Lecture 4 Script

1 Learning with Graph Signals

Slide 1: Learning with Graph Signals - Title Page

(1) This lecture is devoted to the introduction of graph neural networks. We begin with some reminders about empirical risk minimization and introduce the problem of learning with graph signals.

Slide 2: Empirical Risk Minimization

(1) Graph neural networks are the tool we use for machine learning on graphs. And, if you don't mind my reminding, in this course, machine learning is a synonym for empirical risk minimization.

(2) In empirical risk minimization, we are given three elements.

(3) The first elements is a training set T containing observation pairs of the form (x,y), where x is an input, or feature, and y is an output associated with x.

(4) Which, today and in the next few lectures, we are assuming are both of the same length *n*

(5) The second element is a loss function *ell* of *y* comma *y* hat that evaluates the similarity between output y and an estimates y hat of such output.

(6) The third element, which is arguably the most important, is a function class C.

(7) A machine learning problem entails finding.

(8) A function Phi star in the class function C.

(9) That minimizes the loss between the observed output y, and the output Phi of x, predicted by function Phi.

(10) Averaged over the elements of the training set.

(11) When we say machine learning, we refer to this empirical risk optimization problem and the process of finding the function Phi star in the class C is the process of training.

(12) Once we find Phi star of x, we can use it to estimate outputs y hat when inputs x are observed but outputs y are unknown. The goal of ML is for these live operation estimates to be good. Producing good estimates in the training set is useful only insofar as it is conducive to producing good estimates during live operation.

Slide 3: Empirical Risk Minimization

(1) In empirical risk minimization problems, which we rewrite here for reference, the training set and the loss function are given once the problem is given.

(2) Which leaves the function class C as the degree of freedom that is available to the system's designer.

(3) Thus, the problem of designing a machine learning method can be equated to the problem of finding the right function class C.

(4) And since we are interested in graph signals like the ones we show below, graph convolutional filters are a good starting point to search for and appropriate class.

Slide 4: Learning with a Graph Convolutional Filter

(1) Let us then describe a learning system with a convolutional graph filter.

(2) The input signals x are graph signals that are all of them supported on the same common graph with shift operator S.

(3) The same is true of output signals y. They are also graph signals supported on the same graph S. Which is also the same graph that supports the input signals.

(4) Given that inputs and outputs are graph signals supported on S, we choose as function class the set of filters of length capital K that are supported on the graph S.

(5) The function class Phi produces outputs Phi of x, by multiplying x with a polynomial on the shift operator S modulated by coefficients h_k. The polynomial is of order K-1 and the total number of coefficients is K.

(6) This is a function class that is parameterized by both the shift operator S and the filter coefficients h. The shift operator represents given prior information and the filter coefficients h are the parameter we want to learn.

(7) To fix ideas we describe this parametrization with a block diagram.

(8) In which the graph signal x.

(9) Is the input to a graph convolutional filter of length K.

(10) That produces the output z equals Phi of x. The output is parametrized by the shift operator S and the filter coefficients h.

(11) With this choice of parameterization, learning reduces to finding the optimal set of filter coefficients h star that minimize the loss averaged over the training set.

(12) We emphasize that even though the function class is parameterized by both the shift operator S and the filter coefficients h, the optimization is only over the set of coefficients h with the shift operator given. The filter h is learnable. The shift operator is prior information we leverage.

Slide 5: When the Output is Not a Graph Signal: Readout

(1) When the outputs we are trying to predict are not graph signals, we add a readout layer to match dimensions.

(2) Namely, introduce a matrix A with n columns and m rows and use a parametrization

(3) In which we multiply the output of a graph filter, which is a graph signal with n components, by the matrix A.

(4) Thus. We begin with the graph signal x, which we process with a graph filter to produce the signal z. Which is also supported on graph S.

(5) We multiply this signal with A to produce output predictions as the product of A with the output of the graph filter.

(6) While it is possible to make the readout layer a trainable parameter, this is in general inadvisable.

(7) It is more advisable to simply chose a suitable readout matrix A and train the graph filter only

(8) Thus, the ERM problem is not much different from the one before. It is, in fact, the same but with a change in the loss function. Instead of comparing observed outputs y to the output of a graph filter, we compare them to a readout of the output of the graph filter.

(9) The motivation for simply choosing A is that is to retain the advantages of using a parametrization that leverages the graph. And we can get away with using simple readouts because most situations with dimension mismatch require simple readouts.

(10) For instance, if we want to read out the signal value at node i, we choose the i-th vector of the canonical basis. This is useful in recommendation systems. Or in any problem in which we are interested in values of individual nodes.

(11) If we want to classify a signal, a convenient way to do is to read an average. We can read this out with the all one vector.

2 Graph Neural Networks (GNNs)

Slide 6: Graph Neural Networks (GNNs) - Title Page

(1) We have reached the momentous time when we are ready to define graph neural networks.

Slide 7: Pointwise Nonlinearities

(1) We start with a brief parenthesis to discuss pointwise nonlinear functions

(2) That a function is a pointwise nonlinearity means that it when applied to a vector x it is applied to individual components. Without mixing entries

(3) More precisely, the result of apply the pointwise nonlinear function sigma to the vector x.

(4) Which is a vector made up of entries x_1 through x_n.

(5) Is equal to the vector that stacks sigma of x_1, sigma of x_2, all the way through sigma of x_n. It applies the nonlinearity to each entry individually. Entries are not mixed with each other in the application of sigma. The result that appears in position k of the vector sigma of x, is sigma of x_k.

(6) It is pertinent to emphasize that pointwise nonlinearities are sort of the simplest nonlinear function we can apply to a vector x. They are pointwise. Applied entry by entry.

(7) Pointwise nonlinear functions are used in convolutional and non-convolutional neural networks. The most widespread is the rectified linear unit that zeros all the negative components and retains all the positive ones.

(8) The hyperbolic tangent characterized by a sigmoid graph is another choice.

(9) And so is the absolute value of the x. There are a dozen more choices of pointwise nonlinear functions. It is generally accepted that different choices make marginal differences.

(10) An important observation to make about these three specific pointwise nonlinearities and other possible choices is that they all reduce variability. The ReLU eliminates the variations of all negative components. The hyperbolic tangent saturates large entries. The absolute value makes all negative entries positive.

(11) In the parlance of signal processing this is called a demodulation. It generates signals with more energy concentrated in low frequencies. In signals that change more slowly over the graph.

Slide 8: Learning with a Graph Perceptron

(1) We close the parenthesis and return to learning with graph signals. In the previous video we studied how to learn with graph signals using graph filters. This is shown in this block diagram where the input graph signal x is multiplied by a polynomial on the shift operator matrix.

(2) A limitation of graph filters is that they have limited expressive power. A fact that is in turn due to their linearity. Graph filters can only learn linear functions of x.

(3) A first approach at a function class C that can learn nonlinear maps, is the graph perceptron.

(4) To build a graph perceptron we process the output of the graph filter.

(5) With a pointwise nonlinear function sigma.

(6) That sigma is a pointwise nonlinearity is a key restriction. As we have just seen, it means that sigma is applied to the vector x component by component.

(7) A graph perceptron is determined by the choice of graph and filter coefficients. Therefore, the family of graph perceptrons can be written as a family that is parameterized by the shift operator S and the filter coefficients h. As in the case of graph filters, the coefficients h are trainable but the graph S is given prior knowledge.

(8) This is then the block diagram of a graph perceptron. It composes a graph filter with a pointwise nonlinearity. The fact that the nonlinearity is applied componentwise, implies that graph perceptrons are minor modifications of graph filters. Nevertheless, given that we will use perceptrons to build GNNs, it warrants a repeat explanation.

(9) (Empty)

(10) In the block diagram, we begin with the graph signal x as an input.

(11) Which we send into a block where it is processed with a graph filter. This is the same graph filter that we used before.

(12) The output of the graph filter is z. But now, instead of becoming the output of the learning parametrization.

(13) It is fed into a pointwise nonlinear function sigma.

(14) This produce the output, function Phi. At the risk of overstaying my welcome, I emphasize that the nonlinear function sigma is pointwise or elementwise. It applies to the filter output z individually. It is for this reason that we categorize perceptrons as minor modifications of filters.

(15) With the graph perceptron parameterization, the learning problem looks the same as before. We still search for the optimal set of filter coefficients h that minimize the loss averaged over the training set. It's only that now, the function Phi is not simply a graph filter. It is a graph filter that is post-processed with a pointwise or elementwise nonlinearity. As before, the graph shift operator S is part of the parameterization, but it is not part of the optimization.

(16) Although we have emphasized the conceptual proximity of filters and perceptrons there is a substantial difference. The addition of the nonlinearity allows the perceptron to learn nonlinear maps. It therefore renders the model more expressive. It can represent a larger set of functions.

Slide 9 - Graph Neural Networks (GNNs)

(1) The quest for further increases in expressive power is what leads to the introduction of graph neural networks.

(2) To build more complex nonlinear functions, a GNN composes several graph perceptrons.

(3) It stacks perceptrons. It layers a set of graph perceptrons

(4) More precisely, Layer 1 of the GNN.

(5) Takes the input signal x.

(6) And processes the signal with a perceptron. This perceptron is made up of graph filter followed by application of a pointwise nonlinearity sigma. The perceptron is defined by the coefficients of the filter. We denote them by h_1 to clarify that they are specific to the layer.

(7) The Layer 1 perceptron yields an output x_1. Which becomes the output of Layer 1.

(8) Instead of stopping here, The output of Layer 1 becomes and input to Layer 2. It is still the same signal, but It changes roles. If goes from being the blue output of Layer 1 to being the green input to Layer 2.

(9) Layer 2 processes

(10) Its input signal x_1, which a second ago was the output of layer 1,

(11) With a perceptron defined by coefficients h_2. This perceptron is a composition of the graph filter that uses these coefficients with the pointwise nonlinearity sigma. The coefficients are specific to the layer.

(12) The output of the layer 2 perceptron is denoted by x_2. Which is also the output of Layer 2.

(13) The output of layer 2 is now going to become an inout to layer 3. As was the case when we went from Layer 1 to Layer 2, this is just a change of roles. A change of colors. But it’s the same signal.

(14) Layer 3 is now going to process the output of layer 2 with a specific perceptron. It will send the output to Layer 4, where it will be processed by another perceptron and so on. We repeat this process a total of capital L times. This capital L is the depth of the GNN. Its total number of layers. The total number of perceptrons that we compose.

(15) The output of the last layer, is x sub capital L. This is declared to be the output of the GNN.

Slide 10 - The GNN Layer Recursion

(1) According to this description of a GNN, a generic layer of a GNN is a perceptron that takes as input the output of the previous layer. Consider then layer ell, which takes as input the output x_ell-minus-one of layer ell-minus-one.

(2) Layer ell processes

(3) Its input signal x_ell-minus-one.

(4) With a perceptron defined by its own specific filter. Which is defined by a set of coefficients h_ell.

(5) The result of this processing is the output of the layer, x_ell.

(6) Agreeing that the input to layer 1, which is the given signal x, be reinterpreted as the output of the nonexistent layer 0, the equation above provides a recursive definition of GNNs.

(7) If the GNN has capital L layers, the output of the GNN is the result of stopping the recursion after L iterations.

(8) This results in a function class Phi that is parametrized by the shift operator S and the set of filter coefficients h_1 through h_L.

(9) To simplify notation we define the filter tensor calligraphic H and write the function class as parametric on the shift operator S and the filter tensor H

(10) The filter tensor H is the trainable parameter of the GNN. The shift operator is prior information.

Slide 11 - GNN Block Diagram

(1) To cement our understanding of the definition of a graph neural network, consider an example GNN with 3 layers. We represent it with the block diagram shown on the right.

(2) (Empty)

(3) We begin by feeding the input graph signal x to the graph perceptron in layer one. This signal is also understood as the output of the nonexistent Layer 0.

(4) Within Layer 1, the signal x_0 is first processed with a graph filter. This filter uses the given shift operator S and a set of coefficients h_1-k which we will later train.

(5) This graph filter produces an internal output z_1

(6) Which we feed into the pointwise nonlinear function sigma.

(7) The result of applying this pointwise nonlinearity to the output of the graph filter is the signal x_1.

(8) This completes Layer 1. The signal x_1 is the output of Layer 1.

(9) The output of Layer 1 is now feed as an input to Layer 2.

(10) Where, again, it is processed by a filter with a specific set of coefficients h_2-k which we will later train.

(11) To produce an internal output z_2

(12) Which we feed to a pointwise nonlinear function

(13) To produce the signal x_2

(14) This completes the layer and x_2 is declared to be the output of Layer 2.

(15) The output of Layer 2 is not fed as an input to Layer 3.

(16) Where it is processed by a filter with trainable parameters h_3-k

(17) To produce internal output z_3

(18) Which we process with the pointwise nonlinear function sigma

(19) To produce the signal x_3

(20) This completes Layer 3.

(21) Since this is a GNN with 3 layers, the output of layer 3 is declared to be the output of the GNN. If the GNN had more than 3 layers, the process of transmitting layer outputs into layer inputs would continue until we reach the final layer. In any event, we denote the output of the GNN as Phi of x, S and H. In this notation, x is the input signal and S is the shift operator, which we assume given.

(22) The filter tensor H groups all of the filter coefficients. This is the trainable parameter of the GNN.

3 Some Observations about Graph Neural Networks

Slide 12: Some Observations Graph Neural Networks - Title Page

(1) There are several important remarks we have to make about graph neural networks.

Slide 13: The Components of a Graph Neural Network

(1) We take a minute to summarize and highlight the components of a GNN. A GNN with L layers is defined as L recursive compositions of graph perceptrons. In which the input signal x is rewritten as x-sub-zero.

(2) This is a composition of L layers. Each of which is itself a composition.

(3) Of a Filter

(4) With a Pointwise nonlinearity.

(5) The filters that appear in each layer are parametrized.

(6) By sets of coefficients h_l-k. Coefficients are attributes of individual layers.

(7) And they are also parametrized by a graph shift operator S. This is the sam at all layers.

(8) The output of the GNN is a the output x_L of layer L. We represent this map as Phi-of-x-S-and-H.

(9) The learnable parameters in this function class is the filter tensor calligraphic H. Which is a grouping of all the filter coefficients h_l-k that are used across all layers.

Slide 14: Learning with a Graph Neural Network

(1) The problem of learning with a GNN reduces to the problem of finding the tensor H star that minimizes the average loss over the training set. This is analogous to the problems of learning with a perceptron or learning with a graph filter. Except that the tensor H contains coefficients for a group of filters. Instead of containing coefficients for a single filter only.

(2) As was also the case of learning with a perceptron and a graph filter, the graph shift operator S is given. It is not part of the optimization space.

(3) The shift operator is interpreted here as prior information that is given to the GNN for leverage.

Slide 15: Graph Neural Networks and Graph Filters

(1) A subtle point that I don't want to leave unnoticed is that GNNs are minor variations of graph filters. We have said this of perceptrons already. They are, in a sense, the easiest possible modification to transform linear graph filters into a nonlinear function class. The same is true of GNNs

(2) Their only difference with graph filters is the addition of pointwise nonlinearities and layer compositions.

(3) Since the nonlinearities are pointwise, they process signal entries individually. There is no mixing of components carried out by a nonlinear transformation.

(4) All of the component mixing that goes on in a GNN is carried out by linear transformations. More precisely, it is carried out by graph filter. A consequence of this observation is that if we understand the behavior of graph filters, we also understand the behavior of GNNs. There is little difference between one and the other. They are, conceptually, very close relatives.

(5) Now, despite their conceptual proximity, graph neural networks do work better than graph filters in practice. Sometimes, much better. We are showcasing the truth of this statement in labs 2 and 3.

(6) This is, in the face of it, unexpected. How come such a minor variation can produce significant differences in practice? There are good reasons, related to signal invariance as we will explore soon, to expect graph filters to work well. Since GNNs are close to graph filters, we should expect them to work well, too. But not better. Certainly, not much better.

(7) But reality is reality. Experiments are there to be explained. There are somewhat unexpected stability properties of GNNs that explain their better performance relative to graph filters. We will study this in upcoming lectures.

Slide 16: Transference of GNNs Across Graphs

(1) Another subtle point I want to make sure I emphasize, has to do with the transference of graph neural networks across different graphs.

(2) We know that GNN outputs depend on the graph shift operator S.

(3) We can interpret S as a non-trainable parameter that we pass to the GNN.

(4) A parameter that encodes prior information that we feed to the GNN for leverage. This is the perspective we have emphasized so far. We have thought of S as a way of encoding prior information about our signals of interest.

(5) But nothing prevents us from reinterpreting S as an input to the GNN.

(6) This interpretation enables transference across different graphs.

(7) Indeed, for a given filter tensor we can execute the GNN using a graph S as an input or using another graph S-tilde as an input.

(8) This is analogous to transferring the GNN across signals.

(9) For a given filter we can execute the GNN with input signal x and we can execute the GNN for input signal x-tilde

(10) At the end of the day, what matters is that trained GNN is just a filter tensor H-star. The filter tensor can be executed on different graphs in the same way in which it can be executed on different signals. To close this digression on transference of GNNs, let me say that the word transference is reserved to the case when a GNN that has been trained on graph S is executed on graph S-tilde. Alternatively, we may choose to train on a family of graphs. In the same way in which we train on a family of signals. In this latter case we say that the GNN generalizes across different graphs of the family. It is not that

the GNN is transferred to something we haven't seen during training. But that the GNN is sufficiently general that it captures the whole variety of graphs we have seen during training. I find this distinction somewhat pedantic. In both cases we simply interpret the shift operator S as an input. The difference is whether we change it during training or not. But the reviewers of your papers may disagree and there is no cost in using standard language.

Slide 17: CNNs and GNNs

(1) The final point that I want to illustrate has to do with the relationship between CNNs and GNNs.

(2) This diagram illustrates a **G**NN. A **graph** neural network

(3) This other diagram illustrates a **C**NN. A **convolutional** neural network.

(4) See the difference? Let's repeat that.

(5) This is a diagram illustrating a **graph** neural network

(6) And this is a diagram illustrating a **convolutional** neural network.

(7) If you see no difference between the two, it is because there is **no** difference between a **C**NN and a **G**NN.

(8) To recover a CNN, we just particularize the shift operator to the adjacency matrix of the directed line graph. We know that this is true because we have seen that convolutional filters in time are particular cases of graph convolutional filters. This equivalence comes from particularizing the shift operator to the adjacency matrix of the line graph. There is nothing to be said about the pointwise nonlinearity, which is, well, pointwise. It's always the same no matter what. It does not mix components. It is unaware of the underlying structure of the signal.

(9) If you didn't know what CNNs were, this lecture is a Toofer. You get to learn what GNNs are. And you get to learn what CNNs are. If that doesn't excite you, do remember that the ability to recover CNNs from GNNs implies that graph neural networks are proper generalizations of convolutional neural networks. We can obtain the latter as a particular case of the former.

4 Fully Connected Neural Networks

Slide 18: Fully Connected Neural Networks - Title Page

(1) Among many interesting things we said about graph neural networks is that they generalize Convolutional neural networks. A somewhat converse perspective is that GNNs are particular cases of fully connected neural networks. This latter one, is a connection that deserves exploration.

Slide 19: The Road Not Taken: Fully Connected Neural Networks

(1) We have chosen to work with graph filters and graph neural networks because of our interest in graph signals

(2) Like the signals that appear in the word adjacency networks, the wireless communication network, or the recommendation system we illustrate here.

(3) We have argued that this is a good idea because graph neural networks and graph filters are generalizations of convolutional filters and convolutional neural networks.

(4) But we can shed further light in the reasons to choose graph filters and GNNS if we go back and investigate the road not taken. That was the road that could had led us towards fully connected neural networks.

Slide 20: Learning with a Linear Classifier

(1) When we decided to work with graph convolutional filters, we could have chosen to work with arbitrary linear maps . Had we done so, the AI function Phi would belong to the class of arbitrary linear functions given by the product of the signal x with an arbitrary matrix H. This would give a block diagram like the one we show here

(2) In which the input signal x.

(3) Is multiplied by a matrix H.

(4) To produce the output Phi of x comma H as the product H times x.

(5) The resulting learning problem would had been one in which we want to find the matrix H star that minimizes the average loss over the training set across this arbitrary linear parametrization.

Slide 21: Learning with a Linear Perceptron

(1) From a linear classifier, we move to a linear perceptron. As opposed to moving on to a graph perceptron. In this perceptron the output of the linear transformation is processed with a pointwise nonlinearity. Observe how the pointwise nonlinear is always the same.Different information processing architectures are designed by selecting different classes of linear transforms. This arbitrary linear perceptron, can be represented by this block diagram.

(2) In which the signal x.

(3) Is fed into the arbitrary linear transformation.

(4) To produce the intermediate signal z,

(5) Which we process with a pointwise nonlinearity sigma

(6) To yield the output Phi of the arbitrary linear perceptron.

(7) The learning problem is more or less the same as before. We want to find the best matrix H star that minimizes the loss averaged over the training set. The only difference is that the parametrization Phi is slightly different now. It involves the the composition of a linear transform with a pointwise nonlinearity.

Slide 22: Fully Connected Neural Network

(1) The linear perceptron brings us to fully connected neural networks. In the same way in which the graph perceptron brought us to graph neural networks. We can go over quickly because it's more or less the same.

(2) A generic layer of a Fully connected neural network takes as input the output of the previous layer. Same as a GNN. Consider then layer ell, which takes as input the output x_ell-minus-one of layer ell-minus-one.

(3) Layer ell processes

(4) Its input signal x_ell-minus-one.

(5) With a perceptron defined by the linear transformation H_ell. This is a generic linear map. As opposed to the graph filters we used in GNNs.

(6) The result of this processing is the output of the layer, x_ell.

(7) Agreeing that the input to layer 1, which is the given signal x, be reinterpreted as the output of the nonexistent layer 0, the equation above provides a recursive definition of an FCNN.

(8) The output of the fully connected neural network is the result of stopping the recursion after L iterations.

(9) This results in a function class Phi that is parametrized by the set of matrices capital H_1 through capital H_L. Not filter coefficients as in the GNN.

(10) We define the tensor calligraphic H to write the FCNN with more compact notation.

(11) The filter tensor H is the trainable parameter of the GNN.

Slide 23: Fully Connected Neural Network Block Diagram

(1) This was more or less the same description we gave go a GNN. The only thing that changed is that we use arbitrary linear transformations in lieu of graph filters. We see the same in this block diagram of an FCNN with 3 layers. It's the same diagram except that matrices H_ell take the place of graph filters.

(2) (Empty)

(3) Thus, as in the GNN, we begin by feeding the input graph signal x to layer 1.

(4) But this is now processed with a generic matrix multiplication H one times x.

(5) This graph filter produces an internal output z_1

(6) Which we feed into the pointwise nonlinear function sigma. As we did for GNNs.

(7) The output of the pointwise nonlinearity is the signal x_1.

(8) This completes Layer 1.

(9) The output of Layer 1 is now fed as an input to Layer 2.

(10) Where, again, it is processed by a matrix H_2.

(11) To produce an internal output z_2

(12) Which we feed to a pointwise nonlinear function

(13) To produce the signal x_2

(14) This completes the layer and x_2 is declared to be the output of Layer 2.

(15) The output of Layer 2 is not fed as an input to Layer 3.

(16) Where it is processed with matrix H_3

(17) To produce internal output z_3

(18) Which we process with the pointwise nonlinear function sigma

(19) To produce the signal x_3

(20) This completes Layer 3

(21) And the signal x_3 at the output of this layer is declared to be the output of the Fully connected neural network.

(22) The output is parametrized by the filter tensor calligraphic H grouping the matrices of all layers. This is the trainable parameter of the FCNN.

5 Neural Networks vs Graph Neural Networks

Slide 24: Neural Networks vs Graph Neural Networks - Title Page

(1) Having defined fully connected neural networks and graph neural networks, we are ready to compare them.

Slide 25: Which is Better, a GNN or an Arbitrary NN?

(1) Graph neural networks and fully connected neural networks have very similar architectures. They both use layers, which are composed of linear transformations and pointwise nonlinearities. The difference is that arbitrary neural networks utilize arbitrary linear transformations, whereas graph neural networks rely on graph filters. An important question is which of these two architectures we expect to work better.

(2) The first important point to make is that the GNN is a particular case of a fully connected neural network where we impose a particular structure on the linear map. A consequence of this fact is that if we compare

(3) The best possible cost that is attainable by a fully connected neural network

(4) With the best possible cost that is attainable by a GNN,

(5) The cost attained by the neural network is smaller.

(6) This is because the optimization set of the neural network includes the optimization set of the GNN. Whatever transformation can be implemented with a GNN is a particular case of a transformation that can be implemented with an FCNN.

(7) This seems to indicate that the fully connected neural network does better, but this reduction in cost holds for the training set, which does not necessarily translate to the operation of the neural network.

(8) In practice, the GNN does better during operation because it generalizes better to signals or to examples that have not yet been seen,

(9) And, in turn, this happens because it successfully exploits internal symmetries of graph signals that are codified by the graph shift operator. This is a somewhat obscure statement. But it is also the reason why we are studying GNNs instead of just using Fully connected neural networks. We will therefore try to clarify with an example.

Slide 26: Generalization with a Neural Network

(1) The diagrams are a cartoon illustration of a recommendation system where the colored nodes represent available ratings. Our objective is to predict the ratings associated with the clear nodes.

(2) Suppose that we observe ratings with the structure on the left,

(3) But we never get to observe examples like the other two.

(4) That is, the signal on the left is observed during training, but the other two examples are observed only during the execution of the neural network.

(5) From the examples like the one in the left,

(6) The neural network should be able to infer how to fill the ratings for the signal in the middle.

(7) But it's unreasonable that it will learn how to fill the ratings for the signal on the right. There is nothing in this right signal, save for the graph, that can make it learnable from the signal on the left. This is obvious, but if you don't find it obvious, remember that the graph is unknown to the FCNN. All the FCNN knows are the signal indexes. And how is the FCNN going to learn how to fill entry x_6 on the right after it has learnt to fill entry x_3?

Slide 27: Generalization with a Graph Neural Network

(1) But who knows the graph? The GNN does! And because it knows the graph, it will succeed at learning how to fill the signal on the right.

(2) Indeed, if we learn how to fill the signal on the left with a GNN during training.

(3) The GNN will also learn how to fill the signal in the middle. Same as the FCNN.

(4) But it will also learn how to fill the signal on the right. Which the FCNN did not.

(5) This is because the local structure of both signals, the one on the left and the one on the right, are identical. This is a property that the filters that make up the layers to the GNN can exploit. The operations that a graph filter performs to predict the value of x_6 for the signal on the right, are the exact same operations that the graph filter performs to predict the value of x_3 for the signal on the left.

Slide 28: Permutation Equivariance of Graph Neural Network

(1) This is what we mean when we say that GNNs exploit signal symmetries. There are symmetries in signals that make seemingly disparate examples equivalent. Like it happens for the signals on the right and left. They are different. But they can be processed in the same manner. The symmetries effectively multiply the size of the data set.

(2) This intuitive idea will be formalized later in the form of the permutation equivariance of graph neural networks. And it will also connect with stability notions, which is what will allows us to exploit quasi-symmetries as opposed to exploit exact symmetries only. Incidentally, a similar story holds for CNNs. Which instead of permutation quasi equivariance exploit translation quasi equivariance.

6 Graph Filter Banks

Slide 29: Graph Filter Banks - Title Page

(1) Filters isolate signal features. When we consider problems where we foresee multiple features to be of interest, we use filter banks.

Slide 30: Graph Filter Banks

(1) A graph filter bank is a collection of graph filters that we apply to an input signal. We use capital F to denote the total number of filters in the bank.

(2) We index filters in the bank with lowercase f and denote the coefficients of filter f as h sub k super f. The output of this f-th filter is the graph signal z with the filter index f noted as a superscript.

(3) In this block diagram of a filter bank, the input signal is x.

(4) We feed that signal to a graph filter with coefficients h_k superscript 1. We obtain the output z superscript 1. This output is a graph signal, which same as x, is supported on the graph S.

(5) Signal x is, in parallel, also fed to filter 2, which has coefficients h_k-2 and produces output z superscript 2.

(6) And we continue feeding x in parallel to other filters

(7) Until we reach filter uppercase F. Which has coefficients h_k-uppercase F and produces as output the signal z-F

(8) The signal x and all of the outputs z-f are graph signals supported on a common graph with shift operator S. Thus, the output of a filter bank is a collection of F graph signals.

(9) We write these signals as the Matrix graph signal capital Z in which each of the columns of the matrix Z is a graph signal.

Slide 31: Filter Bank Outputs: Multiple Features

(1) This matrix graph signal is a new object. Let's make sure we understand it.

(2) The input of a filter bank is a regular graph signal. As we show in this illustration, each row of the vector x is a component x_i which is associated with node i.

(3) The output Z is a collection of graph signals z-f. The components of the signal z_f have rows z_i-f associated with nodes of the graph. Each node of the graph is now supporting multiple values. The output of the filter bank is like a book with several pages. Each of these pages is a graph signal.

(4) Reading across the pages of this book we obtain a vector z_i supported at each node. There are therefore two ways of reading the matrix graph signal Z.

(5) Different columns of Z are graph signals z-f. They are different pages of the book. They are indexed by lowercase f and there are uppercase F of them

(6) Rows of Z are node features. They are vectors z_i associated with individual nodes. We read across pages of the book. There are n of them. One per node.

Slide 32: Output Energy of a Graph Filter in the GFT Domain

(1) Our introduction of filter banks is intended to introduce multiple feature GNNs. Their definition is all we need to that end. But understanding filter banks is helpful in understanding GNNs. To understand filter banks we look at their representations in the GFT domain. As you should had expected.

(2) Our exploration begins with an ancillary theorem where we evaluate the output energy of a graph filter.

(3) In this theorem we consider a single graph filter h with coefficients h_k

(4) And frequency response tilde h of lambda. This is, we recall, a polynomial with coefficients h_k on a scalar variable.

(5) We are interested in the output z of this graph filter, which is the same polynomial but on variable S. In particular, we want to evaluate its energy.

(6) This energy is the norm of the signal squared,

(7) And the theorem claims that it equals the sum of squares

(8) Of the product between the filter's frequency response evaluated at the eigenvalues of the shift operator

(9) And the respective GFT components of the input signal x. Notice that the sum is over all GFT components I, which is the same as saying that it is over all eigenvalues. Since there is a bijective correspondence between eigenvalues of S and GFT components of a signal.

Slide 33: Proof of Output Energy Theorem

(1) The theorem is true because the GFT is a unitary transform that preserves energy. Consider the GFT tilde z of the filter's output z.

(2) The energy of this GFT

(3) Is the inner product of the GFT with itself

(4) Which using the definition of the GFT can be written as the inner product of V Hermitian Z with itself.

(5) In expanding this product we end up with a V times V Hermiatian product in the middle

(6) Which we know is an identity. We end up with the inner product of z with itself.

(7) Which allows us to conclude that the GFT energy is the same as the energy of z

(8) On the other hand, we know that graph filters are pointwise in the frequency domain. We can therefore write the components of the output GFT tilde z_i as the product of the frequency response tilde h of lambda_i and the input GFT tilde x_i. The three of them evaluated at the same index.

(9) Using this expression the energy in the GFT domain.

(10)Which we can write as the sum of squares of the individual GFT components.

(11) Is the sum of squares of the products between tilde h of lambda_i and tilde x_i.

(12) We have then expressed the energy in the form we want. Except that the expression is in the GFT domain and we wanted to have it in the node domain.

(13) This is not a problem as we have just seen that the GFT preserves energy. Thus, the energy in the node domain equals the energy in the GFT domain, which is a sum of squares of the products between the components of the input GFT and frequency response of the filter evaluated at the respective eigenvalues.

(14) Which is what we wanted to show.

Slide 34: Filter Banks in the Graph Frequency Domain

(1) Out of this theorem we can think of the energy that graph filters let pass as a sort of area under the curve. Area under the frequency response curve.

(2) This diagram here represents the squared GFT of an input signal

(3) Where we add this curve to represent the squared response of a graph filter

(4) To capture the energy of the output we compute the product between the two and sum. We compute the area under the curve. Sort of. We're actually summing spikes. But conceptually close enough.

(5) An interesting aspect of this GFT diagram is that it illustrates that graph filters identify different frequency signatures. This filter lets pass frequencies associated with small lambda only. This is matched to the input signal. Which is made of GFT components associated with small lambda as well. There is substantial energy at the output of this filter when applied to this signal. By measuring the energy at the output of the filter we identify the frequency signature of the input signal.

(6) This other filter lets pass components that are associated with large lambda. When applied to this signal there is little energy at the output. We know that the input does not have this GFT signature.

(7) When we consider a different signal the roles of the filters may get flipped. In this example the first filter lets pass little energy

(8) But the second filter matches the frequency signature of this signal. There is substantial energy at the output.

(9) The effect of a graph filter bank is to scatter the energy of the signal on different outputs z-f as different filters pick up energy that is concentrated in different GFT components.

(10) This allows filter banks to pick up different signal signatures by concentrating energy on different outputs. This is helpful because the detection of a signal becomes elementary. Just identify the filter that has accumulated more energy. In this example the two filters in the bank scatter energy by letting pass low frequency components or high frequency components. This permits elementary separation between signals with energy concentrated in low frequencies and signals with energy concerned in high frequencies.

Slide 35: Filter Banks as a Transforms

(1) The scattering of energy in a graph filter bank is the reason why they are useful in machine learning. Indirectly, it is the reason why GNNs with multiple features are useful. The filter bank is like a transform that represents the signal in an alternative domain.

(2) This is an interpretation they share with the GFT. The GFT rewrites the signal by isolating individual frequency components. Filter banks, isolate groups of frequencies.

(3) Like in this example we have three filters that are trying to capture different signatures. Low, medium, and high frequencies, if you wish.

(4) The energy of the input signal gets scattered into the three different outputs z-f. This is because the energies are the different areas under the frequency response curves and the filters are chosen to accumulate energy on different supports.

(5) We are using the filter bank to facilitate identification of signals with different spectral signatures. It is just matter of comparing the energy of the different outputs

Slide 36: Energy Conservation in Filter Banks

(1) Thinking of filter banks as transforms analogous to the GFT draws attention to energy conservation. As we have seen, the GFT preserves energy. The energy at the output is the same as the energy at the input. This is important because maintaining energy implies that the GFT scatters information across different GFT components. But it doesn’t loose information. Whatever signatures were contained in signal x are still present in its transform.

(2) The same is not necessarily true of filter banks. Unless they are designed to have that property. We do that by imposing restrictions on the filter choice. A filer bank is said to be

a frame if there exist constant little m and uppercase M such that sum of the energies at the output o the filter bank scales the energy of the input signal but no less than little m and not more than capital M. If the two constants lowercase m and uppercase M are moderate, the energy at the output of a frame filter bank is not too different from the energy at the input. This is required to hold for all signals x.

(3) If both constants are 1, the inequalities must hold with equality. When this happens we say the filter bank is a tight frame. The equality implies energy conservation. The energy of the input signal is the same as the sum of the energies of the individual outputs signals of the tight frame filter bank.

(4) The important facts to remember are that no signal is vanquished by a frame, nor is it amplified by an infinite amount, but this is nota s relevant.

(5) And that a tight frame preserves energy

Slide 37: Frames in the Graph Frequency Domain

(1) Frame and tight frame conditions look impossible to check. They must hold for all input signals. But frame and tight frame conditions are easy to check because filters are pointwise operations in the frequency domain. To check if a filter bank is a frame we sum the squared frequency responses of all filters and check that the sum is between lower and upper case M for all lambda

(2) The diagram illustrates a frame because the sum of the squared responses is, as we show, finite and bounded away from 0

(3) A simple sufficient condition to have a frame is for all frequencies to have at least one filter that lets pass that frequency.

Slide 38: Tight Frames in the Graph Frequency Domain

(1) Analogously, to check if we have a tight fame we look at the sum of the squared frequency responses and see if the sum equals one or not. We have a tight frame if the sum is one.

(2) The filters we show here are a tight frame because their squared frequency responses add up to 1.

(3) In a tight frame, all frequencies accumulate unit energy when summing across all filters.

7 Multiple Feature GNNs

Slide 39: Multiple Feature GNNs - Title Page

(1) We defined GNNs by leveraging filters. Now that we have defined filter banks, we use them to define GNNs that process multiple features per layer.

Slide 40: Multiple Feature (Matrix) Graph Signals

(1) The output os a graph filter bank is a collection of multiple graph signals.

(2) Something we represent with a matrix graph signal Z. Each of these columns is a graph signal. Varying from z superscript 1 to z superscript f.

(3) The collection of these F graph signals represents a collection of F features per node. A book with may pages. Each of which is a graph signal.

(4) It equivalently signifies the presence of a vector z_i with F components supported at each node.

(5) We would like to process this graph signal with multiple features. More concretely, we would like to process it with a filter bank.

Slide 41: Multiple-Input-Multiple-Output (MIMO) Graph Filters

(1) Formally, for given input feature x superscript f we consider a bak of capital G filters with coefficients h_k annotated with superscripts f and g. This graph filters applied to signal x^f produce an output signal u-f-g. The output signal depends on the input feature x^f and the filter coefficients h_k-f-g.

Slide 42: Multiple-Input-Multiple-Output (MIMO) Graph Filters

(1) Since we have a total of F input features and each of them is processed with a filter bank containing G features we have a total of F times G filter that are run in parallel to produce as many features. We call this structure a Multiple-Input-Multiple-Output filter. Because we have both, multiple inputs and multiple outputs.

Slide 43: Multiple-Input-Multiple-Output (MIMO) Graph Filters

(1) Using this MIMO filter in a GNN, where we want to stack layers would lead to exponential growth in the number of features. This is not necessarily undesirable. But to exercise more control on the number of output features, we reduce the number of outputs to G by summing across input features for a given g. In the diagram below moving from left to right we vary output features and moving from front to back we vary input features. The outputs of our MIMO filters are signals z-g that sum the u-f-g features from front to back. This particular choice of sum to reduce the number of features at the output of a MIMO filter is more or less arbitrary. But it is chosen so that all input features can be represented in all output features.

(2) For instance, output z superscript 1 sums the intermediate outputs u superscript f, comma 1 for all f. Each of this u-f-1 outputs results from the processing of different input features x-f with a different filter in the MIMO bank.

(3) The same is true of output z-2, which sums outputs u-f-2 for all input feature indexes f

(4) And for all other output features going up the the last output z-capital-G

Slide 44: MIMO Graph Filters with Matrix Graph Signals

(1) This description of MIMO filters makes them look difficult. But they are not. They are cumbersome. They are simply a collection of F times G filters. Or a collection of F filter banks.

(2) We can make them easier if we represent them with matrix notation. To that end, let H_k be a G times F matrix in which the entry in row f column g is the filter coefficient h_k-f-g.

(3) With this notation the MIMO graph filter we just introduced can be written more compactly.

(4) The output is a matrix graph signal Z given

(5) By a sum of diffusion indexes k

(6) Of powers of the shift operators

(7) Multiplying the input signal in matrix form

(8) And multiplying the filter matrix H_k

(9) This is a more compact notation for the MIMO filter. It is equivalent.

(10) If you need help seeing that the definitions are indeed equivalent, just expand the matrices. When we do that we see that the graph filter output z-g has the same expression we covered some seconds ago. This is just algebra. There's no point in doing it here.

(11) What matters is the expression for a MIMO graph filter in which an input matrix signal X with F features is processed to produce an output matrix signal Z with G features. The form of the filter is reminiscent of a polynomial on the shift operator S in that it contains powers of this matrix. The coefficients of a MIMO filter are not a set of scalars lowercase h_k but a set of G times F matrices uppercase H_k that multiply from the right. This structure is equivalent to a set of F filter banks with G filters each. The matrix representation of MIMO filters is convenient for implementations. The filter bank representation is convenient for analyses.

Slide 45: MIMO GNN / Multiple Feature GNN

(1) In a story that should by now sound very familiar, we can build a MIMO GNN by stacking MIMO perceptrons. MIMO perceptrons which, in turn, we can build by composing MIMO filters with pointwise nonlinearities.

(2) The procedure is the same we used to build a GNN out of single-input-single-output graph filters. Layer ell of the GNN processes

(3) The output of the previous layer X_ell-minus-one. This is a matrix graph signal with multiple features. At layer ell this output plays the role of an input.

(4) This input is processed with the MIMO perceptron H_ell. This MIMO perceptron is the composition of a MIMO filter with a pointwise nonlinearity. The coefficients of this MIMO filter are matrices H_ell-k.

(5) The result of this processing is the output of the layer, X_ell. This is also a matrix graph signal with multiple features.

(6) Agreeing that the input to layer 1, which is the given signal X, be reinterpreted as the output of the nonexistent layer 0, the equation above provides a recursive definition of a MIMO GNN. We are writing the input to the GNN as a matrix graph signal with multiple features. But it is allowed to have inputs with single features. Indeed, it is often the case.

(7) The output of the MIMO GNN is the result of stopping the recursion after L iterations. Again, we are writing the output as a matrix graph signal with multiple features. But is allowable to have outputs with single features.

(8) This results in a function class Phi that is parametrized by the shift operator S and the set of filters capital H_1 through capital H_L.

(9) We define the filter tensor calligraphic H to write the MIMO GNN with more compact notation.

(10) The filter tensor H is the trainable parameter of the GNN.

Slide 46: MIMO GNN Block Diagram

(1) This was more or less the same description we gave of GNNs and FCNNs. The only change is the use of MIMO graph filters. We see the same in this block diagram of a MIMO GNN with 3 layers. It's the same diagram except that layers have MIMO perceptrons.

(2) (Empty)

(3) We begin by feeding the input graph signal X to layer 1. This could be a signal with multiple features. We denote the number of features as F_0.

(4) This signal is processed with a MIMO graph filter with matrix coefficients H_1-k.

(5) This MIMO graph filter produces an internal output Z_1. This is a signal which can possibly have a different number of features. We will denote this number of features as F_1.

(6) The output the MIMO filter is now sent through a pointwise nonlinearity.

(7) The output of the pointwise nonlinearity is the multiple feature signal x_1.

(8) This completes Layer 1.

(9) The output of Layer 1 is now fed as an input to Layer 2. This is a signal with F_1 features.

(10) In Layer 2, the signal is processed by a MIMO filter with coefficients H_2-k

(11) To produce an internal output Z_2

(12)Which we feed to a pointwise nonlinear function

(13)To produce the signal X_2

(14)This completes the layer and X_2 is declared to be the output of Layer 2. This is a signal with F_2 features.

(15)The output of Layer 2 is not fed as an input to Layer 3.

(16)Where it is processed with a MIMO filer with coefficients H_3-k

(17)To produce internal output Z_3

(18)Which we process with the pointwise nonlinear function sigma

(19)To produce the signal X_3

(20)This completes Layer 3.

(21)The signal X_3, at the output of Layer 3 is declared to be the output of the MIMO GNN.

(22)The output is parametrized by the filter tensor calligraphic H grouping the MIMO filters of all layers. This is the trainable parameter of the MIMO GNN.