Explore Mixture of Experts in Graph Neural Networks

Xuanyu Zhou
xuanyu98@stanford.edu

Yuanhang Luo
royluo@stanford.edu

1 Abstract

The capacity of a neural network is limited by the parameters it contains. Most state-of-the-art GNNs are limited to very shallow network due to the over-smoothing problem. Mixture of Experts, where parts of the network are activated on a per sample basis, is an effective way to increase the model capacity without a proportional increase in network depth. In this work, we show how to combine mixture of experts with Graph Neural Networks. Specifically, we propose GraphSAGE-MoE, an extension network of GraphSAGE with mixture of experts layers. We apply the network with experts to both graph classification tasks and node classification tasks. In both tasks, our results show that GraphSAGE-MoE could achieve improvement in performance compared to the baselines.

2 Introduction

GNNs have become a prominent research topic in recent years. There are several reasons for this trend, but above all, GNNs promise a natural extension of CNNs to non-euclidean data. While CNNs are very powerful when dealing with grid-like structured data *e.g.* images, it turns out that their performance on more irregular data, *e.g.* point clouds, graphs, *etc.* is sub-par. Since many real-world applications need to leverage such data, GNNs are a very natural fit. There has already been some success in using GNNs to predict individual relations in social networks [1], modeling proteins for drug discovery [2] [3], enhancing predictions of recommendation engines [4] [5], and efficiently segmenting large point clouds [6]. While these works show promising results, they rely on simple and shallow network architectures.

In the case of CNNs, the primary reason for their continued success and state-of-the-art performance on many computer vision tasks is the ability to train very deep network architectures reliably. Surprisingly, it is not clear how to train deep GNN architectures and many existing works have investigated this limitation along with other shortcomings of GNNs [7] [8] [9]. Similar to CNNs, stacking multiple layers in GNNs leads to the vanishing gradient problem. Over-smoothing problem occurs when repeatedly applying GNN many layers [7]. As a result most state-of-the-art GNNs are limited to very shallow network architectures, usually no deeper than 4 layers [9].

In this work, we show how the idea of Mixture of Experts layer can be incorporated into a graph framework. We combine the idea of sparsely Mixture-of-Experts layer(MoE) [10] [11], consisting of many feed-forward sub-networks, with the GNN variants. MoE has been adapted as a way of dramatically increasing model capacity without a proportional increase in network depth. A Trainable gating network determines a sparse combination of these experts to use for each MoE layer. The resulting models have the same depth as GNNs, but has much larger model capacity. We apply the new models, *e.g.* GraphSAGE-MoE[1] to tasks of node classification and graph classification.

3 Related work

3.1 Graph Neural Networks

GNNs use the graph structure and node features X_v to learn a representation vector of a node, h_v, or the entire graph, h_G. Modern GNNs follows a neighborhood aggregation strategy, where we iteratively

[1]The code for GraphSAGE-MoE is uploaded to GitHub at https://github.com/sparkroy/GraphSAGE-MoE

update the representation of a node by aggregating representations of its neighbors. After k iterations of aggregation, a node's representation captures the structural information within its k-hop network neighborhood. Formally, the k-th layer of a GNN is

$$a_v^{(k)} = \text{AGGREGATE}^{(k)}(\{h_u^{(k-1)} : u \in \mathcal{N}(n)\}), h_v^{(k)} = \text{UPDATE}^{(k)}(h_v^{(k-1)}, a_v^{(k)}) \tag{1}$$

where h_v^k is the feature vector of node v at the k-th iteration/layer. We initialize $h_v^{(0)} = X_v$, and $\mathcal{N}(v)$ is a set of nodes adjacent to v. The choice of $\text{AGGREGATE}^{(k)}(\cdot)$ and $\text{UPDATE}^{(K)}(\cdot)$ in GNNs is crucial. A number of architectures for AGGREGATE have been proposed. In the pooling variant of GraphSAGE [12], AGGREGATE has been formulated as

$$a_v^{(k)} = \text{MAX}(\{\text{ReLU}(W \cdot h_u^{(k-1)}), \forall u \in \mathcal{N}(v)\}) \tag{2}$$

where W is a learnable matrix, and MAX represents an element-wise max-pooling. The UPDATE step could be a concatenation followed by a linear mapping $W \cdot [h_v^{(k-1)}, a_v^{(k}]$ as in GraphSAGE. In Graph Convolutional Networks(GCN) [13], the element-wise mean pooling is used instead, and the AGGREGATE and UPDATE steps are integrated as follows:

$$h_v^{(k)} = \text{Relu}(W \cdot \text{MEAN}\{h_u^{(k-1)}, \forall u \in \mathcal{N}(v) \cup \{v\}\}) \tag{3}$$

Many other GNNs can be represented similarly to (1) [14] [15].
For node classification, the node representation $h_v^{(K)}$ of the final iteration is used for prediction. For graph classification, the POOL function aggregates node features from the final iteration to obtain the entire graph's representation h_G:

$$h_G = \text{POOL}(h_V^{(K)} \mid V \in G) \tag{4}$$

POOL can be a simple permutation invariant function such as summation or a more sophisticated graph-level pooling function. [16] [17]

3.2 Mixture of Experts

Jacobs et al. [10] introduced the original formulation of mixture of experts(MoE) models. In this work, they described a learning procedure for systems composed of many separate neural networks, each devoted to subsets of the training data. Later work [18] [19] [20] applied the MoE idea to classic machine learning algorithms such as support vector machines. More recently, several [11] [21] [22] have proposed MoE variants for deep learning in language modeling and image recognition domains.
The original mixture of experts [10] formulation combines a set of experts(classifiers), $E_1, ..., E_C$, using a mixture(gating) function G that returns a distribution over the experts given the input x:

$$y = \sum_{i=1}^{C} G(x)_i E_i(x) \tag{5}$$

Here $G(x)_i$ is the weight assigned to the i^{th} expert E_i. Later work [19] generalized this mixture of experts formulation to a non-probabilistic setting where the gating function G outputs arbitrary weights for the experts instead of probabilities.

4 Dataset

In this project, we use these real world datasets:

- Cora dataset [23]. This is a single graph dataset with nodes representing documents and edges representing citation links.

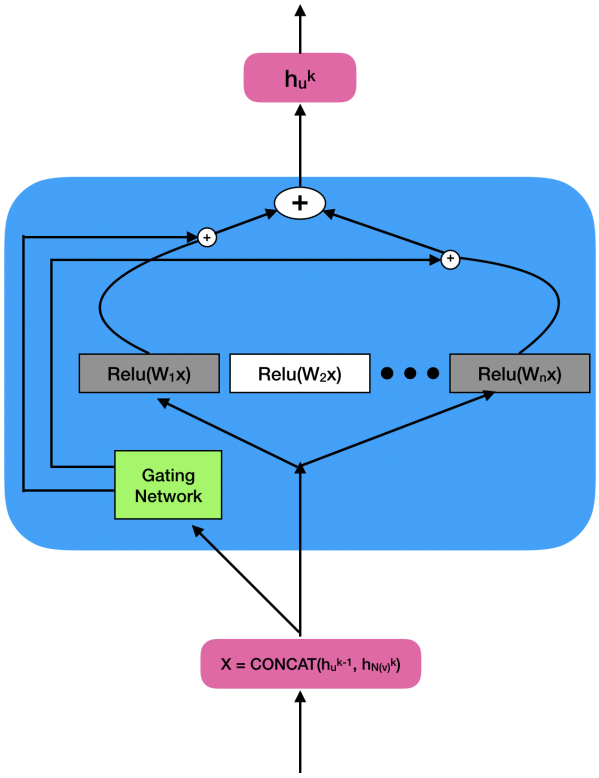

Figure 1: MoE Layer in GraphSAGE-MoE

- Pubmed dataset [24]. This is a single graph dataset with nodes representing documents and edges representing citation links.
- ENZYMES dataset [25]. This is a dataset with 600 graphs.

5 Method

5.1 Math Background

Apart from the math formula introduced in Related Works section, here are some mathematical representations of our project. Define X as the design matrix of input features. $X \in \mathbb{R}^{n \times in}$

$$X^T = [x_1, x_2, ..., x_n] \tag{6}$$

For each x_i, we have:

$$h_i = \sigma(W x_i) \tag{7}$$

where σ is a non-linear activation function.
In matrix form, this is:

$$h = \sigma(XW) \tag{8}$$

For a list of weight matrices $[W_1, W_2, ..., W_m]$,$W_j \in \mathbb{R}^{in \times out}, \forall j \in \{1, 2, ..., m\}$, suppose the corresponding distribution of the matrix for input x_i is the list of k scalar parameters $[r_{i1}, r_{i2}, ..., r_{im}]$. $\sum_{j=1}^{m} r_{ij} = 1$. The first index is the index of input, and the second index is the index of weight matrix. For each x_i We

have

$$h_i = \sigma(\sum_{j=1}^{m} r_{ij} W_j x_i) \tag{9}$$

Let $r_j^T = [r_{1j}, r_{2j}, ...r_{nj}]$ for corresponding weight matrix W_j. Then the above equation can be written in matrix form:

$$h = \sigma(\sum_{j=1}^{m} r_j * XW_j) \tag{10}$$

where r_j is broadcasted to $\mathbb{R}^{n \times out}$, and $*$ is element-wise product of matrices.

5.2 Algorithms

GraphSAGE-MoE As in [12], we use MEAN function to aggregate the features of neighbors, and then concatenate with the node feature at the previous layer.

$$h_{\mathcal{N}(v)}^k = \text{MEAN}_k(\{h_u^{k-1}, \forall u \in \mathcal{N}(v)\}) \tag{11}$$

$$x = \text{CONCAT}(h_v^{k-1}, h_{\mathcal{N}(v)}^k) \tag{12}$$

We mainly explore how mixture of experts can be useful for graph neural network. Specifically, we assume that each input x_i has a distribution of $[r_{i1}, r_{i2}, ..., r_{im}]$ that corresponds to different expert weights $[W_1, W_2, ..., W_m]$. We call these r_{ij} expert coefficients. Expert weights are shared between all inputs x_i, but expert coefficients are different for different x_i. We would like to explore how expert coefficients and expert weights can be learned, and how they can be incorporated into graph neural networks.

5.2.1 Where MoE Layers Are Used

There are two places in the Graph Neural Networks where mixture of experts could be utilized. One is when message is being passed from one node to another, and the other is after messages are aggregated.

a. Message Passing
The passing message stage before being aggregated together can be subject to mixture of experts. Each message passing is determined by the target node's weight distribution. When receiving message, the target node decides which expert weights are more important or selected. This can be represented as

$$M(h_u, h_i, e_{ui}) = \sigma(\sum_{j=1}^{m} r_{ij} W_j h_i) \tag{13}$$

b. Message Aggregated
Another scenario is after messages are aggregated. The node with aggregated messages can choose the importance of expert weights and then form its own state. Figure 1 shows how aggregated message x go through different expert matrices W_j, and form a new state h_u^k. This can be represented as

$$h^k = \sigma(\sum_{j=1}^{m} r_j * XW_j) \tag{14}$$

5.2.2 How Experts Are Mixed

We propose two approaches to define the gating coefficients for expert matrices. One is using probability distribution of roles, and the other is training a gating network to jointly learn sparsely-gating networks with the rest of the graph neural network.

After weights of gating networks are learned, information will be mixed by mixture of experts according to the two scenarios described in 5.2.1.

a. Mixture by Roles
This is a preliminary approach. Given the fact that information of neighbors will be passed to the node, we would like to see if different roles correspond to different expert matrices. More specifically, for each node, we detect the probability distribution of its role across all possible roles. Each probability is an expert coefficient corresponding to a specific expert matrix. In this way, each expert matrix will learn the weight specifically corresponding to a role. The intuition is that different roles might have different kinds of information. For now, we use GraphRole [26] as the role probability extractor for each node.

b. Mixture by Differentiable Gating Network
This approach allows the network to learn expert coefficients and expert weights jointly by training a gating network with the graph neural network. The gating network is a network that tries to learn expert coefficients with expert weights, and the learned expert coefficient matrix is a function of the input feature matrix X. Depending on the number of experts we have, the sparsity of expert coefficient matrix is different. We consider two kinds of gating networks: non-sparse gating with a trainable expert coefficient matrix, and a sparse Noisy Top-K Gating network.

(a) Non-sparse Gating As in [20], a choice of non-sparse gating function is to multiply the input X by a trainable weight matrix W_g. A Softmax function can be applied afterwards to get the probability distribution of the expert coefficients. This network will be more efficient in settings with small number of experts.

(b) Noisy Top-K Gating We adopt the Noisy Top-K Gating function as in [11]. Each single row x_i of input matrix X transforms into $F(x_i)$ first. A new trainable matrix W_{noise} is introduced here to balance the load between different expert weights.

$$F(x) = (x \cdot W_g) + \text{StandardNormal}() \cdot \text{Softplus}((x \cdot W_{noise})) \tag{15}$$

After this similar step as non-sparse gating network, sparsity comes into playing. We select top K best expert coefficients for each x, and drop the other ones. In this way, only a few expert weights are active for each input x, even if the total number of expert weights might be very large. The network finally outputs $G(x)$ for each x as the expert coefficients.

$$G(x) = \text{SOFTMAX}(\text{KeepTopK}(F(x), K)) \tag{16}$$

$$\text{KeepTopK}(v, K)_i = v_i \text{ if } v_i \text{ is in top } K \text{ elements of } v; -\infty \text{ otherwise.} \tag{17}$$

To utilize sparsity, Equation (15) cannot be directly applied to the problem of graph neural networks, due to the variable number of rows in input X. Each row x corresponds to a different set of top K expert weights, and thus for the input matrix X, all expert weights are still under consideration. Thus computation is not saved for sparsity.

To solve this, we first use average pooling over nodes to extract the graph feature from node features when calculating $F(X)$. This will allow us to use matrix multiplication for the gating function and obtain a valid probability distribution over the available experts. Only top K expert weights will be considered for input matrix X, thus saving computation.

$$F(X) = \text{AVERAGE\_POOL}(XW_g) + \text{StandardNormal}() \cdot \text{Softplus}(\text{AVERAGE\_POOL}(XW_{noise})) \quad (18)$$

Training the Gating Network The gate values for the top K experts chosen have nonzero derivatives with respect to the weights in the gating network. The rest will have zero derivatives since we assign $-\infty$ to them in the KeepTopK function and $\text{SOFTMAX}(-\infty) = 0$.

We train the gating network jointly with the rest of the network through backpropagation. Gradients also backpropagate through the gating network to its inputs. For MoE used in Message Aggregation(5.2.1.b), the equation for calculating $h_v^{(k)}$ is equation (19). Where x is calculated as in equation (12).

$$h_v^{(k)} = \sum_{i=1}^{C} G(x)_i \sigma(W_i^k \cdot x) \quad (19)$$

6 Experiments

6.1 Evaluation

We split the dataset into training and testing set. This is done either by slicing dataset for graph classification tasks or using training/testing masks for node classification tasks. Then we evaluate the performance on the testing set by calculating the accuracy of the prediction labels. The baseline model is the corresponding GraphSAGE network without mixture of experts.

6.2 Experiments to Run

There are different experiments to run when exploring the structure of the network. The basic ones are Cartesian products of {Message Passing, Message Aggregated} × {Mixture by Roles, Mixture by Differentiable Gating Network}.

Within Mixture by Differentiable Gating Network, there are choices of: {Non-sparse Gating with softmax, Non-sparse Gating without softmax, Noisy Top-K Gating}. Number of experts and number of top K experts are also hyperparameters for this problem.

6.3 Variation

One variation worth mentioning is that both message passing and message aggregated could use mixture of experts, and the trainable expert coefficient matrix for both stages can be shared. For message passing, the trainable expert coefficient matrix is W_{msg}. For update after message aggregated, the trainable expert coefficient matrices are W_{msg} and W_0. W_{msg} corresponds to h_v^{k-1} for skip connection, and W_0 corresponds to the aggregated message from its neighbors.

6.4 Results and Analysis

We have a large number of experiments to run, and we will be presenting some of the more important results in this report. First are results for Mixture by 4 Roles.

Table 1: Mixture by 4 Roles

testing accuracy			
Dataset	Cora	Pubmed	ENZYMES
Baseline	0.71	0.72	0.31
Mixture by 4 Roles - Message Passing	0.70	0.70	0.33
Mixture by 4 Roles - Message Aggregated	0.71	0.69	0.32

The results show that in Cora, Pubmed, and ENZYMES datasets, using roles extracted by GraphRole [26] to determine the mixture of experts coefficient does not outperform the baseline model. There are several possible reasons: First, We did not tune the hyperparameter enough for better results. Second, the role detection algorithm might not be very accurate. Third, the graphs in the dataset is too small to utilize the increased capacity brought by mixture of experts. Finally, Role distribution is not a very good distribution for mixture coefficients.

Table 2: Softmax

testing accuracy			
Dataset	Cora	Pubmed	ENZYMES
Baseline	0.71	0.72	0.31
4 Experts with Non-sparse Gating - Softmax	0.75	0.74	0.34
4 Experts with Non-sparse Gating - No Softmax	0.72	0.72	0.34

This set of experiments on softmax function in gating network shows that softmax ensures slightly better performance. This might be because softmax helps the learned expert coefficients become probability distribution summing up to one.

Table 3: Non-sparse Gating

testing accuracy			
Dataset	Cora	Pubmed	ENZYMES
Baseline	0.71	0.72	0.31
4 Experts with Non-sparse Gating	0.75	0.74	0.34
8 Experts with Non-sparse Gating	0.76	0.74	0.36
16 Experts with Non-sparse Gating	0.77	0.74	0.42

From this table, we see that GraphSAGE-MoE with 4 experts and Differentiable Gating has slightly higher accuracy than original GraphSAGE in all Cora, Pubmed and ENZYMES datasets, this illustrate how mixture of experts is capable of effectively increasing model capacity. From the experiments of 8 Expert with Non-sparse Gating and 16 Expert with Non-sparse Gating, we find that increasing the number of experts could further improve the model capacity and therefore achieve better performance.

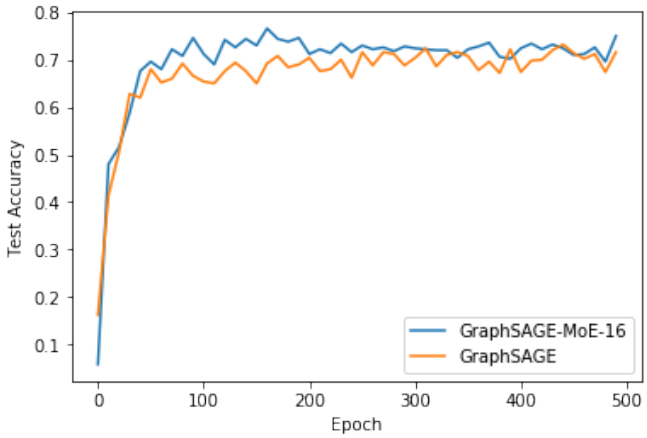

Figure 2: Test Accuracy on Cora Dataset

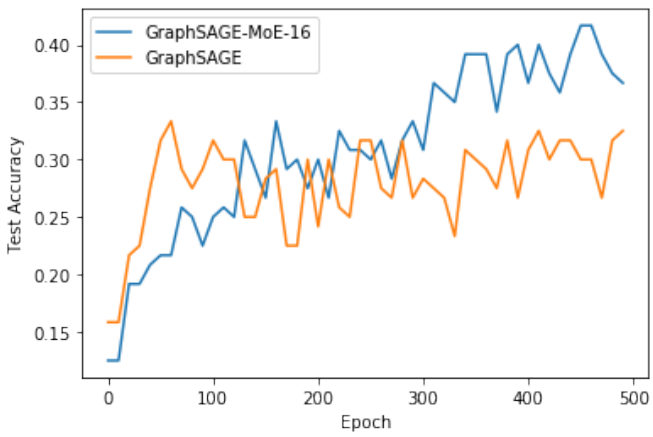

Figure 3: Test Accuracy on ENZYMES Dataset

Table 4: Differentiable Gating

testing accuracy			
Dataset	Cora	Pubmed	ENZYMES
Baseline	0.71	0.72	0.31
4 Experts with Non-sparse Gating	0.75	0.74	0.34
128 Experts with Noisy Top-4 Gating	0.73	0.73	0.43

For 128 experts with Noisy Top-4 Gating, the accuracy on Cora and Pubmed decreases a little bit compared to 4 Experts with Non-sparse Gating. We think this is because the three datasets are relatively small in size and do not require an overly high model capacity. In addition, with more experts in the MoE layers, each expert's weight will be updated less within same training iterations. Therefore, the experts in the noisy top-4 gating with 128 experts could not get well optimized compared to those networks with fewer experts.

Table 5: Message Passing and Aggregated

testing accuracy			
Dataset	Cora	Pubmed	ENZYMES
Baseline	0.71	0.72	0.31
4 Experts with Non-sparse Gating - Message Aggregated	0.75	0.74	0.34
4 Experts with Non-sparse Gating - Shared Weights	0.76	0.73	0.36

Weight sharing for learnable expert coefficient matrices between Message Passing and Message Aggregated do not show a significant difference of accuracy on the datasets compared with using MoE after Message Aggregated. This might be because sharing expert coefficients for both stages will degenerate the network into the case where MoE is only used in one of the stages (Message Passing and Message Aggregated). Alternatively, this might be because using MoE in both stages is redundant and does not perform better than using in only one stage.

7 Conclusion

This work is the first to apply Mixture of Experts to Graph Neural Networks. We also propose GraphSAGE-MoE to illustrate how to design a GraphSAGE with Mixture of Experts at each layer. As seen from experiments on Cora, PubMed and ENZYMES datasets, mixture of experts could effectively increase the model capacity. However, mixture of experts has its largest potential when the datasets are very large. Therefore, adapting Mixture of Experts on larger graph datasets would be a promising direction to explore in the future.

8 Contributions

Xuanyu Zhou: co-propose network structure, co-write the project writeup, implement the experiments of 4, 8 and 16 Mixture of Experts with Differentiable Gating and make the visualizations.
Yuanhang Luo: co-propose network structure, co-write the project writeup, implement message passing and aggregation MoE layers, mixture by roles, noisy top-k gating networks and several variations.

References

[1] Lei Tang and Huan Liu. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 817–826, New York, NY, USA, 2009. ACM.

[2] Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *CoRR*, abs/1707.04638, 2017.

[3] Nikil Wale, Ian Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowl. Inf. Syst.*, 14:347–375, 03 2008.

[4] Federico Monti, Michael M. Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. *CoRR*, abs/1704.06803, 2017.

[5] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. *CoRR*, abs/1806.01973, 2018.

[6] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph CNN for learning on point clouds. *CoRR*, abs/1801.07829, 2018.

[7] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *CoRR*, abs/1801.07606, 2018.

[8] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019.

[9] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *CoRR*, abs/1812.08434, 2018.

[10] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Comput.*, 3(1):79–87, March 1991.

[11] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. 01 2017.

[12] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017.

[13] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[14] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. *CoRR*, abs/1806.03536, 2018.

[15] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. *CoRR*, abs/1704.01212, 2017.

[16] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *CoRR*, abs/1806.08804, 2018.

[17] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *AAAI*, 2018.

[18] Ronan Collobert, Samy Bengio, and Yoshua Bengio. A parallel mixture of svms for very large scale problems. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 633–640. MIT Press, 2002.

[19] Ronan Collobert, Yoshua Bengio, and Samy Bengio. Scaling large learning problems with hard parallel mixtures. *IJPRAI*, 17(3):349–365, 2003.

[20] Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural Comput.*, 6(2):181–214, March 1994.

[21] Sam Gross, Marc'Aurelio Ranzato, and Arthur Szlam. Hard mixtures of experts for large scale weakly supervised vision. *CoRR*, abs/1704.06363, 2017.

[22] Karim Ahmed, Mohammad Haris Baig, and Lorenzo Torresani. Network of experts for large-scale image categorization. *CoRR*, abs/1604.06119, 2016.

[23] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Inf. Retr.*, 3(2):127–163, July 2000.

[24] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *CoRR*, abs/1603.08861, 2016.

[25] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016.

[26] Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. Rolx: Structural role extraction mining in large graphs. In *KDD'12 - 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 1231–1239, 9 2012.