An analysis of the strengths and weaknesses of Graph Convolution Networks for the Statistical Inference of Erdos-Reyni Graphs

Lawrence Stewart
Ecole Normale Supérieure
45 Rue d'Ulm Paris, 75005
`lawrence.stewart@ens.fr`

Abstract

A range of Neural Network architectures all based on an initial graph propagation phase have demonstrated recent success for learning problems on graphical data sets [1]. This initial phase, whether it is an aggregation or some other form of propagation, addresses the need for the model to be order invariant in the case of scalar output, or order equivariant in the case of tensor output [2]. A popular example of this type of architecture is the Graph Convolution Network (GCN), which has demonstrated impressive performance in graph classification problems [3]. This paper evaluates the GCN's learning in a regression setting, namely the inference of the parameter p of an Erdos-Reyni random graph. In particular, we show that the two layer GCN can infer this parameter after training on datasets consisting of Erdos-Reyni graphs of any connectivity provided the range of connectivities in the dataset is not too large.

1 Introduction

1.1 Graph Neural Networks: A brief review

The traditional goal of learning a graph embedding $\psi : G \rightarrow V$, a function that maps a graph to a vector space V gained popularity with numerous breakthroughs using models such as DeepWalk [4], Word Embedding [5], Node2vec [6] and many others [1]. Despite these methods yielding exciting results, they suffered heavily from computational inefficiencies due to the fact that no parameters are shared between nodes in the graph [1]. Furthermore, the direct embedding techniques did not address the desirable characteristic for the model to generalise for dynamic graphs or newly sampled graphs.

Research with the goal of resolving this weakness lead to the development of the Graph Neural Network (GNN) [7], a model which blends graph embeddings [8] with Neural Networks (NNs). The GNN model can learn signals over graphs by first learning an embedding of the graph, representing each node $v \in V$ as a vector $h_v \in \mathbb{R}^s$ which captures all the local information about a neighbourhood of the node. For unweighted graphs this state embedding is a function of the features of node v (denoted x_v), the state embeddings of nodes in some local neighbourhood of v (denoted $h_{co[v]}$) and the features of nodes in a local neighbourhood (denoted $x_{co[v]}$):

$$h_v = f(x_v, h_{co[v]}, x_{co[v]}) \tag{1}$$

A fundamental assumption of the model is that f is a contraction mapping [7]. We use the notation X and H to denote the stacking of all the node state vectors and feature vectors respectively.

The GNN is a two stage model, where the first stage consists of an aggregation phase which computes the iteration $H^{t+1} = f(H^t, X)$ up until $t = T$ for value of T sufficiently large. By the Banach Fixed

Preprint. Work in progress.

Point Theorem this iteration will converge to a graph embedding $H = f(H, X)$. The second stage is the usual feed-forward Neural Network output $O = g(H, X)$, where O is the network output and g is some chosen output function. A typical example of such an output is a pooling followed by some chosen non-linearity.

Despite the GNN's power for modelling structural data, this 'vanilla model' suffers from several limitations [1], with the primary weakness being the assumption that f is a contraction mapping. It was hence desirable to develop a new model, which like the GNN can obtain a stable embeddding of a graph, but which is free from the previous restriction on f.

1.2 Graph Convolution Networks

There have been many proposed solutions to this task and for a comprehensive review of these methods we direct the reader to [1]. This paper focuses on the spectral approach for two reasons, the first being that it has outperformed other methods by a significant margin [3]. The second reason is the fact that the spectral approach scales well with the complexity of the graph, more precisely linearly in the number of graph edges.

Spectral approaches are generalisations of Convolutional Neural Networks (CNNs) [9] to the case where a signal is defined on a graph domain. They work by exploiting the fact that the spectrum of the graph Laplacian captures the global structure of the graph and allows a generalisation of the convolution operation $\star$. To be more explicit, let $g_\theta = diag(\theta)$ be some filter parameterized by a parameter θ in the fourier domain and let $x \in \mathbb{R}^N$ be some signal. Then the graph convolution operation of the filter g_θ and the signal x is defined as:

$$g_\theta \star x = U g_\theta U^T x \tag{2}$$

where U is the matrix of eigenvectors of the symmetric normalised graph Laplacian $L = U \Lambda U^T = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$, D is the degree matrix and A the adjacency matrix of the graph. Note that $U^T x$ is the graph fourier transform of x and one should think of the filter g_θ as a function of Λ. Unfortunately computing $g_\theta \star x$ in the above manner is computationally intense and can result in non-spatially localised filters [1]. Hence [10] suggested an approximation of $g_\theta(\Lambda)$ by a truncated expansion of Chebyshev polynomials $T_k(x)$ up to the k^{th} order:

$$g_{\theta'} \star x \approx \sum_{i=0}^{k} \theta'_k T_k(\tilde{L}) x \tag{3}$$

with $\widetilde{L} = \frac{2}{\lambda_{max}} L - I_N$, where $\theta \in \mathbb{R}^k$ is a vector of Chebyshev coefficients and λ_{max} is the maximum eigenvalue of L. Note that $g_{\theta'}$ is a kth order polynomial in the Laplacian, hence it depends on nodes that are up to k connections away from the centered node.

As in the case of CNNs, one can create a linear formulation by stacking multiple layers, creating a model called a GCN. If we choose to approximate the convolution by degree one Chebyshev Polynomials ($k = 1$) then each layer represents the computation $g_{\theta'} \star x = \theta'_0 x + \theta'_1 \tilde{L} x$. In practice, [3] noted that it is beneficial to constrain the values of θ'_i with a single parameter $\theta = -\theta'_0 = \theta'_1$ to reduce over-fitting of the model, giving the following approximation:

$$g_{\theta'} \star x \approx \theta(I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) x \tag{4}$$

Stacking the operator $g'_\theta \star$ can lead to numerical instabilities and vanishing gradients. To tackle this problem [3] introduced a normalisation trick in which the term $(I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}})$ is replaced by $(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}})$, where $\tilde{A} = A + I$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. The model of the GCN can be then be summarised as follows

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} \Theta^{(l)}) \tag{5}$$

where $H^{(l)}$ denotes the l^{th} layer in the network, $\Theta^{(l)}$ is the weight matrix and σ is some chosen non-linearity.

Whilst the majority of research into GCNs evaluates performance in classification problems, this paper assesses their effectiveness in the domain of statistical inference. More precisely, we evaluate the performance of the GCN on a statistical inference problem based on the Erdos-Reyni graph model, exploring the learning process on datasets consisting of graphs with varying levels of connectivity and comparing performance to the maximum likelihood estimate.

1.3 Erdos-Reyni Random Graph Model

Before defining the inference problem, we will briefly review the Erdos-Reyni random graph model $G(n, p)$. Given a set of vertices V, where $|V| = n$, the set of edges for an Erdos-Reyni random graph $E_{G(n,p)}$ is created as follows: each possible edge is included in the graph with a probability p independent from every other edge, hence the probability of the graph having m edges is $\mathbb{P}(|E| = m) = p^m(1-p)^{\binom{n}{2}-m}$. For low values of p, sparse graphs are obtained more frequently and for high p highly connected graphs are obtained more frequently. More concretely [11]:

1. If $p < \frac{(1-\epsilon)\log n}{n}$ then a graph $G(n, p)$ will almost surely contain isolated verticies.
2. If $p > \frac{(1-\epsilon)\log n}{n}$ then a graph $G(n, p)$ will almost surely be connected.
3. If $np < 1$ then a graph $G(n, p)$ will almost surely have no connected components of size larger than $O(\log n)$.

It is easy to see that at $p = 0.5$ all $2^{\binom{n}{2}}$ graphs are chosen with the same probability.

2 Experiment

2.1 Graphical Data

Let $G(.\,;\,.)$ denote the Erdos-Reyni random model and let $\mathcal{P}$ and $\mathcal{N}$ be two random variables, where $\mathcal{N}$ is a discrete uniform random variable over some range and $\mathcal{P}$ is some continuous probability distribution with range taking values between 0 and 1.

From these two random variables, we can construct the random variable $\mathcal{G} \sim G(\mathcal{N}, \mathcal{P})$. Here we have abused the notation of $G(n, p)$ to say that $\mathcal{G}$ is a $G(n, p)$ model whose n and p values are realisations of the random variables $\mathcal{N}$ and $\mathcal{P}$. For example, if we take $\mathcal{P} \sim$Normal$(\mu, \sigma)|_{[0,1]}$, the restriction of the range of a Normal distribution with mean μ and standard deviation σ to the interval $[0, 1]$ and $\mathcal{N} \sim$ Uniform $(5, 10)$, then sampling from the distribution $\mathcal{G}$ we obtain the graphs displayed in Figure 1 and Figure 2:

Samples of $\mathcal{G} \sim (\mathcal{N}, \mathcal{P})$; $\quad \mathcal{N} \sim$ Uniform$(5, 15)$

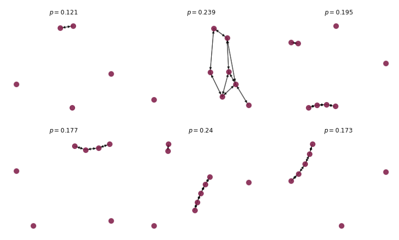


Figure 1: $\mathcal{P} \sim$Normal $(0.2, 0.15)|_{[0,1]}$

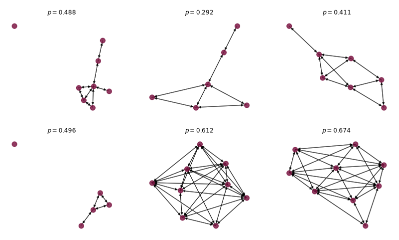


Figure 2: $\mathcal{P} \sim$ Normal $(0.5, 0.15)|_{[0,1]}$

Both of these are typical samples, as when $\mathcal{P} \sim$ Normal $(0.2, 0.15)|_{[0,1]}$ the probability of being within one standard deviation is $\mathbb{P}(0.05 \leq p \leq 0.35) \approx 0.683$ [12], so as discussed in section 1.3 we would expect a mixture of sparse graphs with a few connections as seen in Figure 1. In the case of Figure 2 we have that $\mathbb{P}(0.35 \leq p \leq 0.65) \approx 0.683$, hence we should expect a mix of lightly connected graphs and highly connected graphs.

2.2 Statistical Inference Problem

This paper concerns itself with the effectiveness of GCN's at the following inference problem:

Let $\mathcal{N}$ be a random variable distributed as discrete uniform over some range and $\mathcal{P}$ be a random variable distributed with some continuous probability density function taking range $[0, 1]$. Given a dataset $\mathcal{D} = \{(g_i, p_i)\}_{1 \leq i \leq N}$ of size N, where each g_i is a realisation of $\mathcal{G} = G(\mathcal{N}, \mathcal{P})$ and p_i is the realisation of $\mathcal{P}$, then infer to a high accuracy the value of p for any graph sampled from $\mathcal{G}$. We selected this specific inference problem as the maximum likelihood estimator exists in closed form and furthermore, it is easy to generate datasets for varied distributions of $\mathcal{P}$ whilst training.

In order to evaluate the performance of GCNs at the statistical inference of p, it is desirable to find a collection of datasets that each capture a differing range of connectivity of Erdos-Reyni graphs. A natural choice that satisfies this goal is to consider $\mathcal{P}$ normally distributed at varied means μ with a fixed small variance. Hence, we considered data-sets $\mathcal{D}_i = \{(G_i, p_i)\}_{1 \leq i \leq N}$ all consisting of a fixed $\mathcal{N} \sim$ Uniform $(5, 50)$, with varying distributions of $\mathcal{P}$:

- $\mathcal{D}_1$: $\mathcal{P} \sim$ Normal $(0.2, 0.1)|_{[0,1]}$
- $\mathcal{D}_2$: $\mathcal{P} \sim$ Normal $(0.5, 0.1)|_{[0,1]}$
- $\mathcal{D}_3$: $\mathcal{P} \sim$ Normal $(0.8, 0.1)|_{[0,1]}$
- $\mathcal{D}_4$: $\mathcal{P} \sim$ Uniform $(0, 1)$

The three normal distributions span the varying levels of connectivity, whilst the Uniform distribution is a benchmark for comparing training losses and the ability of the model to infer any equally likely p value.

2.3 Model Architecture

Both [13, 7] have found 2 layers GCNs perform the best with the approximated graph convolution detailed in equation (5). Hence our model uses a two layer GCN followed by a mean pooling $h_g = \frac{1}{|V|} \sum_{v \in V} h_v$ to extract the features, which itself is fed into a linear layer with a sigmoid non-linearity to restrict output between 0 and 1 (see Figure 3). The initial values of h_v are set to be the degree of the vertex as in [7]. Note that changing these initial values to a vector of zeros yielded no difference in results and we believe that this is the case for other initial values provided they are invariant with respect to the graph. The hidden layers of the GCN have dimension 100, and training is done in batches of 64 graphs.

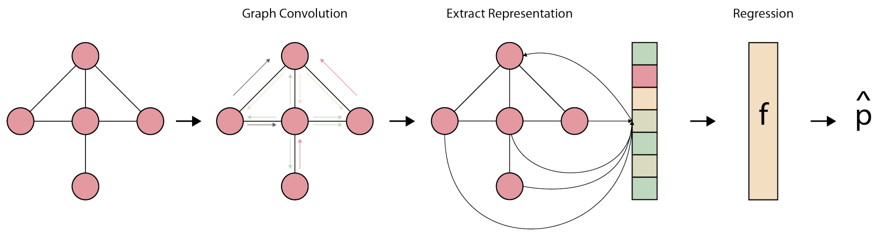


Figure 3: GCN with regression output layer

2.4 Implementation

The code for the experiment is implemented in Pytorch [14] and the graphs use the package Networkx [15]. The implementation of the GCN utilises the library DGL [16] which contains an API for sparse matrix multiplication kernels and graph batching. Our code [17] is readily available and consists of two files:

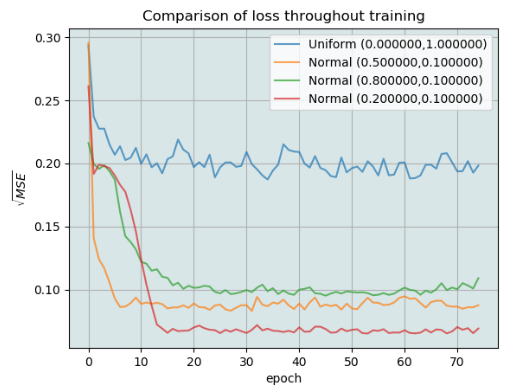
- erdoslib - a module with a selection of functions to create and manipulate Erdos-Reyni graphs datasets.
- train - a file that runs all experiments, training GCN's, saving weights and producing the plots found in this paper.

3 Results

3.1 Model Training

The above model was trained for 75 epochs, with 400 Erdos-Reyni graphs generated for each epoch. The minimised loss function ℓ was the mean square error using the ADAM optimiser [18] with learning rate $\alpha = 10^{-3}$. It was found that increasing the number of epochs did not yield significant further convergence for the model. After training, the model is evaluated on a test set of size 100.

The results are depicted in Figure 4 and Figure 5, where Figure 4 depicts the square root of the loss throughout the training process for each $\mathcal{D}_i$ and Figure 5 depicts the square root of the test and train loss for each dataset.



	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\mathcal{D}_4$
$\sqrt{\ell_{train}}$	0.0650	0.0829	0.0951	0.1872
$\sqrt{\ell_{test}}$	0.0782	0.0960	0.1274	0.2168
$\lvert \sqrt{\ell_{train}} - \sqrt{\ell_{test}} \rvert$	0.0102	0.0131	0.0323	0.0296

Figure 4: Evolution of loss throughout training (left)

Figure 5: Train and Test Losses (right)

3.2 Convergence and Overfitting

It can be seen from Figure 4 that the GCN displayed similar convergence behaviour for $\mathcal{D}_1$,$\mathcal{D}_2$ and $\mathcal{D}_3$. Amongst these three datasets a noticeable trend is that the sparser the graph data, the lower the final loss obtained by the model. Despite converging weakly on $\mathcal{D}_4$, the GCN struggled to obtain a loss similar to that of the other datasets, settling at a loss value roughly four times the magnitude of the loss value of $\mathcal{D}_3$ and oscillating with a larger magnitude.

The GCN generalised well from the training set to the test set for each of the datasets (see Figure 5). In a similar manner to the convergence of the loss, as the graph datasets become more connected, there is a slight decrease in the model's ability to generalise. Whilst the GCN generalised when trained on $\mathcal{D}_4$ rather than on $\mathcal{D}_3$, the reader should note that it is due to the poor convergence of the model on $\mathcal{D}_4$.

3.3 Comparison to Maximum Likelihood Estimator

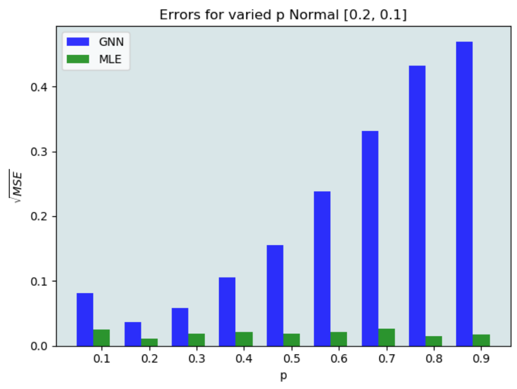
In order to assess the accuracy of the GCN after training on each of the datasets, we chose to compare it to the maximum likelihood estimator $\hat{p}_{MLE}$. For the $G(n,p)$ model, this benchmark estimator has the following form:

$$\hat{p}_{MLE} = \frac{2}{n(n-1)}|E|_G \tag{6}$$

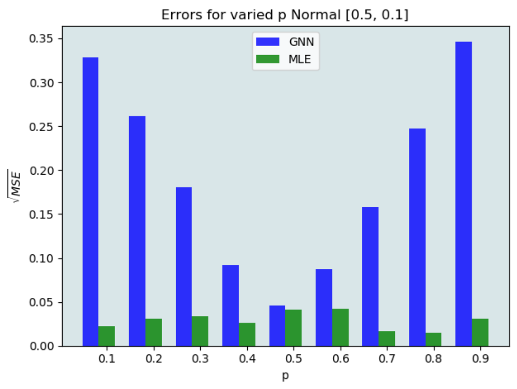
where $|E|_G$ is the number of edges in the graph. The GCN's prediction and maximum likelihood estimate were computed for a set of Erdos-Reyni graphs generated with $p \in \{0.1, 0.2, \ldots, 0.9\}$. The results are displayed in Figure 6.

Figures 4 and 6.a,b,c demonstrate that with a sufficient number of training epochs, the GCN model can learn to infer the p value from a collection of Erdos-Reyni graphs of any connectivity, achieving results close to that of the maximum likelihood estimator. However, the GCN struggles to learn such predictions when trained on a dataset that spans the entire range of graph connectivities i.e. $\mathcal{D}_4$.

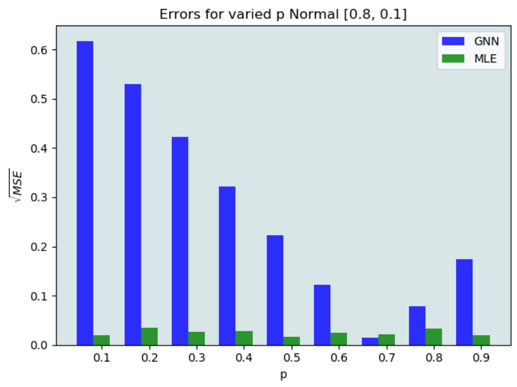
A possible explanation to this behaviour lies in the approximation of the graph convolution using Chebyshev polynomials. As this current architecture uses degree one convolution approximations, the calculation of $g_{\theta'} \star x$ at each of the two layers only depends on nodes adjacent to the centre node. This is of little importance when considering a sparse graph dataset such as $\mathcal{D}_1$, but leads to a slight increase in difficulty when learning to differentiate between a vast range of connectivities as found in $\mathcal{D}_4$. Future work addressing this problem could include searching for a method to take a higher order approximation e.g. taking $k = 2$ or $k = 3$ and finding some heavy regularisation or trick similar to that of section 1 in order to prevent overfitting. However, it is unclear as to whether this approach would be successful.



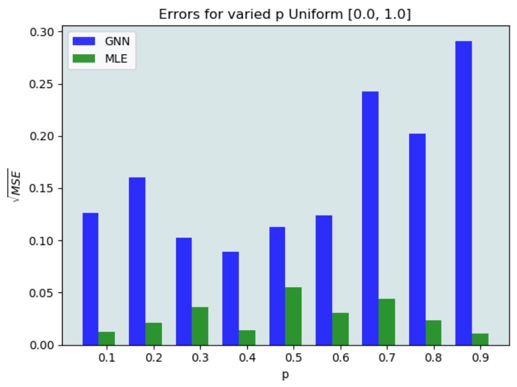
(a)



(b)



(c)



(d)

Figure 6: Prediction errors for dataset

4 Acknowledgements

Thanks go to my supervisor Professor Gabriel Peyre for providing both assistance and guidance throughout this project and also to Nicolas Keriven for many helpful discussions.

References

[1] 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. 1, 2

[2] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. In *International Conference on Learning Representations*, 2019. 1

[3] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016. 1, 2

[4] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. *CoRR*, abs/1403.6652, 2014. 1

[5] Tomas Mikolov, G.s Corrado, Kai Chen, and Jeffrey Dean. Efficient estimation of word representations in vector space. pages 1–12, 01 2013. 1

[6] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. *CoRR*, abs/1607.00653, 2016. 1

[7] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, Jan 2009. 1, 4

[8] H. Cai, V. W. Zheng, and K. C. Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, Sep. 2018. 1

[9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. 2

[10] David Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30:129–150, 12 2009. 2

[11] P. Erdős and A Rényi. On the evolution of random graphs. In *PUBLICATION OF THE MATHEMATICAL INSTITUTE OF THE HUNGARIAN ACADEMY OF SCIENCES*, pages 17–61, 1960. 3

[12] G. Casella and R.L. Berger. *Statistical Inference*. Duxbury advanced series in statistics and decision sciences. Thomson Learning, 2002. 3

[13] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017. 4

[14] 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. 2017. 4

[15] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008. 4

[16] NYU DGL team. Dgl-deepgraphlibrary pytorch. `https://github.com/dmlc/dgl`, 2017. 4

[17] L. Stewart. Repository: Graph convolution networks. `https://github.com/LawrenceMMStewart/GraphConvolutionNetworks`, 2019. 4

[18] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014. 5