and kernel Fisher discriminant analysis [18]. These techniques have been shown to be superior to their linear counterpart in various tasks. In recent years, some successful efforts have also been made to use kernels to provide embeddings of data distributions in a potentially infinite-dimensional feature space [19–21].

This idea to simultaneously leverage the benefits of both compositional embedding and kernel methods for structured data representation and classification is a primary motivation to make a new class of techniques in this area. To this end, two methods are proposed in this paper based on a new approach called Kernel Compositional Embedding (KCE). Generally, a KCE method maps a recursive structured data to an embedding space by recursively mapping and composing its constituents with the incorporation of some implicit operators. These operators are constructed using elements of one or more reproducing kernel Hilbert spaces. In fact, the embedding process of KCE consists of several hierarchical transposition and composition steps. An example conceptual view of a step of this process are shown in Fig. 2. In this example, embedding of a phrase with two words is desired. To this end, one or multiple prior representations of each word are mapped to a single vector element of an intermediate kernel space first. Then the mapped vectors are transposed to a subspace, which syntactically or semantically corresponds to the phrase. Finally, the embedding of the phrase is calculated through a weighted summation of the transposed vectors.

This should be noted that an embedding space in KCE can be potentially infinite-dimensional space depending on the definition of transpose operators. This definition is also the most important difference between the two proposed KCE methods. In the first method, Direct KCE (DKCE), the transpose operators are entirely defined as implicit operators. Hence, the embedding space of this method, which is the same for all structured objects, can potentially has an infinite number of dimensions. Accordingly, to fulfill classification task in DKCE, all operations should be implicitly performed on embeddings. Indirect KCE (IKCE) is the second method in which we deal with explicit low-dimensional embedding spaces. In fact, instead of implicitly performing all operations inside embedding space, only some elements of one or more reproducing kernel Hilbert spaces are used to create the transpose operators in this method.

The contributions of this paper can be summarized as follows:

1. A new approach, KCE, is proposed that fulfills compositional embedding by implicitly incorporating some elements of one or more kernel spaces. In this manner, KCE actually leverages the benefits of kernel methods and compositional embedding simultaneously, which has not been addressed by other compositional methods.
2. A unified formulation and general definition of inference and learning procedures are provided for KCE. Thus, KCE can be seen as a flexible framework whose composition mechanism can be customized by declaring custom variants of its operators.
3. Based on the KCE approach, two new methods are introduced to implicitly or explicitly embed structured data objects and to classify them. To evaluate the performance of these methods and also to illustrate the application of the approach in computational linguistic tasks, we apply our methods in two common problems, i.e., sentiment analysis [22] and natural language inference [23]. The experimental results demonstrate that the classification performance of the KCE methods is better than or competitive to some well-known related techniques.
4. Also, a multiple kernel learning solution is applied to achieve a richer kernel and better performance. In this way, KCE has the capacity of employing various knowledge resources in the form of the kernel. Thus, it can exploit the capacities of other embedding methods.

The remaining sections of the paper are organized as follows. We briefly discuss some previous related works in Section 2. Section 3 is devoted to a detailed description of the KCE approach and the two methods derived from it. Next, we present and discuss in detail the experimental results and some comparisons in Section 4. Finally, Section 5 concludes the paper.

2. Related works

In this section, we review some prominent related works on embedding methods. They provide embeddings for single or structured objects, or even distributions. To fulfill this purpose, they rely on different solutions including concurrency statistics, deep neural networks, and kernel methods.

2.1. Word embeddings

Word embeddings, also called distributed word representations [24], are unsupervised learning methods to represent words by dense, low-dimensional and real-valued vectors. Each dimension of an embedding represents a latent feature of a word, hopefully capturing useful syntactic and semantic properties of a language. Indeed, these methods analyze text data to learn distributed representations of the vocabulary that capture their co-occurrence statistics and are helpful for reasoning about word usage and meaning [25,26]. This is in contrast to traditional approaches that represent text objects as bag-of-words (BoW), term frequency inverse document frequency (TF–IDF) [27] vectors, or their enhanced variants [28,29]. However, such representations cannot accurately capture the similarity between individual words and do not take the semantic structure of a language into consideration.

Although some neural network based approaches, which process massive amounts of textual data to embed word semantics into low-dimensional vectors, have been introduced in the

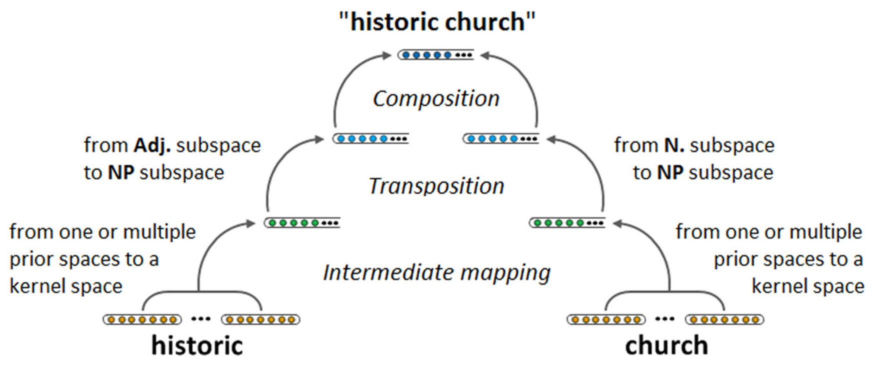
"historic church"
Composition
from Adj. subspace to NP subspace
from N. subspace to NP subspace
Transposition
from one or multiple prior spaces to a kernel space
Intermediate mapping
from one or multiple prior spaces to a kernel space
historic
church

Fig. 2. An example conceptual view of one step of the embedding process in the KCE approach. It consists of three phases. First, one or multiple prior representations of a constituent are mapped to a single vector element of an intermediate kernel space. Then, the mapped vectors are transposed from their syntactically or semantically related subspaces to another subspace. The destination subspace is also corresponds to the target component and can be located in a different space. Finally, the transposed vectors are linearly composed to make the embedding of the phrase. The ellipses at the end of the vectors indicate that they can have infinite length potentially.

last decade [24,30]. Word embeddings have gained universal recognition thanks to the recent considerable interesting work, Word2Vec [31]. It presents two log-linear models, continuous bag-of-words (CBoW) and continuous skip-gram (SG) [32,33], that allow for efficient estimation of continuous-space word representations from massive datasets. A well-known extension of Word2Vec is the FastText method which represents each word as an n-gram of characters [34]. Another prominent word embedding architecture is GloVe (Global Vectors for word representation) [35] which combines global matrix factorization and local context window methods through a bilinear regression model.

Recently, some efforts have also been done to provide contextualized word embeddings by capturing word semantics in different contexts to address the context-dependent nature of words. ELMo (Embeddings from Language Models) [36] is one of these methods that uses a bi-directional recurrent neural network trained on a specific task to create the embeddings. BERT (Bidirectional Encoder Representations from Transformers) [37] is another successful contextualized word embedding model which is based on a multilayer bi-directional transformer-encoder. This transformer neural network uses parallel attention layers rather than sequential recurrence. Besides these popular methods, various approaches have also been proposed to improve the quality of word embeddings, such as exploiting dependency parse-trees [38] or symmetric patterns [39], leveraging sub-word units [40], representing words as probability distributions [41], learning word embeddings in multilingual vector spaces [8,42], or exploiting knowledge resources like WordNet or BabelNet [43–46].

2.2. Compositionality in embedding spaces

Although the word embeddings are sufficient to compute the similarity between words, it is not trivial to capture the meaning of phrases and sentences. To address this issue, modeling and learning compositionality in phrases have received a lot of attention recently [47,48]. Most of the compositionality algorithms and related datasets capture two-word compositions. Particularly, [13] uses two-word phrases and analyze similarities computed by vector addition, multiplication, convolution [49] and others. They measured the similarity between word pairs such as compound nouns or verb–object pairs and compared these with human similarity judgments. Among the ways, simple vector averaging or multiplication performed best. There are some other related models such as holographic reduced representations [50], quantum logic [51], discrete–continuous models [52] and compositional matrix space model [53]; however, these have not been experimentally validated on larger corpora.

Recursive neural network (RNN) [54] is one of the most prominent compositional methods that comprises an architecture in which the same set of weights is recursively applied within a structural setting. Matrix–vector RNN (MV-RNN) [55] is an extension of RNN in which each constituent (a word or longer phrase) has a matrix–vector representation. The vector captures the meaning of constituents, and the matrix captures how each constituent modifies the meaning of other one that it combines with. However, it has a large number of parameters and there are no methods yet for pre-training the word matrices. Another popular generalization on RNN is RNTN (Recursive Neural Tensor Network) [14]. The main idea of RNTN is to use a single, tensor-based composition function to perform better and compose aggregate meaning from smaller constituents.

In another side, recurrent neural networks are a natural choice for sequence modeling tasks, due to their capability for processing arbitrary length sequences. Recently, these networks with LSTM (Long Short-Term Memory) units [56] have re-emerged as a popular architecture, because of their representational power and effectiveness at capturing long-term dependencies [57]. However, LSTM architectures are insufficient to fully capture the semantics of natural languages, due to their inability to account for differences in meaning as a result of differences in word order or syntactic structure. To fill this gap, [15] introduces a generalization of the standard LSTM architecture to tree-structured network topologies called Tree-LSTM. It can incorporate information from multiple child units. Furthermore, to include semantic information in combination of constituent nodes, [58] also proposes multiplicative Tree-LSTM. It injects semantic relation information into every node in the tree that allows the model to have different semantic composition matrices to combine child nodes.

2.3. Kernel embedding

Exploiting complex statistical features are commonplace for the analysis of large volumes of high-dimensional measurements which is required in most modern applications of machine learning. Most existing approaches, including graphical models, rely heavily on parametric assumptions that can result in a model far different from the data generating process. [19] provides a novel, unified kernel framework called kernel embedding of distributions for addressing this challenge. Its key idea is to map conditional distributions into potentially infinite-dimensional feature spaces using kernels, such that subsequent comparisons and manipulations of these distributions are achieved via feature space operations: inner products, distances, projections, and linear transformations. This can be thought of as a generalization of the feature mapping of individual points, as used in classical kernel methods. This very general framework is core to advanced kernel probabilistic methods, including kernel two-sample testing [59], kernel Bayesian inference [20], density estimation [60], component analysis [61], feature discovery [21], and state space filtering [62]. This framework inspires some basic idea of our proposed methods.

3. Kernel compositional embedding

Simultaneously exploiting the advantages of both kernel methods and compositional embeddings to represent and classify data with recursive structure is the primary motivation to make a new class of techniques in this area. In order to reach this purpose, we propose a new approach called Kernel Compositional Embedding (KCE). In KCE, embedding of a structured object is obtained by recursively mapping and composing its constituents with the incorporation of some implicit transpose operators. These operators, which play a fundamental role in KCE, are constructed using elements of one or more reproducing kernel Hilbert spaces. Based on the definition of transpose operators and intermediate mappings, we introduce two methods to accomplish the purpose of KCE. The details of these methods and some essential aspects of KCE are explained in the following subsections.

3.1. Composition

In KCE, the embedding of a composed object is actually obtained by a linear composition of transposed embeddings of its constituents. Since these constituents can also be formed from some sub-objects themselves, this composition will be recursively repeated to obtain embedding of the whole structured object. Eq. (1) illustrates the composition operation in KCE that is done by weighted summation of constituent embeddings, which are already transposed to target subspace,

$$\boldsymbol{\phi}^{(l_p)}(\boldsymbol{x}_p) = \sum_{c\in \boldsymbol{I}_p} \frac{\alpha(s_c, s_p)}{\tilde{\alpha}(s_p; \boldsymbol{I}_p)}\, \boldsymbol{\psi}^{(l_c,l_p)}(\boldsymbol{x}_c, s_c, s_p)\ , \tag{1}$$

where $\boldsymbol{\phi}^{(l_p)}(\boldsymbol{x}_p)$ indicates the embedding vector of pth component of input sample $\boldsymbol{x}$ as parent, which achieved at the embedding level l_p, and $\boldsymbol{I}_p$ is the index set of its children. Moreover, $\boldsymbol{\psi}^{(l_c,l_p)}(\boldsymbol{x}_c, s_c, s_p)$ shows the transposed embedding of cth component of the input sample as one of the children. Also, s_c and s_p are tags referring to the embedding subspaces belong to a child and parent as the source and the destination of a transposition. Furthermore, $\alpha(s_c, s_p)$ and $\tilde{\alpha}(s_p; \boldsymbol{I}_p)$ are importance coefficient and coefficient normalizer, respectively, which must be estimated according to the source and the destination of transposition. As with l_p, l_c also indicates the embedding level of a child. The embedding levels are involved because the target embedding spaces at different composition steps can be distinct.

Actually, in this composition, the child embeddings are mapped from their corresponding embedding subspaces to the parent's subspace first. Then, the parent's embedding is made by weighted summation of the transposed embeddings of its children. The normalized importance coefficients, $\frac{\alpha(s_c,s_p)}{\tilde{\alpha}(s_p;\boldsymbol{I}_p)}$, are the weights to specify the participation ratio of the transposed embeddings in this construction. The transposed embeddings are obtained by the following equation,

$$\boldsymbol{\psi}^{(l_c,l_p)}(\boldsymbol{x}_c, s_c, s_p) = \boldsymbol{T}^{(l_c,l_p)}(s_c, s_p)\, \boldsymbol{\phi}'^{(l_c,l_p)}(\boldsymbol{x}_c)\ , \tag{2}$$

where $\boldsymbol{T}^{(l_c,l_p)}(s_c, s_p)$ is a transpose operator and $\boldsymbol{\phi}'^{(l_c,l_p)}(\boldsymbol{x}_c)$ is an intermediate map of embedding of child c, whose embedding can be obtained from the previous composition steps itself.

According to the elements involved in the above equations, we can describe a structured data input $\boldsymbol{x}$ by a tuple of $(\boldsymbol{v}, \boldsymbol{s}, \boldsymbol{I})$; where $\boldsymbol{v}$ is comprised of raw input values associated with nodes, $\boldsymbol{s}$ includes embedding subspace tags of all nodes, and $\boldsymbol{I}$ consists of index sets of their children. To explain this definition, we apply it on a part of the example sentence shown in Fig. 1. If we consider the phrase "the historic church" as a single structured data input, it is described as follows:

$$\begin{aligned}\boldsymbol{v} &= \Big[\ \text{“the”, “historic”, “church”, “historic church”,}\\ &\quad \text{“the historic church”}\ \Big]\\ \boldsymbol{s} &= \Big[\ \text{Det., Adj., N., NP, NP}\ \Big]\\ \boldsymbol{I} &= \Big[\ \{\},\ \{\},\ \{\},\ \{2, 3\},\ \{1, 4\}\ \Big]\end{aligned}$$

Using this definition, the embedding of the phrase "historic church" is calculated as:

$$\begin{aligned}\boldsymbol{\phi}^{(1)}(\text{“historic church”}) = &\frac{\alpha(\text{Adj., NP})}{\tilde{\alpha}(\text{NP}; \{2,3\})}\, \boldsymbol{\psi}^{(0,1)}(\text{“historic”, Adj., NP})\\ &+ \frac{\alpha(\text{N., NP})}{\tilde{\alpha}(\text{NP}; \{2,3\})}\, \boldsymbol{\psi}^{(0,1)}(\text{“church”, N., NP})\ ,\end{aligned}$$

where

$$\begin{aligned}\boldsymbol{\psi}^{(0,1)}(\text{“historic”, Adj., NP}) &= \boldsymbol{T}^{(0,1)}(\text{Adj., NP})\, \boldsymbol{\phi}'^{(0,1)}(\text{“historic”})\\ \boldsymbol{\psi}^{(0,1)}(\text{“church”, N., NP}) &= \boldsymbol{T}^{(0,1)}(\text{N., NP})\, \boldsymbol{\phi}'^{(0,1)}(\text{“church”})\ .\end{aligned}$$

Similarly, for the phrase "the historic church" we have,

$$\begin{aligned}&\boldsymbol{\phi}^{(2)}(\text{“the historic church”}) =\\ &\quad \frac{\alpha(\text{Det., NP})}{\tilde{\alpha}(\text{NP}; \{1,4\})}\, \boldsymbol{\psi}^{(0,2)}(\text{“the”, Det., NP})\\ &\quad + \frac{\alpha(\text{NP, NP})}{\tilde{\alpha}(\text{NP}; \{1,4\})}\, \boldsymbol{\psi}^{(1,2)}(\text{“historic church”, NP, NP})\ .\end{aligned}$$

In this example, $\boldsymbol{x}_j$ is replaced with $\boldsymbol{v}_j$ for the sake of conciseness. The embedding levels are usually bounded in the KCE methods, $l_j \le l_{\max}$. It means the embedding level for an input sample is increased during bottom-up composition steps until reaching $l_{\max}$. Then all subsequent operations are done in the same embedding space at the level $l_{\max}$.

3.2. Inference

In order to keep generality of the KCE approach in terms of inference and learning problems, we generally define the decision rule of a KCE method as follows:

$$y^* = h(\boldsymbol{x}; \boldsymbol{\theta}) = \operatorname*{argmax}_{y\in\mathcal{Y}}\, g_{\text{KCE}}(\boldsymbol{x}, y; \boldsymbol{\theta})\ , \tag{3}$$

where $g_{\text{KCE}}(\boldsymbol{x}, y; \boldsymbol{\theta})$ is a score function with parameter set $\boldsymbol{\theta}$ that assigns a score to each pair $(\boldsymbol{x}, y)$, a structured input with a candidate label. Also, $\mathcal{Y}$ is the set of all valid labels. In fact, the score function is made up of some functionals of embedding spaces at various levels. Each of these functionals, which is called score functional, is theoretically defined as follows:

$$\mathbf{g}^{(l)}_{\text{KCE}}(\boldsymbol{.}, y; \boldsymbol{\theta}) = \int_{\mathcal{X}^{(l)}} \boldsymbol{\phi}^{(l)}(\boldsymbol{x})\, d\beta^{(l)}(\boldsymbol{x}, y)\ , \tag{4}$$

where $\mathcal{X}^{(l)}$ is the domain of all objects that can be embedded into the embedding space at level l, and $\beta^{(l)}(\boldsymbol{x}, y)$ is a valid measure in that space. In practice, we need an empirical estimate of this score functional to utilize in real applications. This empirical estimate can be computed using training data as follows:

$$\hat{\mathbf{g}}^{(l)}_{\text{KCE}}(\boldsymbol{.}, y; \boldsymbol{\theta}) = \sum_{\tilde{\boldsymbol{x}}\in\tilde{\mathcal{X}}^{(l)}} \beta(\tilde{\boldsymbol{x}}, y)\, \boldsymbol{\phi}^{(l)}(\tilde{\boldsymbol{x}}) \tag{5}$$

In this equation, $\tilde{\mathcal{X}}^{(l)}$ indicates the set of all the sub-samples in the training set, whose embeddings are obtained at embedding level l. Also, $\beta(\tilde{\boldsymbol{x}}, y)$ specifies the participation ratio of a sub-sample in practice. Since each training sub-sample can only embed into one embedding space at a specific level, the parameter l of the measure $\beta^{(l)}(\tilde{\boldsymbol{x}}, y)$ is discarded in this empirical formulation.

In this way, calculating the score function of an input sample is happening via the inner product of an appropriate score functional and the embedding of the sample;

$$\begin{aligned} g_{\text{KCE}}(\boldsymbol{x}, y; \boldsymbol{\theta}) &= \hat{\boldsymbol{g}}_{\text{KCE}}^{(l(\boldsymbol{x}))}(\centerdot, y; \boldsymbol{\theta})^T \boldsymbol{\phi}^{(l(\boldsymbol{x}))}(\boldsymbol{x}) \\ &= \Big(\sum_{\substack{\tilde{\boldsymbol{x}} \in \\ \tilde{\mathcal{X}}^{(l(\boldsymbol{x}))}}} \beta(\tilde{\boldsymbol{x}}, y) \boldsymbol{\phi}^{(l(\boldsymbol{x}))}(\tilde{\boldsymbol{x}}) \Big)^T \boldsymbol{\phi}^{(l(\boldsymbol{x}))}(\boldsymbol{x}) \\ &= \sum_{\substack{\tilde{\boldsymbol{x}} \in \\ \tilde{\mathcal{X}}^{(l(\boldsymbol{x}))}}} \beta(\tilde{\boldsymbol{x}}, y)\ \boldsymbol{\phi}^{(l(\boldsymbol{x}))}(\tilde{\boldsymbol{x}})^T \boldsymbol{\phi}^{(l(\boldsymbol{x}))}(\boldsymbol{x}) , \end{aligned} \tag{6}$$

where $l(\boldsymbol{x})$ indicates that the appropriate score functional is selected according to the embedding level at which the input sample is embedded. In some tasks such as natural language inference (NLI), input of the score function is a pair of structured objects with a single label. To cover the type of problems similar to NLI, the definition of Eq. (4) needs to be updated and generalized as follows:

$$\boldsymbol{g}_{\text{KCE}}^{(l',l'')}(\centerdot, \centerdot, y; \boldsymbol{\theta}) = \int_{\mathcal{X}^{(l')} \times \mathcal{X}^{(l'')}} \boldsymbol{\phi}^{(l')}(\boldsymbol{x}') \otimes \boldsymbol{\phi}^{(l'')}(\boldsymbol{x}'')\, d\beta^{(l',l'')}(\boldsymbol{x}', \boldsymbol{x}'', y) \tag{7}$$

In fact, $\boldsymbol{g}_{\text{KCE}}^{(l',l'')}$ is an element of a second-order tensor embedding space which can be used to evaluate the score function as a linear algebraic operation. Similarly, an empirical estimate of this generalized score functional can be calculated by the following equation.

$$\hat{\boldsymbol{g}}_{\text{KCE}}^{(l',l'')}(\centerdot, \centerdot, y; \boldsymbol{\theta}) = \sum_{\substack{(\tilde{\boldsymbol{x}}', \tilde{\boldsymbol{x}}'') \\ \in \tilde{\mathcal{X}}^{(l',l'')}}} \beta(\tilde{\boldsymbol{x}}', \tilde{\boldsymbol{x}}'', y)\ \boldsymbol{\phi}^{(l')}(\boldsymbol{x}') \otimes \boldsymbol{\phi}^{(l'')}(\boldsymbol{x}'') \tag{8}$$

Using this functional, the score function is evaluated as shown below.

$$\begin{aligned} g_{\text{KCE}}(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, y; \boldsymbol{\theta}) &= \boldsymbol{\phi}^{(l(\boldsymbol{x}^{(1)}))}(\boldsymbol{x}^{(1)})^T \\ &\quad \times \hat{\boldsymbol{g}}_{\text{KCE}}^{(l(\boldsymbol{x}^{(1)}),l(\boldsymbol{x}^{(2)}))}(\centerdot, \centerdot, y; \boldsymbol{\theta})\ \boldsymbol{\phi}^{(l(\boldsymbol{x}^{(2)}))}(\boldsymbol{x}^{(2)}) \\ &= \boldsymbol{\phi}^{(l(\boldsymbol{x}^{(1)}))}(\boldsymbol{x}^{(1)})^T \Big(\sum_{\substack{(\tilde{\boldsymbol{x}}', \tilde{\boldsymbol{x}}'') \\ \in \tilde{\mathcal{X}}^{(l',l'')}}} \beta(\tilde{\boldsymbol{x}}', \tilde{\boldsymbol{x}}'', y)\ \boldsymbol{\phi}^{(l')}(\boldsymbol{x}') \\ &\quad \otimes \boldsymbol{\phi}^{(l'')}(\boldsymbol{x}'') \Big)\ \boldsymbol{\phi}^{(l(\boldsymbol{x}^{(2)}))}(\boldsymbol{x}^{(2)}) \\ &= \sum_{\substack{(\tilde{\boldsymbol{x}}', \tilde{\boldsymbol{x}}'') \\ \in \tilde{\mathcal{X}}^{(l',l'')}}} \beta(\tilde{\boldsymbol{x}}', \tilde{\boldsymbol{x}}'', y)\ \big(\boldsymbol{\phi}^{(l(\boldsymbol{x}^{(1)}))}(\boldsymbol{x}^{(1)})^T \boldsymbol{\phi}^{(l')}(\boldsymbol{x}')\big) \\ &\quad \times \big(\boldsymbol{\phi}^{(l'')}(\boldsymbol{x}'')^T \boldsymbol{\phi}^{(l(\boldsymbol{x}^{(2)}))}(\boldsymbol{x}^{(2)})\big) \end{aligned} \tag{9}$$

These general definitions of the score functions help us to design different ways of implementing the KCE idea while preserving generality of the approach.

3.3. Direct kernel compositional embedding

A method to accomplish the KCE approach is to embed all the structured inputs into a unique embedding space whose number of dimensions can be infinite. In this way, each embedding implicitly incorporates into learning and inference processes using the kernel trick. In other words, all operations for composition and transposition in this method demonstrated in Eqs. (1) and (2) are implicitly performed inside the same embedding space and no intermediate mapping is required. Due to discarding intermediate mappings, this method is called Direct Kernel Compositional Embedding (DKCE). Also, because the target embedding space at all composition steps is the same in DKCE, the level parameterizations can be discarded for this method. Accordingly, the transpose operators of DKCE is generally defined as follows:

$$\begin{aligned} \boldsymbol{T}(s_c, s_p) &= \int_{\mathcal{X} \times \mathcal{X}} \boldsymbol{\phi}(\boldsymbol{x}') \otimes \boldsymbol{\phi}(\boldsymbol{x}'')\, d\nu(\boldsymbol{x}', \boldsymbol{x}'', s_c, s_p) \\ &= \begin{bmatrix} t_{1,1}^{(s_c,s_p)} & t_{1,2}^{(s_c,s_p)} & \cdots \\ t_{2,1}^{(s_c,s_p)} & t_{2,2}^{(s_c,s_p)} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}_{m \times m} \end{aligned} \tag{10}$$

where m is the number of dimensions of the embedding space and $\nu(\boldsymbol{x}', \boldsymbol{x}'', s_c, s_p)$ is an arbitrary measure which is valid in that space. In fact, each transpose operator in DKCE is an element of second-order tensor embedding space whose number of dimensions is infinite potentially, $m = \infty$. For example, if the measure is defined as a conditional probability function as follows:

$$\nu(\boldsymbol{x}', \boldsymbol{x}'', s_c, s_p) = P(\boldsymbol{x}', \boldsymbol{x}'' | s_c, s_p) , \tag{11}$$

then the general definition of the transpose operators becomes a tensor expectation.

$$\boldsymbol{T}(s_c, s_p) = \mathbb{E}_{XX|Z}\big[\phi(X) \otimes \phi(X) \mid s_c, s_p\big] \tag{12}$$

To utilize the transpose operators in real tasks, its empirical estimate can be calculated using sub-samples in the training set as follows:

$$\hat{\boldsymbol{T}}(s_c, s_p) = \sum_{\substack{\boldsymbol{x}', \boldsymbol{x}'' \\ \in \tilde{\mathcal{X}}}} \nu(\boldsymbol{x}', \boldsymbol{x}'', s_c, s_p)\, \boldsymbol{\phi}(\boldsymbol{x}') \otimes \boldsymbol{\phi}(\boldsymbol{x}'') \tag{13}$$

By applying this empirical operator, the transposed embeddings are obtained as shown below.

$$\begin{aligned} \boldsymbol{\psi}(\boldsymbol{x}_c, s_c, s_p) &= \hat{\boldsymbol{T}}(s_c, s_p)\, \boldsymbol{\phi}(\boldsymbol{x}_c) \\ &= \Big(\sum_{\boldsymbol{x}', \boldsymbol{x}'' \in \tilde{\mathcal{X}}} \nu(\boldsymbol{x}', \boldsymbol{x}'', s_c, s_p)\, \boldsymbol{\phi}(\boldsymbol{x}') \otimes \boldsymbol{\phi}(\boldsymbol{x}'') \Big)\, \boldsymbol{\phi}(\boldsymbol{x}_c) \\ &= \sum_{\boldsymbol{x}', \boldsymbol{x}'' \in \tilde{\mathcal{X}}} \nu(\boldsymbol{x}', \boldsymbol{x}'', s_c, s_p) \Big(\boldsymbol{\phi}(\boldsymbol{x}') \otimes \boldsymbol{\phi}(\boldsymbol{x}'') \Big)\, \boldsymbol{\phi}(\boldsymbol{x}_c) \\ &= \sum_{\boldsymbol{x}', \boldsymbol{x}'' \in \tilde{\mathcal{X}}} \nu(\boldsymbol{x}', \boldsymbol{x}'', s_c, s_p)\ \ \boldsymbol{\phi}(\boldsymbol{x}'')^T \boldsymbol{\phi}(\boldsymbol{x}_c)\ \ \boldsymbol{\phi}(\boldsymbol{x}') \end{aligned} \tag{14}$$

Now, each inner product whose both the associated operands, e.g., $\boldsymbol{\phi}(\boldsymbol{x}'')$ and $\boldsymbol{\phi}(\boldsymbol{x}_c)$, are the embeddings of atomic objects, i.e., words, should be substituted with an equivalent evaluation of a kernel function, e.g., $k(\boldsymbol{x}'', \boldsymbol{x}_c)$. If one of the incorporated operands is an embedding of a composed object itself, it needs to be computed in another composition step. This means that it should be replaced with its corresponding composition equation using Eq. (1). Then, the inner product is distributed on embeddings of child objects.

As mentioned, $\boldsymbol{g}_{\text{KCE}}(\centerdot, y; \boldsymbol{\theta})$ is an element of the embedding space itself. Thus, this functional is a potentially infinite-length vector for DKCE and must be applied implicitly too. To obtain the score function of DKCE, it is sufficient to replace the embedding of input sample $\boldsymbol{\phi}(\boldsymbol{x})$ in Eq. (6) with its corresponding DKCE composition which is achieved using Eqs. (1) and (14). The result of this replacement is shown below.

$$g_{\text{DKCE}}(\boldsymbol{x}, y; \boldsymbol{\theta}) = \sum_{\tilde{\boldsymbol{x}} \in \tilde{\mathcal{X}}} \beta(\tilde{\boldsymbol{x}}, y)\ \ \boldsymbol{\phi}(\tilde{\boldsymbol{x}})^T \Big(\sum_{c \in \boldsymbol{I}_p} \frac{\alpha(s_c, s_p)}{\tilde{\alpha}(s_p; \boldsymbol{I}_p)}$$

$$\times \sum_{\substack{\boldsymbol{x}',\boldsymbol{x}'' \\ \in\tilde{\mathcal{X}}}} \nu(\boldsymbol{x}',\boldsymbol{x}'',s_c,s_p)\ \boldsymbol{\phi}(\boldsymbol{x}')^T\boldsymbol{\phi}(\boldsymbol{x}_c)\ \boldsymbol{\phi}(\boldsymbol{x}'')\Big)$$

$$= \sum_{\tilde{\boldsymbol{x}}\in\tilde{\mathcal{X}}} \beta(\tilde{\boldsymbol{x}},y)\ \Big(\sum_{c\in\boldsymbol{I}_p} \frac{\alpha(s_c,s_p)}{\tilde{\alpha}(s_p;\boldsymbol{I}_p)} \sum_{\substack{\boldsymbol{x}',\boldsymbol{x}'' \\ \in\tilde{\mathcal{X}}}} \nu(\boldsymbol{x}',\boldsymbol{x}'',s_c,s_p)$$

$$\times\ \boldsymbol{\phi}(\boldsymbol{x}')^T\boldsymbol{\phi}(\boldsymbol{x}_c)\ \boldsymbol{\phi}(\tilde{\boldsymbol{x}})^T\boldsymbol{\phi}(\boldsymbol{x}'')\Big) \tag{15}$$

This equation can be expanded more by repeating the replacement of $\boldsymbol{\phi}(\boldsymbol{x}_c)$ until no dividable constituent remains. At that point, by substituting all inner products with their corresponding kernel evaluations, e.g., $k(\boldsymbol{x}',\boldsymbol{x}_c)$ for $\boldsymbol{\phi}(\boldsymbol{x}')^T\boldsymbol{\phi}(\boldsymbol{x}_c)$, no implicit element will remain and output value of the score function can be calculated explicitly. These replacements and expanding can be similarly done for Eq. (9).

Although evaluation of this score function can be efficiently computed by a bottom-up fashion, the computational complexity of inference and learning procedures is exponentially increased according to the size of training data. This is a severe challenge in dealing with large datasets that can make the method inapplicable, similar to many other non-parametric approaches. As a way to reduce this complexity, $\tilde{\mathcal{X}}$ can be confined to only include the atomic objects in empirical estimates of the score functional and the transpose operators. Another simplifying way is to consider the transpose operators as an identity matrix for dividable constituents. This strong assumption actually results in $\boldsymbol{\psi}^{(l_c,l_p)}(\boldsymbol{x}_c,s_c,s_p)=\boldsymbol{\phi}(\boldsymbol{x}_c)$ where $\boldsymbol{x}_c$ is a non-atomic component. By applying this assumption, the definition of the score function can be rewritten as Eq. (16), if no child component of $\boldsymbol{x}$ is atomic. We refer to this variant of DKCE as sDKCE, which stands for simplified DKCE. Please note that these strong simplifying assumptions for sDKCE may yield a significant performance degradation.

$$g_{\text{sDKCE}}(\boldsymbol{x},y;\boldsymbol{\theta}) = \sum_{\tilde{\boldsymbol{x}}\in\tilde{\mathcal{X}}} \beta(\tilde{\boldsymbol{x}},y)\ \ \boldsymbol{\phi}(\tilde{\boldsymbol{x}})^T\Big(\sum_{c\in\boldsymbol{I}_p}\frac{\alpha(s_c,s_p)}{\tilde{\alpha}(s_p;\boldsymbol{I}_p)}\boldsymbol{\phi}(\boldsymbol{x}_c)\Big)$$

$$= \sum_{\tilde{\boldsymbol{x}}\in\tilde{\mathcal{X}}} \beta(\tilde{\boldsymbol{x}},y)\sum_{c\in\boldsymbol{I}_p}\frac{\alpha(s_c,s_p)}{\tilde{\alpha}(s_p;\boldsymbol{I}_p)}\ \boldsymbol{\phi}(\tilde{\boldsymbol{x}})^T\boldsymbol{\phi}(\boldsymbol{x}_c) \tag{16}$$

Alg. 1 presents a pseudo-code for sDKCE. It illustrates how the simplified idea of DKCE and its score function can be implemented in a recursive fashion. There are two functions in this algorithm. The first one is declared as SDKCE_SCORE($\boldsymbol{v},\boldsymbol{s},\boldsymbol{I},y$) that implements Eq. (6) by a loop on support samples included in $\boldsymbol{Z}$. The inputs of this function are previously described in Section 3.1. The support samples are actually the atomic objects in the training set that their corresponding entries of the matrix ***beta*** are non-zero after training. The second function is EMBEDDING_INNERPROD($\boldsymbol{v},\boldsymbol{s},\boldsymbol{I},i,\boldsymbol{w}$) that recursively calculates the inner product of implicit embeddings using kernel. In this function, $\boldsymbol{X}$ is $\tilde{\mathcal{X}}$ which indicates the set of atomic objects in the training set. We also use SUM as an auxiliary function to calculate the sum of elements of a vector. According to the algorithm, if we consider the evaluation of kernel function as a single instruction, the time complexity of calculating the sDKCE score function is $O\big(|\boldsymbol{Z}|\times(|\boldsymbol{v}|\times|\tilde{\mathcal{X}}|^2+|\boldsymbol{I}|-|\boldsymbol{v}|)\big)$. Please note that $|\boldsymbol{Z}|$ is equal to $|\tilde{\mathcal{X}}|$ at worst. Also, we always have $|\boldsymbol{I}|<2*|\boldsymbol{v}|$ and $|\boldsymbol{v}|$ is usually less than 100. Therefore, the time complexity of this algorithm can be summarized as $O(|\tilde{\mathcal{X}}|^3)$.

3.4. Indirect kernel compositional embedding

Another way to accomplish the KCE approach, instead of direct embedding into a potentially infinite-dimensional space and implicitly performing all operations, is to utilize some elements of one or more kernel spaces to embed structured objects into some low-dimensional Euclidean spaces. Due to exploiting some kernel spaces to perform intermediate mappings, this method is called Indirect Kernel Compositional Embedding (IKCE). In this method, a composition step is done the same as before, as shown in Eqs. (1) and (2), with the remark that $\boldsymbol{\phi}^{(l_c)}(\boldsymbol{x}_c)$ indicates a finite-length embedding vector, and transpose operators for one or more levels of embedding spaces are defined as follows:

$$\boldsymbol{T}^{(l_c,l_p)}(s_c,s_p)=\begin{bmatrix} t_{1,1}^{(l_c,l_p,s_c,s_p)} & t_{1,2}^{(l_c,l_p,s_c,s_p)} & \cdots \\ \vdots & \vdots & \cdots \\ t_{m,1}^{(l_c,l_p,s_c,s_p)} & t_{m,2}^{(l_c,l_p,s_c,s_p)} & \cdots \end{bmatrix}_{m\times u^{(l_c,l_p)}} \tag{17}$$

Each row in this matrix definition of a transpose operator is a vector in a kernel space with potentially infinite length, $u^{(l_c,l_p)}=\infty$. In other words, a transpose operator in IKCE can generally be seen as a fixed-length vector of functionals belong to one or multiple kernel spaces;

$$\boldsymbol{T}^{(l_c,l_p)}(s_c,s_p)=\begin{bmatrix} \boldsymbol{f}_1^{(l_c,l_p)}(\,.\,;s_c,s_p) \\ \vdots \\ \boldsymbol{f}_m^{(l_c,l_p)}(\,.\,;s_c,s_p) \end{bmatrix} \tag{18}$$

Algorithm 1 Simplified pseudo-code to estimate the score function of sDKCE. Non-italic, bold terms are the keywords of the pseudo-code syntax. Non-italic, non-bold terms with capital letters indicate function names. The italic font represents variables, where scalar variables are non-bold, and vector variables are in bold non-capital letters. Higher-order tensors and set variables are also shown in bold capital letters.

```
1: function SDKCE_SCORE($\boldsymbol{v}, \boldsymbol{s}, \boldsymbol{I}, y$)
2:     $r \leftarrow 0$
3:     for $j \leftarrow 0$ to $|\boldsymbol{Z}|$ do
4:         $r \leftarrow r + \boldsymbol{beta}[j, y] *$ EMBEDDING_INNERPROD($\boldsymbol{v}, \boldsymbol{s}, \boldsymbol{I}$, size($\boldsymbol{I}$), $\boldsymbol{Z}[j]$)
5:     end for
6:     return $r$
7: end function

8: function EMBEDDING_INNERPROD($\boldsymbol{v}, \boldsymbol{s}, \boldsymbol{I}, i, \boldsymbol{w}$)
9:     $k \leftarrow 0$
10:    $ptag \leftarrow \boldsymbol{s}[i]$
11:    $alpha\_sum \leftarrow$ SUM($\boldsymbol{alpha}[ptag]$)
12:    for $c$ in $\boldsymbol{I}$ do
13:        $kc \leftarrow 0$
14:        $ctag \leftarrow \boldsymbol{s}[c]$
15:        if $c < |\boldsymbol{v}|$ then
16:            for $t1 \leftarrow 0$ to $|\boldsymbol{X}|$ do
17:                $kw \leftarrow$ KERNEL($\boldsymbol{w}, \boldsymbol{X}[t1]$)
18:                for $t2 \leftarrow 0$ to $|\boldsymbol{X}|$ do
19:                    $kc \leftarrow kc + \boldsymbol{nu}[ptag, ctag, t1, t2] *$ KERNEL($\boldsymbol{v}[c], \boldsymbol{X}[t2]$) $* kw$
20:                end for
21:            end for
22:        else
23:            $kc \leftarrow$ EMBEDDING_INNERPROD($\boldsymbol{v}, \boldsymbol{s}, \boldsymbol{I}, c, \boldsymbol{w}$)
24:        end if
25:        $k \leftarrow k + (\boldsymbol{alpha}[ptag, ctag] / alpha\_sum) * kc$
26:    end for
27:    return $k$
28: end function
```

where each $\boldsymbol{f}_t^{(l_c,l_p)}(\,.\,; s_c, s_p)$ is a functional in a kernel space and m indicates the number of these functionals. Each of these functionals is theoretically defined as follows:

$$\boldsymbol{f}_t^{(l_c,l_p)}(\,.\,; s_c, s_p) = \int_{\mathcal{X}} \boldsymbol{\phi}'^{(l_c,l_p)}(\boldsymbol{x})\, d\nu_t^{(l_c,l_p)}(\boldsymbol{x}, s_c, s_p)\,, \tag{19}$$

where $\nu_t^{(l_c,l_p)}(\boldsymbol{x}, s_c, s_p)$ is an arbitrary measure which is valid in its corresponding kernel space. Similarly, if this measure is defined as a conditional probability function, $\nu_t^{(l_c,l_p)}(\boldsymbol{x}, s_c, s_p) = P(\boldsymbol{x} \mid s_c, s_p, l_c, l_p)$, the corresponding functional becomes an expectation in its kernel space.

$$\boldsymbol{f}_t^{(l_c,l_p)}(\,.\,; s_c, s_p) = \mathbb{E}^{(t)}_{\boldsymbol{X}|\boldsymbol{Z}}\left[\boldsymbol{\phi}'^{(l_c,l_p)}(\boldsymbol{X}) \mid s_c, s_p, l_c, l_p\right] \tag{20}$$

The empirical estimate of these functionals can be computed using available training sub-samples as shown below.

$$\hat{\boldsymbol{f}}_t^{(l_c,l_p)}(\,.\,; s_c, s_p) = \sum_{\tilde{\boldsymbol{x}} \in \tilde{\mathcal{X}}^{(l_c,l_p)}} \nu_t(\tilde{\boldsymbol{x}}, s_c, s_p)\, \boldsymbol{\phi}'^{(l_c,l_p)}(\tilde{\boldsymbol{x}}) \tag{21}$$

By substituting this transpose operator within Eq. (2), the definition of transposed embedding can be expanded as follows:

$$\begin{aligned}
\boldsymbol{\psi}^{(l_c,l_p)}(\boldsymbol{x}_c, s_c, s_p) &= \hat{\boldsymbol{T}}^{(l_c,l_p)}(s_c, s_p)\, \boldsymbol{\phi}'^{(l_c,l_p)}(\boldsymbol{x}_c) \\
&= \begin{bmatrix} \hat{\boldsymbol{f}}_1^{(l_c,l_p)}(\,.\,; s_c, s_p) \\ \vdots \\ \hat{\boldsymbol{f}}_m^{(l_c,l_p)}(\,.\,; s_c, s_p) \end{bmatrix} \boldsymbol{\phi}'^{(l_c,l_p)}(\boldsymbol{x}_c) \qquad (22) \\
&= \begin{bmatrix} \left(\sum_{\tilde{\boldsymbol{x}} \in \tilde{\mathcal{X}}^{(l_c,l_p)}} \nu_1(\tilde{\boldsymbol{x}}, s_c, s_p)\, \boldsymbol{\phi}'^{(l_c,l_p)}(\tilde{\boldsymbol{x}})\right)^T \boldsymbol{\phi}'^{(l_c,l_p)}(\boldsymbol{x}_c) \\ \vdots \\ \left(\sum_{\tilde{\boldsymbol{x}} \in \tilde{\mathcal{X}}^{(l_c,l_p)}} \nu_m(\tilde{\boldsymbol{x}}, s_c, s_p)\, \boldsymbol{\phi}'^{(l_c,l_p)}(\tilde{\boldsymbol{x}})\right)^T \boldsymbol{\phi}'^{(l_c,l_p)}(\boldsymbol{x}_c) \end{bmatrix} \\
&= \begin{bmatrix} \sum_{\tilde{\boldsymbol{x}} \in \tilde{\mathcal{X}}^{(l_c,l_p)}} \nu_1(\tilde{\boldsymbol{x}}, s_c, s_p)\;\; \boldsymbol{\phi}'^{(l_c,l_p)}(\tilde{\boldsymbol{x}})^T \boldsymbol{\phi}'^{(l_c,l_p)}(\boldsymbol{x}_c) \\ \vdots \\ \sum_{\tilde{\boldsymbol{x}} \in \tilde{\mathcal{X}}^{(l_c,l_p)}} \nu_m(\tilde{\boldsymbol{x}}, s_c, s_p)\;\; \boldsymbol{\phi}'^{(l_c,l_p)}(\tilde{\boldsymbol{x}})^T \boldsymbol{\phi}'^{(l_c,l_p)}(\boldsymbol{x}_c) \end{bmatrix}
\end{aligned}$$

Because embeddings are explicitly achieved here, computational complexity of inference and learning procedures of IKCE can be significantly lower than DKCE. Due to involving the intermediate mappings and the embedding levels in the definition of the transpose operators of IKCE, its score function is defined the same as Eq. (6) or (9) corresponds to the task.

A simplified pseudo-code of IKCE is presented in Alg. 2. The input structured object is shown in a tuple $(\boldsymbol{v}, \boldsymbol{s}, \boldsymbol{I})$ as described in Section 3.1. The first function in the algorithm is declared as IKCE_SCORE$(\boldsymbol{v}, \boldsymbol{s}, \boldsymbol{I}, y)$. It takes as inputs a structured object and a label, then it calculates the score of the input pair following the IKCE method. More precisely, this function first embeds the given structured object using the second declared function, EMBED$(\boldsymbol{v}, \boldsymbol{s}, \boldsymbol{I})$. Then it calculates the score value by weighted summation of inner products of the obtained embeddings and some support embeddings as per Eq. (6). The used support embeddings are selected according to the level l, at which the input structured-object is embedded. The function EMBED implements the recursive composition mechanism of the IKCE method in a dynamic programming manner. To obtain the list of embedding level of all constituents of an input, we use the function LEVELS. The body of this simple function is not presented for conciseness. The third function declared as TRANSPOSE$(\boldsymbol{e}, ptag, ctag, pl, cl)$ is used to perform transpositions in accordance with Eq. (22). In this function, $\boldsymbol{X}[pl, cl]$ indicates the set of sub-samples in $\tilde{\mathcal{X}}$ which embedding levels of them and their parent are cl and pl, respectively. In addition to SUM which previously used in Alg. 1, two other auxiliary functions are used in this algorithm:

Algorithm 2 Simplified pseudo-code of the score function of IKCE. Description of the syntax is the same as Alg. 1.

```
1:  function IKCE_SCORE($\boldsymbol{v}, \boldsymbol{s}, \boldsymbol{I}, y$)
2:      $r \leftarrow 0$
3:      $\boldsymbol{e},\ l \leftarrow$ EMBED($\boldsymbol{v}, \boldsymbol{s}, \boldsymbol{I}$)
4:      for $j \leftarrow 0$ to $|\boldsymbol{Z}[l]|$ do
5:          $r \leftarrow r\ +\ \boldsymbol{beta}[j, y] *$ INNERPROD($\boldsymbol{e}, \boldsymbol{Z}[l, j]$)
6:      end for
7:      return $r$
8:  end function

9:  function EMBED($\boldsymbol{v}, \boldsymbol{s}, \boldsymbol{I}$)
10:     $\boldsymbol{l} \leftarrow$ LEVELS($\boldsymbol{I}$)
11:     $\boldsymbol{E} \leftarrow$ ZEROS($|\boldsymbol{I}|, m$)
12:     for $i \leftarrow 0$ to $|\boldsymbol{I}|$ do
13:         $ptag \leftarrow \boldsymbol{s}[i]$
14:         if $\boldsymbol{l}[i] > 0$ then
15:             $alpha\_sum \leftarrow$ SUM($\boldsymbol{alpha}[ptag]$)
16:             for $c$ in $\boldsymbol{I}[i]$ do
17:                 $ctag \leftarrow \boldsymbol{s}[c]$
18:                 $\boldsymbol{e} \leftarrow \boldsymbol{e} + (\boldsymbol{alpha}[ptag, ctag]\,/\,alpha\_sum) *$
    TRANSPOSE($\boldsymbol{E}[c], ptag, ctag, \boldsymbol{l}[i], \boldsymbol{l}[c]$)
19:             end for
20:         else
21:             $\boldsymbol{e} \leftarrow$ TRANSPOSE($\boldsymbol{v}[i], ptag, 0, 1, 0$)
22:         end if
23:         $\boldsymbol{E}[i] \leftarrow \boldsymbol{e}$
24:     end for
25:     $i \leftarrow |\boldsymbol{I}|-1$
26:     return $\boldsymbol{E}[i],\ \boldsymbol{l}[i]$
27: end function

28: function TRANSPOSE($\boldsymbol{e}, ptag, ctag, pl, cl$)
29:     $\boldsymbol{te} \leftarrow$ ZEROS($m$)
30:     $\boldsymbol{U} \leftarrow \boldsymbol{X}[pl, cl]$
31:     for $f \leftarrow 0$ to $m$ do
32:         for $t \leftarrow 0$ to $|\boldsymbol{U}|$ do
33:             $\boldsymbol{te}[f] \leftarrow \boldsymbol{te}[f] + \boldsymbol{nu}[ptag, ctag, f, t] *$KERNEL($\boldsymbol{e}, \boldsymbol{U}[t]$)
34:         end for
35:     end for
36:     return $\boldsymbol{te}$
37: end function
```

INNERPROD and ZEROS. The former calculates the inner product of two vectors, and the latter creates a zero vector or a zero matrix. Similarly, if we consider the evaluation of kernel function as a single instruction, the time complexity to compute the IKCE score function is $O\big((|\boldsymbol{Z}| + |\boldsymbol{I}| \times n_c \times |\tilde{\mathcal{X}}|) \times m\big)$, where n_c is the maximum number of children of a node. Because $|\boldsymbol{Z}|$ is equal to $|\tilde{\mathcal{X}}|$ at worst and n_c is always less than $|\boldsymbol{I}|$, the time complexity of the algorithm can be summarized as $O(m\, n^2 |\tilde{\mathcal{X}}|)$, where n is the maximum number of constituents of an input.

A challenge in using the IKCE method is how to select several appropriate functionals for its transpose operators, which can be done in various ways. One option is to choose different hyper-parameters for each functional kernel. Another method is to use different basic kernel functions for each or each group of functionals. As another option, we can divide samples in the training set to multiple groups using a clustering algorithm. Then, one or more functionals are considered for each cluster such that the empirical estimate of each functional is computed using the samples belonging to its corresponding cluster. Finally, another simple solution is to assign different initial values to the coefficients to encourage obtaining various functionals after training. According to the complexity of the objective function of the IKCE learning optimization problem, this expectation seems reasonable.

3.5. Learning

The main parameters of a KCE model are $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{N}$, which indicate the matrix of importance coefficients for composition, the measure matrix of score function, and the measure tensor of transpose operators, respectively. We use $\boldsymbol{\theta}$ to refer to all these parameters simultaneously, i.e., $\boldsymbol{\theta} = \{\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{N}\}$. For both the proposed methods, we have $\boldsymbol{\alpha} \in \mathbb{R}^{|S|} \times \mathbb{R}^{|S|}$ and $\boldsymbol{\beta} \in \mathbb{R}^{|\tilde{\mathcal{X}}|} \times \mathbb{R}^{|\mathcal{Y}|}$, where S is the set of subspace tags. But $\boldsymbol{N}$ is differently defined for each of them, because of the difference between their transpose operator definitions. Specifically, we have $\boldsymbol{N} \in \mathbb{R}^{|S|} \times \mathbb{R}^{|S|} \times \mathbb{R}^{|\tilde{\mathcal{X}}|} \times \mathbb{R}^{|\tilde{\mathcal{X}}|}$ and $\boldsymbol{N} \in \mathbb{R}^{|S|} \times \mathbb{R}^{|S|} \times \mathbb{R}^{m} \times \mathbb{R}^{|\tilde{\mathcal{X}}|}$ for DKCE and IKCE, respectively.

In order to estimate the parameters, we define a risk function by inspiring of the max-margin estimation framework [63]. Accordingly, we want the score of the highest scoring correct output y^* to be larger up to a margin defined by the loss Δ; $\forall\, y \in \mathcal{Y}, d \in \{0, \ldots, D\}, i \in \{0, \ldots, n^{(d)}\}$:

$$g_{\text{KCE}}(\boldsymbol{x}_i^{(d)}, y_i^{(d)}; \boldsymbol{\theta}) \geq g_{\text{KCE}}(\boldsymbol{x}_i^{(d)}, y; \boldsymbol{\theta}) + \Delta(y_i^{(d)}, y)\ , \qquad (23)$$

where D is the size of training set and $n^{(d)}$ indicates the number of all labeled constituents of dth sample in the training set. These desiderata lead us to the following regularized risk function:

$$\mathcal{Q}(\boldsymbol{\theta}) = \frac{1}{D}\sum_{d=0}^{D}\sum_{i=0}^{n^{(d)}} \ell\left(\boldsymbol{x}_i^{(d)}, y_i^{(d)}; \boldsymbol{\theta}\right) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|^2 \qquad (24)$$

$$\ell\left(\boldsymbol{x}_i^{(d)}, y_i^{(d)}; \boldsymbol{\theta}\right) = \max_{y\in\mathcal{Y}} \left\{g_{\text{KCE}}(\boldsymbol{x}_i^{(d)}, y; \boldsymbol{\theta}) + \Delta(y_i^{(d)}, y)\right\} - g_{\text{KCE}}(\boldsymbol{x}_i^{(d)}, y_i^{(d)}; \boldsymbol{\theta})$$

For those tasks in which the input is a pair of structured objects, such as two sentences, with a single label, the learning problem is defined as follows:

$$\mathcal{Q}(\boldsymbol{\theta}) = \frac{1}{D}\sum_{d=0}^{D} \ell\left(\boldsymbol{x}^{(d,1)}, \boldsymbol{x}^{(d,2)}, y^{(d)}; \boldsymbol{\theta}\right) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|^2 \qquad (25)$$

$$\ell\left(\boldsymbol{x}^{(d,1)}, \boldsymbol{x}^{(d,2)}, y^{(d)}; \boldsymbol{\theta}\right) = \max_{y\in\mathcal{Y}} \left\{g_{\text{KCE}}(\boldsymbol{x}^{(d,1)}, \boldsymbol{x}^{(d,2)}, y; \boldsymbol{\theta}) + \Delta(y^{(d)}, y)\right\} - g_{\text{KCE}}(\boldsymbol{x}^{(d,1)}, \boldsymbol{x}^{(d,2)}, y^{(d)}; \boldsymbol{\theta})$$

Minimizing each one of these objectives maximizes the correct output's score and minimizes (up to a margin) the score of the highest scoring but incorrect output. The loss function is usually defined as an indicator function, i.e., $\Delta(y_i^{(d)}, y) = \mathbb{I}\{y_i^{(d)} \neq y\}$ or $\Delta(y^{(d)}, y) = \mathbb{I}\{y^{(d)} \neq y\}$.

An important issue in a KCE method is selection of an appropriate kernel. Because its effectiveness heavily depends on the power of the applied kernel [64]. A good way to achieve a rich kernel is multiple kernel learning (MKL) [65,66]. The primary idea of the MKL approach is to combine multiple kernels instead of using a single one. These different kernels can correspond to using information coming from multiple sources, e.g., different representations [67]. Accordingly, an option to use MKL in our methods is to use multiple different word embedding spaces to evaluate basic kernels for atomic constituents (words). Then we use a weighted sum of them as the final kernel evaluations which are required at the last level of compositions. In this way, the weights can be added to the main parameters of the model to be simultaneously estimated with them. Also, hyper-parameters of used basic kernels can be considered as the additional parameters of the model and can be learned with its main parameters. To overcome computation complexity of the learning problem, the parameters related to the kernels and the main parameters of a KCE model can be estimated in two separate phases iteratively.

Table 1
Specifications of the Stanford Sentiment Tree-bank (SST) dataset and the Sentences Involving Compositional Knowledge (SICK) dataset.

Dataset	Size of subsets			Size of label set
	Training	Development	Test	
SST	8544	1101	2210	5
SICK	4500	500	4927	3

Table 2
Optimal values for the hyper-parameters of our models.

Hyper-parameter	IKCE		sDKCE	
	SST	SICK	SST	SICK
Number of functionals of transpose operators	30	50	–	–
Learning rate	0.01	0.01	0.01	0.1
Regularization weight	0.05	0.1	0.05	0.1

4. Experiments

4.1. Dataset and experimental settings

We evaluate our models on Stanford Sentiment Treebank (SST) [14] which is a standard dataset for sentiment analysis. It contains fully labeled parse trees and is built upon 10 662 reviews. Each node of these parse trees has a sentiment label. The sentiment label set is {0, 1, 2, 3, 4}, where the numbers mean very negative, negative, neutral, positive, and very positive, respectively. We used the standard train/dev/test split of the dataset like the compared methods. In addition, we add part-of-speech (POS) tags to all nodes of the trees. It is done using the latest version of Stanford Parser [68]. Because the newer parser generates trees different from those provided in the datasets, we consider the 'unknown' tag for the phrases which exist according to the original trees but are not tagged by the parser. The POS tags are considered as the embedding subspace tags indicating the source and the destination of the transpositions in KCE. Another dataset used in the experiments is Sentences Involving Compositional Knowledge (SICK) [23]. It is a standard dataset for natural language inference task. In this task, the model reads two sentences (a premise and a hypothesis), and outputs a judgment of entailment, contradiction, or neutral. In fact, these labels reflect the relationship between the meanings of the two sentences. The SICK dataset consists of 9927 sentence pairs with the standard train/dev/test split. Similarly, we use the Stanford parser to add parse trees with POS tags to the sentences. Table 1 summarizes the specifications of both the datasets.

To achieve a relatively richer kernel, we use a MKL solution with two Gaussian kernels. Each of the kernel functions is used in a different word embedding space. In other words, we use two different representing vectors for each word to perform final kernel evaluations, as explained in Section 3.5. The pre-trained models used to obtain the word vectors are FastText with 300-dimension [34] and CharNGram with 100-dimension [69]. The FastText model has been trained on the Common Crawl dataset with 840B tokens and 2.2M vocabulary size. The CharN-Gram model has been trained using the case-sensitive English Wikipedia text corpus.

The initial value of the bandwidth parameter of both the Gaussian kernels is set 0.5; their final optimal values are estimated during the kernel learning phase. The different functionals of the transpose operators of our IKCE model are achieved by initializing their measure coefficients with random values and estimating

Table 3
Accuracy on the Stanford Sentiment Tree-bank dataset (numbers in percentage), where NA means "Not Available".

Model	Phrase-level		Root-level	
	5-class	2-class	5-class	2-class
RNN	79.0	86.1	43.2	82.4
MV-RNN	78.7	86.8	44.4	82.9
RNTN	80.7	87.6	45.7	85.4
LSTM	NA	86.7	45.6	85.6
Tree-LSTM	NA	87.9	46.3	85.8
mTree-LSTM	NA	85.7	46.7	85.7
sDKCE	79.4	86.9	41.2	81.5
IKCE	81.7	90.3	48.1	86.3

their optimal values during training. We use the Xavier initialization technique to set initial value of these measure coefficients like the other KCE parameters of our models. Also, to determine the number of these functionals, we train the IKCE model using the training set and evaluate it on the development set for each one of the candidate values, including {10, 20, 30, 40, 50}. Then we use the optimal value of this hyper-parameter. All the estimations are done using the Adam method. Similar to the number of functionals, we determine the learning rate and L2-regularization weight of the optimization method using the candidate value set {0.001, 0.005, 0.01, 0.05, 0.1}. These values are chosen according to the values commonly used in other similar works. The optimal values for the hyper-parameters are summarized in Table 2.

The computational experiments are performed on an Intel Core i7-4790K system with 24 GB RAM and GTX-970 graphics card, using Ubuntu 17.10 operating system. All the implementations are done using the Python programming language and the TensorFlow framework [70]. Because of very high computation complexity of DKCE and some limitations of the experimental platform, we could only fulfill experiments for the sDKCE and IKCE models.

4.2. Evaluation

We compare our models with several well-known related models, which are also evaluated on one or both the datasets. Table 3 presents the main results of comparing the proposed and the related methods on the SST dataset, from the classification accuracy point of view. The results for the related methods including Recursive Neural Network (RNN), Matrix–Vector RNN (MV-RNN), and Recursive Neural Tensor Network (RNTN) are reported in [71]. Also, for the methods including Long Short-Term Memory (LSTM), Tree-LSTM, and multiplicative Tree-LSTM (mTree-LSTM), the reported results are obtained from [58] in which 5-class phrase-level setting is not available.

As shown in Table 3, while the obtained classification accuracy of sDKCE is slightly more than RNN and MV-RNN in 5-class phrase-level setting, RNTN achieves a higher performance compared to it. Also, the IKCE model outperforms all the other models in this settings. In 2-class phrase-level setting, however, the performance of sDKCE is higher than RNN, MV-RNN and even mTree-LSTM; the obtained results for RNTN and Tree-LSTM are slightly higher than sDKCE. The IKCE model also presents better performance in 2-class phrase-level setting compared to the rest with a relatively high difference.

While the sDKCE model presents the comparable results in both the phrase-level settings, it obtains the lowest performance compared to the other methods at the root-level settings. This implies the lack of the simplified composition mechanism of sDKCE to provide the composed embeddings preserving semantic and syntactic information of sentences well. In contrast, the IKCE model outperforms all the other models in root-level settings too. Especially, the achieved accuracy improvement by IKCE in 5-class root-level setting is higher than other settings. Consequently, this illustrates the effectiveness of the IKCE method more clearly, because of dealing with long sentences and more granularity of class labels in 5-class root-level setting.

Generally, the obtained results for the whole phrases in the phrase-level settings do not necessarily reflect the effectiveness of composition mechanism of a KCE method which has gained good performance; because the number of phrases included in different length level sets is imbalanced. In other words, the number of phrases with length n is reduced exponentially by increasing n. Therefore, good classification performance on short phrases can compensate weakness of the applied composition mechanism. By comparing the results of phrase-level and root-level settings, we see this for the sDKCE model. In fact, this is due to the strong simplifying assumption applied on sDKCE that actually leads to ignorance of the subspace tag-aware transpositions. In contrast, the IKCE model does not suffer from this problem and gained the best results in all settings. This issue is more clearly demonstrated in Figs. 3 and 4, and is discussed in further detail below.

Fig. 3 illustrates the impact of phrase-length on performance of the models. Because this information is not reported for the LSTM, Tree-LSTM, and mTree-LSTM methods, they are excluded in this evaluation. Each point in the figure indicates classification accuracy of a model in dealing with all the phrases with the same length. As shown, the obtained accuracies of all the models for the set of single words, phrases with length one, are high and close together. However, the accuracies are gradually degraded by increasing the length of phrases. The degradation rate of the IKCE model's accuracy is lower than the rest. Especially for the phrases shorter than 13 words, IKCE outperforms all the other methods with relatively a significant difference. Also, for the phrases with length longer than 13, the performance of the IKCE model is better or comparable. These observations implies effectiveness of the composition mechanism of IKCE. For the sDKCE model, its performance is higher than RNN, MV-RNN, and RNTN at the lengths between 2 and 7; however, it cannot keep this relative superiority for the longer phrases. Consequently, the simplified composition mechanism of sDKCE cannot preserve semantic and syntactic information of constituents for the long phrases. In other words, this indicates the key role of the transpose operators of the KCE approach that ignoring them can significantly degrade the performance of the model.

Fig. 4 shows the curves of cumulative accuracies of the methods for 5-class phrase-level setting. Similarly, the LSTM, Tree-LSTM, and mTree-LSTM methods are excluded here for the same reason. At each phrase-length n, the cumulative accuracies are obtained for set of the phrases whose lengths are smaller than $n + 1$. As depicted, the cumulative accuracy curve of the IKCE model is consistently higher than other curves with a relatively large gap. While as shown in Fig. 3, the results obtained for the IKCE model in dealing with long phrases does not have a significant difference with the other models. This is due to very high occurrence frequency of short phrases relative to long phrases in the data. So that a small performance improvement for the short phrases can greatly reduce the effect of lower performance for the long phrases on the cumulative accuracy degradation rate. For the same reason, while the obtained accuracy results of the sDKCE model, which shown in Fig. 3, is lower than the rest at the most phrase-lengths greater than 10, the cumulative accuracy curve of this model is also consistently higher than RNN, MV-RNN, and RNTN until the length 19.

Table 4 summarizes the classification performance of the proposed models and some of the other methods for the SICK dataset. The RNN family models are excluded since their performance on

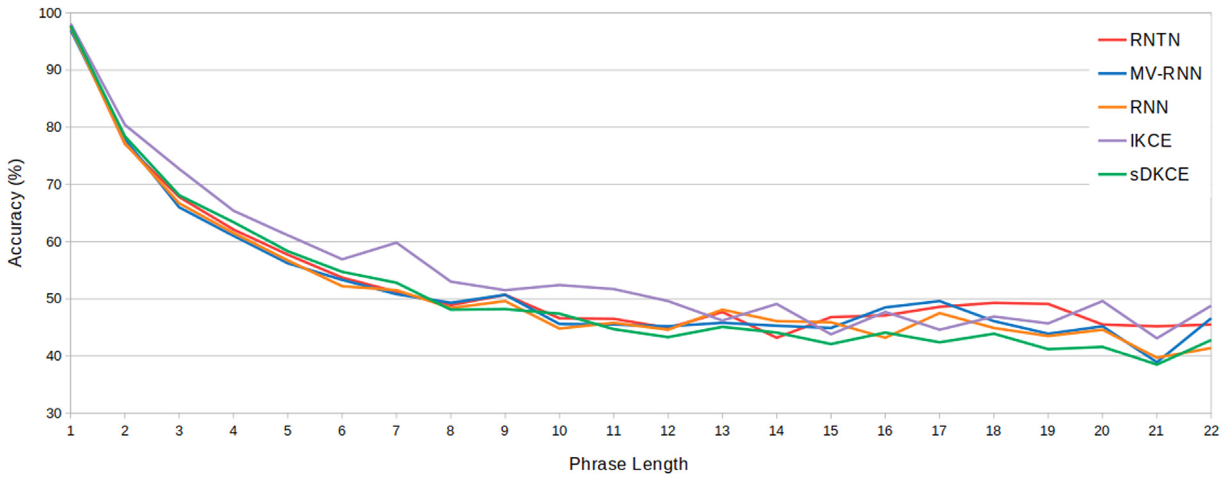

Fig. 3. Accuracy curves for 5-class phrase-level setting. To achieve the results shown in the figure, the phrases in the test set are divided into several subsets based on their length. Each subset contains phrases with the same length. Then for each phrase-length, the accuracies of the models are calculated by applying them on the subset corresponds to that length.

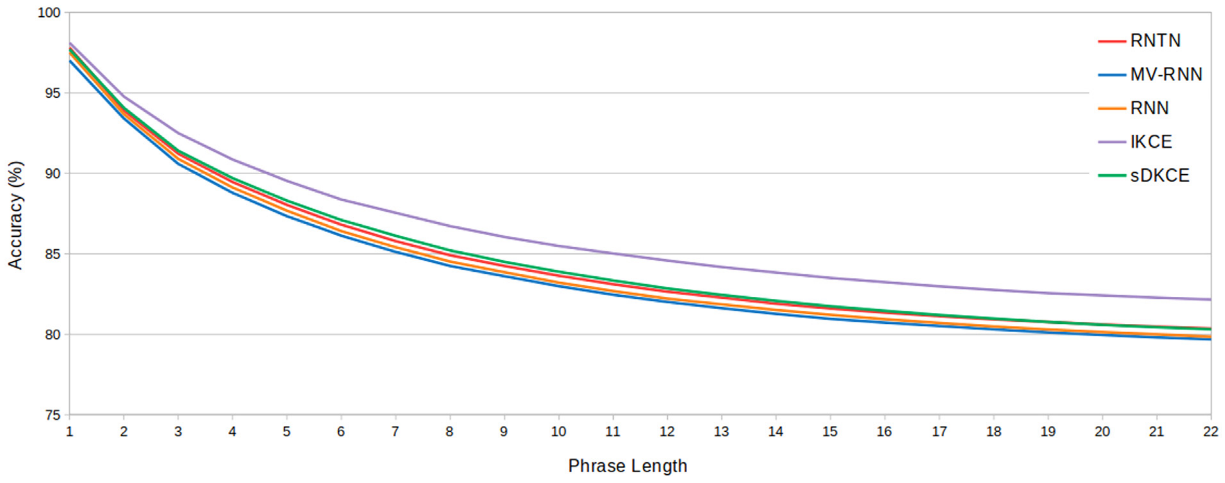

Fig. 4. Cumulative accuracy curves for 5-class phrase-level setting. Similarly, for each phrase-length *n*, a subset is considered which consists of the phrases in the test set with at least *n* words. Then, for each phrase-length, all the models are evaluated using its corresponding subset.

Table 4
Accuracy on the Sentences Involving Compositional Knowledge dataset (numbers in percentage).

Model	All	Long sentence	Negation
LSTM	77.3	74.6	77.5
Tree-LSTM	79.0	78.1	85.3
mTree-LSTM	84.0	81.6	87.8
sDKCE	76.7	73.5	77.1
IKCE	85.1	86.3	88.6

the SICK dataset is not reported. In addition to the standard test set of SICK, we also evaluate the performance of the models on its two different subsets. The first subset, Long Sentence (LS), consists of sentence pairs in the test set where the premise sentence contains at least 18 words. We hypothesize that long sentences are more difficult to handle by sequential models as well as tree-structured models. The second subset, Negation, is a set of sentence pairs where negation words (not, n't or no) do not appear in the premise but appear in the hypothesis. As shown in the table, the IKCE model achieves the highest results, followed by the mTree-LSTM model. For the test set (All) and the Negation subset difference between the performance of IKCE and mTree-LSTM is not significant. But this difference is more for the Long Sentence subset. This shows that the IKCE model can handle long-range dependencies in a sentence more effectively. Similar to the obtained results for the SST dataset, the sDKCE model cannot outperform the LSTM family models here, too; however, its performance is close to the LSTM model. This closeness demonstrates the effectiveness of the KCE idea so that a simple KCE model with a strongly simplified linear composition mechanism but with a suitable kernel can compete with a relatively complex LSTM model. Also, the superiority of IKCE illustrates that if this composition mechanism is purposefully more complicated, a KCE model can outperform even the TreeLSTM models.

5. Conclusion

In this paper, we present two methods to leverage simultaneously the advantages of compositional embedding and kernel methods to embed and classify recursive structured objects. In fact, both the methods operate based on a superior kernel compositional embedding approach. In the first method, DKCE, structured objects are embedded into a potentially infinite dimensional space and the necessary transpose and composition operations are implicitly performed inside that space. The second method, IKCE, uses some elements of one or more kernel spaces to embed structured objects into low-dimensional Euclidean spaces. The experimental results indicate that both the proposed methods, especially IKCE, can provide structured object classifiers whose classification performance is higher or competitive compared to some well-known related methods. Particularly for the short phrases and sentences this superiority is more prominent. In the future work, the KCE approach can be extended to support non-linear composition and transpose operations. Also, applying the proposed methods on huge datasets and other applications dealing with structured objects can be considered for future work.

CRediT authorship contribution statement

Hamed Ganji: Conceptualization, Methodology, Software, Writing - original draft. **Mohammad Mehdi Ebadzadeh:** Supervision. **Shahram Khadivi:** Investigation, Writing - review & editing.

References

[1] P. Goyal, S.R. Chhetri, A. Canedo, Dyngraph2vec: Capturing network dynamics using dynamic graph representation learning, Knowl.-Based Syst. (2019).

[2] N. Guan, D. Song, L. Liao, Knowledge graph embedding with concepts, Knowl.-Based Syst. 164 (2019) 38–44.

[3] H.J. Li, Z. Bu, A. Li, Z. Liu, Y. Shi, Fast and accurate mining the community structure: integrating center locating and membership optimization, IEEE Trans. Knowl. Data Eng. 28 (9) (2016) 2349–2362.

[4] L. Bing, Z.Y. Niu, P. Li, W. Lam, H. Wang, Learning a unified embedding space of web search from large-scale query log, Knowl.-Based Syst. 150 (2018) 38–48.

[5] T. Deng, D. Ye, R. Ma, H. Fujita, L. Xiong, Low-rank local tangent space embedding for subspace clustering, Inform. Sci. 508 (2020) 1–21.

[6] W.Y. Zou, R. Socher, D. Cer, C.D. Manning, Bilingual word embeddings for phrase-based machine translation, in: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, 2013, pp. 1393–1398.

[7] D. Weiss, C. Alberti, M. Collins, S. Petrov, Structured training for neural network transition-based parsing, in: Proceedings of Association for Computational Linguistics (ACL), 2015, pp. 323–333.

[8] M. Pota, F. Marulli, M. Esposito, G. De Pietro, H. Fujita, Multilingual pos tagging by a composite deep architecture based on character-level features and on-the-fly enriched word embeddings, Knowl.-Based Syst. 164 (2019) 309–323.

[9] R.A. Sinoara, J. Camacho-Collados, R.G. Rossi, R. Navigli, S.O. Rezende, Knowledge-enhanced document embeddings for text classification, Knowl.-Based Syst. 163 (2019) 955–971.

[10] T. Hayashi, H. Fujita, Word embeddings-based sentence-level sentiment analysis considering word importance, Acta Polytech. Hung. 16 (7) (2019).

[11] G. Zhou, Y. Zhou, T. He, W. Wu, Learning semantic representation with neural networks for community question answering retrieval, Knowl.-Based Syst. 93 (2016) 75–83.

[12] M. Esposito, E. Damiano, A. Minutolo, G. De Pietro, H. Fujita, Hybrid query expansion using lexical resources and word embeddings for sentence retrieval in question answering, Inform. Sci. 514 (2020) 88–105.

[13] J. Mitchell, M. Lapata, Composition in distributional models of semantics, Cogn. Sci. 34 (8) (2010) 1388–1429.

[14] R. Socher, A. Perelygin, J. Wu, J. Chuang, C.D. Manning, A. Ng, C. Potts, Recursive deep models for semantic compositionality over a sentiment treebank, in: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, 2013, pp. 1631–1642.

[15] K.S. Tai, R. Socher, C.D. Manning, Improved semantic representations from tree-structured long short-term memory networks, in: Proceedings of Association for Computational Linguistics (ACL), 2015, pp. 1556–1566.

[16] V. Vapnik, The Nature of Statistical Learning Theory, Springer, 2000.

[17] B. Schölkopf, A. Smola, K.R. Müller, Nonlinear component analysis as a kernel eigenvalue problem, Neural Comput. 10 (5) (1998) 1299–1319.

[18] G. Baudat, F. Anouar, Generalized discriminant analysis using a kernel approach, Neural Comput. 12 (10) (2000) 2385–2404.

[19] A. Smola, A. Gretton, L. Song, B. Schölkopf, A Hilbert space embedding for distributions, in: International Conference on Algorithmic Learning Theory, Springer, 2007, pp. 13–31.

[20] L. Song, K. Fukumizu, A. Gretton, Kernel embeddings of conditional distributions: A unified kernel framework for nonparametric inference in graphical models, IEEE Signal Process. Mag. 30 (4) (2013) 98–111.

[21] W. Jitkrittum, Z. Szabó, K.P. Chwialkowski, A. Gretton, Interpretable distribution features with maximum testing power, in: Advances in Neural Information Processing Systems, 2016, pp. 181–189.

[22] G. Katz, N. Ofek, B. Shapira, Consent: Context-based sentiment analysis, Knowl.-Based Syst. 84 (2015) 162–178.

[23] M. Marelli, S. Menini, M. Baroni, L. Bentivogli, A sick cure for the evaluation of compositional distributional semantic models, in: Proceedings of LREC, 2014, pp. 216–223.

[24] Y. Bengio, R. Ducharme, P. Vincent, C. Jauvin, A neural probabilistic language model, J. Mach. Learn. Res. 3 (Feb) (2003) 1137–1155.

[25] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Nature 323 (6088) (1986) 533.

[26] J. Li, J. Li, X. Fu, M.A. Masud, J.Z. Huang, Learning distributed word representation with multi-contextual mixed embedding, Knowl.-Based Syst. 106 (2016) 220–230.

[27] H.C. Wu, R.W.P. Luk, K.F. Wong, K.L. Kwok, Interpreting tf-idf term weights as making relevance decisions, ACM Trans. Inf. Syst. (TOIS) 26 (3) (2008) 13.

[28] G. Paltoglou, M. Thelwall, A study of information retrieval weighting schemes for sentiment analysis, in: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, 2010, pp. 1386–1395.

[29] H.J. Li, L. Wang, Multi-scale asynchronous belief percolation model on multiplex networks, New J. Phys. 21 (1) (2019) 015005.

[30] R. Collobert, J. Weston, A unified architecture for natural language processing: Deep neural networks with multitask learning, in: Proceedings of the 25th International Conference on Machine Learning, ACM, 2008, pp. 160–167.

[31] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, in: International Conference on Learning Representations (ICLR).

[32] T. Mikolov, I. Sutskever, K. Chen, G.S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in: Advances in Neural Information Processing Systems, 2013, pp. 3111–3119.

[33] O. Levy, Y. Goldberg, Neural word embedding as implicit matrix factorization, in: Advances in Neural Information Processing Systems, 2014, pp. 2177–2185.

[34] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, A. Joulin, Advances in pre-training distributed word representations, in: Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018), 2018.

[35] J. Pennington, R. Socher, C.D. Manning, Glove: Global vectors for word representation, in: EMNLP, 2014.

[36] M.E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, L. Zettlemoyer, Deep contextualized word representations, in: Proceedings of NAACL, 2018, pp. 2227–2237.

[37] J. Devlin, M.W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of NAACL, 2019, pp. 4171–4186.

[38] O. Levy, Y. Goldberg, Dependency-based word embeddings, in: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), Vol. 2, 2014, pp. 302–308.

[39] R. Schwartz, R. Reichart, A. Rappoport, Symmetric pattern based word embeddings for improved word similarity prediction, in: Proceedings of the Nineteenth Conference on Computational Natural Language Learning, 2015, pp. 258–267.

[40] J. Wieting, M. Bansal, K. Gimpel, K. Livescu, Charagram: Embedding words and sentences via character n-grams, in: Proceedings of Empirical Methods in Natural Language Processing (EMNLP).

[41] B. Athiwaratkun, A.G. Wilson, A. Anandkumar, Probabilistic fasttext for multi-sense word embeddings, in: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Vol. 1, 2018, pp. 1–11.

[42] M. Artetxe, G. Labaka, E. Agirre, A robust self-learning method for fully unsupervised cross-lingual mappings of word embeddings, in: Proceedings of Association for Computational Linguistics (ACL), 2018, pp. 789–798.

[43] H.T. Nguyen, P.H. Duong, E. Cambria, Learning short-text semantic similarity with word embeddings and external knowledge sources, Knowl.-Based Syst. (2019).

[44] N. Mrksic, I. Vulic, D.O. Seaghdha, I. Leviant, R. Reichart, M. Gasic, A. Korhonen, S. Young, Semantic specialisation of distributional word vector spaces using monolingual and cross-lingual constraints, Trans. Assoc. Comput. Linguist. (TACL) (2017).

[45] Z. Bu, H.J. Li, C. Zhang, J. Cao, A. Li, Y. Shi, Graph k-means based on leader identification, dynamic game and opinion dynamics, IEEE Trans. Knowl. Data Eng. (2019).

[46] J. Goikoetxea, A. Soroa, E. Agirre, Bilingual embeddings with random walks over multilingual wordnets, Knowl.-Based Syst. 150 (2018) 218–230.

[47] M. Li, Q. Lu, D. Xiong, Y. Long, Phrase embedding learning based on external and internal context with compositionality constraint, Knowl.-Based Syst. 152 (2018) 107–116.

[48] J. Su, B. Zhang, D. Xiong, Y. Liu, M. Zhang, Alignment-consistent recursive neural networks for bilingual phrase embeddings, Knowl.-Based Syst. 156 (2018) 1–11.

[49] J.M. Eich, A composite holographic associative recall model., Psychol. Rev. 89 (6) (1982) 627.

[50] T.A. Plate, Holographic reduced representations, IEEE Trans. Neural Netw. 6 (3) (1995) 623–641.

[51] D. Widdows, Semantic vector products: Some initial investigations, in: Second AAAI Symposium on Quantum Interaction, Vol. 26, 2008, p. 28th.

[52] S. Clark, S. Pulman, Combining symbolic and distributional models of meaning., in: AAAI Spring Symposium: Quantum Interaction, 2007, pp. 52–55.

[53] S. Rudolph, E. Giesbrecht, Compositional matrix-space models of language, in: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, 2010, pp. 907–916.

[54] R. Socher, J. Pennington, E.H. Huang, A.Y. Ng, C.D. Manning, Semi-supervised recursive autoencoders for predicting sentiment distributions, in: Proceedings of the Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2011, pp. 151–161.

[55] R. Socher, B. Huval, C.D. Manning, A.Y. Ng, Semantic compositionality through recursive matrix-vector spaces, in: Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, Association for Computational Linguistics, 2012, pp. 1201–1211.

[56] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.

[57] W. Zhang, Y. Li, S. Wang, Learning document representation via topic-enhanced lstm model, Knowl.-Based Syst. 174 (2019) 194–204.

[58] N.K. Tran, W. Cheng, Multiplicative tree-structured long short-term memory networks for semantic representations, in: Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics, 2018, pp. 276–286.

[59] A. Gretton, K.M. Borgwardt, M.J. Rasch, B. Schölkopf, A. Smola, A kernel two-sample test, J. Mach. Learn. Res. 13 (Mar) (2012) 723–773.

[60] M. Kanagawa, K. Fukumizu, Recovering distributions from gaussian rkhs embeddings, in: Artificial Intelligence and Statistics, 2014, pp. 457–465.

[61] K. Muandet, D. Balduzzi, B. Schölkopf, Domain generalization via invariant feature representation, in: International Conference on Machine Learning, 2013, pp. 10–18.

[62] M. Kanagawa, Y. Nishiyama, A. Gretton, K. Fukumizu, Filtering with state-observation examples via kernel monte carlo filter, Neural Comput. 28 (2) (2016) 382–444.

[63] B. Taskar, D. Klein, M. Collins, D. Koller, C.D. Manning, Max-margin parsing., in: Proceedings of Empirical Methods in Natural Language Processing (EMNLP), 2004, p. 3.

[64] H. Ganji, S. Khadivi, M.M. Ebadzadeh, Support vector-based fuzzy classifier with adaptive kernel, Neural Comput. Appl. (2017) 1–14.

[65] T. Wang, D. Zhao, Y. Feng, Two-stage multiple kernel learning with multiclass kernel polarization, Knowl.-Based Syst. 48 (2013) 10–16.

[66] T. Wang, J. Lu, G. Zhang, Two-stage fuzzy multiple kernel learning based on hilbert–schmidt independence criterion, IEEE Trans. Fuzzy Syst. 26 (6) (2018) 3703–3714.

[67] M. Gönen, E. Alpaydın, Multiple kernel learning algorithms, J. Mach. Learn. Res. 12 (Jul) (2011) 2211–2268.

[68] C.D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S.J. Bethard, D. McClosky, The Stanford CoreNLP natural language processing toolkit, in: Association for Computational Linguistics (ACL) System Demonstrations, 2014, pp. 55–60.

[69] K. Hashimoto, C. Xiong, Y. Tsuruoka, R. Socher, A joint many-task model: growing a neural network for multiple NLP tasks, in: Proceedings of Empirical Methods in Natural Language Processing (EMNLP), Copenhagen, Denmark, 2017, pp. 446–456.

[70] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, https://www.tensorflow.org/, Software available from tensorflow.org.

[71] R. Socher, Recursive Deep Learning for Natural Language Processing and Computer Vision (Ph.D. thesis), Computer Science Department, Stanford University, 2014.