NEURAL NETWORK ARCHITECTURES FOR SHORT TEXT

By

Dylan Elliott

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Major Subject: COMPUTER SCIENCE

Examining Committee:

Dr. Mohammed Zaki, Thesis Adviser

Dr. Charles Stewart, Member

Dr. Alex Gittens, Member

Rensselaer Polytechnic Institute
Troy, New York

November 2017
(For Graduation December 2017)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

I extend my sincerest gratitude to my adviser, Professor Mohammed Zaki, for his knowledge, guidance and support throughout my study for this work. Dr. Zaki has provided me with a balance of instruction and freedom to which I have been able to explore my interests and expand them into many new topics.

I extend many thanks to my reviewing committee consisting of Dr. Mohammed Zaki, Dr. Charles Stewart and Dr. Alex Gittens for kindly offering their time and attention to the presentation of this work. Their questions and feedback are a valuable resource that I sincerely appreciate.

I also thank the United States Air Force and their Science and Engineering PALACE Acquire program for providing the funding for this degree and a job in which I can enjoy the continuation of exploring my research interests.

Finally, I extend deep thanks to my family, for their endless support for my goals and aspirations. They have selflessly provided the guidance to allow me to pursue a career that I will enjoy for the rest of my life.

ABSTRACT

This thesis presents a comparison of four commonly used neural network models for learning to classify and encode short text sequences. We first evaluate the performance of the models for a supervised classification task on three short text datasets. The results of these tests suggest that performance can be dependent on a combination of the model architecture and the complexity of features desired to be learned. We then train each model on a semi-supervised learning task with a K-means clustering objective for one of the short text datasets, after which we encode the dataset with the trained models and perform clustering on the encoded representations. The results of clustering reveal that a model's performance in the classification task does not necessarily correlate positively to its performance in the semi-supervised task and we relate these observations to data about each model's behavior during learning. Overall we find that if a model does not learn to largely separate its feature representations too quickly, it may have a better chance at clustering due to an increased ability to correct initial alignment mistakes. These insights provide guidance to future work in which more complex models will be used and knowledge bases will be constructed using raw text scraped from the web.

1. INTRODUCTION

1.1 Purpose

We perform a comparison of four neural network architectures used for natural language understanding (NLU) by reviewing their performance on several learning tasks with short text datasets. Specifically, given a labeled dataset of short text sequences (sentences), we use several variants of either a fully-connected neural network (FCNN), a convolutional neural network (CNN) or a recurrent neural network (RNN) to classify the data or learn useful representations for the data through a semi-supervised task. A comparative study of model behavior and performance during each task reveals useful insights into the strengths and shortcomings of each architecture.

It is a well-known fact that efficiently representing short text sequences to a machine is a challenging task, therefore we expect the performance of most NLU models to be largely dependent on the fashion by which the data is presented. Additionally, we observe that the greedy nature of the usually unbounded task setup for neural network training tends to create a bias towards learning spatially dependent features. With careful task design and slight supervision, a neural network can be coerced to extract higher level features from data.

1.2 Significance

Humans and machines continuously work together. The ability of a machine to perform natural language understanding (NLU) tasks at a human level is pivotal in the advancement of artificially intelligent systems and human knowledge. Since it is common to use human-level metrics to evaluate machine learning models, it is only logical that machines should understand the dominant way in which humans communicate and store information—written and spoken natural language. The motivation for using neural architectures for NLU tasks comes from the fact that these models automatically learn to extract relevant features from their input data through an end-to-end trainable task. This offloads the otherwise daunting task of performing manual feature extraction on text data and allows for gradient descent optimization.

Natural language understanding is quickly becoming a requirement for state-of-the art technology that is included in many products from automobiles and consumer electronics to large scale data mining systems. More and more businesses are integrating dialog systems into their products so that their users can enjoy a personal experience rather than having to navigate a website. Successful dialog systems must be able to hold a realistic conversation while helping a user reach their goals. This includes understanding low-level features like sentence topics, understanding high-level features like the goals, or *intents*, within questions, remaining robust to noise like improper grammar and spelling errors, and being specific enough to detect small differences in the inputs like synonyms and changing context. In data mining systems, access to the information stored in text is extremely beneficial. There is a vast amount of information stored in natural language, but only a fraction is conveniently available to us while the rest is contained in high-level semantic forms. Discovery of this information is critical for next generation knowledge extraction systems and the solution to solving this problem has been challenging for machines and humans because of the highly expressive properties of language.

1.3 Difficulty

Consider the sentence, “What was Bob thinking?”. We can quickly detect that this sentence is syntactically correct, and that it is actually a question. It is slightly more challenging to break the sentence into the important words *what*, *Bob* and *thinking*, in order to interpolate that the intent within this question may be to determine the specific thoughts that are in Bob’s head. With prior knowledge about the semantics of modern language, we could speculate that this sentence may not really be a question at all, but more of a statement expressing one’s dislike towards Bob’s decisions. Some NLU systems would have trouble reaching this consideration since none of the words in the sentence directly indicate any negative sentiment towards the subject of the sentence (Bob). It is significantly more challenging to identify the mechanisms responsible for detecting these high-level semantic features from such a simple sequence of words. Many of the detected features that allow these hypotheses to be made by humans, ironically remain elusive to us. This suggests that successful NLU models should discover these features automatically, without the explicit prior knowledge of their existence.

Now consider the sentence, “What was Bob thinking about?”. The sentiment of this new sentence is very different than that of the original, although the literal intent of the two sentences is essentially identical. It is difficult to target which features are responsible for the difference in perception between the two sentences above, therefore attempting to create a rule-based system to detect these differences would probably not generalize well to similar cases in the future. Instead, machine learning methods are used to extract useful features in an unsupervised manner, targeting them through clever design of goals and constraints within the problem.

Since most data is not labeled with respect to the unknown features being investigated, it is common to group data based on learned similarity measures that attempt to capture dependencies between these features. This is the motivation behind clustering, in which data sharing common features is grouped together, usually by transformation to a new space that linearly separates feature dependencies. Discovering this transformation space is a challenging problem, since prior knowledge of the transformation space implies knowledge about the specific features we are looking for.

Currently, many NLU models that achieve state-of-the art performance involve “enriching” text with heavily engineered rule-based systems that require extensive data preprocessing in order to expose useful features [1], [2], [3], [4]. In contrast, recent success is drawing attention towards NLU models that utilize neural network architectures that can learn to automatically detect useful features from text [5], [6], [7], [8], [9]. Analogous to the expansion in progress made while using neural architectures for image analysis, these models also demonstrate their powerful feature extraction abilities on natural language datasets. We focus mainly on solutions of this latter form for the comparisons accomplished in this research.

1.4 Language Space

The space in which natural language resides is very different than that of other data forms like images and sounds. Images and sounds reside within a continuous space whereas the structure of natural language is non-continuous and logically complex, consisting of values only for words that exist within a quantized vocabulary. Short sentences produce additional difficulties, since they comprise a small subset of a relatively large vocabulary. Also, the frequency of which words are used within a vocabulary is very unbalanced, as words that syntactically bind sentences (*the*, *if*, *and*, etc.) are used many more times than most other words. Words that are used very rarely are sometimes vital to understanding important aspects about a sentence, especially when that sentence contains few words.

The properties above violate assumptions that are made for a variety of machine learning problems. Therefore, some models that achieve great performance on non-sparse datasets, like images or very large documents, end up performing poorly when given a short text dataset. It is common practice to attempt to mitigate the effects of sparseness altogether by representing words as vectors rather than indices of a vocabulary. Additionally, many pre-processing techniques exist to shorten vocabularies, such as the removal of infrequent words, words that carry little information and words that are redundant. We will discuss these issues and solutions in more detail in later sections.

1.5 Contributions

This work highlights some of the advances in using neural networks for the extraction of information from text. We perform a survey of some of the traditional methods used for representing text data to machines, and we investigate the problems that result when the data consists of short sentences. The main contribution of this work involves a comparison of several neural network models that achieve state-of-the-art performance in NLU tasks with short text datasets. Initially, we compare models on a fully supervised classification task and we analyze the quality of automatic feature extraction through component analysis. We then use the models to encode text sequences into a low-dimensional embedding space using a semi-supervised learning task. Clustering is performed in this low-dimensional space and results are compared to clustering on traditional text representation methods. We focus on evaluating feature encoded data with clustering techniques because we would like to investigate the ability of the models to learn feature vectors that are separable in the low-dimensional embedded space. From this analysis, we discern relationships from common success points and failures in order to provide insight into the underlying structure of natural language. We conclude by discussing possible improvements that could be made in the quality of extracted features from short text.

2. BACKGROUND

2.1 Clustering

Clustering is the process of organizing data into groups, in which data samples from the same group are more similar to each other than data samples from different groups. The choice of how to represent the data and the measure used for "similarity" is a heavily researched topic involving many different approaches. Usually, data is represented by a vector of extracted features and clustering is performed on these feature vectors. The similarity measure used can range from simple Euclidean distances in vector space, cosine similarity measures, or any other operation that can capture desired similarity relationships between the feature representation of the data samples. Due to the motive of organizing data into groups, clustering is usually performed in an unsupervised manner in which labeled data is unavailable, but techniques exist that leverage small amounts of labeled data for semi-supervised approaches. Ideally, the clusters formed from encoded feature vectors would take the form of simple, convex shapes that can be linearly separated in the feature space.

2.1.1 K-means Algorithm

The K-means clustering algorithm involves grouping samples in feature space by their Euclidean distance to a set of "representative" points in the feature space. These representative points take the values of the cluster means and are continuously updated as samples in the feature space are assigned to their clusters. Initially, the cluster means are chosen randomly and samples in feature space are assigned to the cluster corresponding to the closest cluster mean. After assigning all samples, the cluster means are updated to the mean of all the samples currently belonging to the cluster and the process is repeated to assign each of the samples to clusters corresponding to the updated means. The algorithm is stopped once the difference between cluster means from two consecutive iterations is below some threshold, indicating that samples have been stably assigned to clusters and not much reassignment is further occurring. Since the K-means algorithm uses a distance metric to discriminate between clusters, the algorithm works best for samples in feature space that form convex cluster shapes [10].

2.2 Neural Networks

Neural networks are machine learning models inspired by the networks of neurons that make up the brain. These models consist of a network of neural processing units connected together by weights that produce complex transformations of input data in order to minimize a task-specific objective function.

2.2.1 Fully Connected Neural Networks

A fully connected neural network (FCNN) consists of multiple layers of neurons that have connections only to neurons of adjacent layers in the network. These networks fall within the class of *feed-forward* neural networks because signals strictly propagate forward through the network from inputs to outputs, with no signals between neurons of the same layer. FCNN's have demonstrated the ability to learn mappings of complex, non-linear data to simpler spaces that can be linearly separated. Autoencoders (AE's) are neural networks that are trained to reconstruct the data provided to them as inputs [11]. Usually, an AE is implemented using a FCNN, but it is becoming more popular to use CNN's and RNN's for the main architecture. The input and output layers share the same dimensionality as the input data and any middle, or *hidden*, layers usually have lower dimensionality as the input data. The motivation for using an autoencoder is similar to that of using other dimensionality reduction techniques such as Principle Component Analysis (PCA), since the activations of the neurons in the hidden layer learn to respond to important features and create lower dimensional encodings of the input data.

2.2.2 Convolutional Neural Networks

Convolutional neural networks (CNN's) replace fully-connected neural network layers by sweeping small filters across inputs from a previous layer in order to produce activations for a current layer. The output of each layer within the CNN is calculated by convolving a set of filters for that layer with the output of the previous layer. Each filter is analogous to a neuron with a small amount of inputs (usually 9 or 25) being swept across a field of view as it produces output values from its inputs at each location during the sweep. In order to compress information, it is common to include an additional convolution at each layer that uses filters that output the maximum value of their inputs, or the average value from their inputs. These layers are called max pooling layers and average pooling layers respectively. It is common to include a fully-connected layer or two at the output of a CNN in order to improve expression and transform the output to a vectorized form. Due to the convolution operation and reuse of filters as they sweep across an input, CNN's can have drastically fewer parameters than fully-connected neural networks, yet tend to outperform many fully-connected neural networks models for tasks like image recognition and regression.

2.2.3 Recurrent Neural Networks

A recurrent neural network (RNN) uses feedback loops in order to process time-separated sequences of data alongside its outputs from previous time steps. The most basic RNN is a single neural network that produces an output at time t from its input at time t and its output at time $t-1$. When input sequences become long, this basic model can lose sight of context that is far back in time. Also, during training, gradients through long sequences of time have a tendency to vanish or explode, causing training to slow or even halt. One of the most commonly RNN models used to address these issues is Long Short-Term Memory (LSTM) [12]. The LSTM model was designed to reduce the effects of unstable gradients as well as improve the ability for an RNN to remember information far back in time. An LSTM network consists of multiple neural networks that can store information in weights like long-term memory as well as control how information flows sequentially through time like short-term memory.

2.3 Input Representations for Text and Documents

Systems built for NLU tasks start with a vocabulary, or a data structure in memory of a usually large but finite collection of words. This vocabulary is usually constructed from the words contained within a corpus of text sequences, or from a massive corpus of words scraped from online sources like Wikipedia. Since language vocabularies are very large, it is common practice to use techniques that minimize repetition and remove words of low-importance in attempt to minimize the size of the vocabulary. We give a brief overview of these simplifications below, as well as a review of some of the methods used to represent sentences given a vocabulary.

2.3.1 Vocabulary Simplification

A large contribution to the size of a vocabulary can be due to synonyms that share identical information, or frequently used words that contain little information, but provide syntactic functionality for the rest of the information-rich words. One method for greatly reducing the size of a vocabulary is suffix stripping [13] (usually referred to as *word stemming*), in which any extended words are reduced to their root word before constructing a vocabulary. For example the words, *ran*, *running*, *runner* and *runs* would all be converted to the root word, *run*, removing the need for explicit entries in the vocabulary for each of the variants that include a suffix. Additionally, frequently used words, or *stop-words*, that syntactically bind sentences like *the*, *if* or *and*, can be removed from text data before a vocabulary is constructed in order to reduce its size. Removal of these words may break proper syntax, but they carry little information and therefore most features present in the original text are unaffected. Careful consideration must still take place to decide which stop-words to remove—we certainly would not want to remove words like *where*, *when* or *who* when attempting to extract features used to cluster different types of questions.

2.3.2 Bag of Words

A bag of words (BOW) vector is a vocabulary length vector representation for a sequence of text. The i^{th} element of a BOW vector for a sequence of text takes a value equal to the amount of times the i^{th} word in the vocabulary occurs in that sequence—this is usually referred to as the *Term Frequency* weighting for that word. BOW vectors are sometimes normalized to have values between 0 and 1 and can be further processed to add importance to specific words that hold more information than others.

2.3.3 TF-IDF

It is common to encode a sequence of text with an *Inverse Document Frequency* (TF-IDF) vector, which is similar to BOW but takes the frequency of a word throughout an entire corpus of text sequences into account as well [14], [15]. As in BOW, the weights of the vector for a text sequence are initially calculated according to the term frequency. The weight given to each word's index is then inversely scaled by the frequency of that word appearing within the entire corpus of text sequences—called the *Document Frequency*. The formula we use for the TF-IDF weighting is shown by Equation 2.1, where w_i is the weight for the i^{th} index of the TF-IDF vector, t_i is the term frequency for word i, d_i is the document frequency for word i, N is the total number of documents or text sequences and $|V|$ is the size of the vocabulary. We can see from Equation 2.1 that the importance, or weight, of a word increases with increased frequency of use within a specific text sequence, and decreases with increased frequency of use throughout the entire corpus of text sequences.

$$w_i = t_i \log \frac{N}{d_i} \qquad \forall i = 1...|V| \tag{2.1}$$

2.3.4 Limitations

While interesting results have been achieved when representing long sequences of text with BOW and TF-IDF vectors, representing short sequences of text exposes the limitations of these techniques. Short sequences of text contain a very small subset of the overall vocabulary which results in sentences being represented by sparse high-dimensional vectors. These vectors mostly contain zero values, therefore it can be difficult to efficiently extract information from these representations. BOW and TF-IDF vectors also destroy word order when encoding a text sequence. In long text sequences, dropping word order may not be much of a problem, but for short text sequences the meaning of a sentence can be completely dependent on word order. Finally, since BOW and TF-IDF vectors cannot capture similarities between synonyms, sentences which consist of completely different words, but share the exact same meaning, will share little similarity in BOW or TF-IDF vector space.

2.3.5 Word Vectors

The problems faced with using BOW and TF-IDF vectors are addressed by representing each word within a text sequence with a unique vector of fixed dimension, in which related words result in similar vectors. One popular method for calculating word vectors that satisfies this requirement is the skip-gram model, in which an autoencoder is fed with single words from text and trained to reconstruct nearby surrounding words [16]. As a result, the hidden layer activations of the autoencoder learn to capture syntactic and semantic relationships between words commonly appearing next to each other in the training data. These relationships have even been shown to map arithmetic operations in the word vector space to correct predictions in the language space.

Word vectors usually have dimensions of magnitude 10^1 or 10^2 elements, therefore sparsity is not as much an issue. A sentence can be represented by a sequence of word vectors stored as a matrix and word order can be preserved over rows of the matrix. Additionally, since synonym relationships can be reflected in word vectors, the ability to determine similarity between same-meaning sentences with completely different words is improved. An example of encoding a short text sequence into a word vector matrix, or *word embedding*, can be seen in Figure 2.1. A row in the word embedding is a word vector and a column represents a single feature from the word vector. There are $n = 4$ words for the sentence embedded in Figure 2.1 and each word is encoded by a word vector $w_i \in \mathcal{R}^d$, therefore the embedding is in $\mathcal{R}^{n \times d}$.

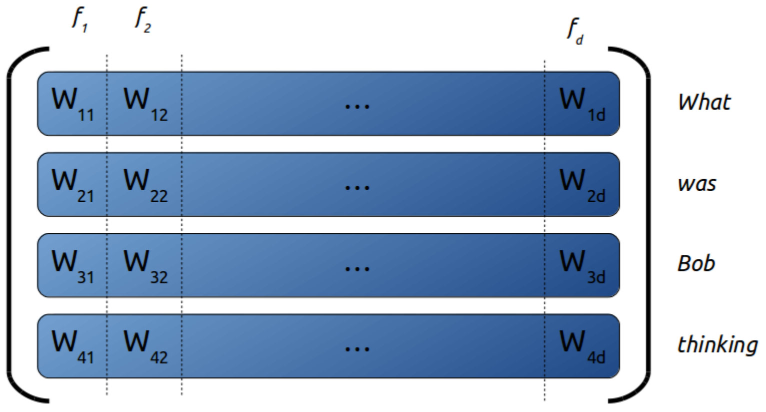

f_1
f_2
f_d
W_{11} W_{12} ... W_{1d}
W_{21} W_{22} ... W_{2d}
W_{31} W_{32} ... W_{3d}
W_{41} W_{42} ... W_{4d}
What
was
Bob
thinking

Figure 2.1: Word embedding matrix.

2.4 Features in Text

A common data preprocessing technique used in the field of NLU is to manually extract features from text in order to“enrich” its representation and amplify important information. Syntactic features represent the grammatical structure of text, therefore these features can usually be extracted efficiently using a rule-based method. Semantic features represent high-level abstract ideas, therefore these features are more difficult to extract and manually defining them is a challenge.

2.4.1 Syntactic Features

Some of the most common syntactic features that are extracted from text data are part-of-speech tags (POS tags), which are word-type categories (noun, adjective, verb, etc.) assigned to each word in a text sequence. From POS tags, higher-level syntactic features can be constructed from groups or patterns of specific tags—as performed in [1]. The information that these features represent is largely dependent on the physical structure of text, therefore it is hard to draw conclusions about higher-level ideas from this information alone.

2.4.2 Semantic Features

There are many semantic features that can be extracted from text, including more that have yet to be discovered. Basic semantic features for text can include topic, sentiment, intention, and if the text is a question, question-type and answer-type [1], [2], [3]. Topic features describe the field of raw information discussed within a text sequence, and can usually be extracted using rule-based methods. Sentiment describes a text sequence by either having a positive attitude or a negative attitude, requiring a slightly richer understanding of the relationships between words in the text. Intention indicates the goals of the text sequence, or the person who produced the text sequence, requiring not only an understanding of the relationships between words in the text, but prior knowledge of patterns present in the language as a whole. Question-type and answer-type features describe the category of information desired from a text sequence query and are closely related to intention.

3. RELATED WORK

3.1 Text Enrichment

There are many research efforts gaining progress in the field of text representation for machines and many popular methods involve the use of manually designed feature extractors for text enrichment. The text is then represented in the new data space defined by these targeted features.

The work in [1] shows the benefits of text enrichment with a model that achieves state-of-the-art results on a coarse-grained question-type and fine-grained question-type classification task. The coarse-grained task contains 6 question-type categories whereas the fine grained task contains 50 question-type categories, derived as sub-categories of the coarse labels. Text enrichment is achieved by representing text samples with one of six *feature-sets*. These feature-sets are derived using a series of manual lower-level feature extraction techniques, such as POS-tagging, named entity tagging and pre-trained classifiers from previous work done by the authors. The text data used for training the model in [1] was manually labeled by the authors and is available for use online [17]. The original questions used for the text data were collected from the TREC competition datasets [18]. Since the work in [1], many approaches for the question-type classification have emerged that similarly involve pre-processing steps of manual feature extraction [19], [20], [21].

The question-type attribute extracted above is leveraged as one of several higher-level features used to represent text for cluster analysis in [2]. In this study, K-means clustering is performed on Yahoo Answers queries by pre-processing the short text sequences through a carefully designed feature generator stream. The enriched representations are then clustered for the analysis. The representation for a short text sequence is a combination of tagged key-topic words, tagged answer-type words, tagged words that modify key-topic or answer-type words, question-type classification, synonyms, and relevance measures returned by search engine queries. Although the authors show that this feature representation can efficiently group syntactically and semantically related short text sequences, the overhead to do so requires the explicit organization of many optimized subsystems.

The work in [3] presents an answer detection and ranking model trained on a dataset of question answer pairs. This model uses many of the same manual feature extraction techniques discussed above in order to train a model to select a group of passages that may contain the answer to an input question.

3.2 Neural Networks and Text Data

It is evident that the understanding of short text data can be improved by pre-processing the text to expose targeted features, but manually identifying some targeted features can be a very challenging task. Several research efforts exist to leverage deep neural networks in order to extract useful features from short texts in an automated fashion.

Although not technically tested with short text datasets, the results in [22] demonstrate that autoencoders equipped with constraints on their hidden states can perform quite well in document classification tasks where some of the documents can be considered relatively short. The data used in the study includes documents with lengths varying from about 2% of the total vocabulary length to about 60% of the total vocabulary length. Although text sequences that are 2% the length of their vocabulary sizes may seem small, the average length of the short text sequences we focus on in this work vary from 0.1% the length of their vocabulary down to 0.01% the length of their vocabulary, which can introduce major issues with sparsity.

To avoid issues of sparsity without needing to prune many non-frequent words from a vocabulary, most neural network models trained on text datasets accept their data in the form of the word embedding in 2.1. Ideally, the word vectors that make up the rows of this matrix are sourced from a library of pre-trained vectors that capture semantic relationships between specific words. Many times though, the word vectors are randomly initialized as part of the model parameters that need to be updated during training. The latter approach may sacrifice a few points of accuracy on some datasets, but allows for training on datasets in which there are no pre-trained word vectors of good quality.

The work in [7] shows that a relatively shallow CNN model can achieve state-of-the-art performance on a number of short text NLU tasks including sentiment analysis and question-type classification. The model convolves an input word embedding with filters of varying height and constant width. This width is equal to the width of the word embedding, producing a single row vector of feature-map outputs for each filter. These outputs are fed to a non-linear activation function and max pooling is performed over the entire activation vector of each filter. Since the max pooling is not computed spatially, but rather over the time dimension of the convolved sequence, this pooling scheme is called *max-over-time pooling* [23] and allows for easy handling of input sequences with varying lengths. Each of these pooling operations produces one output per filter and these outputs are organized into a vector which is fed to a fully connected output layer. By this design, each filter is trained to respond to a specific and continuous sub-sequence of words, or n-gram, and produce an output value conditioned on the presence of that sub-sequence feature.

A comparison between using RNN's and CNN's for several NLU tasks is performed in [24]. In particular, the study uses the same CNN model from [7] and demonstrates that the model performs comparatively to some popularly used RNN models. The authors highlight strengths and weaknesses of both models, specifically noting that the CNN performs as well if not better than RNN models in the tasks that involve short text sequences. They conclude that this is due to the CNN being particularly tuned towards responding to key words from a text sequence, whereas the RNN models tend to create more generalized representations of the sequence as a whole. In contrast, the RNN's tested generally show a slight increase in performance compared to the CNN when tasked with longer input text sequences.

A deeper, slightly more complicated CNN model for NLU understanding is proposed in [6], which introduces multiple layers of *wide convolutions* and a newly defined *k-max pooling* operation which is applied to the convolved feature-maps at each layer. This model, called the dynamic CNN (DCNN), accepts a word embedding matrix identical to the format in [7]. At each layer, including the input layer, padding is added to the top and bottom of the input from the previous layer and convolutions are computed with filters with unit width. Adding the padding along the height dimension of the input represented to each layer allows for the filters to better consider information at the top and bottom boundaries of these representations. The resulting output from a convolution layer is no longer a 1D vector, as in [7], but instead a matrix feature-map that has the same number of columns as the pre-convolved representation, but with a few more rows—hence the name *wide convolution*. The feature-map of each convolution layer is passed to a non-linear activation function and *k-max pooling* is performed on these activations. The *k-max pooling* operation is similar to max-over-time pooling, but instead selects the top k values along the time dimension of the convolved sequence. This operation always produces an output representation with k rows and as many columns as the representation before pooling. The value of k is a function of the current layer within the CNN and its value is diminished to a final minimum value as the final layer of the network is reached. Before the pooling performed at each layer, the authors also introduce an optional *folding* operation, in which every two columns of the activation matrix from that layer are summed. This operation can be used to introduce interdependencies between spatial features in the convolved representations at each layer, without adding any more parameters to the model. Like most CNN models, the final output layer of the DCNN involves vectorizing feature-maps to feed to a fully connected layer. The work in [6] demonstrates that the DCNN model described above achieves state-of-the-art performance on question-type classification tasks and sentiment prediction tasks on short text datasets.

The authors of [5] use the CNN model from [7] and an LSTM model to perform semi-supervised clustering on several labeled short text datasets such as question-types and topics for news article titles. For the LSTM model used in the study, the output is taken as the average of its hidden states from every input word vector of the input text sequence. The models are trained on a semi-supervised clustering task, in which a K-means objective is minimized jointly with a classification objective from a subset of labeled data. A comparative study demonstrates that the CNN model slightly outperforms the LSTM for all datasets tested on the semi-supervised clustering task.

In [8], the DCNN from [6] is trained on short text data and supervised with pre-calculated, binary encodings of the input samples. The binary encodings are constructed by computing locality-preserving projections (outlined in [25]) of keyword features, followed by thresholding the projections on their medians in order to produce the binary values. The input data is then encoded by the last hidden layer of the trained DCNN and K-means clustering is performed on these encodings. The results from [8] show that the DCNN can learn useful encodings for short text without the knowledge of labels, but the quality of the locality-preserving projections are largely dependent on the quality of the keyword features being projected.

The techniques used in [5] and [8] both require a level of supervision that drives the network towards learning a constrained distribution. This supervision may be sufficient to achieve state-of-the-art results as long as the chosen method of supervision is reflective of the semantic relationships we would like to capture between the inputs. In [5], labels are chosen for the supervision, therefore it is clear that the model will be constrained to learn relationships that are relevant to the task. In [8], it may be sufficient to use distances between BOW vectors to capture useful topic-relevant relationships because topics for short text sequences are highly dependent on shared words between the sequences. For higher-level features such as the intent of a question or question-type identification, a BOW distance supervision metric will struggle to capture similarities on sentence pairs that do not share any words.

4. METHODS COMPARED

We compare the performance of four neural neural network models for the short text semi-supervised learning task outlined in [5]. This semi-supervised task is chosen over the learning task in [8] because it is more suited for a wide range of feature extraction learning, rather than depending on what can be supervised from locality measurements within the high-dimensional input space. The four models we compare are selected from the work outlined above and presented with three short text datasets that are fully labeled. In this section we first describe in detail the architecture of each of the four neural network models we test, including any changes made to accommodate our task set-up.

4.1 Neural Bag of Words Network

The neural bag of words (NBOW) model that we use is a standard feed-forward neural network that accepts a vectorized text sequence as input. The architecture for this model is outlined in Figure 4.1 and contains one fully connected hidden layer and one fully connected output layer. An input auxiliary layer is added in order to construct a vectorized text sequence from a word embedding matrix. Traditionally, BOW and TF-IDF vectors have been used as the sentence vector for this type of model rather than vectorizing from a word embedding matrix. When vocabulary sizes are very large though, this approach is not very feasible unless the dictionary is heavily pruned to decrease its size. Even for relatively smaller vocabulary sizes, as we will demonstrate using one of the datasets, BOW and TF-IDF vectors are still too sparse to construct useful representations.

For the NBOW model in Figure 4.1, we vectorize the input word embedding matrix of n word vectors in $\mathcal{R}^d$ by computing the average word vector over the rows of the matrix. For the input word embedding in $\mathcal{R}^{n\times d}$, this produces one sentence vector in $\mathcal{R}^i$ where $i = d$. This sentence vector is provided as the input to a fully connected hidden, or *latent* layer to map the sentence vector to a representation in $\mathcal{R}^h$. This representation is fed to a final output layer which maps the latent

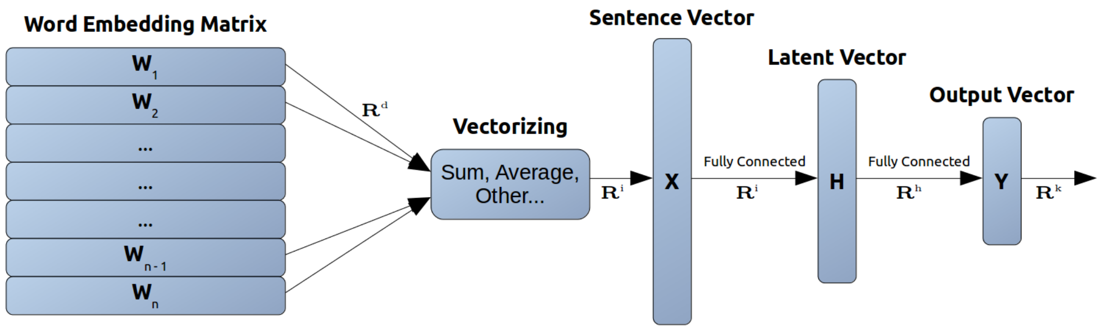


Figure 4.1: Neural bag of words architecture.

representation to an output representation in $\mathcal{R}^k$. The latent layer uses hyperbolic tangent activation for its neurons and the output layer provides linear activation to a classification objective that computes the softmax of the activations. Both fully connected layers also include dropout [26], in which a neuron will remain active during training with some set *keep probability*. Dropout has shown to reduce the negative effects of over-fitting by forcing neurons not to depend too much on other neurons during the training phase.

We include two fully connected layers at the output of the NBOW model because we would like explicit control over the parameter h (the dimensionality of latent vectors encoded by the model). This allows us to encode input text sequences in varying spatial dimensions by extracting the latent vectors of the trained model before they are passed to the output classification layer. For a dataset with K class labels, the output layer used for the classification tasks is always in $\mathcal{R}^k$.

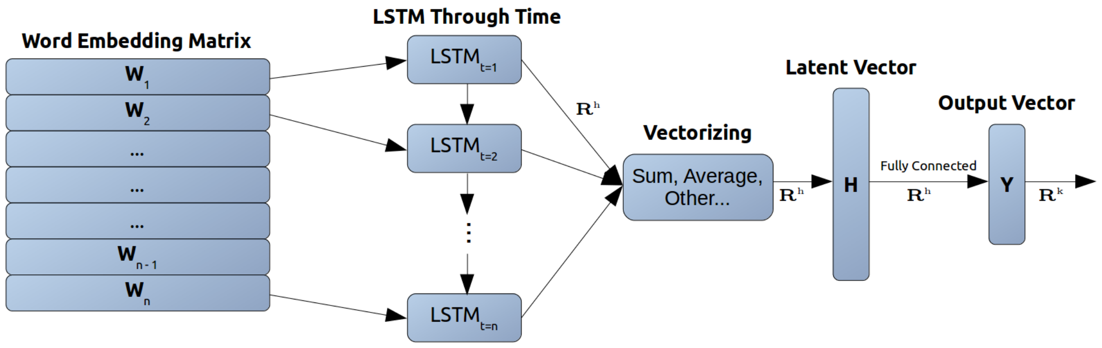


Figure 4.2: Long short-term memory architecture.

4.2 Long Short-Term Memory Network

The LSTM model that we use is set up similarly to the one used in [5], in which the hidden states of the LSTM cell are averaged over time to construct a vector representation of the input text sequence. The architecture for this model is outlined in Figure 4.2. Instead of vectorizing the input word embedding as in the NBOW model, each word vector in the embedding is provided to the LSTM one time-step at a time, as if the LSTM were reading word-by-word from the embedded input sentence. At each time-step, the LSTM cell outputs a hidden state vector in $\mathcal{R}^h$ that is computed from current cell inputs and the hidden state from previous time-steps. After feeding the n word vectors from the input embedding, the resulting n hidden states are averaged to compute one vector in $\mathcal{R}^h$ which is the encoded latent vector of the input embedding. The latent representation is fed to a fully connected output layer to construct an output representation in $\mathcal{R}^k$. The hidden states of the LSTM cells use hyperbolic tangent activation and the output layer of the LSTM model provides linear activation to the same classification objective function used in the NBOW model.

Since we are free to choose the amount of hidden neurons in the LSTM cells of the model, we still have full control over the parameter h in order to control the dimension of the encoded latent space. Also similar to the NBOW model, the output layer dimensionality matches the number of classification labels, k for the input datasets. Like the NBOW model, we are free to use the LSTM model as an encoder by outputting the trained latent representations in $\mathcal{R}^h$, or as a classifier by using the actual model output representations in $\mathcal{R}^k$.

4.3 Max-Over-Time Convolutional Neural Network

The max-over-time convolutional neural network, which we will call the Text CNN (TCNN) from now on, is designed according to the model in [7] and the architecture for this model is outlined in Figure 4.3. Instead of vectorizing the input embedding or consuming the input embedding one word vector at a time, the TCNN performs convolutions over the embedding with filters that have varying height and the same width as the word embedding. We add the padding used in [6] to make

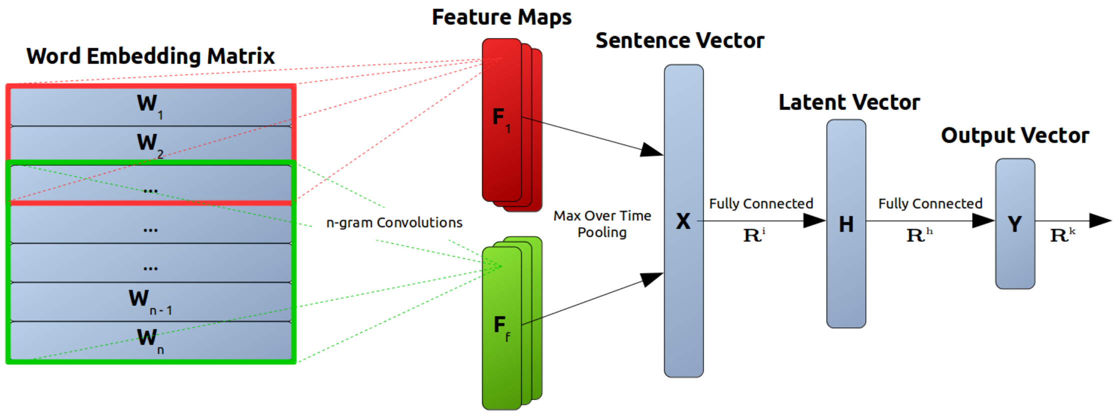


Figure 4.3: Max-over-time convolutional architecture.

the convolutions *wide convolutions* along the rows of the embedding which allows each filter to better consider word vectors at the top and bottom borders of the embedding. For the model we test, we follow the choice of convolution filters in [5] in which there are 500 filters of height 1, 500 filters of height 2 and 500 filters of height 3. Convolution of the input embedding with each filter produces a vector of outputs which are converted to activations with the rectified linear activation function, $relu(x) = max(x, 0)$. Max-over-time pooling is computed on each one of these vector activations, therefore each filter contributes one output value after the pooling operation, resulting in 1500 output values. These output values are vectorized and fed to a fully connected layer with hyperbolic tangent activation that produces a latent representation for the input embedding in $\mathcal{R}^h$. This latent representation is fed to a final fully connected layer with linear activation to produce an output representation for the model in $\mathcal{R}^k$. Once again, by choosing the number of neurons in the latent fully connected layer, we are free to control the dimension of the encoded latent space. This allows us to use the TCNN in a similar manner as the NBOW and LSTM models—either as an encoder to represent input text sequences in $\mathcal{R}^h$ or a classifier by using the actual model output representations in $\mathcal{R}^k$.

4.4 Dynamic Convolutional Neural Network

The dynamic convolutional neural network (DCNN) is designed according to the model in [6] and the architecture of this model is outlined in Figure 4.4. Like

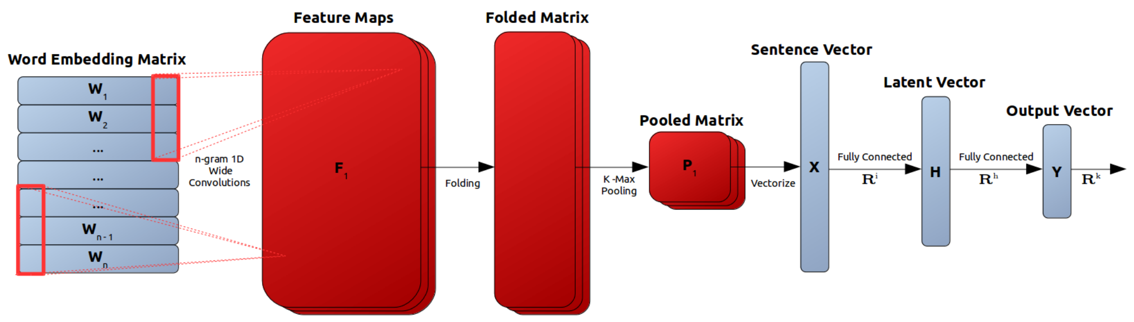


Figure 4.4: Dynamic convolutional architecture.

the TCNN model, the DCNN performs *wide convolutions* over the input embedding to produce a number of feature-maps. For the model we test, we use one convolutional layer and feed the vectorized output of the convolutional layer to two fully connected layers to produce a latent representation and an output representation. The architecture in Figure 4.4 shows only one convolutional layer consisting of the feature-maps, folded matrices and pooled matrices. More than one convolutional layer can be added by convolving another set of features with the pooled matrices to produce a deeper set of secondary feature-maps, folded matrices and pooled matrices. The filters used for convolution in the DCNN have unit width, therefore the feature-maps produced by convolving a matrix with these filters have the same width as the original matrix. Since *wide convolutions* are used, the resulting feature-maps become taller than the matrix that is convolved. We follow the DCNN design in [6] in which the model has only one convolutional layer with 5 filters of height 8 and the value of k for k-max pooling is set to 5. The design uses hyperbolic tangent at all layers and is shown to achieve state-of-the art performance on the Question-Type dataset. The output of the final convolutional layer is vectorized and fed to the first fully connected layer with hyperbolic tangent activation to produce a latent representation in $\mathcal{R}^h$. This latent representation is fed to a final fully connected layer to produce an output with linear activation in $\mathcal{R}^k$. We are able to use the DCNN in a similar manner as the other 3 models, either by using it as an encoder or as a classifier.

5. EXPERIMENTS PERFORMED

In this section, we present the supervised learning task and the semi-supervised learning task we use to construct a fair test bed for the four neural network models described in the previous section. Additionally, we describe the task of clustering the text data using the low-dimensional latent representations learned from the semi-supervised task and we outline the methods used for the evaluation of the clustering results.

5.1 Supervised Learning Task

To compare the performance of the four models described above, we test them on a fully supervised classification task with three short text datasets. The controlled classification task involves training each model with a subset of hyperparameters that are held constant throughout all tests in order to create a similar testing space for all models. The constant parameters are the word vector dimension d, learning rate η, latent vector dimension h and the dropout keep probability $P(keep)$ during training. The values of these parameters are listed in Table 5.1.

We require that each model learns the word embedding during training, as there is a large difference in the quality of pre-trained word vectors available for each of the datasets we use. We follow [5] and set the word vector size parameter $d = 300$, since the authors of this paper notice a steep drop in performance when the size of word vectors is too small. We observe that all the models learn stably when $\eta = 1 \times 10^{-3}$ and learning only slows with smaller values of η without increases in accuracy. For the latent vector dimension for all models, we match the best choice

Hyperparameter	Value
d	300
η	1×10^{-3}
h	100
$P(keep)$	0.5

Table 5.1: Constants for the classification task.

for the dimension used in [5] and set $h = 100$ for all classification experiments. We train all models with dropout keep probability 0.5 and during inference, we do not use any dropout (dropout keep probability 1.0). Each model is presented with roughly 10 epochs of each dataset during training, as we find that beyond this, accuracy does not increase or slight over-fitting may occur.

The outputs of each model we test are the linear activations of the k neurons in the output layer. In order to construct an objective function for the classification task, we convert the outputs to a probability distribution over class labels by adding softmax activation to each of the output neurons. The softmax function $\sigma(a)$ for activation value a is defined in Equation 5.1. This function computes the probability of an element from neural network activation vector $\mathbf{a}$ being "on" through an exponentially weighted calculation with respect to all other elements in the vector. When applied to the linear activations of the k neurons in the output layer of each model, the softmax function scales the outputs to "class label" probabilities that are exponentially proportional to the linear output signal of each neuron. The resulting k softmax activations sum to the value 1, where the activation of neuron j in the output layer corresponds to the probability that the input sample belongs to class j. Ideally, the trained outputs of the classification model would assign a probability of 1 to the output neuron corresponding to the correct class with 0 probability assigned to all other output neurons—this is referred to as a *one-hot* vector. The labels used for supervision are encoded into one-hot vectors containing the value of 1 in the index corresponding to the value of the label. The values of the one-hot labels clearly sum to 1, therefore they can be considered a probability distribution of the same form as the model outputs.

$$\sigma(a_i) = \frac{\exp(a_i)}{\sum_j \exp(a_j)} \tag{5.1}$$

We compute the model loss l_i, of input sample i using the cross entropy function defined in Equation 5.2. In the equation, $\hat{y}_{ij}$ is the softmax activation of the j^{th} neuron in the output layer of the model for input sample i and y_{ij} is the value of the j^{th} index of the one-hot label for input sample i. The softmax cross entropy function is well suited for representing the model performance for the classification task because it computes an error measure between two probability distributions. When training with mini-batches, we feed multiple samples through the model and evaluate the batch loss L, by computing the average over the values of l_i for each sample i, in the mini-batch. Training the model is performed with gradient descent by iteratively updating the parameters of the model, θ, along the opposite direction of the gradient $\partial(L)/\partial(\theta)$.

$$l_i = -\sum_{j}^{k} y_{ij} log(\hat{y}_{ij}) \tag{5.2}$$

5.2 Semi-Supervised Learning Task

We test the four neural network models on the semi-supervised learning task performed in [5] for the Question-Type dataset. The task in [5] is designed such that a neural network model learns latent vector representations from a short text dataset with a limited amount of labeled information. The model learns to group latent representations by minimizing a K-means inspired objective function J, which maintains and updates cluster label assignments and centroids in terms of model weight parameters. This objective function is defined as,

$$J = \alpha \sum_{i}^{N} \sum_{j}^{k} r_{ij}\delta_{ij} + (1-\alpha) \sum_{i}^{L} \{\delta_{ig_i} + \sum_{l \neq g_i} \max(m + \delta_{ig_i} - \delta_{il}, 0)\} \tag{5.3}$$

where $\delta_{ij} = ||f(x_i) - \mu_j||^2$.

In Equation 5.3, r_{ij} is the value of the j^{th} one-hot index for the current cluster assignment for sample i, g_i is the ground truth class label for input sample i mapped to the correct current cluster assignment by solving a minimum-cost matching problem between cluster assignments and ground truth labels for the labeled data, $f(x_i) \in \mathcal{R}^h$ is the latent representation output by the model for input sample i and $\mu_j \in \mathcal{R}^h$ is the current centroid corresponding to cluster j.

The minimum-cost matching problem used to map ground truth labels to their corresponding cluster labels is addressed in [27]. For each pairing of a cluster and the set of all samples belonging to one ground truth label, we calculate a *cost* of that pairing that is minimized when the pair shares a lot of common samples. The best mapping of ground truth labels to cluster labels will therefore have minimum-cost.

A cost measure that achieves this relationship is the size of the symmetric difference between two sets, which is equal to the number of samples not common to both sets. We should note that this approach is complimentary to the measure used in the Cluster Validation chapter of [10] for solving this problem using a maximum matching. More formally, the size of the symmetric difference between sets A and B can be defined as,

$$W_{AB} = |A| + |B| - 2|A \cap B| \tag{5.4}$$

The cost for each pair of cluster assignments and set of samples with a common label is calculated according to Equation 5.4 to produce a $K \times K$ matrix of weights for a clustering problem with K classes and K cluster assignments. The ground truth labels correspond to the columns of the matrix and the cluster assignments correspond to the rows of the matrix. Matrix element W_{ij} is the amount of points not common to both cluster assignment i and ground truth label j. The mapping from ground truth labels to cluster assignment labels can be determined by solving the minimum-cost bipartite matching problem. This problem is solved using the `linear_sum_assignment()` method in the Scipy Optimization and Root Finding library [28]. The solution to this problem gives the mapped cluster assignment g_i for ground truth label i.

The authors in [5] designed the objective function defined by Equation 5.3 to consist of two cleverly designed terms weighted by the parameter $\alpha \in [0, 1]$. The first term computes a loss value according to the unsupervised K-means objective by iteratively updating N one-hot cluster assignments and the k centroids $\mu_j \in \mathcal{R}^h$ $\forall j = 1...k$. The second term utilizes supervision from a subset of labeled data in order to guide the clustering process performed when minimizing the first term. The loss contributed by the second term is a function of the distance between the latent representations of labeled input samples and their correct and incorrect current assigned centroids. The loss term corresponding to the correct centroid of a labeled sample i is the euclidean distance between the latent representation of the sample and its correct centroid. The loss term corresponding to the remaining $k-1$ incorrect centroids of a labeled sample i is computed from the sum of hinge losses with margin m. The hinge loss with margin m for input sample i and incorrect centroid j

Hyperparameter	Value
d	300
h	100
$P(keep)$	0.5

Table 5.2: Constants for the semi-supervised learning task.

contributes loss if the latent representation of sample i is not closer to its correct centroid by margin m than it is to incorrect centroid j. The two main terms of the objective are weighted with parameter α in order to allow control over how much the subset of labeled data contributes to the loss.

Like the supervised classification task, we keep a subset of the hyperparameters constant during the test of the neural network models for the semi-supervised learning task. The hyperparameters that are held constant can be seen in Table 5.2. One exception is made for the DCNN model in which the word vector dimension d must be decreased to 100 due to memory issues with very large multi-dimensional arrays. For each test, we evaluate the quality of the learned latent representations by performing cluster analysis on them. We describe the procedure for this analysis in the next section.

5.3 Clustering and Analysis

We show the effectiveness of the semi-supervised learning task by clustering in the latent representation space learned by each of the models on the Question-Type dataset. In order to observe change in performance rather than raw performance, we also perform clustering in the learned latent space of each model after it has only been pre-trained on the subset of labeled data. We additionally compare both of these results to clustering on the traditional BOW and TF-IDF text representations. We cluster using the K-means algorithm and we evaluate the clusters using external validation measures by utilizing the entirety of the ground truth labels for each dataset. The external validation measures we use include the widely used F-Measure and Adjusted Mutual Information (AMI)—as used in [5]. We describe the mechanisms of both validation measures below.

The F-measure is a widely used cluster validation metric that represents a measure of both the *precision* and *recall* of a cluster assignment. The precision of cluster j is given by the ratio of the maximum number of samples in cluster j belonging to the same ground truth label, to the total number of samples in cluster j. Precision is also referred to as *purity*, in the sense that it gives a measure of how *pure* a cluster is—if all samples in a cluster belong to the same ground truth label, then this is a very good cluster and it can be said to be 100% pure, or have a *purity* of 1 and *precision* of 1. The recall of cluster j, provided the ground truth labels have been mapped to their corresponding cluster labels, is given by the ratio of the number of samples in cluster j belonging to the truth label i mapped to that cluster j, to the total number of samples for ground truth label i. The harmonic mean between the precision of cluster i and recall of cluster i is used to calculate the F-measure for cluster i and the total F-measure is taken as the average over the F-measures of all clusters [10]. The F-measure for a clustering into K clusters is defined as,

$$F = \frac{1}{K}\sum_{i}^{K}\frac{2p_i r_i}{p_i + r_i} \tag{5.5}$$

where p_i is the precision of cluster i and r_i is the recall of cluster i.

The F-measure is defined on $[0, 1]$ and gives a maximum value of 1 when perfect clustering is achieved.

Adjusted Mutual Information (AMI), also widely used as an external cluster validation metric, is a measurement of the information shared between a set of cluster assignments $\mathcal{C}$ and a set of ground truth label assignments $\mathcal{T}$, adjusted for the natural increase in mutual information between random sets as the sizes of the sets increase [29]. We use the definition in [10] for the mutual information between these sets, adjusted slightly for our case where the number of clusters r is the same as the number of unique ground truth labels k, therefore $r = k = K$. The mutual information between $\mathcal{C}$ and $\mathcal{T}$ is defined as,

$$I(\mathcal{C}, \mathcal{T}) = \sum_{i}^{K}\sum_{j}^{K} p_{ij} \log \frac{p_{ij}}{p_{C_i} p_{T_j}} \tag{5.6}$$

where p_{ij} is the probability of selecting a sample from cluster assignment i and with ground truth label j, p_{C_i} is the probability of selecting a sample from cluster i and

p_{T_j} is the probability of selecting a sample with ground truth label j.

To construct a scoring in the range $[0, 1]$, the mutual information is adjusted by considering the entropy of the cluster and ground truth label assignments as well as a measure of the expected mutual information if the clusters were assigned completely at random [29]. The adjustment involves dividing the mutual information $I(\mathcal{C}, \mathcal{T})$ by the maximum between the entropy of $\mathcal{C}$ and the entropy of $\mathcal{T}$ and subtracting from the top and bottom terms of this ratio the expected value of the mutual information, $E[I(\mathcal{C}, \mathcal{T})]$ for random cluster assignment. The equation from [29] for the AMI between a clustering assignment and ground truth labels is defined as,

$$AMI(\mathcal{C}, \mathcal{T}) = \frac{I(\mathcal{C}, \mathcal{T}) - E[I(\mathcal{C}, \mathcal{T})]}{\max(H(\mathcal{C}), H(\mathcal{T})) - E[I(\mathcal{C}, \mathcal{T})]} \tag{5.7}$$

where $H(\mathcal{C})$ and $H(\mathcal{T})$ is the entropy of $\mathcal{C}$ and $\mathcal{T}$ respectively.

Like the F-measure, AMI is defined on $[0, 1]$ and it is clear to see that if clusters are performed at random, the top term in Equation 5.7 will be very close to 0—indicating that we indeed have a poor cluster assignment. For a perfect cluster assignment, AMI has a maximum value of 1.

6. EXPERIMENTAL COMPARISON

We program all of the experiments described in the above section using Python version 2.7 and we use Google's TensorFlow library for the development of the neural network models [30]. Presented below are the results of the initial supervised learning task for classification and the semi-supervised learning task to learn latent representations of sentences. Finally, we report the results of cluster analysis on the semi-supervised latent representations.

6.1 Datasets

The datasets we use for both the supervised classification task and the semi-supervised learning task include the Question-Type dataset constructed in [1], the AG-News dataset of news topics constructed in [9] and accessed from [31], and the StackOverflow dataset of query topics constructed in [8], originally sourced from Kaggle [32].

The Question-Type dataset from [1] consists of short queries coarsely labeled by the type of answer the query is requesting and finely labeled by sub-categories of the coarse labels. Our tests only consider the coarse labels, which include *Abbreviation*, *Entity*, *Description*, *Human*, *Location* and *Number*—resulting in $K = 6$ total categories. The AG-News dataset from [9] consists of short news titles and their descriptions, labeled by their topics, which include *World*, *Sports*, *Business* and *Sci/Tech*—resulting in $K = 4$ total categories. The StackOverflow dataset from [8] consists of short queries from StackOverflow, labeled into $K = 20$ different programming topic categories such as *matlab*, *bash*, etc. All three of the datasets contain mutually exclusive classes in which no sample belongs to more than one class.

Since our motivation is to require that the neural network models automatically extract useful features from the datasets during learning, we perform minimal preprocessing and we do not perform any manual text enrichment. The only preprocessing we perform is to convert the sequence of words in each data sample into

Dataset	Question-Type	StackOverflow	AG-News
N	5,952	20,000	127,600
N_{train}	5,452	16,000	120,000
N_{test}	500	4,000	7,600
n_{max}	39	36	20
n_{avg}	10	8	6
$\|V\|$	8,983	18,927	50,627

Table 6.1: Statistics for the datasets used in testing.

a sequence of numbers to be ingested by the neural network models. We use the Gensim library for Python [33] in order to construct a vocabulary for each dataset and we use the vocabulary indices to convert words into real-valued numbers. These indices are used as the look-up key for the word embedding matrix in each neural network model. Since the datasets are relatively small, we do not perform any stop-word removal, word stemming or vocabulary pruning by removing infrequent words. The statistics for each dataset can be seen in Table 6.1, which shows the total number of samples N, number of samples in the training and testing splits (N_{train} and N_{test}), maximum and average sample length (n_{max} and n_{avg}), and the size of the vocabulary $|V|$ (number of unique words).

6.2 Supervised Classification Results

In this section, we describe the results of the supervised classification task which is used as the initial comparison for the neural network models we are testing. In each test, the hyperparamaters in Table 5.1 and the model designs outlined in Chapter 4 are held constant while we are free to adjust other parameters such as the amount of training iterations and the batch size in order to train on roughly 10 epochs over each dataset. We evaluate the performance of each model by computing the accuracy that the trained model achieves on the testing splits of each dataset. The accuracy is computed as the percentage of correctly classified samples out of all samples in the testing split.

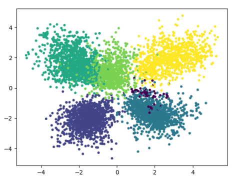
The average accuracy achieved over 5 training periods for each neural network model and for each dataset is shown in Table 6.2, with bold text indicating statistically better performance. We perform paired t-Test cross-validation with

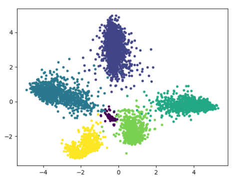
Model/Dataset	Question-Type	StackOverflow	AG-News
NBOW	86.36 ± 0.43	**85.27 ± 0.05**	84.62 ± 0.15
LSTM	**87.48 ± 0.52**	76.20 ± 0.50	84.26 ± 0.14
TCNN	**88.60 ± 0.66**	84.65 ± 0.17	**84.78 ± 0.29**
DCNN	86.04 ± 0.50	**85.38 ± 0.26**	**85.59 ± 0.41**

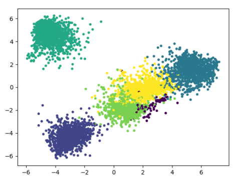
Table 6.2: Accuracy of the models on each of the short text datasets.

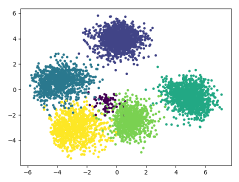
a 99% confidence interval on the accuracy results by taking the model with the highest average performance for each dataset and comparing it pair-wise against the performance of the others. The t-Test, outlined in Chapter 22 of [10], determines if the difference in performance between the winning classifier and a losing classifier in one dataset is statistically significant enough to consider the winning classifier superior over the losing classifier within some confidence interval. For the Question-Type dataset, the t-Test reveals with 99% confidence that the TCNN performs better than the NBOW and DCNN models, but no statistical difference can be determined between the performance of the TCNN over the LSTM model. For the StackOverflow dataset, the t-Test reveals with 99% confidence that the DCNN performs better than the LSTM and TCNN models, but no statistical difference can be determined between the performance of the DCNN over the NBOW model. For the AG-News dataset, the t-Test reveals with 99% confidence that the DCNN model performs better than the NBOW and LSTM models, but no statistical difference can be determined between the performance of the DCNN over the TCNN. It is important to note that the difference in model performances is not very large, with the exception of the LSTM model performance on the StackOverflow dataset. Comparisons between these results and results in the literature that use pre-trained word vectors suggest that the CNN based models gain performance over the other models by more efficiently utilizing word vectors. This would further support some hypotheses that CNN architectures may be better suited for NLU tasks on short text and we would like to look more closely into this behavior in future studies.

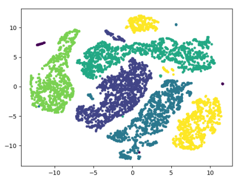
We attempt to learn more about the underlying behavior of each neural network model by projecting its learned latent space in $\mathcal{R}^h$ to $\mathcal{R}^2$ using Principle Component Analysis (PCA) and t-Distributed Stochastic Nearest Neighbor Embeddings

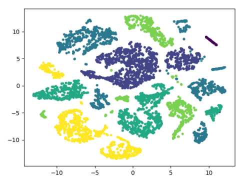

(a) NBOW

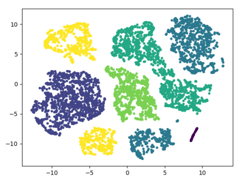

(b) LSTM


(c) TCNN


(d) DCNN


(e) NBOW


(f) LSTM


(g) TCNN

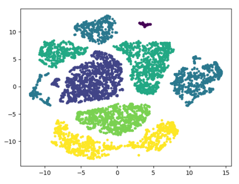

(h) DCNN

Figure 6.1: PCA (top row) and t-SNE (bottom row) projections of learned latent representations of the Question-Type dataset for the supervised task.

(t-SNE) [34]. Figure 6.1 shows the learned latent representations of the training split in the Question-Type dataset after training the models on the supervised task. The latent representations are projected using PCA and t-SNE for 2D visualization. We can see that the Question-Type dataset actually has quite an imbalance between the number of samples from each class. The amount of samples colored as dark purple in the latent visualizations are greatly outnumbered by the samples from other classes. We expect that separating these less-frequently seen samples from more-frequently seen samples will pose as a difficult challenge in the semi-supervised learning task and with clustering.

We perform additional analysis on each model by collecting the average Euclidean distance between the learned latent representations corresponding to the same label and the learned latent representations corresponding to different labels—which we call the average intra-label distance and average inter-label distance, respectively. These statistics, shown in Table 6.3, are collected by sampling the training split latent representations corresponding to the greatest accuracy achieved for each model after completing the supervised learning task. The average intra-label distance and the average inter-label distance give us an idea of how well each model can group similar inputs and separate dissimilar inputs in its latent representation

Model/Dataset	Question-Type	StackOverflow	AG-News
NBOW	1.79/4.89/3.10	2.56/6.60/4.04	3.73/10.66/6.93
LSTM	1.36/6.37/5.01	2.45/6.11/3.66	2.22/4.12/1.90
TCNN	2.31/10.33/8.02	3.10/11.74/8.64	4.73/14.44/9.71
DCNN	2.11/8.28/6.17	2.90/9.95/7.05	3.78/13.84/10.06

Table 6.3: Average intra-label distance/inter-label distance/difference margin of learned latent representations for each dataset and model.

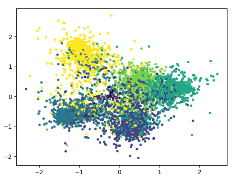
space. Given that these tests are fully supervised, we can use these statistics as general guidelines for what to expect when we test the neural network models on the more error-prone, semi-supervised task. We expect the difference between the inter-label and intra-label distance (Difference Margin value in Table 6.3) to largely dictate the range of values we can set for the margin parameter of Equation 5.3 for the semi-supervised task.

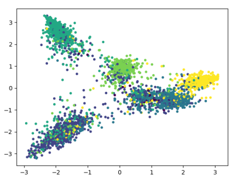
Overall, we see that at least one of the CNN based models tend to be a top performer for each dataset, indicating that fine grained feature extraction from the word embeddings may be an advantage in some circumstances. For the Question-Type dataset, we see that the LSTM model outperforms the DCNN and statistically performs at the same level as the TCNN. This result could be due to the fact that the Question-Type dataset requires the learning of slightly higher level features compared to the other two datasets. These higher level features may be less dependent on individual words and may in fact be extracted more efficiently by considering the sentence as a whole. This capability is noted as one of the strengths of RNN architectures in [24], indicating that our LSTM model may be taking advantage of these higher level features needed for better performance in the Question-Type dataset. The results on the Question-Type dataset also suggest that the TCNN may be better at extracting higher level features than the DCNN. This may seem like a surprising result since the TCNN we test is constructed with single word filters, indicating that it should learn to respond strongly to individual parts of a sentence—but since the TCNN also contains max-over-time pooling, convolutions over full sentences are eventually compressed to one value in the network. This may help the TCNN better extract the whole-sentence level features necessary to achieve slightly better performance on the Question-Type dataset.

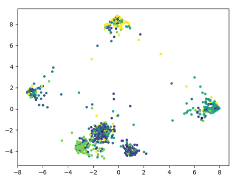
Since the StackOverflow dataset and the AG-News dataset are labeled based on topic information, individual word level features are much more useful for inferring the label of a sample sentence. We see that the DCNN takes advantage of the importance of features at the word level and is a top performer for both the StackOverflow and AG-News datasets. An interesting observation is that the NBOW model outperforms the TCNN model in the StackOverflow dataset but for the AG-News dataset these roles are reversed. We can try to understand these observations by considering some of the characteristics of each of these datasets. The StackOverflow dataset contains 20 different categories, each of which is a very particular programming relevant topic. Most of the time, this topic is entirely dependent on the occurrence of one word (e.g. *java*). Since the NBOW model outperforms the TCNN model on the StackOverflow dataset, the averaging operation that occurs in the NBOW model may preserve more individual word information than the max-over-time pooling operation that occurs in the TCNN model. On the other hand, the AG-News dataset contains only four categories with many different words contributing to a single category. Since the TCNN model outperforms the NBOW model on the AG-News dataset, the max-over-time pooling operation that occurs in the TCNN model may be able to better generalize similar sentences containing different words compared to that of the averaging operation that occurs in the NBOW model.

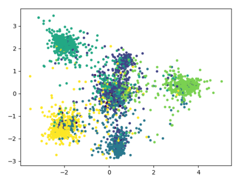
6.3 Semi-Supervised Task Results

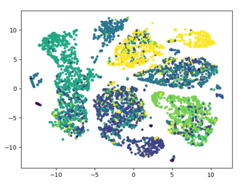
We perform the semi-supervised learning task on the Question-Type dataset by using each neural network model to minimize the objective function in Equation 5.3 with the hyperparameters in Table 5.2 held constant and $d = 100$ for the DCNN model. We follow the choice in [5] of setting $\alpha = 0.01$ in Equation 5.3. From the combined training and testing splits of the Question-Type dataset, we select 10% of the samples and use their labels for the semi-supervision. We follow the procedure in [5] and pre-train each model using the subset of labeled data before further training it on the semi-supervised learning task. Pre-training is done excessively over roughly 100 epochs of the subset of labeled data in order to promote any further increases

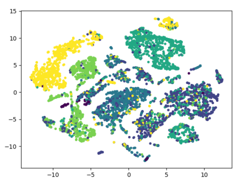

(a) NBOW

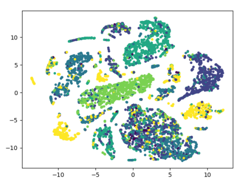

(b) LSTM


(c) TCNN


(d) DCNN


(e) NBOW


(f) LSTM


(g) TCNN

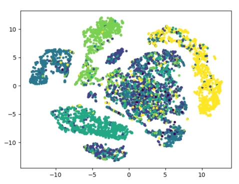

(h) DCNN

Figure 6.2: PCA (top row) and t-SNE (bottom row) projections of learned latent representations of the Question-Type dataset for the semi-supervised task.

in performance to be a result from minimizing Equation 5.3. Finally, we initialize each cluster centroid as the pre-trained latent representation of a randomly chosen labeled sample from each class and the minimization of Equation 5.3 is performed on roughly 40 epochs over the Question-Type dataset for all models.

We find that our intuition of the inter-neighbor and intra-neighbor separation achieved by each model under full supervision plays a key role in determining good values for the margin parameter in the semi-supervised learning task. Since the margin determines how much separation the semi-supervised learning task attempts to achieve between clusters in the latent space, we would not expect a test on the same models to result in greater separation in the semi-supervised task. In fact, we see the effects of these limitations when training on the semi-supervised task; when we set the margin to a value moderately greater than a model's corresponding difference margin value in Table 6.3, the frequency of non-converging training runs is increased. This is understandable since, at this point, we are setting objectives outside the physical range achievable by each model.

Figure 6.2 shows the learned latent representations of the Question-Type dataset projected using PCA and t-SNE for each of our neural network models for the semi-supervised task. We immediately see that the results are much more noisy compared to those in Figure 6.1 for the fully supervised task, but we can generally still observe 6 clusters that have been formed corresponding to the 6 classes in the Question-Type dataset. Another observation we notice is that samples from the least frequent class (dark purple) tend to be grouped into a large and noisy cluster with many other samples from other classes. This cluster seems to be formed from samples that cannot be clustered as easily and they group with the less frequently seen samples because this incurs the least amount of loss in Equation 5.3. We observed during training the TCNN that it was able to achieve lower values of the loss in Equation 5.3 compared to the other models. We also observe that the TCNN creates "tighter" clusters in its PCA projections which would agree with the lower loss values observed because the distance between samples in the same cluster become smaller while the distance between samples in different clusters becomes larger.

6.4 Cluster Evaluation

In order to observe the effectiveness of each model tasked with minimizing the objective defined in Equation 5.3 on the Question-Type dataset, we compare the learned latent representations after just pre-training each model and the learned latent representations after optimization on the full semi-supervised task. The quality of the learned latent representations is measured by performing K-means clustering on them, and we evaluate these clusters using the dataset labels and external cluster metrics. We use the Scikit Learn implementations of the K-means algorithm to perform the clustering [35].

The AMI and F-Measure from clustering on the learned latent representations after only pre-training and after the full semi-supervised task can be seen in Table 6.4. We immediately notice that the results vary greatly depending on which model we utilize. The t-Test from [10] is used again with a 99% confidence interval to compare the cluster metrics from only pre-training with the cluster metrics after completing the full optimization task. This comparison allows us to observe which model's statistically benefit more from minimizing the objective in Equation 5.3. We bold any AMI or F-Measure metrics for the full semi-supervised task that show an increase in performance that is statistically sufficient to consider.

Model	AMI Pre-Train Only	F-Measure Pre-Train Only	AMI Full-Task	F-Measure Full-Task
NBOW	0.425 ± 0.005	0.593 ± 0.014	0.441 ± 0.009	$\mathbf{0.706 \pm 0.025}$
LSTM	0.460 ± 0.007	0.641 ± 0.007	0.492 ± 0.017	$\mathbf{0.699 \pm 0.020}$
TCNN	0.453 ± 0.010	0.632 ± 0.008	0.432 ± 0.010	0.621 ± 0.017
DCNN	0.420 ± 0.012	0.607 ± 0.009	0.433 ± 0.014	0.566 ± 0.025

Table 6.4: AMI/F-Measure on only pre-training and on the full semi-supervised clustering task for each model on Question-Type dataset.

From Table 6.4, we see that only the NBOW and LSTM models show sufficient increases in performance from optimization on the full semi-supervised task and this performance is mainly seen in the F-Measure. This result is quite interesting as it does not agree much with any expectations that would stem from the results from the fully supervised learning tasks. One possible explanation for these observations is that the smaller difference margins in both the NBOW and LSTM models may allow samples that are initially assigned to wrong the cluster to be more easily corrected. In contrast, the larger separation that the CNN based models can achieve in their latent spaces may prevent many of these initial errors from being corrected. This could result in the slight decreases in performance we see for the TCNN and DCNN after optimizing the semi-supervised objective. Overall, these results raise some interesting questions about what kind of models work best for semi-supervised clustering applications—suggesting that simpler models that do not initially "commit" their encodings too strongly may be able to better correct their mistakes.

As a baseline measure for perspective, we additionally encode the Question-Type dataset using the BOW and TF-IDF weighting schemes and cluster on these representations as well. We only use the Question-Type dataset for this perspective because it has the smallest vocabulary and both of these representations are in $\mathcal{R}^{|V|}$ where $|V|$ is the vocabulary size. The AMI and F-Measure achieved by clustering the Question-Type BOW and TF-IDF representations can be seen in Table 6.5. We can see from the table that the sparsity of these representations causes them to be poorly clustered and the TF-IDF weighting scheme is able to offer a very slight increase in performance. Although not a fair comparison to clustering on the

Representation	AMI	F-Measure
BOW	0.140	0.306
TF-IDF	**0.157**	**0.375**

Table 6.5: Cluster evaluation on the BOW and TF-IDF representations of the Question-Type dataset.

representations learned in the semi-supervised learning task, the results in Table 6.5 demonstrate the difficulty of representing short text in the traditional methods normally used to represent full documents.

7. CONCLUSION AND FUTURE WORK

7.1 Conclusion

We have presented four popularly used neural network models for NLU tasks and have compared their performance with short text datasets on a supervised classification task and a semi-supervised clustering task. Each model is utilized on roughly equal footing, in order to evaluate a fair comparison between them.

The results from the supervised classification task establish base-line performance measures that allow us to make quantitative and visual comparisons between the behavior of each model. These comparisons reveal insights into how efficiently each model utilizes its latent representation space and we see the direct results of this behavior by the performance of the same models in a semi-supervised learning task on one of the datasets.

The results from the semi-supervised learning task reveal that a model with stronger discriminative power may not necessarily perform better in a setting where a large amount of initial error could be present. These results give some insight into selecting the type of model used for a semi-supervised learning task—suggesting that it may be wiser in some cases to choose a model with weaker discriminative power.

Overall, this work presents an example in which none of the models that we test can provide all-around superior performance on our tasks. In fact, we find that the specific architecture of each model can offer some advantages to help better exploit the structure of certain problems, which results in a surprisingly varied array of performances.

We discuss below some of the questions that are raised from the results of this work, as well as future directions we would like to take in regards to clustering short text. As work progresses, we will apply these methods to cluster raw data from online forums.

7.2 Pre-Trained Word Vectors

As mentioned earlier, none of our experiments involve the use of pre-trained word vectors, as the quality of available word vectors varies greatly across different domains and we try to maintain a level of fairness throughout all comparative experiments. In most cases, the use of pre-trained word vectors has shown to increase performance in learning tasks and this increase in performance varies depending on the model being used.

An interesting topic for future study would involve quantifying the utilization of pre-trained word vectors for each of the models we have tested in this work. These results would give some insight into which architectures benefit most from an increase in the quality of word vectors.

7.3 IBM HEALS

Much of this research stems from the work being done within IBM's Health Empowerment by Analytics, Learning and Semantics (HEALS) project, which aims to produce a chat-bot framework that personally provides health-related informative guidance to users who may or may not be motivated to improve their health. Currently the project focuses on the constrained domain of pre-diabetes and one of the subsystems involves mining text data from online question forums to construct a knowledge base within this domain. The goal is to cluster this text data into separate groups defined by *intent* in order to anticipate the future structure of conversations the chat-bot will have with its users. Since all of the collected text data is unlabeled, the next step in the intent mining subsystem of the HEALS project will be to construct a subset of manually labeled text samples. This subset would then be used to pre-train our model and seed the centroids for clustering on this new data.

7.4 Alternative Models

Autoencoders are popularly used for dimensionality reduction, therefore they have been studied as tools for unsupervised and semi-supervised clustering as well. One area we would like to explore is the use of adversarial learning in autoencoders, in which the latent representation space of the model is trained to produce outputs drawn from a predetermined probability distribution. By this design, the autoencoder learns to encode input samples into a constrained space which many times can spatially preserve similarities from the inputs. One such constraint that can be used on the latent representation is to draw samples from a classification distribution. The work in [36] demonstrates this idea in which an autoencoder is trained not only on reconstruction, but on a limited subset of labeled samples used to supervise its latent representations. An adversarial component provides constraints on the latent representations of the model in order to encourage it to produce outputs from the classification distribution. The work shows that the semi-supervised model can achieve very high accuracy on image datasets, but extending this ability to text-based datasets is still a challenging problem. This is because an autoencoder that operates on text data would have to learn to reconstruct word vectors, which exist in a non-continuous and high-dimensional space.

The siamese neural network architecture, originally proposed in [37], involves presenting a pair of input samples to a pair of neural networks with tied weights. These neural networks both produce an output and the distance between these outputs is calculated for evaluation of the loss. If the pair of samples are labeled as a "similar" pair, the networks are updated to minimize this distance and if the pair of samples are labeled "dissimilar", the networks are updated to increase this distance. The siamese network therefore can learn highly complex invariant features from data that can simply be labeled by pairwise similarity. This provides many benefits for learning features which are hard to physically define, but easy to label differences from. Examples of this phenomenon occur many times in text, where two sentences can be very similar but not share a single word. We would like to exploit this feature of siamese networks in future work in order to possibly discover more underlying features hidden in the structure of natural language.

REFERENCES

[1] X. Li and D. Roth, "Learning question classifiers," in *Proc. of the 19th Int. Conf. on Computational Linguistics*, vol. 1, 2002, pp. 1–7.

[2] D. Paranjpe, "Clustering semantically similar and related questions," https://nlp.stanford.edu/courses/cs224n/2007/fp/paranjpe.pdf, (Date Last Accessed, November, 10, 2017).

[3] G. Ramakrishnan, S. Chakrabarti, D. Paranjpe, and P. Bhattacharya, "Is question answering an acquired skill?" in *Proc. of the 13th Int. Conf. on World Wide Web*, 2004, pp. 111–120.

[4] Z. Yu, H. Wang, X. Lin, and M. Wang, "Understanding short texts through semantic enrichment and hashing," *IEEE Trans. on Knowledge and Data Eng.*, vol. 28, no. 2, pp. 566–579, Feb. 2016.

[5] Z. Wang, H. Mi, and A. Ittycheriah, "Semi-supervised clustering for short text via deep representation learning," in *Proc. of The 20th SIGNLL Conf. on Computational Natural Language Learning*, 2016, pp. 31–39.

[6] P. Blunsom, E. Grefenstette, and N. Kalchbrenner, "A convolutional neural network for modelling sentences," in *Proc. of the 52nd Annu. Meeting of the Assoc. for Computational Linguistics*, 2014, pp. 655–665.

[7] Y. Kim, "Convolutional neural networks for sentence classification," in *Proc. of the 2014 Conf. on Empirical Methods in Natural Language Process.*, 2014, pp. 1746–1751.

[8] J. Xu, P. Wang, G. Tian, B. Xu, J. Zhao, F. Wang, and H. Hao, "Short text clustering via convolutional neural networks," in *Proc. of NAACL-HLT 2015*, 2015, pp. 62–69.

[9] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in Neural Inform. Process. Syst.*, 2015, pp. 649–657.

[10] M. J. Zaki, W. Meira Jr, and W. Meira, *Data mining and analysis: fundamental concepts and algorithms.* Cambridge, UK: Cambridge University Press, 2014. [Online]. Available: http://www.dataminingbook.info/pmwiki.php

[11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[12] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[13] M. F. Porter, “An algorithm for suffix stripping,” *Program*, vol. 14, no. 3, pp. 130–137, July 1980.

[14] P. Bafna, D. Pramod, and A. Vaidya, “Document clustering: Tf-idf approach,” in *Int. Conf. on Elect., Electron., and Optimization Techn. (ICEEOT)*, 2016, pp. 61–66.

[15] C. D. Manning and H. Schtze, “15 topics in information retrieval,” in *Foundations of Statistical Natural Language Processing.* Cambridge, MA: The MIT Press, 1999, ch. 15, pp. 529–574.

[16] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *CoRR*, vol. abs/1301.3781, Sep. 2013, (Date Last Accessed, November, 18, 2017). [Online]. Available: https://arxiv.org/abs/1301.3781

[17] “Cognitive computation group,” http://cogcomp.org/page/resource_view/49, (Date Last Accessed, November, 8, 2017).

[18] E. M. Voorhees and D. Tice, “Overview of the trec-9 question answering track.” in *Proc. of the 9th Text REtrieval Conf. (TREC-9)*, 2001, pp. 71–80.

[19] P. Blunsom, K. Kocik, and J. R. Curran, “Question classification with log-linear models,” in *Proc. of the 29th Annu. Int. ACM SIGIR Conf. on Res. and Develop. in Inform. Retrieval*, 2006, pp. 615–616.

[20] Z. Huang, M. Thint, and Z. Qin, “Question classification using head words and their hypernyms,” in *Proc. of the Conf. on Empirical Methods in Natural Language Processing*, 2008, pp. 927–936.

[21] D. Zhang and W. S. Lee, “Question classification using support vector machines,” in *Proc. of the 26th Annu. Int. ACM SIGIR Conf. on Res. and Develop. in Inform. Retrieval*, 2003, pp. 26–32.

[22] Y. Chen and M. J. Zaki, “Kate: K-competitive autoencoder for text,” in *Proc. of the 23rd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2017, pp. 85–94.

[23] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, “Natural language processing (almost) from scratch,” *J. of Machine Learning Res.*, vol. 12, pp. 2493–2537, Aug. 2011.

[24] W. Yin, K. Kann, M. Yu, and H. Schütze, “Comparative study of cnn and rnn for natural language processing,” *CoRR*, vol. abs/1702.01923, 2017, (Date Last Accessed, November, 18, 2017). [Online]. Available: https://arxiv.org/abs/1702.01923

[25] X. He and P. Niyogi, “Locality preserving projections,” in *Advances in Neural Inf. Process. Syst.*, 2004, pp. 153–160.

[26] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting.” *J. of Machine Learning Res.*, vol. 15, pp. 1929–1958, Jun. 2014.

[27] H. W. Kuhn, “The hungarian method for the assignment problem,” *Naval Res. Logistics (NRL)*, vol. 2, no. 1-2, pp. 83–97, Mar. 1955.

[28] “Optimization and root finding (scipy.optimize) - scipy v1.0.0 reference guide,” https://docs.scipy.org/doc/scipy/reference/optimize.html, (Date Last Accessed, November, 8, 2017).

[29] N. X. Vinh, J. Epps, and J. Bailey, “Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance,” *J. of Machine Learning Res.*, vol. 11, pp. 2837–2854, Oct. 2010.

[30] “Tensorflow,” https://www.tensorflow.org/, (Date Last Accessed, November, 8, 2017).

[31] “Ag’s corpus of news articles,” http://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html, (Date Last Accessed, November, 8, 2017).

[32] “Kaggle: Your home for data science,” https://www.kaggle.com/, (Date Last Accessed, November, 8, 2017).

[33] “gensim: Topic modelling for humans,” https://radimrehurek.com/gensim/, (Date Last Accessed, November, 8, 2017).

[34] L. Van der Maaten and G. Hinton, “Visualizing data using t-sne,” *J. of Machine Learning Res.*, vol. 9, pp. 2579–2605, Nov. 2008.

[35] “2.3 clustering - scikit learn 0.19.1 documentation,” http://scikit-learn.org/stable/modules/clustering.html, (Date Last Accessed, November, 8, 2017).

[36] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, “Adversarial autoencoders,” *CoRR*, vol. abs/1511.05644, May 2016, (Date Last Accessed, November, 18, 2017). [Online]. Available: https://arxiv.org/abs/1511.05644

[37] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, “Signature verification using a” siamese” time delay neural network,” in *Adv. in Neural Inform. Process. Syst.*, 1994, pp. 737–744.