ON THE USE OF VECTOR REPRESENTATION FOR IMPROVED ACCURACY AND CURRENCY OF TWITTER POS TAGGING

by

David Sampson Samuel

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
December 2016

To Mom and Dad

Table of Contents

List of Tables

List of Figures

Abstract

The scarcity of labelled text corpora has inspired alternative methods for harnessing data for training and development of Natural Language Processing systems geared toward tasks such as Part-of-Speech (POS) tagging, Chunking and Semantic Role Labelling. Of particular interest is the performance of POS taggers on corpora which are largely informal and unstructured such as Twitter posts. In modern business activity, the expansion of social media networks has led to increased 'lead generation' activity; POS taggers form a significant part of such activities. We have trained a neural network based POS tagger using commercially available, labelled Penn Treebank data together with Twitter word embeddings. Word embeddings (or vector representations) are generated from tweets and used for training of the POS tagger. We illustrate the value of harnessing tweets as an unlimited, freely available data source by demonstration of improved performance on tagging of twitter text.

List of Abbreviations and Symbols Used

$\mathbb{R}^n$	The space of all *ordered n-tuples* of real numbers
POS	Part-of-speech
LSA	Latent Semantic Analysis
SRL	Semantic Role Labelling
NER	Named Entity Recognition
JSON	JavaScript Object Notation
URI	Uniform Resource Identifier
HAL	Hyperspace Analogue to Language
PCA	Principal Components Analysis
PPMI	Positive Pointwise Mutual Information
WSJ	Wall Street Journal
PWA	Per-word-Accuracy
NN	Neural Network
θ	Model parameters to be adjusted during training
V	Vocabulary list
d	Dimension of word vector
p	Probability

Glossary of Terms

Corpus	A collection of written text
Part-of-speech	Word category based on its syntactic function
Semantic Role Label	Detection of verb/predicate arguments
Chunking	Detection of parts-of-speech and short phrases
JSON	JavaScript Object Notation
Lead	Person with an interest in a product or service
Lead generation	Initiation of lead
Word embedding	Vector of real numbers representing a word
Twitter	Social media / microblogging platform
Tweet	Twitter post/microblog
Emoticon	Facial expression created from keyboard symbols
Tokenize	Splitting of a sequence of strings into individual parts/tokens
Language model	A statistical distribution over a sequence of words

Acknowledgements

My first note of thanks goes to my supervisor, Prof. Stan J. Matwin, for his patience, understanding and support during my time as a graduate student at the Institute of Big Data Analytics. With respect to basis of this work, I owe a tremendous debt of gratitude to, Dr. Jacek Wolkowicz, whose advice and guidance during my internship at LeadSift Inc. was invaluable. In addition, my fellow classmates deserve my thanks for making these past two years vibrant and enjoyable. Last but not least, I wish to thank my father and mother for their unwavering dedication and sacrifice that has directly and permanently impacted my life and helped guide me toward every success that I have ever achieved.

Chapter 1

Introduction

Natural language processing systems of the past have taken an atomic approach to word representation in that, each word was given some unique identifier which conveyed little, if anything, about the relationships between the words under consideration. As a prime example, the sentence *'Cherries are a highly anticipated summer fruit while December brings us blueberries'* among a given corpus would yield no relationships such as that between cherries and blueberries (ie. that they are both fruits and thus are closer conceptually to each other than to the word *December*). The usage of vector space models whereby words are each associated with a vector (ie. a member of $\mathbb{R}^n$), enables us to capture such relationships and can also mitigate the problem of data sparsity that stems from older methods that assign unique identifiers to words in an effort to construct and train language models.

In considering a distributed representation of words, we may consider the words in a corpus as a sequence of discrete states, each of which may be transformed into vectors of real numbers (also called weights). One might first opt for the simplest 'present or not present' encoding of a word whereby each vector has length equal to vocabulary size and a given word has a 1 in the corresponding element position and zeros elsewhere. As obvious as such an encoding may be, it is equally weak if we consider modelling comparisons other than word equality. We may instead consider a distributed representation of words whereby each word vector has some desired number of dimensions (50 for our purposes) and there is some distribution of real valued weights across all vector elements. This representation therefore yields vectors in which by the distributed nature of the elements, they each contribute to the definition of other words (not only their corresponding word). The underlying goal of word vector construction is then to learn word vectors that capture useful syntactic and semantic relationships that might not be otherwise achieved by using simpler representations. It is also worth noting that word vectors may be used as *inputs to*

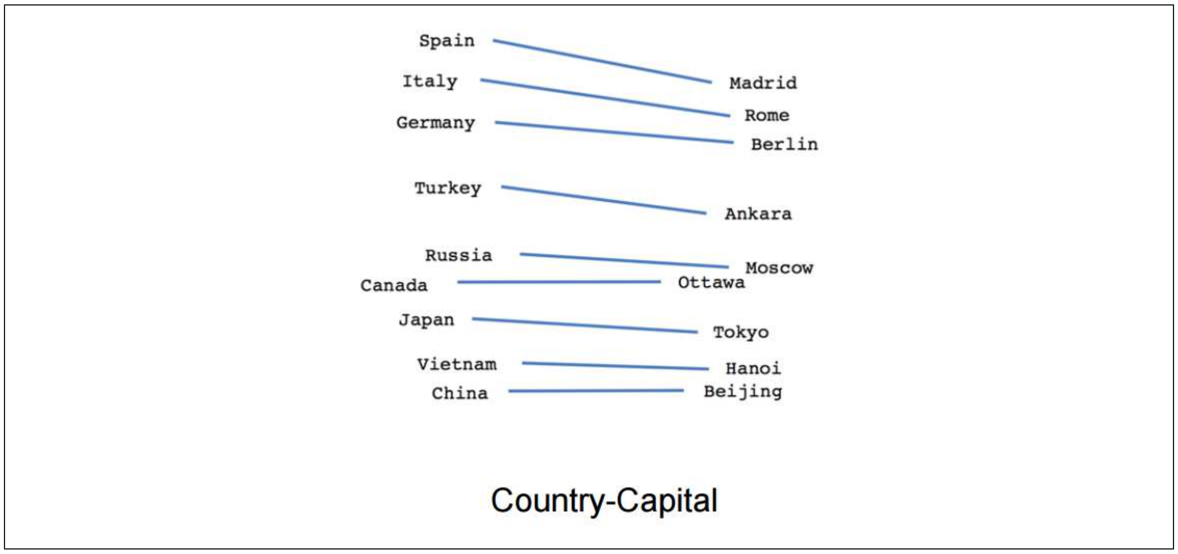


Figure 1.1: Countries and Capitals mapped as vectors in $\mathbb{R}^n$

neural network architectures for tasks such as Part of Speech Tagging and Semantic Role Labelling. This allows for the utilization of potentially unlimited amounts of unlabelled data for training of systems for various NLP tasks.

We refer to word vectors (or interchangeably, distributed representations) as *word embeddings* when we consider the fact that each vector can be located and visualized in multidimensional space. In particular, it is useful to think of the transformation of words to vectors as a mapping from a high dimensional space (where each word corresponds to one dimension) to a correspondingly more dense representation in lower dimensional space. The success of word embeddings learned in an unsupervised manner (such as those obtained via neural network architectures) in various NLP tasks has dwarfed older methods for achieving distributed representations, such as LSA (Latent Semantic Analysis).

The embedding of words in mathematical space (or more precisely, vector spaces) yields models that place words closer together that are more semantically similar and further away if they are not. We might imagine an example of such an embedding whereby countries are closer in distance (and therefore semantically) to their capitals than to other countries' capitals.

The linguistic principle known as the *distributional hypothesis* [56][57] (which theorizes that words used in the same contexts purport similar meanings), has fed various approaches to language modelling that are largely grouped under **predictive methods** (which predict words based on context, that is, based on surrounding words) and **count-based methods** (which count the number of times words co-occur with neighbouring words). We discuss word representation in greater detail later in this work.

In considering applications of word embeddings, many important NLP tasks including but not limited to, POS Tagging, Semantic Role Labelling (SRL), Named Entity Recognition (NER) and Sentiment Analysis, may be mentioned. In this work, we focus on Part of Speech (POS) tagging of Twitter text data. Twitter is a virtually unlimited source of text and has the added benefit of being as current and contemporary as we may choose during data collection. With over 300 million users active worldwide generating thousands of messages with each passing second, Twitter posts are characteristic of high velocity data. Twitter data has been used for various applications (e.g. monitoring earthquakes [35] and predicting flu outbreaks [36]). However, given that machine learning techniques constitute the frontier of advancements in NLP and such methods typically require large sets of labelled text corpora for training, the clear issue is the absence of such labelled datasets for Twitter microblogs. Such text constitutes the text field of tweets where this field typically comprises: short messages; inclusion of URIs; username mentions; topic markers; and threaded conversations. The nature of expressions in such messages is largely colloquial, meaning that there may be significant usage of abbreviated forms (eg. *ftw* (for the win), *omg* (Oh my gosh) etc) and forms of sentences/phrases that do not conform to established grammatical rules. In layman's terms, we may say that Twitter data frequently consists of *broken* speech. Such lack of adherence to formal grammatical rules, coupled with the lack of available labelled corpora and the 140 character limit results in great difficulty in achieving higher accuracy for part-of-speech taggers. In order to circumvent the lack of unlabelled Twitter text corpora but yet harness the potential for learning meaningful semantic relationships between words in contemporary speech, we use distributed word representations for training of our neural network based POS tagger.

1.1 Aims and Objectives

We note that POS Tagging of Twitter sentences is a challenging task in natural language processing. Traditional part-of-speech taggers trained on currently available labelled data do not exhibit good performance on labelled twitter data. Our claim is that harnessing a large set of unlabelled data might yield better tagger performance on Twitter sentences as compared to traditional POS Taggers trained only labelled data. Based on the above introduction to the background of word vector representation and POS Tagging, the overall research aim of this thesis can be stated as:

To train a POS Tagging system in a supervised fashion using Twitter word embeddings and labelled text data to obtain improved tagging performance on Twitter sentences.

In order to address the aim, the following objectives are identified:

1. Investigate and survey the related approaches proposed for word vector generation (or, distributed word representation) and POS Tagging.

2. Propose new approach to improved tagging performance via the use of a part-of-speech tagging system (**nlpnet**[4]) and word vectors generated from unlabelled Twitter text data. **Nlpnet** is a multilayer perceptron based classifier for POS Tagging inspired by an architecture [13] that achieves state-of-the-art results in English. It uses as input vectors built through Vector Space Models and can perform POS tagging, Semantic Role Labelling (SRL) and dependency parsing.

3. Demonstrate the effectiveness of word vector usage on POS tagging of Twitter tweets. We note that scalability is of concern with larger datasets.

1.2 Research Methodology

We have previously trained a POS Tagging system based on neural networks with the use of a labelled dataset from the Wall Street Journal [46]. In this work, we incorporate the use of 3 months worth of Twitter tweets constituting a large body of unlabelled text from everyday social network activity. In order to use this data to train our system, it must be cleaned and a suitable representation constructed. To that end, we employ a single layer neural network architecture[1] that learns a distributed representation for each word in our twitter dataset in an unsupervised manner. These representations can then be used directly as input to the chosen POS tagger [4] for training and subsequent testing against a labelled set of twitter text. This work is undertaken with the goal of demonstrating superior performance with the use of word embeddings constructed from a large, current, unlabelled dataset in comparison with our previous system trained solely on a relatively dated, labelled corpus of news media, as well as in comparison with the OpenNLP POS Tagging system.

1.3 Scientific Contribution

The empirical results of our training and testing of chosen systems are presented in this work. The main contributions of this thesis are as follows:

1. Twitter data collected in a streaming fashion over a period of 3 months is used as a source of unlabelled data at a large scale (ie. millions of tweets) for unsupervised learning of word embeddings using **word2vec**[1]. To the best of our knowledge, word2vec has never been coupled with Twitter data in order to obtain word embeddings.

2. We obtain improved POS Tagging performance on Twitter sentences using the previously unexplored technique of training **nlpnet** [4] initialized with pre-trained word embeddings at the input layer (learned from Twitter data using **word2vec**) and with labelled Wall Street Journal data as our 'supervising' examples.

1.4 Thesis Outline

Chapter 2 discusses related work in each of 3 relevant categories (Word Representation, POS Tagging and Twitter Tagging).
Chapter 3 describes the chosen data and preprocessing tasks as well as models used for obtaining word representations. Chapter 4 presents a detailed overview of the

nlpnet[4] tagger architecture. Chapter 5 is a description and discussion of our empirical results. Our conclusion and directions for future work are given in Chapter 6.

Chapter 2

Related Work

2.1 Word Representation

Among measures of interest in Natural Language processing, semantic similarity of documents/terms is fundamental. Intuitively, if some group of words or documents (say, elements for the sake of brevity) are given unique numerical representations (such as vectors of real numbers) and some distance metric is defined, we then associate smaller distances between elements with greater similarity of those elements with each other than with others at a greater distance away. Semantic similarity is used to represent 'is a' relations between elements. For example, among the words *apples, fruit* and *gardens*, there is no 'is a' relation between gardens and the other words, but an apple is indeed a fruit. This distinction is important in understanding that *semantic similarity* ought not to be interchanged with the more general *semantic relatedness* (which is not used in this work). Word vectors/embeddings are at the forefront of contemporary methods of word representations in NLP due to the rising need for modelling of semantic similarity among words. Such representations are useful in and of themselves as inputs for further tasks including but not limited to information retrieval, sentiment analysis, POS tagging, Semantic Role Labelling (SRL) and Chunking. The computation of distance-based similarity measures is also a basic application of such vectors.

Given the mathematical basis of approaches to word representation in NLP and in keeping with current trends (especially among the Deep Learning research community), we use word embedding we use *word embeddings/vectors* as our terms of choice to describe our distributed word representations in this work. Natural Language Processing deviates from the field of formal linguistics in some sense since it considers a word's context to be the sole source upon which word representations should be built. In practice, this is taken to be some threshold value of words before, after or both with respect to some target word.

We use a real-valued vector for each word of a given vocabulary and refer to these as **word embeddings**, which can in turn be used as features for training algorithms in a variety of tasks, including but not limited to parsing [16], named entity recognition [17] and document classification [18]. In the work of Bengio [19], a multi-clustering idea of distributed representations has been put forward and can be captured by models that produce *dimensions of meaning* rather than simply relying on distance or angle between word vectors as a method of evaluating quality of word representations. Mikolov et. al [1] has developed such a model that probes various dimensions of difference between word vectors as a means of evaluation. In general, two major approaches for learning word vectors are, in temporal order: 1) global methods and 2) local methods. The main difference (and source of sub-par performance for certain tasks) between these models is in the information maintained by each method. Global methods make use of statistical information at the document level (ie, such methods take a *global* view to vector generation) whereas local methods rely on some small window of context that largely neglects statistical information provided outside of the chosen window. We consider works on each method in finer detail but digress to say that the pitfalls of each method have driven the efforts of Pennington et. al [20] to develop a new log-bilinear regression model that achieves superior performance on similarity and named entity recognition tasks via a combination of advantages from both varieties of approaches.

Matrix Factorization Methods.
Global approaches produce large matrices as a result of entire documents being taken as word context. Matrix factorization methods were developed for generating low-dimensional word representations from such large matrices. Global approaches to vector generation can be seen in early works such as (Hyperspace Analogue to Language) HAL [22] and Latent Semantic Analysis [21]. A matrix produced by HAL has elements corresponding to the number of times a word occurs in the context of another word, in contrast to matrices produced via LSA which considers the number of times a word occurs in a given document. In consideration of word co-occurence, it is useful to recall that '**stop-words**' (ie. commonly used words) such as determiners

(like a, an), prepositions (eg. so, but, for and yet) and even some coordinating conjunctions may frequently occur in the context of more important/interesting words but do not contribute much information about co-occurrences between important words. We may therefore find for instance, that the word 'an' occurs more frequently with 'apple' than does 'apple' with 'orchard' even though we are likely to be more interested in examining potential relations between the latter two words. Clearly, methods that rely on word co-occurrence can potentially suffer from disproportionate contributions (to similarity measures) by stop-words thereby giving much weight to similarity based on co-occurrence and virtually no information about the semantic relationship between two given words. Several works [23][24][25] demonstrate a solution to this issue by showing that word co-occurrence counts may be transformed in such a way that they are more evenly spread across some smaller interval. In the most recent of these works, Lebret et. al [25] use the Hellinger PCA (that is, Principal Component Analysis performed using the Hellinger distance [58]) approach to achieving good word representations.

Shallow Window-Based Methods.

Instead of considering an entire document as a word's context, we may instead restrict the context to some smaller sized window of words around the target word to be predicted. The use of neural networks for language modeling has been pushed to large scale application by the work of [26] where instead of focusing on the role of words in a sentence (as in the work of Miikkulainen et. al [30]), a word sequence based statistical distribution is learned. Bengio et al. [26] in addition to coining the term *word embeddings*, train such representations in a neural language model jointly with the model's parameters. However, the utility of *pre-trained embeddings* was first demonstrated by Collobert et. al [27] and their neural network architecture has been the basis of many current approaches to NLP tasks. In an effort to avoid using a full neural network architecture (and therefore to reduce computational complexity) for learning word representations, the skip-gram and continuous bag-of-words (CBOW) models of Mikolov et al. [1] utilize the inner product of word vectors in a simple single-layer architecture. The PPMI (Positive Pointwise Mutual Information) metric has been employed [29] for obtaining word vectors and log-bilinear models [28] have been developed for this task. In the words of Jurafsky [59], the PMI measure 'is a measure of how often two pointwise mutual information events x and y occur, compared with what we would expect if they were independent'. PPMI is obtained by replacing all negative PMI values with zero. The word analogy task has been used as a prominent test case to demonstrate the capacity of these models to learn complex semantic relationships as linear relationships between the word vectors.

GloVe

In the work of Pennington et. al [20], a log-bilinear regression model is proposed as a way of bridging the gap between shallow window-based methods and global approaches. Namely, it is found to be useful to exploit document-wide information (such as largely repetitive occurrences) that has only been available with global approaches. Pennington et. al. present an analysis of model properties deemed essential for the production of linear directions of meaning. Specifically, a weighted least squares model trained on global word-word co-occurrence counts is proposed, which achieves an accuracy of 75% for word analogy tasks. Their log-bilinear model has also been tested on NER and word similarity tasks. Intuitively, GloVe passes over a given corpus once and constructs a matrix X, of co-occurrence counts whereby element x_{ij} of X gives the number of times the represented by row i occurs in the context of the word represented by column j. Their model is then constructed to use X in the course of obtaining distributed word representations that retain at least some information supplied by the co-occurrence values in X.

The gap exploited by our work is given by the fact that Pennington et. al's (**currently state-of-the-art**) Twitter embeddings and the embeddings of prior works have not been coupled with a neural network model for tagging of Twitter tweets.

2.2 POS Tagging

Part-of-Speech (POS) tagging has been widely instrumental both as a task in its own right as well as for use within systems for Named Entity Recognition, Sentiment Analysis, Question Answering, Information retrieval and others. Various approaches to this task have been proposed with varying degrees of success and complexity. Early work such as that of [8] has given an automatic tagging system based on developed 'rules' but methods from the machine learning community have come to be at the forefront of developments in NLP for POS tagging. A major challenge for users of machine learning techniques for this task is availability of training data. Sliding window based methods for part-of-speech tagging have enjoyed great success. Taggers based on this approach assign a single part-of-speech tag to a given word, by looking at a fixed sized "window" of words around the word (for example, 5 words before and after the target word). Motivation for this approach can be attributed to the observation that large proportions of words (estimated to be 30% or more, language dependent [9]) may be assigned more than one possible POS tag when considered out of context. The sliding window provides a fixed context for each word. Toutanova et al. [10] have presented a benchmark setup consisting of Wall Street Journal data. Previous works [13] suggest that systems where bidirectional decoding algorithms coupled with windows of text context give superior POS tagging performance. Some features may include suffix, prefix, capitalization and multiple words (n-grams). In

[10], 97.24% per word accuracy (PWA) is achieved with the use of maximum entropy classifiers together with a bidirectional dependency network [39]. An approach using SVM trained on text windows achieved 97.16% PWA by employing Viterbi decoding for bidirectional inference. In [12], 97.33% PWA was obtained using guided learning. Collobert et. al. [13] have presented a unified neural network architecture which learns internal representations using mostly unlabeled training data and surpasses the benchmark in [10] achieving a PWA of 97.29%.

2.3 Twitter POS Tagging

In considering the generation of POS tags for Twitter text data, two clustering based taggers [40][41] are among the earliest works. Gimpel et. all [40] achieves 92.8% PWA and Ritter et. al [41] achieves 88.4%. Both use clustering, existing (non-Twitter) labelled corpora and hand labelled tweets but only Ritter et. al. used Penn Treebank tags. In [42], Foster et. al. presents results for POS tagging but place greater emphasis on parsing. In [43], POS tagging where noise was a significant factor (noisy tokens) was done by performing extra pre-/post-processing to fine tune the tagger. Clark et. al in [53] present an approach for improving tagging performance with the use of unlabelled data and bootstrap techniques. Finally, [44] propose a novel approach using unlabelled data which involved different POS tagging systems (using different tagsets) coupled with bootstrap techniques. Derczynski et. al. identify errors in existing taggers and make their augmented tagging system [45] publicly available; this system achieves per-word-accuracy of 88.7%.

Chapter 3

Data and Word Representation

3.1 Corpus Treatment

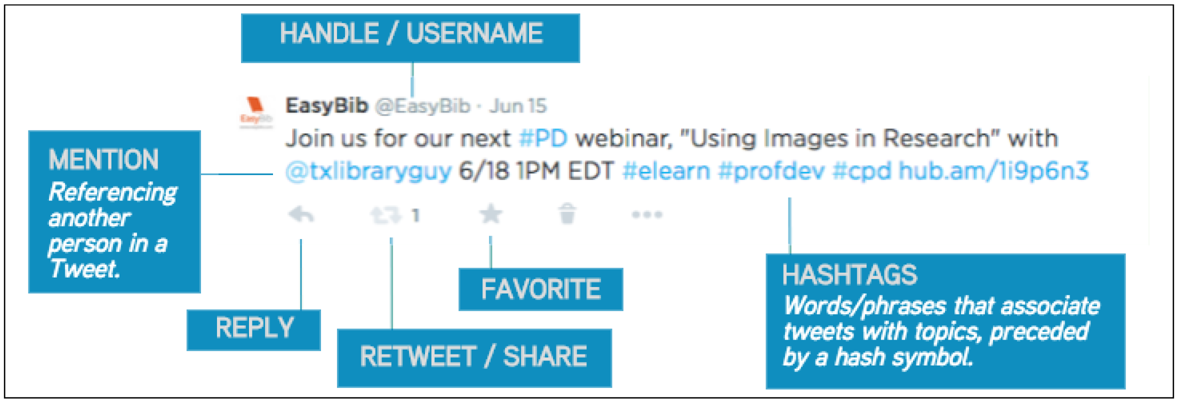
In our pursuit of improved POS Tagging performance on Twitter sentences, we have utilized two sets of data (labelled and unlabelled respectively). Our unlabelled data is fed into **word2vec**[1] for generation of word vectors and subsequently, the generated vectors are used as inputs to **nlpnet**[4] (our POS Tagger) which is trained in a supervised manner using our chosen labelled dataset. We briefly describe our labelled sentences before giving a detailed description of the unlabelled Twitter data that has been leveraged in this work.

Labelled Data

The Penn Treebank (PTB) project has provided 2,499 syntactically labelled (tagged) 98,732 Wall Street Journal (WSJ) stories collected over a period of three years. The corpus contains overall 1147617 tokens and 21717 vocabulary words. This dataset is commercially available and access has been provided for purposes of this work by LeadSift Inc., Halifax NS.

Unlabelled Data

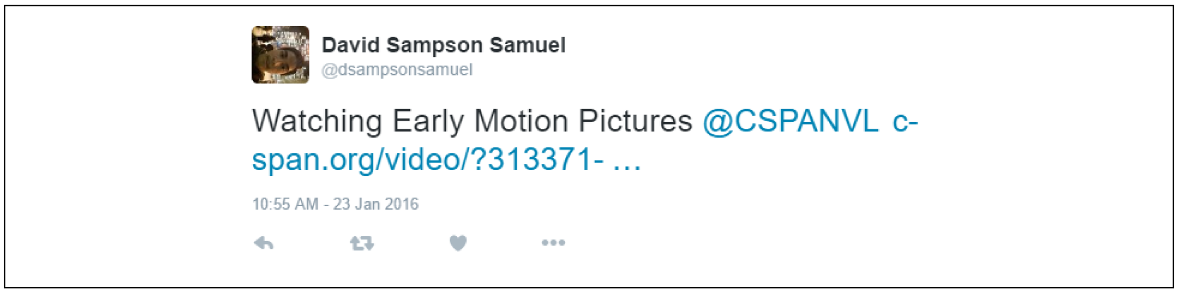
Currently, around 500 million tweets per day (about 200 billion annually)[47] are generated worldwide. We have collected Twitter text data (or, *tweets*) spanning the dates $01-$March-2016 through $31-$May-2016 amounting to approximately $400,000,000$ samples of tweets from users worldwide. This immensely valuable source of data opens a channel through which the thoughts and opinions of millions of persons are made immediately available. Our data is obtained and archived daily in a real-time streaming fashion via the Twitter API. Twitter provides RESTful APIs which allow access to core Twitter data, including user info, locations, update timelines and status data. The API presently supports various data formats such as RSS, XML and


EasyBib @EasyBib · Jun 15

Join us for our next #PD webinar, "Using Images in Research" with @txlibraryguy 6/18 1PM EDT #elearn #profdev #cpd hub.am/1i9p6n3

HANDLE / USERNAME
MENTION
Referencing another person in a Tweet.
REPLY
RETWEET / SHARE
FAVORITE
HASHTAGS
Words/phrases that associate tweets with topics, preceded by a hash symbol.
1

Figure 3.1: Sample tweet showing typical constituent elements of Twitter text


David Sampson Samuel
@dsampsonsamuel

Watching Early Motion Pictures @CSPANVL c-span.org/video/?313371- …

10:55 AM - 23 Jan 2016

Figure 3.2: Sample tweet example of URI usage

JSON. The streaming manner of collection gives our corpus the quality of being truly representative of Twitter subject matter, language and content. This directly reflects our desire for 'currency' (being as contemporary as possible) of sample data. Upon restriction to English language tweets, we obtain our desired sample corpus composed of $59,767,267$ tweets.

As shown in the above figures depicting representative tweet examples, our Twitter text frequently contains hashtags, URLs and username handles as well as twitter **at-mentions** whereby a username handle is referred to within a given tweet. Tweets are also frequently populated with 'emoticons' represented in unicode as well as emoticon approximations composed of ordinary ascii symbols (eg. :) for smiles or *_* as another facial expression). Various preprocessing steps were therefore taken to bring our raw data to usable form. We have

1. Replaced username handles with the token 'USER'
2. Replaced URLs with the token 'URL' (however, in further work, it may be of use to use extract TLD(Top Level Domains) from URLs to be used as entities such as Nouns as they frequently occupy such roles in social media text)
3. Removed all '#' symbols from hashtags and marked our data instead with the tag 'HT'.
4. Removed all unicode emoticons and emoticon 'approximations'
5. Split sentences into tokens using the NLTK tokenizer Python library

3.2 Word Representation

Overview

The typical ANN (Artificial Neural Network) generates embeddings from a vocabulary list of words as an intermediate step in its training. Specifically, each word in some dictionary are assigned to rows of a weight matrix whose columns correspond to the dimensions of each word vector. This matrix is randomly initialized with real values and each row is then used as the numerical representation of the corresponding vocabulary word. These embeddings (or vectors) are then iteratively improved via back-propagation. There are networks that are constructed and trained with the aim of word vector/embedding generation (call this **Type I**) and some merely produce such representations as a by-product (**Type II**). The 'breakthrough' impact of systems such as **word2vec**[1] and **GloVe**[20] lies merely in the difference between *it* as a Type I system having much lower computational complexity than its Type II counterparts. Large vocabularies and Deep Neural Networks have made for computationally intensive (sometimes prohibitively so) systems, and this has fuelled advancement in embedding technologies as the demand for systems with lower computational complexity rises.

With respect to the training objective, word2vec and GloVe differ from regular neural networks since they are designed with the aim of encoding general semantic relationships in their resulting embeddings. Models that utilize regular neural network typically produce embeddings (as an intermediate step) relevant to the specific task for which the network is being used.

Remark: *Semantically coherent word representations are relied on by tasks such as language modelling and can be produced as such in a similar way as compared to Type I models.*

Given that models (such as **word2vec** [4]) for learning of distributed word representations have evolved from earlier approaches known collectively as *language models*, we give an explanation of such models before addressing the structure of **word2vec**.

Notation

Since models are to be compared in this section, we assume the following notational standards: Let there be a training corpus containing a sequence of T training words $w_1, w_2, ..., w_T$ from a vocabulary V of cardinality $|V|$. Also let n be the number of context words with respect to some target word w. Each word is to be associated with an input embedding w_w having d dimensions and an output embedding v'_w. The objective function J_θ (where θ represents the parameters of the model) is to be optimized with respect to θ and for each input x, the score $f_\theta(x)$ will be produced as output.

Language Modelling

Definition: A language model is an algorithm for learning a function, that 'captures the important statistical characteristics of the distribution of sequences of words in a natural language, typically allowing one to make probabilistic predictions of the next word given preceding ones' [60].

The relation between language models and word embedding models is a close one. Language model quality is typically evaluated based on the ability to learn a probability distribution over words in V. Numerous state-of-the-art word embedding models try to predict the next word in a sequence to some extent. In addition, word embedding models are often assessed using perplexity. Perplexity is a cross-entropy based measure also used in language modelling.

Computation of the probability probability of a word w_t given its n previous words, i.e. $p(w_t|w_{t-1}, ..., w_{t-n+1})$ is the general objective of language models. The Markov Property in conjunction with the Chain Rule for Derivatives can be used to obtain the product of probabilities of each word given its n previous words which can then be used as an approximation of the probability of an entire sentence. Such a probability product can be expressed as:

$$p(w_1, ..., w_T) = \prod_i p(w_i|w_{i-1}, ..., w_{i-n+1}) \tag{3.1}$$

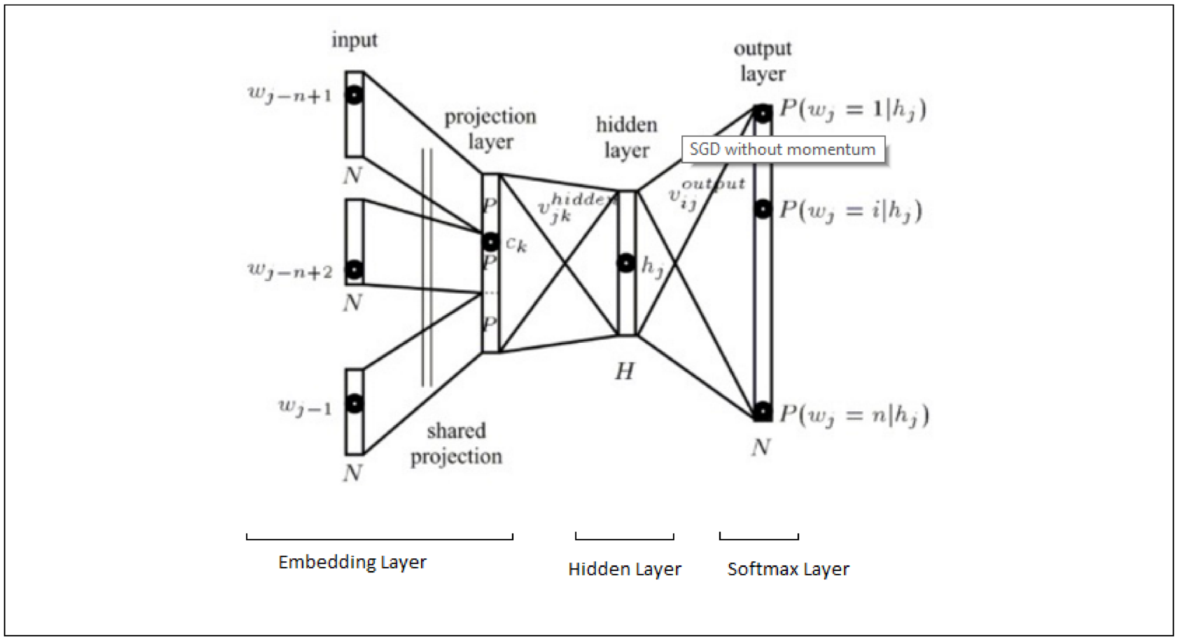


Figure 3.3: Neural Language model structure [26]

For the case of n-gram based language models, a word's probability can be computed from frequencies of its constituent n-grams:

$$p(w_t|w_{t-1}, ..., w_{t-n+1}) = \frac{count(w_{t-n+1}, ..., w_{t-1}, w_t)}{count(w_{t-n+1}, ..., w_{t-1})} \tag{3.2}$$

If $n = 2$ we obtain bigram probabilities, while $n = 5$ for instance, coupled with Kneser-Ney smoothing[31] yields smoothed 5-gram models. In neural networks, the probability of every word w must be calculated at the output layer. This probability $p(w_t|w_{t-1}, ..., w_{t-n+1})$ is calculated using the softmax function in the final layer.

With this softmax layer, the model maximizes the probability of predicting the correct word at each timestep t. The entire model then performs maximization of the averaged log probability of the whole corpus:

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \log p(w_1, ..., w_T) \tag{3.3}$$

In fact, the average of the log probabilities of all words in the corpus given their previous n words may be maximized:

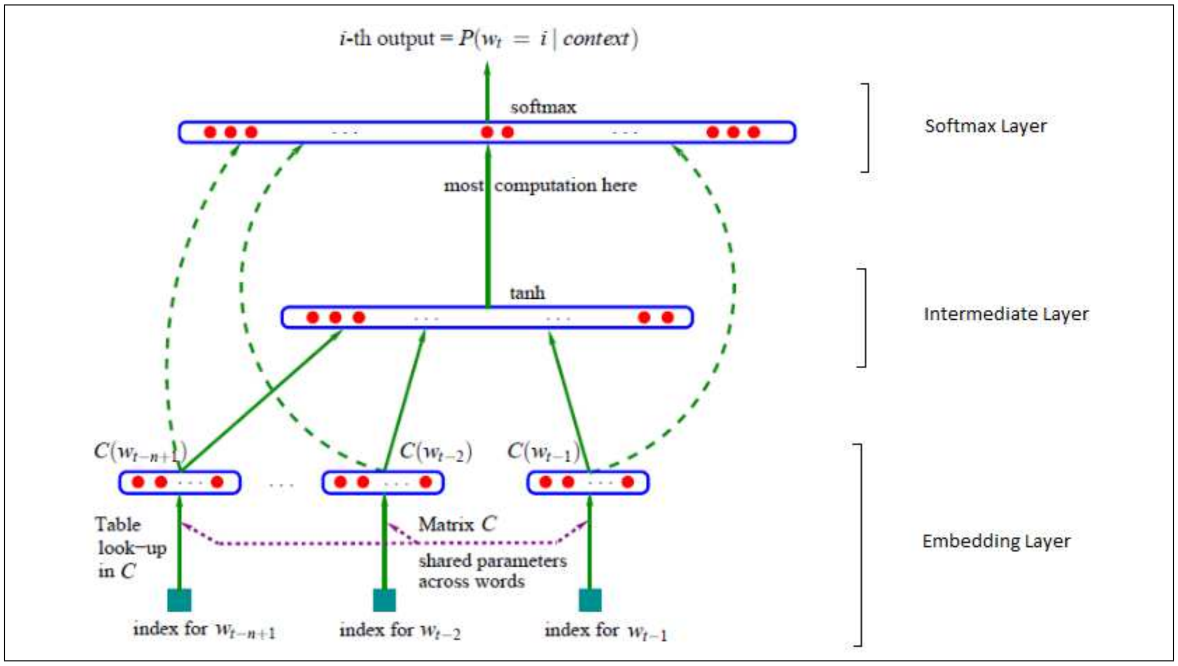


Figure 3.4: Bengio [26] language model

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \log p(w_t|w_{t-1}, ..., w_{t-n+1}) \tag{3.4}$$

Equation (3.3) is obtained from (3.4) by application of the chain rule since $p(w_1, ..., w_T)$ is given by $\Pi_i p(w_t|w_{t-1}, ..., w_{t-n+1})$. Word sampling at test time can be done by choosing the highest probability word $p(w_t|w_{t1}, ..., w_{tn+1})$ for each t.

Classic Neural Language Models

The proposed neural language model by Bengio et al. [26] consists of a one-hidden layer feed-forward neural network and predicts the next word in a sequence as in Figure 3.4.

This model maximizes the neural language model objective function:

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \log f(w_t|w_{t-1}, ..., w_{t-n+1}) \tag{3.5}$$

$f(w_t, w_{t1}, ..., w_{tn+1})$ is the output of the model, i.e. the probability $p(w_t|w_{t1}, ..., w_{tn+1})$ as computed by the softmax layer, and n is the number of previous words fed into the model.

The architecture presented in [26] forms the basis upon which current approaches have gradually improved. The general building blocks of their model, however, are still found in all current neural language and word embedding models. These are:

- **Embedding Layer**: a layer that generates word embeddings by multiplying an index vector with a word embedding matrix;
- **Intermediate Layer(s)**: one or more layers that produce an intermediate representation of the input, e.g. a fully-connected layer that applies a non-linearity to the concatenation of word embeddings of n previous words;
- **Softmax Layer**: the final layer that produces a probability distribution over words in V.

Bengio et. al have pointed out that the intermediate layer may be replaced by an LSTM (Long Short Term Memory) model, which is used by state-of-the-art neural language models [32][33]. In their work, the final softmax layer is identified as the network's main computational bottleneck, since there is a proportional relationship between vocabulary size $|V|$ and the softmax computation. After Bengio et al.'s pioneering work in neural language models, the limits of computing power slowed the progress of model construction for large vocabularies. The work of Collobert and Weston [34] avoids computing the expensive softmax by employing a different objective function. They use a pairwise ranking criterion as follows:

$$J_\theta = \sum_{x \in X} \sum_{w \in V} \max\{0, 1 - f_\theta(x) + f_\theta(x^{(w)})\} \tag{3.6}$$

Collobert and Weston sample correct windows x containing n words from the set of all possible windows X in their corpus. For each window x, they then produce a corrupted, incorrect version $x^{(w)}$ by replacing x's centre word with another word w from V. Their objective now maximises the distance between the scores output by the model for the correct and the incorrect window with a margin of 1. Collobert et. al.'s model architecture, depicted in Figure 3.5, is analogous to Bengio et al.'s[26] model.

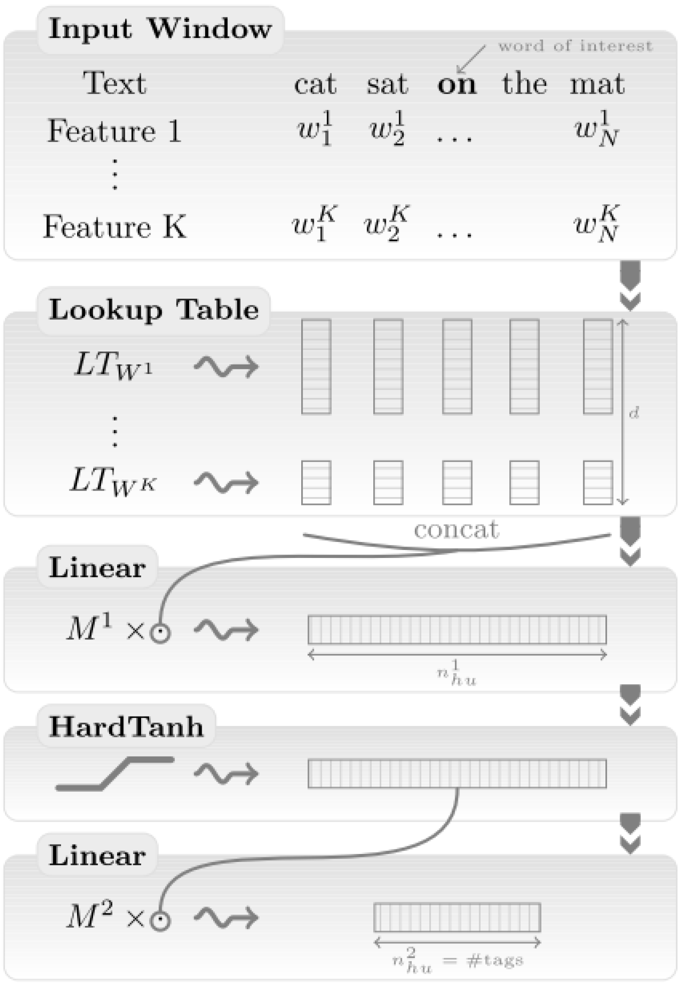


Figure 3.5: Collobert et. al [13] architecture

Their ranking objective offers a significant reduction in the complexity of the softmax, they keep the intermediate fully-connected hidden layer (2.) of Bengio et al. around (the HardTanh layer in Figure 3.5), which constitutes another source of expensive computation.

Word2Vec

Given the reduction in computational complexity offered by our chosen embedding model (the subject of two papers by Mikolov et al. in 2013 [1][2]), and its the demonstrably superior performance of its embeddings on word similarity and word analogy tasks as compared with embeddings from older models, we use **word2vec** for unsupervised learning of vectors from our unlabelled data. Since embeddings are a key building block of deep learning models for NLP, word2vec is often assumed to belong to the same group. Technically though, word2vec's architecture is neither deep nor uses non-linearities (in contrast to Bengio's model and Collobert's model).

Mikolov et al. in [1] propose two architectures for learning word embeddings that are less computationally expensive than previous models. In the work of [2], additional strategies for enhanced training speed and accuracy are proposed. The major benefits of these architectures by Mikolov et. al. are:

- Eradication of the hidden layer bottleneck.
- Allowing the model to consider additional context.

Their model's success can be particularly attributed to certain training strategies. We examine both architectures in detail:

Continuous bag-of-words (CBOW)

Language models look only at the past words for their predictions, as it is evaluated on its ability to predict each next word in the corpus, but a model constructed with the primary goal of generating accurate word embeddings does not have this restriction. Mikolov et al. thus use words both before and after the target word w_t to predict it as depicted in Figure 3.6. This is called the continuous bag-of-words (CBOW), as it uses continuous representations.

The objective function of CBOW in turn is only slightly different from the language model one:

$$J_\theta = \frac{1}{T} \sum_{t=1}^{T} \log p(w_t | w_{t-n}, ..., w_{t-1}, w_{t+1}, ..., w_{t+n}) \tag{3.7}$$

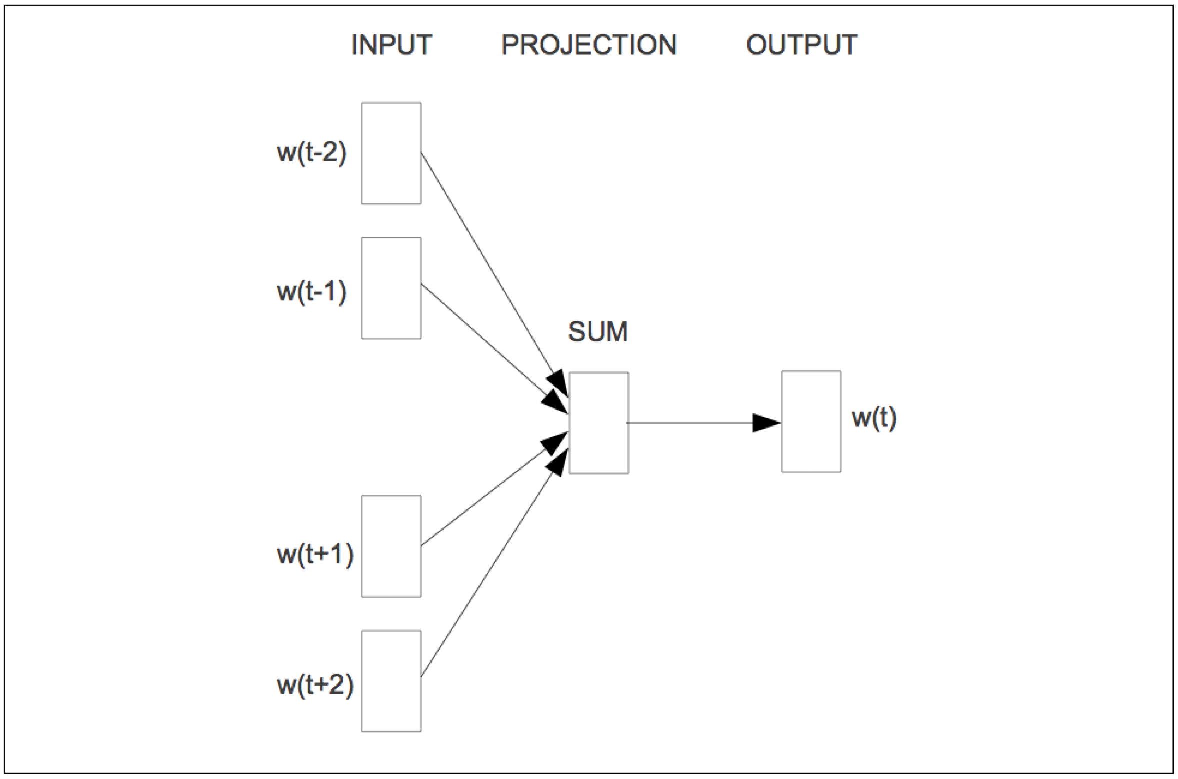


Figure 3.6: Continuous Bag-of-Words (CBOW) model [1]

Instead of feeding n previous words into the model, the model receives a window of n words around the target word w_t at each time step t.

Remark: Given that CBOW is faster than the alternative (Skip-Gram), we use only the former as our chosen word2vec model. However, we give a description and brief discussion of the latter.

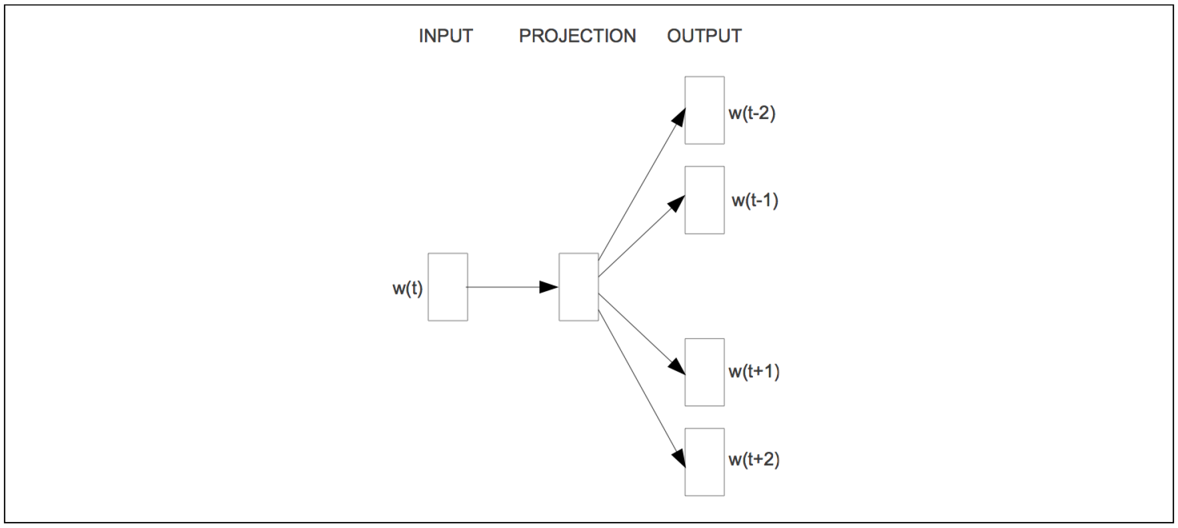


Figure 3.7: Skip-gram model [1]

Skip-gram

While CBOW can be seen as a precognitive language model, skip-gram turns the language model objective on its head: Instead of using the surrounding words to predict the centre word as with CBOW, skip-gram uses the centre word to predict the surrounding words as can be seen in Figure 3.7.

The skip-gram objective thus sums the log probabilities of the surrounding n words to the left and to the right of the target word w_t to produce the following objective:

$$J_\theta = \frac{1}{T} \sum_{t=1}^{T} \sum_{-n \leq j \leq n, \neq 0} \log p(w_{t+j} | w_T) \tag{3.8}$$

To gain a better intuition of how the skip-gram model computes $p(w_{t+j}|w_t)$, let's recall the definition of our softmax:

$$p(w_t | w_{t-1}, ..., w_{t-n+1}) = \frac{\exp(h^T v'_{w_t})}{\sum_{w_i \in V} \exp(h^T v'_{w_i})} \tag{3.9}$$

Instead of computing the probability of the target word w_t given its previous words, we calculate the probability of the surrounding word w_{t+j} given w_t. We can thus simply replace these variables in the equation:

$$p(w_{t+j} | w_t) = \frac{\exp(h^T v'_{w_{t+j}})}{\sum_{w_i \in V} \exp(h^T v'_{w_i})} \tag{3.10}$$

As the skip-gram architecture does not contain a hidden layer that produces an intermediate state vector h, h is simply the word embedding v_{w_t} of the input word w_t. This also makes it clearer why we want to have different representations for input embeddings v_w and output embeddings v_w, as we would otherwise multiply the word embedding by itself. Replacing h with v_{w_t} yields:

$$p(w_{t+j}|w_t) = \frac{\exp(v_{w_t}^T v'_{w_{t+j}})}{\sum_{w_i \in V} \exp(v_{w_t}^T v'_{w_i})} \tag{3.11}$$

Note that the notation in Mikolov's paper differs slightly from this thesis, as they denote the centre word with w_I and the surrounding words with w_O. If we replace w_t with w_I, w_{t+j} with w_O, and swap the vectors in the inner product due to its commutativity, we arrive at the softmax notation in their paper:

$$p(w_O|w_I) = \frac{\exp(v'^T_{w_O} v'_{w_I})}{\sum_{w=1}^{V} \exp(v'^T_{w} v_{w_I})} \tag{3.12}$$

Chapter 4

Nlpnet Tagger Architecture

The POS tagger used in this work is implemented by E. Fonseca [4] and similar to the work in [13]. This tagger (**nlpnet**) multiperceptron neural network based classifier used for POS Tagging, Chunking, Semantic Role Labelling and Parsing. In its operation, **nlpnet** takes a window (of user-defined size) of tokens which are each mapped to vectors of real values. The middle word of this token window is designated as our 'target word' to be tagged. These vectors are our 'feature vectors' which are subsequently concatenated and fed to further layers of the neural network. Each vector element requires one corresponding neuron in the input layer of **nlpnet** in order to be received. The weighted output of the input (or 'embedding') layer is fed to the non-linear hidden layer which produces output that is carried forward to the final (softmax) layer. The softmax layer consists of a number of neurons equal to the size of our part-of-speech tag set. The score at each output neuron determines the probability that the corresponding tag will be assigned to the target word.
The **nlpnet** tagger is available online. We use Fonseca's implemented model initialized with embeddings trained using unlabelled data and model described in Chapter 3. **Nlpnet** also utilizes WSJ data as its labelled data since training of this model is done in supervised manner. This chapter examines the internal tagger architecture.

4.1 Mathematical Notation

This architecture learns various layers of representations (higher levels of representations as we move through each layer) using only sentences as inputs. We use backpropagation to train the network parameters as the deep layers of the network compute task relevant features. In general terms each word is considered by the first layer for feature extraction. The second (hidden) layer considers word context windows (5 windows before and after each target word in our case)in order to extract

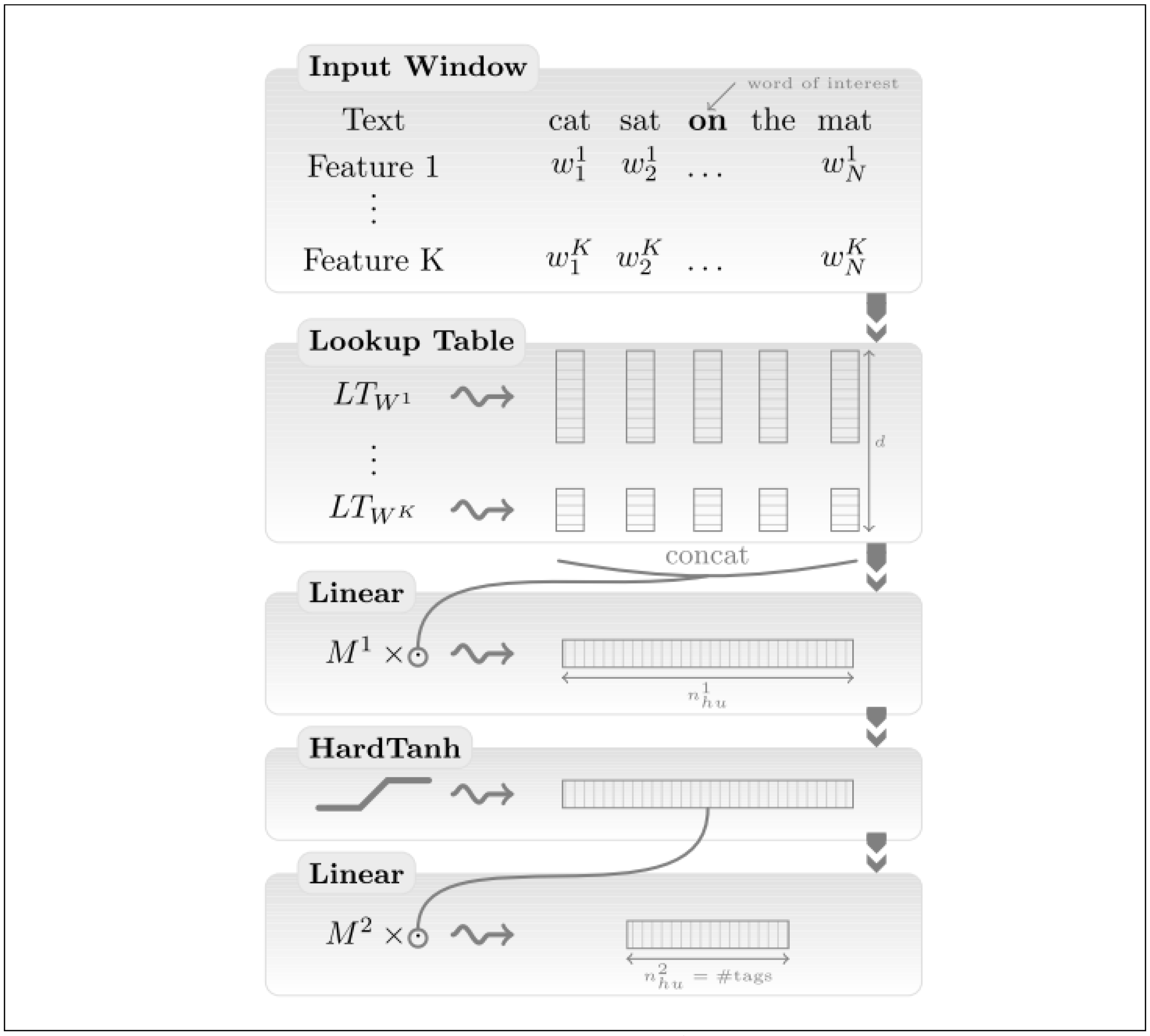


Figure 4.1: Basic Tagger Architecture

local and global structure information for each word. We present each NN layer individually in greater detail.

In accordance with the notation of Collobert et al [13], let a neural network be represented as a function $f_\theta(\cdot)$, with parameters θ. The composition of functions $f_\theta(\cdot) = f_\theta^L(f_\theta^{L-1}(...f_\theta^1(\cdot)))$ can then be interpreted as a representation of multiple network layers, where L is the total number of layers.

Each layer we use will be described in accordance with Figure 1 and Figure 2. With respect to a vector v, let $[v_i]$ denote the scalar at the i^{th} position in $[v]$ and $[x]_i$ be i^{th} element of a given sequence $[x]_i^T = \{x_1, x_2, ..., x_T\}$ where T is the number of terms. In dealing with matrices, the elements of a matrix A is indexed $[A]_{i,j}$ where

i and j represent row and column indexes respectively. The concatenation of d_{win} column vectors around the i^{th} column of our real-valued matrix A (ie. $A \in \mathbb{R}^{d_1 \times d_2}$) is represented as

$$[\langle A \rangle_i^{d_{win}}]^T = ([A]_{1,i-d_{win/2}}, ...[A]_{d_1,i-d_{win/2}}, ...[A]_{1,i+d_{win/2}}, ...[A]_{d_1,i+d_{win/2}}) \tag{4.1}$$

4.2 Word Transformation

A major advantage of this architecture is its low preprocessing requirement of words. That is, we may use words that are closer to raw text so that our method can learn good word representations. Let D be some dictionary of size $|D|$ where each word has an index. The purpose of the first network layer if to map each word index into a feature vector, via a lookup table operation. At first, our lookup table is a matrix initialized randomly with real numbers and then trained iteratively via *backpropagation*. For purposes of this thesis, we make use of unlabelled data obtained from *Twitter* microblogs (ie. text portions of *Tweets*). Instead of using randomly initialized vectors, we may populate our lookup table with our own pre-trained word vectors, as is the case with our usage of **word2vec** for construction of Twitter-based embeddings. Specifically, for each word $w \in D$, an internal d_{wrd}-dimensional feature vector representation is given by the lookup table layer $LT_W(\cdot)$:

$$LT_W(w) = \langle W \rangle_w^1 \tag{4.2}$$

where $W \in \mathbb{R}^{d_{wrd} \times |D|}$ represents a matrix of parameters which must be learned during training, $\langle W \rangle \in \mathbb{R}^{d_{wrd}}$ is the w^{th} column of W and d_{wrd} is the *user defined* word vector size. We consider sentences as sequences of T words $[w]_1^T$ in D from which a matrix is produced as below:

$$LT_W([w]_1^T) = (\langle W \rangle_{[w]_1}^1 \ \langle W \rangle_{[w]_2}^1 ... \langle W \rangle_{[w]_T}^1) \tag{4.3}$$

We use this matrix as input to further neural network layers.

4.3 From Word Vectors to Higher Level Features

Learning higher levels of representation (or in more intuitive terms, learning greater levels of abstraction) is achieved by combination of feature vectors (from the lookup table (3.16)) in subsequent neural network layers and finally, tag decisions for all words are to be produced. The production to tags for each word of a given sentence is a standard problem in machine-learning and natural language processing.

Two main approaches are presented in [13]; the first of these approaches being the word window approach and the second being a sentence level, convolutional approach. Given that our chosen task is POS Tagging, we focus only on the Word Window Approach in greater detail. For purposes of Semantic Role Labelling, the convolutional approach may be considered.

Word Window Approach

This method assumes that the context of a target word is the ideal source of information in predicting an appropriate tag for the target. For a given target word, a fixed size $k_s z$ window of words around (ie. before and after) this word is considered. We pass each word in the window, through the lookup table layer, and obtain a matrix of word features of fixed size $d_{wrd} \times k_{sz}$. Consider the resulting matrix as a $d_{wrd}k_{sz}$-dimensional vector obtained by concatenating each column vector, which can be fed to further neural network layers. Specifically, first network layer gives a word feature window as follows:

$$f_\theta^1 = \left\langle LT_W([w]_1^T) \right\rangle_t^{d_{win}} = (\langle W \rangle_{w_t - d_{win}/2}^1, ..., \langle W \rangle_t^1, ..., \langle W \rangle_{w_t + d_{win}/2}^1) \tag{4.4}$$

Linear Layer.
We may input the fixed size vector f_θ^l into one or more neural network layers which perform affine transformations over their inputs:

$$f_\theta^l = W^l f_\theta^{l-1} + b^l \tag{4.5}$$

where $W^l \in \mathbb{R}^{n_{hu}^l \times n_{hu}^{l-1}}$ and $b^l \in \mathbb{R}^{n_{hu}^l}$ are the network parameters to be trained by backpropagation. Note that the number of hidden units of the l^{th} layer is represented by n_{hu}^l. Given that we use **nlpnet** (the implementation provided by [4]), we instead use the sigmoid function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

where $\sigma(z)$ is a given hidden layer neuron.
Scoring. The last layer L of our network is of size 44 given that the number of POS tags defined according to our gold dataset (WSJ Penn Treebank data) has defined such tags. We may then consider the output each neuron in L as representing the score for each of our 44 part-of-speech tags. A given target word is therefore tagged according to the layer L neuron with the highest score.

4.4 Training

By 'training', we mean the use of stochastic gradient ascent for maximizing the likelihood over the training data. θ represents the trainable network parameters. The training set T is used for training and tuning of parameters in order to maximize the log-likelihood:

$$\theta \to \sum_{(x,y)\in T} \log p(y|x,\theta) \tag{4.6}$$

where x is a training word window or sentence and related features, and y is the corresponding tag. We use the neural network outputs to compute $p(\cdot)$.

Word-Level Log-Likelihood

Upon consideration of an input example x, the network with parameters θ produces a score for each word (of the input sentence) in turn. Specifically, the i^{th} tag is given a score $[f_\theta(x)]_i$ which is then taken to be the conditional tag probability $p(i|x,\theta)$ if the softmax [37] operation is performed over all the tags:

$$p(i|x,\theta) = \frac{e^{[f_\theta]_i}}{\sum_j e^{[f_\theta]_j}} \tag{4.7}$$

Collobert et. al.[13] defines the logadd operation as:

$$\text{logadd} z_i = \log(\sum_i e^{z_i}) \tag{4.8}$$

so as to obtain the log-likelihood for one training example (x, y) as:

$$\log p(y|x,\theta) = [f_\theta]_y - \text{logadd}[f_\theta]_j \tag{4.9}$$

Given the prevalence of correlations between neighbouring word tags in sentences, another approach for neural networks maybe used that takes such dependencies into consideration.

Maximizing Log-Likelihood

If we randomly choose some example (x, y) (where x and y are the training example and tags respectively), we can maximize (3.19) using stochastic gradient [38] as follows:

$$\theta \leftarrow \theta + \lambda \frac{\partial \log p(y|x, \theta)}{\partial \theta} \tag{4.10}$$

where λ is a user specified parameter called the learning rate. Recall that the composition of functions $f_\theta(\cdot) = f_\theta^L(f_\theta^{L-1}(...f_\theta^1(\cdot)))$ can then be interpreted as a representation of multiple network layers, where L is the total number of layers. Out neural network composed with the word-level log-likelihood (3.22). We can therefore obtain the derivative via the chain rule being applied through the network and the word-level log-likelihood (3.22).

Chapter 5

Evaluation and Discussion

In this chapter, we evaluate the effectiveness of our proposed word vector approach to the chosen task. Firstly, in section 4.1, we demonstrate the our choices and results with respect to vector generation. Section 4.2 focuses exclusively on part-of-speech tagging performance comparisons and we present tagged sample tweets deemed interesting with respect to our objectives.

5.1 Twitter-based Vectors

As specified in Section 3.1, we have obtained our word representations from approximately 60 million English language tweets using the Gensim Python implementation [48] of **word2vec**. Before delving into our usage of this model, it was mentioned earlier in Chapter 3 that computational 'bottleneck' for word embedding models has been the complexity of the calculations of the final softmax layer. Different methods of reducing the softmax complexity have been proposed and can be largely categorized as either sampling-based or softmax-based. It is worth noting that sampling-based methods make a difference to our bottleneck at training time but during evaluation of a normalised probability, the full softmax must be computed. Softmax-based approaches such as Hierarchical Softmax have been promoted by Morin et. al [52]. Softmax-based approaches keep the softmax layer intact, but achieve some increase in efficiency by modification of the softmax architecture. Sampling-based methods such as Negative Sampling [51] optimise some another loss function (ie. not softmax but something different) that approximates the softmax. Such optimization is achived though replacement of the normalization term in the softmax denominator with a less complex loss function.

A Word on Negative Sampling

Noise Contrastive Estimation(NCE)[50] yields an approximation of the softmax loss as sample size grows. The method of Negative Sampling (NeS), can be formulated as an approximation to NCE. Specifically, NeS dispenses with the NCE guarantee, since we would like to obtain high-quality word representations but have no concern for lowering perplexity on some test data.

NeS minimizes the negative log-likelihood of training set words via a logistic loss function. From [50], we have that the probability that a word w comes from the empirical training distribution P_{train} given a context c is calculated by NCE as follows:

$$P(y = 1|w, c) = \frac{\exp(h^T v_w^{'})}{\exp(h^T v_w^{'}) + kQ(w)} \tag{5.1}$$

NeS takes $kQ(w)$ (where Q represents the noise distribution) to be 1, thereby avoiding the biggest part of computational cost, which gives:

$$P(y = 1|w, c) = \frac{\exp(h^T v_w^{'})}{\exp(h^T v_w^{'}) + 1} \tag{5.2}$$

Note that if Q is uniform and k is 1, then $kQ(w) = 1$. NEG is shown to be equivalent to NCE when these assumptions are true. Consider (after setting $kQ(w)$) to 1, that $P(y = 1|w, c)$ can be transformed into the sigmoid function:

$$P(y = 1|w, c) = \frac{1}{1 + \exp(-h^T v_w^{'})} \tag{5.3}$$

Using this in our logistic regression loss gives:

$$J_\theta = -\sum_{w_i \in V} [\log \frac{1}{1 + \exp(-h^T v_{w_i}^{'})} + \sum_{j=1}^{k} \log(1 - \frac{1}{1 + \exp(-h^T v_{\hat{w_{ij}}}^{'})})] \tag{5.4}$$

Which can be converted to:

$$J_\theta = -\sum_{w_i \in V} [\log \frac{1}{1 + \exp(-h^T v_{w_i}^{'})} + \sum_{j=1}^{k} \log(\frac{1}{1 + \exp(h^T v_{\hat{w_{ij}}}^{'})})] \tag{5.5}$$

And when $\sigma(x) = 1 + \exp(x)$, the NeS loss is:

$$J_\theta = -\sum_{w_i \in V} [\log \sigma(h^T v_{w_i}^{'}) + \sum_{j=1}^{k} \log \sigma(-h^T v_{\hat{w_{ij}}}^{'})] \tag{5.6}$$

From [1], we can let $h = v_{w_I}$, $v_{w_i} = v_{w_O}$ and $v_{\hat{w_{ij}}} = v_{w_{ij}}$.

As discussed, if $k = |V|$ and Q is uniform, NEC and NeS are deemed equivalent. If these assumptions cannot be made, NeS is just an approximation of NCE, so it does not offer a direct optimization of correct word likelihood. This would be detrimental to language modelling tasks but is a plus for learning word embeddings.

We list the chosen parameters for running Gensim word2vec:

1. *Number of features* - the dimension of generated word vectors; 50 in our usage.
2. *Minimum word count* - ignores words that appear less than this number of times. We choose 10 as our cutoff.
3. *Context* - the maximum distance between the current and predicted word within a sentence. Our window is set to 5.
4. *Down sampling* - threshold for negative sampling taken as 1e-3
5. *Number of epochs* - In practice, for word2vec, this has been set anywhere from 3 to 50. We use Gensim's [48] default of 5 and note that further variation of this quantity may lead to even better word representations.
6. *CBOW or Skip-gram* - According to Mikolov et. al [1], CBOW is faster to train especially when coupled with negative sampling as an approximation of softmax. This choice is also appropriate given that our dataset is relatively large.

Given our parameter choices, we generate vectors for our Twitter vocabulary of size 348197. In Figure 4.1, we show the result of generating vectors on just 1 million tweets (for simplicity of representation) where this plot (and Figure 1 of the Appendix) was obtained via the use of tSNE (t-distributed Stochastic Neighbour Embedding) [49]. We show that social networking terms (tweet, followers, facebook etc.) are clustered together and so are a range of interjections (yeah, heyy, haha etc.). The ability to identify and utilize semantic similarities between such contemporary terminology is invaluable for various NLP tasks, including our chosen tagging objective.

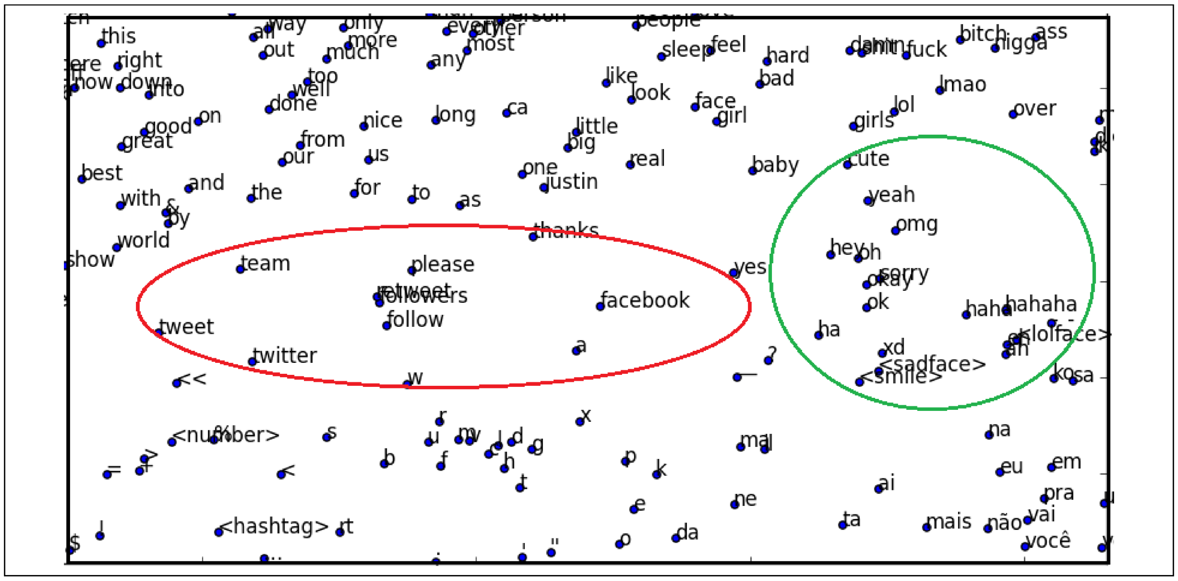


Figure 5.1: Word clusters identified in vector space representation

In tables 4.1 and 4.2, we observe the 10 most similar terms (and their similarity measures) for each of 4 words/expressions in order to demonstrate the contemporary value of harnessing Twitter embeddings. In the case of 'ikr' (interjection of agreement, '*i know right*'), we observe terms such as '*srsly*' or '*ofc*' (short for 'of course') being identified as semantically similar. In today's microblogging culture, the prevalence of such abbreviated or in some cases, incorrectly spelled forms poses the significant challenge for POS Tagging especially with respect to the limited supply of labelled Twitter data for training of taggers via supervised learning. As further examples of similar term identification, we refer to tables 4 and 5 of the Appendix.

Twitter Slang Similarities			
ikr	**Similarity**	*ftw*	**Similarity**
Ikr	0.8678	FTW	0.7139
yess	0.8266	rox	0.6979
IKR	0.8131	comon	0.6825
nooo	0.8114	deym	0.6790
yesss	0.7947	Kreygasm	0.6702
srsly	0.7890	stronk	0.6653
ofc	0.7878	hooo	0.6545
hahaha	0.7878	lookit	0.6479
ahaha	0.7795	arghhhh	0.6432
hahahah	0.7777	ohmy	0.6432

Table 5.1: The 10 closest contemporary slang expressions to the considered slang terms and their similarity measures

Celebrity Similarities			
bieber	**Similarity**	*trump*	**Similarity**
timberlake	0.9434	bernie	0.9072
beiber	0.9246	hillary	0.9063
mendes	0.8537	sanders	0.8776
Beiber	0.7989	cruz	0.8768
gomez	0.7955	clinton	0.8672
biebers	0.7924	romney	0.8635
sivan	0.7808	drumpf	0.8628
malik	0.7719	obama	0.8469
cabello	0.7716	rubio	0.8401
grande	0.7692	kasich	0.8371

Table 5.2: The 10 closest celebrity names to the considered celebrities and their similarity measures

5.2 Tagger Performance

Our chosen taggers were evaluated on 269 tagged Twitter sentences provided as a development set in the work of Derczynski et. al. in [44]. In keeping with the example of [4], we have chosen a maximum entropy model (the OpenNLP tagger [15]) as a suitable candidate for performance comparison since other POS Tagging models employ different tagsets and different test data. Our undertaking of this work is modestly aimed at demonstrating some improvement to our known tagger performance using vectors from a previously untapped source (Twitter).

In testing **nlpnet**[4] POS tagging performance, we have taken a simplified approach:

1. Since our base case (usage at LeadSift) tagging has been done without emphasis on prefix/suffix features, we do not consider such features in this work. However, we include capitalization in training our network. This feature yields 5 additional vector dimensions.

2. For training of the OpenNLP tagger, we use the labelled WSJ dataset and then a combination of WSJ with 551 labelled Twitter sentences from [41].

3. In our calculation of per-word-accuracy, we consider only the first 36 tokens (Penn Treebank token list) presented in the Tag Description tables provided in the Appendix. The justification for such an exclusion lies in our observation that punctuation marks, hashtags and quotes are accurately recognized and tagged regardless of training method and therefore, such tags are deemed inconsequential to our ultimate result.

4. Using the recommendation of [4], we also choose a word window size of 5 and use the learning rates 10^{-3} for two epochs and 10^{-4} for a further 100 epochs. A larger number of epochs and subsequent learning rate changes may be considered. However, we limit the number of passes over our data so as to avoid lengthy training times but yet obtain satisfactory tagger performance.

5. In determining the precise form of our **target outcome**, we demonstrate '*what tags are given to our Twitter sentences before training with embeddings (ie. training on WSJ data only) and after our embeddings are included.*'

Approach			POS
Tagger	Labelled Data	Twitter Embeddings	(PWA %)
Nlpnet (100 hln)	WSJ	No	81.09
Nlpnet (500 hln)	**WSJ**	**No**	**81.77**
Nlpnet (100 hln)	WSJ	Yes	84.40
Nlpnet (500 hln)	**WSJ**	**Yes**	**85.03**
OpenNLP	WSJ	N/A	67.94
OpenNLP	WSJ + Twitter	N/A	74.28

Table 5.3: Tagger Performance in per-word-accuracy percentage. Embeddings are not applicable (N/A) to OpenNLP.

We have utilized their industry standard numbers of hidden layer neurons (hln) in **nlpnet** for comparison on the POS tagging task. In particular, the major resulting factor from variation of this quantity has been the observed difference in rate of tagging. Specifically, our model when trained with 500 hln tags 293 sentences per second (sps) while 100 hln yields 606 tagged sps.) Some difference in per-word-accuracy PWA is observed and reported; though we note that the disparity between our results of interest (PWA with embeddings vs. without embeddings) is emphasized.

In consideration of testing statistical significance (or lack thereof), we use the approach of Fonseca et. al. in [7]. We shuffle the outputs of both models in **bold** in Table 4.3 and estimate the likelihood that two random partitions (each containing all 269 test sentences) have an accuracy difference greater than or equal to that of our observed results. We take the null hypothesis to be that the results of both tagging models are from the same distribution and estimate the probability p that some actual accuracy difference will be obtained. As in the Fonseca et. al's approach we tag each of our 269 sentences and assign the tagged (each sentence is tagged with both models) sequences to two sets with equal probability of assignment. If we repeat this process Y times and let the number y of times in which the difference in accuracy between our sets exceeds or matches the difference in the actual results, we then let $p = \frac{y+1}{Y+1}$. Using $Y = 1000$ we conclude that there is a statistically significant difference between **Nlpnet (500 hln) + WSJ** without embeddings and **Nlpnet (500 hln) + WSJ** with embeddings given that $p < 0.05$.

5.3 Discussion

Upon our review of existing systems for Tweet tagging, we have seen two clustering based taggers; Gimpel et. al. [40] achieve 92.8% accuracy after training on labelled Twitter text and Ritter et. al. [41] obtain 88.4% tag accuracy using labelled and unlabelled Twitter data for training. Derczynski et. al. [44] identify errors in existing taggers and obtain 88.7% per-word-accuracy from their system [45] trained using unlabelled Twitter text and bootstrapping techniques. After evaluating the OpenNLP POS tagger performance on our Twitter test sentences, we attain a maximum accuracy of 74.28% when the labelled Wall Street Journal data is combined with 551 tagged Twitter sentences for training. The newly trained model (based on **nlpnet**) exceeds its own prior performance on Twitter sentence tagging by 3.26% when Twitter embeddings are used. We note that scalability is of concern given the sheer size of data and corresponding lengths of training time involved in using such systems as have been demonstrated in this work. However, we do not attempt to demonstrate computational gains from parallel/distributed approaches to training **word2vec** and/or **nlpnet** but we refer to works such as that of Ji et. al [54] and Sierra-Canto et. al [55] for methods of parallelizing our architectures of interest when needed. Our simple approach to Twitter POS tagging is empirically shown to be positively impacted by our incorporation of **word2vec** trained embeddings in **nlpnet**. Nevertheless, our approach is yet to meet the performance achieved by previously discussed models.

Chapter 6

Conclusion

We have seen that traditional part-of-speech taggers such as **nlpnet**[4], **OpenNLP**[15] (trained on currently available labelled data) do not achieve good performance on Twitter sentences (or more generally, we may say, sentences of the type frequently generated via social media activity). We have attempted to improve upon the performance of our existing **nlpnet** tagger which was trained solely on labelled non-Twitter data.

It has been shown that Twitter data spanning a relatively short time period (3 months) yields a large enough collection of sample sentences of sufficient variety to allow for unsupervised learning of word embeddings that can be used for word similarity/analogy tasks as well as in development/training of systems for other downstream NLP tasks such as Part-of-speech tagging. The sheer size of our leveraged Twitter dataset serves as but one demonstration of the virtually unlimited access to up-to-date (albeit unlabelled) data made possible by high velocity social media traffic generated daily.

The positive change in POS tagging performance on Twitter sentences gives validity to our claim that the inclusion of Twitter-based word embeddings may yield better tagged sentences when used to train systems designed for downstream (ie. beyond simple generation of word embeddings) NLP tasks.

6.1 Future Work

Given the improvements made by our current word embeddings, we propose firstly to expand our Twitter dataset considerably. The current state-of-the-art word embedding algorithm, Glove [20], has been shown to generate vectors of superior quality in various dimensions based on 2 billion tweets. The demonstrated success of their approach through evaluation on Similarity and Named Entity Recognition tasks offers an open challenge to enhance the tagging performance of **nlpnet** [4]. It may be of interest to

(a) Use even larger (eg. 1 billion sentences) samples of Twitter data for training of Word2vec thereby learning word vectors for a larger vocabulary of words. The performance increase from our inclusion of Twitter embeddings indicates that possibly even better POS Tagging performance may be obtained using vectors trained on more data. We also expect to obtain better tagger performance by including labelled Tweets in our supervised approach to training **nlpnet** given that OpenNLP has benefited from the inclusion of a small number of such labelled Tweets.

(b) Compare **nlpnet** performance on Twitter dataset via initialization of lookup table with vectors generated by GloVe vs Word2Vec.

(c) Use generated vectors to investigate possible performance increases for Chunking (Giving labels to "segments of a sentence with syntactic constituents such as noun or verb phrases (NP or VP)".[13]) We note that this task may present a significant challenge given the 140 character limitation on Twitter posts.

(d) Produce semantic role labels (SRL) for constituent parts of Twitter sentences using vector trained models. We note that currently, **nlpnet**[4] assigns the tag 'unknown' to words that it has not previously 'seen' in training when generating SRLs for a given sentence. This effect is minimized for each new term introduced in our vocabulary list (and corresponding lookup table of vectors).

Bibliography

[1] Tomas Mikolov, Kai Chen, Greg Corrado and Jeffery Dean. *Efficient Estimation of Word Representations in Vector Space*. arXiv preprint arXiv:1301.3781, 2013.

[2] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado and Jeffery Dean. *Distributed Representations of Words and Phrases and their Compositionality*. Advances in neural information processing systems 26, pp 3111–3119, Stateline, Nevada USA, Dec 10 2013.

[3] Tomas Mikolov, Wen-tau Tih and Geoffrey Zweig. *Linguistic Regularities in Continuous Space Word Representations*. HLT-NAACL. Vol. 13, pp 746–751, Atlanta, GA USA, June 10—12, 2013.

[4] Fonseca, Erick Rocha, and Joao Lus G. Rosa. *Mac-Morpho revisited: Towards robust part-of-speech tagging*. In Proceedings of the 9th Brazilian Symposium in Information and Human Language Technology, pp. 98–107. Ceara, Brazil, Oct 21–23, 2013.

[5] Fonseca, Erick and Alusio, Sandra M. *A Deep Architecture for Non-Projective Dependency Parsing*. Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing, pp 56–61, NAACL-HLT, June 5, 2015, Denver, Colorado, USA. May 31 —June 05, 2015.

[6] Fonseca, Erick and Rosa, Joo Lus G. *A Two-Step Convolutional Neural Network Approach for Semantic Role Labeling*. The 2013 International Joint Conference on Neural Networks, pp 1–7, Dallas, TX, USA, August 4—9, 2013.

[7] Fonseca ER, Rosa JL, Alusio SM. *Evaluating word embeddings and a revised corpus for part-of-speech tagging in Portuguese*. Journal of the Brazilian Computer Society. Vol 21, Issue 1, pp. 2:1–2:14, Feb 06 2015.

[8] Bick, E. *The parsing system PALAVRAS: automatic grammatical analysis of Portuguese in a constraint grammar framework*. PhD thesis, Department of Linguistics, Aarhus University, Jan 2000.

[9] Enrique Sanchez-Villamil, Mikel L. Forcada, and Rafael C. Carrasco. *Unsupervised Training of a Finite-State Sliding-Window Part-of-Speech Tagger*. J. L. Vicedo et al. (Eds.): EsTAL 2004, LNAI 3230, pp. 454-463, 2004. Springer-Verlag Berlin Heidelberg 2004.

[10] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer. *Feature-rich part-of-speech tagging with a cyclic dependency network*. In Conference of the North American Chapter of the Association for Computational Linguistics & Human

Language Technologies (NAACL-HLT), Edmonton, Canada, May 27—June 01, 2003.

[11] J. Gimenez and L. M'arquez. *SVMTool: A general POS tagger generator based on support vector machines.* In Conference on Language Resources and Evaluation (LREC), Lisbon, Portugal, May 26—28, 2004.

[12] L. Shen, G. Satta, and A. K. Joshi. *Guided learning for bidirectional sequence classification.* In Meeting of the Association for Computational Linguistics (ACL), Prague, Czech Republic, June 25—27, 2007.

[13] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. P. *Natural language processing (almost) from scratch.* Journal of Machine Learning Research, Vol 12, pp 2493—2537, 2011.

[14] Fonseca, E.R., Ao Lus, G., Rosa, J. *Mac-morpho revisited: Towards robust part-of-speech tagging.* In: Proceedings of the 9th Brazilian Symposium in Information and Human Language Technology, pp. 98—107, Oct 21—23, 2013.

[15] The Apache software foundation. URL `http://opennlp.apache.org`, Accessed on 10 Sept 2016.

[16] Richard Socher, John Bauer, Christopher D. Manning,and Andrew Y. Ng. *Parsing With Compositional Vector Grammars.* Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, Sofia, Bulgaria, Volume 1: Long Papers, pp 455–465, Aug 4–9 2013.

[17] Joseph Turian, Lev Ratinov, and Yoshua Bengio. *Word representations: a simple and general method for semi-supervised learning.* In Proceedings of ACL, pages 384—394, Uppsala, Sweden, July 11—16, 2010.

[18] Fabrizio Sebastiani. *Machine learning in automated text categorization.* ACM Computing Surveys, Vol 34, pp. 1-47, 2002.

[19] Bengio, Yoshua. "Learning deep architectures for AI." Foundations and trends in Machine Learning Vol 2, Issue 1, pp. 1—127, Jan 01 2009.

[20] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. *Glove: Global Vectors for Word Representation.* EMNLP. Vol. 14., pp 1532–43, Doha, Qatar, Oct 29 2014.

[21] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. *Indexing by latent semantic analysis.* Journal of the American Society for Information Science, Vol 41, Issue 6, pp 391–407, Sep 01 1990.

[22] Kevin Lund and Curt Burgess. *Producing high-dimensional semantic spaces from lexical co-occurrence.* Behavior Research Methods, Instrumentation, and Computers, Vol 28, pp. 203—208, 1996.

[23] Douglas L. T. Rohde, Laura M. Gonnerman, and David C. Plaut. *An improved model of semantic similarity based on lexical co-occurence*. Communications of the ACM, Vol 8, pp. 627—633, 2006.

[24] John A. Bullinaria and Joseph P. Levy. *Extracting semantic representations from word cooccurrence statistics: A computational study*. Behavior Research Methods, Vol 39, Issue 3, pp 510—526, 2007.

[25] Remi Lebret and Ronan Collobert. *Word embeddings through Hellinger PCA*. In Proceedings of the European Chapter for the Association of Computational Linguistics, pp 482–490, Gothenburg, Sweden, April 26—30, 2014.

[26] Yoshua Bengio, Rejean Ducharme, Pascal Vincent, and Christian Janvin. *A neural probabilistic language model*. Journal of Machine Learing Research, Vol. 3, pp 1137—1155, 2003.

[27] Ronan Collobert and Jason Weston. *A unified architecture for natural language processing: deep neural networks with multitask learning*. In Proceedings of ICML, pages 160—167, 2008.

[28] Andriy Mnih and Koray Kavukcuoglu. *Learning word embeddings efficiently with noise-contrastive estimation*. Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems, pp 2265–2273 .December 5-8, 2013, Lake Tahoe, Nevada, United States.

[29] Omer Levy, Yoav Goldberg, and Israel RamatGan. *Linguistic regularities in sparse and explicit word representations*. Proceedings of the Eighteenth Conference on Computational Natural Language Learning, pp 171–180, CoNLL 2014, Baltimore, Maryland, USA, June 26-27, 2014.

[30] R. Miikkulainen and M.G. Dyer. *Natural language processing with modular neural networks and distributed lexicon*. Cognitive Science, 15, pp 343–399, 1991.

[31] Teh, Yee Whye. *A Bayesian interpretation of interpolated Kneser-Ney*, School of Computing, National University of Singapore, Technical Report TRA2, 2006.

[32] Kim, Yoon, et al. *Character-aware neural language models*. Proceedings of the Thirtieth Conference on Artificial Intelligence, pp 2741–2749, Phoenix, Arizona, USA. , February 12-17 2016.

[33] Jozefowicz, Rafal, et al. *Exploring the limits of language modeling*. arXiv preprint arXiv:1602.02410, 2016.

[34] Collobert, Ronan, and Jason Weston. *A unified architecture for natural language processing: Deep neural networks with multitask learning*. Machine Learning, Proceedings of the Twenty-Fifth International Conference, pp 160–167, Helsinki, Finland, June 5-9, 2008.

[35] T. Sakaki, M. Okazaki, and Y. Matsuo. *Earthquake shakes Twitter users: real-time event detection by social sensors.* In Proceedings of the 19th international conference on World Wide Web (WWW), pages 851—860. ACM, April 26—30, 2010.

[36] A. Culotta. *Towards detecting influenza epidemics by analyzing twitter messages.* In Proceedings of the First Workshop on Social Media Analytics, pages 115—122. ACM,Washington DC, USA, July 25—28, 2010.

[37] J. S. Bridle. *Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition.* In F. Fogelman Soulie and J. Herault, editors, Neurocomputing: Algorithms, Architectures and Applications, pages 227—236. NATO ASI Series, 1990.

[38] L. Bottou. *Stochastic gradient learning in neural networks.* Fourth International Conference Neural Networks & their Applications, Nimes, France, November 4—8, 1991 : proceedings & exhibition catalog = Neuro-Nimes '91 : Journées internationales Les réseaux neuro-mimétiques & leurs applications, Nimes, France, 4-8 novembre 1991 : actes et catalogue de l'exposition sponsored by ARC, JSAI, & SEE ; with the support of EERIE & Multipôle régional du Languedoc-Roussillon. Published 1991 by EC2 in Nanterre . Written in French.

[39] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. *Dependency networks for inference, collaborative filtering, and data visualization.* Journal of Machine Learning Research (JMLR), Vo1, pp 49—75, Oct 2001.

[40] K. Gimpel, N. Schneider, B. OConnor, D. Das, D. Mills, J. Eisenstein, M. Heilman, D. Yogatama, J. Flanigan, and N. Smith. *Part-of-speech tagging for twitter: annotation, features, and experiments.* In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, pages 42—47. ACL, Oregon, USA, June 19—24, 2011.

[41] A. Ritter, S. Clark, O. Etzioni, et al. *Named entity recognition in tweets: an experimental study.* In Proceedings of the Conference on Empirical Methods in Natural Language Processing, pages 1524—1534. ACL, Oregon, USA, June 27—31, 2011.

[42] J. Foster, O. Cetinoglu, J. Wagner, J. Le Roux, S. Hogan, J. Nivre, D. Hogan, and J. van Genabith. *# hardtoparse: POS Tagging and Parsing the Twitterverse.* In Proceedings of the AAAI Workshop on Analyzing Microtext, San Francisco, California USA, Aug 07—11, 2011.

[43] P. Gadde, L. Subramaniam, and T. Faruquie. *Adapting a wsj trained part-of-speech tagger to noisy text: preliminary results.* In Proceedings of the Joint Workshop on Multilingual OCR and Analytics for Noisy Unstructured Text Data, pages 5:15:8. ACM, Sept 17, 2011.

[44] Derczynski, Leon, et al. *Twitter Part-of-Speech Tagging for All: Overcoming Sparse and Noisy Data*. Recent Advances in Natural Language Processing, pp 198–206 , Hissar, Bulgaria, Sep 9-11, 2013,

[45] GATE Twitter Part-of-Speech Tagger. University of Sheffield, `https://gate.ac.uk/wiki/twitter-postagger.html`, Accessed on Sept 28 2016.

[46] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Mareinkiewicz. *Building a large annotated corpus of English: the Penn Treebank*. Computational Linguistics, 19(2) pp 313-330, 1994.

[47] *Internet Live Statistics*, Real Time Statistics Project. `http://www.internetlivestats.com/twitter-statistics/` Accessed on 20 Oct 2016.

[48] Rehurek, Rahim. *Gensim Word2Vec*. Gensim Topic Modelling for Humans, `https://radimrehurek.com/gensim/models/word2vec.html` Accessed on 05 Oct 2016.

[49] Maaten, Laurens van der, and Geoffrey Hinton. *Visualizing data using t-SNE*. Journal of Machine Learning Research Vol 9, pp. 2579—2605, Nov 2008.

[50] Gutmann, Michael, and Aapo Hyvrinen. *Noise-contrastive estimation: A new estimation principle for unnormalized statistical models*.Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pp 297–304, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010.

[51] Rong, Xin. *word2vec parameter learning explained*. arXiv preprint arXiv:1411.2738, 2014.

[52] Morin, Frederic, and Yoshua Bengio. *Hierarchical Probabilistic Neural Network Language Mode*. In Artifical Intelligence and Statistics Conference. Vol. 5, pp 246—252, Barbados, Jan 06 2005.

[53] Clark, A. *Combining distributional and morphological information for part of speech induction*. In Proceedings of the tenth conference of the European chapter of the Association for Computational Linguistics, pages 5966. ACL, Budapest, Hungary, April 12—17, 2003.

[54] Ji, Shihao, Nadathur Satish, Sheng Li, and Pradeep Dubey. *Parallelizing Word2Vec in Shared and Distributed Memory*. arXiv preprint arXiv:1604.04661, 2016.

[55] Sierra-Canto, Xavier, Francisco Madera-Ramirez, and Victor Uc-Cetina. *Parallel training of a back-propagation neural network using CUDA*. The Ninth International Conference on Machine Learning and Applications, pp 307–312 , Washington, DC, USA, 12-14 December 2010.

[56] Rubenstein, Herbert, and Goodenough. *Contextual correlates of synonymy*. Communications of the ACM 8.10, pp 627-633, 1965.

[57] Schtze, Hinrich, and Jan O. Pedersen. *Information retrieval based on word senses.* In *Proceedings of SDAIR'95*, Las Vegas, Nevada, April 24—26, 1995.

[58] Hellinger distance. M.S. Nikulin (originator), Encyclopedia of Mathematics. URL: `https://www.encyclopediaofmath.org/index.php/Hellinger\_distance`, Accessed 28 Nov 2016.

[59] Martin, James H., and Daniel Jurafsky. *Speech and language processing.* International Edition 710, 2000.

[60] Yoshua Bengio. *Neural Net Language Models.* URL: `http://www.scholarpedia.org/article/Neural_net_language_models`, Accessed on 29 Nov 2016.

Appendices

Number	**Tag**	**Description**
1	CC	Coordinating Conjuction
2	CD	Cardinal Number
3	DT	Determiner
4	EX	Existential
5	FW	Foreign word
6	IN	Preposition or subordinating conjunction
7	JJ	Adjective
8	JJR	Adjective, omparative
9	JJS	Adjective, superlative
10	LS	List item marker
11	MD	Modal
12	NN	Noun, singular or mass
13	NNS	Noun, plural
14	NNP	Proper noun, singular
15	NNPS	Proper noun, plural
16	PDT	Predeterminer
17	POS	Possessive ending
18	PRP	Personal pronoun
19	PRP$	Possessive pronoun
20	RB	Adverb
21	RBR	Adverb, comparative
22	RBS	Adverb, superlative

Table 1: POS Tag Descriptions

Number	Tag	Description
23	RP	Particle
24	SYM	Symbol
25	TO	*To*
26	UH	Interjection
27	VB	Verb, base form
28	VBD	Verb, past tense
29	VBG	Verb, gerund or past participle
30	VBN	Verb, past participle
31	VBP	Verb, non-3rd person singular present
32	VBZ	Verb, 3rd person singular present
33	WDT	Wh-determiner
34	WP	Wh-pronoun
35	WP$	Wh-pronoun
36	WRB	Wh-adverb
37	OQUOT	-
38	COMMA	comma
39	COLON	colon
40	DOLLAR	dollar sign
41	PERIOD	period
42	HASH	hashtag
43	QUOT	-
44	LBR	-

Table 2: POS Tag Descriptions cont'd

Twitter Test Set POS Tag Distribution			
Tag	Frequency	Tag	Frequency
VBG	71	NNP	467
VBD	44	VB	453
VBN	34	WRB	10
VBP	67	CC	38
WDT	1	RBS	1
OQUOT	-	PDT	1
JJ	141	DOLLAR	-
WP	10	RBR	2
VBZ	115	PERIOD	-
DT	173	EX	0
LBR	0	IN	248
RP	18	HASH	-
NN	902	CD	31
FW	1	MD	31
POS	17	PRPD	0
TO	72	NNPS	15
COMMA	-	QUOT	-
PRP	167	JJS	3
RB	133	JJR	5
WPD	0	SYM	-
COLON	-	UH	26
NNS	84	LS	0

Table 3: Distribution of POS Tags for all 269 sentences of our chosen Twitter test set. Tags with frequency '-' indicate the tags not considered during testing.

WSJ POS Tag Distribution			
Tag	Frequency	Tag	Frequency
VBG	17863	NNP	114942
VBD	36993	VB	151736
VBN	24264	WRB	2555
VBP	14943	CC	28696
WDT	5205	RBS	516
OQUOT	8636	PDT	434
JJ	79924	DOLLAR	8835
WP	3034	RBR	3683
VBZ	25882	PERIOD	47557
DT	99270	EX	1038
LBR	1719	IN	120226
RP	2042	HASH	180
NN	348180	CD	44321
FW	262	MD	11748
POS	10593	PRPD	10044
TO	27020	NNPS	3020
COMMA	58935	QUOT	6606
PRP	30927	JJS	2345
RB	41690	JJR	3942
WPD	203	SYM	62
COLON	6050	UH	104
NNS	72445	LS	32

Table 4: Distribution of POS Tags for entire WSJ dataset

Messaging Technology Similarities			
sms	**Similarity**	*text*	**Similarity**
SMS	0.7638	reply	0.8854
aadhar	0.7637	txt	0.8721
hotmail	0.7572	FaceTime	0.8076
debits	0.7449	unfollow	0.8009
airtel	0.7399	stalk	0.7980
mpesa	0.7398	delete	0.7835
Whatsapp	0.7362	facetime	0.7647
M-PESA	0.7314	texting	0.7596
irctc	0.7309	subtweet	0.7476
sbi	0.7266	txts	0.7436

Table 5: *Twitter*, *sms* and *text* top 10 similarities

Mobile and Email Technology Similarities			
iphone	**Similarity**	*email*	**Similarity**
macbook	0.8589	e-mail	0.9354
imac	0.7948	query	0.8471
Iphone	0.7932	invoice	0.8203
iPhone	0.7882	address	0.8135
samsung	0.7822	request	0.8099
blackberry	0.7749	msg	0.8057
IPhone	0.7671	website	0.8023
ios	0.7640	acct	0.8006
6plus	0.7529	inbox	0.7951
acer	0.7506	message	0.7911

Table 6: *iPhone* and *email* top 10 similarities

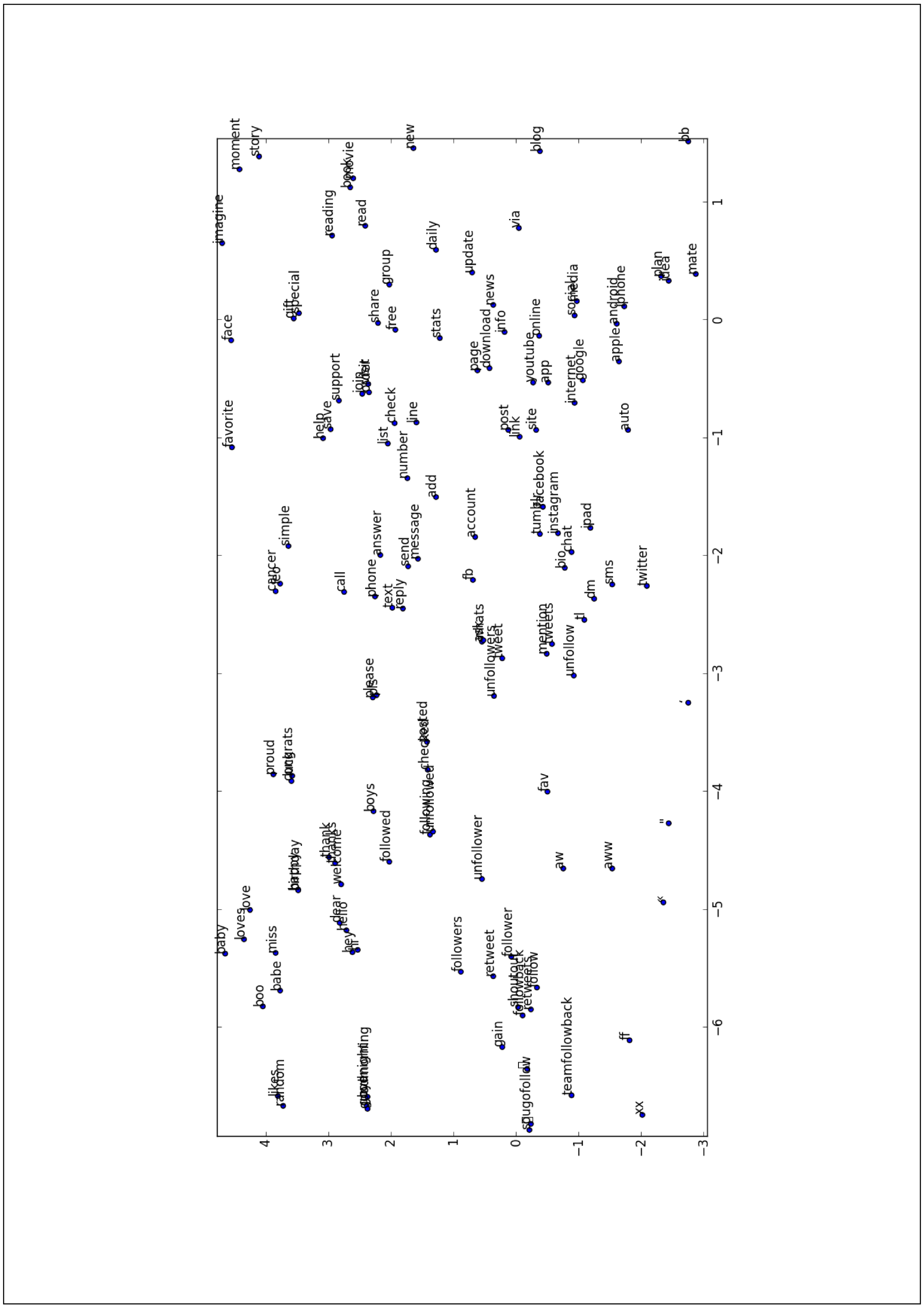


Figure 1: Word clusters identified in vector space representation