# Word Embedding Fine-Tuning using Graph Neural Networks on Local Word Co-Ocurrence Graphs

Stanford CS224N {Default IID} Project
Mentor: Christopher Wolff

**Federico Reyes Gomez**
Department of Computer Science
Stanford University
frg100@stanford.edu

## Abstract

Graph Neural Networks (GNNs) have recently achieved phenomenal success in many fields of machine learning by leveraging the existing structure of data in many applications, leading us to wonder if we can use them for Natural Language Processing. In this work, we attempt to leverage the structure of word co-occurrences to create a graph that we can use to further fine-tune the pre-trained word embeddings that we feed into our downstream models. We test this hypothesis on the Question Answering task using a Bidirectional Attention Flow (BiDAF) model on the Stanford Question Answering Dataset (SQuAD). Initial experiments show very little improvement using this method, likely due to the fact that the GNNs weren't able to learn good information and the model learns to ignore the embeddings.

## 1 Introduction

Graph Neural Networks (GNNs) and Pretrained Language Models have both reached astounding success in recent years. The use of GNNs in NLP has been previously explored but there are still new approaches to try that can lead to further performance advances in the field. Notably, researchers have built text graphs based on full datasets but little work has been done with building text graphs of individual documents to improve the downstream task.

The goal of this project is to investigate whether structural information of text, represented by word co-ocurrence graphs, can be used by Graph Neural Networks to fine-tune pretrained word embeddings for a specific domain and document. This is an interesting goal given the recent successes of Graph Neural Networks in augmenting existing tasks with new structural information that had previously been hard to capture with more traditional models. Given that our goal is to fine-tune the word embeddings, at worst the fine-tuning model will make no changes to the pre-trained word embeddings, but at best can add a non-trivial amount of knowledge that will significantly improve performance of the downstream model on its task. This work is similar to [1] with a slightly different application, namely to improve word embedding fine-tuning in a way that can be directly fed into the downstream model.

## 2 Related Work

Surprisingly, not much work at all has been done on this problem. The closest work to this project is "Bert-Enhanced Text Graph Neural Network for Classification", which seems like the only relevant work to mention. [1].

The contribution of this paper is the augmentation of a BERT [2] baseline with trained structural GNN word embeddings using a jointly trained co-attention mechanism for document classification.

Graph Neural Networks are Deep Learning models that take in a graph $G = (V, E)$ where each node $v_i \in V$ has an associated initial node embedding (feature vector) $x_i^{(0)}$. A GNN is made up of stacked layers of Graph Convolutional operators where each operator takes each node embedding at layer $k-1$, $\mathbf{x}_i^{(k-1)} \in \mathbb{R}^{F_{k-1}}$ and generates new node embeddings $\mathbf{x}_i^{(k)} \in \mathbb{R}^{F_k}$ with the following operation:

$$\mathbf{x}_i^{(k)} = \gamma^{(k)} \left( \mathbf{x}_i^{(k-1)}, \square_{j \in \mathcal{N}(i)} \, \phi^{(k)} \left( \mathbf{x}_i^{(k-1)}, \mathbf{x}_j^{(k-1)}, \mathbf{e}_{j,i} \right) \right) \text{[3]} \tag{1}$$

where $\square$ is either $sum, mean$, or $max$, $\mathcal{N}(i)$ denotes the set of neighbors of node $i$ in graph $G$, $\mathbf{e}_{j,i}$ represents optional edge features, and $\gamma, \phi$ are any differentiable function such as a Multi-Layer Perceptron [3]).

This paper trains the structural GNN word embeddings using local document-level text graphs using a word co-ocurrence graph and a Gated Graph Sequence Neural Network [4]. This means that each word is treated as a node in a graph and two nodes are connected if the corresponding words co-occur in a fixed window. A GNN initialized with node embeddings corresponding to GloVe [5] word embeddings is used along with the co-ocurrence graph structural information to generate new node embeddings that are combined with the outputs of BERT [2]. Previous approaches have utilized word co-ocurrence graphs at a global level but not at a local level and also concatenation of the two embeddings but not the co-attention mechanism proposed by this paper, which involves a multi-head self-attention block like regular BERT [2] does, except that, given BERT [2] word embeddings and GNN word embeddings, the keys and values of both sets of embeddings are passed as input to each other's multi-headed attention block as shown below:

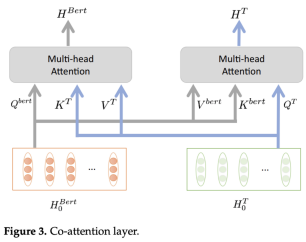

**Figure 3.** Co-attention layer.

Figure 1: Co-Attention

## 3 Approach

The main approach of this work is to explore the use of GNNs, Deep Learning models that take in a graph $G = (V, E)$ where each node $v_i \in V$ has an associated initial node embedding (feature vector) $x_i^{(0)}$. A GNN is made up of stacked layers of Graph Convolutional operators where each operator takes each node embedding at layer $k-1$, $\mathbf{x}_i^{(k-1)} \in \mathbb{R}^{F_{k-1}}$ and generates new node embeddings $\mathbf{x}_i^{(k)} \in \mathbb{R}^{F_k}$ with the following operation:

$$\mathbf{x}_i^{(k)} = \gamma^{(k)} \left( \mathbf{x}_i^{(k-1)}, \square_{j \in \mathcal{N}(i)} \, \phi^{(k)} \left( \mathbf{x}_i^{(k-1)}, \mathbf{x}_j^{(k-1)}, \mathbf{e}_{j,i} \right) \right) \text{[3]} \tag{2}$$

where $\square$ is either $sum, mean$, or $max$, $\mathcal{N}(i)$ denotes the set of neighbors of node $i$ in graph $G$, $\mathbf{e}_{j,i}$ represents optional edge features, and $\gamma, \phi$ are any differentiable function such as a Multi-Layer Perceptron [3]).

The theme of the approaches to this problem is to insert an intermediate GNN layer between the pretrained embedding layer and the downstream model. The graph convolutions are performed on word co-occurrence graphs generated either globally (over all training examples) or locally (over a single batch). The details of the main approaches are as follows:

1. **BiDAF [6]:** This is the starter code model, a Bidirectional Attention Flow model.

2. **Intermediate Local Single-Layer GNN:** This is an extension to the starter code with a single layer of graph convolutions, message passing operations, between the embedding layer and downstream model.
3. **Intermediate Local Multi-Layer GNN:** This approach extends the previous approach by using multiple layers of graph convolutions to create a full GNN model
4. **Intermediate Local Multi-Layer GNN with Skip Connection:** This approach extends approach (2) by adding a skip connection from the word embedding to the output of the GNN embedding to allow the original vectors to flow through if the GNN wasn't useful.
5. **Intermediate Local Multi-Layer GNN with Rescaling:** This approach extends approach (3) by rescaling the output GNN embeddings to be the same norm as the original word vectors to maintain consistency and combining the word embeddings and the GNN embeddings using a Linear layer.
6. **Intermediate Local Multi-Layer GNN with Rescaling and Offset Output:** This approach extends approach (4) by adding the logits of $y_1$ to the logits of $y_2$ before calculating the softmax that gives the final position prediction to weakly condition $y_2$ on $y_1$, hoping this would make it easier for the model to get better scores for F1 and EM.
7. **GNN Encoder:** This approach removes the encoder submodule from the BiDAF model and replaces it with a GNN
8. **Intermediate Context-Query GNN Mixing:** In all previous approaches, we embed the context and query separately. However, similarities between the context and the query may give us information about what words/phrases might be useful for the model. In this approach, we replicate the previous experiments by jointly embedding the context and query using a graph convolutional layer.

All of the approaches described above were coded by me using the PyTorch Geometric (PyG) [3] package.

Code:

`https://drive.google.com/file/d/1exWWhTBWaOvEbuI7n5F_l3l1DR3SXmT3/view?usp=sharing`

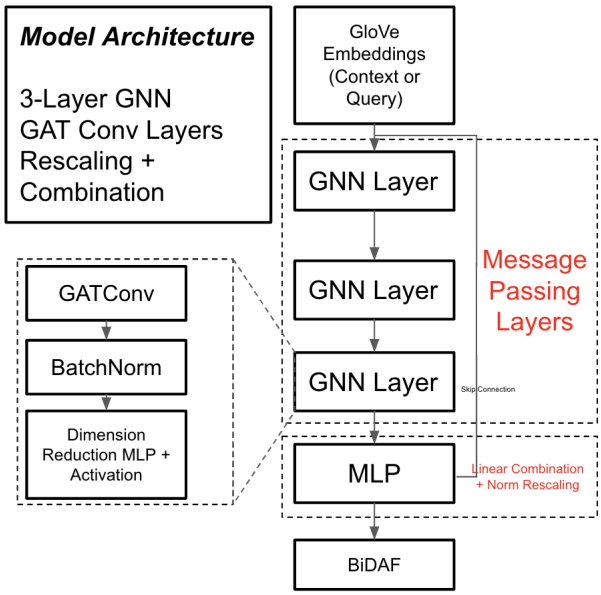


Figure 2: Model Architecture: Intermediate Local Multi-Layer GNN with Rescaling. This process is repeated for both the context and query independently and passed into the BiDAF model as normally

# 4 Experiments

## 4.1 Data

I'm using the default dataset provided with the starter code. Much of the work done was spent preprocessing and caching the word co-ocurrence graphs for each context and each query in the dataset. However, once calculated, they are cached and subsequent runs will use that data.

## 4.2 Evaluation method

We use the evaluation metrics described in the project document: F1, EM, and AvNA (Default IID Project).

## 4.3 Experimental details

We used all the default model hyperparameters and training time parameters for all experiments. For the GNN layer, we used a Graph Attention Network [7] with full counts as edge features since the word co-ocurrence graphs are pretty small. Multi-layer GNNs use 3 layers. The hidden size is the same as the BiDAF [6] model baseline. In order to test the effect of the GNN layer only, we decided to keep most of the parameters similarly to the baseline.

Here is the equation [3] for the GAT Convolutional layer:

$$\mathbf{x}_i' = \alpha_{i,i}\mathbf{\Theta}\mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} \alpha_{i,j}\mathbf{\Theta}\mathbf{x}_j,$$

where the attention coefficients are calculated as follows where $e_i j$ is the word co-ocurrence count betwee word $i$ and $j$:

$$\alpha_{i,j} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top [\mathbf{\Theta}\mathbf{x}_i \,\|\, \mathbf{\Theta}\mathbf{x}_j \,\|\, \mathbf{\Theta}_e \mathbf{e}_{i,j}]\right)\right)}{\sum_{k \in \mathcal{N}(i) \cup \{i\}} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top [\mathbf{\Theta}\mathbf{x}_i \,\|\, \mathbf{\Theta}\mathbf{x}_k \,\|\, \mathbf{\Theta}_e \mathbf{e}_{i,k}]\right)\right)}.$$

## 4.4 Results

| Experiment (Development Set Split) | F1 | EM | AvNA |
|---|---|---|---|
| Lower learning rate ( < 0.5) | too slow | too slow | too slow |
| Higher learning rate (> 0.5) | diverged | diverged | diverged |
| Intermediate Local Single-Layer GNN | 51.47 | 48.97 | 59.57 |
| GNN Encoder (Single Layer) | 52.19 | 52.19 | 52.14 |
| Intermediate Context-Query GNN Mixing | 58.30 | 54.80 | 65.54 |
| Intermediate Local Multi-Layer w Rescaling and Offset Output (batch 1) | 60.29 | 57.25 | 66.58 |
| Intermediate Local Multi-Layer w Rescaling and Offset Output (batch 256) | 60.45 | 57.42 | 66.76 |
| Intermediate Local Multi-Layer w Rescaling and Offset Output (batch 64) | 60.48 | 57.47 | 66.91 |
| Baseline | 60.85 | 57.79 | 67.89 |
| Intermediate Local Multi-Layer w Skip Connection | 60.92 | 57.82 | 67.30 |
| Intermediate Local Multi-Layer w Rescaling | **61.66** | **58.38** | **68.11** |

Our quantitative results were worse than what we expected. With the right architecture, there's a nonzero improvement, but it's less than 1%, which isn't a strong signal that the model works well. However, the simple change of rescaling the GNN outputs had an outsized effect on the effectiveness of the method, so it is possible that more architectures need to be tried in order to find one that works better with the BiDAF model specifically.

| Final Test Set Results | F1 | EM |
|---|---|---|
| Intermediate Local Multi-Layer w Rescaling | **60.891** | **57.058** |

## 5 Analysis

Looking at the results, we can notice that methods that include a skip-connection or Combination Layer that combines the GloVe embeddings with the GNN embeddings do best. This is intentional, since we want to allow the model to not use the GNN embeddings if they're not useful. Our best model uses a Linear layer to combine the embeddings, so we can visualize the weights of that layer to get a sense of what the model is doing:

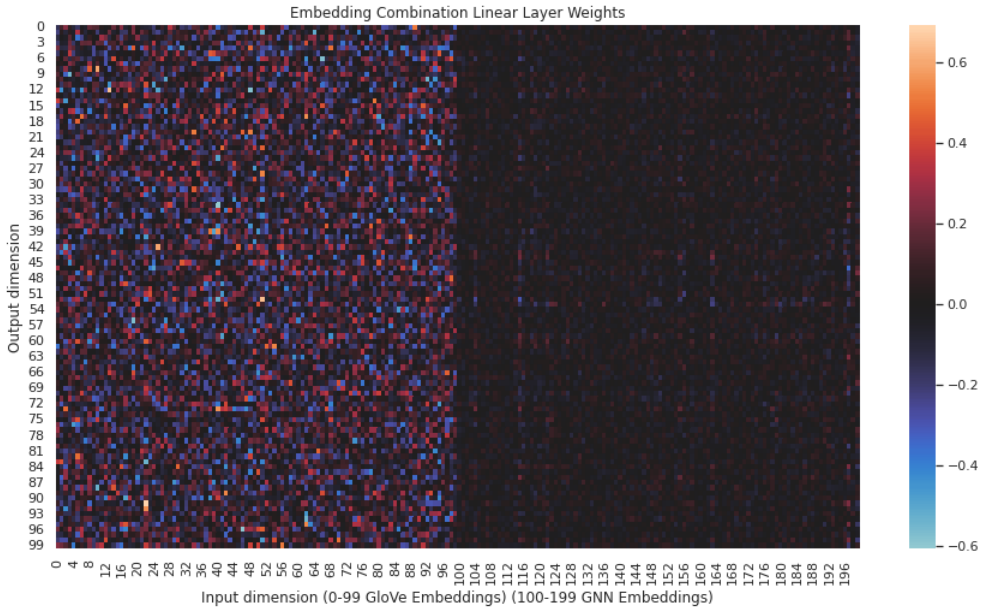


Figure 3: Heatmap of the Combination Layer weights

We see a stark pattern in the weights: the first 100 dimensions of the input correspond to the GloVe embeddings and the last 100 dimensions of the input correspond to the GNN Embeddings. A darker color indicates less of an effect of the input dimension on the output dimension. The first 100 input dimensions are much brighter than the last 100 input dimensions. This indicates that the model is effectively ignoring the GNN Embeddings to pass to the encoder in the model.

### 5.1 Why GNNs didn't work

While it was an interesting idea, our hypothesis that GNNs could leverage the word-coocurrence graph structure seems to not be true. It looks like a GNN, propely tuned and designed, could give some improvements in the single percentage range, but apart from that probably won't be able to have any huge improvment. This could be due to a variety of reasons:

1. **GloVE already contains the same information**. One potential explanation is that GNNs weren't able to add too much to the embeddings with the co-occurrence graph due to the fact that the GloVE vectors already had a much stronger and robust signal about word co-ocurrence information and fine-tuning them just added weak signals and lots of noise. Experimentally we do find that the embeddings are changing, but if this explanation is true, the changes are likely just noise.
2. **We used the wrong GNN**. For most of our experimentation, we used a Graph Attention Network (GAT), due to the fact that (a) the generated graphs were relatively small and (b) we had edge attributes (co-ocurrence counts) to utilize. However, we did not do heavy tuning of

the GNN architecture. We applied standard methods and patterns utilized in state of the art methods like BatchNorm and non-linearities, but tuning GNNs is tricky and small changes in the architecture of the convolutional layer can have a large impact on the results.

3. **This method would have a stronger effect for a downstream model that relies more heavily on word co-ocurrence information like a generative model**. While it's hard to analyze subparts of complex deep learning models like BiDAF we theorize that this method could have a stronger effect on a different task such as text generation. It's possible that training on graphs generated from small documents didn't provide enough information to the BiDAF model to be useful in a difficult question answering task, but training on graphs generated from a single, coherent corpus with a distinguishable style like song lyrics, legal briefings, or instant messaging conversations would have a much more drastic effect on the model performance.

### 5.2 Runtime and Batch Size

Adding a new submodule that needs to either create an adjacency matrix by calculating word co-ocurrences or read that adjacency matrix for a disk, example by example, significantly hurt the runtime of the training by as much as a factor of 10x.

Ideally, we would pass each example through a GNN that only contained the word co-ocurrence graph for that document in order to have the most noiseless signal from the edges. However, since it would take too long to run through each example one by one, we had to use a batch size of $64$. We tried evaluating a trained model with different batch sizes with little to no effect, but we couldn't get to try training a model with a batch size of 1 and a lower learning rate, taking slower steps but closer to the right direction in the loss landscape.

## 6 Conclusion

The goal of this project was to explore the use of GNNs in NLP, specifically as tools to fine-tune pretrained word embeddings with document word co-ocurrence graphs. In our preliminary experiments we found that it is not easy to get a GNN to provide a strong benefit in as difficult a task as SQuAD2.0.

Preprocessing the dataset to generate the correct co-ocurrence graphs took a while, limiting our results, but we experimented with various architectures of different depths, batch sizes, post-processing, and connectivity. We found that a couple of model choices ended up performing better than the baseline, but each by less than a single percent, indicating weak performance gains. It is possible that further tuning of the GNN architecture including pre and post-processing as well as the specific Graph Convolutional Layer will lead to better performance, but this sort of approach does seem fairly limited.

That being said, it would be odd if a method that performs fine-tuning on already-robust word embeddings gives more than a couple percentage points of gain so we can consider this project a mild success. With more time and resources to run extensive experiments with different architecture choices, we may be able to find an architecture that yields consistent performance gains across a variety of tasks and datasets.

## References


[1] Yiping Yang and Xiaohui Cui. Bert-enhanced text graph neural network for classification. *Entropy*, 23, 2021.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[3] PyTorch Geometric creating message passing networks. `https://pytorch-geometric.readthedocs.io/en/latest/notes/create_gnn.html`. Accessed: 2022-02-07.

[4] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks, 2017.

[5] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[6] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016.

[7] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.