# Classifying graphs with deep neural networks

Huy Pham

## 1 Introduction

Many interesting data come in the form of graphs, such as chemicals, drugs, social networks, etc. In this project, we attempt to understand global structures of graphs via several neural network models. Learning graphs is, however, difficult because of the lack of a canonical ordering or representation of the graph. For example, the same graph can be encoded by many different adjacency matrices. It seems that much of the difficulty involved in coming up with a learning algorithm for graphs concerns with this crucial invariance that we want to preserve.

One may attempt to classify graphs using simple network models, such as ConvNet, which operates directly on the graph adjacency matrix. As we remarked, this violates graph invariance (the ordering of nodes should not affect the graph structure) and so does not promise a good result. Furthermore, 2D ConvNet works based on the heuristic that local regions in images can be learned in the same way. However, on high-dimensional graphs (for example, random graphs) which are far from the 2D or 3D grid, the type of information each filter catches based on local nodes (in the 2D representation) is completely incompatible with the graph structure, where a node's neighborhood may consist of many nodes that are far away from it in the node ordering of the adjacency matrix. Therefore, it seems that we would lose important information using convolutional networks this way.

To fix this issue, different approaches attempt to construct graph representations that are better suited to invariant properties of graphs. One possible approach is to use a vector of frequencies of small graphlets as a kernel (for this approach and other variants, see [15]). This is however computationally expensive and highly local. Another approach, PATCHY-SAN [11] performs convolution on the graph by applying a convolutional receptive field to a normalized and canonically ordered neighborhood of each node, thus giving a generalization of the 2D convolution that better respects the graph structure.

Other approaches use ideas inspired by the skip-gram model to learn node embeddings of the graph into a vector space (node2vec [4], subgraph2vec [6]). We can then leverage the node embeddings for a graph embedding. For example, an approach demonstrated in [14] is to first apply node2vec to find the node representations, then apply a PCA to reduce dimension, and extract sequentially from every two layers a 2D encoding of the graph, by discretizing the plane, and count the number of nodes appearing in each discritized bin as the pixel value. With available node embeddings, we can also use simple pooling operations, for example, taking averages of the node features. However, we risk losing important information.

Recent developments have focused on learning data structures that have a fixed underlying graph structure, in particular, generalizing the use of convolutional neural networks over images (which can be thought of as processing vertex features on a 2D-grid graph). By leveraging the inherent two-dimensional Euclidean structure in images, convolutional networks extract better local natural features of images. Graph convolutional networks, [1, 5], generalize this to more complex underlying topology.

Graph convolutional networks operate on a fixed underlying graph, where each data point corresponds to input features on each node in the graph. We represent the graph by its adjacency matrix $A$, input features by a feature matrix $X$. Kipf and Welling [7] give a first order approximation of the graph convolution which only operates on the graph domain. Each iteration of graph convolution is then given (approximated) by

$$H^{(i+1)} = g(D^{-1/2}(\lambda I + A)D^{-1/2}H^{(i)}W^{(i+1)}),$$

where $W^{(i+1)}$ is the weight matrix of corresponding to this $(i+1)$-layer, $D$ is the degree matrix, $g$ is an activation function, and $H^{(0)} = X$. In [7], the algorithm is applied to semi-supervised tasks, where the algorithm tries to minimize the supervised loss over available features while smoothens the labels with respect to the graph structure.

In our context, the task is different and it is not immediately obvious that graph convolutions can be applied directly to graph classification problems, where we want to classify the graphs themselves (rather than some data on the graphs). However, we can learn meaningful information about the graphs by propagating some trivial features through graph convolution layers, and concatenate the resulting features on the vertices. This approach can be found, together with other relevant graph neural network architectures in [2, 10, 13].

In this project, we will test the ability to capture global graph structures of different learning algorithms. Using different architectures, we attempt to classify graphs from synthetic random graph ensembles, specifically those with very similar degree distribution or local neighborhoods, but with very different global structure.

The project code is available at https://github.com/huypham9966/deep-learning.

## 2 Data Sets

We have three different tasks with varying difficulty. In each task, we generate $n$ graphs from one ensemble and $n$ graphs from a different ensemble. We then train the network with the goal of detecting the ensemble from which the graph is generated.

In the first task, the two ensembles are Erdos-Renyi random graphs with different edge densities. For graphs in the first ensemble, each edge is sampled with probability $p_1 = 1/2$. For graphs in the first ensemble, each edge is sampled with probability $p_2 = 1/3$. Classifyng these ensembles is a simple task, corresponding to classifying vectors based on their averages.

In the second task, the two ensembles are (non-uniform) Erdos-Renyi graphs with the same global edge density but with different global structure. In particular, for graphs in the first ensemble, each edge is sampled with probability $p_1 = 1/2$. In the second ensemble, each graph is partitioned to two vertex sets of size $k/2$, each edge within one vertex set is sampled with probability $2/3$, and each edge across the two vertex sets is sampled with probability $1/3$. It is important that, in generating the data, we randomly permute the two sets of $k/2$ vertices to make sure that the model adapts to the invariant node labelings.

In the third task, for graphs in the first ensemble, each edge is sampled with probability $p_1 = 4/9$. In the second ensemble, each graph is partitioned to six vertex sets of size $k/6$, each edge going between two adjacent vertex sets is sampled with probability $2/3$, and other edges are sampled with probability $1/3$.

While the first task is easy, the second and third task are created so that the nodes in two graph ensembles have very similar degree distribution, hence, it is harder to learn about the graph structure through only local information. We focus on the second task, the third task will be used to illustrate generalizability of the architectures across different learning tasks.

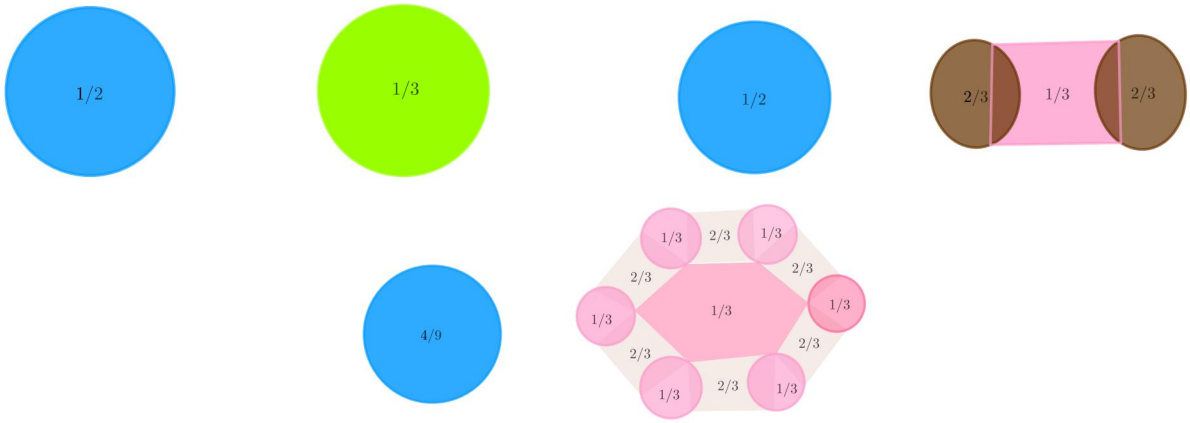


Figure 1: Classification tasks

## 3 Fully Connected and Convolutional Networks

### 3.1 Results

We encode the graphs by adjacency matrix, and use this as a direct input to fully connected and 2D convolutional networks. The fully connected neural networks have 6 hidden layers, containing $[450, 350, 250, 100, 50, 1]$ hidden units. The convolutional neural network has three stacks of two convolution layers followed by a pooling layer each. Then we attach four fully connected layers containing $[350, 100, 50, 1]$ units. We use a training set of size 2000, with graphs having $k = 36$ vertices. We use mini-batch gradient descent with the Adam Optimizer and $L_2$-regularization with batch normalization in each layer.

| Algorithms | Fully Connected | Permuted FC | Convolution | Permuted Conv |
|---|---|---|---|---|
| Test Error (%) | 27.06 | 10.25 | 6.12 | 2.03 |

Figure 2: Test error of Fully Connected and Convolutional Networks on second task

On the first task, both models achieve good performance, with test error around 1%. The second and third tasks are quite comparable for both models. While the naive 2D models do not seem to respect important graph invariances, it is surprising that they can still non-trivially generalize. This is discussed in the next subsection.

### 3.2 Regularization by permutations

There is a trick that can significantly boost the performance of these models. Before each training epoch, we would apply a random permutation of the vertices on the adjacency matrix representation of each graph in the training set. This yields a tremendous improvement in the ability to generalize of both models.

We can think of this as a heuristic to force the network weights to respect the invariance of the vertex permutations (i.e., if we let $f_\pi$ be transformation of the adjacency matrix when we apply the permutation $\pi$ on the vertices, we want the network weights to satisfy $WA = Wf_\pi(A)$). Note that the permutations are generated by $n$ transpositions, each transposition giving a linear constraint on $W$. Thus, the space of weights satisfying graph invariance has codimension $n$. Instead of using this as a hard-constraint, we permute the data in training to force the weight to get closer to the invariance subspace.

In some sense, while the hardest nature of graph is the invariance under vertex permutations, this property also helps us canonically expand the training set to improve generalization.

The number of training examples also have an important influence on the accuracy. Using 1000 examples, the test error of the convolutional network without permutation increases to 20%, while the permuted network has test error 3.3%. If instead we use 4000 examples, the permuted fully connected network has test error 1.8% while the permuted convolutional network has test error 1.7%.

## 4 Graph Convolutions

### 4.1 Models

We consider four models of graph convolutions. Since these models operate on vertex features, we initialize the vertex features at each node to be a constant vector (where all entries are 1). We then iteratively propagate the features through the graph convolution layers, learning the graph structure along the process. After $l$ iterations, we then pool the vertex features (which should now contain information about the graph structure) and build some fully connected layers as classifier. Compared to the previous approach using the adjacency matrix, the graph convolutional approach respects the graph structure and invariance much better, as it does not depend on any special ordering of the vertices. The architectures we consider are inspired by or taken from [2, 7, 10, 13, 16].

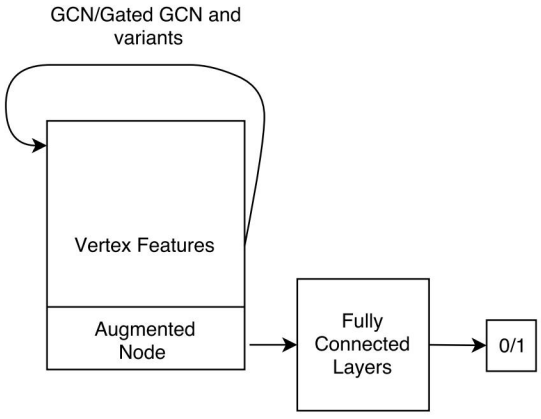


Figure 3: Graph convolution model

Instead of some canonical pooling of vertex features (which can be made adaptive), we use an augmented node that is connected to all other vertices in the graph. As we propagate features, this node will acquire information from all other nodes, and can be thought of as the graph embedding. We later realize that this approach has also been taken in [12].

Next, we discuss the four models we are interested in. We denote by $\mathbf{1}$ the matrix of dimension $n \times d$ where all entries are 1, with $n$ being the number of vertices in the graph, and $d$ being the dimension of the vertex features.

The graph convolution model has propagation rule

$$H^{(i+1)} = g(D^{-1/2}(I + A)D^{-1/2}H^{(i)}W),$$

starting from

$$H^{(0)} = \mathbf{1}.$$

Here, $g$ is a non-linear activation. We use the ReLU for $g$. We can think of the graph convolution as

Inspired by LSTM and , we consider a gated graph convolution model, with propagation rule

$$H^{(i+1)} = c_i \circ g(D^{-1/2}(I + A)D^{-1/2}H^{(i)}W) + (\mathbf{1} - c_i) \circ H^{(i)},$$

where $\circ$ denotes element-wise multiplication, and

$$c_i = g'(D^{-1/2}(\lambda I + A)D^{-1/2}H^{(i)}Z),$$

where $g'$ denotes the sigmoid activation, and

$$H^{(0)} = \mathbf{1}.$$

A natural generalization of convolution on the graph domain can be given by

$$H^{(i+1)} = g((W \circ A)H^{(i)}),$$

$$H^{(0)} = \mathbf{1}.$$

This corresponds to convolving the features in the neighboring vertices of a node with a weight matrix $W$ in each iteration.

We consider a filtered version of this, given by

$$H^{(i+1)} = g((W \circ G_i \circ A)H^{(i)}),$$

$$G_i = g'(AQ_A + H^{(i)}Q_H),$$

$$H^{(0)} = \mathbf{1}.$$

We note that in all of these models, we use graph convolutional layers in the form of unrolled recurrent neural network, with shared weight across all iterations.

## 4.2 Results

The graph convolutional models give good results. The gated convolutional model generalizes slightly better. Both element-wise convolution models are subjected to more overfitting. We also observe that on our random ensembles, we do not need to use many iterations of propagation. This may come from the fact that our graphs are very dense and well-connected. Also, while there are generalizations of the convolution on vertex neighborhoods to higher order neighborhoods, we observe that this does not work well for our task, likely because of the graphs we consider are dense and very well connected in short distances.

On a training set of 2000 graphs of 36 vertices, we use vertex features of dimension 20, and 4 iterations of graph convolutions. We also observe that training the graph convolutional models is much faster than the naive fully connected and convolutional models.

| Algorithms | GCN | Gated GCN | GConv | Filtered GConv |
|---|---|---|---|---|
| Test Error | 4.27 | 3.52 | 6.35 | 6.15 |

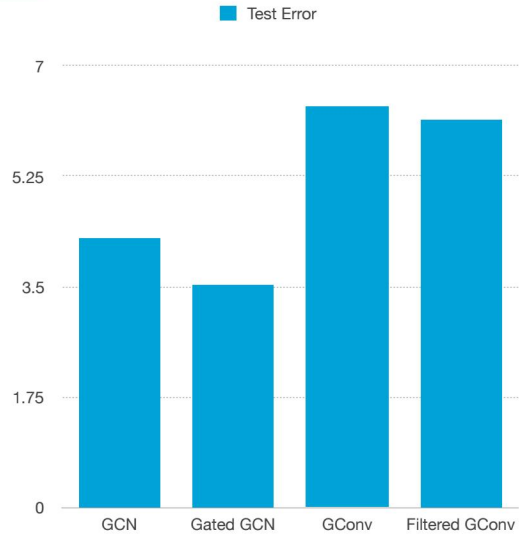


Figure 4: Comparing accuracy of different models on the second task

### 4.3 Learning embeddings across ensembles

One of the properties that give the graph convolutional models a big advantage over the naive models is their ability to generalize across tasks. To illustrate this, we train the graph convolutional models on the second task. Then we fix the convolutional iterations, take the embedding (from the augmented node) at the end, and retrain the fully connected layers with data from the third task. Surprisingly, the algorithms still do quite well. We may hope that the convolutional layers can actually extract some canonical intrinsic properties of graphs that do not overfit to the specific ensembles. If this is true in an appropriate sense, then we may leverage these architectures to learn better graph embeddings.

| Algorithms | GCN | Gated GCN | GConv | Filtered GConv |
|---|---|---|---|---|
| Test Error | 5.25 | 6.30 | 29.30 | 28.65 |

Figure 5: Learning across ensembles: iterations are trained on the second task, fine-tuned for the third task

## 5 Future work

It is interesting to see if we can get information about the graph beyond graph classification from the graph convolutional models. For example, it would be interesting if we can learn the convolution layers so that a clustering algorithm can recover the global clusters of vertices in the graph based on the vertex embeddings. One may attempt to train an end-to-end network that maximizes the embedding distances between vertices in different clusters. If this works, we are also interested in finding a learning algorithm that can segment the graph to clusters so that vertices in the same clusters are very similar (inspired by the regular decomposition of graphs from [3, 9]). We can then pool information in the same way that we do in convolutional networks.

We also want to see whether the graph convolutional models can be used as a more canonical graph embedding algorithm. On our graphs, node2vec did not seem to work well. As the convolutional models generalize well across different tasks, there may be a way to train them to do more canonical embedding. It is also interesting to see what can be changed if we start from another hand-picked initial vertex features, or whether these networks can approximate quantitative properties of graphs (such as counts of subgraphs).

# References


[1] Bruna, Joan, Zaremba, Wojciech, Szlam, Arthur, and LeCun, Yann. “Spectral networks and locally connected networks on graphs.” In International Conference on Learning Representations, 2014.

[2] Duvenaud, David, Maclaurin, Dougal, Aguilera-Iparraguirre, Jorge, Gomez-Bombarelli, Rafael, Hirzel, Timothy, Aspuru-Guzik, Alan, and Adams, Ryan P. Convolutional networks on graphs for learning molecular fingerprints. arXiv preprint arXiv:1509.09292, 2015.

[3] Frieze, Alan, and Ravi Kannan. "The regularity lemma and approximation schemes for dense problems." Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on. IEEE, 1996.

[4] Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks." Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2016.

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

[6] Narayanan, Annamalai, et al. "subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs." arXiv preprint arXiv:1606.08928 (2016).

[7] Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016).

[8] Kipf, Thomas N., and Max Welling. "Variational graph auto-encoders." arXiv preprint arXiv:1611.07308 (2016).

[9] Komlós, János, and Miklós Simonovits. "Szemerédi's regularity lemma and its applications in graph theory." (1996).

[10] Li, Yujia, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. "Gated graph sequence neural networks." arXiv preprint arXiv:1511.05493 (2015).

[11] Niepert, Mathias, Mohamed Ahmed, and Konstantin Kutzkov. "Learning convolutional neural networks for graphs." International conference on machine learning. 2016.

[12] Pham, Trang, Truyen Tran, Hoa Dam, and Svetha Venkatesh. "Graph Classification via Deep Learning with Virtual Nodes." arXiv preprint arXiv:1708.04357 (2017).

[13] Scarselli, Franco, Gori, Marco, Tsoi, Ah Chung, Hagenbuchner, Markus, and Monfardini, Gabriele. The graph neural network model. IEEE Transactions on Neural Networks, 20(1):61–80, 2009.

[14] Tixier, Antoine, Giannis Nikolentzos, Polykarpos Meladianos, Michalis Vazirgiannis, “Graph Classification with 2D Convolutional Neural Networks.” arXiv preprint arXiv:1708.02218 (2018).

[15] Yanardag, Pinar, and S. V. N. Vishwanathan. "Deep graph kernels." Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2015.

[16] Zhou, Zhenpeng, and Xiaocheng Li. "Convolution on Graph: A High-Order and Adaptive Approach." (2018).