MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY BHOPAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING PROJECT

ON

Text Classification using Convolutional Neural Networks

SUBMITTED IN PARTIAL FULFILLMENT FOR THE DEGREE OF BACHELOR OF TECHNOLOGY

SUBMITTED BY:

PRAKHAR AGARWAL	**141112023**
R. VINAY	**141112082**
DEVKARAN NIRWAN	**141112087**
PUSHPENDRA SINGH	**141112046**

UNDER THE GUIDANCE OF

DR. SWETA JAIN

SESSION 2016-17

MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY BHOPAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that **Prakhar Agarwal, R. Vinay, Devkaran Nirwan and Pushpendra Singh** are students of B.Tech 3rd Year (Computer Science & Engineering), have successfully completed their project "**TEXT CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS**" in partial fulfillment of their minor project in Computer Science & Engineering.

Dr. Sweta Jain	Dr. Sanyam Shukla
(Project Guide)	(Project Coordinator)

MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY BHOPAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, hereby, declare that the following report which is being presented in the Minor Project Documentation entitled "TEXT CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORK" is the partial fulfillment of the requirements of the third year (sixth semester) Minor Project in the field of Computer Science and Engineering. It is an authentic documentation of our own original work carried out under the able guidance of Dr. Sweta Jain and the dedicated co-ordination of Prof. Sanyam Shukla. The work has been carried out entirely at Maulana Azad National Institute of Technology, Bhopal. The following project and its report, in part or whole, has not been presented or submitted by us for any purpose in any other institute or organization.

We, hereby, declare that the facts mentioned above are true to the best of our knowledge. In case of any unlikely discrepancy that may possibly occur, we will be the ones to take responsibility.

PRAKHAR AGARWAL	**141112023**
R. VINAY	**141112082**
DEVKARAN NIRWAN	**141112087**
PUSHPENDRA SINGH	**141112046**

ACKNOWLEDGEMENT

With due respect, we express our deep sense of gratitude to our respected Dr. Sweta Jain, for her valuable help and guidance. We are thankful for the encouragement that she has given us in completing this project successfully. Her rigorous evaluation and constructive criticism was of great assistance.

It is imperative for us to mention the fact that this minor project could not have been accomplished without the periodic suggestions and advice of our project co-ordinator Prof. Sanyam Shukla.

We are also grateful to our respected director Dr. Narendra S Chaudhary for permitting us to utilize all the necessary facilities of the college.

Needless to mention is the additional help and support extended by our respected HOD, Dr. R. K. Pateriya, in allowing us to use the departmental laboratories and other services.

We are also thankful to all the other faculty, staff members and laboratory attendants of our department for their kind co-operation and help.

Last but certainly not the least; we would like to express our deep appreciation towards our family members and batch mates for providing the much-needed support and encouragement.

Contents

List of Figures

List of Tables

Abstract

Classification is an important task in Machine Learning, where it is identified to which of a set of categories a new observation belongs; making decisions on the basis of training data containing observations whose category membership is known. Classification has various applications in fields such as e-mail filtering, medical diagnosis, financial areas, etc. These applications require highly accurate classification to be useful. Hence, it is necessary to evolve current classification algorithms to produce better and better results. Thus, there is a need to analyze various algorithms which can perform better compared to others in the concerned areas. This problem has been the basis of work of this project.

Convolutional Neural Networks(CNNs) were responsible for major breakthroughs in Image Classification and are the core of most Computer Vision systems today, from Facebook's automated photo tagging to self-driving cars.

Here, we report on a series of experiments with convolutional neural networks (CNN) for sentence and document classification tasks. These neural networks are trained on the top of without pre-trained word vectors (random initialization) and pre-trained word vectors (glove and word2vec).

We additionally compared the results of the model obtained from CNN with Support Vector Machines and Naïve Bayes classifier. The CNN models discussed herein has better accuracy than the SVM and Naïve Bayes.

Chapter1: Introduction

1.1 Machine Learning

Machine learning is a type of artificial intelligence (AI) that provides computers with the ability to learn without being explicitly programmed. Machine learning focuses on the development of computer programs that can teach themselves to grow and change when exposed to new data. In general, a computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E . For example- Suppose an email program watches which emails to not mark as spam, and based on that learns how to better filter spam. Here the task T, performance P and experience E is given below.

Task, T- It involves classifying emails as spam or not spam.

Performance, P - The fraction of emails correctly classified as spam/not spam.

Experience, E - Watching you label emails as spam or not spam.

1.2 Supervised Learning

Supervised learning is the learning process when the correct outcome variable is known, and that information is explicitly used in training (Supervised) the model. In supervised learning, each example is a pair consisting of an input object and a desired output value. A supervised learning algorithm learn from training dataset and produces an inferred function, which can be used for mapping new examples.

Supervised learning has two types.

1. Regression-In regression the output for an input example has continuous set of values. Example-Housing price prediction

2. Classification-In classification the output for an input example has discrete set of values mapped as different class label.
Example-Predicting whether a person has cancer or not

1.2.1 Semi-Supervised learning

Semi-supervised learning is a class of supervised learning tasks and techniques that also make use of unlabeled data for training – typically a small amount of labeled data with a large amount of unlabeled data. Semi-supervised learning falls between unsupervised learning (without any labeled training data) and supervised learning (with completely labeled training data). Semi-supervised learning may refer to either transductive learning or inductive learning.

1.3 Unsupervised learning

Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labeled responses.

The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data. The clusters are modeled using a measure of similarity which is defined upon metrics such as Euclidean or probabilistic distance.

Common clustering algorithms include:

- **Hierarchical clustering**: builds a multilevel hierarchy of clusters by creating a cluster tree
- **k-Means clustering**: partitions data into k distinct clusters based on distance to the centroid of a cluster
- **Gaussian mixture models**: models clusters as a mixture of multivariate normal density components
- **Self-organizing maps**: uses neural networks that learn the topology and distribution of the data
- **Hidden Markov models**: uses observed data to recover the sequence of states.

1.4 Word Embedding

Word embedding is the collective name for a set of language modeling and feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers.

1.5 Classification

In machine learning and statistics, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known.
An algorithm that implements classification, especially in a concrete implementation, is known as a classifier. In the terminology of machine learning, classification is considered an instance of supervised learning, i.e. learning where a training set of correctly identified observations is available.

1.6 Feed forward neural networks

A feedforward neural network is an artificial neural network wherein connections between the units do not form a cycle. As such, it is different from recurrent neural networks.

The feedforward neural network was the first and simplest type of artificial neural network
 devised. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

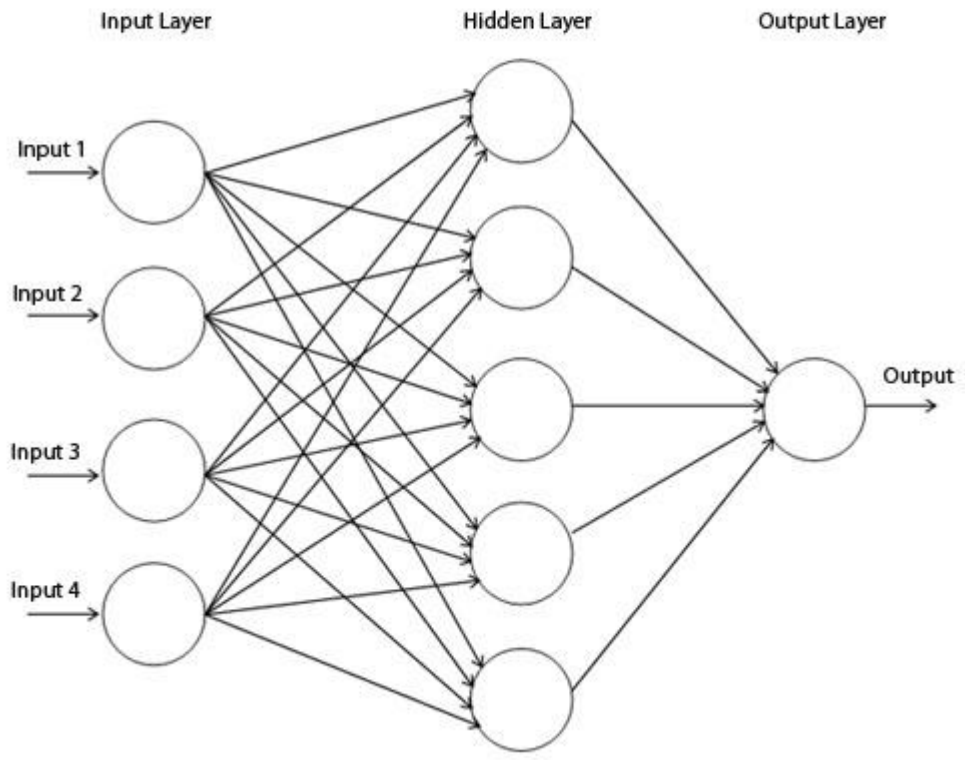


Figure 1: Feedforward Neural Network

1.7 Convolutional neural network

In machine learning, a convolutional neural network (CNN, or ConvNet) is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex.

1.7.1The CNN Architecture

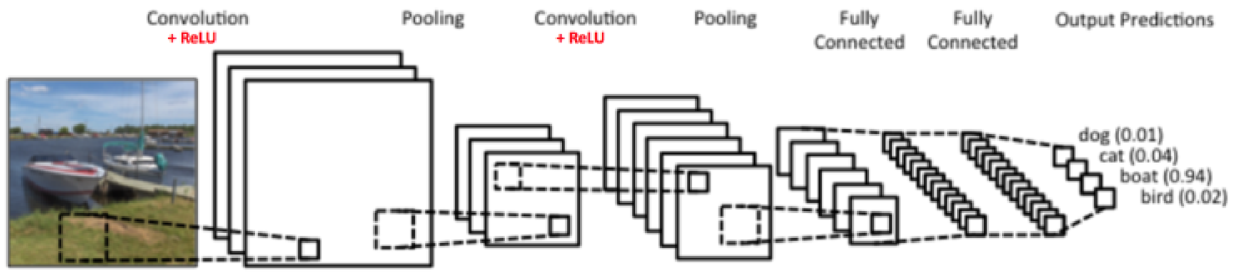


Figure 2: CNN Architecture

The architecture of a typical CNN is composed of multiple layers where each layer performs a specific function of transforming its input into a useful representation. There are 3 major types of layers that are commonly observed in complex neural network architectures:

1.7.2 Convolutional Layer

Also referred to as Conv. layer, it forms the basis of the CNN and performs the core operations of training and consequently firing the neurons of the network. It performs the convolution operation over the input volume and consists of a 3-dimensional arrangement of neurons .

If we have N×N square neuron layer which is followed by our convolutional layer. If we use an m×m filter ω, our convolutional layer output will be of size:

$$(N-m+1)\times(N-m+1)$$

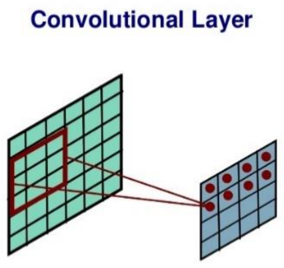


Figure 3: Convolution Layer

In order to compute the pre-nonlinearity input to some unit $\mathbf{x}_{ij}^{l}$ in our layer, we need to sum up the contributions (weighted by the filter components) from the previous layer cells:

$$x_{ij}^{l}=\sum_{a=0}^{m-1}\sum_{b=0}^{m-1}\omega_{ab}y^{\ell-1}_{(i+a)(j+b)}.$$

Then, the convolutional layer applies its nonlinearity:

$$y^{\ell}_{ij}=\sigma(x^{\ell}_{ij}).$$

1.7.3 The ReLu (Rectified Linear Unit) Layer

ReLu refers to the Rectifier Unit, the most commonly deployed activation function for the outputs of the CNN neurons. Mathematically, it's described as:

$$\max(0,x)$$

1.7.4 The Pooling Layer

The pooling layer is usually placed after the Convolutional layer. Its primary utility lies in reducing the spatial dimensions (Width x Height) of the Input Volume for the next Convolutional Layer. It does not affect the depth dimension of the Volume.

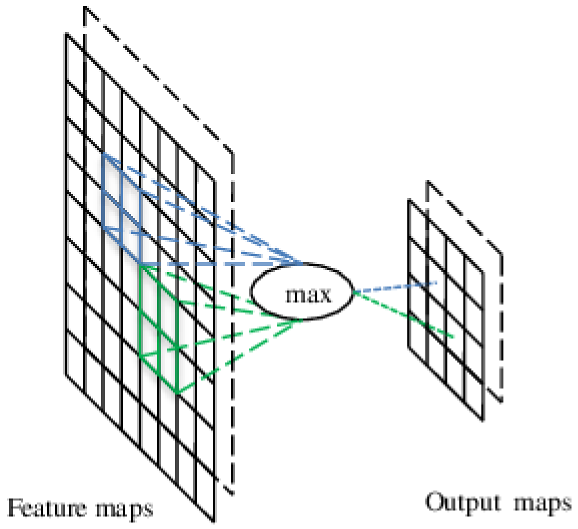


Figure 4: Max Pooling Layer

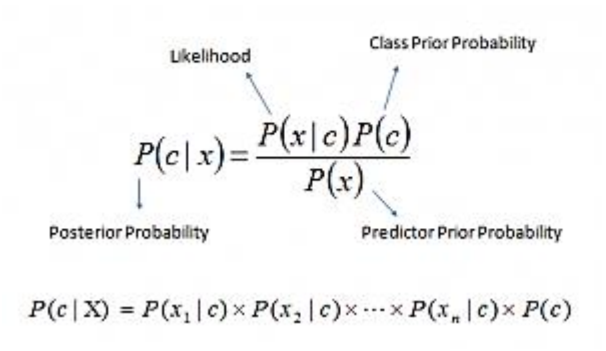
The operation performed by this layer is also called 'down-sampling', as the reduction of size leads to loss of information as well. However, such a loss is beneficial for the network for two reasons:

- the decrease in size leads to less computational overhead for the upcoming layers of the network;
- it works against over-fitting.

The max-pooling layers are quite simple, and do no learning themselves. They simply take some k×k region and output a single value, which is the maximum in that region. For instance, if their input layer is a N×N layer, they will then output a N/k×N/k layer, as each k×k block is reduced to just a single value via the max function.

1.8 Naive Bayes

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. It is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features.



$$P(c \mid x) = \frac{P(x \mid c)\,P(c)}{P(x)}$$

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Figure 5: Naïve Bayes

- $P(c|x)$ is the posterior probability of *class* (c, *target*) given *predictor* (x, *attributes*).
- $P(c)$ is the prior probability of *class*.
- $P(x|c)$ is the likelihood which is the probability of *predictor* given *class*.
- $P(x)$ is the prior probability of *predictor*.

For some types of probability models, naive Bayes classifiers can be trained very efficiently in a supervised learning setting. In many practical applications, parameter estimation for naive Bayes models uses the method of maximum likelihood; in other words, one can work with the naive Bayes model without accepting Bayesian probability or using any Bayesian methods.

1.9 Support vector machine

In machine learning, support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging
to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. An

SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

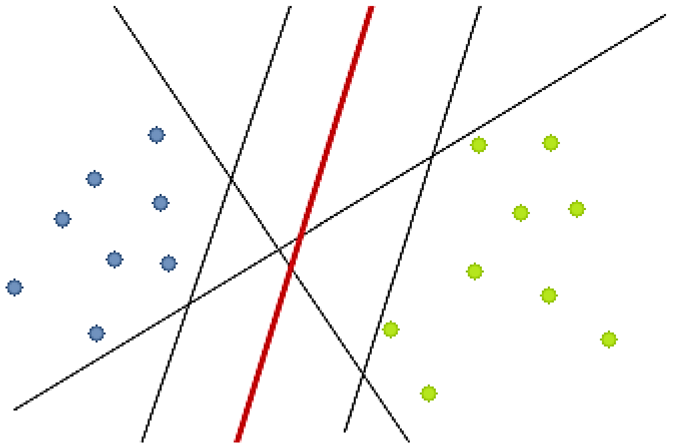


Figure 6: Optimal Separating Hyperspace

SVM searches for a separating hyperplane, which separates positive and negative examples from each other with maximal margin, in other words, the distance of the decision surface and the closest example is maximal.

Chapter2: Literature Survey

The following works have been analyzed in the field of our study: -

Alex Krizhevsky[2012] et. al discussed the architecture of the network (which was called AlexNet). They have used a relatively simple layout, compared to modern architectures. The network was made up of 5 conv layers, max-pooling layers, dropout layers, and 3 fully connected layers. The network they designed was used for classification with 1000 possible categories. They trained the network on ImageNet data, which contained over 15 million annotated images from a total of over 22,000 categories. They have used ReLU for the nonlinearity functions (found to decrease training time as ReLUs are several times faster than the conventional tanh function).

Vandana Korde et al (2012) discussed that the text mining studies are gaining more importance recently because of the availability of the increasing number of the electronic documents from a variety of sources which include unstructured and semi structured information. The main goal of text mining is to enable users to extract information from textual resources and deals with the operations like, retrieval, classification (supervised, unsupervised and semi supervised) and summarization, Natural Language Processing (NLP), Data Mining, and Machine Learning techniques work together to automatically classify and discover patterns from the different types of the documents.

Zakaria Elberrichi, et al (2008) says that in this paper, a new approach is proposed for text categorization based on incorporating background knowledge (WordNet) into text representation with using the multivariate, which consists of extracting the K better features characterizing best the category compared to the others. The experimental results with both Reuters21578 and 20Newsgroups datasets show that incorporating background knowledge in order to capture relationships between words is especially effective in raising the macro-averaged F1 value. The main difficulty is that a word usually has multiple synonyms with somewhat different meanings and it is not easy to automatically find the correct synonyms to use.

Vishwanath Bijalwan et.al in his paper have first categorize the documents using KNN based machine learning and then return the most relevant documents. In this paper author conclude that KNN shows the maximum accuracy as compared to the Naive Bayes and Term-Graph. The disadvantage of KNN classifier is that its time complexity is high but gives a enhanced accuracy than others. In this paper the author rather than implementing the traditional Term-Graph used with AFOPT used TermGraph with other methods. This hybrid shows a better result than the traditional combination. Finally author made an information retrieval application using Vector Space Model to give the result of the query entered by the client by showing the relevant document

Chapter3: Proposed Work

3.1 Motivation

The Convolutional Neural Networks have been successfully used for classifying images and have achieved state of the art results in many standard classification problems.

More recently researchers also started to apply CNNs to problems in Natural Language Processing and gotten some interesting results. Keeping the same spirit, we have also implemented text classification using Convolutional Neural Networks and have done a comparative analysis for the same with Naïve Bayes classifier and Support Vector Machine Classifier.

3.2 Datasets Used

1. MR dataset:-
   The dataset contains 10,662 example review sentences, half positive and half negative. The dataset has a vocabulary of size around 20k. The dataset doesn't come with an official train/test split, so we have simply used 10% of the data as a dev set.

2. 20 Newsgroup dataset:
   The 20 Newsgroups data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups.
   Organization

   The data is organized into 20 different newsgroups, each corresponding to a different topic. Here is a list of the 20 newsgroups, partitioned (more or less) according to subject matter:

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

3.3 Word Embeddings Used

1. Word2Vec: Word2vec is a two-layer neural net that processes text. Its input is a text corpus and its output is a set of vectors: feature vectors for words in that corpus. While Word2vec is not a deep neural network, it turns text into a numerical form that deep nets can understand

   The output of the Word2vec neural net is a vocabulary in which each item has a vector attached to it, which can be fed into a deep-learning net or simply queried to detect relationships between words.

   Source: https://code.google.com/archive/p/word2vec/

2. Glove : GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

   Source: https://nlp.stanford.edu/projects/glove/

3.4 Proposed Model

Data and Preprocessing
The MR dataset has a vocabulary of size around 20k. The dataset doesn’t come with an official train/test split, so we simply use 10% of the data as a dev set.

The 4 categories chosen from the 20 Newsgroup Dataset are:

- alt.atheism

- comp.graphics

- sci.med

- soc.religion.christian

The total vocabulary size for these chosen categories is 35769.

Data preprocessing steps:

1. Loading of positive and negative sentences from the raw data files (in case of mr dataset) and loading of documents from the specified categories in case of 20 newsgroup dataset.
2. Cleaning of the textual data is performed.
3. Pad each sentence to the maximum sentence length. We append special <PAD> tokens to all other sentences to make them upto the maximum sentence length. Padding sentences to the same length is useful because it allows us to efficiently batch our data since each example in a batch must be of the same length.
4. Build a vocabulary index and map each word to an integer between 0 and the maximum vocabulary size-1. Each sentence becomes a vector of integers.

3.5 The Layers

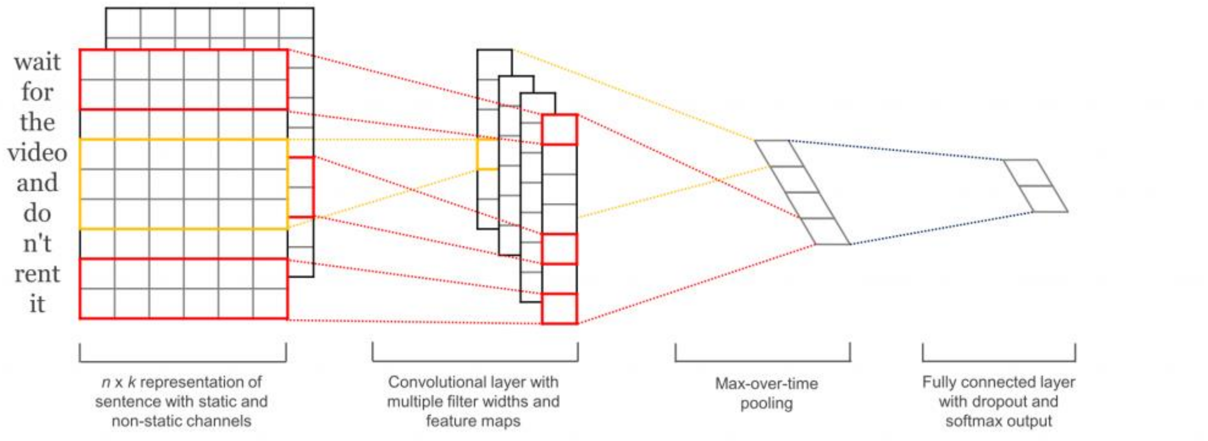

wait
for
the
video
and
do
n't
rent
it
n x *k* representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 7: Model for text classification

1. Embedding Layer

   The first layer we define is the embedding layer, which maps vocabulary word indices into low-dimensional vector representations. It’s essentially a lookup table that we learn from data

2. Convolution Layer

   The next layer performs convolutions over the embedded word vectors using multiple filter sizes. For example, sliding over 3, 4 or 5 words at a time.
3. Activation Layer

   The experiment is performed with rectified linear unit (ReLU) activation function. This activation is used due to following reasons:
   1. Training a network when ReLU is used is faster, and it is more biological inspired.

2. One major benefit is the reduced likelihood of the gradient to vanish. In this regime the gradient has a constant value. In contrast, the gradient of sigmoids becomes increasingly small as the absolute value of x increases. The constant gradient of ReLUs results in faster learning.
3. The other benefit of ReLUs is sparsity. The more units that exist in a layer the more sparse the resulting representation. Sigmoids on the other hand are always likely to generate some non-zero value resulting in dense representations. Sparse representations seem to be more beneficial than dense representations.

4. Max Pooling Layer
   Next, we max-pool the result of the convolutional layer into a long feature vector.
5. Fully Connected Layer
   The outputs obtained from different max-pooled feature maps are flattened to create the fully connected layer. We add dropout regularization, and classify the result using a softmax layer.

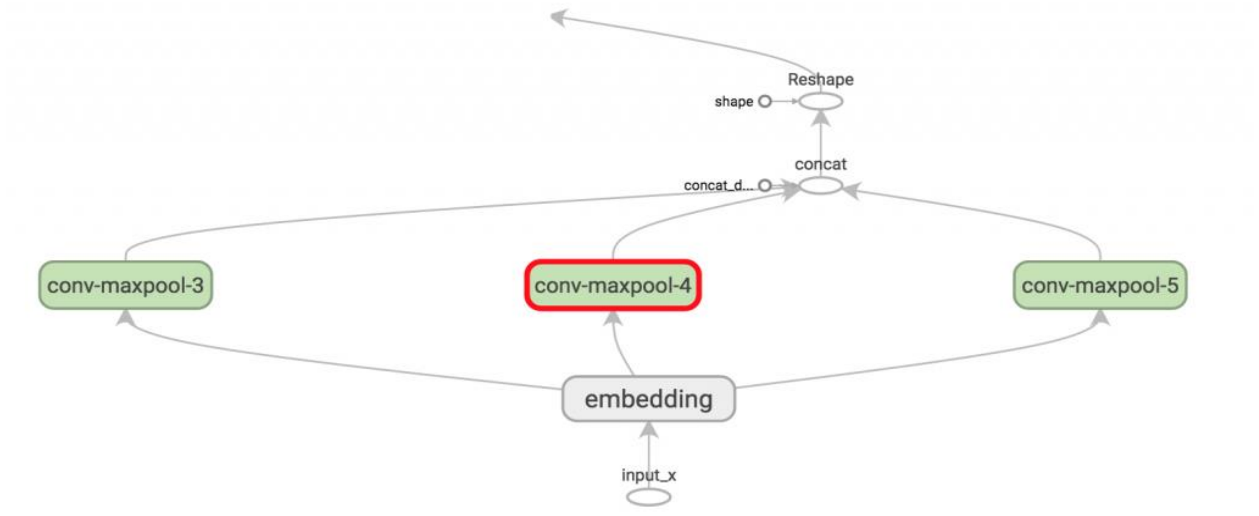


Figure 8: Graph of the model

6. Dropout Layer
   A dropout layer stochastically "disables" a fraction of its neurons. This prevent neurons from co-adapting and forces them to learn individually useful features. The fraction of neurons we keep enabled is defined by the dropout_keep_prob input to our network. We had set this to 0.5 during training, and to 1 (disable dropout) during evaluation.

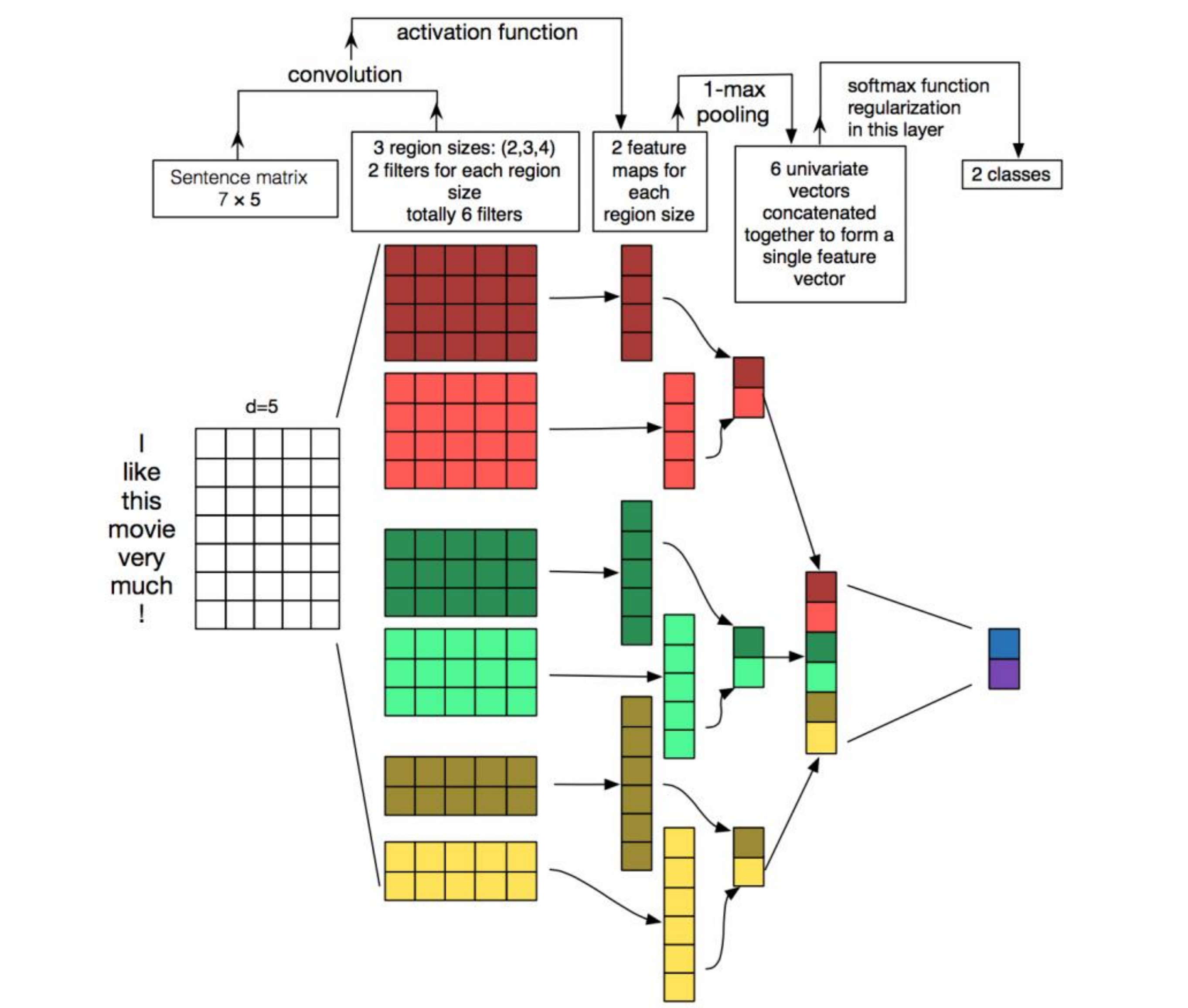

activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I like this movie very much !

Figure 9: Illustration of the CNN architecture for text classification

Chapter4: Implementation

4.1 Hardware and Software

While performing the experiments, the following environment was used-

1. Tensorflow 1.0
2. Python 3.x
3. Windows 10
4. Intel Core-i5 processor machine.

4.2 Experimental Procedure

The main aim of the experiment is to perform text classification using Convolutional Neural Networks.

Datasets used for text classification:-

1. Movie Review dataset of Rotten Tomatoes:
   Sentiment analysis on movie reviews (positive or negative reviews).
   Source: https://www.cs.cornell.edu/people/pabo/movie-review-data/

2. 20 Newsgroup dataset:
   Document classification (out of the 20, 4 categories are taken for the purpose of classification)
   Source: http://scikit-learn.org/stable/datasets/twenty_newsgroups.html

Two methods of classification with Neural Nets are used:

1. Without using pre-trained word embeddings :-
   The word vectors are randomly initialized.
2. With using pre-trained word embeddings:-
   The word vectors are initialized from the popular word embedding like word2vec and glove

Moreover, the results obtained from of this cnn text classification are compared with naive bayesian and support vector machine (SVM) using the scikit-learn.

4.3 Libraries Used

Tensorflow:

TensorFlow is an open source software library released in 2015 by Google to make it easier for developers to design, build, and train deep learning models. At a high level, **TensorFlow** is a Python library that allows users to express arbitrary computation as a graph of data flows.

Numpy:

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Yaml:

YAML is a human-readable data serialization language. It is commonly used for configuration files, but could be used in many applications where data is being stored (e.g. debugging output) or transmitted (e.g. document headers).

Scikit Learn:

Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

4.4 Important Functions used

4.4.1 Tensorflow

1. tf.nn.embedding_lookup

This function is used to perform parallel lookups on the list of tensors in parameters.

2.tf.nn.relu(features, name=None)

Computes rectified linear: max(features,0)

3. tf.nn.conv2d

Computes a 2-D convolution given 4-D input and filter tensors.

This operation performs the following:

Flattens the filter to a 2-D matrix

Extracts image patches from the input tensor to form a virtual

For each patch, right-multiplies the filter matrix and the image patch vector.

4. tf.nn.max_pool

Performs the max pooling on the input.

5. tf.nn.dropout

Computes dropout.

With probability keep_prob, outputs the input element scaled up 1/keep_prob, otherwise outputs 0. The scaling is so that the expected sum is unchanged.

6. tf.nn.l2_loss

L2 Loss.

Computes half the L2 norm of a tensor without the sqrt:

Output = sum(t**2) / 2

7. tf.nn.softmax_cross_entropy_with_logits

Computes softmax cross entropy between logits and labels.

Measures the probability error in discrete classification tasks in which the classes are mutually exclusive (each entry is in exactly one class).

8. tf.learn.preprocessing.VocabularyProcessor(max_document_length)
learn.preprocessing.VocabularyProcessor converts a sentence given to it into a vector of words . The parameter max_document_length makes sure that all the documents are represented by a vector of length max_document_length either by padding numbers if their length is shorter than max_document_length and clipping them if their length is greater than max_document_length

9. tf.train.AdamOptimizer

Optimizer that implements the Adam algorithm.

Adam is a stochastic gradient descent algorithm based on estimation of 1st and 2nd-order moments. The algorithm estimates 1st-order moment (the gradient mean) and 2nd-order moment (element-wise squared gradient) of the gradient using exponential moving average, and corrects its bias. The final weight update is proportional to learning rate times 1st-order moment divided by the square root of 2nd-order moment.

4.4.2 Scikit Learn

1. CountVectorizer:

CountVectorizer implements both tokenization and occurrence counting in a single class

- **tokenizing** strings and giving an integer id for each possible token, for instance by using white-spaces and punctuation as token separators.
- **counting** the occurrences of tokens in each document.

2. TfidfTransformer:

Transform a count matrix to a normalized tf or tf-idf representation.Tf means term-frequency while tf-idf means term-frequency times inverse document-frequency. This is a common term weighting scheme in information retrieval, that has also found good use in document classification.

3. MultinomialNB:

Naive Bayes classifier for multinomial models

The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work.

Multinomial Naive Bayes is a specialized version of Naive Bayes that is designed more for text documents. Whereas simple naive Bayes would model a document as the presence and absence of particular words, multinomial naive bayes explicitly models the word counts and adjusts the underlying calculations to deal with in.

4. Pipeline:

Python scikit-learn provides a Pipeline utility to help automate machine learning workflows. Pipelines work by allowing for a linear sequence of data transforms to be chained together culminating in a modeling process that can be evaluated.

Pipeline can be used to chain multiple estimators into one. This is useful as there is often a fixed sequence of steps in processing the data, for example feature selection, normalization and classification.

5. SGDClassifier

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as (linear) Support Vector Machines and Logistic Regression.

The class **SGDClassifier** implements a plain stochastic gradient descent learning routine which supports different loss functions and penalties for classification.

6. metrics.classification_report:

Build a text report showing the main classification metrics

7. metrics.confusion_matrix:

Compute confusion matrix to evaluate the accuracy of a classification

Chapter 5: Result Analysis

After the experimentation following results have been obtained: -

1. MR Dataset

    a. Without using pre-trained word vectors:-

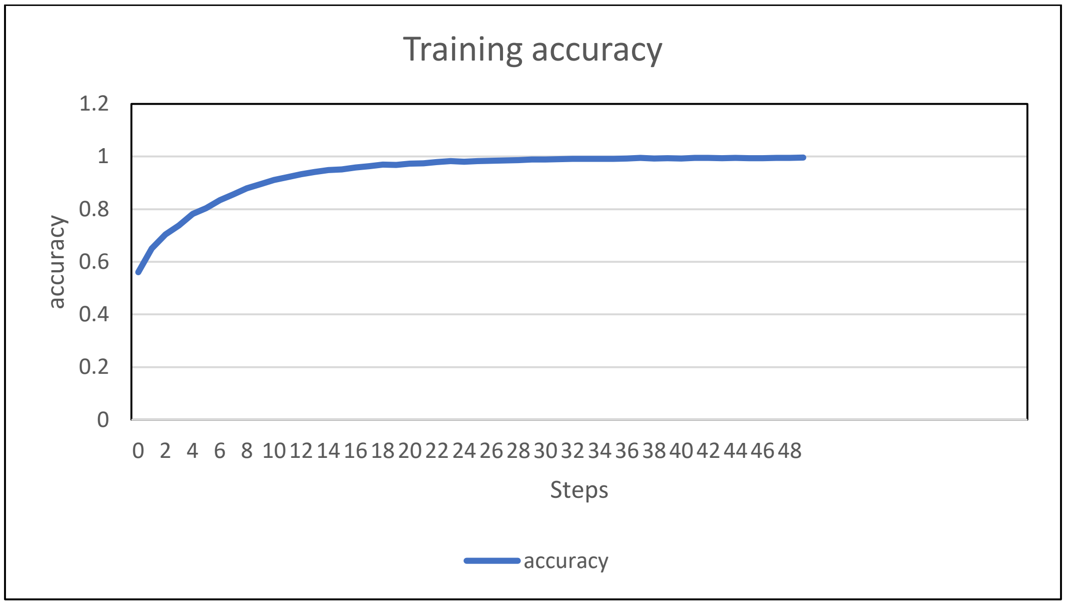


Figure 10 :MR dataset without pre –trained word vectors: Plots of steps vs accuracy for training data

Final Value = 0.9962
Relative Time for Completion = 1hr20m24s

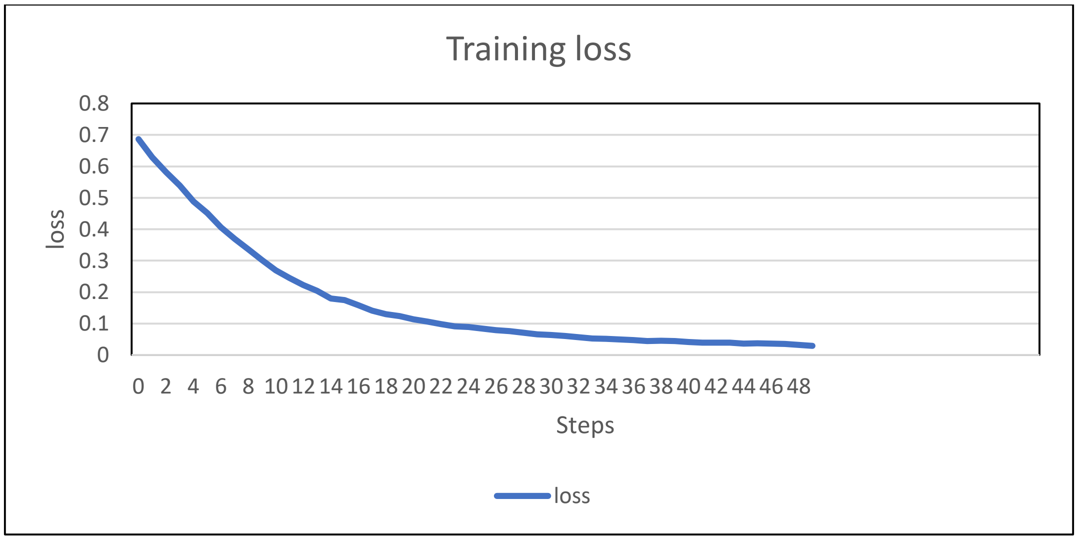


Figure 11 :MR dataset without pre –trained word vectors: Plots of steps vs loss for training data

Final Value = 0.02988
Relative time for completion = 1hr20m24s

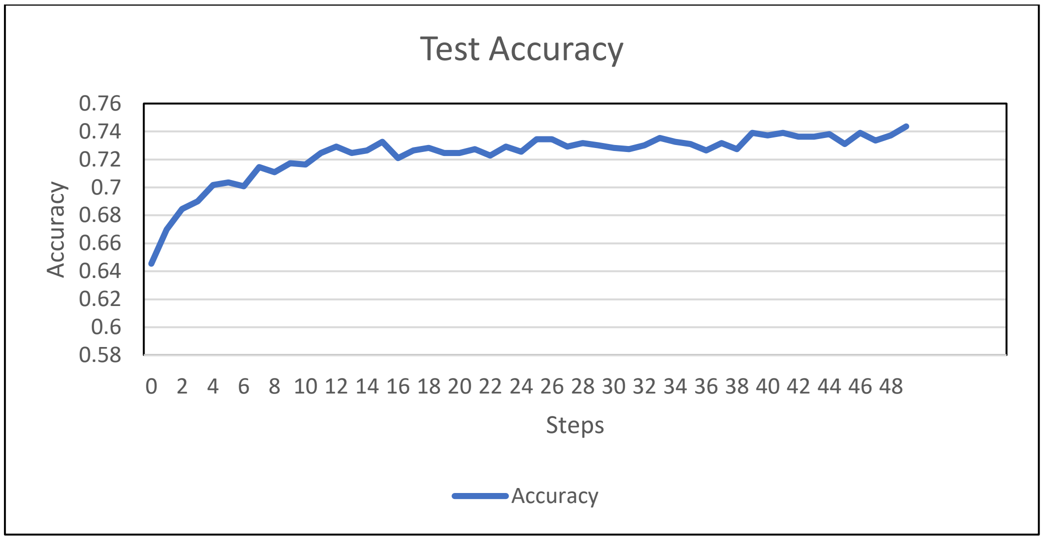


Figure 12:MR dataset without pre –trained word vectors: Plots of steps vs accuracy for test data

Final Value = 0.7436
Relative time for completion = 1hr20m24s

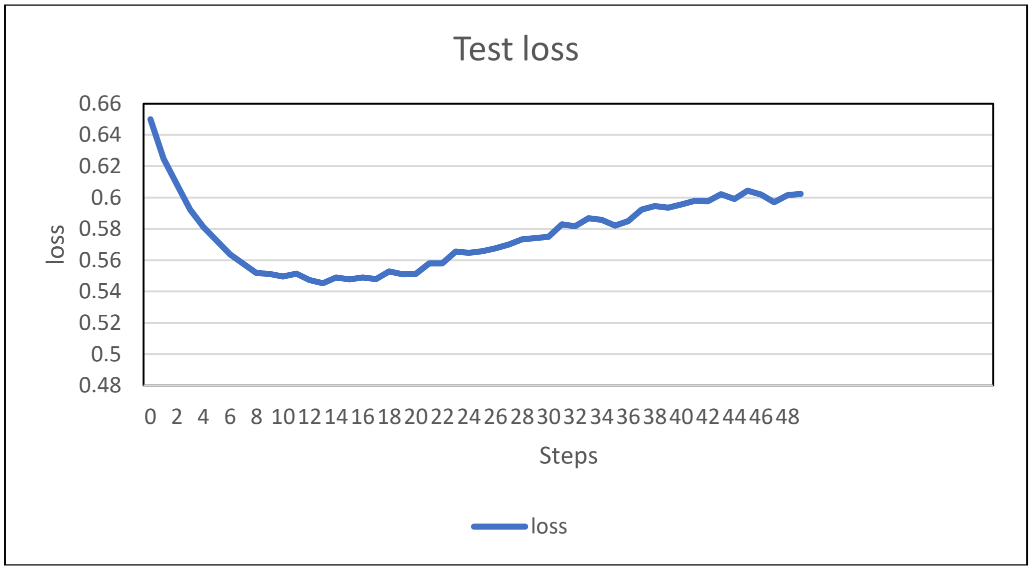


Figure 13:MR dataset without pre –trained word vectors: Plots of steps vs loss for test data

Final Value = 0.6024
Relative time for completion = 1hr20m24s

b. Using pre-trained word vectors:-

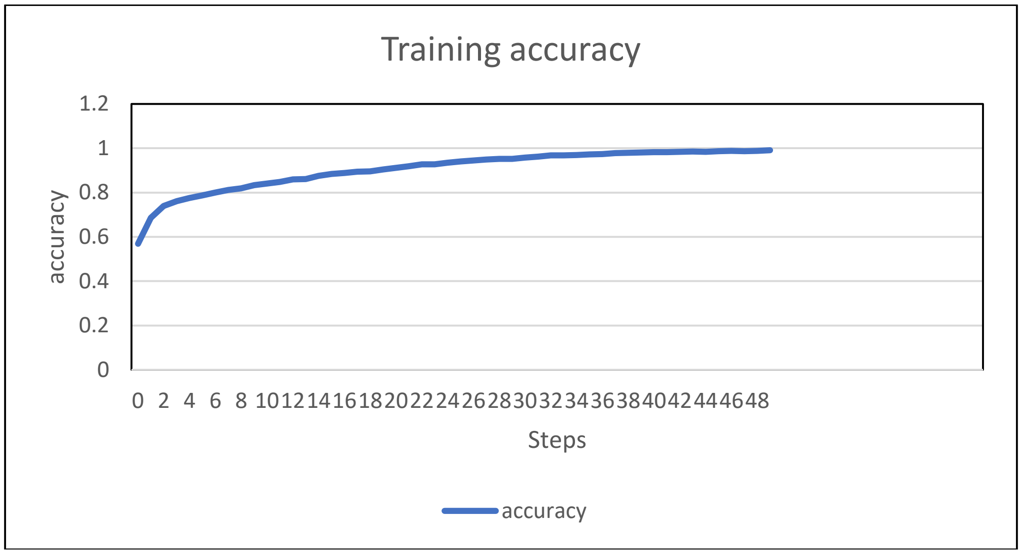


Figure 14 :MR dataset with pre –trained word vectors: Plots of steps vs accuracy for training data

Final Value = 0.9909
Relative time for completion = 1hr23m14s

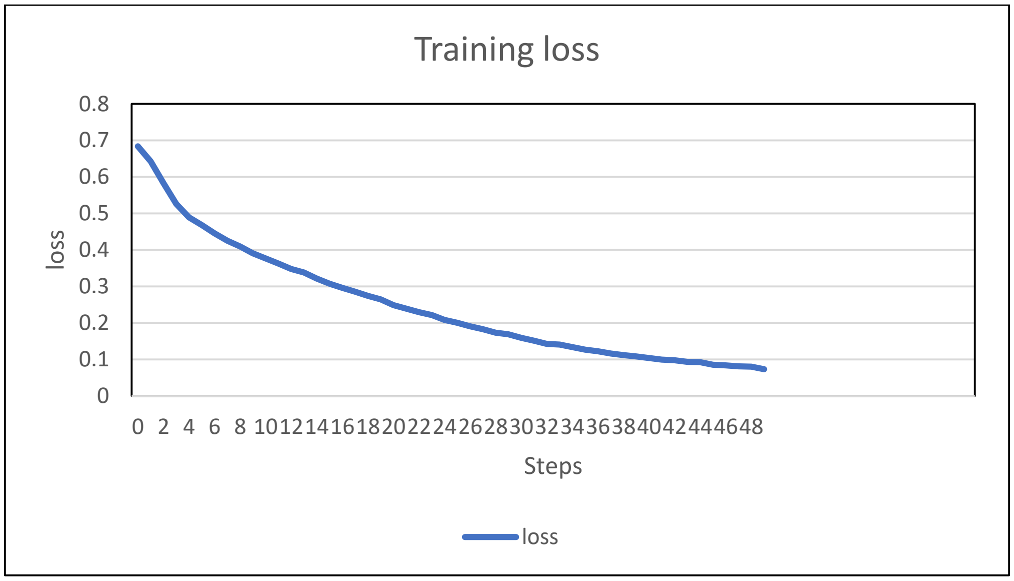


Figure 15 :MR dataset with pre –trained word vectors: Plots of steps vs loss for training data

Final Value = 0.07280
Relative time for completion = 1hr23m14s

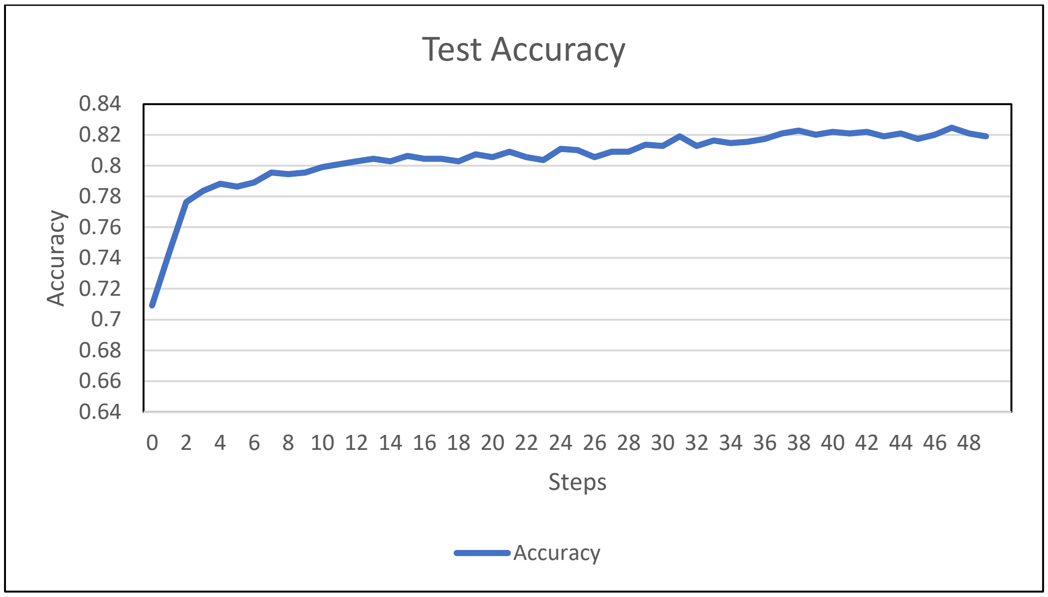


Figure 16 :MR dataset with pre –trained word vectors: Plots of steps vs accuracy for test data

Final Value = 0.8191
Relative time for completion = 1hr23m14s

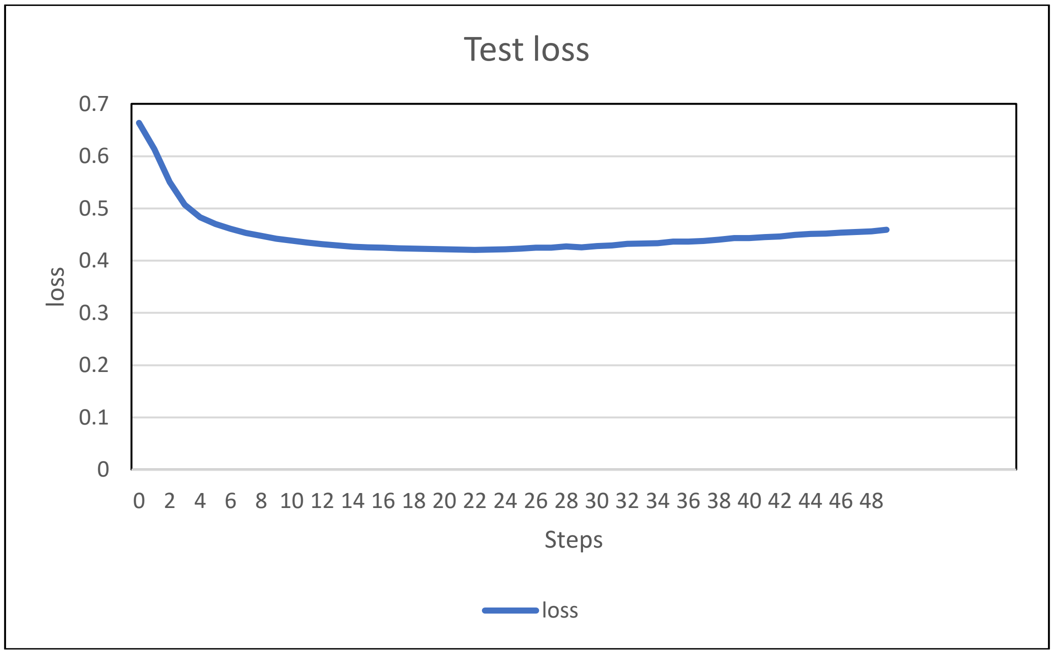


Figure 17 :MR dataset with pre –trained word vectors: Plots of steps vs loss for test data

Final Value = 0.4590
Relative time for completion = 1hr23m14s

Table1: Training accuracy for MR dataset: -

Without using pre-trained word vectors	Using pre-trained word vectors
74.36%	**81.91%**

2. 20 Newsgroup Dataset

a. Without using pre-trained word vectors

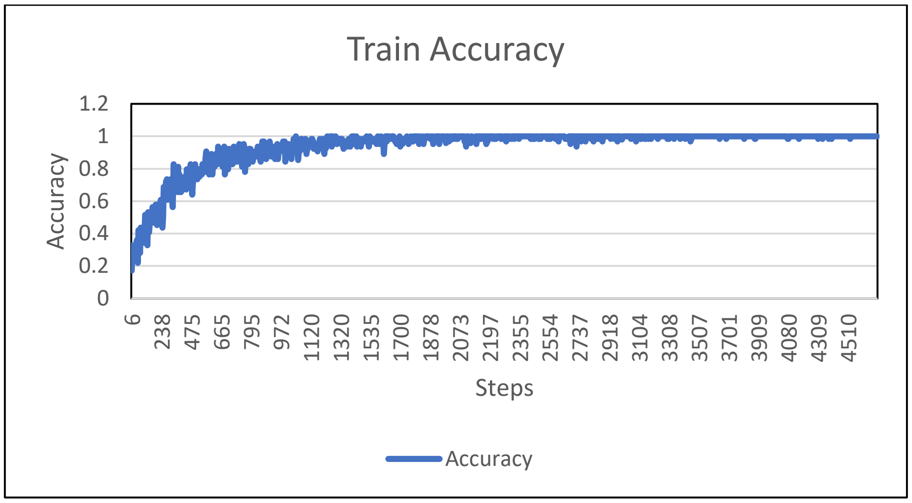


Figure18: 20 Newsgroup without word vector: Plots of steps vs accuracy for train data

Final Value=1.000

Relative time for completion :20h 0m 48s

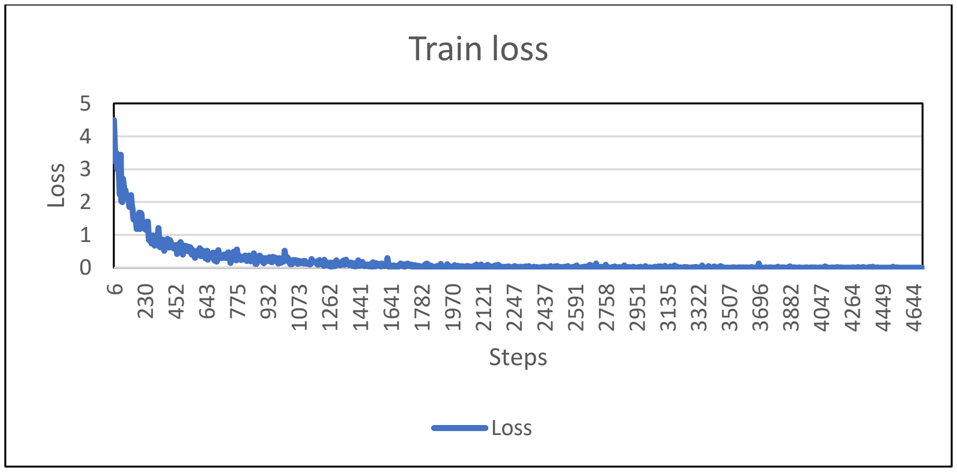


Figure 19: Newsgroup without word vector: Plots of steps vs loss for train data

Final Value=6.0336e-4

Relative time for completion =20h 0m 48s

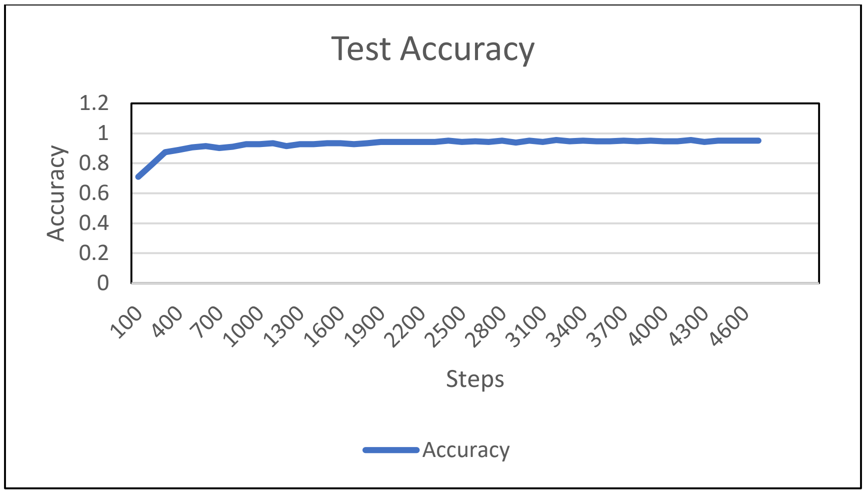


Figure 20 :20 Newsgroup without word vector: Plots of steps vs accuracy for test data

Final Value=0.9511

Relative time for completion =19h 37m 13s

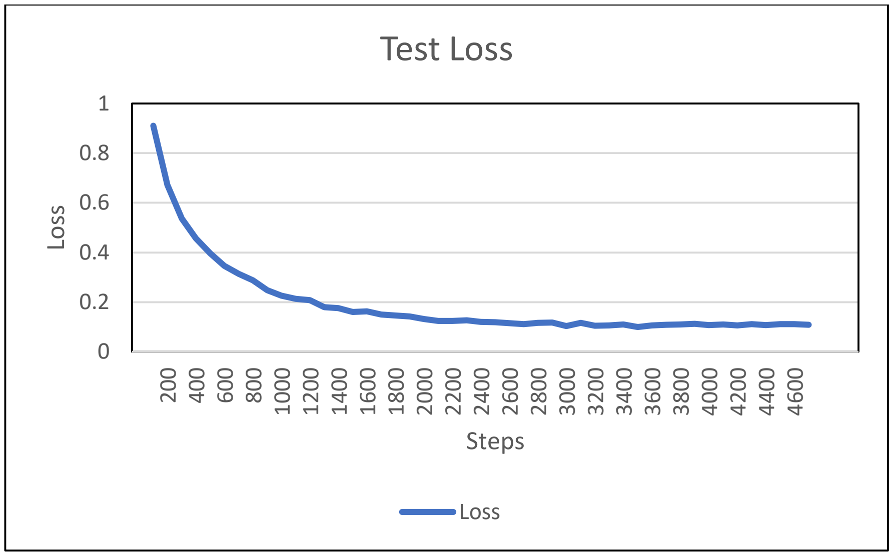


Figure 21:Newsgroup without word vector: Plots of steps vs loss for test data

Final Value=0.1091

Relative time for completion=19h 37m 13s

b. Using pre-trained word vectors

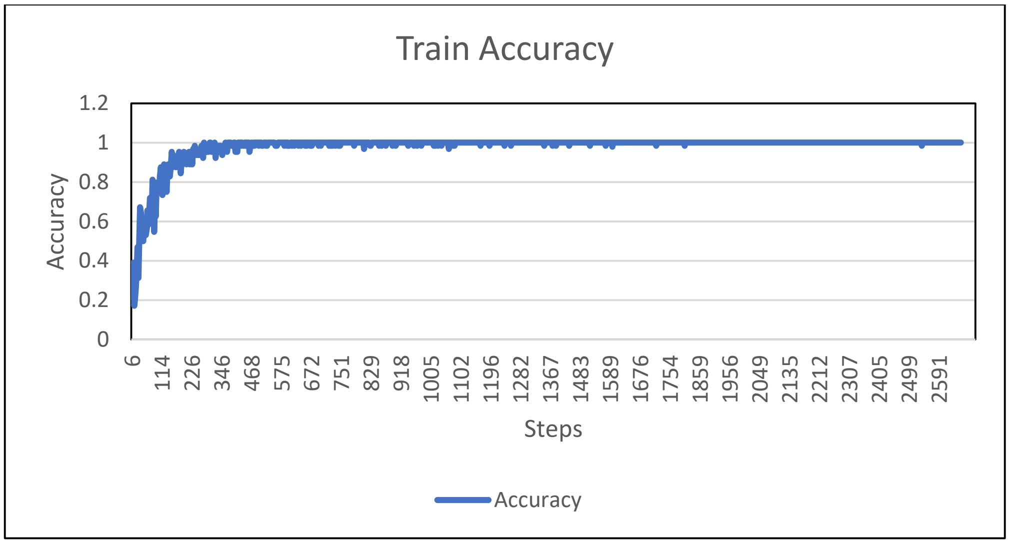


Figure 22: 20 Newsgroup with word vector: Plots of steps vs accuracy for train data

Final Value=1.000

Relative time for completion :23h 37m 11s

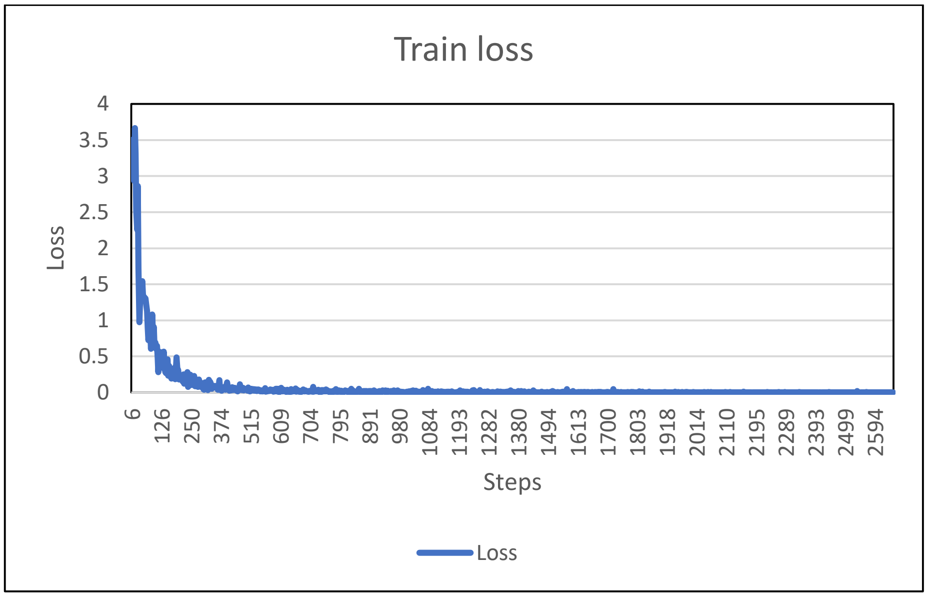


Figure 23: Newsgroup with word vector: Plots of steps vs loss for train data

Final Value=8.3148e-4

Relative time for completion=23h 37m 11s

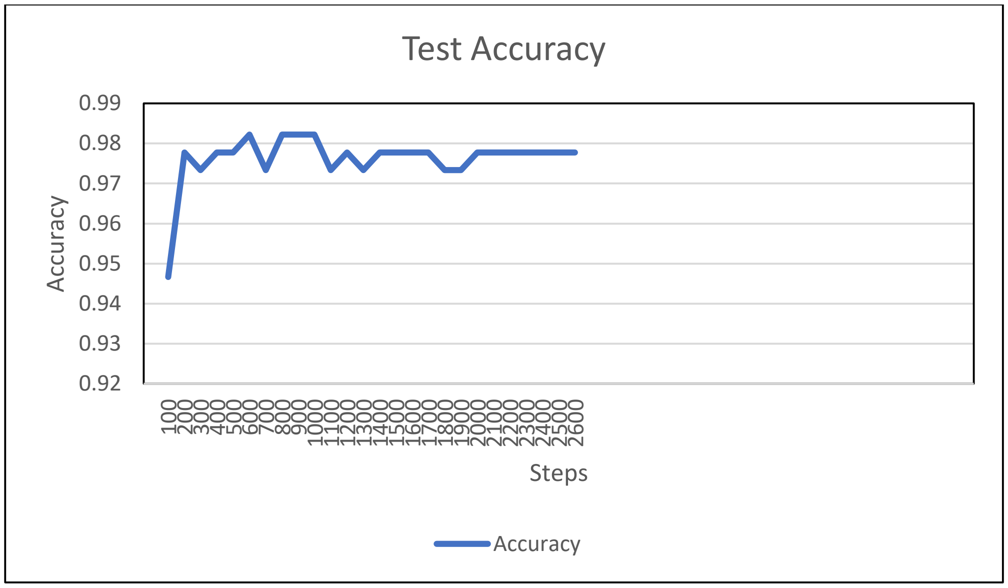


Figure 24 :20 Newsgroup with word vector: Plots of steps vs accuracy for test data

Final Value=0.9778

Relative time for completion=21h 58m 13s

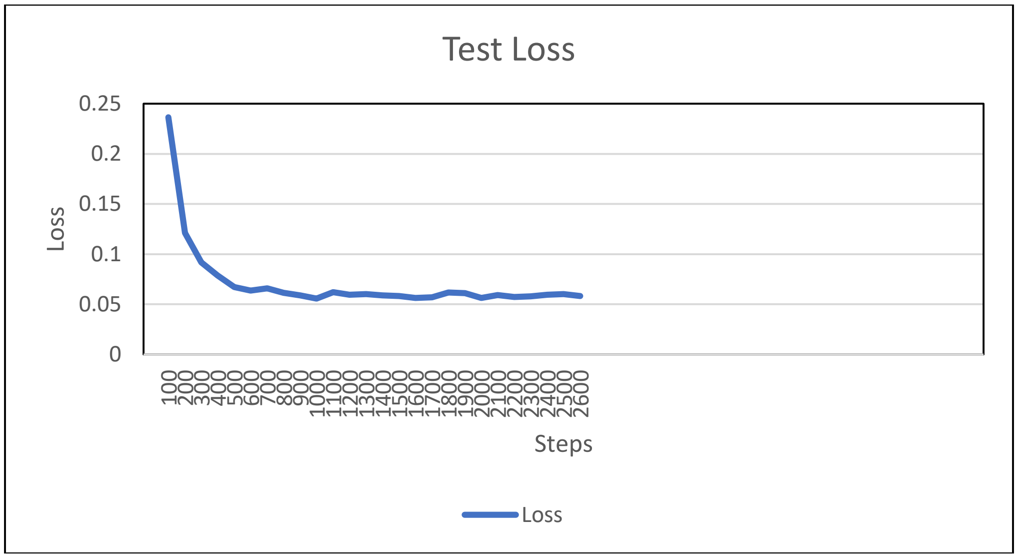


Figure 25:Newsgroup with word vector: Plots of steps vs loss for test data

Final Value=0.1091

Relative time for completion=19h 37m 13s

Table2: Training accuracy for 20 Newsgroup dataset: -

Naïve Bayes	SVM	CNN without using pre-trained word vectors	CNN using pre-trained word vectors
83.0%	**91.0%**	**95.11%**	**97.78%**

Chapter 6: Conclusion

In the present work we have described a series of experiments with convolutional neural networks for text classification tasks: sentiment analysis (MR dataset) and document classification (20 NewsGroup Dataset).

These CNN's were built on top of with and without pre-trained word embeddings. The classification accuracy achieved by the convolutional neural network has increased with the use of pre-trained word embeddings.

We have shown that such CNN's though require more time to train but have far better accuracy than Support Vector Machines and Naïve Bayes classifier.

Moreover, our results add to the well-established evidence that unsupervised pre-training of word vectors is an important ingredient in deep learning for NLP.

References

[1] Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014), 1746–1751.

[2] Zhang, X., & LeCun, Y. (2015). Text Understanding from Scratch. arXiv E-Prints, 3, 011102.

[3] Thorsten Joachims: Text categorization with support vector machines: learning with many relevant features, Proc. of ECML-98, 10th European Conference on Machine Learning, Springer Verlag, Heidelberg, DE, 1998, pp. 137-142

[4] The 20 Newsgroups data set http://people.csail.mit.edu/jrennie/20Newsgroups/

[5] Krizhevsky, A., Sutskever, I. and Hinton, G. E.
ImageNet Classification with Deep Convolutional Neural Networks
NIPS 2012: Neural Information Processing Systems, Lake Tahoe, Nevada

[6] Vandana Korde et al Text classification and classifiers:” International Journal of Artificial Intelligence & Applications (IJAIA), Vol.3, No.2, March 2012”.

[7] "Zakaria Elberrichi,Abdelattif Rahmoun, and Mohamed Amine Bentaalah""Using WordNet for Text Categorization""The International Arab Journal of Information Technology, Vol. 5, No. 1, January 2008".

[8] Vishwanath Bijalwan, Vinay Kumar, Pinki Kumari, Jordan Pascual. “KNN based Machine Learning Approach for Text and Document Mining”, 2014,Vol.7,No.1,pp.6170.

Appendix

Codes

1. CNN Architecture for Text Classification

```
import tensorflow as tf
import numpy as np


class TextCNN(object):
  """
  A CNN for text classification.
  Uses an embedding layer, followed by a convolutional, max-pooling and softmax layer.
  """
  def __init__(
   self, sequence_length, num_classes, vocab_size,
   embedding_size, filter_sizes, num_filters, l2_reg_lambda=0.0):

    # Placeholders for input, output and dropout
    self.input_x = tf.placeholder(tf.int32, [None, sequence_length], name="input_x")
    self.input_y = tf.placeholder(tf.float32, [None, num_classes], name="input_y")
    self.dropout_keep_prob = tf.placeholder(tf.float32, name="dropout_keep_prob")

    # Keeping track of l2 regularization loss (optional)
    l2_loss = tf.constant(0.0)

    # Embedding layer
    with tf.device('/cpu:0'), tf.name_scope("embedding"):
      self.W = tf.Variable(
        tf.random_uniform([vocab_size, embedding_size], -1.0, 1.0),
        name="W")
      self.embedded_chars = tf.nn.embedding_lookup(self.W, self.input_x)
      self.embedded_chars_expanded = tf.expand_dims(self.embedded_chars, -1)

    # Create a convolution + maxpool layer for each filter size
    pooled_outputs = []
    for i, filter_size in enumerate(filter_sizes):
      with tf.name_scope("conv-maxpool-%s" % filter_size):
        # Convolution Layer
        filter_shape = [filter_size, embedding_size, 1, num_filters]
```

```
            W = tf.Variable(tf.truncated_normal(filter_shape, stddev=0.1), name="W")
            b = tf.Variable(tf.constant(0.1, shape=[num_filters]), name="b")
            conv = tf.nn.conv2d(
                self.embedded_chars_expanded,
                W,
                strides=[1, 1, 1, 1],
                padding="VALID",
                name="conv")
            # Apply nonlinearity
            h = tf.nn.relu(tf.nn.bias_add(conv, b), name="relu")
            # Maxpooling over the outputs
            pooled = tf.nn.max_pool(
                h,
                ksize=[1, sequence_length - filter_size + 1, 1, 1],
                strides=[1, 1, 1, 1],
                padding='VALID',
                name="pool")
            pooled_outputs.append(pooled)

    # Combine all the pooled features
    num_filters_total = num_filters * len(filter_sizes)
    self.h_pool = tf.concat(pooled_outputs, 3)
    self.h_pool_flat = tf.reshape(self.h_pool, [-1, num_filters_total])

    # Add dropout
    with tf.name_scope("dropout"):
        self.h_drop = tf.nn.dropout(self.h_pool_flat, self.dropout_keep_prob)

    # Final (unnormalized) scores and predictions
    with tf.name_scope("output"):
        W = tf.get_variable(
            "W",
            shape=[num_filters_total, num_classes],
            initializer=tf.contrib.layers.xavier_initializer())
        b = tf.Variable(tf.constant(0.1, shape=[num_classes]), name="b")
        l2_loss += tf.nn.l2_loss(W)
        l2_loss += tf.nn.l2_loss(b)
        self.scores = tf.nn.xw_plus_b(self.h_drop, W, b, name="scores")
        self.predictions = tf.argmax(self.scores, 1, name="predictions")

    # CalculateMean cross-entropy loss
    with tf.name_scope("loss"):
            losses = tf.nn.softmax_cross_entropy_with_logits(logits=self.scores,
labels=self.input_y)
            self.loss = tf.reduce_mean(losses) + l2_reg_lambda * l2_loss

        # Accuracy
        with tf.name_scope("accuracy"):
            correct_predictions = tf.equal(self.predictions, tf.argmax(self.input_y, 1))
            self.accuracy = tf.reduce_mean(tf.cast(correct_predictions, "float"),
name="accuracy")
```

2.The code to run the model

```
import tensorflow as tf
import numpy as np
import os
import time
import datetime
import data_helpers
from text_cnn import TextCNN
from tensorflow.contrib import learn
import yaml

# Parameters
# ==================================================

# Data loading params
tf.flags.DEFINE_float("dev_sample_percentage", .1, "Percentage of the training data to use
for validation")

# Model Hyperparameters
tf.flags.DEFINE_boolean("enable_word_embeddings", True, "Enable/disable the word
embedding (default: True)")
tf.flags.DEFINE_integer("embedding_dim", 64, "Dimensionality of character embedding
(default: 128)")
tf.flags.DEFINE_string("filter_sizes", "3,4,5", "Comma-separated filter sizes (default:
'3,4,5')")
tf.flags.DEFINE_integer("num_filters", 64, "Number of filters per filter size (default: 128)")
tf.flags.DEFINE_float("dropout_keep_prob", 0.5, "Dropout keep probability (default: 0.5)")
tf.flags.DEFINE_float("l2_reg_lambda", 0.0, "L2 regularization lambda (default: 0.0)")
```

```
# Training parameters
tf.flags.DEFINE_integer("batch_size", 64, "Batch Size (default: 64)")
tf.flags.DEFINE_integer("num_epochs", 200, "Number of training epochs (default: 200)")
tf.flags.DEFINE_integer("evaluate_every", 100, "Evaluate model on dev set after this many
steps (default: 100)")
tf.flags.DEFINE_integer("checkpoint_every", 100, "Save model after this many steps
(default: 100)")
tf.flags.DEFINE_integer("num_checkpoints", 5, "Number of checkpoints to store (default:
5)")
# Misc Parameters
tf.flags.DEFINE_boolean("allow_soft_placement", True, "Allow device soft device
placement")
tf.flags.DEFINE_boolean("log_device_placement", False, "Log placement of ops on devices")

FLAGS = tf.flags.FLAGS
FLAGS._parse_flags()
print("\nParameters:")
for attr, value in sorted(FLAGS.__flags.items()):
    print("{}={}".format(attr.upper(), value))
print("")

with open("config.yml", 'r') as ymlfile:
    cfg = yaml.load(ymlfile)

dataset_name = cfg["datasets"]["default"]
if FLAGS.enable_word_embeddings and cfg['word_embeddings']['default'] is not None:
    embedding_name = cfg['word_embeddings']['default']
    embedding_dimension = cfg['word_embeddings'][embedding_name]['dimension']
else:
    embedding_dimension = FLAGS.embedding_dim

# Data Preparation
# ==================================================

# Load data
print("Loading data...")
datasets = None
if dataset_name == "mrpolarity":
    datasets =
data_helpers.get_datasets_mrpolarity(cfg["datasets"][dataset_name]["positive_data_file"]["
path"],
                                                    cfg["datasets"][dataset_name]["negative_data_file"]["path"])
```

```
elif dataset_name == "20newsgroup":
    datasets = data_helpers.get_datasets_20newsgroup(subset="train",
                                                     categories=cfg["datasets"][dataset_name]["categories"],
                                                     shuffle=cfg["datasets"][dataset_name]["shuffle"],
                                                     random_state=cfg["datasets"][dataset_name]["random_state"])
x_text, y = data_helpers.load_data_labels(datasets)

# Build vocabulary
max_document_length = max([len(x.split(" ")) for x in x_text])
vocab_processor = learn.preprocessing.VocabularyProcessor(max_document_length)
x = np.array(list(vocab_processor.fit_transform(x_text)))

# Randomly shuffle data
np.random.seed(10)
shuffle_indices = np.random.permutation(np.arange(len(y)))
x_shuffled = x[shuffle_indices]
y_shuffled = y[shuffle_indices]

# Split train/test set

dev_sample_index = -1 * int(FLAGS.dev_sample_percentage * float(len(y)))
x_train, x_dev = x_shuffled[:dev_sample_index], x_shuffled[dev_sample_index:]
y_train, y_dev = y_shuffled[:dev_sample_index], y_shuffled[dev_sample_index:]
print("Vocabulary Size: {:d}".format(len(vocab_processor.vocabulary_)))
print("Train/Dev split: {:d}/{:d}".format(len(y_train), len(y_dev)))


# Training
# ==================================================

with tf.Graph().as_default():
    session_conf = tf.ConfigProto(
      allow_soft_placement=FLAGS.allow_soft_placement,
      log_device_placement=FLAGS.log_device_placement)
    sess = tf.Session(config=session_conf)
    with sess.as_default():
        cnn = TextCNN(
            sequence_length=x_train.shape[1],
            num_classes=y_train.shape[1],
            vocab_size=len(vocab_processor.vocabulary_),
            embedding_size=embedding_dimension,
            filter_sizes=list(map(int, FLAGS.filter_sizes.split(","))),
            num_filters=FLAGS.num_filters,
            l2_reg_lambda=FLAGS.l2_reg_lambda)

        # Define Training procedure
        global_step = tf.Variable(0, name="global_step", trainable=False)
        optimizer = tf.train.AdamOptimizer(1e-3)
        grads_and_vars = optimizer.compute_gradients(cnn.loss)
        train_op = optimizer.apply_gradients(grads_and_vars, global_step=global_step)

        # Keep track of gradient values and sparsity (optional)
        grad_summaries = []
        for g, v in grads_and_vars:
            if g is not None:
                grad_hist_summary = tf.summary.histogram("{}/grad/hist".format(v.name), g)
                sparsity_summary = tf.summary.scalar("{}/grad/sparsity".format(v.name),
tf.nn.zero_fraction(g))
                grad_summaries.append(grad_hist_summary)
                grad_summaries.append(sparsity_summary)
        grad_summaries_merged = tf.summary.merge(grad_summaries)

        # Output directory for models and summaries
        timestamp = str(int(time.time()))
        out_dir = os.path.abspath(os.path.join(os.path.curdir, "runs", timestamp))
        print("Writing to {}\n".format(out_dir))

        # Summaries for loss and accuracy
        loss_summary = tf.summary.scalar("loss", cnn.loss)
        acc_summary = tf.summary.scalar("accuracy", cnn.accuracy)

        # Train Summaries
        train_summary_op = tf.summary.merge([loss_summary, acc_summary,
grad_summaries_merged])
        train_summary_dir = os.path.join(out_dir, "summaries", "train")
        train_summary_writer = tf.summary.FileWriter(train_summary_dir, sess.graph)

        # Dev summaries
        dev_summary_op = tf.summary.merge([loss_summary, acc_summary])
        dev_summary_dir = os.path.join(out_dir, "summaries", "dev")
        dev_summary_writer = tf.summary.FileWriter(dev_summary_dir, sess.graph)

        # Checkpoint directory. Tensorflow assumes this directory already exists so we need to
create it
```

```
        checkpoint_dir = os.path.abspath(os.path.join(out_dir, "checkpoints"))
        checkpoint_prefix = os.path.join(checkpoint_dir, "model")
        if not os.path.exists(checkpoint_dir):
          os.makedirs(checkpoint_dir)
        saver = tf.train.Saver(tf.global_variables(), max_to_keep=FLAGS.num_checkpoints)

        # Write vocabulary
        vocab_processor.save(os.path.join(out_dir, "vocab"))

        # Initialize all variables
        sess.run(tf.global_variables_initializer())
        if FLAGS.enable_word_embeddings and cfg['word_embeddings']['default'] is not None:
          vocabulary = vocab_processor.vocabulary_
          initW = None
          if embedding_name == 'word2vec':
            # load embedding vectors from the word2vec
            print("Load word2vec file
{}".format(cfg['word_embeddings']['word2vec']['path']))
            initW = data_helpers.load_embedding_vectors_word2vec(vocabulary,
                                                                 cfg['word_embeddings']['word2vec']['path'],
                                                                 cfg['word_embeddings']['word2vec']['binary'])
            print("word2vec file has been loaded")
          elif embedding_name == 'glove':
            # load embedding vectors from the glove
            print("Load glove file {}".format(cfg['word_embeddings']['glove']['path']))
            initW = data_helpers.load_embedding_vectors_glove(vocabulary,
                                                              cfg['word_embeddings']['glove']['path'],
                                                              embedding_dimension)
            print("glove file has been loaded\n")
          sess.run(cnn.W.assign(initW))

        def train_step(x_batch, y_batch):
          """
          A single training step
          """
          feed_dict = {
            cnn.input_x: x_batch,
            cnn.input_y: y_batch,
            cnn.dropout_keep_prob: FLAGS.dropout_keep_prob
          }
          _, step, summaries, loss, accuracy = sess.run(
            [train_op, global_step, train_summary_op, cnn.loss, cnn.accuracy],
            feed_dict)
          time_str = datetime.datetime.now().isoformat()
          print("{}: step {}, loss {:g}, acc {:g}".format(time_str, step, loss, accuracy))
          train_summary_writer.add_summary(summaries, step)

        def dev_step(x_batch, y_batch, writer=None):
          """
          Evaluates model on a dev set
          """
          feed_dict = {
            cnn.input_x: x_batch,
            cnn.input_y: y_batch,
            cnn.dropout_keep_prob: 1.0
          }
          step, summaries, loss, accuracy = sess.run(
            [global_step, dev_summary_op, cnn.loss, cnn.accuracy],
            feed_dict)
          time_str = datetime.datetime.now().isoformat()
          print("{}: step {}, loss {:g}, acc {:g}".format(time_str, step, loss, accuracy))
          if writer:
            writer.add_summary(summaries, step)

        # Generate batches
        batches = data_helpers.batch_iter(
          list(zip(x_train, y_train)), FLAGS.batch_size, FLAGS.num_epochs)
        # Training loop. For each batch...
        for batch in batches:
          x_batch, y_batch = zip(*batch)
          train_step(x_batch, y_batch)
          current_step = tf.train.global_step(sess, global_step)
          if current_step % FLAGS.evaluate_every == 0:
            print("\nEvaluation:")
            dev_step(x_dev, y_dev, writer=dev_summary_writer)
            print("")
          if current_step % FLAGS.checkpoint_every == 0:
            path = saver.save(sess, checkpoint_prefix, global_step=current_step)
            print("Saved model checkpoint to {}\n".format(path))
```

3.Code for SVM and Naïve Bayes on 20 Newsgroup

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.linear_model import SGDClassifier
from sklearn import metrics

import numpy as np

random_state = 42

categories = ['alt.atheism', 'comp.graphics', 'sci.med', 'soc.religion.christian']

""" Retrieve train newsgroup dataset """
twenty_train = fetch_20newsgroups(subset='train',
                                  # remove=('headers', 'footers', 'quotes'),
                                  categories=categories,
                                  shuffle=True,
                                  random_state=random_state)

print("Following newsgroups will be tested: {}".format(twenty_train.target_names))
print("Number of sentences: {}".format(len(twenty_train.data)))

""" Retrieve test newsgroup dataset """
twenty_test = fetch_20newsgroups(subset='test',
                                 # remove=('headers', 'footers', 'quotes'),
                                 categories=categories,
                                 shuffle=True,
                                 random_state=random_state)
docs_test = twenty_test.data

""" Naive Bayes classifier """
bayes_clf = Pipeline([('vect', CountVectorizer()),
                      ('tfidf', TfidfTransformer()),
                      ('clf', MultinomialNB())
                      ])
bayes_clf.fit(twenty_train.data, twenty_train.target)
""" Predict the test dataset using Naive Bayes"""
predicted = bayes_clf.predict(docs_test)
print('Naive Bayes correct prediction: {:4.2f}'.format(np.mean(predicted == twenty_test.target)))
print(metrics.classification_report(twenty_test.target, predicted,
target_names=twenty_test.target_names))

""" Support Vector Machine (SVM) classifier"""
```

```
svm_clf = Pipeline([('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', SGDClassifier(loss='hinge', penalty='l2', alpha=1e-3, n_iter=   5, random_state=42)),
])
svm_clf.fit(twenty_train.data, twenty_train.target)
""" Predict the test dataset using Naive Bayes"""
predicted = svm_clf.predict(docs_test)
print('SVM correct prediction: {:4.2f}'.format(np.mean(predicted == twenty_test.target)))
print(metrics.classification_report(twenty_test.target, predicted,
target_names=twenty_test.target_names))

print(metrics.confusion_matrix(twenty_test.target, predicted))
```