Graph Kernel and Graph Neural Network in Molecular Dynamics

Hy Truong Son

Department of Computer Science, The University of Chicago, IL, USA

hytruongson@uchicago.edu

Abstract

Density Functional Theory (DFT) is the most successful and widely used approach to compute the electronic structure of matter. However, DFT is significantly expensive in computation. In this paper, to address the DFT's time-compexity limitation, we apply multiple Machine Learning methods to estimate the solution of DFT. We extend the state-of-the-art Weisfeiler-Lehman graph kernel and generalize the convolution operation into graph neural network. We obtain a very promising result in Harvard Clean Energy Project molecular dataset.

Index Terms: density functional theory, kernel method, graph kernel, graph neural network, molecular dynamics

1. Introduction

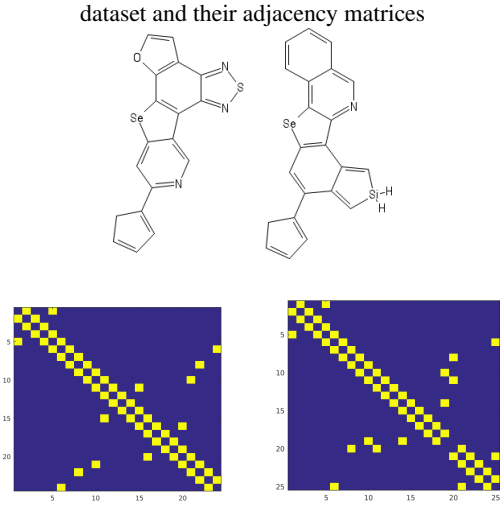
In the field of Machine Learning, standard objects such as vectors, matrices, tensors were carefully studied and successfully applied into various areas including Computer Vision, Natural Language Processing, Speech Recognition, etc. However, none of these standard objects are efficient in capturing the structures of molecules, social networks or the World Wide Web which are not fixed in size. This arises the need of graph representation and extensions of Support Vector Machine and Convolution Neural Network to graphs.

To represent graphs in general and molecules specifically, the proposed models must be *permutation-invariant* and *rotation-invariant*. In addition, to apply kernel methods on graphs, the proposed kernels must be *positive semi-definite*. Many graph kernels and graph similarity functions have been introduced by researchers. Among them, one of the most successful and efficient is the Weisfeiler-Lehman graph kernel which aims to build a multi-level, hierarchical representation of a graph [1]. However, a limitation of kernel methods is quadratic space usage and quadratic time-complexity in the number of examples. In this paper, we address this drawback by proposing Dictionary Weisfeiler-Lehman graph features in combination with Morgan circular fingerprints in section 3.1.

The common idea of family of Weisfeiler-Lehman graph kernel is hashing the sub-structures of a graph. Extending this idea, we come to Graph Neural Network in which the *fixed* hashing function is replaced by a *learnable* one as a non-linearity mapping. In the context of graphs, the sub-structures can be considered as a set of vertex feature vectors. We ultilize the convolution operation by introducing higher-order representations of a set of vectors in section 3.2.

We applied our methods to the Harvard Clean Energy Project (HCEP) molecular dataset [2]. The visualizations, experiments and results are detailed in chapter 4. We introduce our Deep Learning framework in C++ named **GraphFlow** that supports symbolic differentiation and dynamic computation graph in chapter 5.

Figure 1. Two molecules from Harvard Clean Energy Project dataset and their adjacency matrices



$C_{18}H_9N_3OSSe$ $C_{22}H_{15}NSeSi$

2. Related works

The notion of graph kernel was first introduced by Kondor and Lafferty in their ICML 2002 paper [3]. Borgwardt and colleagues proposed the state-of-the-art Weisfeiler-Lehman graph kernel based on the famous Weisfeiler-Lehman test for graph isomorphism [1]. N. M. Kriege, P. L. Giscard and R. C. Wilson applied the Hungarian matching algorithm to define a more robust Weisfeiler-Lehman graph kernel [4]. The convolution on graphs as well as Graph Neural Network for learning molecular fingerprints are mentioned by other authors [5], [6], and [7].

3. Methods

3.1. Graph Kernel

3.1.1. Positive Semi-definite Kernel

Given two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Each vertex is associated with a feature vector $f : V \to \Omega$. A *positive semi-definite* graph kernel between G_1 and G_2 is defined as:

$$\mathcal{K}_{graph}(G_1, G_2) = \frac{1}{|V_1|} \cdot \frac{1}{|V_2|} \cdot \sum_{v_1 \in V_1} \sum_{v_2 \in V_2} k_{base}(f(v_1), f(v_2))$$

where k_{base} is the base kernel and can be:

- Linear: $k_{base}(x, y) = \langle x, y \rangle_{norm} = x^T y / (\|x\| \cdot \|y\|)$
- Quadratic: $k_{base}(x, y) = (\langle x, y \rangle_{norm} + q)^2$
- RBF: $k_{base}(x, y) = \exp(-\gamma \|x - y\|^2)$

3.1.2. Weisfeiler-Lehman Graph Feature

We combine Weisfeiler-Lehman graph kernel [1] and Morgan circular fingerprints into the Dictionary Weisfeiler-Lehman graph feature algorithm. To express the idea of this combination, we define a Weisfeiler-Lehman subtree level l rooted at a vertex v as the shortest-path subtree that includes all vertices reachable from v by a shortest-path of length at most 2^l. Each substree is represented by a multi-set of vertex labels. We build the Weisfeiler-Lehman dictionary by finding all subtree representations of every graph in the dataset. The graph feature or fingerprint is a frequency vector in which each component corresponds to the frequency of a particular dictionary element.

Formally, we have a set of N graphs $\mathcal{G} = \{G^{(1)}, .., G^{(N)}\}$ where $G^{(i)} = (V^{(i)}, E^{(i)})$ $(1 \leq i \leq N)$. Let $f^G : V \to \Omega$ be the initial feature vector for each vertex of graph $G = (V, E)$. Let $S_k^G(v)$ be the set of vectors for vertex $v \in V$ of graph G at Weisfeiler-Lehman level k.

function Dictionary Weisfeiler-Lehman
01. Universal dictionary: $\mathcal{D} \leftarrow \emptyset$
02. for $i = 1 \to N$:
03. *Initialize the WL level 0*
04. for each $v \in V^{(i)}$:
05. $S_0^{G^{(i)}}(v) \leftarrow \{f^{G^{(i)}}(v)\}$
06. $\mathcal{D} \leftarrow \mathcal{D} \cup \{S_0^{G^{(i)}}(v)\}$
07. end for
08. *Build the WL level* 1, 2, .., K
09. for $k = 1 \to K$:
10. for each $v \in V^{(i)}$:
11. $S_k^{G^{(i)}}(v) \leftarrow S_{k-1}^{G^{(i)}}(v)$
12. for each $(u, v) \in E^{(i)}$:
13. $S_k^{G^{(i)}}(v) \leftarrow S_k^{G^{(i)}}(v) \cup S_{k-1}^{G^{(i)}}(u)$
14. end for
15. $\mathcal{D} \leftarrow \mathcal{D} \cup \{S_k^{G^{(i)}}(v)\}$
16. end for
17. end for
18. end for
19. $\mathcal{S} \leftarrow \{S_0^{G^{(1)}}, .., S_0^{G^{(N)}}, .., S_K^{G^{(1)}}, .., S_K^{G^{(N)}}\}$
20. **return** $\mathcal{D}, \mathcal{S}$
end function

3.1.3. Histogram-Alignment Graph Feature

Optimal-assignment Weisfeiler-Lehman graph kernel [4] ultilizes the original Weisfeiler-Lehman graph kernel by applying the Hungarian matching algorithm for bipartite graphs to find the optimal matching between two sets of vertex feature vectors of two graphs. One drawback of this approach is the time complexity of the matching algorithm. To address this problem, we construct a multi-level histogram of frequencies as the fingerprint for each graph. This method is known as *Histogram-alignment graph feature*.

3.2. Graph Convolution Neural Network

3.2.1. Higher-order Representation

Given a set of vectors in d-dimensional $\mathcal{X} = \{x_1, x_2, .., x_n\}$. The first-order representation of $\mathcal{X}$ is defined as the sum of all vectors in $\mathcal{X}$:

$$\Psi_1 = \sum_{i=1}^{n} x_i \in \Re^d$$

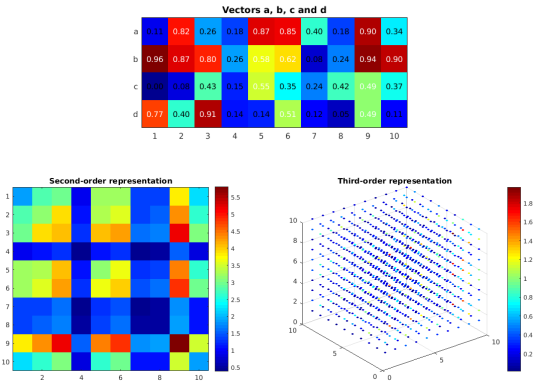
The second-order representation of $\mathcal{X}$ is defined as the sum of outer-products of all distinct pairs of vectors in $\mathcal{X}$:

$$\Psi_2 = \sum_{(i,j)\in[n]\times[n]} x_i \otimes x_j \in \Re^{d\times d}$$

Similarly, the third-order representation of $\mathcal{X}$ is defined as the sum of outer-products of all distinct triples of vectors in $\mathcal{X}$:

$$\Psi_3 = \sum_{(i,j,k)\in[n]\times[n]\times[n]} x_i \otimes x_j \otimes x_k \in \Re^{d\times d\times d}$$

Figure 2. Higher-order representations of vectors a, b, c, and d



3.2.2. Supervised Feature Training

We construct the Graph Neural Network as a multi-level hierarchy. The bottom level gets the input from vertex feature vectors and outputs the first hidden states. The middle levels produce hidden states based on the previous level. The top level is a fully-connected layer. We can take the final hidden states just before the top level as the graph feature vector or fingerprint.

Formally, each level $l \in \{0, .., L\}$ is associated with two matrices $W_l^{(1)} \in \Re^{h\times d}$ and $W_l^{(2)} \in \Re^{h\times h}$ where h is the size of hidden dimensions and d is the size of input feature. Level 0 has only matrix $W_l^{(1)}$. These matrices represent for learnable *hashing function* by Back-Propagation. Let the hidden state of vertex $v \in V$ in level $l \in \{0, .., L\}$ be denoted by ϕ_l^v. Let the learning target (for regression tasks) be $T \in \Re$ (for example, atomic energy). On top of the graph neural network is a linear regressor with the weight vector $U \in \Re^h$.

Instead of considering only the neighborhood of a vertex when constructing the sub-structure representation as [5], [6], and [7], we extend the notion of convolution to a Receptive Field that allows us to capture more graph information. A receptive field of vertex v at level l is defined as the set of all vertices that are reachable from v with a shortest-path of length at most l:

$$\mathcal{R}_l(v) = \{u \in V | d(u, v) \leq l\}$$

where $d : V \times V \to \mathbb{N}$ denotes the shortest-path distance.

To address the poor input vertex labeling problem as a one-shot vector, we initialize the input feature vector of each vertex as a hierarchical multi-level histogram of the subtree rooted at the corresponding vertex. The idea is based on Optimal-assignment Weisfeiler-Lehman graph kernel [4].

The Graph Convolution Neural Network based on Weisfeiler-Lehman iterations is described as the following pseudo-code:

function Graph Convolution Neural Network
01. Level 0: $\phi_0^v \leftarrow \sigma(W_0^{(1)} f(v))\ (\forall v \in V)$
02. for each level $l = 1 \rightarrow L$:
03. for each $v \in V$:
04. *Collect feature in the Receptive Field:*
05. $\Phi \leftarrow \bigcup_{u \in \mathcal{R}_l(v)} \phi_{l-1}^u$
06. *First-order representation* Ψ_1
07. $\Psi_1 \leftarrow \sum_i \Phi_i \in \Re^h$
08. *Second-order representation* Ψ_2
09. $\Theta \leftarrow \sum_{(i,j)} \Phi_i \otimes \Phi_j \in \Re^{h \times h}$
10. $\Psi_{2,i} \leftarrow \sum_k \Theta_{i,k} + \Theta_{k,i}$
11. *Third-order representation* Ψ_3
12. $\Theta \leftarrow \sum_{(i,j,v)} \Phi_i \otimes \Phi_j \otimes \Phi_v \in \Re^{h \times h \times h}$
13. $\Psi_3 \leftarrow$ Select the top h magnitudes from 3D tensor Θ
14. *Compute the hidden states at level* l:
15. $\phi_l^v \leftarrow \sigma(W_l^{(1)} f(v) + W_l^{(2)} \Psi)$
16. end for
17. end for
18. Graph feature: $\phi_G \leftarrow \cup_{v \in V} \phi_L^v$
19. Linear regression: $U \leftarrow argmin_{u \in \Re^h} (u^T \phi_G - T)^2$
end function

For short-hand notations, we call our Graph Convolution Neural Network with first, second, and third-order representations as GCN 1D, GCN 2D, and GCN 3D, respectively.

Given the trained Graph Convolution Neural Network, we extract the last hidden states as the graph feature vectors. Standard Machine Learning models such as Linear regression, Gaussian Process, and Support Vector Machine are all applicable with these feature vectors.

4. Experiments and Results

4.1. Dataset

The Harvard Clean Energy Project (HCEP) dataset [2] contains 2.3 million molecules that are potential future solar energy materials. By expensive Density Functional Theory, the authors of HCEP computed the Power Conversion Energy (PCE) for each molecule. PCE values are continuous ranging from 0 to 11. For our experiments, we extract 50,000 molecules and set 30,000 molecules for training, 10,000 molecules for validating, and 10,000 molecules for testing. Each molecule is represented by a SMILES code. We transform the SMILES codes into adjacancy matrices where each atom is a vertex and the atom type is the input vertex label.

For the Dictionary Weisfeiler-Lehman graph kernel, we did an investigation on the number of different receptive fields. In the set of 50,000 molecules that we selected, there are in total 11,172 different receptive fields of level 3. This number is very manageable as the size of the dictionary. We also applied *sparse* Support Vector Regression to these sparse feature vectors and obtained good results.

4.2. Visualization

For the purpose of visualization, we rounded PCE values into the nearest integers. We applied Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE) [8] on the feature vectors produced by both Weisfeiler-Lehman graph kernel and our Graph Convolution Neural Networks.

Figure 3. Weisfeiler-Lehman feature vectors with PCA and t-SNE visualizations

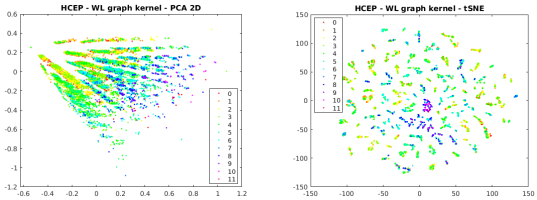
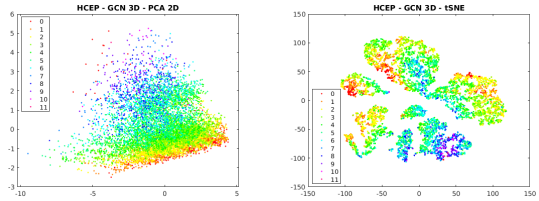


Figure 4. GCN 3D feature vectors with PCA and t-SNE visualizations



We observed that our GCN models give a much better visualization in 2-dimensional visualization. We can see the separations among the clusters where each cluster is associated with a rounded integer PCE value.

4.3. Regression tasks

We apply Linear regression, Gaussian Process and SVM for the task of regressing PCE values. We find the optimal hyperparameters in the validation set. For linear regression, the regularization parameter $C \in \{0, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 100\}$. For Gaussian Process, we choose Radial Basis Function (RBF) kernel with the optimal $\sigma = 2$, the noise parameter is selected from the range from 0 to 100. For SVM, the RBF kernel is chosen with $\sigma = 0.01$, and regualarization parameter $C = 100$.

For each representation, we select the best architecture among our graph convolution neural networks of levels 1, 2, and 3. The number of hidden states is fixed as 10. The activation function for the non-linearity mapping is chosen as Softmax. This choice of activation function outperforms Sigmoid and ReLU. Softmax can be interpreted as a *maximum pooling* layer that selects the highest magnitude signal in the hidden states.

Regarding training our neural networks, we employ Stochastic Gradient Descent with Momentum. The learning rate is initially 10^{-4} and linearly decayed to 10^{-6} in 10^6 iterations. The batch size is fixed as 100. The number of epochs is 100 in all settings.

We estimate the performance of our models in both Mean Average Error (MAE) and Root Mean Square Error (RMSE). In

both metrics, Graph Convolution Neural Network outperforms Weisfeiler-Lehman graph kernel. Among first, second, and third-order representations, the first-order GCN 1D gives the best accuracy.

We observe that trainings for higher-order representations are increasingly harder since the size of representation grows exponentially. Some techniques have been used to solve this problem, for example customized *maximum pooling* for the third-order representation. More training is probably required to obtain the full potential of higher-order representations.

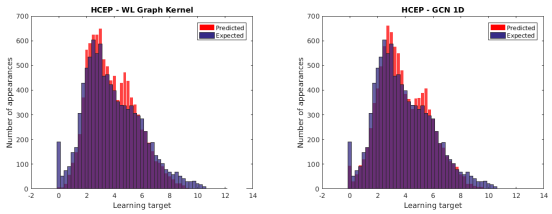
Table 1. Mean Average Error (MAE)

	Linear regression	Gaussian Process	SVM
WL	0.805135	0.760736	0.878423
GCN 1D	**0.539711**	**0.528121**	**0.493815**
GCN 2D	0.624043	0.653314	0.560620
GCN 3D	0.594986	0.602821	0.551861

Table 2. Root Mean Square Error (RMSE)

	Linear regression	Gaussian Process	SVM
WL	1.096222	1.093145	1.202667
GCN 1D	**0.800283**	**0.833758**	**0.775039**
GCN 2D	0.895965	0.940610	0.843593
GCN 3D	0.876125	0.883956	0.834215

Figure 5. Distributions of the predicted PCE values and the expected ones. Left: Weisfeiler-Lehman. Right: GCN 1D



One possible reason that the first-order representation

5. Conclusion and Future Research

We extended the state-of-the-art Weisfeiler-Lehman graph kernel and generalized convolution operation for Graph Neural Networks in order to approximate Density Functional Theory. We obtained very promising results on the Harvard Clean Energy Project dataset.

We are developing our custom Deep Learning framework in C++ named **GraphFlow** which supports symbolic differentiation and dynamic computation graph. We expect that this framework will enable us to design more flexible, efficient Graph Neural Networks at a large scale in the future.

6. Acknowledgements

I would like to acknowledge my advisor Professor Risi Kondor for his valuable instructions and especially for his great ideas in generalizing convolution operations in graphs. I also want to thank other members of Machine Learning group at the University of Chicago for their dedicated support.

Some of the neural network training in this paper was done using the Midway cluster at the UChicago Computing Research Center.

7. References

[1] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, K. M. Borgwardt, "Weisfeiler-Lehman Graph Kernels", *Journal of Machine Learning Research*, vol. 12, 2011.

[2] J. Hachmann, R. O. Amaya, S. A. Evrenk, C. A. Bedolla, R. S. S. Carrera, A. G. Parker, L. Vogt, A. M. Brockway, and A. A. Guzik, "The Harvard Clean Energy Project: Large-Scale Computational Screening and Design of Organic Photovoltaics on the World Community Grid", *The Journal of Physical Chemistry Letters*, pp. 2241–2251, 2011.

[3] R. I. Kondor, and J. Lafferty, "Diffusion Kernels on Graphs and Other Discrete Structures", *International Conference on Machine Learning (ICML)*, 2002.

[4] N. M. Kriege, P. L. Giscard, R. C. Wilson, "On Valid Optimal Assignment Kernels and Applications to Graph Classification", *Neural Information Processing Systems (NIPS)*, 2016.

[5] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, P. Riley, "Molecular Graph Convolutions: Moving Beyond Fingerprints", *Journal of Computer-Aided Molecular Design*, vol. 30, pp. 595–608, 2016.

[6] D. Duvenaud, D. Maclaurin, J. A. Iparraguirre, R. G. Bombarelli, T. Hirzel, A. A. Guzik, R. P. Adams, "Convolutional Networks on Graphs for Learning Molecular Fingerprints", *Neural Information Processing Systems (NIPS)*, 2015.

[7] T. N. Kipf, M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks", *International Conference on Learning Representations (ICLR)*, 2017.

[8] L. V. D. Maaten, G. Hinton, "Visualizing Data using t-SNE", *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.