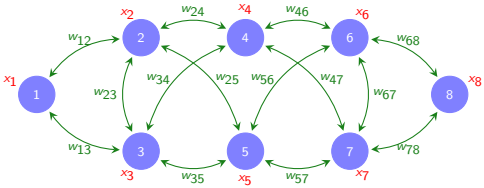
First Midterm

- We are halfway through the course. It is time for a midterm

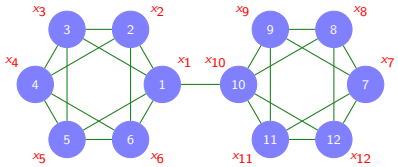
What?	$\Rightarrow$	Machine learning on graphs
Why?	$\Rightarrow$	Generic models of signal structure
		Models of distributed physical infrastructure
How?	$\Rightarrow$	Graph Neural Networks
Does this work?	$\Rightarrow$	Yes: Examples
Should this work?	$\Rightarrow$	Yes: Equivariance and Stability

- We saw several systems (and there are many more) that can be modeled with graph signals
- Use graphs as generic descriptors of signal structure with signal values associated to nodes and edges expressing expected similarity between signal components

A signal supported on a graph



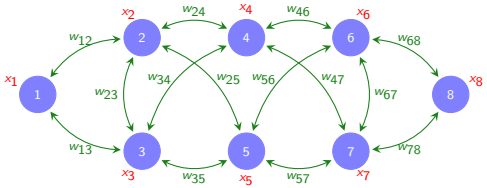
Another signal supported on another graph



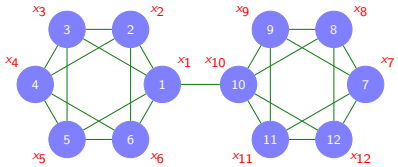
- Nodes are customers. Signal values are product ratings. Edges are cosine similarities of past scores

- We saw several systems (and there are many more) that can be modeled with graph signals
- Use graphs as generic descriptors of signal structure with signal values associated to nodes and edges expressing expected similarity between signal components

A signal supported on a graph



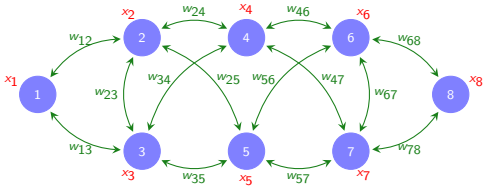
Another signal supported on another graph



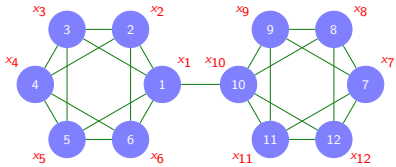
- Nodes are drones. Signal values are velocities. Edges are sensing and communication ranges

- We saw several systems (and there are many more) that can be modeled with graph signals
- Use graphs as generic descriptors of signal structure with signal values associated to nodes and edges expressing expected similarity between signal components

A signal supported on a graph

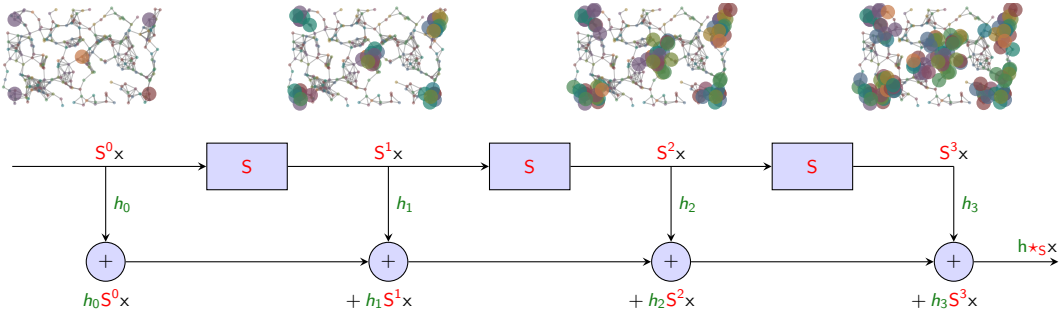


Another signal supported on another graph

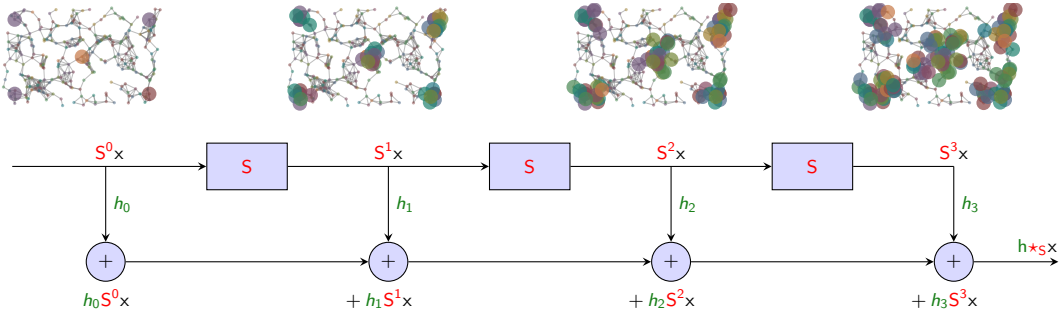


- Nodes are transceivers. Signal values are QoS requirements. Edges are wireless channels strength

- A Graph filter on shift operator S with coefficients h_k is a polynomial on S $\Rightarrow \mathrm{H}(\mathrm{S}) = \sum_{k=0}^{\infty} h_k \mathrm{S}^k$
- A convolutional filter depends on the filter coefficients h_h and the graph shift operator S



- A graph convolution is the result of applying a graph filter $\Rightarrow \mathbf{h} \star_{\mathbf{S}} \mathbf{x} = \mathbf{H}(\mathbf{S})\mathbf{x} = \left[\sum_{k=0}^{\infty} h_k \mathbf{S}^k\right]\mathbf{x}$
- A weighted linear combination of the elements of the diffusion sequence $\equiv$ Shift. Scale. Sum



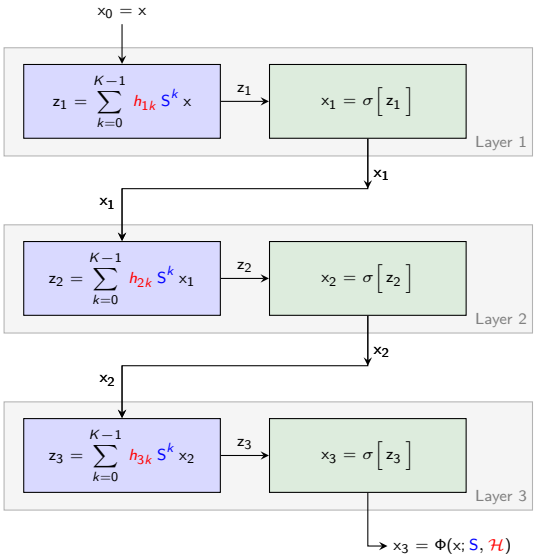
- A GNN with L layers follows L recursions of the form

$$\mathbf{x}_\ell = \sigma\Big[\mathbf{z}_\ell\Big] = \sigma\left[\sum_{k=0}^{K-1} h_{\ell k}\, \mathbf{S}^k\, \mathbf{x}_{\ell-1}\right]$$

- The output of the GNN is the output of layer L

$$\mathbf{x}_L = \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$$

- Depends on filter tensor and shift operator



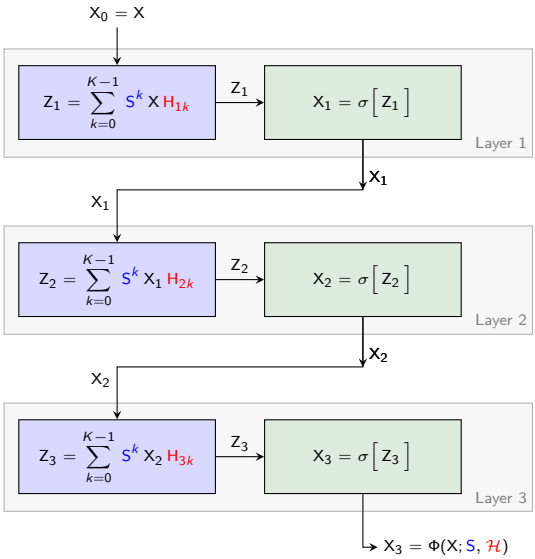
- Improve expressive power with MIMO graph filters

$$\mathbf{X}_\ell = \sigma\Big[\mathbf{Z}_\ell\Big] = \sigma\left[\sum_{k=0}^{K-1} \mathbf{S}^k \mathbf{X}_{\ell-1} \mathbf{H}_{\ell k}\right]$$

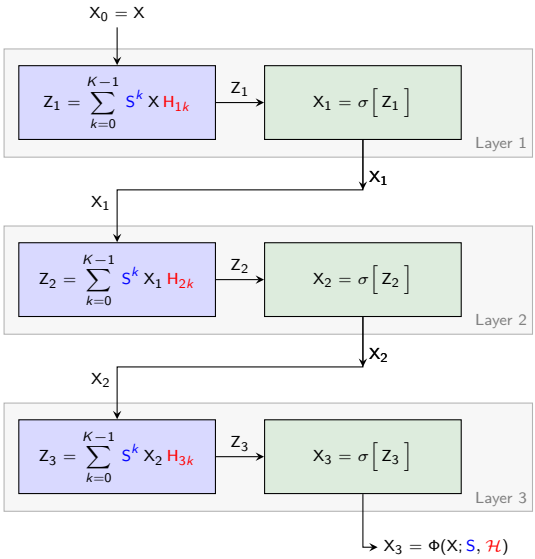
- The output of GNN is the output of layer L

$$\mathbf{X}_L = \Phi(\mathbf{X}; \mathbf{S}, \mathcal{H})$$

- Depends on filter tensor and shift operator



- Graph neural networks are...
  - ⇒ Minor variations of graph filters
    - ⇒ Pointwise nonlinearities and compositions
  - ⇒ Transferable across different graphs
  - ⇒ Generalizations of CNNs (line graph)
  - ⇒ Particularizations of fully connected NNs



Does This Work and Should This work?

- Does this work? $\Rightarrow$ Recommendation Systems. Wireless Communication Networks

- Should this work?

  $\Rightarrow$ Graph filters and GNNs are permutation equivariant. They leverage symmetries

  $\Rightarrow$ GNNs have a better stability vs discriminability tradeoff They leverage quasi-symmetries

Learning Ratings in Recommendation Systems

- Formulate recommendation systems as ERM problems that predict ratings that users give to items

- In a recommendation system, we want to predict the rating a user would give to an item
- Collect ratings that some users give to some items $\Rightarrow$ These are rating histories
- Exploit product similarities to predict ratings of unseen user-item pairs
- Example 1 $\Rightarrow$ In an online store items are products and users are customers
- Example 2 $\Rightarrow$ In a movie repository items are movies and users are watchers

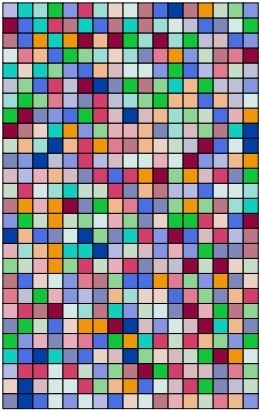
- For all items i and users u there exist ratings $\Rightarrow y_{ui}$

  $\Rightarrow$ User rating vector y_u has entries y_{ui}

- We only observe a subset of ratings $\Rightarrow x_{ui}$

  $\Rightarrow$ Observed user rating vector x_u has entries x_{ui}

  $\Rightarrow$ We assume $x_{ui} = 0$ if item i is unrated by user u



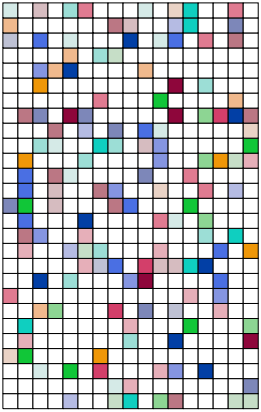
- For all items i and users u there exist ratings $\Rightarrow y_{ui}$

  $\Rightarrow$ User rating vector $\mathbf{y}_u$ has entries y_{ui}

- We only observe a subset of ratings $\Rightarrow x_{ui}$

  $\Rightarrow$ Observed user rating vector $\mathbf{x}_u$ has entries x_{ui}

  $\Rightarrow$ We assume $x_{ui} = 0$ if item i is unrated by user u



Product Ratings as Graph Signals

- Construct product similarity graph with weights w_{ij} represent likelihood of similar scores
- Interpret vector of ratings y_u of user u as a graph signal supported on the product similarity graph
- The observed ratings x_u of user u are a subsampling of this graph signal.
- Our goal is to learn to reconstruct the rating graph signal y_u from the observed ratings x_u
- Build similarity graph using available ratings. Use of expert knowledge is common as well

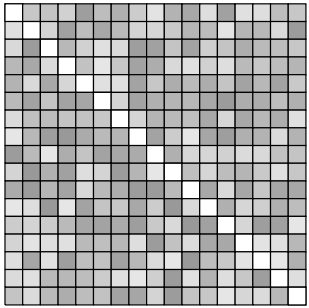
- Consider pair of products i and j. Restrict attention to set of users that rated both products $\Rightarrow \mathcal{U}_{ij}$
- Mean ratings restricted to users that rated products i and j

$$\mu_{ij} = \frac{1}{\#(\mathcal{U}_{ij})} \sum_{u \in \mathcal{U}_{ij}} x_{ui} \qquad \mu_{ji} = \frac{1}{\#(\mathcal{U}_{ij})} \sum_{u \in \mathcal{U}_{ji}} x_{uj}$$

- Similarity score = correlation restricted to users in $\mathcal{U}_{ij}$

$$\sigma_{ij} = \frac{1}{\#(\mathcal{U}_{ij})} \sum_{u \in \mathcal{U}_{ij}} \Big(x_{ui} - \mu_{ij} \Big) \Big(x_{uj} - \mu_{ji} \Big)$$

- Weights = normalized correlations $\Rightarrow w_{ij} = \sigma_{ij} \Big/ \sqrt{\sigma_{ii}\sigma_{jj}}$



- Given observed ratings x_u the AI produces estimates $\Phi(x_u)$. We want $\Phi(x_u)$ to approximate y_u

$$\ell\Big(y_u, \Phi(x_u)\Big) = \frac{1}{2}\Big\| y_u - \Phi(x_u) \Big\|^2$$

- In reality, we want to predict the rating of specific item i

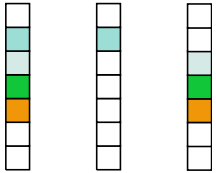
$$\ell\Big(y_u, \Phi(x_u)\Big) = \frac{1}{2}\Big(e_i^T y_u - e_i^T \Phi(x_u)\Big)^2$$

- Where e_i is a vector in the canonical basis $\Rightarrow (e_i)_i = 1$, $(e_i)_j = 0$ for $j \neq i$

- For each item i let $\mathcal{U}_i$ be the set of users that have rated i. Construct training pairs (x, y) with

$$\mathrm{y} = \left(\mathrm{e}_i^T \mathrm{x}_u\right) \mathrm{e}_i \qquad \mathrm{x} = \mathrm{x}_u - \mathrm{y} \qquad \text{for all } u \in \mathcal{U}_i, \text{ for all } i$$

- Extract the rating x_{ui} of item i. Record into graph signal y
- Remove rating x_{ui} from x_u. Record to graph signal x
- Repeat for all users in the set $\mathcal{U}_i$ of users that rated i
- Repeat for all items $\Rightarrow$ Training set $\mathcal{T}$



- Parametrized AI $\Phi(\mathbf{x}_u) = \Phi(\mathbf{x}_u; \mathcal{H})$. We want to find solution of the ERM problem

$$\mathcal{H}^* = \underset{\mathcal{H}}{\operatorname{argmin}} \sum_{(\mathbf{x},y)\in\mathcal{T}} \left(\mathbf{e}_i^T \mathbf{y} - \mathbf{e}_i^T \Phi(\mathbf{x}; \mathcal{H}) \right)^2$$

- Two bad ideas $\Rightarrow$ Linear regression. Fully connected neural networks

- Two good ideas $\Rightarrow$ Graph filters. Graph neural networks

Learning Ratings with Graph Filters and GNNs

- We use graph filters and graph neural networks to learn ratings in recommendation systems
- We contrast with the use of linear regression and fully connected neural networks

- Use MovieLens-100k as benchmark $\Rightarrow 10^6$ ratings given by $U = 943$ users to $M = 1,682$ movies
- The ratings for each movie are between 1 and 5. From one star to five starts
- Train and test several machine learning parametrizations.

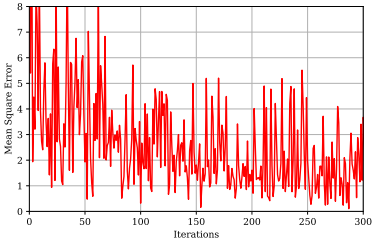
- We predict ratings using AI that results from solving the ERM problem

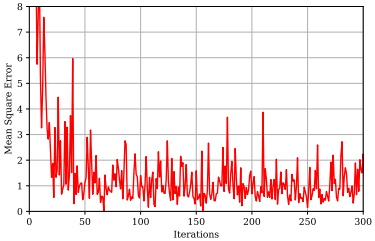
$$\mathcal{H}^* = \underset{\mathcal{H}}{\text{argmin}} \sum_{(x,y)\in\mathcal{T}} \left(\mathbf{e}_i^T \mathbf{y} - \mathbf{e}_i^T \mathbf{\Phi}(\mathbf{x}; \mathcal{H}) \right)^2$$

- Parameterizations that ignore data structure$=$ $\Rightarrow$ Linear regression. Fully connected NNs

- Parameterizations that leverage data structure$=$ $\Rightarrow$ Graph filters. Graph NNs

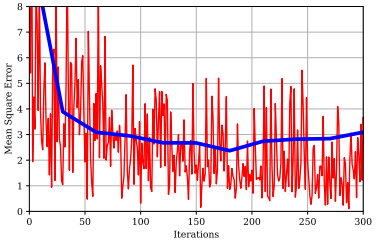
- Linear regression reduces training MSE to about 2. Quite bad for ratings that vary from 0 to 5
- Graph filter reduces training MSE to about 1. Not too good. Humans are not that predictable

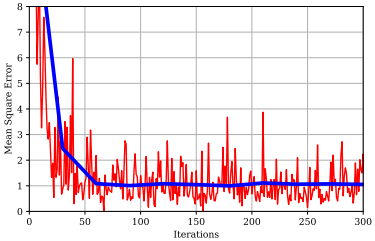




- Graph filter outperforms linear regression $\Rightarrow$ Leverages underlying permutation symmetries

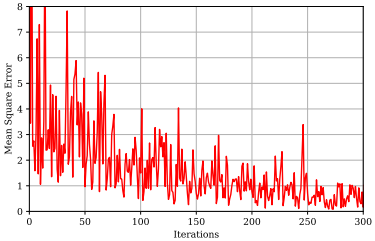
- Linear regression works even worse in the test set
- The test MSE of the graph filter is about the same as the training MSE. It generalizes

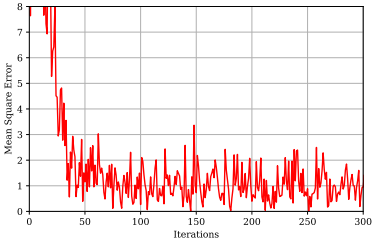




- Graph filter outperforms linear regression $\Rightarrow$ Leverages underlying permutation symmetries

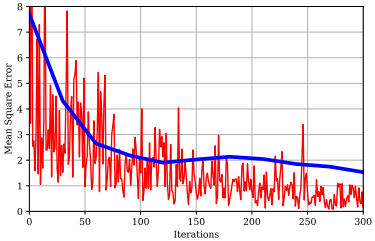
- The fully connected NN reduces the MSE to about 0.8. Looks like a great accomplishment.
- Graph NN reduces test MSE to about 0.9. Not bad. But not as good as the fully connected NN

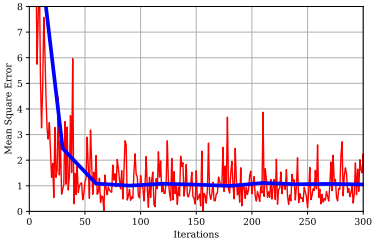




- Graph NN outperforms fully connected NN $\Rightarrow$ Leverages underlying permutation symmetries

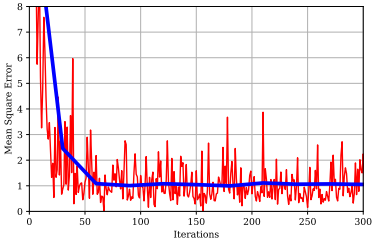
- But the fully connected NN does not do well in the test set. It does not generalize
- The test MSE of the graph NN is about the same as the training MSE. It generalizes

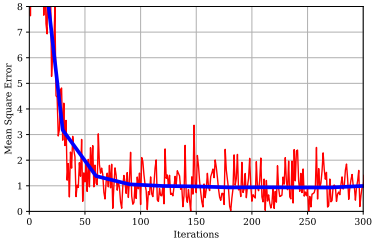




- Graph NN outperforms fully connected NN $\Rightarrow$ Leverages underlying permutation symmetries

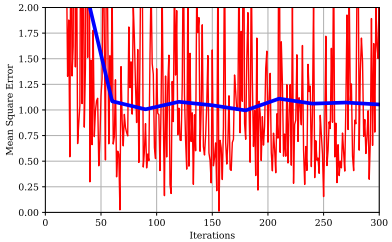
- The graph filter and the GNN do well in the training and test set. They generalize well
- The GNN does a little better. Not by much. But an extra 10% is not irrelevant

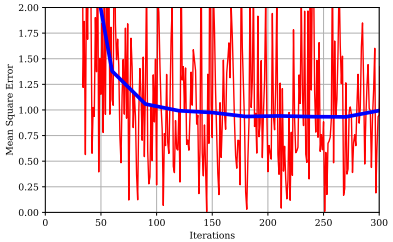




- GNN outperforms graph filter $\Rightarrow$ The GNN has a better stability-discriminability tradeoff

- The graph filter and the GNN do well in the training and test set. They generalize well
- The GNN does a little better. Not by much. But an extra 10% is not irrelevant

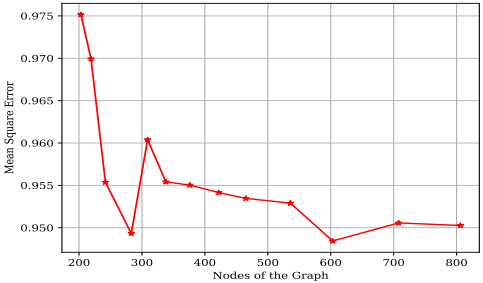




- GNN outperforms graph filter $\Rightarrow$ The GNN has a better stability-discriminability tradeoff

- A GNN can be trained on a graph with a small number of nodes ...

  $\Rightarrow$ And transferred to a graph with a (much) larger number of nodes. Without retraining



- In this recommendation system, transference incurs no MSE degradation $\Rightarrow$ MSE is further reduced

- The engineer is satisfied $\Rightarrow$ Proposed a method (GNNs). They demonstrated that it works
- The scientist is curious $\Rightarrow$ The method worsk. But they don't know why.
- This is the motivation for out analyses. Which we will proceed to review

Permutation Equivariance

- Graph filters and GNN are equivariant to permutations $\Rightarrow$ They can exploit signal symmetries

- Predicting that they should outperform linear regression and fully connected neural networks

- It is ready to show that graph filters are equivariant to permutations of the input signals

Theorem (Permutation equivariance of graph filters)

Consider consistent permutations of the shift operator $\hat{S} = P^T S P$ and input signal $\hat{x} = P^T x$. Then

$$H(\hat{S})\hat{x} = P^T H(S) x$$

- Relabeling the input signal results in a consistent relabeling of the output signal
- Graph filters leverage permutation symmetries of graphs and graph signals

- It is equally ready to show that GNNs are also equivariant to permutations of the input signals

Theorem (Permutation equivariance of graph neural networks)

Consider consistent permutations of the shift operator $\hat{\mathbf{S}} = \mathbf{P}^T \mathbf{S} \mathbf{P}$ and input signal $\hat{\mathbf{x}} = \mathbf{P}^T \mathbf{x}$. Then

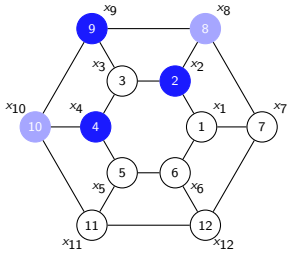
$$\Phi(\hat{\mathbf{x}}; \hat{\mathbf{S}}, \mathcal{H}) = \mathbf{P}^T \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$$

- Relabeling the input signal results in a consistent relabeling of the output signal
- Graph neural networks leverage permutation symmetries of graphs and graph signals

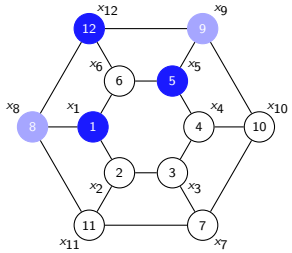
- Graph filters and GNNs, perform label-independent processing of graph signals

$\Rightarrow$ Permute input and shift $\equiv$ Relabel input $\Rightarrow$ Permute output $\equiv$ Relabel output

Graph signal $\mathbf{x}$ Supported on $\mathbf{S}$



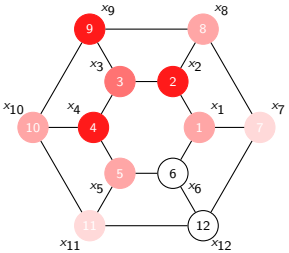
Graph signal $\hat{\mathbf{x}} = \mathbf{P}^T\mathbf{x}$ supported on $\hat{\mathbf{S}} = \mathbf{P}^T\mathbf{S}\mathbf{P}$



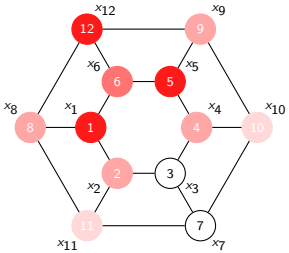
- Graph filters and GNNs, perform label-independent processing of graph signals

$\Rightarrow$ Permute input and shift $\equiv$ Relabel input $\Rightarrow$ Permute output $\equiv$ Relabel output

GNN output $\Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$ supported on $\mathbf{S}$

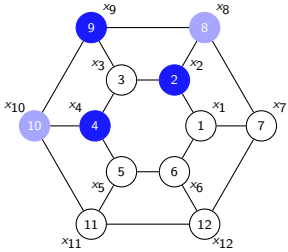


GNN $\Phi(\hat{\mathbf{x}}; \hat{\mathbf{S}}, \mathcal{H}) = \mathbf{P}^T \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$ on $\hat{\mathbf{S}} = \mathbf{P}^T \mathbf{S} \mathbf{P}$

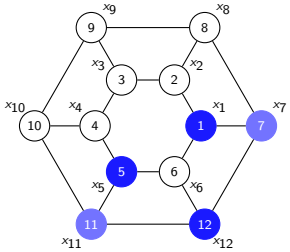


- Equivariance lets graph filters and GNNs exploit permutation symmetries of graphs and graph signals
- By symmetry we mean that the graph can be permuted onto itself $\Rightarrow \mathbf{S} = \mathbf{P}^T\mathbf{S}\mathbf{P}$
- Equivariance theorem implies $\Rightarrow \Phi\Big(\mathbf{P}^T\mathbf{x};\, \mathbf{S}, \mathcal{H}\Big) = \Phi\Big(\mathbf{P}^T\mathbf{x};\, \mathbf{P}^T\mathbf{S}\mathbf{P}, \mathcal{H}\Big) = \mathbf{P}^T\Phi\Big(\mathbf{x};\, \mathbf{S}, \mathcal{H}\Big)$

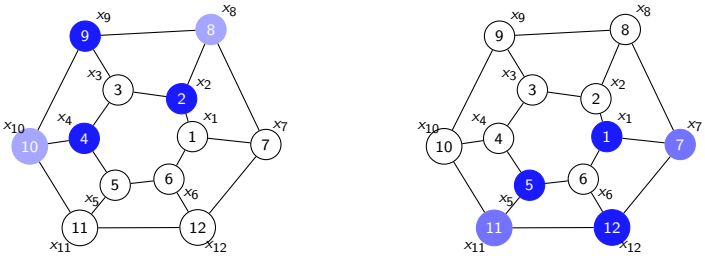
From observing $\mathbf{x}$ supported on $\mathbf{S}$



Learn to process $\mathbf{P}^T\mathbf{x}$ supported on $\mathbf{S} = \mathbf{P}^T\mathbf{S}\mathbf{P}$



- Graph not symmetric but close to symmetric $\Rightarrow$ Deformed version of a permutation of itself

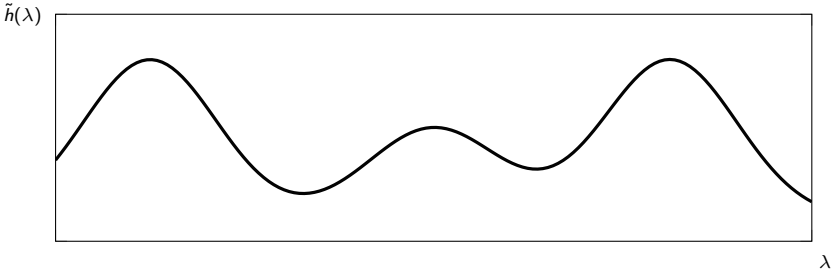


- Quasi-Symmetry, not symmetry $\Rightarrow$ Stability to deformations that are close to permutation.
- GNNs have better stability properties than graph filters $\Rightarrow$ Better at leveraging quasi-symmetries.

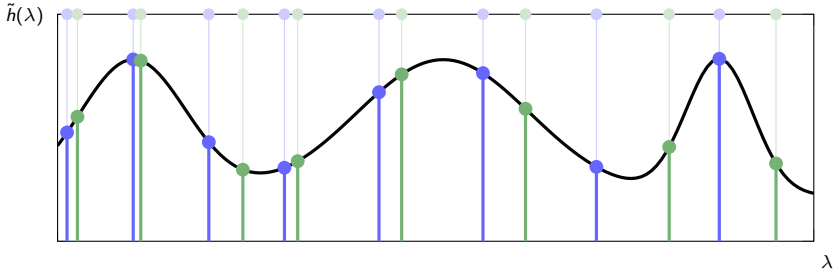
Stability of Graph Filters to Graph Perturbations

- Graph filters that are integral Lipschitz are stable to relative deformations of the graph

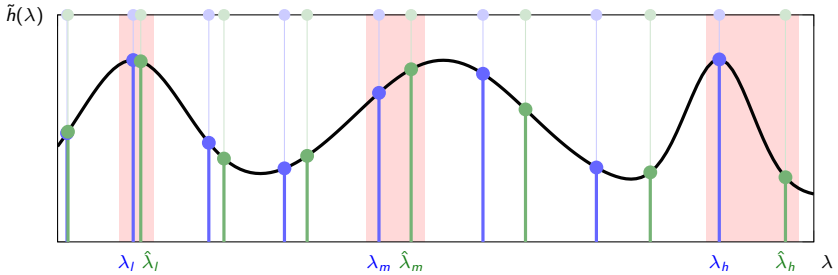
- Graph filters are operators defined on graph shift operators $\Rightarrow \mathbf{H}(\mathbf{S}) = \sum_{k=1}^{\infty} h_k \mathbf{S}^k$
- They are completely characterized by their frequency responses $\Rightarrow \tilde{h}(\lambda) = \sum_{k=1}^{\infty} h_k \lambda^k$



- Graph $\mathbf{S}$ has eigenvalues λ_i $\Rightarrow$ The response is instantiated at these eigenvalues $\tilde{h}(\lambda_i) = \sum_{k=1}^{\infty} h_k \lambda_i^k$
- Graph $\hat{\mathbf{S}}$ has eigenvalues $\hat{\lambda}_i$ $\Rightarrow$ The response is instantiated at these eigenvalues $\tilde{h}(\hat{\lambda}_i) = \sum_{k=1}^{\infty} h_k \hat{\lambda}_i^k$



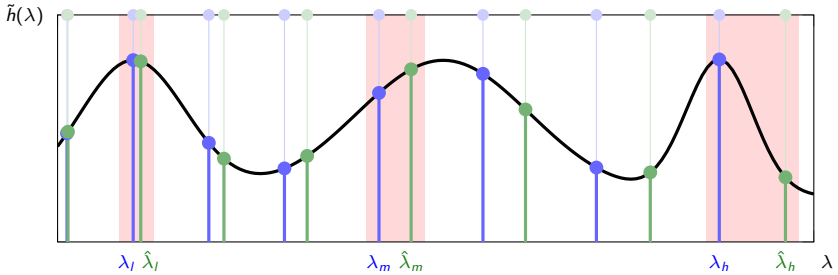
- Meaningful perturbations of a shift operator operator are relative $\Rightarrow \mathsf{P}^\top \hat{\mathsf{S}} \mathsf{P} = \mathsf{S} + \mathsf{ES} + \mathsf{SE}$
- Conceptually, we learn all there is to be learnt from dilations $\Rightarrow \hat{\mathsf{S}} = \mathsf{S} + \epsilon \mathsf{S}$
- Eigenvalues are dilated $\lambda_i \to \hat{\lambda}_i = (1+\epsilon)\lambda_i$. Frequency response instantiated on dilated eigenvalues



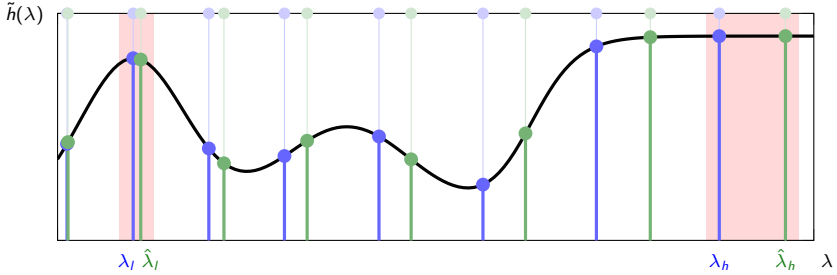
- Higher eigenvalues move more. Signals with high frequency components are more difficult to process

  $\Rightarrow$ Even small perturbations yield large differences in the filter values that are instantiated

  $\Rightarrow$ We think we instantiate $h\Big(\lambda_i\Big)$ $\Rightarrow$ But in reality we instantiate $h\Big(\hat{\lambda}_i\Big) = h\Big((1+\epsilon)\,\lambda_i\Big)$



- To attain stable graph signal processing we need integral Lipschitz filters $\Rightarrow \left| \lambda \tilde{h}'(\lambda) \right| \leq C$
- Either the eigenvalue does not change because we are considering low frequencies
- Or the frequency response does not change when we are considering high frequencies



- But only when they illustrate theorems! We've proven that integral Lipschitz filters are stable

Theorem (Integral Lipschitz Filters are Stable to Relative Perturbations)

Consider graph filter $\mathbf{H}$ along with shift operators $\mathbf{S}$ and $\hat{\mathbf{S}}$ having n nodes. If it holds that:

(H1) $\mathbf{S}$ and $\hat{\mathbf{S}}$ are related by $\mathbf{P}^T\hat{\mathbf{S}}\mathbf{P} = \mathbf{S} + \mathbf{E}\mathbf{S} + \mathbf{S}\mathbf{E}$ with $\mathbf{P}$ a permutation matrix

(H2) Error matrix has norm $\|\mathbf{E}\| = \epsilon$ and eigenvector misalignment constant δ relative to $\mathbf{S}$

(H3) The filter is integral Lipschitz with constant C

Then, the operator distance modulo permutation between filters $\mathbf{H}(\mathbf{S})$ and $\mathbf{H}(\hat{\mathbf{S}})$ is bounded by

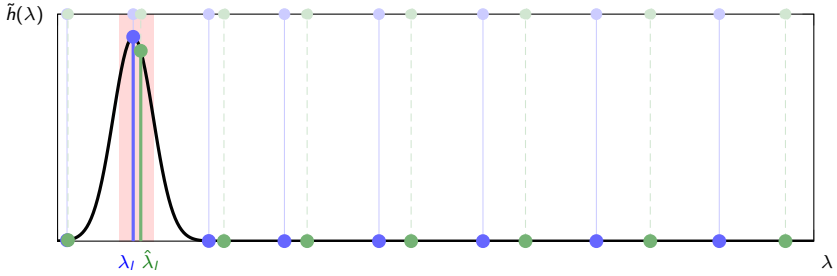
$$\left\| \mathbf{H}(\hat{\mathbf{S}}) - \mathbf{H}(\mathbf{S}) \right\|_{\mathcal{P}} \leq 2C \left(1 + \delta\sqrt{n}\right) \epsilon + \mathcal{O}(\epsilon^2).$$

Stability and Discriminability are Incompatible in Graph Filters

- The stability vs discriminability tradeoff depends on the frequency components of the signal

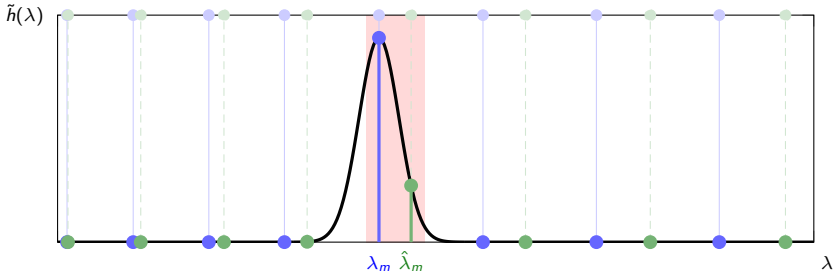
- At low frequencies a sharp highly discriminative filter is also highly stable

$\Rightarrow$ Ideal response $h\Big(\lambda_l\Big)$ is very close to perturbed response $h\Big(\hat{\lambda}_l\Big) = h\Big((1+\epsilon)\,\lambda_l\Big)$



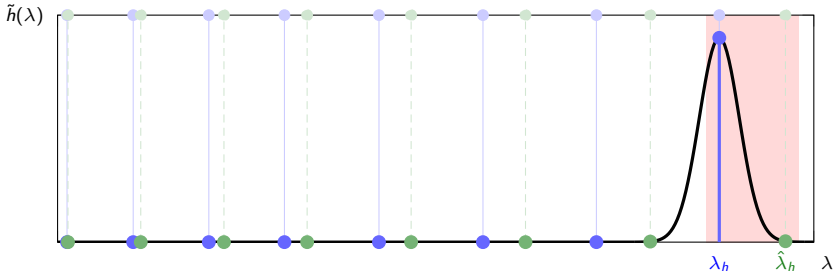
- At intermediate frequencies a sharp highly discriminative filter is somewhat stable

$\Rightarrow$ Ideal response $h\Big(\lambda_m\Big)$ is somewhat close to perturbed response $h\Big(\hat{\lambda}_m\Big) = h\Big((1+\epsilon)\,\lambda_m\Big)$



- At high frequencies a sharp highly discriminative filter is unstable. It becomes useless

$\Rightarrow$ Ideal response $h\Big(\lambda_h\Big)$ is very different from perturbed response $h\Big(\hat{\lambda}_h\Big) = h\Big((1+\epsilon)\,\lambda_h\Big)$

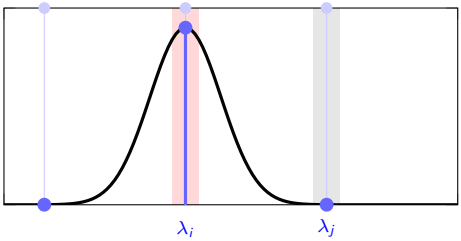


Fact: It is impossible to discriminate high frequency components with a stable filter

We can have a filter that is discriminative. Or a filter that is stable. But not one that is both.

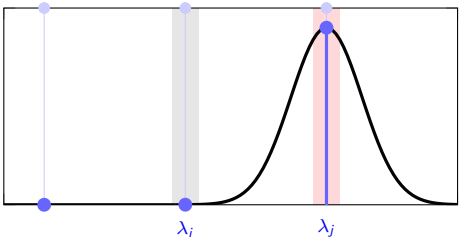
Filter that learns to identify $x_i = v_i$

Input	Output
$x_i = v_i$	$y_i = \tilde{h}(\lambda_i)v_i \approx v_i$
$x_j = v_j$	$y_j = \tilde{h}(\lambda_j)v_j \approx 0$

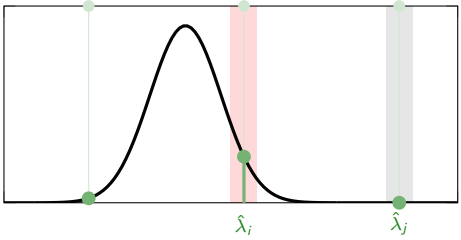


Filter that learns to identify $x_j = v_j$

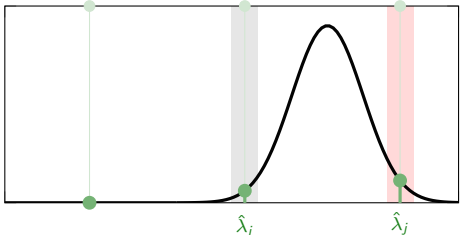
Input	Output
$x_i = v_i$	$y_i = \tilde{h}(\lambda_i)v_i \approx 0$
$x_j = v_j$	$y_j = \tilde{h}(\lambda_j)v_j \approx v_j$



Filter that learns to identify $x_i = v_i$	
Input	Output
$x_i = v_i$	$y_i = \tilde{h}(\hat{\lambda}_i)v_i = (\text{small})v_i$
$x_j = v_j$	$y_j = \tilde{h}(\hat{\lambda}_j)v_j \approx 0$



Filter that learns to identify $x_j = v_j$	
Input	Output
$x_i = v_i$	$y_i = \tilde{h}(\hat{\lambda}_i)v_i = (\text{small})v_i$
$x_j = v_j$	$y_j = \tilde{h}(\hat{\lambda}_j)v_j = (\text{small})v_j$



Fact: This is not what we would had expected a priori

It is reasonable to expect discriminability to be in conflict with stability. But this is not what happens. They are plainly incompatible.

- There is some tradeoff. Increasing C improves discriminability
  - $\Rightarrow$ But the effect is marginal at high frequencies
  - $\Rightarrow$ Given a stable filter there's always a graph with features that can't be discriminated by the filter
- The tradeoff is different at different frequencies. We can be more stable at low frequencies

The Stability vs Discriminability Tradeoff of GNNs

- The effect of pointwise nonlinearities is to move high frequencies into lower parts of the spectrum

  $\Rightarrow$ Where they can be discriminated with stable filters at deeper layers

- Nonlinearity is pointwise $\Rightarrow$ Same stability theorem that holds for graph filters, also holds for GNNs.

Theorem (GNN Stability to Relative Perturbations)

Consider a GNN operator $\Phi(\cdot; \mathbf{S}, \mathcal{H})$ along with shifts operators $\mathbf{S}$ and $\hat{\mathbf{S}}$ having n nodes. If:

(H1) Shift operators are related by $\mathbf{P}^T \hat{\mathbf{S}} \mathbf{P} = \mathbf{S} + \mathbf{E}\mathbf{S} + \mathbf{S}\mathbf{E}$ with $\mathbf{P}$ a permutation matrix

(H2) The error matrix $\mathbf{E}$ has norm $\|\mathbf{E}\| = \epsilon$ and eigenvector misalignement δ relative to $\mathbf{S}$

(H3) The GNN has L single-feature layers with integral Lipschitz filters with constant C

(H4) Filters have unit operator norm and the nonlinearity is normalized Lipschitz

The operator distance modulo permutation between $\Phi(\cdot; \mathbf{S}, \mathcal{H})$ and $\Phi(\cdot; \hat{\mathbf{S}}, \mathcal{H})$ is bounded by

$$\left\| \Phi(\cdot; \hat{\mathbf{S}}, \mathcal{H}) - \Phi(\cdot; \mathbf{S}, \mathcal{H}) \right\|_{\mathcal{P}} \leq 2C \left(1 + \delta\sqrt{n}\right) L\epsilon + \mathcal{O}(\epsilon^2).$$

- Fact: It is possible to discriminate high frequency components with a stable GNN

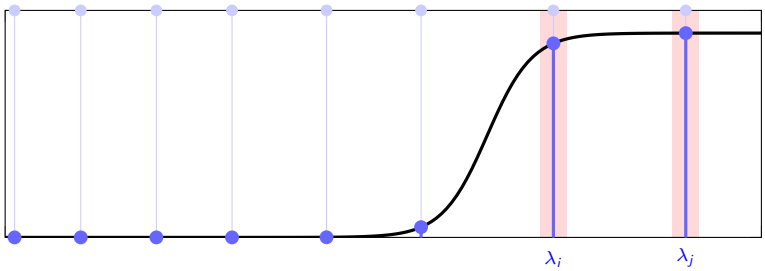
  $\Rightarrow$ GNNs inherits the (good) stability properties of graph filters

  $\Rightarrow$ But they don't have to inherit the (bad) discriminability properties of graph filters

- Pointwise nonlinearities demodulate high frequencies $\Rightarrow$ Stable discrimination in deeper layers

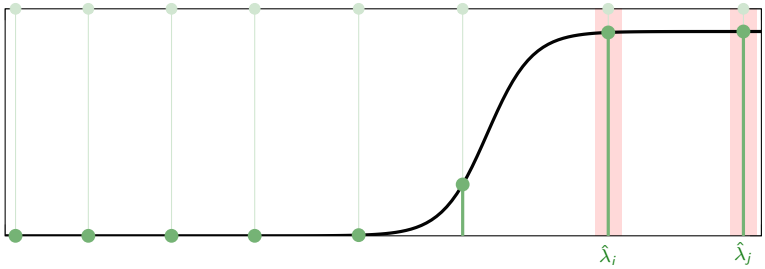
Filter that learns to isolate $x_i = v_i$ and $x_j = v_j$	
Input	Output
$x_i = v_i$	$y_i = \tilde{h}(\lambda_i)v_i \approx v_i$
$x_j = v_j$	$y_j = \tilde{h}(\lambda_j)v_j \approx v_j$

- Separates them from the rest. But it doesn't discriminate between them



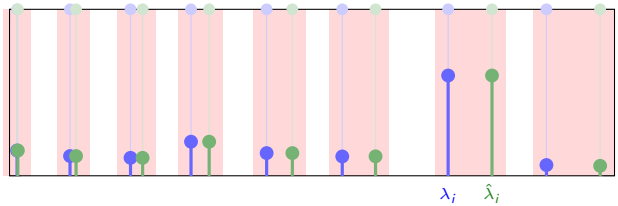
Filter that learns to isolate $x_i = v_i$ and $x_j = v_j$	
Input	Output
$x_i = v_i$	$y_i = \tilde{h}(\hat{\lambda}_i)v_i \approx v_i$
$x_j = v_j$	$y_j = \tilde{h}(\hat{\lambda}_j)v_j \approx v_j$

- It is, however, stable to deformations.

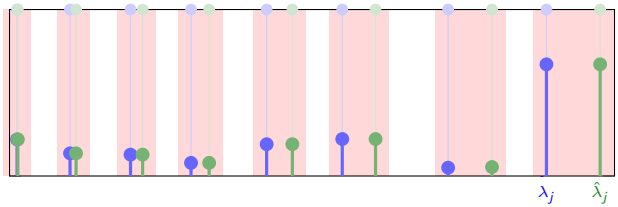


- Nonlinearities $\sigma(\mathbf{v}_i)$ and $\sigma(\mathbf{v}_j)$ spread energy across all frequencies
- Some energy where it used to be
- Some energy at other high frequencies
- Some energy at medium frequencies
- Some energy at low frequencies
- Where it can be discriminated with a stable filter in Layer 2

Spectrum of nonlinearity applied to $\mathbf{v}_i \Rightarrow \mathbf{V}^H\sigma(\mathbf{v}_i)$



Spectrum of nonlinearity applied to $\mathbf{v}_j \Rightarrow \mathbf{V}^H\sigma(\mathbf{v}_j)$



Fact: It is possible to discriminate high frequency components with a stable GNN

A GNN can be discriminative. And it can be stable. Stability and discriminability are compatible

- Stable layers can't discriminate high frequency components. They must use integral Lipschitz filters
- But nonlinearities are low pass $\Rightarrow$ They demodulate high frequencies into low frequencies

  $\Rightarrow$ ReLU: $\max(0, x)$. Absolute value: $|x|$. Hyperbolic tangent: $(e^{2x} - 1)/(e^{2x} + 1)$.
- Where a deeper layer can discriminate them with a stable filter

Equivariance, Stability, and Transference

Good and Bad Questions

- Some questions are good. Some questions are bad. Some are so bad they don't even have an answer
- Does this work and should this work are not scientific questions. They can't be proven or disproven

  $\Rightarrow$ The questions are so bad they don't even have an answer

- The theorems we have proven make scientific predictions. Which we can validate experimentally

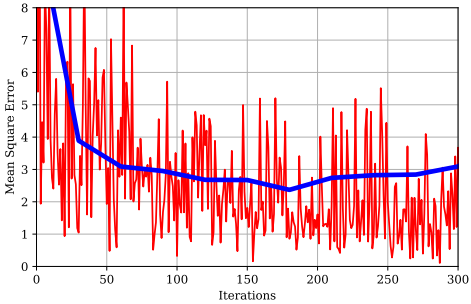
Theorem: Graph filters are equivariant to permutations of the shift operator

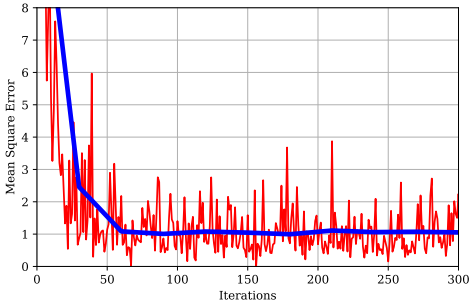
Theorem: GNNs are equivariant to permutations of the shift operator

Prediction: Graph filters outperform generic linear regression

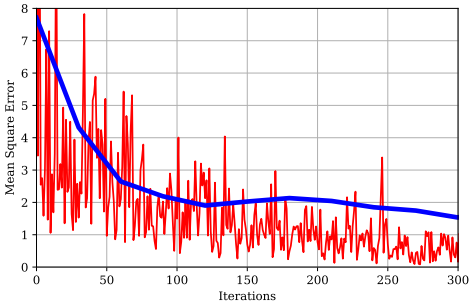
Prediction: GNNs outperform generic fully connected neural networks (FCNNs)

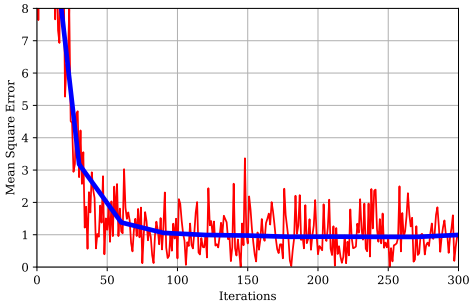
- Graph filters outperform linear regression when learning rating predictions in recommendation systems





- GNNs outperform FCNNs when learning rating predictions in recommendation systems





- Permutation equivariance of the task is crucial. You can't use a GNN just because you have a graph

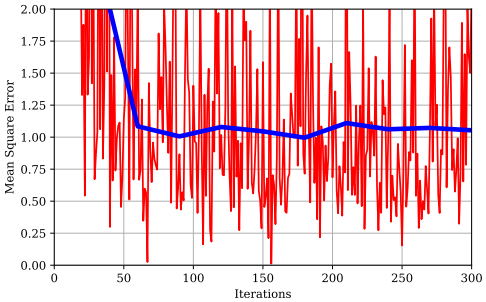
Theorem: Graph filters are stable to graph deformations if they are integral Lipschitz

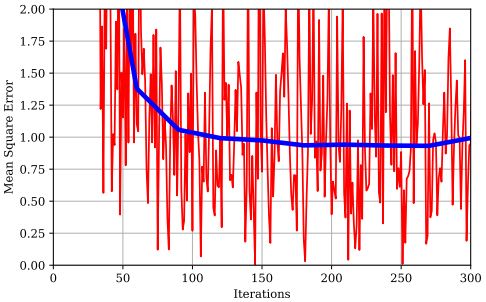
Theorem: GNNs made up of integral Lipschitz layers are stable to graph deformations

Prediction: GNNs outperform graph filters

They are equality stable but better at discriminating high frequency components

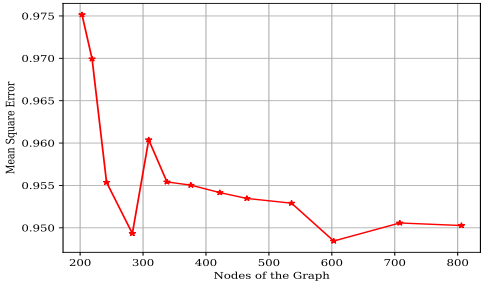
- GNNs outperform graph filters when learning rating predictions in recommendation systems





- The difference is small $\Rightarrow$ Rating prediction is largely a low frequency problem

- It is possible to transfer a GNN to a graph with a (much) larger number of nodes



- This child need a mom $\Rightarrow$ We study transference with graphon filters and graphon neural networks