All content in this week's tutorial
(both Sheldon and Xin Xia)
will not in your exam.

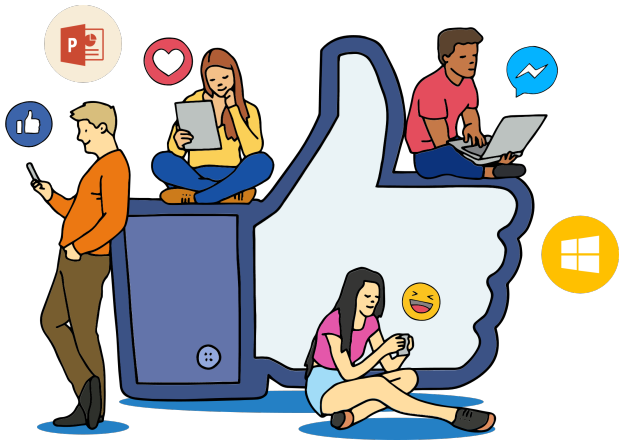
You see this slide because you come here early. Let's wait for other students to come

INFS7450 SOCIAL MEDIA ANALYTICS
Tutorial Week 10

Xiangguo (Sheldon) Sun

School of ITEE

The University of Queensland



Summary of Project 1

Ground truth:

- nx.betweenness_centrality(G)
- nx.pagerank(G)
- All the parameters are set up by default.
- Python 3.6, network 2.3

Node betweenness:
[107, 1684, 3437, 1912, 1085, 0, 698, 567, 58, 428]
Pagerank:
[3437, 107, 1684, 0, 1912, 348, 686, 3980, 414, 698]

- We don't care about the orders.
- Ground truth means ground truth. There's no complete truth in our life. Ground truth means the baseline or the relatively objective criterion. Tt is unique for the given task. No matter whether the python version impacts, there is only one version of ground truth.
- Some students may have different nodes when it comes to 10^{th} and 11^{th}, because they have very similar values. If you generate 9 of the above, you can also get full mark in the result part.

Summary of Project 1

Marking criteria (Total marks: 15):

- Task 1: 8 marks = 3 marks (code) + 3 marks (results) + 2 marks (report)
- Task 2: 7 marks = 2 marks (code) + 3 marks (results) + 2 marks (report)
- Your results should be reproducible and your codes should be readable. If your codes cannot be executed or generate the results as reported, the corresponding marks for the code and results will be deducted.
- We will evaluate your submitted results via calculating the Jaccard Similarity between the submitted results and the ground truth. That means your mark for each task will be calculated by:

  Result Mark = Jaccard Similarity (Submitted Results, Ground Truth) * 3

Some excellent work from our students!

Student: 44749826 Miss Jawaher

PART ONE: results

107 1684 3437 1912 1085 0 698 567 58 428
[107, 1684, 3437, 1912, 1085, 0, 698, 567, 58, 428]

107 3437 0 1684 1912 348 414 3980 686 698
[3437, 107, 1684, 0, 1912, 348, 686, 3980, 414, 698]

The results are consist with our ground truth

Some excellent work from our students!

Student: 44749826 Miss Jawaher

PART TWO: Codes

```
Degree = Nei_graph(D)
# Initialize the pagerank values
initialPagerank = dict.fromkeys(Degree, 1.0 / len(nx.nodes(G)))
# power iteration, iterate until convergence
for iterations in range(max_iter):
    Initial = initialPagerank
    initialPagerank = dict.fromkeys(Initial.keys(), 0)
    for n in initialPagerank:
        for neighbor in Degree[n]:
            # initialPagerank[neighbor] = initialPagerank[neighbor] + alpha * Degree[n][neighbor][weigh
            initialPagerank[neighbor] = initialPagerank[neighbor] + Degree[n][neighbor][weight]
        pagerank_node = beta + alpha * initialPagerank[neighbor] # Updating the pagerank for the node
    # Setting the initial pagerank with the new pagerank for future iterations
    for node in nx.nodes(G):
        initialPagerank[neighbor] = pagerank_node
    # check convergence, l2 norm
    #val = math.sqrt(sum([(initialPagerank[n]**2 + Initial[n]) for n in initialPagerank]))
    # check convergence, l1 norm
```

- The codes are readable and she gave me some annotation so that I can understand her idea easily.
- The source codes are reproductive.

Some excellent work from our students!

Student: 44749826 Miss Jawaher

PART THREE: REPORT

THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

Jawaher A. Al-Ghamdi
ID, 44749826
Queensland University

j.alghamdi@uqconnect.edu.au

Social Media Analytics

INFS7450 : Assignment 1 :
Due Apr 23, 2020

Contents

Task 1

This dataset consists of data that has been anonymized by replacing the Facebook-internal ids for each user with a new value. The data contains of 4039 nodes, 88234 edges in total. In other words,the network consists of 4,039 nodes, connected via 88,234 edges. Each line of the data represents an undirected link starting from one node to another. In task1, we will discuss the **Betweenness Centrality**.

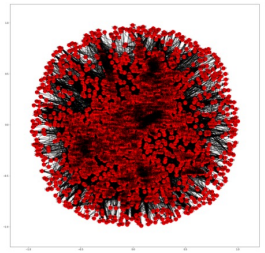


Figure 1: Network Visualization

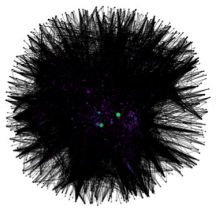


Figure 2: Network Visualization

In the graph theory, betweenness centrality is one a of the centrality measures in a graph based on shortest paths. For every pair of vertices in a connected graph, there exists at least one shortest path between the vertices such that either the number of edges that the path passes through (for unweighted graphs) or the sum of the weights of the edges (for weighted graphs) is minimized. In other words, The betweenness centrality for each vertex is the number of these shortest paths that pass through the vertex. Figure 2 shows the important nodes by the size of the node in terms of betweenness centrality But how did we implement it?

Task 1 [] continued on next page...

Some excellent work from our students!

Student: 44749826 Miss Jawaher

PART THREE: REPORT

Answer

Betweenness centrality of the node is defined as the number of shortest paths that go through the node v. The following formula is used to compute the betweenness centrality of the node.
$C_b(vi) = \Sigma_{(s)\neq(v)\neq(t)} \frac{\sigma st(vi)}{\sigma st}$
where $\sigma st(vi)$ refers to the number of shortest paths from s to t pass through vi. and σst indicates the number of shortest paths from vertex s to t w/o passing through vi. [1]
Indeed, we want to calculate the betweenness centrality of the node efficiently and to do so we will implement what is so-called **Brandes algorithm** by utilizing the following formula
$g(v) = \Sigma_{(s)\neq(v)\neq(t)} \frac{\sigma st(v)}{\sigma st} = \delta_s(vi) = \Sigma_{w:vi\in pred(s,w)} \frac{\sigma_s(vi)}{\sigma_s w}(1+\delta_s(w))$. [1]

where $\delta_s(vi)$ indicates the dependence of s to vi and $\frac{\sigma_s(vi)}{\sigma_s w}$ refers to the shortest path from s to vi divided by the shortest path from s to w. Recalling (from the lecture) that we can calculate the betweenness centrality using many ways but what we need to do is to utilize an efficient way to implement our method and that is what indeed inspired me about using **Brandes algorithm**. There are two steps in Brandes Algorithm which are *forward* and *backward* step. We calculate the shortest paths from the starting node to all other nodes in the graph in the forward step where we apply BFS algorithm, and , we compute the dependence between the starting node s to vi in the backward step where we need to make sure that the nodes are visited in the correct order on the backward step and to do so, in our code, we have created a stack and a queue , we dequeue the elements from the queue and push them to the stack in the forward step and then visiting the nodes in the order they are popped off the stack in the backward pass. we firstly use a dictionary in order to return the Betweenness Centrality for the nodes. The parameters we used in the code are:

- The dependence of s to the last node (last node visited in BFS) is always zero.
- pred(s,w) is the set of predecessors of w in the shortest paths from s to w.
- vi is one of parents nodes of w.
- if w is not vi's child then it can be ignored.

Implemented steps are:

- For every node v in V, set$C_B(w) = 0$.
- For each node s in V :
  - set $\sigma(v)$ to zero for all nodes v in V.
  - Use the BFS algorithm
  - while Q is not empty, do:
    - dequeue v from Q and push v onto a stack S
    - For each node w such that is an edge in E from v to w,do:
      - if dist[w] is infinity, then
      - set dist[w] to dist[v] + 1
      - enqueue w
      - if dist[w] = dist[v]+1 then
      - set $\sigma(s,w)$ to $\sigma(s,w)+\sigma(s,v)$
      - append v to Pred[w]
  - while S is not empty,pop w off S
    - for all nodes v in Pred(w) set $\sigma(v)$ to $\sigma(v) + \frac{\sigma_s(vi)}{\sigma_s w}(1+\delta_s(w)$.
    - unless $w = s$, set $C_B(w) = C_B(w) + \sigma(w)$.
- $C_B(w)$ gives the final result

Task 2

To overcome the shortages of Katz centrality, pagerank centrality measure were proposed. The problem of Katz index is that in directed graphs, once a node becomes an authority(high centrality), it passes all its centrality along all of its out-links and that is less desirable since not everyone known by well-known person is well-known. Therefore, pagerank, as aforementioned above, were proposed to fill this gap. In pagerank, the value of passed centrality is divided by the number of out-degree of that node, that is, each connected neighbor gets a fraction of the source node's centrality. In task2, we will discuss the **Pagerank Centrality**. But how did we implement pagerank algorithm ?

Answer

Page rank
We used the power iteration of the following formula in order to calculate the page rank method
$C_p(vi) = \alpha \sum_{j=1}^{n} A(j,i) \frac{C_p(vi)}{d_j^{out}} + \beta$
Similar to the betweenness centrality, there are many ways to calculate the pagerank of the nodes but some of them are time-consuming. We use the following power iteration to implement page rank method on **directed graph** as the following steps:

- Initializing all nodes to be 1 divided by the number of nodes
- Start iterating according to the max-iter which is limited to 100
- Apply the formula $C_p(vi) = \alpha \sum_{j=1}^{n} A(j,i) \frac{C_p(vi)}{d_j^{out}} + \beta$ iteratively

We use the following power iteration to implement page rank method on **undirected graph** as the following steps:

- Initializing all nodes to be 1 divided by the number of nodes
- Set $C^0 = 1$, K =1
- Start iterating according to the max-iter which is limited to 100
- $C^(k) = \alpha A^T D^{-}1 + \beta 1$
- If $||C^(k) - C^(k-1)||_1 > \epsilon$
- $k = k + 1$, goto 1

We normalize using L1 norm rather than L2 norm.
Similar to the Katz centrality but the difference is that in katz we don't really consider the degree of the node contrary to the pagerank which requires that as the value of passed centrality is divided by the number of out-degree of the node. In this project, page rank was implemented for both directed and an **undirected graph**. We first of all, initialize the pagerank for each node to be 1 divided by the total number of nodes (e.g. if we have 4 nodes then the initial score for each node will be 1/4). Then, we start with any node and see its degree(i.e.in-links,neighbors) and then we look at how many out-links from this neighbors. If, for instance, we have node A which has 1 in-link from B, then we need to look at B and calculate how many out-links from B. Then, we divided the pagerank(B) by the number of its out-links, that is the pagerank of A in this case.We do this iteratively for all nodes in the graph. We set max-iter to 100. Each time we compare the current value with the previous value, if the value is greater than the ϵ then we continue the iteration, otherwise we stop.

	Top 10 nodes									
Betweenness Centrality	107	1648	3437	1912	1085	0	698	567	58	428
PageRank Centrality	3437	107	1684	0	1912	686	3980	348	414	483

Table 1: The top 10 nodes in both centrality measures.

Some excellent work from our students!

Student: 44749826 Miss Jawaher

PART THREE: REPORT

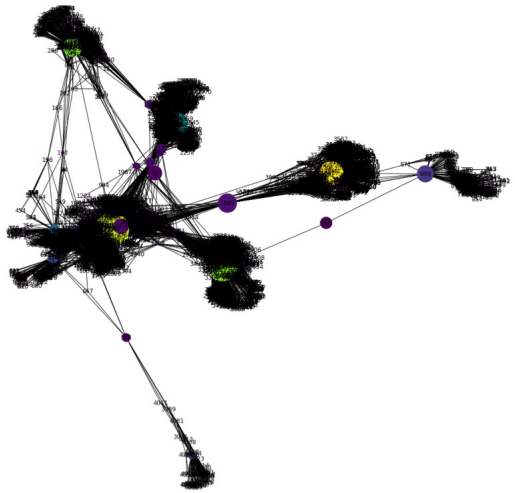


Figure 3: Nodes Visualization

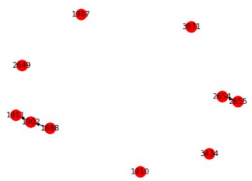


Figure 4: Pagerank centrality on Digrapg

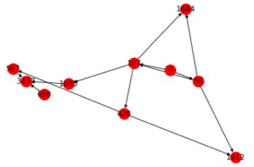


Figure 5: Beteenness centrality

Summary

`Centrality` measures are one of the most important requirements in social media analytics. We have computed the **betweenness centrality** and the **page rank centrality** for the dataset and we have generated the top 10 nodes in both measures. surprisingly, some nodes are having higher centrality in both measures such as node 3437 and 107 ..etc. In figure3, we try to visualize the most important nodes that has high betweenness centrality and pagerank by the size and the color of the nodes respectively. In the table, it can be observed that there are some nodes have highest *betweenness centrality* and *pagerank centrality*. node 107 is the most important node in terms of *betweenness centrality* as it is linked by a node which is in turn linked by other important node. By looking at figure 5, node 107 and 1912 are very important since when we remove them, the graph will be disconnected. By looking at the page rank centralises, it can be seen the node 3437 has the highest *pagerank centrality* and also one of the top 5 nodes with highest *betweenness centrality*. Table 1 shows the top 10 nodes by both *betweenness centrality* and *pagerank centrality*.

Some excellent work from our students!

Student: 44749826 Miss Jawaher

Any shortcoming?

- The report didn't follow our template. However, she supplied with another version of report using our template. Besides, I don't think this report mislead my understanding. Actually the student's own format is more professional than our template. So it doesn't reduce my good impression.
- The codes are based on *Functional Programming*. Although we prefer to *OOP (object-oriented programming),* but our project is small-scale and relatively simple, which means functional programming is also a good choice. So it doesn't reduce my good impression.

Highlight?

- The report gave me deep impression. She did a lot of work and she also draw the graph and her results.
- Her code is independent with existing work and readable.
- The results are good enough.

In summary, I'd like to give full mark of her project work

Some excellent work from our students!

Student: 45209167 Mr. Haifeng Hu

PART ONE: results

107 1684 3437 1912 1085 0 698 567 58 428
3437 107 1684 0 1912 348 686 3980 414 483

[107, 1684, 3437, 1912, 1085, 0, 698, 567, 58, 428]
[3437, 107, 1684, 0, 1912, 348, 686, 3980, 414, 698]

The results are consist with our ground truth
Although there exist only one different, but as we
previously said, he can also get full mark in this part

Some excellent work from our students!

Student: 45209167 Mr. Haifeng Hu

PART TWO: Codes

- The codes are highly independent of other third part libraries.
- They are based on our previous tutorial (graph_me.py), and they didn't use any valid function from networkX!
- The source codes are reproductive, OOP, and readable.

```
class MyGraph:
    def __init__(self, edges=None, weighted=True):...

    def to_undirected(self):...

    def weight(self, u_name, v_name):
        return self.DiAdj[self.name2index[u_name], self.name2index[v_name]]

    def has_node(self, n):
        return n in self.node_names

    def has_edge(self, u, v):
        return v in self.DiGraph[u]

    def graph_transpose(self):...

    def degree(self, u=None, mode='out'):...
```

```python
class MyGraph:
    def __init__(self, edges=None, weighted=True):...

    def to_undirected(self):...

    def weight(self, u_name, v_name):
        return self.DiAdj[self.name2index[u_name], self.name2index[v_name]]

    def has_node(self, n):
        return n in self.node_names

    def has_edge(self, u, v):
        return v in self.DiGraph[u]

    def graph_transpose(self):...

    def degree(self, u=None, mode='out'):...

    def traverse(self, mode='bfs', src='s'):...

    def extract_min(self, Q, dis):...

    def prim(self, r):...

    def Dijkstra(self, r):...

    def out_degree_centrality(self):...

    def closeness_centrality(self, u=None):...
```

```
    def node_betweenness_centralities(self):
        # initialize centralities for each node
        centralities = {}
        for name in self.node_names:
            centralities[name] = 0

        for s in self.node_names:
            # initialization
            color = defaultdict(str)  # visited or not
            dis = defaultdict(float)  # distance from s
            pi = defaultdict(str)  # predecessor key<->value, value is the predecessor of the key
            sigma = defaultdict(int)  # count of shortest path
            for name in self.node_names:
                color[name] = 'white'
                dis[name] = math.inf  # math.inf
                pi[name] = []
                sigma[name] = 0

            # forward step, calculate sigma
            sigma[s] = 1
            color[s] = 'gray'
            dis[s] = 0
            Q = [s]
            T = []
            order_list = [s]
            while len(Q) > 0:
                # pop the first item from the queue
                u = Q.pop(0)
                for v in self.DiGraph[u]:
                    # not visited
```

```
            for v in self.DiGraph[u]:
                # not visited
                if color[v] == 'white':
                    order_list.append(v)
                    color[v] = 'gray'
                    dis[v] = dis[u] + 1
                    Q.append(v)
                    T.append((u, v))
                if dis[v] == dis[u] + 1:
                    sigma[v] += sigma[u]
                    pi[v].append(u)
            color[u] = 'black'

        #backward step
        delta = {}
        # initial delta value is all 0, the last node in the BFS search is 0
        # no shortest path from starting node s that passes through the last node
        for name in self.node_names:
            delta[name] = 0.0
        while order_list:
            #pop the last
            w = order_list.pop()
            for v in pi[w]:
                delta[v] = delta[v] + (sigma[v] / sigma[w]) * (1 + delta[w])
            print(centralities[w])
            if w != s:
                #aggregate the delta value on the route
                centralities[w] = centralities[w] + delta[w]

    # it is a undirected graph, therefore should be a 0.5 scale
    for i in centralities:
```

```
        # it is a undirected graph, therefore should be a 0.5 scale
        for i in centralities:
            centralities[i] = centralities[i] * 0.5
            return centralities

        return centralities

    def page_rank(self, alpha, beta, epsilon):
        # rank dictionary for each node's page rank centrality value
        rank = {}
        # initialize the value = 1 for each node
        for node in self.node_names:
            rank[node] = 1
        while True:
            current_epsilon = 0
            current_rank = {}
            for src in self.node_names:
                sum = 0
                # calculate each node's page rank for the current loop
                for tar in self.DiGraph[src]:
                    sum = sum + rank[tar]/len(self.DiGraph[tar])
                current_rank[src] = sum*alpha + beta
                current_epsilon = current_epsilon + np.square(current_rank[src] - rank[src])
            # calculate the norm of rank difference between two loop runs
            current_epsilon = np.sqrt(current_epsilon)
            # check exit condition is met or not
            if current_epsilon < epsilon:
                break
            else:
                rank = current_rank
```

Some excellent work from our students!

Student: 45209167 Mr. Haifeng Hu

PART THREE: REPORT

Report (Project One)

Student Name: Haifeng Hu
Student ID: 45209167
Student Email: haifeng.hu@uqconnect.edu.au

I've created two methods to calculate the betweenness centrality, and page rank centrality for nodes, respectively. The two methods lies in the MyGraph class, which is inherited from the Graph class provided by the tutor.
The main program takes the graph file name in the command line argument and read the provided graph text file, and then the graph is initialized by the existing functions from MyGraph class. The betweenness centrality and page rank centrality methods are called and the top 10 nodes are returned and written into the output file.

Task 1

I am using Brandes Algorithm to the betweenness centrality. The algorithm consists of two steps.

- The first step is the forward step, where I enhances the BFS search in order to calculate the sigma value of each node for a given start node s. sigma value is the number of shortest paths from starting node s to a given node v.
- The second step is the backward step, where from the last node of the BFS search, the delta value for node v_i is calculated based on the following formula

$$\delta_s(v_i) = \sum_{w:\ v_i \in pred(s,w)} \frac{\sigma_{sv_i}}{\sigma_{sw}}(1 + \delta_s(w))$$

  - σ_{sv_i} is the number of shortest paths from starting node s to node v_i
  - σ_{sw} is the number of shortest paths from starting node s to node w
  - Node v_i is the predecessors of node w

  It is worth to mention that the last node of the BFS search route from node s has the delta value 0 because there is no shortest path from starting node s that passes through the last node.

The centrality value of each node is aggregated based on the following formula

$$C_b(v_i) = \sum_{s \neq t \neq v_i} \frac{\sigma_{st}(v_i)}{\sigma_{st}} = \sum_{s \neq v_i} \delta_s(v_i)$$

$$\delta_s(v_i) = \sum_{t \neq v_i} \frac{\sigma_{st}(v_i)}{\sigma_{st}}$$

The dependence of s to v_i

Each node in the graph needs to go through the steps above as the starting node s.

Result: Top 10 nodes
107 1684 3437 1912 1085 0 698 567 58 428

Task 2

I am using power iteration to calculate the page rank centrality for nodes.
Each node's centrality is initialized as 1, for each loop, calculate each node's centrality based on the previous loop's centrality value. The exit condition is when the centrality values converges, this is checked by the difference of two loop's centrality values' norm is less than a given epsilon. The algorithm is as follows:

1. Set $c^{(0)} = 1, k = 1$
2. $c^{(k)} = \alpha A^T D^{-1} c^{(k-1)} + \beta$
3. If $\| c^{(k)} - c^{(k-1)} \| \geq \varepsilon$
   k = k + 1, go to 1

In the algorithm, A is the adjacency matrix of the graph, D is the diagonal matrix with the outdegrees in the diagonal.
The parameter α, β and ε is provided from input, $\alpha = 0.85, \beta = 0.15, \varepsilon = 0.001$.

Result: Top 10 nodes
3437 107 1684 0 1912 348 686 3980 414 483

Summary

For the task 1, use small dataset to test whether the logic works makes much sense. By creating a small graph like the lecture notes and check the result is correct helps a lot.

Reference

Some excellent work from our students!

Student: 45209167 Mr. Haifeng Hu

Any shortcoming?

- The result of task 2 has only one different node in 10th position, but I don't think this should reduce my impression of this work. And according to our previous said, we will give the full remark.

Highlight?

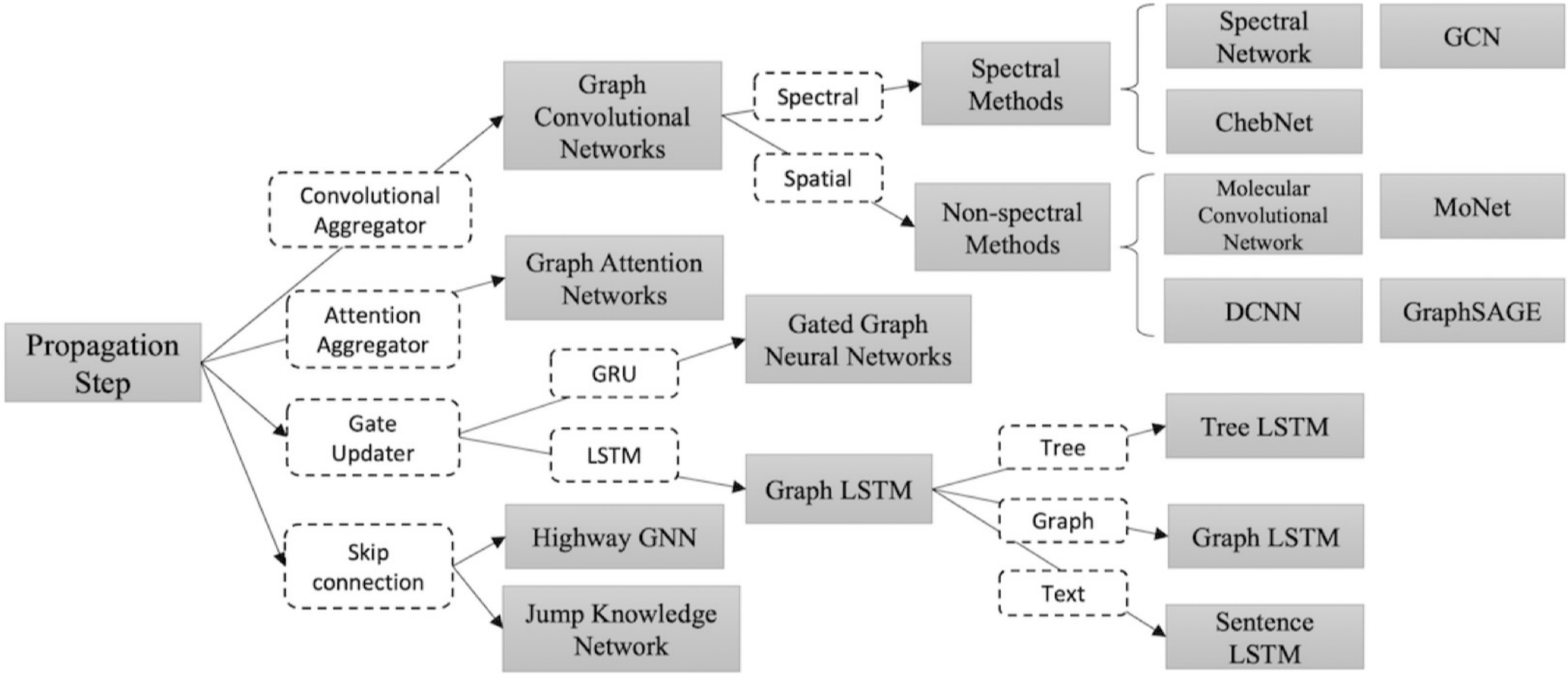
- The project source code gave me deep impression.
- The report is brief but precise.
- The results are good enough.

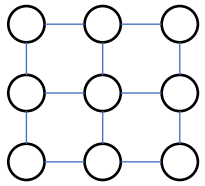
In summary, I'd like to give full mark of his project work

Graph Convolution Networks: A Spectral Method

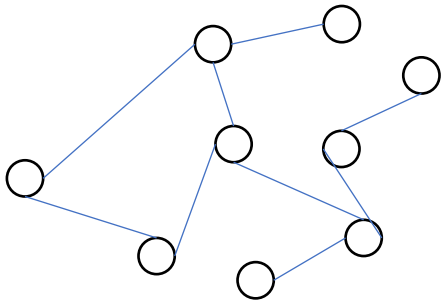
Xiangguo Sun

sunxiangguodut@qq.com


Propagation Step
Convolutional Aggregator
Attention Aggregator
Gate Updater
Skip connection
Graph Convolutional Networks
Graph Attention Networks
GRU
LSTM
Highway GNN
Jump Knowledge Network
Gated Graph Neural Networks
Graph LSTM
Spectral
Spatial
Spectral Methods
Non-spectral Methods
Tree
Graph
Text
Spectral Network
ChebNet
Molecular Convolutional Network
DCNN
Tree LSTM
Graph LSTM
Sentence LSTM
GCN
MoNet
GraphSAGE



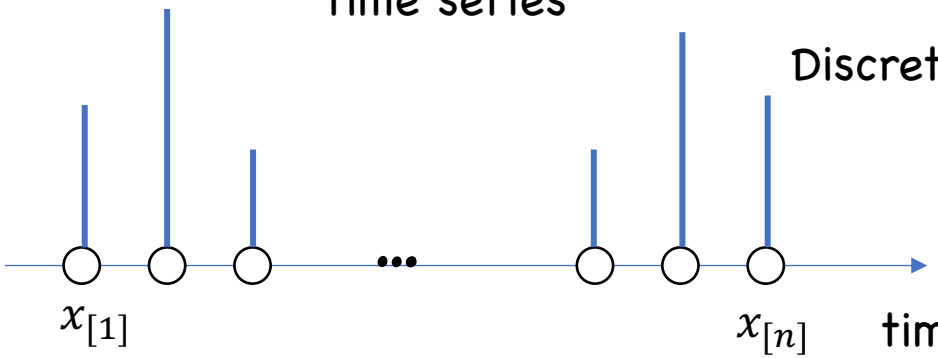
CNN



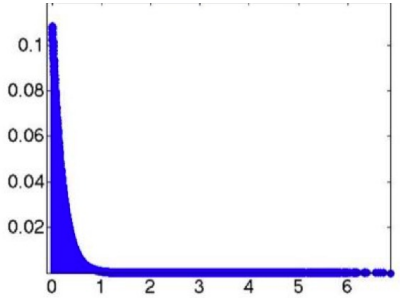
How about this?

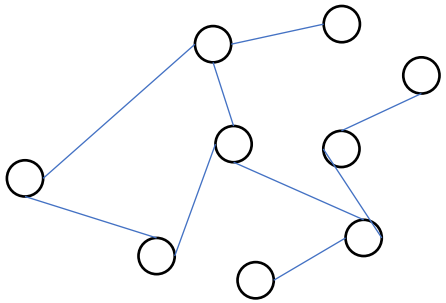
Graph Convolution Networks in Spatial

Time series



Discrete Fourier Transform



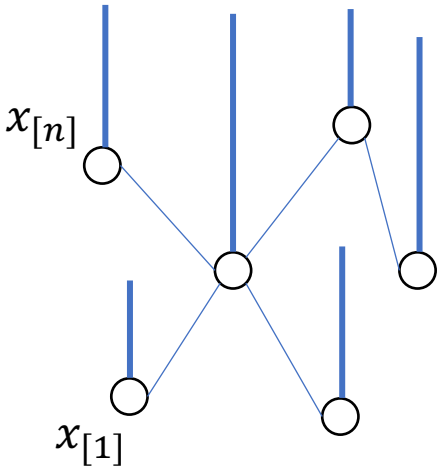


How about this?

Graph Convolution Networks in Spatial

Spatial domain

Spatial domain



How about this?

Graph Convolution Networks in Spectrum

Spatial domain

Spectral domain

Graph Convolution Networks in Spectrum

A. Prerequisite Knowledge

B. Fourier Transform

C. Graph Laplacian and its Properties

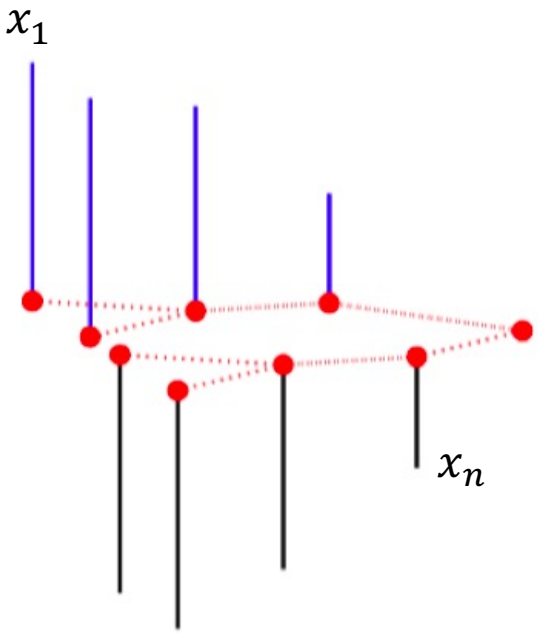
D. Graph Fourier Transform

E. Spectral Graph Convolution

F. Convolution Kernel

Prerequisite Knowledge

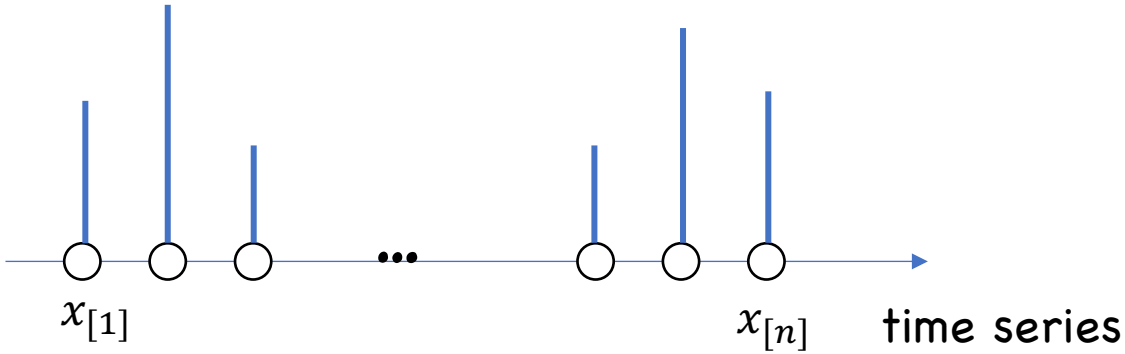
Graph signal



$\hat{x} = (x_1, x_2, \cdots, x_n)$

Not sequence

Time series



$\hat{x} = (x_1, x_2, \cdots, x_n)$

sequence

- consider a real $n \times n$ matrix A, i.e., $A \in \mathbb{R}^{n \times n}$
- $\lambda \in \mathbb{C}$ is an eigenvalue of A

  if there exists $\mathbf{x} \in \mathbb{C}^n$, $\mathbf{x} \neq \mathbf{0}$

  such that

$$A\mathbf{x} = \lambda\mathbf{x}$$

- such a vector $\mathbf{x}$ is called eigenvector of λ
- alternatively,

$$(A - \lambda I)\mathbf{x} = \mathbf{0} \quad \text{or} \quad \det(A - \lambda I) = 0$$

- it follows that A has n eigenvalues

  (possibly complex and possibly with multiplicity > 1)

- consider a real and symmetric $n \times n$ matrix A

  (e.g., the adjacency matrix of an undirected graph)
- then
  - all eigenvalues of A are real
  - eigenvectors of different eigenvalues are orthogonal

    i.e., if $\mathbf{x}_1$ an eigenvector of λ_1

    and $\mathbf{x}_2$ an eigenvector of λ_2

    then $\lambda_1 \neq \lambda_2$ implies $\mathbf{x}_1 \perp \mathbf{x}_2$ (or $\mathbf{x}_1^T \mathbf{x}_2 = 0$)

- A is positive semi-definite if $\mathbf{x}^T A \mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbb{R}^n$
- a symmetric positive semi-definite real matrix has

  real and non negative eigenvalues

Fourier Transform

$$F(\lambda) = \int_{-\infty}^{\infty} f(t)\, e^{-(i\lambda t)} dt$$

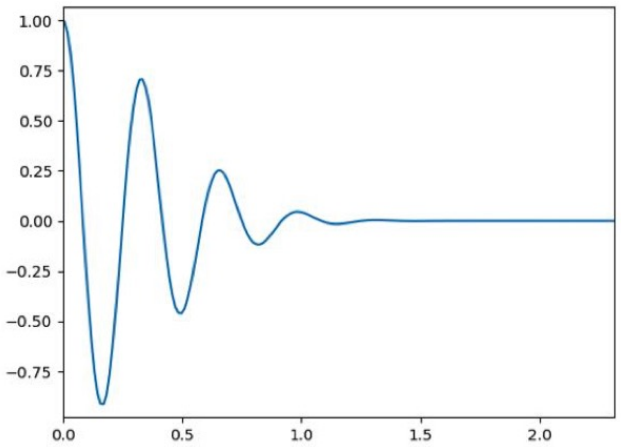
Fourier Transform

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} F(\lambda) e^{(i\lambda x)} d\lambda$$

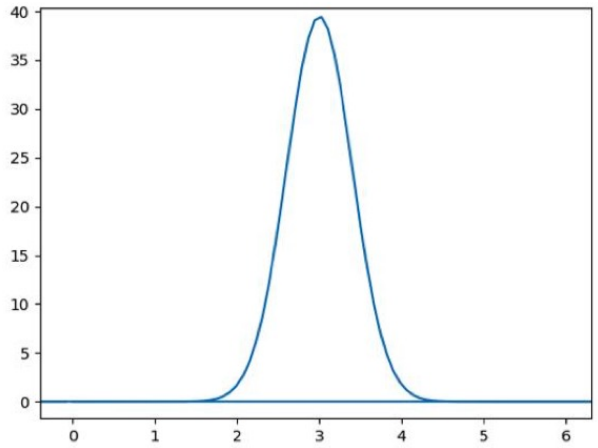
Inverse Fourier Transform

We will adopt the notation

$$\mathcal{F}[f] = F \qquad \mathcal{F}^{-1}[F] = f$$



$$f(x) = \cos(2\pi(3x))e^{-\pi x^2}$$



$$\mathcal{F}[f]$$

Discrete Fourier Transform

DFT transforms a sequence of N complex numbers $\mathbf{x} = (x_0, x_1, \cdots, x_{N-1})$ into another sequence of complex numbers, $X(k) = (X_0, X_1, \cdots, X_{N-1})$.

Let:

$$\mathbf{u_k} = (e^{-\frac{i2\pi}{N}k0}, e^{-\frac{i2\pi}{N}k1}, \cdots, e^{-\frac{i2\pi}{N}kj}, \cdots, e^{-\frac{i2\pi}{N}k(N-1)})$$

where $\mathbf{u}_k(j) = e^{-\frac{i2\pi}{N}kj}$, then

$$X(k) = \sum_{j=0}^{N-1} x_j e^{-\frac{i2\pi}{N}kj} = \sum_{j=0}^{N-1} x_j \mathbf{u}_k(j) = < \mathbf{u_k}, \mathbf{x} >$$

Fourier basis

$$F(\lambda) = \int_{-\infty}^{\infty} f(t) e^{-(i\lambda t)} dt$$

Fourier Transform

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} F(\lambda) e^{(i\lambda x)} d\lambda$$

Inverse Fourier Transform

We will adopt the notation

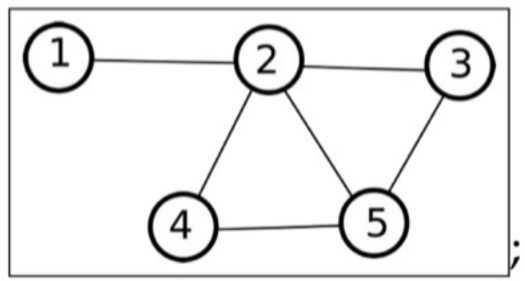
$$\mathcal{F}[f] = F \qquad \mathcal{F}^{-1}[F] = f$$

Graph Laplacian and its Properties

$$\mathbf{L} = \mathbf{D} - \mathbf{A}$$

A is the adjacency matrix and **D** is a diagonal matrix with

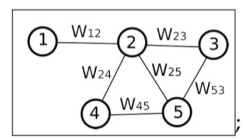
$$d_{ii} = \sum_j a_{ij}$$



;

$$\underbrace{\begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 4 & -1 & -1 & -1 \\ 0 & -1 & 2 & 0 & -1 \\ 0 & -1 & 0 & 2 & -1 \\ 0 & -1 & -1 & -1 & 3 \end{bmatrix}}_{\mathbf{L}} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix}}_{\mathbf{D}} - \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}}_{\mathbf{A}}$$

If the graph has weights w_{ij} associated to the edges, then the graph Laplacian can incorporate such weights.



;

$$\mathbf{L} = \mathbf{D} - \mathbf{W}$$

$$\begin{bmatrix} w_{12} & -w_{12} & 0 & 0 & 0 \\ -w_{21} & \sum_{j\neq 2} w_{2j} & -w_{23} & -w_{24} & -w_{25} \\ 0 & -w_{32} & \sum_{j\neq 3} w_{3j} & 0 & -w_{35} \\ 0 & -w_{42} & 0 & \sum_{j\neq 4} w_{4j} & -w_{45} \\ 0 & -w_{52} & -w_{53} & -w_{54} & \sum_{j\neq 5} w_{5j} \end{bmatrix}$$

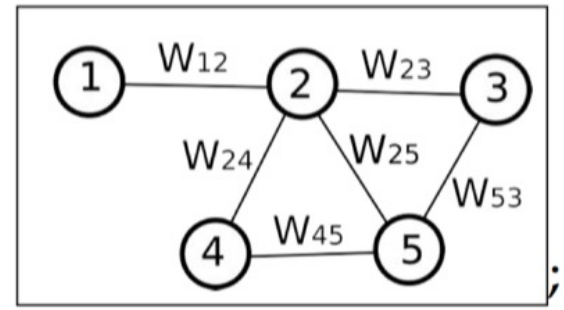
One can also defined normalized versions of the adjacency and Laplacian matrices. There are two normalized versions of the adjacency (Laplacian) matrix that are typically used in the literature—the symmetric normalized adjacency (Laplacian) matrix, $\mathbf{A}^{\mathsf{S}}(\mathbf{L}^{\mathsf{S}})$ and the random walk normalized adjacency (Laplacian) matrix $\mathbf{A}^{\mathsf{RW}}(\mathbf{L}^{\mathsf{RW}})$.These are defined as follows,

$$\mathbf{A}^{\mathsf{S}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}, \quad (2.3)$$
$$\mathbf{A}^{\mathsf{RW}} = \mathbf{D}^{-1}\mathbf{A}, \quad (2.4)$$
$$\mathbf{L}^{\mathsf{S}} = \mathbf{I}_N - \mathbf{A}^{\mathsf{S}}, \quad (2.5)$$
$$\mathbf{L}^{\mathsf{RW}} = \mathbf{I}_N - \mathbf{A}^{\mathsf{RW}}, \quad (2.6)$$

where $\mathbf{I}_N$ is the identity matrix of size N. For the rest of the thesis, we will only work with the non-normalized adjacency and Laplacian matrices unless stated otherwise.

Graph Laplacian and its Properties

If the graph has weights w_{ij} associated to the edges, then the graph Laplacian can incorporate such weights.

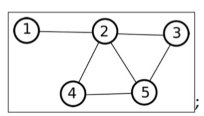


$$\mathbf{L} = \mathbf{D} - \mathbf{W}$$

$$\begin{bmatrix} w_{12} & -w_{12} & 0 & 0 & 0 \\ -w_{21} & \sum_{j\neq 2} w_{2j} & -w_{23} & -w_{24} & -w_{25} \\ 0 & -w_{32} & \sum_{j\neq 3} w_{3j} & 0 & -w_{35} \\ 0 & -w_{42} & 0 & \sum_{j\neq 4} w_{4j} & -w_{45} \\ 0 & -w_{52} & -w_{53} & -w_{54} & \sum_{j\neq 5} w_{5j} \end{bmatrix}$$

$$\mathbf{L} = \mathbf{D} - \mathbf{A}$$

A is the adjacency matrix and **D** is a diagonal matrix with $d_{ii} = \sum_j a_{ij}$



$$\underbrace{\begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 4 & -1 & -1 & -1 \\ 0 & -1 & 2 & 0 & -1 \\ 0 & -1 & 0 & 2 & -1 \\ 0 & -1 & -1 & -1 & 3 \end{bmatrix}}_{\mathbf{L}} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix}}_{\mathbf{D}} - \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}}_{\mathbf{A}}$$

One can also defined normalized versions of the adjacency and Laplacian matrices. There are two normalized versions of the adjacency (Laplacian) matrix that are typically used in the literature—the symmetric normalized adjacency (Laplacian) matrix, $\mathbf{A}^{\mathsf{S}}(\mathbf{L}^{\mathsf{S}})$ and the random walk normalized adjacency (Laplacian) matrix $\mathbf{A}^{\mathsf{RW}}(\mathbf{L}^{\mathsf{RW}})$.These are defined as follows,

$$\mathbf{A}^{\mathsf{S}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}, \quad (2.3)$$
$$\mathbf{A}^{\mathsf{RW}} = \mathbf{D}^{-1}\mathbf{A}, \quad (2.4)$$
$$\mathbf{L}^{\mathsf{S}} = \mathbf{I}_N - \mathbf{A}^{\mathsf{S}}, \quad (2.5)$$
$$\mathbf{L}^{\mathsf{RW}} = \mathbf{I}_N - \mathbf{A}^{\mathsf{RW}}, \quad (2.6)$$

where $\mathbf{I}_N$ is the identity matrix of size N. For the rest of the thesis, we will only work with the non-normalized adjacency and Laplacian matrices unless stated otherwise.

Graph Laplacian and its Properties

One can also defined normalized versions of the adjacency and Laplacian matrices. There are two normalized versions of the adjacency (Laplacian) matrix that are typically used in the literature— the symmetric normalized adjacency (Laplacian) matrix, $\mathbf{A}^{\mathsf{S}}(\mathbf{L}^{\mathsf{S}})$ and the random walk normalized adjacency (Laplacian) matrix $\mathbf{A}^{\mathsf{RW}}(\mathbf{L}^{\mathsf{RW}})$.These are defined as follows,

$$\mathbf{A}^{\mathsf{S}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}, \tag{2.3}$$

$$\mathbf{A}^{\mathsf{RW}} = \mathbf{D}^{-1}\mathbf{A}, \tag{2.4}$$

$$\mathbf{L}^{\mathsf{S}} = \mathbf{I}_N - \mathbf{A}^{\mathsf{S}}, \tag{2.5}$$

$$\mathbf{L}^{\mathsf{RW}} = \mathbf{I}_N - \mathbf{A}^{\mathsf{RW}}, \tag{2.6}$$

where $\mathbf{I}_N$ is the identity matrix of size N. For the rest of the thesis, we will only work with the non-normalized adjacency and Laplacian matrices unless stated otherwise.

Graph Laplacian and its Properties

1. Laplacian Matrix $\mathbf{L}$ is a symmetric matrix, which means is has real eigenvalues and eigenvectors, forming an orthogonal basis.
2. Laplacian Matrix $\mathbf{L}$ is positive semidefinite (assuming non-negative edge weights), which means all eigenvaluses are non-negative, and the following equations holds

$$\mathbf{x}^T\mathbf{L}\mathbf{x} = \sum_{(i,j)\in E} \omega_{ij}(x_i - x_j)^2 \geq 0$$

3. The smallest eigenvalue of $\mathbf{L}$ is 0, the corresponding eigenvectors is the constant one vector $\mathbf{1}$.
4. $\mathbf{L}$ has n non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq, \cdots, \leq \lambda_n$.
5. The multiplicity of the eigenvalues zero of the Laplacian matrix $\mathbf{L}$, is the number of connected components in the graph G.
6. It can be shown that $(\mathbf{L}^k)_{ij} = 0$ when the shortest distance between nodes i and j is greater than k.

Graph Laplacian and its Properties

Laplacian matrix has another name Laplacian Operator, denoted by Δ.

Laplacian matrix $\mathbf{L}$ can be viewed as an operator on the space of functions $\mathbf{x} : V(G) \to \mathbb{R}$ which satisfies

$$\mathbf{Lx}(u) = \sum_{v=1}^{n} \omega_{uv}(x_u - x_v)$$

The output of this operation essentially retains weighted differences of signal values on each node with respect to their neighboring nodes.

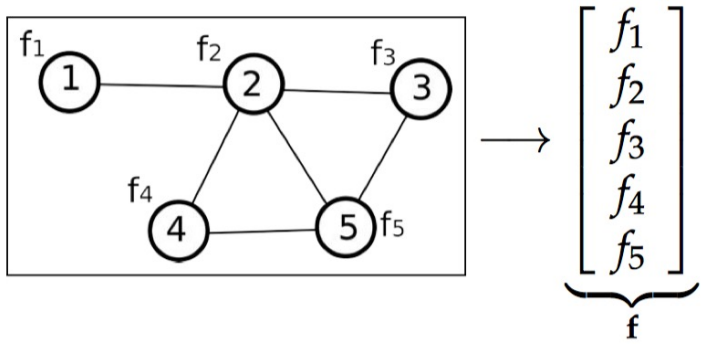
Graph Laplacian and its Properties

$$
\begin{aligned}
\mathbf{Lx} &= \mathbf{Dx} - \mathbf{Wx} \\
&= \begin{pmatrix} d_1 & & \\ & \ddots & \\ & & d_n \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} - \begin{pmatrix} \omega_{11} & \cdots & \omega_{1n} \\ & \cdots & \\ \omega_{n1} & \cdots & \omega_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \\
&= \begin{pmatrix} d_1 x_1 \\ \vdots \\ d_n x_n \end{pmatrix} - \begin{pmatrix} \omega_{11} x_1 + \omega_{12} x_2 + \cdots + \omega_{1n} x_n \\ \vdots \\ \omega_{n1} x_1 + \omega_{n2} x_2 + \cdots + \omega_{nn} x_n \end{pmatrix} = \begin{pmatrix} d_1 x_1 - \sum_{i=1}^{n} \omega_{1i} x_i \\ \vdots \\ d_u x_u - \sum_{i=1}^{n} \omega_{ui} x_i \\ \vdots \\ d_n x_n - \sum_{i=1}^{n} \omega_{ni} x_i \end{pmatrix}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{Lx}(u) &= d_u x_u - \sum_{i=1}^{n} \omega_{ui} x_i \\
&= \sum_{v=1}^{n} \omega_{uv} x_u - \sum_{i=1}^{n} \omega_{ui} x_i \\
&= \sum_{v=1}^{n} \omega_{uv} (x_u - x_v)
\end{aligned}
$$

Graph Laplacian and its Properties

Let $G = (V, E)$ be a graph with node set V and edge set E. We can define a scalar function $f : V \to \mathbb{R}$ that associates a scalar to each node of the graph.

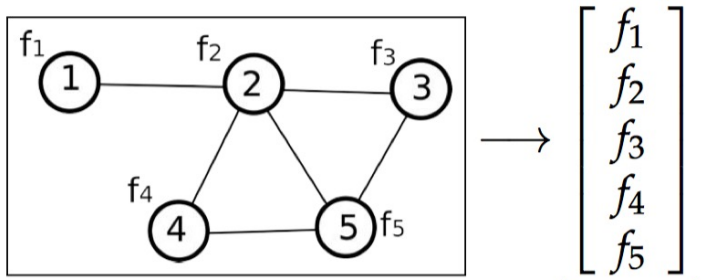


$$\longrightarrow \underbrace{\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{bmatrix}}_{\mathbf{f}}$$

$$\begin{aligned}
\mathbf{f}^T\mathbf{L}\mathbf{f} &= \mathbf{f}^T\mathbf{D}\mathbf{f} - \mathbf{f}^T\mathbf{W}\mathbf{f} \\
&= \sum_{i=1}^{n} d_i f_i^2 - \sum_{i,j=1}^{n} f_i f_j \omega_{ij} \\
&= \frac{1}{2}\left(\sum_{i=1}^{n} d_i f_i^2 - 2\sum_{i,j=1}^{n} f_i f_j \omega_{ij} + \sum_{j=1}^{n} d_j f_j^2\right) \\
&= \frac{1}{2}\sum_{i,j=1}^{n} \omega_{ij}(f_i - f_j)^2 \\
&= \sum_{(i,j)\in E}^{n} \omega_{ij}(f_i - f_j)^2
\end{aligned}$$

Graph Laplacian and its Properties

Let $G = (V, E)$ be a graph with node set V and edge set E. We can define a scalar function $f : V \to \mathbb{R}$ that associates a scalar to each node of the graph.



$$\rightarrow \underbrace{\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{bmatrix}}_{\mathbf{f}}$$

This is so called the **Smoothness Function** on a Graph

$$\begin{aligned}
\mathbf{f}^T\mathbf{L}\mathbf{f} &= \mathbf{f}^T\mathbf{D}\mathbf{f} - \mathbf{f}^T\mathbf{W}\mathbf{f} \\
&= \sum_{i=1}^{n} d_i f_i^2 - \sum_{i,j=1}^{n} f_i f_j \omega_{ij} \\
&= \frac{1}{2}\left(\sum_{i=1}^{n} d_i f_i^2 - 2\sum_{i,j=1}^{n} f_i f_j \omega_{ij} + \sum_{j=1}^{n} d_j f_j^2\right) \\
&= \frac{1}{2}\sum_{i,j=1}^{n} \omega_{ij}(f_i - f_j)^2 \\
&= \sum_{(i,j)\in E}^{n} \omega_{ij}(f_i - f_j)^2
\end{aligned}$$

Graph Laplacian and its Properties

This is so called the **Smoothness Function** on a Graph

In the field of machine learning, graph Laplacians are not only used for clustering, but also emerge for many other tasks such as semi-supervised learning or manifold reconstruction. In most applications, graph Laplacians are used to encode the assumption that data points which are "close" (i.e., w_{ij} is large) should have a "similar" label (i.e., $f_i \approx f_j$). A function $\mathbf{f}$ satisfies this assumption if $w_{ij}(f_i - f_j)^2$ is small for all i, j, that is $f^T L f$ is small. With this intuition one can for example use the quadratic form $f^T L f$ as a regularizer in a transductive classification problem.

Graph Laplacian and its Properties

This is so called the **Smoothness Function** on a Graph

In the field of machine learning, graph Laplacians are not only used for clustering, but also emerge for many other tasks such as semi-supervised learning or manifold reconstruction. In most applications, graph Laplacians are used to encode the assumption that data points which are "close" (i.e., w_{ij} is large) should have a "similar" label (i.e., $f_i \approx f_j$). A function $\mathbf{f}$ satisfies this assumption if $w_{ij}(f_i - f_j)^2$ is small for all i, j, that is $f^T L f$ is small. With this intuition one can for example use the quadratic form $f^T L f$ as a regularizer in a transductive classification problem.

One other way to interpret the use of graph Laplacians is by the **smoothness assumptions** they encode. A function f which has a low value of $f^T L f$ has the property that it varies only "a little bit" in regions where the data points lie dense (i.e., the graph is tightly connected), whereas it is allowed to vary more (e.g., to change the sign) in regions of low data density. In this sense, a small value of $f^T L f$ encodes the so called "cluster assumption" in semi-supervised learning, which requests that the decision boundary of a classifier should lie in a region of low density.

Graph Laplacian and its Properties

This is so called the **Smoothness Function** on a Graph

One other way to interpret the use of graph Laplacians is by the **smoothness assumptions** they encode. A function f which has a low value of $f^T L f$ has the property that it varies only "a little bit" in regions where the data points lie dense (i.e., the graph is tightly connected), whereas it is allowed to vary more (e.g., to change the sign) in regions of low data density. In this sense, a small value of $f^T L f$ encodes the so called "cluster assumption" in semi-supervised learning, which requests that the decision boundary of a classifier should lie in a region of low density.

for
s,
large)
small
f as

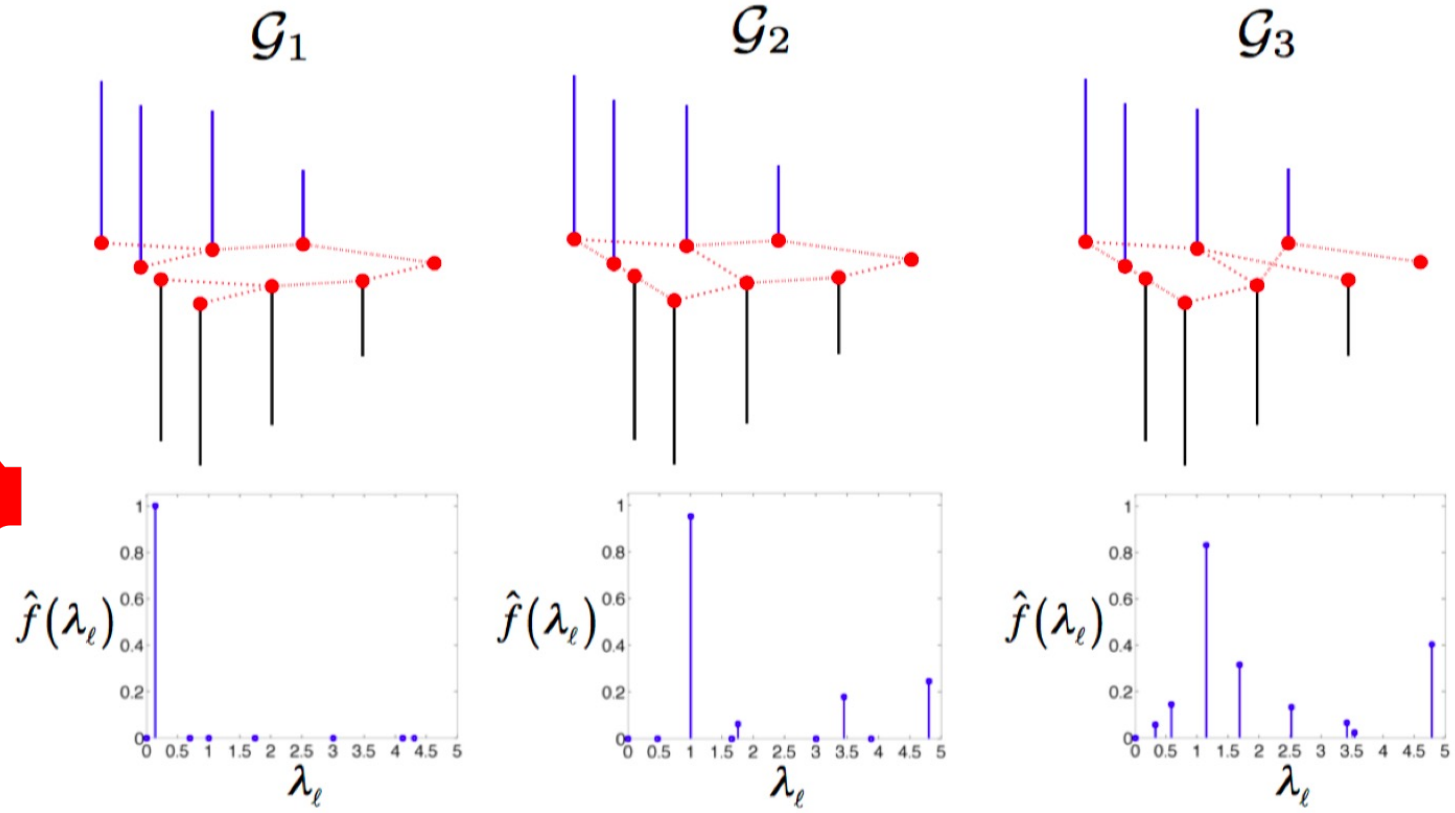
An intuition often used is that graph Laplacians formally look like a continuous Laplace operator (and this is also where the name "graph Laplacian" comes from). To see this, transform a local similarity w_{ij} to a distance d_{ij} by the relationship $w_{ij} = 1/d_{ij}^2$ and observe that

$$\omega_{ij}(f_i - f_j)^2 \approx (\frac{f_i - f_j}{d_{ij}})^2$$

Graph Laplacian and its Properties

More lower frequencies, more smooth

Compare subgraph structures from their **frequency** spectrum



Smoothness of a graph signal

$\mathbf{f}^{\mathrm{T}}\mathcal{L}_1\mathbf{f} = 0.14$ $\mathbf{f}^{\mathrm{T}}\mathcal{L}_2\mathbf{f} = 1.31$ $\mathbf{f}^{\mathrm{T}}\mathcal{L}_3\mathbf{f} = 1.81$

Graph Fourier Transform

What's the so called "frequency" on a Graph exactly means?

Graph Fourier Transform

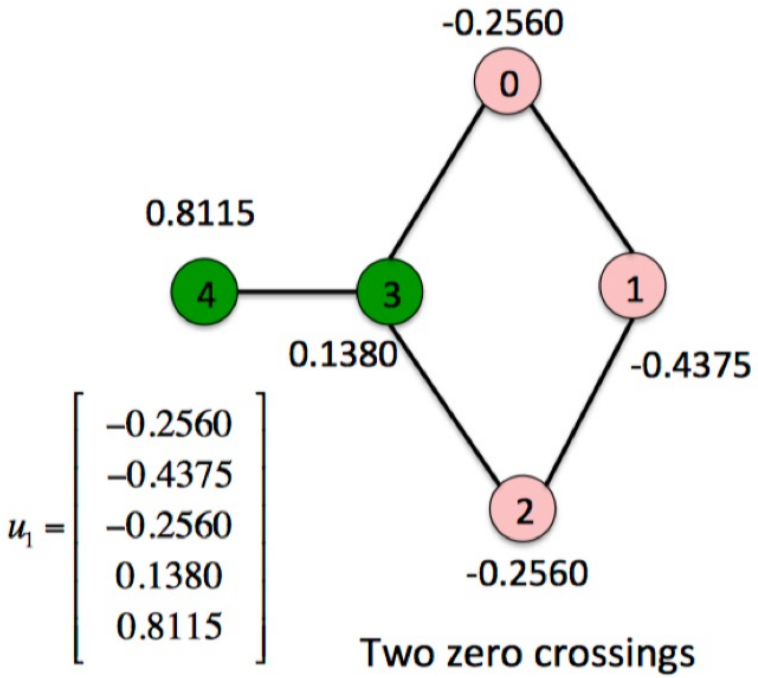
What's the so called "frequency" on a Graph exactly means?

Eigenvalues

$$\Lambda = \begin{bmatrix} 0.0000 \\ 0.8299 \\ 2.0000 \\ 2.6889 \\ 4.4812 \end{bmatrix}$$

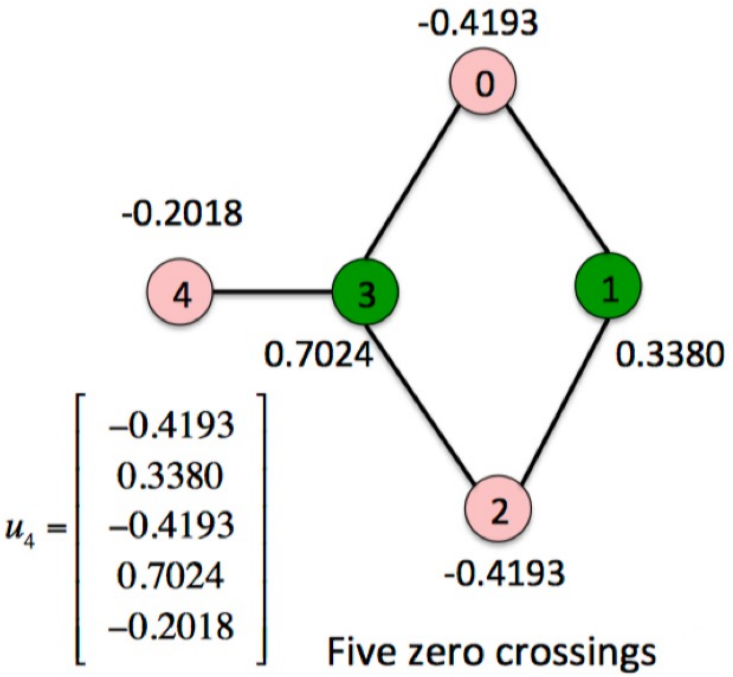
Eigenvectors

$$U = \begin{bmatrix} -0.4472 & -0.2560 & 0.7071 & 0.2422 & -0.4193 \\ -0.4472 & -0.4375 & 0.0000 & -0.7031 & 0.3380 \\ -0.4472 & -0.2560 & -0.7071 & 0.2422 & -0.4193 \\ -0.4472 & 0.1380 & 0.0000 & 0.5362 & 0.7024 \\ -0.4472 & 0.8115 & 0.0000 & -0.3175 & -0.2018 \end{bmatrix}$$



$$u_1 = \begin{bmatrix} -0.2560 \\ -0.4375 \\ -0.2560 \\ 0.1380 \\ 0.8115 \end{bmatrix}$$

Two zero crossings



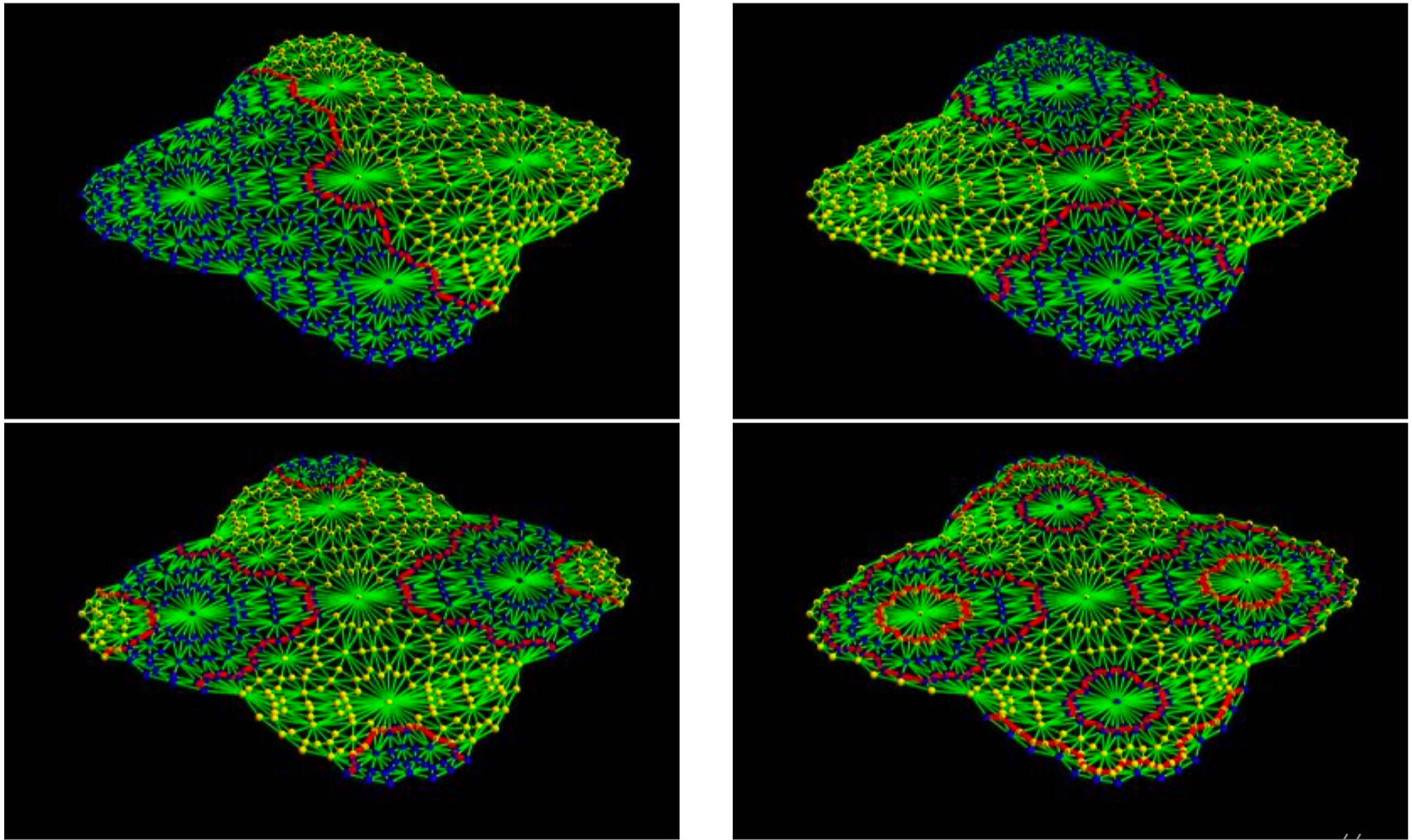
$$u_4 = \begin{bmatrix} -0.4193 \\ 0.3380 \\ -0.4193 \\ 0.7024 \\ -0.2018 \end{bmatrix}$$

Five zero crossings

$$x^T L x = \sum_{(i,j)\in E} |x_i - x_j|^2$$

$$u^T L u = u^T \lambda u = \lambda ||u||^2 = \lambda = \sum_{(i,j)\in E} |u_i - u_j|^2$$

Graph Fourier Transform



Graph Fourier Transform

The Laplacian matrix $\mathbf{L}$ of an undirected graph is a symmetric positive semi-definite matrix. The spectral decomposition theorem guarantees the existence of an orthonormal matrix $\mathbf{U}$ that diagonalizes $\mathbf{L}$

$$\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathbf{H}}$$

whereΛ is a diagonal matrix of non-negative real eigenvalues. The columns of $\mathbf{U}$ which are the eigenvectors $\{\mathbf{u}_0, \mathbf{u}_1, \cdots, \mathbf{u}_{N-1}\}$ corresponding to the ordered eigenvalues $\{0 \leq \lambda_0, \leq \lambda_1, \cdots, \leq \lambda_{N-1}\}$, constitute an orthonormal basis for $\mathbb{R}^N$.

Graph Fourier Transform

GFT transforms a graph signal $\mathbf{x} = (x_0, x_1, \cdots, x_{N-1})$ (each x_i corresponds to a node in the graph) into another sequence numbers, $X(k) = (X_0, X_1, \cdots, X_{N-1})$,which is defined by

$$X(k) =< \mathbf{u_k}, \mathbf{x} >= \sum_{j=0}^{N-1} x_j \mathbf{u}_k(j)$$

Where $\mathbf{u}_k$ is eigenvectors corresponding toeigenvalue λ_k of Laplacian matrix $\mathbf{L}$ of the graph.

For all "frequencies", GFT has:

$$\mathbf{X^{GFT}} = U^H \mathbf{x}$$

DFT transforms a sequence of N complex numbers $\mathbf{x} = (x_0, x_1, \cdots, x_{N-1})$ into another sequence of complex numbers, $X(k) = (X_0, X_1, \cdots, X_{N-1})$.

Let:

$$\mathbf{u_k} = (e^{-\frac{i2\pi}{N}k0}, e^{-\frac{i2\pi}{N}k1}, \cdots, e^{-\frac{i2\pi}{N}kj}, \cdots, e^{-\frac{i2\pi}{N}k(N-1)})$$

where $\mathbf{u}_k(j) = e^{-\frac{i2\pi}{N}kj}$, then

$$X(k) = \sum_{j=0}^{N-1} x_j e^{-\frac{i2\pi}{N}kj} = \sum_{j=0}^{N-1} x_j \mathbf{u}_k(j) =< \mathbf{u_k}, \mathbf{x} >$$

In summary, the eigenvalues of L play the role of **frequencies** and the eigenvectors the **Fourier basis**

Graph Fourier Transform

This analogy can also happened in the DTFT or CTFT

CTFT $$X(\omega) = \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt$$

DFT $$X(k) = \sum_{j=0}^{N-1} x_j e^{-\frac{i2\pi}{N} kj}$$

DTFT $$X(\omega) = \sum_{n=-\infty}^{\infty} x[n] e^{-j\omega n}$$

GFT $$X(k) = \sum_{j=0}^{N-1} x_j \mathbf{u}_k(j)$$

Spectral Graph Convolution

$$\mathcal{F}[f(i)] = f(\hat{\lambda}_l)$$

GFT:

$$\hat{f}(\lambda_l) = \sum_{i=1}^{n} f(i) u_l(i)$$

IGFT:

$$f(i) = \sum_{\lambda=0}^{n-1} \hat{f}(\lambda_l) u_l(i)$$

$$\mathcal{F}[f(i) * h(i)] = \mathcal{F}[f(i)]\mathcal{F}[h(i)] = \hat{f}(\lambda_l)\hat{h}(\lambda_l)$$

Spectral Graph Convolution

$$\mathcal{F}[f(i) * h(i)] = \mathcal{F}[f(i)]\mathcal{F}[h(i)] = \hat{f}\,(\lambda_l)\hat{h}(\lambda_l)$$

$$\begin{aligned} f(i) * h(i) &= IGFT[\hat{f}\,(\lambda_l)\hat{h}(\lambda_l)] \\ &= \sum_{l=0}^{n-1} \hat{f}\,(\lambda_l)\hat{h}(\lambda_l)u_l(i) \\ &= \sum_{l=0}^{n-1}\sum_{j=1}^{n} f(j)u_l(j)\hat{h}(\lambda_l)u_l(i) \end{aligned}$$

Spectral Graph Convolution

$$
\begin{aligned}
f(i) * h(i) &= IGFT[\hat{f}(\lambda_l)\hat{h}(\lambda_l)] \\
&= \sum_{l=0}^{n-1} \hat{f}(\lambda_l)\hat{h}(\lambda_l)u_l(i) \\
&= \sum_{l=0}^{n-1}\sum_{j=1}^{n} f(j)u_l(j)\hat{h}(\lambda_l)u_l(i)
\end{aligned}
$$

$$
\mathbf{f} * \mathbf{h} = \begin{pmatrix}
\sum_{l=0}^{n-1}\sum_{j=1}^{n} f(j)u_l(j)\hat{h}(\lambda_l)u_l(1) \\
\sum_{l=0}^{n-1}\sum_{j=1}^{n} f(j)u_l(j)\hat{h}(\lambda_l)u_l(2) \\
\cdots \\
\sum_{l=0}^{n-1}\sum_{j=1}^{n} f(j)u_l(j)\hat{h}(\lambda_l)u_l(i) \\
\cdots \\
\sum_{l=0}^{n-1}\sum_{j=1}^{n} f(j)u_l(j)\hat{h}(\lambda_l)u_l(n)
\end{pmatrix}
$$

Spectral Graph Convolution

$$\mathbf{f} * \mathbf{h} = \begin{pmatrix} \sum_{l=0}^{n-1} \sum_{j=1}^{n} f(j) u_l(j) \hat{h}(\lambda_l) u_l(1) \\ \sum_{l=0}^{n-1} \sum_{j=1}^{n} f(j) u_l(j) \hat{h}(\lambda_l) u_l(2) \\ \cdots \\ \sum_{l=0}^{n-1} \sum_{j=1}^{n} f(j) u_l(j) \hat{h}(\lambda_l) u_l(i) \\ \cdots \\ \sum_{l=0}^{n-1} \sum_{j=1}^{n} f(j) u_l(j) \hat{h}(\lambda_l) u_l(n) \end{pmatrix}$$

$$\mathbf{h}_\theta(\mathbf{\Lambda}) = \begin{pmatrix} \hat{h}_0(\lambda_0), & 0, & \cdots, & 0 \\ 0, & \hat{h}_\theta(\lambda_1), & \cdots, & 0 \\ & \cdots & & \\ 0, & \cdots & \cdots, & \hat{h}_{n-1}(\lambda_{n-1}) \end{pmatrix}$$

Spectral Graph Convolution

$$\mathbf{f} * \mathbf{h} = \begin{pmatrix} \sum_{l=0}^{n-1} \sum_{j=1}^{n} f(j) u_l(j) \hat{h}(\lambda_l) u_l(1) \\ \sum_{l=0}^{n-1} \sum_{j=1}^{n} f(j) u_l(j) \hat{h}(\lambda_l) u_l(2) \\ \cdots \\ \sum_{l=0}^{n-1} \sum_{j=1}^{n} f(j) u_l(j) \hat{h}(\lambda_l) u_l(i) \\ \cdots \\ \sum_{l=0}^{n-1} \sum_{j=1}^{n} f(j) u_l(j) \hat{h}(\lambda_l) u_l(n) \end{pmatrix}$$

$$\mathbf{h}_\theta(\mathbf{\Lambda}) = \begin{pmatrix} \hat{h}_0(\lambda_0), & 0, & \cdots, & 0 \\ 0, & \hat{h}_\theta(\lambda_1), & \cdots, & 0 \\ & \cdots & & \\ 0, & \cdots & \cdots, & \hat{h}_{n-1}(\lambda_{n-1}) \end{pmatrix}$$

$$\mathbf{f} * \mathbf{h} = \mathbf{U} \mathbf{h}_\theta(\mathbf{\Lambda}) \mathbf{U}^\mathbf{T} \mathbf{f}$$

Spectral Graph Convolution

$$\mathbf{f} * \mathbf{h} = \mathbf{U}\mathbf{h}_\theta(\mathbf{\Lambda})\mathbf{U}^\mathbf{T}\mathbf{f}$$

$$\mathbf{h}_\theta(\mathbf{\Lambda}) = \begin{pmatrix} \hat{h}_0(\lambda_0), & 0, & \cdots, & 0 \\ 0, & \hat{h}_\theta(\lambda_1), & \cdots, & 0 \\ & \cdots & & \\ 0, & \cdots & \cdots, & \hat{h}_{n-1}(\lambda_{n-1}) \end{pmatrix}$$

$$\mathbf{U} = \begin{pmatrix} | & | & & u_l(1) & & | \\ | & | & & | & & | \\ u_0 & u_1 & \cdots & u_l & \cdots & u_{n-1} \\ | & | & & | & & | \\ | & | & & u_l(n) & & | \end{pmatrix}_{n \times n} \quad \mathbf{U}^\mathbf{T} = \begin{pmatrix} - & - & u_0^T & - & - \\ - & - & u_1^T & - & - \\ & & \cdots & & \\ u_l(1) & - & u_l^T & - & u_l(n) \\ & & \cdots & & \\ - & - & u_{n-1}^T & - & - \end{pmatrix}_{n \times n} \quad \mathbf{f} = \begin{pmatrix} f(1) \\ f(2) \\ \cdots \\ f(i) \\ \cdots \\ f(n) \end{pmatrix}_{n \times 1}$$

Spectral Graph Convolution

$$\mathbf{f} * \mathbf{h} = \mathbf{U}\mathbf{h}_\theta(\mathbf{\Lambda})\mathbf{U}^\mathbf{T}\mathbf{f}$$

$$\mathbf{h}_\theta(\mathbf{\Lambda}) = \begin{pmatrix} \hat{h}_0(\lambda_0), & 0, & \cdots, & 0 \\ 0, & \hat{h}_\theta(\lambda_1), & \cdots, & 0 \\ & \cdots & & \\ 0, & \cdots & \cdots, & \hat{h}_{n-1}(\lambda_{n-1}) \end{pmatrix}$$

$$\mathbf{U}^\mathbf{T} = \begin{pmatrix} - & - & u_0^T & - & - \\ - & - & u_1^T & - & - \\ & & \cdots & & \\ u_l(1) & - & u_l^T & - & u_l(n) \\ & & \cdots & & \\ - & - & u_{n-1}^T & - & - \end{pmatrix}_{n \times n} \quad \mathbf{f} = \begin{pmatrix} f(1) \\ f(2) \\ \cdots \\ f(i) \\ \cdots \\ f(n) \end{pmatrix}_{n \times 1}$$

$$\mathbf{U}^\mathbf{T}\mathbf{f} = \begin{pmatrix} \sum_{j=1}^n f(j)u_0(j) \\ \sum_{j=1}^n f(j)u_1(j) \\ \cdots \\ \sum_{j=1}^n f(j)u_l(j) \\ \cdots \\ \sum_{j=1}^n f(j)u_{n-1}(j) \end{pmatrix}_{n \times 1}$$

Spectral Graph Convolution

$$\mathbf{f} * \mathbf{h} = \mathbf{U}\mathbf{h}_\theta(\mathbf{\Lambda})\mathbf{U}^\mathbf{T}\mathbf{f}$$

$$\mathbf{h}_\theta(\mathbf{\Lambda}) = \begin{pmatrix} \hat{h}_0(\lambda_0), & 0, & \cdots, & 0 \\ 0, & \hat{h}_\theta(\lambda_1), & \cdots, & 0 \\ & \cdots & & \\ 0, & \cdots & \cdots, & \hat{h}_{n-1}(\lambda_{n-1}) \end{pmatrix} \qquad \mathbf{U}^\mathbf{T}\mathbf{f} = \begin{pmatrix} \sum_{j=1}^n f(j)u_0(j) \\ \sum_{j=1}^n f(j)u_1(j) \\ \cdots \\ \sum_{j=1}^n f(j)u_l(j) \\ \cdots \\ \sum_{j=1}^n f(j)u_{n-1}(j) \end{pmatrix}_{n\times 1}$$

$$\mathbf{h}_\theta(\mathbf{\Lambda})\mathbf{U}^\mathbf{T}\mathbf{f} = \begin{pmatrix} \hat{h}(\lambda_0)\sum_{j=1}^n f(j)u_0(j) \\ \cdots \\ \hat{h}(\lambda_l)\sum_{j=1}^n f(j)u_l(j) \\ \cdots \\ \hat{h}(\lambda_{n-1})\sum_{j=1}^n f(j)u_{n-1}(j) \end{pmatrix}_{n\times 1}$$

Spectral Graph Convolution

$$\mathbf{f} * \mathbf{h} = \mathbf{U}\mathbf{h}_\theta(\mathbf{\Lambda})\mathbf{U}^\mathbf{T}\mathbf{f}$$

$$\mathbf{U} = \begin{pmatrix} | & | & & u_l(1) & & | \\ | & | & & | & & | \\ u_0 & u_1 & \cdots & u_l & \cdots & u_{n-1} \\ | & | & & | & & | \\ | & | & & u_l(n) & & | \end{pmatrix}_{n \times n}$$

$$\mathbf{h}_\theta(\mathbf{\Lambda})\mathbf{U}^\mathbf{T}\mathbf{f} = \begin{pmatrix} \hat{h}(\lambda_0) \sum_{j=1}^{n} f(j) u_0(j) \\ \cdots \\ \hat{h}(\lambda_l) \sum_{j=1}^{n} f(j) u_l(j) \\ \cdots \\ \hat{h}(\lambda_{n-1}) \sum_{j=1}^{n} f(j) u_{n-1}(j) \end{pmatrix}_{n \times 1}$$

$$\mathbf{U}\mathbf{h}_\theta(\mathbf{\Lambda})\mathbf{U}^\mathbf{T}\mathbf{f} = \begin{pmatrix} \sum_{l=0}^{n-1} u_l(1) \hat{h}(\lambda_0) \sum_{j=1}^{n} f(j) u_0(j) \\ \cdots \\ \sum_{l=0}^{n-1} u_l(i) \hat{h}(\lambda_l) \sum_{j=1}^{n} f(j) u_l(j) \\ \cdots \\ \sum_{l=0}^{n-1} u_l(n) \hat{h}(\lambda_{n-1}) \sum_{j=1}^{n} f(j) u_{n-1}(j) \end{pmatrix}_{n \times 1}$$

Spectral Graph Convolution

$$\mathbf{f} * \mathbf{h} = \mathbf{U}\mathbf{h}_\theta(\mathbf{\Lambda})\mathbf{U}^\mathbf{T}\mathbf{f}$$

$$\mathbf{U}\mathbf{h}_\theta(\mathbf{\Lambda})\mathbf{U}^\mathbf{T}\mathbf{f} = \begin{pmatrix} \sum_{l=0}^{n-1} u_l(1)\hat{h}(\lambda_0)\sum_{j=1}^{n} f(j)u_0(j) \\ \cdots \\ \sum_{l=0}^{n-1} u_l(i)\hat{h}(\lambda_l)\sum_{j=1}^{n} f(j)u_l(j) \\ \cdots \\ \sum_{l=0}^{n-1} u_l(n)\hat{h}(\lambda_{n-1})\sum_{j=1}^{n} f(j)u_{n-1}(j) \end{pmatrix}_{n\times 1}$$

$$\mathbf{f} * \mathbf{h} = \begin{pmatrix} \sum_{l=0}^{n-1}\sum_{j=1}^{n} f(j)u_l(j)\hat{h}(\lambda_l)u_l(1) \\ \sum_{l=0}^{n-1}\sum_{j=1}^{n} f(j)u_l(j)\hat{h}(\lambda_l)u_l(2) \\ \cdots \\ \sum_{l=0}^{n-1}\sum_{j=1}^{n} f(j)u_l(j)\hat{h}(\lambda_l)u_l(i) \\ \cdots \\ \sum_{l=0}^{n-1}\sum_{j=1}^{n} f(j)u_l(j)\hat{h}(\lambda_l)u_l(n) \end{pmatrix}$$

$$\sum_{l=0}^{n-1}\sum_{j=1}^{n} f(j)u_l(j)\hat{h}(\lambda_l)u_l(i)$$

$$=$$

$$\sum_{l=0}^{n-1} u_l(i)\hat{h}(\lambda_l)\sum_{j=1}^{n} f(j)u_l(j)$$

Spectral Graph Convolution

$$\mathbf{f} * \mathbf{h} = \mathbf{U}\mathbf{h}_\theta(\mathbf{\Lambda})\mathbf{U}^\mathbf{T}\mathbf{f}$$

Spectral approaches work with a spectral representation of the graphs. [35] proposed the spectral network. The convolution operation is defined in the Fourier domain by computing the eigendecomposition of the graph Laplacian. The operation can be defined as the multiplication of a signal $\mathbf{x} \in \mathbb{R}^N$ (a scalar for each node) with a filter $\mathbf{g}_\theta = \text{diag}(\theta)$ parameterized by $\theta \in \mathbb{R}^N$:

$$\mathbf{g}_\theta \star \mathbf{x} = \mathbf{U}\mathbf{g}_\theta(\mathbf{\Lambda})\mathbf{U}^T\mathbf{x} \tag{10}$$

Graph Neural Networks: A Review of Methods and Applications

Convolution Kernel

Recall our previously mentioned GTF (Graph Fourier Transform), that is

$$\hat{f}(\lambda_l) = \sum_{i=1}^{n} f(i) u_l(i)$$

As we talked before, the eigenvalues of $\mathbf{L}$ play the role of frequencies and the eigenvectors the Fourier basis. The graph Fourier transform (and its inverse) gives a way to represent a signal in two different domains: **the vertex domain** and **the graph spectral domain**.

Convolution Kernel

We can also leverage its inverse transform to synthesize signals.

$$f(i) = \sum_{l=0}^{n-1} \hat{f}\left(\lambda_l\right) u_l(i)$$

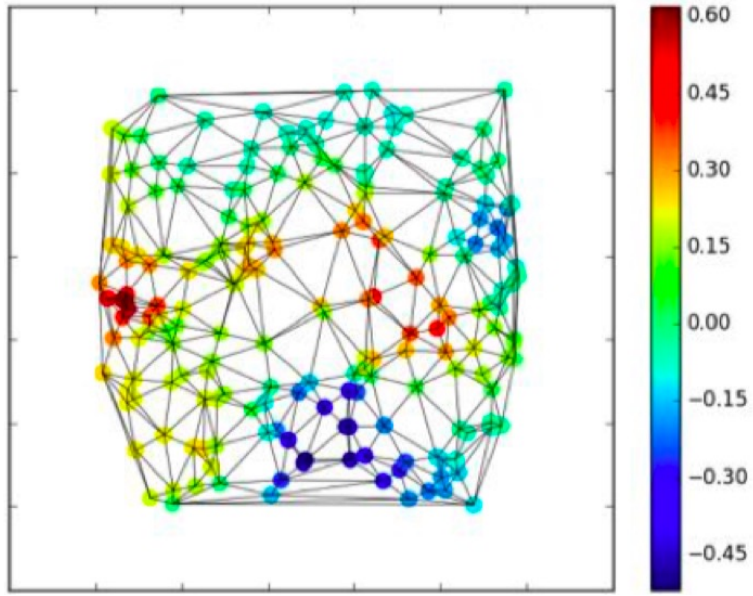
$$\Downarrow$$

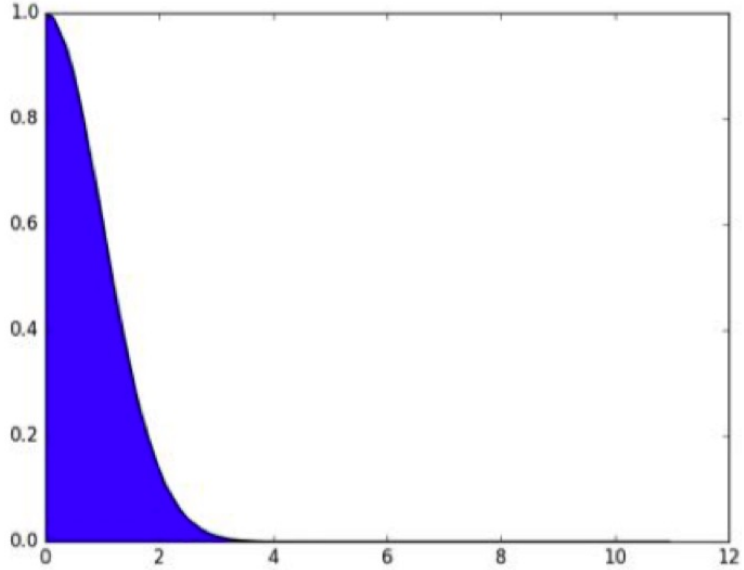
$$f(i) = \sum_{l=0}^{n-1} \hat{g}(\lambda_l) u_l(i)$$

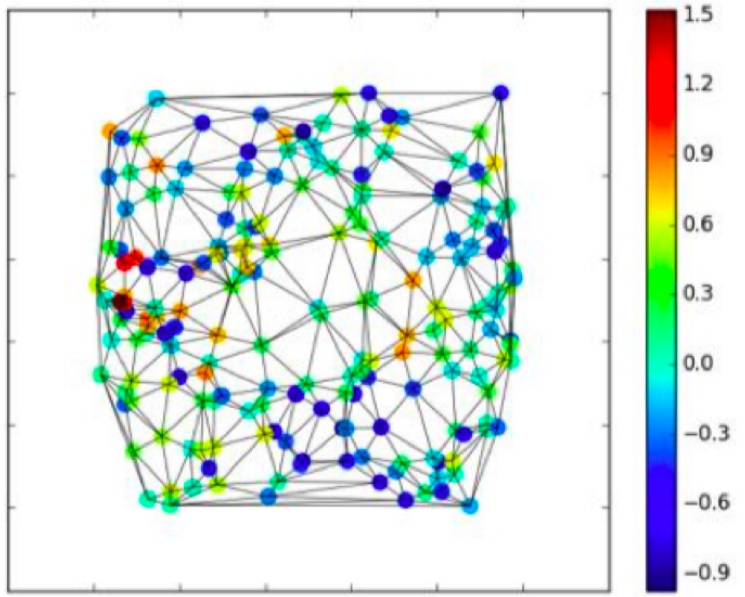
We can generate signals in the graph domain by using a kernel in the spectral domain. For example, in the same graph, we design different g, then we can get different embedding results:

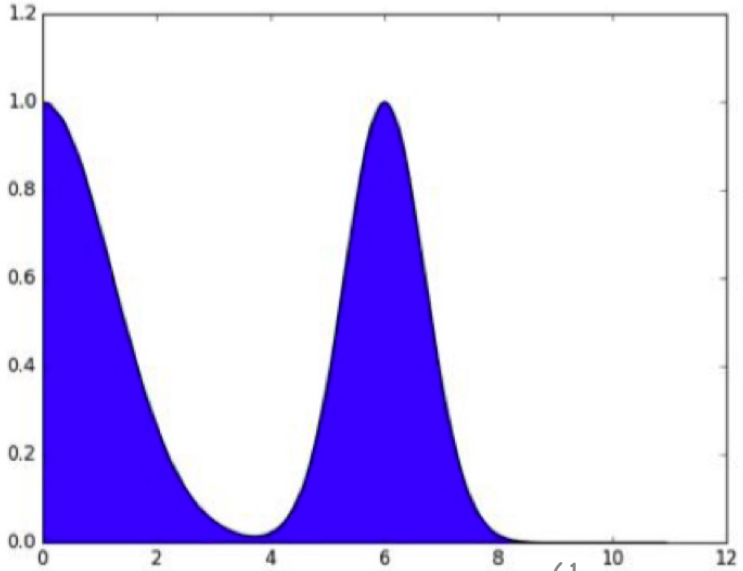
Convolution Kernel

$$f(i) = \sum_{l=0}^{n-1} \hat{g}(\lambda_l) u_l(i)$$









Convolution Kernel

Now let's look at how the output signal is generated when we design a typical kernel, that is $\hat{h}(\lambda_l) = \sum_{k=0}^{K} a_k \lambda_l^k$ (polynomial on the spectral domain).

As we deduced in section 9 Spectral Graph Convolution, the output signal is the graph convolution of f and h, that is

$$
\begin{aligned}
f(i) * h(i) &= IGFT[\hat{f}(\lambda_l)\hat{h}(\lambda_l)] \\
&= \sum_{j=1}^{n} f(j) \sum_{k=0}^{K} a_k \sum_{l=0}^{n-1} \lambda_l^k u_l(i) u_l(j)
\end{aligned}
$$

Convolution Kernel

$$f(i) * h(i) = IGFT[\hat{f}(\lambda_l)\hat{h}(\lambda_l)]$$
$$= \sum_{j=1}^{n} f(j) \sum_{k=0}^{K} a_k \sum_{l=0}^{n-1} \lambda_l^k u_l(i) u_l(j)$$

Notice that $\mathbf{L}^k = U\Lambda^k U^h$ (We talked about this in Section 7), then we have:

$$(\mathbf{L}^k)_{ij} = \sum_{l=0}^{n-1} \lambda_l^k u_l(i) u_l(j)$$

Let

$$b_{ij} = \sum_{k=0}^{K} a_k \mathbf{L}_{ij}^k$$

Notice that $\mathbf{L}^k = U\Lambda^k U^h$ (We talked about this in Section 7), then we have:

Convolution Kernel

$$f(i) * h(i) = IGFT[\hat{f}(\lambda_l)\hat{h}(\lambda_l)]$$
$$= \sum_{j=1}^{n} f(j) \sum_{k=0}^{K} a_k \sum_{l=0}^{n-1} \lambda_l^k u_l(i) u_l(j)$$

$$(\mathbf{L}^k)_{ij} = \sum_{l=0}^{n-1} \lambda_l^k u_l(i) u_l(j)$$

$$b_{ij} = \sum_{k=0}^{K} a_k \mathbf{L}_{ij}^k$$

Then

$$f(i) * h(i) = \sum_{j \in \mathcal{N}(i)}^{n} b_{ij} f(j)$$

It can be shown that $(\mathbf{L}^k)_{ij} = 0$ when the shortest distance between nodes i and j is greater than k. We talked about this property in section 5.2

Convolution Kernel

Then

$$f(i) * h(i) = \sum_{j \in \mathcal{N}(i)}^{n} b_{ij} f(j)$$

It can be shown that $(\mathbf{L}^k)_{ij} = 0$ when the shortest distance between nodes i and j is greater than k. We talked about this property in section 5.2

> Therefore, when the filter is a polynomial in the spectral domain, the filtered signal in each node i is a linear combination of the original signal in the neighborhood of i.

Convolution Kernel

The following papers all mention spectral graph convolution operation with their own designed kernels:

1. M. Henaff, J. Bruna, and Y. Lecun, "Deep convolutional networks on graph-structured data." arXiv: Learning, 2015.
2. D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," Applied and Computational Harmonic Analysis, vol. 30, no. 2, pp. 129–150, 2011.
3. M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," NIPS 2016, pp. 3844–3852, 2016.
4. T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," ICLR 2017, 2017.
5. Y. C. Ng, N. Colombo, and R. Silva, "Bayesian semi-supervised learning with graph gaussian processes," in NeurIPS 2018, 2018, pp. 1690–1701.

Now let's select a part of work above for further discussion:

> T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," ICLR 2017, 2017.

Further Reading

- Spectral decomposition
- Laplacian matrix and random walks
- Harmonic Analysis on Graphs and Networks

References

Papers

1. Graph Neural Networks: A Review of Methods and Applications, Zhou, Jie.
2. Discrete signal processing on graphs graph fourier transform, Aliaksei Sandryhaila
3. A tutorial on spectral clustering, Ulrike von Luxburg
4. Discrete Signal Processing on Graphs, Aliaksei Sandryhaila
5. Deep Convolutional Networks on Graph structured data, Mikael Henaff
6. Semi-supervised classification with graph convolutional networks, Thomas N.Kipf
7. Spectral convolution networks, Maria Francesca
8. Deep collective classification in heterogeneous information networks, Yizhou Zhang
9. Spectral networks and locally connected networks on graphs, Joan Bruna

Lectures

1. Spectral graph analysis, Aristides Gionis
2. Graph Signal Processing, Prof. Luis Gustavo Nonato
3. Spectral Graph Theory, Lecture 5, David P. Willianmson
4. Lectures on Spectral Graph Theory， Fan R. K. Chung
5. Harmonic Analysis on Graphs and Networks, Pierre Vandergheynst
6. Spectral Convolution Networks, Maria Francesca, Arthur Hughes, David Gregg

Websites

1. https://en.wikipedia.org/wiki/Discrete_Fourier_transform
2. https://www.zhihu.com/question/22085329
3. https://www.cnblogs.com/pinard/p/6221564.html
4. https://en.wikipedia.org/wiki/Laplace_operator
5. https://en.wikipedia.org/wiki/Algebraic_connectivity
6. https://en.wikipedia.org/wiki/Discrete_Laplace_operator
7. https://blog.csdn.net/chensi1995/article/details/77232019

Books

1. Pattern Recognition and Machine Learning,
2. Graph Structured Data Viewed Through a Fourier Lens, Venkatesan Ekambaram, Page 13

Discrete nodal domain theorems. Linear Algebra and its Applications, 336(1):51–60, 2001.

4. Spectral graph theory, Daniel Spielman
5. Matrix analysis and applied linear algebra,Carl D Meyer

See you next week☺

May you smile like a child