Graph Neural Networks

Alejandro Ribeiro
Electrical and Systems Engineering, University of Pennsylvania
aribeiro@seas.upenn.edu

Thanks: Joan Bruna, Luiz Chamon, Fernando Gama, Gabriel Egan, Daniel Lee, Antonio Garcia Marques, Mark Eisen, Elvin Isufi, Arbaaz Khan, Vijay Kumar, Geert Leus, George Pappas, Jimmy Paulos, Luana Ruiz, Santiago Segarra, Kate Tolstaya, Clark Zhang

Support: ARO W911NF1710438, Intel ISTC-WAS, NSF CCF 1717120, ARL DCIST CRA W911NF-17-2-0181

September 6, 2019

Machine Learning for Graph Signals

Authorship Attribution

Learning Decentralized Controllers in Distributed Systems

Learning Optimal Resource Allocations in Wireless Communications Networks

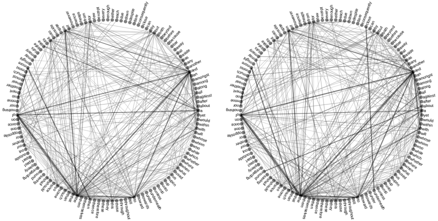
Invariance and Stability Properties of Graph Neural Networks

Concluding Remarks

Machine Learning on Graphs

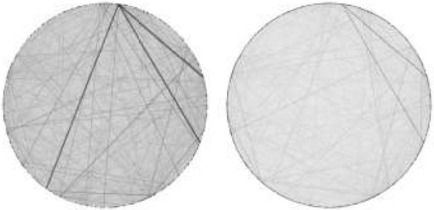
- Graphs are generic models of signal structure that can help to learn in several practical problems

Authorship Attribution



Segarra et al ’16, doi.org/10.1353/shq.2016.0024

Recommendation Systems

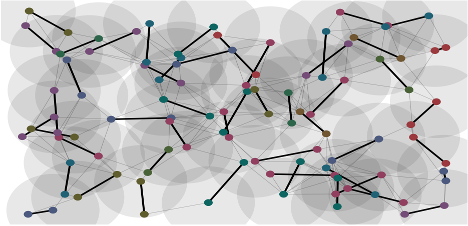


Ruiz et al ’18, arxiv.org/abs/1903.12575

- Graphs are generic models of signal structure that can help to learn in several practical problems

Decentralized Control of Autonomous Systems

Wireless Networks (Eisen et al '19)

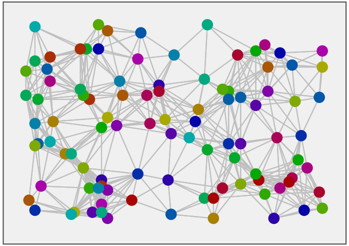


Tolstaya et al '19, arxiv.org/abs/1903.10527

Eisen-Ribeiro '19, arxiv.org/abs/1909.01865

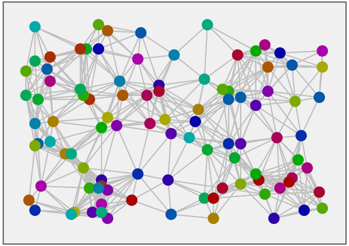
- There is overwhelming empirical and theoretical justification to choose a neural network (NN)

Challenge is we want to run a NN over this

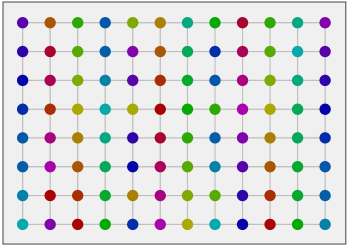


- There is overwhelming empirical and theoretical justification to choose a neural network (NN)

Challenge is we want to run a NN over this

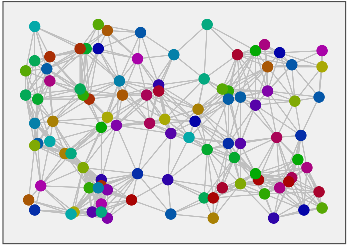


But we are good at running NNs over this

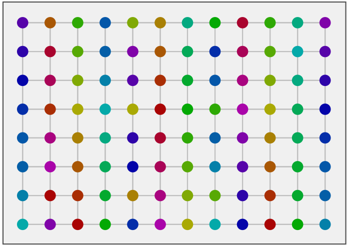


- There is overwhelming empirical and theoretical justification to choose a neural network (NN)

Challenge is we want to run a NN over this



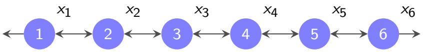
But we are good at running NNs over this



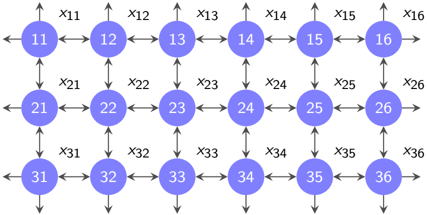
- Generalize convolutions to graphs and compose graph filters with pointwise nonlinearities

- We can describe discrete time and space using graphs that support time or space signals

Description of time with a line graph



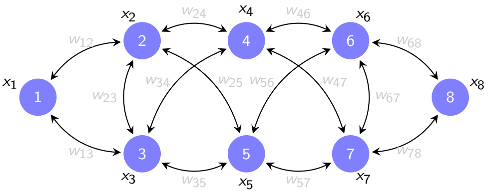
Description of images (space) with a grid graph



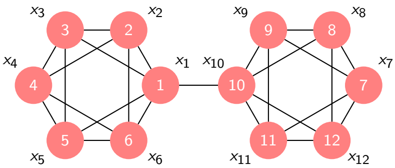
- Either convolution is a polynomial on the respective adjacency matrix $\Rightarrow \mathbf{z} = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$

- In general, we can describe signals with arbitrary structure with a suitable graph
  - $\Rightarrow$ With edges that represent an expectation of similarity between components of the signal

A signal supported on a graph



Another signal supported on another graph



- Again, convolution is a polynomial on the respective adjacency matrix $\Rightarrow \mathbf{z} = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$

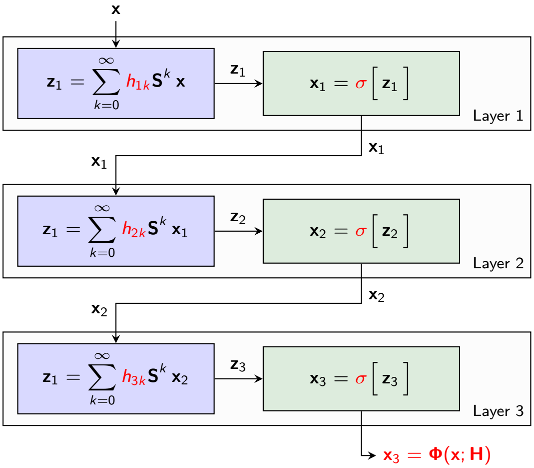
Convolutional Neural Networks (CNNs)

- Compose a cascade of layers

- Themselves compositions of chosen convolutional filters with pointwise nonlinearities

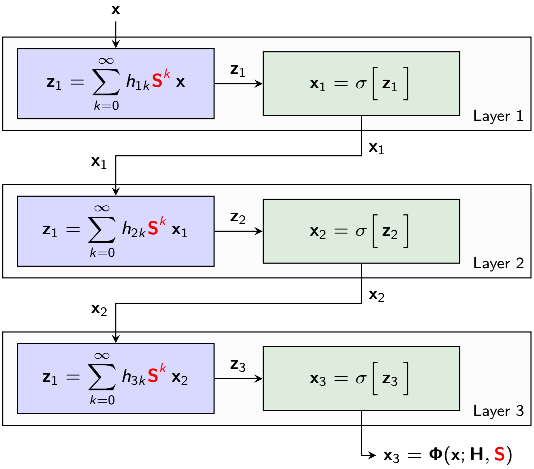
- Output is a function of filter tensor $\mathbf{H}$

- A CNN is a minor variation of a convolutional filter. Just add nonlinearity and stir



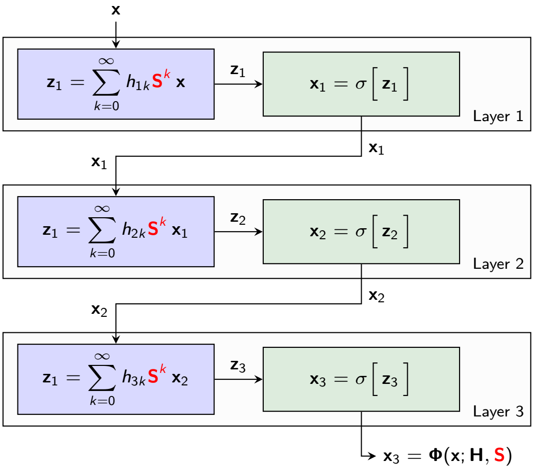
, ,

- Compose a cascade of layers
- Themselves compositions of chosen graph filters with pointwise nonlinearities
- Output is a function of filter tensor $\mathbf{H}$
- A GNN is a minor variation of a graph filter. Just add nonlinearity and stir



Gama-Marques-Leus-Ribeiro, *Convolutional Neural Network Architectures for Signals Supported on Graphs*, TSP 2019, arxiv.org/abs/1805.00165

- It's the same thing $\Rightarrow$ We just redefined what it means to do a convolution
- Output is a function of graph shift operator $\mathbf{S}$
- In practice we use multiple features per layer, pooling and readout layers
  - $\Rightarrow$ But just for polishing around the edges



Gama-Marques-Leus-Ribeiro, *Convolutional Neural Network Architectures for Signals Supported on Graphs*, TSP 2019, arxiv.org/abs/1805.00165

Machine Learning for Graph Signals

Authorship Attribution

Learning Decentralized Controllers in Distributed Systems

Learning Optimal Resource Allocations in Wireless Communications Networks

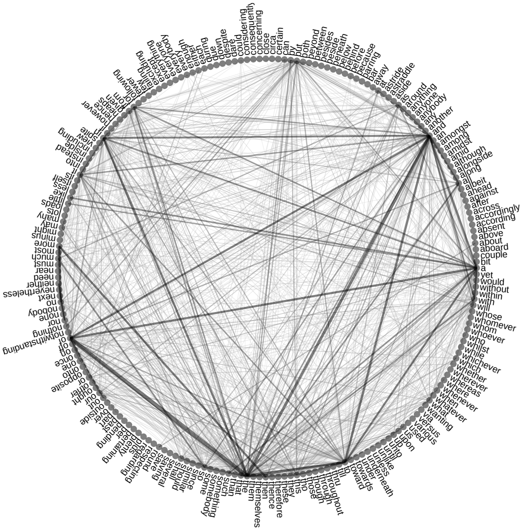
Invariance and Stability Properties of Graph Neural Networks

Concluding Remarks

- Function words are those that don't carry meaning
- Their use depends on the language's grammar
- Different authors use slightly different grammar
- Capture with a word adjacency network (WAN)

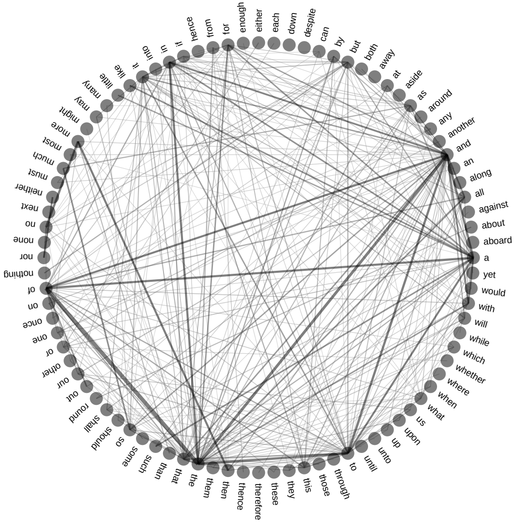
  ⇒ How often pairs of words appear together

Segarra-Eisen-Ribeiro, *Authorship Attribution through Function Word Adjacency Networks*, TSP 2015, arxiv.org/abs/1805.00165

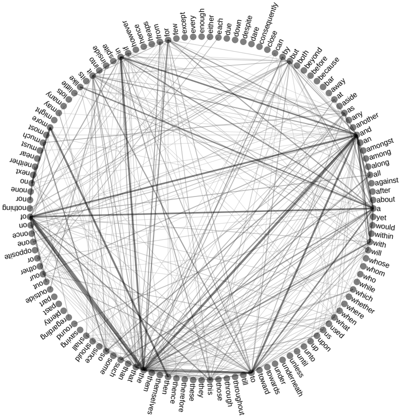


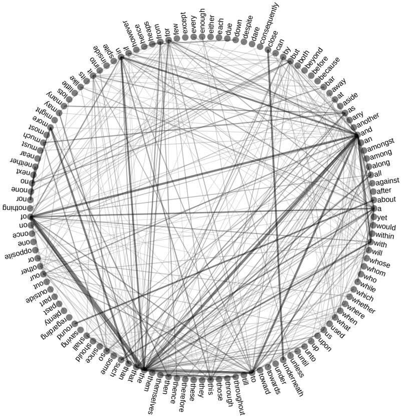
- Function words are those that don't carry meaning
- Their use depends on the language's grammar
- Different authors use slightly different grammar
- Capture with a word adjacency network (WAN)
  - ⇒ How often pairs of words appear together

Segarra-Eisen-Ribeiro, *Authorship Attribution through Function Word Adjacency Networks*, TSP 2015, arxiv.org/abs/1805.00165



- Shakespeare's and Marlowe's WANs are sufficiently different to ascertain their collaboration on Henry VI

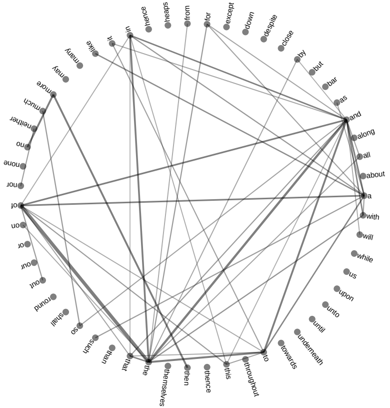


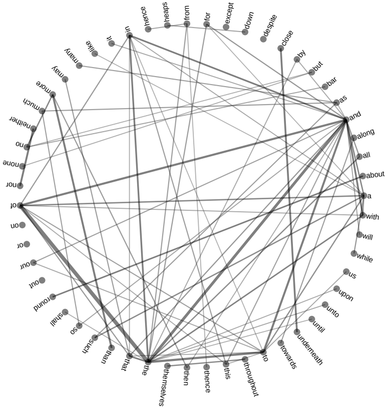


Segarra-Eisen-Egan-Ribeiro, *Attributing the Authorship of the Henry VI Plays by Word Adjacency*, Shakespeare Quarterly 2016, doi.org/10.1353/shq.2016.0024

Henry the VI is a Collaborative Effort by Shakespeare and Marlowe

- Shakespeare’s and Marlowe’s WANs are sufficiently different to ascertain their collaboration on Henry VI



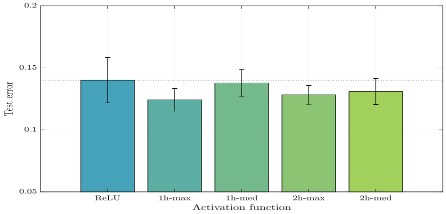


Segarra-Eisen-Egan-Ribeiro, *Attributing the Authorship of the Henry VI Plays by Word Adjacency*, Shakespeare Quarterly 2016, doi.org/10.1353/shq.2016.0024

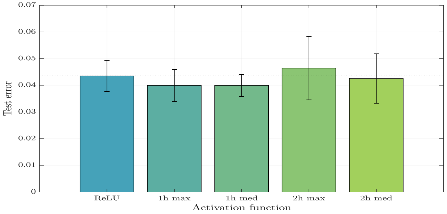
- When texts are long we can attribute by comparing word networks of different texts
- When texts are short comparing networks is unreliable

  $\Rightarrow$ Compare histograms of different texts defined as graph signals over WANs

- Pickup pages (1K words) written by E. Brontë or J. Austen from a pool of 22 contemporaries



Emily Brönte



Jane Austen

- Different GNN architectures all achieve good error rates $\Rightarrow \sim 12\%$ (Brönte) and $\sim 4\%$ (Austen)

Ruiz-Gama-Marques-Ribeiro, *Invariance-Preserving Localized Activation Functions for Graph Neural Networks*, arxiv.org/abs/1903.12575

Machine Learning for Graph Signals

Authorship Attribution

Learning Decentralized Controllers in Distributed Systems

Learning Optimal Resource Allocations in Wireless Communications Networks

Invariance and Stability Properties of Graph Neural Networks

Concluding Remarks

Decentralized Coordination of a Robot Swarm

- We want the team to coordinate on their individual velocities without colliding with each other

- This is a very easy problem to solve if we allow for centralized coordination $\Rightarrow \mathbf{u}_i = \sum_{i=1}^{N} \mathbf{v}_i$
- But it is very difficult to solve if we do do not allow for centralized coordination $\Rightarrow \mathbf{u}_i = \ldots$

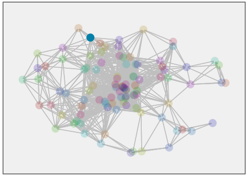
Tolstaya-Gama-Paulos-Pappas-Kumar-Ribeiro, *Learning Decentralized Controllers for Robot Swarms with Graph Neural Networks*, arxiv.org/abs/1909.01865

- Or, the team has to maintain a formation while they fly without colliding and tolerating wind

- This is also an easy to solve problem with centralized coordination and difficult to solve without

Khan-Tolstaya-Kumar-Ribeiro, *Graph Policy Gradients for Large Scale Robot Control*, arxiv.org/abs/1907.03822

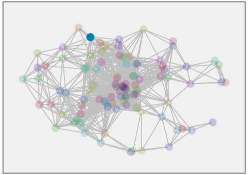
- The challenge in designing behaviors for distributed systems is the partial information structure



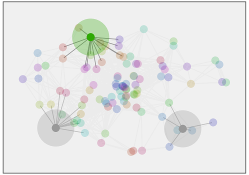
$\mathbf{x}_{i(n)}$

- Node i has access to its own local information at time n $\Rightarrow \mathbf{x}_{i(n)}$

- The challenge in designing behaviors for distributed systems is the partial information structure



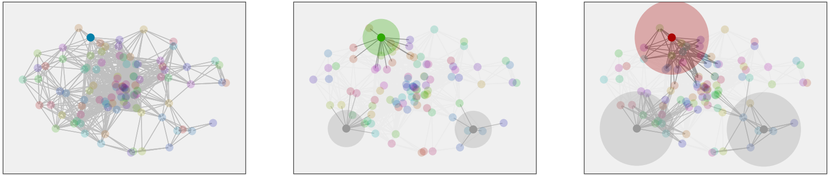
$\mathbf{x}_{i(n)}$



$\mathbf{x}_{j(n-1)}$ for $j \in \mathcal{N}_i^1$

- Node i has access to its own local information at time $n \Rightarrow \mathbf{x}_{i(n)}$
- And the information of its 1-hop neighbors at time $n-1 \Rightarrow \mathbf{x}_{j(n-1)}$ for all $j \in \mathcal{N}_i^1$

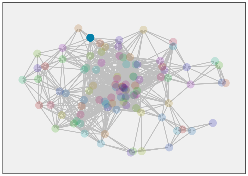
- The challenge in designing behaviors for distributed systems is the partial information structure



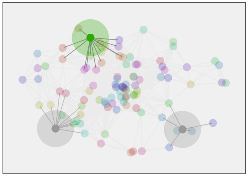
$\mathbf{x}_{i(n)}$ $\quad$ $\mathbf{x}_{j(n-1)}$ for $j \in \mathcal{N}_i^1$ $\quad$ $\mathbf{x}_{j(n-2)}$ for $j \in \mathcal{N}_i^2$

- Node i has access to its own local information at time $n \Rightarrow \mathbf{x}_{i(n)}$
- And the information of its 1-hop neighbors at time $n-1 \Rightarrow \mathbf{x}_{j(n-1)}$ for all $j \in \mathcal{N}_i^1$
- And the information of its 2-hop neighbors at time $n-2 \Rightarrow \mathbf{x}_{j(n-2)}$ for all $j \in \mathcal{N}_i^2$

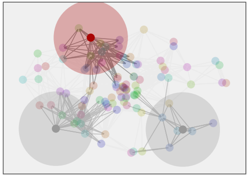
- The challenge in designing behaviors for distributed systems is the partial information structure



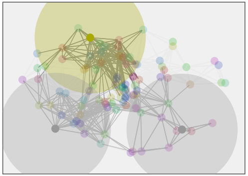
$\mathbf{x}_{i(n)}$



$\mathbf{x}_{j(n-1)}$ for $j \in \mathcal{N}_i^1$



$\mathbf{x}_{j(n-2)}$ for $j \in \mathcal{N}_i^2$



$\mathbf{x}_{j(n-3)}$ for $j \in \mathcal{N}_i^3$

- Node i has access to its own local information at time n $\Rightarrow \mathbf{x}_{i(n)}$
- And the information of its 1-hop neighbors at time $n-1$ $\Rightarrow \mathbf{x}_{j(n-1)}$ for all $j \in \mathcal{N}_i^1$
- And the information of its 2-hop neighbors at time $n-2$ $\Rightarrow \mathbf{x}_{j(n-2)}$ for all $j \in \mathcal{N}_i^2$
- And the information of its 3-hop neighbors at time $n-3$ $\Rightarrow \mathbf{x}_{j(n-3)}$ for all $j \in \mathcal{N}_i^3$

- Control actions can only depend on information history $\Rightarrow \mathcal{H}_{in} = \bigcup_{k=0}^{K-1} \left\{ \mathbf{x}_{j(n-k)} : j \in \mathcal{N}_i^k \right\}$

- Optimal controller is famously difficult to find. Even for very simple linear systems

  $\Rightarrow$ Witsenhausen, H. "A counterexample in stochastic optimum control" (February 1968)

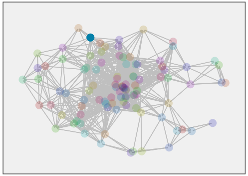
- When optimal solutions are out of reach we resort to heuristics $\Rightarrow$ data driven heuristics

- The centralized optimal control policy $\pi^*(\mathbf{x}_n)$ can be computed during training time
- Introduce parametrization and learn decentralized policy that imitates centralized policy

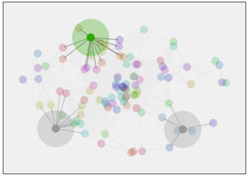
$$\mathbf{H}^* = \underset{\mathbf{H}}{\operatorname{argmin}}\, \mathbb{E}^{\pi^*}\Big[\mathcal{L}\Big(\pi\big(\mathcal{H}_{in}, \mathbf{H}\big), \pi^*(\mathbf{x}_n)\Big)\Big]$$

- Need parametrization $\mathbf{H}$ adapted to the information structure $\mathcal{H}_{in}$ $\Rightarrow$ Graph filters and GNNs

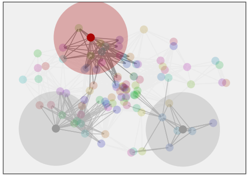
The Aggregation Sequence (Diffusion Sequence)



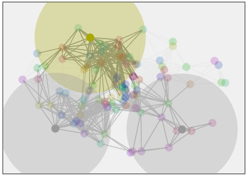
$\mathbf{y}_{0n} = \mathbf{x}_n$



$\mathbf{y}_{1n} = \mathbf{S}\mathbf{y}_{0(n-1)}$



$\mathbf{y}_{2n} = \mathbf{S}\mathbf{y}_{1(n-2)}$



$\mathbf{y}_{3n} = \mathbf{S}\mathbf{y}_{2(n-3)}$

- Aggregate information at nodes through successive averaging with graph adjacency $\mathbf{S}$

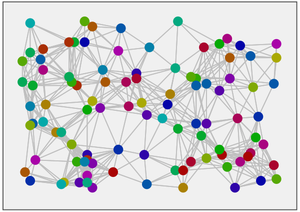
$$\mathbf{y}_{kn} = \mathbf{S}\mathbf{y}_{(k-1)(n-1)} \quad \Rightarrow \quad \Big[\mathbf{y}_{kn}\Big]_i = \Big[\mathbf{S}\mathbf{y}_{k-1(n-1)}\Big]_i = \sum_{j=1, j\in\mathcal{N}_{in}} \Big[\mathbf{S}\Big]_{ij}\Big[\mathbf{y}_{k-1n-1}\Big]_j$$

- Computed with local operations that respect information structure of distributed system

- GNNs operate on the diffusion sequence $\Rightarrow$ Which respects the partial information structure $\mathcal{H}_{in}$
- From the perspective of an individual node, the processing of the aggregation sequence is such that
  - $\Rightarrow$ If two agents observe the same input
  - $\Rightarrow$ Their K-hop neighbors observe the same inputs
  - $\Rightarrow$ And the local neighborhood structures of the graph are the same
- Then the output of the control policy is the same at both nodes. As it should. Or not.
- Aggregation GNN is permutation covariant $\Rightarrow$ Permute graph and input $\equiv$ permute output
- Permutation covariance is not a choice. It is a necessity for offline training

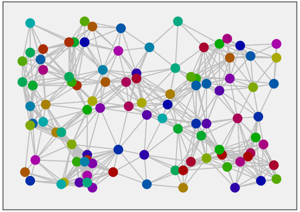
- If we want to train offline and execute online we can't assume the graph is the same

- Train online on a graph like this

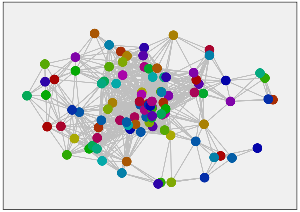


- If we want to train offline and execute online we can't assume the graph is the same

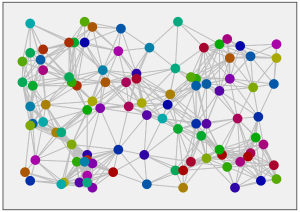
- Train online on a graph like this



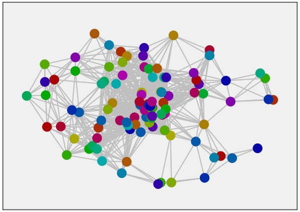
- And execute offline on a graph like this



- If we want to train offline and execute online we can't assume the graph is the same
- Train online on a graph like this



- And execute offline on a graph like this



- GNNs run on the aggregation sequence $\Rightarrow$ It can be ran independently of graph's structure.
- Permutation covariance says that if graphs are similar GNN outputs will be similar.
- Thereby producing similar policies and ensuring generalization across different graphs

- The GNN learns to imitate the central policy. Outperforms existing distributed control methods

- It transfers $\Rightarrow$ All of these different spatial configurations are using the same GNN tensor

Machine Learning for Graph Signals

Authorship Attribution

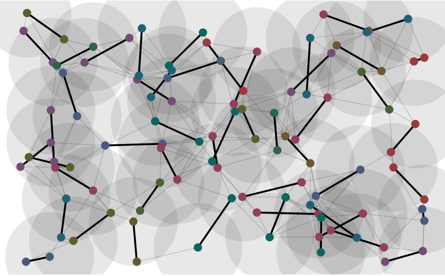
Learning Decentralized Controllers in Distributed Systems

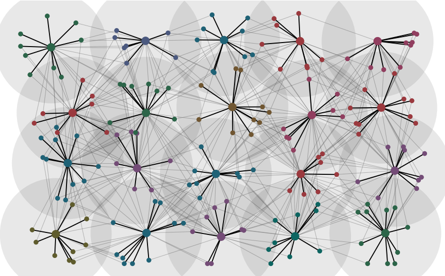
Learning Optimal Resource Allocations in Wireless Communications Networks

Invariance and Stability Properties of Graph Neural Networks

Concluding Remarks

- Groups of communicating pairs. Either ad-hoc network pairs. Or user-base station pairs





- There is interference crosstalk from other communicating pairs that we can describe with a graph

Eisen-Ribeiro, *Optimal Wireless Resource Allocation with Random Edge Graph Neural Networks*, arxiv.org/abs/1909.01865

- Pairs communicate over a time varying fading channel. Pair i is interfered by neighboring pairs j

  $\Rightarrow$ Channel is h_i for communicating pair i. Transmitter allocates power $p_i(\mathbf{h})$. $\mathbf{h} = [h_1; \ldots; h_n]$

  $\Rightarrow$ Channel crosstalk from pair j to receiver of pair i is h_{ji}. Nonzero when $j \in n(i)$

- We want to select a power allocation that maximizes communication rates in some sense

$$\mathbf{p}^*(\mathbf{h}) = \underset{\mathbf{p}(\mathbf{h})}{\operatorname{argmax}}\ \mathbb{E}_{\mathbf{h}} \left[\sum_i \log \left(1 + \frac{h_i p_i(\mathbf{h})}{1 + \sum_{j \in n(i)} h_{ij} p_j(\mathbf{h})} \right) \right]$$

- This is a problem we know well. We can't solve it exactly but we can approximate it (WMMSE)

- There are two drawbacks to the use of WMMSE and other model based solutions

  ⇒ We can model the rate function but there is a mismatch between reality and model

$$c_i = \log\left(1 + \frac{h_i p_i(\mathbf{h})}{1 + \sum_{j\in n(i)} h_{ij} p_j(\mathbf{h})}\right) + f(\mathbf{h}, \mathbf{p}(\mathbf{h}))$$

  ⇒ We modularize design but in reality we can have $\sim 10^3$ base stations with $\sim 10^5$ active users

- To some extent, both drawbacks can be ameliorated with an ML parametrization

  ⇒ The function f is unknown but it is possible to probe the environment to evaluate $f(\mathbf{x}, \mathbf{u})$

  ⇒ The optimization problem is too difficult (large scale). The parametrization may make it easier

- ML parametrizations are justified in large scale wireless communications with uncertain models

Interference Channel as a Statistical Learning Problem

- Unsupervised statistical learning data to actions that minimize and expected loss

$$\mathbf{u}^*(\mathbf{x}) = \operatorname*{argmin}_{\mathbf{u}(\mathbf{x})} \mathbb{E}_{\mathbf{x}}\Big[\ f\Big(\mathbf{x},\ \mathbf{u}(\mathbf{x})\Big)\ \Big]$$

- Given fading channel we search for a power allocation that maximizes expected sum rates

$$\mathbf{p}^*(\mathbf{h}) = \operatorname*{argmax}_{\mathbf{p}(\mathbf{h})} \mathbb{E}_{\mathbf{h}}\left[\sum_i \log\left(1 + \frac{h_i p_i(\mathbf{h})}{1 + \sum_{j\in n(i)} h_{ij} p_j(\mathbf{h})}\right)\right]$$

- Fading channel $\sim$ Data. Power allocation $\sim$ classifier. Capacity function $\sim$ loss.

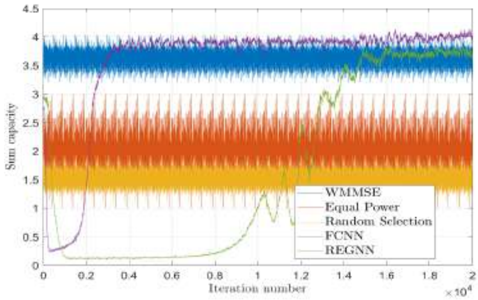
- Parametrize with a Neural Network. Or parametrize with a Graph Neural Network. Either way

$$^* = \operatorname*{argmax}_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{h}}\left[\sum_i \log\left(1 + \frac{h_i p_i(\mathbf{h},\ )}{1 + \sum_{j\in n(i)} h_{ij} p_j(\mathbf{h},\ )}\right)\right]$$

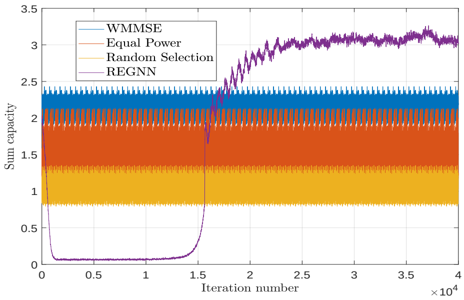
Eisen-Zhang-Chamon-Lee-Ribeiro, *Learning Optimal Resource Allocations in Wireless Systems*, TSP 2019, arxiv.org/abs/1807.08088

- GNNs, fully connected neural networks, and WMMSE performance on networks of varying size

Ad-hoc network with 20 pairs



Ad-hoc network with 50 pairs

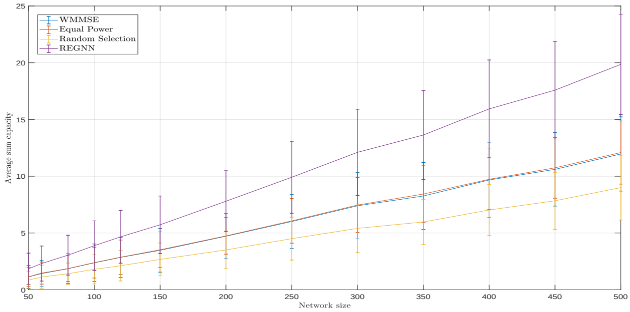


- A GNN parametrized resource allocation scales to large networks $\Rightarrow$ The only solution that scales

Eisen-Ribeiro, *Optimal Wireless Resource Allocation with Random Edge Graph Neural Networks*, arxiv.org/abs/1909.01865

- GNN built for 50 pairs generalizes to larger networks

  $\Rightarrow$ Performance of GNN trained with 50 nodes exectured on networks with up to 500 nodes



- No need for retraining $\Rightarrow$ exploits permutation equivariance of graph convolution

Machine Learning for Graph Signals

Authorship Attribution

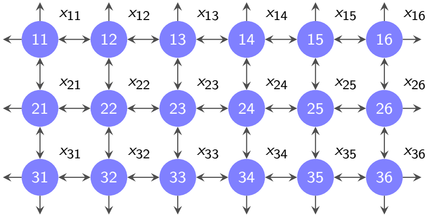
Learning Decentralized Controllers in Distributed Systems

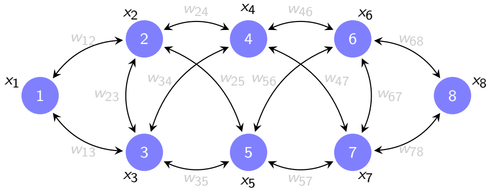
Learning Optimal Resource Allocations in Wireless Communications Networks

Invariance and Stability Properties of Graph Neural Networks

Concluding Remarks

- The engineer is satisfied. Proposed a technique. Showed it worked. But the scientist is not
- Whether on lines, grids, or arbitrary graphs we write convolutions as polynomials $\mathbf{z} = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$





- What is good about GNNs and graph filters that makes them good at machine learning on graphs?

- Given a training set of input-output example pairs $\mathcal{T}\{(\mathbf{x}, \mathbf{y})\}$ and a loss function $f(\cdot, \mathbf{y})$
- Find the best arbitrary linear regressor for the average loss $\Rightarrow \mathbf{H}^* = \underset{\mathbf{H}}{\operatorname{argmin}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} f\Big(\mathbf{H}\mathbf{x}, \mathbf{y}\Big)$
- Or, the best convolution regressor regressor $\Rightarrow \mathbf{H}^* = \underset{\mathbf{h}}{\operatorname{argmin}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} f\left(\sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}, \mathbf{y}\right)$
- The linear regressor is better than the convolution by definition. Both are linear. One is generic
  - $\Rightarrow$ This is true in the test set. In reality, the convolution does better. It generalizes
- We know why this happens $\Rightarrow$ The convolution is equivariant to time shifts
  - $\Rightarrow$ CNNs inherit this property from filters. Explaining their good performance (Mallat '12)

Graph Convolutions are Useful Because of Permutation Equivariance

- Define the graph convolution operator $\Phi(\mathbf{x}; \mathbf{S}, \mathbf{H}) = \sum_{k=0}^{\infty} h_k \, \mathbf{S}^k \, \mathbf{x}$

- Φ depends on input signal $\mathbf{x}$, graph shift operator $\mathbf{S}$ and filter tensor $\mathbf{H} = \{h_k\}_{k=0}^{\infty}$

Theorem (Gama, Ribeiro, Bruna)

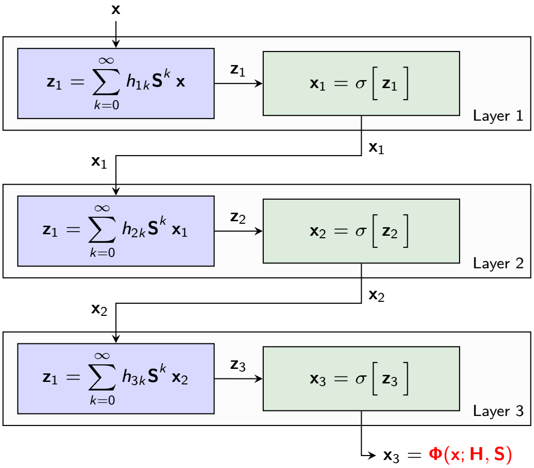
Graph convolutions are equivariant to permutations. For graphs with permuted shift operators $\hat{\mathbf{S}} = \mathbf{P}^T\mathbf{S}\mathbf{P}$ *and permuted graph signals* $\hat{\mathbf{x}} = \mathbf{P}^T\mathbf{x}$ *it holds*

$$\Phi(\hat{\mathbf{x}}; \hat{\mathbf{S}}, \mathbf{H}) = \mathbf{P}^T \Phi(\mathbf{x}; \mathbf{S}, \mathbf{H})$$

Proof $\Rightarrow$ $\Phi(\hat{\mathbf{x}}; \hat{\mathbf{S}}, \mathbf{H}) = \sum_{k=0}^{\infty} h_k \, \hat{\mathbf{S}}^k \hat{\mathbf{x}} = \sum_{k=0}^{\infty} h_k \, (\mathbf{P}^T\mathbf{S}\mathbf{P})^k \mathbf{P}^T\mathbf{x} = \mathbf{P}^T \left(\sum_{k=0}^{\infty} h_k \, \mathbf{S}^k \mathbf{x} \right) = \Phi(\mathbf{x}; \mathbf{S}, \mathbf{H})$

Gama-Bruna-Ribeiro, *Stability Properties of Graph Neural Networks*, arxiv.org/abs/1905.04497

- GNN compose a cascade of layers
- Themselves compositions of graph filters with pointwise nonlinearities
- A pointwise operation does not mix components. It's independent of the graph.
- GNN retains permutation equivariance



Theorem (Gama, Ribeiro, Bruna)

GNNs are equivariant to permutations. For graphs with permuted shift operators $\hat{\mathbf{S}} = \mathbf{P}^T\mathbf{S}\mathbf{P}$ *and permuted graph signals* $\hat{\mathbf{x}} = \mathbf{P}^T\mathbf{x}$ *it holds*

$$\Phi(\hat{\mathbf{x}}; \hat{\mathbf{S}}, \mathbf{H}) = \mathbf{P}^T\Phi(\mathbf{x}; \mathbf{S}, \mathbf{H})$$

where $\Phi(\hat{\mathbf{x}}; \hat{\mathbf{S}}, \mathbf{H})$ *is the output of processing* $\hat{\mathbf{x}}$ *on* $\hat{\mathbf{S}}$ *with GNN* $\mathbf{H}$ *and* $\Phi(\mathbf{x}; \mathbf{S}, \mathbf{H})$ *is the output of processing* $\mathbf{x}$ *on* $\mathbf{S}$ *with the same GNN* $\mathbf{H}$.

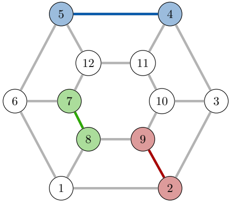
- Signal Processing with Graph Neural Networks is independent of labeling

Gama-Bruna-Ribeiro, *Stability Properties of Graph Neural Networks*, arxiv.org/abs/1905.04497

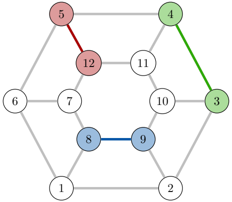
Equivariance to Permutations is More Valuable than Apparent

- Invariance to node relabelings allows GNNs to exploit internal symmetries of graph signals
- Although different, signals on (a) and (b) are permutations of one other

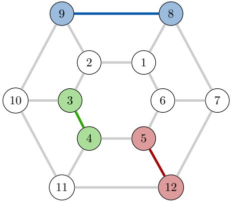
$\Rightarrow$ Permutation invariance means that the GNN can learns to classify (b) from seeing (a)



(a)



(b)



(c)

- Permutation Equivariance is not a good idea in all problems $\Rightarrow$ Edge-Variant GNNs

Isufi-Gama-Ribeiro, *Generalizing Graph Convolutional Neural Networks with Edge-Variant Recursions on Graphs*, arxiv.org/abs/1903.01298

Equivariance to Permutations is a Property of Graph Filters

- Permutation equivariance is a property of graph convolutions inherited to GNNs
  - $\Rightarrow$ Q1: What is good about pointwise nonlinearities?
  - $\Rightarrow$ Q2: What is wrong with linear graph convolutions?
- A2: They can be unstable to perturbations of the graph if we push their discriminative power
- A1: They can be made stable to perturbations while retaining discriminability
- Beautifully, these questions can be answered with an analysis in the spectral domain

- Graph convolution is a polynomial on the shift operator $\Rightarrow \mathbf{y} = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$

- Decompose operator as $\mathbf{S} = \mathbf{V}^H \mathbf{\Lambda} \mathbf{V}$ to write the spectral representation of the graph convolution

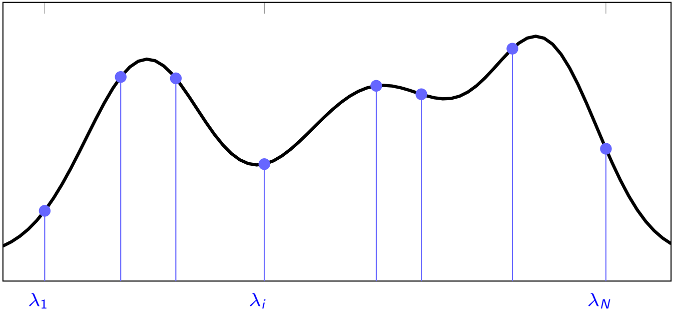
$$\mathbf{V}^H \mathbf{y} \ = \ \sum_{k=0}^{\infty} h_k (\mathbf{V}^H \mathbf{S} \mathbf{V})^k \ \mathbf{V}^H \mathbf{x} \qquad \Rightarrow \qquad \tilde{\mathbf{y}} \ = \ \sum_{k=0}^{\infty} h_k \mathbf{\Lambda}^k \ \tilde{\mathbf{x}}$$

- where we have used the graph Fourier transform (GFT) definitions $\tilde{\mathbf{x}} = \mathbf{V}^H \mathbf{x}$ and $\tilde{\mathbf{y}} = \mathbf{V}^H \mathbf{y}$

- Graph convolution is a pointwise operation in the spectral domain

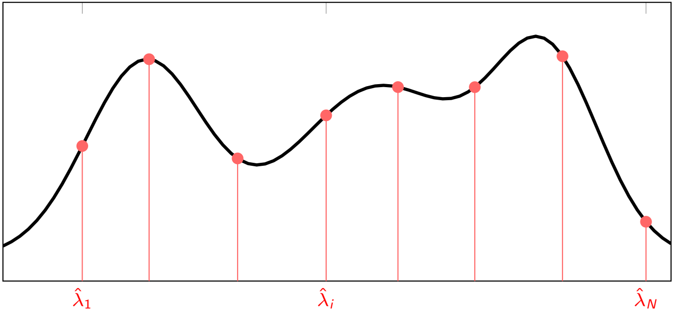
$\Rightarrow$ Determined by the (graph) frequency response $\Rightarrow \sum_{k=0}^{\infty} h_k \lambda_k^k$

- We can reinterpret the frequency response as a polynomial on continuous λ $\Rightarrow$ $\tilde{h}(\lambda) = \sum_{k=0}^{\infty} h_k \lambda^k$



- Frequency response is the same no matter the graph $\Rightarrow$ It's instantiated on its particular spectrum

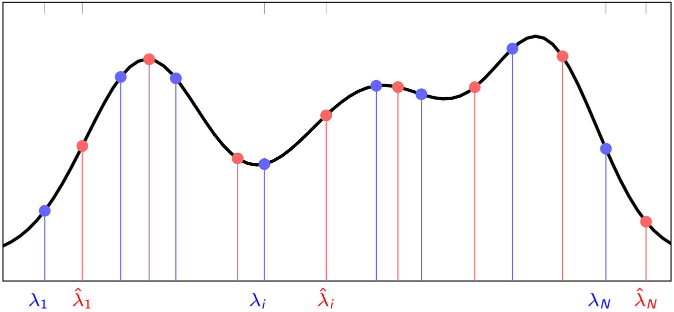
- We can reinterpret the frequency response as a polynomial on continuous λ $\Rightarrow$ $\tilde{h}(\lambda) = \sum_{k=0}^{\infty} h_k \lambda^k$



- Frequency response is the same no matter the graph $\Rightarrow$ It's instantiated on its particular spectrum

Graph Frequency Response

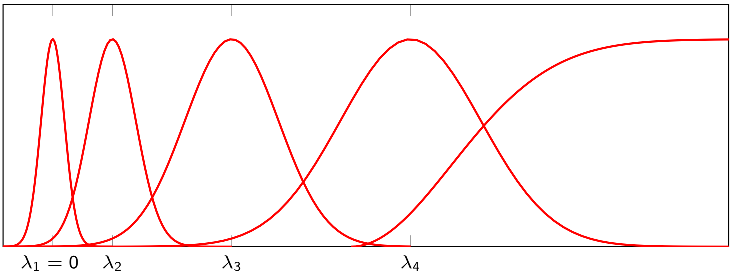
- We can reinterpret the frequency response as a polynomial on continuous λ $\Rightarrow \tilde{h}(\lambda) = \sum_{k=0}^{\infty} h_k \lambda^k$



- Frequency response is the same no matter the graph $\Rightarrow$ It's instantiated on its particular spectrum

Restricting the Class of Allowable Filters

- Let $h(\lambda)$ be the frequency response of filter $\mathbf{H}$. We say $\mathbf{H}$ is integral Lipschitz if $|\lambda h'(\lambda)| \leq C$



- Integral Lipschitz filters have to be wide for large λ $\Rightarrow$ They can't discriminate
- But they can be thin for low λ $\Rightarrow$ They can discriminate. Arbitrarily discriminate

- To measure graph perturbations introduce a relative perturbation model $\Rightarrow \hat{\mathbf{S}} = \mathbf{E}^H\mathbf{S} + \mathbf{S}\mathbf{E}$

- Since graphs $\mathbf{S}$ and $\hat{\mathbf{S}}$ are identical if they are permutations of each other define the set

$$\mathcal{E} = \Big\{ \mathbf{E} : \mathbf{P}^T\hat{\mathbf{S}}\mathbf{P} \;=\; \mathbf{E}^H\mathbf{S} + \mathbf{S}\mathbf{E} \text{ for some } \mathbf{P} \in \mathcal{P} \Big\} \qquad \mathcal{P} = \text{set of permutation matrices}$$

- Smallest $\mathbf{E}$ matrix that maps $\mathbf{S}$ into a permutation of $\hat{\mathbf{S}}$ is the relative distance between $\mathbf{S}$ and $\hat{\mathbf{S}}$

$$d(\mathbf{S}, \hat{\mathbf{S}}) \;=\; \min_{\mathbf{E}\in\mathcal{E}} \|\mathbf{E}\|$$

- This distance measures relative differences between graphs modulo permutations

  $\Rightarrow$ In particular, it is small if the graphs are close to being permutations of each other

- The nonlinearity σ is said to be normalized Lipschitz if $\Rightarrow \|\sigma_\ell(\mathbf{x}) - \sigma(\hat{\mathbf{x}})\| \leq \|\mathbf{x} - \hat{\mathbf{x}}\|$

Theorem

Consider a GNN with L layers having integral Lipschitz filter $\mathbf{H}_\ell$ and normalized Lipschitz nonlinearities σ_ℓ. Graphs $\mathbf{S}$ and $\hat{\mathbf{S}}$ are such that their relative distance satisfies $d(\mathbf{S}, \hat{\mathbf{S}}) \leq \epsilon/2$ and the matrix $\mathbf{E}$ that achieves minimum distance satisfies $\|\mathbf{E}/\|\mathbf{E}\| - \mathbf{I}\| \leq \epsilon$. It holds that for all signals $\mathbf{x}$

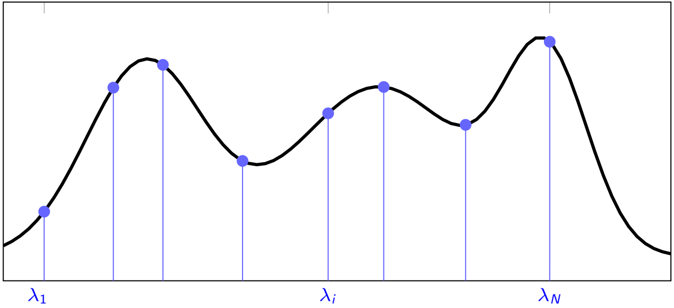
$$\min_{\mathbf{P}\in\mathcal{P}} \|\Phi(\hat{\mathbf{x}}; \hat{\mathbf{S}}, \mathbf{H}) - \mathbf{P}^T\Phi(\mathbf{x}; \mathbf{S}, \mathbf{H})\| \leq CL\epsilon + \mathcal{O}(\varepsilon^2)$$

- GNNs can be made stable to graph perturbations if filters are integral Lipschitz

- Requires validity of the structural perturbation constraint $\|\mathbf{E}/\|\mathbf{E}\| - \mathbf{I}\| \leq \epsilon$

Gama-Bruna-Ribeiro, *Stability Properties of Graph Neural Networks*, arxiv.org/abs/1905.04497

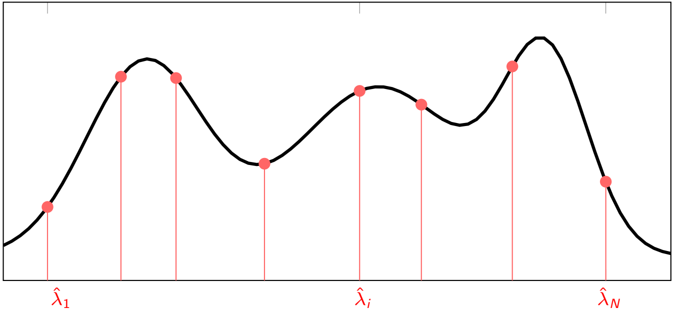
- The GNN stability theorem is elementary to prove for an edge dilation $\Rightarrow$ multiply edges by $\alpha \approx 1$
- An edge dilation just produces a spectrum dilation $\Rightarrow$ If $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^H$ then $\hat{\mathbf{S}} = \mathbf{V}(\alpha\mathbf{\Lambda})\mathbf{V}^H$



- Small deformations may result in large filter variations for large λ if filter is not integral Lipschitz

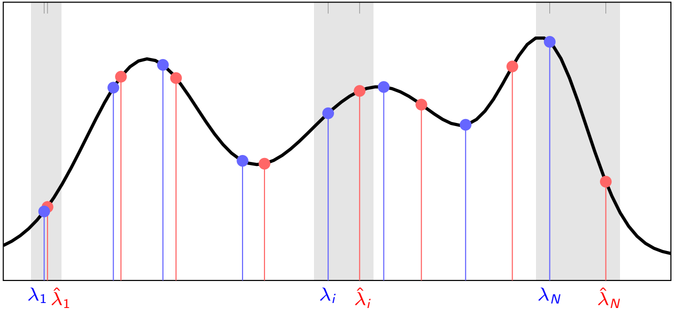
Proof of Stability Theorem

- The GNN stability theorem is elementary to prove for an edge dilation $\Rightarrow$ multiply edges by $\alpha \approx 1$
- An edge dilation just produces a spectrum dilation $\Rightarrow$ If $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^H$ then $\hat{\mathbf{S}} = \mathbf{V}(\alpha\mathbf{\Lambda})\mathbf{V}^H$



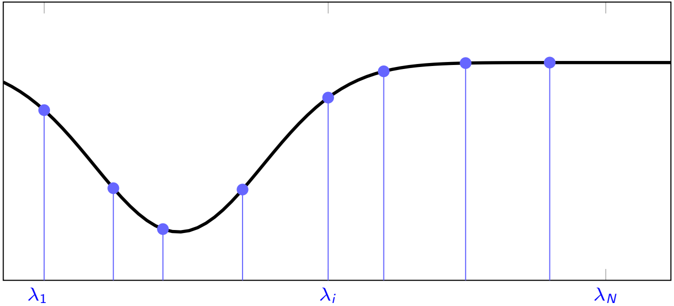
- Small deformations may result in large filter variations for large λ if filter is not integral Lipschitz

- The GNN stability theorem is elementary to prove for an edge dilation $\Rightarrow$ multiply edges by $\alpha \approx 1$
- An edge dilation just produces a spectrum dilation $\Rightarrow$ If $\mathbf{S} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^H$ then $\hat{\mathbf{S}} = \mathbf{V}(\alpha\boldsymbol{\Lambda})\mathbf{V}^H$



- Small deformations may result in large filter variations for large λ if filter is not integral Lipschitz

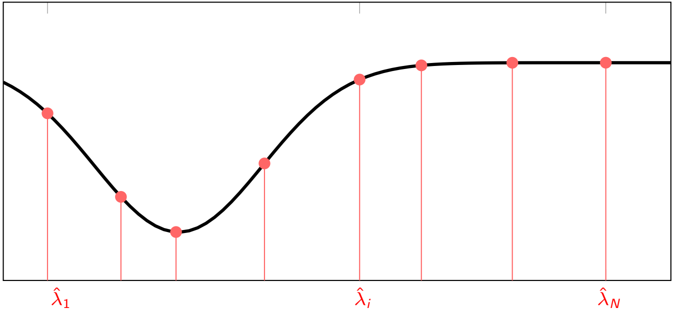
- The GNN stability theorem is elementary to prove for an edge dilation $\Rightarrow$ multiply edges by $\alpha \approx 1$
- An edge dilation just produces a spectrum dilation $\Rightarrow$ If $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^H$ then $\hat{\mathbf{S}} = \mathbf{V}(\alpha\mathbf{\Lambda})\mathbf{V}^H$



- Lipschitz filter is always stable $\Rightarrow$ Either the eigenvalue doesn't move. Or the filter doesn't move

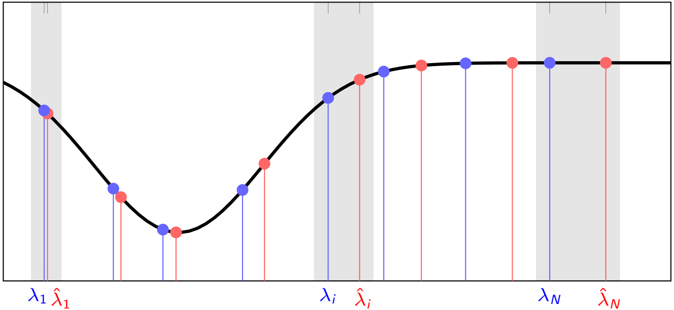
Proof of Stability Theorem

- The GNN stability theorem is elementary to prove for an edge dilation $\Rightarrow$ multiply edges by $\alpha \approx 1$
- An edge dilation just produces a spectrum dilation $\Rightarrow$ If $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^H$ then $\hat{\mathbf{S}} = \mathbf{V}(\alpha\mathbf{\Lambda})\mathbf{V}^H$



- Lipschitz filter is always stable $\Rightarrow$ Either the eigenvalue doesn't move. Or the filter doesn't move

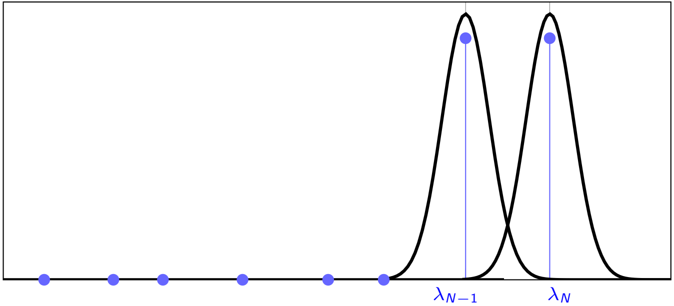
- The GNN stability theorem is elementary to prove for an edge dilation $\Rightarrow$ multiply edges by $\alpha \approx 1$
- An edge dilation just produces a spectrum dilation $\Rightarrow$ If $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^H$ then $\hat{\mathbf{S}} = \mathbf{V}(\alpha\mathbf{\Lambda})\mathbf{V}^H$



- Lipschitz filter is always stable $\Rightarrow$ Either the eigenvalue doesn't move. Or the filter doesn't move

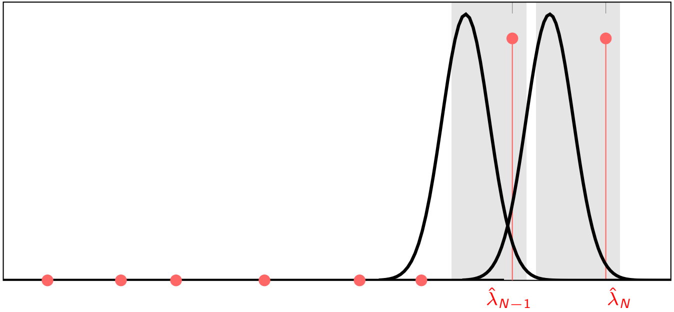
Discriminative Graph Filter Banks are Unstable

- Q2: What is wrong with linear graph convolutions?
- They can't simultaneously be stable to deformations and discriminate features at large eigenvalues



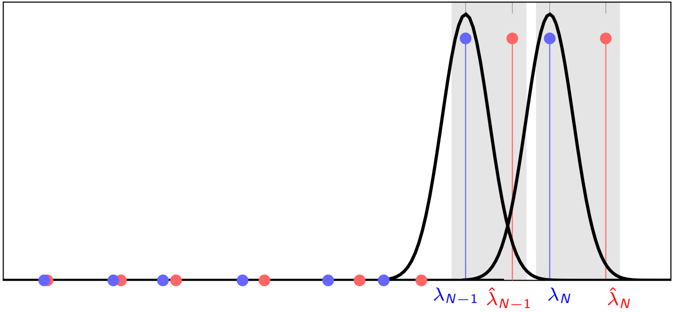
- Limits their value in machine learning problems where features at large eigenvalues are important

- Q2: What is wrong with linear graph convolutions?
- They can't simultaneously be stable to deformations and discriminate features at large eigenvalues



- Limits their value in machine learning problems where features at large eigenvalues are important

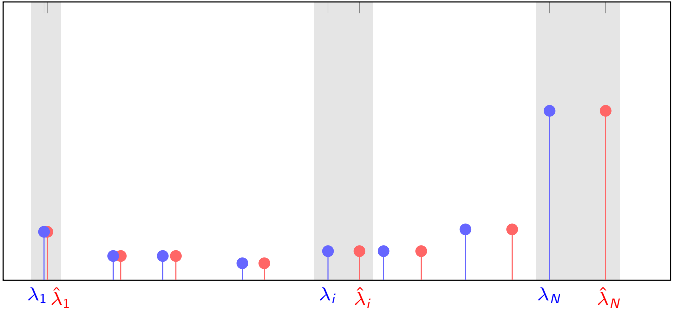
- Q2: What is wrong with linear graph convolutions?
- They can't simultaneously be stable to deformations and discriminate features at large eigenvalues



- Limits their value in machine learning problems where features at large eigenvalues are important

Nonlinearities Create Low Frequency Components

- Q1: What is good about pointwise nonlinearities?
- Preserves permutation equivariance while generating low graph frequency components
  - $\Rightarrow$ Which we can discriminate with stable filters

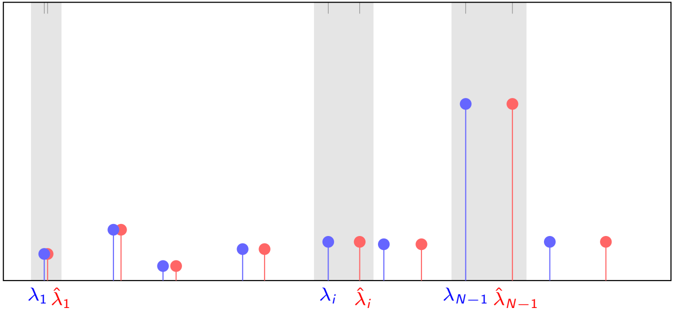


Spectrum of rectified graph signal

$$\mathbf{x}_{\text{relu}} = \max(\mathbf{x}, 0)$$

- The nonlinearity demodulates. It creates low frequency garbage. But stable garbage

- Q1: What is good about pointwise nonlinearities?
- Preserves permutation equivariance while generating low graph frequency components
  - $\Rightarrow$ Which we can discriminate with stable filters



Spectrum of rectified graph signal

$$\mathbf{x}_{\text{relu}} = \max(\mathbf{x}, 0)$$

- The nonlinearity demodulates. It creates low frequency garbage. But stable garbage

Machine Learning for Graph Signals

Authorship Attribution

Learning Decentralized Controllers in Distributed Systems

Learning Optimal Resource Allocations in Wireless Communications Networks

Invariance and Stability Properties of Graph Neural Networks

Concluding Remarks

- The promise of Machine Learning is boundless
- Machine Learning (ML) is a set of tools for solving optimization problems with data
  - $\Rightarrow$ Other names for ML are regression, pattern recognition, or statistical signal processing
- Thus, the use of ML is justified when models are unavailable or inaccurate
- Or when they are too complex so we are better off using them as generators of simulated data
- Arguably, there are no systems in which models are available, accurate, and simple

- The promise of Machine Learning is boundless

- The reality of Machine Learning is not so boundless.

- The reality of Machine Learning is not so boundless. However remarkable and impressive.

- The reality of Machine Learning is not so boundless. However remarkable and impressive.
- In 2019 Machine Learning $\equiv$ Deep Learning $\equiv$ Convolutional Neural Networks (CNNs)
  - $\Rightarrow$ If they rely on convolutions, we expect CNNs to work for Euclidean signals only (time, images)
  - $\Rightarrow$ Recent remarkable successes of Neural Networks are for image and speech processing, indeed
- Fully connected neural networks do not scale. They work for small scale problems only
- In fact, no ML method works in high dimensions if we can't exploit signal structure

(i) The promise of machine learning / statistical signal processing really is boundless

(ii) Realizing this promise requires success beyond Euclidean signals in time and space

$\Rightarrow$ True even if we just want to better our abilities in Euclidean signal processing

(iii) To succeed in non-Euclidean processing we have to operate from foundational principles

$\Rightarrow$ Humans live in Euclidean time and space. Out intuition does not necessarily carry.

(iv) The key to machine learning in non-Euclidean domains is to exploit signal structure $\Rightarrow$ A graph

- Graph Neural Networks (GNN) generalize Convolutional Neural Networks (CNN) to graph signals
  - $\Rightarrow$ They leverage structure $\Rightarrow$ Machine learning in high dimensions necessitates structure
- GNNs show particular promise in distributed collaborative intelligent systems
  - $\Rightarrow$ Data needed for their execution respects the information structure of distributed systems
- GNNs can be made discriminative and stable to deformations of the graph
  - $\Rightarrow$ A property that linear filters can't have. Explains their better performance
  - $\Rightarrow$ Analogous to stability of CNNs versus instability of convolutional filters