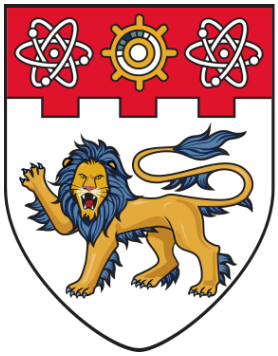

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SCE17-0432

Graph Convolutional Neural Networks for Text Categorization

SUYASH LAKHOTIA

Submitted in Partial Fulfilment of the Requirements for the Degree of Bachelor of Engineering (Computer Science) from Nanyang Technological University

Supervised by Assoc. Prof. Xavier Bresson

March 2018

School of Computer Science and Engineering

Abstract

Text categorization is the task of labelling text data from a predetermined set of thematic labels. In recent years, it has become of increasing importance as we generate large volumes of data and require the ability to search through these vast datasets with flexible queries. However, manually labelling text data is an extremely tedious task that is prone to human error. Thus, text classification has become a key focus of machine learning research, with the goal of producing models that are more efficient and accurate than traditional methods.

This project explores the recently enhanced deep learning techniques of convolutional neural networks and their fusion with graph analysis (i.e. graph convolutional neural networks) in the field of text categorization and compares their performance to established baseline models and simpler multilayer perceptrons. We show through experiments on three major text classification datasets (Rotten Tomatoes Sentence Polarity, 20 Newsgroups and Reuters Corpus Volume 1) that graph convolutional neural networks can naturally work in the space of words represented as a graph and perform with greater or similar test accuracy when compared to standard convolutional neural networks and simpler baseline models.

A Note on Notation

Abbreviated model architectures in this report abide by the following notation:

- Fully-Connected Layers: FC[No. of Units]
- Convolutional Layers: CL[Filter Width]_[No. of Features]
- Graph Convolutional Layers: GC[Filter Size / Support Size]_[No. of Features]
- (Graph) Pooling Layers: P[Pooling Size]
- Multiple Layers: [Layer 1]-[Layer 2]

Please note that dropout layers, reshaping/concatenation operations and the output softmax layer are ignored when the architecture is noted in its abbreviated form.

Acknowledgments

I would like to sincerely express my gratitude to Prof. Xavier Bresson for his constant motivation and guidance throughout the course of my Final Year Project. He has been extremely supportive and helped me overcome numerous obstacles I encountered while pursuing the project. This project could not have been completed without his expertise, advice and mentorship.

Apart from him, I would also like to extend my gratitude to the School of Computer Science and Engineering at Nanyang Technological University, Singapore for providing me with the opportunity to pursue this project and granting me the computational resources required for the experiments cited in this report.

Last but not the least, I would like to thank my family and friends for their encouragement and support over the course of this project.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Motivation

Over the last couple decades, the field of content-based document management tasks collectively known as information retrieval has gained prominent status due to the increased availability of documents in their digital form and the ensuing need to access them in flexible ways. One of the key techniques used to handle and organize text data is text categorization (or text classification, topic spotting), which is the task of labelling natural language texts with thematic categories from a predefined set [22]. This method of 'tagging' text data has wide applicability in real world applications as it allows users to classify news stories, efficiently search for information on the Internet and catalog customer reviews as positive or negative, to cite a few examples. Because building text classifiers by hardcoding the rules is difficult, time-consuming and may lead to inconsistent results, it is beneficial to learn these classifiers from existing labelled data using machine learning.

In the field of text categorization, text data is often represented as document vectors, which are essentially histograms of the words in the document that match a fixed vocabulary extracted from the corpus. There are several existing machine learning algorithms that have proven to be useful in classifying document vectors, most notably the application of Linear Support Vector Classifiers and Multinomial Naïve Bayes Classifiers. The same document vectors have also been successfully used to train neural networks made up of fully-connected layers (i.e. multilayer perceptrons) to classify text data. However, with recent strides in deep learning, various models and algorithms have surfaced that not only take advantage of the document vectors but also the underlying vocabulary of the raw documents by performing composition over the vocabulary's corresponding word embeddings [5]. Pre-trained word vectors, such as word2vec [16] or GloVe [20], encode the semantic relationships between words, thus, allowing the model to explore associations between the semantics of a text and its assigned label.

One such example of this is the application of convolutional neural networks (CNNs) in the field of text categorization. CNNs are a class of deep, feed-forward artificial neural networks

inspired by biological processes that have primarily been popularized due to their success in image, video and sound tasks. They work by translating convolutional filters across data represented as a grid, revealing local features that are shared across the data domain [12]. CNNs can be applied on text data by replacing the original words with their respective word vectors (thus forming a 2-D grid) and sliding convolutional filters across this sequence of embeddings, extracting potentially useful information.

However, while CNNs have been extremely successful with applications where the underlying data representation has a grid structure, they fall short when certain datasets can be better represented by higher-dimensional data structures like graphs. As a result, novel deep learning techniques have emerged in recent years through the fusion of traditional CNNs and graph analysis to tackle this very problem [3] [6]. Graph convolutional neural networks (Graph CNNs or GCNNs) use filters that translate in the graph's spatial domain and use the same motivations of local features and translational invariance as standard CNNs. GCNNs can be applied to the field of text categorization by feeding the network document vectors as well as a feature graph constructed from the word embeddings of the underlying vocabulary, allowing the network to take advantage of the semantic relationships between the word vectors as well as the relationship between the frequency of the words and the class (i.e. category or topic) of the document.

1.2 Purpose and Scope

The purpose of this project is to (1) implement existing deep learning models in the field of text categorization, (2) improve their architecture and (3) compare their performance on popular datasets against each other as well as against standard baseline models to build the best possible model(s) for text categorization, with a primary focus on graph convolutional neural networks.

This project explores three major deep learning architectures for text categorization, namely F. Chollet's CNN [4], Y. Kim's CNN [10] and Graph CNN (ChebNet [6], Spline [3] and Non-Parametric Fourier). We compare their relative performance on three benchmark text classification datasets – Pang and Lee's Rotten Tomatoes sentence polarity dataset [17] [18],

20 Newsgroups [21] and Reuters Corpus Volume 1 [14]. The pre-trained word embeddings used throughout this project are Google's publicly released word2vec word vectors (for 3 million words and phrases), which were originally trained on a portion of the Google News dataset (about 100 billion words) [15].

Additionally, at the end of the implementation, training and selection of the models, we build a client-side application to better demonstrate the capabilities of the chosen model(s).

1.3 Contributions

This project was completed successfully and all of its objectives were fulfilled. The primary contributions of the project are listed below:

- All three deep learning models and three baseline models were implemented in Python 3 using TensorFlow & scikit-learn (Chapter 3) and the entire codebase has been released as open-source software.[1]
- Improvements were made to the deep learning models by modifying their architecture, strengthening regularization and tweaking weight initialization in addition to other minor changes (Chapter 3).
- The models were comprehensively tested on the three benchmark datasets and their performance was compared against each other as well as published results in existing literature (Chapters 4 & 5).
- Our machine learning pipeline (preprocessing + model) for graph convolutional neural networks outperformed test accuracies quoted in existing literature for 20 Newsgroups in [6] and RCV1 in [7] (Chapters 4 & 5).
- A client-side application for news classification was built for selected models trained on the RCV1 dataset (Chapter 6).

1.4 Project Timeline

This project was worked on from August 2017 to March 2018, including the time needed for us to familiarize ourselves with the subject matter. Progress on this project was tracked using

[1] `https://github.com/SuyashLakhotia/TextCategorization`

a blog[2], where summaries of the completed tasks and interesting findings were uploaded on a weekly basis. Meetings with the supervising professor were scheduled on a bi-weekly basis during the initial stages of the project and on a weekly basis towards the end to clarify doubts and discuss the results of the project thus far.

1.5 Organization of Report

The remainder of this report is organized as follows:

- Chapter 2 provides background information and elaborates on the intuitions behind the baseline models, neural networks, convolutional neural networks, graph convolutional neural networks and word embeddings.
- Chapter 3 describes the specific deep learning models implemented and tested in this project as well as our improvements to these models.
- Chapter 4 details the datasets used to benchmark the models, the preprocessing of these datasets and the performance results of the models.
- Chapter 5 discusses the results from Chapter 4 and provides insights into the differences in the models' performance.
- Chapter 6 gives a brief overview of the client-side application built for the best models trained on the RCV1 dataset.
- Chapters 7 & 8 conclude this report and provide suggestions for future work in the field of deep learning for text categorization.

[2] http://suyashlakhotia.com/FYPSnippets/

2. Background / Literature Review

2.1 Linear Support Vector Classifiers

Linear Support Vector Classifiers are a class of machine learning algorithms that aim to define a linear boundary (i.e. linear hyperplane) in the multi-dimensional feature space to separate samples belonging to different classes. The goal of a Linear SVC (in the case of linearly separable classes) is to define a linear hyperplane that maximizes the margin between the samples of different classes, improving generalization. However, since most real-world datasets are not truly linearly separable, slack variables are introduced that allow a certain number of samples to lie on the wrong side of the hyperplane, creating a "soft margin".

Mathematically, for a two-class linearly separable classification problem, Linear SVCs define a linear hyperplane such that for any sample $\boldsymbol{x}$:

$$\boldsymbol{x}^T\boldsymbol{w} + b \geq +a \quad \text{for} \quad y = +1$$
$$\boldsymbol{x}^T\boldsymbol{w} + b \leq -a \quad \text{for} \quad y = -1$$

Figure 1: Linear SVC Equations for Linearly Separable Classes

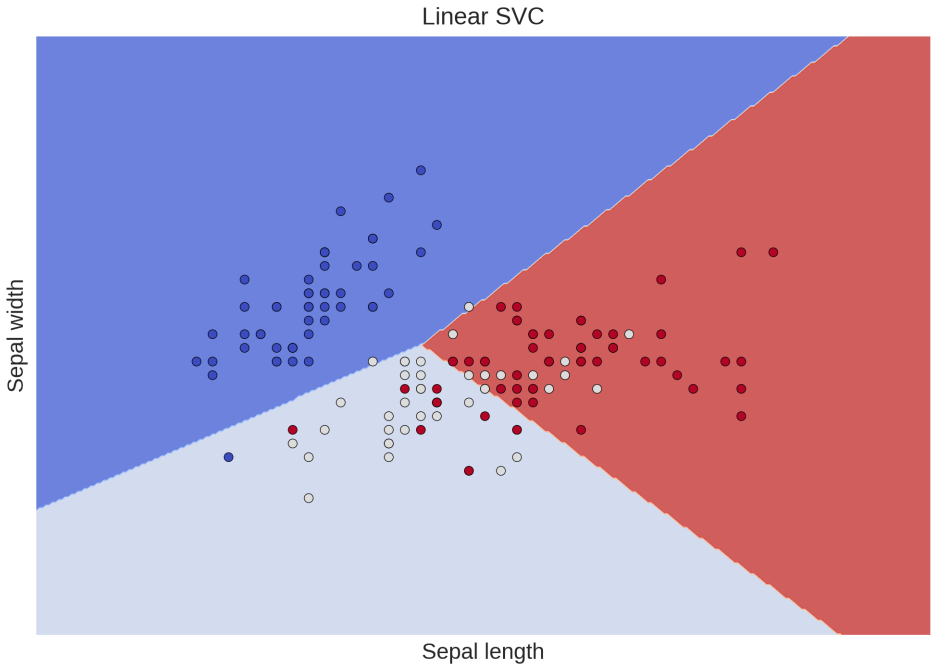

Figure 2: Linear SVC on Iris Dataset (2 Features, 3 Classes)

2.2 Multinomial Naïve Bayes Classifiers

Naïve Bayes classifiers are a set of supervised machine learning algorithms based on Bayes' theorem with the assumption of independence between every pair of features. In spite of their simplicity, naïve Bayes classifiers have been shown to perform well on several real-world tasks, including document classification and spam filtering [26].

Mathematically, for any input sample $\boldsymbol{x}$ with n features:

$$\hat{y} = \underset{y}{\operatorname{argmax}}\, P(y) \prod_{i}^{n} P(x_i|\, y)$$

Figure 3: Naïve Bayes Classifier

A multinomial naïve Bayes classifier differs from a traditional naïve Bayes classifier in that it makes the assumption that the distribution of $P(x_i|\, y)$ is multinomial, which is helpful when dealing with features like word frequencies.

2.3 Artificial Neural Networks

Artificial neural networks (or neural networks) are the foundation of most deep learning algorithms and a popular choice in machine learning due to their flexibility and large learning capacity. They are inspired by their biological counterparts found in the human brain in that their basic units (known as neurons) perform a summation over their weighted inputs ($\boldsymbol{x}^T\boldsymbol{w}$) and a bias term ($b$), forming the synaptic input. This is then passed through an activation function (f) whose output (h) is the output of the neuron.

$$h = f\left(\sum_{1}^{n} w_i x_i + b\right)$$

Figure 4: Output of a Basic Neuron

Neurons typically have non-linear activation functions, which allow them to perform well on data that is not linearly separable. Popular activation functions include the sigmoid function, tanh function and step function. In this report, we'll be using the Rectified Linear Unit or ReLU

activation function $f(u) = \max\{0, u\}$ for neurons in hidden layers and the softmax function for the final classification in the output layer. The softmax function (Figure 5) outputs the probability that synaptic input u belongs to class j in a K-class classification problem.

$$\sigma(u)_j = P(y = j \mid u) = \frac{e^{u_j}}{\sum_{k=1}^{K} e^{u_k}}$$

Figure 5: Softmax Function

Neurons are typically stacked into layers, which are sequentially applied on the input, forming a neural network. The initial layer, known as the input layer, corresponds to the input features and in the case of classification, the output layer is usually a softmax layer with as many neurons as number of classes. Fully-connected layers in between the two are known as hidden layers and form the architecture for a multilayer perceptron (MLP). Note that MLP is a commonly-used misnomer for a more complicated neural network and is not necessarily related to a single-layer perceptron.

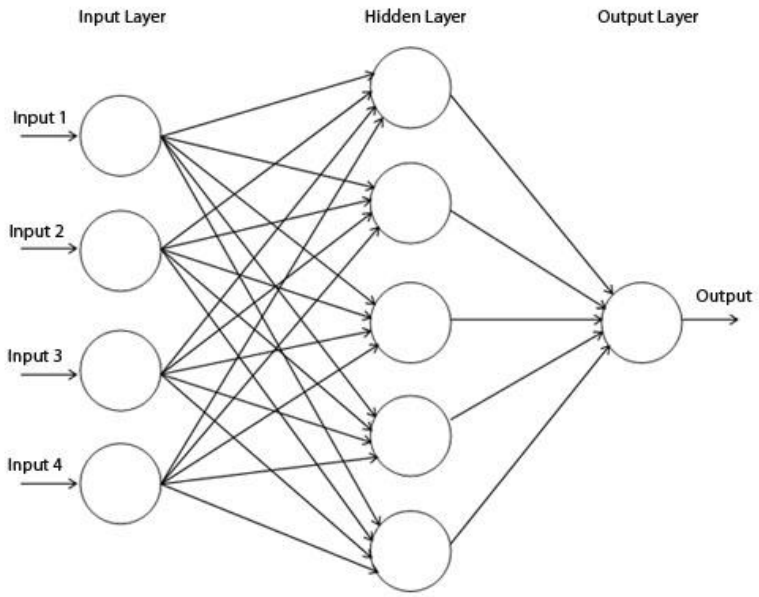

Figure 6: Architecture of a Multilayer Perceptron

Neural networks are trained by systematically adjusting the weights and biases of every neuron in the network using the error backpropagation algorithm, which employs gradient descent to modify the parameters of the network in order to minimize the loss.

2.4 Convolutional Neural Networks

Convolutional neural network (CNN) is a neural network architecture that has found great triumph in tasks involving image, sound and video data [13]. CNNs have especially been successful in image classification tasks due to their properties of local connectivity, shared parameters and assumption of translational invariance.

Unlike regular neural networks where each neuron is fully connected to all the neurons in the previous layer, convolutional neural networks exploit spatially local correlations by forcing local connectivity between neurons of adjacent layers. That is, the receptive fields of neurons are limited, forming a filter that is replicated across the entire visual field. These replicated units share the same weights and biases and form feature maps. By replicating the filter parameters in this manner, features are detected regardless of their position in the visual field (i.e. translational invariance). The ability of CNNs to extract features and compose them to multi-scale hierarchical patterns is one of its core strengths.

Convolutional neural networks typically consist of alternating convolutional and pooling layers followed by fully-connected layers. The convolutional layer learns features over small patches of the image and these learned features are then convolved with the image to obtain the feature maps. Pooling layers, on the other hand, are used to downsample the output from the convolutional layer by either taking the maximum (max-pooling) or the mean (average-pooling) within disjoint pooling regions. During backpropagation, the error terms are upsampled at the pooling layer. An example of a popular convolutional neural network architecture for handwritten character recognition is shown in Figure 7.

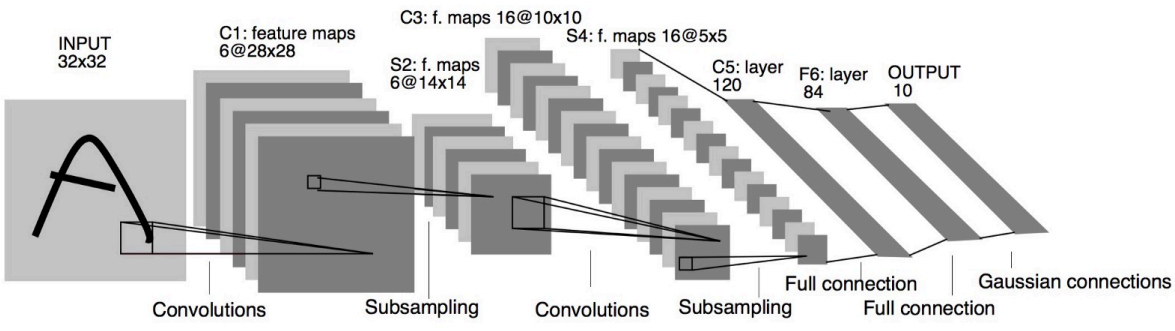

Figure 7: LeNet5 Architecture [12]

2.5 Graph Convolutional Neural Networks

While neural networks perform extremely well in several applications, generalizing well-established model architectures to work with arbitrarily structured graphs has proven to be a hard problem due to the increased complexity. One such architecture aimed at tackling this challenge in the recent past is graph convolutional neural networks or GCNNs. As the name suggests, GCNNs take inspiration from traditional CNNs in that they make the same statistical assumptions of stationarity and compositionality through local features but instead of working on an n-dimensional grid, they perform convolution and pooling operations on a graph's spatial domain [3].

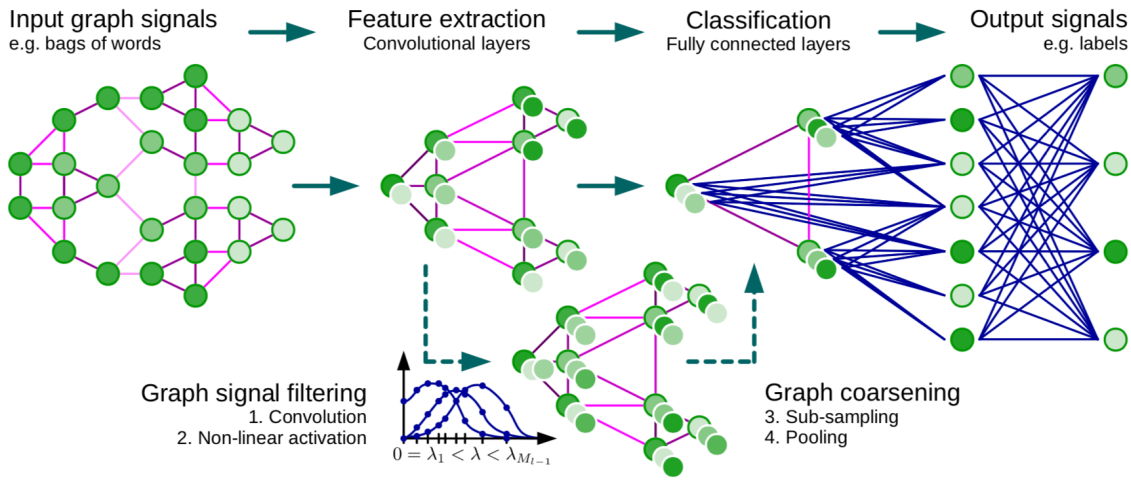

Figure 8: Graph Convolutional Neural Network for Text Classification [6]

Because CNNs were originally designed to work with grid-structured data, the convolution and pooling operations had to be re-designed for their application in GCNNs. The methods quoted in this report use a spectral approach to solve this problem by performing the convolution operation on the Fourier domain [3], where strictly localized filters work in a ball of radius K i.e. K hops from the central vertex (K = filter size or support size). Over the past couple years, graph convolution operations have been made much more efficient, with the key focus of this report being on the O(n) Chebyshev filter introduced in [6].

When applied to text classification, GCNNs take in two inputs. The first is the Laplacians of the feature graph, which is a static nearest-neighbor graph composed of all the words in the

chosen vocabulary that is constructed using word embeddings. That is, the distance between two nodes in the graph corresponds to the distance between their respective word vectors. This feature graph is constructed once using the training set and is provided to the graph convolutional and pooling layers. The second input is the document vector for every sample, which acts as the input graph signal since each feature in the document vector represents a value for each word in the vocabulary i.e. each node in the feature graph.

2.6 Word Embeddings

Word embedding is a technique in natural language processing that maps words and phrases to vectors of real numbers. Mathematically, this means embedding from a space with one dimension per word to a continuous vector space with much lower dimensionality. When used as the underlying input representation, word embeddings have proven to increase performance in several NLP tasks such as syntactic parsing and sentiment analysis [23].

One of the methods used to generate word embeddings or word vectors is using neural networks. The network is fed a large corpus of text and it produces a vector space of several hundred dimensions, where each word is assigned a unique vector. There are two popular types of word vectors being used today – word2vec & GloVe. Briefly, word2vec uses a predictive model based on the surrounding words (i.e. context) to extract word vectors [16] while GloVe is a technique that performs dimensionality reduction on the co-occurrence matrix of the vocabulary [20]. We'll only be using word2vec word embeddings throughout this report, specifically Google's publicly released word2vec vectors.

The advantage of word2vec vectors is that the neural network generating the embeddings is trained to reconstruct linguistic contexts of words, preserving semantic analogies. For example, $\mathrm{vector(Paris)} - \mathrm{vector(France)} + \mathrm{vector(Indonesia)}$ should result in a vector very close to $\mathrm{vector(Jakarta)}$. Therefore, word vectors are positioned in the vector space such that words that share similar context are located close to each other. This is especially helpful in the field of text categorization, where the context of the words in the vocabulary can help the model pick up similar keywords unique to a class.

3. Models

3.1 Baseline Models

To provide a benchmark for the more complex models presented in this report, two baseline models were implemented using the scikit-learn library for Python – a Linear Support Vector Classifier (Linear SVC) and a Multinomial Naïve Bayes Classifier (Multinomial NB).

Support Vector Machines (SVMs) are generally effective in high-dimensional spaces and are capable of accurately separating samples into their respective classes. The implementation we employed (sklearn.svm.LinearSVC) uses a "one-vs-the-rest" multi-class strategy, which trains as many models as number of classes. The Linear SVC models quoted in this report were trained with the default hyperparameter values as set in the library (squared hinge loss function, penalty parameter of 1.0), unless stated otherwise.

The Multinomial Naïve Bayes Classifier implements the Naïve Bayes algorithm for multinomially distributed data and is commonly used in text classification. While the implementation we used (sklearn.naive_bayes.MultinomialNB) usually requires integer features, fractional counts have been shown to work as well and are used in this project. All Multinomial NB models referenced in this report were trained with an additive Lidstone smoothing of 0.01 and default values for the remaining hyperparameters, unless stated otherwise.

3.2 Multilayer Perceptron

In order to provide another benchmark for the results of the experiments presented in this report, a multilayer perceptron (MLP) and a neural network without hidden layers (i.e. simply a softmax layer) were implemented using the TensorFlow library in Python (as were all the other neural network models). Any hidden layers in the network use the ReLU activation function and apply dropout for regularization while the output layer is a standard softmax layer for classification. This model and all the other neural network models in this report were trained using the Adam optimizer [11] as it has been proven to be extremely effective at

learning deep models, providing separate learning rates for each network parameter that are also adapted individually. The optimizer was used with the default decay rates & epsilon value as defined in TensorFlow and an initial learning rate of 0.001 for all the neural network models in this report.

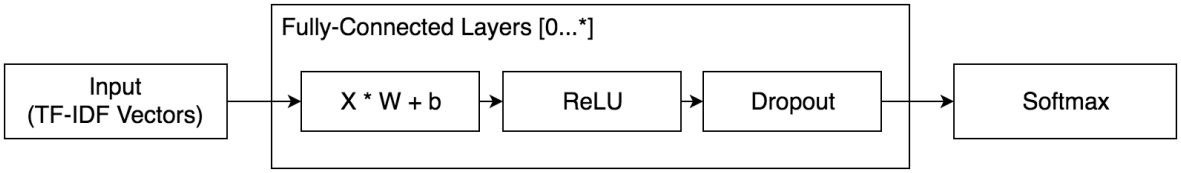

Figure 9: Multilayer Perceptron for Text Categorization

3.3 F. Chollet CNN

3.3.1 Model Description

Francois Chollet's CNN (FC-CNN) for text data [4] is a deep model that applies consecutive convolutional and pooling layers followed by fully-connected layers on sequences of word embeddings. The input provided is a (truncated or padded) sequence of word identifiers from a fixed vocabulary, which are replaced with their respective word embeddings (Google's 300-dimensional word2vec embeddings) in the Embedding Layer, forming a tensor of shape [sequence_length x embedding_size] for each sample. This is then passed through the network as shown in Figure 10 below.

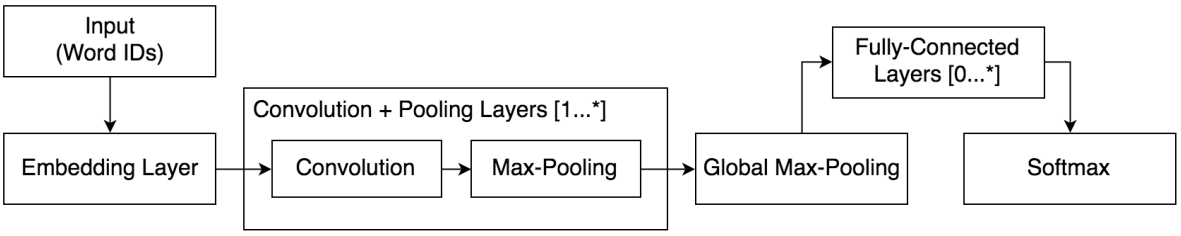

Figure 10: F. Chollet CNN for Text Categorization

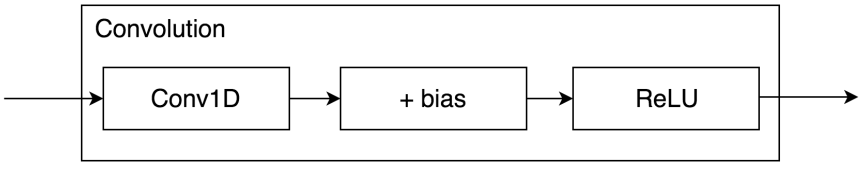

Figure 11: Convolution Operation

The convolution and pooling operations in FC-CNN are the 1-D variants of the traditional operations where the filters match the height of the embeddings and the width determines how many words are convolved at a time. Figure 12 shows how a convolutional filter would traverse a sentence, where each box represents an iteration of the filter from left to right.

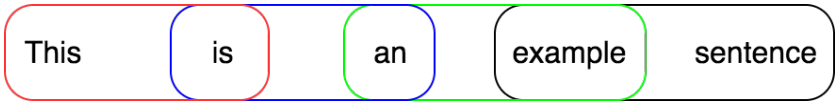

Figure 12: Convolutional Filter with Width of 2 and Stride of 1

The output from the sequential convolutional and pooling layers goes through global max-pooling and fully-connected layers before being classified by a softmax layer. The fully-connected layers use the ReLU activation function and apply dropout as in the MLP model.

3.3.2 Changes from Original Implementation

This model was adapted from Francois Chollet's blog post on The Keras Blog [4], where he also presents an implementation of the model in Keras. The architecture of the model remains the same with the exception of the incorporation of dropout in the fully-connected layers for regularization. Chollet also uses GloVe embeddings extracted from Wikipedia that remain static during training while our implementation fine-tunes Google's word embeddings for the task at hand, which should generally boost model performance as noted in [10]. Finally, Chollet employs the RMSProp learning algorithm for his version of the model as opposed to the Adam optimizer used in our implementation.

3.4 Y. Kim CNN

3.4.1 Model Description

Yoon Kim's CNN (YK-CNN) for text data was introduced in [10] and further analyzed in [27]. It is similar to Chollet's more traditional architecture, however, instead of sequentially applying the convolution and pooling operations, they are applied in parallel to the same input (i.e. sequence of word embeddings) and the outputs are concatenated before being fed to fully-connected layers and classified by the output softmax layer. The convolution operations work in an identical fashion to Chollet's CNN (see Figure 12), however, the pooling operation is

global to reduce the axes of the data and output the same number of features from each Convolution + Max-Pooling combination for concatenation.

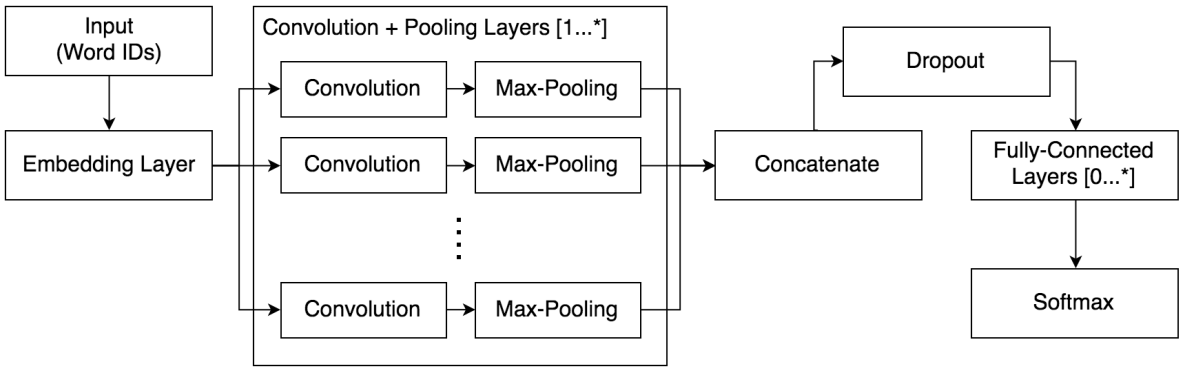

Figure 13: Y. Kim CNN for Text Categorization

The parallel convolutional layers allow the network to intuitively look at sequences of word embeddings from the perspective of different n-grams (corresponding to different filter widths), extracting diverse contextual features for the network to learn from.

3.4.2 Changes from Original Implementation

Our implementation of Yoon Kim's CNN is based on Denny Britz's TensorFlow implementation in [2] (Kim's original implementation is in Theano), which uses the Adam optimizer instead of Adadelta as in the original paper, removes L2-constraints on the weights (shown to have no effect in [27]) and changes the weight initializations from random uniform distributions (for convolutional layers) and random normal distributions (for fully-connected layers) to uniform Xavier initialization for all layers.

3.5 Graph CNN

3.5.1 Model Description

Graph convolutional neural networks (GCNNs) are one of the latest breakthroughs in deep learning and the primary focus of this report. Our implementation of graph convolutional neural networks is based on the paper by M. Defferrard, X. Bresson & P. Vandergheynst [6] and the supporting codebase available on GitHub.

GCNNs take in the feature graph and the input graph signal and apply localized graph filters in order to extract meaningful features from the data. In the case of text categorization, the feature graph is constructed from the pre-trained word embeddings of the vocabulary. That is, the nodes in the graph represent the words in the vocabulary while the edges between the nodes correspond to the similarity between the words, which is computed using the cosine similarity between the respective word embeddings. Only the top N edges are retained (i.e. nearest-neighbor graph) to restrict the number of edges in the graph. The graph signal passed for each sample is the document vector, which is composed of the same vocabulary as present in the feature graph.

The architecture of the model is fairly similar to traditional CNNs, with consecutive graph convolutional layers and graph pooling layers followed by a dropout layer, fully-connected layers and an output softmax layer for classification. The fully-connected layers in the GCNN are identical to the ones used in MLP (i.e. ReLU & dropout), however, batch normalization is also applied before the non-linearity (explained in Section 3.5.3).

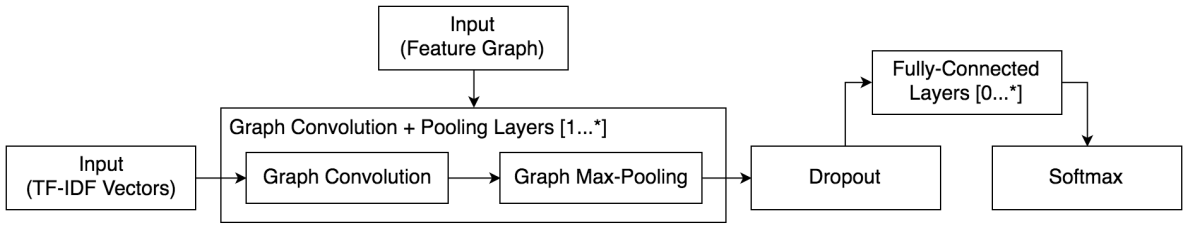

Figure 14: Graph CNN for Text Categorization

3.5.2 Filters Implemented

There are three main graph filters implemented and tested for GCNNs in this report – Chebyshev (filter for ChebNet) [6], Spline [3] & Non-Parametric Fourier. The implementations used are identical to the ones provided in the codebase for [6] and further details about the filters (including the underlying theory and math) can be found in [6] and [3]. They are also briefly touched upon in Section 2.5 of this report.

3.5.3 Improvements

3.5.3.1 Dropout

The first improvement made to the original architecture was the addition of a dropout layer after the graph convolutional and pooling layers. Dropout is a powerful regularization tool for neural networks and prevents the neurons from co-adapting, forcing the network to learn general characteristics from the data and reducing overfitting [24].

3.5.3.2 Batch Normalization

The second improvement made was the addition of batch normalization [8] to the fully-connected layers in the network. This was done because adding even a single fully-connected layer after the graph convolutional and pooling layers was consistently preventing the network from converging and learning anything useful when the vocabulary size was considerably large (tested with 10,000 words).

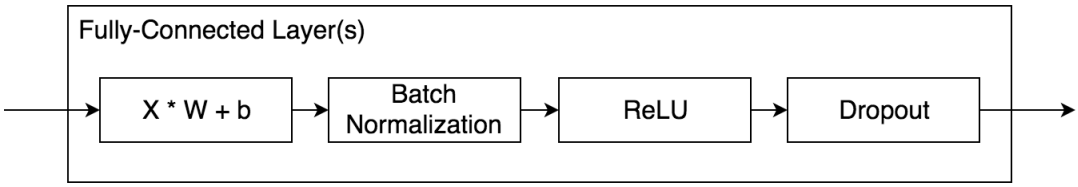

Figure 15: Modified Fully-Connected Layers for GCNNs

The additional dropout layer from 3.5.3.1 became even more important after batch normalization was added as the network immediately began overfitting the data when the dropout rate was 0 (i.e. dropout keep probability of 1.0) and a fully-connected layer was added (GC15_5-FC100). The results of the different test accuracies for GC15_5-FC100 on 20 Newsgroups with a vocabulary size of 10,000 can be found in Table 1.

Dropout Keep Probability	Batch Normalization	Max. Test Accuracy
0.5	Y	69.84
1.0	Y	10.78
0.5	N	5.37
1.0	N	5.37

Table 1: Effect of Dropout & Batch Norm on GC15_5-FC100 (Chebyshev)

3.5.3.3 Other Improvements

tf-idf document vectors are used as opposed to count vectors used in the implementation by M. Defferrard et al. as they better represent data for text classification (further elaborated on in Section 4.1.1).

Additionally, all the weights in our network are initialized using the Xavier initialization as opposed to being sampled from a truncated normal distribution with a standard deviation of 0.1 as in the original implementation, which is empirically shown to be better (~1% increase in performance) when the network has no fully-connected layers and have little to no effect when the network has fully-connected layers, as can be seen in Table 2.

Model Architecture	Weight Initialization	Max. Test Accuracy
GC5_32	Uniform Xavier	71.65
GC5_32	Truncated Normal	70.78
GC15_5-FC100	Uniform Xavier	69.84
GC15_5-FC100	Truncated Normal	70.26

Table 2: Effect of Weight Initialization (Chebyshev, 20 Newsgroups, |V|= 10,000)

Overall, our implementation has a maximum test accuracy of 71.65% as opposed to 68.26% quoted in [6] when trained with the same model architecture (i.e. GC5_32) and identical hyperparameters (16-NN feature graph, Adam optimizer, initial learning rate of 0.001) on the 20 Newsgroups dataset. Our implementation also has a significantly higher test accuracy on the RCV1 dataset even with a simpler model architecture than the results quoted in [7], though it should be noted that there is a slight variation in the dataset split and we use different hyperparameters to train our model.

Dataset (Vocabulary Size)	Model Architecture (Filter)	Test Accuracy
20 Newsgroups (10,000)	GC5_32 (Chebyshev) [6]	68.26 [6]
	GC5_32 (Chebyshev)	71.65
RCV1 (2,000)	GC[?]_4-P4-FC1000 (Spline) [7] [3]	69.41 [7]
	GC5_32 (Chebyshev)	89.68

Table 3: Comparison of GCNN Test Accuracies with Existing Literature

[3] [?] is used because the filter size (as used in our implementation) is not clearly defined in the paper.

4. Experiments

4.1 Preprocessing

In order to keep the results of our experiments consistent and focused on the models as opposed to the representation, the preprocessing of the raw text datasets is kept simple and constant throughout all the runs.

First, the raw text is cleaned using simple regular expressions that delete any characters that don't match `/[^A-Za-z0-9(),!?'$]/`, replace any sequence of one or more digits with "NUM" and separate punctuation & clitics from their respective adjacent words. The entire text is then converted into lowercase for consistent frequency counts.

Next, the cleaned text is tokenized and vectorized to count vectors using scikit-learn's CountVectorizer, which also removes any English stop words. Once the count vectors are derived, the vocabulary is reduced by keeping only the top N most frequent words. Due to the reduction in vocabulary size, some samples in the dataset may become too short (due to excessive use of infrequent words) and are subsequently removed as outliers.

Finally, the input data for the model is generated. For YK-CNN & FC-CNN, the cleaned text is tokenized and converted into sequences of word IDs (extracted from the vocabulary) after which it is padded or truncated to fit the chosen sequence length. For the other models (MLP, GCNN, Linear SVC & Multinomial NB), the count vectors are transformed into normalized tf-idf vectors using scikit-learn's TfidfTransformer.

4.1.1 count vs. tf-idf

Throughout the experiments mentioned in this report, I have represented the documents as normalized tf-idf (term frequency – inverse document frequency) document vectors. tf-idf document vectors scale down the impact of tokens (i.e. words) that occur very frequently throughout a corpus and hence, allow unique keywords to have significantly greater weight, which is essential for text classification. Unlike count vectors, which simply take into account the frequencies of the tokens (i.e. tf), tf-idf multiplies the token's frequency with the token's

inverse document frequency. This is shown in Figure 16, where c is the corpus, t is the token, n is the total number of documents in the corpus and $\mathrm{df}(c,t)$ or document frequency is the number of documents that contain the token.

$$\mathrm{tfidf}(c,t) = \mathrm{tf}(t) \times \mathrm{idf}(c,t)$$
$$\mathrm{idf}(c,t) = \log\left[\frac{1+n}{1+\mathrm{df}(c,t)}\right] + 1$$

Figure 16: tf-idf Equations

In order to prove tf-idf document vectors perform better than count vectors for the task of text categorization, we ran experiments on the benchmark models and ChebNet using the 20 Newsgroups dataset with a vocabulary size of 10,000. The results in Table 4 empirically show that tf-idf vectors consistently outperform count vectors by approximately 3% – 8% for all of the models.

MODEL NAME	TEST ACCURACY	
	Count Vectors	tf-idf Vectors
Linear SVC	60.54	69.72
Multinomial NB	64.52	69.54
Softmax	68.36	70.93
MLP (FC360)	70.80	73.08
Graph CNN (Chebyshev, GC5_32)	67.95	71.65

Table 4: Count Vectors vs. tf-idf Vectors for 20 Newsgroups (|V| = 10,000)

4.2 Rotten Tomatoes Sentence Polarity

The Rotten Tomatoes sentence polarity (RT Polarity) dataset is Pang and Lee's movie review sentiment polarity dataset [17] [18], which consists of 5,331 positive and 5,331 negative movie reviews. For all the models, the data is shuffled (with a fixed random seed) and 10% of the dataset is used as the test set (1,066 documents), which is identical to the split used in [10] and [27].

In the following experiments, the original vocabulary size of 18,121 words is shrunk to 5,000 by selecting the most frequent words. From the reduced vocabulary, 4,817 word embeddings

are retrieved from Google's word2vec and the rest are randomly initialized. The traditional CNN models (FC-CNN & YK-CNN) employ a maximum sequence length of 56 words (maximum sentence length in dataset) and the Graph CNN models use a 16-NN graph constructed from the vocabulary's word vectors. All the neural network models were trained for 200 epochs.

MODEL NAME	ARCHITECTURE	ACCURACY
Linear SVC	-	75.33
Multinomial NB	-	77.20
Softmax	-	77.67
MLP	FC100	77.49
	FC1000	77.11
FC-CNN	CL5_128-P5-CL5_128-P5-FC128	76.92
YK-CNN	CL3_128-CL4_128-CL5_128	77.20
GCNN (Chebyshev)	GC5_32	77.86
GCNN (Spline)	GC5_32	78.24
GCNN (Fourier)	GC5000_32	77.20

Table 5: Test Accuracies of Models on RT Polarity

Model	Test Accuracy	No. of Parameters	Training Time
Linear SVC	75.33	N/A	0.06s [4]
Multinomial NB	77.20	N/A	0.01s [4]
Softmax	77.67	10,002	1.07s
FC100	77.49	500,302	3.81s
FC1000	77.11	5,003,002	19.66s
GC5_32 (Chebyshev)	77.86	320,194	5.16s
GC5_32 (Spline)	78.24	320,194	7.94s
GC5000_32 (Fourier)	77.20	480,034	10.28s

Table 6: No. of Parameters & Approx. Training Time per Epoch for RT Polarity

4.3 20 Newsgroups

The 20 Newsgroups dataset [21] comprises of around 18,000 newsgroups posts on 20 topics split into train (10,116 documents) and test (7,068 documents) subsets based upon whether the messages were posted before or after a particular date. Some of the topics are very

[4] The training time quoted for the baseline models is the **total** training time.

closely related to each other (e.g. rec.autos & rec.motorcycles) while some topics are highly unrelated (e.g. misc.forsale & sci.space). The headers, footers and quotes (newsgroup-related metadata) of each document were removed to make the performance metrics more realistic.

4.3.1 Hyperparameter Optimization

In order to perform an unbiased comparison of ChebNet to the benchmark models (Linear SVC, Multinomial NB & MLP), we tuned the hyperparameters of all four models using a validation set (10% of train set, ~1,000 documents) with a vocabulary size of 10,000.

4.3.1.1 Linear SVC

The only hyperparameter to be tuned for Linear SVC is the penalty parameter C of the error term, which was tested in the range of 0.1 to 50.0 in intervals of 0.1. The maximum validation accuracy of 78.34% was obtained with a C value of 41.0.

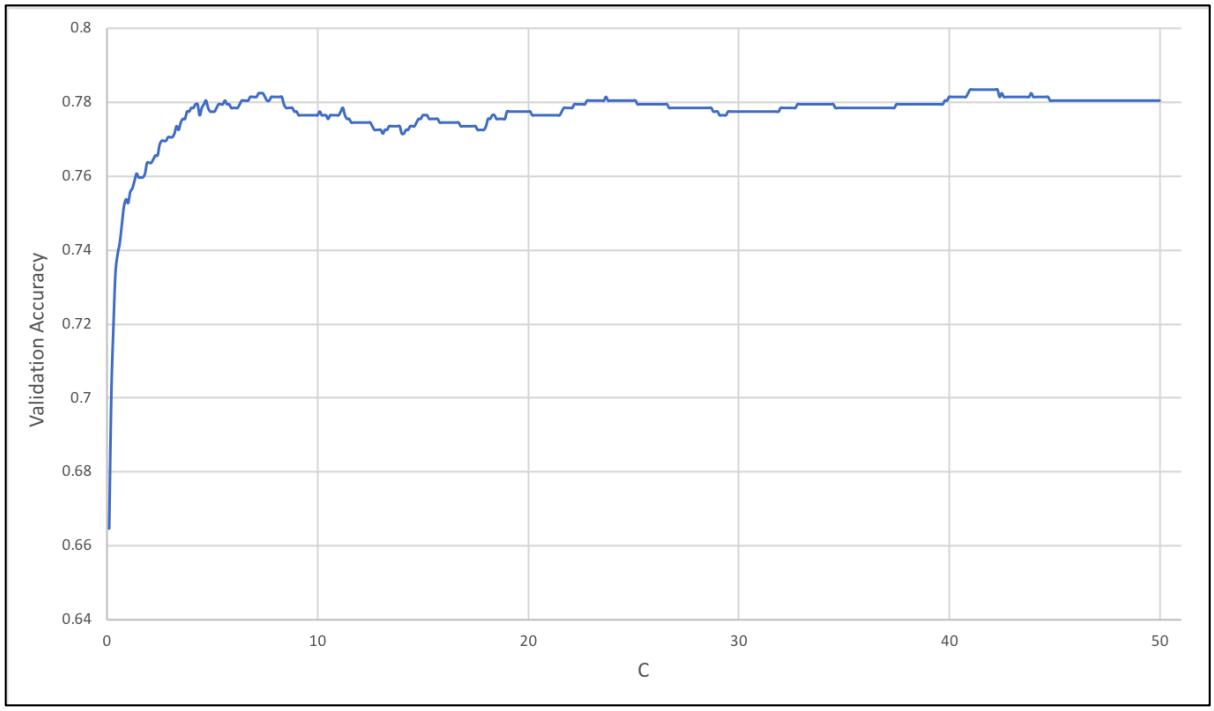

Figure 17: Penalty Parameter C vs. Validation Accuracy for Linear SVC on 20 Newsgroups

4.3.1.2 Multinomial NB

The only hyperparameter to be tuned for the Multinomial Naïve Bayes Classifier is the additive smoothing parameter α, which accounts for features that may not be present in

training samples. The values tested were between 1e-10 (minimum possible value) and 1e-01 for Lidstone smoothing and 1.0 for Laplace smoothing. The maximum validation accuracy of 75.57% was obtained when α was equal to 1e-03.

Alpha (α)	Validation Accuracy
1e-10	68.84
1e-09	69.44
1e-08	70.03
1e-07	70.92
1e-06	72.21
1e-05	73.29
1e-04	74.28
1e-03	**75.57**
1e-02	74.97
1e-01	71.22
1.0	56.18

Table 7: Tuning Alpha for Multinomial NB on 20 Newsgroups

4.3.1.2 Multilayer Perceptron

The MLP model was tuned by experimenting with the number of neurons in a single hidden layer and testing a couple architectures with two hidden layers. The accuracies listed below in Table 8 show the optimal architecture to be FC100. All models were trained for 100 epochs.

Architecture	Validation Accuracy
Softmax	76.95
FC100	**80.12**
FC250	80.02
FC500	80.02
FC1000	79.62
FC2000	79.82
FC2000-FC500	78.73
FC2000-FC1000	79.03

Table 8: Validation Accuracies of MLP on 20 Newsgroups

4.3.1.3 Graph CNN (Chebyshev)

The hyperparameters tuned for ChebNet were the number of edges in the feature graph and the filter size & number of features for a single graph convolutional layer. Through previous experiments, we observed no increase in performance with the addition of graph pooling, fully-connected layers or multiple graph convolutional layers (further discussed in Section 5.4). A grid search was performed on the values listed in Table 9 and the model with the highest validation accuracy of 79.92% was G2_32 with an 8-NN feature graph. The complete results can be found in Appendix A.1.

Hyperparameter	Values Tested
Number of Edges	4, **8**, 16
Filter Size (or Support Size)	**2**, 4, 5
Number of Features	8, 16, **32**

Table 9: Hyperparameter Values for Grid Search on Graph CNN (Chebyshev)

4.3.2 Final Results

The original vocabulary size of 72,366 words is shrunk to 10,000 by selecting the top words by frequency. From the reduced vocabulary, 9,164 word embeddings are retrieved from Google's word2vec while 836 word embeddings are randomly initialized. The traditional CNN models (FC-CNN & YK-CNN) employ a sequence length of 1,000 words and the Graph CNN models use an 8-NN feature graph. All neural network models were trained for 200 epochs.

MODEL NAME	ARCHITECTURE	ACCURACY
Linear SVC	[C = 41.0]	69.54
Multinomial NB	[α = 0.001]	68.45
Softmax	-	70.93
MLP	FC100	72.92
FC-CNN	CL5_128-P5-CL5_128-P5-CL5_128-P5-FC128	63.20
YK-CNN	CL3_128-CL4_128-CL5_128	71.39
GCNN (Chebyshev)	GC2_32	71.51
GCNN (Spline)	GC2_32	71.35
GCNN (Fourier)	GC10000_32	70.46

Table 10: Test Accuracies of Models on 20 Newsgroups

Model	Test Accuracy	No. of Parameters	Training Time
Linear SVC	69.54	N/A	10.84s [5]
Multinomial NB	68.45	N/A	0.05s [5]
Softmax	70.93	200,020	3.82s
FC100	72.92	1,002,120	7.71s
GC2_32 (Chebyshev)	71.51	6,400,116	41.38s
GC2_32 (Spline)	71.35	6,400,116	65.86s
GC10000_32 (Fourier)	70.46	6,720,052	84.81s

Table 11: No. of Parameters & Approx. Training Time per Epoch for 20 Newsgroups

4.4 Reuters Corpus Volume 1

Reuters Corpus Volume I (RCV1) is an archive of over 800,000 manually categorized newswire stories made available by Reuters Ltd. for research purposes [14]. The original dataset was retrieved as a large set of raw XML files, from which the documents and labels were extracted with the help of the publicly released online appendices of the original paper, which contain text files that list the IDs of the documents to be considered in the RCV1-v2 version of the dataset and the mappings of documents to assigned classes.

The original documents pose a multiclass problem with 103 classes where a document can belong to more than one class and the classes are arranged in a hierarchical manner. In order to convert the dataset to a single class classification problem, only the 55 second-level classes are considered, out of which one class is removed as it covers nearly 25% of the data and a couple classes are removed because they have less than 10 documents (after documents with multiple classes are removed from the dataset). Thus, the final dataset contains 420,282 documents from 52 unique classes. This is similar to the split used in [7] and originally used by Srivastava in [24].

The resulting dataset is chronologically split equally into train and test sets with 210,141 documents each. A chronological split is chosen instead of shuffling the dataset and splitting randomly because a majority of text classification tasks require training on currently available material and then applying the system to documents that are received later. Furthermore, a

[5] The training time quoted for the baseline models is the **total** training time.

chronological split decreases the probability of duplicate or near-duplicate documents such as news articles about the same event to be split between the train and test subsets, providing a more realistic performance measure.

4.4.1 Hyperparameter Optimization

In order to perform an unbiased comparison of ChebNet to the benchmark models (Linear SVC, Multinomial NB & MLP), we tuned the hyperparameters of all four models using a validation set (10% of train set, ~20,000 documents) with a vocabulary size of 10,000.

4.4.1.1 Linear SVC

The only hyperparameter to be tuned for Linear SVC is the penalty parameter C of the error term, which was tested in the range of 0.1 to 50.0 in intervals of 0.1. The maximum validation accuracy of 90.79% was obtained with a C value of 14.3.

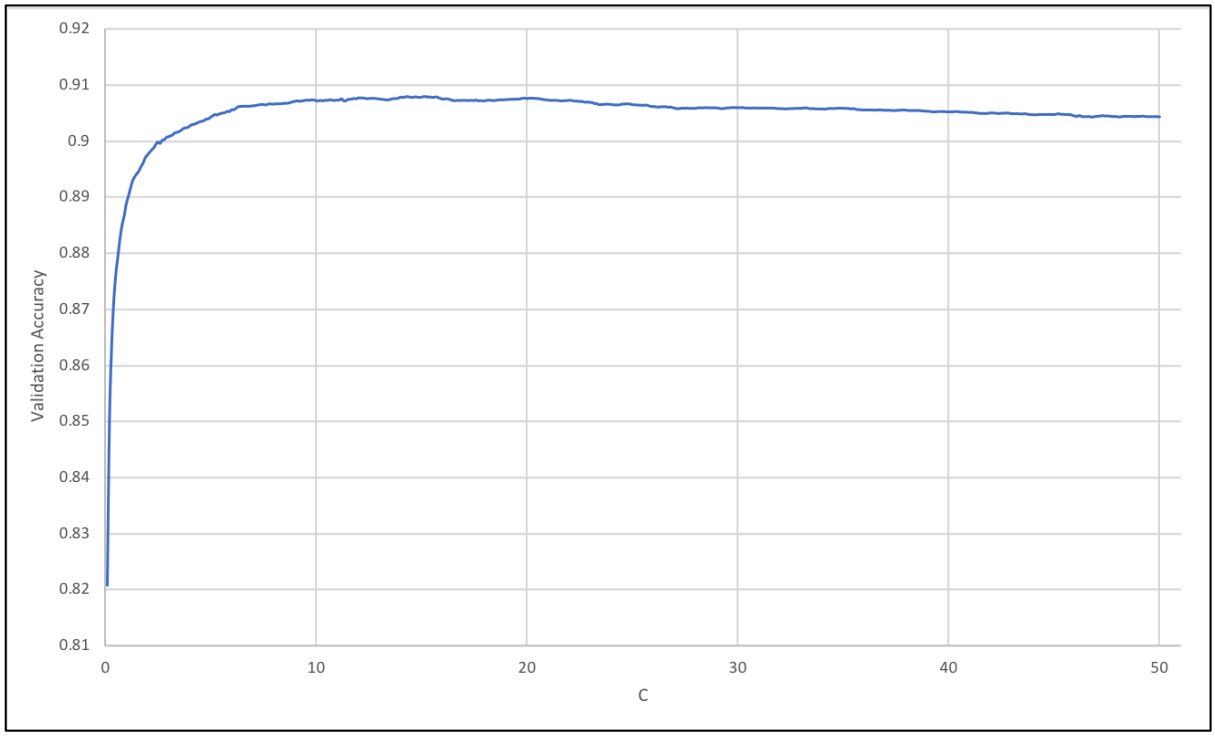

Figure 18: Penalty Parameter C vs. Validation Accuracy for Linear SVC on RCV1

4.4.1.2 Multinomial NB

The only hyperparameter to be tuned for the Multinomial Naïve Bayes Classifier is the additive smoothing parameter α, which accounts for features that may not be present in training samples. The values tested were between 1e-10 (minimum possible value) and 1e-01 for Lidstone smoothing and 1.0 for Laplace smoothing. The maximum validation accuracy of 67.48% was obtained when α was equal to 1e-10.

Alpha (α)	Validation Accuracy
1e-10	**67.48**
1e-09	67.46
1e-08	67.43
1e-07	67.36
1e-06	67.34
1e-05	67.22
1e-04	67.15
1e-03	66.83
1e-02	66.27
1e-01	63.38
1.0	52.99

Table 12: Tuning Alpha for Multinomial NB on RCV1

4.4.1.2 Multilayer Perceptron

The MLP model was tuned by experimenting with the number of neurons in a single hidden layer and testing a couple architectures with two hidden layers. The accuracies listed below in Table 13 show the optimal architecture to be FC500. All models were trained for 50 epochs.

Architecture	Validation Accuracy
Softmax	90.41
FC100	91.01
FC250	91.02
FC500	**91.05**
FC1000	90.93
FC2000	90.96
FC2000-FC500	90.52
FC2000-FC1000	90.57

Table 13: Validation Accuracies of MLP on RCV1

4.4.1.3 Graph CNN (Chebyshev)

The hyperparameters tuned for ChebNet were the number of edges in the feature graph and the filter size & number of features for a single graph convolutional layer. Through previous experiments, we observed no increase in performance with the addition of graph pooling, fully-connected layers or multiple graph convolutional layers (further discussed in Section 5.4). A grid search was performed on the values listed in Table 14 and the model with the highest validation accuracy of 91.04% was GC4_8 with a 16-NN feature graph. The complete results can be found in Appendix A.2.

Hyperparameter	Values Tested
Number of Edges	4, 8, **16**
Filter Size (or Support Size)	2, **4**, 5
Number of Features	**8**, 16, 32

Table 14: Hyperparameter Values for Grid Search on Graph CNN (Chebyshev)

Additionally, a search was performed on the dropout values for the chosen hyperparameters from the grid search above. The optimal dropout keep probability was 0.2 with a resulting validation accuracy of 91.19%.

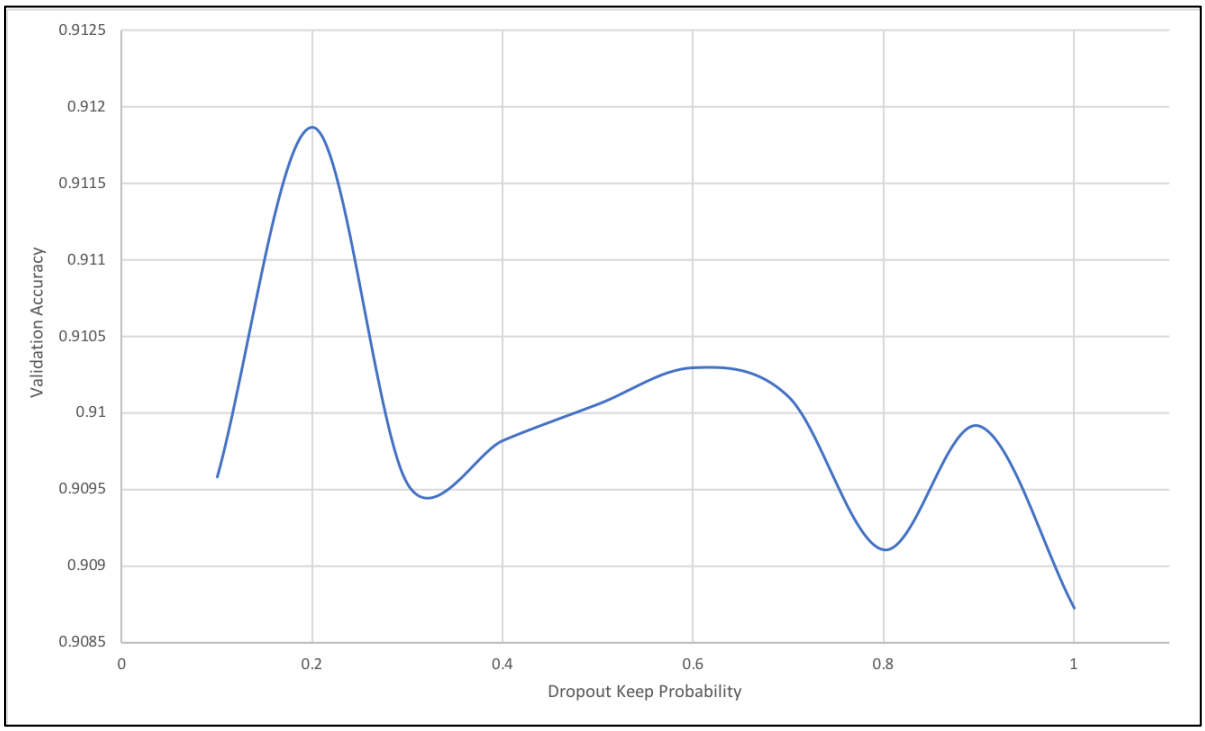

Figure 19: Dropout vs. Validation Accuracy for GC4_8 on RCV1

4.4.2 Final Results

The original vocabulary size of 270,589 words is shrunk to 10,000 by selecting the top words by frequency. From the reduced vocabulary, 8,870 word embeddings are retrieved from Google's word2vec while 1,130 word embeddings are randomly initialized. The traditional CNN models (FC-CNN & YK-CNN) employ a maximum sequence length of 1,000 words and the Graph CNN models use a 16-NN graph constructed from the embeddings of the vocabulary. All neural network models were trained for 50 epochs.

MODEL NAME	ARCHITECTURE	ACCURACY
Linear SVC	[C = 14.3]	91.19
Multinomial NB	[α = 1e-10]	68.44
Softmax	-	90.86
MLP	FC500	91.44
FC-CNN	CL5_128-P5-CL5_128-P5-CL5_128-P5-FC128	89.94
YK-CNN	CL3_128-CL4_128-CL5_128	90.75
GCNN (Chebyshev)	GC4_8 [Dropout Keep = 0.2]	91.33
GCNN (Spline)	GC4_8	91.29
GCNN (Fourier)	GC10000_8	91.35

Table 15: Test Accuracies of Models on RCV1

Model	Test Accuracy	No. of Parameters	Training Time
Linear SVC	91.19	N/A	305.30s [6]
Multinomial NB	68.44	N/A	0.99s [6]
Softmax	90.86	520,052	445.10s
FC500	91.44	5,026,552	860.00s
GC4_8 (Chebyshev)	91.33	4,160,092	1769.18s
GC4_8 (Spline)	91.29	4,160,092	2567.82s
GC10000_8 (Fourier)	91.35	4,240,060	2997.90s

Table 16: No. of Parameters & Approx. Training Time per Epoch for RCV1

[6] The training time quoted for the baseline models is the **total** training time.

5. Discussion of Results

5.1 Relative Performance of Models

We see from the results of the experiments on RT Polarity (Table 5), 20 Newsgroups (Table 10) and RCV1 (Table 15) that all the deep models tested in this report with the exception of F. Chollet's CNN consistently outperform or at least perform as well as the two baseline models (Linear SVC & Multinomial NB). However, the CNNs and GCNNs tend to fall short of the simpler MLP models, which have significantly fewer parameters to train (as can be seen in Table 6, Table 11 and Table 16). This is in stark contrast to the results reported in [6] where GC5_32 (Chebyshev) is said to outperform multilayer perceptrons (specifically, FC2500 and FC2500-500). The difference in results can be attributed to the use of tf-idf vectors as opposed to count vectors in our implementation and the much smaller hidden layers tested in our experiments. This is also in line with the results obtained during hyperparameter optimization in Table 8 and Table 13, where more complex MLP models resulted in lower validation accuracies on 20 Newsgroups and RCV1.

The similarity in performance (difference of less than 3%) for most of the models on all three datasets suggests that either the current data representation has been saturated or we still have not found the correct architecture needed for the task of text categorization, especially one that can potentially have a test accuracy close to 100%. This is especially true for the deep models, which take a considerably longer time to train and are much more complex than their simpler counterparts but fail to produce significantly higher test accuracies.

5.2 Relative Performance on Datasets

Looking at the performance of the models across the three datasets, we see that our models do exceedingly well on the RCV1 dataset as compared to the other two datasets. This is probably because of two important reasons. The first is the abundance of high quality training data in RCV1, which consists of professionally written news articles whereas the other two datasets contain user-submitted online texts. The second is the difference in the themes of the datasets and the resulting separation of classes. In RT Polarity, for example, the sentences

used to positively or negatively describe a movie may be quite similar and don't always use easily identifiable keywords. In contrast, samples from RCV1 (which is a news corpus) are very topical and use a high number of class-specific keywords, which are further amplified due to the use of tf-idf document vectors.

An interesting anomaly in the experimental results for RCV1 is the considerably lower test accuracy of the Multinomial Naïve Bayes Classifier (68.44%) as compared to the other models (~90%). This is probably due to the fact that the RCV1 dataset is particularly unbalanced, having 31,305 samples in its largest class and only 22 samples in its smallest, which is not ideal for a naïve Bayes model as the resulting probabilities may be skewed.

5.3 Effect of Graph CNN Filter

When comparing the different graph convolutional neural networks tested in this report, we see that using different filters has a notable effect on the final accuracies of the resulting models. While the Chebyshev and Spline filters tend to result in models with similar test accuracies, the non-parametric Fourier filter fails to perform as well despite being more computationally complex and inefficient in comparison. This observation is identical to the one reported in [6]. Additionally, from the results in Table 6, Table 11 and Table 16, we see that the Chebyshev filter is substantially more computationally efficient than the other two filters, however, it is important to reemphasize that neither of the GCNNs are as efficient as the MLP models with similar (or greater) test accuracies.

5.4 Graph CNN (Chebyshev)

Similar to the observations made about MLP, we see that the performance of ChebNet does not improve with increasing model complexity in Table 17 (next page). The experiments show that increasing the number of feature maps, incorporating pooling or the addition of fully-connected layers does not improve test accuracy and may also degrade the performance of the model. We also see that a smaller support size generally improves model performance, however, it is important to note that using a support size of 1 prevented the model from learning anything useful. The network also fails to learn anything useful when multiple graph

convolutional and pooling layers are added (>2 for 20 Newsgroups). This is probably due to the lack of hierarchical information in text data and a drastic loss of information with the application of consecutive convolution and pooling operations.

Architecture	Test Accuracy
GC2_32 (Highest Validation Accuracy)	71.51
GC5_8 (Highest Test Accuracy)	71.76
GC5_32 (Architecture in [6])	71.65
GC2_4	71.62
GC2_4-P4	69.21
GC2_4-P4-FC1000	62.99
GC2_4-GC2_8	70.63
GC15_5	70.45
GC15_5-FC100	69.84

Table 17: Test Accuracies of GCNN Architectures (20 Newsgroups, |V| = 10,000)

6. Client Application

A simple client application was built for the models trained on the RCV1 dataset and open-sourced on GitHub.[7] The application can be used to categorize news articles provided they match at least one of the RCV1 classes. It is built on top of Flask, a Python micro-framework, on server-side and uses vanilla HTML, CSS and JavaScript on the client-side. A high-level architecture and overview of the flow of the client application is shown in Figure 20. Screen captures of the client application are shown in Figure 21 and Figure 22 (next page).

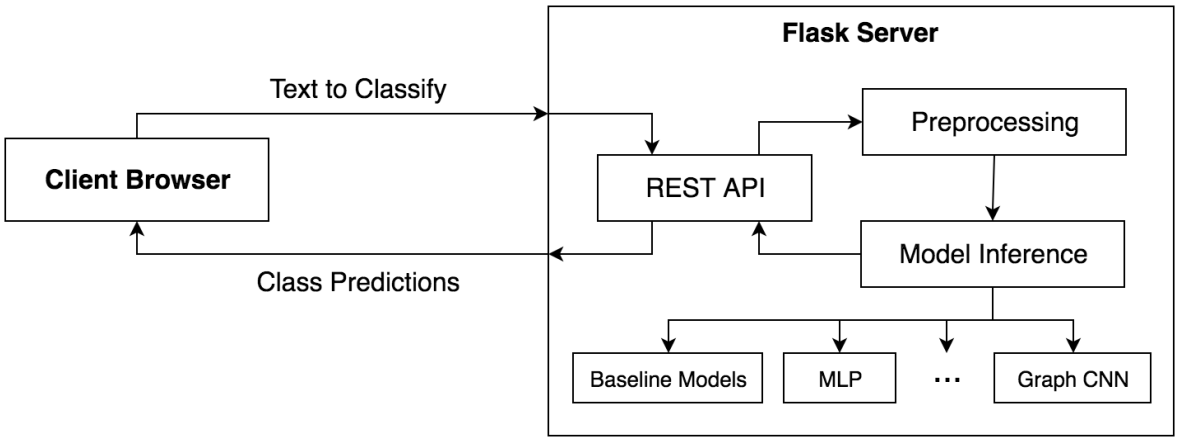

Figure 20: Architecture of Client Application

Figure 21: Screenshot of Client Web App

[7] https://github.com/SuyashLakhotia/TextCategorization-Demo

Figure 22: Classifying a News Article from Bloomberg

7. Conclusion

In this report, we have presented three major deep learning models for text categorization and compared their performance on three benchmark datasets – RT Polarity, 20 Newsgroups and RCV1 after improving on their initial implementations.

We observe that graph convolutional neural networks achieve a similar performance as standard convolutional neural networks when using the same vocabulary size and pre-trained word embeddings. However, while GCNNs and CNNs outperform the baseline models (which take an extremely short time to train), they are only as good or sometimes slightly worse than simpler multilayer perceptrons with a single hidden layer that have significantly fewer trainable parameters.

Nonetheless, the experiments in this report prove the ability of GCNNs to extract meaningful local features through graph convolutional layers and showcase the effectiveness of the Chebyshev filter introduced in [6], which performs better than the other filters tested and is also much more computationally efficient.

The results of the experiments also shed light on the powerful capability of tf-idf document vectors in capturing the necessary information needed for text classification, producing simple models with impressive test accuracies.

8. Future Work

While this project has been successful in comparing Graph CNNs to other machine learning models for the purpose of text classification, below are several suggestions for future research in this field:

1. **Advanced Data Exploration:** Further data exploration is required to understand the distribution of the representations versus the labels. It is important to recognize if the models are performing well solely on well-separated classes or are able to differentiate between similar classes as well.

2. **Different Data Representations:** As we saw in our experiments, Graph CNNs are not able to completely take advantage of the combination of information they possess (i.e. tf-idf document vectors and word embeddings) when compared to the other models. Perhaps more expressive features are necessary that are hierarchical in nature (for convolutional operations) which would allow GCNNs to truly benefit from its capabilities. Furthermore, vocabulary indexes in the document vectors can be re-arranged, possibly on a scale of sentiment or another semantic attribute, to encode features in local neighborhoods.

3. **Hyperparameter Tuning / Architecture Experiments:** Additional exhaustive experiments are needed to test different hyperparameters and model architectures. Due to limited time and resources, we chose only a subset of the possible hyperparameter values in our experiments and trained without cross-validation.

4. **Additional Models:** There were numerous deep models for text categorization that were excluded from this project. This includes LSTMs & bidirectional LSTMs, one-hot-CNNs & bag-of-words-CNNs [9] and character-level-CNNs [28]. These models should be compared as well in future experiments in order to gain a broader perspective. Additionally, it may be beneficial to investigate the effect of using graph convolutional neural networks purely as feature generators for simpler models such as Linear SVC.

5. **Additional Datasets:** While the datasets chosen from this project dealt with diverse themes (review polarity, forum posts, news articles), the size of the datasets were not similar and may have skewed certain results due to lack of training data. Thus, future

experiments would require datasets that deal with diverse themes and have similar quality and quantity of training data to better compare the relative performance of the models on different datasets.

6. **Better Preprocessing:** While the preprocessing of the datasets in this project was kept extremely naïve for research purposes, it is important to preprocess the text data using more robust methods for actual production models. For example, better tokenization and entity recognition would help differentiate between "…several deep wells…" and "…said Mr. Wells…".

References

[1] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015. Software available from tensorflow.org.

[2] D. Britz. Implementing a CNN for Text Classification in TensorFlow. *WildML*, 2015. Available at: http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/.

[3] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral Networks and Deep Locally Connected Networks on Graphs. *arXiv:1312.6203*, 2014.

[4] F. Chollet. Using pre-trained word embeddings in a Keras model. *The Keras Blog*, 2016. Available at: https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html.

[5] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural Language Processing (almost) from Scratch. *arXiv:1103.0398*, 2011.

[6] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. *arXiv:1606.09375*, 2017.

[7] M. Henaff, J. Bruna, and Y. LeCun. Deep Convolutional Networks on Graph-Structured Data. *arXiv:1506.05163*, 2015.

[8] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167*, 2015.

[9] R. Johnson and T. Zhang. Effective Use of Word Order for Text Categorization with Convolutional Neural Networks. *arXiv:1412.1058*, 2015.

[10] Y. Kim. Convolutional Neural Networks for Sentence Classification. *arXiv:1408.5882*, 2014.

[11] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980*, 2014.

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE,* 86(11), pp. 2278-2324, 1998.

[13] Y. LeCun, Y. Bengio, and G. Hinton. Deep Learning. In *Nature*, 521(7553), pp. 436-444, 2015.

[14] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A New Benchmark Collection for Text Categorization Research. In *Journal of Machine Learning Research 5*, pp. 361-397, 2004.

[15] T. Mikolov et al. word2vec: Pre-trained word and phrase vectors. 2013. Available at: https://code.google.com/archive/p/word2vec/

[16] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed Representations of Words and Phrases and their Compositionality. *arXiv:1310.4546*, 2013.

[17] B. Pang and L. Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of ACL 2004*, 2004.

[18] B. Pang and L. Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of ACL 2005*, 2005.

[19] F. Pedregosa et al. Scikit-learn: Machine Learning in Python. In *Journal of Machine Learning Research 12*, pp. 2825-2830, 2011. Software available from scikit-learn.org.

[20] J. Pennington, R. Socher, and C. D. Manning. GloVe: Global Vectors for Word Representation. 2014.

[21] J. Rennie. 20 Newsgroups. Available at: http://qwone.com/~jason/20Newsgroups/.

[22] F. Sebastiani. Machine Learning in Automated Text Categorization. In *ACM Computing Surveys Vol. 34 No. 1*, pp. 1-47, 2002.

[23] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Empirical Methods on Natural Language Processing*, 2013.

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salahkhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. In *Journal of Machine Learning Research 15*, pp. 1929-1958, 2014.

[25] S. Walt, S. Chris Colbert, and G. Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation. In *Computing in Science & Engineering Vol. 13*, pp. 22-30, 2011. Software available from numpy.org.

[26] H. Zhang. The Optimality of Naïve Bayes. In *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference*, 2004.

[27] Y. Zhang and B. C. Wallace. A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. *arXiv:1510.03820*, 2016.

[28] X. Zhang, J. Zhao and Y. LeCun. Character-level Convolutional Networks for Text Classification. *arXiv:1509.01626*, 2016.

Appendix A: Hyperparameter Grid Search

A.1 Graph CNN on 20 Newsgroups

Below are the results of a grid search performed for Graph CNN (Chebyshev) on the 20 Newsgroups dataset with a vocabulary size of 10,000. Each model was trained for 100 epochs with an initial learning rate of 0.001 and dropout of 0.5.

No. of Edges	Filter Size	No. of Features	Validation Accuracy
4	2	8	0.7823936696340257
4	2	16	0.7883283877349159
4	2	32	0.7893175074183977
4	4	8	0.7843719090009891
4	4	16	0.7952522255192879
4	4	32	0.7912957467853611
4	5	8	0.7794263105835806
4	5	16	0.7873392680514342
4	5	32	0.7873392680514342
8	2	8	0.7942631058358062
8	2	16	0.7942631058358062
8	2	32	0.7992087042532147
8	4	8	0.7942631058358062
8	4	16	0.7932739861523245
8	4	32	0.7912957467853611
8	5	8	0.7893175074183977
8	5	16	0.7883283877349159
8	5	32	0.7883283877349159
16	2	8	0.7952522255192879
16	2	16	0.7972304648862513
16	2	32	0.7962413452027696
16	4	8	0.7893175074183977

16	4	16	0.7903066271018794
16	4	32	0.7893175074183977
16	5	8	0.7893175074183977
16	5	16	0.7893175074183977
16	5	32	0.7903066271018794

A.2 Graph CNN on RCV1

Below are the results of a grid search performed for Graph CNN (Chebyshev) on the RCV1 dataset with a vocabulary size of 10,000. Each model was trained for 20 epochs with an initial learning rate of 0.001 and dropout of 0.5.

No. of Edges	Filter Size	No. of Features	Validation Accuracy
4	2	8	0.90768059388978772
4	2	16	0.90853716569905774
4	2	32	0.90972684876748833
4	4	8	0.90948891215380223
4	4	16	0.90825164176263440
4	4	32	0.90901303892643004
4	5	8	0.90967926144475109
4	5	16	0.90934615018559051
4	5	32	0.90777576853526221
8	2	8	0.91015513467212339
8	2	16	0.90953649947653947
8	2	32	0.90958408679927671
8	4	8	0.90982202341296281
8	4	16	0.90887027695821831
8	4	32	0.90877510231274383
8	5	8	0.90977443609022557
8	5	16	0.90963167412201384
8	5	32	0.90749024459883887

16	2	8	0.89492719139621202
16	2	16	0.90753783192157611
16	2	32	0.90896545160369280
16	4	8	0.91039307128580949
16	4	16	0.90934615018559051
16	4	32	0.90853716569905774
16	5	8	0.91025030931759776
16	5	16	0.90991719805843718
16	5	32	0.90806129247168554