THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LEVERAGING GRAPH NEURAL NETWORKS FOR EFFICIENT WORD REPRESENTATIONS

RYAN E. HIMES
Spring 2024

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Computer Science
with honors in Computer Science

Reviewed and approved* by the following:

Dr. Truong Tran
Professor of Computer Science
Assistant Professor
Thesis Supervisor

Dr. Ting He
Professor of Electrical Engineering and Computer Science
Associate Professor
Honors Adviser

*Signatures are on file in the Schreyer Honors College.

Abstract

Sentence structure can consist of a complex structure of tokens and relationships between tokens which can be hard to represent with only a sequential representation. To capture more information about sentence structure we propose representing a sentence as a graph of tokens and relationships between tokens to learn dynamic word embeddings. The graph structure is used as input for a graph neural network (GNN) to learn syntactic and semantic information about the sentence by learning a next word prediction task to learn the embeddings. We also experiment with using the graph structure as input for different natural language classification tasks. Results show that using the GNN ARMAConv on natural language classification tasks can increase accuracy over a sequential representation. Also, using the newly trained dynamic word embeddings achieves a higher validation accuracy compared to all other word embeddings tested on the tweet disaster classification data set.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to start by thanking Professor Tran for his continued support throughout writing this thesis and introducing me to research. Learning prior skills has helped me through the process of this project.

Next, Professor Grosh and Professor AB Shafaye for helping me learn which specific sub-fields of Computer Science interested me. Their classes and teaching styles inspired me and I don't think I would enjoy Computer Science as much without their efforts.

Also, Professor Ting He for her continued help throughout my time at Penn State University Park and for her time teaching communication networks.

Finally, my family for helping me throughout my time at Penn State. I would not have been able to get anywhere in life without their help.

Chapter 1

Introduction

Next word prediction is an important task in natural language processing (NLP), not only for natural language generation but also to convert words into compact numerical vectors called word embeddings. Words that have a similar meaning should be represented with similar vectors, usually compared using cosine similarity. With this characteristic, word embeddings are able to increase accuracy on downstream tasks such as text classification when used as input to different machine learning algorithms compared to using words directly mapped to a single number as input.

Previous research on how to generate word embeddings has mostly focused on representing a sentence as a linear sequence of words. While this representation can capture a large portion of a sentence's meaning, it fails to capture dependencies between words. To fix this issue, a dependency graph can better represent a sentence revealing dependencies between words which can be represented as a graph and give more insight on the semantic structure of a sentence. [1] This representation leads to the conclusion of using a graph neural network (GNN) on the sentence for different NLP tasks including next word prediction.

Applying GNNs on NLP tasks has been proposed by Wu et. al. [2] in their survey. One of the main takeaways from the survey is that there are many gaps in the proposed methods on applying GNNs to NLP tasks and the methods that were proposed have not been explored in much depth. Testing if tools made for different tasks can be applied to NLP has been beneficial in the past, such as applying convolutional neural networks (CNNs), which may be the same case with GNNs if enough research is put into the effort.

Instead of starting from scratch on GNN applications, model architecture that has been used for sequential data can be modified to include a GNN so graphs can be used as input. In specific, the ELMo architecture [3], is modified to include a GNN which takes as input a sentence represented as a graph to predict the next word.

This paper contributes the following:

- Test a Variation of the Dependency Parse Based Graph Creation Process Proposed by Xu et. al. [4]
- Propose using Bi-LSTMs Combined with GNNs for NLP Tasks
- Propose Two Different Methods of Generating Dynamic Word Embeddings using the GNN ARMAConv
- Compare the Proposed Embeddings with Previous Embeddings on Downstream Classification Tasks
- Compare the Produced Dynamic Embeddings using Cosine Similarity in Different Contexts

Chapter 2

Literature Review

A one hot encoding representation of a word contains a vector of zeros that has a dimension size of how many unique words appear in the data set. A one is placed in a unique position for each word the encoding is representing. In other words each dimension in the vector space is reserved for every unique word in the data set. These representations are not sustainable in space because of the number of unique words. One hot encodings also fail to learn semantic meaning of words since every vector points along the axis of the given dimension.

The first efficient static representation of words was created by Mikolov et al. in 2013. Their embeddings solved the two problems that faced one hot encodings by capturing semantic meaning and compressing a word's vector representation into a smaller number of dimensions. To capture semantic meaning, the authors create a context set for each word containing words that are a certain distance away from the target word in different sentences. The context is used as input for a feed forward network to predict the target word which is called continuous bag of words. Another method of producing embeddings introduced in the paper is the skip-gram model which takes the target word as input and predicts the set of context words. The embeddings are produced by the output of the last hidden layer before the softmax layer of the feed forward network. Formally the skip-gram's objective function with a sequence of words $w_1, w_2, w_3, ..., w_T$ is to maximize the function where c is the linear context window size:

$$\max \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t) \tag{2.1}$$

The softmax probability can be represented as:

$$p(w_O | w_I) = \frac{\exp(v'_{w_O} \top v_{w_I})}{\sum_{w=1}^{W} \exp(v'_w \top v_{w_I})} \tag{2.2}$$

Where v'_w is the output/context word's vector representation, v_w is the input/target word, and W is the number of words in the vocabulary. This probability function ensures that a word will have a similar vector representation to the vector representation of words in its context and not a

similar representation to words that do not appear in its context.

By following this method the authors were able to create a single embedding for most words in the English dictionary. Using word embeddings for downstream tasks has achieved better results for NLP tasks compared to one hot encoding representations as well as consuming less space. Word2Vec representations still has two downsides, the representations are static while meanings of different words are dynamic depending on their semantic context and words that appear in the context once have the same weight as words that appear in the context multiple times. [5]

The authors of GloVe fixed the second problem of Word2Vec by keeping a counter for every word that appears in a target word's context instead of only keeping a set of context words. GloVe embeddings outperformed Word2Vec embeddings on downstream tasks. These word representations still share the same problem as Word2Vec, not having dynamic representation of words for different semantic contexts. [6]

The context2Vec model fixed the problem of static word embeddings by creating dynamic embeddings that depend on the current sentence a word is used in. Context2Vec uses a Bi-LSTM to create a context using the whole sentence opposed to a smaller linear context window. The last hidden fully connected layer is used for the embedding. Compared to static embedding methods, this method needs the entire sentence so words that have different meanings in different sentences now have different embeddings. Words like *bank* can have multiple embeddings depending on the specific context. [7]

The ELMo model takes inspiration from context2Vec, but instead of using one Bi-LSTM the model uses two and gets the embeddings from the values that both Bi-LSTMs produce along with the values before the first Bi-LSTM instead of only the last hidden layer. The authors observed that the first Bi-LSTM was able to learn syntactic information while the second was able to learn

semantic information, which are both important in modeling natural language. [3]

Levy et al. rethought of the way to create context for a target word opposed to using a linear window used in Word2Vec, by using a dependency parse of the sentence. Words connected by a dependency edge are used as context words. This method produced accurate word representations but different word representations compared to Word2Vec. With the target word *Hogwarts*, the most similar words using Word2Vec were other Harry Potter characters and locations while the dependency embeddings found different location words more similar. [8]

GNNs are a generalized form of CNNs applied to graph structured data using message passing. Message passing allows GNNs to pass data from each node to its neighboring nodes to gain surrounding information. The first created GNN was the graph convolutional network (GCN) which passes its data inversely proportional to the node's degree. [9] The equation for message passing by a GCN is shown in equation 2.3.

$$X' = \frac{1}{\sqrt{\hat{D}}}(A + I)\frac{1}{\sqrt{\hat{D}}}XW + b \tag{2.3}$$

Where X' is the node's new value matrix, $\hat{D}$ is the degree matrix, A is the adjacency matrix, I is the identity matrix, X is the node's current value matrix, W is a trainable weight matrix, and b is a trainable bias vector. [10]

Wu et al. discussed a high level overview of current research and ideas of GNNs applied to NLP such as different approaches to the graph creation process. The survey shows that there are many topics that have not been tested or have not been tested in much depth. [2]

Filippo Bianchi et al. proposed a new type of GNN using the auto-regressive moving average (ARMA) technique called ARMAConv. ARMAConv is able to train quickly, learn long distance

dependencies, and is more robust to noise. ARMAConv achieved better results than all previous GNN methods on all tasks tested except for one. [11]

Chapter 3

Methodology

3.1 Data Set

To train the word embeddings using next word prediction, the Wikipedia Sentences data set is used. [12] The data set contains about 7.8 million sentences from Wikipedia. Duplicates, sentences with less than 3 characters, and sentences with more than 255 characters are removed. The sentences are then sorted in alphabetical order.

3.1.1 Text Cleaning

Cleaning the text can remove unnecessary information that can add noise from the sentence. The process for cleaning text follows the following pattern.

Expand Contractions: Expanding contractions reduces the number of vocabulary words the model has to learn while keeping the same sentence structure and meaning, allowing it to learn relationships between words easier. For example the word *can't* will be expanded to *can not*.

Remove Numbers: In natural language most of the time numbers create unnecessary noise within the text. Since there are an infinite amount of numbers for the model to learn, numbers provide very little to no information about surrounding words. So they are removed to reduce the vocabulary size and reduce noise.

Remove Punctuation: Punctuation also contributes unneeded noise to the sentence so it is removed. Punctuation, like a comma, provides very little to sentence structure while creating more unnecessary vocabulary. An example of this is *they,* and *they*. The extra comma at the end of *they* creates an additional vocabulary word for the model to learn even though the two words have the same meaning.

Lowercase Sentence: Converting the sentence to lowercase achieves the same goal as removing punctuation, to reduce the number of vocabulary words that have the same meaning. For example, if a word starts a sentence such as *They* it has the same meaning as *they*, but the model sees these as two different vocabulary words.

Remove Extra Spaces: Some of the previous steps may result in extra spaces being left in the

sentence which are removed to keep the data uniform.

3.1.2 Data Set Vocabulary

Since the data set is used for a next word prediction the number of words to predict can grow large with words that can be hard to learn since they do not appear often. To fix this, words that appear less than ten times in the data set are not included in the final set of words to predict. Every word in the set of words to predict are then mapped to a unique number. With this process the remaining vocabulary contains a total of $189,010$ unique words to predict.

3.1.3 Creating Next Word Prediction Data Set

Each sentence can be used as multiple data points to predict the next word. For every word in the sentence, if it appears in the reduced vocabulary, the words that appear before in the sentence are added to the sentence category in the data set while the number assigned to the current word is added to the target category. This does not occur if the current word is the first word since there is no context before it. The objective of the data set is, given the sentence category predict the target category. Using this method, a total of $137,872,139$ sentences with target words are created.

3.2 Creating the Graph

The graph structure uses vertices to represent word embeddings and edges to represent relationships between the words. The structure used is similar to the dependency parse feature structure proposed by Xu et al. [4]. To create their graph, they first run a dependency parse on the sentence connecting words that modify each other with a directed edge pointing to the modified word. After running a dependency parse, words that appear next to each other are connected by an undirected edge. This captures both the dependency between words as well as which words appear next to each other.

This process to convert a sequence of words $T = w_1, w_2, ..., w_n$ to a graph, $G(T)$, can be

represented as:

$$G(T) = (V, E)$$

$$V = \{v_{w_i} | i \leq n \wedge i \in \mathbb{N}\}$$

$$E = \{(u, k) | u, k \leq n \wedge u, k \in \mathbb{N}\} \cup D(T)$$

Where v_{w_i} is the embedding representation of the word w_i and $D(T)$ is the set of edges returned from the dependency parse.

The Spektral library requires vertex values and an adjacency matrix. Our graph construction follows the same process, but all edges are undirected. This is done to allow message passing in both directions between vertices while being processed by the GNN. The value stored in each vertex is the embedding for the given word, so while training the dynamic word embeddings Word2Vec embeddings are used. During text classification different embeddings are used for vertex values. A GCN-filter implemented in the Spektral library is then applied on all of the created graphs.[10][9] Figure 3.1 shows a graph representation of multiple different sentences.

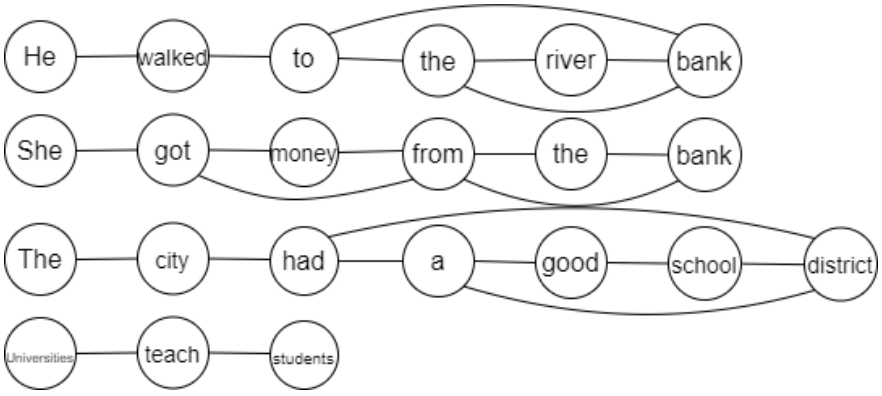

Figure 3.1: Different Sentences Constructed as a Graph Using the Method in Section 3.2

3.3 Generating Word Embeddings

This section presents three different architectures that were trained and how the embeddings are retrieved from the models. One architecture is used as a baseline, not containing a GNN, while

the other two use a GNN. This is done so the baseline and experimental architectures are trained on the same training data. All models are implemented using the Keras [13] and Spektral [10] libraries.

Since the embeddings are trained using a graph structure, the objective and probability functions change compared to the equations used in Word2Vec.[14] Given a sequence of words $T = w_1, w_2, ..., w_{i-1}$ the objective function's goal is to maximize the probability of the next word, w_i, given the constructed graph $G(T) = (V, E)$ described in Section 3.2. Formally this can be represented in equation 3.1.

$$\max \frac{1}{n} \sum_{i=2}^{n} \log p(w_i | G(T)) \tag{3.1}$$

To maximize the probability of the next word, w_i, given the graph $G(T)$, the vector representation of the next word needs to be similar to the vectors in the graph while being different from the vector representation of all other words. Formally this can be represented in equation 3.2 where W is the set of all words in the vocabulary and v_w represents the vector value of the word w.

$$p(w_i | G(T)) = \frac{\sum_{j \in V} \exp(v_{w_i} \top j)}{\sum_{k \in W} \exp(v_k \top v_{w_i})} \tag{3.2}$$

Each architecture follows a similar process on how they are trained and how the final word embeddings are obtained. First the input sentence is converted into static embeddings using Word2Vec [5]. If the architecture uses a graph neural network the embeddings are turned into a graph using the method in Section 3.2. The objective of each architecture is to predict the next word given the previous words as input. After the training process, the embeddings of a sentence are obtained from the outputs of different layers within the network which are different for each architecture to create the dynamic embeddings using a sentence as input. Each process produces a list of embeddings which each correspond with the given word of the input sentence by index. For each dynamic embedding the Word2Vec static embedding is concatenated to the beginning. This gives the embedding the advantage of Word2Vec being trained on a large amount of data while also

making the entire embedding dynamic. This also increases the amount of vocabulary words that are covered since two different sets of vocabulary are used, one from Wikipedia and the other from Word2Vec.

3.3.1 ELMo-Like Baseline Embeddings

The baseline used is based off of the structure of the ELMo architecture. The input feeds into the first Bi-LSTM. The output is then passed through a second Bi-LSTM into a final fully connected layer with a softmax activation function for prediction. The second Bi-LSTM only outputs a single vector, the last vector produced by the forward and backward LSTM concatenated. Every layer uses a $tanh$ activation function except the last layer which uses a softmax activation function. The structure is shown as the first diagram in Figure 3.2.

To obtain the embeddings of a sentence the output of the two Bi-LSTMs are added together, then a $tanh$ function is applied. When predicting the embeddings, all the vectors produced by the second Bi-LSTM are used not just the last. This structure is shown as the first diagram in Figure 3.3.

3.3.2 ARMAConv Embeddings

This structure contains a single ARMAConv layer to test if GNNs are capable of training embeddings on their own. A graph is created with the input sentence and Word2Vec static embeddings. The created graph is used by the ARMAConv and global attention sum pool layers [10] into a fully connected layer with a softmax activation function to predict the next word. Using a GNN like ARMAConv allows connected vertices to message pass to each other which allows a context to be learned making the embeddings dynamic. The structure is shown as the second diagram in Figure 3.2.

To obtain the embeddings produced by the input sentence, the vertex values from the graph outputted by ARMAConv are used as input to a $tanh$ function. The pooling layer is not used since it can remove useful information from the embedding. This structure is shown as the second

diagram in Figure 3.3.

3.3.3 ARMAConv+ELMo Embeddings

This model follows the same process of the ELMo-Like baseline but with the added ARMAConv layer before the two Bi-LSTMs. Like previously, the second Bi-LSTM only outputs a single vector. The structure is shown as the last diagram in Figure 3.2.

The obtain embeddings from the model the output of the ARMAConv layer and the two Bi-LSTMs are added together and then a $tanh$ function is applied. Like the previous ELMo-Like Baseline structure while producing embeddings, all the vectors produced by the second Bi-LSTM are used. This structure is shown as the last diagram in Figure 3.3.

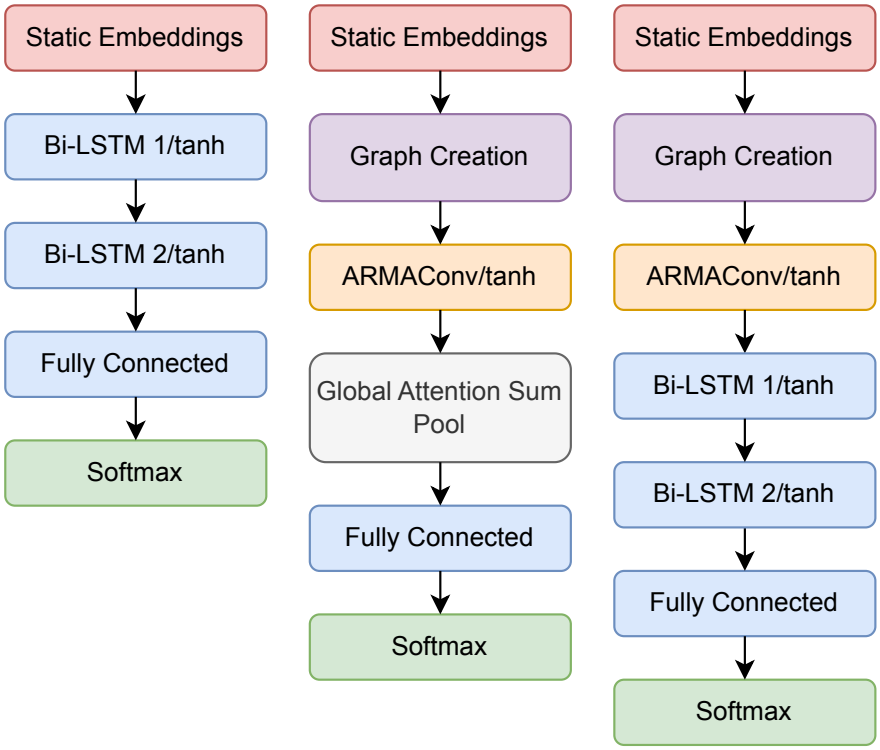

Figure 3.2: Structures of ELMo-Like Baseline, ARMAConv, and ARMAConv+ELMo Used While Training Next Word Prediction

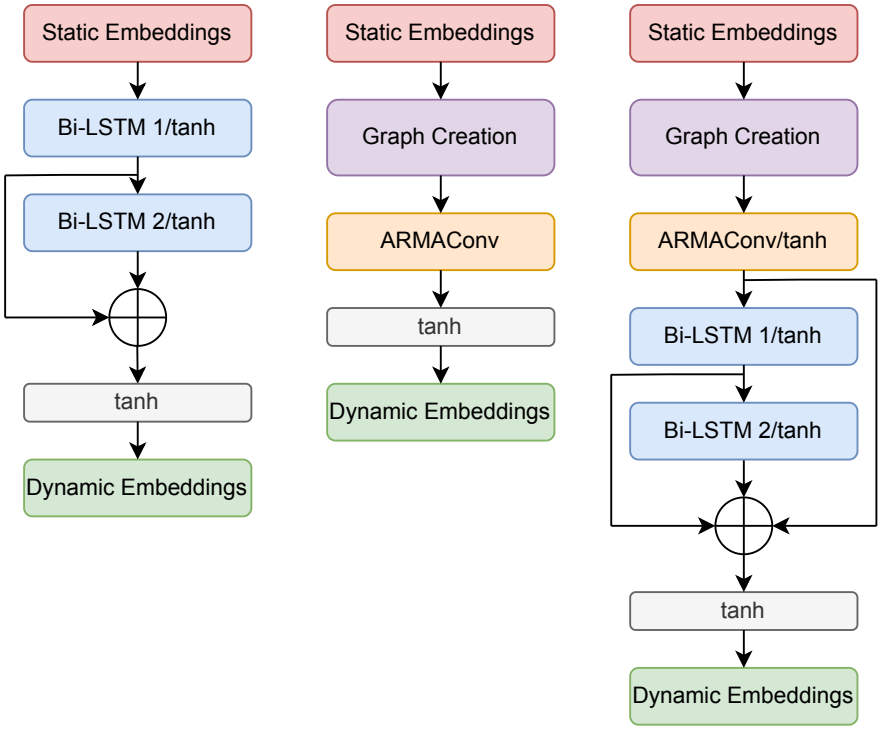

Figure 3.3: Structures to Obtain the Dynamic Embeddings from ELMo-Like Baseline, ARMAConv, and ARMAConv+ELMo Using Layers Trained in Figure 3.2

Chapter 4

Results

4.1 Supervised Classification

To test the trained embeddings, multiple text classification data sets are used. For each data set different embeddings are used with different neural network structures. The embeddings used are Word2Vec, GloVe, ELMo-Like baseline, ARMAConv, and ARMAConv+ELMo. The structures used are three CNNs, two Bi-LSTMs, three CNNs followed by two Bi-LSTMs, an ARMAConv layer followed by a global attention sum pool layer, and an ARMAConv layer followed by two Bi-LSTMs. The goal of using ARMAConv with Bi-LSTMs is to learn sequential representation from the Bi-LSTMs while also having the advantage of the dependency graph structure from ARMAConv. Every layer uses a $tanh$ activation function and the structures that use a GNN use the graph structure proposed in Section 3.2. The different structures are shown in Figure 4.1.

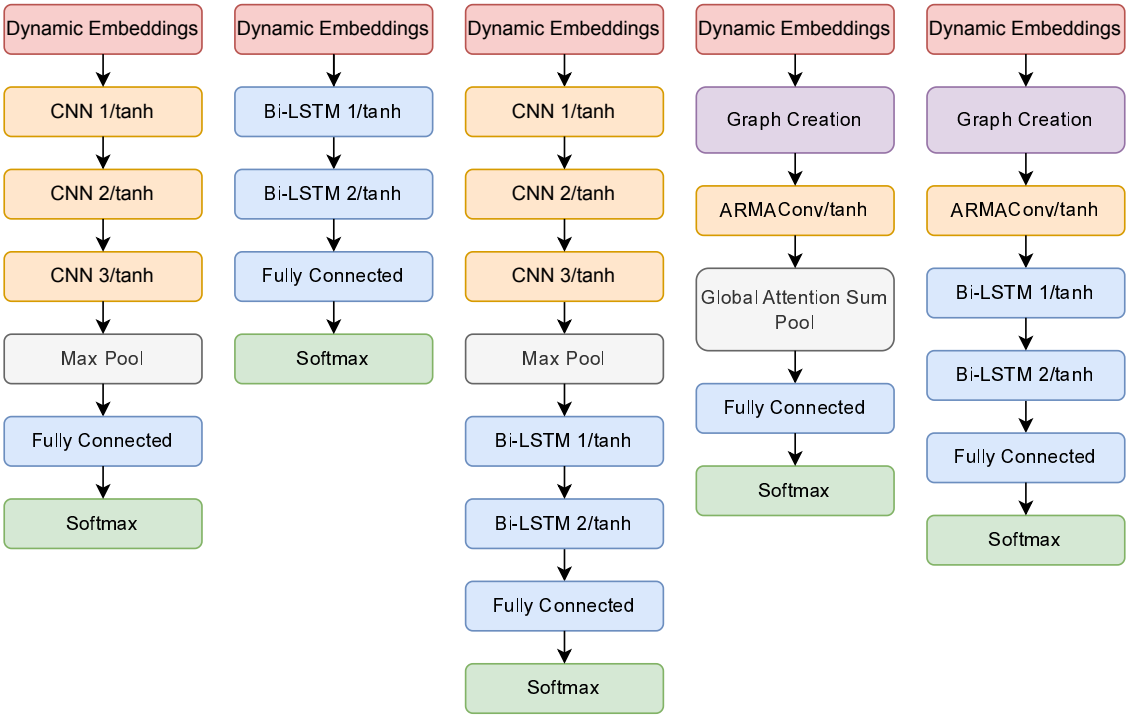

Figure 4.1: Different Structures for Sentence Classification

4.1.1 Sentiment Analysis

The chosen sentiment analysis data set used consists of sentences that are labeled with the emotion the sentence is trying to convey. There are a total of 6 different classes used to classify the sentences, anger, fear, sadness, joy, love, and surprise. The class distribution of the different sentences are shown in Figure 4.2. [15]

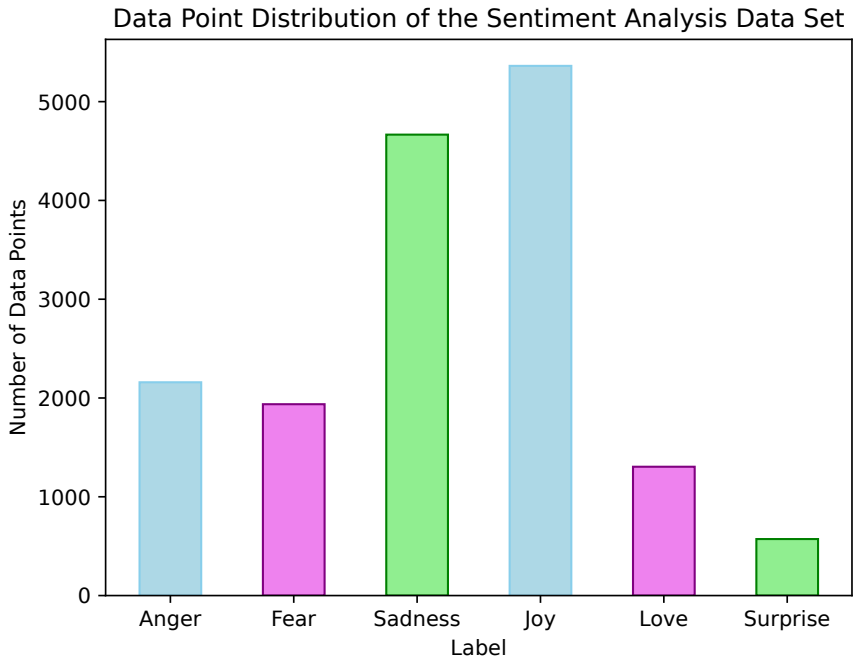

Figure 4.2: Data Point Distribution of the Sentiment Analysis Data Set

The results of testing on the data set are shown below in Table 4.1 and Table 4.2 using the different structures. The different rows represent the word embedding used while the different columns represent the different structures of the network shown in Figure 4.1 to predict the sentence classification.

Table 4.1: Maximum Validation Accuracies of Different Embeddings and Structures on Sentiment Analysis

Embedding	CNNs	Bi-LSTMs	CNNs+Bi-LSTMs	ARMAConv	ARMAConv+Bi-LSTMs
Word2Vec	82.55	93.25	92.20	92.25	**94.10**
GloVe	82.70	92.80	90.40	91.35	**93.55**
ELMo-Like Baseline	87.75	93.20	92.15	92.10	**93.45**
ARMAConv	81.40	92.95	91.65	92.05	**93.10**
ARMAConv+ELMo	82.95	92.95	91.20	90.85	**93.30**

Table 4.2: Validation Accuracies After 50 Epochs Using Different Embeddings and Structures on Sentiment Analysis

Embedding	CNNs	Bi-LSTMs	CNNs+Bi-LSTMs	ARMAConv	ARMAConv+Bi-LSTMs
Word2Vec	80.60	92.40	91.15	90.50	**93.70**
GloVe	80.70	92.50	89.35	89.50	**93.35**
ELMo-Like Baseline	87.75	92.45	91.70	90.55	**92.75**
ARMAConv	79.70	**92.55**	91.40	90.05	91.60
ARMAConv+ELMo	82.80	91.90	90.35	89.00	**93.30**

The validation accuracy over epochs is shown in Figure 4.3 and Figure 4.4 for the sentiment analysis data set.

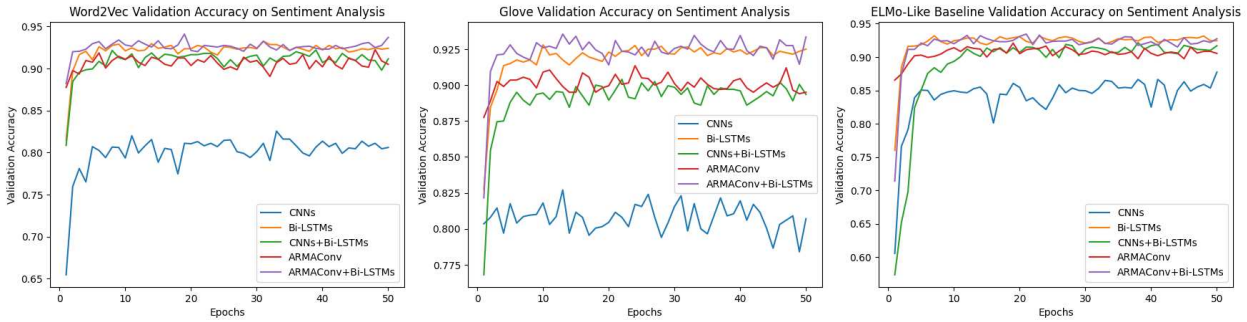

Figure 4.3: Validation Accuracy Over Epochs With Embeddings That Do Not Use ARMAConv on Sentiment Analysis

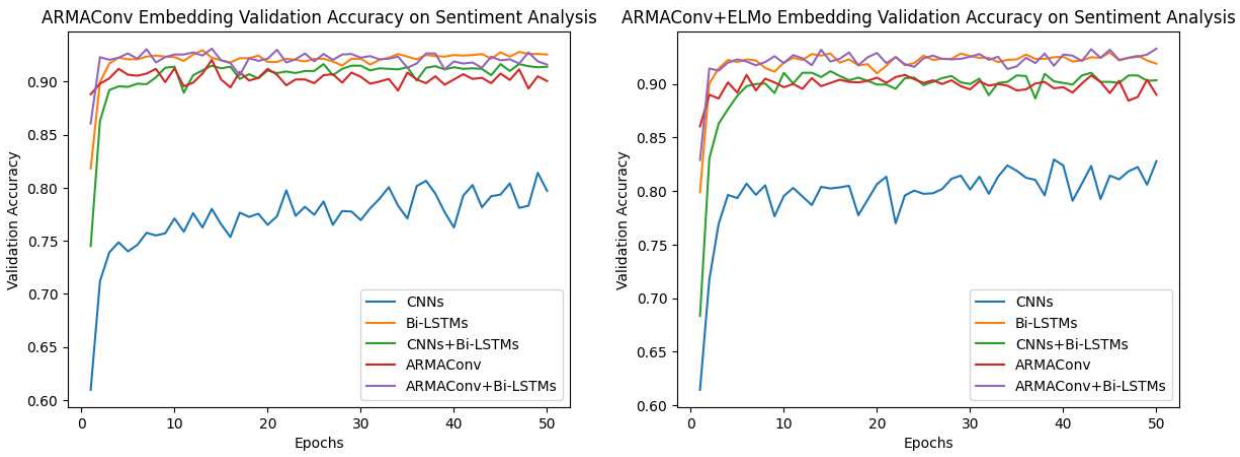

Figure 4.4: Validation Accuracy Over Epochs With Embeddings That Use ARMAConv on Sentiment Analysis

From the data, using the structure ARMAConv+Bi-LSTMs achieves both the highest validation accuracy as well as has the highest validation accuracy after 50 epochs using most embeddings. The static embeddings, Word2Vec and GloVe, achieve a higher validation accuracy compared to the trained dynamic embeddings.

4.1.2 Disaster Tweet Classification

The disaster tweet classification data set contains the tweet, keyword, and location the tweet was sent to classify whether the tweet is describing a disaster or not. This task can be challenging since people can use expressive language while describing either a disaster or not. While training, only the tweet was used as input to isolate the impact of using natural language as input. The data distribution between the two classes is shown in Figure 4.5. [16]

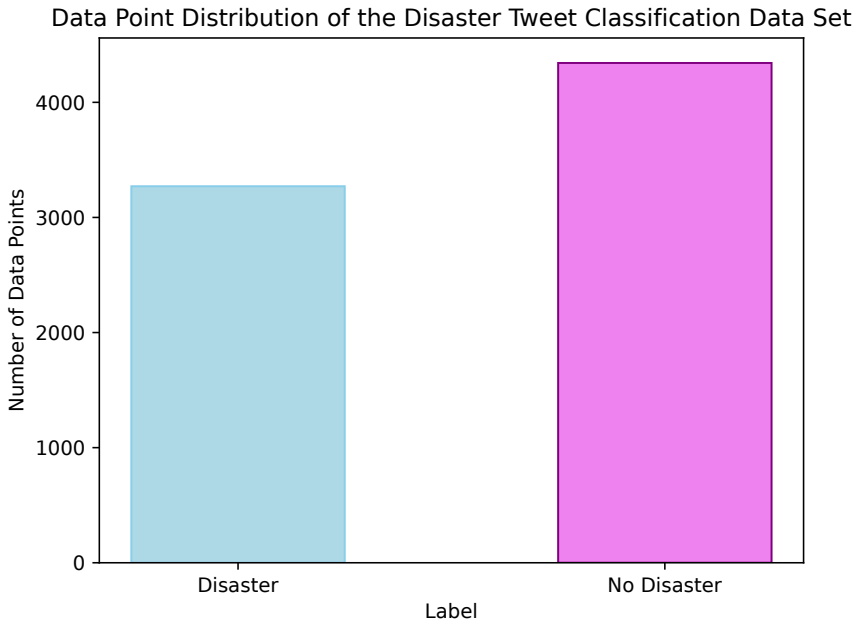

Figure 4.5: Data Point Distribution of the Disaster Tweet Data Set

The results in maximum validation accuracy are shown in Table 4.3 and the validation accuracy after 50 epochs is shown in Table 4.4. The rows represent the different embeddings used while the columns represent the different sentence classification model structure.

Table 4.3: Maximum Validation Accuracies of Different Embeddings and Structures on Tweet Disaster Classification

Embedding	CNNs	Bi-LSTMs	CNNs+Bi-LSTMs	ARMAConv	ARMAConv+Bi-LSTMs
Word2Vec	79.12	79.91	79.51	**80.04**	79.78
GloVe	76.89	79.32	78.33	79.91	**80.63**
ELMo-Like Baseline	79.38	79.51	78.92	80.50	**80.89**
ARMAConv	79.19	79.58	79.71	**81.09**	80.11
ARMAConv+ELMo	78.73	79.71	80.30	**81.68**	80.50

Table 4.4: Validation Accuracies After 50 Epochs Using Different Embeddings and Structures on Tweet Disaster Classification

Embedding	CNNs	Bi-LSTMs	CNNs+Bi-LSTMs	ARMAConv	ARMAConv+Bi-LSTMs
Word2Vec	75.11	**77.35**	76.03	75.25	76.17
GloVe	74.66	75.71	74.85	75.38	**76.63**
ELMo-Like Baseline	70.52	74.39	72.42	**76.89**	76.43
ARMAConv	73.54	76.36	75.71	**77.41**	76.43
ARMAConv+ELMo	75.31	75.97	75.71	**77.41**	76.03

The validation accuracy over epochs is shown in Figure 4.6 and Figure 4.7 on the Tweet classification data set.

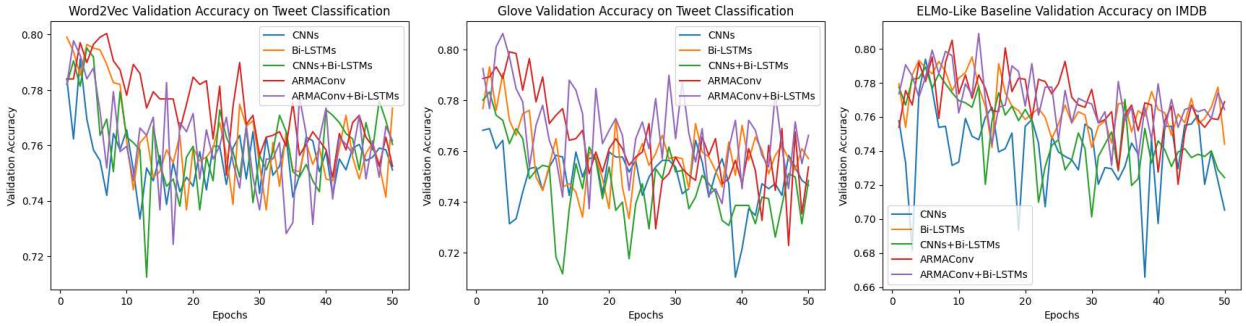

Figure 4.6: Validation Accuracy Over Epochs With Embeddings That Do Not Use ARMAConv on Tweet Classification

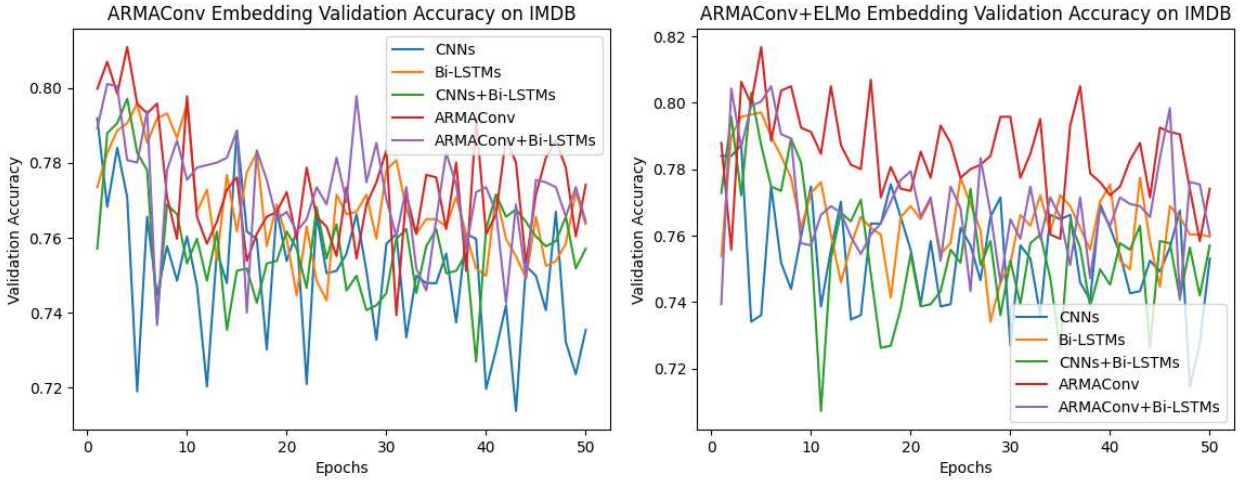

Figure 4.7: Validation Accuracy Over Epochs With Embeddings That Use ARMAConv on Tweet Classification

The data shows that using the structures ARMAConv and ARMAConv+Bi-LSTMs gives an increase in accuracy compared to other structures for most embeddings. For this data set, the ARMAConv and ARMAConv+ELMo embeddings achieve the best maximum validation accuracy and best validation accuracy after 50 epochs.

4.1.3 IMDB Genre Classification

The IMDB data set contains the title and description of a movie to predict the genre. There are 27 different genres to classify in the data set such as action, adventure, and family. For training, only the description is used to predict the genre for the same reason as the tweet classification data set, to isolate the effects of natural language on sentence classification. Since the data set contains such a large amount of information, the first $10,000$ data points are used. The data point distribution for the different classes are shown in Figure 4.8 which shows a class imbalance of throughout the data set which can make classification harder. [17]

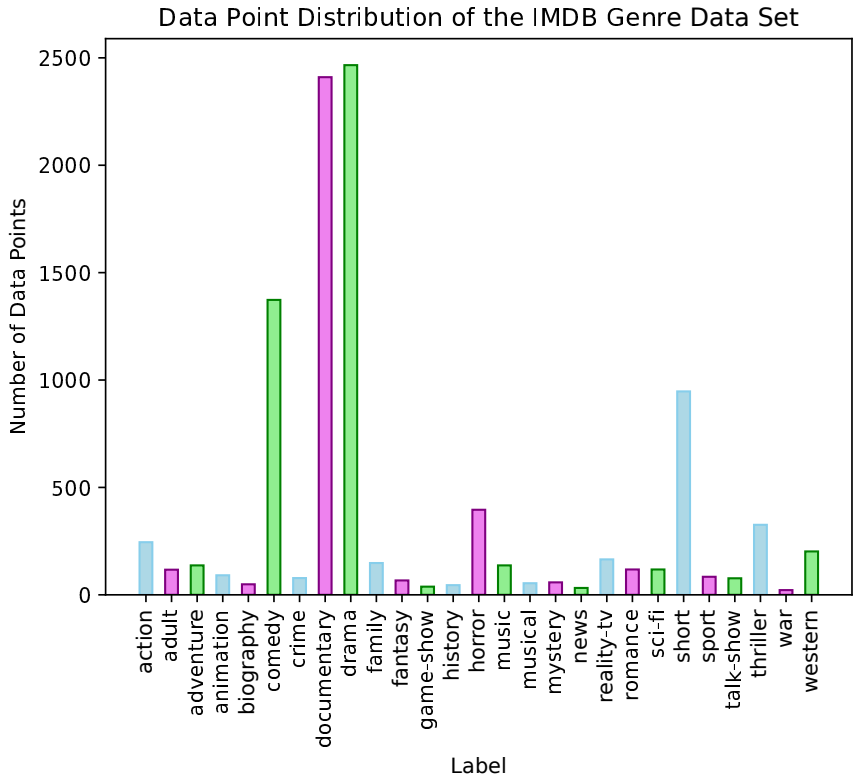

Figure 4.8: Data Point Distribution of the IMDB Data Set

The results are shown in Table 4.5 and Table 4.6 where the rows represent the embedding used as input while the columns represent the different structures used for sentence classification.

Table 4.5: Maximum Validation Accuracies of Different Embeddings and Structures on IMDB Genre Classification

Embedding	CNNs	Bi-LSTMs	CNNs+Bi-LSTMs	ARMAConv	ARMAConv+Bi-LSTMs
Word2Vec	46.25	51.75	48.45	**52.10**	51.60
GloVe	44.75	51.45	48.00	51.65	**52.10**
ELMo-Like Baseline	39.25	49.10	44.40	**49.70**	48.00
ARMAConv	45.30	**51.10**	48.30	50.60	51.05
ARMAConv+ELMo	41.25	50.40	47.15	**50.55**	49.25

Table 4.6: Validation Accuracies After 50 Epochs Using Different Embeddings and Structures on IMDB Genre Classification

Embedding	CNNs	Bi-LSTMs	CNNs+Bi-LSTMs	ARMAConv	ARMAConv+Bi-LSTMs
Word2Vec	39.45	**47.45**	43.80	43.65	47.20
GloVe	42.20	**48.70**	45.30	42.35	45.95
ELMo-Like Baseline	33.50	**46.80**	38.70	44.30	41.85
ARMAConv	41.75	**49.35**	41.15	41.65	47.60
ARMAConv+ELMo	35.15	**47.05**	38.75	41.70	42.70

The validation accuracy over epochs is shown in Figure 4.9 and Figure 4.10 on the IMDB genre classification data set.

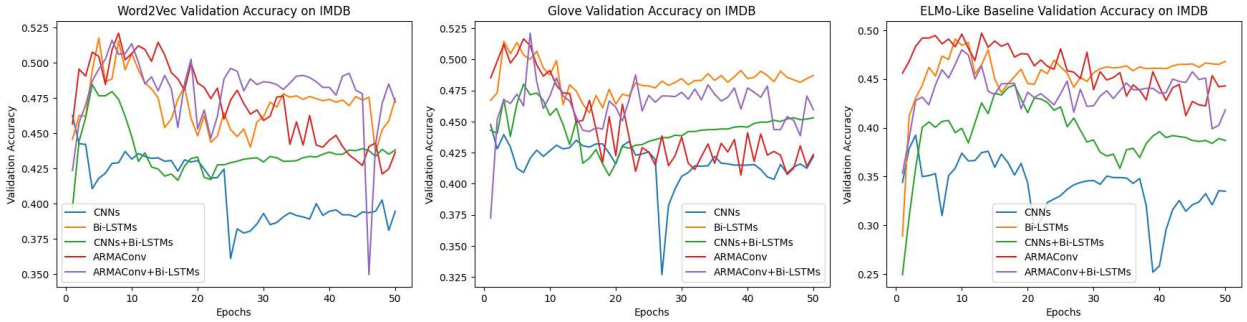

Figure 4.9: Validation Accuracy Over Epochs With Embeddings That Do Not Use ARMAConv on IMDB Genre Classification

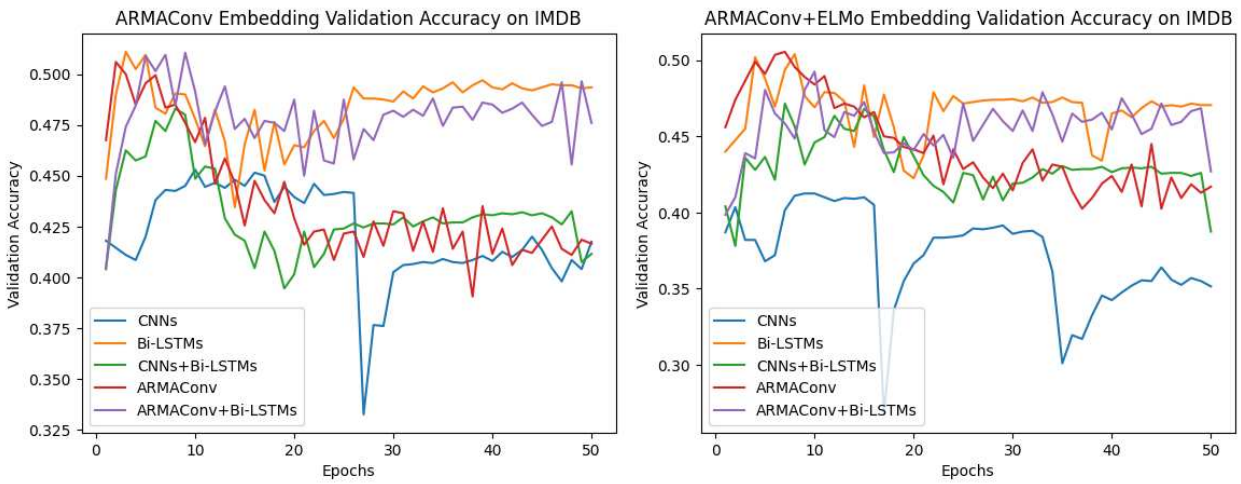

Figure 4.10: Validation Accuracy Over Epochs With Embeddings That Use ARMAConv on IMDB Genre Classification

For the IMDB genre classification data set, using a Bi-LSTM structure is more stable since it achieves the highest validation accuracy after 50 epochs. From the validation accuracy curves some structures lose validation accuracy while training, especially the ARMAConv structure, while using Bi-LSTMs kept a steady validation accuracy after many epochs. For embeddings, Word2Vec and GloVe achieve the highest maximum validation accuracy while the ARMAConv embedding achieves the highest validation accuracy after 50 epochs.

4.1.4 Supervised Learning Analysis

On both the sentiment analysis data set and tweet classification data set, using a graph as input increases validation accuracy, while on the IMDB genre classification data set using a Bi-LSTM achieves better results. A CNN+Bi-LSTM structure does not achieve the highest validation accuracy on any experiment while an ARMAConv+Bi-LSTM structure does multiple times. This may point towards ARMAConv being able to learn more information for the Bi-LSTM compared to a CNN in the context of the run experiments. This may be attributed to the dependency parse connections that are created during the graph creation process. From these observations, using a graph as input for sentence classification tasks shows to be the most beneficial for a majority of data sets used. The structure that achieves the highest validation accuracy varies due to the

unpredictability of neural networks for different data.

The validation accuracy of different embeddings varies with no indication to a clear best embedding. This can also be attributed to word embeddings being unpredictable since a neural network trained them. For the two proposed word embeddings, ARMA and ARMA+ELMo embeddings, they achieve the highest accuracy on the tweet classification data set which indicates that the embedding structure has potential on other data sets as well.

4.2 Word Embedding Comparison

Each embedding produced by the different architectures have shared characteristics such as the output shape and each entry being normalized between -1 and 1. The embeddings also differ from each other such as the specific values that are produced in each dimension of the embedding. The following show the first 5 values from the produced dynamic portion of the embeddings for the word ran in the sentence *He ran down the street* for the ELMo-Like baseline, ARMA, and ARMA+ELMo embeddings.

$$[5.01 * 10^{-2}, -4.75 * 10^{-4}, -6.33 * 10^{-1}, -2.47 * 10^{-4}, 3.67 * 10^{-2}, ...] \tag{4.1}$$

$$[1.47 * 10^{-1}, 1.95 * 10^{-1}, -5.57 * 10^{-2}, -1.90 * 10^{-1}, 1.34 * 10^{-1}, ...] \tag{4.2}$$

$$[1.47 * 10^{-1}, 1.95 * 10^{-1}, -5.57 * 10^{-2}, -1.90 * 10^{-1}, 1.34 * 10^{-1}, ...] \tag{4.3}$$

Along with testing downstream tasks, observing the vectors produced can give insight on how the embeddings represent words. To do this two tests are done, comparing how similar the vectors of different words are and comparing the vectors of the same words but in different contexts. To compare the embeddings, cosine similarity is used which is shown in equation 4.2 where A and B are vectors and $||A||$ is the magnitude of vector A.

$$\frac{A \cdot B}{||A|| \cdot ||B||} \tag{4.4}$$

Since all of the trained word embedding models take in a full sentence and output the embeddings of all of the words in the sentence the following sentences are used as inputs into every model are:

1. He walked to the river bank.
2. She got money from the bank.
3. The fish swam in the shore's bank.
4. The city had a good school district.
5. Universities teach students.

To differentiate between the same words in different sentences the subscript shows what sentence the word is from. For example $bank_1$ refers to the word $bank$ in the first sentence.

4.2.1 Word2Vec

Table 4.7: Comparisons of Static Word2Vec Embeddings

First Word	Second Word	Cosine Similarity
bank_1	bank_2	1
bank_1	bank_3	1
school_4	universities_5	.314
district_4	universities_5	.156

Since Word2Vec is a static embedding technique, even though the word *bank* appears in different contexts it is assigned the same embedding representation. The embeddings still represent semantic meanings between words, since the words *school* and *universities* are more similar than *district* and *universities*. GloVe word embeddings are not analyzed since they share the same quality of being static.

4.2.2 ELMo-Like Baseline

Table 4.8: Comparisons of Dynamic Embeddings using ELMo-Like Baseline

First Word	Second Word	Cosine Similarity
bank_1	bank_2	.983
bank_1	bank_3	.988
school_4	universities_5	.934
district_4	universities_5	.922

Since ELMo-Like baseline uses trained layers to predict the embedding from a next word prediction instead of directly mapping a word to an embedding it is a dynamic embedding. This can be shown since the the word *bank* has different embeddings for each of the different contexts it appears in. The word *bank* in sentences 1 and 3 refer to a bank pertaining to water while the word *bank* in sentence 2 pertains to a bank that holds money. The similarity of the two embeddings pertaining to a *bank* of water is higher than the similarity between the two that pertain to different meanings of the word *bank*. Like Word2Vec, the similarities of *school* and *universities* is higher than *district* and *universities* showing that the embeddings also capture semantic information.

4.2.3 ARMAConv Embeddings

Table 4.9: Comparisons of Dynamic Embeddings using ARMAConv Embeddings

First Word	Second Word	Cosine Similarity
bank_1	bank_2	.859
bank_1	bank_3	.872
school_4	universities_5	.286
district_4	universities_5	.248

The embeddings are able to capture semantic meaning since the similarity between *school* and *universities* is greater than the similarity between *district* and *universities*. Like the ELMo-Like Baseline the two words *bank* that both refer to the same type of bank has a higher cosine similarity compared to the the cosine similarity of *bank* when it refers to different types of banks.

4.2.4 ARMAConv+ELMo Embeddings

Table 4.10: Comparisons of Dynamic Embeddings using ARMAConv+ELMo Embeddings

First Word	Second Word	Cosine Similarity
bank_1	bank_2	.883
bank_1	bank_3	.749
school_4	universities_5	.539
district_4	universities_5	.526

The words *school* and *universities* has a larger cosine similarity compared to *district* and *universities* which is expected. This shows that the embedding is capable of learning semantic information about words. When comparing the variations of the word *bank* with each other, the two *bank* words that appear in different contexts have a larger cosine similarity compared to the two that refer to the same type of bank. This may be because more training time is required since learning differences between the same words given different contexts is a complex task for a neural network.

Chapter 5

Future Research and Conclusion

5.1 Future Research

GNNs applied to NLP tasks are a relatively new development, so further testing is needed to determine if GNNs can be beneficial for more NLP tasks. Steps such as graph creation and model structure still has much to explore since there are many different structural combinations.

Graph creation has the most to explore with many different tools that are available to help construct a graph representation. The dependency parse used in this study only used the structure but not the dependency type values in the parse which should be explored on how to properly represent and use those values. Other parse methods that have been proposed but not tested extensively such as constituency graphs, AMR graphs, and similarity graphs can be tested to determine if they work better compared to a dependency graph. The most promising graph creation method that was covered in the survey written by Wu et al. is a dynamic graph which learns its structure/adjacency matrix within the neural network using trainable weights. [2]

Prior model structures can give insight on how to structure GNNs for NLP tasks. This study only studies integrating a GNN into a model structure like the already existing ELMo structure while many other structures exist.[3] The most promising current structure in NLP are transformers using self attention. The graph attention network leverages self attention which could be beneficial when studied on how to apply them to NLP tasks. [18]

5.2 Conclusion

Sentences are more complex than a simple sequential representation of words. Instead, sentences can be represented as words and how each word depends other words in the sentence using a graph constructed from a dependency parse. When used as input to the GNN ARMAConv, ac-

curacy can increase compared to using a sequential representation. Embeddings produced with ARMAConv has shown to outperform other embeddings on one of the tested data sets. Applying a graph structure to NLP tasks shows to be promising but has not been extensively tested making room for future research into different applied structures.

Bibliography

[1] Matthew Honnibal and Ines Montani. Spacy · industrial-strength natural language processing in python, 2015.

[2] Lingfei Wu, Yu Chen, Kai Shen, Xiaojie Guo, Hanning Gao, Shucheng Li, Jian Pei, Bo Long, et al. Graph neural networks for natural language processing: A survey. *Foundations and Trends® in Machine Learning*, 16(2):119–328, 2023.

[3] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. corr abs/1802.05365 (2018). *arXiv preprint arXiv:1802.05365*, 2018.

[4] Kun Xu, Lingfei Wu, Zhiguo Wang, Mo Yu, Liwei Chen, and Vadim Sheinin. Exploiting rich syntactic information for semantic parsing with graph-to-sequence model. *arXiv preprint arXiv:1808.07624*, 2018.

[5] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[6] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[7] Oren Melamud, Jacob Goldberger, and Ido Dagan. context2vec: Learning generic context

embedding with bidirectional lstm. In *Proceedings of the 20th SIGNLL conference on computational natural language learning*, pages 51–61, 2016.

[8] Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 302–308, 2014.

[9] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[10] Daniele Grattarola and Cesare Alippi. Graph neural networks in tensorflow and keras with spektral [application notes]. *IEEE Computational Intelligence Magazine*, 16(1):99–106, 2021.

[11] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph neural networks with convolutional arma filters. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3496–3507, 2021.

[12] Mike Ortman. Wikipedia sentences, Aug 2018.

[13] François Chollet et al. Keras. https://keras.io, 2015.

[14] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[15] Praveen. Sentiment data set, Apr 2020.

[16] Phil Culliton Yufeng Guo Addison Howard, devrishi. Natural language processing with disaster tweets, 2019.

[17] Radmirkaz. Genre classification dataset imdb, Jun 2021.

[18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.