Investigating limitations of global and local pooling in generalizable graph neural networks for materials property prediction

Oscar Rivera
Fung Group at the School of Computational Science and Engineering at the Georgia Institute of Technology

Abstract

Recent artificial intelligence breakthroughs have brought more attention to machine learning (ML) and ML research. New models, architectures, and theories are being proposed with increasing frequency, and one such model architecture, the graph neural network (GNN), is likely the most compelling. GNNs can be applied to numerous fields and often outperform competitors, especially when given ample data. The present research focuses on the pooling step of GNNs (also known as the readout step), though pooling is a broader ML problem as well. Specifically, a combinational pooling method is proposed as a means to capture more complex graphical information. Indeed, on certain tasks, the combinational approach outperforms the baseline, but it is not a silver-bullet solution to pooling. Moreover, the combination of pooling methods is compared to several proposed state-of-the-art (SOTA) pooling methods. Again, these SOTA methods do not always outperform the baseline. Reasons for the inconsistency of methods, including the difficulty of the materials datasets employed in the present research, are discussed, and directions for future research are proposed.

1 Introduction

GNNs, a relatively new kind of representation of data in machine learning, are positioned as one of the most promising avenues for ML research[1-3]. GNNs can be applied to a wide variety of fields, including social networks, molecular chemistry, and text generation. For a social network, the graph's nodes represent individuals, and the graph's edges represent the connections between people; that is, the two people 'follow' each other (Fig. 1a). In molecular chemistry and materials science, atoms are represented as a graph's nodes and bonds between atoms as the graph's edges (Fig. 1b)[4].

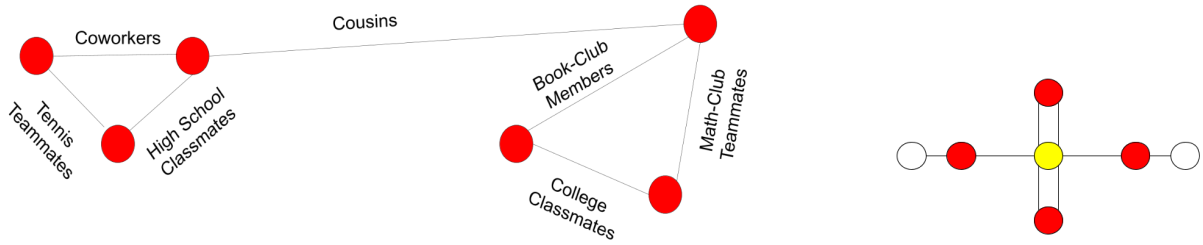

Fig. 1. (a) Left: A social network with six members with edge types that represent the relationships between those members. (b) Right: A molecule of sulfuric acid (H_2SO_4).

The GNN method of representing molecules is also consistent with recent developments in machine learning, which have shown that end-to-end models are preferable to using

human-made feature (that is, property) representations[4]. However, these end-to-end models have larger data requirements than their feature-based predecessors. To compensate for this data requirement, GNN architectures often use pre-training, or learning from large, unlabeled datasets, before testing on smaller, more specific datasets[3]. Without pre-training, however, it appears that the performance benefit from using GNNs over traditional models is greatest on datasets that are over 3000 samples[5].

In the context of drug design, computational chemistry can help speed up the drug discovery process, which can take up to six years, by optimizing for target properties of novel molecules[6]. Machine learning models, though often less reliable than experimental testing, make it easier to identify which materials should be tested empirically because ML simulations are significantly faster than in-lab investigations. In other words, it is useful to use ML in a high throughput manner, testing the most promising materials from the simulations more extensively in the lab[7].

Though GNNs are relevant in a variety of disciplines, the present research considers computational chemistry applications, specifically periodic crystal (that is, materials) applications, as opposed to closely related molecular applications. A molecular graph corresponds to atoms in a molecule, whereas a periodic crystal graph corresponds to atoms in a unit cell (Fig. 2).

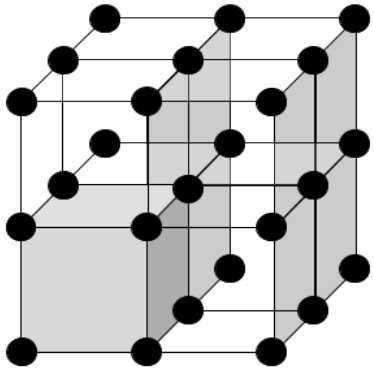

Fig. 2. The gray box represents one unit cell, the smallest non-repeating set of atoms in this simple cubic crystalline structure. The individual unit cell comprises the graph used in this research.

In the context of materials prediction, the datasets used are log kvrh, dielectric, perovskites, phonons, 2D, surface, and exfoliation. The target properties of these datasets are bulk modulus, refractive index, formation energy for perovskite cell, frequency at the last phonon PhDOS peak, work function, molecular adsorption energy, and exfoliation energy,

respectively[7-9]. These datasets represent a broad spectrum of materials and materials properties and are regularly used to evaluate ML approaches to computational chemistry and materials prediction. Proposed approaches, including the pooling approaches employed in this paper, are often expected to benefit performance on most or all of the datasets to be considered effective.

MatDeepLearn, the ML framework employed in the present research, functions as a benchmark for GNN methods, architectures, and designs. The synthesis of experimental and theoretical fields through computational science is a promising and active domain: labs use high throughput computation to determine promising routes for further materials research. This approach has led to the development of lithium-ion batteries and hydrogen storage materials, among others[10]. MatDeepLearn provides the computational aspect of this kind of experimental-theoretical research; given structures and target properties, it transforms structural data into graph-formatted data, provides various ML learning algorithms, and enables easy hyperparameter tuning (the ability to change training parameters, like the number of graphs to be processed at a time)[11].

The graph neural network architecture in the present research consists of pre-graph-convolution processing in which linear layers process the node features and attempt to learn from the identity of each node in a graph. Having at least one linear layer has been shown to be optimal in GNN architectures[12]. Generally, these models randomize weights, or the strength of connections between data, and update those weights to decrease the loss (difference in actual and predicted values). It has been proposed to use linear layers before graph convolutions in conjunction with nonlinear activation functions, functions that increase expressibility by non-linearizing data[13]. Learning in linear layers is a process that was first employed in the 1960s. Soon after, the use of stochastic gradient descent in neural networks - a method for finding local minima in the loss that, when paired with backpropagation (updating weights based on predictions), learns more appropriate weight embeddings - was proposed[14].

After the pre-graph-convolution processing step, a graph convolution, an integral of a function multiplied by another function's reflection, is performed over the node features. In this phase, also known as the message-passing phase, instead of attempting to minimize the difference between predicted and target values, each atom or node learns from its neighboring nodes. In a graph convolution, one node may consider its neighbors by finding the mean of surrounding node features, also taking into account the strength of the edge between them[15]. This

graph convolution is one of the most promising methods for learning from graphs. Previously, learning from graphs was done via graph kernel, a method of using a grid of neighboring values to come up with a single representation for that neighborhood[16].

After the graph convolution layers, the feature vector (a numerical representation of properties) of each atom is reduced to the feature vector of each graph in the pooling step; for instance, a model will consider a unit cell as a whole instead of considering each atom of that unit cell. Since 2020, the importance of expressive pooling functions, functions that capture complexity by considering graph topology, node embeddings, and node importance, has been doubted in the context of GNNs, following the publication of a paper that argued that the importance of GNNs stems from their graph convolution layers, not the pooling layers that follow. This 2020 paper found that when nodes are randomly rearranged, the model performs just as well as it would when nodes are not rearranged, given an expressive pooling method. Thus, the paper concluded that expressive pooling functions that considered topology and other graph characteristics were not beneficial[17]. These results were subsequently refuted because the architecture employed had linear layers before pooling, meaning that the predictions being made were less predicated on the pooling itself. The authors of this refutation found that using rearranged graphs given an expressive pooling operator impaired performance. Moreover, it was demonstrated that expressive pooling methods achieved the best performance over traditional methods and were often faster[18].

Some of the most common yet rudimentary pooling operations include finding the sum of each atom in the graph, finding the mean of each atom in the graph, and finding the maximum of each atom in the graph. It has been shown that using a sum operation is preferable for predicting extensive molecular properties, properties based on the size of the graph[12]. However, the literature on pooling remains limited. In most cases, GNN architectures choose between mean pooling and sum pooling. Because of the complexity of the previous graph convolution layer, it is likely that an overly simplistic pooling method would minimize important graphical details. For example, in a graph of sulfuric acid (Fig. 1b), hydrogen atoms comprise 28.6% of the molecule by number of atoms but only 2.05% of the molecule by mass; here, a mean pooling method will underestimate the importance of the hydrogen atoms, despite their importance to the function of the molecule.

A combination of operations to capture graphical complexity is proposed. Simple mathematical operations such as mean, maximum, minimum, standard deviation, and add (that is, sum) are considered, as well as other recently proposed methods that claim to capture certain graphical details or to retain nuance. After pooling, each graph represents a unit cell, but each unit cell still has a non-one-size feature vector; that is, each unit cell is represented by more than one number. In order to make a prediction, final linear layers are used to ensure each unit cell only has one number representing it. In this step, the efficacy of employing an encoder-decoder architecture following the pooling step is tested. In this architecture, the predictions of the final linear layers are compared to the predictions before those linear layers by adding the difference to the model's loss calculation. It is possible that more importance will be attached to the capturing of important information in the output of each pooling method.

Because using several pooling methods increases the size of each feature vector, experiments are done with reducing this size gradually instead of all at once. For instance, a feature vector containing the maximum, minimum, and mean of a graph, instead of solely the mean of a graph, will be three times as large. As a result, the efficacy of gradually reducing the size of the feature vector is tested; it is likely that a gradual reduction will place less stress on the first linear layer following pooling. For instance, an architecture with three post-GNN linear layers might reduce the feature vector dimension of 300 to 100 entirely in the first linear layer, and the other two layers process only the feature vector of size 100. The performance may improve if the first layer decreases the size of the vector from 300 to 200, the second layer reduces the size of the vector from 200 to 100, and the third layer processes the vector of size 100. Regardless of architecture choice, there is also one final linear layer that reduces the size of the feature vector to 1 so a prediction can be made. The last layer is not included in the post-GNN linear layer count.

2 Procedure

The model in the present research has three pre-GNN linear layers built with PyTorch, a common library for machine learning functions and tools, followed by four graph convolution layers, which use PyTorch Geometric[19]. Finally, tests were conducted with three post-GNN linear layers following the model's pooling method. The batch size for all trials is 100 (that is, the number of graphs to be processed at once), and each trial lasts 200 epochs (that is, iterations over

the entire dataset). All experiments were run virtually using the NERSC supercomputer, including an NVIDIA A100 (40 GB) GPU and an AMD EPYC 7713 CPU.

Pooling methods are compared in various contexts: for instance, the performance of pooling after two different graph convolution layers (CGCNN and TorchMD)[20,21] are compared because, theoretically, if one graph convolution layer captures more complexity than another, there is less room for improvement for the pooling methods with that architecture. Moreover, two different edge, or bond, calculation methods are tested. The first explicitly assigns all of the edges in the graph's predetermined cutoff radius (OCP). The second assigns only the shortest of the possible edges in this radius (MDL) (Fig. 3).

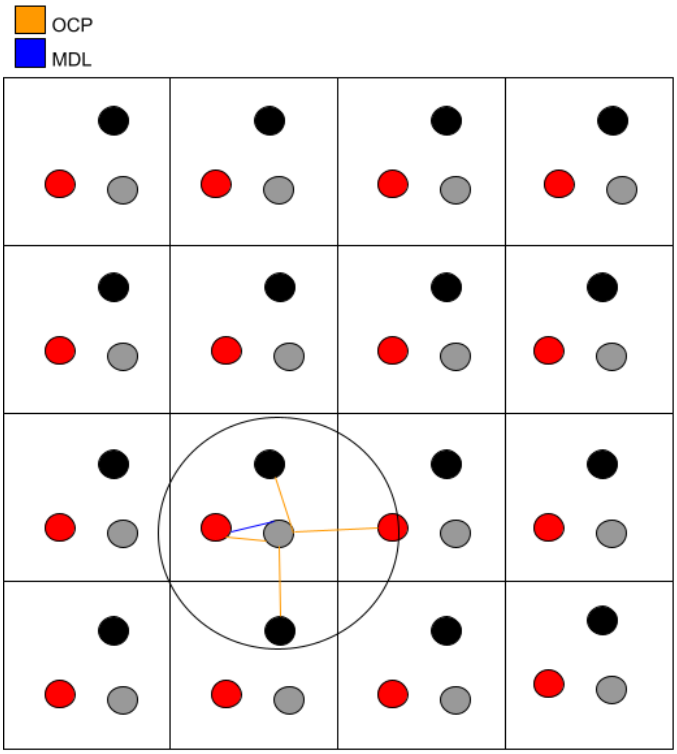

Fig. 3. A 2D representation of a 3D crystal structure. The circle represents the predetermined cutoff radius, which would be a sphere in a 3D graph. The image only considers one atom (the gray atom at the center of the circle). This process is repeated for every atom. MDL connects the considered atom only to its closest neighbor within the cutoff radius. OCP connects this atom to all of its neighbors within the cutoff radius.

The MDL method is less computationally intensive but will generally create a graph with less expressive capability because it has less information than OCP. In the context of different graph convolution layers and different edge calculation methods, pooling methods are tested on various chemical datasets: 2D, dielectric, perovskites, phonons, exfoliation, surface, and log kvrh. Each test has five trials with five predetermined seeds (handpicked parameters that

determine how to randomly split the dataset), which are consistent among all trials and tests. For one combinational pooling method that increases the size of the feature dimension, an encoder-decoder architecture on the post-pooling linear layers and a gradual reduction in the size of the feature dimension were tested.

Two different graph convolution layers, also known as models, were tested: CGCNN and TorchMD. Methods that performed best on CGCNN and MDL were chosen for testing with the OCP edge calculation method and CGCNN model. Finally, the best method from all CGCNN (both OCP and MDL) testing and the method proposed in the present research were used on the TorchMD model (both OCP and MDL). In summary, methods were tested in the following order, with successful methods receiving additional, more extensive testing:

1. Testing on all datasets with CGCNN and MDL.
2. Testing on all datasets with CGCNN and OCP.
3. Testing on all datasets with TorchMD using both MDL and OCP.

When testing, there are three losses: training, testing, and validation. In the present research, the training, testing, and validation splits are 0.80, 0.15, and 0.05, respectively. In each epoch, 80% of the data is used for training (updating the model), 15% for testing (determining how well the model performs with the learned parameters and hyperparameters), and 5% for validation (updating the hyperparameters of the model, like how much the model should change to reduce its loss, known as learning rate). Only the test error that corresponds to the minimum validation error from each trial is used to calculate the average percent improvement over mean pooling, the baseline. The figures presented in the results section of the present research are percent improvements, where negative numbers represent an increase in performance and positive numbers represent a decrease in performance.

3 Results & Discussion

Table 1 presents the results of combining methods using the CGCNN model and MDL edge-calculation method. The encoder-decoder architecture and gradually decreasing linear layers are paired with mean, add, maximum, and minimum pooling (MAMM). One of the proposed SOTA pooling methods in recent literature, graph multiset transformer (GMT), is also combined with mean pooling. To find the optimal amount of pooling operators, MAMM is compared to mean & add and MAMM & standard deviation. MAMM had the best average performance over all datasets, outperforming the second-best method by 0.5%. Though both the

encoder-decoder architecture and gradual decrease worked in some instances, like the 2D dataset, MAMM alone had the best average performance and was thus used for comparison in the following experiments.

Table 1. These results are based on the average percent change versus the baseline mean pooling operator for five trials on each dataset. This is using the MDL edge calculation method and CGCNN architecture.

	Gradual	MAMM	Encoder-Decoder & Gradual	Encoder-Decoder	MAMM + Standard Deviation	GMT, Mean	Mean, Add
2D	-5.054	-2.189	-5.280	-5.675	-1.930	-6.985	-0.406
Dielectric	-8.811	-6.595	-3.101	-5.440	-2.973	-4.258	0.947
Perovskites	-2.651	1.056	7.638	12.051	-4.122	-3.497	-7.857
Phonons	5.928	-4.411	10.073	8.469	6.544	-3.208	7.096
Exfoliation	8.269	5.501	10.221	14.282	9.916	13.316	20.994
Surface	-9.495	-10.256	12.265	5.568	-11.364	-0.200	-1.151
Log Kvrh	5.920	5.475	5.510	6.596	4.399	-3.077	1.270
Average:	-0.842	-1.631	5.332	5.121	0.067	-1.130	2.985

MAMM and the standard deviation operation may have shown a slight performance decrease compared to the baseline because the standard deviation operation is too complex to be understood by the model as it employs two averages: the average of values and the average of distances between values. MAMM likely outperformed the combination of mean and add because the model understood the importance of minimum and maximum values in some cases. The information learned from the GMT approach, even though increasing performance alone (Table 2), was negatively affected by the inclusion of mean pooling.

Table 2 compares several pooling methods proposed in the recent literature. As is the case in Table 1, no method outperforms the baseline on all benchmark datasets. On the 2D dataset, for instance, memory pooling outperforms the second-best method by over 3%, whereas memory pooling suffers a 75% decrease in performance on the phonons dataset. Dense diff performs similarly, ranging from a 4% improvement to a 77% decrease in performance.

Table 2. Results based on the average percent change versus the baseline mean pooling operator for five trials on each dataset. This is using the MDL edge calculation method and CGCNN architecture.

	Dense Diff	Set2set	Memory	Set Transformer	GMT
2D	-4.074	-3.602	-7.146	-0.881	-2.863
Perovskites	-3.321	-3.477	-1.502	-6.909	-9.968
Dielectric	-0.103	-1.308	6.157	-4.835	-6.064
Phonons	77.146	1.468	75.51	-2.299	-2.564
Exfoliation	25.714	4.309	47.119	7.248	11.634
Surface	-2.895	-8.428	-2.065	-14.508	-11.9
Log Kvrh	6.194	5.86	5.074	5.236	0.29
Average:	14.095	-0.74	17.593	-2.421	-3.062

Set transformer (ST) and GMT record the most impressive average performances of 2.421% improvement and 3.062% improvement, respectively, despite both methods hurting performance on the log kvrh and the exfoliation datasets. These two methods are chosen for more extensive testing.

The other recently proposed methods (dense diff, memory, and set2set) employ various pooling approaches, including attention-based approaches in the cases of memory and set2set pooling. Attention methods use two matrix multiplications to avoid prioritizing the data that is analyzed by the model last, a common problem in ML. The first matrix multiplication determines which nodes are important; the second scales the values of nodes by their importance[22]. Dense differentiable pooling (dense diff) attempts to capture graphical complexity by applying pooling in multiple steps instead of all at once[23]. Set2set uses attention to recursively choose the most important nodes[24]. Memory pooling assigns nodes to groups, called clusters, and then uses attention to find the most important nodes in those clusters[25].

A trend present in Table 1 and Table 2 is the substantial performance benefit of some methods on the surface dataset. Specifically, these methods are ones that prioritize or consider certain nodes that are deemed important. ST, GMT, and set2set (Table 2) all use attention operators to locate important nodes[22], whereas the information in MAMM is enough to capture those important nodes (Table 1). This approach works for the surface dataset because predictions

are based on the properties of the surfaces of alloys[9], so atoms on or close to the surface are more important than those toward the center of the material. However, when all nodes are equally important, these methods fail: for instance, ST, GMT, and MAMM all hurt performance on the log kvrh dataset, which measures the effect of applying pressure to a material on the volume of that material (Tables 1 and 2).

The further experimentation results for ST, GMT, and MAMM are presented in Table 3. The three methods are tested using the OCP edge calculation method and the CGCNN message-passing architecture. Both MAMM and GMT are presented using the TorchMD model with the MDL and OCP edge calculation methods. Since ST performed worse than GMT using CGCNN (Table 3) and both have similar transformer-based architectures (that is, a complex application of attention), ST was not tested (No Test) using the more computationally expensive TorchMD model.

Table 3. For each model and edge-calculation method, the average of five trials on five pre-determined seeds is averaged again over each of the following datasets: perovskites, 2D, surface, phonons, dielectric, exfoliation, and log kvrh.

	ST	GMT	MAMM
OCP CGCNN	0.629	-6.112	-0.524
MDL CGCNN	-2.421	-3.063	-1.631
OCP TorchMD	No Test	-2.943	14.434
MDL TorchMD	No Test	-7.153	3.510

Comparing the three methods, GMT seems to be the best option as it is the only method that consistently offers improvement; MAMM decreases performance by 14.4% and 3.5% on TorchMD using OCP and MDL, respectively; ST performs worse than both GMT and MAMM on CGCNN using OCP, reducing performance by 0.6%. Similar to the pooling method's performance being dependent on the dataset, as shown in Tables 1 and 2, it is also dependent on model design and edge-calculation method, as shown in Table 3.

Figures 4a and 4b demonstrate the range of performance of GMT, ST, and MAMM using CGCNN. For MDL and OCP, no method consistently records the lowest of all metrics. The metrics presented in box-and-whisker format are maximum, third-quartile value, median, first-quartile value, and minimum.

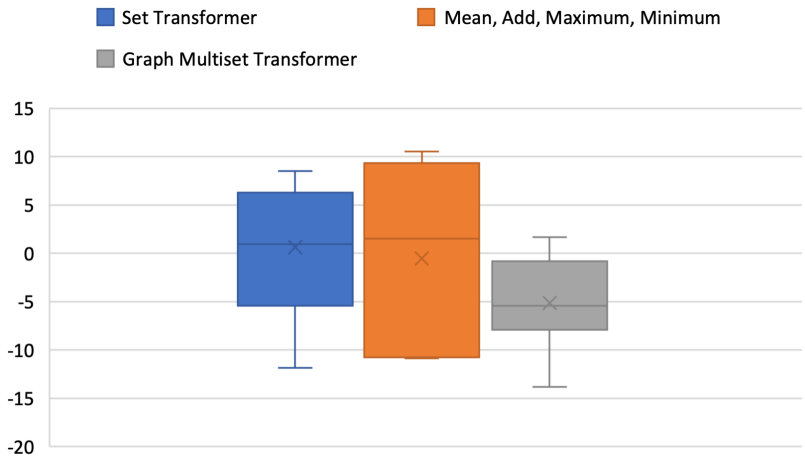

Fig. 4. (a) The five-number summary is shown by a box and whisker plot. The data come from experiments on each of seven different datasets with a CGCNN model. These tests are done with the OCP edge calculation method.

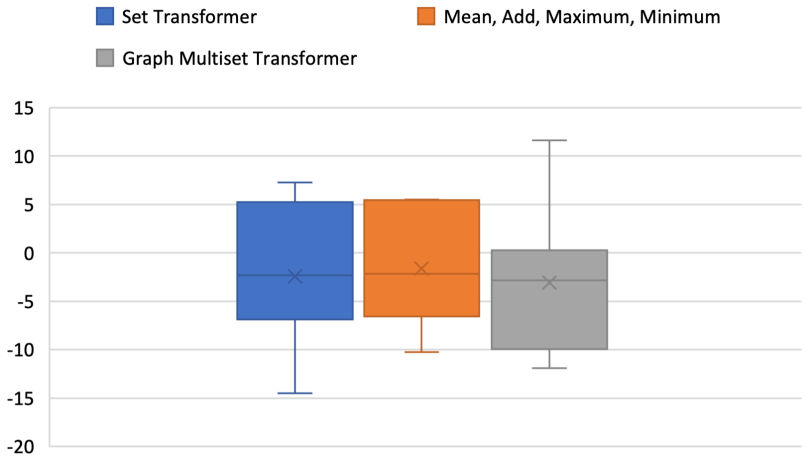

Fig. 4. (b) The five-number summary is shown by a box and whisker plot. The data come from experiments on each of seven different datasets with a CGCNN model. These tests were conducted with the MDL edge calculation method.

The complexity of the pooling situation is made apparent in Figures 4a and 4b. Though Tables 2 and 3 suggest GMT is the optimal pooling method, Figures 4a and 4b suggest that this is only sometimes the case. As shown in Figure 4a, GMT exceeds alternative methods by recording the lowest value on several statistical measures (third-quartile value, maximum, median, and minimum) using the OCP edge calculation method. Also dominant with the MDL method, GMT records the lowest first-quartile value, median, and third-quartile value (Fig. 4b). However, by some metrics, GMT is not always the clear winner. For example, GMT has the worst performance with the MDL method, whereas ST has the best performance (Fig. 4b).

Moreover, MAMM has a first-quartile value of approximately 3.5% lower than GMT with the OCP edge calculation method (Fig. 4a).

Like Figures 4a and 4b, Figures 5a and 5b demonstrate the range of performance of pooling methods. Even though GMT outperforms MAMM in Figures 5a and 5b by all metrics (those being maximum, third-quartile value, median, first-quartile value, and minimum), over half of the GMT plot is above 0, representing a performance decrease on over half of the benchmark datasets. Even though GMT performs, on average, better than the baseline for both edge calculation methods on TorchMD (Table 3), the results in Figures 5a and 5b suggest that blindly choosing GMT would hurt performance in many cases.

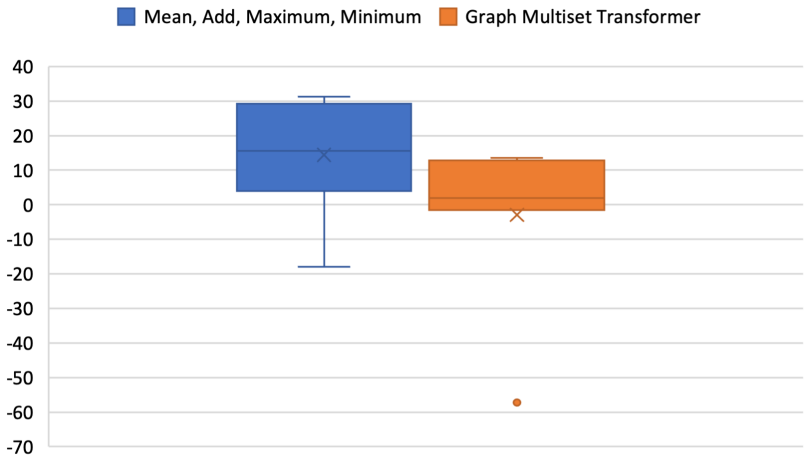

Fig. 5. (a) The five-number summary is shown by a box and whisker plot. The data come from experiments on each of seven different datasets with a TorchMD model. These tests are done with the OCP edge calculation method.

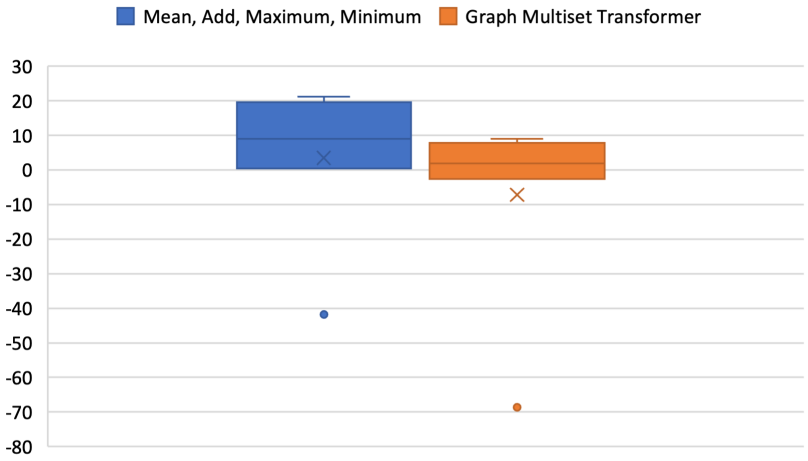

Fig. 5. (b) The five-number summary is shown by a box and whisker plot. The data come from experiments on each of seven different datasets with a TorchMD model. These tests were conducted with the MDL edge calculation method.

The importance of architecture specificities manifests again when a TorchMD model is used instead of a CGCNN model. As shown in Table 3, GMT performs, on average, better than mean pooling for TorchMD with OCP and MDL. However, as shown by Figures 5a and 5b, GMT is almost exclusively worse than mean pooling, but one significantly outlying performance resulted in an average increase in performance. For five of seven datasets, mean pooling (the baseline) outperformed GMT using TorchMD on both OCP and MDL. Though its performance using the CGCNN model was promising (Table 1), GMT suffered with the TorchMD model (Figures 5a and 5b). Again, the inconsistency of the results of GMT, though better on average than mean pooling, suggests that the pooling problem is not solved. The same conclusion is true for ST and MAMM, which most often performed worse than GMT (Tables 1, 2, and 3).

4 Conclusions

As demonstrated by the present research, pooling is a difficult problem confronting the GNN and ML communities, especially the materials property prediction domain. Individually, choosing a non-optimal pooling operation can lead to a 77% decrease in model performance. GMT and ST, though promising for graph learning, do not consistently outperform mean pooling. This may be due to the domain of some of the datasets tested in the present research, in which there are more possible edges between nodes than in other chemical datasets because crystal representations are considered, not just molecule representations, as is the case in the GMT and ST papers[26,27]. This periodic crystalline structure may increase the difficulty of the pooling step because there is a larger amount of information stored between nodes. The demonstrated importance of the dataset to the pooling step further shows that, though there are pooling methods that can achieve up to 14% improvement in some cases, the pooling bottleneck is not yet solved. Moreover, the range of performance across presented datasets indicates that there is more work to be done on the pooling step and domain knowledge is still paramount when designing and conducting experiments.

As the most impressive pooling methods in the molecular domain show weakness when translated to the periodic crystal domain, more work is necessary for pooling in the materials domain. Because of the difficulty in this domain, it would be useful to create transformer methods similar to GMT and ST more tailored for periodic crystal problems. The mixed success of combining pooling methods in the materials domain suggests it may be useful to test how well combinational methods, like MAMM proposed in the present research, perform in other

domains. As ST and GMT seem to perform better in non-materials research, this trend might extend to combinational methods.

Also, further testing how the size of the post-pooling node representation affects the efficacy of pooling methods may provide more insight into the nature of optimal pooling. Combining pooling methods increases the difficulty of the task of summarizing data in the final linear layers, so the pooling itself may be negatively affected by the additional demand on linear layers. The post-pooling approaches proposed in the present research, employing linear layers with gradually decreasing size and an encoder-decoder architecture, failed to benefit performance, so it would be useful to examine how to most efficiently use additional data in the pooling step without overwhelming the linear layers.

Ideally, a generalizable ML model for materials property prediction will emerge in the long term. Since this model needs to be generalizable to all materials benchmarks in order to be truly useful, a pooling method that consistently outperforms its competitors is vital. Several methods have been proposed to address this necessity in related domains, yet none are equally applicable to all materials systems. The present research highlights the urgency for ML researchers to continue to explore the pooling space by applying novel knowledge and breakthrough methods. The pooling problem is an integral part of the larger materials property prediction problem, and solving it will enable maximally reliable models capable of supercharging research in fields ranging from vaccine design to energy storage.

5 Acknowledgements

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231 using NERSC award BES-ERCAP0028447.

Bibliography

1. Gong, W & Yan, Q. Graph-based deep learning frameworks for molecules and solid-state materials. *Computational Materials Science*. **195**, 110332 (2021).

2. Schmidt, J. et al. Recent advances and applications of machine learning in solid-state materials science. *Npj Comput*. *Mater*. **5**, 83 (2019).

3. Zhou, J. et al. Graph neural networks: A review of methods and applications. *AI Open* **1**, (2020).

4. Reiser, P. et al. Graph neural networks for materials science and chemistry. *Commun Mater*. **3**, 93 (2022).

5. Wu, Z. et al. MoleculeNet: a benchmark for molecular machine learning. *Royal Society of Chemistry* **2**, (2018).

6. Keith, J. et al. Combining Machine Learning and Computational Chemistry for Predictive Insights Into Chemical Systems. *PubMed* (2021).

7. Dunn, A. et al. Benchmarking materials property prediction methods: the Matbench test set and Automatminer reference algorithm. *Npj Comput. Mater.* **6**, 138 (2020).

8. Haastrup, S. et al. The Computational 2D Materials Database: high-throughput modeling and discovery of atomically thin crystals. *2D Mater.* **5**, 042002 (2018).

9. Mamun, O. et al. High-throughput calculations of catalytic properties of bimetallic alloy surfaces. *Sci. Data* **6**, 76 (2019).

10. Jain, A. et al. Commentary: The Materials Project: A materials genome approach to accelerating materials innovation. *APL Mater.* **1**, 011002 (2013).

11. Fung, V. et al. Benchmarking graph neural networks for materials chemistry. *Npj Comput Mater*. **7**, 84 (2021).

12. Sanchez-Gonzalez, A. et al. Learning to Simulate Complex Physics with Graph Networks. *ICML* (2020).

13. You, J. et al. Design space for graph neural networks. *NeurIPS* **33**, (2020).

14. S.-I. Amari. Learning Patterns and Pattern Sequences by Self-Organizing Nets of Threshold Elements. *IEEE Transactions on Computers* **C-21**, 11, 1197-1206 (1972).

15. Schütt, K. et al. Learning representations of molecules and materials with atomistic neural networks. Preprint at https://arxiv.org/abs/1812.04690 (2018).

16. Vishwanathan, S. et al. Graph Kernels. *Journal of Machine Learning Research* **11**, 40, 1201−1242 (2010).
17. Mesquita, D. et al. Rethinking pooling in graph neural networks. *NeurIPS* **33**, (2020).
18. Bianchi, F. & Lachi, V. The expressive power of pooling in Graph Neural Networks. Preprint at https://arxiv.org/abs/2304.01575 (2023).
19. Fey, M. & Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. Preprint at https://arxiv.org/abs/1903.02428 (2019).
20. Xie, T. & Grossman, J. C. Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. *Phys. Rev. Lett.* **120**, 145301 (2018).
21. Doerr, S. et al. TorchMD: A Deep Learning Framework for Molecular Simulations. *Journal of Chemical Theory and Computation* **17**, 4, 2355-2363 (2021).
22. Vaswani, A. et al. Attention Is All You Need. *NeurIPS* **30**, (2017).
23. Ying, Z. et al. Hierarchical Graph Representation Learning with Differentiable Pooling. *NeurIPS* **31**, (2018).
24. Vinyals, O. et al. Order Matters: Sequence to sequence for sets. *ICLR* (2015).
25. Khasahmadi, A. et al. Memory-Based Graph Networks. *ICLR* (2020).
26. Baek, J. et al. Accurate Learning of Graph Representations with Graph Multiset Pooling. *ICLR* (2021).
27. Lee, J. et al. Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks. *ICML* (2019).