PiNet: A Permutation Invariant Graph Neural Network for Graph Classification

Peter Meltzer[1,2], **Marcelo Daniel Gutierrez Mallea**[2] and **Peter J. Bentley**[1,2]
[1]Department of Computer Science, University College London, UK
[2]Braintree Ltd., London, UK
{p.meltzer, p.bentley}@cs.ucl.ac.uk, {p.meltzer, p.bentley, m.gutierrez}@braintree.com

Abstract

We propose an end-to-end deep learning learning model for graph classification and representation learning that is invariant to permutation of the nodes of the input graphs. We address the challenge of learning a fixed size graph representation for graphs of varying dimensions through a differentiable node attention pooling mechanism. In addition to a theoretical proof of its invariance to permutation, we provide empirical evidence demonstrating the statistically significant gain in accuracy when faced with an isomorphic graph classification task given only a small number of training examples.

We analyse the effect of four different matrices to facilitate the local message passing mechanism by which graph convolutions are performed vs. a matrix parametrised by a learned parameter pair able to transition smoothly between the former. Finally, we show that our model achieves competitive classification performance with existing techniques on a set of molecule datasets.

1 Introduction

Graph classification, the problem of predicting a label to each graph in a given set, is of significant interest in the bio- and chemo-informatics domains, among others; with typical applications in predicting chemical properties [Li and Zemel, 2016], drug effectiveness [Neuhaus *et al.*, 2009], protein functions [Shervashidze *et al.*, 2009], and classification of segmented images [Scarselli *et al.*, 2009].

There are two major challenges faced by graph classifiers: First, a problem of ordering, i.e. the ability to recognise isomorphic graphs as equivalent when the order of their nodes/edges are permuted, and second, in how to handle instances of varying dimensions, i.e. graphs with different numbers of nodes/edges. In image classification, the ordering of pixels is given, and instances differing in size may be scaled; however, for graphs the ordering of nodes/edges is typically arbitrary, and finding the analogous transformation to scaling an image is evidently non-trivial.

Typical approaches to solving these challenges include kernel methods, in which implicit kernel spaces circumvent the need to map each instance to a fixed size, ordered representation for classification [Zhang *et al.*, 2018], and deep learning architectures with some explicit feature extraction method whereby a fixed size representation is constructed for each graph and passed to a CNN (or similar) classification model [Niepert *et al.*, 2016]. While the deep learning approaches often outperform the kernel methods with respect to scalability in the number of graphs, they require suitable fixed size representation for each graph, and typically cannot guarantee that isomorphic graphs will be interpreted as the same.

In order to address these issues, we propose PiNet; an end-to-end deep learning graph convolution architecture with guaranteed invariance to permutation of nodes in the input graphs. To present our model, we first review relevant literature, then present the architecture with proof of its invariance to permutation. We conduct three experiment to evaluate PiNet's effectiveness: We verify the utility in its invariance to permutation with a graph isomorphism classification task, we then test its ability to learn appropriate message passing matrices, and we perform a benchmark test against existing classifiers on a set of standard molecule classification datasets. Finally, we draw our conclusions and suggest further work.

2 Background

2.1 Graph Kernels

A graph kernel ϕ is a positive semi-definite function that maps graphs belonging to a space $\mathcal{G}$ to an inner product in some Hilbert space $\mathcal{H}$, $\phi : \mathcal{G} \rightarrow \mathcal{H}$. Graph classification can be performed in the mapped space $\mathcal{H}$ with standard classification algorithms, or using the *kernel trick* (with SVMs, for example) the mapped feature space may be exploited implicitly. In this sense, kernel methods are well suited to deal with the high and variable dimensions of graph data, where explicit computation of such a feature space may not be possible.

Despite the large number of graph kernels found in the literature, they typically fall into just three distinct classes [Shervashidze *et al.*, 2011]: Graph kernels based on random walks and paths [Borgwardt *et al.*, 2005], graph kernels based on frequencies of limited size subgraphs or graphlets [Shervashidze *et al.*, 2009], and graph kernels based on subtree patterns where a similarity matrix between two graphs is defined by the number of matching subtrees in each graph [Harchaoui and Bach, 2007].

Although kernels are well suited to varying dimensions of graphs, their scalability is limited. In many cases they scale poorly to large graphs [Shervashidze *et al.*, 2010] and given their reliance on SVMs or full computation of a kernel matrix, they become intractable for large numbers of graphs.

2.2 Graph Neural Networks

Convolutional and recurrent neural networks, while successful in many domains, struggle with a graph inputs because of the arbitrary order in which the nodes of each instance may

appear [Ying *et al.*, 2018]. For node level tasks (i.e. node classification and link prediction) graph neural networks (GNNs) [Scarselli *et al.*, 2009] handle this issue well by integrating the neural network structure with that of the graph. In each layer, state associated with each node is propagated to its neighbours via a learned filter and then a non linear function is applied. Thus the network's inner layers learn a latent representation for each node. For example, the GCN [Kipf and Welling, 2016] uses a normalised version of the graph adjacency matrix to propagate node features while learning spectral filters. The GCN model has received significant attention in recent years with several extensions already in the literature [Hamilton *et al.*, 2017; Atwood and Towsley, 2016; Romero, 2018].

However, for graph level tasks the GNN model and its variants do not handle permutations of the nodes well. For example, for a pair of isomorphic graphs, corresponding nodes would receive corresponding outputs, but for the graph as whole, the node level outputs will not necessarily be given in the same order, thus two graphs may be given shuffled representations presenting an obvious challenge for a downstream classifier. One approach to solving this problem is proposed by [Verma and Zhang, 2018] where a permutation invariant layer based on computing the covariance of the data is added to the GCN architecture; however, computation of the covariance matrix $O(n^3)$ in the number of nodes, thus not an attractive solution for large graphs.

2.3 Mixed Models

Combining elements of kernel methods and neural networks, mixed models use an explicit method in order to generate vectorized representations of the graphs which are then passed to a conventional neural network. One benefit to this approach being that explicit kernels are typically more efficient when dealing with large numbers of graphs [Kriege *et al.*, 2015].

For example, PATCHY-SAN [Niepert *et al.*, 2016] extracts fixed size localized patches by applying a graph labelling procedure given by the WL algorithm [Weisfeiler and Lehman, 1968] and a canonical labeling procedure from [McKay, 1981] to order the nodes. It then uses these patches to form a 3-dimensional tensor for each graph that is passed to a standard CNN for classification.

A similar procedure is presented in [Gutierrez Mallea *et al.*, 2019] where, in order to overcome the potential loss of information associated with the CNN convolution operation, a Capsule network is used to perform the classification.

3 PiNet

Formally, we consider the problem of predicting a label $\mathbf{y}$ for an unseen test graph, given a training set $\mathcal{G}$ with corresponding labels $\mathcal{Y}_L$. Each graph $G \in \mathcal{G}$ is defined as the pair $G = (\mathbf{A}, \mathbf{X})$, where $\mathbf{A} \in \mathbb{R}^{N\times N}$ is the graph adjacency matrix, and $\mathbf{X} \in \mathbb{R}^{N\times d}$ is a corresponding matrix of d-dimensional node features. We fix N to be the maximum number of nodes in each of the graphs in $\mathcal{G}$, padding empty rows and columns of $\mathbf{A}$ and $\mathbf{X}$ with zeros. Note, however, that these zero entries do not form part of the final graph representations used by the model.

3.1 Model Architecture

PiNet is an end-to-end deep neural network architecture that utilizes the permutation equivariance of graph convolutions in order to learn graph representations that are invariant to permutation.

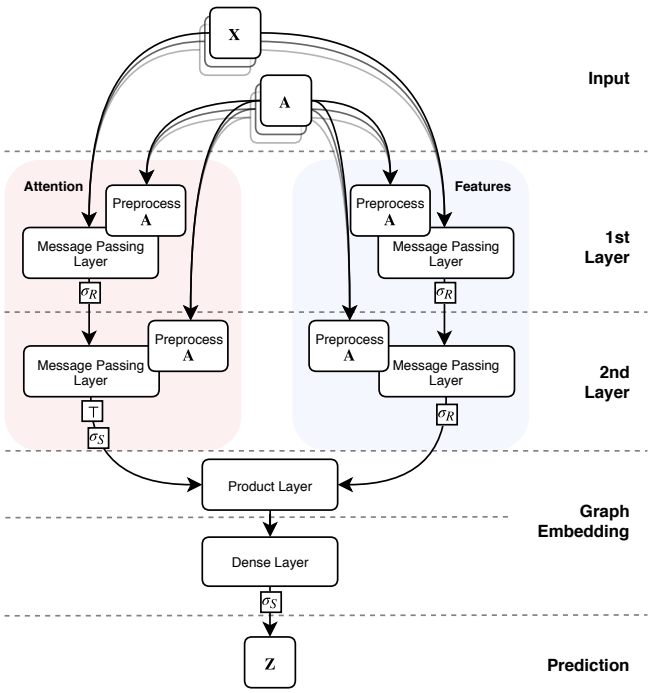

Figure 1: Model Architecture. $\mathbf{A}$ is the adjacency matrix of a single graph, $\mathbf{X}$ the corresponding feature matrix, and $\mathbf{Z}$ the predicted label(s) for the complete batch of input graphs.

As shown in Figure 1, our model consists of a pair of double-stacked message passing layers combined by a matrix product. Let σ_R and σ_S denote the rectified linear unit and softmax activations functions, $\tilde{\mathbf{A}}$ the preprocessed adjacency matrix, and $F_X^{(1)}$and $F_A^{(1)}$the dimensions per node of the latent representation of the features and attention stacks respectively. The features and attention stacks each output a tensor (Z_X and Z_A) with each element corresponding to an input graph given by the functions $z_X : (\mathbb{R}^{N\times N}, \mathbb{R}^{N\times d}) \to \mathbb{R}^{N\times F_X^{(1)}}$ and$z_A : (\mathbb{R}^{N\times N}, \mathbb{R}^{N\times d}) \to \mathbb{R}^{F_A^{(1)}\times N}$respectively, where

$$z_X(\mathbf{A}, \mathbf{X}) = \sigma_R\left(\tilde{\mathbf{A}}\ \sigma_R\left(\tilde{\mathbf{A}}\ \mathbf{X}\mathbf{W}_X^{(0)}\right)\mathbf{W}_X^{(1)}\right), \quad (1)$$

$$z_A(\mathbf{A}, \mathbf{X}) = \sigma_S\left(\left[\tilde{\mathbf{A}}\ \sigma_R\left(\tilde{\mathbf{A}}\ \mathbf{X}\mathbf{W}_A^{(0)}\right)\mathbf{W}_A^{(1)}\right]^\top\right), \quad (2)$$

and

$$\tilde{\mathbf{A}} = (p\mathbf{I} + (1-p)\mathbf{D})^{-\frac{1}{2}}(\mathbf{A} + q\mathbf{I})(p\mathbf{I} + (1-p)\mathbf{D})^{-\frac{1}{2}}, \quad (3)$$

where $\mathbf{I}$ is the identity matrix of order N, $\mathbf{D}$ is the degree matrix such that

$$\mathbf{D}_{ij} = d(\mathbf{A})_{ij} = \begin{cases} \sum_{k=1}^{N} \mathbf{A}_{ik} & \text{if } i = j, \\ 0 & \text{otherwise,} \end{cases} \quad (4)$$

and $0 \le p, q \le 1$.

The use of softmax activation on the attention stack applies the constraint that outputs sum to 1, thus preventing all node attention weightings from dropping to 0 and keeping the resulting products within a reasonable range.

The trainable parameters p and q offer an extra attention mechanism that enables the model to weigh the importance

of symmetric normalisation of the adjacency matrix, and the addition of self loops. Table 1 shows the four matrices given by the extreme cases of p and q; however, intermediate combinations are of course possible. Also note that p and q may be different for each message passing layer.

Matrix	Definition	p	q
Adjacency	$\mathbf{A}$	1	0
$\mathbf{A}$ with S.L.	$\mathbf{A}+\mathbf{I}$	1	1
Sym Norm Adjacency	$\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$	0	0
Sym Norm $\mathbf{A}$ with S.L.	$\mathbf{D}^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I})\mathbf{D}^{-\frac{1}{2}}$	0	1

Table 1: The four message passing matrices given by the extreme values of p and q, i.e. $(p, q) \in \{0, 1\} \times \{0, 1\}$.

As seen in Figure 2, the weighting on the self loops given by q allows the model to include each node's own state in the previous layer as an input to the next layer.

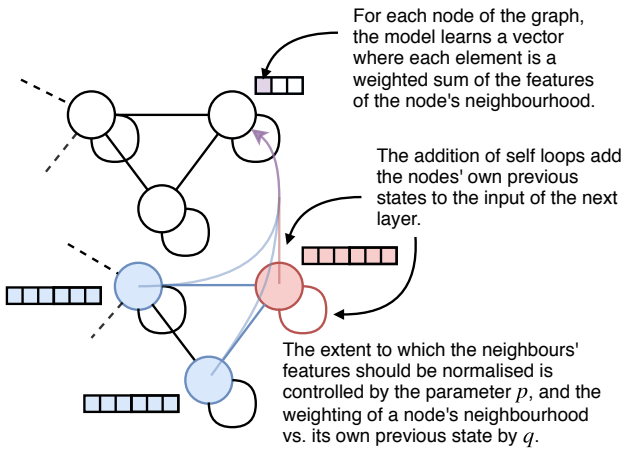

Figure 2: Layer propagation

The final output of the model is the matrix $\mathbf{Z}$, where each row i is the predicted label given by the function

$$z(\mathbf{A}, \mathbf{X}) = \sigma_S \left[g\left(z_A\left(\mathbf{A}, \mathbf{X}\right) \cdot z_X\left(\mathbf{A}, \mathbf{X}\right)\right) \mathbf{W}_D \right] \in \mathbb{R}^C, \quad (5)$$

where $g : \mathbb{R}^{F_A^{(1)} \times F_X^{(1)}} \to \mathbb{R}^{F_A^{(1)} \cdot F_X^{(1)}}$ is a reshape function. Note that since Z_A has been transposed, the columns of each matrix (corresponding directly to the nodes of the graph) align exactly with the rows of each matrix in Z_X, thus $\forall i \in |\mathcal{G}|$ the product $[Z_A]_i \cdot [Z_X]_i \in \mathbb{R}^{F_A^{(1)} \times F_X^{(1)}}$ aligns the weights for each node learned by the attention stack with the latent features learned by the features stack, resulting in a weighted sum of the features of each node without being affected by permutations of the nodes at input.

Finally, to train the model, we minimise the categorical cross-entropy

$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{C} \mathbf{Y}_{lf} \ln \mathbf{Z}_{lf}, \quad (6)$$

where $\mathbf{Y}$ is the target labels, and C the number of classes.

3.2 Permutation Invariance

Here, following the necessary definitions, we provide proof of our model's invariance to permutation of the nodes in the input graphs.

Definition 3.1 (Permutation)**.** A permutation π is a bijection of a set S onto itself, such that it moves an object at position i to position $\pi(i)$. For example, a permutation of π to the vector $(x_1, x_2, \ldots, x_n)$ would be $(x_{\pi^{-1}(1)}, x_{\pi^{-1}(2)}, \ldots, x_{\pi^{-1}(n)})$.

Definition 3.2 (Permutation Matrix)**.** A permutation matrix $\mathbf{P}_\pi \in \{0, 1\}^{n \times n}$ is an orthogonal matrix such that:

$$[\mathbf{P}_\pi \mathbf{X}]_{ij} = \mathbf{X}_{\pi^{-1}(i)j}, \qquad \mathbf{X} \in \mathbb{R}^{n \times m} \quad (7)$$

$$\left[\mathbf{X}\mathbf{P}_\pi^\top\right]_{ij} = \mathbf{X}_{i\pi^{-1}(j)}, \qquad \mathbf{X} \in \mathbb{R}^{m \times n} \quad (8)$$

thus, for a square matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$,

$$\left[\mathbf{P}_\pi \mathbf{A} \mathbf{P}_\pi^\top\right]_{ij} = \mathbf{A}_{\pi^{-1}(i)\pi^{-1}(j)}. \quad (9)$$

Definition 3.3 (Graph Permutation)**.** We define the permutation of π to a graph G as a mapping of the node indices, i.e.

$$\mathbf{P}_\pi G = (\mathbf{P}_\pi \mathbf{A} \mathbf{P}_\pi^\top, \mathbf{P}_\pi \mathbf{X}) \quad (10)$$

Definition 3.4 (Permutation Invariance)**.** Let $\mathcal{P}_n$ be the set of all valid permutation matrices of order n, then a function f is invariant to row permutation iff

$$f(\mathbf{X}) = f(\mathbf{P}_\pi \mathbf{X}), \qquad \forall \mathbf{X} \in \mathbb{R}^{n \times m}, \mathbf{P}_\pi \in \mathcal{P}_n, \quad (11)$$

and f is invariant to column permutation iff

$$f(\mathbf{X}) = f(\mathbf{X}\mathbf{P}_\pi^\top), \qquad \forall \mathbf{X} \in \mathbb{R}^{m \times n}, \mathbf{P}_\pi \in \mathcal{P}_n. \quad (12)$$

Definition 3.5 (Permutation Equivariance)**.** Let $\mathcal{P}_n$ be the set of all valid permutation matrices of order n, then a function f is *equivariant* to row permutation iff

$$\mathbf{P}_\pi f(\mathbf{X}) = f(\mathbf{P}_\pi \mathbf{X}), \qquad \forall \mathbf{X} \in \mathbb{R}^{n \times m}, \mathbf{P}_\pi \in \mathcal{P}_n, \quad (13)$$

similarly, for column permutation $f(\mathbf{X})\mathbf{P}_\pi^\top = f(\mathbf{X}\mathbf{P}_\pi^\top)$.

Lemma 3.1. *For any matrices $\mathbf{A} \in \mathbb{R}^{n \times m}$ and $\mathbf{B} \in \mathbb{R}^{n \times p}$, and any permutation π, the product $\mathbf{A}^\top \cdot \mathbf{B}$ remains unchanged by a permutation of π applied to the rows of $\mathbf{A}$ and $\mathbf{B}$, i.e.*

$$(\mathbf{P}_\pi \mathbf{A})^\top \cdot \mathbf{P}_\pi \mathbf{B} = \mathbf{A}^\top \cdot \mathbf{B}. \quad (14)$$

Proof. Consider the vector product

$$\mathbf{a}^\top \cdot \mathbf{b} = \sum_k^n \mathbf{a}_k \mathbf{b}_k. \quad (15)$$

We permute the rows of $\mathbf{a}$ and the columns of $\mathbf{b}$

$$\mathbf{P}_\pi \mathbf{a}^\top \cdot \mathbf{b}\mathbf{P}_\pi^\top = \sum_k^n \mathbf{a}_{\pi^{-1}(k)} \mathbf{b}_{\pi^{-1}(k)} \quad (16)$$

and observe a reordering of terms, in which the factor pairs remain in correspondence. Since addition is commutative, then

$$\sum_k^n \mathbf{a}_{\pi^{-1}(k)} \mathbf{b}_{\pi^{-1}(k)} = \sum_k^n \mathbf{a}_k \mathbf{b}_k \quad (17)$$

$$\implies \mathbf{P}_\pi \mathbf{a}^\top \cdot \mathbf{b}\mathbf{P}_\pi^\top = \mathbf{a}^\top \cdot \mathbf{b}. \quad (18)$$

By the same logic, we see that

$$\left[(\mathbf{P}_\pi\mathbf{A})^\top \cdot \mathbf{P}_\pi\mathbf{B}\right]_{ij} = \sum_k^n \mathbf{A}_{\pi^{-1}(k)i}\mathbf{B}_{\pi^{-1}(k)j} \tag{19}$$

$$= \sum_k^n \mathbf{A}_{ki}\mathbf{B}_{kj} \tag{20}$$

$$= \left[\mathbf{A}^\top \cdot \mathbf{B}\right]_{ij} \tag{21}$$

$$\implies (\mathbf{P}_\pi\mathbf{A})^\top \cdot \mathbf{P}_\pi\mathbf{B} = \mathbf{A}^\top \cdot \mathbf{B} \tag{22}$$

□

Lemma 3.2. *If $\tilde{\mathbf{A}}$ is the preprocessed version of $\mathbf{A}$, then $\mathbf{P}_\pi\tilde{\mathbf{A}}\mathbf{P}_\pi^\top$ is the preprocessed version of $\mathbf{P}_\pi\mathbf{A}\mathbf{P}_\pi^\top$, i.e. If*

$$\tilde{\mathbf{A}} = (p\mathbf{I} + (1-p)d(\mathbf{A}))^{-\frac{1}{2}}(\mathbf{A} + q\mathbf{I})(p\mathbf{I} + (1-p)d(\mathbf{A}))^{-\frac{1}{2}} \tag{23}$$

then

$$\mathbf{P}_\pi\tilde{\mathbf{A}}\mathbf{P}_\pi^\top = (p\mathbf{I} + (1-p)d(\mathbf{P}_\pi\mathbf{A}\mathbf{P}_\pi^\top))^{-\frac{1}{2}}(\mathbf{P}_\pi\mathbf{A}\mathbf{P}_\pi^\top + q\mathbf{I})$$
$$(p\mathbf{I} + (1-p)d(\mathbf{P}_\pi\mathbf{A}\mathbf{P}_\pi^\top))^{-\frac{1}{2}}, \tag{24}$$

where $d(\mathbf{A})$ is the diagonal degree matrix of $\mathbf{A}$ as defined in Equation 4.

Proof. From Equation 4,

$$d(\mathbf{P}_\pi\mathbf{A}\mathbf{P}_\pi^\top)_{ij} = \begin{cases} \sum_{k=1}^N \mathbf{A}_{\pi^{-1}(i)\pi^{-1}(k)} & \text{if } i = j, \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

$$= \mathbf{P}_\pi d(\mathbf{A})\mathbf{P}_\pi^\top \tag{26}$$

Considering each factor of Equation 24 (RHS), we observe that

$$(p\mathbf{I} + (1-p)d(\mathbf{P}_\pi\mathbf{A}\mathbf{P}_\pi^\top))^{-\frac{1}{2}}$$
$$= (p\mathbf{I} + (1-p)\mathbf{P}_\pi d(\mathbf{A})\mathbf{P}_\pi^\top)^{-\frac{1}{2}} \tag{27}$$

$$= (p\mathbf{I} + \mathbf{P}_\pi\left[(1-p)d(\mathbf{A})\right]\mathbf{P}_\pi^\top)^{-\frac{1}{2}} \tag{28}$$

$$= (\mathbf{P}_\pi\left[p\mathbf{I} + (1-p)d(\mathbf{A})\right]\mathbf{P}_\pi^\top)^{-\frac{1}{2}} \tag{29}$$

and since the matrix is diagonal

$$= \mathbf{P}_\pi(p\mathbf{I} + (1-p)d(\mathbf{A}))^{-\frac{1}{2}}\mathbf{P}_\pi^\top. \tag{30}$$

We also have

$$(\mathbf{P}_\pi\mathbf{A}\mathbf{P}_\pi^\top + q\mathbf{I}) = \mathbf{P}_\pi(\mathbf{A} + q\mathbf{I})\mathbf{P}_\pi^\top. \tag{31}$$

By Equation 30 and 31,

$$(p\mathbf{I} + (1-p)d(\mathbf{P}_\pi\mathbf{A}\mathbf{P}_\pi^\top))^{-\frac{1}{2}}(\mathbf{P}_\pi\mathbf{A}\mathbf{P}_\pi^\top + q\mathbf{I})$$
$$(p\mathbf{I} + (1-p)d(\mathbf{P}_\pi\mathbf{A}\mathbf{P}_\pi^\top))^{-\frac{1}{2}}$$
$$= \mathbf{P}_\pi(p\mathbf{I} + (1-p)d(\mathbf{A}))^{-\frac{1}{2}}\mathbf{P}_\pi^\top\mathbf{P}_\pi(\mathbf{A} + q\mathbf{I})\mathbf{P}_\pi^\top$$
$$\mathbf{P}_\pi(p\mathbf{I} + (1-p)d(\mathbf{A}))^{-\frac{1}{2}}\mathbf{P}_\pi^\top, \tag{32}$$

and since $\mathbf{P}_\pi$ is orthogonal, $\mathbf{P}_\pi^\top\mathbf{P}_\pi = \mathbf{I}$, so

$$= \mathbf{P}_\pi(p\mathbf{I}+(1-p)d(\mathbf{A}))^{-\frac{1}{2}}(\mathbf{A}+q\mathbf{I})(p\mathbf{I}+(1-p)d(\mathbf{A}))^{-\frac{1}{2}}\mathbf{P}_\pi^\top \tag{33}$$

□

Theorem 3.3. *For any input graph G, and any permutation π applied to G, the output of PiNet is equal, i.e.*

$$z(G) = z(\mathbf{P}_\pi G), \tag{34}$$

where $z : (\mathbb{R}^{N\times N}, \mathbb{R}^{N\times d}) \to \mathbb{R}^C$ is the forward pass function of PiNet given in Equation 5.

Proof. By Equation 1, Definition 3.3 and Lemma 3.2,

$$z_X(\mathbf{P}_\pi G) = z_X(\mathbf{P}_\pi\mathbf{A}\mathbf{P}_\pi^\top, \mathbf{P}_\pi\mathbf{X}) \tag{35}$$

$$= \sigma_R\left(\mathbf{P}_\pi\tilde{\mathbf{A}}\mathbf{P}_\pi^\top\ \sigma_R\left(\mathbf{P}_\pi\tilde{\mathbf{A}}\mathbf{P}_\pi^\top\ \mathbf{P}_\pi\mathbf{X}\mathbf{W}_X^{(0)}\right)\mathbf{W}_X^{(1)}\right) \tag{36}$$

$\mathbf{P}_\pi$ is orthogonal, so $\mathbf{P}_\pi^\top\mathbf{P}_\pi = \mathbf{I}$, giving

$$z_X(\mathbf{P}_\pi G)$$
$$= \sigma_R\left(\mathbf{P}_\pi\tilde{\mathbf{A}}\mathbf{P}_\pi^\top\ \sigma_R\left(\mathbf{P}_\pi\tilde{\mathbf{A}}\ \mathbf{X}\mathbf{W}_X^{(0)}\right)\mathbf{W}_X^{(1)}\right) \tag{37}$$

Since σ_R is an element-wise operation,

$$\sigma_R(\mathbf{P}_\pi\mathbf{X}) = \mathbf{P}_\pi \cdot \sigma_R(\mathbf{X}), \tag{38}$$

then

$$z_X(\mathbf{P}_\pi G)$$
$$= \sigma_R\left(\mathbf{P}_\pi\tilde{\mathbf{A}}\mathbf{P}_\pi^\top\ \mathbf{P}_\pi \cdot \sigma_R\left(\tilde{\mathbf{A}}\ \mathbf{X}\mathbf{W}_X^{(0)}\right)\mathbf{W}_X^{(1)}\right) \tag{39}$$

$$= \sigma_R\left(\mathbf{P}_\pi\tilde{\mathbf{A}}\ \sigma_R\left(\tilde{\mathbf{A}}\ \mathbf{X}\mathbf{W}_X^{(0)}\right)\mathbf{W}_X^{(1)}\right) \tag{40}$$

$$= \mathbf{P}_\pi \cdot \sigma_R\left(\tilde{\mathbf{A}}\ \sigma_R\left(\tilde{\mathbf{A}}\ \mathbf{X}\mathbf{W}_X^{(0)}\right)\mathbf{W}_X^{(1)}\right) \tag{41}$$

$$= \mathbf{P}_\pi \cdot z_X(\mathbf{A}, \mathbf{X}) \tag{42}$$

$$= \mathbf{P}_\pi \cdot z_X(G). \tag{43}$$

By the same logic as Equation 35 to 40,

$$z_A(\mathbf{P}_\pi G)$$
$$= \sigma_S\left(\left[\mathbf{P}_\pi\tilde{\mathbf{A}}\mathbf{P}_\pi^\top\ \sigma_R\left(\mathbf{P}_\pi\tilde{\mathbf{A}}\mathbf{P}_\pi^\top\ \mathbf{P}_\pi\mathbf{X}\mathbf{W}_X^{(0)}\right)\mathbf{W}_X^{(1)}\right]^\top\right) \tag{44}$$

$$= \sigma_S\left(\left[\mathbf{P}_\pi\tilde{\mathbf{A}}\ \sigma_R\left(\tilde{\mathbf{A}}\ \mathbf{X}\mathbf{W}_X^{(0)}\right)\mathbf{W}_X^{(1)}\right]^\top\right) \tag{45}$$

$$= \left[\sigma_{S'}\left(\mathbf{P}_\pi\tilde{\mathbf{A}}\ \sigma_R\left(\tilde{\mathbf{A}}\ \mathbf{X}\mathbf{W}_X^{(0)}\right)\mathbf{W}_X^{(1)}\right)\right]^\top, \tag{46}$$

where $\sigma_{S'}$ is a column-wise softmax

$$\sigma_{S'}(\mathbf{X})_{ij} = \frac{e^{\mathbf{X}_{ij}}}{\sum_k e^{\mathbf{X}_{ik}}}. \tag{47}$$

When the rows of its input are permuted by π,

$$\sigma_{S'}(\mathbf{P}_\pi\mathbf{X})_{ij} = \frac{e^{\mathbf{X}_{\pi^{-1}(i)j}}}{\sum_k e^{\mathbf{X}_{\pi^{-1}(i)k}}} \tag{48}$$

we observe that rows of the output are also permuted by π, thus by Equation 46 and 48

$$z_A(\mathbf{P}_\pi G) = \left[\mathbf{P}_\pi \cdot \sigma_{S'}\left(\tilde{\mathbf{A}}\ \sigma_R\left(\tilde{\mathbf{A}}\ \mathbf{X}\mathbf{W}_X^{(0)}\right)\mathbf{W}_X^{(1)}\right)\right]^\top. \tag{49}$$

From Equation 5

$$z(\mathbf{P}_\pi G) = \sigma_S\left[g\left(z_A\left(\mathbf{P}_\pi G\right) \cdot z_X\left(\mathbf{P}_\pi G\right)\right)\mathbf{W}_D\right], \tag{50}$$

and by Equation 43 and 49 we see that

$$z_A\left(\mathbf{P}_\pi G\right) \cdot z_X\left(\mathbf{P}_\pi G\right) = \left[\mathbf{P}_\pi \cdot \sigma_{S'}\left(\tilde{\mathbf{A}}\, \sigma_R\left(\tilde{\mathbf{A}}\, \mathbf{X}\mathbf{W}_X^{(0)}\right)\mathbf{W}_X^{(1)}\right)\right]^\top \cdot \left[\mathbf{P}_\pi \cdot z_X(G)\right], \quad (51)$$

which by Lemma 3.1

$$= \left[\sigma_{S'}\left(\tilde{\mathbf{A}}\, \sigma_R\left(\tilde{\mathbf{A}}\, \mathbf{X}\mathbf{W}_X^{(0)}\right)\mathbf{W}_X^{(1)}\right)\right]^\top \cdot \left[z_X(G)\right] \quad (52)$$

$$= \sigma_S\left(\left[\tilde{\mathbf{A}}\, \sigma_R\left(\tilde{\mathbf{A}}\, \mathbf{X}\mathbf{W}_X^{(0)}\right)\mathbf{W}_X^{(1)}\right]^\top\right) \cdot \left[z_X(G)\right] \quad (53)$$

$$= z_A(G) \cdot z_X(G) \quad (54)$$

Finally, by Equation 50 and 51 to 54,

$$z(\mathbf{P}_\pi G) = \sigma_S\left[g\left(z_A\left(G\right) \cdot z_X\left(G\right)\right)\mathbf{W}_D\right] \quad (55)$$

$$= z(G). \quad (56)$$

□

3.3 Implementation

To implement PiNet we use Keras + Tensorflow. The model operates on batches and uses a mixture of Scipy sparse matrices and Numpy arrays to represent the graphs and their features. Full source code for PiNet is available at LINK(reveals authors).

4 Experiments

We conduct three experiments: We empirically verify the utility of our model's invariance to permutation with a graph isomorphism classification task, we evaluate the effect of different message passing matrices and the model's ability to select an appropriate message passing configuration, and we compare the model's classification performance against existing graph classifiers on a set of standard molecule classification datasets. We next describe the data used followed by a description of each experiment.

4.1 Datasets

For the isomorphism test, we generate a dataset of 500 graphs. To create sufficient challenge, in each randomly sampled Erdos Reny [Erdõs and Rényi, 1960] graph, we fix the number of nodes, and the node degree distributions, to be constant. We generate the dataset according to Algorithm 1, with the parameters: $N = 50$, $C = 5$, $N_g = 100$, and $p = 0.15$.

For the final two experiments we use the binary classification molecule datasets detailed in Table 2.

	MUTAG	NCI1	NCI109	PTC	PROTEINS
$\lvert\mathcal{G}\rvert$	188	4110	4127	344	1113
Max. $\lvert V\rvert$	28	111	111	109	620
Mean $\lvert V\rvert$	18	29.8	29.6	25.56	39.06
d	7	37	38	18	3
% of +ve	66.49	50.05	50.38	39.51	59.57

Table 2: Binary classification molecule datasets. $|\mathcal{G}|$ is the number of graphs, $|V|$ the number of nodes, and d the dimensions of the node features.

Algorithm 1 Graph Dataset Generation

Input: Number of nodes N, number of classes C, number of graphs per class N_g, edge probability p
seedGraph ← SampleErdosRenyiGraph(N, p)
while seedGraph not fully connected **do**
  SampleErdosRenyiGraph(N, p)
end while
D ← Array[]
for each c **in** C **do**
  G_c ← Array[]
  S ← getDegreeSequence(seedGraph)
  sampleGraph ← GenerateGraph(S)
  for each i **in** N_g **do**
    p_g ← permute(sampleGraph.adjacencyMatrix)
    p_g ← relabel(pg.nodes)
    $G_c[i]$ ← p_g
  end for
  $D[c]$ ← G_c
end for
Output: D

4.2 Isomorphism Test

We test PiNet's ability to recognise isomorphic graphs - specifically, unseen permutations of given training graphs. For a baseline, we test against two variants of the GCN [Kipf and Welling, 2016], one in which the graph level representation is given by a sum of the node representations, and the other in which we apply a dense layer directly to the node level outputs of the GCN. We also compare against two state of the art graph classifiers: the WL Kernel [Shervashidze *et al.*, 2011] and PATCHY-SAN [Niepert *et al.*, 2016]. We perform 10 trials for every training sample size.

4.3 Message Passing Mechanisms

We study the impact on classification accuracy of the four matrices shown in Table 1 that facilitate message passing between nodes, alongside the parametrised matrix shown in Equation 3, in which the extent to which to normalise neighbours' values and include the node's own state as input are controlled by the learned parameters p and q. Note that p and q are learned for each message passing layer and are not required to be the same, however, for each of the matrices from Table 1 we test, we use the same matrix in all four message passing layers.

For each matrix, we perform a 10-fold cross validation and record the classification accuracy. For all runs we use the following hyper-parameters: batch size = 50, epochs = 200, first layer latent feature size $F_X^{(0)}$= $F_A^{(0)}$= 100, second layer latent feature size $F_X^{(1)}$= $F_A^{(1)}$= 64, and learning rate = 10^{-3}.

4.4 Comparison Against Existing Methods

As with the isomorphism test, we compare against the GCN with a dense layer applied directly to the node outputs as well as with a sum of the node representations, the Weisfeiler Lehman graph kernel, and PATCHY-SAN. For each model we perform 10-fold cross validation on each dataset.

For PiNet, we use the same hyper-parameters as described in Section 4.3. We test the model with p and q as trainable values, and also search over the space $(p, q) \in \{0, 1\} \times \{0, 1\}$. For the GCN we use two layers of sizes 100 and 64 hidden units, with a learning rate of 10^{-3}, and for PATCHY-SAN we search over two labelling procedures, betweenness centrality [Brandes, 2001] and NAUTY [McKay, 1981] canonical labelling.

5 Results

5.1 Isomorphism Test

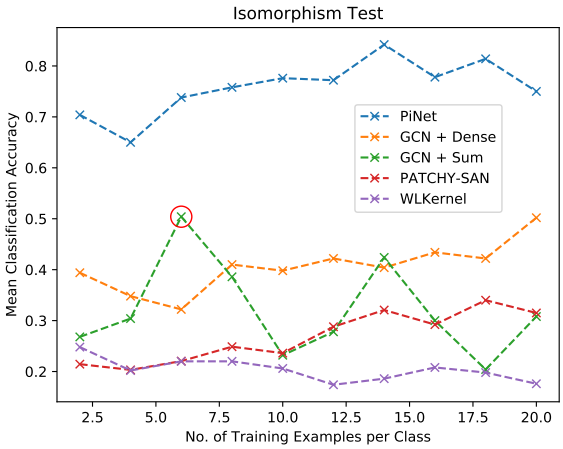

Figure 3: Mean classification accuracy on isomorphic graph classes.

As seen in Figure 3, PiNet outperforms all competitors tested. Using an independent two-sample t-test we observe statistical significance (p-value < 0.05) in all cases except a single point (circled in red). PiNet fails to achieve 100% accuracy since the neural network learns a surjective function, thus with so few training examples in some cases complete multiple classes become indistinguishable.

5.2 Message Passing Mechanisms

In Figure 4 we observe that the optimal message passing matrix (when p and q are fixed for all layers, and $(p, q) \in \{0, 1\} \times \{0, 1\}$) varies depending on the particular set of graphs given. With p and q as trainable parameters ($(p, q) \in [0, 1] \times [0, 1]$, and (p, q) may be different for each layer), we see that for the MUTAG and PROTEINS datasets the model learns values that outperform those found with our manual search. For the others, however, the model is unable to find the optimal ps and qs, suggesting that the model finds only local minima. We note however, that in every case tested, the model is able to learn the values p and q that give better than average classification performance when compared with our manual search space.

5.3 Comparison Against Existing Methods

As shown in Table 3, PiNet achieves competitive classification performance on all datasets tested. An independent two-sample t-test indicates PiNet achieves a statistically significant (p-value < 0.05) gain in only a few cases (*); however, with competitive performance on all datasets it demonstrates a robustness to the properties of the different sets of graphs.

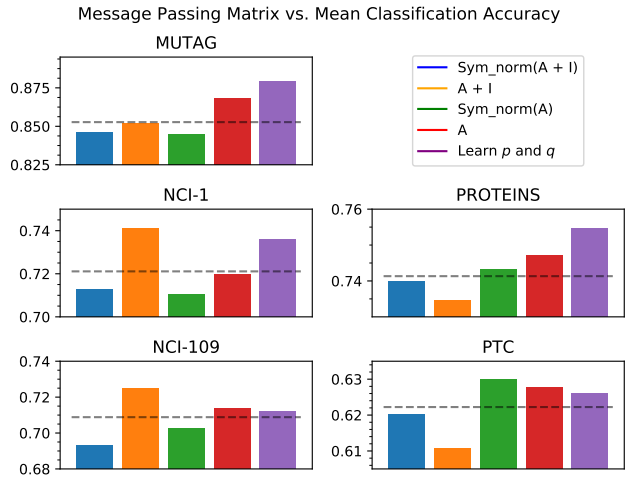

Figure 4: Each plot shows the mean classification accuracy for each message passing matrix of our search space, alongside the accuracy of PiNet when p and q are learned during training. The dashed lines indicate the mean accuracy of the manual search.

6 Conclusion

We have proposed PiNet, an end-to-end deep neural network graph classifier invariant to permutations of nodes in the input graphs. We have provided theoretical proof of its invariance to permutation, and demonstrated the utility in such a property empirically with a graph isomorphism classification task against a set of existing graph classifiers, achieving a statistically significant gain in classification accuracy on a range of small training set sizes. The permutation invariance is achieved through a differentiable attention mechanism in which the model learns the weight by which the states associated with each node should be aggregated into the final graph representation.

We have demonstrated that PiNet is able to learn an effective parametrisation of a message passing matrix that enables it to adapt to different types of graphs with a flexible state propagation and diffusion mechanism. Finally, we have shown PiNet's robustness to the properties of different sets of graphs in achieving consistently competitive classification performance against a set of existing techniques on five commonly used molecule datasets.

For future work we plan to explore more advanced aggregation mechanisms by which the latent representations learned for each node of the graph may be combined.

Acknowledgements

We thank Braintree Ltd. for providing the full funding for this work.

	MUTAG	NCI-1	NCI-109	PROTEINS	PTC
GCN + Dense	$.86 \pm .06$	$.73 \pm .03$	$.72 \pm .02$	$.71 \pm .04$	$.63 \pm .07$
GCN + Sum	$.86 \pm .05$	$.72 \pm .03$	$.73 \pm .03$	$.74 \pm .04$	$.61 \pm .05$
PATCHY-SAN	$.85 \pm .06$	$.58 \pm .02$	$.58 \pm .03$	$.70 \pm .02$	$.58 \pm .02$
WLKernel	$.68 \pm .00^*$	$.53 \pm .02^*$	$.53 \pm .03^*$	$.61 \pm .01^*$	$.62 \pm .03$
PiNet (Manual p and q)	$.87 \pm .08$	$.74 \pm .03$	$.73 \pm .03$	$.75 \pm .06$	$.63 \pm .06$
PiNet (Learned p and q)	$.88 \pm .07$	$.74 \pm .02$	$.71 \pm .04$	$.75 \pm .06$	$.63 \pm .04$

Table 3: Mean classification accuracies for each classifier. For manual search the values p and q as follows: MUTAG and PROTEINS $p = 1, q = 0$, NCI-1 and NCI-109 $p = q = 1$, PTC $p = q = 0$. * indicates PiNet (both models) achieved statistically significant gain.

References

[Atwood and Towsley, 2016] James Atwood and Don Towsley. Diffusion-Convolutional Neural Networks. In *30th Conference on Neural Information Processing Systems (NIPS 2016)*, 2016.

[Borgwardt *et al.*, 2005] Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V.N. Vishwanathan, Alex J. Smola, and Hans Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 2005.

[Brandes, 2001] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.

[Erdõs and Rényi, 1960] P Erdõs and A Rényi. On evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, (5):17–61, 1960.

[Gutierrez Mallea *et al.*, 2019] Marcelo Daniel Gutierrez Mallea, Peter Meltzer, and Peter J Bentley. Capsule Neural Networks for Graph Classification using Explicit Tensorial Graph Representations. 2019.

[Hamilton *et al.*, 2017] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. In *31st Conference on Neural Information Processing Systems (NIPS 2017)*, 2017.

[Harchaoui and Bach, 2007] Zaïd Harchaoui and Francis Bach. Image classification with segmentation graph kernels. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007.

[Kipf and Welling, 2016] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. sep 2016.

[Kriege *et al.*, 2015] Nils Kriege, Marion Neumann, Kristian Kersting, and Petra Mutzel. Explicit Versus Implicit Graph Feature Maps: A Computational Phase Transition for Walk Kernels. *Proceedings - IEEE International Conference on Data Mining, ICDM*, 2015-Janua(January):881–886, 2015.

[Li and Zemel, 2016] Yujia Li and Richard Zemel. GATED GRAPH SEQUENCE NEURAL NETWORKS. In *International Conference on Learning Representations (ICLR)*, 2016.

[McKay, 1981] Brendan D McKay. Practical graph isomorphism, 1981.

[Neuhaus *et al.*, 2009] Michel Neuhaus, Kaspar Riesen, and Horst Bunke. Novel kernels for error-tolerant graph classification. *Spatial Vision*, 22(5):425–441, sep 2009.

[Niepert *et al.*, 2016] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning Convolutional Neural Networks for Graphs. 1, 2016.

[Romero, 2018] Adriana Romero. Graph Attention Networks. In *International Conference on Learning Representations (ICLR)*, number 2015, pages 1–11, 2018.

[Scarselli *et al.*, 2009] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 20(1):61–80, 2009.

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

[Shervashidze *et al.*, 2010] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Karsten M Borgwardt, Kurt Mehlhorn, Karsten M Borgwardt Shervashidze, and Van Leeuwen. Weisfeiler-Lehman graph kernels. Technical report, 2010.

[Shervashidze *et al.*, 2011] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011.

[Verma and Zhang, 2018] Saurabh Verma and Zhi-Li Zhang. Graph Capsule Convolutional Neural Networks. 2018.

[Weisfeiler and Lehman, 1968] B Yu Weisfeiler and A. A. Lehman. Reduction of a graph to a canonical form and an algebra which appears in the process. *Nauchno-Technicheskaya Informatsiya, Ser. 2*, 9:12, 1968.

[Ying *et al.*, 2018] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical Graph Representation Learning with Differentiable Pooling. *32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.*, 2018.

[Zhang *et al.*, 2018] Zhen Zhang, Mianzhi Wang, Yijian Xiang, Yan Huang, and Arye Nehorai. RetGK: Graph Kernels based on Return Probabilities of Random Walks. *32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.*, 2018.