ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0.
Available from http://www.i6doc.com/en/.

Embeddings and Representation Learning for Structured Data

Benjamin Paaßen[1] Claudio Gallicchio[2] Alessio Micheli[2]
Alessandro Sperduti[3] *

1 - Bielefeld University - CITEC
Inspiration 1, 33619 Bielefeld - Germany

2- University of Pisa - Department of Computer Science
Largo Bruno Pontecorvo 3, 56127 Pisa - Italy

3- University of Padova - Department of Mathematics
Via Trieste 63, 35121 Padova, Italy

Abstract. Performing machine learning on structured data is complicated by the fact that such data does not have vectorial form. Therefore, multiple approaches have emerged to construct vectorial representations of structured data, from kernel and distance approaches to recurrent, recursive, and convolutional neural networks. Recent years have seen heightened attention in this demanding field of research and several new approaches have emerged, such as metric learning on structured data, graph convolutional neural networks, and recurrent decoder networks for structured data. In this contribution, we provide an high-level overview of the state-of-the-art in representation learning and embeddings for structured data across a wide range of machine learning fields.

Traditional machine learning has mostly focused on the question of how to solve problems like classification or regression for fixed, manually engineered data representations [7]. By contrast, representation learning focuses on the challenge of obtaining a vectorial representation in the first place, such that subsequent problems become easy to solve [7]. Such an alternative view is particularly helpful for processing structured data, i.e. sequences, trees, and graphs, where vectorial representations are not immediately available [8].

A wide range of machine learning fields has attempted to construct such vectorial representations for structured data. In this contribution, we provide a high-level overview of these approaches, highlighting shared foundations and properties. Thus, we hope to provide readers with a rich toolbox to handle structured data and sufficient context knowledge to select the fitting method for any given situation.

We begin by introducing key concepts of structured data and vectorial representations before we dive into the various approaches which have been proposed to achieve such representations. This paper concludes with an overview of the contributions in this special session.

*Funding by the CITEC center of excellence (EXC 277) is gratefully acknowledged.

ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0. Available from http://www.i6doc.com/en/.

1 Background

In this section, we introduce terms that are shared among all methods for representation learning and embeddings for structured data. We begin by defining structured data itself. In particular, we define a graph as a triple $\mathcal{G} = (V, E, \xi)$, where $V = \{v_1, \ldots, v_T\}$ is a finite set of nodes, $E \subseteq V \times V$ is a finite set of edges, and $\xi : V \to \mathbb{R}^k$ is a mapping which assigns some vectorial label $\xi(v)$ to each node $v \in V$.

We call a node v a *parent* of another node u if $(v, u) \in E$. Conversely, we call u a *child* of v. We denote the set of all parents of u as $\mathcal{P}_u$, the set of all children of v as $\mathcal{C}_v$, and we define the *neighborhood* of u as $\mathcal{N}(u) := \{u\} \cup \mathcal{P}_u \cup \mathcal{C}_u$.

We call a graph a *tree* if exactly one node exists which has no parents (which we call the *root*) and every other node has exactly one parent. We call all nodes without children *leaves*. As a notational shorthand, we also denote trees in a recursive fashion. In particular, if $u \in V$ is the root of a tree and $v_1, \ldots, v_C$ are its children, then the recursive tree notation is $u(\tilde{x}_1, \ldots, \tilde{x}_C)$, where $\tilde{x}_i$ is the recursive tree notation for the subtree rooted at v_i.

We call a graph a *sequence* if it is a tree with exactly one leaf or if it is the empty graph $\epsilon = (\emptyset, \emptyset, \xi)$. As a notational shorthand, we denote a sequence as $v_1, \ldots, v_T$ where $(v_t, v_{t+1}) \in E$.

Now, we turn to embeddings and representations. Let $\mathcal{X}$ be some arbitrary set. Then, we call a mapping $\phi : \mathcal{X} \to \mathbb{R}^n$ a n-dimensional *embedding* of $\mathcal{X}$. For any $x \in \mathcal{X}$ we call $\phi(x)$ the *representation* of x.

2 Approaches for Embeddings of Structured Data

Approaches for embeddings of structured data can be distinguished along multiple axes. For example, we can distinguish according to the kind of structured data that a method can process - sequences, trees, or full graphs -, whether it generates an implicit or an explicit representation, whether it generates fixed or learned representations, whether nodes or entire structures are embedded, and whether decoding is possible or not.

We begin our list with *kernels* and *distances*, which compute pairwise measures of proximity between structured data based on a pre-defined and fixed algorithm that implicitly corresponds to a vectorial representation. By contrast, *neural networks* learn explicit vectorial representations which are learned from data. In particular, *recurrent neural networks* are designed to process sequential data, *recursive neural networks* for tree-structured data, and *graph convolutional neural networks* for nodes of general graphs. However, there also exist extensions to process nodes of graphs via recurrent neural networks or entire graphs via recursive neural networks.

Note that all of these methods are initially limited to *encoding* a structured datum into a vectorial representation and can not *decode* a vector back into structured data. Our final section covers recent approaches to perform such decodings. A quick overview of methods surveyed in this paper is in Table 1.

ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0.
Available from http://www.i6doc.com/en/.

	sequences	**trees**	**graphs**	**nodes**
		kernels		
fixed	n-grams, substrings [9]	subtrees [10], reservoir activation [11]	shortest paths [12], Weisfeiler-Lehmann [13]**[1]**	Laplacian [14], neighborhood hashing [15]
learned	-	Markov models [16]	-	-
meta	multiple kernel learning [17, 18], **feature space composition [2]**			
		distances		
fixed	string edits [19], alignments [20, 21]	tree edits [22]	-	-
learned	string edits [23], **tf-idf [3]**	tree edits [24]	-	-
meta	dimensionality reduction [25, 26, 27], multiple metric learning [28, 29]			
		neural networks		
encoding	echo state [30, 31, 32], recurrent [33, 34, 35]**[4]**	recursive [36, 37, 38]	recursive [39, 40], hierarchical convolutional [41]	recurrent [42, 43, 44], constructive [45], convolutional [46, 47, 48]
decoding	sequence to sequence [49, 50]	(doubly) recurrent [51, 52, 53], grammar-based [54, 55]	adjacency matrix [56], edge sequence [57, 58] **[5]**	-

Table 1: An overview of the approaches surveyed in this paper. Methods are sorted into columns according to the kind of structured data they process - either sequences, trees, graphs, or nodes within graphs. Each block in the table marks a different class of method, either kernels, distances, or neural networks. In kernels and distances, three rows distinguish between fixed representations, learned representations, and meta-representations built on pre-existing representations. For neural networks, we distinguish between networks that focus on encoding and networks which decode. Contributions of this special session are highlighted via bold print.

ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0. Available from http://www.i6doc.com/en/.

Kernels: We call a mapping $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ over some set $\mathcal{X}$ a *kernel* iff an embedding $\phi : \mathcal{X} \to \mathbb{R}^n$ exists (for possibly infinite n), such that for all $x, y \in \mathcal{X}$ it holds: $k(x, y) = \phi(x)^T \cdot \phi(y)$. Therefore, every kernel on structured data relies - implicitly or explicitly - on an embedding for structured data. In the past decade, a diverse range of structure kernels have emerged, but the conceptual basis is typically the same. A structure kernel defines a class of m (possibly infinite) characteristic substructures $\mathcal{G}_1, \ldots, \mathcal{G}_m$ and defines the embedding ϕ as the sum of the embeddings for all these substructures. More precisely, if $h : \mathcal{X} \to \mathbb{N}^m$ is a mapping of structured data to histograms over the selected substructures and $f : \{\mathcal{G}_1, \ldots, \mathcal{G}_m\} \to \mathbb{R}^n$ is an embedding for the pre-defined set of substructures, then the overall embedding ϕ is given as $\phi(\mathcal{G}) = \sum_{i=1}^{m} h(\mathcal{G})_i \cdot f(\mathcal{G}_i)$. Examples of such substructures include string n-grams, substrings, random walks, shortest paths, or subtrees [9, 10, 12, 13]. Most recently, kernels have also been constructed based on reservoir activations for nodes [11] or learned substructures, such as Markov Model hidden states for nodes [16]. The embedding f for the substructures can be as simple as mapping the ith substructure to the ith unit vector, i.e. ϕ just counts the substructures. Multiple works are specifically devoted to constructing node-specific kernels based on the graph Laplacian or comparing node neighborhoods [14, 15].

Note that, in most cases, it is infeasible to explicitly compute histograms over the substructures due to large or infinite m. Therefore, most kernels are computed directly as $k(x, y) = \sum_{i=1}^{m} \sum_{j=1}^{m} h(x)_i \cdot h(y)_j \cdot k'(\mathcal{G}_i, \mathcal{G}_j)$, e.g. via some dynamic programming scheme [13]. As such, the embedding remains implicit and can not be directly exploited for subsequent learning.

Instead of summing up embeddings of base kernels, one can also concatenate such embeddings, which is the basis for *multiple kernel learning* (MKL). Given a set of base embeddings for graphs $f_1, \ldots, f_m$, MKL learns factors $\alpha_1, \ldots, \alpha_m \in \mathbb{R}^+$ for these embeddings and defines the overall ϕ as $\phi(\mathcal{G}) = \left(\sqrt{\alpha_1} \cdot f_1(\mathcal{G}), \ldots, \sqrt{\alpha_m} \cdot f_m(\mathcal{G})\right)$, such that the resulting learned kernel is given as $k(x, y) = \sum_{i=1}^{m} \alpha_i \cdot f_i(x)^T \cdot f_i(y)$ [17, 18].

Distances: Distance measures on structured data typically quantify distance in terms of effort that is needed to transform one datum into another by means of discrete edit operations such as node deletions, insertions, or replacements. This framework includes measures like the string edit distance, dynamic time warping, alignment distances, or the tree edit distance [19, 20, 21, 22]. These measures are all non-negative, self-equal, and symmetric, but do not necessarily conform to the triangular inequality and are thus not necessarily proper metrics [29]. As with kernels, distances are related to embeddings but are typically computed directly via dynamic programming. In particular, it can be shown that for any self-equal and symmetric function d there exist two embeddings ϕ^+ and ϕ^-, such that for all $x, y \in \mathcal{X}$ it holds $d(x, y)^2 = \|\phi^+(x) - \phi^+(y)\|^2 - \|\phi^-(x) - \phi^-(y)\|^2$ [26]. We can make this embedding explicit by dimensionality reduction methods such as multi-dimensional scaling or t-SNE [25, 27].

It is worth noting that edit distances can be learned in a supervised fashion

ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0.
Available from http://www.i6doc.com/en/.

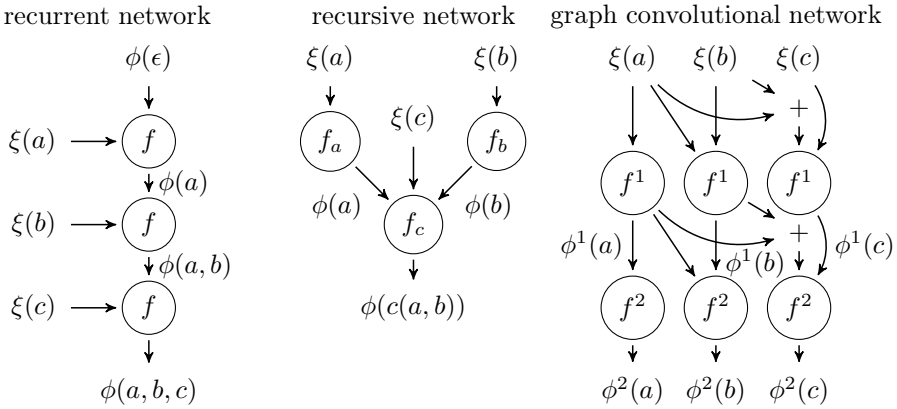

Figure 1: An illustration of the three encoder networks presented in the paper, namely a recurrent network which encodes the sequence a, b, c via Equation 1 (left), a recursive network that encodes the tree $c(a, b)$ via Equation 2 (center), and a two-layer graph convolutional neural network that encodes the nodes of the graph $\mathcal{G} = (\{a, b, c\}, \{(a, b), (a, c), (b, c)\})$ via Equation 3 (right). Function applications/neurons are indicated by circles. The computational flow is indicated by arrows.

by manipulating the costs of single edits to facilitate classification [23, 24].

Recurrent Neural Networks: A recurrent neural network (RNN) maps sequential data $\bar{x} = \vec{x}_1, \ldots, \vec{x}_T \in \mathbb{R}^k$ to a representation $\phi(\bar{x}) \in \mathbb{R}^n$ by means of the recursive equation

$$\phi(\vec{x}_1, \ldots, \vec{x}_t) = f\big(\vec{x}_t, \phi(\vec{x}_1, \ldots, \vec{x}_{t-1})\big), \tag{1}$$

where f is some mapping $f : \mathbb{R}^k \times \mathbb{R}^n \to \mathbb{R}^n$ and where $f(\epsilon)$ is typically defined as the zero vector (also refer to Figure 1, left).

In recurrent neural networks, the function f is a neural network layer, e.g. a classic sigmoid layer of the form $f(\vec{x}_t, \vec{h}_{t-1}) = \sigma\big(\boldsymbol{U} \cdot \vec{x}_t + \boldsymbol{W} \cdot \vec{h}_{t-1}\big)$, where $\boldsymbol{U} \in \mathbb{R}^{n \times k}$ and $\boldsymbol{W} \in \mathbb{R}^{n \times n}$ are weight matrices and σ is a sigmoid function such as the tanh function. A challenge in learning such networks are vanishing or exploding gradients over time, which can be addressed, for instance, by the following strategies. First, one can decide to adapt neither either $\boldsymbol{U}$ nor $\boldsymbol{W}$ but to initialize them in a randomized or deterministic fashion [30, 31], also in a deep setting [32]. Second, one can replace a standard sigmoid layer with a gated architecture that can ignore irrelevant parts of the sequence and thus maintain memory over longer time without being unstable [33, 34, 35]. For example, gated recurrent units [34] define the recurrent function $f(\vec{x}_t, \vec{h}_{t-1}) = \vec{z}(\vec{x}_t, \vec{h}_{t-1}) \odot \vec{h}_t + [\vec{1} - \vec{z}(\vec{x}_t, \vec{h}_{t-1})] \odot \sigma\big(\boldsymbol{U} \cdot \vec{x}_t + \boldsymbol{W} \cdot [\vec{r}(\vec{x}_t, \vec{h}_{t-1}) \odot \vec{h}_{t-1}]\big)$, where

ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0. Available from http://www.i6doc.com/en/.

$\odot$ denotes the element-wise product and where $z(\vec{x}_t, \vec{h}_{t-1}) \in [0,1]^n$ as well as $\vec{r}(\vec{x}_t, \vec{h}_{t-1}) \in [0,1]^n$ are so-called gates, computed via standard sigmoid layers as above.

The objective function of recurrent neural networks is typically to map the input sequence $\bar{x}$ to an output sequence $\bar{y} = \vec{y}_1, \ldots, \vec{y}_T \in \mathbb{R}^l$ by means of an output sigmoid layer $g : \mathbb{R}^n \to \mathbb{R}^l$ such that $g(\vec{h}_t) \approx \vec{y}_t$ for all $t \in \{1, \ldots, T\}$. However, recurrent neural networks can also be trained to auto-encode sequences or to decode to other sequences [49].

Note that RNNs can also be applied to embed nodes in a graph by considering the embedding $\phi(v)$ of a node $v \in V$ as part of a state vector in a RNN [42, 43, 44]. More precisely, let $\phi^t(v)$ denote the embedding of node v at time t. Then, we can construct the node embedding at time $t+1$ via the equation $\phi^{t+1}(v) = \sum_{u \in \mathcal{N}(v)} \hat{f}\big(\xi(v), \phi^t(v), \xi(u), \phi^t(u)\big)$ or, similarly, via the equation $\phi^{t+1}(v) = \hat{f}\Big(\xi(v), \phi^t(v), \sum_{u \in \mathcal{N}(v)} \xi(u), \sum_{u \in \mathcal{N}(v)} \phi^t(u)\Big)$ for some mapping $\hat{f} : \mathbb{R}^k \times \mathbb{R}^n \times \mathbb{R}^k \times \mathbb{R}^n \to \mathbb{R}^n$. In both cases, this collapses to a recurrent neural network according to Equation 1 if we consider the concatenation of all node labels $\xi(v)$ as the input for each time step t and the concatenation of all embeddings $\phi^t(v)$ as state vector at time t. The training of this network is typically guided by some supervised objective as in regular RNNs [42, 43, 44].

As [42] have shown, letting this network run is guaranteed to converge to a fix point if $\hat{f}$ is a contractive map. In other words, one can let the network run for a sufficiently large time and then use the resulting embedding at that time as an approximation of the fix point and thus as an embedding of the nodes [44].

Recursive Neural Networks: Recursive neural networks are an extension of recurrent neural networks for tree structured data. Given a tree $v(\tilde{x}_1, \ldots, \tilde{x}_C)$, a recursive neural network is defined by the recursive equation

$$\phi\big(v(\tilde{x}_1, \ldots, \tilde{x}_C)\big) = f_v\big(\xi(v), \phi(\tilde{x}_1), \ldots, \phi(\tilde{x}_C)\big), \tag{2}$$

where f_v is typically a sigmoid layer (also refer to Figure 1, center) [36, 37, 38]. In other words, the encoding starts with the leaves of the tree and then processes the tree bottom-up until an embedding for the entire tree is obtained at the root. Extensions of recursive neural networks to the treatment of directed postional acyclic graphs (DPAGs) have been introduced in [39, 40].

Note that the construction of f_v by be challenging of the children have no clear positional order or if the number of children is not consistent among nodes. Such problems can be addressed by using order-invariant operators like sum or product to aggregate child embeddings, to fill missing children with special tokens like zero vectors, to normalize the trees to binary structure, or to learn specific functions for different kinds of nodes with different number of children.

Graph Convolutional Neural Networks: Graph Convolutional Neural Networks (GCNs) generate embeddings of nodes in graphs similarly to RNNs but via a layered feedforward architecture. In particular, let $\phi(v)^l$ denote the embedding

ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0. Available from http://www.i6doc.com/en/.

of node v in layer l of the network, where $\phi(v)^0 = \xi(v)$. Then, the embedding in layer $l+1$ is obtained via the equation

$$\phi(v)^{l+1} = f^{l+1}\big(\phi(v)^l, \sum_{u \in \mathcal{P}(v)} \alpha(u,v) \cdot \phi(u)^l\big), \tag{3}$$

where f^l is a sigmoid layer and $\alpha(u,v)$ is some connectivity factor depending on the graph structure (also refer to Figure 1, right) [46]. Note that this equation differs from the RNN equation of [42, 43, 44] in that we use different parameters for each layer. Also note that the embedding in the lth layer integrates information from nodes up to distance l in the graph.

The idea to treat the mutual dependencies (graph cycles) through different neural network layers and to extend the nodes embedding by composition of the information of previous layers was originally introduced (and formally proved) in the context of constructive approaches [45]. Therein, the concept/terminology of visiting (the nodes) of the graphs corresponds to the terminology of convolution over (the nodes) of the graphs used in GCN. Indeed, the main differences between the model in [45] (NN4G) and GCN are related to the use of an incremental construction of the deep NN for NN4G instead of the end-to-end training of GCN (with advantage for NN4G in terms of divide et impera automatic design and layer by layer learning). A recent model proposal using the costruction of NN4G in the context of generative models is in [59].

As with RNNs, GCNs are trained in a supervised fashion where the last layer of a GCN is considered as the output of the network. Further, we can train GCNs semi-supervised by augmenting the supervised loss with a term that forces neighboring nodes to have similar encodings [46]. While vanilla GCNs are limited to graphs for which the structure is known a priori, multiple authors have recently extended GCNs to unknown structure, either by normalizing the neighborhood or by using attention mechanisms [47, 48].

Importantly, the embeddings of GCNs can also be aggregated to achieve an embedding for the entire graph by iteratively clustering nodes to coarser structures and aggregating the embeddings inside structures by a pooling network [41].

Decoder Networks: Decoding vectorial representations back into structured data poses a significant challenge as decoding trees or full graphs is provably harder compared to encoding them [37]. Therefore, present decoding approaches focus on decoding sequential data via recurrent neural networks.

Most prominently, sequence-to-sequence (seq2seq) learning first encodes a sequence as a vector via a recurrent neural and then applies a second recurrent neural network which decodes the sequence step by step until it returns a special end-of-sentence token [49].

Given the success of this scheme for hard machine learning tasks such as machine translation or caption generation [49, 50], researchers have also attempted to apply it to trees or graphs by encoding these structures as sequences. In particular, we can re-write graphs as a sequences of nodes and edges if we impose

ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0. Available from http://www.i6doc.com/en/.

a order on the graph's nodes, e.g. via breadth-first-search [57, 58]. After this re-representation, we can train a recurrent neural network to generate one edge at a time and thus reconstruct the entire graph until an end-of-sentence token is generated [57, 58].

Alternatively, we can exploit grammatical knowledge about the domain. If our data can be described by a context-free grammar (as in the case of chemical molecules or computer programs), generating a structured datum reduces to a sequence of grammar rule applications. Therefore, we can train a recurrent neural network which outputs the current sequence of grammar rules to decode a given datum, which will then also be guaranteed to be syntactically correct [51, 52, 54]. Indeed, grammatical structures can even be used to impose semantic constraints, such as chemical bond properties [55].

3 Special Session Contributions

The contributions in this special session cover a wide range of approaches for representation learning and embeddings for structured data, namely two kernel approaches [1, 2], one distance approach [3], one sequence encoding approach via recurrent neural networks [4], one graph decoding approach via recurrent neural networks [5], and an extension of non-negative matrix factorization to uncover structure in high-dimensional vectorial data [6] (also refer to Table 1).

In more detail, [1] presents a variation of the Weisfeiler-Lehman graph kernel [13] which combines the concept of optimal assignments with multiple kernel learning [18]. In particular, they define the kernel between two graphs $\mathcal{G}$ and $\mathcal{G}'$ as $k(\mathcal{G}, \mathcal{G}') = \max_{M \subseteq \mathcal{B}(\mathcal{G},\mathcal{G}')} \sum_{(u,v)\in M} k'(u, v)$ where $\mathcal{B}(\mathcal{G}, \mathcal{G}')$ is the set of all possible bijections between the nodes of $\mathcal{G}$ and $\mathcal{G}'$ and where k' is a weighted Weisfeiler-Lehman base kernel over nodes. Recall that the Weisfeiler-Lehman kernel counts subtree patterns in the neighborhood of a node. The kernel variation employed by [1] applies weights to these subtree patterns and then optimizes these weights via multiple kernel learning.

[2] propose a scheme which generates a more expressive feature space from a base node kernel by means of a sum of outer products. In particular, the encoding is defined as $\phi(\mathcal{G}) = \sum_{v\in V} f(v) \cdot \sum_{i=1}^{D} \sum_{u\in\mathcal{N}^i(v)} f(u)^T$, where f is the embedding of the base node kernel and $\mathcal{N}^i(v)$ is the i-hop neighborhood of node v. They also consider a version of this kernel where features are selected based on their discriminative value in a linear classifier.

[3] perform metric learning to weigh text features obtained via tf-idf and latent semantic analysis and obtain an explicit, low-dimensional embedding via t-SNE [27].

[4] encode a snapshot of a driving scene incorporating a variable number of vehicles via the semantic pointer architecture [60] and encode a sequence of such snapshots via long short-term memory networks [33, 35] in order to predict the future movement of a single vehicle.

With the the aim is to provide an adpative approach to graph generation from arbitrary distributions, [5] first represent graphs as sequences of edges,

ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0. Available from http://www.i6doc.com/en/.

where the edges are ordered according to their starting node, and auto-encode graphs as vectors by means of a sequence-to-sequence [49] network consisting of two gated recurrent units [34], where the former encodes an edge sequence as a vector and the second decodes the edge sequence from the code vector.

Finally, [6] proposes a variation of hierarchical alternating least squares (HALS) to infer non-negative polynomial signals which best explain a given data set of sequences in the sense that the Euclidean distance between the observed sequences and the sequences produced by a linear combination of non-negative polynomials on the same interval is as small as possible.

Overall, the contributions in this special session push the boundaries of embeddings for structured data forward across a wide range of approaches. This reflects the more intense recent interest in such embeddings in the research community overall and gives hope for further progress in the future.

References

[1] Nils Kriege. Deep weisfeiler-lehman assignment kernels via multiple kernel learning. In *Proc. ESANN*, 2019. Special Session Contribution.

[2] Nicolò Navarin, Dinh van Tran, and Alessandro Sperduti. On the definition of complex structured feature spaces. In *Proc. ESANN*, 2019. Special Session Contribution.

[3] Jelle Bakker and Kerstin Bunte. Efficient learning of email similarities for customer support. In *Proc. ESANN*, 2019. Special Session Contribution.

[4] Florian Mirus, Peter Blouw, Terrence Stewart, and Jörg Conradt. Predicting vehicle behaviour using lstms and a vector power representation for spatial positions. In *Proc. ESANN*, 2019. Special Session Contribution.

[5] Davide Bacciu, Alessio Micheli, and Marco Podda. Graph generation by sequential edge prediction. In *Proc. ESANN*, 2019. Special Session Contribution.

[6] Cécile Hautecoeur and François Glineur. Nonnegative matrix factorization with polynomial signals via hierarchical alternating least squares. In *Proc. ESANN*, 2019. Special Session Contribution.

[7] Yoshia Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[8] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *Bulletin of the Technical Committee on Data Engineering*, 40(3):52–74, 2017.

[9] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2(Feb):419–444, 2002.

[10] Fabio Aiolli, Giovanni Da San Martino, and Alessandro Sperduti. An efficient topological distance-based tree kernel. *IEEE Transactions on Neural Networks and Learning Systems*, 26(5):1115–1120, 2015.

[11] Davide Bacciu, Claudio Gallicchio, and Alessio Micheli. A reservoir activation kernel for trees. In *Proc. ESANN*, pages 29–34, 2016.

[12] Karsten M. Borgwardt, Nicol N. Schraudolph, and S.v.n. Vishwanathan. Fast computation of graph kernels. In *Proc. NIPS*, pages 1449–1456, 2007.

[13] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

[14] Risi Kondor and Horace Pan. The multiscale laplacian graph kernel. In *Proc. NIPS*, pages 2990–2998, 2016.

[15] Nicolò Navarin and Alessandro Sperduti. Approximated neighbours MinHash graph node kernel. In Michel Verleysen, editor, *Proc. ESANN*, pages 281–286, 2017.

[16] Davide Bacciu, Alessio Micheli, and Alessandro Sperduti. Generative kernels for tree-structured data. *IEEE Transactions on Neural Networks and Learning Systems*, 29(10):4932–4946, 2018.

[17] Mehmet Gönen and Ethem Alpaydın. Multiple kernel learning algorithms. *Journal of Machine Learning Research*, 12(Jul):2211–2268, 2011.

[18] Fabio Aiolli and Michele Donini. EasyMKL: a scalable multiple kernel learning algorithm. *Neurocomputing*, 169:215–224, 2015.

[19] Vladimir Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1965.

[20] T. K. Vintsyuk. Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57, 1968.

[21] Osamu Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, 1982.

[22] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989.

[23] Aurélien Bellet, Amaury Habrard, and Marc Sebban. A survey on metric learning for feature vectors and structured data. *arXiv*, abs/1306.6709, 2014.

[24] Benjamin Paaßen, Claudio Gallicchio, Alessio Micheli, and Barbara Hammer. Tree Edit Distance Learning via Adaptive Symbol Embeddings. In *Proc. ICML*, pages 3973–3982, 2018.

[25] John W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, 18(5):401–409, 1969.

[26] Elzbieta Pekalska and Robert Duin. *The Dissimilarity Representation for Pattern Recognition*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2005.

[27] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[28] Babak Hosseini and Barbara Hammer. Efficient metric learning for the analysis of motion data. In *Proc. DSAA*, pages 1–10, 2015.

[29] David Nebel, Marika Kaden, Andrea Villmann, and Thomas Villmann. Types of (dis-)similarities and adaptive mixtures thereof for improved classification learning. *Neurocomputing*, 268:42–54, 2017.

ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0. Available from http://www.i6doc.com/en/.

[30] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, 2004.

[31] Ali Rodan and Peter Tiňo. Simple deterministically constructed cycle reservoirs with regular jumps. *Neural Computation*, 24(7):1822–1852, 2012.

[32] Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. Deep reservoir computing: A critical experimental analysis. *Neurocomputing*, 268:87–99, 2017.

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

[34] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proc. EMNLP*, pages 1724–1734, 2014.

[35] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, 2017.

[36] A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

[37] Barbara Hammer. Recurrent networks for structured data - A unifying approach and its properties. *Cognitive Systems Research*, 3(2):145 – 165, 2002. Integration of Symbolic and Connectionist Systems.

[38] Claudio Gallicchio and Alessio Micheli. Tree echo state networks. *Neurocomputing*, 101:319 – 337, 2013.

[39] A. Micheli, D. Sona, and A. Sperduti. Contextual processing of structured data by recursive cascade correlation. *IEEE Transactions on Neural Networks*, 15(6):1396–1410, Nov 2004.

[40] Barbara Hammer, Alessio Micheli, and Alessandro Sperduti. Universal approximation capability of cascade correlation for structures. *Neural Computation*, 17(5):1109–1159, 2005.

[41] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Proc. NeurIPS*, pages 4805–4815, 2018.

[42] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[43] Claudio Gallicchio and Alessio Micheli. Graph echo state networks. In *Proc. IJCNN*, pages 1–8, 2010.

[44] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. In *Proc. ICLR*, 2016.

[45] Alessio Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

[46] Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proc. ICLR*, 2017.

[47] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proc. NIPS 2017*, pages 1024–1034, 2017.

[48] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *Proc. ICLR*, 2018.

[49] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Proc. NIPS*, pages 3104–3112, 2014.

[50] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proc. ICML*, pages 2048–2057, 2015.

[51] David Alvarez-Melis and Tommi Jaakkola. Tree-structured decoding with doubly-recurrent neural networks. In *Proc. ICLR*, 2017.

[52] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *Proc. ICML*, pages 2323–2332, 2018.

[53] Xinyun Chen, Chang Liu, and Dawn Song. Tree-to-tree neural networks for program translation. In *Proc. NeurIPS*, pages 2552–2562, 2018.

[54] Matt J. Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. In *Proc. ICML*, pages 1945–1954, 2017.

[55] Hanjun Dai, Yingtao Tian, Bo Dai, Steven Skiena, and Le Song. Syntax-directed variational autoencoder for structured data. In *Proc. ICLR*, 2018.

[56] Martin Simonovsky and Nikos Komodakis. Towards variational generation of small graphs. In *Proc. ICLR*, 2018.

[57] Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander Gaunt. Constrained graph variational autoencoders for molecule design. In *Proc. NeurIPS*, pages 7806–7815, 2018.

[58] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *Proc. ICML*, pages 5708–5717, 2018.

[59] Davide Bacciu, Federico Errica, and Alessio Micheli. Contextual graph Markov model: A deep and generative approach to graph processing. In *Proc. ICML*, pages 294–303, 2018.

[60] Chris Eliasmith. *How to build a brain: A neural architecture for biological cognition*. Oxford University Press, 2013.