Graph Based Neural Architectures

TEAM 25
Team Linear Aggression

SMAI Project Final Report

Detailed Work Distribution

Introduction

As mentioned in the proposal, our primary goal is to learn about neural architectures used on graph-based data. Text data can be represented as linear graphs. In contrast, images can be described as rectangular graphs. Though ample neural architecture exists that performs well on those graphs, more complicated graph structures require GNNs.

Our project tackles two conceptual problems: first, is there a better way to represent the irregular graph-like structures when it comes to neural architectures, and second, can we create an architecture such that we could "attend" to specific nodes more than the others.

We explore Graph Convolutional Networks (GCNs), a simple graph-based neural architecture with multiple layers. It modifies the node's features based on the adjacency matrix as we go through numerous layers via learn able matrices. Next, we try to answer our second question by building Graph Attention Network (GAT) on top of GCNs to focus on certain edges more, which may or may not help the model. We also visualise the said attention so we can understand the process happening.

Finally, we try to recreate glove embedding for text, to better understand how different structures of languages effect quality of embedding, and how attention interacts with different words given the context.

Task	**Member(s)**
Read the papers on GCN and GAT	All
Explore papers on other graph based neural architectures	Shivansh, Tanishq
Implementation of GCN: Data preprocessing & handling	Shivansh
Implementation of GCN: Model creation & training	Tanishq
Exploring the use of attention in graph models	Shivansh, Tanishq
Implementation of GAT: Model creation & training	Shivansh
Attention Visualizations	Tanishq
Hyperparameter tuning, results and analysis of GCN & GAT on Cora dataset	Shivansh, Tanishq
Task Recognition to interrelate SMAI with Natural Language Processing	Prajneya
Creation of dataset for graph implementation and analysis	Prajneya
Applying & analyzing GCN and GAT performance on the dataset	Prajneya
Experimental Analysis of GCN and GAT performance on original dataset	Mayank
Exploration of model performance with multiple layers	Mayank

Literature Review

Project Proposal Drafting

Initial Implementation of GCN and GAT

Dataset

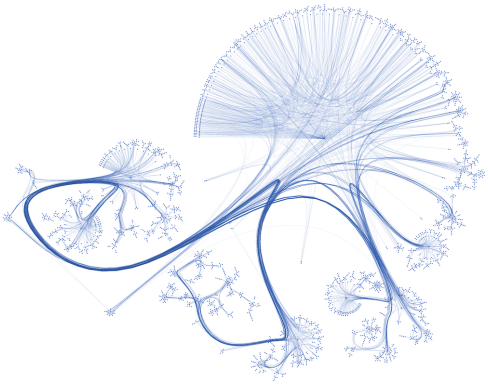
Graph Neural Network may or may not work differently than a traditional neural network based on the dataset. Sometimes you may have multiple samples of various graphs representing the main thing, in which case we can easily do the standard 80-20 train-test split without any loss of information. Such a learning mechanism is known as inductive learning.

Conversely, in some cases, we have a massive graph as the dataset, which we cannot split without loss of information. In such a case, we send the whole data to training (we show the model all the data) but have masks defined. These masks are used so that the backpropagation happens only to specifically selected notes, while other node features, weights and biases are not updated. Such a learning mechanism is called transductive learning.

We have opted to explore transductive learning via the Cora dataset. The Cora dataset represents multiple research publications from various fields as a network of citations. Here, two publications have an edge if one has cited the other (for simplicity, edges are assumed to be non-directional). Each node has a feature vector assigned to it, a binary value based on one hot encoding of words existing in the publication. The words are selected after removing the stop words from all the corpora and ignoring words with less than ten frequencies.

Details about the dataset:

- **2708** Nodes (each node is a research paper)
- **1433** Unique words per document
- **5278** Edges (citation connections)
- **7 Classes:** Neural Networks, Probabilistic Methods, Genetic Algorithms, Theory, Case Based, Reinforcement Learning and Rule Learning
- Class distribution: { 2: 818, 3: 426, 1: 418, 6: 351, 0: 298, 4: 217, 5: 180}

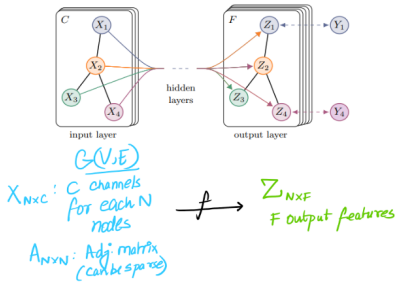


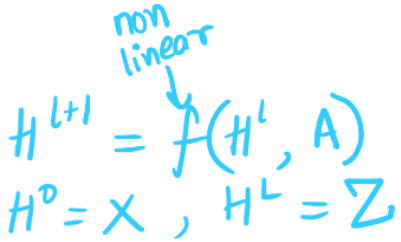
This is a super cool visualisation of the Cora Dataset done using yEd Live and a GraphML file. Here we can see how interconnected the nodes truly are to different nodes.

Understanding Graphs

A normal graph in this case is assumed to be $G(V, E)$ where, G represents the Graph, V represents the vertices (or nodes), and E represents Edges connecting the vertices. Here, the edges are assumed to be undirected for the sake of simplicity.

The adjacency matrix is represented as $A \in R^{N \times N}$, where N is the number of nodes in the graphs. Adjacency matrix is, naturally, symmetrical in nature. To add self loops to the adjacency matrix we perform a small trick $A = A + I$





Issues

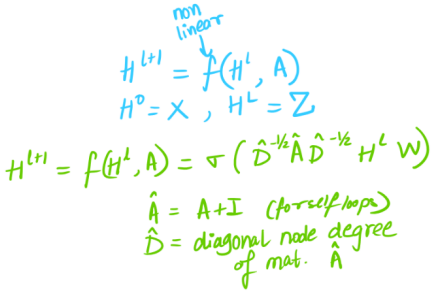
1. A will not consider the node features itself, unless there are self loops

2. A can scale the feature vectors arbitrarily

Resolutions

1. That is the reason we added self loops, $A = A + I$

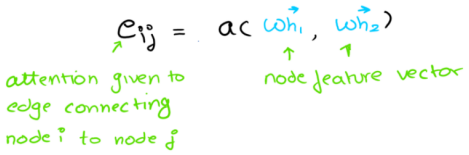
2. We the symmetrically normalise the matrix $A = D^{\frac{-1}{2}} A D^{\frac{-1}{2}}$

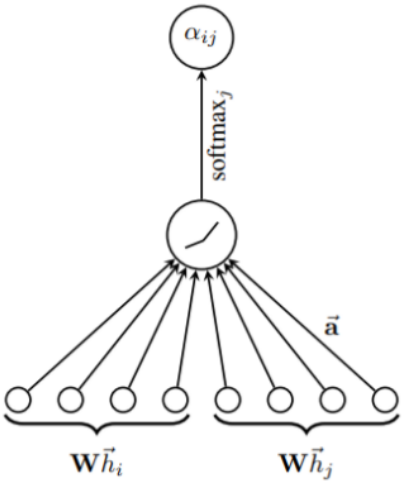


Implementation of GCN is available on: GCN Implementation Colab Link

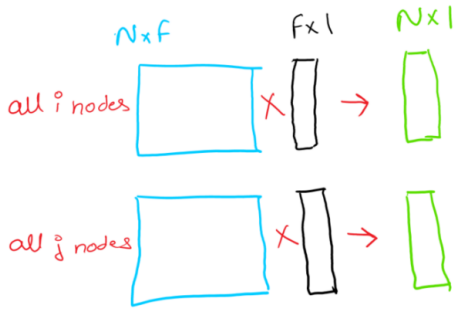
Attention

Attention is essential to what we are trying to do, by concentrating on certain nodes more we are trying to find out if there are some nodes which are more important than the others. Attention is represented by a , and is a single layer feed-forward neural network. It takes in 2 node features as input and gives attention between the two nodes as the output.





One could also assume a to be a learnable weight matrix which needs to be multiplied with the features. Once we get the softmax, we apply LeakyReLU (with default parameters) as the activation function.



This is how we calculate the attention, we multiple all the nodes at once with each other and hence calculate attention for all nodes together. We then sum up the last two vectors as $a + a.T$ and get a $N \times N$ matrix as our attention matrix, which we then multiply with our adjacency matrix A to get a symmetric matrix, since attention should only exist between nodes which are connected.

Implementation of GAT is available on: GAT Implementation Colab Link

Results

Hyperparameters

We tried 2 things with both the datasets:

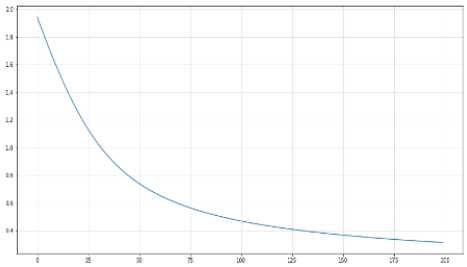
Hyperparameters as mentioned in the paper:

- Training Nodes: 140 (20 from each label)
- Test Nodes: 1000 (randomly selected)
- Epochs: 200
- Layers: 2
- Hidden Features: 32

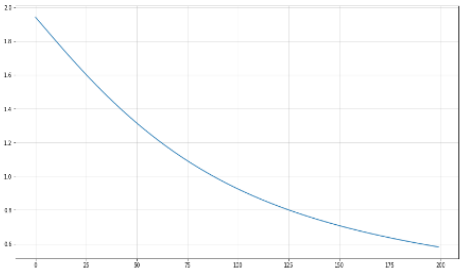
Hyperparameters which gave best results:

- Training Nodes: 1647 (at max 250 from each label)
- Test Nodes: 1061 (remaining)
- Hidden Features: 1024

Loss



GCN



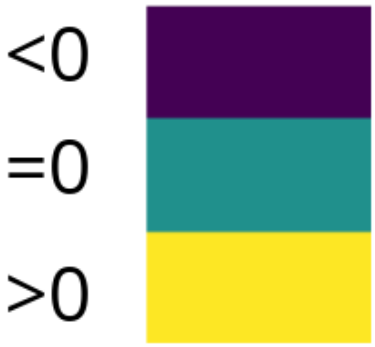
GAT

Interpretations

As we can see, the loss for GCN is lower (and decreases faster) compared to that of GAT for the same number of training data and epochs. There are a couple of plausible explanations for this phenomenon:

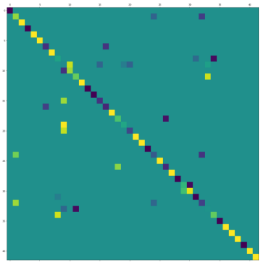
1. We observed with our experiments that as the number of layers of GAT increases the loss decreases as well (we could do an experiment on how the loss changes as we increase layers and compare between GCN and GAT)
2. Another explanation is that since initially a matrix is randomized, probably GAT would perform better as we increase the number of epochs
3. We observe that test accuracy for GCN: 76.5%, and GAT: 82.38%, hence it could also be that GCN overfits hence gives lesser loss.

Understanding Attention

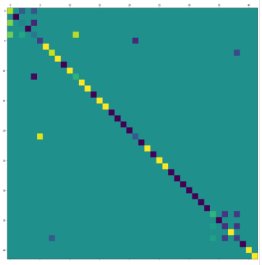


We compare the difference between the first layer of the model and the last layer (first second layer in our case). We do so to see how much the attention matrix changes in a single iteration, later, we plan to do this for more number of layers. Since a 2708x2708 matrix visualisation is not easy for the human eye to understand or make sense of, we take the first 42 nodes for each class, and hence have 7 42x42 matrices. We expect to see high changes on self loops.

Class 0



Class 2



Images of other classes are available in the presentation as well as in the github link (which is private as of now)

In the execution part of attention, we have initially kept the attention matrix a randomized, hence, even in the losses we see that GAT has higher losses initially which then goes lower. Since we are plotting change in the matrices, this does not play a huge role here. Highest change (whether negative or positive) occur in self edges, which means that the nodes most effecting a node's class is most probably itself

As seen in the images of attention plotted for same labels, all attention matrices are symmetric in nature. This can be explained since they are being element-wise multiplied by adjacency matrix, hence we expect to see a symmetrical matrix. Now, one would notice that though these images are indeed symmetrical matrices, but the colours are different. One simple hypothesis explaining this observation is that since we are just observing first-degree connections, some nodes might be more important than others.

Insights

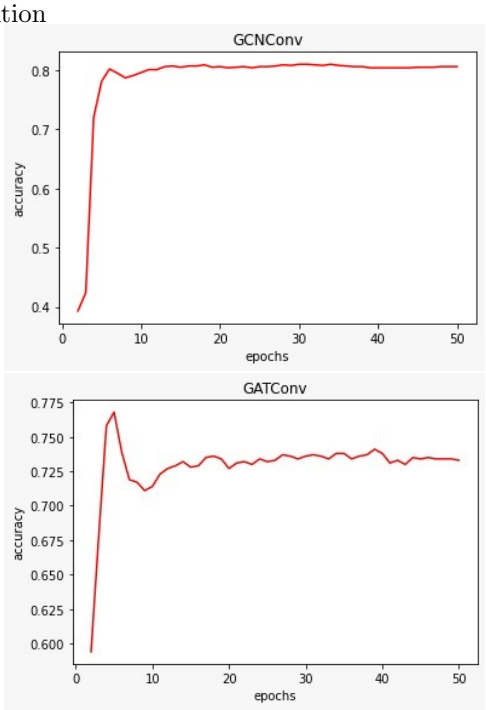
Although we compared the above two models based on their performance over word embedding equations, the results become even more interesting when we look into the attention given over individual words in the GAT approach.

We plotted attention between words over several words and analysed how the model used attention to determine context and learn newer features.

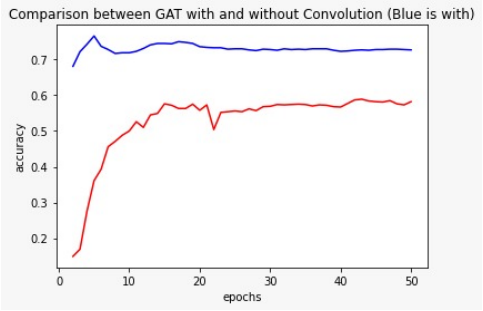
Experimental Analysis

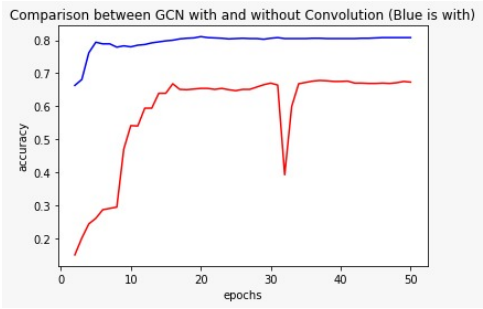
Building upon our previous work, we ran several experiments on a small subset of CORA dataset using the out-of-the-box layers from pytorch.geometric. This meant that our models could we run in a shorter time, and we could model the accuracy over the number of epochs. The experiments were:

- Performance of model with/without Attention

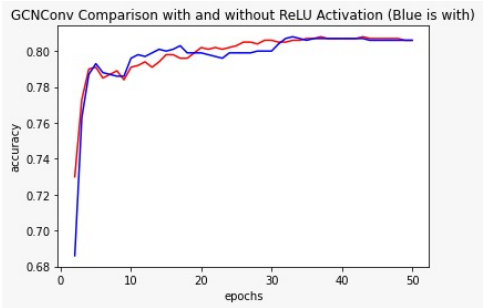


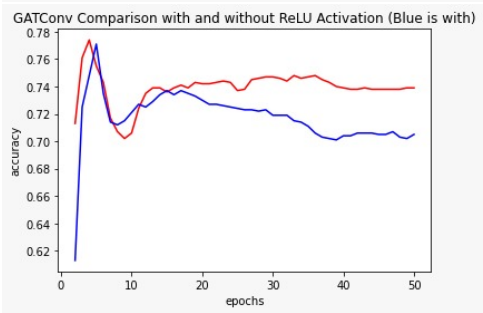
- Performance of Model with/without Convolution



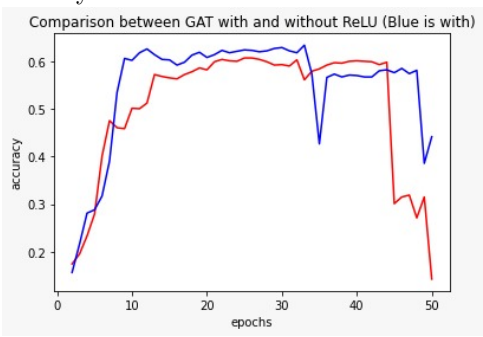


- Convolution layers with and without ReLU





- GAT Layer with and without ReLU



There are several insights that can be made on this.

- We see how the GCN with a convolution layer outperforms the one with attention. This reaffirms an often ignored fact in Deep Learning i.e a more complicated model may not always outperform a simpler model. The model with attention quickly overfits.

- We can also see that the convolution layers outperforms the graph without. The core insight here is that the addition of convolution (which in turn involves the insight of neighbors, and not just the node itself) is a very powerful tool for GNNs. This does not change even when attention is added.

- With the ReLU activation function, we see how a complicated model compares to a simpler one. For example, with convolutions, there is close to no difference in accuracy, but when we add the attention layer with convolution, the accuracy droops. However, without the convolution layer, ReLU outperforms the one without.

The analysis can be found on: Exploratory Analysis Colab Link

Applications

This involved looking at tasks that could be used to evaluate the above implemented models while explicitly visualising its performance over attention-based NLP Tasks. Initial ideas were Text Classification, Relation Extraction etc. We explored how GAT could be applied to such tasks, since there was existing work of GCN over these domains, but no mention of GATs. Afterwards, we realised that even after implementing GAT over such tasks, although we can achieve a proper accuracy metric to compare the two models, no new insight would be collected to analyse attention in these tasks.

We finally came up with a novel idea of "predicting word embeddings", where although the goal of the model was defined to predict word embeddings, the overall goal was to identify semantic relations in a text corpora that is represented by a graph. Our hypothesis was that the inclusion of attention over words connected to one another by syntactic relations will contribute to semantic relations as well. It can be formally stated as:

Predicting word embeddings of word connected in a semantic network constructed via the graphical analog of a textual corpus

Methodology

Instead of just applying the already implemented models on text chunks, our aim was to look at an interdisciplinary cross of graphs and NLP, and analyse how attention can be utilised in graphs to have a new outlook at how context plays an important role in determining word relations.

To achieve this, we first needed a graph-based dataset, and since we are looking at context and attention between words, we need the nodes of this graph to be individual word tokens and the edges such that they depict the connection between words in a sentence (in accordance with the text corpus).

Therefore, before moving to the graph algorithms, the following steps were undertaken:

1. **Graph Construction**: We took the Chapter 1 of *The Hunger Games Part 1 Novel* and used it as our text corpora. We constructed a graph from this corpora where each word was a node and each edge represented the fact that the word occurs after/before the other in a sentence in the corpus.

2. **Feature Matrix Addition**: Each node had a set of words, and to represent its features we used the GLoVe vector embeddings trained on Wikipedia 2015 and Gigaword5 (consisting of 6B tokens and 50 dimensions) and accordingly a feature matrix was constructed.

3. **Node Splitting**: Since our initial end goal is to "predict" word embeddings, 25% of the words in the vocabulary had their features taken from the GLoVE embedding, while the rest 75% were randomly initialised (values uniformly ranging between -1 and 1). The training data had a split of 25% of the total vocabulary while the test was remaining 50% words.

Once the above ideas were established, a dataset needed to be created which represented a textual data as a graph. This graph needed to satisfy the following properties:

1. Nodes must be words

2. Edges should represent syntactic properties, that is, words occurring together in sentences

3. Feature embeddings should represent semantic features of words

We then used our pre-existing implementations of GCN and GAT to have an in-depth analysis of the results we get from each of these algorithms with a linguistic outlook.

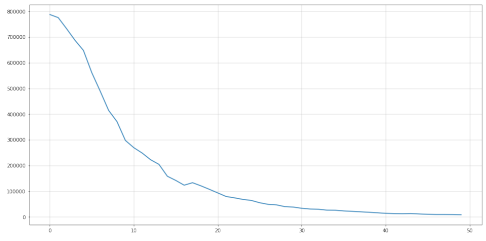
Performance

GCN

Hyperparameters which gave best results:

- Training Nodes: 368
- Test Nodes: 733
- Hidden Features: 1024
- Layers: 2
- Epochs: 500

Loss



Analysis

Since we are dealing with word embeddings of 50 dimensions, comparing accuracies needs to be done in a non-traditional manner. A very famous equation that comes across in daily NLP tasks while discussing about word embeddings that comes up is the addition and subtraction of individual embeddings that make semantic sense. One such example is:

W(FATHER)-W(MOTHER)+W(WOMAN) = W(MAN)

where W(word) represents the word-embedding of the "word".

Utilising this equation, we compare the MSE between our predictions of such an equation across various words:

Following are the MSE Errors for each word:

- MAN: 684.1567
- THINK: 722.6337
- APPLE: 2009.2284
- GUARDING: 2727.8784
- UGLIEST: 2820.1250
- CAT: 2820.7300
- SIXTEEN: 2203.2769

It can be clearly seen that MAN has the least MSE as compared to other words in the list. This does not provide a direct metric to the accuracy of our model as compared to the already existing ones out there, but shows that the model is preserving semantic relations between words when represented as a graph.

If we try to look at how well the model predicted actual word embeddings as comparaed to the GLoVe model, the numbers do not show a good relation, which is totally expected. Although our initial goal for the model was to "predict" word embeddings, it must be noted that the goal of the project was to understand how the graph algorithms implemented in this project can provide an outlook to existing linguistic tasks. This becomes even more evident when we look at how the GAT model performed in the next section.

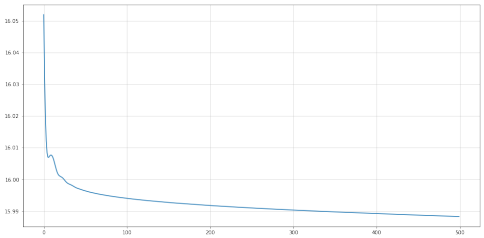
This application of GCN is available on: GCN Application Colab Link

GAT

Hyperparameters which gave best results:

- Training Nodes: 368
- Test Nodes: 733
- Hidden Features: 1024
- Layers: 2
- Epochs: 500

Loss



Analysis

Continuing with a similar take as explained in the GCN section above, we have the following MSE Errors for each word:

- MAN: 0.0013
- THINK: 0.0034
- APPLE: 0.0101
- GUARDING: 0.0033
- UGLIEST: 0.0035
- CAT: 0.0048
- SIXTEEN: 0.0035

Here as well, we can see that the MSE for MAN is the least as compared to the other words in the list, however the MSE has reduced significantly as compared to the GCN Model.

This application of GAT is available on: GAT Application Colab Link

Insights

It is interesting to note the difference in semantic space structures we are observing here. Glove is a frequency based word embedding space, where the co-occurrence frequency determines word representation based on probabilities. Due to the markov rule, only immediate neighbours are considered to be valid (bigrams), On the other hand, Graphs (especially GAT) handles language a bit more elegantly. It does not consider co-occurence as a measure at all, since each node is connected

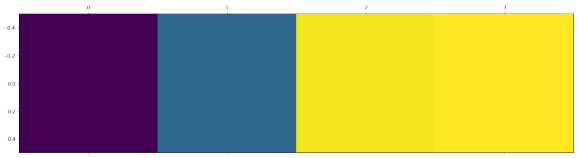
with equal edge weights, and this enables it to not fall prey to bias within small text datasets. Since different nodes are connected to each node, words other than immediate neighbours are also able to influence the representation of main node over multiple iterations which improves the context representations and understanding. Finally, GAT's use of attention really improves our model since there is no inherent favoritism, so all the weights are identified over multiple iterations due to loss calculations only.

Attention Visualisation

Visualising attention over words and their neighbours (in accordance with the text corpus) are as follows (the color coding is the same as described in the *Understanding Attention* section above):

Woman

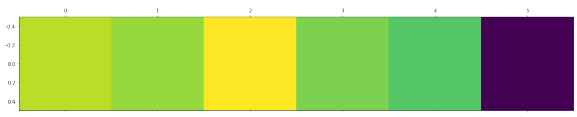
Neighbouring words: [who, the, upbeat, old]



Observation: We see that the model prefers to have a more positive influence of *upbeat* and *old* over **woman** rather than stopwords like *who* and *the*. This is very interesting as the main character in the Novel is a woman who is quite upbeat. While she is not "old", the model prefers this because of the presence of phrases such as "sixteen years old", as well as the fact that the mother of the character is also a woman who is characterised as "old" in the novel.

Think

Neighbouring words: [i, good, would, he, the, squirrel]



Observation: We see that the model puts a negative influence of *squirrel* on **think**. This is because in the actual text,

Just a squirrel - Think the old man

is present. The model realised that squirrel is actually spaced by a hyohen and thus should not be related with the next phrase, even though we did not give this information to the model explicitly.

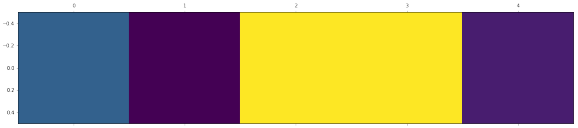
Apples

Neighbouring words: [harvest, but]

Observation: It is very clear that the model gives *harvest* a positive influence over **apples** while a negative influence to a stopword like *but*. This shows that the model understands the concept that apples can be harvested.

Hunger

Neighbouring words: [and, of, happy, game, the]



Observation: It can be seen that the words *happy* and *games* are given a positive influence on **hunger** while *of* has a more negative influence. This highlights the fact that the model recognizes the fact that the usage of the word *hunger* is not actually related to the noun or verb as in "hunger of a man", but a term used as an adjective to describe the "games"

in the novel, thus again capturing context with the usage of context.

These observations above very clearly signify the fact that the inclusion of attention in the model not only allows it to form closer semantic relations between words, but also helps it to include context given any text corpus.

Conclusion

To conclude, we will go through the insights we have gained via this project:
- A proper understanding of graph structures and transductive learning which is used in implementation of the GCN and GAT models
- Attention within the graph structure, how it works and how it favours different classes and nodes.
- Analysis of differences between GAT and GCNs, understanding how GAT works in relation to GCN due to higher number of learnable parameters, initial randomization of attention and different number of layers and epochs.
- Understand semantic space representation of a language, and how graphs understand language differently as compared to nomral frequency based metrics like Glove
- Difference in understanding of context in semantic representation via Attention within Graphs.

References

[1] Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016).

[2] Veličković, Petar, et al. "Graph attention networks." arXiv preprint arXiv:1710.10903 (2017).

[3] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems. 2017.

[4] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global vectors for word representation." Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.

*****