# LanczosNet: Multi-Scale Deep Graph Convolutional Networks

**Renjie Liao**$^{1,2,3}$, **Zhizhen Zhao**$^{4}$, **Raquel Urtasun**$^{1,2,3}$, **Richard S. Zemel**$^{1,3,5}$
University of Toronto$^{1}$, Uber ATG Toronto$^{2}$, Vector Institute$^{3}$,
University of Illinois at Urbana-Champaign$^{4}$, Canadian Institute for Advanced Research$^{5}$
{rjliao, urtasun, zemel}@cs.toronto.edu, zhizhenz@illinois.edu

## Abstract

Relational data can generally be represented as graphs. For processing such graph structured data, we propose LanczosNet, which uses the Lanczos algorithm to construct low rank approximations of the graph Laplacian for graph convolution. Relying on the tridiagonal decomposition of the Lanczos algorithm, we not only efficiently exploit multi-scale information via fast approximated computation of matrix power but also design learnable spectral filters. We benchmark our model against 8 recent deep graph networks on QM8 quantum chemistry dataset. Experimental results show that our model achieves the state-of-the-art performance.

## 1 Introduction

Data with complex relational and rich probabilistic structure is ubiquitous in social networks, gene expression regulatory networks, protein-protein interactions, and many other physical systems. Since relational data can generally be represented as graphs, how to reason and learn with graph structured data has become a central research question of several branches of machine learning, including statistical relational learning [1], structured prediction [2] and deep learning [3]. In the context of deep learning, we can roughly categorize the recent endeavors of modeling relational data into supervised/semi-supervised and unsupervised methods. For the former, a majority of work focuses on node/edge/graph classification and regression [4, 5, 6, 7, 3]. For the latter, unsupervised node/graph embedding learning [8, 9] is common. Recently, generative models for graphs, such as molecule generation, has drawn some attention [10, 11, 12].

From a modeling perspective, deep learning models on graphs can be grouped into two classes. The first class stems from graph signal processing (GSP) [13] which tries to generalize convolution operators from traditional signal processing to graphs. Following this line, many graph convolution based deep network models emerge. [14, 15] are among the first to explore Laplacian based graph convolution within the context of deep networks. Meanwhile, [16] performs graph convolution directly based on the adjacency matrix to predict molecule fingerprints. [17] proposes a strategy to form same sized local neighborhoods and then apply convolution like regular CNNs. Chebyshev polynomials are exploited by [18] to construct localized polynomial filters for graph convolution and are later simplified in graph convolutional networks (GCN) [19]. Further accelerations for GCN based on importance sampling and control variate techniques have been proposed by [20, 21]. [22] introduces an attention mechanism to GCN to learn weights over edges. Notably, [23] proposes diffusion convolutional neural networks (DCNN) which uses diffusion operator for graph convolution. Lanczos method has been explored for graph convolution in [24] for the purpose of acceleration.

The prominent example of the second class is graph neural networks (GNN) [25] which generalize recurrent neural networks (RNN) to arbitrary graphs and exploit the synchronous schedule to propagate information on graphs. [26] later proposes the gated graph neural networks (GGNN) which improves GNN by adding gated recurrent unit and training the network with back-propagation through time. [27] introduces random subgraph sampling and explores different aggregation functions to scale

GNN to large graphs. [28] proposes asynchronous propagation schedules based on graph partitions to improve GNN.

In this paper, we follow the line of graph convolution based models and propose the Lanczos network (LanczosNet). First, based on the tridiagonal decomposition implied by the Lanczos algorithm, our model exploits the low rank approximation of the graph Laplacian. This approximation facilitates efficient computation of matrix powers thus gathering multi-scale information easily compared to previous models. Second, we design learnable spectral filters based on the approximation which effectively increase model capacity whereas previous work often adopt non-learnable spectral filters. We benchmark against 8 recent graph networks, including both convolutional and RNN based methods, on quantum chemistry graph problem, and achieve state-of-the-art results.

## 2 Lanczos Networks

A graph $\mathcal{G}$ with $N$ nodes is denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, A)$, where $A \in \mathbb{R}^{N \times N}$ is an adjacency matrix which could either be binary or real valued. $X \in \mathbb{R}^{N \times F}$ is the compact representation of node features. For any node $v \in \mathcal{V}$, we denote its feature as a row vector $X_{v,:} \in \mathbb{R}^{1 \times F}$. We use $X_{:,i}$ to denote the $i$-th column of $X$. Following [19], we add self-loops, i.e., changing $A$ to $A + I$, and use the affinity matrix $S = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ instead of the conventional graph Laplacian.

**Algorithm 1** : Lanczos Algorithm

```
1: Input: $S, x, K, \epsilon$
2: Initialization: $\beta_0 = 0$, $q_0 = 0$, and $q_1 = x / \|x\|$
3: For $j = 1, 2, \dots, K$:
4:     $z = Sq_j$
5:     $\gamma_j = q_j^\top z$
6:     $z = z - \gamma_j q_j - \beta_{j-1} q_{j-1}$
7:     $\beta_j = \|z\|_2$
8:     If $\beta_j < \epsilon$, quit
9:     $q_{j+1} = z / \beta_j$
10:
11: $Q = [q_1, q_2, \cdots, q_K]$
12: Construct $T$ following Eq. (4)
13: Eigen decomposition $T = BRB^\top$
14: Return $V = QB$ and $R$.
```

**Algorithm 2** : LanczosNet

```
1: Input: Signal $X$, Lanczos output $V$ and $R$, scale index sets $\mathcal{S}$ and $\mathcal{I}$,
2: Initialization: $Y_0 = X$
3: For $\ell = 1, 2, \dots, \ell_c$:
4:     $Z = Y_{\ell-1}$, $\mathcal{Z} = \{\emptyset\}$
5:     For $j = 1, 2, \dots, \max(\mathcal{S})$:
6:         $Z = SZ$
7:         If $j \in \mathcal{S}$:
8:             $\mathcal{Z} = \mathcal{Z} \cup Z$
9:     For $i \in \mathcal{I}$:
10:        $\hat{R}(\mathcal{I}_i) = f_{\mathcal{I}_i}([R^{\mathcal{I}_1}, \dots, R^{\mathcal{I}_{K-1}}])$
11:        $\mathcal{Z} = \mathcal{Z} \cup V\hat{R}(\mathcal{I}_i)V^\top Y_{\ell-1}$
12:    $Y_\ell = \text{concat}(\mathcal{Z})W_\ell$
13:    If $\ell < L$
14:        $Y_\ell = \text{Dropout}(\sigma(Y_\ell))$
15: Return $Y_{\ell_c}$.
```

### 2.1 Lanczos Algorithm

Given the affinity matrix $S$ and node features $x \in \mathbb{R}^{N \times 1}$, the $N$-step Lanczos algorithm computes an orthogonal matrix $Q$ and a symmetric tridiagonal matrix $T$, such that $Q^\top S Q = T$. We denote $Q = [q_1, \cdots, q_N]$ where column vector $q_i$ is the $i$-th Lanczos vector. The diagonal and super-diagonal of $T$ are $\{\gamma_1, \dots, \gamma_N\}$ and $\{\beta_1, \dots, \beta_{N-1}\}$ respectively. $T$ is illustrated in the appendix. One can verify that $Q$ forms an orthonormal basis of the Krylov subspace $\mathcal{K}_N(S, x)$ and the first $K$ columns of $Q$ forms the orthonormal basis of $\mathcal{K}_K(S, x)$. The Lanczos algorithm is shown in detail in Alg. 1. Intuitively, if we investigate the $j$-th column of the system $SQ = QT$ and rearrange terms, we obtain $\beta_j q_{j+1} = Lq_j - \beta_{j-1} q_{j-1} - \gamma_j q_j$, which clearly explains lines 4 to 6 of the pseudocode, i.e., it tries to solve the system in an iterative manner. Note that the most expensive operation in the algorithm is the matrix-vector multiplication in line 4. After obtaining $T$, we can compute the Ritz values and Ritz vectors which approximate the eigenvalues and eigenvectors of $S$ by diagonalizing the matrix $T$.

### 2.2 LanczosNet

In this section, we first show the construction of the localized polynomial filter based on the Lanczos algorithm's output and discuss its limitations. Then we explain how to construct the spectral filter using a particular low rank approximation and how to further make the filter learnable. At last, we elaborate how to construct multi-scale graph convolution and build a deep network.

**Localized Polynomial Filter** For ease of demonstrating the concept of *Krylov subspace*, we consider a pair of input and output channels $(i, j)$. We denote the input as $X_{:,i} \in \mathbb{R}^{N \times 1}$ and the output as $Y_{:,j} \in \mathbb{R}^{N\times 1}$. Executing the Lanczos algorithm for $K$ steps with the starting vector as the normalized $X_{:,i}$, one can obtain the orthonormal basis $\tilde{Q}$ of $\mathcal{K}_K(S, X_{:,i})$ and the corresponding tridiagonal matrix $\tilde{T}$. Recall that in the localized polynomial filtering, given the orthonormal basis of $\mathcal{K}_K(S, X_{:,i})$, one can write the graph convolution as $Y_j = \tilde{Q}\boldsymbol{w}_{i,j}$ where $\tilde{Q} \in \mathbb{R}^{N\times K}$ depends on the $X_{:,i}$ and $\boldsymbol{w}_{i,j} \in \mathbb{R}^{K\times 1}$ is the learnable parameter. This filter has the benefit that the corresponding learnable coefficients are compact due to the orthonormal basis. However, if one wants to stack multiple graph convolution layers, the dependency of $\tilde{Q}$ on $X_{:,i}$ implies that a separate run of Lanczos algorithm is necessary for each graph convolution layer which is computationally demanding.

**Spectral Filter** Ideally, we would like to compute Lanczos vectors only once during the inference of a deep graph convolutional network. Luckily, this can be achieved if we take an alternative view of Lanczos algorithm. In particular, we can choose a random starting vector with unit norm and treat the $K$ step Lanczos layer's output as the low rank approximation $S \approx QTQ^\top$. Note that here $Q \in \mathbb{R}^{N\times K}$ has orthonormal columns and does not depend on the node features $X_i$ and $T$ is a $K \times K$ tridiagonal matrix. Following [29], we prove the theorem below to bound the approximation error.

**Theorem 1.** *Let $U\Lambda U^\top$ be the eigendecomposition of an $N \times N$ symmetric matrix $S$ with $\Lambda_{i,i} = \lambda_i$, $\lambda_1 \geq \cdots \geq \lambda_N$ and $U = [u_1, \ldots, u_N]$. Let $\mathcal{U}_j \equiv \mathrm{span}\{u_1, \ldots, u_j\}$. Assume $K$-step Lanczos algorithm starts with vector $v$ and outputs the orthogonal $Q \in \mathbb{R}^{N\times K}$ and tridiagonal $T \in \mathbb{R}^{K\times K}$. For any $j$ with $1 < j < N$ and $K > j$, we have,*

$$\|S - QTQ^\top\|_F^2 \leq \sum_{i=1}^{j} \lambda_i^2 \left( \frac{\sin(v, \mathcal{U}_i) \prod_{k=1}^{j-1} (\lambda_k - \lambda_N)/(\lambda_k - \lambda_j)}{\cos(v, u_i) T_{K-i}(1 + 2\gamma_i)} \right)^2 + \sum_{i=j+1}^{N} \lambda_i^2,$$

*where $T_{K-i}(x)$ is the Chebyshev Polynomial of degree $K - i$ and $\gamma_i = (\lambda_i - \lambda_{i+1})/(\lambda_{i+1} - \lambda_N)$.*

We leave the proof to the appendix. Note that the term $(\sum_{i=j+1}^{N} \lambda_i^2)^{1/2}$ is the Frobenius norm of the error between $S$ and the best rank-$j$ approximation $S_j$. We decompose the tridiagonal matrix $T = BRB^\top$, where the $K \times K$ diagonal matrix $R$ contains the Ritz values and $B \in \mathbb{R}^{K\times K}$ is an orthogonal matrix. We have a low rank approximation of the affinity matrix $S \approx VRV^\top$, where $V = QB$. Therefore, we can rewrite the graph convolution as,

$$Y_j = [X_i, SX_i, \ldots, S^{K-1}X_i]\boldsymbol{w}_{i,j} \approx [X_i, VRV^\top X_i, \ldots, VR^{K-1}V^\top X_i]\boldsymbol{w}_{i,j}, \tag{1}$$

The difference between $Y_j = \tilde{Q}\boldsymbol{w}_{i,j}$ and Eq. (1) is that the former uses the orthonormal basis while the latter uses the approximation of the direct basis of $\mathcal{K}_K(S, X_{:,i})$. Since we explicitly operate on the approximation of spectrum, i.e., Ritz value, it is a spectral filter. Such a filtering form will have significant computational benefits while considering the long range/scale dependency due to the fact that the $t$-th power of $S$ can be approximated as $S^t \approx VR^tV^\top$, where we only need to raise the diagonal entries of $R$ to the power $t$.

**Learning the Spectral Filter** Following the previous filter, one can naturally design learnable spectral filters. Specifically, we use $K$ different spectral filters of which the $k$-th output $\hat{R}(k) = f_k([R, R^1, \ldots, R^{K-1}])$, where $f_k$ is a multi-layer perceptron (MLP), $R$ and $\hat{R}(k)$ are the diagonal vector of the corresponding diagonal matrix. Therefore, by properly constructing $\hat{R}(k)$ as a diagonal matrix, we have the following filtering,

$$Y_j = [X_i, V\hat{R}(1)V^\top X_i, \ldots, V\hat{R}(K-1)V^\top X_i]\boldsymbol{w}_{i,j}. \tag{2}$$

Note that it includes the polynomial filter as a special case. When positive semi-definiteness is a concern, one can apply an activation function like ReLU [30] to the output of the MLPs.

**Multi-scale Graph Convolution** Using any above filter, one can construct a deep graph convolutional network which leverages multi-scale information. Taking the learnable spectral filter as an example, we can write one graph convolution layer in a compact way as below,

$$Y = \left[L^{\mathcal{S}_1}X, \ldots, L^{\mathcal{S}_M}X, V\hat{R}(\mathcal{I}_1)V^\top X, \ldots, V\hat{R}(\mathcal{I}_N)V^\top X\right] W, \tag{3}$$

where weight $W \in \mathbb{R}^{(M+E)D\times O}$, $\mathcal{S}$ is a set of $M$ short scale parameters and $\mathcal{I}$ is a set of $E$ long scale parameters. We consider a non-negative integer as scale parameter, e.g., $\mathcal{S} = \{0, 1, \ldots, 5\}$,

$\mathcal{I} = \{10, 20, \dots, 50\}$. Note that the convolution corresponding to short scales is similar to [23] where the number of matrix-vector multiplications is tied to the maximum scale of $\mathcal{S}$. In contrast, the convolution of long scales decouples the Lanczos step $K$ and scale parameters $\mathcal{I}$, thus permitting great freedom in tuning scales as hyperparameters. One can choose $K$ properly to balance the computation cost and the accuracy of the low rank approximation. If the maximum eigenvalue of $S$ is 1, we can even raise the power to infinity, which corresponds to the equilibrium state of diffusion process on the graph. We can stack multiple such graph convolution layers and add nonlinear activation functions, e.g., ReLU, and Dropout in between to form a deep network. The inference algorithm of such a deep network is shown in Alg. 2. With the top layer representation, one can use softmax to perform classification or a fully connected layer to perform regression. The Lanczos algorithm is run beforehand once per graph to form the network and will not be invoked during inference and learning.

## 3 Experiments

In this section, we compare our model against 8 recent graph networks, including graph convolution networks for fingerprint (GCN-FP) [16], gated graph neural networks (GGNN) [26], diffusion convolutional neural networks (DCNN) [23], Chebyshev networks (ChebyNet) [18], graph convolutional networks (GCN) [19], message passing neural networks (MPNN) [31], graph sample and aggregate (GraphSAGE) [27], graph attention networks (GAT) [22]. We test them on supervised regression of molecule property on QM8 quantum chemistry dataset [32]. For fair comparison, we only tune model-related hyperparameters in all our experiments and share the others. Please refer to the appendix for more details on hyperparameters. We implement all methods using PyTorch 0.4.0 [33] and will release the code to allow others to experiment with this suite of methods.

### 3.1 Quantum Chemistry

| Methods | Validation MAE | Test MAE |
|---|---|---|
| GCN-FP | $15.06 \pm 0.04$ | $14.80 \pm 0.09$ |
| GGNN | $12.94 \pm 0.05$ | $12.67 \pm 0.22$ |
| DCNN | $10.14 \pm 0.05$ | $9.97 \pm 0.09$ |
| ChebyNet | $10.24 \pm 0.06$ | $10.07 \pm 0.09$ |
| GCN | $11.68 \pm 0.09$ | $11.41 \pm 0.10$ |
| MPNN | $11.16 \pm 0.13$ | $11.08 \pm 0.11$ |
| GraphSAGE | $13.19 \pm 0.04$ | $12.95 \pm 0.11$ |
| GAT | $11.39 \pm 0.09$ | $11.02 \pm 0.06$ |
| LanczosNet | $\mathbf{9.65 \pm 0.19}$ | $\mathbf{9.58 \pm 0.14}$ |

Table 1: MAE error ($\times 1.0e^{-3}$) on QM8.

We benchmark all algorithms on the QM8 quantum chemistry dataset which comes from a recent study on modeling quantum mechanical calculations of electronic spectra and excited state energy of small molecules [32]. The setup of QM8 is as follows. Atoms are treated as nodes and they are connected to each other following the structure of the corresponding molecule. Each edge is labeled with a chemical bond. Note that two atoms in one molecule can have multiple edges belong to different chemical bonds. Therefore a molecule is actually modeled as a multigraph. We also use explicit hydrogen in molecule graphs as suggested in [31]. Since some models cannot leverage feature on edges easily, we use the molecule graph itself as the only input information for all models so that it is a fair comparison. As demonstrated in our ablation studies, learning node embeddings for atoms is very helpful. Therefore, we augment all competitors and our models with this component. The task is to predict 16 different quantities of electronic spectra and energy per molecule graph which boils down to a regression problem. There are 21786 molecule graphs in total of which the average numbers of nodes and edges are around 16 and 21. There are 6 different chemical bonds and 70 different atoms throughout the dataset. We use the split provided by DeepChem [1] which have 17428, 2179 and 2179 graphs for training, validation and testing respectively. Following [31, 34], we use mean squared error (MSE) as the loss for training and weighted mean absolute error (MAE) as the evaluation metric. We use the same random seed for all methods. The validation and test MAE of all methods are shown in Table 1. As you can see, LanczosNet achieves better performances than all other competitors. Note that DCNN also achieves good performance with the carefully chosen scale parameters since it is somewhat similar to our model in terms of leveraging multi-scale information.

## 4 Conclusion

In this paper, we propose LanczosNet which provides the efficient multi-scale graph convolution for processing graph-structured data. By representing relational data as graphs, we expect more progress can be made for applications like link prediction, entity classification and learning first order logic.

[1]https://deepchem.io/

## References

[1] Lise Getoor and Ben Taskar. *Introduction to statistical relational learning*, volume 1. MIT press Cambridge, 2007.

[2] Gökhan BakIr, Thomas Hofmann, Bernhard Schölkopf, Alexander J Smola, Ben Taskar, and SVN Vishwanathan. *Predicting structured data*. MIT press, 2007.

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

[4] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *JMLR*, 2010.

[5] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 2017.

[6] Victor Garcia and Joan Bruna. Few-shot learning with graph neural networks. In *ICLR*, 2018.

[7] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. vd Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. *arXiv preprint arXiv:1703.06103*, 2017.

[8] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, 2016.

[9] Alberto Garcia Duran and Mathias Niepert. Learning graph representations with embedding propagation. In *NIPS*, 2017.

[10] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. In *ICLR Workshop*, 2018.

[11] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *ICLR*, 2018.

[12] Jiaxuan You, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec. Graphrnn: A deep generative model for graphs. *arXiv preprint arXiv:1802.08773*, 2018.

[13] Antonio Ortega, Pascal Frossard, Jelena Kovačević, José MF Moura, and Pierre Vandergheynst. Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 2018.

[14] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *ICLR*, 2014.

[15] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[16] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*, 2015.

[17] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *ICML*, 2016.

[18] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 2016.

[19] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[20] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *ICLR*, 2018.

[21] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *ICML*, 2018.

[22] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[23] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *NIPS*, 2016.

[24] Ana Susnjara, Nathanael Perraudin, Daniel Kressner, and Pierre Vandergheynst. Accelerated filtering on graphs using lanczos method. *arXiv preprint arXiv:1509.04537*, 2015.

[25] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE TNN*, 2009.

[26] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *ICLR*, 2016.

[27] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

[28] Renjie Liao, Marc Brockschmidt, Daniel Tarlow, Alexander L Gaunt, Raquel Urtasun, and Richard Zemel. Graph partition neural networks for semi-supervised classification. In *ICLR Workshop*, 2018.

[29] Horst D Simon and Hongyuan Zha. Low-rank matrix approximation using the lanczos bidiagonalization process with applications. *SIAM Journal on Scientific Computing*, 2000.

[30] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[31] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

[32] Raghunathan Ramakrishnan, Mia Hartmann, Enrico Tapavicza, and O Anatole von Lilienfeld. Electronic spectra from tddft and machine learning in chemical space. *The Journal of chemical physics*, 2015.

[33] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS Workshop*, 2017.

[34] Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 2018.

[35] Beresford N Parlett. *The Symmetric Eigenvalue Problem*. SIAM, 1980.

[36] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

## 5 Appendix

### 5.1 Tridiagonal Matrix $T$

$$T = \begin{bmatrix} \gamma_1 & \beta_1 & & \\ \beta_1 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_{N-1} \\ & & \beta_{N-1} & \gamma_N \end{bmatrix}. \tag{4}$$

### 5.2 Low Rank Approximation

We first state the following Lemma from [35] without proof and then prove our Theorem 1 following [29].

**Lemma 1.** *Let $A \in \mathbb{R}^{N \times N}$ be symmetric and $v$ an arbitrary vector. Define Krylov subspace $\mathcal{K}_m \equiv span\{v, Av, \ldots, A^{m-1}v\}$. Let $A = U\Lambda U^\top$ be the eigendecomposition of $A$ with $\Lambda_{i,i} = \lambda_i$ and $\lambda_1 \geq \cdots \geq \lambda_n$. Denoting $U = [u_1, \ldots, u_N]$ and $\mathcal{U}_j = \mathrm{span}\{u_1, \ldots, u_j\}$, then*

$$\tan(u_j, \mathcal{K}_m) \leq \frac{\sin(v, \mathcal{U}_j) \prod_{k=1}^{j-1} (\lambda_k - \lambda_n)/(\lambda_k - \lambda_j)}{\cos(v, u_j) T_{m-j}(1 + 2\gamma)},$$

*where $T_{m-j}(x)$ is the Chebyshev Polynomial of degree $m - j$ and $\gamma = (\lambda_j - \lambda_{j+1})/(\lambda_{j+1} - \lambda_N)$.*

**Theorem 1.** *Let $U\Lambda U^\top$ be the eigendecomposition of an $N \times N$ symmetric matrix $S$ with $\Lambda_{i,i} = \lambda_i$, $\lambda_1 \geq \cdots \geq \lambda_N$ and $U = [u_1, \ldots, u_N]$. Let $\mathcal{U}_j \equiv \mathrm{span}\{u_1, \ldots, u_j\}$. Assume $K$-step Lanczos algorithm starts with vector $v$ and outputs the orthogonal $Q \in \mathbb{R}^{N \times K}$ and tridiagonal $T \in \mathbb{R}^{K \times K}$. For any $j$ with $1 < j < N$ and $K > j$, we have,*

$$\|S - QTQ^\top\|_F^2 \leq \sum_{i=1}^{j} \lambda_i^2 \left( \frac{\sin(v, \mathcal{U}_i) \prod_{k=1}^{j-1} (\lambda_k - \lambda_N)/(\lambda_k - \lambda_j)}{\cos(v, u_i) T_{K-i}(1 + 2\gamma_i)} \right)^2 + \sum_{i=j+1}^{N} \lambda_i^2,$$

*where $T_{K-i}(x)$ is the Chebyshev Polynomial of degree $K - i$ and $\gamma_i = (\lambda_i - \lambda_{i+1})/(\lambda_{i+1} - \lambda_N)$.*

*Proof.* From Lanczos algorithm, we have $SQ = QT$. Therefore,

$$\begin{aligned} \|S - QTQ^\top\|_F^2 &= \|S - SQQ^\top\|_F^2 \\ &= \|S(I - QQ^\top)\|_F^2 \end{aligned} \tag{5}$$

Let $P_Q^\perp \equiv I - QQ^\top$, the orthogonal projection onto the orthogonal complement of subspace $\mathrm{span}\{Q\}$. Relying on the eigendecomposition, we have,

$$\begin{aligned} \|S - QTQ^\top\|_F^2 &= \|U\Lambda U^\top (I - QQ^\top)\|_F^2 \\ &= \|\Lambda U^\top (I - QQ^\top)\|_F^2 \\ &= \|(I - QQ^\top) U\Lambda\|_F^2 \\ &= \| \left[\lambda_1 P_Q^\perp u_1, \ldots, \lambda_N P_Q^\perp u_N\right] \|_F^2, \end{aligned} \tag{6}$$

where we use the fact that $\|RA\|_F^2 = \|A\|_F^2$ for any orthogonal matrix $R$ and $\|A^\top\|_F^2 = \|A\|_F^2$.

Note that for any $j$ we have,

$$\begin{aligned} \left\| \left[\lambda_1 P_Q^\perp u_1, \ldots, \lambda_N P_Q^\perp u_N\right] \right\|_F^2 &= \sum_{i=1}^{N} \lambda_i^2 \|P_Q^\perp u_i\|^2 \\ &\leq \sum_{i=1}^{j} \lambda_i^2 \|P_Q^\perp u_i\|^2 + \sum_{i=j+1}^{N} \lambda_i^2, \end{aligned} \tag{7}$$

where we use the fact that for any $i$, $\|P_Q^\perp u_i\|^2 = \|u_i\|^2 - \|u_i - P_Q^\perp u_i\|^2 \leq \|u_i\|^2 = 1$.

Note that we have $\text{span}\{Q\} = \text{span}\{v, Sv, \dots, S^{K-1}v\} \equiv \mathcal{K}_K$ from the Lanczos algorithm. Therefore, we have,

$$\|P_Q^{\perp} u_i\| = |\sin(u_i, \mathcal{K}_K)| \leq |\tan(u_i, \mathcal{K}_K)|. \tag{8}$$

Applying the above lemma with $A = S$, we finish the prove.

□

## 5.3 Lanczos Algorithm

Utilizing exact arithmetic, Lanczos vectors are orthogonal to each other. However, in floating point arithmetic, it is well known that the round-off error will make the Lanczos vectors lose orthogonality as the iteration proceeds. One could apply a full Gram-Schmidt (GS) process $z = z - \sum_{i=1}^{j-1} z^\top q_i q_i$ after line 6 of Alg. 1 to ensure orthogonality. Other partial or selective re-orthogonalization could also be explored. However, since we found the orthgonality issue does not hurt overall performance with a small iteration number, e.g., $K = 20$, and the full GS process is computationally expensive, we do not add such a step. Although some customized eigendecomposition methods, e.g., implicit QL [36], exist for tridiagonal matrix, we leave it for future exploration due to its complicated implementation.

## 5.4 Experiments

For ChebyNet, we do not use graph coarsening in all experiments due to its demanding computation cost for large graphs. Also, for small molecule graphs, coarsening generally does not help since it loses information compared to directly stacking another layer of original graph.

## 5.5 Hyperparameters

We now report the important hyperparameters chosen via cross-validation for each method. All methods are trained with Adam with learning rate $1.0e^{-4}$ and no weight decay. The maximum number of epoch is set to 200. Early stop with window size 10 is also adopted. For convolution based methods, we found 7 layers work the best. We augment all methods with node embedding and add edge types by either feeding a multiple-channel graph Laplacian matrix or directly adding a separate message function per edge type. For all methods, no dropout is used since it slightly hurts the performance. In GCN-FP, we set the hidden dimension to 128. In GGNN, we set the hidden dimension to 128, the propagate step to 15 and aggregation function to average. In DCNN, we set the hidden dimension to 128 and use diffusion scales $\{3, 5, 7, 10, 20, 30\}$. In ChebyNet, we set the polynomial order to 5, the hidden dimension to 128. In GCN, we set the hidden dimension to 128. In MPNN, we use GRU as update function, set the number of propagation to 7, set the hidden dimension to 128, use a 1-layer MLP with 1024 hidden units and ReLU nonlinearity as the message function and set the number of unroll step of *Set2Vec* to 10. In GraphSAGE, we set the number of sampled neighbors to 40, the hidden dimension to 128 and the aggregation function to average. In GAT, we set the number of heads of all 7 layers to 8 and hidden dimension per head to 16. In LanczosNet, we do not use short diffusion scales and set long ones to $\{1, 2, 3, 5, 7, 10, 20, 30\}$. The hidden dimension is 128. Lanczos step is 20. 1-layer MLP with 128 hidden units and ReLU nonlinearity is used as the spectral filter.

## 5.6 Ablation Study

We also did a thorough ablation study of our modeling components on the validation set of QM8.

**Multi-Scale Graph Convolution:** We first study the effect of multi-scale graph convolution. In order to rule out the impact of other factors, we use LanczosNet variant, fix the spectral filter and use the one-hot encoding as the node embedding. The results are shown in the first row of Table 2. Using long scales for graph convolution clearly helps on this task. Combining both short and long scales further improves results.

**Lanczos Step:** We then investigate the Lanczos step since it will have an impact on the accuracy of the low rank approximation with Lanczos tridiagonalization. The results are shown in the second row of Table 2. We can see that performance saturates at a relatively small Lanczos step like 10 which makes sense since the average number of nodes in this dataset is around 16.

| Model | Graph Kernel | Node Embedding | Spectral Filter | Short Scales | Long Scales | Lanczos Step | Validation MAE ($\times 1.0e^{-3}$) |
|---|---|---|---|---|---|---|---|
| LNet | | one-hot | fix | {1, 2, 3} | | | 10.71 |
| LNet | | one-hot | fix | {3, 5, 7} | | | 10.60 |
| LNet | | one-hot | fix | | {10, 20, 30} | 20 | 10.54 |
| LNet | | one-hot | fix | {3, 5 ,7} | {10, 20, 30} | 20 | **10.41** |
| LNet | | one-hot | fix | | {10, 20, 30} | 5 | 10.49 |
| LNet | | one-hot | fix | | {10, 20, 30} | 10 | **10.44** |
| LNet | | one-hot | fix | | {10, 20, 30} | 20 | 10.54 |
| LNet | | one-hot | fix | | {10, 20, 30} | 40 | 10.49 |
| LNet | | one-hot | fix | {3, 5 ,7} | {10, 20, 30} | 20 | 10.41 |
| LNet | | one-hot | 1-MLP | {3, 5 ,7} | {10, 20, 30} | 20 | **10.08** |
| LNet | | one-hot | 3-MLP | {3, 5 ,7} | {10, 20, 30} | 20 | 10.44 |
| LNet | | one-hot | 5-MLP | {3, 5 ,7} | {10, 20, 30} | 20 | 10.54 |

Table 2: Ablation study on QM8 dataset. Empty cell means that the component is neither using nor applicable. 1-MLP means a MLP with 1 hidden layer.

**Fixed vs. Learned Spectral Filter:** We then study whether learning spectral filter will help improve performance. The results are shown in the third row of Table 2. Adding a 1-layer MLP does significantly help reduce the error compared to a fixed spectral filter. Note that here 1-MLP refers to a MLP with one hidden layer, 128 hidden units and ReLU nonlinearity. However, using a deeper MLP does not seem to be helpful which might be caused by the challenges in optimization.