



Function Space Pooling for Graph Convolutional Networks

Padraig Corcoran

► To cite this version:

Padraig Corcoran. Function Space Pooling for Graph Convolutional Networks. 4th International Cross-Domain Conference for Machine Learning and Knowledge Extraction (CD-MAKE), Aug 2020, Dublin, Ireland. pp.473-483, 10.1007/978-3-030-57321-8_26 . hal-03414745

HAL Id: hal-03414745

<https://inria.hal.science/hal-03414745>

Submitted on 4 Nov 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

Function Space Pooling For Graph Convolutional Networks

Padraig Corcoran

School of Computer Science and Informatics,
Cardiff University, Wales, UK.
`corcoranp@cardiff.ac.uk`

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, this set of vertex representations must be integrated or pooled to form a graph representation. In this article we propose a novel pooling method which maps a set of vertex representations to a function space representation. This method is distinct from existing pooling methods which perform a mapping to either a vector or sequence space. Experimental graph classification results demonstrate that the proposed method generally outperforms most baseline pooling methods and in some cases achieves best performance.

Keywords: graph neural network · vertex pooling · function space.

1 Introduction

Many real world systems have a relational structure which can be modelled as a graph. These include physical systems where the bodies and joints correspond to the vertices and edges respectively [20]; robot swarms where robots and communication links correspond to the vertices and edges respectively [21]; and topological maps where locations and paths correspond to the vertices and edges respectively [4]. Given this, there exists great potential for the application of machine learning to graphs. With the great successes of neural networks and deep learning to the analysis of images and natural language, 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 for many tasks [25].

Graph convolutional is a neural network architecture commonly applied to graphs which consists of a sequence of convolutional layers. The output of a sequence of such layers is a set of vertex representations where each element in this set encodes properties of a corresponding vertex and the vertices in its neighbourhood. In their seminal work, Gilmer et al. [9] showed that many different types of convolutional layers can be formulated in terms of a framework containing two steps. In the first step message passing is performed where each

vertex receives messages from adjacent vertices regarding their current representation. In the second step, each vertex performs an update of its representation which is a function of its current representation and the messages it received in the previous step. Graph convolution is fundamentally different to the more commonly used image convolution. Unlike an image where each pixel will have an equal number of adjacent pixels (excluding boundary pixels), each vertex in a graph may have a different number of adjacent vertices. Furthermore, unlike an image where the set of pixels adjacent to a given pixel can be ordered, the set of vertices adjacent to a given vertex cannot be easily ordered. Given these facts, generalizing image convolution methods to graphs is non-trivial.

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 to form a graph representation. We refer to this integration step as *pooling* and it represents the focus of this article. Note that, this step is sometimes referred to as global pooling. Performing pooling represents a challenging problem for a couple of reasons. Firstly, the size of the set of vertex representations will equal the number of vertices in the graph in question and this number will vary from graph to graph. Furthermore, the elements in this set will not be ordered. Therefore the set of vertex representations cannot be directly fed as input to feed-forward or recurrent architecture which require as input an element in a vector space of fixed dimension and an element in a sequence space respectively.

Commonly employed pooling methods include computing summary statistics of the set of vertex representations such as the mean or sum. 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 [27]. To overcome this issue and increase discrimination power a number of authors have proposed more sophisticated pooling methods. For example, Ying et al. [29] proposed a pooling method which performs a hierarchical clustering of the set of vertex representations to produce an element in a vector space of fixed dimension.

In this article we propose a novel pooling method which maps a set of vertex representations to a function space representation. This method is illustrated in Figure 1 in the context of a complete graph classification architecture. The proposed pooling 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 function space representations [1].

The layout of this paper is as follows. Section 2 reviews related works on graph convolution architectures and pooling methods. Section 3 describes the proposed

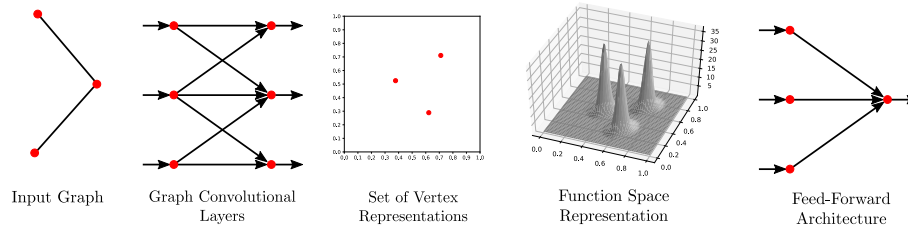


Fig. 1. The proposed pooling method is illustrated in the context of a complete graph classification architecture. The input graph is first fed to a sequence of graph convolutional layers which outputs a set of vertex representations. The number of elements in this set equals the number of vertices in the original graph. This set is next mapped to a function space representation. This function space representation is then fed to a feed-forward architecture which outputs a predicted graph class.

pooling method. Section 4 presents an evaluation of this method. Finally section 5 draws some conclusions from this work.

2 Background & Related Works

In the following two subsections we review related works on graph convolution architectures and pooling methods.

2.1 Graph Convolution Architectures

There exist a wide array of graph convolution architectures. In this section we only review those architectures representing theoretical breakthroughs and state of the art. However the interested reader can consult the following review papers for greater details [31, 26]. Hamilton et al. [10] proposed a graph convolution layer known as GraphSAGE which updates a vertex representation by first performing an aggregation of adjacent vertex representations. This aggregation is then concatenated with the current representation of the vertex in question before applying a linear transformation and non-linearity. The authors considered the aggregation functions of mean vertex representation and LSTM (Long Short-Term Memory) applied to a random ordering of vertex presentations. Xu et al. [27] proposed to apply a multi-layer perceptron, as opposed to a single layer which is most common, to the aggregation of adjacent vertex representations and demonstrated that this improve discrimination power. In a later work the same authors [28] proposed an architecture known as the jumping knowledge architecture which allows vertices to aggregate information from neighbouring vertices over different ranges. The authors showed that this architecture allows deeper convolutional architectures to be used and outperforms the use of residual connections commonly used in computer vision applications [12]. Given the successes of attention based architectures in natural language processing [22],

Velickovic et al. [23] proposed an attention based architecture for graphs. For a given vertex this architecture allows different weights to be specified for different adjacent vertices.

2.2 Pooling Methods

There exist two main categories of pooling methods: those which map the set of vertex representations to a vector space of fixed dimension and those which map the set of vertex representations to a sequence space. The output of these mappings can then be fed as input to a feed-forward or recurrent architecture respectively. We now review pooling methods belonging to each of these categories.

The simplest pooling methods for mapping to a vector space of fixed dimension involve computing summary statistics such as mean and sum of vertex representations [7]. Despite the simple nature of these methods, a recent study by Luzhnica et al. [18] demonstrated that in some cases they can outperform more complex methods. To improve discrimination power more sophisticated pooling methods have been proposed. The SortPooling method proposed by Zhang et al. [30] first sorts the vertices with respect to structural roles in the graph. The vertex representations corresponding to the first k vertices in this order are then concatenated to give a fixed dimensional vector. The value k is a fixed hyper-parameter in the model. Set2Set is a general approach for producing a fixed dimensional vector space representation of a set which is invariant to the order in which the elements are processed [24]. Gilmer et al. [9] proposed to use this method to perform pooling. Ying et al. [29] proposed a pooling method known as DiffPool which performs a hierarchical clustering of vertex representations and returns an element in a fixed dimensional vector space. Kearnes et al. [13] proposed a pooling method based on fuzzy histograms. This method has similarities to that proposed in this article but is formulated in terms of fuzzy theory as opposed to function spaces. The method proposed in this article is in turn distinct. Tarlow et al. [17] proposed a pooling method which outputs an element in sequence space. Finally, all of the above pooling methods are supervised methods. Many unsupervised pooling methods have also been proposed but we do not review them here [2].

3 Function Space Pooling

In this section we present the proposed pooling method. Let graph $G = (V, E)$ denote a graph we wish to classify where V and E are the corresponding sets of vertices and edges respectively. Let $l : V \rightarrow \Sigma$ denote a vertex labelling function that assigns each vertex $v \in V$ a label $l(v)$ in the finite set Σ .

Let D be the set of vertex representations output from a sequence of convolutional layers applied to G . We assume that each element in this set is an element of \mathbb{R}^n where n is a fixed hyper-parameter. The proposed pooling method takes as input D and returns an element in a function space. That is, the method is a

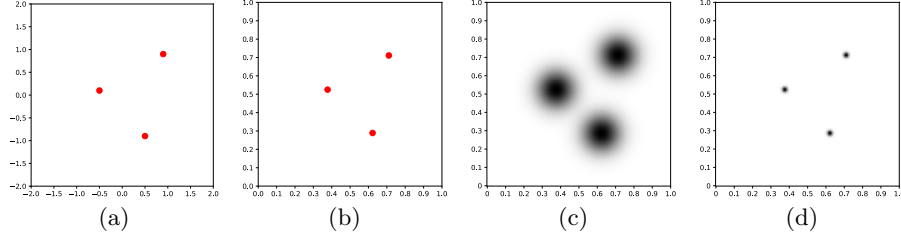


Fig. 2. A set D of vertex representations output from a sequence of convolutional layers is displayed in (a) where each element is represented by a red dot. The result of applying the map S to the set D is the set $s(D)$ displayed in (b). The result of applying the map F to $S(D)$ with the parameter $\sigma = 0.005$ is the function $\rho : I \rightarrow \mathbb{R}$ displayed in (c). The result of applying the map F to $S(D)$ with the parameter $\sigma = 0.0001$ is the function $\rho : I \rightarrow \mathbb{R}$ displayed in (d).

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 space of sets which we denote Ω . Let $\text{Sigmoid} : \mathbb{R}^n \rightarrow I$ be the n -dimensional Sigmoid function defined in Equation 1 where $I = \{(x_1, \dots, x_n) \in [0, 1]^n\}$ 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 $S : \Omega \rightarrow \Omega$. To illustrate this map consider Figure 2(a) which displays an example set D containing three elements in \mathbb{R}^n where $n = 2$. The result of applying the map S to this set is illustrated in Figure 2(b).

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

Let $g_u : \mathbb{R}^n \rightarrow \mathbb{R}$ 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 .

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

In the second step of the proposed pooling method we apply a map $F : \Omega \rightarrow L^p(I)$ to $S(D)$. Here $L^p(I)$ is the space of real valued functions on I equipped with the L^p -norm defined in Equation 3 [5]. Note that, function addition and subtraction is performed pointwise in this space.

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

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 2(b). Figure

2(c) displays the function $\rho : I \rightarrow \mathbb{R}$ 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 $\rho : I \rightarrow \mathbb{R}$ is defined in Equation 4

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

The elements of $L^p(I)$, and in turn the function representation $\rho : I \rightarrow \mathbb{R}$, 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 2(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 takes a value 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 2(c) and 2(d) display the functions $\rho : I \rightarrow \mathbb{R}$ resulting from applying the map F to the set $S(D)$ in Figure 2(b) with σ parameter values of 0.005 and 0.0001 respectively.

The parameter σ may be interpreted as follows. As the value of σ approaches 0 the function representation $\rho : I \rightarrow \mathbb{R}$ becomes a sum of indicator functions on the set D of vertex representations. In this case 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 ∞ , differences between the functions are gradually smoothed out and in turn the distance between the functions gradually reduces. Therefore, one can view the parameter σ as controlling the discrimination power of the method.

4 Results

To evaluate the proposed pooling method we considered the task of graph classification on a number of datasets. The layout of this section is as follows. Section 4.1 describes the neural network architecture used in all experiments. Section 4.3 describes the datasets considered. Section 4.2 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 a number of baseline methods.

4.1 Network Architecture

Recall from section 3 that $G = (V, E)$ denotes a graph we wish to classify and $l : V \rightarrow \Sigma$ denotes a vertex labelling function where Σ is a finite set. In order to perform classification of G the following feed-forward neural network architecture was used which consists of six layers.

The first two layers are convolutional layers similarly to the GraphSAGE convolutional layers [10]. Only two convolutional layers were used because a number of studies have found that the use of two layers empirically gives best performance [16].

Let \cdot denote matrix multiplication and CONCAT denote horizontal matrix concatenation. The k th convolutional layer is implemented using Equation 5 where A is the adjacency matrix corresponding to G , W_k are the layer weights and b_k are the layer biases. The weights W_k is a matrix of dimension $2d_{k-1} \times d_k$ where d_k the dimension of the k th layer. The biases b_k is a vector of dimension d_k . The term h_0 denotes a matrix of size $|V| \times |\Sigma|$ where each matrix row equals the one-hot-encoding of an individual vertex label. The term h_k denotes a matrix of size $|V| \times d_k$ where each matrix row equals the representation of an individual vertex output from the k th convolutional layer and d_k is the dimension of this representation. Note that, since two convolutional layers are used, k takes values in the set $\{1, 2\}$. 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.

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

The third architecture layer is a fully connected linear layer of dimension 10. The fourth layer is the pooling method used. The fifth layer is another fully connected linear layer of dimension 20. The final layer is a softmax function and returns a probability distribution over the classes. The output of the first linear layer equals 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 dimension $3^{10} = 59049$.

4.2 Optimization

The model parameters to be optimized in the architecture of section 4.1 are the weights and biases of the convolutional layers, the weights and biases of the linear layers and the parameter σ of the pooling method. In all experiments the neural network parameters were initialized as follows. All weight matrices in the convolutional and linear layers were initialized using Kaiming initialization [11]. All biases in the convolutional and linear layers were initialized to zero. Finally, the parameter σ in Equation 2 of the pooling layer was initialized to 0.125.

For loss function Cross Entropy plus an L^2 regularization term with 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} [15]. In all experiments optimization was performed for 350 data epochs and the model which achieved the minimum loss during this process was returned. In all cases the optimization procedure converged well before 350 data epochs.

| Pooling Method | MUTAG | PROTEINS | ENZYMES |
|----------------|-----------------------------------|-----------------------------------|-----------------------------------|
| Sum | 0.66 ± 0.60 | 0.60 ± 0.18 | 0.26 ± 0.07 |
| Mean | 0.78 ± 0.18 | 0.58 ± 0.16 | 0.30 ± 0.05 |
| DiffPool | 0.85 ± 0.11 | 0.73 ± 0.04 | 0.32 ± 0.07 |
| SortPooling | 0.74 ± 0.11 | 0.72 ± 0.05 | 0.23 ± 0.04 |
| Set2Set | 0.73 ± 0.08 | 0.72 ± 0.04 | 0.30 ± 0.07 |
| Function Space | 0.83 ± 0.11 | 0.73 ± 0.19 | 0.32 ± 0.06 |

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

4.3 Datasets

To evaluate the proposed pooling method we used three graph classification datasets. The datasets in question are commonly used to evaluate graph classification methods and were obtained from the TU Dortmund University graph dataset repository [14].

The first dataset was the MUTAG dataset which consists of 188 graphs corresponding to chemical compounds where there are 7 distinct types of vertices. The classification problem is binary and concerns predicting if a chemical compound has mutagenicity or not [6].

The second dataset was the PROTEINS dataset which consists of 1113 graphs corresponding to protein molecules where there are 3 distinct types of vertices. The classification task is binary and concerns predicting if a protein is an enzyme or not [3].

The third dataset was the ENZYMES dataset which consists of 600 graphs corresponding to enzymes where there are 3 distinct types of vertices. The classification task is multi-class and concerns predicting enzyme class where there are 6 distinct classes.

4.4 Classification Accuracy

The proposed pooling method was benchmarked against the following five baseline pooling methods: mean vertex representation, sum of vertex representations, DiffPool by Ying et al. [29], SortPooling by Zhang et al. [30] and Set2Set by Vinyals et al. [24]. As described in the background and related works section of this paper, these are some of the most commonly used pooling methods.

For all baseline models we used a neural network architecture similar to that described in section 4.1 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. All experiments were implemented in Python3 using the PyTorch library [19] and run on an Nvidia GeForce RTX 2080 GPU. For the baseline pooling methods we used the corresponding implementations available in the *PyTorch Geometric* Python library [8].

For each dataset considered we computed the mean accuracy of 10-fold cross validation for each pooling method. The results of this analysis are displayed in Table 1. For each dataset, the proposed pooling method outperformed most baseline methods and achieved equal best performance on two of the three datasets. This demonstrates the utility of the proposed pooling method.

5 Conclusions

Pooling is a fundamental type of layer in graph neural networks which involves compute a representation of the set of vertex representations output from a sequence of convolutional layers. In this work we proposed a novel pooling method which computes a function space representation of the set of vertex representations. This method is distinct from existing pooling methods which compute either a vector or sequence space representation.

Experimental results on a number of graph classification benchmark datasets demonstrate that the proposed method generally outperforms most baseline pooling methods and in some cases achieves best performance. The benchmark datasets in question contain graphs corresponding to molecules and chemical compounds which are the most common types of dataset used to evaluate graph classification methods. Despite this fact, the proposed pooling method is general in nature and can be applied to any type of graph. Finally, the authors hope this work will serve as a platform for future work investigating the use of function space representations for pooling.

References

1. Adams, H., Emerson, T., Kirby, M., Neville, R., Peterson, C., Shipman, P., Chepushtanova, S., Hanson, E., Motta, F., Ziegelmeier, L.: Persistence images: A stable vector representation of persistent homology. *The Journal of Machine Learning Research* **18**(1), 218–252 (2017)
2. Bai, Y., Ding, H., Qiao, Y., Marinovic, A., Gu, K., Chen, T., Sun, Y., Wang, W.: Unsupervised inductive whole-graph embedding by preserving graph proximity. *First International Workshop on Deep Learning on Graphs: Methods and Applications* (2019)
3. Borgwardt, K.M., Ong, C.S., Schönauer, S., Vishwanathan, S., Smola, A.J., Kriegel, H.P.: Protein function prediction via graph kernels. *Bioinformatics* **21**(suppl.1), i47–i56 (2005)
4. Chen, K., de Vicente, J.P., Sepulveda, G., Xia, F., Soto, A., Vázquez, M., Savarese, S.: A behavioral approach to visual navigation with graph localization networks. *arXiv preprint arXiv:1903.00445* (2019)

5. Christensen, O.: Functions, spaces, and expansions: mathematical tools in physics and engineering. Springer Science & Business Media (2010)
6. Debnath, A.K., Lopez de Compadre, R.L., Debnath, G., Shusterman, A.J., Hansch, C.: 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)
7. Duvenaud, D., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., Adams, R.: Convolutional networks on graphs for learning molecular fingerprints. In: *Advances in neural information processing systems*. pp. 2224–2232 (2015)
8. Fey, M., Lenssen, J.E.: Fast graph representation learning with PyTorch Geometric. In: *ICLR Workshop on Representation Learning on Graphs and Manifolds* (2019)
9. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. pp. 1263–1272 (2017)
10. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: *Advances in Neural Information Processing Systems*. pp. 1024–1034 (2017)
11. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: *Proceedings of the IEEE international conference on computer vision*. pp. 1026–1034 (2015)
12. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 770–778 (2016)
13. Kearnes, S., McCloskey, K., Berndl, M., Pande, V., Riley, P.: Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design* **30**(8), 595–608 (2016)
14. Kersting, K., Kriege, N.M., Morris, C., Mutzel, P., Neumann, M.: Benchmark data sets for graph kernels (2016), <http://graphkernels.cs.tu-dortmund.de>
15. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)* (2014)
16. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations* (2017)
17. Li, Y., Tarlow, D., Brockschmidt, M., Zemel, R.: Gated graph sequence neural networks. In: *Proceedings of International Conference on Learning Representations* (2016)
18. Luzhnica, E., Day, B., Liò, P.: On graph classification networks, datasets and baselines. In: *ICML Workshop on Learning and Reasoning with Graph-Structured Representations* (2019)
19. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in PyTorch. In: *NIPS Autodiff Workshop* (2017)
20. Sanchez-Gonzalez, A., Heess, N., Springenberg, J.T., Merel, J., Riedmiller, M., Hadsell, R., Battaglia, P.: Graph networks as learnable physics engines for inference and control. In: *International Conference on Machine Learning*. pp. 4467–4476 (2018)
21. Tolstaya, E., Gama, F., Paulos, J., Pappas, G., Kumar, V., Ribeiro, A.: Learning decentralized controllers for robot swarms with graph neural networks. *arXiv preprint arXiv:1903.10527* (2019)

22. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. pp. 6000–6010. Curran Associates Inc. (2017)
23. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph Attention Networks. International Conference on Learning Representations (2018)
24. Vinyals, O., Bengio, S., Kudlur, M.: Order matters: Sequence to sequence for sets. International Conference on Learning Representations (2016)
25. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. arXiv preprint arXiv:1901.00596 (2019)
26. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. arXiv preprint arXiv:1901.00596 (2019)
27. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: International Conference on Learning Representations (2019)
28. Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.i., Jegelka, S.: Representation learning on graphs with jumping knowledge networks. In: International Conference on Machine Learning. pp. 5449–5458 (2018)
29. Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., Leskovec, J.: Hierarchical graph representation learning with differentiable pooling. In: Advances in Neural Information Processing Systems. pp. 4800–4810 (2018)
30. Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An end-to-end deep learning architecture for graph classification. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
31. Zhang, Z., Cui, P., Zhu, W.: Deep learning on graphs: A survey. arXiv preprint arXiv:1812.04202 (2018)