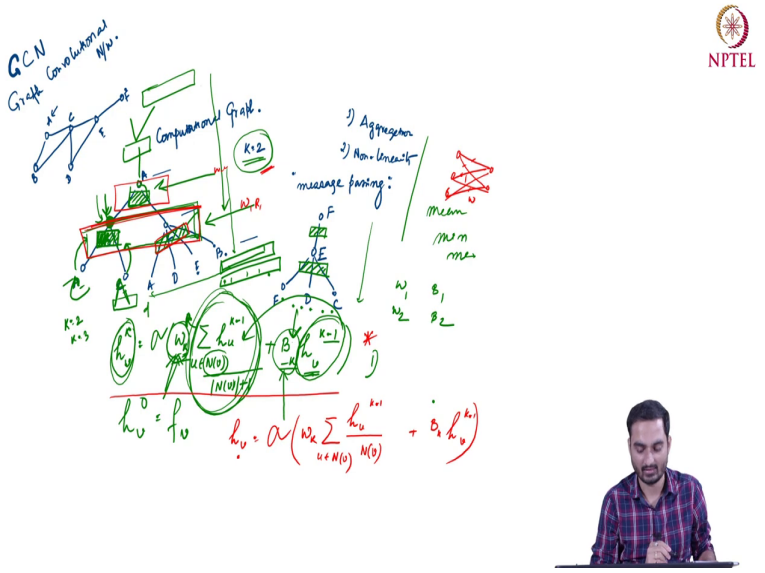
Social Network Analysis
Prof. Tanmoy Chakraborty
Department of Computer Science and Engineering
Indraprastha Institute of Information Technology, Delhi

Chapter - 09
Lecture - 10

Welcome back; so, we have been discussing graph neural network, particularly graph convolutional network GCN.

(Refer Slide Time: 00:30)



And a quick recap what we have discussed in the last lecture is that GCN is a basically a message passing paradigm which for every node it basically generates a computational graph and the computational graph is driven by the topology of the graph. And for every node right we have a separate topological structure we have topology separate computational graph and we specify the depth of that computational graph.

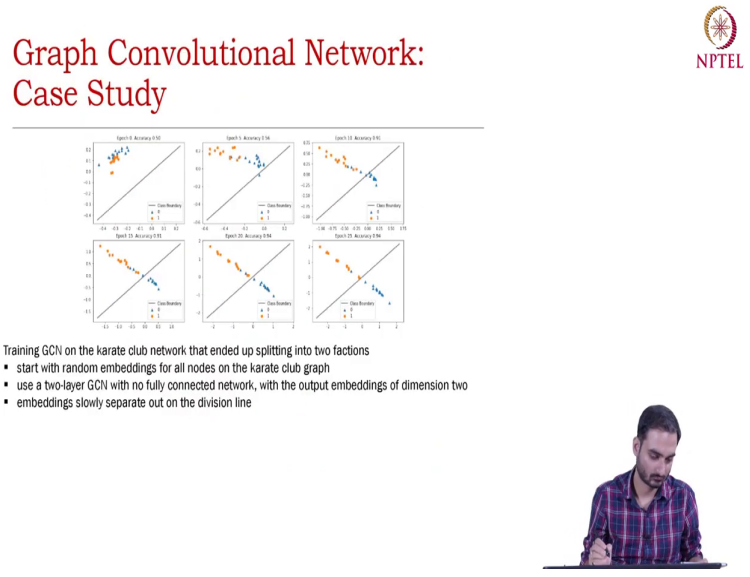
And we also have you know convolution operations at different places right, basically for every depth we have a convolution operation. And in the convolution operations we have two parameters that we train one is W another is B. So, we discussed this equation right, this is the very important equation that hidden state at a particular depth or at a particular time of a node v is essentially is a function of the neighbors right.

We basically take the aggregation of messages from the neighbor's; aggregation if the aggregation is just a mean, it is basically going to be h of u k minus 1 where u is the; u is the neighbor of v and then we normalize it by the number of neighbors right. And apart from that we also have the hidden state of that particular vortex v right, at the previous time stamp h k minus 1 right.

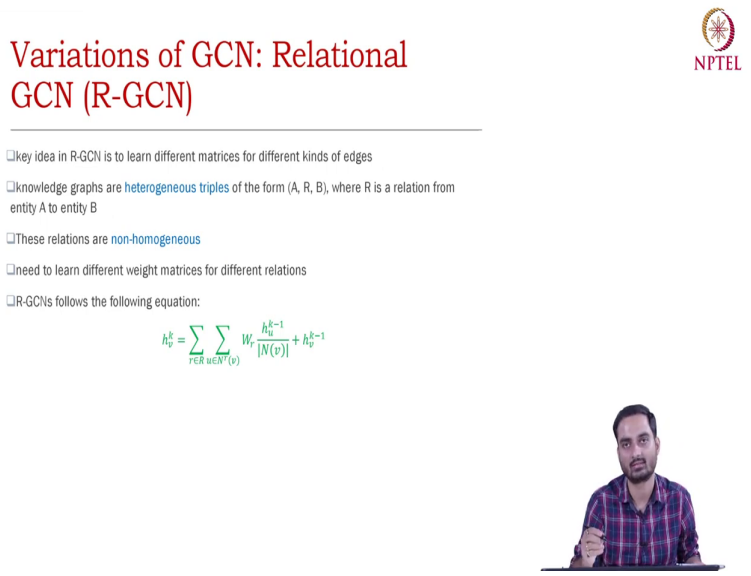
And this is parameterized by W K and this is B K, we wanted to keep these two things separate. And W K and B K are learnable parameters and the entire things will be passed through a non-linearity sigma this is the idea. And so for every depth we have separate W and v, for depth 1 we have W 1 B 1, for depth 2 we have W 2 B 2 right which we learn.

So, these W's are essentially you know you can think of this as neural network weights; for example, say you have this kind of network structure right you can think of; you can think of these are the weights of the connections as W's right. So, you can think of it basically as two different convolution operation happening here at the depth 1, here at depth 2 right.

(Refer Slide Time: 03:07)

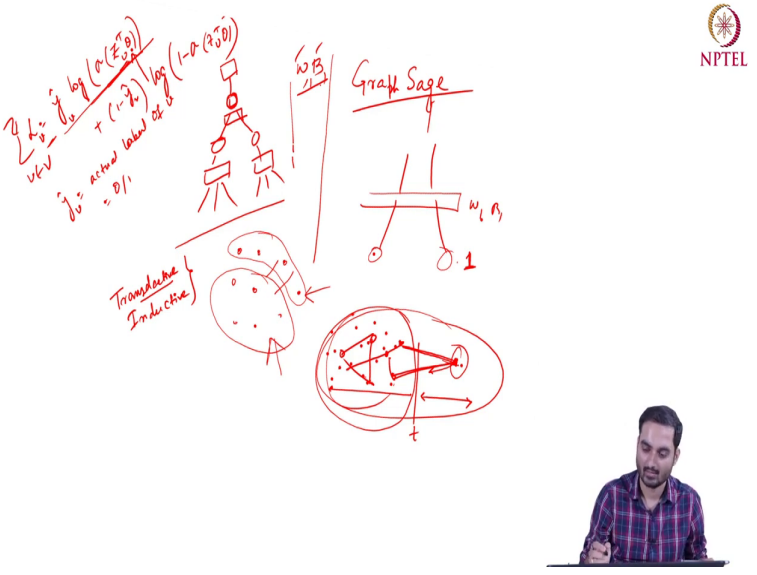


(Refer Slide Time: 03:10)



So, now and we also discussed how we train GCN, we use supervised setting; for example, the task is node classification or say link prediction you basically come up with a loss and Let us say; let us say the loss is a simple cross entropy loss.

(Refer Slide Time: 03:31)



So, what we will say is that say at the final stage right you have say this is a computational graph right ok; let us say like this and say you want to predict whether this node is fraud or genuine. So, you have some classification model here, it can be logistic regression, this can

be a simple neural network multilayer perceptron type. And let us say the loss is a simple cross entropy loss where you say that the loss is essentially write like this.

So, for a particular node v, let us say you are doing it for a particular node v this is y hat v, y hat v is the actual level right actual level of v, it can be say, it can be 0 or 1 right. And you have the output let us say it is a simple let us say it is a simple logistic regression kind of stuff. So, z v is the embedding that you get right for this node at the final stage and this theta is essentially the logistic regression parameters right. So, this is the parameter of the classifier remember this is not the parameter of the embedding method right.

So, and then you pass it through a sigmoid this is basically logistic regression then you have a log loss right. So, this plus 1 minus this right 1 minus that log of 1 minus sigmoid ok. And this is this loss is for a particular node v right. And then you basically sum it up for all the nodes write all the nodes in v and you then take the derivative with respect to each and every component of W and B, because W and B are matrices that you learn that you want to learn.

So, for every element of W and B you take the derivative of this, derivative of this with respect to that element and you back propagate the error right. So, now this is GCN, normal GCN there are multiple variations as I mentioned right, one such variation is called graph sage ok. So, graph sage is essentially a more generalized version of graph convolution network, remember the beauty of the graph convolution network is that it can also be useful for inductive learning right.

I hope you have heard about something called transductive learning and inductive learning ok, these are the very common terminologies in machine learning, these are two learning paradigms. So, in transductive learning, the idea is that let us say right that the test set is known to you ok. Let us say you have a graph right and you and this is the entire graph is known to you right and you are say running deep walk or node two-way kind of approaches on the entire graph right.

And you get some embeddings, and you also know the labels, let us say it is a classification problem let us say you know the labels of some of the nodes in the graph right and your task is to get the label of the other nodes right. So, from the labeling point of view this this is your training set, this is your test set right, but remember the test set is also known to you.

Therefore, when you train your graph learning graph embedding method you gave the inter graph as an input to the embedding method right. So, that deep work node two be like this kind of methods are run on the entire graph right and you got the embeddings right. So, you somehow use the topological structure of the test set as well to get the embedding of the trained set right; so, this is called transductive learning when you know the test set also if the test set is known to you. In case of inductive learning the test set is not known to you.

So, you have to take the decision, you have to take run all your learning methods on the training set, you generate some embeddings whatever generate some representations. And when a new test set comes in you have to also generate the representation of that additional that test node right and you perform classification ok.

Let us say; let us say you have an evolving graph right; so, you know the graph till t right, and you have trained your embedding methods on these nodes right. I mean on the given part of the graph, when a new test set comes in right you do not know because this set this node was not a part of the training set right. So, if you use node to way kind of methods how do you get the embedding of this unknown node, you cannot do that right.

So, this is inductive learning, in the inductive learning the test set is not known to us I mean if you think of a normal machine learning right kind of applications mostly inductive learning right; so, but inductive learning is difficult because you do not know the test set right. Now, think of a graph like this right and let us say your method has already been trained on this graph and a new node comes in this node gets attached to this node and this node right.

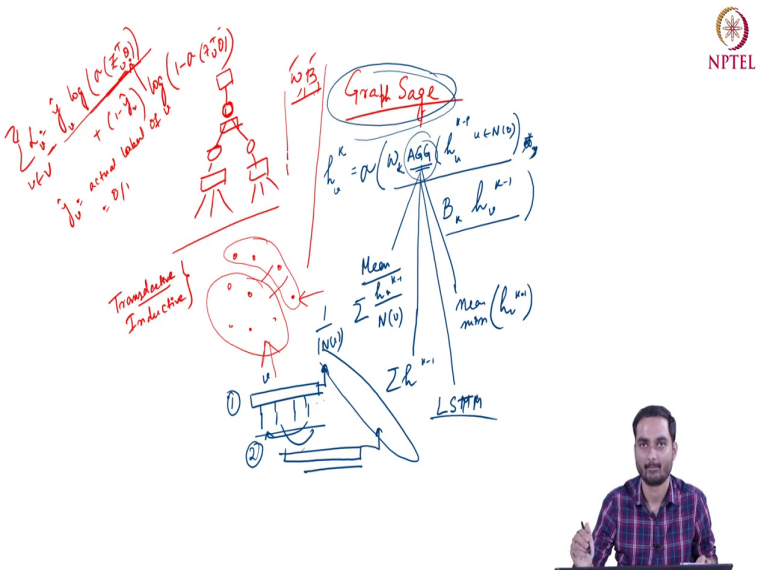
You have to run your method again right, said deep work or node to be on the entire graph and that is troublesome, because think of Facebook, Twitter like networks evolving over time. So, every time if you keep on you know running graph methods graph learning methods from scratch you are gone right. So, this is a terrible approach right, what you can do here? You can use GCN ok, because GCN the two trainable parameters W and B these two parameters are same across all the nodes right; so, on the training set you can easily learn W and B right.

Now, when a test set comes in this node gets attached to these two nodes. Now, for this node you basically get an computational graph for this node you get a computational graph. Now, you know that for the computational graph for label 1 this is W 1 B 1, you already got it W 1 B 1 is same across all the computational graph. So, what is your task now? Your task is just to feed right the representations of the neighbors right.

If the representation not available you just use you know either an identity matrix or identity feature, I mean all ones not identity all one feature vector or if the vector is available feature vector is available you pass it right. First to W 1 W B W 1 and B 1, then you pass it to W 2 B 2 and you get the representation for this node ok; so, GCN is also suitable for inductive learning right.

Now, in graphs sage what they suggested is that you know you just take the same GCN paradigm, but in a different way right. What you do here you now GCN let me just go back remember this formula right; so, in GCN what we do? We make it even more generalized right.

(Refer Slide Time: 12:04)



What we say is that the embedding of a node v at depth K or at time K would be something like this. So, in GCN we used some sort of weighted average and weight was 1 by Nv, weight was same for all the neighbors, aggregated then weighted aggregated and then averaged some sort of weighted average.

So, here we make it generic, we are saying that let us use any aggregation method ok. And what is the parameter, and what are the arguments? The arguments are the neighbor's embeddings of embedding of neighbors of v at time K minus 1 right. Now, this aggregation function can be mean can be sum can be m i n min can be max anything right. And let us make it parameterized W K, and you have the additional part right which is h v K minus 1 B K.

Let us not sum them up we are not taking the sum unlike there in case of GCN where we just took the sum, here we keep them separate. When we sum them the information about the individual components will be gone right, we wanted to keep this thing separate. Therefore, here we concatenate them instead of taking the sum right, this part and this part will be concatenated comma right appended then pass it to sigmoid.

This is graph sage right, although graph sage was developed just to show the inductive power of it right; but of course, GCN can also be used for the inductive learning. And graph sage paper also theoretically show that what could be the possible aggregation operations that one can think off. The first one is just simple a mean right, we have already seen what is it, it is basically this one right.

It can also be just a sum ok; it can also be it can also be a pooling kind of operation it is can be max pool right or min pool, max pool or min pool can also be possible they do also generate the same size vector right. Max pool or mean pool of the neighbor's, neighbor embeddings till time K minus 1 at time K minus 1 right. Even it can also be an LSTM right, it can also be another neural network like an RNN kind of network.

So, a long short-term memory you all know what is LSTM what you do here is that you since let us say for vortex v you have four neighbors right you pass it through LSTM right and the final output would be the output of the LSTM. Remember this aggregation operation should be order invariant right, but you may ask LSTM is order variant right.

Because, if you change the order of the neighbor's, it will change it will produce different output that is true. So, therefore, what you do if you really want to use LSTM here you basically you know what we do you shuffle right your neighbors multiple times. So, this is LSTM configuration 1 you have one sequence of neighbors you get the output. You have LSTM configuration 2 another sequence of neighbors you have with some output and so on and so forth, and then you aggregate this output right.

So, if you shuffle multiple times and feeding and feed them to LSTMs different LSTMs and aggregate them, somehow that order invariance order invariant property you know somehow you know can be preserved right that is the idea you can use LSTM here. So, graph sage turned out to be very useful particularly in a network like Pinterest right, Pinterest is also a network where you share images and they showed an algorithm called pin sage which basically uses graph sage for Pinterest kind of social network right.

So, this is about graph sage there are many things which I intentionally skipped particularly the theoretical part of it, it turned out that sum is better than mean right. Particularly if you look at the aggregation different aggregation operations, there are theoretical results showing that this aggregation is better than that aggregation and why right; for example, sum is better than mean.

So, in the next lecture we will briefly talk about another part which is called graph attention network gat in short. We look at how attention mechanism can be used for graph embedding ok.

Thank you.