ESTIMATION OF CHEEGER CONSTANT USING MACHINE LEARNING

A REPORT

submitted in partial fulfillment of the requirements

for the award of the dual degree of

Bachelor of Science-Master of Science

in

PHYSICS

by

SHIVAM PAL

(15148)

DEPARTMENT OF PHYSICS
INDIAN INSTITUTE OF SCIENCE EDUCATION AND RESEARCH BHOPAL
BHOPAL - 462066

Indian Institute of Science Education and Research Bhopal
(Established by MHRD, Govt. of India)

CERTIFICATE

This is to certify that **Shivam Pal**, BS-MS (Dual Degree) student in department of Physics, has completed bonafide work on the thesis entitled **'Estimation of Cheeger Constant using Machine Learning'** under my supervision and guidance.

May 2020
IISER Bhopal

Dr. Ambar Jain

Committee Member	Signature	Date

ACADEMIC INTEGRITY AND COPYRIGHT DISCLAIMER

I hereby declare that this project report is my own work and due acknowledgement has been made wherever the work described is based on the findings of other investigators. This report has not been accepted for the award of any other degree or diploma at IISER Bhopal or any other educational institution. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission.

I certify that all copyrighted material incorporated into this document is in compliance with the Indian Copyright (Amendment) Act (2012) and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and safeguard IISER Bhopal from any claims that may arise from any copyright violation.

May 2020 **Shivam Pal**
IISER Bhopal

ACKNOWLEDGEMENT

I am very thankful to my supervisors *Dr. Ambar Jain* and *Prof. Kashyap Rajeevsarathy* for giving me the opportunity to work on this project. I am also thankful to IISER Bhopal computer center for providing best computing resources.

ABSTRACT

Computing the Cheeger Constant (Isoperimetric Constant) of an arbitrary connected graph is generally a computationally hard problem. Cheeger inequality shows the dependence of second largest eigenvalue to Cheeger Constant. We use Machine Learning techniques to estimate Cheeger Constant of a graph using its spectrum.

LIST OF SYMBOLS OR ABBREVIATIONS

$h(G)$	Cheeger Constant of Graph G
$h_{est.}$	Estimator of Cheeger constant
Spectrum	Set of Eigenvalues
Δh	fraction deviation in Cheeger constant
$\langle \Delta h \rangle$	Average fractional deviation in Cheeger constant

LIST OF TABLES

CONTENTS

1. INTRODUCTION

There are numerous known applications of graph theory in computer science, linguistics, social sciences, physics, chemistry etc. In computer science many problems such as networking, data mining, scheduling, optimization etc are solved using graphs. In statistical physics graphs are used in percolation theory [5]. Graphs are also applied in quantum physics for perfect transfer of qubit states in the network of spin-1/2 particles [7].
A *graph* is a mathematical object that is consists of vertices and edges (see formal definition 2.1). In this thesis, we are particularly interested in a quantity related to the graphs called *Cheeger constant* (see definition 2.20). Cheeger constant is a measure of the connectivity of a graph. Computing the true value of the Cheeger constant of a graph involves subset enumeration. So, as the size of the graph grows, computing its Cheeger constant becomes computationally difficult. In fact, it is a well known NP-hard problem [2, 3, 4]. Alternatively, there are well known inequalities (see 2.23) that gives the lower and upper bounds for Cheeger constant. For *regular graph* (see difinition 2.4) the inequality indicates the dependence of Cheeger constant to the first (λ_0) and second (λ_1) eigenvalue of adjacency matrix. Which motivates us to ask the following questions.

- Does the dependence of Cheeger constant on λ_0 and λ_1 stronger than what known bounds indicate?
- Is this dependence predominantly linear or non-linear?
- How does Cheeger constant depends on other eigenvalues of the adjacency matrix?
- Can these dependencies be used to estimate the Cheeger constant with greater efficiency for large size graphs where calculating its true value is computationally difficult?

In this thesis, We did numerical analysis via machine learning techniques to test the following hypothesis.

1. Cheeger constant of the graph has predominant linear dependence on λ_0 and λ_1. Moreover, as size of the graph increases, this dependence appears to approach the linear function $\frac{1}{2}\lambda_0 - \frac{1}{3}\lambda_1$. Furthermore, there is also some non-linear component to it.

2. The Dependence of Cheeger constant to other eigenvalues of adjacency matrix is not very significant.

3. These dependence can be used together with deep neural networks (for capturing the non-linear component) to estimate the Cheeger constant effectively and efficiently for large size graphs where it is computationally difficult for classical algorithms.

The outline of the thesis is as follows. In the next chapter, we have given some basic definitions from the graph theory and known bounds for Cheeger constant. In the third chapter we have given the overview of machine learning and machine learning models (linear regression, neural network) that are mainly related to regression task and used for this analysis. In the fourth chapter we have done the numerical analysis for Cheeger constant thoroughly.

2. GRAPH THEORY

2.1 Basic Definitions

Graphs are mathematical structures used to model pairwise relations between objects. We will now formally define an undirected graph.

Definition 2.1. An **undirected, simple graph** G is a pair (V, E), where:

- V is a nonempty set called the vertex set, and
- $E \subset \{\{x, y\} : x, y \in V \text{ and } x \neq y\}$ is called an edge set.

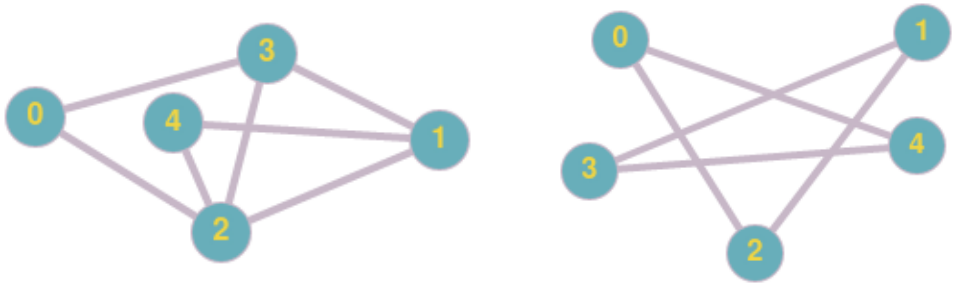
Example 2.2. In figure (1), graph $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are undirected, simple graphs.

Where $V_1 = V_2 = \{0, 1, 2, 3, 4\}$;
$E_1 = \{\{0, 3\}, \{0, 2\}, \{3, 2\}, \{3, 1\}, \{4, 2\}, \{4, 1\}, \{1, 2\}\}$;
$E_2 = \{\{0, 2\}, \{0, 4\}, \{3, 4\}, \{3, 1\}, \{2, 1\}\}$.

Definition 2.3. The **degree** $deg(v)$ of a vertex v of a graph $G = (V, E)$ is defined by

$$deg(v) = |\{e \in E : v \in e\}|.$$

Here if S is a set, then $|S|$ is the cardinality of that set.



(a) Graph G_1

(b) Graph G_2

Fig. 1: Simple undirected Graph Examples

Definition 2.4. A graph $G = (V, E)$ is called k-**regular** if

$$deg(v) = k, \forall v \in V.$$

Example 2.5. In Figure (1), graph G_2 is a 2-regular graph because all the vertex of G_2 has degree 2.

Remark 2.6. Let $G = (V, E)$ be a k-regular graph and $|V| = n$ then possible values of

$$k = \begin{cases} 1, 2, 3, ..., n-1 & \text{if} \quad n \text{ is even} \\ 2, 4, 6, ..., n-1 & \text{if} \quad n \text{ is odd} \end{cases} . \tag{2.1}$$

Definition 2.7. Let $G = (V, E)$ be a graph. A **walk** $w = (v_1, e_1, v_2, e_2, ..., v_n, e_n, v_{n+1})$ in G is an alternating sequence of vertices and edges in V and E respectively so that

$$\forall i = 1, ..., n : \{v_i, v_{i+1}\} = e_i.$$

A walk is called closed if $v_1 = v_{n+1}$ and open otherwise. A walk consisting of only one vertex is called *trivial*.

Definition 2.8. Let $G = (V, E)$ be a graph. A **path** in G is a *non-trivial walk* with no vertex and no edge repeated.

Definition 2.9. Let $G = (V, E)$ be a graph. G is called **connected graph** if for every pair of vertex v_1, v_2 in V, there exits a *walk* between them.

Definition 2.10. Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. Then G and G' are **isomorphic** if there exist a *bijective* mapping $f : V \to V'$ such that for all v_1, v_2 in V

$$\{v_1, v_2\} \in E \iff \{f(v_1), f(v_2)\} \in E'.$$

Here the mapping f is called **graph isomorphism**.

Definition 2.11. Given a graph $G = (V, E)$ with n vertices, the **adjacency matrix** is defined as

$$A := (a_{ij})_{n \times n};$$

$$\text{where} \quad a_{ij} = \begin{cases} 1 & \text{if} \quad \{i, j\} \in E \\ 0 & \text{else} \end{cases} .$$

Remark 2.12. For a simple and undirected graph $a_{ij} = a_{ji}$ so A will be symmetric matrix with all diagonal element equals to zero.

Remark 2.13. The eigenvalues $\lambda_i(G)$, $0 \leq i \leq n$ of the adjacency matrix A of G are real and can be ordered as:

$$\lambda_0(G) \geq \lambda_1(G) \geq \ldots \geq \lambda_{n-1}(G).$$

Remark 2.14. If two graphs are *isomorphic* then their eigenvalues will be same.

Example 2.15. In figure (1), the adjacency matrix of graph G_1 and G_2 are

$$A_1 = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix} \quad \text{and} \quad A_2 = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \text{ respectively.}$$

Definition 2.16. Given a graph $G = (V, E)$ with n vertices, the **degree matrix** is defined as

$$D := (d_{ij})_{n \times n};$$

$$\text{where} \quad d_{ij} = \begin{cases} deg(v_i) & \text{if} \quad i = j \\ 0 & \text{else} \end{cases}.$$

Example 2.17. In figure (1), degree matrix of graph G_1 and G_2 are

$$D_1 = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix} \quad \text{and} \quad D_2 = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix} \text{ respectively.}$$

Definition 2.18. Given a graph $G = (V, E)$ with n vertices, the **laplacian matrix** is defined as

$$L := D - A;$$

Where D and A are Degree and Adjacency Matrix respectively.

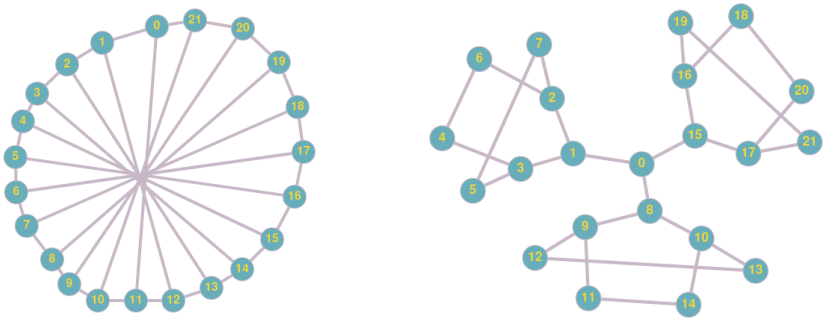
Example 2.19. In figure (1), laplacian matrix of graph G_1 and G_2 are

$$L_1 = \begin{pmatrix} 2 & 0 & -1 & -1 & 0 \\ 0 & 3 & -1 & -1 & -1 \\ -1 & -1 & 4 & -1 & -1 \\ -1 & -1 & -1 & 3 & 0 \\ 0 & -1 & -1 & 0 & 2 \end{pmatrix} \quad \text{and} \quad L_2 = \begin{pmatrix} 2 & 0 & -1 & 0 & -1 \\ 0 & 2 & -1 & -1 & 0 \\ -1 & 0 & 2 & 0 & -1 \\ 0 & -1 & 0 & 2 & -1 \\ -1 & 0 & 0 & -1 & 2 \end{pmatrix}$$

respectively.

2.2 Connectivity

A graph is an abstract mathematical object. Any real-world problem, where we are interested in the relationship between objects, can be modeled using graphs. One such example is communication networks. In the communication network, we can map the nodes of the network (e.g., computers, telephones) to vertices of the graph and connection between nodes (e.g., fiber optic cables, telephone lines) to edges of the graph. Two vertexes can communicate with each other if they are adjacent (there is an edge between them). Now suppose we are asked to construct a communication network which is consists of 22 vertices, and each vertex is connected to exactly 3 other vertexes (3-regular graph of size 22). We can do this in 2 different ways shown in figure 2 graph X and Y.



(a) Graph X (b) Graph Y

Fig. 2: 3-regular graph of size 22

Now, we can ask which graph out of X, and Y is *better* as a communication network. Here *better* is in the sense of more reliable means, how the removal of some edges in the network affects the communication in the network.
We can see that graph Y has a substantial bottleneck in the sense that removal of any edge connected from vertex 0 makes it a disconnected graph. In contrast, in graph X, removal of any two edges do not make it disconnected. Thus graph X has better connectivity and reliability as compared to graph Y. Numerically, connectivity and reliability of a graph can be measured using the quantity called *Cheeger Constant*. Formal definition of Cheeger constant is given below.

Definition 2.20. (**Cheeger Constant**) Let $G = (V, E)$ be a finite, simple, connected and undirected graph. For each $F \subset V$, let $\partial F := \{\{u, v\} \in$

$E(X) : u \in F, v \in V \setminus F\}$. Then the number

$$h(G) = \min_{F \subset V, |F| \leq \frac{|V|}{2}} \frac{|\partial F|}{|F|}, \tag{2.2}$$

is called the *Cheeger constant* (or the *isoperimetric constant* or the *edge expansion constant*) of the graph G.

Remark 2.21. As we can see from the definition, the calculation of Cheeger constant involves subset enumeration. Consequently, as a graph increases in size, the number of subsets grows exponentially, which makes the calculation of the Cheeger Constant computationally complex.

Remark 2.22. Cheeger constant of disconnected graph is zero.

Remark 2.23. (**Bounds for Cheeger Constant**) Although computation of exact Cheeger constant for large size graph is computationally hard problem but there are well known bounds for it.
Let $G = (V, E)$ be a finite, simple, connected and undirected k-regular graph with $|V| = n$. The eigenvalues $\lambda_i(G)$, $0 \leq i \leq n$ of the adjacency matrix $A(G)$ of G are real and can be ordered as:

$$k = \lambda_0(G) \geq \lambda_1(G) \geq \ldots \geq \lambda_{n-1}(G) \geq -k.$$

Then the Cheeger constant $h(G)$ of G is related to the *spectral gap* $k - \lambda_1(G)$ of G by the following inequality [1]:

$$\frac{k - \lambda_1(G)}{2} \leq h(G) \leq \sqrt{2k(k - \lambda_1(G))}. \tag{2.3}$$

Moreover, Mohar [4] showed that

$$h(G) \leq \begin{cases} \frac{k}{2}\left[\frac{n}{n-1}\right], & \text{if } n \text{ is even, and} \\ \\ \frac{k}{2}\left[\frac{n+1}{n-1}\right] & \text{if } n \text{ is odd.} \end{cases}, \tag{2.4}$$

and when $G \neq K_1$, K_2, or K_3, he showed that

$$h(G) \leq \sqrt{k^2 - \lambda_1(G)^2} \tag{2.5}$$

and

$$h(G) \leq \sqrt{\beta(2k - \beta)}. \tag{2.6}$$

Where β is second smallest eigenvalue of laplacian matrix and K_i is fully connected graph of size i.

3. MACHINE LEARNING

Machine learning is synonyms for *finding patterns* in the *data*. There are different types of learning depending upon the type of data, such as *supervised learning, unsupervised learning, reinforcement learning* etc.
In **supervised machine learning**, we have data (pairs of input and output) $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$ where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. There exist a function $f : \mathcal{X} \to \mathcal{Y}$ that satisfies all input, output pairs but its exact mathematical form is unknown. The whole purpose of machine learning is to find out this unknown function. We choose a hypothesis set $\mathcal{H} = \{h\}$, then using a *learning algorithm*, a hypothesis $g : \mathcal{X} \to \mathcal{Y}$ is selected from $\mathcal{H}$ that approximates the true function f. Hypothesis set $\mathcal{H}$ and *learning algorithm* together called *machine learning model*. For example, we choose the *neural network* as a hypothesis set and *back-propagation* as a learning algorithm. When $\mathcal{Y}$ is continuous, it is called a *regression* problem. When $\mathcal{Y}$ is discrete, it is called a *classification* problem.

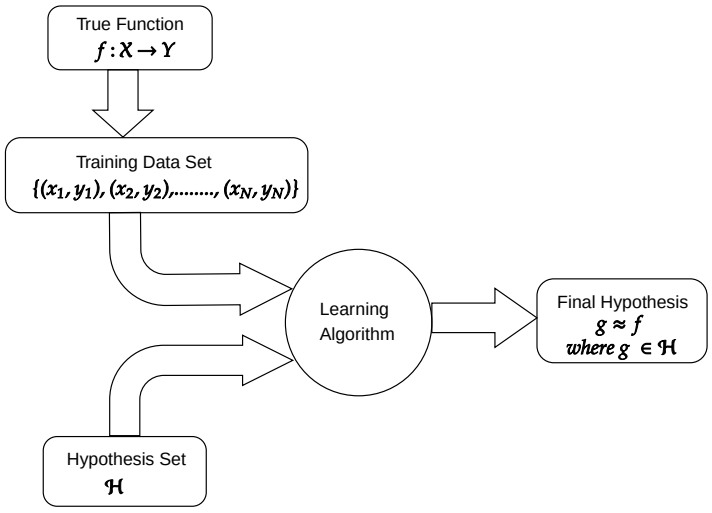


Fig. 3: Overview of Machine Learning

To measure the **accuracy of machine learning model**, we usually split the entire data set into two parts, **training data set** and **test data set**. The machine learning model is trained via a learning algorithm on the training data set only. Then the output of the test data set is predicted via the trained model. And error in the test data set is calculated by comparing the predicted output and true output of the test data set. The comparison is made via a suitable metric that depends on the nature of the output. For continuous output, we may choose the accuracy metric to be mean absolute difference between predicted output and true output. Similarly, error in the training data set can also be calculated. If the error in both training and test data set is high, then the trained model is **under-fitted**. Whereas if error in the training data set is low while in the test data set is high, then the trained model is **over-fitted**. A good machine learning model will have low error in both training and test data set.

3.1 Linear Regression

Linear regression is a machine learning model where hypothesis set $\mathcal{H}$ is a set of linear functions. Let's assume the data $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_N, y_N)\}$ where $\mathbf{x}_i \in \mathbb{R}^D$ and $y_i \in \mathbb{R}$. Then $\mathcal{H} = \{h(\mathbf{x}, \mathbf{w})\}$ where $\mathbf{w} \in \mathbb{R}^{D+1}$ and

$$h(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{i=1}^{D} w_i x_i. \tag{3.1}$$

Thus for each values of $\mathbf{w}$, we have a linear function in $\mathcal{H}$. Now our goal is to find the $\mathbf{w}$ for which the quantity

$$E(\mathbf{w}) := \sum_{i=1}^{N} (y_i - h(\mathbf{x}_i, \mathbf{w}))^2 \tag{3.2}$$

is minimum. $E(\mathbf{w})$ is called *loss function* or *cost function*. Process of finding $\mathbf{w}$ for minimum error function is called *optimization*. Here we can optimize it by setting the gradient of $E(\mathbf{w})$ with respect to $\mathbf{w}$ equal to zero.

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = 0$$

Then,solving for $\mathbf{w}$, we obtain

$$\mathbf{w}^* = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y} \tag{3.3}$$

where

$$\Phi = \begin{pmatrix} 1 & \mathbf{x}_1^T \\ 1 & \mathbf{x}_2^T \\ \vdots & \vdots \\ 1 & \mathbf{x}_N^T \end{pmatrix} \quad \text{and} \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}.$$

Here Φ is $N \times (D+1)$ matrix, called *design matrix*. Because here we have chosen $E(\mathbf{w})$ to the square error loss function, we can solve it for $\mathbf{w}$ analytically. For an arbitrary loss function, it may not be solvable analytically, but another optimization algorithm called *gradient descent* can be used in that case. The only condition for gradient descent to work is that loss function must be continuous and differentiable. In the next section, the gradient descent algorithm is explained.

3.2 Neural Networks

The idea of the neural network is closely related to the biological neurons. We will explain the neural network in the following steps.

3.2.1 Model

Perceptron or a artificial neuron is a building block of neural network. A artificial neuron takes a set of inputs x_1 to x_d then each of these inputs is multiplied with corresponding weights w_1 to w_d and summed up. Bias w_0 is added to the sum and the sum is passed through a activation function to get the output $\hat{y}$ of the neuron.

$$\hat{y} = g(w_0 + \sum_{i=1}^{d} w_i x_i) \tag{3.4}$$

where g is *activation function*. Purpose of activation function is to introduce non-linearity into the model. Here are few example of non-linear activation functions. Visual representation of equation: 3.4 is shown in figure 6.

1. **Sigmoid** activation function $g(x) = \frac{1}{1+e^{-x}}$
2. **Hyperbolic Tangent** $g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
3. **Rectified Linear Unit**(ReLU) $g(x) = max(0, x)$

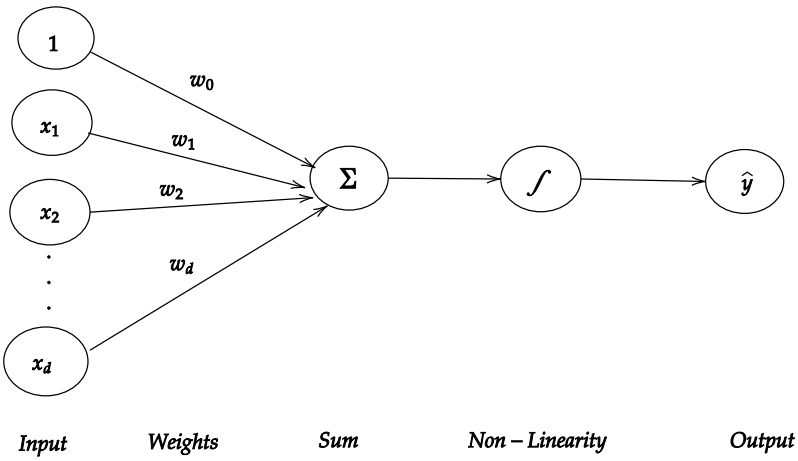


Fig. 4: Single neuron in artificial neural network

Output of a neuron can be used as input for the other neuron and in this way a deep neural network can be constructed. Different type of activation function can be used in different neurons. In the figure 5 we have shown a example of deep neural network with 3 hidden layers, where first hidden layer has 8 neuron and second and third hidden layer has 6 neurons. Neurons from second hidden layers gets input from first hidden layer similarly second hidden layer works as input for the neurons in third hidden layer.

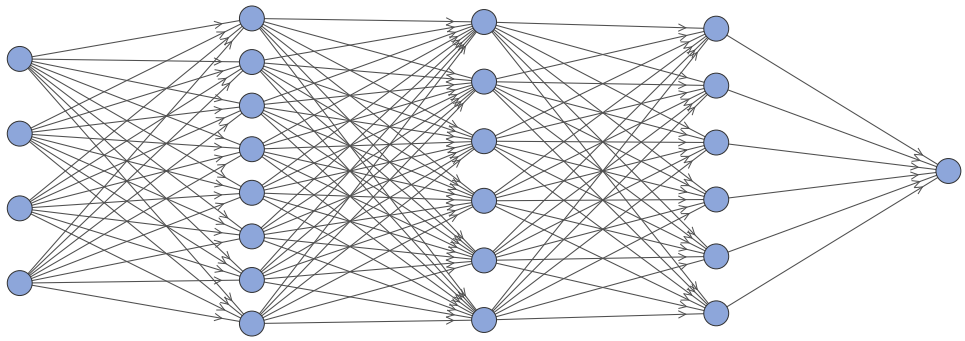


Fig. 5: Example of a deep neuron network with 3 hidden layer

In figure: 5, Each node in hidden layers represent a neuron and nodes in input layer represent input feathers of input vector. The representation power of these multi-layer networks of neurons is enormous. In fact, according to universal approximation theorem neural network with a single hidden layer containing enough neurons can be used to approximate any continuous function to any desired precision.

3.2.2 Training Neural Network (optimization)

Weights and biases in each neuron are the parameters of neural network. Let's $\mathbf{W}$ be parameter matrix and $f(\mathbf{x}, \mathbf{W})$ be neural network model where $\mathbf{x}$ in input vector corresponding to output y . Loss function can be defined similar to equation 3.2 except model is now changed. Now we have to find $\mathbf{W}$ such that

$$\mathbf{W}^* = \underset{\mathbf{W}}{\mathbf{argmin}} \quad E(\mathbf{W}) \tag{3.5}$$

Where, $E(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(\mathbf{x_i}, \mathbf{W}))^2$

For optimization gradient descent algorithm or its other variant can be used. Basic gradient descent algorithm is shown below. η in the algorithm is called *learning rate* and it is responsible for convergence. Small values of η can take many steps before reaching minima while large values can overshot local minima. Thus often in practice other variant of gradient descent algorithm (based on adaptive learning rate) are used such as ADAM, Momentum based Gradient Descent,etc. In gradient descent algorithm, each time when weights are being updated, $\frac{\partial E(W)}{\partial W}$ is computed. For large training data set, this computation can be costly so entire dataset can be randomly spitted into different batches and each time one batch is used to update weights instead of using entire dataset which makes training fast. Number of data points in a batch is called **batch size**. For batch size equal to 1, algorithm is called as *stochastic gradient descent*. When all the batches one by one (one pass of training dataset) are completed updating weights, it is called one **epoch**. Often many epochs are needed depending upon batch size for better optimization.

Algorithm 1: Gradient Descent Algorithm

1 **Initialize weights randomly,** $W \sim \mathbb{N}(0, \sigma^2)$
2 **until convergence**
3 | **Compute gradient,** $\frac{\partial E(W)}{\partial W}$
4 | **Update weights,** $W \leftarrow W + \eta\frac{\partial E(W)}{\partial W}$
5 **End**
6 **return** W

In the next chapter we have done the numerical analysis using linear regression and neural network. For the practical purpose, we use a machine learning framework called tensorflow for the analysis.

4. NUMERICAL ANALYSIS OF CHEEGER CONSTANT

For this analysis we have considered simple, un-directed, un-weighted, connected and regular graphs only.

4.1 Data Set

We generate k-regular graphs of node size 12 to 30 for all possible values of k shown in 2.1. Graphs are generated using a Python package that implements the algorithm described in [6]. Corresponding each graph, its Cheeger Constant is computed using the algorithm given in the appendix I. Because we generated graphs randomly, it is possible to get duplicate and isomorphic graphs. We computed the eigenvalues of adjacency matrix of each graph and the graphs having same set of eigenvalues (duplicate and isomorphic grphs) were removed from the dataset. Thus our final dataset consist of eigenvalues(sorted in descending order) and Cheeger constant for each graph. Similarly the dataset for other node size is generated. The number of graphs generated in each dataset is shown in the table 4.2. A small chunk of dataset for regular graph of size 12 is given in table 4.1.

λ_0	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8	λ_9	λ_{10}	λ_{11}	Cheeger
2.0	1.732	1.732	1.000	1.000	0.000	-0.000	-1.000	-1.000	-1.732	-1.732	-2.000	0.333333
3.0	2.285	1.750	1.241	0.618	0.194	-0.421	-1.000	-1.373	-1.618	-2.073	-2.603	0.600000
3.0	2.000	1.814	1.000	0.732	-0.000	0.000	-0.471	-1.000	-2.000	-2.343	-2.732	0.666667
3.0	2.273	1.438	1.323	0.545	0.429	-0.271	-1.000	-1.000	-1.902	-2.140	-2.695	0.600000
3.0	2.224	1.956	1.241	0.209	0.000	0.000	-1.000	-1.338	-1.710	-1.827	-2.755	0.600000

Tab. 4.1: Small chunk of the dataset of node size 12.

In table 4.1, Each row is a graph and columns are sorted eigenvalues. Last column is Cheeger constant of the graph. First graph is 2-regular and others are 3-regular.

4.2 Known bounds

For each graph G in each dataset of node size $n = 12$ to 30, we compute the lower bound on $h(G)$ as given by Eqn. (2.3), and an upper bound, which is the lowest of the upper bounds appearing in equations (2.3) to (2.6). For each of these estimators, we calculate its deviation Δh from the true value of $h(G)$ as given in the equation below:

$$\Delta h_{\text{est.}} = \left| \frac{h_{\text{est.}} - h(G)}{h(G)} \right| \tag{4.1}$$

where, $h_{\text{est.}}$ refers to the estimator of $h(G)$. For the analysis in this section, corresponds to either the upper bound or the lower bound. The mean values of $\Delta h_{\text{est.}}$ for each n is shown in Table 4.2 below.

n	# of Graphs	$\langle \Delta h_{\text{lower}} \rangle$	$\langle \Delta h_{\text{upper}} \rangle$
12	15176	0.18	0.61
13	55128	0.23	0.61
14	115663	0.18	0.60
15	118702	0.22	0.63
16	22635	0.18	0.65
17	20024	0.21	0.61
18	35774	0.18	0.66
19	20436	0.20	0.59
20	56016	0.18	0.64
21	1606	0.19	0.56
22	1626	0.17	0.61
23	1636	0.19	0.55
24	1825	0.16	0.59
25	1385	0.18	0.53
26	1829	0.16	0.57
27	1722	0.17	0.52
28	1097	0.16	0.57
29	958	0.20	0.62
30	872	0.16	0.59

Tab. 4.2: Graph data considered in the analysis and the average deviation in bounds.

In table 4.2, The second column shows the number of graphs considered in the analysis for each n. For $n \leq 20$, at least 20,000 graphs were considered for each n with exception to $n = 12$, where the total number of available graphs is less than 20,000. For $n > 21$, we tried to accumulate at least about 1000 graphs with the exceptions of $n = 29$ and $n = 30$. We note that the lower bound deviates from the true value of $h(G)$ by about 20%, while the upper bound deviates at about 60%. This deviation marginally reduces, on average, for large values of n. The table indicates that the bounds considered are not efficient estimators for $h(G)$. In the next section, we consider linear regression to construct a better estimator for $h(G)$.

4.3 Linear regression analysis and prediction

In this section we want to determine weather the relationship between $h(G)$ and $\lambda_0(G)$ and $\lambda_1 G$ is predominantly linear. We assume the linear model

$$h_{\text{est.}}(G) = \sum_{i=0}^{m} a_i \lambda_i(G) + c. \tag{4.2}$$

For each $m \in \{0, 1, 2, 3\}$, we fitted the mode for all the datasets of node size $n = 12$ to 30 using the procedure given in the section 3.1 and mean deviation $(\Delta h_{est.})$ given by Eq 4.1 is calculated. Results are shown in the figure 6.

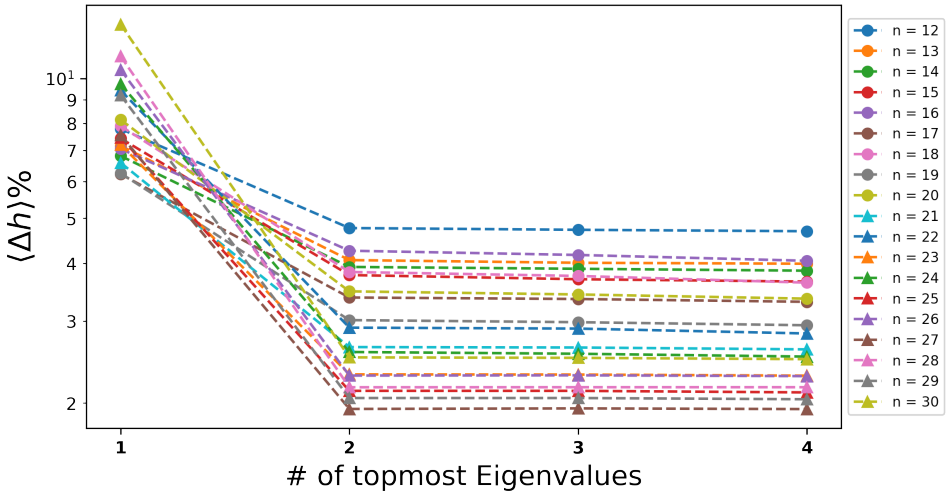


Fig. 6: Linear regression on Cheeger constant.

Figure 6 shows average deviation of $h(G)$ from the estimate obtained through a linear fit for topmost, top two, top three and top four eigenvalues. Points are joined by lines to guide the eye. Log scale is used on the y-axis to stretch the scale. There is no considerable improvement in the Cheeger estimate from linear regression beyond $\lambda_0(G)$ and $\lambda_1(G)$. It is evident from the figure 6 that adding the third and fourth eigenvalue to the analysis does not significantly reduce $\langle \Delta h \rangle$. This shows that a linear function of just the two largest eigenvalues estimates $h(G)$ fairly accurately Interestingly, the average deviation of $\langle \Delta h \rangle$ reduces gradually with increase in n. This observation confirms that the relationship between the two largest eigenvalues and $h(G)$ is mostly linear.

The regression coefficients of $\lambda_0(G)$ appears to converge to $\frac{1}{2}$ as n increases, while the coefficient of $\lambda_1(G)$ appears to converge to $-\frac{1}{3}$. The coefficients a_0,

a_1 and c of the model $a_0\lambda_0(G) + a_1\lambda_1(G) + c$ are plotted in Fig. 7 below for each n along with lines corresponding to $1/2$ and $-1/3$ for reference.

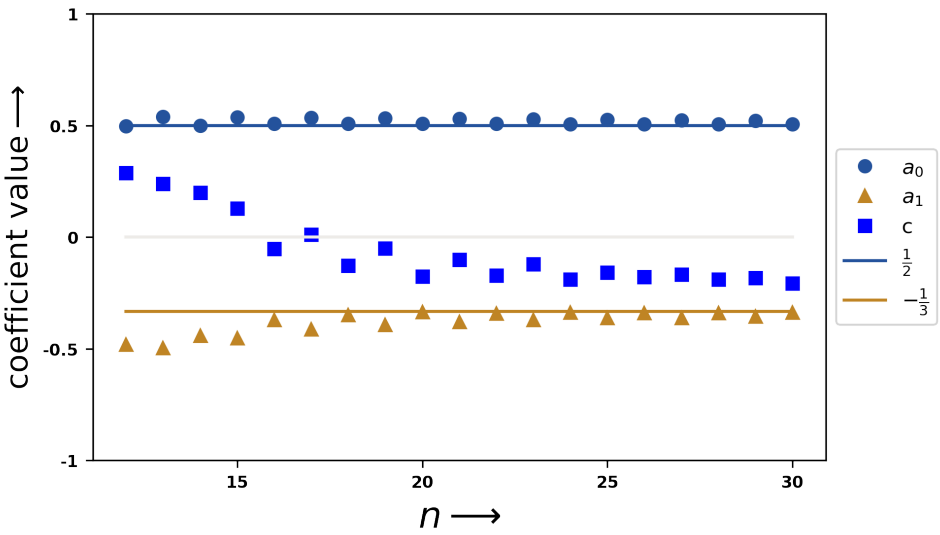


Fig. 7: Coefficients of linear regression on Cheeger constant.

This suggests a universality in the linear relationship, which is almost independent of n. This observation motivates us to test the linear model for prediction of Cheeger constant for larger n where it is computationally challenging to calculate $h(G)$. We train the linear regression model on the available data for $n = 12, 13, 16, 17$ and then use it to predict Cheeger constant for other n. Using the trained linear model as the estimator, we show a mean deviation of $\langle \Delta h \rangle$ in Fig.8 below.

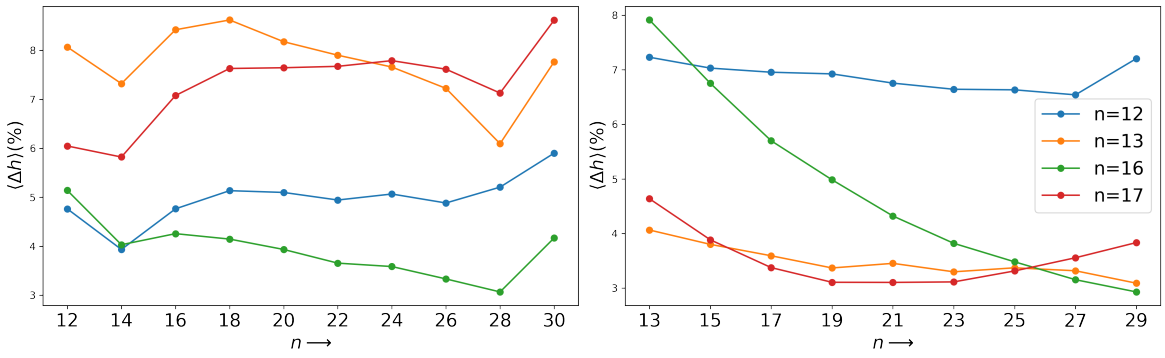


Fig. 8: Predicting with linear regression : Linear models trained on Cheeger constant data for n = 12, 13, 16, 17 are used to predict Cheeger constants for graphs of other n.

The left panel shows prediction for even n, while the right panel shows prediction for odd n. We make the following observations

1. In general, for large n, linear regression with $\lambda_0(G)$ and $\lambda_1(G)$ appears to be a reasonable estimator for $h(G)$.

2. The prediction is slightly more accurate when regression on odd n (resp. even n) is used to predict the $h(G)$ for larger values of odd n (resp. even n.)

3. Average deviation Δh is typically 4-5% for odd-odd and even-even predictions for the entire range of n considered.

4. It also appears that $n = 16$ and $n = 17$ linear models are slightly better over $n = 12$ and $n = 13$ models respectively for even-sized and odd-sized graphs respectively.

4.4 Estimation of Cheeger constant using Neural Network

In this section, we study the data on Cheeger constant using machine learning methods with deep neural networks, mainly to answer following two questions

1. Does $h(G)$ have a non-linear dependence on $\lambda_0(G)$ and $\lambda_1(G)$?
2. Does $h(G)$ has any significant dependence on other eigenvalues?

We expect that machine learning techniques will be able to identify non-linear dependencies that were not visible through linear regression. We randomly take 40% of our dataset for $12 \leq n \leq 30$ and train a deep neural network shown in Fig. 9 below [1] using ADAM optimizer.

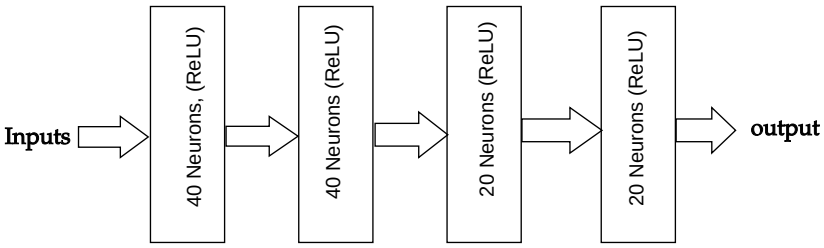


Fig. 9: Neural Network architecture used.

The remaining 60% of the dataset is used for validation. The trained neural net essentially provides an approximate non-linear map between the input eigenvalues and the expected Cheeger constant. The validation ensures that there is no memorization done by the neural net and that it is truly capturing features of the data. Fig. 10 below shows training and validation histograms of Δh for $n = 12$, for both the cases of trainings done with the largest two and the largest four eigenvalues.

[1] We have observed that other similar choices of neural net produce similar results presented in this section, as is the case with any machine learning problem.

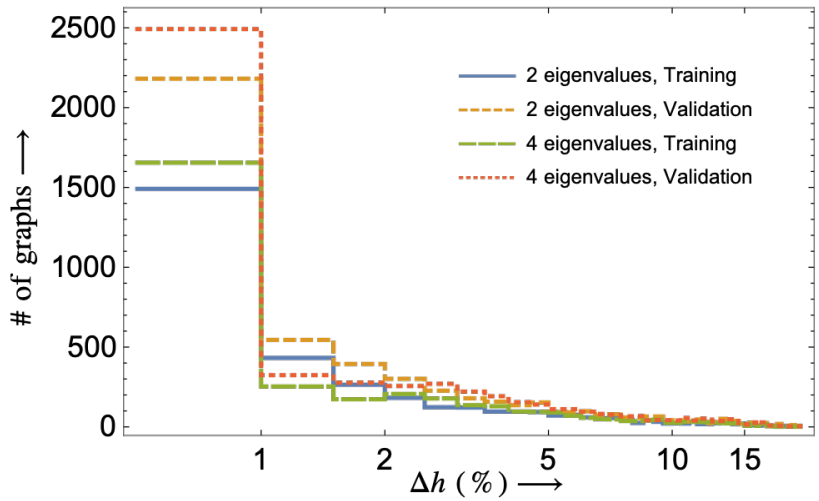


Fig. 10: Training and Validation Histograms for training deep neural network on $n = 12$ graph data.

In figure 10, each bin size corresponds to 1% of Δh. Mean deviation for both training and validation for both cases of 2 and 4 eigenvalues is about 2.5%. We make the following observations.

1. $\lambda_0(G)$ and $\lambda_1(G)$ have a very strong correlation with $h(G)$. Furthermore, there appears to be a small non-linear dependence on $\lambda_0(G)$ and $\lambda_1(G)$ which accounts for about 2.5% improvement over the linear regression. The average deviation Δh is about 2.5% in both the training and validation data sets for deep neural net (DNN) model while it was about 5% for the linear model.

2. We do not observe any significant improvement for the estimation of $h(G)$ while considering top four eigenvalues over $\lambda_0(G)$ and $\lambda_1(G)$. In both cases $\langle \Delta h \rangle \approx 2.5\%$ with small fluctuations in each attempt of training.

3. It is also interesting to observe that while the average deviation $\langle \Delta h \rangle$ is small, the deviation in the entire data never exceeds 25%. The similar exercise done for other graph sizes between $n = 13$ to $n = 20$ has similar results. Mean and standard deviation for Δh for these cases is plotted in Fig. 11 below for both training and validation, reaffirming the observations made above.

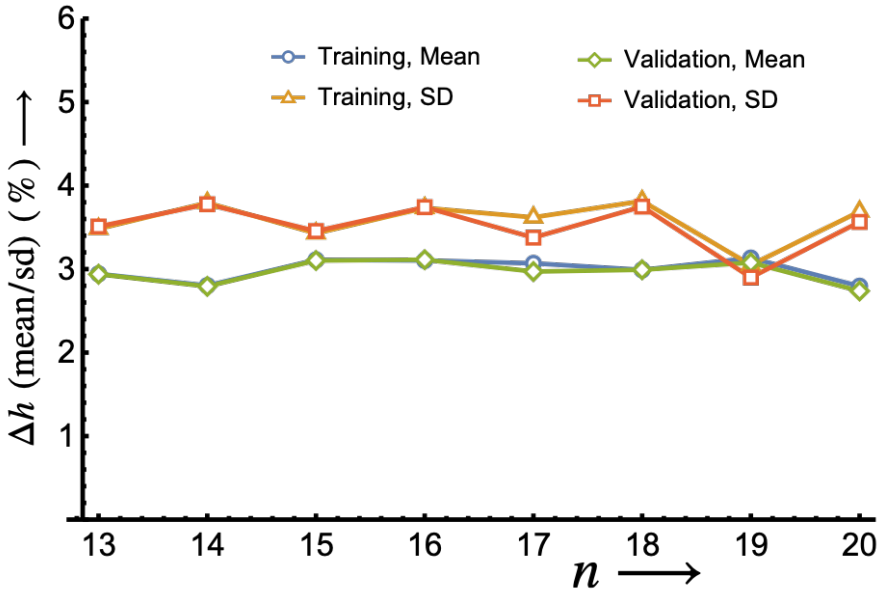


Fig. 11: Training and Validation mean standard deviations for neural network model trained with two eigenvalues for $n = 13$ to 20.

4. Studying the trained deep neural network reveals that $h(G)$ has largely linear dependence on λ_0 and λ_1 when the spectral gap is large, while it exhibits non-linear dependence when the spectral gap is small.

We conclude that $\lambda_0(G)$ and $\lambda_1(G)$ suffice to estimate $hG)$ reliably.

4.5 Predicting $h(G)$ using Neural Networks

The most interesting application of this work is to predict Cheeger constant for large regular graphs, where it is computationally inefficient to calculate Cheeger constant but computationally efficient to calculate the spectrum. To achieve this, we train a neural net for small graphs where it is possible to calculate Cheeger constant in reasonable computation time. We then use this trained net to predict Cheeger constant for the large graph. We moderately train the deep neural network shown in the previous section for 50 epochs[2] on $\lambda_0(G)$ and $\lambda_1(G)$ of the spectrum and Cheeger constant data for graphs of sizes 12 and 16 for even-sized graphs and sizes 13 and 17 for odd-sized graphs. Again for training here we have taken only 40% of the available data. Each training results in a new model, so we train the network for each n a few times then take the trained model that yields the least validation error on the same n. We use the trained nets to predict $h(G)$ for graphs of other sizes which we compare to its true value and obtain Δh. The average deviation $\langle \Delta h \rangle$ with respect to n is shown in Fig. 12 below, where we also show prediction done by linear regression method of Sec. 4.3 for contrast.

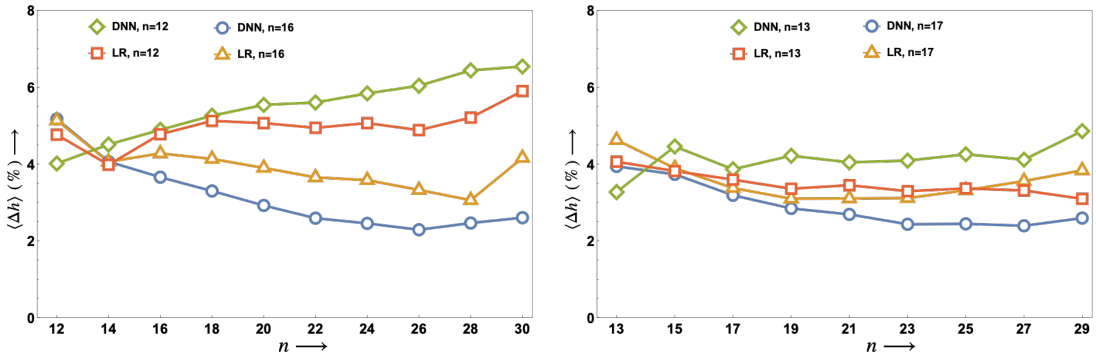


Fig. 12: Comparison of Deep Neural Network with Linear Regression for predicting Cheeger constant.

Left panel of figure 12 shows mean prediction deviation for even n when deep neural network (DNN) models and linear regression (LR) models for $n = 12$ or $n = 16$ are used. Right panel of figure 12 shows mean prediction deviation

[2] The training was stopped after 50 epochs as compared to about 500 epochs (optimization stopping automatically when loss stops improving) done in the previous section. This ensures that the network learns the significance of top two eigenvalues and not the information about the n. Maximal training to about 500 epochs optimizes the network to estimate Cheeger constant for a given n, but is bad for predicting Cheeger constant of other n.

for odd n when deep neural network (DNN) models and linear regression (LR) models for $n = 13$ or $n = 17$ are used. Here are our observations

1. We note that $n = 16$ works better than $n = 12$ for predicting Cheeger constants for higher even n, and similarly $n = 17$ works better than $n = 13$ for predicting Cheeger constant for higher odd n.

2. Although the graphs are not shown here, but we have verified that to predict for even n training on even n works better than training on odd n, and vice versa. This is consistent with observations of Sec. 4.3.

3. We also note that deep neural net based model provides better prediction compared to linear regression model with a consistent improvement as n increases. Particularly, the models trained on $n = 16$ and $n = 17$ data predict Cheeger constants for the graphs of sizes 29 and 30 respectively, to within 3% accuracy on an average.

4. While we observe low average Δh the standard deviation in Δh is also low at about 4% throughout the range of the n, thus guaranteeing reliability on predictions. This is shown in Fig. 13 below.

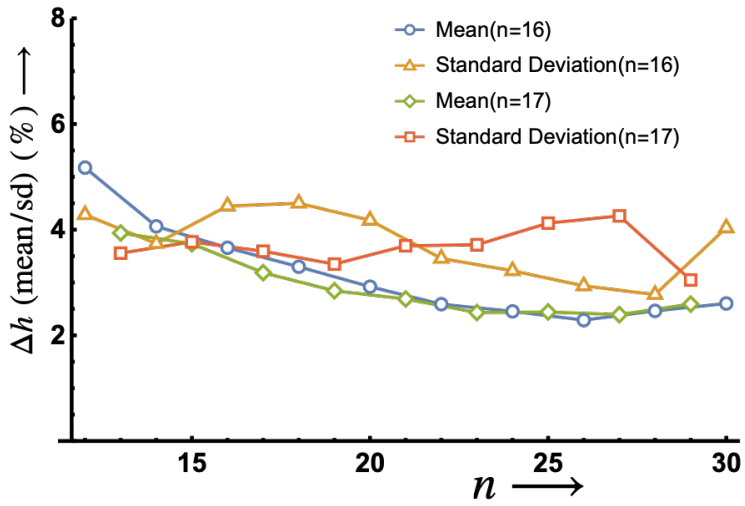


Fig. 13: Prediction Statistics for $n = 16$ and $n = 17$ DNN models. Mean deviation stays between 2% and 4% for all higher n while standard deviation is about 4%.

5. CONCLUSION

In this thesis, we have studied the relevance of the spectrum of a graph G in estimating $h(G)$. We find that $\lambda_0(G)$ and $\lambda_1(G)$ is strongly correlated with $h(G)$, where the correlation is largely linear with a small non-linearity as confirmed by the machine learning analysis. We have also showed that a moderate training makes a deep neural network learn about the relationship between the two largest eigenvalues and $h(G)$ which can then be used to estimate Cheeger constant of a larger graph with high accuracy. This is particularly useful when in a scenario when a choice between large regular graphs is to be made and computational time is limited.

It is very interesting to observe that the quantity $h(G)$, is computationally complex in classical domain, seems to have very simple relation in spectral domain. This can be very helpful in other combinatorial optimization problems too.

Appendices

I Algorithm for Calculating Cheeger Constant

Algorithm 2: Compute Cheeger constant

Input: Graph G
Output: The Cheeger constant of graph G

```
1  Function GetCheeger(G):
2      V ← G.nodes()
3      l ← V.length()
4      Cheeger ← l^2
5      for i ← 1 to ⌊l/2⌋ do
6          foreach j ∈ GetSubset(V,i) do
7              p ←GetBoundaryLength(G,j)
8              s ← p/i
9              if s < Cheeger then
10                 Cheeger ← s
11             end
12         end
13     end
14     return Cheeger
15 End Function
1  Function GetBoundaryLength(G,S):
2      V ← G.nodes()
3      l ← 0
4      foreach i ∈ S do
5          k ← G.neighbors(i)
6          foreach j ∈ k do
7              if j ∈ V \ S then
8                  l ← l + 1
9              end
10         end
11     end
12     return l
13 End Function
```

BIBLIOGRAPHY

[1] Norman Biggs. *Algebraic graph theory*. Cambridge Mathematical Library. Cambridge University Press, Cambridge, second edition, 1993.

[2] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoret. Comput. Sci.*, 1(3):237–267, 1976.

[3] Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.

[4] Bojan Mohar. Isoperimetric numbers of graphs. *J. Combin. Theory Ser. B*, 47(3):274–291, 1989.

[5] Eric Reich. Graph Theory with Applications to Statistical Mechanics. https://web.wpi.edu/Pubs/E-project/Available/E-project-091114-230704/unrestricted/report.pdf, 2014. [Online; accessed 2020].

[6] A. Steger and N. C. Wormald. Generating random regular graphs quickly. volume 8, pages 377–396. 1999. Random graphs and combinatorial structures (Oberwolfach, 1997).

[7] Dragan Stevanovic. Application of graph spectra in quantum physic. http://elib.mi.sanu.ac.rs/files/journals/zr/22/zbr14085.pdf, 2016.