Spectral Clustering, Graph Neural Networks, and Transformers

CS 145: Introduction to Data Mining (Spring 2024)

Yanqiao Zhu

UCLA

May 17, 2024

Announcements

- HW4 scores have been released
- HW5 solution will be released tomorrow
- No more assignments!

Midterm Exam

Date
- May 20, 12:00PM–1:45PM
- In-person only; no online or make-up exams offered

Format
- Close-book but double-sided letter-sized cheat sheets allowed, **up to 10 pages**
- Simple calculators allowed
- Internet access strictly prohibited

Scope
- Supervised Learning: Linear Regression, Logistic Regression, MLP, Gradient Descent, Backpropagation, Regularization, Batch/Layer Norm, Decision Trees, Random Forests, Mixture-of-Expert, Ensemble (Bagging, Boosting, Adaboost), K-Nearst Neighbor
- Unsupervised Learning: K-Means, Gaussian Mixture Models, EM Algorithm, (Variational) Auto Encoders
- Graphs and Networks: Random Walks, Spectral Clustering, Graph Representation Learning

Midterm Exam

Structure

- Part A. True/False Questions ($8 * 2 = 16$ points)
- Part B. Multiple-Choice Questions ($5 * 2 = 10$ points)
- Part C. Fill-in-the-Blank Questions ($21 * 1 = 21$ points)
- Part D. Open-Answer Questions (63 points)
  - Problem 17. Linear Regression with Regularization (16 points)
  - Problem 18. Multilayer Perceptron and Backpropagation (15 points)
  - Problem 19. Decision Trees and Bagging (16 points)
  - Problem 20. K-Means and Gaussian Mixture Models (16 points)

Scores

- 110 points in total, with the additional **10** points serving as bonus points

Outline

1. Spectral Clustering

2. Graph Neural Networks

3. Transformers

Graph Representations and Notations

- A graph is denoted as $G = (V, E)$, where V is the set of vertices (nodes) and E is the set of edges
- $|V| = n$ is the number of vertices, and $|E| = m$ is the number of edges
- Adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$: $\boldsymbol{A}_{ij} = 1$ if $(i, j) \in E$, and 0 otherwise
- Degree matrix $\boldsymbol{D} \in \mathbb{R}^{n \times n}$: diagonal matrix with $\boldsymbol{D}_{ii} = \sum_{j=1}^{n} \boldsymbol{A}_{ij}$

Spectral Clustering

- Spectral clustering: K-Means in a transformed space
  - Preprocessing: Construct a graph and compute its Laplacian matrix
  - Transformation: Compute eigenvectors of the Laplacian matrix
  - Clustering: Run K-Means on the eigenvectors
- Motivation: Solve graph partitioning problems by optimizing graph cut objectives

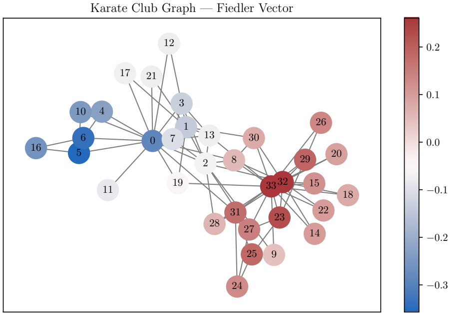
	Construct Laplacian		Compute Eigenvectors		K-Means Clustering	
Graph $G = (V, E, \boldsymbol{W})$	$\longrightarrow$	Laplacian Matrix $\boldsymbol{L}$ (or $\boldsymbol{L}_{\mathrm{sym}}$, $\boldsymbol{L}_{\mathrm{rw}}$)	$\longrightarrow$	Eigenvectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ (or $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k$)	$\longrightarrow$	Clusters $C_1, C_2, \ldots, C_k$

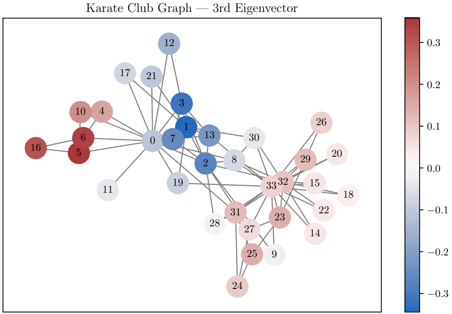
Graph Laplacian and Spectrum

- Undirected weighted graph $G = (V, E, \boldsymbol{W})$
- Degree matrix $\boldsymbol{D}$: diagonal matrix with $\boldsymbol{D}_{ii} = \sum_{j=1}^{n} \boldsymbol{W}_{ij}$
- Unnormalized Laplacian: $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$
- Normalized Laplacians:
  - Symmetric: $\boldsymbol{L}_{\text{sym}} = \boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2}$
  - Random walk: $\boldsymbol{L}_{\text{rw}} = \boldsymbol{D}^{-1}\boldsymbol{L}$
- Quadratic form: $\boldsymbol{x}^\top \boldsymbol{L}\boldsymbol{x} = \sum_{(i,j)\in E} \boldsymbol{W}_{ij}(x_i - x_j)^2$
- Spectrum: eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ and eigenvectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$
- Key properties:
  - $\boldsymbol{L}$, $\boldsymbol{L}_{\text{sym}}$, $\boldsymbol{L}_{\text{rw}}$: symmetric, positive semi-definite
  - $\lambda_1 = 0$, $\boldsymbol{v}_1 = \mathbf{1}$ (constant vector)
  - Number of connected components = multiplicity of $\lambda_1 = 0$
  - Fiedler vector (eigenvector $\boldsymbol{v}_2$) for bi-partitioning

Graph Laplacian and Spectrum

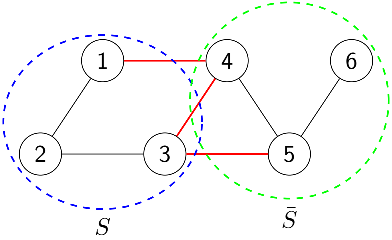
- Eigenvalues and eigenvectors of these matrices encode important structural properties of the graph
- The Fiedler vector shows a gradual change across the graph, indicating a smooth transition of values





Graph Cuts and Optimization

- Graph cut: cut$(S, \bar{S}) = \sum_{i \in S, j \in \bar{S}} \boldsymbol{W}_{ij}$
- Ratio cut objective: $\sum_{i=1}^{k} \frac{\text{cut}(S_i, \bar{S}_i)}{|S_i|}$
  - Relaxation: $\min_{\boldsymbol{X}} \text{tr}(\boldsymbol{X}^\top \boldsymbol{L} \boldsymbol{X})$ s.t. $\boldsymbol{X}^\top \boldsymbol{X} = \boldsymbol{I}$
  - Solution: $\boldsymbol{X} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k]$
- Normalized cut objective: $\sum_{i=1}^{k} \frac{\text{cut}(S_i, \bar{S}_i)}{\text{vol}(S_i)}$, where vol$(S_i) = \sum_{j \in S_i} \boldsymbol{D}_{jj}$
  - Relaxation: $\min_{\boldsymbol{X}} \text{tr}(\boldsymbol{X}^\top \boldsymbol{L}_{\text{rw}} \boldsymbol{X})$ s.t. $\boldsymbol{X}^\top \boldsymbol{D} \boldsymbol{X} = \boldsymbol{I}$
  - Solution: $\boldsymbol{X} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k]$, where $\boldsymbol{L}_{\text{rw}} \boldsymbol{u}_i = \lambda_i \boldsymbol{D} \boldsymbol{u}_i$



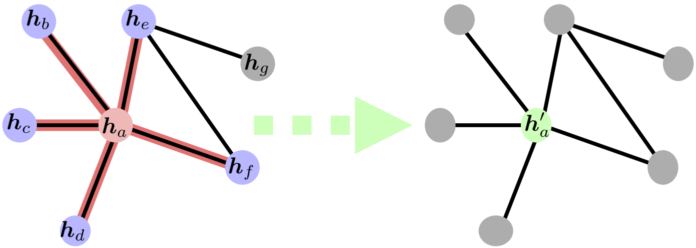
Spectral Clustering

Spectral Clustering Algorithms

- Unnormalized spectral clustering (Ratio cut):
  1. Construct unnormalized Laplacian $\boldsymbol{L}$
  2. Compute first k eigenvectors of $\boldsymbol{L}$: $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$
  3. Run K-Means on the rows of $\boldsymbol{X} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k]$
- Normalized spectral clustering (Normalized cut):
  1. Construct normalized Laplacian $\boldsymbol{L}_{\text{rw}}$ (or $\boldsymbol{L}_{\text{sym}}$)
  2. Compute first k generalized eigenvectors: $\boldsymbol{L}_{\text{rw}}\boldsymbol{u}_i = \lambda_i \boldsymbol{D}\boldsymbol{u}_i$
  3. Run K-Means on the rows of $\boldsymbol{X} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k]$

Graph Neural Networks (GNNs)

- GNNs generalize convolutional operations to the graph domain
- Key idea: neighborhood aggregation
  - Update node representations by aggregating information from neighboring nodes
  - Repeated for multiple layers to capture higher-order dependencies



Graph Neural Networks (GNNs)

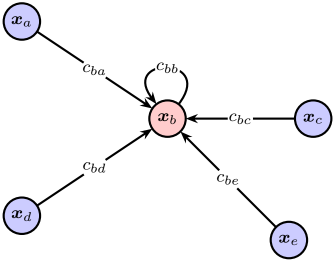
- Formally, a GNN layer can be decomposed into three functions:

$$\boldsymbol{h}_i^{(k)} = \overbrace{\gamma^{(k)}}^{\text{Update function}} \left(\boldsymbol{h}_i^{(k-1)}, \underbrace{\bigoplus_{j \in \mathcal{N}(i)}}_{\text{Aggregate function}} \overbrace{\phi^{(k)}}^{\text{Message function}} \left(\boldsymbol{h}_i^{(k-1)}, \boldsymbol{h}_j^{(k-1)} \right) \right)$$

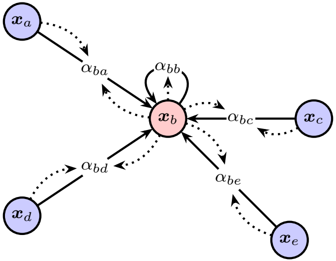
Three "Flavors" of GNNs

- Graph Convolutional Network (GCN):
  - Aggregate neighborhood information using convolutional filters
  - Simplifies spectral graph convolutions using first-order approximation
  - $\boldsymbol{H}^{(l+1)} = \sigma(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)})$
- Graph Attention Network (GAT):
  - Introduces attention mechanism to assign different weights to neighbors
  - $\boldsymbol{h}_i^{(l+1)} = \sigma(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(l)}\boldsymbol{W}^{(l)}\boldsymbol{h}_j^{(l)})$
  - $\alpha_{ij}^{(l)} = \frac{\exp(\text{LeakyReLU}(\boldsymbol{a}^{(l)\top}[\boldsymbol{W}^{(l)}\boldsymbol{h}_i^{(l)}\|\boldsymbol{W}^{(l)}\boldsymbol{h}_j^{(l)}]))}{\sum_{k\in\mathcal{N}(i)}\exp(\text{LeakyReLU}(\boldsymbol{a}^{(l)\top}[\boldsymbol{W}^{(l)}\boldsymbol{h}_i^{(l)}\|\boldsymbol{W}^{(l)}\boldsymbol{h}_k^{(l)}]))}$
- Message Passing Neural Network (MPNN):
  - Unifies various GNN architectures under a general message passing framework
  - $\boldsymbol{m}_i^{(l+1)} = \sum_{j\in\mathcal{N}(i)} M^{(l)}(\boldsymbol{h}_i^{(l)}, \boldsymbol{h}_j^{(l)}, \boldsymbol{e}_{ij})$
  - $\boldsymbol{h}_i^{(l+1)} = U^{(l)}(\boldsymbol{h}_i^{(l)}, \boldsymbol{m}_i^{(l+1)})$

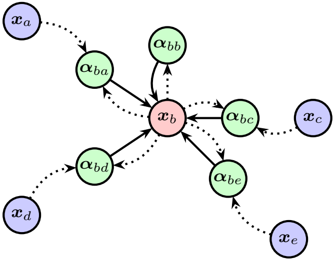
Three “Flavors” of GNNs



Convolutional



Attentional



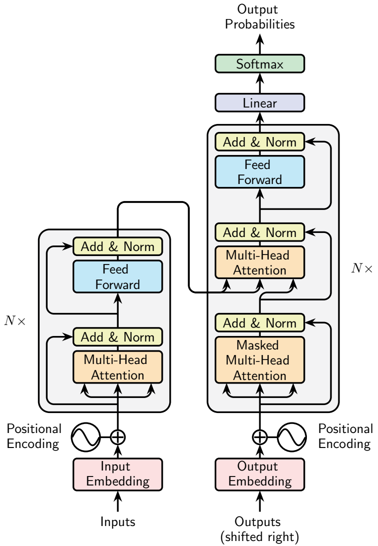
Message-Passing

Graph Tasks with GNN Embeddings

- Node-level tasks:
  - Node classification: $\hat{y}_i = f_c(\boldsymbol{h}_i)$, where f_c is a classifier
  - Node regression: $\hat{y}_i = f_r(\boldsymbol{h}_i)$, where f_r is a regression model
- Link-level tasks:
  - Link prediction: $\hat{y}_{ij} = f_l(\boldsymbol{h}_i \parallel \boldsymbol{h}_j)$, where f_l is a binary classifier and $\parallel$ denotes concatenation
  - Edge classification: $\hat{y}_{ij} = f_e(\boldsymbol{h}_i \parallel \boldsymbol{h}_j)$, where f_e is a classifier
- Community-level tasks:
  - Community detection: $\{C_1, \ldots, C_K\} = \text{Clustering}(\{\boldsymbol{h}_i \mid i \in V\})$
  - Subgraph classification: $\hat{y}_{S_k} = f_s(\boldsymbol{h}_{S_k})$, where $\boldsymbol{h}_{S_k} = \text{Aggregate}(\{\boldsymbol{h}_i \mid i \in S_k\})$ and f_s is a classifier
- Graph-level tasks:
  - Graph classification: $\hat{y}_{G_i} = f_g(\boldsymbol{h}_{G_i})$, where $\boldsymbol{h}_{G_i} = \text{Aggregate}(\{\boldsymbol{h}_i \mid i \in V_i\})$ and f_g is a classifier
  - Graph regression: $\hat{y}_{G_i} = f_r(\boldsymbol{h}_{G_i})$, where $\boldsymbol{h}_{G_i} = \text{Aggregate}(\{\boldsymbol{h}_i \mid i \in V_i\})$ and f_r is a regression model

Transformers

- Transformers are a type of deep learning model that rely heavily on the attention mechanism
- Introduced by Vaswani et al. (2017) for sequence-to-sequence tasks, particularly in natural language processing



Tokenization

- Tokenization is the process of breaking down a sequence (e.g., a sentence) into smaller units called tokens
- Tokens can be:
  - Words
  - Subwords (e.g., WordPiece, BPE)
  - Characters
- Each token is mapped to a unique integer ID, which is then used to look up the corresponding embedding vector

The quick brown fox jumps over the lazy dog

↓

[The] [quick] [brown] [fox] [jumps] [over] [the] [lazy] [dog]

↓

[12] [542] [1201] [783] [3120] [4500] [12] [7891] [1345]

Attention Mechanism

- Attention is a mechanism that allows the model to focus on different parts of the input sequence when generating each output token
- Mathematically, attention can be described as a weighted sum of values, where the weights are computed based on the query and keys:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V}$$

  - $\boldsymbol{Q}$ (Query): The current hidden state
  - $\boldsymbol{K}$ (Keys): The hidden states of the input sequence
  - $\boldsymbol{V}$ (Values): The hidden states of the input sequence (same as keys)
- The attention weights are computed by taking the dot product between the query and keys, scaled by $\sqrt{d_k}$ (the dimensionality of the keys), and then passed through a softmax function

Positional Encoding

- Positional encoding is a way to inject position information into the input embeddings
- It allows the model to learn relative positions of tokens in a sequence
- Positional encodings are typically sinusoidal functions of different frequencies

$$\mathsf{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\mathsf{model}}}}\right)$$
$$\mathsf{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\mathsf{model}}}}\right)$$

  - pos: The position in the sequence
  - i: The dimension index
  - d_{model}: The dimensionality of the embeddings

Next Token Prediction

- Transformers are often used in a language modeling setup, where the goal is to predict the next token in a sequence given the previous tokens
- To perform next token prediction, the model:
  1. Passes the input tokens through the Transformer layers
  2. Takes the final hidden state corresponding to the last input token
  3. Passes this hidden state through a linear layer followed by a softmax to produce a probability distribution over the vocabulary
  4. Chooses the token with the highest probability as the predicted next token
- During training, the model is typically trained using a cross-entropy loss between the predicted probabilities and the actual next token
- During inference, the predicted next token can be sampled from the output distribution or chosen deterministically (e.g., by taking the argmax)

Transformer Architectures

- Encoder-only models:
  - Stack of encoder layers
  - Self-attention and feed-forward networks
  - Used for tasks like classification and extraction
- Decoder-only models:
  - Stack of decoder layers
  - Masked self-attention and feed-forward networks
  - Used for tasks like language modeling and generation
- Encoder-decoder models:
  - Encoder stack processes input sequence
  - Decoder stack attends to encoder output and generates output
  - Used for tasks like translation and summarization

Summary

Spectral Clustering

- Clustering based on graph Laplacian eigenvectors
- Partitions data using graph cut objectives
- Relies on the spectrum of the graph Laplacian matrix
- Algorithms: unnormalized, normalized spectral clustering

Graph Neural Networks

- Neural networks operating on graph-structured data
- Aggregate neighborhood information to update node representations
- Variants: convolutional, attentional, message-passing
- Applications: node classification, link prediction, graph classification

Transformers

- Sequence-to-sequence models based on attention mechanisms
- Architectures: encoder-only, decoder-only, encoder-decoder
- Key components: tokenization, attention mechanisms, positional encoding
- Applications: language modeling, machine translation, text generation