Graph Regression and Classification using Permutation Invariant Representations

Naveed Haghani,[1] Maneesh Singh,[2] Radu Balan,[3*]

[1] University of Maryland, Applied Mathematics, Applied Statistics and Scientific Computing (AMSC) Program, College Park, MD 20742

[2] Verisk Analytics, Jersey City, NJ 07310

[3] University of Maryland, Department of Mathematics and CSCAMM, College Park, MD 20742

nhaghan1@umd.edu, msingh@verisk.com, rvbalan@umd.edu

Abstract

We address the problem of graph regression using graph convolutional neural networks and permutation invariant representation. Many graph neural network algorithms can be abstracted as a series of message passing functions between the nodes, ultimately producing a set of latent features for each node. Processing these latent features to produce a single estimate over the entire graph is dependent on how the nodes are ordered in the graph's representation. We propose a permutation invariant mapping that produces graph representations that are invariant to any ordering of the nodes. This mapping can serve as a pivotal piece in leveraging graph convolutional networks for graph classification and graph regression problems. We tested out this method and validated our solution on the QM9 dataset.

Introduction

In recent years, graph convolutional networks have gained much traction in a variety of graph inference problems, most prominently node classification and link prediction. Graph convolutions typically address such problems by producing a set of latent embeddings for each node on a graph. For a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a set of nodes and $\mathcal{E}$ is a set of edges, let $A \in \mathbb{R}^{n \times n}$ with $n = |\mathcal{V}|$ be the associated adjacency matrix for $\mathcal{G}$. Also let $z^{(i)} \in \mathbb{R}^r$ be an r length feature vector for the i^{th} node $v_i \in \mathcal{V}$. The set of all feature vectors can be represented by the matrix $Z \in \mathbb{R}^{n \times r}$ where the i^{th} row of Z is given by ${z^{(i)}}^T$. A graph convolutional network, Γ, takes in a graph along with an associated feature matrix and produces a latent embedding matrix $X \in \mathbb{R}^{n \times d}$, where the i^{th} row of X corresponds to the latent feature set for the i^{th} node: $\Gamma(A, Z) = X$.

For node classification, these embeddings can directly be employed to classify a node. For link prediction, the embeddings offer a vector representation for every node allowing a similarity metric to quantify any pair of nodes' association with one another. In this paper, we address the problem of graph regression using graph convolutional networks (Gilmer et al. 2017). Given a set of latent embeddings X produced by a graph convolutional network, a deep network classifier can learn a continuous variable over this latent feature set. This problem, however, incurs an added challenge. Any traditional deep network applied over the latent embeddings (e.g. flattened rows fed through a fully connected network) would be sensitive to the ordering of the nodes and the order in which their data is concatenated. Two distinct permutations of the same graph data would thus, in all likelihood, produce different results. In this paper we address this problem directly and study its effect on the overall graph regression problem.

We conceive that the problem of permutation sensitivity can be addressed in any of four broadly different ways. First, one can find a universally consistent scheme to order the nodes for every graph in a given dataset. Such a solution though would be problem specific and any particular problem domain is not guaranteed to lend itself to an effective scheme. Second, one can use data augmentation to feed the deep network numerous permutations of each graph in the training set. We believe this method has the potential to be very effective but may not scale well for datasets with larger graph sizes. Third, if an intermediate pooling algorithm could be developed that itself is invariant to the node ordering, successive pooling operations could be employed until the problem is scaled down far enough to make the issue negligible. Finally, one can employ a permutation invariant function that, when employed within the network, ensures the network's output is necessarily consistent across all possible permutations of any given graph. Our contribution primarily focuses on this fourth approach.

In this paper we present two novel permutation invariant mapping functions; one based on feature orderings and one based on gaussian kernels. When these mapping are applied to the latent node embeddings produced by a graph convolutional network, we get a permutation invariant representation for the graph. We use these permutation invariant mappings to build an end-to-end deep network graph classifier. We offer theoretical guarantees as to the stability of both of our mappings and injectivity of one of our mappings. We also test these methods against previous permutation invariant functions presented in the literature as well as a data agumentation approach that avoids the use of any permutation invariant mapping. We test our results on the QM9 dataset, a supervised problem of predicting a physical property of

*Author partially supported by NSF DMS-1816608 and NSF-DMS 2108900 grants, and by a Simons Foundation fellowship

Copyright © 2022, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

chemical compounds.

Prior Works on Permutation Invariant Maps

Vinyals, Bengio, and Kudlur (2015) designed the set to set algorithm to map any set of vectors into a single vector invariant to any ordering of the given vectors. The algorithm works by applying a weighted sum of the feature set over all the nodes. This sum is passed through an Long-Short Term Memory (LSTM) network to derive a new vector that is subsequently cross-compared with each node's feature set. This process is repeated an arbitrary number of times until the output of the LSTM is finally taken as the embedding.

Li et al. (2015) employs two separate neural networks, both applied over the feature set for each node. One neural network produces a set of new embeddings, the other serves as an attention mechanism whose output is multiplied to these new embeddings. The result here is then summed over over all the nodes.

Zhang et al. (2018) introduced SortPooling. SortPooling is a method to order the latent embeddings of a graph. It takes a specific column/feature set and orders all the rows according to the values each node takes in this column.

The sorting embedding has been used in applications under the name of "Pooling Map" (Zaheer et al. 2017), based on "Max Pooling".

A naïve extension of the unidimensional map $\downarrow\colon \mathbb{R}^n \to \mathbb{R}^n, \downarrow(x) = (x_{\pi(1)}, \dots, x_{\pi(n)})^T, x_{\pi(1)} \geq \cdots \geq x_{\pi(n)}$ to the case $d > 1$ might employ the lexicographic order: order monotone decreasing the rows according to the first column, and break the tie by going to the next column. While this gives rise to an injective map, it is easy to see it is not even continuous, let alone Lipschitz. Our paper introduces a novel method that bypasses this issue.

Problem

Take the equivalence relation $\sim$ on $\mathbb{R}^{n\times d}$ defined such that given $V, V' \in \mathbb{R}^{n\times d}$, then $V = V'$ if $V' = \pi V$ for some $\pi \in S_n$, the set of all $n \times n$ permutation matrices. Our goal is to find a mapping $\phi : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times D} \sim \mathbb{R}^m$ with $m = nD$ such that:

- Permutation invariance; if $V \sim V'$, then $\phi(V) = \phi(V')$
- Injectivity modulo permutations; if $\phi(V) = \phi(V')$, then $V \sim V'$
- (bi)-Lipschitz continuity with constants $A, B > 0$: for every $V, V' \in \mathbb{R}^{n\times d}$,

$$A \min_{\Pi\in S_n} \|V - \Pi V'\|_F^2 \leq \|\phi(V) - \phi(V')\|_2^2 \leq B \min_{\Pi\in S_n} \|V - \Pi V'\|_F^2 \tag{1}$$

where F denotes the Frobenius norm.

We wish to find embeddings that effectively distinguish between graphs-feature pairings that are non-isomorphic. This is achieved by ensuring our mapping is injective, i.e. given a permutation invariant representation the original embedding can feasibly be reproduced up to a row-wise permutation. We also aim to produce a stable, i.e. Lipschitz, mapping bounding the magnitude of change in the output between two distinct inputs passed through it, facilitating a deep network's ability to learn functions over this space of representations.

Our Methodology

Our deep learning framework for graph classification is comprised of three components; a graph convolutional newtork Γ, a permutation invariant mapping ϕ, and a deep neural network approximator η. Given a graph's adjacency matrix (or weight matrix) A and an associated feature matrix Z, the end-to-end deep network produces an estimate $\hat{p}$ meant to approximate some given global graph attribute p. The graph convolutional network produces a set of latent embeddings X for the graph. The permutation invariant mapping translates this latent embedding matrix into a new representation $\phi(X) \in \mathbb{R}^m$ consistent over all possible node orderings. Finally, a deep network estimates the regressed variable this representation, and thus the graph, predicts. Given labeled data, the entire network can be trained in a supervised manner. The three successive processes are expressed by the equations: $\Gamma(A, Z) = X, \qquad \phi(X) = Y, \qquad \eta(Y) = \hat{p}$

In this paper for Γ we use the Graph Convolutional Network (GCN) outlined in Kipf and Welling (2016). For our deep neural network regressor η we use a simple multilayer perceptron. The permutation invariant mapping ϕ implements two general approaches, ordering and summing.

Ordering Method Let $X \in \mathbb{R}^{n\times d}$ be our set of latent embeddings produced by applying a graph convolutional network, where n represents the number of nodes in the graph and d represents the number of latent features.

In our ordering method, we construct the permutation invariant representation by first embedding X linearly into a larger dimensional space: $X \mapsto XR \in \mathbb{R}^{n\times D}$ and then act with the sorting operator$\downarrow$ columnwise, independently from one column to another: $\phi : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times D} \sim \mathbb{R}^{nD}$, $\phi(X) = [\downarrow(Xr_1) \cdots \downarrow(Xr_D)]$, where $R = [r_1 \cdots r_D]$ are the columns of the encoding key R. The idea here is that the redundancy obtained from $X \cdot R$ will embed the row-wise relationships that would otherwise be lost by the ordering scheme if applied directly to X.

This technique can be easily shown to be permutation invariant. If D is sufficiently large, and columns of R form a full spark frame, it can be shown :

Theorem 1. *(Balan, Haghani, and Singh 2021)*

1. *If $D = 1 + (d-1)n!$ and $R \in \mathbb{R}^{d\times D}$ so that each submatrix of d columns is full rank, the map $\phi : \mathbb{R}^{n\times d}/\sim \to \mathbb{R}^{n\times D}$, $X \mapsto \phi(X) = \downarrow(XR)$ is injective and bi-Lipschitz. In this case $m = (1 + (d-1)n!)n$.*
2. *Let $\phi : \mathbb{R}^{n\times d}/\sim \to \mathbb{R}^{n\times(d+1)}$ be given as before with $R = [I_d\ 1]$ a single column of ones added to the identity matrix, then ϕ is Lipschitz everywhere with Lipschitz constant $K = 1$ and injective almost everywhere.*

This theorem guarantees the existence of a bi-Lipschitz map provided we choose a large enough embedding dimension m.

Kernels Method We define a kernel function ν designed to work as a similarity metric between two vectors a and t, $\nu : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R},\ \nu(a, t) = e^{-\|a-t\|^2}$.

This function offers a similarity score we can apply over the feature set of each node. If we are given a set of m vectors in $\mathbb{R}^d$, we can use ν to produce a similarity score for each pairing between these vectors and each node's feature set. Let $x^{(i)} \in \mathbb{R}^d$ represent the column vector representation of the feature set for the i^{th} node, i.e. the i^{th} row of X transposed to a column vector. If we are given a set of vectors $a_i \in \mathbb{R}^d$ for $i \in 1, \ldots, m$, we obtain our permutation invariant representation $Y \in \mathbb{R}^m$ by the formula: $y_i = \sum_{k=1}^{n} \nu(a_i, x^{(k)})$, for $1 \leq i \leq m$. For a given set of kernel vectors $a_i \in \mathbb{R}^d$ for $i \in 1, \ldots, m$, this mapping is always Lipschitz:

Theorem 2. *Let $\phi : \mathbb{R}^{n \times d} / \sim \to \mathbb{R}^m$ be the kernel scheme defined above mapping $X \in \mathbb{R}^{n \times d}$ to $Y \in \mathbb{R}^m$. Then ϕ is Lipschitz.*

Experiments

In this section we take an empirical look at the permutation invariant mappings presented in this paper. We focus on the problem of graph regression, for which we employ the quantum chemistry QM9 dataset (Ramakrishnan et al. 2014). In this problem, our goal is to estimate a function $F : (A, Z) \to p$, where (A, Z) characterizes a graph where $A \in \mathbb{R}^{n \times n}$ is an adjacency matrix and $Z \in \mathbb{R}^{n \times r}$ is an associated feature matrix where the i^{th} row encodes an array of r features associated with the i^{th} node. We wish to estimate a continuous scalar output $p \in \mathbb{R}_+$ associated with the entire graph. The entire end-to-end network is shown in Figure 1.

In our experiments, we employ a Graph Convolutional Network (GCN) (Kipf and Welling 2016) for Γ. For ϕ we employ both permutation invariant mappings, orderings and kernels. We compare against the case where no permutation invariant mapping is used (i.e. ϕ is set to the identity). We also compare against the case where no permutation invariant mapping is used, but data augmentation is used in its place. Our data augmentation scheme works as follows. We take the training set and create multiple permutations of the adjacency and associated feature matrix for each graph in the training set. We add each permuted graph to the training set to be included with the original graphs. In our experiments we use four added permutations for each graph when employing data augmentation. As well, we compare against a sum pooling method which sums the feature values across the set of nodes: $\phi_{sum\ pooling} = \mathbf{1}_{n \times 1}^T X$.

For the kernels method, kernel vectors are generated randomly, where each element of each vector is drawn from a standard normal distribution. Each resultant vector is then normalized to produce a kernel vector of magnitude one. When inputting the embedding X to the kernels based mapping, we first normalized the embedding for each respective node. The ordering and identity-based mappings have the notable disadvantage of not producing the same output embedding size for different sized graphs. To accommodate this and have consistently sized inputs for η, we choose to zero-pad $\phi(X)$ for these methods to produce a vector in $\mathbb{R}^m$, where $m = Nd$ and N is the size of the largest graph in the dataset.

Figure 1: End-to-end deep learning network.

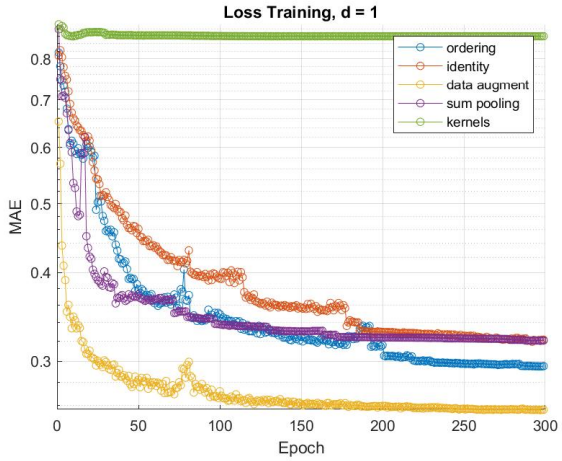
Graph Regression

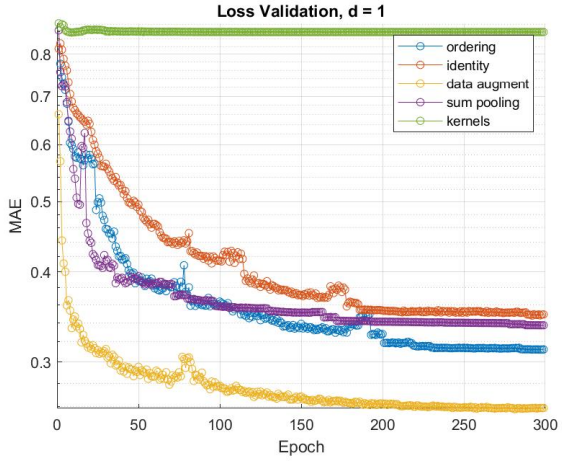
For our experiments in graph regression we consider the QM9 dataset (Ramakrishnan et al. 2014), though our regression methods were not optimized for QM9 specifically. This dataset consists of 134 thousand molecules represented as graphs, where the nodes represent atoms and edges represent the bonds between them. Each graph has between 4 and 29 nodes, $4 \leq n \leq 29$. Each node has 11 features, $r = 11$. We hold out 20 thousand of these molecules for evaluation purposes. The dataset includes 19 quantitative features for each molecule. For the purposes of our study, we focus on electron energy gap (unit eV), which is $\Delta\varepsilon$ in (Faber et al. 2017) whose chemical accuracy is $0.043eV$. The best existing regressor for this feature is enn-s2s-ens5 in (Gilmer et al. 2017) and has a Mean Absolute Error (MAE) of $0.0529eV$. We run the end to end model with three GCN layers in Γ, each with 50 hidden units. η consists of three multi-layer perceptron layers, each with 150 hidden units. We use rectified linear units as our nonlinear activation function. Finally, we vary the size of the node embeddings d that is outputted by Γ. We set d equal to 1 and 10.

For each method and embedding size we train for 300 epochs. Note though that the data augmentation method will have experienced five times as many training steps due to the increased size of its training set. We use a batch size of 128 graphs. We track the mean absolute error through the course of training. We look at this performance metric on the training set, on the holdout set, and on a random node permutation of the holdout set (see Figures 2 and 3).

Discussion

From the results we see that both the ordering and kernels method performed well for $d = 10$, though falling short of data augmentation which performed the best on both training data and holdout data. For $d = 1$, the kernels method failed to train adequately. The identity mapping performed relatively well on training data and even the holdout data, however it lost its performance on permuted holdout data. The identity mapping's failure to generalize across permutations of the holdout set is likely exacerbated by the fact that the QM9 data as presented to the network comes ordered in its node positions from heaviest atom to lightest. Data augmentation notably kept its performance despite this due to training on many permutations of the data. Our data augmentation method achieved a MAE of $0.1705eV$, which is 3.96 times larger than the chemical accuracy. This is worse than the enn-s2s-ens5 method in Gilmer et al. (2017) (current best method) that achieved a MAE 1.23 larger than the chemical accuracy, but better than the Coulomb Matrix (CM) representation in Rupp et al. (2012) that achieved a MAE 5.32 larger than the chemical accuracy whose features were optimized for this task.





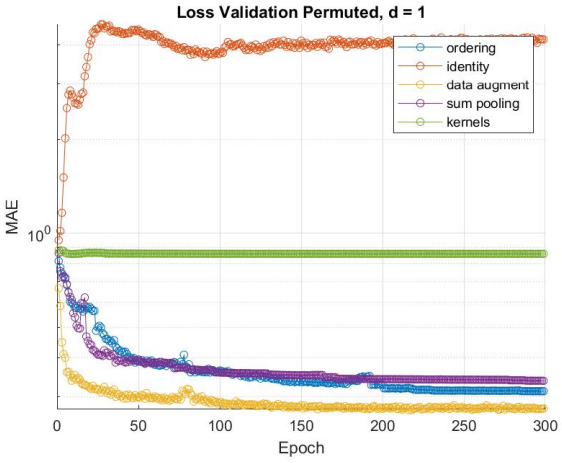
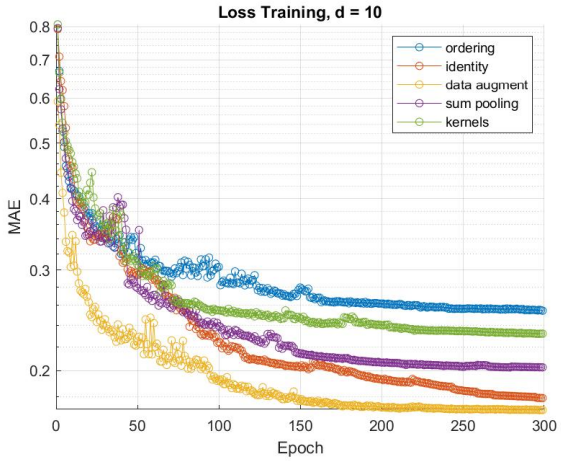
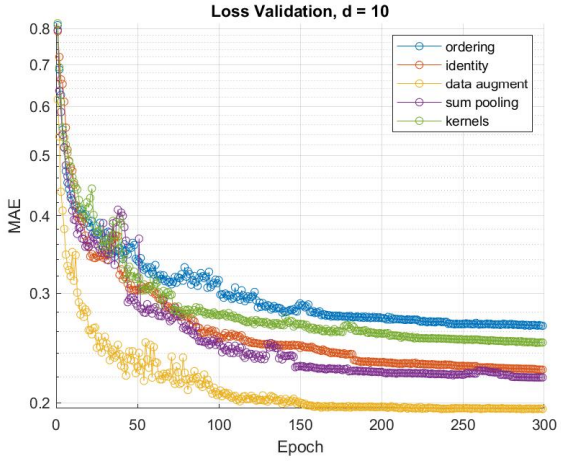


Figure 2: MAE vs Epoch, embedding size $d = 1$





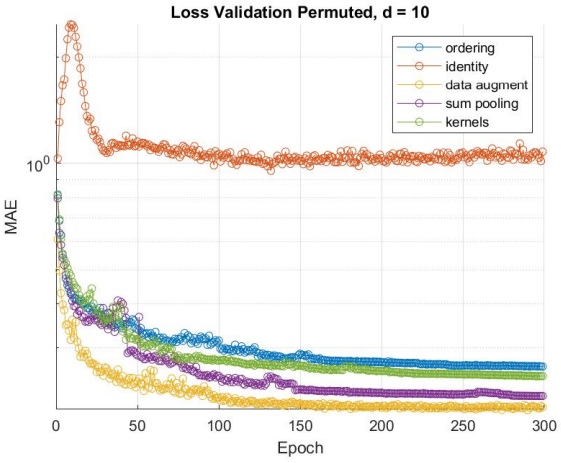


Figure 3: MAE vs Epoch, embedding size $d = 10$

References

Balan, R.; Haghani, N.; and Singh, M. 2021. Permutation Invariant Representations with Applications to Graph Deep Learning. *preprint*.

Faber, F. A.; Hutchison, B., Luke Huang; Gilmer, J.; Schoenholz, S. S.; Dahl, G. E.; Vinyals, O.; Kearnes, S.; Riley, P. F.; and von Lilienfeld, O. A. 2017. Machine learning prediction errors better than DFT accuracy. *arXiv e-prints*.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural Message Passing for Quantum Chemistry. *arXiv e-prints*, arXiv:1704.01212.

Kipf, T. N.; and Welling, M. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv e-prints*, arXiv:1609.02907.

Li, Y.; Tarlow, D.; Brockschmidt, M.; and Zemel, R. 2015. Gated Graph Sequence Neural Networks. *arXiv e-prints*, arXiv:1511.05493.

Ramakrishnan, R.; Dral, P. O.; Rupp, M.; and Von Lilienfeld, O. A. 2014. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1): 1–7.

Rupp, M.; Tkatchenko, A.; Müller, K.-R.; and von Lilienfeld, O. A. 2012. Fast and Accurate Modeling of Molecular Atomization Energies with Machine Learning. *Phys. Rev. Lett.*, 108: 058301.

Vinyals, O.; Bengio, S.; and Kudlur, M. 2015. Order Matters: Sequence to sequence for sets. *arXiv e-prints*, arXiv:1511.06391.

Zaheer, M.; Kottur, S.; Ravanbakhsh, S.; Póczos, B.; Salakhutdinov, R.; and Smola, A. J. 2017. Deep Sets. *CoRR*, abs/1703.06114.

Zhang, M.; Cui, Z.; Neumann, M.; and Chen, Y. 2018. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*.