Deep Learning of Graphs with Ngram Convolutional Neural Networks

Zhiling Luo, Ling Liu, *Fellow, IEEE,* Jianwei Yin, *Member, IEEE* Ying Li, and Zhaohui Wu

Abstract—Convolutional Neural Network (CNN) has gained attractions in image analytics and speech recognition in recent years. However, employing CNN for classification of graphs remains to be challenging. This paper presents the Ngram graph-block based convolutional neural network model for classification of graphs. Our Ngram deep learning framework consists of three novel components. First, we introduce the concept of n-gram block to transform each raw graph object into a sequence of n-gram blocks connected through overlapping regions. Second, we introduce a diagonal convolution step to extract local patterns and connectivity features hidden in these n-gram blocks by performing n-gram normalization. Finally, we develop deeper global patterns based on the local patterns and the ways that they respond to overlapping regions by building a n-gram deep learning model using convolutional neural network. We evaluate the effectiveness of our approach by comparing it with the existing state of art methods using five real graph repositories from bioinformatics and social networks domains. Our results show that the Ngram approach outperforms existing methods with high accuracy and comparable performance.

Index Terms—Classification of Graphs, Convolutional Neural Network, N-gram modeling, Subgraph Pattern Extraction

1 Introduction

Many complex problems in business, science and engineering can be formulated as graph problems and solved by using graph analytic algorithms. The problem of classification of graphs treats graphs as complex objects and constructs deep learning models to learn classification of graphs based on common substructure patterns hidden in those graphs. For example, MUTAG dataset consists of many nitro compounds where class labels can indicate whether the compound has a mutagenic effect on a bacterium [1]. Another example is mapping unseen compounds to their level of activity against cancer cells [2].

Conventional approaches to classification of complex objects, such as protein structures, social graphs, images, rely on designing adequate similarity function(s) to measure the similarity distance between two complex objects and then use the off-the-shelf classification algorithms [3]. Based on graph-similarity computation models, existing approaches to classification of graphs can be broadly classified into two categories. (1) The *local subgraph based* approaches compare graphs based on the existence or count of small substructures [4][5][6][7]. The main challenge is to identify the significant subgraph structures as the signature features for classification of arbitrary graphs in the training set [4][5]. Then by representing each graph as a feature vector with each element denoting the weight on the respective subgraph structure, existing off-the-shelf machine learning algorithms can be applied. A main problem with using such subgraph structures as signatures is the restriction of using very small subgraphs with a few nodes (window size of < 10 nodes) due to the combinatorial complexity of subgraph enumeration for large window size. Consequently, these approaches fail to capture the complex structure patterns of graphs. This limitation can lead to high error ratio due to missing of the subgraph patterns that are critical to classification but cannot be captured by using the small window size. (2) The *global similarity-based* approaches compute the pairwise similarity (distance) of graphs [8][9][10][8], typically by first encoding the subgraph features and then creating the distance matrix to record pairwise similarity for every pair of graphs, before employing the off-the-shelf supervised learning algorithms, e.g., kNN and SVM, on the distance matrix. Graph kernel [8] and graph embedding [11] are the two most recent representative methods in this category. Deep Graph Kernel (DGK) [8] is the state-of-art approach in graph kernel family. The key idea is to leverage the dependency information between substructures by learning their latent representations. Their framework, especially when using Weiseiler-Lehman kernel [12], achieves some competitive result on open benchmarks. Therefore, the DGK approach is one of the popular methods used for experimental comparison in terms of efficiency and effectiveness in literature [11], [13], [14].

However, existing approaches in both categories suffer from some serious drawbacks. First, comparing to classification of text, image, video and scene datasets, feature extractions for graphs pose some unique challenges. Graphs consist of two types of primitive elements: vertices and edges. Analyzing graphs as whole objects requires capturing not only the shallow features from explicit topological structure of a graph but also the deep features from the implicit (hidden) correlation structures at both vertex and edge level. Thus, it is hard to represent graphs in a deterministic feature space [7]. Second, capturing the implicit structural correlation patterns is critical for high quality classification of graphs. Neither small and fixed size of subgraph pattern matching (local) nor pairwise similarity of graphs (global) are sufficient for capturing the complex hidden correlation patterns for classification of graphs that have different size and different structural complexity.

To address the above challenges, we develop n-gram graph-block based convolutional neural network (CNN) for deep learning of graphs. The NgramCNN approach allows us to capture complex patterns of graphs with small-size substructure filters by leveraging multiple convolution layers and feed-forwarding iterative composition of simple patterns. Our NgramCNN approach is novel in three aspects: First, we introduce the concept of **n-gram blocks** to transform each raw graph into a sequence of n-gram blocks connected through overlapping regions. By leveraging the

overlapping n-gram blocks based approach, we can effectively encode the complex structural correlation patterns over the collection of graphs in different sizes and different topology complexity. Second, we introduce the concept of **diagonal convolution with n-gram normalization** to extract interesting local patterns and connectivity features hidden in these n-gram blocks. By leveraging n-gram normalization and diagonal convolution as the optimized initialization step, our approach allows early pruning of those blocks that contain no edge connectivity between any pair of vertices in a graph and thus do not contribute to the classification of the given collection of graphs. Third, we conduct deep learning of global substructure patterns through a sequence of convolutional layers by leveraging local patterns and iteratively refined weight filters. To the best of our knowledge, this work is the first to develop a deep learning framework using Ngram graph blocks based diagonal CNN for classification of graphs. We compare our approach with the existing state of art graph classification methods using five real-world graph datasets from bioinformatics and social networks domains. Our experimental results show that the Ngram approach outperforms existing methods with high accuracy and comparable performance.

The rest of this paper is organized as follows. Section 2 briefly reviews the related work. Section 3 introduces Ngram diagonal CNN model and its deep learning framework. Section 4 reports our experimental evaluation and Section 5 concludes the paper.

2 Related Work and Overview

Convolutional neural network (CNN) [15] is a type of artificial neural networks (ANNs), widely used in many science and business domains for image recognition, speech recognition, drug discovery, protein structure mining, complex manufacturing analytics, and computer-assisted game play (e.g., Google Alphago).

A CNN consists of a sequence of stacked convolutional layers, and every layer transforms one volume of activations to another through a differentiable function. Each layer is made up of a set of neurons that have learnable weights and biases. Each neuron is fully connected to all neurons in the previous layer. Neurons in a single layer function independently and do not share any connections. The last fully-connected layer is called the "output layer" and it represents the class scores. Typically, a CNN architecture consists of input and three main types of layers: Convolutional Layer, Pooling Layer, and Fully-Connected Layer [16]. In the context of image recognition, the input of $32 \times 32 \times 3$ will hold the raw pixel values of an image of width 32, height 32, and with three color channels R, G, B. A convolution layer will compute the output of neurons that are connected to local regions in the input by performing a dot product between their weights and a small region they are connected to in the input volume, and each neuron learns by optionally following it with a nonlinearity tuned at each iteration through gradient descent based refinement of weights and biases. Through these neurons, a CNN transforms an original image layer by layer from the original pixel values to the final class scores. Some layers contain parameters and other do not. Convolution layer and fully connected layer perform transformations that are a function of both the activations in the input volume, and the parameters (the weights and biases of the neurons) that are refined in each iteration of the deep learning process. These parameters will be trained with gradient descent iteratively so that the class scores that the CNN computes are consistent with the labels in the training set for each image.

CNN has had some noticeable success in deep learning over sequential data, e.g., text [17], image and grid data [18], video and stream data [19] as well as large scale Scene analysis [20]. Recent work extends the baseline CNNs to classification of graphs. [21] addresses the window size issue by constructing a deep CNN that can capture more complex graph patterns through adding multiple convolution layers.[22] employs convolutional type operations on graphs to develop a differentiable variant for specific graph feature in the context of molecular fingerprints. Mikael et. al [23] extends spectral networks by employing graph estimation for better classification on text categorization, computational biology and computer version. [24] introduces Graph LSTM (Long Short-Term Memory) as the generalization from sequential data to graph structured data. [13] is the first to apply CNN to classification of graph objects in a graph repository. PSCN encodes nodes by the neighborhood nodes within the system-default window size k, and then applies a standard 1-Dimension convolutional neural network. PSCN achieves better results on the open datasets comparing with the Deep Graph Kernel [8]. However, it still suffers from some drawbacks. First, the selection of neighborhood is determined by the window-size k, which is less than 10, because a larger window-size k will result in unacceptable running time and memory usage. Second, PSCN cannot perform deep learning effectively with the small window size k because they lose the complex subgraph features when some input graphs in the repository have the dense connectivity features that are beyond the pre-defined system default window size. Third, the classification results of PSCN are sensitive to the labeling approach, in which the nodes in neighborhood are ranked, since their labeling approach works on one dataset and may fail on another.

To tackle the problems of PSCN, we design and implement an unified NgramCNN framework for classification of graphs with high accuracy. Our NgramCNN is scalable to various graph datasets with graphs of varying sizes and complexity, and it outperforms PSCN and Deep Graph kernel in terms of accuracy by careful integration of four optimizations: the n-gram vertex blocks of equal size, the n-gram normalization, the diagonal convolution layer, and the extraction of complex subgraph structures by stacking subsequent convolution layers on top of our diagonal convolution layer. A unique characteristics of our NgramCNN, comparing with these existing approaches, is the design of a graph specific convolutional neural network structure for focused learning of complex and deep subgraph patterns. For example, we normalize each graph by capturing the local connectivity through vertex (node) ID ordering. This approach has two key challenges: determine the sequence of nodes from which to create neighborhood substructure and define a unique mapping to transform a graph from graph representation to vector representation such that nodes with similar structural roles in the neighborhood substructures are positioned similarly in the vector representation. In addition to introducing the n-gram vertex blocks to capture local connectivity of complex graphs of different sizes through small subgraph structure of fixed size, our NgramCNN offers

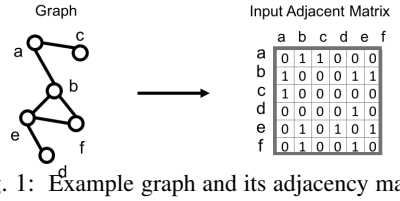


Fig. 1: Example graph and its adjacency matrix.

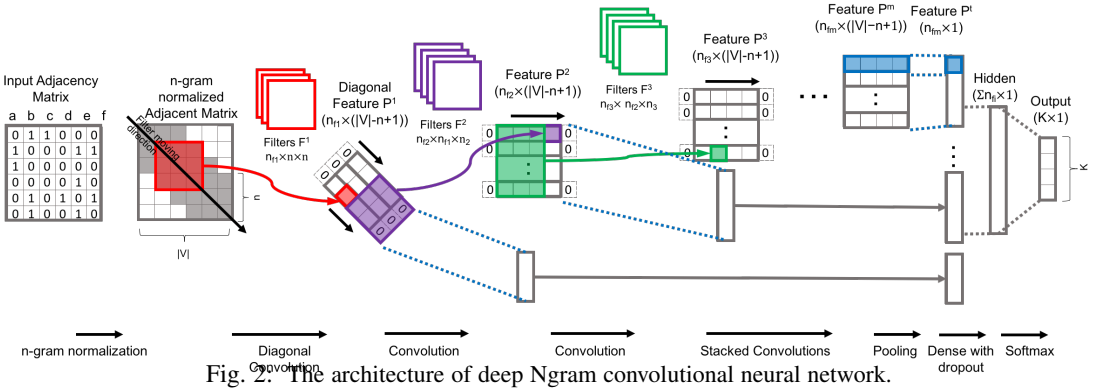


Fig. 2. The architecture of deep Ngram convolutional neural network.

another advantage over existing methods. We introduce n-gram normalization to construct the diagonal representation of input graph to enable our NgramCNN start with a diagonal convolution layer. In summary, by encoding graph specific properties into the deep learning architecture, NgramCNN can make the forward function more effectively focused on the important computations and parameter tuning in the network, which contribute significantly to the objective function of the classification of graphs in the given collection. Our experimental evaluation shows that our approach outperforms existing ones with high accuracy and comparable performance.

Orthogonal to deep learning models for classification of a repository of graphs, another thread of graph analytics research has been centered on mining a big graph by treating each graph as a dataset of heterogeneous vertices and edges. The graph analysis objective is to label vertices and edges in the input graph with a given set of class labels based on pairwise vertex similarity computed using either a graph traversal model [25] or a pairwise vertex attribute based similarity measure or a hybrid approach [26].

3 Learning Ngram CNN for Graphs

3.1 Concepts and Definitions

A graph G is defined by (V, E) where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. $\forall v_i, v_j \in V$, if $(v_i, v_j) \in E$, then (v_i, v_j) denotes an edge in E connecting vertex v_i with vertex v_j. A graph $G = (V, E)$ can be represented by an adjacency matrix $\mathbf{A}$ of size $|V| \times |V|$, where $A_{i,j} = 1$ if $(v_i, v_j) \in E$, and $A_{i,j} = 0$ otherwise.

Figure 1 shows an example graph of six vertices and its corresponding adjacency matrix of size 6×6. For an undirected graph $G = (V, E)$, if $\forall (v_i, v_j) \in E$, $v_i \neq v_j$, then G has no self-edge and its adjacent matrix is a symmetrical matrix with value "0" in its diagonal elements. A basic property of adjacency matrix is that by switching two columns and corresponding rows of an adjacency matrix, we get another adjacency matrix representing the same graph.

Let $\mathcal{R}$ denote a collection of graphs, $\mathcal{L}$ denote the set of K class labels ($|\mathcal{L}| = K$) and $CL(G)$ denote the class label for an instance graph $G \in \mathcal{R}$. The problem of classification of $\mathcal{R}$ is defined as the graph labeling problem, which partitions $\mathcal{R}$ into a disjoint partitioning with K class labels, satisfying that $\forall G \in \mathcal{R}$, $\exists l \in \mathcal{L}$ such that $CL(\mathrm{G}) = l$. When $|\mathcal{L}| = 2$, we refer to the classification problem as the binary classification of $\mathcal{R}$.

Definition 1 (n-gram block) *Let $G = (V, E)$ denote a graph with $V = \{v_i | 1 \leq i \leq |V|\}$, and $\mathbf{A}$ denote an adjacency matrix of G with size $|V| \times |V|$. A n-gram block of $\mathbf{A}$ is defined as a square matrix block B of size $n \times n$ ($2 \leq n \leq |V|$), where $V_B = \{v_{i_1}, v_{i_2}, \ldots, v_{i_n}\}$, $1 \leq i_q \leq |V|$, $B_{i_p,i_q} = 1$ if $A_{i_p,i_q} = 1$ and $B_{i_p,i_q} = 0$ if $A_{i_p,i_q} = 0$.*

This definition states that a n-gram block in the adjacency matrix $\mathbf{A}$ of G corresponds to a subgraph of G. Let $S_B = (V_B, E_B)$ denote the subgraph S_B, where $V_B \subseteq V$, $E_B \subseteq V_B \times V_B \subseteq E$, $B_{i,j} = 1$ if $(v_i, v_j) \in E_B$ and $B_{i,j} = 0$ otherwise. For example, the matrix with shaded 3×3 blocks in Figure 1 shows an example n-gram block ($n = 3$).

3.2 Ngram CNN Architecture

Figure 2 illustrates the architecture of our Ngram convolutional neural network. An NgramCNN is a specific CNN for deep learning of graphs and is composed of the initial convolutional layer, and a series of feed-forward deep convolutional layers, followed by pooling layer and dropout layer before outputting by softmax. The initial convolutional layer is defined by the basic convolution fabric, which extracts common patterns found within local regions of the input graph, which is represented by a 2D adjacency matrix. Each convolutional layer employs a set of learnable weights, called filters or kernels, and each filter is convolved across the width and height of the input features, computing the inner product between the entries of the filter and the input, and producing a 2D activation map of that filter, shown as feature P^i ($1 \leq i \leq m$). In each convolutional layer, the CNN learns from two perspectives: (1) The 2D activation map P^i is used to learn the filters that activate at some spatial position in the input and extract more complex type of feature. (2) The output produced by the convolution layer is obtained by stacking the activation maps for all filters along the depth dimension. We can interpret each output entry as an output of a neuron, which can learn over a small region in the input, and by sharing learned feature parameters with other neurons in the same activation map, more complex type of feature can be captured and represented.

As shown in Figure 2, our NgramCNN architecture consisting of three core components: (1) The n-gram normalization, which sorts the nodes in the adjacency matrix of a graph and produces the n-gram normalized adjacency matrix. (2) A special convolution layer, called diagonal convolution, which extracts the common subgraph patterns found within local regions of the input graph. (3) A stacked deep convolution structure, which is built on top of diagonal convolution and repeated by a series of convolution layers and a pooling layer. Note that given a graph repository, we need to create an adjacent matrix of size $\alpha \times \alpha$ initially, prior to enter the two phases supervised learning model (training and

testing). α is the largest size of all the vertex sets for all the graphs in the given repository. In the subsequent sections, α and $|V|$ are used interchangeably when no confusion occurs.

3.3 n-gram Convolution Fabric

A graph can be represented by different adjacency matrices according to different ways to order its vertex set. Given an adjacency matrix, those matrix cells filled with value of "1" represent the connectivity between the vertices of the corresponding graph. One way to sort the vertices of a graph is to ensure that most of the pairwise connected vertices are close to one another. This is the main idea behind the concept of n-gram normalization.

Definition 2 (n-gram normalization) *Let $G = (V, E)$ denote a graph in the collection $\mathcal{R}$, $\mathbf{A}$ denote an adjacency matrix of G with size $|V| \times |V|$, and $\mathcal{S} = \{B_h | 1 \leq h \leq |\mathcal{S}|\}$ denote the collection of n-gram blocks extracted from $\mathbf{A}$ ($|\mathcal{S}| \leq |V| \times |V|$). The n-gram normalization of G is defined by a one-to-one vertex to vertex ID mapping on the vertex set V, defined by $M^n : V \rightarrow \{1, ..., |V|\}$ such that $\forall (v_i, v_j) \in E$, if $B \in \mathcal{S}$ and $v_i, v_j \in V_B$, then $|M^n(v_i) - M^n(v_j)| + 1 \leq n$.*

Intuitively, the vertex to vertex ID mapping function M^n is a sort function on the vertex set V. There are a number of different ways to sort the vertex set of a graph based on the semantic attributes of vertices or the traversal orders of the graph. In this paper, we consider a hybrid breath-first and depth-first traversal order, aiming at preserving the locality property of the graph, such that vertices within the same n-gram block are in close vicinity in G. The n-gram normalization aims to find the set of n-gram blocks, denoted by $\mathcal{R}_{\mathcal{B}}(G)$, in an adjacency matrix of G such that each $B \in \mathcal{R}_{\mathcal{B}}(G)$ is of size n and all n vertices in B have their vertex IDs sorted in an ascending order and the pairwise value difference in their positions in the adjacency matrix is no larger than n.

Consider the example graph in Figure 1 with six vertices and six edges. For ease of illustration, we use six letters *a, b, c, d, e, f* to denote the six vertices. Assume the original node sequence is $abcdef$, following the alphabetical order and six edges are (a, b), (a, c), (b, e), (b, f), (e, f) and (e, d). The adjacency matrix of the graph generated using the given vertex ID ordering is shown in Figure 1, next to the graph.

Clearly, this vertex order fails to capture the structural locality of the graph. This is because b, e and f form a dense subgraph structure in the raw input graph G (a triangle shape), which is an interesting subgraph structure in graph mining [27]. However, their positions in the adjacency matrix of Figure 2(a) are 2, 5 and 6, which are not close to one another within 3-hop distance even though they can be traversed within 3-hop distance in the original graph. This motivates us to introduce the n-gram normalization, which sorts the vertex set of G to ensure that the dense structures, such as the subgraph with vertices b, e, f, can be captured by the n-gram normalized adjacency matrix.

Consider the same example, the n-gram normalization with $n = 3$ over G can be achieved by the following vertex ID mapping $M(c) = 1$, $M(a) = 2$, $M(b) = 3$, $M(f) = 4$, $M(e) = 5$, and $M(d) = 6$. This new vertex ID mapping sorts the vertex set in the sequence of $cabfed$. The adjacency matrix corresponding to $cabfed$ is a 3-gram normalized matrix for this example graph because for edges (a, b), (a, c), (b, f), (e, f) and (e, d), their position difference is 1, and for edge (b, e), the position difference between their sorted order is $5 - 3 = 2$.

From this example, we show that the new vertex ID mapping helps to generate a 3-gram normalized adjacency matrix, which is more compact than the adjacency matrix generated by following the previous vertex ID ordering. We refer to this optimization as the n-gram normalization and the adjacency matrix obtained through the n-gram optimization as the n-gram normalized adjacency matrix. The n-gram normalized adjacency matrix typically has elements of "0" in the diagonal line, the left-bottom corner and the right-top corner of the matrix, and all the "1" elements are positioned in a belt area around the diagonal line. We conjecture that the n-gram normalization helps preserving the dense subgraph structures and the spatial patterns in local regions of a graph.

3.4 Augmentation by Isomorphism

Another important property of n-gram normalization is that given a raw graph G, there exists more than one way to order vertices of a graph by their numerical vertex ID, and hence there exists more than one n-gram normalized adjacency matrices. Furthermore, any two n-gram normalized adjacency matrices constructed from the same graph G are isomorphic.

Theorem 3.1 (Canonical Isomorphism) *Let X and Y are the two n-gram normalized matrices of a given graph G. X and Y are canonically isomorphic with respect to graph structure of G.*

Sketch of PROOF. According to category theory [28], a morphism $f : X \rightarrow Y$ in a category is an isomorphism if it admits a two-sided inverse, namely, there is another morphism $g : Y \rightarrow X$ in that category such that $gf = 1_x$ and $fg = 1_y$, where 1_x and 1_y are the identity morphisms of X and Y, respectively. A **canonical isomorphism** is a canonical map that is an isomorphism. Two objects are said to be **canonically isomorphic** if there is a canonical isomorphism between them. Given that X and Y are the two n-gram normalized adjacency matrices from G, we can define the morphism $f : X \rightarrow Y$ and the morphism $g : X \rightarrow Y$ through G and its n-gram normalization. Thus, there is a canonical map between X and Y that arises naturally from the definition or the construction of the two n-gram adjacency matrices X and Y. We call the canonical map the structure map or structure morphism because the map comes with the given graph structure of G, which is canonical for all n-gram normalized adjacency matrices of G. **QED.**

Figure 3(a) shows two different 3-gram normalized adjacency matrices for the same input graph on the left, both matrices are constructed from the same raw graph G with six vertices a, b, c, d, e, f. Each is constructed by following a different vertex ordering, and the n-gram blocks are highlighted in the belt area. The two different vertex ID orderings are $acbefd$ and $cabefd$, which result in two different 3-gram normalized adjacency matrices respectively.

In Ngram CNN, we leverage this canonical isomorphism property to produce different matrix representations of the same raw graph object G, each of which shows one alternative perspective of graph G in the matrix format. These isomorphic versions of G are used to augment the training set in the preprocessing stage of our Ngram CNN deep learning process, with the goal of optimizing the accuracy of our deep learning with Ngram CNN.

Augmentation of training datasets is a popular technique in CNN based image recognition [15]. Different parameters are used for augmenting training data with the goal of improving accuracy by examining raw images from different angles through

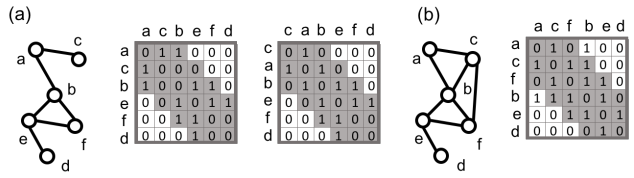


Fig. 3: (a)The strict n-gram normalized adjacency matrix with $n = 3$ and a isomorphic variant. (b) The relaxed n-gram normalized adjacency matrix with $n = 3$.

augmentation of raw input images as a preprocessing step prior to CNN learning.

3.5 Normalization Optimization

A main advantage of n-gram normalization is to prune out those left bottom and upper right cells that are filled by value of "0" in the diagonal convolution layer as these cells do not contribute significantly to the classification of graph G. Concretely, without n-gram normalization, one would need to examine a total of $(|V| - n + 1)^2$ n-gram blocks in each convolution layer for each iteration. With n-gram normalization, one can reduce the total number of n-gram blocks to be examined during deep learning from $(|V| - n + 1)^2$ to the number of normalized n-gram blocks, which is $|V| - n + 1$. Consider our running example in Figure 1(a): with $n = 3$, we obtain 4 $(|V| - n + 1)$ 3-gram normalized blocks, reducing the total number of 3-gram blocks to be examined for deep learning from 16 (i.e., $(|V| - n + 1)^2$) to 4, a 25% cost reduction. The perfect n-gram normalization for a graph G typically generates a n-gram normalized adjacency matrix such that only the cells in the diagonal n-gram block defined belt area have value "1" except the diagonal line and all the cells outside the diagonal n-gram block defined belt area have value "0".

However, given a repository of graphs and a system-defined n for performing n-gram normalization on every graph in the repository, it is possible that we may not be able to construct the perfect n-gram normalization for every graph such that all "1" elements in the adjacency matrix fall into the resulting n-gram blocks in the n-gram diagonal belt area after applying the n-gram normalization. Figure 3(b) shows an illustrative example. For the raw graph on the left, a n-gram normalized adjacency matrix is constructed on the right for $n = 3$. If the normalization were to capture all the n-gram blocks containing "1" elements, we would need to add another two n-gram blocks in addition to the four grey-colored 3-gram blocks after the normalization. For applications that have higher demand on runtime efficiency, one can reduce the number of basic convolution fabrics generated by using the relaxed n-gram normalization.

Definition 3 (Relaxed n-gram normalization) *Let $G = (V, E)$ denote a graph in the collection $\mathcal{R}$, $\mathbf{A}$ denote an adjacency matrix of G, and $\mathcal{S}$ denote the set of n-gram blocks of $\mathbf{A}$. Let P denote the upper bound on the number of normalized n-gram blocks required in the output of n-gram normalization. Let count(B) output the number of "1" element in the n-gram block B. The* relaxed n-gram normalization *with upper bound P is a one-to-one vertex-to-vertex ID mapping $M^n : V \rightarrow \{1, ..., |V|\}$ such that the following two conditions are met:*

(i) $\exists \mathcal{S}^P \subseteq \mathcal{S}$ such that $\forall B \in \mathcal{S}^P$, $\forall B' \in \mathcal{S} \setminus \mathcal{S}^P$, count($B$)$\geq$ count(B').

(ii) If $B = (V_B, E_B) \in S^P$ and $v_i, v_j \in V_B$, then $|M^n(v_i) - M^n(v_j)| + 1 \leq n$.

This definition introduces a relaxed normalization condition such that only those n-gram blocks in $\mathcal{S}^P$ needs to meet the condition (ii) and $\mathcal{S}^P$ contains the n-gram blocks that have larger number of "1" elements, representing significant subgraph structures in G, which are more densely connected. The relaxed n-gram normalization helps the users of our NgramCNN system to optimize the model performance with bounded accuracy by selecting an optimal P and a small window size n when performing deep learning of classification on a collection of graphs.

Figure 3(b) shows a relaxed 3-gram normalized adjacency matrix for the example graph on the left. Note that two elements of value "1" are omitted in the final result of the normalization. Thus, for window size $n = 3$, the 3-gram relaxed normalization enables us to only examine a total of four normalized 3-gram blocks instead of a total of sixteen 3-gram blocks. If we choose the window size $n = 4$ instead, then we need to examine a total of three normalized 4-gram blocks, instead of a total of nine 4-gram blocks ($(|V| - n + 1)^2 = (6 - 4 + 1)^2 = 9$). One way to compare the relaxed 3-gram normalized adjacency matrix with the 4-gram normalized adjacency matrix is to measure (i) the number of cells with value "1" outside the diagonal belt area and (ii) the number of cells with value "0" inside the diagonal belt area. For example, the number of cells with value "1" in the relaxed 3-gram normalized adjacency matrix is two and the number of cells with value "0" inside the diagonal 3-gram block belt area is four, as shown in Figure 3(b). In comparison, by choosing window size $n = 4$, we can obtain a perfect 4-gram normalization with no value "1" cells outside the diagonal belt area. But the number of cells with value "0" inside the diagonal 4-gram block belt area is eight.

This motivates us to define quantitative metrics that can guide us to select the best n with respect to the effect of n-gram normalization. Our experiments with real world datasets also show that the best window size n varies from dataset to dataset and larger n may not always deliver the high accuracy results for CNN deep learning [29].

The first quantitative measure is the loss of a relaxed n-gram normalization, which is defined by the number of cells with value "1" outside the diagonal n-gram belt area. Formally, $\boldsymbol{A}$ denote the n-gram normalized adjacency matrix of size $|V| \times |V|$, and let $A_{i,j}$ denote the cell at the i^{th} row and the j^{th} column of $\boldsymbol{A}$ $(1 \leq i, j \leq |V|)$. The loss of a relaxed n-gram normalization, denoted by $Loss(\boldsymbol{A})$, is defined as follows:

$$LS(\boldsymbol{A}, n) = \sum_{i=1}^{n} \sum_{j=i+n}^{|V|} A_{i,j} + \sum_{i=n+1}^{|V|} \sum_{j=1}^{i-n} A_{i,j} \qquad (1)$$

The second quantitative measure is the ratio of the cells with value "0" in the diagonal n-gram block belt area, denoted by ZR. It can be computed by counting the total number of cells in the diagonal belt area, denoted by TC, and the total number of cells with value "1" in the diagonal belt area, denoted by T1. The total number of cells with value "0" in the diagonal belt area is the difference between TC and T1. To calculate TC, we use $C_{i,j}$ $(1 \leq i, j \leq |V|)$ to denote the matrix of ones. Thus, the ratio ZR can

be computed as follows:

$$TC(\boldsymbol{A},n)=\sum_{i=1}^{n}\sum_{j=1}^{|V|-n+i-1}C_{i,j}+\sum_{i=n+1}^{|V|}\sum_{j=i-n+1}^{|V|}C_{i,j}$$
$$T1(\boldsymbol{A},n)=\sum_{i=1}^{n}\sum_{j=1}^{|V|-n+i-1}A_{i,j}+\sum_{i=n+1}^{|V|}\sum_{j=i-n+1}^{|V|}A_{i,j} \quad (2)$$
$$ZR(\boldsymbol{A},n)=\frac{TC\times T1}{TC}$$

In our Ngram CNN deep learning system, we use these two quantitative measures to facilitate the selection of the window size n by sampling a small number of graphs randomly from a given graph dataset. For each sample graph, we switch any two columns and corresponding rows. If we get a lower loss measure after this switching, we choose this switching. If we get an equal or higher loss after switching, we give up the switching. When the loss measure is zero, we compare the ZR measure and choose the low ZR measure as the better switching. This column/row switching process iterates and terminates when all possible column/row switching is examined and we cannot get a lower loss and lower ZR score.

3.6 Diagonal Convolution

There are several ways to optimize a CNN to improve deep learning quality and efficiency. We introduce a special *diagonal convolution* to optimize the first convolution layer by leveraging the n-gram normalized adjacency matrix of input graph to capture the substructure patterns and to perform some early pruning of unnecessary computations. Before defining the diagonal convolution, we briefly review the basic convolution operation in a typical CNN through a simplified example. Figure 4(a) shows a basic convolution operation on a 2×4 grid, which represents a 2×4 feature space. The convolution filter (also called kernel in literature), highlighted in red in Figure 4(a), is the major component in the convolution operation. In this example, two filters in grid size of 2×2 are used such that each filter has 4 weight parameters. For each filter, to apply it on the 2×4 grid, it needs to be applied for 3 times, each time it applies to a window size of 2×2, following the arrow direction from left to right sequentially. In each step, by applying the filter to one of the three sliding windows of size 2×2, the inner product between the filter and the sequential 2×2 window in the filter position is calculated. We get 2×3 results.

To prevent the feature dimension reduction in the convolution process, for this example, we want to ensure that the final result has 2×4 features, instead of reducing to 2×3, a zero-padding technique is widely used in CNN. As shown in Figure 4(a), extra 4 zeros are appended at the head and the tail of the sequential grid data, marked with dash line, such that the filters also perform the convolution on the expended data of size 2×6 and get the result in size of 2×5. After calculating the inner produce, for each element in the result part, an activation unit, e.g., sigmoid function, is applied. It ensures that each element is in the domain of $[-1,1]$. This convolution is called sequential convolution, which is widely used in natural language processing [17]. Other type of convolution operation includes 2D convolution used in image [18] and video processing [19], and tree-based convolution in programming language processing [30]. However, these existing convolution operations are not directly applicable to the graph structures in a meaningful way.

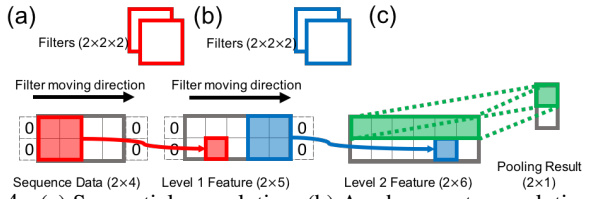


Fig. 4: (a) Sequential convolution. (b) A subsequent convolution. (c) Pooling.

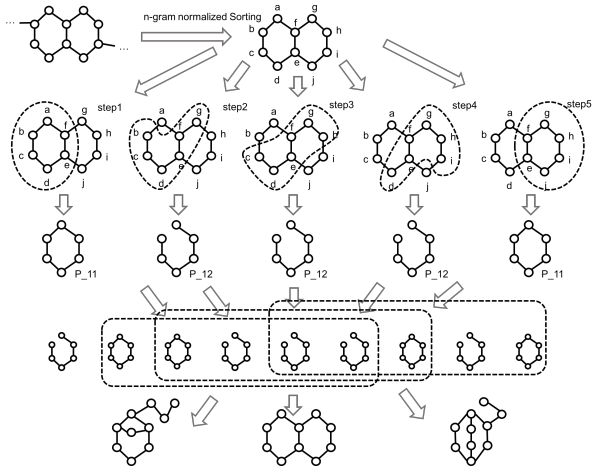


Fig. 5: Diagonal Convolution.

This motivates us to introduce a new convolution operation in our framework to handle graph data. Recall that the n-gram normalization ensures that all the connection information captured by elements of value "1" in the adjacency matrix is recorded in the belt area around diagonal line, as shown by highlighted regions in Figure 2(b). The key idea of diagonal convolution is that we use a group of convolution filters in size of $n\times n$ to feed-forward filtering by moving along the diagonal line of the n-gram normalized adjacency matrix and at each step we perform convolution operation on an n-gram normalized block of size $n\times n$.

Formally, consider the n-gram normalized adjacency matrix $\mathbf{A}^n$ of size $(|V|\times|V|)$, we take a total of $n_0\geq 1$ convolution filters denoted by $F^{0,i}, i\in\{1,...,n_0\}$ in size of $(n\times n)$, then the diagonal feature of filter $F^{0,i}$ at step $j\in\{1,...,|V|-n\}$ is

$$P^0_{i,j}=\alpha(\langle F^{0,i},A^N_{[j:j+n,j:j+n]}\rangle)=\alpha(\sum_{p=1}^{n}\sum_{q=1}^{n}F^{0,i}_{p,q}A^n_{j+p,j+q}) \quad (3)$$

where $\alpha(\cdot)$ is the activation unit function, e.g., sigmoid. Therefore, The feature obtained from the diagonal convolution is in the size of $n_0\times(|V|-n+1)$. In the subsequent discussion, we use $\mathcal{P}^0$ to denotes the diagonal features $\{P^0_{i,j}\}$ and use $\mathcal{F}^0$ to denote the filter parameters $\{F^{0,i}\}$.

Figure 5 illustrates the meaning of diagonal convolution by a graph of 10 nodes, namely $|V|=10$. It is observed that the graph has two rings of size six nodes, and two nodes are shared by these two ring structures. To capture such a ring based graph pattern, existing approaches usually require to have the window size larger than 10 [6][7]. However, our n-gram CNN approach can be effective even when the window size n is as small as six. Consider the original graph on the top left in Figure 5, we sort the nodes by the n-gram normalization with $n=6$ and get the order labeled graph on the top right. We use $abcdefghi$ to denote the sequence of sorted nodes. Then we take diagonal convolution

with filters in size of 6×6, namely $n = 6$. The filter can move by $|V| - n + 1 = 10 - 6 + 1 = 5$ steps. The five figures in the center of Figure 5 shows how each of the five filters covers (captures) the different patterns of the graph in each step. For example, in the first step, the filter stops at $A^n_{[1:6,1:6]}$ and it covers all the connections between any pair of nodes marked by a, b, c, d, e, f. As shown in step 1 of Figure 5, the filter highlighted by the dash line, covers the ring consisting of nodes a, b, c, d, e, f. More interestingly, using the diagonal convolution operation, different subgraph structures (features) can be captured by the same filter. For instance, steps 1 and 5 capture the same graph structure: the six-node ring. At the same time, steps 2,3 and 4 capture another same type of graph structure: the six-node line.

3.7 Go Deeper by Stacked Convolution Layers

Given that the window size of n is not sufficient to capture all substructure features for a graph of size larger than n, such as the example graph of size 10, to capture more complex substructure features beyond the window size of n, we introduce a deep convolutional structure on top of the diagonal convolution layer. Concretely, by taking the diagonal features $\mathcal{P}^0$ as input, we use a sequential diagonal convolution to get the first deep feature $\mathcal{P}^1$. Then by adding more sequential convolution layers repeatedly, we can get deeper features $\mathcal{P}^2$,$\mathcal{P}^3$,...,$\mathcal{P}^m$.

Figure 4(b) illustrates the detailed structure of how the multiple convolution layers are connected with one another. The second convolution layer takes the output from the first convolution layer as its input and applies the two filters on it. This process is repeatedly applied in the feed-forward deep feature learning structure, as shown in Figure 2(b). Table 1 shows the configuration setting in each convolution layer.

Note that for each convolution except diagonal convolution, we need to set the height of the filters to be the number of filters in the previous convolution layer. For example, for the convolution layer 2, the filter size is $n_1 \times s_2$, which means that the filter height is the same as the number of filters (n_1) in the convolution layer 1. This ensures that in the deep feature learning structure, filters in each layer can make the forward-move by one dimension.

Formally, for ith convolution layer, we take feature $\mathcal{P}^{i-1}$ in size of $n_{i-1} \times (|V| - n + 1)$ as input, extend it with zero-padding $(s_i - 1)/2$ on the left and zero-padding $(s_i - 1)/2$ on the right and get the $\hat{\mathcal{P}}^{i-1}$ in size of $n_{i-1} \times (|V| - n + s_i)$. Then we apply n_i filters $\mathcal{F}^i$ in size of $(n_{i-1} \times s_i)$, and get the feature $\mathcal{P}^i$. We define the elements of $\mathcal{P}^i$ as follows:

$$P^i_{j,k} = \alpha(\langle F^{i,j}, \hat{P}^{i-1}_{[1:n_{i-1},n:n+s_i-1]} \rangle) \tag{4}$$

3.8 Features in Stacked Convolution Layers

In stacked convolution layers, the filters, namely F^i for i'th convolution layer, represent the complex subgraph structures, denoted as $\mathcal{G}^i$, composed from previous relatively simpler subgraph structures. In this procedure, the derived filters, $\hat{F}^i$ are introduced, representing the composed value in matrix. We provide an illustrative example in Fig. 6 to describe the features learned through diagonal convolution and stacked convolution layers. Fig. 6 (a) shows the raw input graph with 14 vertices connected into two rings. Fig. 6 (b) shows the diagonal convolution filters and Fig. 6 (c) show the subgraph structure corresponding to each of the two diagonal convolution filters. The filter values are selected as 0 or 1 to simplify the calculation. Fig. 6 (d) shows the diagonal features computed via inner product with the filters in Fig. 6 (b). The values are obtained by the convolution Formula 3 and the cells surrounded by the dash line are zero-padding. The stacked convolution filters are illustrated in Fig. 6 (e) and the derived filters are shown in Fig. 6 (f). The values are also 0 or 1 for simple calculation. Fig. 6 (g) is the subgraph structures for the stack convolution. The features obtained after stack convolution are presented in Fig. 6 (h).

For diagonal convolution, the filters are exactly the derived filters because diagonal convolution layer is the first convolution layer. Note that each cell in filters has domain from -1 to 1, we can simply assign the cell whose value greater than 0 as 1, assign the cell whose value smaller or equal 0 as 0, to obtain subgraph structure directly. Fig. 6 (b) shows the diagonal convolution filter and Fig. 6 (c) represents the simple subgraph structures extracted from the diagonal convolution operations. We can see that one subgraph is a six-node line and another is a six-node ring. Both are the basic subgraph patterns (components) for the complex structures.

For stacked convolution layers, we need to construct the derived filters $\hat{F}^i$ for F^i and draw the subgraph structure $\mathcal{G}^i$ from $\hat{F}^i$. Recall that F^i contains n_i filters, which are both in size of $n_{i-1} \times s_i$. j'th row of F^i represents the weights of $\hat{F}^{i-1,j}$ to $\hat{F}^i$. Let $|\hat{F}^i|$ denote the heights and widths of $\hat{F}^i$. Formally,

$$\hat{F}^{i,j}_{[p,q]} = \begin{cases} 0 \text{ if } |p-q| \geq |\hat{F}^i| \text{or} p = q \\ \sum_{k=q-\hat{F}^{i-1}+1}^{p} \sum_{v=1}^{n_{i-1}} F^{i,j} \hat{F}^{i-1,v}_{[p-k,q+1-k]} \text{ else if } q > p \\ \sum_{k=p-\hat{F}^{i-1}+1}^{q} \sum_{v=1}^{n_{i-1}} F^{i,j} \hat{F}^{i-1,v}_{[q-k,p+1-k]} \text{ else if } p > q \end{cases} \tag{5}$$

The intuitive understanding is to stack the derived filters $\hat{F}^{i-1}$ in last layers along the diagonal direction with weights in F^i as shown in Fig. 6 (i). Consider the stacked two filters in Fig. 6 (e), the height of these filters should be equal to the number of filters in previous convolution layer, namely diagonal convolution, which is 2. The width of these two filters are 5, and each row represents the contribution of a subgraph structure in the previous (diagonal) convolution on the current complex subgraph structure. The first subgraph structure is the six-node line with five-edge, as shown in Fig. 6 (c). Therefore, the first row of the filter in stacked convolution layer, shown in Fig. 6 (e), whose values are "0","1","1","1","0", denotes the contribution of the subgraph of the six-node with five-edge to the derived filters given in Fig. 6 (f). Similarly, the second row of the filter in stacked convolution layer, in Fig. 6 (e), whose values are "1","0","0","0","1", denotes the contribution of the subgraph of six-node ring to the derived filters, given in Fig. 6 (f). Then we stack the derived filters in Fig. 6 (i) from previous convolution layer and obtain the derived filters in the current convolution layer shown in Fig. 6 (f). The two subgraph structures obtained based on convolution operation with derived filters in Fig. 6 (f) are given in Fig. 6 (g) by only drawing the edge with weight larger than 0.

3.9 Pooling Layer

After going deeper through the m convolution layers with system supplied parameter m , we obtain the deep feature set $\mathcal{P}^0$,..., $\mathcal{P}^m$. To integrate the extracted features, the pooling layer is employed. The pooling operation, denoted as $pool(\cdot)$, is designed as a form of non-linear down-sampling. Pooling is typically performed by progressively reducing the number of parameters and thus the spatial size of the feature representation, which helps to identify the approximate location of one feature relative to the other features.

TABLE 1: Ngram CNN configuration and feature size in each layer

Schema	#Filter	Filter size	Zero-padding	Feature size
Input				$\|V\| \times \|V\|$
Diagonal Convolution	n_{f0}	$n \times n$	0	$n_{f0} \times (\|V\| - n + 1)$
Convolution Layer 1	n_{f1}	$n_{f0} \times s_1$	$s_1 - 1$	$n_{f1} \times (\|V\| - n + 1)$
Convolution Layer 2	n_{f2}	$n_{f1} \times s_2$	$s_2 - 1$	$n_{f2} \times (\|V\| - n + 1)$
Convolution Layer 3	n_{f3}	$n_{f2} \times s_3$	$s_3 - 1$	$n_{f3} \times (\|V\| - n + 1)$
...	...	...	...	...
Convolution layer m	n_{fm}	$n_{f(m-1)} \times s_m$	$s_m - 1$	$n_{fm} \times (\|V\| - n + 1)$
Hidden Layer				$\sum(n_{fi})$
Output				K

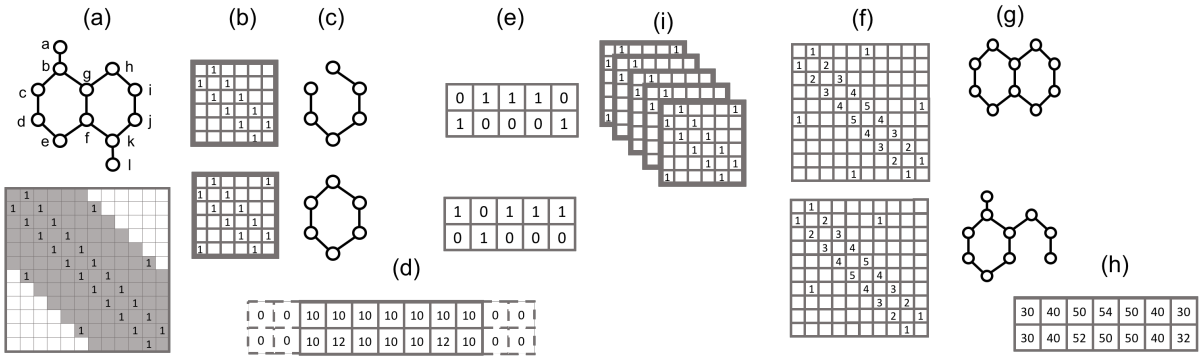


Fig. 6: (a)Example graph (b)Diagonal convolution filters (c)Corresponding subgraph structures (d)Diagonal features (e)Stacked convolution filters (f)Derived filters (g)Corresponding subgraph, and (h) High level features

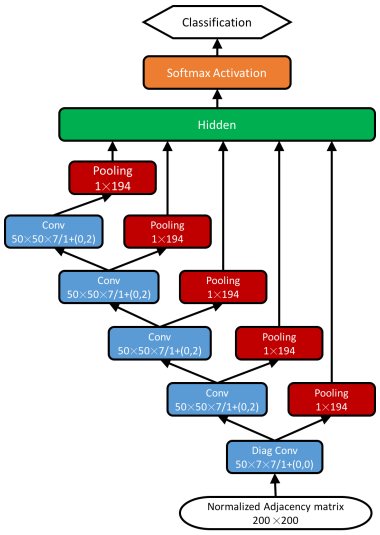


Fig. 7: The configuration of 5 convolution layers NgramCNN on graph dataset whose max node number is 200.

The two popular types of pooling methods are max pooling and average pooling. Max pooling takes the largest number from a set of input parameters as the pooling result while average pooling takes the average number from the input data. In our NgramCNN approach, max pooling is used. We add the pooling layer for each deep feature set $\mathcal{P}^i$ where i from 0 to m. For $\mathcal{P}^i$ whose size is $n_{i-1} \times (|V| - n + 1)$, we take max-pooling on each row. Therefore we get a vector of size $n_{i-1} \times 1$. Note that for diagonal convolution feature, n_0 is exactly n.

The relation between stacked convolution layers and pooling layers is illustrated in Figure 7, in which the arrow denotes the data flow direction between layers, blue layers are convolution layers, and green layers are pooling layers. This figure illustrates the configuration of a NgramCNN with five convolution layers on a graph dataset with the maximum graph size of 200 vertices. With the initial adjacency matrix is 200 by 200 in size, The normalized graphs, represented by the adjacency matrix, are taken as input for the diagonal convolution layer, whose configuration is $50 \times 7 \times 7/1 + (0, 0)$. It means 50 convolution filters with size 7×7, with stride as 1 and no zero padding (0,0). This layer sends its results to a pooling layer, whose size is 1×194. In addition, the results from diagonal convolution are also taken to the second convolution layer whose configuration is $50 \times 50 \times 7/1 + (0, 2)$. It means 50 convolution filers whose size is 50×7 and stride is 1. The zero padding is 0 on the first dimension and 2 on the second dimension. After 5 convolution layers, the dense hidden layer summarizes all of the pooling results and outputs the classification result after a softmax activation.

Note that for a graph dataset with irregular sizes of graphs, we need to find the window size n for this dataset in such a way that this parameter n is not the worst choice for this dataset. When we set n too small, it may result in many graphs requiring to employ relaxed normalization, which means that more connection structure information may get lost due to relaxed n-gram normalization. Also small n may imply that the diagonal convolution may over-fit, since less possible subgraph structure features are being captured. Therefore, we set n according to the maximum graph size, say $|V_{max}|$ instead of using the smallest graph in the dataset. For the small graph, e.g., 2 nodes graph, we append 0, make the graph size equal to $|V_{max}|$, to ensure that (a) the existing connecting information in the raw input graph is maintained, (b) the appended 0 will not destroy or change the original graph structures. A intuitive understanding of setting the parameter n is to add several independent nodes which are disconnected to the original graph. Clearly, difference datasets will have different optimal n.

3.10 Regularization and Output Layer

After several convolutional and max pooling layers, the high-level reasoning can be done via fully connected layers. Neurons in a fully connected layer have full connections to all activations in the previous layer. These activations can be computed with a matrix multiplication followed by a bias offset. The main parameters used in this hidden dense layer are weight parameters $\mathcal{W}_h$, bias parameter b_h and dropout with ratio γ. Dropout is a widely used technique in neural network to prevent overfitting [31].

In the classification output layer, we perform multinomial logistic regression through another full connection on weight parameter $\mathcal{W}_s$, bias parameter b_s and softmax function. The softmax function computes the probability distribution over the vector $\mathbf{x}$ of class labels and the input to the function is the result of K distinct linear functions, and the predicted probability for the j'th class ($j = 1, , K$) given the vector $\mathbf{x}$:

$$Pr(y = j|\mathbf{x}) = \frac{e^{x_j}}{\sum_l^K e^{x_l}} \tag{6}$$

where $K = |\mathcal{L}|$, $\mathcal{L}$ is the given set of class labels represented in a vector format, denoted by $\mathbf{x}$.

3.11 Training

The model is trained to minimize the cross-entropy cost:

$$\mathcal{C} = -log \prod_{i=1}^{|\mathcal{R}|} Pr(y_i|\mathcal{A}_i) \tag{7}$$

where $|\mathcal{R}|$ is the total number of graphs in the training set $\mathcal{R}$, $\mathcal{A}_i$ denotes the adjacency matrix of the i^{th} graph in $\mathcal{R}$, y_i denotes the i^{th} class label in $\mathbf{x}$, and θ contains all the parameters optimized by the Ngram CNN: $\theta = \{\mathcal{F}; \mathcal{W}_h; b_h\}$ $\mathcal{F} = \mathcal{F}^0, \mathcal{F}^1, ..., \mathcal{F}^m$ are the convolution filters, $\mathcal{W}_h$ and b_h are the hidden layer weight and bias. The parameters are optimized with stochastic gradient descent (SGD). The back-propagation algorithm is employed to compute the gradients. A number of speed up modifications to SGD are proposed recently, including momentum [32], Adagrad [33] and Adadelta [34]. Comparing with SGD, Adagrad scales the learning rate dynamically. Adadela uses the history gradient and weight to speed up the convergence. Although we use SGD in the first prototype implementation of our Ngram CNN, we plan to optimize SGD by incorporating and extending Adagrad.

4 Experiments

This section reports the evaluation results of our NgramCNN method by comparing it with the state-of-the art approaches to classification of graphs in terms of classification accuracy and time complexity.

4.1 Datasets

Three bioinformatics datasets: MUTAG, PTC and PROTEINS are used in our experimental evaluation. MUTAG is a dataset with 188 nitro compounds where classes indicate whether the compound has a mutagenic effect on a bacterium [1]. PTC is a dataset of 344 chemical compounds that reports the carcinogenicity for male and female rats [35]. PROTEINS is a collection of graphs, in which nodes are secondary structure elements and edges indicate neighborhood in the amino-acid sequence or in 3D space [36].

In addition, two social network datasets, IMDB-BINARY and IMDB-MULTI, are also used in our experimental comparison. IMDB-BINARY is a movie collaboration dataset where actor/actress and genre information of different movies are collected on IMDB [8]. For each graph, nodes represent actors/actress and the edge connected between them if they appear in the same movie. The collaboration network and ego-network for each actor/actress are generated. The ego-network is labeled with that the genre it belongs to. IMDB-MULTI is the multi-class version since a movie can belong to several genres at the same time. IMDB-BINARY is the binary class version which has the set of ego-networks derived from Comedy, Romance and Sci-Fi genres.

4.2 Experiment Setup

We compared NgramCNN with three state-of-art approaches:

- Deep Graph Kernel (DGK) [8] achieves the best classification accuracy over graph kernel approaches.
- PATCHY-SAN (PSCN) [13] applies CNN on graph classification and is competitive with deep graph kernel.
- Multi-task Learning (MTL) [14] takes the high quality subgraph feature for classification by learning the regularized multiple tasks.

Our approach is implemented based on the neural network framework Chainer (http://chainer.org) by Python 2.7. We obtained MTL source code from the authors and compiled it by MATLAB 2016a. We did not have the source code of DGK and PSCN. The result of DGK and the result of PSCN are the highest accuracy results reported in the literature. The execution time for DGK and PSCN is taken from [13] with 64G RAM and a single 2.8 GHz CPU as the machine configuration of their experiments. All experiments are executed on the same server with 32GB memory, 2.4 GHz Intel CPU, and NVidia GeForce 970 GPU.

For NgramCNN, we compared the performance using 2 convolution layers as the shallow configuration, denoted by NgramCNN-L2, and deep configuration of 5 convolution layers, denoted by NgramCNN-L5. We set a variable n for the n-gram normalization from 3 to 17. Also the filter size s_i used at each convolution layer is tuned from $\{3, 5, 7, 9, 11, 13, 15, 17, 19\}$. The number of convolution filters is tuned from $\{20, 30, 40, 50, 60, 70, 80\}$ at each layer. The dropout ratio is set as 0.5 and max training iteration limit is set as 200. 10-fold valification is employed with training set and testing set are randomly divided in ratio of $7 : 3$.

Given the test collection of graphs in size of N, each graph G_i with class label y_i and predicted class $\hat{y}_i$ by classifier, the accuracy measure is formalized as follows:

$$Accuracy = \frac{\sum_{i=1}^{N} \delta(y_i = \hat{y}_i)}{N} \tag{8}$$

where the indicator function $\delta(\cdot)$ gets value "1" if the condition is true, and gets value "0" otherwise.

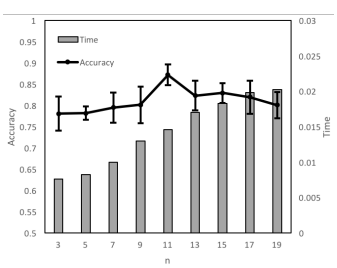
4.3 Performance Comparison Results

We first report the experimental comparison of our approach with three representative methods: DGK, PSCN and MTL. Table 2 shows the characteristics of the five datasets used in the experiments and reports the average accuracy and the standard deviation of the comparison results. All experiments were ran for ten times in the same setting.

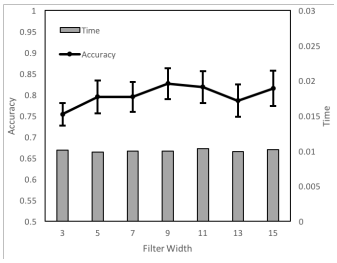
For dataset MUTAG, compared to the best result of PSCN at 92.63%, NgramCNN-L5 (5 convolution layers) obtained the accuracy of 94.99%, higher than PSCN. NgramCNN-L2 achieved

TABLE 2: properties of the datasets and accuracy for NgramCNN and three exiting state of the art approaches

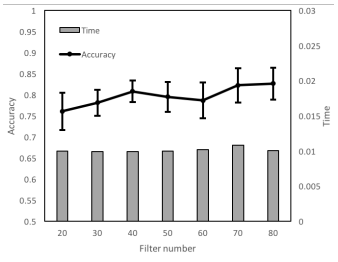
Datasets	MUTAG	PTC	PROTEINS	IMDB-BINARY	IMDB-MULTI
Graph#	188	344	1113	1000	1500
Class#	2	2	2	2	3
Max.Node#	28	109	620	136	89
Avg.Node#	17.9	25.5	39.1	19.7	13.0
DGK	$82.94 \pm 2.68(5s)$	$59.17 \pm 1.56(30s)$	$73.30 \pm 0.82(143s)$	66.96 ± 0.56	44.55 ± 0.52
PSCN	$92.63 \pm 4.21(3s)$	$60.00 \pm 4.82(6s)$	$75.89 \pm 2.76(30s)$	71.00 ± 2.29	45.23 ± 2.84
MTL	$82.81 \pm 1.22(0.006s)$	$54.46 \pm 1.45(0.045s)$	$59.74 \pm 1.06(0.014s)$	53.23 ± 1.23	33.52 ± 2.31
NgramCNN-L2	$92.32 \pm 4.10(0.01s)$	$62.50 \pm 4.51(0.10s)$	$74.99 \pm 2.13(0.39s)$	63.43 ± 2.50	46.22 ± 1.15
NgramCNN-L5	$\mathbf{94.99 \pm 5.63}(0.01s)$	$\mathbf{68.57 \pm 1.72}(0.08s)$	$\mathbf{75.96 \pm 2.98}(0.60s)$	$\mathbf{71.66 \pm 2.71}$	$\mathbf{50.66 \pm 4.10}$



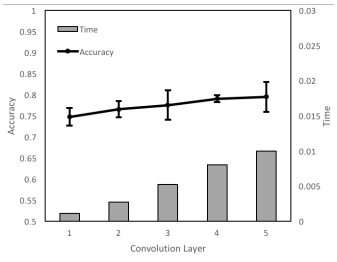
(a) Varying n on MUTAG



(b) Varying filter width on MUTAG

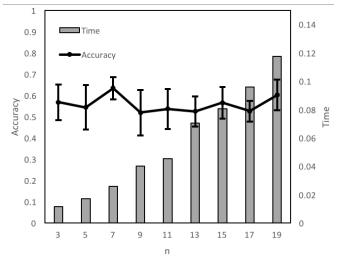


(c) Varying filter number on MUTAG

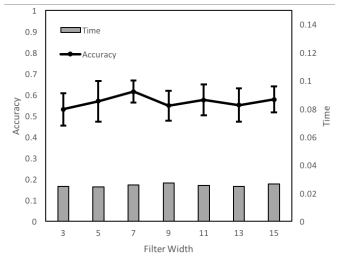


(d) Varying convolution layer number on MUTAG

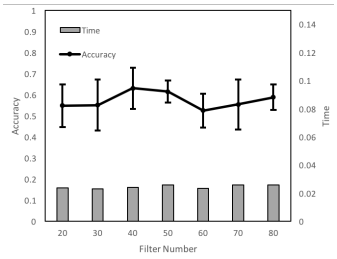
Fig. 8: The accuracy and running time on MUTAG.



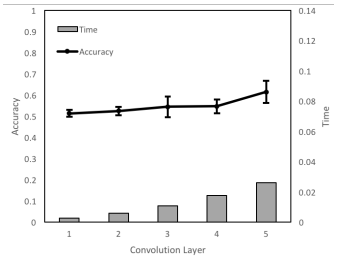
(a) Varying n on PTC



(b) Varying filter width on PTC

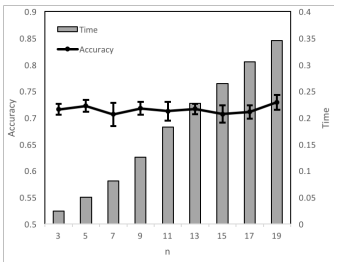


(c) Varying filter number on PTC

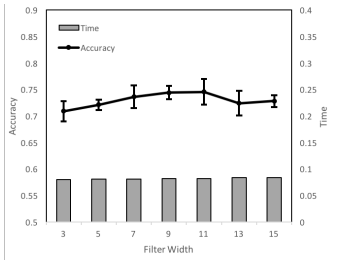


(d) Varying convolution layer number on PTC

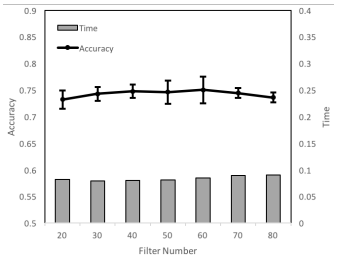
Fig. 9: The accuracy and running time on PTC.



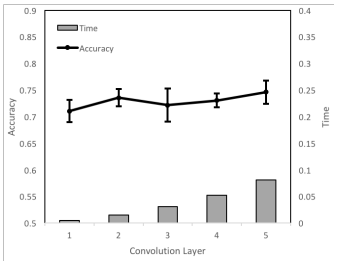
(a) Varying n on PROTEINS



(b) Varying filter width on PROTEINS



(c) Varying filter number on PROTEINS



(d) Varying convolution layer number on PROTEINS

Fig. 10: The accuracy and running time on PROTEINS.

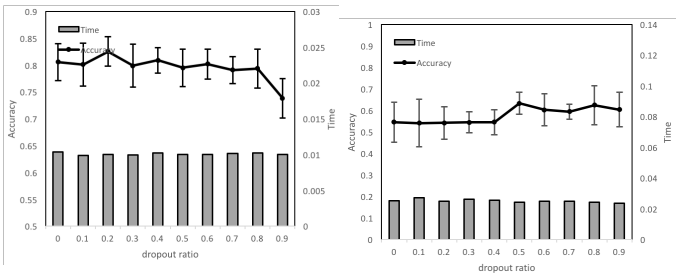
the accuracy of 92.32%, very similar to PSCN. For PTC dataset, DGK and PSCN obtained similar accuracy measure of around 60%. Our NgramCNN-L2 achieved 62.50% and NgramCNN-L5 achieved 64.99%, which is the best accuracy to date on this dataset, with the best of our knowledge. For dataset PROTEINS, NgramCNN-L5 achieved the highest accuracy of 75.96%, which is slightly higher than the best result of 75.89% by PSCN. For the two social network datasets, NgramCNN has a competitive accuracy result of 71.66% for IMDB-BINARY, higher than the best of PSCN at 71.00% and has achieved the highest accuracy of 50.66% for IMDB-MULTI, compared to the best of PSCN at 45% and the best of DGK at 44%.

4.4 Parameters Selection

In this section we study the impact of parameter configuration on the accuracy of the classification result and the time complexity performance of NgramCNN. Concretely, we examine three four configuration parameters in the NgramCNN framework:

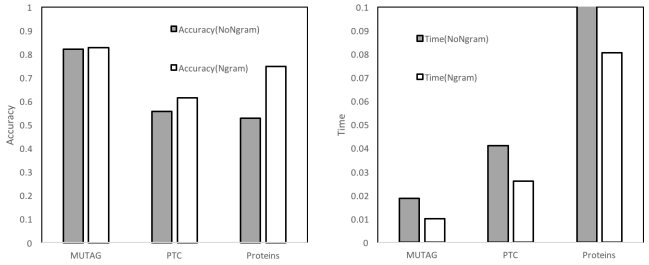
Ngram window size parameter n.

This is the key parameter for determining how good our Ngram



(a) Varying dropout ratio on Mutag (b) Varying dropout ratio on PTC

Fig. 11: The accuracy and running time on different dropout rate.



(a) Accuracy with/without ngram (b) Time with/without ngram

Fig. 12: The accuracy and running time with/without ngram.

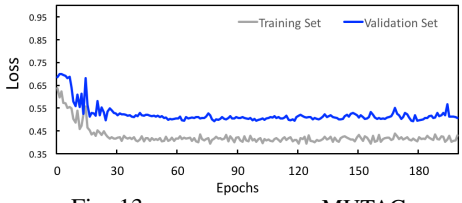


Fig. 13: convergence on MUTAG

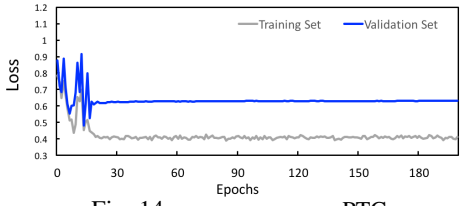


Fig. 14: convergence on PTC

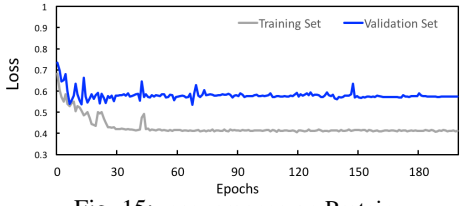


Fig. 15: convergence on Proteins

CNN model can cover the most significant subgraph patterns in the given graph dataset. Because a small n may result in the fact that most graphs would need to use the relaxed normalization to construct their n-gram normalized adjacency matrices. Consequently, we may loss more structural connectivity information, which can be critical for classification of graph dataset. On the other hand, a big n will lead to high computation cost due to large window-size challenge as mentioned in the Related work. Figure 8 (a) shows the accuracy and executing time of NgramCNN varying n on dataset MUTAG. In this experiment, the number of convolution filters is set to 50 for all experiments and the stacked convolution filter width is set to 7. The accuracy and execution time are both the average value in 10 runs with the same experimental setting. From both Figure 8 (a), Figure 9 (a) and Figure 10 (a), we observe that the accuracy is insensitive to the increase of n while the execution time is more sensitive and grows significantly as the parameter n increases from 3 to 11 for both MUTAG dataset and PTC dataset. Thus, setting smaller n is more desirable.

Stacked Convolution Filter Width s_i.

Although different convolution layers can set different filter width, in this set of experiments, we set the same width for all layers to simply the discussion. Setting a larger width s_i means that each filter can capture more complex subgraph structure features. Also the complex subgraph structure features have higher possibility in combination. However, it is also hard to determine the filter width to cover all the possible combinations. In this experiment, we set $n = 7$, filter number by 50 and vary filter width from 3 to 15. Note that due to zero-padding, we can only use the filter with odd value, namely 3,5,7,9,11,13,15. We also performed 10 runs for each measurement collected under the same setting and take the average value in accuracy and executing time. Figure 8 (b), Figure 9 (b) and Figure 10 (b) illustrate the results on MUTAG, PTC and Proteins respectively. It shows that on MUTAG, the accuracy grows as we increase filter width from 3 to 9 and become more stable as we increase the filter width from 9 to 15. This indicates that 9 is an approximately optimal setting of filter width because the running time on 9 is smaller than that on the filter width of 9 and 15. Similar to MUTAG, PTC dataset shows that the best setting of the filter width is 7, because setting filter width as 9,11 and 13 respectively gets similar accuracy but takes longer running time compared to small filter width of 7. While in Proteins dataset, namely Figure 10 (b), we can see that optimal filter width is 11.

Filter Number n_{fi}

This parameter determines how many features are captured by NgramCNN in each layer. Similar to filter width, we set the same filter number for all convolution layers, including diagonal convolution layer and stacked convolution layers. In this experiment, we set n by 7, filter width by 7 and vary filter number from 20 to 80. Each measurement is collected by 10 runs and the average value of accuracy and running time are reported. Figure 8 (c) shows the result on MUTAG and Figure 9 (c) shows the result of PTC. And Figure 10 (c) shows the result in Proteins. We make an interesting observation: a larger filter number, for example, 60 in Fig. 8 (c), may result in much worse classification accuracy for both datasets. This is because the more filters are used, the more weights need to be trained. Thus, it is easier to get overfitted in training with larger filter number.

Convolution Layer number

Like most deep neural network, the convolution layer number is sensitive to the final classification accuracy, namely how deep should the network set. As showed in Tab. 2, the NgramCNN-L5, namely 5 convolution layer version can achieve a better accuracy, comparing with NgramCNN-L2, namely 2 convolution layer version. For better observing the efficiency and effectively of our approach on different convolution layer number, we made a group of experiment varying the convolution layer from 1 to 5 on MUTAG, PTC and Proteins. Figure 8 (d), Figure 9 (d) and Figure 10 illustrate the accuracy and executing time of our approach on MUTAG, PTC and Proteins, respectively. Note that in this experiment, all other parameters are fixed as the default value. n and filter width are set as 7, filter number is set as 50. An interesting fact is that, without tuning other parameters, increasing convolution layer number will not increase the accuracy explicitly. In Figure 8 (d), the accuracy on 5-convolution layer is similar to 2-

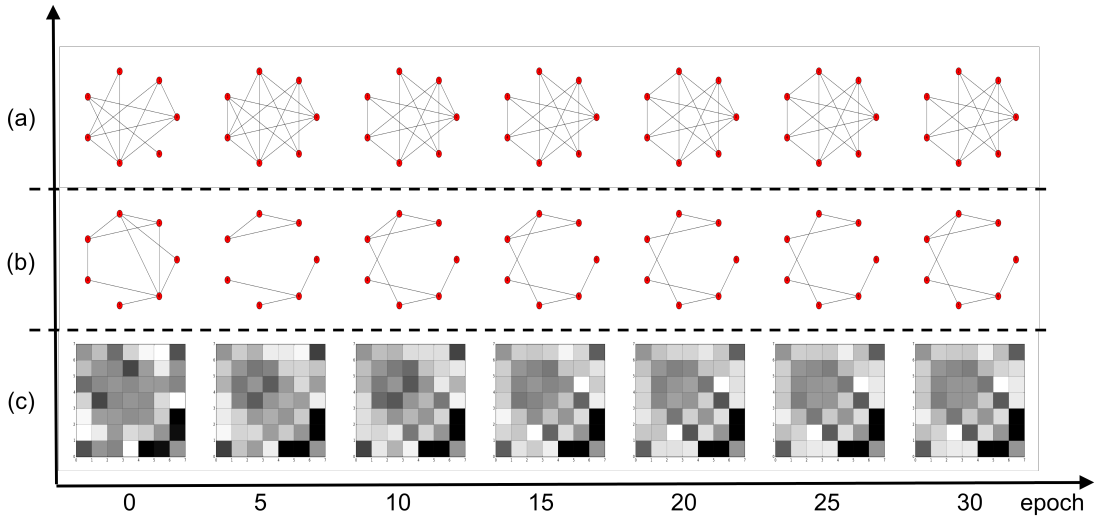


Fig. 16: (a) The positive graph structure, (b) The negative graph structure and (c) filter value.

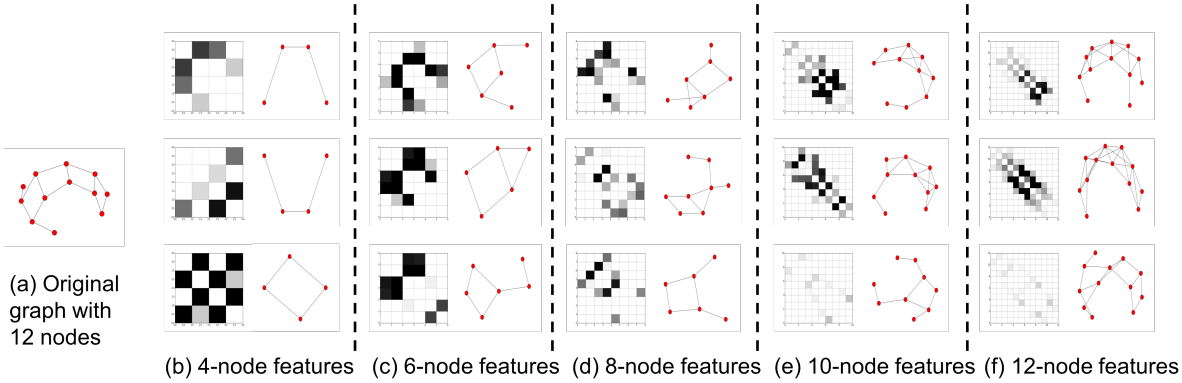


Fig. 17: (a) The original graph, (b) Features on first convolution layer, (c) Features on second convolution layer, (d) Features on third convolution layer, (e) Features on fourth convolution layer, (f) Features on fifth convolution layer.

convolution layer version. It is because without increasing the filter number and filter width, the deeper convolution network cannot take advantage of its capacity in representing more complex features. In Figure 10 (d), the accuracy on 5-convolution layer is even worse than 2-convolution layer version. It means that the current parameter setting in n, filter width and number works well on 2-convolution layer and limits the performance on 5-convolution. Therefore, in this situation, we need enlarge the other parameters for 5-convolution layer version on Proteins dataset.

Dropout Ratio

We have shown that increasing filter width, filter size and convolution layers may not improve the performance. The next set of experiments is to study the impact of overfitting by varying dropout ratios with batch normalization. Figure 11 shows the results on MUTAG and PTC. The x-axis varies the dropout ratio, the left y-axis measures the accuracy and the right y-axis measures the running time. Figure 11(a) shows that the accuracy increases when the dropout ratio is from 0 to 0.2 and the accuracy reduces when the dropout ratio is from 0.2 to 0.9 for MUTAG. Figure 11(b) shows the measurements for PTC: the accuracy is stable when dropout ratio is from 0 to 0.4, increases when the dropout ratio is from 0.4 to 0.5, and decreases slightly when the dropout ratio is from 0.5 to 0.9. This set of experiments indicates that when the dropout ratio is set to 0.2, NgramCNN get the best fit on MUTAG and the optimal dropout ratio for PTC is 0.5.

4.5 Ngram

By introducing ngram in NgramCNN, we largely reduce the parameter number and calculating complexity in nerual network part. To detailed report this contribution, we conduct the experiment by comparing the naive CNN without ngram and our NgramCNN. Specifically, in the approach without ngram, we apply the 2-dimension convolution layer on adjacent matrix directly. There are two major difference in impliment: 1. Without Ngram, the diagonal convolution layer is replaced by a 2-dimension convolution layer. 2. Without Ngram, the pooling layers are 2-dimension pooling. The configuration of the experiment is $n = 7$, filter width as 7 and filter number as 50, for both NgramCNN and naive version. The results are reported in Fig. 12b and 12a. Figure 12a is the accuracy on these two approaches. We can see that by introducing ngram, the accuracy gets higher. In Fig. 12b, the computing time of naive without ngram version is larger than NgramCNN. It means that with ngram, convolutional nerual network get a higher accuracy and lower running time.

4.6 Convergence

We report the convergence process of loss of both training set and validation set on MUTAG, PTC and Proteins, in Fig. 13,14,15. The grey line is the loss on training set and blue line is loss on validation set. We can see that in both three datasets, the loss reduces at first and get stable from 30 epochs. And just like most machine learning approaches, especially neural network, the loss

on training set can get a lower value than validation set. It is because the training procedure apply Stochastic Gradient Descent on loss of training set not the validation set.

4.7 Feature Training

In this experiment, we report a diagonal convolution filter with its value and its represented subgraph structures. The experiment is done on MUTAG dataset, with n set to 7, filter width set to 7 and filter number set to 20. We trace the filter from initial state to trained state. Note that in each epoch in the training procedure, the filter is modified slightly. Figure 16 reports the results, in which the x-axis is the epoch number from 0 to 30. Epoch = 0 means the initial value, which is sampled from a normal distribution. Figure 16 (c) shows the raw filter value, which is a 7×7 matrix. Each cell representing the corresponding position in the filter matrix. The darker the cell is, the bigger the value is. In other words, the darkest cell has the value more close to 1 while the white cell has the value more close to -1, the grey cell has the value around 0. We can see that in initial stage, more cells are grey, with values around 0. As we move forward with the training procedure, some dark cells become lighter and some light cells become darker, especially in the left top part. While the darkest cells, on the right bottom part, keep dark during the training. It means that these cells play important roles in classification of the given dataset of graphs. This is because the back propagation only modifies the cells that are non-contributing o the classification of the input graph.

For better understanding the subgraph structure, we draw the positive subgraph and negative subgraph in Fig. 16(a) and (b) respectively. The positive subgraph is drawn by setting the cell as 1 if its value is bigger than 0 and as 0 if its value is smaller or equal to 0. We call this subgraph a positive subgraph because it represents the edges that should appear. In the contrast, the negative subgraph is drawn by setting the cell as 1 if its value is smaller or equal to 0 and as 1 if its value is bigger than 0. The negative subgraph denotes the edges that should not appear. We can see that, both positive graph and negative graph do change gradually from the initial state in the training procedure and arrive at the stable structures at the end of the training. It means that the training procedure eventually reaches the convergence.

4.8 Feature Visualization.

Figure 17 illustrates the subgraph features captured in different convolution layers. Figure 17(a) presents the input graph of 12 nodes. We show that using NgramCNN, the window-size of $n = 4$ is sufficient to capture and extract the dense structures of this graph. We use NgramCNN-L5, set the diagonal convolution filer of size 4×4, the n-gram window size by $n = 4$, and set the rest 4 convolution layer filters in size of 3. Thus, the feature size in each layer is $4, 6, 8, 10, 12$. Figure 17(b),(c),(d),(e),(f) show the patterns learned at each of the five convolution layers respectively. The adjacency matrix shows the existing probability of each edge, the darker the cell is, the higher probability that the corresponding edge is captured by this filter. In the first layer shown in Figure 17(b), only the basic four node patterns can be handled. Moving forward to the second layer shown in Figure 17(c), the filters can capture and represent the six-node patterns, which are composed by the first layer features. By further adding more convolution layers, the more complicated subgraph patterns can be captured and represented. Finally, in Figure 17(f), the 12-node feature is captured, which is quite similar to the original input graph in Figure 17(a).

In summary, we demonstrate through extensive experiments with real world datasets that NgramCNN is competitive and outperforms the representative state-of-art approaches, represented by DGK, PSCN and MTL, for classification of graphs. Furthermore, the deep convolution NgramCNN-L5 outperforms the shallow NgramCNN-L2 for most datasets.

5 Conclusion

We have presented the Ngram convolutional neural network model for classification of graphs. This paper makes three original contributions: First, we introduce the concept of n-gram block to transform raw graph object into a sequence of n-gram blocks connected through overlapping regions. Second, we introduce n-gram normalization and diagonal convolution operation to extract local patterns and connectivity features hidden in the n-gram blocks. Third but not the least, we develop NgramCNN structure to extract global patterns based on the local patterns and a series of stacked convolutional layers built on top of our diagonal convolution. Our experiments show that NgramCNN outperforms existing methods with high accuracy and comparable performance.

Acknowledgment

The first author performed this work when he is a visiting PhD student at the school of Computer Science in Georgia Institute of Technology, supported by the scholarship from CSC. The second author is partially supported by NSF under Grants CNS 1115375, IIP 1230740, NSF 1547102 and SaTC 1564097. The project is supported by National Natural Science Foundation of China under Grant (No.61272129), National Science and Technology Supporting Program of China (NO. 2015BAH18F02). Dr. Jianwei Yin is the corresponding author.

References

[1] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity," *Journal of medicinal chemistry*, vol. 34, no. 2, pp. 786–797, 1991.

[2] L. Schietgat, J. Ramon, and M. Bruynooghe, "A polynomial-time maximum common subgraph algorithm for outerplanar graphs and its application to chemoinformatics," *Annals of Mathematics and Artificial Intelligence*, vol. 69, no. 4, pp. 343–376, 2013.

[3] N. Jin, C. Young, and W. Wang, "Gaia: graph classification using evolutionary computation," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 879–890.

[4] H. Saigo, S. Nowozin, T. Kadowaki, T. Kudo, and K. Tsuda, "gboost: a mathematical programming approach to graph classification and regression," *Machine Learning*, vol. 75, no. 1, pp. 69–89, 2009.

[5] X. Kong and P. S. Yu, "Semi-supervised feature selection for graph classification," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2010, pp. 793–802.

[6] J. Tang and H. Liu, "An unsupervised feature selection framework for social media data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 12, pp. 2914–2927, 2014.

[7] S. Pan, J. Wu, and X. Zhu, "Cogboost: boosting for fast cost-sensitive graph classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 11, pp. 2933–2946, 2015.

[8] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1365–1374.

[9] H. Kashima, K. Tsuda, and A. Inokuchi, "Kernels for graphs," *Kernel methods in computational biology*, vol. 39, no. 1, pp. 101–113, 2004.

[10] K. Riesen and H. Bunke, "Graph classification by means of lipschitz embedding," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 6, pp. 1472–1483, 2009.

[11] A. Narayanan, M. Chandramohan, L. Chen, Y. Liu, and S. Saminathan, "subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs," *arXiv preprint arXiv:1606.08928*, 2016.

[12] N. Shervashidze and K. M. Borgwardt, "Fast subtree kernels on graphs." in *NIPS*, 2009, pp. 1660–1668.

[13] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proceedings of the 33nd International Conference on Machine Learning, ICML*, 2016, pp. 2014–2023.

[14] S. Pan, J. Wu, X. Zhu, C. Zhang, and S. Y. Philip, "Joint structure feature exploration and regularization for multi-task graph classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 3, pp. 715–728, 2016.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[16] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[17] X. Zheng, H. Peng, Y. Chen, P. Zhang, and W. Zhang, "Character-based parsing with convolutional neural network," in *Proceedings of the 24th International Conference on Artificial Intelligence*. AAAI Press, 2015, pp. 1054–1060.

[18] M. L.-J. Yann and Y. Tang, "Learning deep convolutional neural networks for x-ray protein crystallization image analysis," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[19] S. Hong, T. You, S. Kwak, and B. Han, "Online tracking by learning discriminative saliency map with convolutional neural network," in *Proceedings of the 32nd International Conference on Machine Learning, ICML*, 2015, pp. 597–606.

[20] M. Liang, X. Hu, and B. Zhang, "Convolutional neural networks with intra-layer recurrent connections for scene labeling," in *Advances in Neural Information Processing Systems*, 2015, pp. 937–945.

[21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[22] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in Neural Information Processing Systems*, 2015, pp. 2224–2232.

[23] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.

[24] X. Liang, X. Shen, J. Feng, L. Lin, and S. Yan, "Semantic object parsing with graph lstm," in *European Conference on Computer Vision*. Springer, 2016, pp. 125–143.

[25] Y. Zhou, L. Liu, and D. Buttler, "Integrating vertex-centric clustering with edge-centric clustering for meta path graph analysis," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1563–1572.

[26] Y. Zhou and L. Liu, "Social influence based clustering of heterogeneous information networks," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013, pp. 338–346.

[27] Z. Zeng, J. Wang, L. Zhou, and G. Karypis, "Coherent closed quasi-clique discovery from large dense graph databases," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 797–802.

[28] S. Awodey, "Isomorphisms," 2006.

[29] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[30] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin, "Convolutional neural networks over tree structures for programming language processing," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[31] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[32] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning." in *Proceedings of the 30th International Conference on Machine Learning (ICML)*, vol. 28, 2013, pp. 1139–1147.

[33] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[34] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[35] H. Toivonen, A. Srinivasan, R. D. King, S. Kramer, and C. Helma, "Statistical evaluation of the predictive toxicology challenge 2000–2001," *Bioinformatics*, vol. 19, no. 10, pp. 1183–1193, 2003.

[36] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. suppl 1, pp. i47–i56, 2005.

Zhiling Luo is a Research Associate in Computer Science at Zhejiang University, China. He received his B.S. and Ph.D. degree in Computer Science from Zhejiang University in 2012 and 2017, respectively. He was the visiting scholar of Georgia Institute of Technology, US, in 2016. His research interests include service computing, machine learning and data mining.

Ling Liu is a Professor in the School of Computer Science at Georgia Institute of Technology. She directs the research programs in Distributed Data Intensive Systems Lab (DiSL). Prof. Liu is an elected IEEE Fellow, a recipient of IEEE Computer Society Technical Achievement Award in 2012, and a recipient of the best paper award from a dozen of top venues, including ICDCS 2003, WWW 2004, 2005 Pat Goldberg Memorial Best Paper Award, IEEE Cloud 2012, IEEE ICWS 2013, ACM/IEEE CCGrid 2015, IEEE Symposium of BigData 2016. In addition to service as general chair and PC chairs of numerous IEEE and ACM conferences in data engineering, Prof. Liu has served on editorial board of over a dozen international journals. Prof. Liu's current research is primarily sponsored by NSF, IBM and Intel.

Jianwei Yin is currently a professor in the College of Computer Science, Zhejiang University, China. He received his Ph.D. in Computer Science from Zhejiang University in 2001. He is the visiting scholar of Georgia Institute of Technology, US, in 2008. His research interests include service computing, cloud computing and information integration. Currently Prof. Yin is the AE of IEEE Transactions on Service Computing.

Ying Li received the B.S., M.S. and Ph.D. degrees in computer science from Zhejiang University, China, in 1994, 1997 and 2000, respectively. He is currently an associate professor with the College of Computer Science, Zhejiang University. He is currently leading some research projects supported by National Natural Science Foundation of China and National High-tech RD Program of China (863 Program). His research interests include service computing, business process management and compiler.

Zhaohui Wu received the B.S. and Ph.D. degrees in computer science from Zhejiang University, Hangzhou, China, in 1988 and 1993, respectively. He is currently a Professor with the College of Computer Science, Zhejiang University. His research interests include distributed artificial intelligence, semantic grid, and pervasive computing. Dr. Wu is a Standing Council Member of the China Computer Federation.