Graph Neural Networks

Charilaos I. Kanatsoulis, Navid NaderiAlizadeh,

Alejandro Parada-Mayorga, Alejandro Ribeiro, and Luana Ruiz

gnn.seas.upenn.edu

2023 International Conference on Acoustics, Speech, and Signal Processing

Rhodes, Greece – June 6, 2023

Day 1: Machine Learning on Graphs

Charilaos I. Kanatsoulis, Navid NaderiAlizadeh,

Alejandro Parada-Mayorga, Alejandro Ribeiro, and Luana Ruiz

gnn.seas.upenn.edu

2023 International Conference on Acoustics, Speech, and Signal Processing

Rhodes, Greece – June 6, 2023

Visit our course website: `https://gnn.seas.upenn.edu`

How to use this site (I am not at Penn)

If you are a not a student at Penn, the instructors are honored that you consider the materials worth checking. We wish we had the time to work with you, alas, we do not. However, we think there is quite a lot that you can learn by watching the recorded video lectures and by working on the lab assignments. We have designed the materials with the goal of making them useable in self directed learning. How much we have succeeded at that is for you to decide.

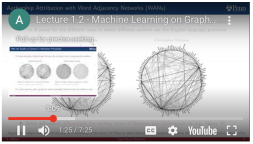
When you check the video lectures you will see that they come with a handout and a script. The handout and the script are designed to be used in conjunction with the videos. The videos are the union of the handout and the script. In addition to video lectures, you will find lab assignments and their solutions accessible through the lab webpage. The labs are designed with a complexity progression in mind. Start from Lab 1 if you have never worked in machine learning. Start from Lab 2 if you have ever encountered GNNs. Labs 3 and onwards are serious applications of GNNs to practical problems.

If you are hardcore and would rather read papers, this post has links to the papers that have inspired this course. These papers are part of the work on graph neural networks going on at Alelab. They are not a comprehensive literature review. You can find a little bit of that in this tutorial article in the signal processing magazine and in this more comprehensive review in the Proceedings of the IEEE.

If there is something you think we could to to help. We are happy to hear suggestions.

Video 1.2 – Machine Learning on Graphs: The Why

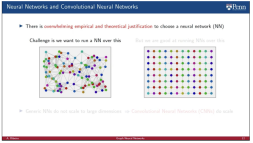
We care about GNNs because they enable machine learning on graphs. But why should we care about machine learning on graphs? We dwell here on the whys of machine learning on graphs. Why is it interesting? Why do we care? The reason we care is simple: Because graphs are pervasive in information processing.



• Covers Slides 6-10 in the handout.

Video 1.3 – Machine Learning on Graphs: The How

Having discussed the why, we tackle the how. How do we do machine learning on graphs? The answer to this question is pretty easy: We should use a neural network. We should do this, because we have overwhelming empirical and theoretical evidence for the value of neural networks. Understanding this evidence is one of the objectives of this course. But before we are ready to do that, there is a dealbreaker challenge potentially lurking in the shadows: Neural Networks must exploit structure to be scalable.



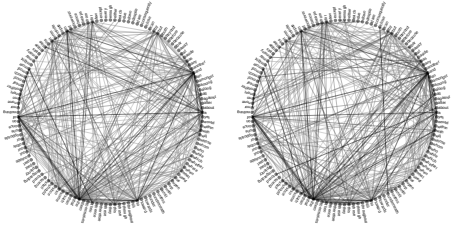
• Covers Slides 11-13 in the handout.

Machine Learning on Graphs: Why?

Why Are Graphs so Common in Information Processing?

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

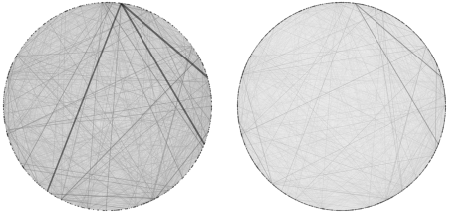
Authorship Attribution



Identify the author of a text of unknown provenance

Segarra et al '16, arxiv.org/abs/1805.00165

Recommendation Systems



Predict the rating a customer would give to a product

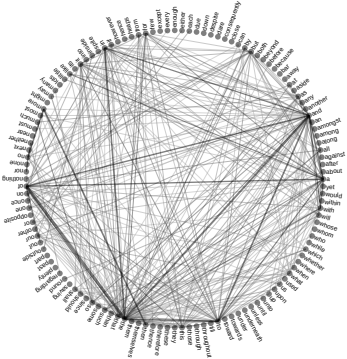
Ruiz et al '18, arxiv.org/abs/1903.12575

- In both cases there exists a graph that contains meaningful information about the problem to solve

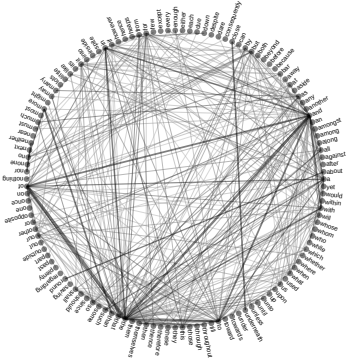
Authorship Attribution with Word Adjacency Networks (WANs)

- Nodes represent different function words and edges how often words appear close to each other
  - ⇒ A proxy for the different ways in which different authors use the English language grammar

William Shakespeare



Christopher Marlowe



- WAN differences differentiate the writing styles of Marlowe and Shakespeare in, e.g., Henry VI

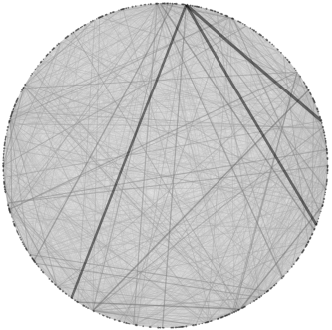
Segarra-Eisen-Egan-Ribeiro, *Attributing the Authorship of the Henry VI Plays by Word Adjacency*, Shakespeare Quarterly 2016, doi.org/10.1353/shq.2016.0024

Recommendation System with Collaborative Filtering

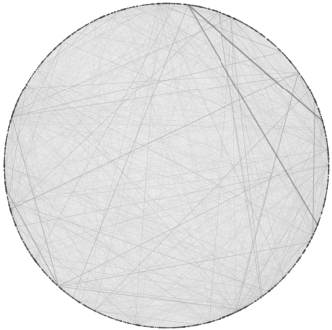
- Nodes represent different customers and edges their average similarity in product ratings

  $\Rightarrow$ The graph informs the completion of ratings when some are unknown and are to be predicted

Variation Diagram for Original (sampled) ratings



Variation Diagram for Reconstructed (predicted) ratings



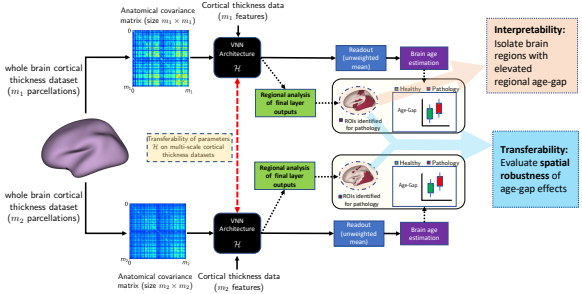
- Variation energy of reconstructed signal is (much) smaller than variation energy of sampled signal

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

Brain Networks: Brain Age Scoring (Thursday 11:35, Nafsika B)

Ageing is a risk factor for neurodegeneration and biological age (brain age) is elevated compared to chronological age in pathology.

Hence, **Age-Gap** (brain age – chronological age) is a biomarker of interest.



Cortical Thickness Brain Signals. GNN on anatomical covariance matrix leverages cortical thickness (CT) features to predict brain age.

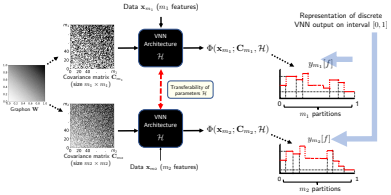
Regional age-gap is defined by the difference between GNN prediction and outputs at the final layer of GNN.

Elevated brain age gap effect is driven by regional age-gap effects in impacted regions.

Interpretable regional profile to elevated brain age.

Regions with elevated age-gap in Alzheimer's Disease

GNN can be transferred across different graphs



Sihag-Mateos-McMillan-Ribeiro, *coVariance Neural Networks.*, arxiv.org/abs/2205.15856

Graphs in Multiagent Physical Systems

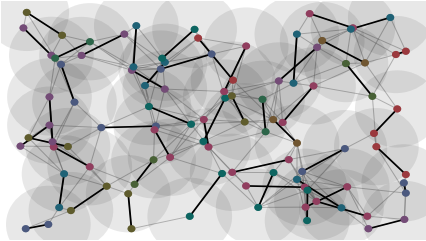
- Graphs are more than data structures $\Rightarrow$ They are models of physical systems with multiple agents

Decentralized Control of Autonomous Systems

Coordinate a team of agents without central coordination

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

Wireless Communications Networks



Manage resources in wireless communications

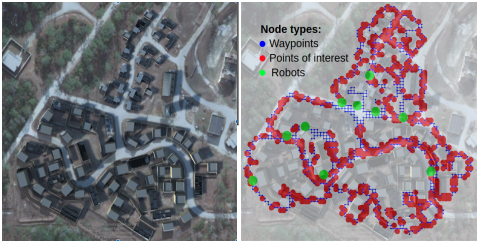
Eisen-Ribeiro '19, arxiv.org/abs/1909.01865

- The graph is the source of the problem $\Rightarrow$ Challenge is that goals are global but information is local

Graphs in Multiagent Physical Systems

- Graphs are more than data structures $\Rightarrow$ They are models of physical systems with multiple agents

Decentralized Control of Autonomous Systems



Collaborative navigation of roads with a team of agents

Tolstaya et al '21, arxiv.org/abs/2011.01119

Wireless Communications Networks

Mobile infrastructure on demand to support a task team

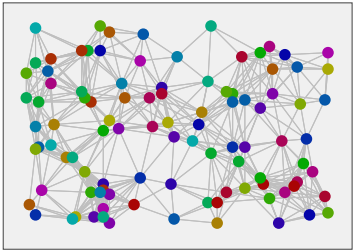
Mox et al '22, arxiv.org/abs/2112.07663

- The graph is the source of the problem $\Rightarrow$ Challenge is that goals are global but information is local

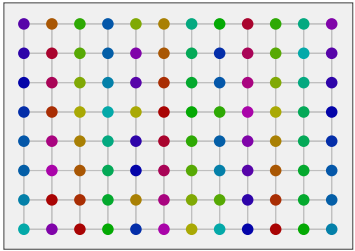
Machine Learning on Graphs: How?

- 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

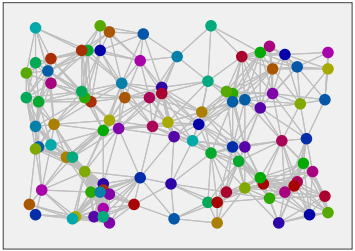


- Generic NNs do not scale to large dimensions $\Rightarrow$ Convolutional Neural Networks (CNNs) do scale

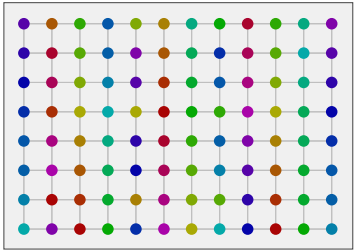
Convolutional Neural Networks and Graph Neural Networks

- CNNs are made up of layers composing convolutional filter banks with pointwise nonlinearities

Process graphs with graph convolutional NNs



Process images with convolutional NNs



- Generalize convolutions to graphs $\Rightarrow$ Compose graph filter banks with pointwise nonlinearities
- Stack in layers to create a graph (convolutional) Neural Network (GNN)

Convolutions in Time, in Space, and on Graphs

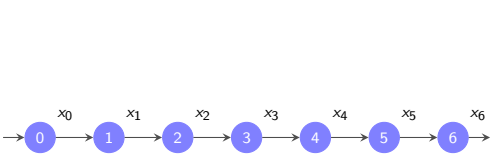
- How do we generalize convolutions in time and space to operate on graphs?

  $\Rightarrow$ Even though we do not often think of them as such, convolutions are operations on graphs

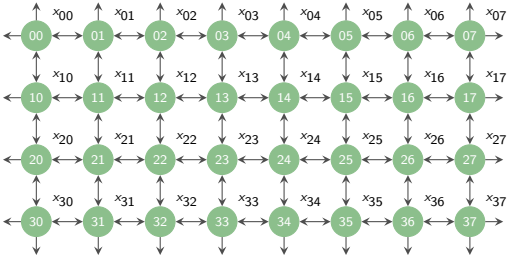
Time and Space are Representable by Graphs

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

Description of time with a directed line graph



Description of images (space) with a grid graph

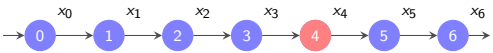


- Line graph represents adjacency of points in time. Grid graph represents adjacency of points in space

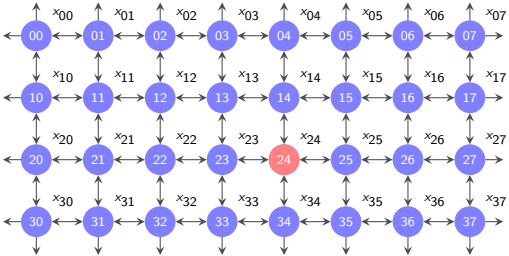
Convolutions in Time and Space

- Use line and grid graphs to write convolutions as polynomials on respective adjacency matrices $\mathbf{S}$

Description of time with a directed line graph



Description of images (space) with a grid graph

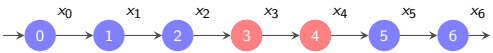


- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0 \mathbf{S}^0 \mathbf{x} + h_1 \mathbf{S}^1 \mathbf{x} + h_2 \mathbf{S}^2 \mathbf{x} + h_3 \mathbf{S}^3 \mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$

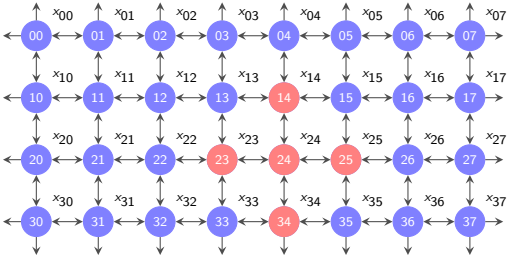
Convolutions in Time and Space

- Use line and grid graphs to write convolutions as polynomials on respective adjacency matrices $\mathbf{S}$

Description of time with a directed line graph



Description of images (space) with a grid graph

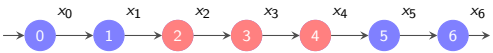


- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0 \mathbf{S}^0 \mathbf{x} + h_1 \mathbf{S}^1 \mathbf{x} + h_2 \mathbf{S}^2 \mathbf{x} + h_3 \mathbf{S}^3 \mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$

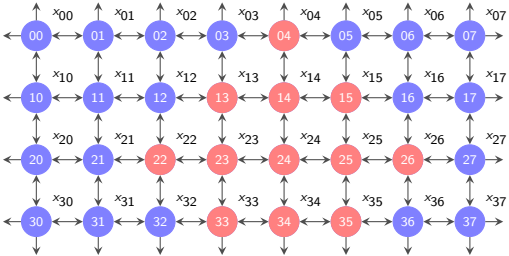
Convolutions in Time and Space

- Use line and grid graphs to write convolutions as polynomials on respective adjacency matrices $\mathbf{S}$

Description of time with a directed line graph



Description of images (space) with a grid graph

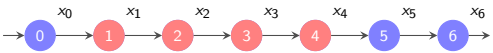


- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0\,\mathbf{S}^0\mathbf{x} + h_1\,\mathbf{S}^1\mathbf{x} + h_2\,\mathbf{S}^2\mathbf{x} + h_3\,\mathbf{S}^3\mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k\,\mathbf{S}^k\mathbf{x}$

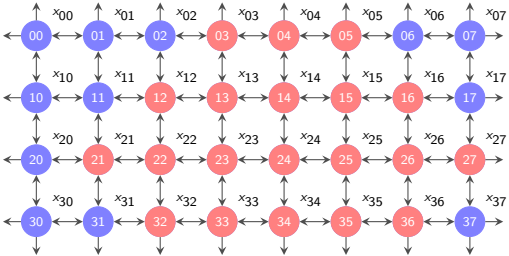
Convolutions in Time and Space

- Use line and grid graphs to write convolutions as polynomials on respective adjacency matrices $\mathbf{S}$

Description of time with a directed line graph



Description of images (space) with a grid graph

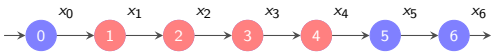


- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0\,\mathbf{S}^0\mathbf{x} + h_1\,\mathbf{S}^1\mathbf{x} + h_2\,\mathbf{S}^2\mathbf{x} + h_3\,\mathbf{S}^3\mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k\,\mathbf{S}^k\mathbf{x}$

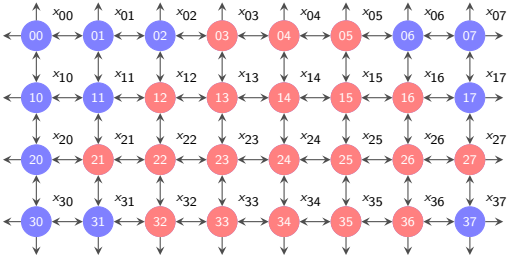
Convolutions in Time and Space

- Use line and grid graphs to write convolutions as polynomials on respective adjacency matrices $\mathbf{S}$

Description of time with a directed line graph



Description of images (space) with a grid graph

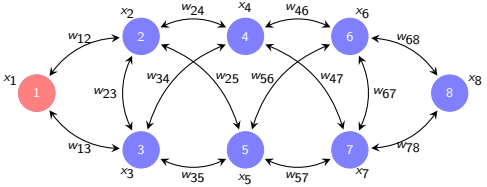


- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0\,\mathbf{S}^0\mathbf{x} + h_1\,\mathbf{S}^1\mathbf{x} + h_2\,\mathbf{S}^2\mathbf{x} + h_3\,\mathbf{S}^3\mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k\,\mathbf{S}^k\mathbf{x}$

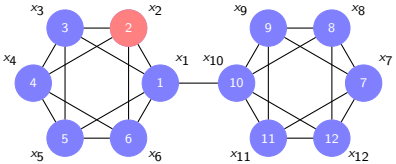
Convolutions on Graphs

- For graph signals we define graph convolutions as polynomials on matrix representations of graphs

A signal supported on a graph



Another signal supported on another graph



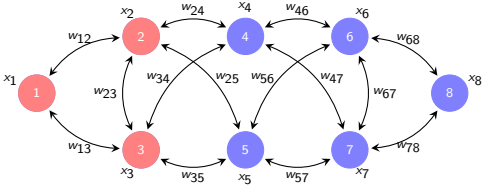
- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0 \mathbf{S}^0 \mathbf{x} + h_1 \mathbf{S}^1 \mathbf{x} + h_2 \mathbf{S}^2 \mathbf{x} + h_3 \mathbf{S}^3 \mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$

- Graph convolutions share the locality of conventional convolutions. Recovered as particular case

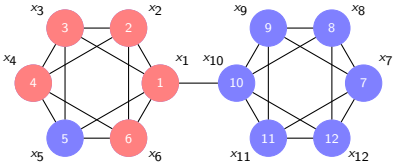
Convolutions on Graphs

- For graph signals we define graph convolutions as polynomials on matrix representations of graphs

A signal supported on a graph



Another signal supported on another graph



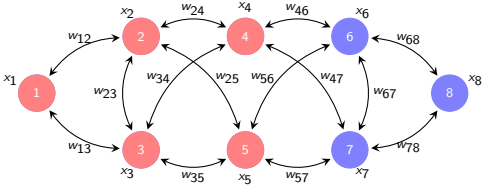
- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0 \mathbf{S}^0 \mathbf{x} + h_1 \mathbf{S}^1 \mathbf{x} + h_2 \mathbf{S}^2 \mathbf{x} + h_3 \mathbf{S}^3 \mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$

- Graph convolutions share the locality of conventional convolutions. Recovered as particular case

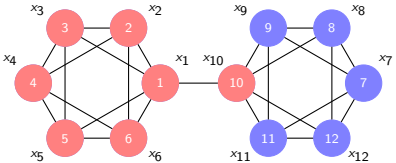
Convolutions on Graphs

- For graph signals we define graph convolutions as polynomials on matrix representations of graphs

A signal supported on a graph



Another signal supported on another graph



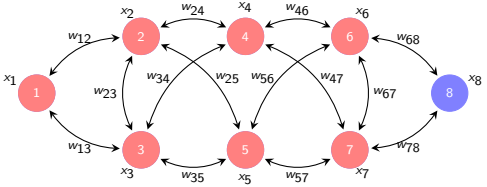
- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0 \mathbf{S}^0 \mathbf{x} + h_1 \mathbf{S}^1 \mathbf{x} + h_2 \mathbf{S}^2 \mathbf{x} + h_3 \mathbf{S}^3 \mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$

- Graph convolutions share the locality of conventional convolutions. Recovered as particular case

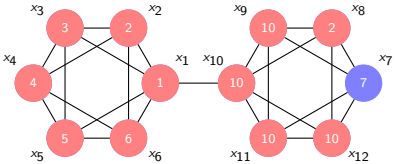
Convolutions on Graphs

- For graph signals we define graph convolutions as polynomials on matrix representations of graphs

A signal supported on a graph



Another signal supported on another graph



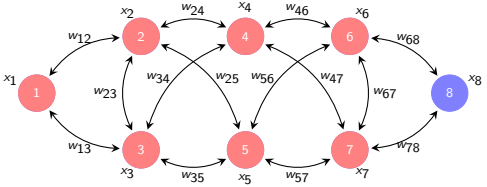
- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0 \mathbf{S}^0 \mathbf{x} + h_1 \mathbf{S}^1 \mathbf{x} + h_2 \mathbf{S}^2 \mathbf{x} + h_3 \mathbf{S}^3 \mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$

- Graph convolutions share the locality of conventional convolutions. Recovered as particular case

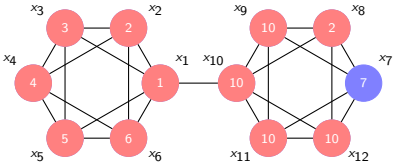
Convolutions on Graphs

- For graph signals we define graph convolutions as polynomials on matrix representations of graphs

A signal supported on a graph



Another signal supported on another graph

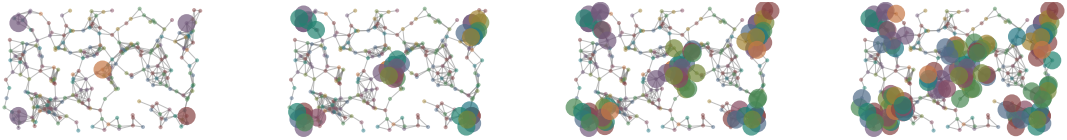


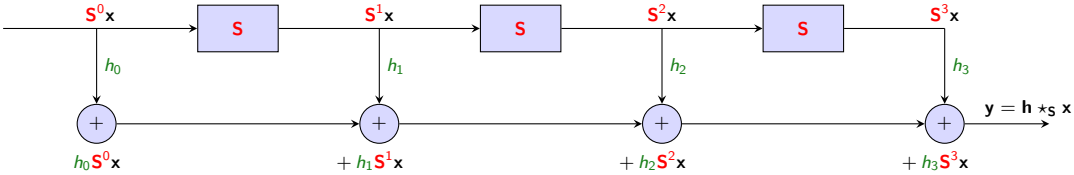
- Filter with coefficients h_k $\Rightarrow$ Output $\mathbf{z} = h_0 \mathbf{S}^0 \mathbf{x} + h_1 \mathbf{S}^1 \mathbf{x} + h_2 \mathbf{S}^2 \mathbf{x} + h_3 \mathbf{S}^3 \mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$

- Graph convolutions share the locality of conventional convolutions. Recovered as particular case

Graph Convolutional Filters as Diffusion Operators

- A graph convolution is a weighted linear combination of the elements of the diffusion sequence
- Can represent graph convolutions with a shift register $\Rightarrow$ Convolution $\equiv$ Shift. Scale. Sum





Definition (Convolution)

A convolutional filter is a polynomial on a shift operator with coefficients h_k $\Rightarrow \mathbf{z} = \sum_{k=0}^{\infty} h_k \mathbf{S}^k \mathbf{x}$

- It is the same algebraic object whether we consider time, space, or graphs
- They all have compositionality (operator powers) and some kind of equivariance
- They all admit a frequency representation

  $\Rightarrow$ Filters are pointwise operators in the eigenvector basis of the shift operator

Parada Mayorga-Ribeiro , *Algebraic Neural Networks: Stability to Deformations*, arxiv.org/abs/2009.01433

Definition (Algebraic Convolutions with Multiple Features)

Input signal $\mathbf{X} \in \mathbb{R}^{N \times F}$ with F features. Output signal $\mathbf{Z} \in \mathbb{R}^{N \times G}$ with G features. Filter coefficients $\mathbf{H}_k$ are $F \times G$ matrices. The convolutional filter with coefficients $\mathbf{H}_k$ is

$$\mathbf{Z} = \sum_{k=0}^{\infty} \mathbf{S}^k \times \mathbf{X} \times \mathbf{H}_k$$

- It has the same algebraic structure of a regular filter with scalar coefficients.
- Retains compositionality, equivariance, and existence of a frequency representation
- Filters with multiple features are more expressive. The ones we use to build GNNs and CNNs

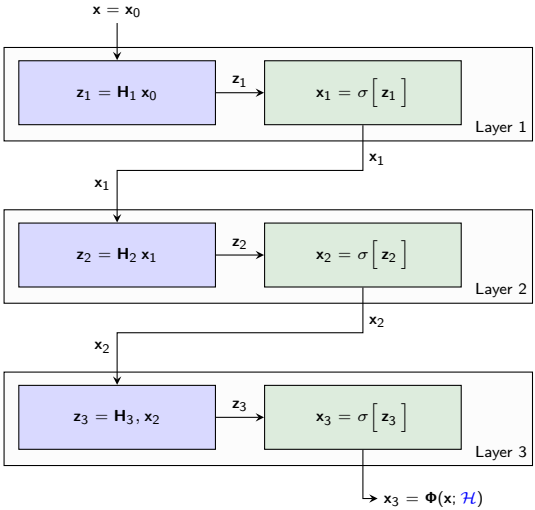
Convolutional Neural Networks and Graph Neural Networks

- CNNs and GNNe are minor variations of linear convolutional filters

  $\Rightarrow$ Compose filters with pointwise nonlinearities and compose these compositions into several layers

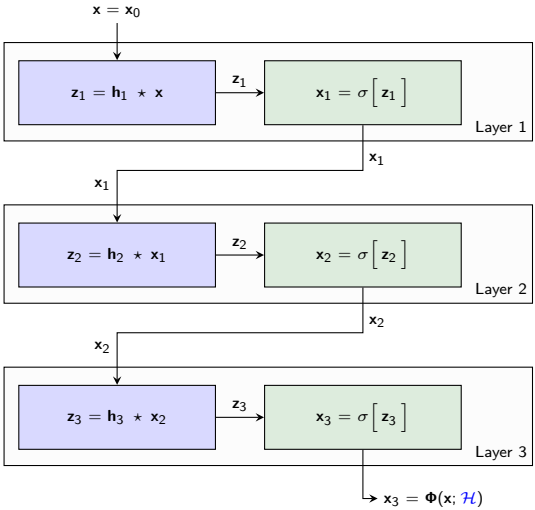
Neural Networks (NNs)

- A neural network composes a cascade of layers
- Each of which are themselves compositions of linear maps with pointwise nonlinearities
- Does not scale to large dimensional signals $\mathbf{x}$



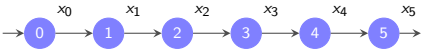
Convolutional Neural Networks (CNNs)

- A convolutional NN composes a cascade of layers
- Each of which are themselves compositions of convolutions with pointwise nonlinearities
- Scales well. The Deep Learning workhorse
- A CNNs are minor variation of convolutional filters
  - $\Rightarrow$ Just add nonlinearity and compose
  - $\Rightarrow$ They scale because convolutions scale

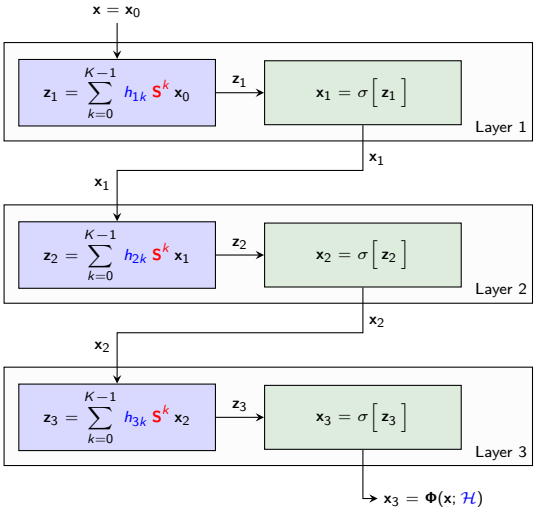


When we Think of Time Signals as Supported on a Line Graph

- Those convolutions are polynomials on the adjacency matrix of a line graph

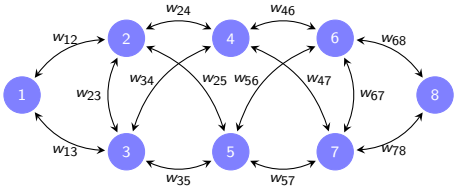


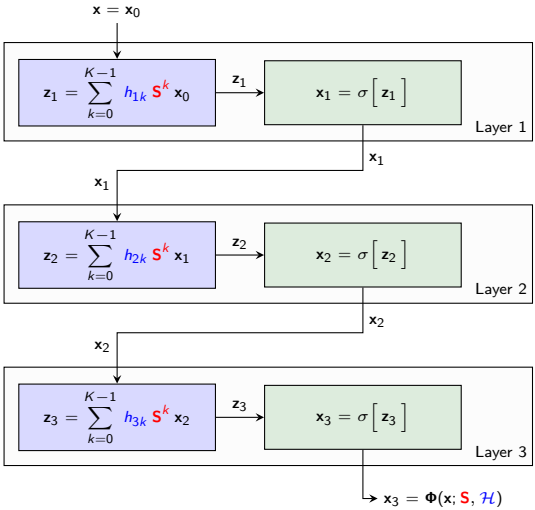
- Just another way of writing convolutions and Just another way of writing CNNs
- But one that lends itself to generalization



Graph Neural Networks (GNNs)

- The graph can be any arbitrary graph
- The polynomial on the matrix representation $\mathbf{S}$ becomes a graph convolutional filter

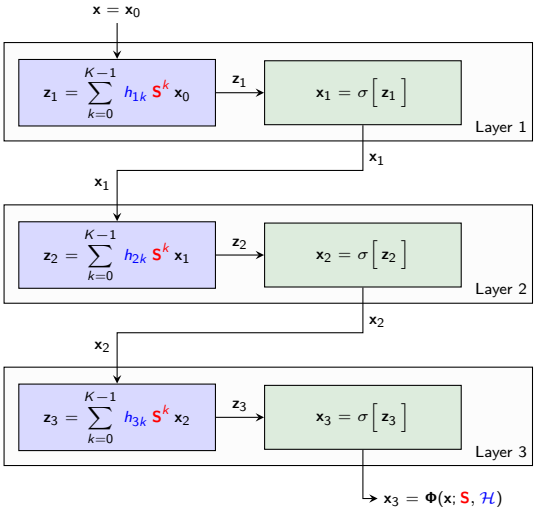




Gama-Marques-Leus-Ribeiro, *Convolutional Neural Network Architectures for Signals Supported on Graphs*, TSP 2019, arxiv.org/abs/1805.00165

Graph Neural Networks (GNNs)

- A graph NN composes a cascade of layers
- Each of which are themselves compositions of graph convolutions with pointwise nonlinearities
- A NN with linear maps restricted to convolutions
- Recovers a CNN if **S** describes a line graph

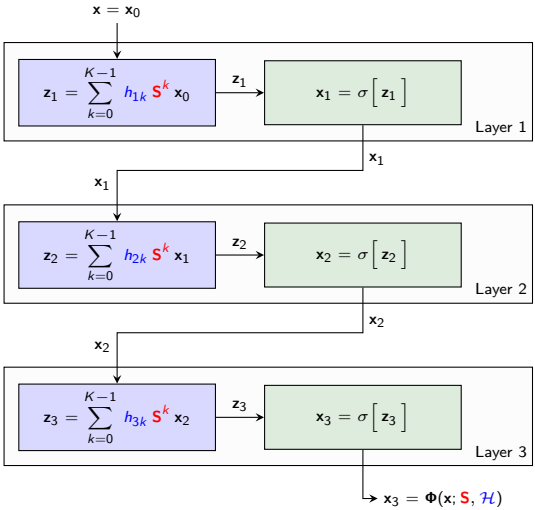


Gama-Marques-Leus-Ribeiro, *Convolutional Neural Network Architectures for Signals Supported on Graphs*, TSP 2019, arxiv.org/abs/1805.00165

Graph Neural Networks (GNNs)

- There is growing evidence of scalability.
- A GNN is a minor variation of a graph filter

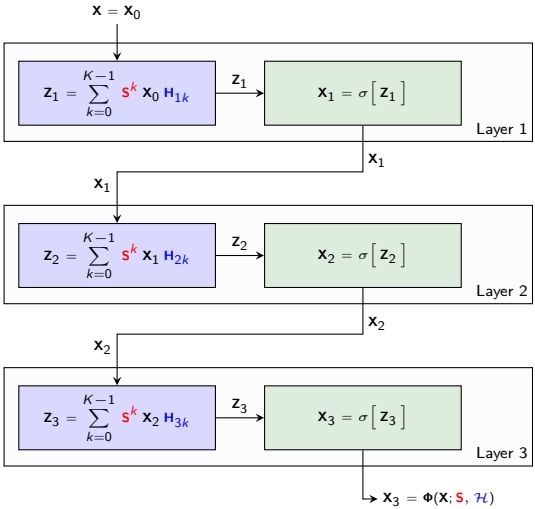
  $\Rightarrow$ Just add nonlinearity and compose
- Both are scalable because they leverage the signal structure codified by the graph



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

Graph Neural Networks with Multiple Features

- In practice we use layers with multiple features
- This is to increase representation power but it does not affect our fundamental observations



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

Equivariance and Stability Properties of GNNs

Gama-Bruna-Ribeiro, *Stability Properties of Graph Neural Networks*, TSP 2020, arxiv.org/abs/1905.04497

Gama-Isufi-Leus-Ribeiro, *Graphs, Convolutions, and Neural Networks: From Graph Filters to Graph Neural Networks*, SPMag 2020, arxiv.org/abs/2003.03777

Ruiz-Gama-Ribeiro, *Graph Neural Networks: Architectures, Stability and Transferability*, PIEEE 2021 arxiv.org/abs/2008.01767

Permutation Equivariance and Stability

Fact 1

Graph filters and GNNs "work." Outperform general linear transforms and fully connected NNs.

Fact 2

GNNs outperform graph filters in most learning tasks.

Permutation Equivariance and Stability

Fact 1: Graph Filters and GNNs are Permutation Equivariant

Graph filters and GNNs leverage symmetries of graph signals

Fact 2

GNNs outperform graph filters in most learning tasks.

Permutation Equivariance and Stability

Fact 1: Graph Filters and GNNs are Permutation Equivariant

Graph filters and GNNs leverage symmetries of graph signals

Fact 2: Stability Properties of GNNs

GNNs can be simultaneously discriminative and stable to deformations. Graph filters cannot.

Permutation Equivariance and Stability

Fact 1: Graph Filters and GNNs are Permutation Equivariant

Graph filters and GNNs leverage symmetries of graph signals

Permutation Equivariance of Graph Neural Networks

- 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

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

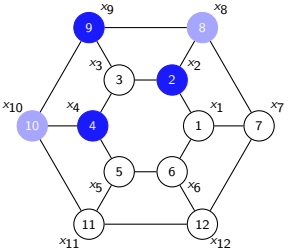
- Relabeling the input signal results in a consistent relabeling of the output signal

Signal Processing with Graph Filters and GNNs is Independent of Labeling

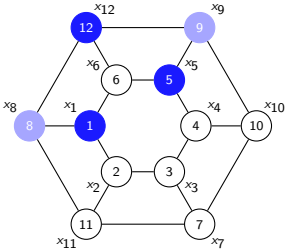
- Graph filters and GNNs, perform label-independent processing of graph signals

$\Rightarrow$ 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 $\hat{\mathbf{S}} = \mathbf{P}^T\mathbf{S}\mathbf{P}$

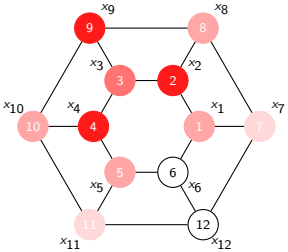


Signal Processing with Graph Filters and GNNs is Independent of Labeling

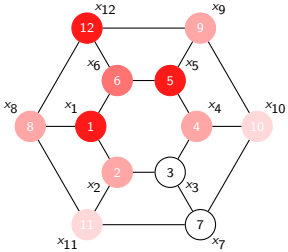
- Graph filters and GNNs, perform label-independent processing of graph signals

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

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



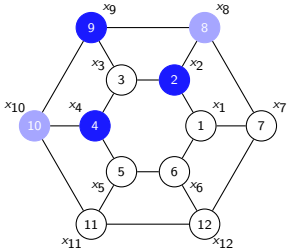
GNN $\Phi(\hat{\mathbf{x}}; \hat{\mathbf{S}}, \mathcal{H}) = \mathbf{P}^T \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H})$ on $\hat{\mathbf{S}} = \mathbf{P}^T \mathbf{S} \mathbf{P}$



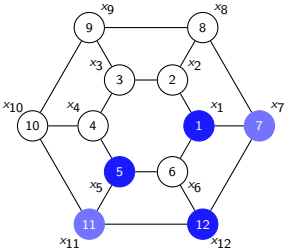
Graph Filters and GNNs Exploit Permutation Symmetries

- 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 $\mathbf{S} = \mathbf{P}^T\mathbf{S}\mathbf{P}$

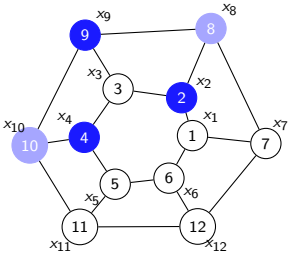


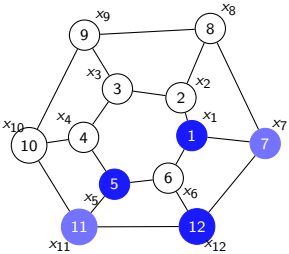
Fact 2: Stability Properties of GNNs

GNNs can be simultaneously discriminative and stable to deformations. Graph filters cannot.

It is Quasi-Symmetry we Want to Exploit. Not Symmetry

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

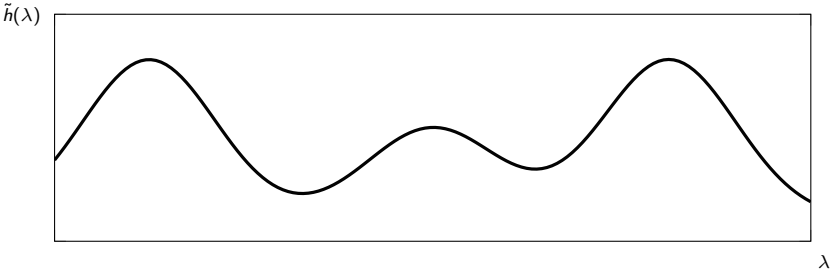




- Quasi-Symmetry, not symmetry $\Rightarrow$ Stability to deformations that are close to permutation.
- GNNs have better stability properties than graph filters $\Rightarrow$ Better at leveraging quasi-symmetries.

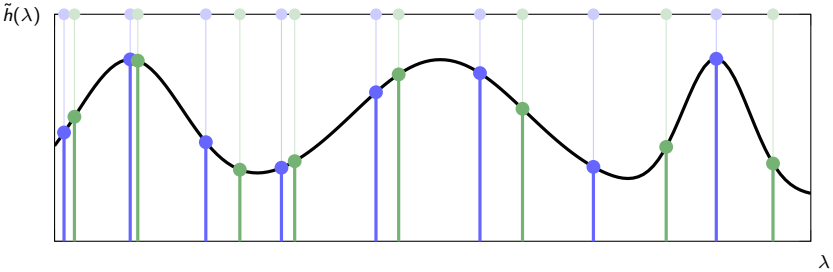
Frequency Response of a Graph Filter

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



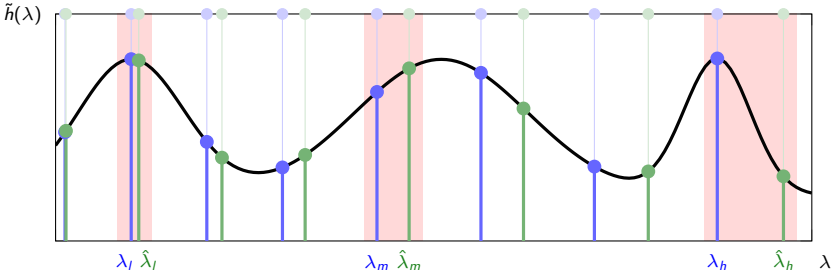
The Effect of the Graph

- 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 $\tilde{h}(\hat{\lambda}_i) = \sum_{k=1}^{\infty} h_k \hat{\lambda}_i^k$



Relative Perturbations of a Shift Operator

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

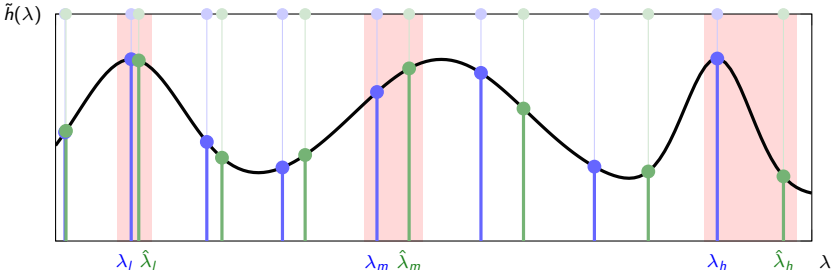


Higher Frequencies are More Difficult to Process

- Large eigenvalues move more. Signals with high frequencies are more difficult to process

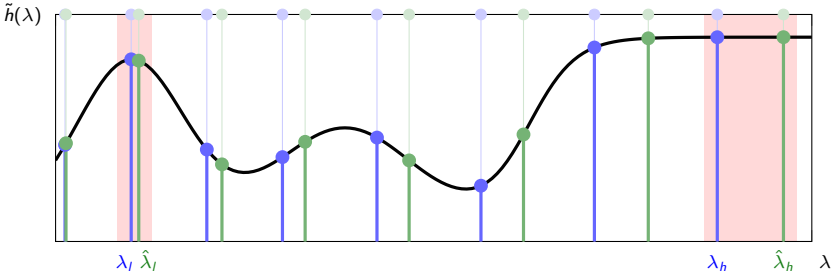
  $\Rightarrow$ Even small perturbations yield large differences in the filter values that are instantiated

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



Stability Requires Integral Lipschitz Filters

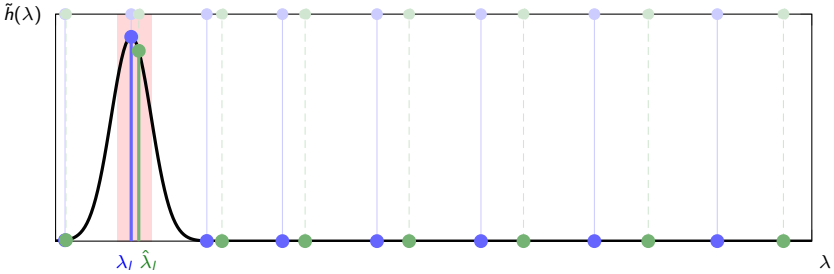
- To attain stable graph signal processing we need integral Lipschitz filters $\Rightarrow \left| \lambda \tilde{h}'(\lambda) \right| \leq C$
- Either the eigenvalue does not change because we are considering low frequencies
- Or the frequency response does not change when we are considering high frequencies



Discriminative Filter at Low Frequencies

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

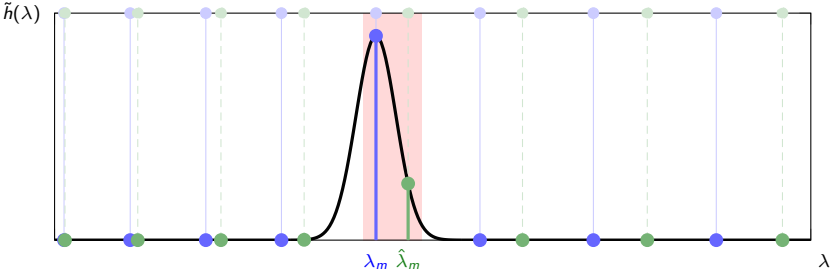
$\Rightarrow$ Ideal response $h\left(\lambda_l\right)$ is very close to perturbed response $h\left(\hat{\lambda}_l\right) = h\left((1+\epsilon)\,\lambda_l\right)$



Discriminative Filter at Medium Frequencies

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

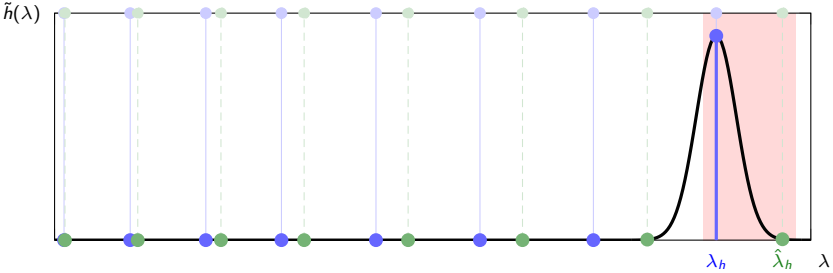
$\Rightarrow$ Ideal response $h\left(\lambda_m\right)$ is somewhat close to perturbed response $h\left(\hat{\lambda}_m\right) = h\left((1+\epsilon)\,\lambda_m\right)$



Discriminative Filter at High Frequencies

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

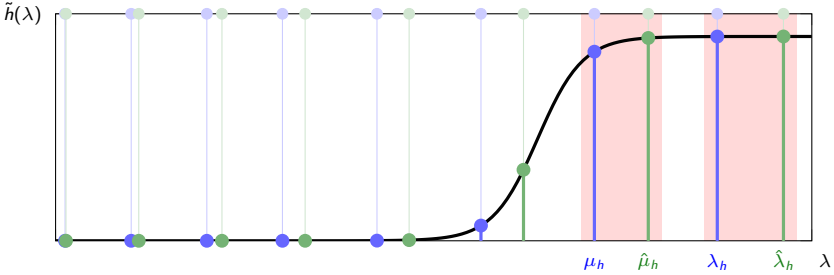
$\Rightarrow$ Ideal response $h\left(\lambda_h\right)$ is very different from perturbed response $h\left(\hat{\lambda}_h\right) = h\left((1+\epsilon)\,\lambda_h\right)$



Discriminative Filter at High Frequencies

- We can have stability to deformations if we use an integral Lipschitz filters $\Rightarrow \left| \lambda \tilde{h}'(\lambda) \right| \leq C$

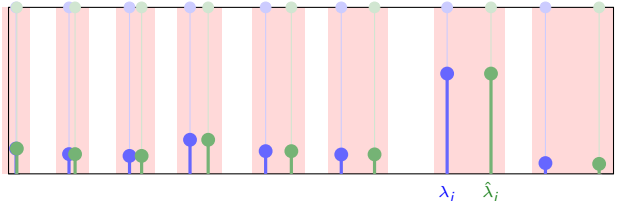
  $\Rightarrow$ But this precludes the discrimination of high frequency components



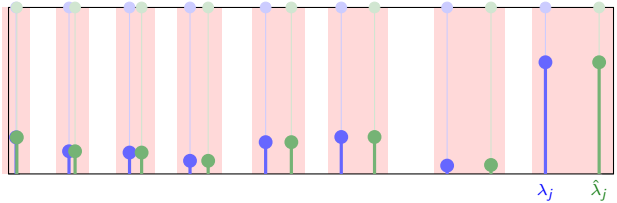
Pointwise Nonlinearities are Frequency Mixers

- 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 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 $\mathbf{v}_j$ $\Rightarrow \mathbf{V}^H\sigma(\mathbf{v}_j)$



Stability vs Discriminability Tradeoff of GNNs

Fact 2: Stability Properties of GNNs

GNNs can be simultaneously discriminative and stable to deformations. Graph filters cannot.

Stability vs Discriminability Tradeoff of GNNs

Fact 2: Stability Properties of GNNs

For the same sensitivity to deformations, GNNs are more discriminative than graph filters

Theorem (GNN Stability to Relative Perturbations)

Consider a GNN operator $\mathbf{\Phi}(\cdot; \mathbf{S}, \mathbf{A})$ 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 $\mathbf{\Phi}(\cdot; \mathbf{S}, \mathbf{A})$ and $\mathbf{\Phi}(\cdot; \hat{\mathbf{S}}, \mathbf{A})$ is bounded by

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

Gama-Bruna-Ribeiro, *Stability Properties of Graph Neural Networks*, TSP 2020, arxiv.org/abs/1905.04497

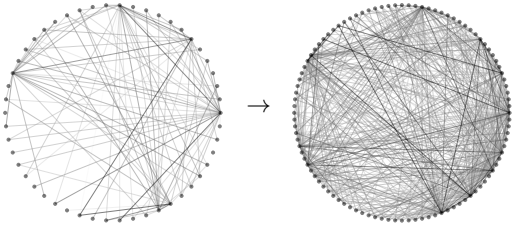
Transferability Properties of Graph Neural Networks

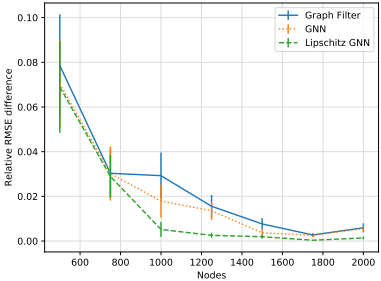
- A GNN that is trained in a graph $\mathbf{S}$ can be executed on any other graph $\hat{\mathbf{S}}$

  $\Rightarrow$ In particular, we can execute it in a much larger graph

Transferability of Graph Neural Networks

- Transferability of graph neural networks is ready to verify in practice $\Rightarrow$ recommendation system

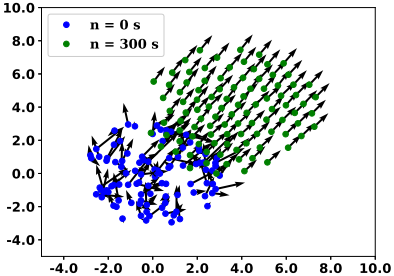


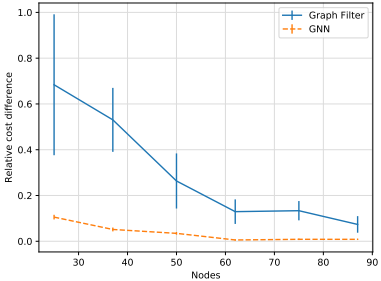


- Performance difference on training and target graphs decreases as size of training graph grows
- GNNs appear to be more transferable than graph convolutional filters $\Rightarrow$ better ML model

Transferability of Graph Neural Networks

- Transferability of graph neural networks is ready to verify in practice $\Rightarrow$ decentralized robot control





- Performance difference on training and target graphs decreases as size of training graph grows
- GNNs appear to be more transferable than graph convolutional filters $\Rightarrow$ better ML model

Do Graph Neural Networks Scale?

Q1: We have empirically observed that GNNs transfer at scale. Why do they?

Q2: Can success of GNNs on moderate-size graphs be used to create success at large-scale?

- To answer these questions, turn to CNNs $\Rightarrow$ known to scale well for images and time sequences

Convolutional Neural Networks Have Limits

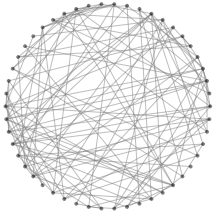
- Discrete time/image signals converge to continuous time/image signals $\Rightarrow \downarrow$ intrinsic dimension

143×95 $\rightarrow$ 205×136 $\rightarrow$ 294×195 $\rightarrow$ 600×399

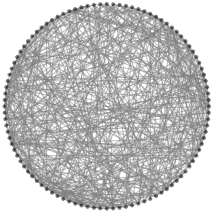
$\Rightarrow$ From SP theory, CNNs have well-defined limits on the limits of images and time signals

- A1: Intrinsic dimensionality of the problem is less than the size of the image
- A2: Training with small images is sufficient $\Rightarrow$ CIFAR 10 images are 32×32

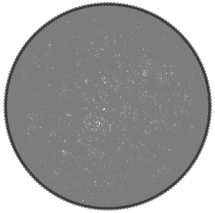
- Graphs also have limit objects that effectively limit their dimensionality $\Rightarrow$ one is the graphon



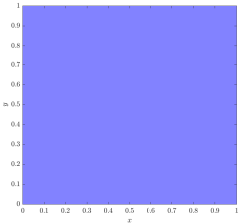
$n = 50$ nodes $\rightarrow$



$n = 100$ nodes $\rightarrow$



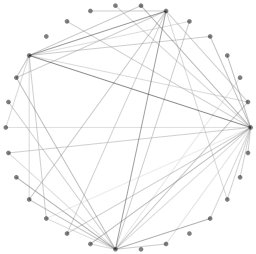
$n = 200$ nodes $\rightarrow$



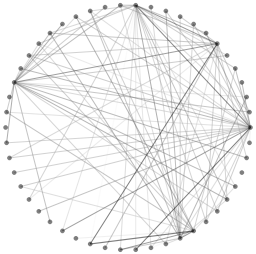
Graphon $W(u, v) = p$

- A graphon can be thought of as a graph with an uncountable number of nodes

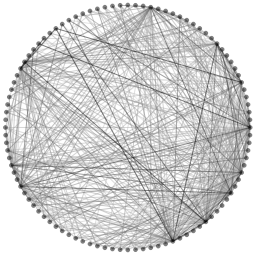
- Graphs however do not have the Euclidean structure time and image signals have in the limit



$n = 30$ products



$n = 50$ products



$n = 100$ products

- So do graph convolutions and graph neural networks converge to limits on the graphon?

Graph Neural Networks Have Limits

Q1: We have empirically observed that GNNs scale. Why do they scale?

- A1: Because graph convolutions and GNNs have well-defined limits on graphons

L. Ruiz et al, *Graphon Signal Processing*, TSP 2021, https://arxiv.org/abs/2003.05030

L. Ruiz et al, *Transferability Properties of Graph Neural Networks*, https://arxiv.org/abs/2112.04629

Q2: Can success of GNNs on moderate-size graphs be used to create success at large-scale?

- A2: Yes, as GNNs are transferable $\Rightarrow$ can be trained on moderate-size and executed on large-scale

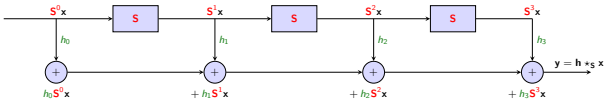
J. Cerviño et al, *Learning by Transference: Training Graph Neural Networks on Growing Graphs.*, https://arxiv.org/abs/2106.03693

Graphon filters and Graphon Neural Networks (WNNs)

Graphon convolutional filters and graph convolutional filters are the same algebraic object. Which is also the same algebraic object of a standard convolutional filter.

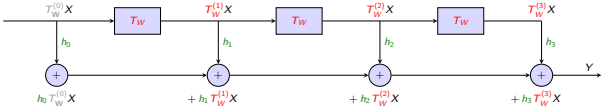
Graph convolutional filters are polynomials on a matrix representation of the graph acting on input signal.

The coefficients of the filter are the coefficients of the polynomial.



Graphon convolutional filters are polynomials on the graphon integral operator acting on input signal.

The coefficients of the filter are the coefficients of the polynomial.



Graphon integral operator: $T_W X \;:\; (T_W)X(v) \;=\; \int_0^1 \mathbf{W}(u,v)\,X(u)\,du$

WNNs are compositions of layers. Themselves compositions of graphon filters with pointwise nonlinearities

[Ruiz et al '20] Graphon Signal Processing, https://arxiv.org/abs/2003.05030

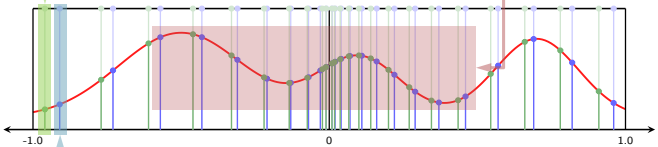
Frequency Representation of Graphon Filters

Graphon filters admit a frequency representation. Same as graph filters. Same as standard convolutions

They are still the same algebraic object: They are polynomials of scalar variables

Representation of graph filter is instantiated at graph eigenvalues

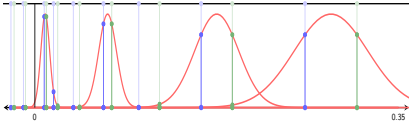
Representation of graphon filter is instantiated at graphon eigenvalues



Since graph eigenvalues converge to graphon eigenvalues convergence of graph to graphon filters follows.

The catch is that we have accumulation of eigenvalues around zero.

Thus, we can't transfer filters that attempt to discriminate these eigenvalues. There is a transferability vs discriminability tradeoff



[Ruiz et al '21] Transferability Properties of Graph Neural Networks, https://arxiv.org/abs/2112.04629

Transferability of Graph Filters and GNNs

We derive a finite sample transferability bound from a graph with m nodes to a graph with n nodes

Transferability of a filter depends on the Lipschitz constant of the frequency response of the graph (and graphon) filter

Theorem (Graph Filter Transferability)

Consider graph signals $(\mathbf{S}_n, \mathbf{x}_n)$ and $(\mathbf{S}_m, \mathbf{x}_m)$ sampled from graphon signal (W, X) along with convolution outputs $\mathbf{y}_n = \mathbf{H}(\mathbf{S}_n)\mathbf{x}_n$ and $\mathbf{y}_m = \mathbf{H}(\mathbf{S}_m)\mathbf{x}_m$. The difference norm of the respective graphon induced signals is bounded by

$$\|Y_n - Y_m\| \leq 2A_x \left(A_h + \pi \frac{\max(B_{nc}, B_{mc})}{\min(\delta_{nc}, \delta_{mc})} \right) \left(\frac{1}{n} + \frac{1}{m} \right) \|X\| + A_x(A_h c + 2) \left(\frac{1}{n} + \frac{1}{m} \right) + 4A_h c \|X\|$$

Same bound holds for GNNs because the pointwise nonlinearity transfers verbatim because it does not mix components

[Ruiz et al '20] Graphon Neural Networks and the Transferability of Graph Neural Networks, https://papers.nips.cc/paper/2020/hash/12bcd658ef0a540cabc36cdf2b1046fd-Abstract.html

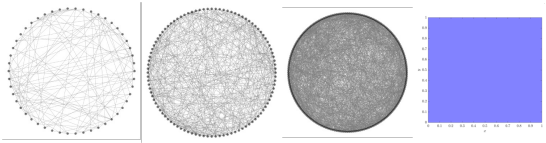
[Ruiz et al '21] Transferability Properties of Graph Neural Networks, https://arxiv.org/abs/2112.04629

Learning by Transference in Stochastic Graph Models

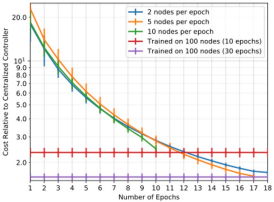
Transferability can be leveraged to learn in a sequence of growing graphs. We say that we learn by transference.

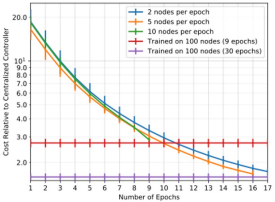
We consider graphs of growing sizes and use the GNN trained on a smaller graph as a warm start to learn the optimal GNN for a larger graph.

Faster training. Enables training in large scale graphs.



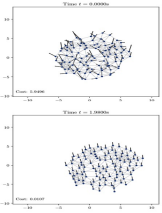
10^2 nodes 10^3 nodes 10^4 nodes graphon





Training with growing graphs learns GNNs with the same performance

Computational cost is reduced by a 5.67 factor. More possible if graph is larger



[Cerviño et al '21] Learning by Transference: Training Graph Neural Networks on Growing Graphs, https://arxiv.org/abs/2106.03693

Graph Neural Networks Architectures, Stability, and Transferability

- Graph neural networks compose layers, which compose graph filters with pointwise nonlinearities
- Graph filters are algebraically identical to standard convolutions $\Rightarrow$ Polynomials $\equiv$ Compositions
- Graph filters are stable to deformations of the graph that are close to perturbations
- Graph filters are transferable from medium scale graph to large scale graphs
- Stability and transferability properties follow from spectral representations of graph filters

- Important real life application problems are naturally associated to data with high dimensionality

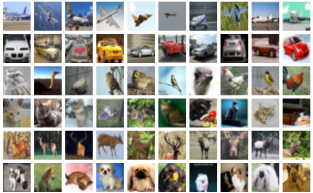
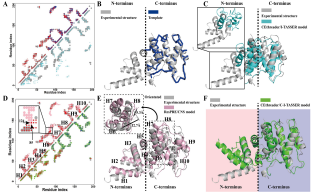


Image Classification
A. Krizhevsky, *CIFAR*, 2009

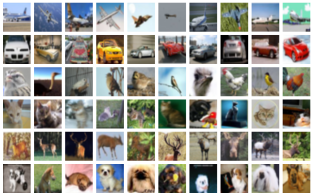
Controlling Robot Swarms
E. Tolstaya, et al., , arxiv.org/abs/1903.10527



Protein Structure Prediction
J. Jumper, et. al., *Nature vol 596*, 2021

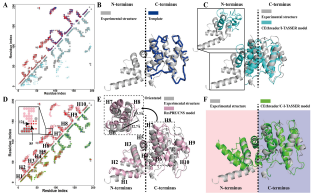
- Scalable learning is difficult $\Rightarrow$ Learning with high dimensional inputs is (much) more challenging

- We know we can learn at scale with Convolutional Neural Networks adapted to several domains



Euclidean
Convolutional Neural Networks

Graph
Convolutional Neural Networks



Group
Convolutional Neural Networks

- One reason why CNNs are effective solutions $\Rightarrow$ Symmetries and equivariances on each domain

All convolutions share a common algebraic structure from which they inherit common stability properties and (perhaps) common transferability properties.

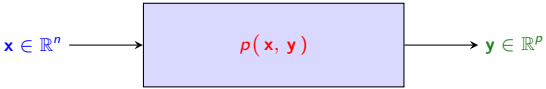
Parada Mayorga-Ribeiro , *Algebraic Neural Networks: Stability to Deformations*, arxiv.org/abs/2009.01433

Statistical Learning

- Before we talk about GNNs, we need to specify what we mean by learning

  $\Rightarrow$ Statistical Learning and Empirical Learning

- Observations (inputs) $\mathbf{x}$ and information (outputs) $\mathbf{y}$ are related by a statistical model $p(\mathbf{x}, \mathbf{y})$



- Given that the universe (nature) associates inputs $\mathbf{x}$ and outputs $\mathbf{y}$ according to distribution $p(\mathbf{x}, \mathbf{y})$

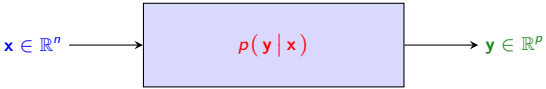
  $\Rightarrow$ The AI should predict $\mathbf{y}$ from $\mathbf{x}$ with the conditional distribution $\Rightarrow \mathbf{y} \sim p(\mathbf{y} \mid \mathbf{x})$

  $\Rightarrow$ Or, if we want deterministic output, a conditional expectation $\Rightarrow \mathbf{y} = \mathbb{E}[\mathbf{y} \mid \mathbf{x}]$

- There is a lot to say about statistical estimation but this is beyond the scope of this course

Statistical Models and Statistical Estimation

- Observations (inputs) $\mathbf{x}$ and information (outputs) $\mathbf{y}$ are related by a statistical model $p(\mathbf{x}, \mathbf{y})$



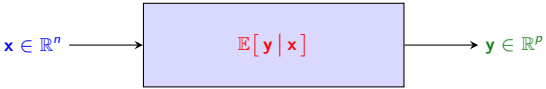
- Given that the universe (nature) associates inputs $\mathbf{x}$ and outputs $\mathbf{y}$ according to distribution $p(\mathbf{x}, \mathbf{y})$

  $\Rightarrow$ The AI should predict $\mathbf{y}$ from $\mathbf{x}$ with the conditional distribution $\Rightarrow \mathbf{y} \sim p(\mathbf{y} \mid \mathbf{x})$

  $\Rightarrow$ Or, if we want deterministic output, a conditional expectation $\Rightarrow \mathbf{y} = \mathbb{E}[\mathbf{y} \mid \mathbf{x}]$

- There is a lot to say about statistical estimation but this is beyond the scope of this course

- Observations (inputs) $\mathbf{x}$ and information (outputs) $\mathbf{y}$ are related by a statistical model $p(\mathbf{x}, \mathbf{y})$



- Given that the universe (nature) associates inputs $\mathbf{x}$ and outputs $\mathbf{y}$ according to distribution $p(\mathbf{x}, \mathbf{y})$

  $\Rightarrow$ The AI should predict $\mathbf{y}$ from $\mathbf{x}$ with the conditional distribution $\Rightarrow \mathbf{y} \sim p(\mathbf{y} \mid \mathbf{x})$

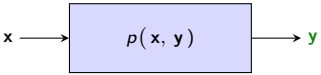
  $\Rightarrow$ Or, if we want deterministic output, a conditional expectation $\Rightarrow \mathbf{y} = \mathbb{E}[\mathbf{y} \mid \mathbf{x}]$

- There is a lot to say about statistical estimation but this is beyond the scope of this course

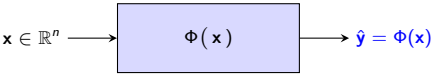
Estimation Loss

- AI is not perfect. Nature and AI may produce different outputs when presented with the same input

Nature relates $\mathbf{x}$ and $\mathbf{y}$ with distribution $p(\mathbf{x}, \mathbf{y})$



The AI relates $\mathbf{x}$ and $\hat{\mathbf{y}}$ with function $\Phi(\mathbf{x})$



- Loss function $\ell(\mathbf{y}, \hat{\mathbf{y}}) = \ell(\mathbf{y}, \Phi(\mathbf{x}))$ measures cost of predicting $\hat{\mathbf{y}} = \Phi(\mathbf{x})$ when actual output is $\mathbf{y}$

  $\Rightarrow$ In estimation problems we often use quadratic loss $\Rightarrow \ell(\mathbf{y}, \hat{\mathbf{y}}) = \|\mathbf{y} - \hat{\mathbf{y}}\|_2^2$

  $\Rightarrow$ In classification problems we often use hit loss $\Rightarrow \ell(\mathbf{y}, \hat{\mathbf{y}}) = \|\mathbf{y} - \hat{\mathbf{y}}\|_0 = \#(\mathbf{y} \neq \hat{\mathbf{y}})$

Statistical Risk Minimization

- Average the loss $\ell(\mathbf{y}, \Phi(\mathbf{x}))$ over nature's distribution $p(\mathbf{x}, \mathbf{y})$ and choose best estimator/classifier

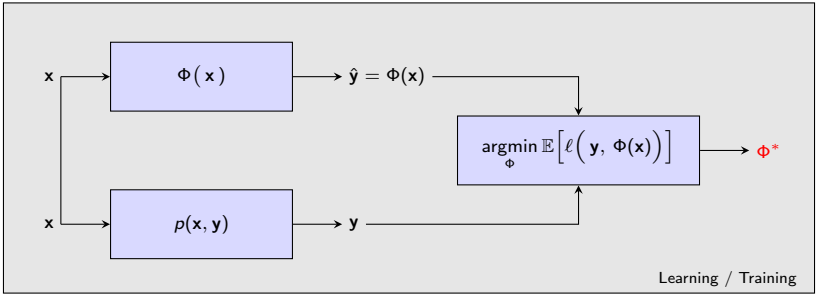
$$\Phi^* = \underset{\Phi}{\operatorname{argmin}} \; \mathbb{E}_{p(\mathbf{x},\mathbf{y})} \Big[\ell\Big(\mathbf{y}, \Phi(\mathbf{x}) \Big) \Big]$$

- Predict $\Phi(\mathbf{x})$. Nature draws $\mathbf{y}$. Evaluate loss ℓ. Take loss expectation over distribution $p(x, y)$

  $\Rightarrow$ Optimal estimator is the function with minimum average cost over all possible estimators.

- This optimization program is called the statistical risk minimization (SRM) problem

- Learning, or Training, is the process of solving the statistical risk minimization problem



- Outcome of learning is function Φ^* with minimum average statistical loss $\Rightarrow$ We learn to estimate $\mathbf{y}$

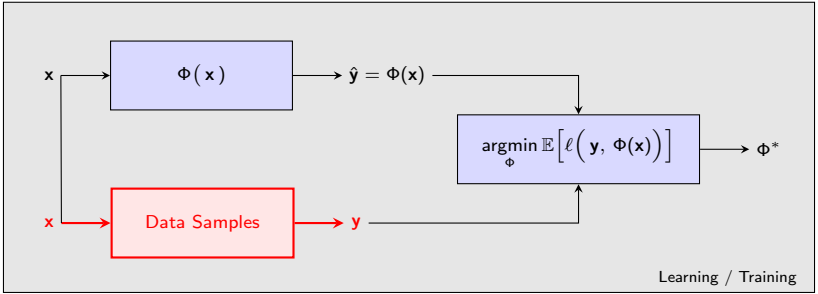
  $\Rightarrow$ During execution time, we just evaluate $\Phi^*(\mathbf{x})$ to predict output associated with input $\mathbf{x}$

Empirical Risk Minimization

- Learning bypasses models. It tries to imitate observations. Let us formulate mathematically.

Artificial Intelligence (AI) / Machine Learning (ML)

- AI and ML in this course refer to the pipeline where we learn from data samples. Not distributions



- AI learns to imitate input-output pairs observed in nature.

Approximating the Statistical Risk

- Statistical Risk Minimization works on the cost averaged over the distribution of inputs and outputs

$$\Phi^* = \underset{\Phi}{\operatorname{argmin}}\, \mathbb{E}_{p(\mathbf{x},\mathbf{y})}\Big[\ell\Big(\mathbf{y},\, \Phi(\mathbf{x})\Big)\Big]$$

- This expectation can be approximated with data

  $\Rightarrow$ Acquire training set with Q pairs $(\mathbf{x}_q, \mathbf{y}_q) \in \mathcal{T}$ drawn independently from distribution $p(\mathbf{x}, \mathbf{y})$

  $\Rightarrow$ For sufficiently large Q we can approximate $\Rightarrow \mathbb{E}_{p(\mathbf{x},\mathbf{y})}\Big[\ell\Big(\mathbf{y},\, \Phi(\mathbf{x})\Big)\Big] \approx \dfrac{1}{Q}\displaystyle\sum_{q=1}^{Q} \ell\Big(\mathbf{y}_q,\, \Phi(\mathbf{x}_q)\Big)$

  $\Rightarrow$ This is just the law of large numbers. True under very mild conditions

Empirical Risk Minimization (ERM)

- Replace statistical risk minimization (SRM) with empirical risk minimization (ERM)

$$\Phi_S^* = \underset{\Phi}{\operatorname{argmin}}\, \mathbb{E}_p(\mathbf{x}, \mathbf{y})\Big[\ell\Big(\mathbf{y}, \Phi(\mathbf{x})\Big)\Big] \quad \Rightarrow \quad \Phi_E^* = \underset{\Phi}{\operatorname{argmin}}\, \frac{1}{Q}\sum_{q=1}^{Q} \ell\Big(\mathbf{y}_q, \Phi(\mathbf{x}_q)\Big)$$

- Since the objectives are close, one would think the optima are close $\Rightarrow \Phi_S^* \approx \Phi_E^*$

  $\Rightarrow$ Alas, this it not true $\Rightarrow \Phi_S^* \not\approx \Phi_E^*$ $\Rightarrow$ Statistical and empirical risk minimizers need not be close

- In fact, the solution of ERM is trivial $\Rightarrow$ Make $\Phi(\mathbf{x}_q) = \mathbf{y}_q$ for all pairs in the training set

- As trivial as nonsensical $\Rightarrow$ Yields no information about observations outside the training set

ERM with Learning Parametrizations

- Our first attempt at learning from data led to an ERM problem that does not make sense
- The search for a problem that makes sense brings us to the notion of learning parametrizations

Learning Parametrization

- A sensical ERM problem, requires the introduction of a function class $\mathcal{C}$

$$\Phi^* = \underset{\Phi \in \mathcal{C}}{\operatorname{argmin}} \frac{1}{Q} \sum_{q=1}^{Q} \ell\Big(\mathbf{y}_q, \Phi(\mathbf{x}_q) \Big)$$

- For example, we can select the class of linear functions $\Phi(\mathbf{x}) = \mathbf{H}\mathbf{x}$ and solve for

$$\mathbf{H}^* = \underset{\mathbf{H}}{\operatorname{argmin}} \frac{1}{Q} \sum_{q=1}^{Q} \ell\Big(\mathbf{y}_q, \mathbf{H}\,\mathbf{x}_q \Big)$$

- This choice of parametrization may be good or bad. But at least is sensical

  $\Rightarrow$ Good or bad, having $\mathbf{H}^*$ allows estimates $\hat{\mathbf{y}} = \mathbf{H}^*\mathbf{x}$ for observations $\mathbf{x}$ outside the training set

- Selecting $\mathcal{C}$ to contain sufficiently smooth functions makes SRM and ERM close

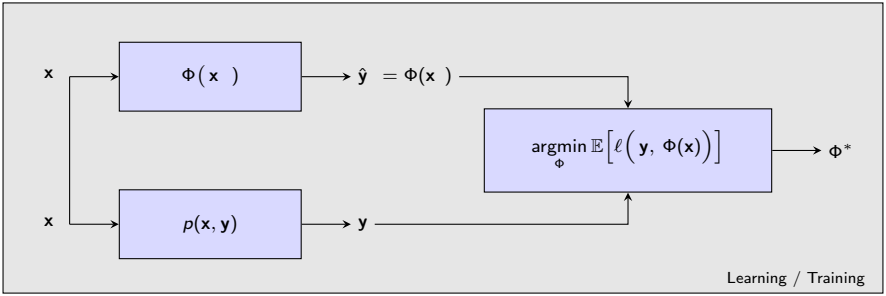
$$\underset{\Phi \in \mathcal{C}}{\text{argmin}}\, \mathbb{E}_p(\mathbf{x}, \mathbf{y}) \Big[\ell\Big(\mathbf{y},\, \Phi(\mathbf{x}) \Big) \Big] \quad \approx \quad \underset{\Phi \in \mathcal{C}}{\text{argmin}}\, \frac{1}{Q} \sum_{q=1}^{Q} \ell\Big(\mathbf{y}_q,\, \Phi(\mathbf{x}_q) \Big)$$

- Fundamental theorem of statistical learning $\Rightarrow$ ERM is a valid approximation of SRM

- Need to identify the appropriate function class $\mathcal{C}$ $\Rightarrow$ But this problem is unavoidable

Learning / Training (For Real)

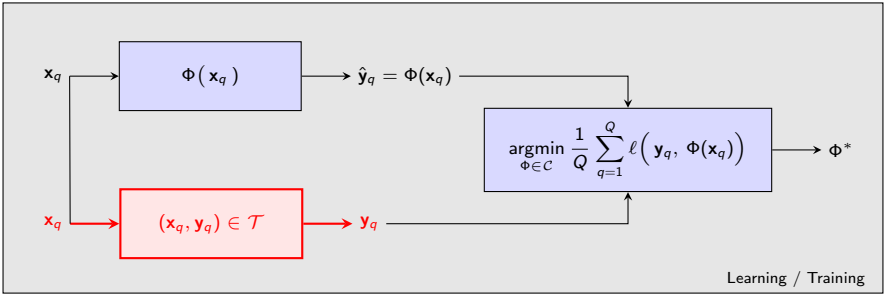
- SRM learns from model $\Rightarrow$ Parametrized ERM learns from data $\Rightarrow$ Three differences:



Learning / Training (For Real)

- SRM learns from model $\Rightarrow$ Parametrized ERM learns from data $\Rightarrow$ Three differences:

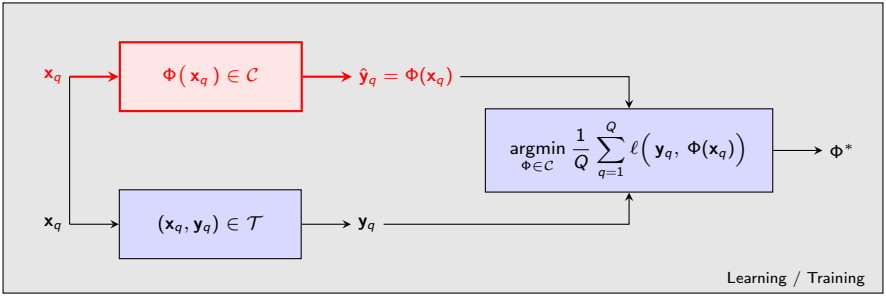
  $\Rightarrow$ The distribution is unknown $\Rightarrow$ We have access to a training set of data samples



Learning / Training (For Real)

- SRM learns from model $\Rightarrow$ Parametrized ERM learns from data $\Rightarrow$ Three differences:

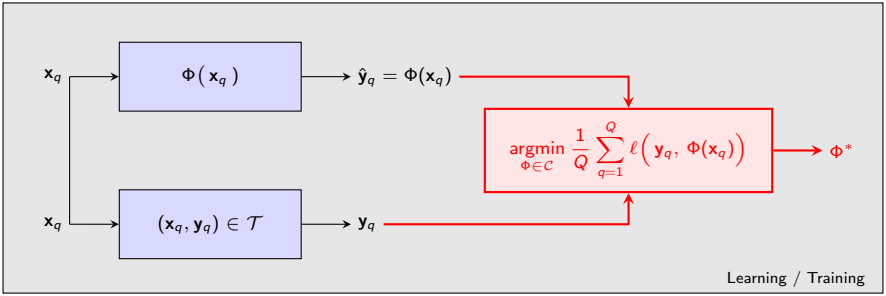
  $\Rightarrow$ The nonparametric ERM problem is nonsensical $\Rightarrow$ We restrict the function class



Learning / Training (For Real)

- SRM learns from model $\Rightarrow$ Parametrized ERM learns from data $\Rightarrow$ Three differences:

  $\Rightarrow$ The statistical risk $\Rightarrow$ Is replaced by the empirical risk



The Meaning of ML or AI in this Course

- Here, Machine learning (ML) $\equiv$ Artificial Intelligence (AI) $\equiv$ Empirical Risk Minimization (ERM)

$$\mathbf{\Phi}^* = \underset{\mathbf{\Phi}\in\mathcal{C}}{\text{argmin}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \ell\Big(\mathbf{y}, \mathbf{\Phi}(\mathbf{x})\Big) = \underset{\mathbf{\Phi}\in\mathcal{C}}{\text{argmin}} \frac{1}{Q}\sum_{q=1}^{Q} \ell\Big(\mathbf{y}_q, \mathbf{\Phi}(\mathbf{x}_q)\Big)$$

- The components of ERM are a dataset, a loss function and, most importantly, a function class

- Make parametrization more explicit $\Rightarrow$ Parameter $\mathbf{H} \in \mathbb{R}^p$ to span function class $\Phi(\mathbf{x}; \hat{\mathbf{H}})$

$$\mathbf{H}^* = \underset{\mathbf{H}}{\text{argmin}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \ell\Big(\mathbf{y}, \mathbf{\Phi}(\mathbf{x}; \mathbf{H})\Big)$$

- Designing an ML / AI system means selecting the appropriate function class $\mathcal{C}$ $\Rightarrow$ What else?

  $\Rightarrow$ The function class determines generalization from inputs in training set to unseen inputs

Machine Learning is Model Free but Not Model Free

- Machine learning does not require a model relating inputs $\mathbf{x}$ to outputs $\mathbf{y}$
- But we need to know a class of functions to which the model belongs
  - $\Rightarrow$ For example, we need to know the model relating inputs to outputs is linear
- Model also needs to be sufficiently simple to operate with insufficient data
  - $\Rightarrow$ This is where we leverage structure using convolutional architectures such as CNNs and GNNs

Learning Ratings in Recommendation Systems

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

Recommendation Systems

- 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

Ratings and Sampled Ratings

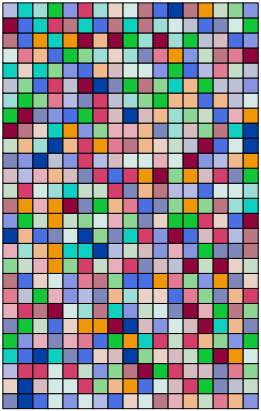
- For all items i and users u there exist ratings $\Rightarrow y_{ui}$

  $\Rightarrow$ 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



Ratings and Sampled Ratings

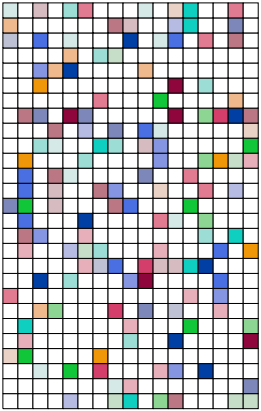
- For all items i and users u there exist ratings $\Rightarrow y_{ui}$

  $\Rightarrow$ 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 $\mathbf{y}_u$ of user u as a graph signal supported on the product similarity graph
- The observed ratings $\mathbf{x}_u$ of user u are a subsampling of this graph signal.
- Our goal is to learn to reconstruct the rating graph signal $\mathbf{y}_u$ from the observed ratings $\mathbf{x}_u$
- Build similarity graph using available ratings. Use of expert knowledge is common as well

Product Similarity Graph

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

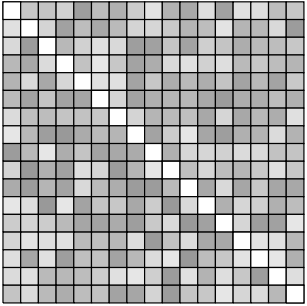
- Mean ratings restricted to users that rated products i and j

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

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



Loss for Measuring Rating Prediction Quality

- Given observed ratings $\mathbf{x}_u$ the AI produces estimates $\Phi(\mathbf{x}_u)$. We want $\Phi(\mathbf{x}_u)$ to approximate $\mathbf{y}_u$

$$\ell\Big(\mathbf{y}_u, \Phi(\mathbf{x}_u)\Big) = \frac{1}{2}\Big\| \mathbf{y}_u - \Phi(\mathbf{x}_u) \Big\|^2$$

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

$$\ell\Big(\mathbf{y}_u, \Phi(\mathbf{x}_u)\Big) = \frac{1}{2}\Big(\mathbf{e}_i^T \mathbf{y}_u - \mathbf{e}_i^T \Phi(\mathbf{x}_u) \Big)^2$$

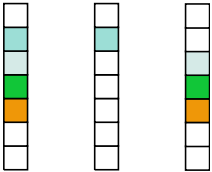
- Where $\mathbf{e}_i$ is a vector in the canonical basis $\Rightarrow (\mathbf{e}_i)_i = 1$, $(\mathbf{e}_i)_j = 0$ for $j \neq i$

Training Set

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

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

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



Learning Rating Predictions

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

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

- 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

Movie Ratings Dataset

- 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.

Empirical Risk Minimization

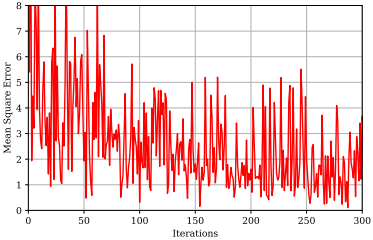
- We predict ratings using AI that results from solving the ERM problem

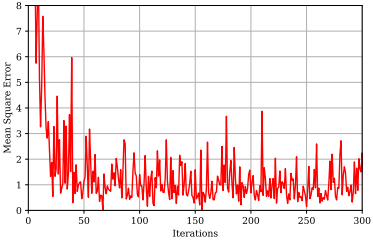
$$\mathcal{H}^* = \underset{\mathcal{H}}{\text{argmin}} \sum_{(x,y)\in\mathcal{T}} \left(\mathbf{e}_i^T \mathbf{y} - \mathbf{e}_i^T \Phi(\mathbf{x}; \mathcal{H}) \right)^2$$

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

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

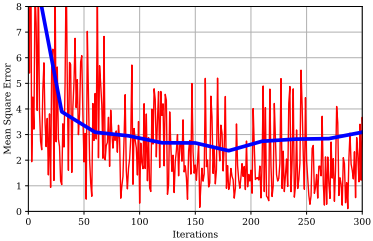
- Linear regression reduces training MSE to about 2. Quite bad for ratings that vary from 0 to 5
- Graph filter reduces training MSE to about 1. Not too good. Humans are not that predictable

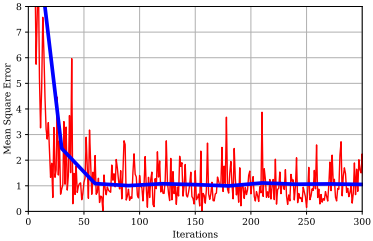




- Graph filter outperforms linear regression $\Rightarrow$ Leverages underlying permutation symmetries

- 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

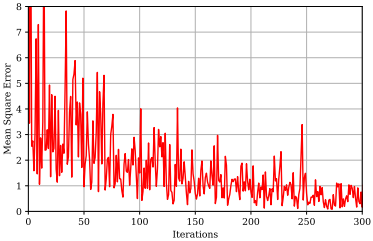


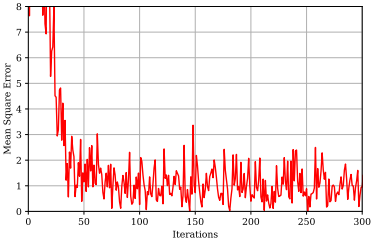


- Graph filter outperforms linear regression $\Rightarrow$ Leverages underlying permutation symmetries

Fully Connected NNs and Graph NNs

- 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

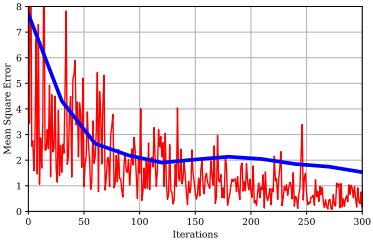


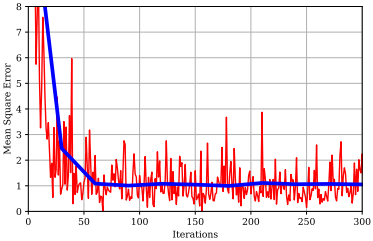


- Graph NN outperforms fully connected NN $\Rightarrow$ Leverages underlying permutation symmetries

Fully Connected NNs and Graph NNs

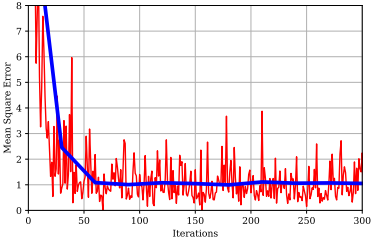
- But the fully connected NN does not do well in the test set. It does not generalize
- The test MSE of the graph NN is about the same as the training MSE. It generalizes

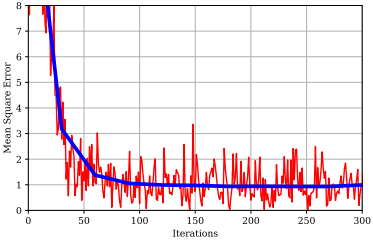




- Graph NN outperforms fully connected NN $\Rightarrow$ Leverages underlying permutation symmetries

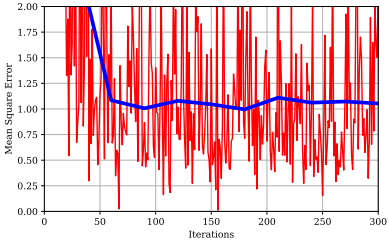
- The graph filter and the GNN do well in the training and test set. They generalize well
- The GNN does a little better. Not by much. But an extra 10% is not irrelevant

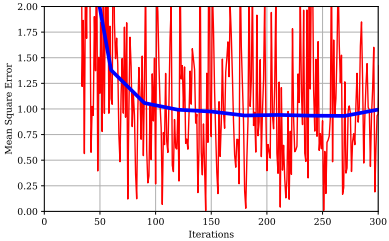




- GNN outperforms graph filter $\Rightarrow$ The GNN has a better stability-discriminability tradeoff

- 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



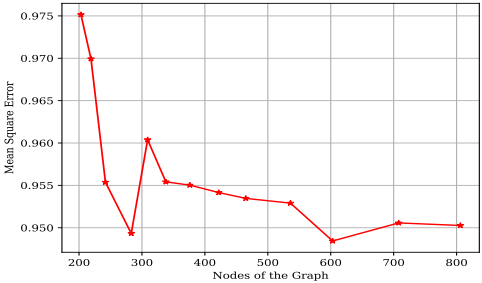


- GNN outperforms graph filter $\Rightarrow$ The GNN has a better stability-discriminability tradeoff

Transferability

- 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



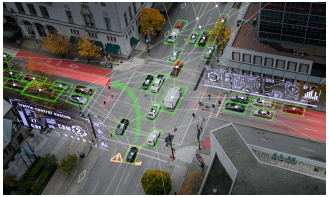
- In this recommendation system, transference incurs no MSE degradation $\Rightarrow$ MSE is further reduced

Wireless Resource Management with GNNs

- GNNs can enable scalable resource management in autonomous wireless communication networks.

Towards Next-Generation Wireless Networks

- Wireless networks are growing beyond humans' ability to design and manage them → 5G, WiFi 6

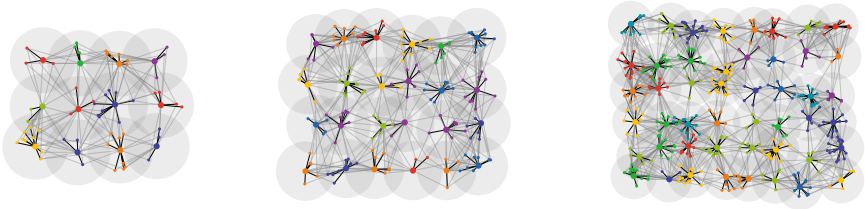


- To address increasing complexity of wireless networks, we will make them autonomous → 6G, WiFi 7

  ⇒ An autonomous wireless network makes (at least some) decisions without human intervention.

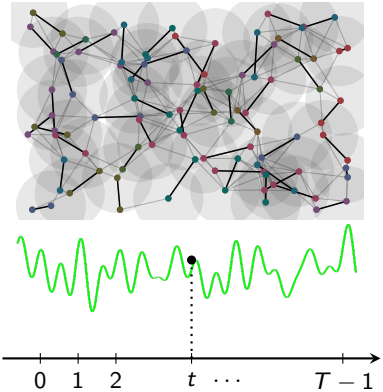
Autonomous Wireless Networks

- Operational decisions in wireless networks are solutions of large constrained optimization problems.
- Solving these problems is very challenging, leading to the design and use of heuristic methods.



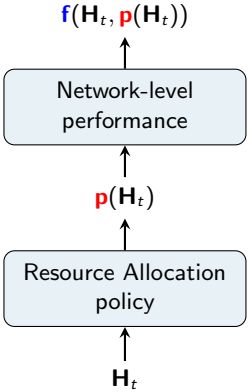
- Leverage data to learn better autonomous network management policies using machine learning.

Wireless Resource Allocation Under Requirements



$$\max_{\{\mathbf{p}(\mathbf{H}_t)\}_{t=0}^{T-1}} \quad \mathcal{U}\left(\frac{1}{T}\sum_{t=0}^{T-1} \mathbf{f}(\mathbf{H}_t, \mathbf{p}(\mathbf{H}_t))\right)$$

$$\text{s.t.} \quad \mathbf{g}\left(\frac{1}{T}\sum_{t=0}^{T-1} \mathbf{f}(\mathbf{H}_t, \mathbf{p}(\mathbf{H}_t))\right) \geq \mathbf{0}$$



NaderiAlizadeh-Eisen-Ribeiro, *State-Augmented Learnable Algorithms for Resource Management in Wireless Networks*, IEEE TSP, arxiv.org/abs/2207.02242

Policy Parameterization

- Resource allocation decisions must be recalculated for any given network state $\mathbf{H}$.

  $\Rightarrow$ This makes learning and deploying such a policy infeasible in practice.

- We parameterize the resource allocation policy, replacing $\mathbf{p}(\mathbf{H})$ with $\mathbf{p}(\mathbf{H};\boldsymbol{\theta})$.

- With parameterization, we do not need to solve the problem online to find optimal decisions.

Unparameterized Formulation

$$\max_{\{\mathbf{p}(\mathbf{H}_t)\}_{t=0}^{T-1}} \quad \mathcal{U}\left(\frac{1}{T}\sum_{t=0}^{T-1}\mathbf{f}(\mathbf{H}_t,\mathbf{p}(\mathbf{H}_t))\right)$$
$$\text{s.t.} \quad \mathbf{g}\left(\frac{1}{T}\sum_{t=0}^{T-1}\mathbf{f}(\mathbf{H}_t,\mathbf{p}(\mathbf{H}_t))\right) \geq \mathbf{0}$$

Parameterized Formulation

$$P^\star = \max_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \quad \mathcal{U}\left(\frac{1}{T}\sum_{t=0}^{T-1}\mathbf{f}(\mathbf{H}_t,\mathbf{p}(\mathbf{H}_t;\boldsymbol{\theta}))\right)$$
$$\text{s.t.} \quad \mathbf{g}\left(\frac{1}{T}\sum_{t=0}^{T-1}\mathbf{f}(\mathbf{H}_t,\mathbf{p}(\mathbf{H}_t;\boldsymbol{\theta}))\right) \geq \mathbf{0}$$

This is an Unsupervised Learning Problem

Empirical Risk Minimization

$$\max_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \quad -\frac{1}{N}\sum_{i=0}^{N-1} \ell\left(\psi\left(\mathbf{x}_i;\boldsymbol{\theta}\right)\right)$$

Parameterized Resource Allocation

$$\max_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \quad \mathcal{U}\left(\frac{1}{T}\sum_{t=0}^{T-1}\mathbf{f}(\mathbf{H}_t,\mathbf{p}(\mathbf{H}_t;\boldsymbol{\theta}))\right)$$

$$\text{s.t.} \quad \mathbf{g}\left(\frac{1}{T}\sum_{t=0}^{T-1}\mathbf{f}(\mathbf{H}_t,\mathbf{p}(\mathbf{H}_t;\boldsymbol{\theta}))\right) \geq \mathbf{0}$$

- Inclusion of constraints makes this problem fundamentally different from a regular learning problem.

Learning in the Lagrangian Dual Domain

- We move to the dual domain, and associate non-negative dual variables $\boldsymbol{\mu}$ to the constraints.

- The Lagrangian function can then be written as

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\mu}) = \mathcal{U}\left(\frac{1}{T}\sum_{t=0}^{T-1}\mathbf{f}(\mathbf{H}_t, \mathbf{p}(\mathbf{H}_t; \boldsymbol{\theta}))\right) + \boldsymbol{\mu}^T\mathbf{g}\left(\frac{1}{T}\sum_{t=0}^{T-1}\mathbf{f}(\mathbf{H}_t, \mathbf{p}(\mathbf{H}_t; \boldsymbol{\theta}))\right).$$

- We then seek to maximize the Lagrangian over $\boldsymbol{\theta}$, while minimizing it over $\boldsymbol{\mu}$, i.e.,

$$D^\star = \min_{\boldsymbol{\mu}\geq 0}\max_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\mu}).$$

Iterative Unsupervised Primal-Dual Updates

- The model parameters $\boldsymbol{\theta}$ and dual variables $\boldsymbol{\mu}$ can be iteratively updated via a primal-dual method.

- We define an iteration duration T_0 between consecutive updates, and an iteration index k.

$$\boldsymbol{\theta}_k = \arg\max_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \left[\mathcal{U}\left(\frac{1}{T_0} \sum_{t=kT_0}^{(k+1)T_0-1} \mathbf{f}(\mathbf{H}_t, \mathbf{p}(\mathbf{H}_t; \boldsymbol{\theta})) \right) + \boldsymbol{\mu}_k^T \mathbf{g}\left(\frac{1}{T_0} \sum_{t=kT_0}^{(k+1)T_0-1} \mathbf{f}(\mathbf{H}_t, \mathbf{p}(\mathbf{H}_t; \boldsymbol{\theta})) \right) \right]$$

$$\boxed{k \leftarrow k+1}$$

$$\boldsymbol{\mu}_{k+1} = \left[\boldsymbol{\mu}_k - \eta_{\boldsymbol{\mu}} \mathbf{g}\left(\frac{1}{T_0} \sum_{t=kT_0}^{(k+1)T_0-1} \mathbf{f}(\mathbf{H}_t, \mathbf{p}(\mathbf{H}_t; \boldsymbol{\theta}_k)) \right) \right]_+$$

- Constraint slacks are the gradient of the Lagrangian with respect to the dual variables.

Theoretical Guarantees of Primal-Dual Updates

Theorem (NaderiAlizadeh-Eisen-Ribeiro)

The sequence of decisions made by the primal-dual updates is both feasible, i.e.,

$$\lim_{T\to\infty} \mathbf{g}\left(\frac{1}{T}\sum_{t=0}^{T-1} \mathbf{f}\left(\mathbf{H}_t, \mathbf{p}\left(\mathbf{H}_t; \boldsymbol{\theta}_{\lfloor t/T_0 \rfloor}\right)\right)\right) \geq \mathbf{0}, \quad a.s.$$

and near-optimal, i.e.,

$$\lim_{T\to\infty} \mathbb{E}\left[\mathcal{U}\left(\frac{1}{T}\sum_{t=0}^{T-1} \mathbf{f}\left(\mathbf{H}_t, \mathbf{p}\left(\mathbf{H}_t; \boldsymbol{\theta}_{\lfloor t/T_0 \rfloor}\right)\right)\right)\right] \geq P^\star - \frac{c\eta_\mu G^2}{2}.$$

- There are no restrictions on the convexity of $\mathbf{f}$ and the parameterization $\mathbf{p}(\cdot;\boldsymbol{\theta})$.
- Time averages of instantaneous performance metrics are feasible and near-optimal.

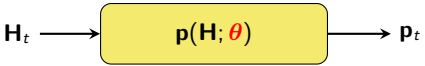
  $\Rightarrow$Time averages of parameters are not near-optimal. We cannot stop training at a finite iteration.

NaderiAlizadeh-Eisen-Ribeiro, *State-Augmented Learnable Algorithms for Resource Management in Wireless Networks*, IEEE TSP, arxiv.org/abs/2207.02242

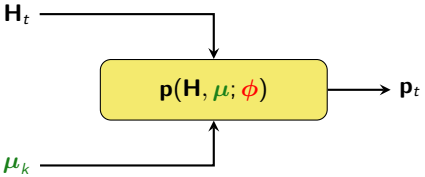
State-Augmented Algorithm

- We propose to use both network state $\mathbf{H}$ and dual variables $\boldsymbol{\mu}$ as input to the policy.
- We leverage a revised state-augmented parameterization $\mathbf{p}(\mathbf{H}, \boldsymbol{\mu}; \boldsymbol{\phi})$ to replace $\mathbf{p}(\mathbf{H}; \boldsymbol{\theta})$.

Regular Parameterization



State-Augmented Parameterization



State-Augmented Primal and Dual Dynamics

- The revised parameterization leads to the augmented Lagrangian

$$\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\mu}) = \mathcal{U}\left(\frac{1}{T}\sum_{t=0}^{T-1} \mathbf{f}(\mathbf{H}_t, \mathbf{p}(\mathbf{H}_t, \boldsymbol{\mu}; \boldsymbol{\phi}))\right) + \boldsymbol{\mu}^T \mathbf{g}\left(\frac{1}{T}\sum_{t=0}^{T-1} \mathbf{f}(\mathbf{H}_t, \mathbf{p}(\mathbf{H}_t, \boldsymbol{\mu}; \boldsymbol{\phi}))\right).$$

- During training, we search for the parameters that maximize the augmented Lagrangian:

$$\boldsymbol{\phi}^\star = \arg\max_{\boldsymbol{\phi} \in \boldsymbol{\Phi}} \mathbb{E}_{\boldsymbol{\mu}}\left[\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\mu})\right].$$

  $\Rightarrow$ This resolves the need to re-optimize the model parameters for any given set of dual variables.

- We use the Lagrangian maximizers to run the dual updates online:

$$\boldsymbol{\mu}_{k+1} = \left[\boldsymbol{\mu}_k - \eta_{\boldsymbol{\mu}} \mathbf{g}\left(\frac{1}{T_0}\sum_{t=kT_0}^{(k+1)T_0-1} \mathbf{f}(\mathbf{H}_t, \mathbf{p}(\mathbf{H}_t, \boldsymbol{\mu}_k; \boldsymbol{\phi}^\star))\right)\right]_+.$$

Theoretical Guarantees of State Augmentation

- ϵ-universal parameterization $\mathbf{p}(\mathbf{H}, \boldsymbol{\mu}; \boldsymbol{\phi})$: For any $\mathbf{H}$ and $\boldsymbol{\theta}(\cdot)$, there exists $\boldsymbol{\phi}$ s.t.

$$\mathbb{E}\left\|\mathbf{p}(\mathbf{H}, \boldsymbol{\mu}; \boldsymbol{\phi}) - \mathbf{p}(\mathbf{H}; \boldsymbol{\theta}(\boldsymbol{\mu}))\right\|_\infty \leq \epsilon.$$

- M-Lipschitz continuity of $\mathbf{f}$: For any $\mathbf{H}$, $\mathbf{p}_1$ and $\mathbf{p}_2$, $\mathbb{E}\left\|\mathbf{f}(\mathbf{H}, \mathbf{p}_1) - \mathbf{f}(\mathbf{H}, \mathbf{p}_2)\right\|_\infty \leq M\mathbb{E}\left\|\mathbf{p}_1 - \mathbf{p}_2\right\|_\infty$.

Theorem (NaderiAlizadeh-Eisen-Ribeiro)

The sequence of decisions made by the proposed state-augmented algorithm is both feasible, i.e.,

$$\lim_{T\to\infty} \mathbf{g}\left(\frac{1}{T}\sum_{t=0}^{T-1} \mathbf{f}\left(\mathbf{H}_t, \mathbf{p}\left(\mathbf{H}_t, \boldsymbol{\mu}_{\lfloor t/T_0 \rfloor}; \boldsymbol{\phi}^\star\right)\right)\right) \geq \mathbf{0}, \quad a.s.$$

and near-optimal, i.e.,

$$\lim_{T\to\infty} \mathbb{E}\left[\mathcal{U}\left(\frac{1}{T}\sum_{t=0}^{T-1} \mathbf{f}\left(\mathbf{H}_t, \mathbf{p}\left(\mathbf{H}_t, \boldsymbol{\mu}_{\lfloor t/T_0 \rfloor}; \boldsymbol{\phi}^\star\right)\right)\right)\right] \geq P^\star - \frac{c\eta_\mu G^2}{2} - M\epsilon.$$

- The decisions made by our method are close to those made by the original primal-dual iterations.

NaderiAlizadeh-Eisen-Ribeiro, *State-Augmented Learnable Algorithms for Resource Management in Wireless Networks*, IEEE TSP, arxiv.org/abs/2207.02242

Power Control in Interference Channels

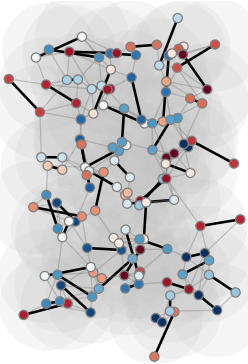
- We focus on multi-user interference channels with m transmitter-receiver pairs.
- The performance function for the ith receiver represents its Shannon capacity,

$$f_i(\mathbf{H}_t, \mathbf{p}) = \log_2\left(1 + \frac{p_i \left|h_{ii,t}\right|^2}{\frac{N}{P_{\max}} + \sum_{j=1, j\neq i}^{m} p_j \left|h_{ji,t}\right|^2}\right).$$

- Considering a sum-rate utility and minimum-rate constraints leads to

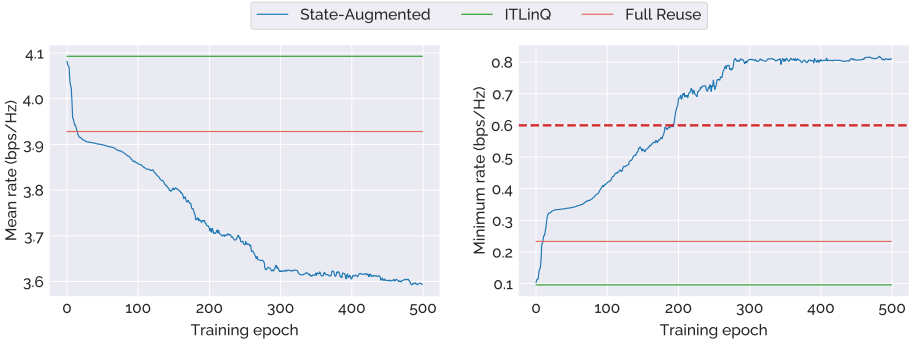
$$\max_{\{\mathbf{p}(\mathbf{H}_t)\}_{t=0}^{T-1}} \quad \frac{1}{T}\sum_{t=0}^{T-1}\sum_{i=1}^{m} f_i(\mathbf{H}_t, \mathbf{p}(\mathbf{H}_t)),$$

$$\text{s.t.} \quad \frac{1}{T}\sum_{t=0}^{T-1} \mathbf{f}(\mathbf{H}_t, \mathbf{p}(\mathbf{H}_t)) \geq f_{\min}\mathbf{1}_m.$$

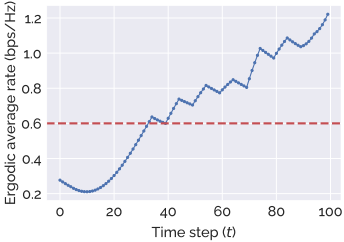


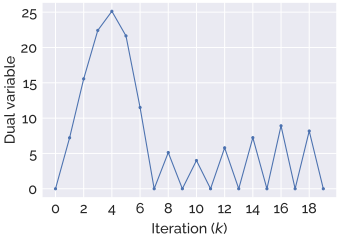
Our Method Satisfies All Minimum-Rate Constraints

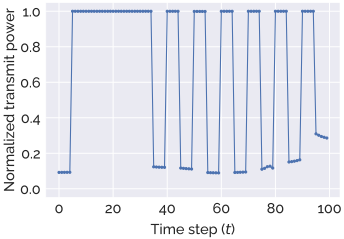
Performance is shown in a 50-user interference channel with minimum-rate constraints of $f_{\min} = 0.6$ bps/Hz.



State Augmentation Leads to Policy Switching





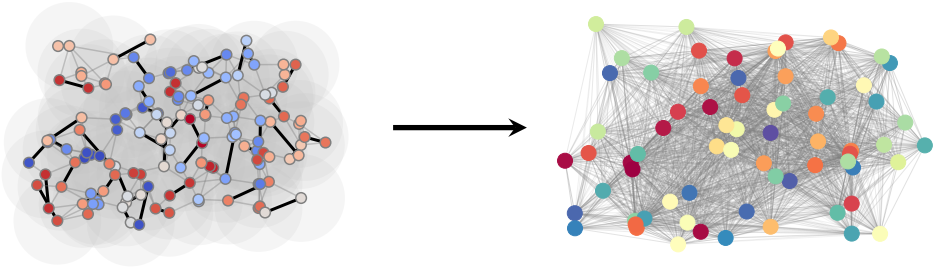


$$\lim_{T\to\infty} \mathbf{g}\left(\frac{1}{T}\sum_{t=0}^{T-1} \mathbf{f}\left(\mathbf{H}_t, \mathbf{p}\left(\mathbf{H}_t, \boldsymbol{\mu}_{\lfloor t/T_0 \rfloor}; \phi^\star\right)\right)\right) \geq \mathbf{0}, \quad a.s.$$

$$\lim_{T\to\infty} \mathbb{E}\left[\mathcal{U}\left(\frac{1}{T}\sum_{t=0}^{T-1} \mathbf{f}\left(\mathbf{H}_t, \mathbf{p}\left(\mathbf{H}_t, \boldsymbol{\mu}_{\lfloor t/T_0 \rfloor}; \phi^\star\right)\right)\right)\right] \geq P^\star - \frac{c\eta_\mu G^2}{2} - M\epsilon.$$

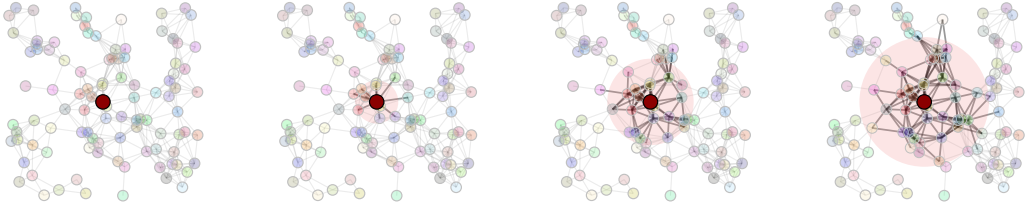
Modeling Interference Channels as Graphs

- We model the interference channel at each time step t as a graph $\mathcal{G}_t = (\mathcal{V}, \mathcal{E}, \mathbf{Y}_t, w_t)$.
  - $\Rightarrow$ $\mathcal{V} = \{1, 2, \ldots, m\}$ denotes the set of transceiver nodes, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes the set of edges.
  - $\Rightarrow$ $\mathbf{Y}_t \in \mathbb{R}^{m \times 1}$ denotes the initial node features, which we set to the dual variables: $\mathbf{Y}_t = \boldsymbol{\mu}_{\lfloor t/T_0 \rfloor}$.
  - $\Rightarrow$ $w_t : \mathcal{E} \to \mathbb{R}$ denotes the edge weight function, which we define as $w_t(i,j) \propto \log\left(P_{\max}|h_{ij,t}|^2/N\right)$.



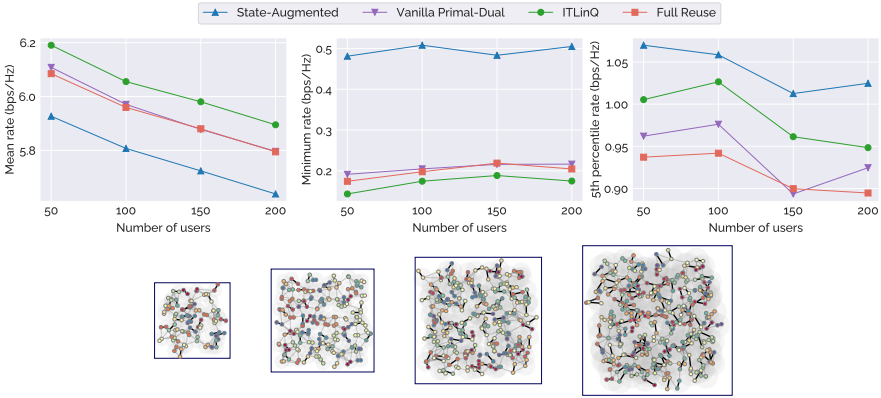
Graph Neural Network (GNN) Parameterizations

- We leverage graph neural networks (GNNs) to parameterize the resource allocation policies.
- Final node features at the output of the GNN are converted to resource allocation decisions.



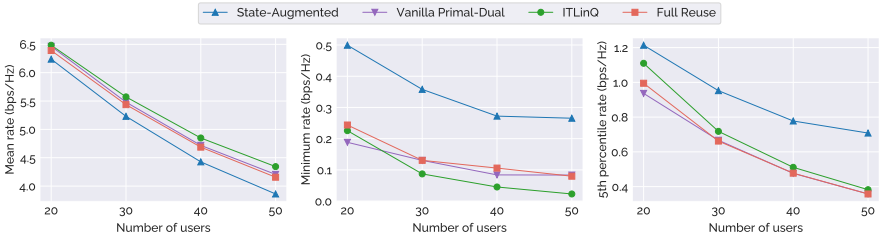
Scalability With Constant Network Density

- The network area size increases proportionally to the number of transmitter-receiver pairs.
- Policies are evaluated on the same network size that they have been trained on.



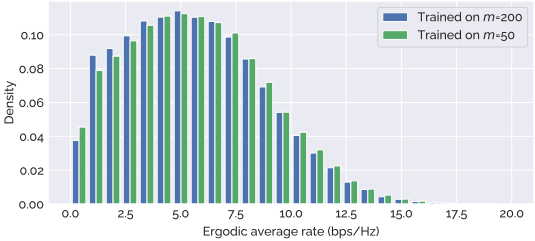
Scalability With Variable Network Density

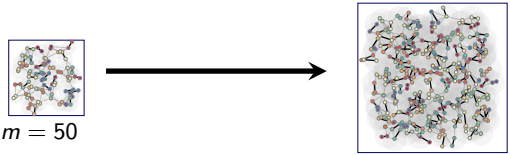
- The network area size is fixed regardless of the number of transmitter-receiver pairs.
- Policies are evaluated on the same network size that they have been trained on.



Transferability With Constant Network Density

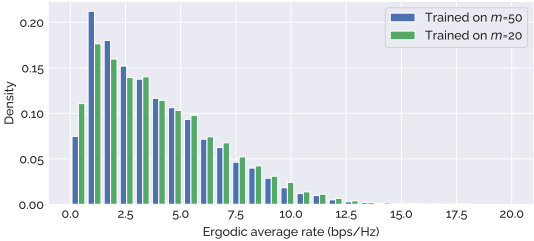
Policies are evaluated on a family of networks with $m = 200$ transmitter-receiver pairs.

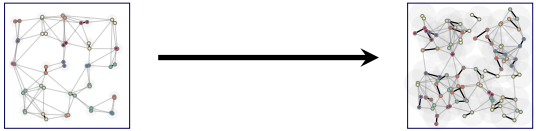




Transferability With Variable Network Density

Policies are evaluated on a family of networks with $m = 50$ transmitter-receiver pairs.





$m = 20$ $m = 50$

Federated Learning with GNNs

- GNNs can enable distributed training of models in a federated learning scenario.

Hadou-NaderiAlizadeh-Ribeiro, *Stochastic Unrolled Federated Learning*, arxiv.org/abs/2305.15371

Federated Learning

- A group of agents attempt to learn a shared model $\mathbf{w}^\star$ with minimium average loss across agents:

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{(\mathbf{x},y)\sim\mathfrak{D}_i}[\ell(f_{\mathbf{w}}(\mathbf{x}), y)].$$

- Considering a graph structure, we can have a constrained formulation:

$$\begin{aligned} \min_{\mathbf{w}_1,\ldots,\mathbf{w}_n\in\mathbb{R}^d} \quad & g(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{(\mathbf{x},y)\sim\mathfrak{D}_i}[\ell(f_{\mathbf{w}_i}(\mathbf{x}), y)], \\ \text{s.t.} \quad & \mathbf{w}_i = \frac{1}{|\mathcal{N}_i|}\sum_{j\in\mathcal{N}_i} \mathbf{w}_j, \quad \text{for all } i = 1,\ldots,N. \end{aligned}$$

- A major challenge: High communication cost between the agents (and a central server).

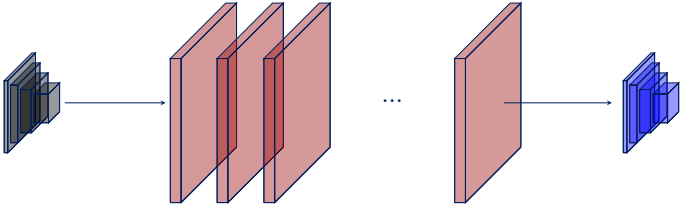
Learning to Optimize via Algorithm Unrolling

- Instead of training the model $\mathbf{W}$ directly, we train a meta model $\mathbf{\Phi}(\mathbf{W}_0, \mathcal{D}; \theta)$, whose output is $\mathbf{W}^\star$:

$$\mathbf{W}^\star = \mathbf{\Phi}(\mathbf{W}_0, \mathcal{D}; \theta^\star) \quad \text{where} \quad \theta^\star = \arg\min_{\theta \in \mathbb{R}^p} \quad \mathbb{E}\big[g(\mathbf{\Phi}(\mathbf{W}_0, \mathcal{D}; \theta))\big].$$

- The meta model takes as input the initial model $\mathbf{W}_0$ and a set of local datasets $\mathcal{D}$.
- We parameterize the meta model using L layers to mimic update rules of an iterative algorithm:

$$\mathbf{W}_l = \phi_l(\mathbf{W}_{l-1}, \mathcal{D}; \boldsymbol{\theta}_l), \quad l = 1, \ldots, L.$$



Stochastic UnRolled Federated learning (SURF)

- Instead of the whole datasets $\mathcal{D}$, we feed stochastic batches of data $\mathcal{B}_l$ to the meta model:

$$\mathbf{W}_l = \phi_l(\mathbf{W}_{l-1}, \mathcal{D}; \boldsymbol{\theta}_l) \quad \rightarrow \quad \mathbf{W}_l = \phi_l(\mathbf{W}_{l-1}, \mathcal{B}_l; \boldsymbol{\theta}_l).$$

- We encourage the model parameters to improve after every layer using descending constraints:

$$\begin{aligned} \min_{\boldsymbol{\theta} \in \mathbb{R}^p} \quad & \mathbb{E}\big[g(\boldsymbol{\Phi}(\mathbf{W}_0, \mathcal{B}; \theta))\big] \\ \text{s.t.} \quad & \mathbb{E}\Big[\|\nabla g(\mathbf{W}_l)\| \; - (1-\epsilon)\, \|\nabla g(\mathbf{W}_{l-1})\|\Big] \leq 0, \;\; \text{for all} \;\; l = 1, \ldots, L, \\ & \mathbf{W}_l = \phi_l(\mathbf{W}_{l-1}, \mathcal{B}_l; \boldsymbol{\theta}_l), \quad \text{for all} \;\; l = 1, \ldots, L. \end{aligned}$$

Hadou-NaderiAlizadeh-Ribeiro, *Stochastic Unrolled Federated Learning*, arxiv.org/abs/2305.15371

Unrolling Distributed Gradient Descent via GNNs

- Distributed gradient descent (DGD) is a distributed iterative algorithm with the update rule:

$$\mathbf{w}_i(l) = \sum_{j \in \mathcal{N}_i} s_{ij} \mathbf{w}_j(l-1) - \beta \nabla g_i(\mathbf{w}_i(l-1)), \quad i = 1, \ldots, N.$$

- DGD relies on communication among agents, and local updates of the model using local data.
- We replace the first term with a GNN layer and the second term with a local FCNN:

$$\mathbf{W}_l = \sum_{k=0}^{K-1} h_{kl} \mathbf{S}^k \mathbf{W}_{l-1} - \sigma\left([\mathbf{W}_{l-1}, \mathcal{B}_l] \mathbf{M}_l + \mathbf{b}_l\right)$$

Hadou-NaderiAlizadeh-Ribeiro, *Stochastic Unrolled Federated Learning*, arxiv.org/abs/2305.15371

Empirical Evaluation of SURF + DGD with GNNs

- Accuracy levels evaluated over randomly selected 3-class subsets of CIFAR-10 with 100 agents.

Training Algorithm	Accuracy	#Layers/Iterations
Centralized	25.81 ± 13.92	10
FedAvg	15.53 ± 12.29	10
SURF + DGD + GNN	$\mathbf{90.83} \pm \mathbf{04.35}$	10
Centralized	$\mathbf{92.71} \pm \mathbf{03.26}$	300
FedAvg	90.35 ± 03.69	300

Transferability of SURF + DGD with GNNs

- The trained meta-GNN transfers to different numbers of agents, dataset sizes, and topologies.