Substructure Assembling Network for Graph Classification

Xiaohan Zhao
Snap Inc.
homeisland03@gmail.com

Bo Zong
NEC Laboratories, America
bzong@nec-labs.com

Ziyu Guan
Northwest University of China
ziyuguan@nwu.edu.cn

Kai Zhang
Temple University
zhang.kai@temple.edu

Wei Zhao
Xidian University
ywzhao@mail.xidian.edu.cn

Abstract

Graphs are natural data structures adopted to represent real-world data of complex relationships. In recent years, a surge of interest has been received to build predictive models over graphs, with prominent examples in chemistry, computational biology, and social networks. The overwhelming complexity of graph space often makes it challenging to extract interpretable and discriminative structural features for classification tasks. In this work, we propose a novel neural network structure called Substructure Assembling Network (SAN) to extract graph features and improve the generalization performance of graph classification. The key innovation of our work is a unified substructure assembling unit, which is a variant of Recurrent Neural Network (RNN) designed to hierarchically assemble useful pieces of graph components so as to fabricate discriminative substructures. SAN adopts a sequential, probabilistic decision process, and therefore it can tune substructure features in a finer granularity. Meanwhile, the parameterized soft decisions can be continuously improved with supervised learning through back-propagation, leading to optimizable search trajectories. Overall, SAN embraces both the flexibility of combinatorial pattern search and the strong optimizability of deep learning, and delivers promising results as well as interpretable structural features in graph classification against state-of-the-art techniques.

Introduction

Many real-world data have complex structures in which instances are naturally represented as graphs, such as molecules in chemical compounds or proteins, dependence graphs in computer programs, social interactions in online social networks. There is an increasing demand for building predictive models over these graph datasets. For example, inference on the characteristics of a new compound could help in clinical trials (Cumming et al. 2013); Identification over critical subgraphs in program flow graphs could help locate bugs in complex programs (Cheng et al. 2009); Prediction on the biochemical properties of compounds library can greatly facilitate discovery of pharmacological targets (Burbidge, Trotter, and Holden 2000).

Compared with vectorial data, the discrete and semi-structured representation of graphs makes it much more challenging to estimate the mapping between instances and their learning targets. As connectivity and size of graphs could vary significantly across instances, how to capture the key structural information from graph-based objects remains an open problem in the machine learning community.

Various approaches have been proposed to build predictive models on graph datasets. For example, one interesting idea is to explicitly extract local graph patterns as features, such that each graph can be represented as a bag of features that naturally fit in traditional predictive models (e.g., support vector machines) (Yan and Han 2002). Though convenient, the number of potential graph patterns can grow exponentially with graph size and node/edge attributes. Another popular approach is to design graph kernels to measure the similarity between graphs (Neumann et al. 2016), followed by a kernel-based classifier. A graph kernel implicitly decomposes a graph into local structures, such as random walk paths or subtrees, and computes their similarity score.

One critical problem in pattern mining and graph kernel methods is that their feature extraction (either explicit or implicit) is independent from the followup classification or model learning tasks. In the context of graphs where the number of features extracted can be quite huge or even approach infinity (e.g., graph kernels), such unguided feature extraction step can be particularly cumbersome in identifying discriminative graph patterns that are relevant to learning tasks. This, in turn, could severely hamper the generalization performance no matter which classifier is adopted.

Recently, there has been a surge of interest in exploiting the power of neural networks to map graph objects to continuous feature spaces (Niepert, Ahmed, and Kutzkov 2016; Defferrard, Bresson, and Vandergheynst 2016; Duvenaud et al. 2015; Bruna et al. 2014; Scarselli et al. 2009; Li et al. 2016). Although these methods have gained great success in a number of benchmark graph classification tasks, open challenges still exist due to the overwhelming structural complexity of graphs. One limitation of the existing methods is that it can be hard for them to tune graph features at a fine granularity. (1) For spectral methods (Bruna et al. 2014; Defferrard, Bresson, and Vandergheynst 2016), a small adjustment in spectral space results in a global impact to graph feature selection. (2) For the methods in graph space (Simonovsky and Komodakis 2017; Li et al. 2016; Duvenaud et al. 2015), they usually assume that feature selection behaves identically for neighbor nodes with the same

node/edge attributes. This assumption restricts the flexibility and potential to construct discriminative features, since neighbor nodes with the same node/edge attributes are either all selected or all filtered.

In this paper, we propose a novel neural network model called Substructure Assembling Network (SAN) that hierarchically extracts discriminative and interpretable local graph patterns/features to improve the generalization performance of graph classification. The key innovation of our framework is a unified Substructure Assembling Unit (SAU), which is a variant of RNN that performs node-centric edge selection at a fine granularity and assembles useful pieces of graph components to fabricate discriminative substructure features. In particular, SAN adopts a sequential probabilistic decision process; therefore, it enables progressive edge selection on nodes and generates substructure features for graphs of arbitrary sizes. Meanwhile, as classification error can be back-propagated and form a feedback loop, the parameterized, soft decision process can be continuously improved using class labels as its guidance. Overall, SAN embraces both the flexibility of combinatorial pattern search and the strong optimizability of deep learning, and delivers promising results as well as interpretable structural features on graph classification tasks against state-of-the-art methods.

Substructure Assembling Network

In this section, we introduce the detailed design of the Substructure Assembling Network (SAN).

Graph Representation

We first formalize the graph representations considered in this paper. Here, we are interested in general, directed graphs where each node and edge are associated with their attributes (either discrete or continuous). A graph G is denoted by a triplet (V, E, R) where

- V is the set of nodes. $\mathbf{v}_i \in V$ is a vector that encodes attributes of node i;
- E is the set of edges. $\mathbf{e}_{i,j} \in E$ is a vector that encodes attributes of the edge from node i to j;
- R is the set of edge residues, where $r_{i,j} \in R$ is for the edge from i to j. Edge residues take values in $[0, 1]$ and constrain the edge selection process in SAN so that each edge will not be repeatedly selected.

In addition, neighbors of node i is denoted by a sequence of nodes $N_i = \langle j \mid \mathbf{e}_{i,j} \in E \rangle$, where $N_i(k)$ is the k-th neighbor of node i, and $|N_i|$ indicates the size of the neighborhood. We will discuss the ordering of neighborhoods in the next section. Since we focus on graph classification, training data are of the form $\{(G_i, \mathbf{y}_i)\}$ where $\mathbf{y}_i$ is the label vector for G_i in a one-hot coding scheme.

Global Overview

A global picture of SAN is presented in Figure 1. SAN is a deep neural network that takes a graph G of arbitrary size as input and predicts its label $\mathbf{y}$. Conceptually, SAN consists of three major components: SAU layer(s), pooling layer, and fully connected layer.

- SAU layers are the core components for extracting discriminative graph features. The basic building block is called Substructure Assembling Unit (SAU), which is a variant of Recurrent Neural Network (RNN) that performs on each node and is shared among nodes. For each node, SAU sequentially makes progressive decisions on edge and neighbor node selection to form discriminative local substructure. Multiple SAU layers can be cascaded to hierarchically build large substructures from small ones. As shown in Figure 1, the center node in the input graph selects to absorb the two nodes to its left in the first SAU layer and then incorporates the substructure centered at the lower-right node in the second layer.
- The pooling layer aggregates diverse substructures (one for each node as the center) obtained from the last SAU layer into a fixed-length vectorial representation suitable for subsequent classification.
- The fully connected layer is a standard multi-layer neural network that maps input features to class labels.

In the following, we discuss details in these three layers.

SAU layers

SAUs are inspired by two observations from combinatorial algorithms on substructure pattern mining (Yan and Han 2002): (1) any substructure can be assembled by a sequence of edge selection decisions on individual nodes (sequential structure selection); and (2) complex substructure features can be constructed from simpler ones (hierarchically).

We adopt a sequential, probabilistic decision process in the design of SAUs, so that they can explore highly complex graph space, while simultaneously achieving optimizable substructure selection by utilizing class labels as the guidance. Figure 2 gives a simple illustration. Here we have the neighborhood of a node v_i (top left figure), and suppose we want to select a one-hop substructure (top right figure) as the feature. In combinatorial algorithms, typically, hard decisions ($\mathrm{H}(\cdot)$ in Figure 2) are made via discrete operations (e.g., heuristic search), where it is difficult to continuously improve the decisions using labels from classification tasks. In our framework, we employ a probabilistic method to mimic the edge selection process. Instead of hard decisions, soft decisions are made in terms of the conditional probability that a neighbor component should be selected ($\mathrm{P}(\cdot)$ in Figure 2). As a result, prediction errors can be back-propagated to individual SAUs, so as to iteratively improve the parameterized soft decision process, and identify discriminative substructures automatically.

In the following, we discuss in more detail on how SAUs sequentially make progressive decisions for neighbor substructure selection . Typically, we will stack L ($L \geq 1$) SAU layers. Each SAU layer acts as a filter on the output of the previous SAU layer, so that larger substructures can be assembled from smaller ones. Let $G^l = (V^{(l)}, E, R^{(l)})$ be the output of the l-th SAU layer, where $V^{(l)}$ contains feature vectors of the assembled substructures and $R^{(l)}$ includes the updated edge residues after the l-th SAU layer. The input to the first SAU layer is $G^0 = (V^{(0)} = V, E, R^{(0)} = R)$. Inside each layer, a SAU runs concurrently on individual nodes

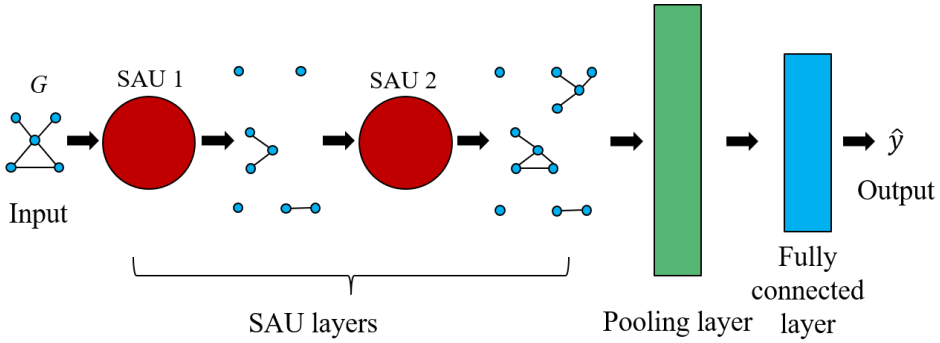

Figure 1: Overview: An example of SAN with two SAU layers

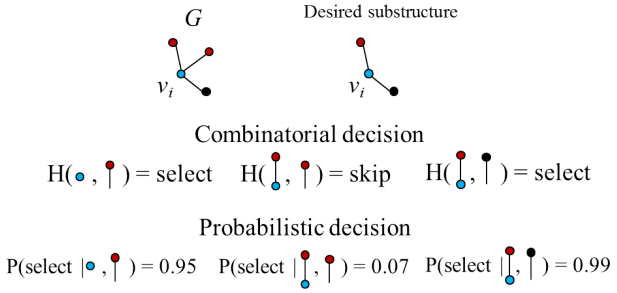

Figure 2: Substructure selection in combinatorial algorithms (hard decision) and substructure assembling networks (probabilistic/soft decision).

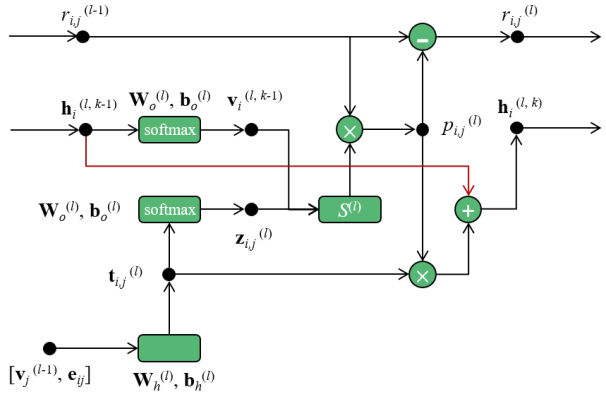

Figure 3: The graphical representation of SAU's computation for node i on its k-th neighbor node j. Note: The red line indicates it passes by other lines without intersection.

using a shared set of parameters. Without loss of generality, here we focus on how the l-th SAU layer operates on the neighborhood of node-i step by step.

Initialization. Given the input to the l-th SAU layer, G^{l-1}, we consider the substructure representation centered at node i, $\mathbf{v}_i^{(l-1)} \in V^{(l-1)}$, which is projected into the hidden space and the output space by the l-th layer SAU:

$$\mathbf{h}_i^{(l,0)} = \mathbf{W}_h^{(l)} \cdot [\mathbf{v}_i^{(l-1)}, \mathbf{0}] + \mathbf{b}_h^{(l)}, \quad (1)$$

$$\mathbf{v}_i^{(l,0)} = softmax(\mathbf{W}_o^{(l)} \cdot \mathbf{h}_i^{(l,0)} + \mathbf{b}_o^{(l)}), \quad (2)$$

where $\mathbf{W}_h^{(l)}$, $\mathbf{b}_h^{(l)}$ are parameters that control the mapping from the input space[1] to the hidden space, and $\mathbf{W}_o^{(l)}$, $\mathbf{b}_o^{(l)}$ are parameters making projections from the hidden space to the output space. We do not employ popular activation functions (e.g., $tanh$) for the hidden space, as it is used for approximating substructure assembling via addition operations in vector space. Like word2vec (Mikolov et al. 2013), an unbounded vectorial space is needed. The output space uses $softmax$ to generate higher level features. Softmax models substructure feature distribution as well as normalizes [2] feature vectors. Compared with other candidates such as L1 and L2 normalization, softmax significantly speeds up learning progress, as it could learn sparse feature representations quickly and distill salient features more efficiently.

Progressive edge selection. After the center node i is initialized, the l-th layer SAU then starts processing the neighborhood sequence N_i. Recall that $N_i(k)$ represents the k-th node in the neighborhood sequence. We also use j to replace $N_i(k)$ when it gives more convenient sub-indexes. The sequential procedure is summarized in 5 steps as follows. The corresponding computation graph is depicted in Figure 3.

- **Step 1**: For $N_i(k) = j$, we project its representation, namely $\mathbf{v}_j^{(l-1)}$ with $\mathbf{e}_{i,j}$, into the hidden space and then to the output space as,

$$\mathbf{t}_{i,j}^{(l)} = \mathbf{W}_h^{(l)} \cdot [\mathbf{v}_j^{(l-1)}, e_{i,j}] + \mathbf{b}_h^{(l)} \quad (3)$$

$$\mathbf{z}_{i,j}^{(l)} = softmax(\mathbf{W}_o^{(l)} \cdot \mathbf{t}_{i,j}^{(l)} + \mathbf{b}_o^{(l)}) \quad (4)$$

[1] The input feature space is a concatenation of $\mathbf{v}_i^{(l)}$ and the concerned edge feature vector, e.g. $[(\mathbf{v}_i^{(l)})^T \ \mathbf{e}_{i,j}^T]^T$; when starting from the center node where no edge is considered yet, the edge part is simply padded with zeros as in Equation (1)

[2] Normalization is important here. As the size of a node's neighborhood could be arbitrary, as SAU progresses, the values of hidden states could be at very different scales for individual nodes.

- **Step 2**: Project the hidden vector $\mathbf{h}_i^{(l,k-1)}$ representing the substructure assembled after considering the $(k-1)$-th neighbor $N_i(k-1)$ to the output feature space,

$$\mathbf{v}_i^{(l,k-1)} = softmax(\mathbf{W}_o^{(l)} \cdot \mathbf{h}_i^{(l,k-1)} + \mathbf{b}_o^{(l)}) \quad (5)$$

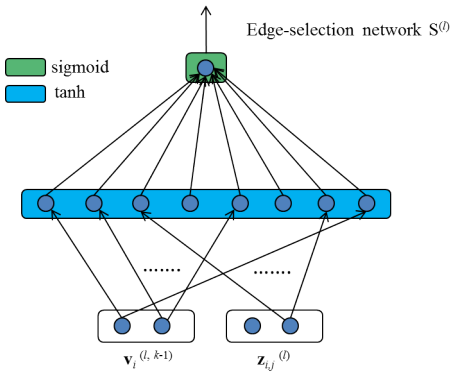

Figure 4: A two-layer edge-selection network models distribution of edge selection decisions, where $tanh$ is used in the hidden layer, and $sigmoid$ is applied to the output layer.

- **Step 3**: Given $\mathbf{v}_i^{(l,k-1)}$ (substructure assembled so far) and $\mathbf{z}_{i,j}^{(l)}$ (substructure at node j), compute the probability that the substructure at node j should be incorporated,

$$p_{i,j}^{(l)} = r_{i,j}^{(l-1)} \cdot S^{(l)}(\mathbf{v}_i^{(l,k-1)}, \mathbf{z}_{i,j}^{(l)}). \quad (6)$$

  Here $S^{(l)}(\cdot)$ is an edge-selection module computing the probability that edge e_{ij} should be selected in the l-th SAU layer. It is a multi-layer network that takes $\mathbf{v}_i^{(l,k-1)}$ and $\mathbf{z}_{i,j}^{(l)}$ as input, and outputs a numerical value in the range of $[0, 1]$. The reason that we use $\mathbf{v}_i^{(l,k-1)}$ and $\mathbf{z}_{i,j}^{(l)}$ rather than $\mathbf{h}_i^{(l,k-1)}$ and $\mathbf{t}_{i,j}^{(l)}$ as input is that, the former pair provides a normalized higher level view of the substructures. The latter pair could be in very different scales, making it difficult for the edge-selection network to learn selection decision distributions. Figure 4 shows a two-layer implementation, where $tanh$ is used in the hidden layer and $sigmoid$ in the output layer.

  In Equation (6), the edge selection probability $p_{i,j}^{(l)}$ is re-scaled by $r_{i,j}^{(l-1)}$, the "edge residue" for $\mathbf{e}_{i,j}$. $r_{i,j}^{(l-1)}$ is equal to one minus the cumulative probability of edge $e_{i,j}$ being selected in the previous $l-1$ SAU layers, and is used to inform the l-th layer on how much consideration can be given to this edge. Edge residues provide constraints to selection probabilities, so as to avoid duplicate edge selection across different SAU layers in a soft manner. The edge residues are initialized to all 1's and updated in each SAU layer based on the edge-selection decisions in that layer, according to Equation (8) in Step 5.

- **Step 4**: Assemble substructure from node j by

$$\mathbf{h}_i^{(l,k)} = \mathbf{h}_i^{(l,k-1)} + p_{i,j}^{(l)} \cdot \mathbf{t}_{i,j}^{(l)} \quad (7)$$

  where we apply linear operations in vector space to approximate substructure assembling in graph space.

- **Step 5**: Update edge residues by

$$r_{i,j}^{(l)} = r_{i,j}^{(l-1)} - p_{i,j}^{(l)} \quad (8)$$

Finally, $\mathbf{v}^{(l)} = \mathbf{v}^{(l,|N_i|)}$ is the output for node-i after the progressive edge selection in the l-th SAU, which then serves as the input to the subsequent layer.

Pooling layer

The pooling layer provides a uniform view for graphs of arbitrary sizes. Suppose that SAN employs L SAUs. The pooling layer takes $V^{(L)}$ as its input, and aggregates substructure features from individual nodes into a fixed-length feature vector for an input graph as follows.

$$\mathbf{z}_p = \sum_{\mathbf{v}_i^{(L)} \in V^{(L)}} \mathbf{v}_i^{(L)} \quad (9)$$

$$\mathbf{g} = softsign(\mathbf{W}_p \cdot \mathbf{z}_p + \mathbf{b}_p) \quad (10)$$

where $\mathbf{W}_p$ and $\mathbf{b}_p$ are model parameters.

In Equation (9), we use sum for aggregation. While max aggregation could be an alternative which serves as feature indicators, sum is preferred in SAN as it performs as both feature indicators and feature strength counters. Another possibility is to use weighted average for aggregation. In our exploration, we find it brings little improvement compared with sum, as the parameters in SAU layers can automatically tune the strength in individual feature channels.

In Equation (10), we make another projection to standardize feature representations. Similar to $tanh$, the output of $softsign$ in each dimension is in $(-1, 1)$. $softsign$ is preferred, as it does not saturate as easily as $tanh$.

Fully connected layer

Given the output $\mathbf{g}$ from the pooling layer, SAN employs a fully connected layer to predict class label:

$$\hat{\mathbf{y}} = softmax(\mathbf{W}_f \cdot \mathbf{g} + \mathbf{b}_f) \quad (11)$$

where $\mathbf{W}_f, \mathbf{b}_f$ are model parameters, $\hat{\mathbf{y}}$ is a vector for class prediction, and the number of dimensions for $\hat{\mathbf{y}}$ equals to the number of classes in a classification task.

SAN Training

In this section, we highlight key points in SAN training.

Meta parameters. The dimensionalities of hidden space and output feature space are two meta-parameters for each SAU. Intuitively, the higher the dimensionality, the more diverse substructure features can be taken into account for subsequent classification tasks, at the cost of higher computational complexity and risk of overfitting.

Loss function. We employ cross entropy to evaluate the error made by a parameter setting in SAN. Suppose that an input graph G is labeled as $\mathbf{y}$ and $\hat{\mathbf{y}}$ is the label prediction generated by SAN. In this case, the loss is quantified as

$$L = -\mathbf{y} \cdot \log \hat{\mathbf{y}} \quad (12)$$

With this loss function, we further leverage gradient descent based back-propagation to iteratively optimize parameters.

Neighborhood shuffling. When processing the neighbors of a node, the order of its neighborhood sequence can have a direct impact on the learning result. One possible solution is to fix the order by converting l-hop structural characteristics of each neighbor into a ranking score, when applying the $(l+1)$-th SAU. However, we find that this strategy can easily lead to model overfitting, especially when the training data is limited. Therefore, we choose to teach the model to handle arbitrary neighborhood orders by randomly shuffling neighborhood sequences for each training epoch. This can effectively generalize SAN for neighborhood of different permutations, and also overcome the small dataset issue for training deep neural networks.

Experimental Results

In this section, we present an experimental study on classification accuracy and interpretability of SAN.

Dataset

In the evaluation, we focus on public benchmark datasets for graph classification methods.

- MUTAG (Debnath et al. 1991) includes a set of graphs, each of which represents nitro compounds, and their labels indicate if they have mutagenic effect on bacteria;
- NCI1 and NCI109 (Wale, Watson, and Karypis 2008) are graph representations of chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines, respectively;
- ENZYMES (Borgwardt et al. 2005) contains graph representations of tertiary structure of 6 classes of enzymes;
- D&D (Dobson and Doig 2003) includes structures of enzymes and non-enzymes proteins, where nodes are amino acids, and edges indicate spatial closeness between nodes.

More details of the datasets are shown in Table 1.

	MUTAG	NCI1	NCI109	ENZYMES	D&D
# graphs	188	4110	4127	600	1178
Avg. # nodes	17.93	29.87	29.68	32.63	284.32
# node attributes	7	37	38	3	82
Avg. # edges	19.79	32.3	32.13	62.14	715.66
# edge attributes	11	3	3	–	–
# classes	2	2	2	6	2

Table 1: Statistics of the benchmark datasets. In these datasets, node and edge attributes are categorical.

Baseline methods

In this study, we compare SAN with state-of-the-art deep learning methods and graph kernels.

- **Graph kernels**. The following representative techniques are considered: (1) deep graph kernel (DG) (Yanardag and Vishwanathan 2015), (2) Weisfeiler-Lehman graph kernel (WL) (Shervashidze et al. 2011), (3) graphlet count kernel (GLC) (Shervashidze et al. 2009), (4) shortest-path kernel (SP) (Borgwardt and Kriegel 2005), and (5) random-walk kernel (RW) (Gärtner, Flach, and Wrobel 2003).
- **Deep graph learning**. We include the following methods: (1) edge-conditioned convolutional network (ECC) (Simonovsky and Komodakis 2017), (2) structure2vector (S2V) (Dai, Dai, and Song 2016), (3) PSCN (Niepert, Ahmed, and Kutzkov 2016), and (4) diffusion-convolutional neural networks (DCNN) (Atwood and Towsley 2016).

In addition, to compare SAU with other widely adopted recurrent unit alternatives, such as Gated Recurrent Unit (GRU) (Chung et al. 2014) and Long-Short Term Memory (LSTM) (Hochreiter and Schmidhuber 1997), we implement two baselines under the SAN framework: LSTM-SAN and GRU-SAN. In LSTM-SAN (GRU-SAN), we replace SAU with LSTM (GRU): (1) Hidden state update operations in LSTM (GRU) are employed to model substructure feature assembling; (2) After substructure assembling, the last hidden state is projected to an output space which serves as the input for a subsequent layer, as SAU does in Equation (5).

SAN configuration

We first briefly introduce the symbols used to present network structures for SAN instances.

- $\text{sau}(x, y, z)$ indicates an SAU layer with x dimensions in its hidden space, y dimensions in its output space, and z units for the hidden layer of its edge-selection network. In the study, we observe two-layer edge-selection networks are able to provide satisfying performance, while additional layers cannot provide any significant improvement.
- $\text{p}(x)$ stands for a pooling layer that first performs aggregation by Equation (9) and then projects the aggregated vectors into an x-dimensional output space by Equation (10);
- $\text{fc}(x)$ denotes a fully connected layer with an x-dimensional output space. Note that in SAN, x also denotes the number of classes in a classification task;
- $\text{d}(p)$ stands for a dropout layer with keep-probability p in training phases.

For each dataset, the specific SAN structure is varied by its input complexity, such as the number of nodes (edges) and the number of node (edge) attributes per graph. By cross-validation, we take the following SAN configurations in the evaluation.

- MUTAG. An SAN of $\text{sau}(64, 32, 64) \rightarrow \text{sau}(64, 32, 64) \rightarrow \text{p}(64) \rightarrow \text{d}(0.5) \rightarrow \text{fc}(2)$ is taken for this dataset. Using two layers of SAUs, we are able to extract high-quality features and achieve decent accuracy.
- NCI1. We adopt $\text{sau}(64, 32, 64) \rightarrow \text{sau}(64, 32, 64) \rightarrow \text{sau}(64, 32, 64) \rightarrow \text{p}(64) \rightarrow \text{d}(0.5) \rightarrow \text{fc}(2)$ for this dataset. With the third SAU, more complex substructure features are learned, bringing better classification performance. Meanwhile, we observe SAUs of larger hidden and output space (e.g., $\text{sau}(128, 64, 128)$) could not lead to significant improvement.

- NCI109. We also use $\text{sau}(64, 32, 64) \rightarrow \text{sau}(64, 32, 64) \rightarrow \text{sau}(64, 32, 64) \rightarrow \text{p}(64) \rightarrow \text{d}(0.5) \rightarrow \text{fc}(2)$, as NCI1 and NCI109 share similar input complexity.
- ENZYMES. $\text{sau}(64, 32, 64) \rightarrow \text{sau}(64, 32, 64) \rightarrow \text{sau}(64, 32, 64) \rightarrow \text{p}(64) \rightarrow \text{d}(0.5) \rightarrow \text{fc}(6)$ is employed for this dataset.
- D&D. We configure an SAN of $\text{sau}(64, 32, 64) \rightarrow \text{sau}(64, 32, 64) \rightarrow \text{p}(64) \rightarrow \text{d}(0.5) \rightarrow \text{fc}(2)$ for this dataset. In the exploration, we observe little improvement is brought by more complex SAUs or additional SAUs.

For LSTM-SAN and GRU-SAN, we replace SAU with LSTM and GRU, respectively, using same meta parameters for fair comparison.

We implement SAN and its variants in Tensorflow (Abadi et al. 2016), and use Adam optimizer with its default setting (Kingma and Ba 2015). Each SAU layer is pre-trained for 2,000 epochs, followed by end-to-end fine-tuning on the whole SAN for 10,000 epochs.

Classification accuracy evaluation

Following (Shervashidze et al. 2011), we perform 10-fold cross validation and present the average accuracy.

As shown in Table 2, SAN achieves the best accuracy on multiple datasets and delivers overall competitive classification performance.

- Compared with SAN variants LSTM-SAN and GRU-SAN, SAN outperforms them over all datasets, with up to 16% improvement. Indeed, LSTM and GRU have achieved great success in applications such as NLP and speech recognition; however, they may not be good fit in the case of substructure assembling: (a) Unlike the aforementioned applications, an edge selection decision in substructure assembling makes permanent impact to the later decisions, so the gating mechanisms, such as forget gate in LSTM, could be counter intuitive in the scenario of substructure feature learning; (b) While more gates bring unnecessary parameters, it becomes more difficult to train a network with increased overfitting risk. This result also confirms our intuition behind the design of SAU.
- Compared with state-of-the-art deep learning methods for graph classification, SAN also achieves overall the best performance. Unlike PSCN, SAN performs on raw graph data without a complex node-ordering step, and outperforms it in all cases. Compared with ECC, SAN performs significantly better on MUTAG, ENZYMES, and D&D, and delivers similar performance on NCI1 and NCI109. This suggests the progressive substructure assembling in SAU indeed brings additional discriminative features to classification tasks. In the case of S2V, SAN achieves better performance on MUTAG, NCI1, and ENZYMES with comparable performance on NCI109 and D&D. Unlike S2V that aims to learn graph embedding, SAN explicitly learns substructure assembling process for better classification performance, where the edge residue mechanism in SAU makes it easier to interpret the learned features.
- SAN also demonstrates competitive classification performance in comparison with the state-of-the-art kernel methods. In particular, SAN outperforms WL in the case of MUTAG, ENZYMES, and D&D, with comparable performance on NCI1 and NCI109.

Interpretable substructure features in SAN

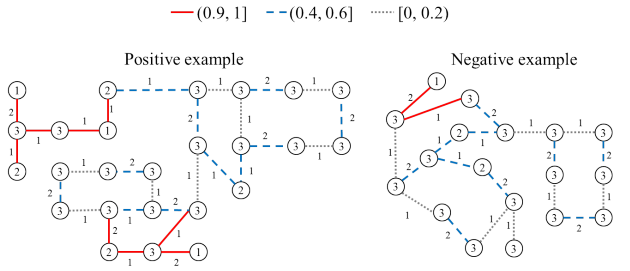

Figure 5: SAN discovers discriminative features for positive and negative graphs in NCI1, where integers are node/edge attributes from the benchmark dataset. Colors on edges indicate how much they contribute for a classification task, where red solid ones contribute the most and grey dotted ones contribute the least.

We perform a case study on two NCI1 graphs (one positive and one negative) to show the interpretability of learned features. By the edge residues generated from the last SAU, we are able to derive contribution weights of edges for the underlying classification task, where a contribution weight is a numerical value in the range of $[0, 1]$ and a larger contribution weight means an edge contributes more. In the example shown in Figure 5, contribution weights fall into three bins: (1) edges of weights in the range of $(0.9, 1]$ are red solid lines; (2) edges of weights in the range of $(0.4, 0.6]$ are blue broken lines; and (3) edges of weights in the range of $(0, 0.2]$ are grey dotted lines. In this case, substructures of red solid edges serve as the most discriminative substructure features, and provide the strongest signals for its classification. Unfortunately, as node/edge attributes in the benchmark dataset are digitized into integers, we are unable to recover their original attributes in Figure 5.

Related Work

Existing techniques for graph classification can be grouped into two categories: kernel methods and deep learning.

Kernel methods. Graph kernels compute similarity between graphs, and then we can leverage kernel-based supervised techniques such as SVM (Cortes and Vapnik 1995) to learn boundaries between different classes. Graph kernels vary based on graph features used for measuring similarity, ranging from random walks (Gärtner, Flach, and Wrobel 2003), shortest paths (Borgwardt and Kriegel 2005), to graph substructures (Shervashidze et al. 2009; Yanardag and Vishwanathan 2015). Although Graph kernels have achieved great success, they suffer two major limitations: (1) they usually involve high computation complexity with scalability concern; and (2) similarity matrix computation and classification learning are two independent steps, where graph

		MUTAG	NCI1	NCI109	ENZYMES	D&D
Deep Learning	SAN	**94.30**	83.84	81.78	**63.17**	81.37
	LSTM-SAN	80.21	76.43	75.26	51.18	74.67
	GRU-SAN	81.33	74.32	75.33	53.33	72.00
	PSCN	92.63	78.59	–	–	77.12
	S2V	88.28	83.72	82.16	61.10	**82.22**
	ECC	89.44	83.80	82.14	53.50	74.10
	DCNN	66.98	62.61	62.86	18.10	–
Graph Kernel	DG	87.44	80.31	80.32	53.43	–
	WL	83.78	**84.55**	**84.49**	59.05	79.78
	GLC	75.61	66.00	66.59	32.70	78.59
	RW	80.72	64.34	63.51	21.68	71.70
	SP	87.28	73.47	73.07	41.68	78.45

Table 2: Classification accuracy comparison between SAN and baseline methods: (1) Average accuracy after 10-fold cross validation is reported; (2) we only include the results from the variant of the best performance for each baseline method.

features are fixed beforehand and feature selection cannot be guided by followup classification tasks.

Deep learning. Deep learning suggests the potential to enable joint training that guides both feature and classification learning. However, as input graphs are allowed to be of arbitrary size and permutation invariant, it is difficult to directly apply off-the-shelf deep learning frameworks, such as CNN, to graph classification problems.

A few studies attempt to address this challenge by graph spectral methods (Defferrard, Bresson, and Vandergheynst 2016; Bruna et al. 2014): Although it is hard to align nodes across graphs, it is still possible to align channels in graph spectral domain. While such deep learning methods are effective to filter information channels in spectral domain and learn global graph features, it is difficult to perform local substructure feature learning, which limits their performance in classification tasks.

To further improve classification accuracy, recent works start investigating how to design network structures that enable automated substructure feature learning. Multiple studies (Simonovsky and Komodakis 2017; Li et al. 2016; Duvenaud et al. 2015) assume feature selection processes behave identically for neighbor nodes with the same node/edge attributes and propose approaches that filter neighbor information according to neighbor nodes' node/edge attributes or filter neighbor information in a uniform way, which may not necessarily hold in practice. Meanwhile, it is also difficult for these methods to distinguish neighbor selection in the cases where edge labels are missing or all edges share an identical label. Niepert et al. (Niepert, Ahmed, and Kutzkov 2016) develop a framework that first aligns nodes across input graphs and then employs CNN for training and classification. The performance of this technique highly depends on node alignment results in the first step. Unfortunately, node alignment problem is NP-hard and difficult to guarantee effective node alignment, which potentially undermines its performance for different datasets. Unlike these works, our technique has no need for a node alignment step and utilizes RNN-like network structures to assemble substructure features by a sequence of edge selection decisions, where each decision is made by jointly considering edge label and neighbor state information. Moreover, with a hierarchical network structure, our technique is able to deliver high accuracy with interpretable graph substructure features.

In addition, deep learning methods are also adopted in node embedding learning. Given a set of nodes in a graph, node embedding learning aims to unveal hidden representations for nodes in either a supervised (Moore and Neville 2017; Kipf and Welling 2017) or an unsupervised (Cao, Lu, and Xu 2016; Tang et al. 2015) manner so that their proximity is preserved. The discovered node representations can serve as an important source of features for tasks such as node classification (Perozzi, Al-Rfou, and Skiena 2014; Chang et al. 2015; Grover and Leskovec 2016), link prediction (Li et al. 2014; Wang, Cui, and Zhu 2016), and community detection (Tian et al. 2014; Ribeiro, Saverese, and Figueiredo 2017). Compared with these works, our work pursues a different goal: Given a given set of graphs, we aim to learn proximity among graphs.

Conclusion

In this work, we propose the Substructure Assembling Network to extract interpretable, discriminative structural features for the challenging task of graph classification, which has a wide variety of real-world applications. Our approach inherits the advantage of both combinatorial pattern search and deep learning. Using public benchmark datasets, we demonstrate the great potential of SAN in extracting discriminative and interpretable graph features to boost classification performance for graphs.

Acknowledgments

This research was partially supported by the National Natural Science Foundation of China (Grant Nos. 61522206, 61373118, 61672409), the Major Basic Research Project of Shaanxi Province (Grant No. 2017ZDJC-31), and the Science and Technology Plan Program in Shaanxi Province of China (Grant No. 2017KJXX-80).

References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; et al. 2016. Tensorflow: A system for large-scale machine learning. In *OSDI*.

Atwood, J., and Towsley, D. 2016. Diffusion-convolutional neural networks. In *NIPS*.

Borgwardt, K. M., and Kriegel, H.-P. 2005. Shortest-path kernels on graphs. In *ICDM*.

Borgwardt, K. M.; Ong, C. S.; Schönauer, S.; Vishwanathan, S.; Smola, A. J.; and Kriegel, H.-P. 2005. Protein function prediction via graph kernels. *Bioinformatics*.

Bruna, J.; Zaremba, W.; Szlam, A.; and LeCun, Y. 2014. Spectral networks and locally connected networks on graphs. In *ICLR*.

Burbidge, R.; Trotter, M.; and Holden, S. 2000. Drug design by machine learning: support vector machines for pharmaceutical data analysis. *Computers and Chemistry*.

Cao, S.; Lu, W.; and Xu, Q. 2016. Deep neural networks for learning graph representations. In *AAAI*, 1145–1152.

Chang, S.; Han, W.; Tang, J.; Qi, G.-J.; Aggarwal, C. C.; and Huang, T. S. 2015. Heterogeneous network embedding via deep architectures. In *KDD*, 119–128.

Cheng, H.; Lo, D.; Zhou, Y.; Wang, X.; and Yan, X. 2009. Identifying bug signatures using discriminative graph mining. In *ISSTA*.

Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Cortes, C., and Vapnik, V. 1995. Support-vector networks. *Machine learning* 20(3):273–297.

Cumming, J. G.; Davis, A. M.; Muresan, S.; Haeberlein, M.; and Chen, H. 2013. Chemical predictive modelling to improve compound quality. *Nature reviews Drug discovery* 12(12):948–962.

Dai, H.; Dai, B.; and Song, L. 2016. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning*, 2702–2711.

Debnath, A. K.; Lopez de Compadre, R. L.; Debnath, G.; Shusterman, A. J.; and Hansch, C. 1991. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*.

Dobson, P. D., and Doig, A. J. 2003. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology* 330(4):771–783.

Duvenaud, D. K.; Maclaurin, D.; Iparraguirre, J.; Bombarell, R.; Hirzel, T.; Aspuru-Guzik, A.; and Adams, R. P. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*.

Gärtner, T.; Flach, P.; and Wrobel, S. 2003. On graph kernels: Hardness results and efficient alternatives. *Learning Theory and Kernel Machines* 129–143.

Grover, A., and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *KDD*, 855–864.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation*.

Kingma, D., and Ba, J. 2015. Adam: A method for stochastic optimization. In *ICLR*.

Kipf, T. N., and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

Li, X.; Du, N.; Li, H.; Li, K.; Gao, J.; and Zhang, A. 2014. A deep learning approach to link prediction in dynamic networks. In *SDM*, 289–297.

Li, Y.; Tarlow, D.; Brockschmidt, M.; and Zemel, R. 2016. Gated graph sequence neural networks. In *ICLR*.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*, 3111–3119.

Moore, J., and Neville, J. 2017. Deep collective inference. In *AAAI*, 2364–2372.

Neumann, M.; Garnett, R.; Bauckhage, C.; and Kersting, K. 2016. Propagation kernels: efficient graph kernels from propagated information. *Machine Learning*.

Niepert, M.; Ahmed, M.; and Kutzkov, K. 2016. Learning convolutional neural networks for graphs. In *ICML*.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *KDD*, 701–710.

Ribeiro, L. F.; Saverese, P. H.; and Figueiredo, D. R. 2017. struc2vec: Learning node representations from structural identity. In *KDD*, 385–394.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2009. The graph neural network model. *IEEE Transactions on Neural Networks*.

Shervashidze, N.; Vishwanathan, S.; Petri, T.; Mehlhorn, K.; and Borgwardt, K. 2009. Efficient graphlet kernels for large graph comparison. In *AISTATS*.

Shervashidze, N.; Schweitzer, P.; Leeuwen, E. J. v.; Mehlhorn, K.; and Borgwardt, K. M. 2011. Weisfeiler-lehman graph kernels. *JMLR*.

Simonovsky, M., and Komodakis, N. 2017. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *CVPR*.

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. Line: Large-scale information network embedding. In *WWW*, 1067–1077.

Tian, F.; Gao, B.; Cui, Q.; Chen, E.; and Liu, T.-Y. 2014. Learning deep representations for graph clustering. In *AAAI*.

Wale, N.; Watson, I. A.; and Karypis, G. 2008. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*.

Wang, D.; Cui, P.; and Zhu, W. 2016. Structural deep network embedding. In *KDD*, 1225–1234.

Yan, X., and Han, J. 2002. gspan: Graph-based substructure pattern mining. In *ICDM*.

Yanardag, P., and Vishwanathan, S. 2015. Deep graph kernels. In *KDD*, 1365–1374.