Learning Convolutional Neural Networks for Graphs

Mathias Niepert MATHIAS.NIEPERT@NECLAB.EU
Mohamed Ahmed MOHAMED.AHMED@NECLAB.EU
Konstantin Kutzkov KONSTANTIN.KUTZKOV@NECLAB.EU
NEC Labs Europe, Heidelberg, Germany

Abstract

Numerous important problems can be framed as learning from graph data. We propose a framework for learning convolutional neural networks for arbitrary graphs. These graphs may be undirected, directed, and with both discrete and continuous node and edge attributes. Analogous to image-based convolutional networks that operate on locally connected regions of the input, we present a general approach to extracting locally connected regions from graphs. Using established benchmark data sets, we demonstrate that the learned feature representations are competitive with state of the art graph kernels and that their computation is highly efficient.

1. Introduction

With this paper we aim to bring convolutional neural networks to bear on a large class of graph-based learning problems. We consider the following two problems.

1. Given a collection of graphs, learn a function that can be used for classification and regression problems on unseen graphs. The nodes of any two graphs are *not* necessarily in correspondence. For instance, each graph of the collection could model a chemical compound and the output could be a function mapping unseen compounds to their level of activity against cancer cells.

2. Given a large graph, learn graph representations that can be used to infer unseen graph properties such as node types and missing edges.

We propose a framework for learning representations for classes of directed and undirected graphs. The graphs may have nodes and edges with multiple discrete and continuous attributes and may have multiple types of edges. Similar to convolutional neural network for images, we construct locally connected neighborhoods from the input graphs. These neighborhoods are generated efficiently and serve as the receptive fields of a convolutional architecture, allowing the framework to learn effective graph representations.

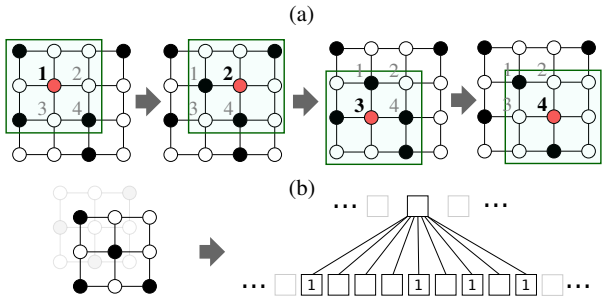

Figure 1. A CNN with a receptive field of size 3x3. The field is moved over an image from left to right and top to bottom using a particular stride (here: 1) and zero-padding (here: none) (a). The values read by the receptive fields are transformed into a linear layer and fed to a convolutional architecture (b). The node sequence for which the receptive fields are created and the shapes of the receptive fields are fully determined by the hyper-parameters.

The proposed approach builds on concepts from convolutional neural networks (CNNs) (Fukushima, 1980; Atlas et al., 1988; LeCun et al., 1998; 2015) for images and extends them to arbitrary graphs. Figure 1 illustrates the locally connected receptive fields of a CNN for images. An image can be represented as a square grid graph whose nodes represent pixels. Now, a CNN can be seen as traversing a node sequence (nodes 1-4 in Figure 1(a)) and generating fixed-size neighborhood graphs (the 3x3 grids in Figure 1(b)) for each of the nodes. The neighborhood graphs serve as the receptive fields to read feature values from the pixel nodes. Due to the implicit spatial order of the pixels, the sequence of nodes for which neighborhood graphs are created, from left to right and top to bottom, is uniquely determined. The same holds for NLP problems where each sentence (and its parse-tree) determines

Proceedings of the 33^{rd} International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48.

a sequence of words. However, for numerous graph collections a problem-specific ordering (spatial, temporal, or otherwise) is missing and the nodes of the graphs are not in correspondence. In these instances, one has to solve two problems: (i) Determining the node sequences for which neighborhood graphs are created and (ii) computing a normalization of neighborhood graphs, that is, a unique mapping from a graph representation into a vector space representation. The proposed approach, termed PATCHY-SAN, addresses these two problems for arbitrary graphs. For each input graph, it first determines nodes (and their order) for which neighborhood graphs are created. For each of these nodes, a neighborhood consisting of exactly k nodes is extracted and normalized, that is, it is uniquely mapped to a space with a fixed linear order. The normalized neighborhood serves as the receptive field for a node under consideration. Finally, feature learning components such as convolutional and dense layers are combined with the normalized neighborhood graphs as the CNN's receptive fields.

Figure 2 illustrates the PATCHY-SAN architecture which has several advantages over existing approaches: First, it is highly efficient, naively parallelizable, and applicable to large graphs. Second, for a number of applications, ranging from computational biology to social network analysis, it is important to visualize learned network motifs (Milo et al., 2002). PATCHY-SAN supports feature visualizations providing insights into the structural properties of graphs. Third, instead of crafting yet another graph kernel, PATCHY-SAN learns application dependent features without the need to feature engineering. Our theoretical contributions are the definition of the normalization problem on graphs and its complexity; a method for comparing graph labeling approaches for a collection of graphs; and a result that shows that PATCHY-SAN generalizes CNNs on images. Using standard benchmark data sets, we demonstrate that the learned CNNs for graphs are both efficient and effective compared to state of the art graph kernels.

2. Related Work

Graph kernels allow kernel-based learning approaches such as SVMs to work directly on graphs (Vishwanathan et al., 2010). Kernels on graphs were originally defined as similarity functions on the nodes of a single graph (Kondor & Lafferty, 2002). Two representative classes of kernels are the skew spectrum kernel (Kondor & Borgwardt, 2008) and kernels based on graphlets (Kondor et al., 2009; Shervashidze et al., 2009). The latter is related to our work, as it builds kernels based on fixed-sized subgraphs. These subgraphs, which are often called motifs or graphlets, reflect functional network properties (Milo et al., 2002; Alon, 2007). However, due to the combinatorial complexity of subgraph enumeration, graphlet kernels are restricted to

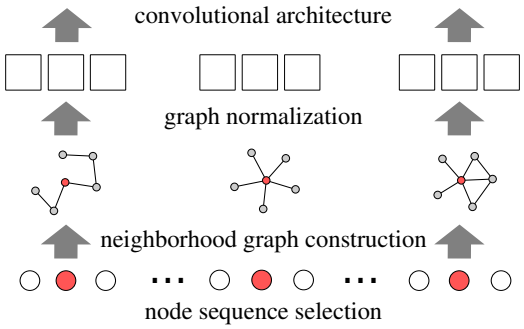

Figure 2. An illustration of the proposed architecture. A node sequence is selected from a graph via a graph labeling procedure. For some nodes in the sequence, a local neighborhood graph is assembled and normalized. The normalized neighborhoods are used as receptive fields and combined with existing CNN components.

subgraphs with few nodes. An effective class of graph kernels are the Weisfeiler-Lehman (WL) kernels (Shervashidze et al., 2011). WL kernels, however, only support discrete features and use memory linear in the number of training examples at test time. PATCHY-SAN uses WL as one possible labeling procedure to compute receptive fields. Deep graph kernels (Yanardag & Vishwanathan, 2015) and graph invariant kernels (Orsini et al., 2015) compare graphs based on the existence or count of small substructures such as shortest paths (Borgwardt & Kriegel, 2005), graphlets, subtrees, and other graph invariants (Haussler, 1999; Orsini et al., 2015). In contrast, PATCHY-SAN learns substructures from graph data and is not limited to a predefined set of motifs. Moreover, while all graph kernels have a training complexity at least *quadratic* in the number of graphs (Shervashidze et al., 2011), which is prohibitive for large-scale problems, PATCHY-SAN scales *linearly* with the number of graphs.

Graph neural networks (GNNs) (Scarselli et al., 2009) are a recurrent neural network architecture defined on graphs. GNNs apply recurrent neural networks for walks on the graph structure, propagating node representations until a fixed point is reached. The resulting node representations are then used as features in classification and regression problems. GNNs support only discrete labels and perform as many backpropagation operations as there are edges and nodes in the graph *per learning iteration*. Gated Graph Sequence Neural Networks modify GNNs to use gated recurrent units and to output sequences (Li et al., 2015).

Recent work extended CNNs to topologies that differ from the low-dimensional grid structure (Bruna et al., 2014; Henaff et al., 2015). All of these methods, however, assume one global graph structure, that is, a correspondence of the vertices across input examples. (Duvenaud et al., 2015) perform convolutional type operations on graphs, developing a differentiable variant of one specific graph feature.

3. Background

We provide a brief introduction to the required background in convolutional networks and graph theory.

3.1. Convolutional Neural Networks

CNNs were inspired by earlier work that showed that the visual cortex in animals contains complex arrangements of cells, responsible for detecting light in small local regions of the visual field (Hubel & Wiesel, 1968). CNNs were developed in the 1980s and have been applied to image, speech, text, and drug discovery problems (Atlas et al., 1988; LeCun et al., 1989; 1998; 2015; Wallach et al., 2015). A predecessor to CNNs was the Neocognitron (Fukushima, 1980). A typical CNN is composed of convolutional and dense layers. The purpose of the first convolutional layer is the extraction of common patterns found within local regions of the input images. CNNs convolve learned filters over the input image, computing the inner product at every image location in the image and outputting the result as tensors whose depth is the number of filters.

3.2. Graphs

A graph G is a pair (V, E) with $V = \{v_1, ..., v_n\}$ the set of vertices and $E \subseteq V \times V$ the set of edges. Let n be the number of vertices and m the number of edges. Each graph can be represented by an adjacency matrix $\mathbf{A}$ of size $n \times n$, where $\mathbf{A}_{i,j} = 1$ if there is an edge from vertex v_i to vertex v_j, and $\mathbf{A}_{i,j} = 0$ otherwise. In this case, we say that vertex v_i has *position* i in $\mathbf{A}$. Moreover, if $\mathbf{A}_{i,j} = 1$ we say v_i and v_j are *adjacent*. Node and edge attributes are features that attain one value for each node and edge of a graph. We use the term attribute value instead of label to avoid confusion with the graph-theoretical concept of a labeling. A walk is a sequence of nodes in a graph, in which consecutive nodes are connected by an edge. A path is a walk with distinct nodes. We write $\mathbf{d}(u, v)$ to denote the distance between u and v, that is, the length of the shortest path between u and v. $N_1(v)$ is the 1-neighborhood of a node, that is, all nodes that are adjacent to v.

Labeling and Node Partitions. PATCHY-SAN utilizes graph labelings to impose an order on nodes. A graph labeling ℓ is a function $\ell : V \to S$ from the set of vertices V to an ordered set S such as the real numbers and integers. A graph labeling procedure computes a graph labeling for an input graph. When it is clear from the context, we use *labeling* to refer to both, the graph labeling and the procedure to compute it. A ranking (or coloring) is a function $\mathbf{r} : V \to \{1, ..., |V|\}$. Every labeling induces a ranking $\mathbf{r}$ with $\mathbf{r}(u) < \mathbf{r}(v)$ if and only if $\ell(u) > \ell(v)$. If the labeling ℓ of graph G is injective, it determines a total order of G's vertices and a unique adjacency matrix $\mathbf{A}^\ell(G)$ of G where vertex v has position $\mathbf{r}(v)$ in $\mathbf{A}^\ell(G)$. Moreover, every graph labeling induces a partition $\{V_1, ..., V_n\}$ on V with $u, v \in V_i$ if and only if $\ell(u) = \ell(v)$.

Examples of graph labeling procedures are node degree and other measures of centrality commonly used in the analysis of networks. For instance, the *betweeness centrality* of a vertex v computes the fractions of shortest paths that pass through v. The Weisfeiler-Lehman algorithm (Weisfeiler & Lehman, 1968) is a procedure for partitioning the vertices of a graph. It is also known as color refinement and naive vertex classification. Color refinement has attracted considerable interest in the ML community since it can be applied to speed-up inference in graphical models (Kersting et al., 2009; 2014) and as a method to compute graph kernels (Shervashidze et al., 2011). PATCHY-SAN applies these labeling procedures, among others (degree, page-rank, eigenvector centrality, etc.), to impose an order on the nodes of graphs, replacing application-dependent orders (temporal, spatial, etc.) where missing.

Isomorphism and Canonicalization. The computational problem of deciding whether two graphs are isomorphic surfaces in several application domains. The graph isomorphism (GI) problem is in NP but not known to be in P or NP-hard. Under several mild restrictions, GI is known to be in P. For instance, GI is in P for graphs of bounded degree (Luks, 1982). A canonicalization of a graph G is a graph G' with a fixed vertex order which is isomorphic to G and which represents its entire isomorphism class. In practice, the graph canonicalization tool NAUTY has shown remarkable performance (McKay & Piperno, 2014).

4. Learning CNNs for Arbitrary Graphs

When CNNs are applied to images, a receptive field (a square grid) is moved over each image with a particular step size. The receptive field reads the pixels' feature values, for each channel once, and a patch of values is created for each channel. Since the pixels of an image have an implicit arrangement – a spatial order – the receptive fields are always moved from left to right and top to bottom. Moreover, the spatial order uniquely determines the nodes of each receptive field and the way these nodes are mapped to a vector space representation (see Figure 1(b)). Consequently, the values read from two pixels using two different locations of the receptive field are assigned to the same relative position if and only if the pixels' structural roles (their spatial position within the receptive field) are identical.

To show the connection between CNNs and PATCHY-SAN, we frame CNNs on images as identifying a sequence of nodes in the square grid graph representing the image and building a normalized neighborhood graph – a receptive

Algorithm 1 SELNODESEQ: Select Node Sequence

input: graph labeling procedure ℓ, graph $G = (V, E)$, stride s, width w, receptive field size k
$V_{\texttt{sort}}$ = top w elements of V according to ℓ
$i = 1, j = 1$
while $j < w$ **do**
  if $i \leq |V_{\texttt{sort}}|$ **then**
    $\mathsf{f} = \text{RECEPTIVEFIELD}(V_{\texttt{sort}}[i])$
  else
    $\mathsf{f} = \text{ZERORECEPTIVEFIELD}()$
  apply f to each input channel
  $i = i + s, j = j + 1$

Algorithm 2 NEIGHASSEMB: Neighborhood Assembly

input: vertex v, receptive field size k
output: set of neighborhood nodes N for v
$N = [v]$
$L = [v]$
while $|N| < k$ and $|L| > 0$ **do**
  $L = \bigcup_{v \in L} N_1(v)$
  $N = N \cup L$
return the set of vertices N

field – for each node in the identified sequence. For graph collections where an application-dependent node order is missing and where the nodes of any two graphs are not yet aligned, we need to determine for each graph (i) the sequences of nodes for which we create neighborhoods, and (ii) a unique mapping from the graph representation to a vector representation such that nodes with similar structural roles in the neighborhood graphs are positioned similarly in the vector representation.

We address these problems by leveraging graph labeling procedures that assigns nodes from two different graphs to a similar relative position in their respective adjacency matrices if their structural roles within the graphs are similar. Given a collection of graphs, PATCHY-SAN (SELECT-ASSEMBLE-NORMALIZE) applies the following steps to each graph: (1) Select a fixed-length sequence of nodes from the graph; (2) assemble a fixed-size neighborhood for each node in the selected sequence; (3) normalize the extracted neighborhood graph; and (4) learn neighborhood representations with convolutional neural networks from the resulting sequence of patches.

In the following, we describe methods that address the above-mentioned challenges.

4.1. Node Sequence Selection

Node sequence selection is the process of identifying, for each input graph, a sequence of nodes for which receptive fields are created. Algorithm 1 lists one such procedure. First, the vertices of the input graph are sorted with respect to a given graph labeling. Second, the resulting node sequence is traversed using a given stride s and for each visited node, Algorithm 3 is executed to construct a receptive field, until exactly w receptive fields have been created. The stride s determines the distance, relative to the selected node sequence, between two consecutive nodes for which a receptive field is created. If the number of nodes is smaller than w, the algorithm creates all-zero receptive fields for padding purposes.

Several alternative methods for vertex sequence selection are possible. For instance, a depth-first traversal of the input graph guided by the values of the graph labeling. We leave these ideas to future work.

4.2. Neighborhood Assembly

For each of the nodes identified in the previous step, a receptive field has to be constructed. Algorithm 3 first calls Algorithm 2 to assembles a local neighborhood for the input node. The nodes of the neighborhood are the candidates for the receptive field. Algorithm 2 lists the neighborhood assembly steps. Given as inputs a node v and the size of the receptive field k, the procedure performs a breadth-first search, exploring vertices with an increasing distance from v, and adds these vertices to a set N. If the number of collected nodes is smaller than k, the 1-neighborhood of the vertices most recently added to N are collected, and so on, until at least k vertices are in N, or until there are no more neighbors to add. Note that at this time, the size of N is possibly different to k.

4.3. Graph Normalization

The receptive field for a node is constructed by *normalizing* the neighborhood assembled in the previous step. Illustrated in Figure 3, the normalization imposes an order on the nodes of the neighborhood graph so as to map from the unordered graph space to a vector space with a linear order. The basic idea is to leverage graph labeling procedures that assigns nodes of two different graphs to a similar relative position in the respective adjacency matrices if and only if their structural roles within the graphs are similar.

To formalize this intuition, we define the optimal graph normalization problem which aims to find a labeling that is optimal relative to a given collection of graphs.

Problem 1 (Optimal graph normalization)**.** *Let $\mathcal{G}$ be a collection of unlabeled graphs with k nodes, let ℓ be an injective graph labeling procedure, let $\mathbf{d_G}$ be a distance measure on graphs with k nodes, and let $\mathbf{d_A}$ be a distance measure on $k \times k$ matrices. Find $\hat{\ell}$ such that*

$$\hat{\ell} = \arg\min_{\ell} \mathbb{E}_{\mathcal{G}} \left[\left| \mathbf{d_A} \left(\mathbf{A}^{\ell}(G), \mathbf{A}^{\ell}(G') \right) - \mathbf{d_G}(G, G') \right| \right].$$

The problem amounts to finding a graph labeling procedure ℓ, such that, for any two graphs drawn uniformly at

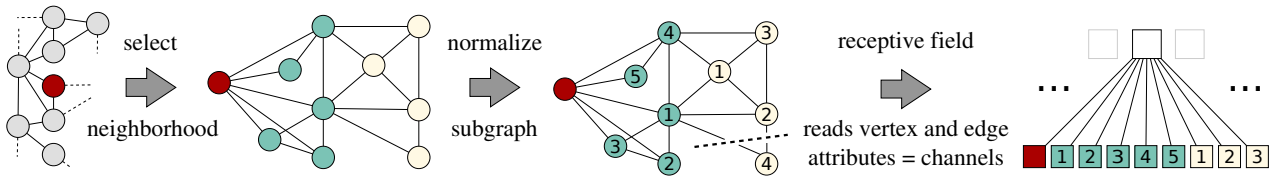

Figure 3. The normalization is performed for each of the graphs induced on the neighborhood of a root node v (the red node; node colors indicate distance to the root node). A graph labeling is used to rank the nodes and to create the normalized receptive fields, one of size k (here: $k = 9$) for node attributes and one of size $k \times k$ for edge attributes. Normalization also includes cropping of excess nodes and padding with dummy nodes. Each vertex (edge) attribute corresponds to an input channel with the respective receptive field.

Algorithm 3 RECEPTIVEFIELD: Create Receptive Field

input: vertex v, graph labeling ℓ, receptive field size k
$N = \text{NEIGHASSEMB}(v, k)$
$G_{\text{norm}} = \text{NORMALIZEGRAPH}(N, v, \ell, k)$
return G_{norm}

Algorithm 4 NORMALIZEGRAPH: Graph Normalization

input: subset of vertices U from original graph G, vertex v, graph labeling ℓ, receptive field size k
output: receptive field for v
compute ranking $\mathbf{r}$ of U using ℓ, subject to $\forall u, w \in U : \mathbf{d}(u, v) < \mathbf{d}(w, v) \Rightarrow \mathbf{r}(u) < \mathbf{r}(w)$
if $|U| > k$ **then**
  $N =$ top k vertices in U according to $\mathbf{r}$
  compute ranking $\mathbf{r}$ of N using ℓ, subject to $\forall u, w \in N : \mathbf{d}(u, v) < \mathbf{d}(w, v) \Rightarrow \mathbf{r}(u) < \mathbf{r}(w)$
else if $|V| < k$ **then**
  $N = U$ and $k - |U|$ dummy nodes
else
  $N = U$
construct the subgraph $G[N]$ for the vertices N
canonicalize $G[N]$, respecting the prior coloring $\mathbf{r}$
return $G[N]$

random from $\mathcal{G}$, the expected difference between the distance of the graphs in vector space (with respect to the adjacency matrices based on ℓ) and the distance of the graphs in graph space is minimized. The optimal graph normalization problem is a generalization of the classical graph canonicalization problem. A canonical labeling algorithm, however, is optimal only for isomorphic graphs and might perform poorly for graphs that are similar but not isomorphic. In contrast, the smaller the expectation of the optimal normalization problem, the better the labeling aligns nodes with similar structural roles. Note that the similarity is determined by $\mathbf{d_G}$.

We have the following result concerning the complexity of the optimal normalization problem.

Theorem 1. *Optimal graph normalization is NP-hard.*

Proof: By reduction from subgraph isomorphism. □

PATCHY-SAN does *not* solve the above optimization problem. Instead, it may compare different graph labeling methods and choose the one that performs best relative to a given collection of graphs.

Theorem 2. *Let $\mathcal{G}$ be a collection of graphs and let $(G_1, G'_1), ..., (G_N, G'_N)$ be a sequence of pairs of graphs sampled independently and uniformly at random from $\mathcal{G}$. Let $\hat{\theta}_\ell := \sum_{i=1}^{N} \mathbf{d_A}\left(\mathbf{A}^\ell(G_i), \mathbf{A}^\ell(G'_i)\right)/N$ and $\theta_\ell := \mathbb{E}_{\mathcal{G}}\left[\left|\mathbf{d_A}\left(\mathbf{A}^\ell(G), \mathbf{A}^\ell(G')\right) - \mathbf{d_G}(G, G')\right|\right]$. If $\mathbf{d_A} \geq \mathbf{d_G}$, then $\mathbb{E}_{\mathcal{G}}[\hat{\theta}_{\ell_1}] < \mathbb{E}_{\mathcal{G}}[\hat{\theta}_{\ell_2}]$ if and only if $\theta_{\ell_1} < \theta_{\ell_2}$.*

Theorem 2 enables us to compare different labeling procedures in an unsupervised manner via a comparison of the corresponding estimators. Under the assumption $\mathbf{d_A} \geq \mathbf{d_G}$, the smaller the estimate $\hat{\theta}_\ell$ the smaller the absolute difference. Therefore, we can simply choose the labeling ℓ for which $\hat{\theta}_\ell$ is minimal. The assumption $\mathbf{d_A} \geq \mathbf{d_G}$ holds, for instance, for the edit distance on graphs and the Hamming distance on adjacency matrices. Finally, note that all of the above results can be extended to directed graphs.

The graph normalization problem and the application of appropriate graph labeling procedures for the normalization of local graph structures is at the core of the proposed approach. Within the PATCHY-SAN framework, we normalize the neighborhood graphs of a vertex v. The labeling of the vertices is therefore constrained by the graph distance to v: for any two vertices u, w, if u is closer to v than w, then v is always ranked higher than w. This definition ensures that v has always rank 1, and that the closer a vertex is to v in G, the higher it is ranked in the vector space representation.

Since most labeling methods are not injective, it is necessary to break ties between same-label nodes. To do so, we use NAUTY (McKay & Piperno, 2014). NAUTY accepts prior node partitions as input and breaks remaining ties by choosing the lexicographically maximal adjacency matrix. It is known that graph isomorphism is in PTIME for graphs of bounded degree (Luks, 1982). Due to the constant size k of the neighborhood graphs, the algorithm runs in time polynomial in the size of the original graph and, on average, in time linear in k (Babai et al., 1980). Our experiments verify that computing a canonical labeling of the graph neigborhoods adds a negligible overhead.

Algorithm 4 lists the normalization procedure. If the size of the input set U is larger than k, it first applies the ranking based on ℓ to select the top k nodes and recomputes a ranking on the smaller set of nodes. If the size of U is smaller than k, it adds disconnected dummy nodes. Finally, it induces the subgraph on the vertices N and canonicalizes the graph taking the ranking $\mathbf{r}$ as prior coloring.

We can relate PATCHY-SAN to CNNs for images as follows.

Theorem 3. *Given a sequence of pixels taken from an image. Applying* PATCHY-SAN *with receptive field size* $(2m-1)^2$*, stride* s*, no zero padding, and* 1*-WL normalization to the sequence is identical (up to a fixed permutation of the receptive field) to the first layer of a* CNN *with receptive field size* $2m-1$*, stride* s*, and no zero padding.*

Proof: It is possible to show that if an input graph is a square grid, then the 1-WL normalized receptive field constructed for a vertex is always a square grid graph with a unique vertex order. □

4.4. Convolutional Architecture

PATCHY-SAN is able to process both vertex and edge attributes (discrete and continuous). Let $\mathtt{a}_v$ be the number of vertex attributes and let $\mathtt{a}_e$ be the number of edge attributes. For each input graph G, it applies normalized receptive fields for vertices and edges which results in one $(w, k, \mathtt{a}_v)$ and one $(w, k, k, \mathtt{a}_e)$ tensor. These can be reshaped to a $(wk, \mathtt{a}_v)$ and a $(wk^2, \mathtt{a}_e)$ tensors. Note that $\mathtt{a}_v$ and $\mathtt{a}_e$ are the number of input channels. We can now apply a 1-dimensional convolutional layer with stride and receptive field size k to the first and k^2 to the second tensor. The rest of the architecture can be chosen arbitrarily. We may use merge layers to combine convolutional layers representing nodes and edges, respectively.

5. Complexity and Implementation

PATCHY-SAN's algorithm for creating receptive fields is highly efficient and naively parallelizable because the fields are generated independently. We can show the following asymptotic worst-case result.

Theorem 4. *Let* N *be the number of graphs, let* k *be the receptive field size,* w *the width, and* $O(f(n,m))$ *the complexity of computing a given labeling* ℓ *for a graph with* n *vertices and* m *edges.* PATCHY-SAN *has a worst-case complexity of* $O(Nw(f(n,m) + n\log(n) + \exp(k)))$ *for computing the receptive fields for* N *graphs.*

Proof: Node sequence selection requires the labeling of each input graph and the retrieval of the k highest ranked nodes. For the creation of normalized graph patches, most computational effort is spent applying the labeling procedure ℓ to a neighborhood whose size may be larger than k. Let $\overline{d}$ be the maximum degree of the input graph G, and U the neighborhood returned by Algorithm 2. We have $|U| \leq (k-2)\overline{d} \leq n$. The term $\exp(k)$ comes from the worst-case complexity of the graph canonicalization algorithm NAUTY on a k node graph (Miyazaki, 1997). □

For instance, for the Weisfeiler-Lehman algorithm, which has a complexity of $O((n+m)\log(n))$ (Berkholz et al., 2013), and constants $w \ll n$ and $k \ll n$, the complexity of PATCHY-SAN is linear in N and quasi-linear in m and n.

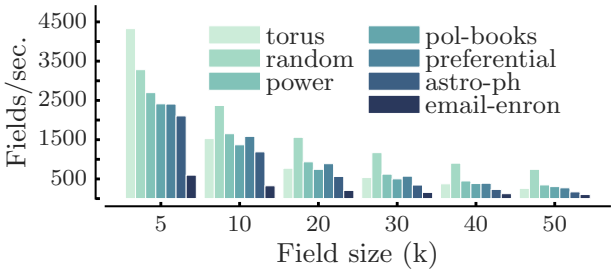

Figure 4. Receptive fields per second rates on different graphs.

6. Experiments

We conduct three types of experiments: a runtime analysis, a qualitative analysis of the learned features, and a comparison to graph kernels on benchmark data sets.

6.1. Runtime Analysis

We assess the efficiency of PATCHY-SAN by applying it to real-world graphs. The objective is to compare the rates at which receptive fields are generated to the rate at which state of the art CNNs perform learning. All input graphs are part of the collection of the Python module GRAPH-TOOL[1]. For a given graph, we used PATCHY-SAN to compute a receptive field for *all* nodes using 1-WL normalization. **torus** is a periodic lattice with $10,000$ nodes; **random** is a random undirected graph with $10,000$ nodes and a degree distribution $P(k) \propto 1/k$ and $k_{\max} = 3$; **power** is a network representing the topology of a power grid in the US; **pol-books** is a co-purchasing network of books about US politics published during the 2004 presidential election; **preferential** is a preferential attachment network model where newly added vertices have degree 3; **astro-ph** is a coauthorship network between authors of preprints posted on the astrophysics arxiv (Newman, 2001); **email-enron** is a communication network generated from about half a million sent emails (Leskovec et al., 2009). All experiments were run on commodity hardware with 64G RAM and a single 2.8 GHz CPU.

[1] https://graph-tool.skewed.de/

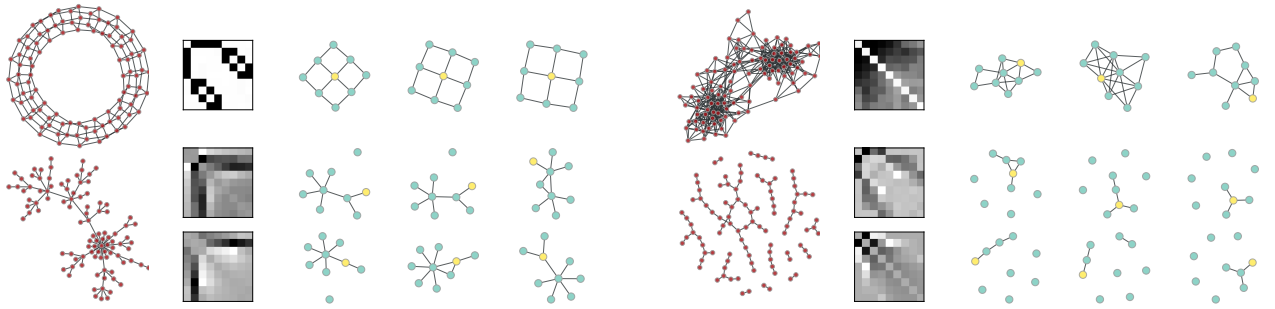

Figure 5. Visualization of RBM features learned with 1-WL normalized receptive fields of size 9 for a torus (periodic lattice, top left), a preferential attachment graph (Barabási & Albert 1999, bottom left), a co-purchasing network of political books (top right), and a random graph (bottom right). Instances of these graphs with about 100 nodes are depicted on the left. A visual representation of the feature's weights (the darker a pixel, the stronger the corresponding weight) and 3 graphs sampled from the RBMs by setting all but the hidden node corresponding to the feature to zero. Yellow nodes have position 1 in the adjacency matrices. (Best seen in color.)

Figure 4 depicts the receptive fields per second rates for each input graph. For receptive field size $k = 5$ and $k = 10$ PATCHY-SAN creates fields at a rate of more than $1000/s$ except for **email-enron** with a rate of $600/s$ and $320/s$, respectively. For $k = 50$, the largest tested size, fields are created at a rate of at least $100/s$. A CNN with 2 convolutional and 2 dense layers learns at a rate of about 200-400 training examples per second on the same machine. Hence, the speed at which receptive fields are generated is sufficient to saturate a downstream CNN.

6.2. Feature Visualization

The visualization experiments' aim is to qualitatively investigate whether popular models such as the restricted Boltzman machine (RBM) (Freund & Haussler, 1992) can be combined with PATCHY-SAN for unsupervised feature learning. For every input graph, we have generated receptive fields for all nodes and used these as input to an RBM. The RBM had 100 hidden nodes and was trained for 30 epochs with contrastive divergence and a learning rate of 0.01. We visualize the features learned by a single-layer RBM for 1-WL normalized receptive fields of size 9. Note that the features learned by the RBM correspond to reoccurring receptive field patterns. Figure 5 depicts some of the features and samples drawn from it for four different graphs.

6.3. Graph Classification

Graph classification is the problem of assigning graphs to one of several categories.

Data Sets. We use 6 standard benchmark data sets to compare run-time and classification accuracy with state of the art graph kernels: MUTAG, PCT, NCI1, NCI109, PROTEIN, and D&D. MUTAG (Debnath et al., 1991) is a data set of 188 nitro compounds where classes indicate whether the compound has a mutagenic effect on a bacterium. PTC consists of 344 chemical compounds where classes indicate carcinogenicity for male and female rats (Toivonen et al., 2003). NCI1 and NCI109 are chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines (Wale & Karypis, 2006). PROTEINS is a graph collection where nodes are secondary structure elements and edges indicate neighborhood in the amino-acid sequence or in 3D space. Graphs are classified as enzyme or non-enzyme. D&D is a data set of 1178 protein structures (Dobson & Doig, 2003) classified into enzymes and non-enzymes.

Experimental Set-up. We compared PATCHY-SAN with the shortest-path kernel (SP) (Borgwardt & Kriegel, 2005), the random walk kernel (RW) (Gaertner et al., 2003), the graphlet count kernel (GK) (Shervashidze et al., 2009), and the Weisfeiler-Lehman subtree kernel (WL) (Shervashidze et al., 2011). Similar to previous work (Yanardag & Vishwanathan, 2015), we set the height parameter of WL to 2, the size of the graphlets for GK to 7, and chose the decay factor for RW from $\{10^{-6}, 10^{-5}, ..., 10^{-1}\}$. We performed 10-fold cross-validation with LIB-SVM (Chang & Lin, 2011), using 9 folds for training and 1 for testing, and repeated the experiments 10 times. We report average prediction accuracies and standard deviations.

For PATCHY-SAN (referred to as PSCN), we used 1-WL normalization, a width w equal to the average number of nodes (see Table 1), and receptive field sizes of $k = 5$ and $k = 10$. For the experiments we only used node attributes. In addition, we ran experiments for $k = 10$ where we combined receptive fields for nodes and edges using a merge layer ($k = 10^{\text{E}}$). To make a fair comparison, we used a

Data set	MUTAG	PCT	NCI1	PROTEIN	D & D
Max	28	109	111	620	5748
Avg	17.93	25.56	29.87	39.06	284.32
Graphs	188	344	4110	1113	1178
SP [7]	85.79 ± 2.51	58.53 ± 2.55	73.00 ± 0.51	75.07 ± 0.54	> 3 days
RW [16]	83.68 ± 1.66	57.26 ± 1.30	> 3 days	74.22 ± 0.42	> 3 days
GK [37]	81.58 ± 2.11	57.32 ± 1.13	62.28 ± 0.29	71.67 ± 0.55	78.45 ± 0.26
WL [38]	80.72 ± 3.00 $(5s)$	56.97 ± 2.01 $(30s)$	80.22 ± 0.51 $(375s)$	72.92 ± 0.56 $(143s)$	77.95 ± 0.70 $(609s)$
PSCN k=5	91.58 ± 5.86 $(2s)$	59.43 ± 3.14 $(4s)$	72.80 ± 2.06 $(59s)$	74.10 ± 1.72 $(22s)$	74.58 ± 2.85 $(121s)$
PSCN k=10	88.95 ± 4.37 $(3s)$	62.29 ± 5.68 $(6s)$	76.34 ± 1.68 $(76s)$	75.00 ± 2.51 $(30s)$	76.27 ± 2.64 $(154s)$
PSCN k=10^E	92.63 ± 4.21 $(3s)$	60.00 ± 4.82 $(6s)$	78.59 ± 1.89 $(76s)$	75.89 ± 2.76 $(30s)$	77.12 ± 2.41 $(154s)$
PSLR k=10	87.37 ± 7.88	58.57 ± 5.46	70.00 ± 1.98	71.79 ± 3.71	68.39 ± 5.56

Table 1. Properties of the data sets and accuracy and timing results (in seconds) for PATCHY-SAN and 4 state of the art graph kernels.

Data set	GK [37]	DGK [44]	PSCN k=10
COLLAB	72.84 ± 0.28	73.09 ± 0.25	72.60 ± 2.15
IMDB-B	65.87 ± 0.98	66.96 ± 0.56	71.00 ± 2.29
IMDB-M	43.89 ± 0.38	44.55 ± 0.52	45.23 ± 2.84
RE-B	77.34 ± 0.18	78.04 ± 0.39	86.30 ± 1.58
RE-M5k	41.01 ± 0.17	41.27 ± 0.18	49.10 ± 0.70
RE-M10k	31.82 ± 0.08	32.22 ± 0.10	41.32 ± 0.42

Table 2. Comparison of accuracy results on social graphs [44].

single network architecture with two convolutional layers, one dense hidden layer, and a softmax layer for all experiments. The first convolutional layer had 16 output channels (feature maps). The second convolutional layer has 8 output channels, a stride of $s = 1$, and a field size of 10. The convolutional layers have rectified linear units. The dense layer has 128 rectified linear units with a dropout rate of 0.5. Dropout and the relatively small number of neurons are needed to avoid overfitting on the smaller data sets. The only hyperparameter we optimized is the number of epochs and the batch size for the mini-batch gradient decent algorithm RMSPROP. All of the above was implemented with the THEANO (Bergstra et al., 2010) wrapper KERAS (Chollet, 2015). We also applied a logistic regression (PSLR) classifier on the patches for $k = 10$.

Moreover, we ran experiments with the same set-up[2] on larger social graph data sets (up to 12000 graphs each, with an average of 400 nodes), and compared PATCHY-SAN with previously reported results for the graphlet count (GK) and the deep graphlet count kernel (DGK) (Yanardag & Vishwanathan, 2015). We used the normalized node degree as attribute for PATCHY-SAN, highlighting one of its advantages: it can easily incorporate continuous features.

Results. Table 1 lists the results of the experiments. We omit the results for NCI109 as they are almost identical to NCI1. Despite using a one-fits-all CNN architecture, the CNNs accuracy is highly competitive with existing graph kernels. In most cases, a receptive field size of 10 results in the best classification accuracy. The relatively high variance can be explained with the small size of the benchmark data sets and the fact that the CNNs hyperparameters (with the exception of epochs and batch size) were not tuned to individual data sets. Similar to the experience on image and text data, we expect PATCHY-SAN to perform even better for large data sets. Moreover, PATCHY-SAN is between 2 and 8 times more efficient than the most efficient graph kernel (WL). We expect the performance advantage to be much more pronounced for data sets with a large number of graphs. Results for betweeness centrality normalization are similar with the exception of the runtime which increases by about 10%. Logistic regression applied to PATCHY-SAN's receptive fields performs worse, indicating that PATCHY-SAN works especially well in conjunction with CNNs which learn non-linear feature combinations and which share weights across receptive fields.

PATCHY-SAN is also highly competitive on the social graph data. It significantly outperforms the other two kernels on four of the six data sets and achieves ties on the rest. Table 2 lists the results of the experiments.

7. Conclusion and Future Work

We proposed a framework for learning graph representations that are especially beneficial in conjunction with CNNs. It combines two complementary procedures: (a) selecting a sequence of nodes that covers large parts of the graph and (b) generating local normalized neighborhood representations for each of the nodes in the sequence. Experiments show that the approach is competitive with state of the art graph kernels.

Directions for future work include the use of alternative neural network architectures such as RNNs; combining different receptive field sizes; pretraining with RBMs and autoencoders; and statistical relational models based on the ideas of the approach.

[2]Due to the larger size of the data sets, we removed dropout.

ACKNOWLEDGMENTS

Many thanks to the anonymous ICML reviewers who provided tremendously helpful comments.

References

Alon, Uri. Network motifs: theory and experimental approaches. *Nature Reviews Genetics*, 8(6):450–461, 2007.

Atlas, Les E., Homma, Toshiteru, and Marks, Robert J. II. An artificial neural network for spatio-temporal bipolar patterns: Application to phoneme classification. In Anderson, D.Z. (ed.), *Neural Information Processing Systems*, pp. 31–40. 1988.

Babai, László, Erdős, Paul, and Selkow, Stanley M. Random graph isomorphism. *SIAM J. Computing*, 9(3):628–635, 1980.

Barabási, Albert-Laszlo and Albert, Réka. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

Bergstra, James, Breuleux, Olivier, Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Desjardins, Guillaume, Turian, Joseph, Warde-Farley, David, and Bengio, Yoshua. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, 2010.

Berkholz, Christoph, Bonsma, Paul S., and Grohe, Martin. Tight lower and upper bounds for the complexity of canonical colour refinement. In *Proceedings of the European Symposium on Algorithms*, pp. 145–156, 2013.

Borgwardt, Karsten M. and Kriegel, Hans-Peter. Shortest-path kernels on graphs. In *Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM)*, pp. 74–81, 2005.

Bruna, Joan, Zaremba, Wojciech, Szlam, Arthur, and LeCun, Yann. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations*, 2014.

Chang, Chih-Chung and Lin, Chih-Jen. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, 2011.

Chollet, François. keras. https://github.com/fchollet/keras, 2015.

Debnath, Asim Kumar, de Compadre, Rosa L. Lopez, Debnath, Gargi, Shusterman, Alan J., and Hansch, Corwin. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *J. Med. Chem.*, (34):786–797, 1991.

Dobson, Paul D. and Doig, Andrew J. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771 – 783, 2003.

Duvenaud, David K, Maclaurin, Dougal, Iparraguirre, Jorge, Bombarell, Rafael, Hirzel, Timothy, Aspuru-Guzik, Alan, and Adams, Ryan P. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*, pp. 2215–2223, 2015.

Freund, Yoav and Haussler, David. Unsupervised learning of distributions of binary vectors using two layer networks. In *Advances in Neural Information Processing Systems*, pp. 912–919, 1992.

Fukushima, Kunihiko. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

Gaertner, Thomas, Flach, Peter, and Wrobel, Stefan. On graph kernels: Hardness results and efficient alternatives. In *Proceedings of the 16th Annual Conference on Computational Learning Theory*, pp. 129–143, 2003.

Haussler, David. Convolution kernels on discrete structures. Technical report, Department of Computer Science, University of California at Santa Cruz, 1999.

Henaff, Mikael, Bruna, Joan, and LeCun, Yann. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

Hubel, David H. and Wiesel, Torsten N. Receptive fields and functional architecture of monkey striate cortex. *Journal of Physiology (London)*, 195:215–243, 1968.

Kersting, Kristian, Ahmadi, Babak, and Natarajan, Sriraam. Counting belief propagation. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 277–284, 2009.

Kersting, Kristian, Mladenov, Martin, Garnett, Roman, and Grohe, Martin. Power iterated color refinement. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1904–1910, 2014.

Kondor, Risi and Borgwardt, Karsten M. The skew spectrum of graphs. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, pp. 496–503, 2008.

Kondor, Risi and Lafferty, John. Diffusion kernels on graphs and other discrete input spaces. In *Proceedings of*

the 19th International Conference on Machine Learning (ICML), pp. 315–322, 2002.

Kondor, Risi, Shervashidze, Nino, and Borgwardt, Karsten M. The graphlet spectrum. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pp. 529–536, 2009.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, 1989.

LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

LeCun, Yann, Bengio, Yoshua, and Hinton, Geoffrey. Deep learning. *Nature*, 521:436–444, 2015.

Leskovec, Jure, Lang, Kevin J, Dasgupta, Anirban, and Mahoney, Michael W. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

Li, Yujia, Tarlow, Daniel, Brockschmidt, Marc, and Zemel, Richard. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Luks, Eugene M. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, (25):42–65, 1982.

McKay, Brendan D. and Piperno, Adolfo. Practical graph isomorphism, {II}. *Journal of Symbolic Computation*, 60(0):94 – 112, 2014.

Milo, Ron, Shen-Orr, Shai, Itzkovitz, Shalev, Kashtan, Nadav, Chklovskii, Dmitri, and Alon, Uri. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

Miyazaki, Takunari. The complexity of mckays canonical labeling algorithm. In *Groups and Computation II*, volume 28, pp. 239–256, 1997.

Newman, Mark EJ. The structure of scientific collaboration networks. *Proceedings of the National Academy of Sciences*, 98(2):404–409, 2001.

Orsini, F., Frasconi, P., and Raedt, L. De. Graph invariant kernels. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 678–689, 2015.

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

Shervashidze, Nino, Vishwanathan, S.V.N., Petri, Tobias H., Mehlhorn, Kurt, and Borgwardt, Karsten M. Efficient graphlet kernels for large graph comparison. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 488–495, 2009.

Shervashidze, Nino, Schweitzer, Pascal, van Leeuwen, Erik Jan, Mehlhorn, Kurt, and Borgwardt, Karsten M. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561, 2011.

Toivonen, Hannu, Srinivasan, Ashwin, King, Ross D, Kramer, Stefan, and Helma, Christoph. Statistical evaluation of the predictive toxicology challenge 2000–2001. *Bioinformatics*, 19(10):1183–1193, 2003.

Vishwanathan, S. V. N., Schraudolph, Nicol N., Kondor, Risi, and Borgwardt, Karsten M. Graph kernels. *J. Mach. Learn. Res.*, 11:1201–1242, 2010.

Wale, Nikil and Karypis, George. Comparison of descriptor spaces for chemical compound retrieval and classification. In *Proceedings of the International Conference on Data Mining (ICDM)*, pp. 678–689, 2006.

Wallach, Izhar, Dzamba, Michael, and Heifets, Abraham. Atomnet: A deep convolutional neural network for bioactivity prediction in structure-based drug discovery. *CoRR*, abs/1510.02855, 2015.

Weisfeiler, Boris and Lehman, AA. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9): 12–16, 1968.

Yanardag, Pinar and Vishwanathan, S.V.N. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1365–1374, 2015.