DDGK: Learning Graph Representations for Deep Divergence Graph Kernels

Rami Al-Rfou
Google AI
Mountain View, CA
rmyeid@google.com

Dustin Zelle
Google AI
New York, NY
dzelle@google.com

Bryan Perozzi
Google AI
New York, NY
bperozzi@acm.org

ABSTRACT

Can neural networks learn to compare graphs without feature engineering? In this paper, we show that it is possible to learn representations for graph similarity with neither domain knowledge nor supervision (i.e. feature engineering or labeled graphs). We propose Deep Divergence Graph Kernels, an unsupervised method for learning representations over graphs that encodes a relaxed notion of graph isomorphism. Our method consists of three parts. First, we learn an encoder for each anchor graph to capture its structure. Second, for each pair of graphs, we train a cross-graph attention network which uses the node representations of an anchor graph to reconstruct another graph. This approach, which we call *isomorphism attention*, captures how well the representations of one graph can encode another.

We use the attention-augmented encoder's predictions to define a divergence score for each pair of graphs. Finally, we construct an embedding space for all graphs using these pair-wise divergence scores.

Unlike previous work, much of which relies on 1) supervision, 2) domain specific knowledge (e.g. a reliance on Weisfeiler-Lehman kernels), and 3) known node alignment, our unsupervised method jointly learns node representations, graph representations, and an attention-based alignment between graphs.

Our experimental results show that Deep Divergence Graph Kernels can learn an unsupervised alignment between graphs, and that the learned representations achieve competitive results when used as features on a number of challenging graph classification tasks. Furthermore, we illustrate how the learned attention allows insight into the the alignment of sub-structures across graphs.

KEYWORDS

Graph Kernels; Graph Neural Networks; Representation Learning; Similarity and Search

ACM Reference Format:
Rami Al-Rfou, Dustin Zelle, and Bryan Perozzi. 2019. DDGK: Learning Graph Representations for Deep Divergence Graph Kernels. In *Proceedings of the 2019 World Wide Web Conference (WWW'19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3308558.3313668

WWW '19, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.3313668

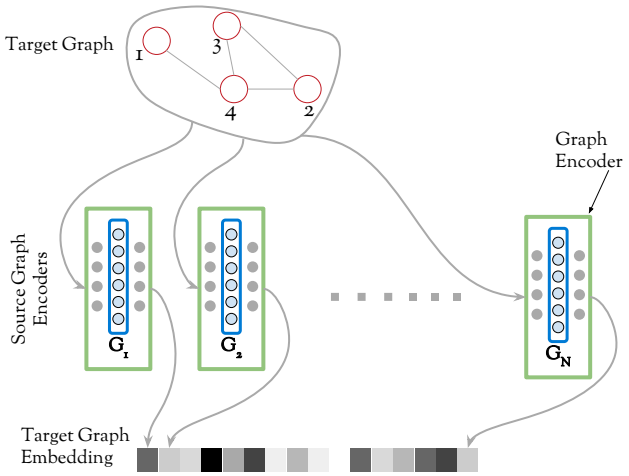


Figure 1: Our method of learning graph representations by measuring the divergence of a target graph across a population of source graph encoders. First, we train a graph encoder for each graph in our source graph population $\{G_1, G_2, ..., G_N\}$. Second, for each of these encoders we measure the divergence of the target graph from the associated source graph. Finally, these divergence scores are used to compose the vector representation of the target graph.

1 INTRODUCTION

Deep learning methods have achieved tremendous success in domains where the structure of the data is known a priori. For example domains like speech and language have intrinsic sequential structure to exploit, while computer vision applications have spatial structure (images) and perhaps temporal structure (videos). In all these cases, our intuition guides us to build models and learning algorithms based on the structure of the data. For example, translation invariant convolution networks might search for shapes regardless of their physical position in an image, or recurrent neural networks might share a common latent representation of a concept across distant time steps or diverse domains such as languages. In contrast, graph learning represents a more general class of problems because the structure of the data is free from any constraints. A neural network model must learn to solve both the desired task at hand (e.g. node classification) and to represent the structure of the problem itself – that of the graph's nodes, edges, attributes, and communities.

Despite the challenges, there has been a recent surge of interest in applying neural network models to such graph-structured

data [26, 30, 40, 50, 63]. While initial approaches like DeepWalk [40] focused on generic representations of graph primitives (e.g. a graph's nodes [40] or edges [1]), present approaches ignore learning general graph and node representations in favor of maximizing accuracy on a set of narrow classification tasks. These approaches, broadly referred to as Graph Neural Networks (GNNs), seek to leverage the structure between data items as a scaffolding to perform computation (e.g. message passing, gradient updates, etc). The parameters and the activations, use the structure during training, but are tuned primarily to classify the graph's nodes, edges, and/or attributes.

While much effort has focused on unsupervised learning of node representations [12, 24, 40, 48], edge representations [1], or latent community structure [13, 52, 62], relatively little work has focused on the unsupervised learning of representations for entire graphs – a problem of practical interest in domains such as information retrieval, biology, and natural language processing [5, 23]. In cases where GNNs have been applied to the task of learning similarity between graphs, the approaches considered generally come in two flavors: an end-to-end *supervised graph classification* or *graph representation learning*.

In supervised graph classification, the task is to solve an end-to-end whole-graph classification problem (i.e. the problem of assigning a label to the entire graph). These supervised approaches [33, 36, 45, 61] learn an intermediate representation of an entire graph as a precondition in order to solve the classification task. This learned representation can be used to compare similarity between graphs, but is heavily biased towards maximizing performance on the classification task of interest.

The second class of approaches focuses on the more general problem of learning graph representations [43]. While much exciting progress has been made in this area, the existing approaches suffer from one or more of the following limitations. First, many existing methods rely on feature engineering, such as the graph's clustering coefficient, its motif distribution, or its spectral decomposition, to represent graphs [6, 47, 56]. By limiting the features that they consider, these methods are limited to composing only known graph signals. Second, many of these approaches [36, 61] have sought to encode algorithmic heuristics from the graph isomorphism literature (especially the intuition encoded in the Weisfeiler Lehman algorithm [42]). Relying heavily on existing heuristics to solve a hard problem raises an important question: how well can a learning-only approach solve a classic algorithmic problem? Finally, other work in this area of graph similarity assumes that identical nodes in both graphs share the same id (i.e. the alignment is already given). While this can be useful for calculating a similarity score, we find the general problem more compelling.

In this work, we propose a method of learning graph representations driven by the similarity between a pair of graphs as measured by the divergence in their structures. We show the representations learned through our method, Deep Divergence Graph Kernels (DDGK), capture the attributes of graphs by using them as features for several classification problems. In addition, we show that our representations capture the local similarity of graph pairs and the global similarity across families of graphs.

DDGK has three key differentiators. First, it makes no assumptions about the structure of the matching problem. In order to solve the matching problem, we propose an attention mechanism: *isomorphism attention* to align the nodes across graph pairs. Second, DDGK does not rely on any existing heuristics for graph similarity. Instead, we learn the kernel method jointly with the node representation and alignment networks. This allows the model the freedom to learn representations that best preserve the graph, and does not impose artificial oversights. Finally, as an unsupervised method, the representations it learns emphasize structural similarity, and does not correlate with a downstream labeling tasks. This is especially useful for ranking tasks where labeling may not be available. To summarize, our main contributions are:

- **Deep Divergence Graph Kernels**: A novel method for learning unsupervised representations of graphs and their nodes. Our kernel is learnable and does not depend on feature engineering or domain knowledge.
- **Isomorphism Attention**: A cross-graph attention mechanism to probabilisticly align representations of nodes between graph pairs. These attention networks allow for great interpretablity of graph structure and discoverablilty of similar substructures.
- **Experimental results**: We show that DDGK both encodes graph structure to distinguish families of graphs, and when used as features, the learned representations achieve competitive results on challenging graph classification problems like predicting the functional roles of proteins.

2 LEARNING GRAPH REPRESENTATIONS

In this section, we lay out the problem definition of representing graphs and the connection between our representations and the kernel framework.

2.1 Problem Definition

A graph is defined to be a tuple $G = (V, E)$, where V is the set of vertices and E is the set of edges, $E \subseteq (V \times V)$. A graph G can have an attribute vector Y for each of its nodes or edges. We denote the attributes of node v_i as y_i, and denote the attributes of an edge (v_i, v_j) as y_{ij}.

Given a family of graphs $G_0, G_1, \ldots, G_N$ we aim to learn a continuous representation for each graph $\Psi(G) \in \mathbb{R}^N$ that encodes its attributes and its structure. For this representation to be useful, it has to be comparable to other graph representations. However, it is likely that our method of graph encoding will produce one of many equally good representations each time we run it. For example we can get two different, but equal, representations by permuting the dimensions of the first one. Those representations are not comparable given they exist in two different spaces.

To avoid this problem, we seek to develop an equivalence class across all possible encodings of a graph. Essentially, two encodings of a graph are equivalent if they lead to the same pair-wise similarity scores when used to compare the graph to all other graphs in the set. We note that this issue arises when working with embedding based representations across domains, and several equivalence methods have been proposed [3, 58].

2.2 Embedding Based Kernels

In this work, we study the development of graph kernels, which are functions to compute the pairwise similarity between graphs. Specifically, given two graphs G_1, G_2, a classic example of a kernel defined over graph pairs is the geometric random walk kernel [9] as shown in Eq. 1:

$$k_\times(G_1, G_2) = e^T(I - \lambda A_\times)^{-1}e, \tag{1}$$

where $A_\times$ is the adjacency matrix of the product graph of G_1 and G_2, and λ is a hyper-parameter which encodes the importance of each step in the random walk. We aim to learn an embedding based kernel function $k()$ as a similarity metric for graph pairs, defined as the following:

$$k(G_1, G_2) = ||\Psi(G_1) - \Psi(G_2)||^2 \tag{2}$$

For a dataset of N *source*[1] graphs $\mathcal{S}$ and M *target* graphs ($\mathcal{T}$), for any member of the target graph set we define the i^{th} dimension of the representation $\Psi(G \in \mathcal{T}) \in \mathbb{R}^N$ to be:

$$\Psi(G)_i = \sum_{v_j \in V_T} f_{g_i}(v_j), \tag{3}$$

where $g_i \in \mathcal{S}$ and $f_{g_i}()$ is a predictor of some structural property of the graph G but parameterized by the graph g_i. We note that the source and target graphs sets $(\mathcal{S}, \mathcal{T})$ could be disjoint, overlapping, or equal.

3 LEARNING TO ALIGN GRAPH REPRESENTATIONS

We propose to learn a graph representation by comparing it to a population of graphs. To compare the similarity of a pair of graphs (*source*, *target*), we rely on deep neural networks to measure the divergence between their structure and attributes. First, we learn the structure of the source graph by passing it through a graph encoder. Second, to measure how much the target graph diverges from the source graph, we use the source graph encoder to predict the structure of the target graph. If the pair is similar, we expect the source graph encoder to correctly predict the target graph's structure. In this section, we develop the three key components necessary to learn the similarity between a pair of graphs.

First, in Section 3.1, we discuss encoding graphs. The quality of the graph representation depends on the extent to which the encoder of each source graph is able to discover its structure.

Second, in Section 3.2, we propose a cross-graph attention mechanism to learn a soft alignment between graphs. This is necessary because a target graph may not share its vertex ids with any of the source graphs – indeed, they could even have differing number of nodes! Therefore, we need to learn an alignment between the nodes of the target graph and each source graph. This leads to an alignment that is not necessarily a one-to-one correspondence.

Third, in Section 3.3 we introduce additional constraints on the cross-graph attention learning. For example, let us assume that $v_i \in V_{G_1}$ is assigned to $u_j \in V_{G_2}$. While both v_i and u_j may be structurally similar, they may belong to different node classes as indicated by their attributes. These attributes may be of significant

[1]In this paper, we use source and anchor interchangeably when referring to the encoded graph.

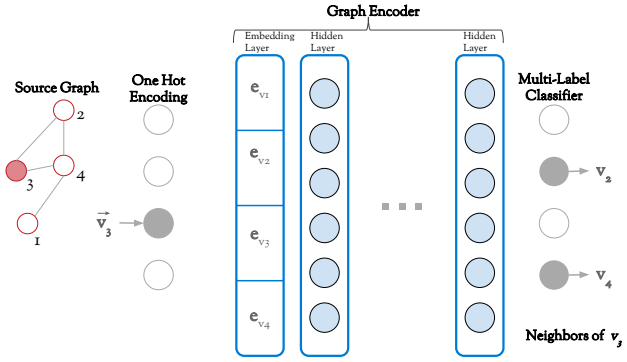


Figure 2: A Node-To-Edges Encoder. Here the input graph contains 4 vertices, and the encoder has to predict the neighbors of vertex v_3. First, v_3 is represented by a one-hot encoding $\vec{v}_3$. Second, $\vec{v}_3$ is multiplied by a linear embedding layer. Third, this embedding $\mathbf{e}_{v_3}$ is passed to a DNN which produces scores for each vertex in V. Finally, these scores are normalized using the *sigmoid* function to produce the final predictions, in this case, $\{v_2, v_4\}$.

importance to the nature of the graph. For instance, swapping one element for another in a graph representing a molecule could drastically change its chemical structure.

We will see how these pairwise alignments can produce divergence scores suitable for Graph Kernels in Section 4.

3.1 Graph Encoding

To learn the structure of a graph, we train an encoder capable of reconstructing such structure given partial or distorted information. In this paper, we choose a *Node-To-Edges* encoder (Figure 2) for its simplicity, but we note that additional choices are certainly possible (see Section 7 for more discussion).

Node-To-Edges Encoder. - In this setup, an encoder is given a single vertex and it is expected to predict its neighbors. This can be modeled as a multilabel classification task since the predictions are not mutually exclusive. Specifically, we are maximizing the following objective function $J(\theta)$,

$$J(\theta) = \sum_i \sum_{\substack{j \\ e_{ij} \in E}} \log \Pr(v_j \mid v_i, \theta). \tag{4}$$

Each vertex v_i in the graph is represented by one-hot encoding vector $\vec{v}_i$. Then to embed the vertex we multiply its encoding vector with a linear layer $\mathbf{E} \in \mathbb{R}^{|V| \times d}$ resulting in an embedded vertex $\mathbf{e}_{v_i} \in \mathbb{R}^d$, where $|V|$ is the number of vertices in the graph, and d is the size of the embedding space.

For graphs with a large number of nodes, we can replace this multiplication with a table lookup, extracting one row from the embedding matrix. This embedding vector represents the feature set given to the encoder tasked with predicting all adjacent vertices. Our encoder H, is implemented as a fully connected deep neural network (DNN) with an output layer of size $|V|$ and trained as a multilabel classifier.

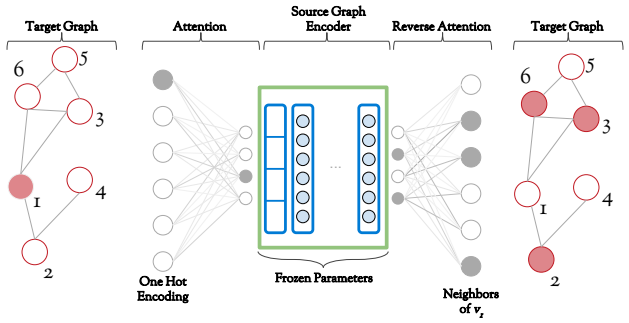


Figure 3: Attention layers mapping the target graph nodes onto the source graph. The augmented encoder has to predict the neighbors of node 1 in the target graph. First, node 1 is passed to the attention layer which assigns it mainly to node 3 of the source graph. Second, the source graph encoder learned earlier (in Figure 2) that the neighbors of node 3 are {2, 4}. Finally, the reverse attention network maps nodes {2, 4} of the source graph to nodes {2, 3, 6} of the target graph which are the neighbors of node 1.

3.2 Cross-Graph Attention

So far, we have developed a utility to encode individual graphs. However, we seek to develop a method which can compare pairs of graphs, which may differ in size (differing node sets) and structure (differing edge sets). For this to happen we need a method of learning an alignment between the graphs. Ideally this method will operate in the absence of a direct mapping between nodes.

In other areas, attention models have been proposed to align structured data. For example, attention models have been proposed to align pairs of images and text [55], pairs of sentences for translation [49], and pairs of speech and transcription [16]. Inspired by these efforts, we formalize the problem of aligning two graphs as that of attention. We propose an attention mechanism, *isomorphism attention*, that aligns the nodes of a target graph against those of a source graph.

3.2.1 Isomorphism Attention. Given two graphs S (*source graph*) and T (*target graph*), we propose a model that allows bi-directional mapping across the pair's nodes. This requires two separate attention networks. The first network allows nodes in the target graph to *attend* to the nodes in the source graph. The second network, allows neighborhood representations in the source graph to *attend* to neighborhoods in the target graph.

We denote the first attention network as $(\mathcal{M}_{T\to S})$, which assigns every node in the target graph $(u_i \in T)$ a probability distribution over the nodes of the source graph $(v_j \in S)$. This attention network will allow us to pass the nodes of the target graph as an input to the source graph encoder. We implement this attention network using a multiclass classifier,

$$\Pr(v_j \mid u_i) = \frac{e^{\mathcal{M}_{T\to S}(v_j, u_i)}}{\sum_{v_k \in V_S} e^{\mathcal{M}_{T\to S}(v_k, u_i)}}. \tag{5}$$

The second network is a *reverse attention* network $(\mathcal{M}_{S\to T})$ which aims to learn how to map a neighborhood's representation in the source graph to a neighborhood in the target graph. By adding both attention networks to the source graph encoder, we will be able to construct a target graph encoder that is able to predict the neighbors of each node – but utilizing the structure of the source graph. We implement the reverse attention as a multilabel classifier,

$$\Pr(u_j \mid \mathcal{N}(v_i)) = \frac{1}{1 + e^{-\mathcal{M}_{S\to T}(u_j, \mathcal{N}(v_i))}}. \tag{6}$$

Figure 3 shows the attention network $(\mathcal{M}_{T\to S})$ receiving a one-hot encoding vector representing a node (u_i) in the target graph and mapping it onto the most structurally similar node (v_j) from the source graph. The source graph encoder, then, predicts the neighbors of v_j, $\mathcal{N}(v_j)$. The *reverse attention* network $(\mathcal{M}_{S\to T})$, takes $\mathcal{N}(v_j)$ and maps them to the neighbors of u_i, $\mathcal{N}(u_i)$.

Both attention networks may be implemented as linear transformations $\mathbf{W}_A \in \mathbb{R}^{|V_Q|\times|V_P|}$. In the case that either $|V_P|$ or $|V_Q|$ are prohibitively large, the attention network parameters can be decreased by substituting a DNN with hidden layers of fixed size. This will reduce the attention network size from $\Theta(|V_P| \times |V_Q|)$ to $\Theta(|V_P| + |V_Q|)$.

3.3 Attributes Consistency

Labeled graphs are not defined only by their structures, but also by the attributes of their nodes and edges. The attention network assigns each node in the target graph a probability distribution over the nodes of the source graph. There might be several, equally good, nodes in the source graph with similar structural features. However, these nodes may differ in their attributes. To learn an alignment that preserves nodes and edges attributes, we add regularizing losses to the attention and reverse-attention networks.

More specifically, we refer to the nodes as v and u for the source and target graphs, respectively. We refer to the set of attributes as $\mathcal{Y}$ and the distribution of attributes over the graph nodes as $(Q_n = \Pr(y_i \mid u))$. Given that the attention network $\mathcal{M}_{T\to S}$ learns the distribution $\Pr(u_k \mid v_j)$, we can calculate a probability distribution over the attributes as inferred by the attention process as the following:

$$Q_n(y_i|u_j) = \sum_k \mathcal{M}_{T\to S}(y_i|v_k) \Pr(v_k \mid u_j). \tag{7}$$

We define, the attention regularizing loss over the nodes attributes to be the average cross entropy loss between the observed distribution of attributes and the inferred one (See Eq. 8).

$$L = \frac{1}{|V_T|} \sum_j^{|V_T|} \sum_i \Pr(y_i \mid u_j) \log(Q_n(y_i|u_j)), \tag{8}$$

where $|V_T|$ is the number of nodes in the target graph.

For preserving edge attributes over nodes, we define $Q_e(y_i \mid u) = \Pr(y_i \mid u)$ to be the normalized attributes count over all edges connected to the node u. For instance, if a node u has 5 edges with 2 of them colored red and the other three colored yellow, $Q_e(red \mid u) = 0.4$ By replacing Q_n with Q_e in Equations 7 and 8, we create a regularization loss for edge attributes.

We also introduce these regularization losses for *reverse attention* networks. Reverse attention networks maps a neighborhood

in the source graph to a neighborhood in the target graph. The distribution of attributes over a node's neighborhood will be the frequency of each attribute occurrence in the neighborhood normalized by the number of attributes appearing in the neighborhood. For edges, the node's neighborhood edges are the edges appearing at 2-hops distance from the node. Similarly, we can define the probability of the edges attributes by normalizing their frequencies over the total number of attributes of edges connected to the neighborhood.

4 DEEP DIVERGENCE GRAPH KERNELS

So far, we have proposed a method for learning representations of graphs, and an attention mechanism for aligning graphs based on a set of encoded graph representations. Here we discuss our proposed method for using the alignment to construct a graph kernel based on divergence scores. First, in Section 4.1, we show how we can utilize the divergence scores to construct a full graph representation. Divergence is driven by the target graph structure and attribute prediction error as calculated using a source graph encoder. Next we introduce DDGK, our method for learning graph representations for Deep Divergence Graph Kernels in Section 4.3. Then in Section 4.4 we discuss how we train these representations. Finally we discuss the scalability of this approach in Section 4.5.

4.1 Graph Divergence

In Section 3 we presented a method to align two graphs by using a source graph encoder, augmented with attention layers, to encode a target graph. Here, we propose to use the ability of the augmented encoder at predicting the structure of the target graph as a measure of those graphs similarity. To explain, let us assume the trivial case where both the source and target graphs are identical. First, we train the source graph encoder. Second, we augment it with attention networks and train it to predict the structure of the target graph. The attention networks will (ideally) learn the identity function. Therefore, the source graph encoder is able to encode the target graph as accurately as encoding itself. We would reasonably conclude that these graphs are similar.

We aim to learn a metric that measures the divergence score between a pair of graphs $\{S, T\}$. If two graphs are similar, we expect their divergence to be correspondingly low. We refer to the encoder trained on a graph S as H_S and the divergence score given to the target graph T to be

$$\mathcal{D}'(T \parallel S) = \sum_{v_i \in V_T} \sum_{\substack{j \\ e_{ji} \in E_T}} -\log \Pr(v_j \mid v_i, H_S) \tag{9}$$

Given that H_S is not a perfect predictor of the graph S structure, we can safely assume that $\mathcal{D}'(S \parallel S) \neq 0$. To rectify this problem we define

$$\mathcal{D}(S \parallel T) = \mathcal{D}'(S \parallel T) - \mathcal{D}'(S \parallel S), \tag{10}$$

which sets $\mathcal{D}(S \parallel S)$ to zero.

We note that this definition is not symmetric (as $\mathcal{D}(T \parallel S)$ might not necessarily equal to $\mathcal{D}(S \parallel T)$). If symmetry is required, we can define $\mathcal{D}(S, T) = \mathcal{D}(S \parallel T) + \mathcal{D}(T \parallel S)$.

input : Set of N source graphs $\mathcal{S}$
Set of M target graphs $\mathcal{T}$
Learning rate α
Encoding epochs τ
Scoring epochs ρ
output: All graph representations $\Psi \in \mathbb{R}^{M \times N}$

1 // *learn graph encodings*
2 **foreach** $g_i \in \mathcal{S}$ **do**
3 $V, E \leftarrow g_i$
4 **for** *step* $\leftarrow 0$ **to** τ **do**
5 $J(\theta) = -\sum_s \sum_{\substack{t \\ e_{st} \in E}} \log \Pr(v_t \mid v_s, \theta)$
6 $\theta = \theta - \alpha * \frac{\partial J}{\partial \theta}$
7 **end**
8 encodings$[i] \leftarrow \theta$
9 **end**
10 **foreach** $g_i \in \mathcal{T}$ **do**
11 $V, E \leftarrow g_i$
12 **foreach** $\theta_j \in$ encodings **do**
13 // *learn cross-graph attention* $\mathcal{M}_{T \to S}$ *and* $\mathcal{M}_{T \to S}$
14 **for** *step* $\leftarrow 0$ **to** ρ **do**
15 $J(\mathcal{M}_{T \to S}, \mathcal{M}_{S \to T}) =$
 $-\sum_s \sum_{\substack{t \\ e_{st} \in E}} \log \Pr(v_t | v_s, \theta_j, \mathcal{M}_{T \to S}, \mathcal{M}_{S \to T})$
16 $\mathcal{M}_{T \to S} = \mathcal{M}_{T \to S} - \alpha * \frac{\partial J}{\partial \mathcal{M}_{T \to S}}$
17 $\mathcal{M}_{S \to T} = \mathcal{M}_{S \to T} - \alpha * \frac{\partial J}{\partial \mathcal{M}_{S \to T}}$
18 **end**
19 // *calculate graph divergences*
20 $\Psi[i, j] \leftarrow J(\mathcal{M}_{T \to S}, \mathcal{M}_{S \to T})$
21 **end**
22 **end**
23 **return** Ψ

Algorithm 1: DDGK: An unsupervised algorithm for learning graph representations.

4.2 Graph Embedding

Given a set of source graphs, we can establish a vector space where each dimension corresponds to one graph in the source set. Target graphs are represented as points in this vector space where the value of the i_{th} dimension for a given target graph T_j is $\mathcal{D}(T_j \parallel S_i)$.

More formally, for a set of N source graphs we can define our target graph representation to be:

$$\Psi(G_T) = [\mathcal{D}(T \parallel S_0), \mathcal{D}(T \parallel S_1), \ldots, \mathcal{D}(T \parallel S_N)] \tag{11}$$

To create a kernel out of our graph embeddings, we use the Euclidean distance measure as outlined in Eq 2. This distance measure will guarantee a positive definite kernel [25, 54].

4.3 Algorithm : DDGK

We present pseudo-code for DDGK in Algorithm 1. The algorithm has two parts. First, a *Node-To-Edges* encoder is trained for all source graphs (Algorithm 1 line 5 and line 6). Second, cross-graph attentions are learned for all target-source graph pairs (Algorithm 1 line 15, line 16 and line 17). We implement DDGK using a deep

neural network for its *Node-To-Edges* encoder and linear transformations for its isomorphism attention.

4.4 Training

We implement our models using TensorFlow [44], calculate our gradients using backpropagation, and update our parameters using Adam [29]. We train each source graph on its adjacency matrix for a constant number of iterations.

4.4.1 Target Graph Encoding. Here, the augmented encoder has to predict the neighboring vertices for each vertex in the target graph with the help of the attention and reverse-attention layers. To learn the augmented target graph encoder (which consists of the source graph encoder with the additional attention layers), we use the following procedure:

(1) First, freeze the parameters of the source graph encoder.
(2) Second, add two additional networks, one for attention and another for reverse attention mapping between the target graph nodes to the source graph nodes and vice versa.
(3) Third, add the regularizing losses to preserve the nodes or edges attributes if they are available.
(4) Fourth, train the augmented encoder on the input, which is the adjacency matrix of the target graph, and a node attribute and/or edge attribute matrix (if available).

Finally, once the training of the attention layers is done, we use the augmented encoder to compute the divergence between the graph pair as discussed in 4.1.

4.5 Scalability

We start by defining the following quantities: N the number of source graphs in the dataset, M the number of target graphs in the dataset, V the average number of nodes, τ the number of epochs to encode source graphs, ρ the number of epochs to encode target graphs, l the number of encoder hidden layers, m the number of attention hidden layers, and d the embedding and hidden layer size

Our method relies on pairwise similarity, therefore, we will have $M \times N$ computations that each involves scoring a target graph against one source graph. Training a source graph encoder requires τ steps that each involves $2{\times}V{\times}d{+}l{\times}d^2$ computations. In addition to running the source graph encoder, the target graph alignment learns the attention networks which represents $\rho{\times}(2{\times}d{\times}V + m{\times}d^2)$. If we define $T = max(\rho, \tau)$, $k = max(l, m)$, and $M = N$ then the total computation cost is $\Theta(N^2 \times T \times (V \times d + k \times d^2)$. Because V is likely much larger than d^2, we interpret the computational complexity as $O(TN^2V)$.

In Section 5.4, we explore the effect of sampling to hasten DDGK's runtime on large datasets. We show that not all $M \times N$ comparisons are necessary to achieve high performance: empirically, it seems that less than 20% of source graphs are required, significantly speeding our approach.

5 EXPERIMENTS

In this section, we demonstrate our method through a series of qualitative and quantitative experiments. First, we show how our attention based alignment works under different conditions. Then, we show how our representations are capable of capturing the structure of the space of graphs by applying hierarchical clustering on the kernel space. Finally, we show that the learned graph embeddings represent a sufficient feature set to predict the graph label on several challenging classification problems in real-world biological datasets.

5.1 Cross-Graph Attention

In this qualitative experiment, we seek to understand how two graphs are related to each other. Comparing different patterns between different graphs is an important application in domains such as biology.

Figure 4a shows two identical unlabeled barbell graphs. Each graph consists of two rings of size 5 connected with the edge $(0, 5)$ and $(10, 15)$. The upper graph represents the target graph while the lower one represents the source graph. The edges connecting the source and target graphs represent the strongest attention weights for each node in the target graph. The heatmap shows the full attention matrix for more thorough analysis. Aligning these identical graphs is an easy task for the naked eye. However, our method can find many possible symmetries to exploit while still achieving perfect predictions. For example, nodes in the left ring can attend to the right ring of the source graph and vice versa.

Figure 4b shows the previous setup with labeled graph nodes. This introduces a regularizing loss to preserve the node attributes. The attention heat map shows significant weights for the upper left and lower rights quadrants. The right ring does not attend to nodes in the left ring anymore, and vice versa. Still we can see the method exploiting symmetries within the same ring.

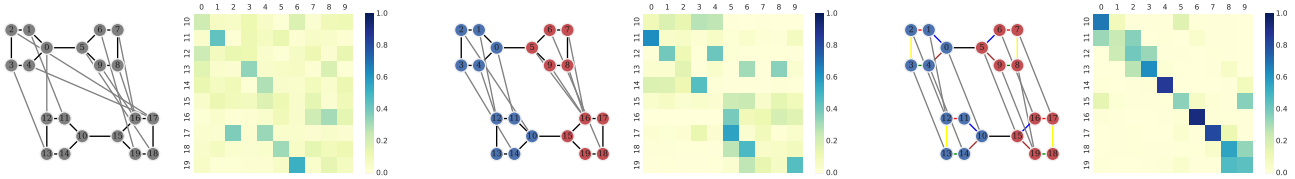
Finally, by also adding edge labels, the alignment problem is constrained enough that the attention heatmap is concentrated along the diagonal (See Figure 4c). We can observe that the attention edges correspond in a one-to-one relationship between the target and source graphs. This synthetic experiment shows the effect of attribute preserving losses on learning the alignment between graphs.

5.2 Hierarchical Clustering

To understand the global structure of the graph embedding space, we explore it qualitatively using hierarchical clustering. First, we create a dataset which is a composition of 6 different families of graphs. Three graph families are mutated graphs and three families are subset of a larger set of realistic graphs. From each family we sample 5 graphs, creating a universe of 30 graphs. Then, we embed the graphs using our method constructing a graph embedding space. Finally, we cluster the embeddings according to their pairwise euclidean distances.

5.2.1 Mutated Graphs. For these datasets we start with a known graph and generate a sequence of mutations to produce a family of graphs. In particular, we consider the following graphs.

- C. Elegans [53]: represents the neural network of the C. Elegans worm.
- Karate Club [60]: social network of friendships between 34 members of a karate club.
- Word Network [35]: adjacency network of common adjectives and nouns in the novel David Copperfield by Charles Dickens.



(a) No attributes. (b) Labeled nodes. (c) Labeled nodes and edges.

Figure 4: The effect of attributes preserving losses on the attention networks. Our method is given a pair of identical graphs, the upper graph represents the target and the other represents the source graph. Each graph consists of two rings of size 5 connected with one edge ((0, 5) and (10, 15) respectively). We visualize the strongest attention weights as cross-graph edges. On the right of each figure we visualize the rest of the attention weights as a heatmap. When the graph attends to itself without attribute preserving losses, there are several solutions that are equally good because of several symmetries available. Once we add the nodes attributes, we can see an immediate effect where the nodes from the same label class attend only to each other. This behavior further intensifies after also adding the edge attributes.

In order to generate a family $G_1 \cdots G_k$ for each original graph G_0, we employ the following mutation procedure. At each of the k time steps, there is a $p = 0.5$ chance of performing an edge deletion or addition. For additions, we select the two nodes to connect from any unlinked nodes according to the preferential attachment model characterized by G_0 [17]. For deletions, we select an edge at random and remove it. We run this procedure for 4 times with $k = 50$ time steps, creating a family of 5 related graphs. The initial seed for any of these mutations is denoted by the suffix "-0".

5.2.2 Realistic Graphs. We randomly pick 5 graphs from three of the real-world families of graphs we consider (**D&D**, **PTC**, and **NCI1**). See Section 5.3.2 for more information about these graphs.

Figure 5 shows the result of clustering the pairwise distances between our graph embeddings. We are able to retrieve perfect clusters of {`c-elegans`, `words`} where there are clusters of size of 5 that consist only of graphs of the same type. For {**NCI1**, **D&D**}, we can cluster 4 graphs out of 5 before adding a graph which is out of the family.

5.3 Graph Classification

Our learned graph representations respect both attributes and structure. They can be used for graph classification tasks where the graph structure, node attributes, and/or edge attributes convey meaning or function. To demonstrate this, we use DDGK representations of several chemo- and bio- informatics datasets as features for classification tasks. We report our results against both unsupervised and supervised methods.

5.3.1 Hyper-parameters Search. To choose DDGK hyperparameters (See Table 1), we perform grid searches for each dataset. We create splits of each dataset to avoid over-fitting, they are: {`train`, `dev`, `test`}. We use the scikit-learn SVM [39] as our classifier, and we vary the kernel choices between {`linear`, `rbf`, `poly`, `sigmoid`} and the regularization coefficient C between 10 and 10^9. We choose the hyper-parameters of both DDGK and the classifier that maximize the accuracy on the `dev` dataset.

5.3.2 Datasets. Four benchmark graph classification datasets from chemo- and bio-informatics domains are used. The datasets include **D&D**, **NCI1**, **PTC** and **MUTAG**. All datasets include node labels.

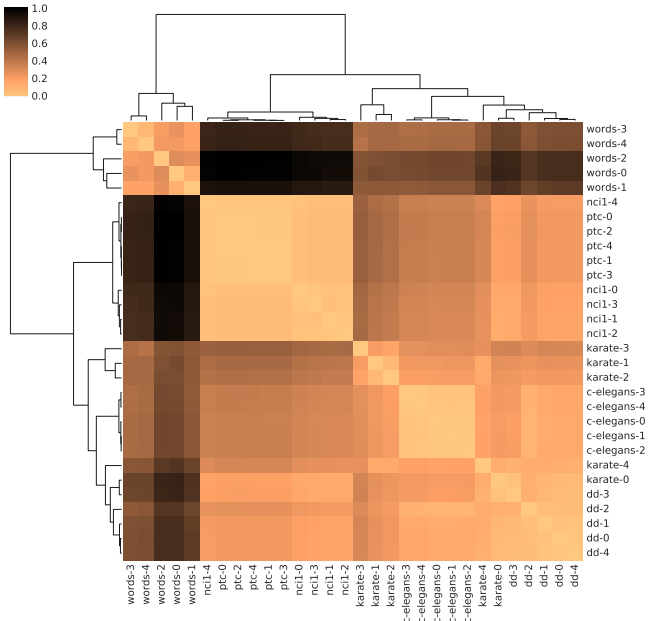


Figure 5: A hierarchical clustering of the graph kernel space for several different graph families. It shows 30 graphs that belong to 6 different families. The values of the matrix are the pairwise Euclidean distances between the graph embeddings.

The **PTC** and **MUTAG** datasets also include edge labels. Table 2 shows network statistics for each dataset. The datasets:

- **D&D** [20]: contains 1178 proteins labeled as enzymes or non-enzymes.
- **NCI1** [51]: contains 4110 chemical compounds labeled as active or inactive against non-small cell lung cancer.
- **PTC** [46]: contains 344 chemical compounds labeled according to their carcinogenicity in male rats.

Hyper-Parameter	Values
Node embedding	$2, 4, 8, 16, 32$
Encoder layers	$1, 2, 3, 4$
Learning rate	$10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1$
Encoding epochs	$100, 300, 600$
Scoring epochs	$100, 300, 600$
Node preserving loss coefficient	$0, 0.25, 0.5, 1.0, 1.5, 2.0$
Edge preserving loss coefficient	$0, 0.25, 0.5, 1.0, 1.5, 2.0$

Table 1: Values used during our grid search for DDGK graph representations learning hyper-parameters.

	# Graphs	Average Nodes	Average Edges	# Labels	# Node Labels	# Edge Labels
D&D	1178	284	716	2	89	–
NCI1	4110	30	32	2	37	–
PTC	344	14	15	2	18	4
MUTAG	188	18	20	2	7	4

Table 2: Statistics of the chemo- and bio-informatics datasets.

- **MUTAG** [19]: contains 188 mutagenic aromatic and heteroaromatic compounds labeled according to their mutagenic effect on a specific gram negative bacterium.

5.3.3 Results. The results of these experiments are presented in Table 3. We see that DDGK is quite competitive, with higher average performance on both the **D&D** and **MUTAG** datasets than any of the baselines. This is especially surprising given that the supervised methods have additional information available to them. We note that DDGK achieves its strong results without engineered features, or access to information from Weisfeiler-Lehman kernels. For **PTC**, we also see that DDGK attains competitive performance against all other methods, only being outperformed by 2 of the 9 baselines. Finally, on **NCI1**, we see that DDGK performs better than the method using the most similar kind of information (`node2vec`), but find that baselines using the WL kernel perform best on this dataset (indeed, the WL kernel itself takes the top two spots). We find this dependence quite interesting, and will seek to characterize it better in future work.

5.4 Dimension Sampling

So far, we have been setting the source graphs set to be equal to the target graphs set. This pairwise computation is quite expensive for large datasets. To reduce the computational complexity of our method, we study the effect of sub-sampling the dimensions of our graph embedding space on the quality of graph classification.

To do that, we construct a source graph set that is a subset of the original graph set. We learn divergence scores for all target graphs against this subset. We use the reduced embeddings as features to predict graph categories. Figure 6 shows that we are able to achieve stable and competitive results with less than 20% of the graphs being used as source graphs.

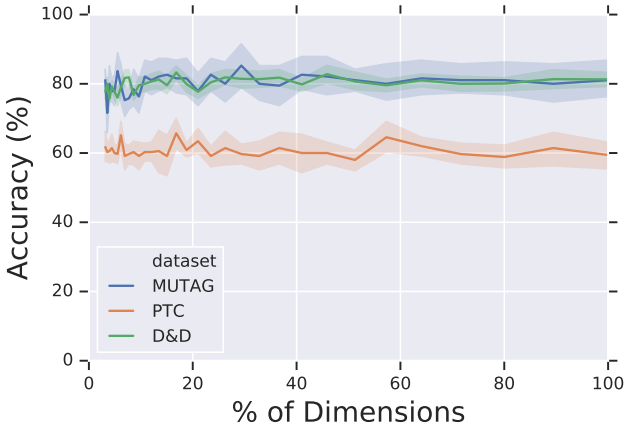


Figure 6: Effect of sub-sampling source graphs on graph classification tasks. Here we vary the number of source graphs available to each method, and observe that very few dimensions are needed to achieve our final classification performance (less than $< 20\%$ of the dimensions for the datasets considered).

6 RELATED WORK

The main differences between our proposed method and previous work can be summarized as follows:

(1) We are an unsupervised method, taking only a graph as input.
(2) We use no domain-specific information about what primitives are important in a graph, using only the edges.
(3) We use no algorithmic insights from the literature in graph isomorphism (e.g. the Weisfeiler-Lehman kernel).
(4) We assume nothing about the mapping of node ids between graphs, instead learning the alignment.

While many approaches exist that contain at least one of these differentiators, we are, to the best of our knowledge, the only proposed method that meets all four of these conditions. In this section we will briefly cover related work in graph similarity and other applications of neural networks to graph representation.

6.1 Unsupervised Graph Similarity

We divide our brief survey of the literature into three kinds of unsupervised methods for graph similarity. The first seeks to explicitly define a kernel over graph features, or use the intuition from such a kernel as part of the representation learning process. The second focuses on the representation of individual elements of the graph, learning primitives that maximize some kind of reconstruction of the graph. The third group of work constructs a similarity function between graphs by an explicit vector of statistical features constructed by the graph.

Traditional Graph Kernels: There has been considerable work done on unsupervised methods for graph kernel learning. Initial efforts in the area focused on theoretical views of the problem, defining graph similarity via the Graph Edit Distance [22] or the size of

Method		Unsupervised	No Weisfeiler-Lehman	No Feature Engineering	D&D	NCI1	PTC	MUTAG
PSCN	[36]				76.27 ± 2.64	76.34 ± 1.68	62.29 ± 5.68	88.95 ± 4.37
DGCNN	[61]				–	74.44 ± 0.40	58.59 ± 2.40	85.83 ± 1.60
SP Kernel	[8]	✓			79.00 ± 0.60	74.50 ± 0.30	58.90 ± 2.20	83.00 ± 1.40
WL Kernel	[32]	✓			79.00 ± 0.40	85.80 ± 0.20	61.30 ± 1.40	86.00 ± 1.70
WL-OA	[32]	✓			79.20 ± 0.40	**86.10** ± 0.20	63.60 ± 1.50	86.00 ± 1.70
DGK	[56]	✓			–	80.30 ± 0.40	60.10 ± 2.50	87.40 ± 2.70
graph2vec	[34]	✓			–	73.22 ± 1.90	60.17 ± 6.90	83.15 ± 9.20
S2S-N2N-PP	[43]	✓			–	83.72 ± 0.40	**64.54** ± 1.10	89.86 ± 1.10
node2vec	[24]	✓	✓		–	61.91 ± 0.30	55.60 ± 1.40	82.01 ± 1.00
DDGK	(this paper)	✓	✓	✓	**83.14** ± 2.72	68.10 ± 2.30	63.14 ± 6.57	**91.58** ± 6.74

Table 3: Average accuracy in ten-fold cross validation on our graph classification task. Methods are grouped by their level of supervision during the similarity metric learning, whether they use algorithm insights the Weisfeiler-Lehman algorithm, and whether they use feature engineering (e.g. graph motifs, random walks, etc.). Baseline results taken from [32, 36, 43] (missing results are missing from these works). We note that DDGK performs surprisingly competitively for an unsupervised method with no hard-coded insights.

the Maximum Common Subgraph [11] between graphs. Unfortunately these problems are both NP-Complete in the general case, require a known correspondence between the nodes of the two graphs of interest.

Many approaches are built around the graph similarity measure computed by the Weisfeiler-Lehman (WL) subtree graph kernel [32, 42]. At its core, the WL algorithm collapses the labels of a node's neighbors into a ordered sequence, and then hashes that sequence into a new label for the node. This process repeats iteratively to average information over the neighborhood together. Other functions that use different types of predefined features for graph similarity, such as shortest-paths kernels [8], and random walk kernels [28] have also been proposed, but their naive implementations suffer from high asymptotic complexity ($O(n^4)$ and $O(n^6)$, respectively). Faster implementations of these kernels have been proposed [10, 27]. Some unsupervised methods also focus on extending the algorithm intuition of these classic approaches to the problem. For instance [43] learns a representation for each position in a WL ordering jointly while learning a graph representation.

Unlike all of these approaches, our method deliberately avoids algorithmic insights. Our proposed isomorphism attention mechanism allows capturing higher-order structure between graphs (beyond immediate neighborhoods).

Node embedding methods: Since DeepWalk [40] proposed embedding the nodes via a sequence of random walks, the problem of node representation learning has received considerable attention [2, 7, 15, 21, 24, 41, 48]. In general, all of these methods utilize insights about similarity functions which are important to the graph. While these methods seek the best way to represent nodes, the representations are learned independently between graphs, which makes them generally unsuitable for graph similarity computations. For more information on this area, we recommend recent surveys in the area [14, 18]. Unlike these methods, our goal is to learn representations of graphs, not of nodes.

Graph statistics: Finally, another family of unsupervised graph similarity measures define a hand-engineered feature vector to compute graph similarity. The NetSmilie method [6] operates by constructing a fixed size feature value of graph statistics and uses this as a similarity embedding over graphs. Similarly, DetlaCon [31] defines the similarity over two graphs with known node-to-node mapping via the similarity in their propagation of belief, and [38] proposes a number of similarity measures over directed web graphs.

Unlike these methods, DDGK does not explicitly engineer its features for the problem. Instead, the similarity is learned function directly from the edges present in the adjacency matrix, with no assumptions about which features are important for the application task.

6.2 Supervised Graph Similarity

The first class of supervised methods uses some supervision to inform a similarity function constructed over different hand-engineered graph features.

A number of supervised approaches also utilize intuitions from the Weisfeiler-Lehman graph kernel. `Patchy-SAN` [37] proposes an approach for convolutional operations on graph structured data. The core of their method uses the ordering from the WL kernel to order the nodes of a rooted subgraph into a sequence, and then apply standard 1-dimensional convolutional filters. This approach is further generalized by [61], who use the WL ordering to sort a graph sample in a pooling layer. Another branch of work has focused on extending the Graph Convolutional Networks (GCNs) proposed by [30] to perform supervised classification of graphs. Proposed extensions include a pooling architecture that learns a soft clustering of the graph [59], or a two-tower model which frames

graph similarity as link prediction between GCN representations [4]. Interestingly, it has been shown that many of these methods are not necessarily more expressive than the original Weisfeiler-Lehman subtree kernel itself [33].

Unlike all of these approaches, our method learns representations of graphs without supervision — we use no labels about the class label of a graph, and no external information about which pairs of graphs are related. Our proposed isomorphism attention mechanism allows capturing higher-order structure between graphs (beyond immediate neighborhoods).

7 EXTENSIONS & FUTURE WORK

Here we briefly discuss a number of areas of future investigation for our method.

7.1 Graph Encoders

Given the choice of input and reconstructed output, several additional graph encoders are possible, in addition to the Nodes-To-Edges encoder which we used in this work. To mention a few options:

Edge-To-Nodes Encoder. - This encoder is trained to predict the source and destination vertices given a specific edge. Similar to the *Node-To-Edges* encoder, this could be expressed as a multilabel classification task with the following objective function,

$$J(\theta) = \sum_{e_{ij} \in E} \log \Pr(v_i \mid e_{ij}, \theta) + \log \Pr(v_j \mid e_{ij}, \theta) \tag{12}$$

Note that the number of edges in a graph could grow quadratically, therefore, iterating over the edges is more expensive than the nodes.

Neighborhood Encoder. - In this case, the encoder is trained to predict a set of vertices or edges that are beyond the immediate neighbors. Random walks could serve as a mechanism to calculate a neighborhood around a specific node or edge. Given a partial random walk, the encoder has to predict the vertices that could have been visited within a specific number of hops.

$$J(\theta) = \sum_{\substack{(v_1, v_2, \cdots, v_i) \\ \sim RandomWalk(G,E,V)}} \log \Pr\left(v_j \mid (v_1, v_2, \cdots, v_i, \theta)\right) \tag{13}$$

7.2 Attention Mechanism

We proposed a simple attention mechanism which uses node-to-node alignment. As we discussed in Section 4.5, we could replace the linear layer with a deep neural network to reduce the size of the model if scability is an issue. While node-to-node alignment enhances the interpretability of our models, subgraph alignment could lead to better and easier understanding of how two graphs are similar. Hierarchical attention models [57] could lead to higher levels of abstractions which could learn community structure and which communities are similar across a pair of graphs. Hierarchy has already been used within the context of learning better node embeddings, for example [59] showed that a better understanding of the graph substructure can lead to better graph classification. Therefore, we believe extending the work beyond node-to-node alignment will significantly improve our results.

7.3 Regularization

We proposed attribute based losses to regularize our isomorphism attention mechanism. The graph encoder capacity was adjusted according to the source graph size. However, the source graph encoder could still suffer from overfitting which would reduce its utility in recognizing similar target graphs. Therefore, further research is necessary to understand the relation between the encoder training characteristics and the quality of the generated divergence scores

7.4 Feature Engineering

In this work we have focused on developing an approach for representing graphs that operated without any feature engineering or algorithmic insights. While this willful ignorance has allowed us to design a new paradigm for graph similarity, we suspect that there are many fruitful combinations of this idea with other approaches for graph classification. For example, the graph embeddings we learn could be used as additional features for approaches based on learning supervised classifiers over graphs.

8 CONCLUSION

In this work, we have shown that neural networks can learn powerful representations of graphs without explicit feature engineering. Our proposed method, Deep Divergence Graph Kernels, learns an encoder for each graph to capture its structure, and uses a novel *isomorphism preserving attention mechanism* to align node representations across graphs without the use of supervision. We show that representing graphs by their divergence from different source graphs provides a powerful embedding space over families of graphs. Our proposed model is both flexible and amenable to extensions. We illustrate this by proposing extensions to handle many commonly occurring varieties of graphs, including graphs with attributed nodes, and graphs with attributed edges.

Our experimental analysis shows that despite being trained with only the graph's edges (and no feature engineering) the learned representations encode a variety of local and global information. When the representations produced by DDGK are used as features for graph classification methods, we find them to be competitive with challenging baselines which use at least one of graph labels, engineered features, or the Weisfeiler-Lehman framework. In addition to being powerful, DDGK models are incredibly informative. The learned isomorphism attention weights allow a level of insight into the alignment between a pair of graphs, which is not possible with other deep learning methods developed for graph similarity.

Unsupervised representation learning for graphs is an important problem, and we believe that the method of Deep Divergence Graph Kernels we have introduced here is an exciting step forward in this area. As future work, we will investigate 1) enhanced method for choosing informative source from the space of all graphs, 2) improving the architecture of our encoders and attention models, 3.) making it easier to reproduce research results in the area of graph similarity, and 4) making graph similarity models even easier to understand.

REFERENCES

[1] Sami Abu-El-Haija, Bryan Perozzi, and Rami Al-Rfou. 2017. Learning Edge Representations via Low-Rank Asymmetric Projections. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM '17)*. ACM, New York, NY, USA, 1787–1796. https://doi.org/10.1145/3132847.3132959

[2] Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alexander A Alemi. 2018. Watch Your Step: Learning Node Embeddings via Graph Attention. In *Advances in Neural Information Processing Systems*. 9198–9208.

[3] Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2016. Learning principled bilingual mappings of word embeddings while preserving monolingual invariance. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2289–2294. https://doi.org/10.18653/v1/D16-1250

[4] Yunsheng Bai, Hao Ding, Song Bian, Yizhou Sun, and Wei Wang. 2018. Graph Edit Distance Computation via Graph Neural Networks. *arXiv preprint arXiv:1808.05689* (2018).

[5] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018).

[6] Michele Berlingerio, Danai Koutra, Tina Eliassi-Rad, and Christos Faloutsos. 2012. Netsimile: A scalable approach to size-independent network similarity. *arXiv preprint arXiv:1209.2684* (2012).

[7] Aleksandar Bojchevski and Stephan Günnemann. 2017. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815* (2017).

[8] Karsten M Borgwardt and Hans-Peter Kriegel. 2005. Shortest-path kernels on graphs. In *Data Mining, Fifth IEEE International Conference on*. IEEE, 8–pp.

[9] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. 2005. Protein function prediction via graph kernels. *Bioinformatics* 21, suppl_1 (2005), i47–i56.

[10] Karsten M Borgwardt, Nicol N Schraudolph, and SVN Vishwanathan. 2007. Fast computation of graph kernels. In *Advances in neural information processing systems*. 1449–1456.

[11] Horst Bunke, Pasquale Foggia, Corrado Guidobaldi, Carlo Sansone, and Mario Vento. 2002. A comparison of algorithms for maximum common subgraph on randomly connected graphs. In *SSPR*. Springer, 123–132.

[12] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep Neural Networks for Learning Graph Representations. In *Proceedings of the Association for the Advancement of Artificial Intelligence*.

[13] Sandro Cavallari, Vincent W Zheng, Hongyun Cai, Kevin Chen-Chuan Chang, and Erik Cambria. 2017. Learning community embedding with community detection and node embedding on graphs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 377–386.

[14] Haochen Chen, Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2018. A Tutorial on Network Embeddings. *arXiv preprint arXiv:1808.02590* (2018).

[15] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. 2018. Harp: Hierarchical representation learning for networks. (2018).

[16] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. 2015. Attention-Based Models for Speech Recognition. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 577–585. http://papers.nips.cc/paper/5847-attention-based-models-for-speech-recognition.pdf

[17] Fan Chung, Fan RK Chung, Fan Chung Graham, Linyuan Lu, Kian Fan Chung, et al. 2006. A Generative Model - the Preferential Attachment Scheme. In *Complex graphs and networks*. Number 107. American Mathematical Soc., Chapter 3.

[18] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2018. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering* (2018).

[19] Asim Kumar Debnath, Rosa L. Lopez de Compadre, Gargi Debnath, Alan J. Shusterman, and Corwin Hansch. 1991. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry* 34, 2 (1991), 786–797. https://doi.org/10.1021/jm00106a046 arXiv:https://doi.org/10.1021/jm00106a046

[20] Paul D Dobson and Andrew J Doig. 2003. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology* 330, 4 (2003), 771–783.

[21] Alessandro Epasto and Bryan Perozzi. 2019. Is a Single Embedding Enough? Learning Node Representations that Capture Multiple Social Contexts. *WWW* (2019).

[22] Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. 2010. A survey of graph edit distance. *Pattern Analysis and applications* 13, 1 (2010), 113–129.

[23] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1263–1272.

[24] A. Grover and J. Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[25] Bernard Haasdonk and Claus Bahlmann. 2004. Learning with distance substitution kernels. In *Joint Pattern Recognition Symposium*. Springer, 220–227.

[26] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[27] U Kang, Hanghang Tong, and Jimeng Sun. 2012. Fast random walk graph kernel. In *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM, 828–838.

[28] Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. 2003. Marginalized kernels between labeled graphs. In *Proceedings of the 20th international conference on machine learning (ICML-03)*. 321–328.

[29] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[30] T. Kipf and M. Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv preprint arXiv:1609.02907*.

[31] Danai Koutra, Joshua T Vogelstein, and Christos Faloutsos. 2013. Deltacon: A principled massive-graph similarity function. In *Proceedings of the 2013 SIAM International Conference on Data Mining*. SIAM, 162–170.

[32] Nils M Kriege, Pierre-Louis Giscard, and Richard Wilson. 2016. On valid optimal assignment kernels and applications to graph classification. In *Advances in Neural Information Processing Systems*. 1623–1631.

[33] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2018. Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks. *arXiv preprint arXiv:1810.02244* (2018).

[34] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. 2017. graph2vec: Learning Distributed Representations of Graphs. *CoRR* abs/1707.05005 (2017). arXiv:1707.05005 http://arxiv.org/abs/1707.05005

[35] M. E. J. Newman. 2006. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E* 74 (Sep 2006), 036104. Issue 3. https://doi.org/10.1103/PhysRevE.74.036104

[36] M. Niepert, M. Ahmed, and K. Kutzkov. 2016. Learning Convolutional Neural Networks for Graphs. In *International Conference on Machine Learning (ICML)*.

[37] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *International conference on machine learning*. 2014–2023.

[38] Panagiotis Papadimitriou, Ali Dasdan, and Hector Garcia-Molina. 2010. Web graph similarity for anomaly detection. *Journal of Internet Services and Applications* 1, 1 (2010), 19–30.

[39] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.

[40] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[41] Bryan Perozzi, Vivek Kulkarni, Haochen Chen, and Steven Skiena. 2017. Don't Walk, Skip!: Online Learning of Multi-scale Network Embeddings. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*. ACM, 258–265.

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

[43] Aynaz Taheri, Kevin Gimpel, and Tanya Berger-Wolf. 2018. Learning Graph Representations with Recurrent Neural Network Autoencoders. In *KDD'18 Deep Learning Day*.

[44] TensorflowTeam. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. http://tensorflow.org/ Software available from tensorflow.org.

[45] Antoine J-P Tixier, Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. 2018. Graph Classification with 2D Convolutional Neural Networks. (2018).

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

[47] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alex Bronstein, and Emmanuel Müller. 2018. NetLSD: Hearing the Shape of a Graph. *KDD* (2018).

[48] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2017. VERSE: Versatile Graph Embeddings from Similarity Measures. (2017).

[49] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*. 5998–6008.

[50] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* 1, 2 (2017).

[51] Nikil Wale and George Karypis. 2006. Comparison of descriptor spaces for chemical compound retrieval and classification. In *Proceedings - Sixth International Conference on Data Mining, ICDM 2006*. 678–689. https://doi.org/10.1109/ICDM.2006.39

[52] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. Community Preserving Network Embedding.. In *AAAI*. 203–209.

[53] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world'networks. *nature* 393, 6684 (1998), 440.

[54] Lingfei Wu, Ian En-Hsu Yen, Fangli Xu, Pradeep Ravikuma, and Michael Witbrock. 2018. D2KE: From Distance to Kernel and Embedding. *arXiv preprint arXiv:1802.04956* (2018).

[55] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*. 2048–2057.

[56] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1365–1374.

[57] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 1480–1489.

[58] Zi Yin and Yuanyuan Shen. 2018. On the Dimensionality of Word Embedding. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.). Curran Associates, Inc., 887–898. http://papers.nips.cc/paper/7368-on-the-dimensionality-of-word-embedding.pdf

[59] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. *arXiv preprint arXiv:1806.08804* (2018).

[60] Wayne W. Zachary. 1977. An Information Flow Model for Conflict and Fission in Small Groups. *Journal of Anthropological Research* 33, 4 (1977), 452–473.

[61] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification.

[62] Vincent W Zheng, Sandro Cavallari, Hongyun Cai, Kevin Chen-Chuan Chang, and Erik Cambria. 2016. From node embedding to community embedding. *arXiv preprint arXiv:1610.09950* (2016).

[63] Dingyuan Zhu, Peng Cui, Daixin Wang, and Wenwu Zhu. 2018. Deep Variational Network Embedding in Wasserstein Space. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2827–2836.