Visualize Graph Convolutional Network

By Mridula Gupta

Motivation

In recent times, with the increase in computational power, structured information, and data collection techniques, deep learning methods have become significantly popular among the AI community. One such widely used deep learning model is AlexNet, winner of the 2012 ImageNet Challenge. As image classification error was significant low (around 15.3%) for AlexNet [1] on ImageNet dataset, several studies extended pre-trained AlexNet to solve other critical problems of the domain. Typically, any deep neural network consists of multiple hidden convolutional layers (AlexNet had five convolutional and three fully connected layers), and a large number of parameters (approx. 63 million for AlexNet). Henceforth, even though we can accurately learn the parameters of a deep learning model, it is hard to understand the underlying training process and visualize the large set of parameters at any particular point during the training.

Approach

To make deep learning models intuitive and their training transparent, several AI and visualization experts developed visualization tools and libraries, specifically for deep learning models. For instance, Tensorflow Playground [2] provides a learning environment for the Neural Network where users can choose and modify the parameters (such as learning rate, number of hidden layers, etc.) and visualize the output for all the neurons, impact of change in parameters on weights, and the final output over epochs. In this project, we aim to create one such visualization tool for Graph Convolutional Network (GCN) [3] - a deep learning approach used for semi-supervised classification of graph networks. GCN takes graph network as the input and comprises of multiple conventional layers, as shown in Figure-1a. Often, the model implementation incorporates ReLu for non-linearity, dropout to avoid overfitting, and softmax, in the final stage, computes the probability for the classification of the nodes. The network learns the weights through backpropagation. To the best of our knowledge, there exists no way to visualize the parameters and underlying learning process of Graph Convolutional Networks.

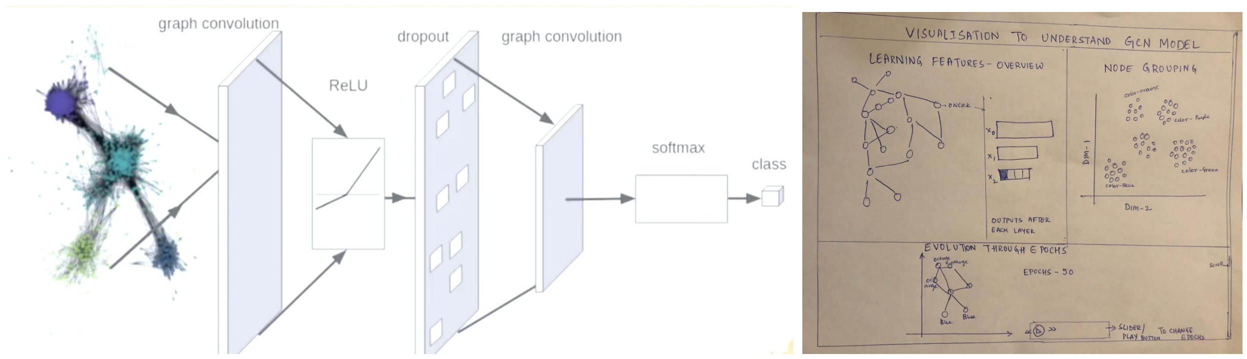
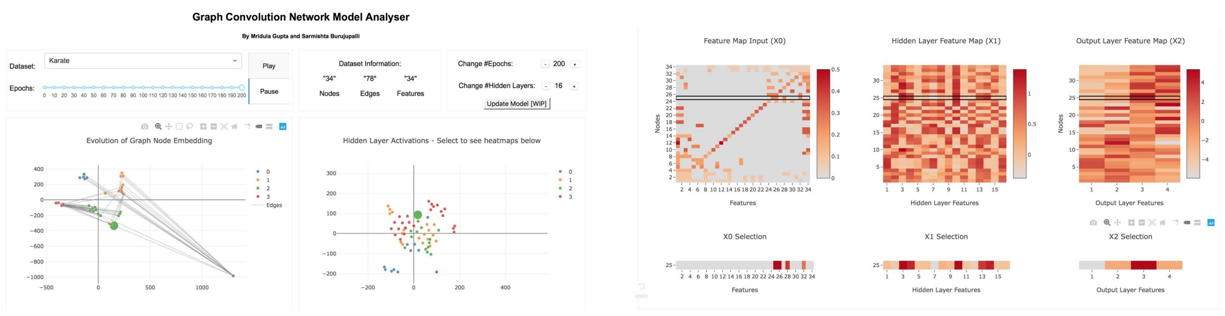


Figure-1: (a) [Left] Graph convolutional networks (GCN) comprises of multiple convolutional layers. We provide a graph network as an input where each node is a data point (depends on the dataset) and edges indicate relation between the nodes [4]. (b) [Right] Paper prototype after initial brainstorming.

In GNC, nodes of the input graph represent the units, and edges indicate the connection (non-directional and unweighted) between any two nodes. In GCN, each node has its feature matrix. In our visualization, we intend to show the classification of nodes across the layers, starting from first convolution layer, until we reach the output layer, and the significance of each feature (input and intermediate layers) as the training progresses over epochs. Figure-1b shows the paper prototype of the dashboard design after initial brainstorming. The top left chart shows the structure of the graph after the final classification. If we select any node in the graph, the feature map (input and intermediate layers) will pop up. On the top right side, we show the tSNE visualization of the node activations after the first convolutional layer. The bottom chart depicts the evolution of the graph network classification with respect to the training epochs. We can control the epochs through a slider.



Graph Convolution Network Model Analyser
By Mridula Gupta and Sarmishta Burujupalli
Dataset: Karate
Epochs:
Play
Pause
Dataset Information:
"34" Nodes
"78" Edges
"34" Features
Change #Epochs: 200
Change #Hidden Layers: 16
Update Model [WIP]
Evolution of Graph Node Embedding
Edges
Hidden Layer Activations - Select to see heatmaps below
Feature Map Input (X0)
Hidden Layer Feature Map (X1)
Output Layer Feature Map (X2)
Nodes
Features
Hidden Layer Features
Output Layer Features
X0 Selection
X1 Selection
X2 Selection

Figure-2: (a) [Left] We realised prototype through Dash, a python-based visualization tool. On the left plot, we depict evolution of graph network and on the right plot, we present tSNE chart. (b) [Right] User can study significant of features for each node, both - over epochs and across layers.

In the final revision, we updated the design and combined the first graph (which represented the final output) with the evolution graph. In addition to that, we added a slider to visualize the training progression of remaining charts (tSNE activations and the feature heatmaps) over epochs and make the dashboard more intuitive. By default, all the charts animate over epochs.

Progress

We implemented our dashboard [4] in Dash by Plotly, which is a Python-based visualization tool, hosted on Heroku. Our dashboard mainly comprises of four sections -

1. Control Panel - In this section, we provided epoch slider controls, the dropdown menu for dataset selection (discussed later), relevant stats about the data, and an interface to update the model inputs - the number of epochs and the number of hidden layers.[1]
2. Evolution of Node Embeddings - Underneath the control panel (on the left side of Figure 2a), in our first visualization - the graph network - we depict the spatial structure of the network along with node classification progression with respect to the epochs.[2]
3. Hidden Layer Activations - The second visualization (on the right side of Figure 2a) classifies all the nodes within the model. Here, it shows the output just after the first hidden layer using tSNE, which reduces the dimensionality of the features in the model. Nodes in the same group are seen coming together in the animation as the number of epochs increases.[3]
4. Feature Maps Across Layers - The last visualization is the set of feature maps present across the model. The first heatmap is the input feature map which is the product of the adjacency matrix (indicates connections within the graph) and the feature map. The second heatmap is the output after the first convolutional layer. Similarly, the next heatmap is the output after the final convolutional layer. Within the visualization, we separately highlighted the selected layer to bring in the clarity.[4]

Dataset

For this tool, we chose Karate [6] and Terrorist [7] Datasets which were very different in size and the type of features. The Karate Dataset consists of around 35 nodes with each node indicating a member of the club, and a connection between any two nodes represents a pair of members who interacted outside the club. For Karate Dataset, the feature matrix is a unit matrix. The input to the GCN model is a feature matrix multiplied with edges adjacency matrix

[1] In current implementation, the update model button is not yet functional.
[2] We used *DGL* library to train the model and *networkx* library to visualize the graph structure.
[3] We have used tSNE approach with scatter plot to create this visualization.
[4] We have used heatmap for this visualization.

with Laplace transformation similar to the paper. To test visualization at a scale, we used Terrorist Dataset, a bigger dataset not used in the paper. It consists of around 13k nodes and 100 features. Each node represents a terrorist attack and connections represent co-located attacks. The feature maps are the attributes of the particular attack. Similar pre-processing steps are done on this dataset as well.

Next Steps

To summarize, we implemented a visualization dashboard for Graph Convolutional Network for two datasets. Through the dashboard, users can visualize the training of the model at various stages, both spatially and temporally. Users can interact with by updating model parameters and also focus on a particular node. As of now, the work is in preliminary stages and has the ability for both improvement and extension. Next, we discuss possible future extensions of this work, in detail.

1. Better Design - We can redesign the dashboard to make it simpler and more intuitive. For instance, we could have static charts and show the training updates through with small multiples. Besides, we could link the feature maps with each other.
2. Real-life Use Case - One of the most important updates is to implement real-life use case. For instance, we plan to implement the dashboard for facebook dataset with each profile as a node, friends as edges, and profile features (like the number of friends, relationship status, etc.) for features. We could use the network to classify the nodes into specific categories, such as political allegiance, product advertisement, among others.
3. Model Update Functionality - We created the dashboard with future updates in mind. So we used Dash for development which can communicate with Python server and handle model updates on the fly. We have already integrated the basic interface for the model updates in the dashboard.
4. User Study - Eventually, we plan to set up a user study to get user feedback and improve the dashboard for a better understandability and more intuitive dashboard interface.

References

[1] https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
[2] https://www.tensorflow.org/guide/summaries_and_tensorboard
[3] https://openreview.net/pdf?id=SJU4ayYgl
[4] https://www.youtube.com/watch?v=UAwrDY_Bcdc&t=2s
[5] https://my-dash-app-gcn3.herokuapp.com
[6] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.68.6623&rep=rep1&type=pdf
[7] https://linqs.soe.ucsc.edu/data