arXiv:1905.06259v1 [cs.LG] 15 May 2019

Function Space Pooling For Graph Convolutional Networks

Padraig Corcoran

School of Computer Science and Informatics
Cardiff University

May 16, 2019

Abstract

Convolutional layers in graph neural networks are a fundamental type of layer which output a representation or embedding of each graph vertex. The representation typically encodes information about the vertex in question and its neighbourhood. If one wishes to perform a graph centric task such as graph classification the set of vertex representations must be integrated or pooled to form a graph representation. We propose a novel pooling method which transforms a set of vertex representations into a function space representation. Experiential results demonstrate that the proposed method outperforms standard pooling methods of computing the sum and mean vertex representation.

1 Introduction

Many real world data such as social networks, collections of documents and chemical structures are naturally represented as graphs. Consequently there exists great potential for the application of machine learning to graphs. Given the great successes of neural networks or deep learning to the analysis of images, there has recently been much research considering the application or generalization of neural networks to graphs. In many cases this has resulted in state of the art performance in many tasks (Wu et al., 2019).

Graph convolutional is a neural network architecture commonly applied to graphs. This architecture consists of a sequence of convolutional layers where each layer iteratively updates a representation or embedding of each vertex. This update is achieved through the application of an operation which considers the current representation of each vertex plus the current representation of its adjacent neighbours (Gilmer et al., 2017). The output of a sequence of convolutional layers is a representation of each vertex which encodes properties of the vertex in question and vertices in its neighbourhood.

If one wishes to perform a vertex centric task such as vertex classification, then one may operate directly on the set of vertex representations output from a sequence of convolutional layers. However, if one wishes to perform a graph centric task such as graph classification, then the set of vertex representations

must somehow be integrated or pooled to form a graph representation. Pooling represents a challenging problem because there exists no vertex ordering and different graphs may have a different number of vertices. Commonly employed pooling methods include computing the mean or sum of vertex representations. However these simple pooling methods are not a *complete invariant* in the sense that many different sets of vertex representations may result in the same graph representation leading to weak discrimination power (Xu et al., 2018). To overcome this issue and increase discrimination power a number of authors have proposed more sophisticated pooling methods. For example Ying et al. (2018) proposed a pooling method which performs a hierarchical clustering of vertex representations.

In this article we propose a novel pooling method which maps a set of vertex representations to a function space representation which forms a vector space. This method is parameterized by a single learnable parameter which controls the discrimination power of the method. This makes the method applicable to both finer and coarser classification tasks which require greater and less discrimination power respectively. The proposed pooling method is inspired by related methods in the field of applied topology which map sets of points in $\mathbb{R}^2$ to a function space representation (Adams et al., 2017).

The layout of this paper is as follows. Section 2 reviews related works on pooling methods. Section 3 describes the proposed pooling method. Section 4 presents an evaluation of this method. Finally section 5 draws some conclusions from this work.

2 Related Pooling Methods

The simplest and mostly commonly used pooling methods involve computing basic summary statistics such as mean and sum of vertex representations (Duvenaud et al., 2015). To improve discrimination power more sophisticated pooling methods have been proposed. The SortPooling method first sorts the vertices with respect to structural roles in the graph (Wu et al., 2019). The vertex representations corresponding to the first k vertices in this order are then used as input to a traditional one dimensional convolutional network. The value k is a fixed hyper-parameter in the model. Set2set is a general approach for embedding a set in a manner which is invariant to element order (Vinyals et al., 2015). Gilmer et al. (2017) proposed to use this method to perform pooling. Ying et al. (2018) proposed a pooling method which performs a hierarchical clustering of vertex representations. Kearnes et al. (2016) proposed a pooling method based on fuzzy histograms. This method has similarities to that proposed in this article but is formulated is in terms of fuzzy theory as opposed to function spaces. The method proposed in this article is in turn distinct. All of the above pooling methods are supervised methods. Bai et al. (2019) proposed a pooling method which is unsupervised.

3 Function Space Pooling

Let graph $G = (V, E)$ be a graph where V and E are the corresponding sets of vertices and edges respectively. Let D be the set of vertex representations

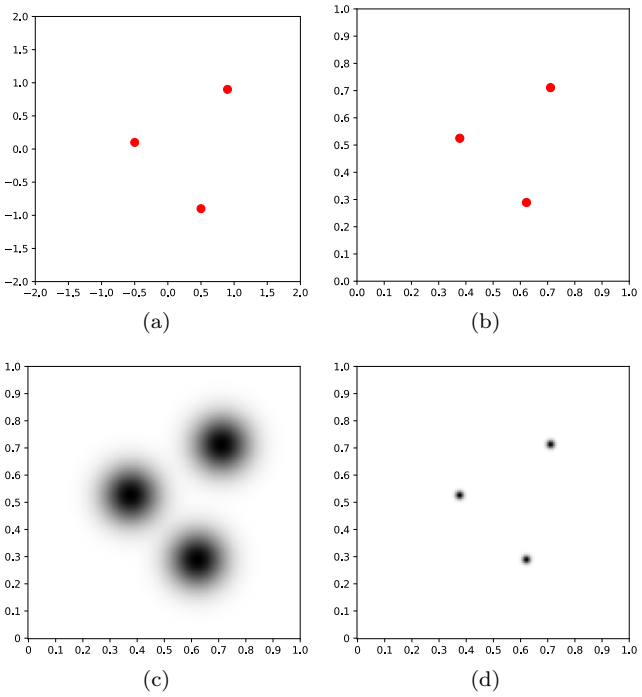

Figure 1: A set D of vertex representations output from a sequence of convolutional layers is displayed in (a) where each element is represented by a red dot. The result of applying the map S to the set D is the set $s(D)$ displayed in (b). The result of applying the map F to $S(D)$ with the parameter $\sigma = 0.005$ is the function $\rho : I \to \mathbb{R}$ displayed in (c). The result of applying the map F to $S(D)$ with the parameter $\sigma = 0.0001$ is the function $\rho : I \to \mathbb{R}$ displayed in (d).

output from a sequence of convolutional layers applied to G. We assume that each vertex representation is an element of $\mathbb{R}^n$. The proposed pooling method takes as input D and returns a function. That is, the method is a map from the space of sets to the space of functions. It contains two steps which we now describe in turn.

The set of vertex representations D is an object in the category of sets which we denote Ω. Let Sigmoid : $\mathbb{R}^n \to I$ be the n-dimensional Sigmoid function defined in Equation 1 where $I = \{(x_1, \ldots, x_n) \in [0,1]^n\}$ is the n-dimensional interval. In the first step of the proposed pooling method we apply the n-dimensional Sigmoid elementwise to D to give a map $S : \Omega \to \Omega$. To illustrate this map consider Figure 1(a) which displays an example set D containing three elements in $\mathbb{R}^2$. The result of applying the map S to this set is illustrated in Figure 1(b).

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

Let $g_u : \mathbb{R}^n \to \mathbb{R}$ be a probability distribution. For the purposes of this work we used the n-dimensional Gaussian distribution defined in Equation 2 with mean u and variance σ^2.

$$g_u(x) = \frac{1}{2\pi\sigma^2} e^{-((x-u)^T(x-u))/2\sigma^2} \tag{2}$$

In the second step of the proposed pooling method we apply a map $F : \Omega \to L^p(I)$ to $S(D)$. Here $L^p(I)$ is the vector space of functions $I \to \mathbb{R}$ where the p-th power of the absolute value is Lebesgue integrable and is equipped with the L^p-norm defined in Equation 3 (Christensen, 2010). Note that, function addition and subtraction is performed pointwise.

$$\|f\|_p = \left(\int_I |f(x)|^p dx \right)^{1/p} \tag{3}$$

The function resulting from the map F is defined in Definition 1. To illustrate this map consider again the example set $S(D)$ illustrated in Figure 1(b). Figure 1(c) displays the function $\rho : I \to \mathbb{R}$ resulting from applying the map F to this set with a σ parameter value of 0.005.

Definition 1. *For $D \in \Omega$ the corresponding function representation $\rho : I \to \mathbb{R}$ is defined in Equation 4*

$$\rho(z) = \sum_{u \in S(D)} g_u(z) \tag{4}$$

The elements of $L^p(I)$, and in turn the function representation $\rho : I \to \mathbb{R}$, are infinite dimensional vector spaces. That is, there are an infinite number of elements in the domain I of ρ. We approximate this function as a finite dimensional vector space by discretizing the function domain using a regular grid of elements. For example, the image in Figure 1(c) corresponds to a discretizing of the function domain using a 250×250 grid.

The proposed pooling method is parameterized by σ in the probability distribution of Equation 2 where this parameter is in the range $[0, \infty]$. As the value of σ approaches 0 the probability distribution approaches an indicator function on the domain I. On the other hand, as the value of σ approaches ∞

the probability distribution approaches a uniform function on the domain I. For example, Figures 1(c) and 1(d) display the functions $\rho : I \to \mathbb{R}$ resulting from applying the map F to the set $S(D)$ in Figure 1(b) with σ parameter values of 0.005 and 0.0001 respectively.

The parameter σ may be interpreted in a couple of ways. As the value of σ approaches 0 the function representation $\rho : I \to \mathbb{R}$ approaches a complete invariant. That is, distinct sets D map to distinct functions where the distance between these functions as defined by the norm in Equation 3 is greater than zero. On the other hand, as σ approaches ∞, the distance between these functions reduces. An alternative way of interpreting the parameter σ is as follows. As mentioned above, as the value of σ approaches ∞ the probability distribution approaches a uniform function on the domain I. In this case the proposed pooling method differs from computing the sum of the elements of D by a multiplicative constant only.

4 Results

To evaluate the performance of the proposed pooling method we considered the task of graph classification. The layout of this section is as follows. Section 4.1 describes the datasets considered. Section 4.2 describes the feed-forward network architecture used in all experiments. Section 4.3 describes the optimization method used to optimize the network parameters. Finally section 4.4 presents the classification accuracy achieved by the proposed pooling method relative to two benchmark methods.

4.1 Datasets

The first dataset consider was the MUTAG dataset which consists of 188 chemical compounds where the classification problem is binary and concerns predicting if a chemical compound has mutagenicity or not (Debnath et al., 1991). Each chemical compound is represented as a graph where there are 7 distinct types of vertices.

The second dataset consider was the PROTEINS dataset which consists of 1113 proteins where the classification task is binary and concerns predicting if a protein is an enzyme or not (Borgwardt et al., 2005). Each protein is represented as a graph where there are 3 distinct types of vertices. Both of the datasets considered are commonly used to evaluate graph neural networks (Fey and Lenssen, 2019).

4.2 Network Architecture

The feed-forward network architecture used consists of the following six layers. The first two layers are convolutional layers. A number of studies have found that two convolutional layers empirically gives best performance (Kipf and Welling, 2016). The third layer is a fully connected linear layer. The fourth layer is the pooling method used. The fifth layer is another fully connected linear layer. The final layer is a softmax function.

The convolutional layers used are similarly to the GraphSAGE convolutional layers (Hamilton et al., 2017). Let h_k denote the matrix containing the vertex

representation in the kth convolutional layer. Each matrix row corresponds to the representation of an individual vertex. Let $\cdot$ denote matrix multiplication and CONCAT denote horizontal matrix concatenation. The kth convolutional layer is implemented using Equation 5 where W_k and b_k are the corresponding weights and biases respectively. The weights W_k is a matrix of dimension $2d_{k-1} \times d_k$ where d_k the dimension of the kth layer. The biases b_k is a vector of dimension d_k. The dimension of the input layer d_0 is equal to the number of vertex types since one-hot encoding was used. The dimensions of the two convolutional layers d_1 and d_2 were both set to 20.

$$\begin{aligned} h_k &\leftarrow \text{CONCAT}(h_{k-1}, A \cdot h_{k-1}) \\ h_k &\leftarrow \text{ReLu}\left(W_k \cdot h_k + b_k\right) \end{aligned} \tag{5}$$

The dimensions of the first and second linear layers were set to 10 and 20 respectively. The output of the first linear layer is the input to the pooling method. Therefore the multi-dimensional interval corresponding to the domain of the function ρ in Definition 1 is of dimension 10. We approximate this function as a finite dimensional vector space by discretizing the function domain using a regular grid with 3 elements in each dimension. This gives a finite dimensional vector space of size $3^{10} = 59049$.

4.3 Optimization

The model parameters to be optimized in the architecture of section 4.2 are the weights and biases of the convolutional and linear layers plus the parameter σ of the pooling method. For loss function a Cross Entropy loss term plus an L^2 regularization term was used. In all experiments a regularization weight of 0.2 was used. The Adam optimization algorithm was used to optimize all model parameters with a learning rate of 1×10^{-3}. In all experiments optimization was performed using 350 epochs.

Pooling Method	**MUTAG**	**PROTEINS**
Sum	65.6 ± 13	60.1 ± 18
Mean	78.1 ± 18	57.7 ± 16
Function Space	$\mathbf{83.3 \pm 11}$	$\mathbf{72.8 \pm 19}$

Table 1: For each of the MUTAG and PROTEINS datasets, the mean classification accuracy of 10-fold cross validation for each pooling method are displayed.

4.4 Classification Accuracy

The proposed pooling method was benchmarked against the methods of computing the mean and sum of vertex representations. As discussed in section 2, these are some of the most commonly used pooling methods. For each benchmark pooling method the corresponding network architecture was identical to that described in section 4.2 with the exception that the pooling layer was replaced and the dimension of the linear layer before this layer was changed from 10 to 20. For both datasets considered we computed the mean accuracy of

10-fold cross validation for each pooling method and the results of this analysis are displayed in Table 1. For both datasets, the proposed pooling method outperformed both benchmark methods.

5 Conclusions

We propose a novel pooling method for convolutional layers in graph neural networks which involves computing a function space representation of vertex representations. Experimental results demonstrate the proposed method outperforms the commonly employed pooling methods of computing the mean and sum of vertex representations.

References

Henry Adams, Tegan Emerson, Michael Kirby, Rachel Neville, Chris Peterson, Patrick Shipman, Sofya Chepushtanova, Eric Hanson, Francis Motta, and Lori Ziegelmeier. Persistence images: A stable vector representation of persistent homology. *The Journal of Machine Learning Research*, 18(1):218–252, 2017.

Yunsheng Bai, Hao Ding, Yang Qiao, Agustin Marinovic, Ken Gu, Ting Chen, Yizhou Sun, and Wei Wang. Unsupervised inductive whole-graph embedding by preserving graph proximity. *arXiv preprint arXiv:1904.01098*, 2019.

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

Ole Christensen. *Functions, spaces, and expansions: mathematical tools in physics and engineering*. Springer Science & Business Media, 2010.

Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2): 786–797, 1991.

David Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

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

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

Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8):595–608, 2016.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *arXiv preprint arXiv:1806.08804*, 2018.