Lecture 8 Script

8.1 First Midterm

Slide 1: First Midterm - Title Page

(1) We are halfway through the course and it is time for us to have a midterm. But I am grateful that you are giving me the privilege of listening to what I have to say. So I will answer the questions myself.

Slide 2: Quiz

(1) There are 5 questions in our quiz: What? Why? How? Does this work? And Should this work?

(2) The answer to the question of what.

(3) Is that this is a course about machine learning on graphs.

(4) The answer to the question of why.

(5) Is that graphs are generic models of signal structure and structure is the key to scalability in machine learning.

(6) Graphs are also models of distributed physical infrastructure. Such as wireless networks, power grids or collaborative systems. In which the graph is not only helpful but inherent. .

(7) The answer to the question of how.

(8) Is that we use graph neural networks.

(9) Does this work? Do GNNs work?

(10) Yes. We have worked out the solution to a couple of practical problems. We are going to study a couple more. And we could study another dozen

(11) And should this work? Should we expect GNNs to solve the problems they solve?

(12) Yes. We have offered theoretical explanations in the form of permutation equivariance and stability to deformations of the support. All of these answers are correct. But we can do better. Let us work together on a more detailed answer.

Slide 3: How? ⇒ Signals Supported on Graphs

(1) Regarding the how question, the first component of our answer is to introduce the notion of a graph signal. Which we have used to great success as a model of several systems of practical importance.

(2) In a graph signal we use a graph as a generic descriptor of signal structure.

(3) With signal values

(4) Associated to nodes

(5) And edges expressing similarity between components of the signal

(6) These are therefore the three components of a graph signal model. Nodes that support signal components and edges that describe their expected proximity.

(7) If nodes are customers, signal values are ratings, and edges are cosine similarities. We have a description of a recommendation system

(8) If nodes are transceivers, signals are QoS requirements and edges are wireless channel strengths. We have a model of a wireless communication network.

(9) If nodes are drones, signals are velocities, and edges are sensing and communication ranges. We have a model of am autonomous system.

Slide 4: How? ⇒ Graph Convolutional Filters

(1) The second component of our answer to the question of how is to design graph convolutional filters. Which we define as a polynomial on the shift operator S that utilizes coefficients h_k.

(2) The convolutional filter depends on the filter coefficients h_k and on the shift operator S. The effect of these two parameters is somewhat separate. A fact we have exploit for transference and that is also crucial for analysis.

Slide 4: How? ⇒ Graph Convolutions

(1) A related notion is that of a graph convolution. Which is the result of applying a graph filter to a graph signal. This is just a matrix-vector multiplication.

(2) But it is also a weighted linear combination of elements of the diffusion sequence. We never compute graph filters. We compute graph convolutions because they are computationally more efficient. A graph convolution recursively applies three operations: Shift. Scale. And sum.

(3) We begin with the signal x. Which is re-baptized here as the zeroth element of the diffusion sequence.

(4) We scale it by h_0.

(5) And sum towards the output.

(6) We then multiply the signal x by the shift operator. We call this a shift. The result of which is the first element of the diffusion sequence.

(7) We scale it by h_1

(8) And sum it towards the output

(9) We now apply the shift a second time. This produces element 2 of the diffusion sequence

(10) We scale it by h_2

(11) And sum towards the output

(12) We shift a third time to produce element 3 of the diffusion sequence.

(13) We scale if by h_3

(14) And sum it towards the output.

(15) Since this is a filter of order 3, the summation of these four scaled and shifted elements of the diffusion sequence is the graph convolution of the filter h with the signal x instantiated on the graph S.

Slide 5: How? ⇒ Graph Neural Networks (GNNs)

(1) The third piece of the answer to the question of how is where things start to get interesting. This is where we introduce a graph neural network. A GNN for short.

(2) A GNN with L layers is defined as the recursive composition of L perceptrons. The figure is an example with 3 Layers. Each perceptron is itself the composition:

(3) Of a graph filter.

(4) With a pointwise nonlinearity.

(5) The Layer L perceptron takes as input the signal x_ell-minus-one. Which is the output of Layer ell minus one

(6) It computes the graph convolution with the Layer ell filter. Which has coefficient s h_ell-k.

(7) The filter's output is passed to the pointwise nonlinearity sigma.

(8) To produce the Layer ell output x_ell.

(9) The output of the GNN is the output of the layer capital L. The last layer of the GNN.

(10) Which is a function of the filter tensor H, that groups all filter coefficients h_ell-k. And the shift operator S. As is the case of graph filters, the effect of these two parameters is somewhat separate. A fact we exploit for transference and that is also crucial for analysis.

Slide 6: How? ⇒ Graph Neural Networks with Multiple Features

(1) These GNNs with single features are interesting conceptual objects but from a practical perspective they are pretty useless. The ones we use in practice are GNNs with multiple features.

(2) They are the same except that the filters at each layer are MIMO graph filters that take a collection of features as an input and output another collection of features. But other than that it is the same architecture.

(3) We stack layers.

(4) Each of which is the composition of a graph filter.

(5) With a pointwise nonlinearity.

(6) The only modification is that the graph filters are MIMO graph filters.

(7) The rest of the processing architecture is the same.

(8) In particular, the output of the GNN is still the output of the capital ell-th layer.

(9) Which is a function of the filter tensor and the shift operator. We are not saying much about GNNs with multiple features in our lectures. This is because the conceptual differences between single feature and multiple feature GNNs are few. But do not interpret the paucity of mention as a statement of importance. GNNs with multiple features are the ones that are relevant in practice. In fact, the opposite is true of the labs. We barely mention GNNs with single features in them. We are interested in GNNs with single features only because they help us in understanding GNNs with multiple features.

Slide 7: How? ⇒ Some Observations About Graph Neural Networks

(1) Let's close the how questions with some observations. Graph neural networks are:

(2) First and foremost, minor variations of graph filters.

(3) We just add nonlinearities and compositions. If you want to understand a GNN, you need to understand graph filters.

(4) GNNs are transferable across different graphs. A trained GNN is a trained tensor. It can be instantiated in any graph we please. Of course, we want to transfer across graphs that we expect to be similar in some way or another.

(5) GNNs are generalizations of CNNs, Because we recover a CNN by particularizing to the line graph.

(6) And they are also particularizations of fully connected neural networks. As all neural networks are.

Slide 8: Does This Work and Should This Work?

(1) This completes the answer to the question of how and leads to the more significant questions in the quiz. Does this work? And should this work?

(2) Does this work. Do GNNs work?

(3) The answer is yes. We have worked out their application to recommendation systems and are working out their application to the allocation of resources in wireless communications. We will review the use of GNNs in recommendation systems in this midterm.

(4) And should this work? Should we expect GNNs to solve the problems they solve? Yes. We have offered two theoretical explanations:

(5) We have shown that graph filters and GNNs are permutation equivariant. This property allows them to leverage signal symmetries. It explains why graph filters outperform linear regression and why GNNs outperform fully connected neural networks. This is legit science by the way. Our theory makes a prediction. This prediction is ratified experimentally.

(6) We have also shown that GNNs have a better stability versus discriminability tradeoff relative to graph filters. This implies that GNNs are better than graph filters at leveraging quasi-symmetries. And explains why GNNs outperform graph filters. This is also legit science. Our theory makes a prediction. This prediction is ratified experimentally. We will review permutation equivariance and stability properties of graph filters and GNNs in this midterm.

8.2 Learning Ratings in Recommendation Systems

Slide 9: Learning Ratings in Recommendation Systems - Title Page

(1) To illustrate the use of graph filters and GNNs, we formulate recommendation systems as empirical risk minimization problems.

Slide 10: Recommendation Systems

(1) The goal of a recommendation system is to predict the rating that a user would give to a certain item

(2) To solve this problem we collect rating histories. They are ratings that users have given to items in the past.

(3) The idea is to exploit the similarities between users and items to predict the ratings of unobserved user-item pairs.

(4) There are plenty of examples of recommendation systems. In an online store, items are product and users are shoppers

(5) In a a movie repository, the movies are the items and the watchers are the users.

Slide 11: Ratings and Sampled Ratings

(1) The underlying assumption of a recommendation system is that for all items i and users u, there exist ratings y_u-i

(2) For each user u we can can define a rating vector y_u grouping all the ratings of this user.

(3) However, many of these ratings are unobserved. We actually observe a subset of ratings x_u-i. This is typically a small subset of ratings. The majority of products are never rated by a given user.

(4) The available ratings of user u are grouped in the rating vector x_u

(5) Where we convene the ratings x_u-i equals 0 for all items i that unrated by user u.

Slide 12 Product Ratings as Graph Signals

(1) In order to predict ratings we build a product similarity graph with edge weights w_i-j representing the likelihood of observing similar scores.

(2) With this graph available the ratings vector y_u of user u is a graph signal supported on the product similarity graph.

(3) And the observed ratings x_u of user u are a subsampling of this graph signal.

(4) Our goal is to learn to reconstruct the rating graph signal y_u from the observed ratings x_u.

(5) This procedure requires that we build a graph of product similarities. We will do that here using the rating histories themselves.

(6) But it is also possible to build the graph using expert knowledge.

Slide 13: Product Similarity Graph

(1) To construct the product similarity graph, we begin by considering the pair of products i and j. We then consider the set of users U_i-j that have rated both items.

(2) We then compute the mean ratings restricted to users that rated products i and j.

(3) The mean for item i is the average of user ratings of product i. But the average is restricted to the set U_i-j. This is why we call the average mu_i-j.

(4) Likewise, the mean for item j is the average of user ratings of products j. But the average is restricted to the set U_i-j and this is why we denote it as mu_j-i.

(5) Once we compute both means mu_i-j and mu_j-i, we compute the correlation coefficient between the ratings of products I and j. The correlation coefficient is restricted to the set of users that have rated both prodcuts.

(6) To obtain the edge weights of the product similarity graph, we compute the normalized correlation by dividing the correlation coefficient by the square root of the multiplication of the auto correlations of each item i and j.

Slide 14: Loss for Measuring Rating Prediction Quality

(1) To write a learning formulation we also need to define a loss. Given observed ratings x u the AI produces estimates phi of x u. These estimates are ones we want to compare with the ratings y_u. We do that with the square norm loss.

(2) This is not wrong, but in reality we want to predict the ratings of specific items i. Thus, it is more convenient to define a loss in which we compare the prediction and the true

rating for a specific item i. We do that by multiplying both, the AI prediction and the true rating,

(3) With the canonical basis vector e_i. This vector is all zero except for its i-th entry and it therefore selects the ratings of item i.

Slide 15: Training Set

(1) Up until now we have constructed an abstract formulation. For a concrete formulation we use the set of existing ratings to construct a training set. Begin considering a given item i and let U_i be the set of users that have rated i.

(2) We construct training pairs x comma y in which

(3) The vector y extracts the rating of item i from the vector of ratings x_u of user u.

(4) And the vector x is constructed from x_u by removing this rating. This allows us to train for a situation in which we predict the rating that user u would give to item i, utilizing all of the other available ratings of user u.

(5) We repeat this process for all users u that have rated this product.

(6) And repeat the process for all items to construct the training set T.

Slide 16: Learning Rating Predictions

(1) With this training set we consider an AI parametrized by calligraphic H and attempt to find a solution of an ERM problem in which we use the loss we introduced a minute ago and the training set we just described.

(2) We will investigate two bad ideas: The use of linear regression and the use of fully connected neural networks as learning parameterizations.

(3) We will then investigate two good ideas: The use of graph filters and graph neural networks as learning parameterizations

8.3 Learning Ratings with Graph Filters and GNNs

Slide 17: Learning Ratings with Graph Filters and GNNs - Title Page

(1) We use graph filters and graph neural networks to learn ratings in recommendation systems.

(2) We contrast their performance with the use of linear regression and fully connected neural networks.

Slide 18: Movie Ratings Dataset

(1) We use the MovieLens 100K dataset as benchmark. This dataset contains 10^5 ratings that 943 users assigned to 1,682 movies

(2) The ratings for each movie are between 0 and 5. From no starts. To five stars. Which your TA Juan Cerviño thinks you should find familiar.

(3) We will train and test several parameterizations.

Slide 19: Empirical Risk Minimization

(1) We pose the recommendation systems as a learning problem is which we use the mean squared error loss restricted to the prediction of the rating of item i. The training set contains pairs x, y in which we are given ratings of different users and we attempt to predict the ratings of specific items. We want an AI that produces good estimates averaged over all items and over all users that are part of the training set.

(2) We consider two parameterizations that ignore the data structure.

(3) Linear Regression and Fully connected neural networks.

(4) As well as two parameterizations that leverage data structure.

(5) Graph Filters and Graph Neural Networks. The goal of running these four experiments and to compare their performances.

Slide 20: Linear Regression and Graph Filters

(1) Begin by considering linear regression. It brings down the MSE to about 2. This is quite bad for ratings that vary from 0 to 5.

(2) The graph filter reduces the training MSE to about 1. This is not too good. But the challenge here is that humans are not that predictable. An MSE of 1 is considered good in a recommendation system.

(3) If we consider the test set we see that the performance of linear regression is even worse.

(4) Whereas the test MSE for the graph filter is about the same as the training MSE. The graph filter generalizes well from the training set to the test set.

(5) The most important observation to make here is that the graph filter outperforms linear regression. Both architectures are linear. But the graph filters leverages the underlying permutation symmetry of the rating prediction problem.

Slide 21: Fully Connected NNs and Graph NNs

(1) When we look at the comparison of fully connected neural networks and graph neural networks the fully connected NN reduces the training MSE to about 0.8. This looks like a great accomplishment.

(2) Particularly when we compare it with the training MSE of the Graph Neural Network . Which is reduced to about 1.

(3) But the fully connected NN does not do well in the test set. The MSE reduction is illusory. The fully connected NN does not generalize.

(4) The GNN on the other hand has a test MSE that is about the same as the training MSE. The GNN generalizes to the test set.

(5) The important observation to make is that the Graph NN outperforms the fully connected NN. The GNN leverages the underlying permutation symmetry of the rating prediction problem.

Slide 22: Graph Filters and Graph Neural Networks

(1) Or final comparison is between the graph filter and the GNN. The graph filter.

(2) And the GNN.

(3) Both do well in the training and test set. They generalize well.

(4) The GNN does a little better. Not by much. But an extra 10% performance is not irrelevant.

(5) The important point to make is that the GNN outperforms the graph filter. This is because the GNN offers a better stability versus discriminability tradeoff.

Slide 23: Transferability

(1) Our final experiment is to observe that a GNN can be trained on a graph with a small number of nodes.

(2) And transferred to a graph with a larger number of nodes. Without having to retrain the filter tensor.

(3) In the case of this recommendation system we grow the graph from 200 to 800 users.

(4) And observe that there is no degradation in the MSE. In fact, the MSE is further reduced. This is a phenomenon fro which we still don't have an expla

Slide 24: The Engineer and The Scientist

(1) The Engineer is satisfied.

(2) They proposed the use of GNNs in recommendation system and they demonstrated that it works.

(3) The scientist is curious

(4) The method works but they don't know why.

(5) This is the motivation for out analyses. Which we will proceed to review in the next part of this midterm.

8.4 Permutation Equivariance

Slide 25: Permutation Equivariance - Title Page

(1) Graph filters and GNNs are equivariant to permutations. A property that lets them leverage signal symmetries

(2) A fact that **predicts** that GNNs and graph filters should outperform linear regression and fully connected neural networks. Which we have verified experimentally.

Slide 26: Permutation Equivariance of Graph Filters

(1) It is ready to show that graph filters are permutation equivariance.

(2) That is, if we consider a permutation of the input graph signal.

(3) Which entails a permutation of the signal

(4) And the shift operator

(5) The outputs of the graph filter are related by the same permutation.

(6) (Empty)

(7) The straightforward implication of this theorem is that relabeling the input to a graph filter results in a consistent relabeling of its output.

(8) A not so straightforward conclusion of this theorem is that graph filters leverage permutation symmetries of graphs and graph signals

Slide 27: Permutation Equivariance of Graph Neural Networks

(1) It is just as ready to prove that graph neural networks are permutation equivariant. In fact, the property is inherited from graph filters. The addition of a pointwise nonlinearity has no effect on equivariance. Because it is, well, pointwise.

(2) In any event, if we consider permutations of the input graph signal.

(3) The outputs of the GNN instantiated in the graph S-hat with input signal x-hat.

(4) Is equal to the the corresponding permutation.

(5) Applied to the output of the GNN instantiated in the graph S with input signal x.

(6) (Empty)

(7) As in the case of graph filters the straightforward implication of this theorem is that relabeling the input to a GNN results in a consistent relabeling of its output.

(8) And the not so straightforward conclusion of the theorem is that graph neural networks leverage permutation symmetries of graphs and graph signals.

Slide 28: Signal Processing with graph filters and GNNs is Independent of Labeling

(1) The straightforward conclusion of the equivariance theorems is, as we have said, that graph filters and GNNs perform label-independent processing.

(2) As a reminder, this is because a permutation of the input and the shift is tantamount to a relabeling of the graph. It's the same signal on the same graph. We just changed the names os the nodes. We relabeled the nodes.

(3) The permutation equivariance of GNNs entails that the same is true of the output.

(4) Namely, that if we consider the result of processing x on the graph S with a GNN with filter tensor H

(5) And the result of processing the relabeled signal x on the relabeled graph S with a GNN with the same filter tensor H.

(6) The output of the latter is a relabeling of the output of the former. This is something the theorem claims is true. But it is also something that we want to happen. We want the processing to be independent of labels. GNNs attain that. And the same is true of graph filters.

Slide 29: Equivariance to Permutations and Signal Symmetries

(1) The not so straightforward consequence of the equivariance theorems is that graph filters and GNNs can exploit permutation symmetries of graphs and graph signals.

(2) By symmetries, we mean situations in which a graph is invariant under the action of a permutation. This means it is possible to rewrite S as P-transpose-S-P for some

permutation P. This is possible for the graph we show on the left. Indeed, consider a specular symmetry around a 45-degree line. So that node 9 moves onto node 12, node 3 moves onto node 6 and so on. This is a permutation but the permutation does not change the graph.

(3) The equivariance theorem applied in cases where these symmetries exist, says that:

(4) A permutation of the shift and the signal applied at the input of the GNN.

(5) Is the same as a permutation of the signal (and the shift) applied at the output of the GNN.

(6) But we also know that the shift permutation is moot because the shift operator is invariant to this permutation. P-transpose-S-P is just S.

(7) We have therefore shown that when a graph is symmetric, the GNN output associated with a permutation of the input x is equal to a permutation of the output of the GNN. But without permuting the shift. Which we don't have to permute. Because it is symmetric.

(8) We can therefore claim that a GNN learns to process the permuted signal P-transpose-x supported on S

(9) From learning the processing of the non-permuted signal x. This is profound. It means we can learn to process P-transpose-x without ever seeing examples of this form. We extrapolate from learning how to process x. If this is a recommendation system, we are learning to predict ratings for user number 6. Without ever observing a single rating for this customer. We are just inferring from what we know about customer number 3. You can think of this effect as a multiplication of the number of entries in the dataset. Whenever you observe a signal for customer 3 you are also seeing a signal for customer 6. And whenever you observe a signal for customer 6, you are also observing a signal for customer 3.

Slide 30: It is Quasi-Symmetry we want to exploit. Not Symmetry

(1) This explanation of the value of symmetry has a flaw: Symmetric graphs are very rare. They just don't appear in reality. What does appear in reality are graphs that are close to symmetric.

(2) As the one we show in this figure.

(3) This graph can't be permuted into itself as the one we have just shown.

(4) But the permutation results in a graph that is a deformed version of itself. The graph is not perfectly symmetric. But it is close to symmetric. It is quasi-symmetric.

(5) Thus, it is quasi-symmetry that we want to exploit. Not symmetry. We want to have stability to deformations of a shift operator that are close to permutations.

(6) Quasi-symmetry is what sets GNNs and graph filters apart. Both operators are permutation equivariant. They are equally good at leveraging symmetries. But the GNN exhibits better stability properties with respect to deformations. This means they can exploit quasi-symmetries better. And it explains why they work better than graph filters.

8.5 Stability of Graph Filters to Graph Perturbations

Slide 31: Stability of Graph Filters to Graph Perturbations - Title Page

(9) Our most interesting theoretical conclusion has been to prove the stability of integral Lipschitz filters to relative deformations of the graph.

Slide 32: Frequency Response of a Graph Filter

(1) Our analyses are undertaken in the spectral domain. They are beautiful because once we move into the spectral domain, they become very simple.

(2) Indeed, graph filters are operators defined as functions of given graph shift operators. They depend on a set of given coefficients h_k and the graph shift operator S. It is not easy to understand what the effect of a graph filter is on a given graph signal.

(3) But in the frequency domain the filter is completely characterized by its frequency response. Which is just a polynomial on a scalar variable lambda. It becomes independent of the graph. And it becomes very easy to understand the effect that a filter has on a given graph signal.

Slide 33: The Effect of the Graph

(1) The graph does not disappear in the spectral domain, but we have seen that its effect is pretty simple.

(2) The graph has a specific set of eigenvalues lambda_i. The frequency response is instantiated at them. When the filter with coefficients h_k is run on this graph, the effect it has on a signal depends on the values of the frequency response evaluated at the lambda_i eigenvalues.

(3) When given a different graph with a different set of eigenvalues lambda-hat_i, the frequency response is evaluated at the new set. The effect the filter has on a signal is different because the frequency response is instantiated on a different set.

(4) This is the fundamental insight in the analysis of graph perturbations. To analyze how a filter changes when we change the graph, we just need to study how eigenvalues of the shift operator change. We need to compare the two different sets of eigenvalues that appear in this figure

Slide 34: Relative Perturbations of a Shift Operator

(1) In our study of perturbations we saw that relative perturbations of graph filters are the ones that are most meaningful. We add a multiplicative error term that is symmetric and modulo permutation.

(2) Conceptually, we saw that we can learn all that there is to be learnt from the study of dilations. Where the error matrix is epsilon times identity. Which implies the shift is scaled, or dilated, by 1 plus epsilon.

(3) When we consider this specific perturbation, the eigenvalues of the shift operator are themselves dilated. This results in the frequency response being evaluated at the dilated eigenvalues.

(4) Thus, if we are given a graph with eigenvalues lambda_i.

(5) The dilated shift operator has eigenvalues lambda-hat_i. Each of which is an eigenvalue lambda_i dilated by 1 plus epsilon.

Slide 35: Higher Frequencies are More Difficult to Process

(1) This observation leads to a key insight. Which is that high eigenvalues move more.

(2) Or conversely, that low eigenvalues move little. The low eigenvalue lambda_i here barely moves when we dilate the graph

(3) The medium eigenvalue lambda_m moves a little more.

(4) And the high eigenvalue lambda_h moves quite a lot.

(5) Thus, signals with high frequency components are more difficult to process with a graph filter. This is inherent. There is nothing we can do to mitigate this problem. Except, of course, to use something other than a graph filter.

(6) The source of the complication is that even small perturbations of the shift operator yield large differences in the values of the filter's frequency response that are instantiated.

(7) We think we are are instantiating h of lambda_i.

(8) But in reality, we are instantiating h of lambda_i-hat. We are instantiating the frequency response at a dilated eigenvalue. And the difference between these two eigenvalues and their corresponding responses can be quite a lot for a large frequency. In practice, this means that graph filters can be very sensitive to deformations of the graph. And that they have very little room to exploit quasi-symmetries. The graph has to be very close to a regular symmetric graph for a filter to leverage symmetries.

Slide 36: Stability Requires Integral Lipschitz Filters

(1) The solution to this conundrum is to use use integral Lipschitz filters. In which we limit the derivative of the filter's response to be inversely proportional to lambda. These filters attain stable graph signal processing.

(2) This is true because the condition forces the filter to be flat at high frequencies. Therefore, if we consider the eigenvalues of a graph and its dilated version as we show in this figure.

(3) Either the eigenvalue does not change because we are considering low frequencies.

(4) Or the frequency response does not change when we are considering high frequencies. The eigenvalue can move quite a lot. But the frequency response doesn't change because doing so would violate the integral Lipschitz condition. This creates discriminability challenges as we are going to review in the next section.

Slide 37: Pictures are Worth a Thousand Theorems

(1) Before we look at that. Let me emphasize that pictures are worth a thousand theorems, but only if you have a theorem to back them up. But we do. We have proven that integral Lipschitz filters are stable. In this theorem:

(2) We consider relative perturbations of the shift operator

(3) And restrict the filters to be integral Lipschitz.

(4) If these conditions are satisfied the filters are proven stable modulo permutation with respect to the norm of the perturbation.

(5) The pictures we have drawn, are a proof of this theorem for the case of dilations. The generic proof, as we have seen, relies on analogous arguments. The important steps on the proof rely on the observation that high eigenvalues move more than low eigenvalues and that the integral Lipschitz conditions cancels this extra eigenvalue variability with a reduction on the filter's variability.

8.6 Stability and Discriminability are Incompatible in Graph Filters

Slide 38: Stability and Discriminability are Incompatible in Graph Filters

(1) The movement of eigenvectors proportional to their eigenvalues implies that stability and discriminability are incompatible in graph filters. At best, we can claim a limited tradeoff.

(2) Or to be perfectly precise, The stability vs discriminability tradeoff depends on the frequency components of the graph signal.

Slide 39: Discriminative Filter at Low Frequencies

(1) Indeed, at low frequencies, a filter that is sharp and, therefore, highly discriminative, is also highly stable.

(2) The ideal response h of lambda_ell.

(3) And the perturbed response h of lambda_ell-hat, evaluated at the dilated eigenvalue.

(4) Are very close to each other. If the filter captures a spectral feature at this frequency in the ideal graph S, the filter also captures the feature in the perturbed graph S-hat.

Slide 40: Discriminative Filter at Medium Frequencies

(1) At medium frequencies, a filter that is sharp and highly discriminative, is not so stable but is not very unstable either. Let's say that it is somewhat stable.

(2) The ideal response h of lambda_m.

(3) And the perturbed response h of lambda_m-hat, evaluated at the dilated eigenvalue.

(4) Are somewhat close to each other. If the filter captures a spectral feature at this frequency in the ideal graph S, the filter also captures some energy of the feature in the perturbed graph S-hat. Not as well as it would capture the feature in graph S. But it doesn't miss the feature entirely.

Slide 41: Discriminative Filter at High Frequencies

(1) At high frequencies, however, a highly discriminative sharp, is unstable. To the point that it becomes useless.

(2) This is because the ideal response h of lambda_h.

(3) And the perturbed response h of lambda_h-hat, evaluated at the dilated eigenvalue.

(4) Are very different to each other. The filter may capture a spectral feature at this frequency in the ideal graph S. But it misses it entirely in the perturbed graph S-hat. A small shake of the graph makes the filter useless. The filter can leverage signal symmetries only when they are almost perfect symmetries.

Slide 42: Stability vs Discriminability Non-Tradeoff of Graph Filters

(1) This is therefore a fact: It is impossible to discriminate high frequency components with a stable filter.

(2) We can have a filter that is discriminative

(3) Or a filter that is stable

(4) But not one that is both. This fact is of sufficient importance to deserve a second look.

Slide 43: Separate High Frequency Signals with Lipschitz Filter - No Deformation

(1) Suppose that we want to discriminate two features, which, for the sake of argument, we assume are perfectly aligned with the eigenvectors v_i and v_j of the shift operator. Eigenvector v_i is associated with eigenvalue lambda_i and eigenvector v_j with eigenvalue lambda_j. The GFTs of these signals are one for components i or j and 0 for all other components. They are spikes placed at lambda_i or lambda_j, if we can stretch the language a little. We have learned the filters we show below that discriminate signal x_i from signal x_j.

(2) To explain what we mean by “discriminate” let’s look at the input-output relationship for the filter that learns to identify x_i. The signal aligned with eigenvector v_i. Which in the GFT domain is a spike at lambda_i.

(3) When we hit this filter with input signal x_i

(4) The output is a scaled version of this signal. With the scaling given by the frequency response evaluated at lambda_i.

(5) When we hit the filter with input signal x_j

(6) The output is a scaled version of this signal but the scaling coefficient is the frequency response evaluated at lambda_j. Which is roughly equal to zero. Therefore, the output of this filter has high energy when we hit it with v_i and low energy when we hit it with v_j. Thus, the filter isolates v_i from v_j. It discriminates v_i from v_j.

(7) The opposite is true of the filter that learned to identify x_j. The signal aligned with eigenvector v_j. The spike at lambda_j.

(8) When we hit this filter with v_j, the signal and the filter match.

(9) The output is a scaling of v_j with scaling coefficient given by the frequency response evaluated at lambda_j. Since this is close to one we have significant energy at the output.

(10) When we hit the filter with v_i

(11) The scaling coefficient is the frequency response evaluated at lambda_i. This is a value close to zero. Thus, there is little energy at the output. The filter isolates v_j from v_i. It discriminates v_j from v_i. Put together, these two filters discriminate signals v_i and v_j.

Slide 44: Separate High Frequency Signals with Lipschitz Filter - After Deformation

(1) But this discrimination is lost when the shift operator is perturbed. For instance, consider the filter that isolates v_i. But we run this filter on the dilated graph.

(2) When we hit the filter with inout v_i, the output is a scaling, as it was before.

(3) But the scaling factor is the response of the filter evaluated at lambda_i-hat. If this is a high eigenvalue and the filter is not integral Lipschitz, this scaling can be very different. In the example shown this is the case. The scaling factor is a small number and we therefore have little energy at the output.

(4) If we hit the filter with v_j

(5) The output is still close to zero. But this is little consolation. We are failing at capturing the signal v_i. Which is matters here.

(6) For the filter that isolates v_j the situation is analogous.

(7) When we hit if with v_j, the output is a scaling. As in all other filters.

(8) But, as is the case of the filter that isolates v_i, the scaling coefficient is the response evaluated at the dilated eigenvalue. This is a small constant because we are considering a high eigenvalue and a filter that is not integral Lipschitz. There is little energy at the output.

(9) When we hit the filter with v_i

(10) We now have some energy making it to the output. This filter is failing at the discrimination of the signal v_j from the signal v_i. There is not much difference between the energies at the output when we hit the filter with v_j relative to when we hit the filter with v_i.

Slide 45: Stability vs Discriminability Non-Tradeoff of Graph Filters

(1) An important fact to emphasize is that this is **not** what we would had expected a priori.

(2) It is reasonable to expect discriminability to be in **conflict** with stability.

(3) But this is not what happens.

(4) They are plainly incompatible.

(5) Having made this point let's clarify that there is **some** tradeoff. Increasing the Lipschitz constant C of the integral Lipschitz filter does improve discriminability. At the cost of reducing stability.

(6) But the effect is marginal at high frequencies.

(7) It is also true that given a stable filter there is always a graph with features that can't be discriminated by the filter. We just need to move up in the frequency domain

(8) To be perfectly precise, what we have shown is that the stability-discriminability tradeoff is different for different frequencies. We can be more stable, arbitrarily more stable in fact, at low frequencies for a given discriminability. But the data is what it is and graph filters can't move high frequencies into lower parts of the spectrum. But pointwise nonlinearities can! This is what graph neural networks do as we will explain in the next section.

8.7 The Stability vs Discriminability Tradeoff of GNNs

Slide 46: The Stability vs Discriminability Tradeoff of GNNs - Title Page

(1) We will review here that the reason why GNNs outperform graph filters is that pointwise nonlinearities move high frequencie components into lower parts of the spectrum.

(2) Where we can use stable filters to discriminate them at deeper layers of the GNN stack.

Slide 47: GNNs Inherit the Stability Properties of Graph Filters

(1) Since the nonlinearity is pointwise, it is impervious to changes in the shift operator. Therefore, the same stability theorem that holds for graph filters, also holds for GNNs.

(2) If we consider relative perturbations modulo permutation

(3) And individual Layers that are made up of integral Lipschitz filters

(4) We can establish the stability of GNNs relative to the norm of the perturbation.

(5) The stability constant is essentially the same.

(6) Except that it is scaled by the number of layers because the error propagates accumulates.

(7) We say the stability of the GNN is inherited from the stability of the graph filter. Because it is due to the stability of the filter that the GNN is stable.

Slide 48: Stability vs Discriminability Tradeoff of GNNs

(1) A fact we can state in contrast to graph filters is that it is possible to discriminate high frequency components with a stable GNN.

(2) This is because the GNN inherits the stability properties of graph filters. This is good. Having stable learning is useful. Necessary, even.

(3) But GNNs do **not** have to inherit the bad discriminability properties of graph filters.

(4) They can rely on pointwise nonlinearities to demodulate high frequencies into low frequencies. Where they can be discriminated with a stable filter at deeper layers.

Slide 49: Isolate High Frequency Signals - No Deformation

(1) To explain this latter statement, consider the problem of discriminating signals x_i and x_j that are aligned with the eigenvectors v_i and v_j of the shift operator. In the GFT domain these signals are represented by spikes at eigenvalues lambda_i and lambda_j, which are the corresponding associated eigenvalues.

(2) Since we are assuming that these eigenvalues are large and that we want to have a stable filter, we can't have a filter that separates them perfectly. But we can have a filter that isolates them from the rest.

(3) If we hit this filter with input signal x_i.

(4) The output is a scaled version of v_i. The scaling coefficient is the filter's response evaluated at lambda_i, which is a value close to one in the example shown.

(5) If we hit the filter with input signal x_j.

(6) The output is a scaled version of v_j. The scaling coefficient is the filter's response evaluated at lambda_j, which is also a value close to one in the example shown.

(7) This filter isolates signals v_i and v_j from the rest but it doesn't discriminate between them.

Slide 50: Isolate High Frequency Signals - After Deformation

(1) It is however, a filter that is stable to deformations.

(2) Indeed if we consider a scaling of the spectrum, the filter's frequency response is instantiated at dilated eigenvalues. But since the filter is integral Lipschitz, the dilation of the eigenvalues doesn't change the value of the filter's response.

(3) If we hit the filter with signal v_i

(4) The output is still close to v_i. Because the filter's response evaluated at the dilated eigenvalue lambda_i-hat is still close to one. The eigenvalue has moved substantially. But the integral Lipschitz condition has kept the filter's response unchanged.

(5) Likewise, when we hit the filter with signal v_j

(6) The output is still close to v_j. Because the filter's response evaluated at the dilated eigenvalues is still close to one. Because even though the eigenvalue moves, the filter does not. Because it is integral Lipschitz.

Slide 51: Pointwise Nonlinearities are Frequency Mixers

(1) The filter is stable but it does not discriminate between v_i and v_j. Here is where the nonlinearity and deeper layers come into the picture. When we apply the nonlinearity to the output of the filter, we obtain spectral profiles for sigma of v_i and sigma of v_j that spread energy across all frequencies.

(2) More to the point, suppose that the input to Layer 1 is the eigenvector v_i. As we explain in the previous slide, the output of the filter is, roughly, v_i itself. It gets scaled by a frequency response close to 1. We then apply the nonlinearity sigma to compute the output of Layer 1.

(3) The projection of sigma of v_i on the spectral domain of the shift operator yields the GFT V-Hermitian times sigma of v_i.

(4) This is represented in this figure for the shift S and for the dilated shift S-hat. The very important point here is that since the filter in Layer 1 is stable to dilations, the output of

Layer 1 is more or less the same. Whether we run the filter on the graph S or we run the filter on the dilated graph S-hat.

(5) Analogously, when the input to Layer 1 is the eigenvector v_j, the output of the filter is, roughly, v_j itself. It gets scaled by a frequency response that is also close to 1. We then apply the nonlinearity sigma to compute the output of Layer 1. This is therefore sigma of v_j.

(6) The projection of sigma of v_j on the spectral domain of the shift operator yields the GFT V-Hermitian times sigma of v_j.

(7) This is represented in this figure for the shift S and for the dilated shift S-hat. As is the case when the input is v_i, the very important point here is that since the filter in Layer 1 is stable to dilations, the output of Layer 1 is more or less the same. Whether we run the filter on the graph S or we run the filter on the dilated graph S-hat.

(8) Thus, energy at the output of Layer 1 is spread across several frequencies in a manner that is stable to deformations. Layer 1 outputs are the same if the filters are instantiated on S or the dilated S-hat. And their spectral representations have components at all frequencies.

(9) In the case of both signals, some energy is where it used to be.

(10) Some energy is at other high frequencies

(11) Some energy moves to medium frequencies

(12) And some energy moves to low frequencies.

(13) The latter of which can be discriminated with a stable filter in Layer 2. We should remark that these two are actual GFTs of nonlinearities applied to the two largest eigenvectors of a random graph with 8 nodes. Their frequency components have a certain garbage aspect to them. But it is stable garbage. It can be discriminated with stable filters. And the process can be repeated. We can isolate again high frequencies with a stable filter and process them with nonlinearities so that we can discriminate them better at Layer 3.

Slide 52: Stability vs Discriminability Tradeoff of GNNs

(1) Fact Indeed: It is possible to discriminate high frequency components with a stable GNN

(2) A GNN can be discriminative.

(3) And it can be stable at the same time.

(4) Stability and discriminability are compatible.

(5) **Individual** layers cannot discriminate high frequency components. They must use integral Lipschitz filters if they are ti be stable.

(6) But nonlinearities are low-pass. They demodulate high frequency components into low frequencies. All common nonlinearities reduce variability. They are low-pass demodulators.

(7) And in moving energy to low frequencies they enable stable discrimination. They enable a legitimate tradeoff between stability and discriminability that is impossible to attain with graph filters.

8.8 Equivariance, Stability, and Transference

Slide 53: Equivariance Stability and Transference - Title Page

(1) We close the lecture and the first part of the course with some reflections on equivariance, stability, and transference.

Slide 54: Good and Bad Questions

(1) My first reflection is about question classes. Some questions are good. Some questions are bad. And some questions are so bad they don't even have an answer.

(2) Questions like "does this work" or "should this work" are not scientific questions. They cannot be proven or disproven by experiment.

(3) They are in the category of questions that are so bad they don't have an answer.

(4) The theorems we have proven don't ascertain responses to a broad unspecific question like "should this work." But they do make scientific predictions. In the sense of claims that we can validate, indeed, have validated experimentally.

Slide 55: Permutation Equivariance: Predictions

(1) In our study of permutation equivariance we presented two theorems which borne out two predictions.

(2) We proved that graph filters are equivariant to permutations of the shift operator.

(3) And that GNNs, too, are equivariant to permutations of the shift operator.

(4) Out of the first theorem we predict that graph filters should outperform generic linear regression in problems that exhibit permutation equivariance.

(5) And of the second theorem we predict that **graph** neural networks should outperform generic **fully connected** neural networks in problems that exhibit permutation equivariance.

Slide 56: Permutation Equivariance: Experiment 1

(1) We run an experiment that demonstrated that, as predicted, graph filters outperform linear regression when learning rating predictions in recommendation systems.

Slide 57: Permutation Equivariance: Experiment 2

(1) And we also run an experiment that demonstrated that, as predicted, GNNs outperform fully connected NNs when learning rating predictions in recommendation systems.

(2) Incidentally, the equivariance of the task is crucial here. Since the filling in of ratings predictions is equivariant to graph permutations, the GNN does well. But you should **not expect** a GNN to work better than a fully connected NN if your problem is **not equivariant** to permutations. You cannot use a GNN just because a graph appears. You need a graph. And you need permutation equivariance of the task.

Slide 58: Stability to Deformations: Prediction

(1) In our study of the stability properties of GNNs we proved two theorems that put together lead to one prediction.

(2) We showed that graph filters are stable to relative deformations of the graph support if they are integral Lipschitz.

(3) And that GNNs whose layers are made up of integral Lipschitz filters are also stable to relative deformations of the graph support.

(4) These two theorems put together lead to the prediction that GNNs outperform graph filters.

(5) Because they are equally stable if they use filters with the same Lipschitz constant but the GNN is better at discriminating high frequencies. We have more discriminability for the same amount of stability.

Slide 59: Stability to Deformations: Experiment

(1) And we ran an experiments that confirms this prediction. GNNs outperform graph filters when learning rating predictions in recommendation systems.

(2) The difference is small. This is mostly because rating prediction is a low frequency problem. We similarity graph is such that rating signals have low variability over the graph. The GNN is not expected to do that much better than a simple graph filter.

Slide 60: An Orphan experiment

(1) We have also seen that it is possible to transfer a GNN to a graph with a much larger number of nodes. We can train in small networks. And transfer to large networks.

(2) This is an orphan experiment. It does not validate a prediction that follows from any piece of our theory. To give this child a mom, we will study transference with the introduction of graphon filters and graphon neural networks