Measuring Brand Perception on Twitter using Context-Dependent Skip-grams

Yunjae Lee
St John's College

A dissertation submitted to the University of Cambridge in partial fulfilment of the requirements for the degree of Master of Philosophy in Advanced Computer Science

University of Cambridge
Computer Laboratory
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom

Email: yl457@cam.ac.uk

April 10, 2020

Declaration

I Yunjae Lee of St John's College, being a candidate for the M.Phil in Advanced Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: 11,513

Signed:

Date:

Abstract

In this thesis, I explore the possibility of computationally analysing people's perception of brands using a Neural Network Language Model. Current marketing professional employ a range of inaccurate and inefficient methodologies to gather user feedback in measuring brand perception, including surveys and focus groups. To automate this process, I use a state of the art neural language model called the Skip-gram model and train it on few hundred gigabytes of text mined from Twitter. An immediate benefit of this model is that it enables 'on-line learning', in that a trained model can continue training as more dataset becomes available. This forms stark contrast with qualitative methods such as surveys, as the cost of gathering more user data is marginal with this model and the model updates as it learns from more recent data.

More specifically, the skip-gram learns low-dimensional, efficient vector representation of words which are shown to be powerful in various NLP tasks including word similarity tasks, POS tagging and parsing. This thesis augments the skip-gram model with embedding matrices for different user *contexts*: *e.g.* gender, age and location. Such metadata of users is available in Twitter as users provide information about themselves in semi-organised manner. I mine and preprocess 8.8 billion tokens from 623 million tweets to obtain a training data with each sentence tagged with a set of contexts in which it is uttered.

The model learns context-specific representation of words and successfully reflect variations of language that correspond to particular demographic groups. The evaluation chapter presents both quantative and qualitative comparison of three models and confirms that the best-performing model learns context-dependent word semantics.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Brand is a collection of experiences regarding the product, employees and services offered by a company. Competing companies in the same product category often offer goods with a similar price range and functionalities. As a result, brand can be a key differentiating component in consumers' choice of one product over another [1]. Blind taste tests often reveal how customers fail to tell the brands apart from the products alone. A recent study shows that when given three tastings of unlabelled beer, two of which from the same brand, most people could not reliably single out the third tasting [2]. The degree to which brand affects purchase decisions is also demonstrated by another study with wine, where wine drinkers are shown to be willing to pay more for a label that is difficult to pronounce [3].

Marketing professionals employ various methodologies to discover how their brand is perceived. These include personal interviews and focus groups, where a group of people are asked to discuss their opinions of a certain company and its services. However, such qualitative techniques are not only inefficient in gathering data but also inaccurate, bringing into question the statistical significance of the hypothesis derived as a result.

This thesis proposes to automate this process by building a language model from an English corpus mined from Twitter. As social networks become more pervasive and personal, on platforms such as Twitter, utterances are now often paired with metadata about the speaker, *e.g.* time, location, gender and even politifical preferences. With a corpus of Tweets tagged with such a rich set of information, I trained a neural network language model that is sensitive to the contexts of the speaker. The aim of this project is then to design a system capable of answering empirical questions such as "what is the brand most associated with cars for women aged between 20 and 25 living in New York?" I will demonstrate the validity of the model by comparing its performance to that of context-insensitive models, on test data designed to measure how well models capture the inter-speaker variability of language. From the brand analytics standpoint, this method provides insights into consumer perception for specific demographics, enabling highly fine-grained, targeted marketing. Also, the language model used in this thesis enables *online learning*; in other words, incoming training data can be fed into the model to continue training and update its parameters. Hence, it is also possible to examine the temporal change in brand semantics over time.

I used a neural network language model known as the *Skip-gram model*, augmented with context information. I give a brief overview of vector space models and neural language models in Chapter 2, as well as a high-level overview of the Skip-gram model architecture. User-composed contents in online microblogs contain a huge degree of noise; hence compiling a sizeable and meaningful dataset is a challenge. Chapter 3 explains the procedure of mining a corpus from Twitter and preprocessing the data to obtain high quality corpora. I will describe the model architecture and the training process in Chapter 4, where I also detail various optimisation techniques used to either reduce the space requirements and speed up the training process. Finally, I confirm the capability of the model to represent linguistic variation amongst different demographics via a quantative evaluation with several carefully designed test sets. I finish the dissertation with some interesting qualitative results and perception rankings of brands in the US across 18 product categories computed by the model.

The main contributions of this work are:

- Extending the state-of-the-art language model to accommodate different categories of user contexts.
- Suggesting an efficient and accurate technique in computational brand analytics that represents online user sentiment well.
- Showing that it is possible extract a huge corpus from Twitter whose content are meaningful enough to construct a language model.

Chapter 2

Background

This chapter gives a brief overview of the background of this work. Earlier work on Computational Brand Analytics (CBA) using machine learning techniques is described in §2.1. This is followed by a survey of recent work on social applications of Natural Language Processing (NLP). After a few pointers into papers used in text normalisation, I move on to briefly outline several approaches to language modelling, finishing the chapter with a description of specific model implementation adopted as a starting point of this project.

2.1 Brand analytics and social NLP

Early work on Computational Brand Analytics dates back to Kuehn's work [4], in which he described brand shifting as a stochastic learning process based on consumers' past purchase patterns. Gupta (1988) took this hypothesis further and modelled the effect of sales promotion in detail, with the impact of market indices expressed as an exponential interpurchase time model, a multinomial logit model for brand selection and cumulative logit model for quantity of purchase [5].

More recent work in marketing research applied Bayesian Inference to predict product sales dynamics from Facebook applications data before launching, with the hypothesis that true consumer network characterstics can be inferred from aggregated diffusion patterns observed in the market [6]. Tirunillai *et. al* (2014) applied Latent Dirichlet Allocation (LDA) to manually collected 350,000 customer reviews in five market categories (personal computers, cellular phones, footwear, toys and data storage) [7, 8]. They showed that market sentiment can be inferred from user generated contents (*e.g.* social networks, product reviews and bulletin boards) by extracting a few key latent dimensions within consumer satisfaction space about the quality of products.

As more of life is presented online and recorded in natural language, there has been active research in the NLP literature to make interesting inferences in social science. Thomas *et al.* (2006) used Support Vector Machines (SVM) and Graph Mincuts to determine whether Congress suports or opposes the proposed legislation from the debate transcript [9]. Another recent work attempts to predict the success of a book based on statistical analysis of its literary style with SVM [10].

However, to my knowledge, there has been no published work that applies of ML and NLP techniques to modelling brand perception of specific demographics. The goal of this thesis is a brand analytics model that produces informative and insightful results from a marketing perspective.

2.2 Vector space and language model

Vector space model

Distributional semantics is a branch of linguistic semantics that is based on the distributional hypothesis: ‘words that often appear with similar contexts have similar meaning’ [11]. Here contexts mean words that appear close to the target word (usually within a few words). Distributional semantic analysis is often implemented concretely with a Vector Space Model (VSM), where a word is represented by a vector residing in a high dimensional space.

Each dimension of the vector space could either be mapped to an actual word (the magnitude of a vector on the dimension then denotes the relative frequency with which a target word co-occurs with this particular context word), or some abstract meaning that does not correspond to an actual word but could potentially be expressed with multiple different words. Both Latent Semantic Analysis (LSA) and neural network word embeddings involve dimensions in vector space that do not correspond to concrete words.

Vector space models have an appealing property that similarity between two words can easily derived from the corresponding vectors. The most common choice is cosine similarity, which for vectors **a** and **b**:

$$sim(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \cdot \|\mathbf{b}\|} = \cos(\theta),$$

where θ is the angle between the two vectors. The similarity value ranges from -1 to 1 and is greater if two words are more similar in meaning. For example, we would expect $sim(dog, cat) > sim(dog, democracy)$ for any reasonable model.

N-gram language model

In statistical language processing, n-gram models have been the state of the art for language modelling until recent years. On the most basic level, training an n-gram model amounts to counting the relative frequencies of word sequences of length n appearing in a document. The model then predicts the probability of a new document w_1^T as

$$\hat{P}(w_1^T) = \prod_{t=1}^{T} \hat{P}(w_t | w_{t-n+1}^{t-1}) \tag{2.1}$$

where w_i is the i^{th} word in the document, T is the length of the document and $w_i^j = (w_i, w_{i+1}, ..., w_j)$. $\hat{P}$ denotes estimated probability. This Markov model serves as an approximation of a true language model, but gives surprisingly good results. Concretely, back-off techniques are often used for good performance in order to combat data sparsity.

There are two obvious drawbacks to the n-gram model. Firstly, the n-gram model is an approximation to the true language model therefore is inevitably limited, as natural language writings are often coherent and humans take advantage of not only the previous word but also the previous sentence and the previous paragraph to understand the next word. Also, n-gram models are unable to take word similarity into context when making predictions: having seen *I have to walk my dog at the park* in the training corpus before, we expect the model to predict *I have to walk my cat in the garden* with reasonably high probability. This, however, is not achieved by n-gram models if the number of co-occurrences for one bigram is smaller than the other. The neural language model, explained in the next section, overcomes this by internally constructing a mapping from words to a vector representation, then performing prediction using the representations. As a result, different words with similar meanings get mapped to close vector representations first.

Neural language model

Recent years have seen an active increase of applications of neural networks in language modelling, since the pioneering work by Bengio, Ducharme, Vincent and Jauvin [12]. Similarly to n-gram models, Bengio's neural language model represents the conditional probability of the next word given the previous sequence of words.

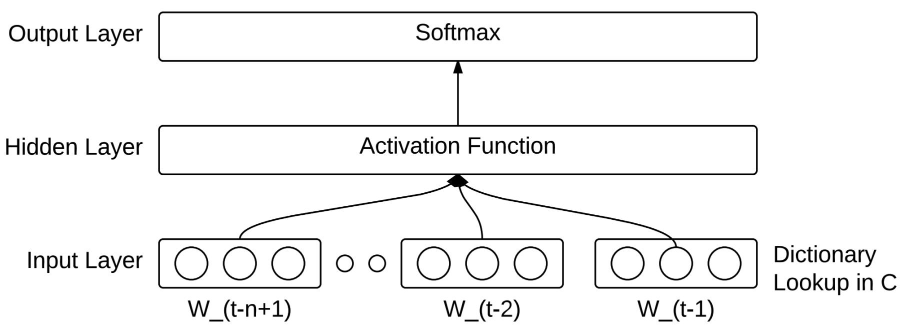


Figure 2.1: Neural network architecture. At the input layer are one-hot vectors that index into particular words in the embedding matrix C. The word embeddings are concatenated and passed to the activation function.

A neural language model has two parts: an encoder and a predictor (see Figure 2.1). An encoder, often called *autoencoder* in the neural network literature, is an unsupervised feedforward network that learns sparse representations of words. A trained autoencoder produces a mapping C from a word $w_t \in V$ to a vector representation, such that $C(w_t) \in \mathbb{R}^m$. This serves as a dimensionality reduction technique as the learned word representation (called *word embeddings*) is often low-dimensional hence compact. Using the word embeddings learned by the autoencoder, the predictor learns the function f that predicts the next word, given the embeddings of previous $n-1$ words:

$$\hat{P}(w_1^T) = \prod_{t=1}^{T} f(C(w_t), C(w_{t-1}), \cdots, C(w_{t-n+1})),$$

where $f(w_t, \ldots, w_{t-n+1}) = \hat{P}(w_t | w_{t-n+1}^{t-1})$ and $\sum_{i=1}^{|V|} f(i, w_{t-1}, \ldots, w_{t-n+1}) = 1$.

Stochastic gradient ascent and backpropagation algorithm is often used for training, with a goal to maximise the regularised log likelihood of the training corpus, *i.e.*

$$L = \frac{1}{T}\sum_t f(w_t, w_{t-1}, \ldots, w_{t-n+1}; \theta) + R(\theta),$$

where θ is the parameter set including the embedding matrix C and some additional parameters related to the predictor function [13]. The regularisation term is denoted as $R(\theta)$. At the output layer, the neural network computes the softmax function of the incoming activation to ensure that all probabilities sum to one.

$$\hat{P}(w_t|w_{t-1}, \ldots, w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum_i^{|V|} e^{y_i}}, \tag{2.2}$$

where y_i for each word i is computed as $y = Ax + \tanh(b + Cx)$ with parameters A, b and C.

Stochastic gradient ascent on the neural network is performed to learn the parameters θ, with an update rule:

$$\theta \leftarrow \theta + \eta \frac{\partial \log \hat{P}(w_t|w_{t-1}, \ldots, w_{t-n+1})}{\partial \theta}, \tag{2.3}$$

where η is the learning rate. Gradient ascent approaches local maxima by taking small steps towards the direction where the ascent is steepest. Note that update to the parameter is made after each training example, approximating the true gradient with the gradient of a single example. This implies that only a small fraction of the parameters need to be updated after each example.

Skip-gram language model

Recently, Mikolov *et al.* developed a novel neural network model called the skip-gram model [14]. Their approach uses every word to predict its every neighbour word (the standard neural language model in contrast only uses previous words to predict the current word), where a neighbour word is a word appearing within a predefined distance to the context word. This structure hence allows *skips* between the neighbour word and the context word. Therefore, the skip-gram model moves away from the n-gram paradigm by allowing prediction of a word with a word that occurs later.

This model gives state of the art performance in word similarity tasks while requiring lower computational cost compared to other neural architectures. The authors also demonstrate the ability of the model to capture interesting relationships with learnt word embeddings, *e.g.* *vec('King')* − *vec('Man')* + *vec('Woman')* is closest to *vec('Queen')*.

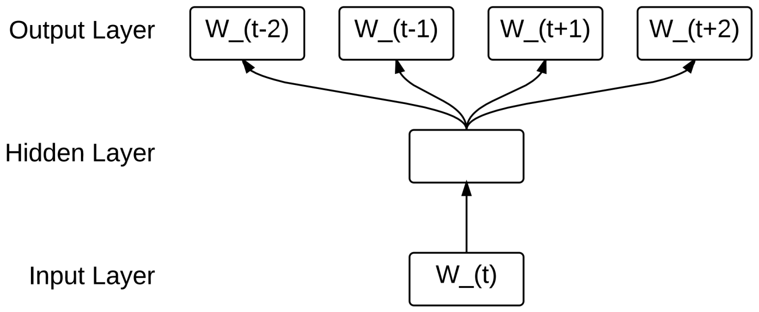


Figure 2.2: Skip-gram network architecture. Every word is used to predict the identities of its every neighbour word in both directions (not just the words after it).

Recent advances

While the skip-gram remains the state-of-the-art for word embeddings, it has also been shown that learning embeddings of paragraphs gives superior performance in text classification and sentiment analysis tasks compared to the bag-of-words approach [15]. On the other hand, Vilnis *et al.* demonstrated the advantages of moving away from learning representations of words as points in vector space, and instead suggested learning a Gaussian distribution over word embeddings to better handle uncertainty of word meanings.

They show that representing entailment between concepts as KL divergence between two distributions gives good performance in entailment prediction tasks [16].

2.3 Resources

Mikolov *et al.* released an implementation of the skip-gram model in C, which takes a corpus as an input and produces a dictionary of learned word embeddings as output[1]. Rehurek took this implementation and ported it into Python. By optimising heavy loops in the core backpropagation module in Cython and multithreading, his implementation offers both readability and performance improvement compared to the original release in C.[2] Cython is a C-extension for Python that allows calling C/C++ code from Python code, achieving C-level performance.[3]. It also interacts efficiently with large datasets in Python such as NumPy arrays. As Python provides well-integrated libraries with to interact with Twitter API and SQLite[4] database API, I took Rehurek's implementation of the skip-gram model as a starting point and augmented it with functionalities to support context-dependent word embedding matrices.

[1]`https://code.google.com/p/word2vec/`
[2]`https://github.com/piskvorky/gensim/blob/develop/gensim/models/word2vec.py`
[3]`http://cython.org/`
[4]`https://www.sqlite.org/`

Chapter 3

Data Mining and Preprocessing

From the very outset of the project, it was clear that obtaining a huge volume of tweets tagged with meaningful context information is key to its success. I outline the work involved in obtaining such a corpus in three parts: mining the tweets, preprocessing the content of the tweets and finally predicting the metadata of users based on available user information.

Cleansing social media data with a high degree of noise constitutes a research topic of its own. As social media data is heavily used in the NLP community, increasing amount of research is conducted in text normalisation directed specifically towards microblogs . Yang and Eisenstein (2013) presented a log-linear framework with Monte Carlo training for unsupervised text normalisation [17]. Several simple decision list classifiers were used in this thesis to predict several types of user metadata.

I finish the chapter with a detailed breakdown of demographic statistics of the training data, namely distributions of location, age and gender of the tweets corpus.

3.1 Mining the corpus

This process involved collecting a corpus of tweets of 8.8 billion tokens from 623 million tweets gathered over a period of approximately three months (August to November 2014[1]). Tweepy library[2] was used to access the Twitter API from Python, and collected data was compiled into an SQLite database.

I decided to perform tweet content normalisation and user metadata prediction separately, rather than both at the same time for each tweet. As most users tweet multiple times, it saves computation to perform user metadata prediction only once. This implied maintaining two separate databases: one for tweets and another for users who composed the tweets.

Tweets database

During initial mining from the Twitter API, I extracted seven fields for each tweet, with description provided below.

Field	Datatype	Description
`id_str`	`text`	Numeric string identifier for individual tweets.
`coordinates`	`text`	Location of the tweet in coordinates.
`created_at`	`text`	Time of authorship.
`place`	`text`	Location of the tweet in plain text.
`retweeted`	`integer`	Number of retweets received.
`content`	`text`	Content of the tweet.
`uid_str`	`text`	Numeric string identifier for individual users.

Table 3.1: Tweets database fields

Once the tweets database has been compiled, I normalised `content` of each tweet. I fed the normalised content into the neural network when training

[1]This thesis started as a summer internship at the Natural Language and Information Processing Group of the Computer Laboratory during this period, with Douwe Kiela and Steven Ghoussain.

[2]`https://github.com/tweepy/tweepy`

the model.

Users database

After the tweets database has been mined, I took the unique set of `uid_str` in the tweets database. Then, for each `uid_str` I extracted seven additional fields from Twitter and compiled them into a users database.

Field	Datatype	Description
`uid_str`	`text`	Numeric string identifier of the user.
`name`	`text`	Full name.
`screen_name`	`text`	Username on Twitter.
`location`	`text`	Location of the user.
`time_zone`	`text`	Timezone of the user's location.
`lang`	`text`	The language of user's choice.
`followers_count`	`integer`	The number of people following the user.
`description`	`text`	A plain text description about the user given by himself/herself.

Table 3.2: Users database fields

Once the database such as Table 3.2 has been compiled for all users in the tweets database, user metadata was predicted from the information available in the users database. Location was inferred from `time_zone` and `location`, gender was deduced from `name` and age estimated from the `description` field (see Section 3.3 for details).

3.2 Tweet normalisation

Preprocessing a corpus of this size will not be trivial. To obtain a dataset of reasonable quality within a given timeframe, a careful balance between normalisation quality and latency was needed. A pipeline[3] of preprocessing techniques was performed to the `content` of each tweet in the following order.

[3]Implemented by Steven Ghoussain during the summer internship.

Some key decisions regarding the precision of normalisation are discussed below.

Removing duplicate characters

A phenomenon particularly common in tweets is duplication of certain characters to express excitement or agreement, such as `loool` and `sweeeeeet!`. This was addressed with a concise and efficient character-level script that reduces substrings consisting of more than two consecutive, identical characters into a single character. Hence, `loool` and `greaaaaat` are mapped to `lol` and `great` respectively.

Expanding abbreviations and contractions

Abbreviations and contractions are extensively used in online microblogs. To correctly expand such occurrences back to their standard forms, I constructed a list of major contractions[4] and online acronyms[5]. For example, `brb` and `y'all could've` were converted to `be right back` and `you all could have`.

Spelling Correction

Another key source of noise in online microblogs is spelling mistakes. To address this, I have used an out-of-the-box solution, a Python spell-checker package known as Pyenchant[6] that also suggests likely words given an incorrect spelling of a word. For example, given `helo`, some examples of suggested possible alternative words are: `hello, halo` and `hell`.

Lowercasing

[4]`http://grammar.about.com/od/words/a/EnglishContractions.htm`

[5]Used entries combined from `http://www.netlingo.com/acronyms.php` and `http://www.webopedia.com/quick_ref/textmessageabbreviations.asp`

[6]`http://pythonhosted.org/pyenchant/`

Online microblog users tend to ignore casing rules and write entirely in lowercase, and this is more pronounced among personal users than corporate or group users on social media. Therefore, following a common approach in Tweet normalisation, I have opted to lowercase all tokens for simplicity and performance, at the expense of some minor loss of accuracy.

Named entity recognition

Original	Preprocessed
`yeeeees`	`yes`
`shouldve`	`should have`
`yolo`	`you only live once`
`pizza hut`	`pizza_hut`
`@disneyparks`	`disneyland`
`$six`	`six_flags`

Table 3.3: Tweet normalisation results.

As this project attempts to model the semantics of brand names in a context-sensitive way, it is of prime importance to correctly recognise key named entities in tweets. Stanford's CRF NER system was initially chosen to perform the recognition task[7]. However, visual inspection of results revealed that Stanford's system is too inadequate for recognising entities in online microblogs, presumably because the system was trained on newswire transcripts and articles which are vastly different from the majority of written language found online. Another issue with Stanford's NER system was that its latency was too big to run on the entire tweets database.

Therefore, I have constructed a named entity dictionary of key interest words. These words include brand names, sports team names and additional words that are expected to bear varying semantic representation for different demographics.

[7]`http://nlp.stanford.edu/software/CRF-NER.shtml`

A key function of Twitter is that users can personally 'mention' other users by prepending the @ sign to the desired person's username. As many companies and sports teams have accounts on Twitter, this functionality is actively utilised by users to mention companies in their discourse. Another way to discuss (usually the financial aspects of) a company on Twitter is to use the $ symbol prepended to the NASDAQ ticker symbol of the company. See the following microblogs for examples of both.

```
Custom @adidas gear showed up today!  Even our big men have
swag!  Look good, Play good!  #BeTheStandard #SpeedKills.
```

```
Jeez that interview last night on @MadMoneyOnCNBC with Doug
Parker from $AAL really crushed this group
```

As both methods are major channels of communication on Twitter, these occurrences were identified and converted to original names of entities in the recognition stage. See Table 3.3 for the input and output of the normalisation module.

3.3 User metadata prediction

With the user database mined as in Table 3.2, the metadata of each user was predicted from the information available in the database, to be later merged with each tweet composed by that user so that each utterance in the final database is tagged with corresponding user metadata. Users on Twitter provide personal information in a semi-organised manner, with related information often scattered across several fields (*e.g.* some users describe their location in the `description` field, rather than `location`).

Location

First and foremost, a gazeteer of location names was compiled, containing: 4409 major US cities, 749 major Canadian cities, 1258 major UK cities and 71 major world cities by population, in addition to 194 countries and 64 American/Canadian states.

User location was classified by detecting location keywords in `location` and `time_zone` fields. Hypothesied user location was arranged into three new fields: `city, state` and `country`, where the `state` is only meaningful in certain countries, namely US and Canada. A high-level algorithm for predicting a user's location is given in Algorithm 1.

`location`	`time_zone`	`city`	`state`	`country`
	Netherlands			netherlands
	Bangkok	bangkok		thailand
Indiana	US & Canada		indiana	us
Boulder	US & Canada	boulder	colorado	us
Houston, TX	US & Canada	houston	texas	us

Table 3.4: Examples of location prediction results. Three columns on the right list *inferred* user properties.

Table 3.4 shows predicted locations of users from some example input fields; namely, five different types of location descriptions are captured by the prediction module.

1. US/Canada
   (a) (`city, state`): If a token is a valid state in US, the previous token is taken to be `city`. Then, `city, state` and `country` are updated accordingly.
   (b) `city`: if a token is recognised as a valid city in US, then `city, state` and `country` are updated accordingly.
   (c) `state`: if a token is a valid US state, `state` and `country` are updated accordingly.

2. World

    (a) `country`: if a token is a valid country name, `country` is updated accordingly.

    (b) `city`: if a token is avalid city name, `city` and `country` are updated accordingly.

Algorithm 1 Outline of location prediction algorithm.

```
1. Search for country names in location and time_zone.
2. If country is either US, Canada or not found, search for US/Canadian
   state names.
3. If state is found, the word preceding the state is assumed to be city.
4. If a country but no state is found, search for world city names (non
   US/Canada).
```

Gender

Gender classification was performed based on users' first names in the `name` field. A simple decision list classifier was constructed using the US census data regarding popularities of male and female first names[8]. The data gives frequency in percent of a given first name for a given gender, *e.g.* 'James' consisted of approximately 3.32% of all American males at the time of the survey. For gender-ambiguous first names such as Charlie, two genders were compared based on the relative frequency and the gender with the higher relative frequency was taken. See Table 3.5 for some examples for gender prediction.

Approximately 6% of the corpus was tagged with hypothesised gender information (see Figure 3.2b).

Age[9]

[8]`http://www.census.gov/topics/population/genealogy/data/1990_census/1990_census_namefiles.html`

[9]Implemented by Steven Ghoussain during the summer internship.

`name`	`gender`
Cristian Rivas	male
Quinton Cameron	male
Robbie Macgregor	female
jen	female
adriana	female

Table 3.5: Examples of gender prediction results. The `gender` field was predicted from the `name` field.

Twitter does not ask for users' date of birth or age during signup. Hence, this information tends to be included in the `description` field, often at the beginning rather than at the end. The age predictor was designed to detect possible occurences of age in the first four characters in the `description` field.

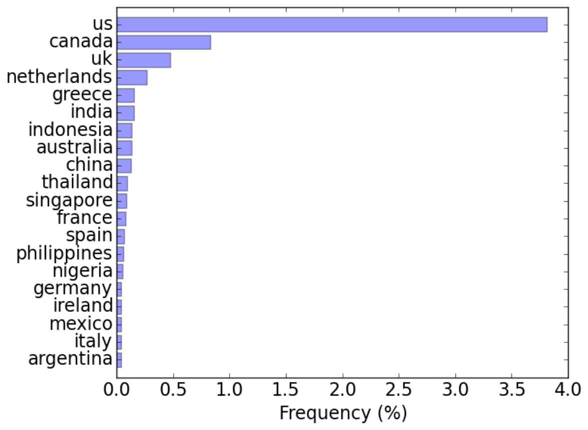
`description`	`age`
18, #SandrasSoldiersForever	18
I'm 16 #TeamMindless.MUSIC IS LIFE.	16
3rd Year CTP student. Chairman of @MMUCBasiks.	
10 days till I meet my idols	

Table 3.6: Examples of age prediction results. `age` column shows hypothesised age of the user.

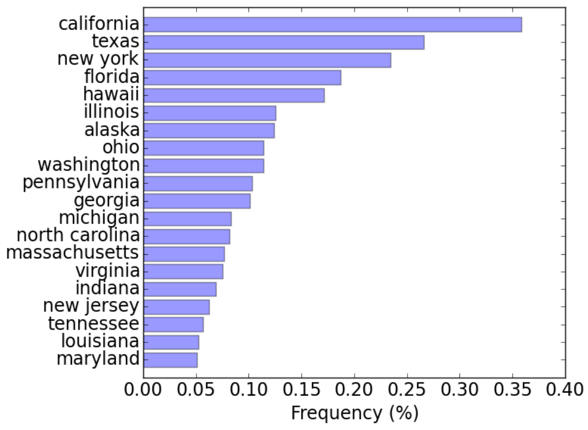
Note that a basic sanity check was performed so that numbers that can clearly not be an age were eliminiated, including ordinal numbers and quantities (*e.g.* 12 months). Approximately 0.25% of the tweets database (22 million tweets) were tagged with age metadata.

3.4 Tweet demographic statistics

Once every tweet has been normalised and every user's metadata predicted, each tweet in the tweets database was tagged with corresponding user's metadata (if available). Among 623 million tweets that have been compiled, the frequency of those with context information was determined and plotted on a

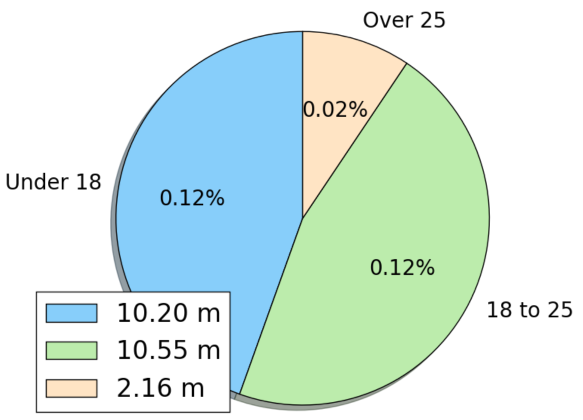


(a) Top 20 countries with the largest number of tweets.

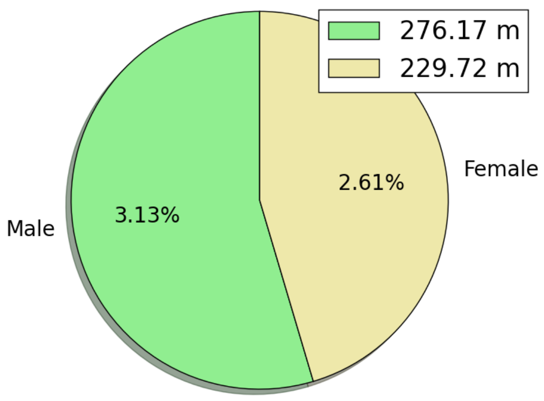


(b) Top 20 states with the largest number of tweets within US/Canada.

Figure 3.1: Distribution of countries and US/Canadian states.



(a) Age distribution of tweets



(b) Gender distribution of tweets.

Figure 3.2: Distribution of age and gender.[10]

chart. Figure 3.1a gives the histogram for the distribution of countries among tweets geotagged with country. Unsurprisingly, the US has the largest number of tweets, followed by Canada, UK and Netherlands. Within US and Canada, California constitutes the largest state in terms of the number of tweets, with Texas, New York and Florida next in rank. The top five results for states aligns with the order of US states by population.

[10]Pie patches show proportions with respect to entire tweets. Numbers in legends denote the number of tweets with corresponding context.

In terms of age distribution, the young demographic did comprise the majority of the online population, with people under 25 making up more than 90% of people with predicted age. Gender distribution shows that tweets composed by males slightly outnumber tweets by their female counterparts.

Chapter 4

Model Design and Training

This chapter gives the detail of the model structure, including the vocabulary encoding, network architecture, regularisation and training procedure. At a high-level, the model's architecture is based on Mikolov's skip-gram model, augmented with additional embeddings at the input layer such that the model is now sensitive to user contexts; hence the model presented in this thesis is called the *Context-dependent skip-gram* (CDSG).

4.1 Model

Hierarchical softmax

The original neural probabilistic language model as presented by Bengio *et al.* suffers from a highly inefficient softmax normalisation at the output layer (see Equation 2.2), as the final conditional probabilities are normalised over $|V|$ different possible words and the size of the vocabulary makes the computation impractical (often 10^5–10^7 terms).

Several workarounds have been suggested for this problem, one of which is known as the *hierarchical softmax* developed by Morin and Bengio [18]. Hierarchical softmax is a computationally efficient approximation to the full softmax, where a binary tree is built with all $|V|$ words in the vocabulary as the leaf nodes. The inner nodes denote some semantic grouping of their child nodes. Given a binary representation $(b_m(w), b_{m-1}(w), \ldots, b_1(w))$ of a word $w \in V$, where $b_j(w)$ denotes the j$^{\text{th}}$ node on the path from the root to the leaf node w, the conditional probability of the next word can be approximated as:

$$P(w|w_{t-1}, \ldots, w_{t-n+1}) = \prod_{j=1}^{m} P(b_j(w)|b_{j-1}(w), \ldots, b_1(w), w_{t-1}, \ldots, w_{t-n+1}). \tag{4.1}$$

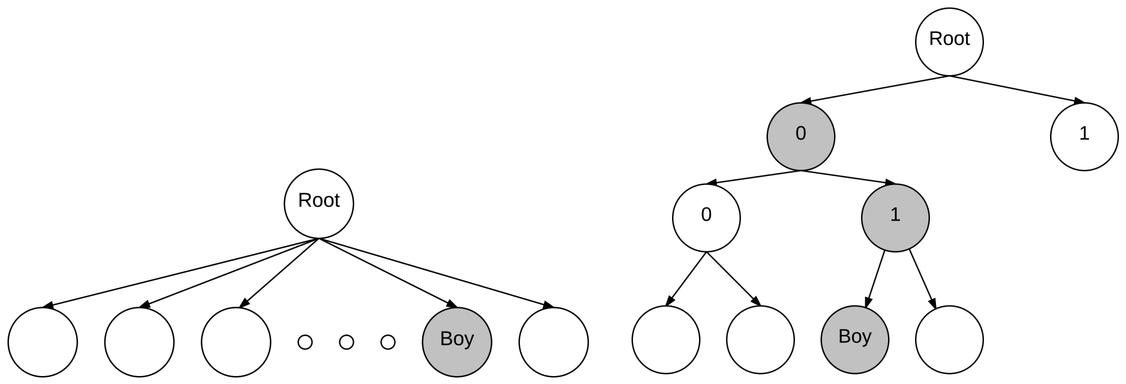
By keeping the binary tree balanced, this structure provides an exponential speed-up, on the order of $\frac{|V|}{\log_2 |V|}$. Note that we now need to maintain an *additional embedding matrix* for all nodes in the binary tree. In Equation 4.1, $w, w_{t-1}, \ldots, w_{t-n+1}$ are standard word embedding vectors, whereas $b_j(w), b_{j-1}(w), \ldots, b_1(w)$ are individual bits of Huffman node embeddings, explained further in the next section.

In their paper introducing negative sampling, Mikolov *et al.* give the following formulation for hierarchical softmax in the skip-gram model [19]. Standard softmax output in the skip-gram model defines $p(w_{t+j}|w_t)$ as:

$$p(w_O|w_I) = \frac{\exp({v'_{w_O}}^\intercal v_{w_I})}{\sum_{w=1}^{|V|} \exp({v'_w}^\intercal v_{w_I})}, \tag{4.2}$$

where v_w and v'_w are standard embeddings and Huffman node embedding of w, respectively. The dot product ${v'_{w_O}}^\intercal v_{w_I}$ gives a measure of 'similarity' between the target and the context word. The intuition is that the higher the similarity is, the more weight the target word receives during prediction.

In a binary tree vocabulary where each word w can be reached by some



(a) Original vocabulary as flat list.

(b) Binary tree vocabulary with hierarchical softmax.

Figure 4.1: Vocabulary structures of orignal neural network language model and hierarchical sampling.

path from the root, let $n(w, j)$ be the j-th node on the path from root to w. Also let $L(w)$ be the length of the path to w. Hence, $n(w, 1) = root$ and $n(w, L(w)) = w$. For any inner node n, let $ch(n)$ be the left child of node n. Then, the conditional probability can be given as:

$$P(w|w_I) = \prod_{j=1}^{L(w)-1} \sigma(\llbracket n(w, j+1) = ch(n(w, j)) \rrbracket \cdot {v'_{n(w,j)}}^{\intercal} v_{w_I}), \tag{4.3}$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$ and $\llbracket x \rrbracket = 1$ if x is true and -1 if otherwise. v_{w_I} and $v'_{n(w,j)}$ are standard embedding of the input word and Huffman embedding of the j-th node on the path to the output word. Similarly to Equation 4.2, the dot product represents a measure of similarity between the input and output word, with the output word decomposed into several layers of clusters, each making different contributions to the semantic representation of the output word.

The sigmoid function σ acts as a normalisation technique, as $\sigma(x) + \sigma(-x) = 1$. Hence, for a given inner node w_1 with child nodes w_2 and w_3, $\sigma(w_2|w_1) + \sigma(w_3|w_1) = 1$, therefore all transition probabilities for every node in each level in the tree sum to 1.

Vocabulary

For the binary encoding of words, an optimal prefix code known as the Huffman encoding scheme was chosen [20]. Constructing a Huffman binary tree starts by sorting all words by their frequency, making the two least frequent words into leaf nodes. Then their parent node is created with the sum of its children's frequencies as its frequency. Excluding the two leaves (but including their parent node), two least frequent words are again selected, and the process is repeated until a single root note remains with the sum of every word's frequency.

I first constructed a Huffman binary encoding of every vocabulary word, then built a binary tree structure based on the encoding, as in Figure 4.1b. Then, each bit of the Huffman embedding of a word represents a 'left or right' decision in the tree, which when combined in series ultimately forms the semantic representation of the word.

Network Architecture

Figure 4.2 shows the architecture of the context-dependent skip-gram. It has three sets of parameters (or *weights* in the neural network literature): main word embeddings, context word embeddings and Huffman node embeddings. Main word embedding is a matrix $M_{main} \in \mathbb{R}^{|V|\times k}$ whose i-th row contains a k-dimensional vector for the i-th word in the vocabulary. An embedding for a particular word can be accessed with a *one-hot vector*: $o' = (0, 0, 0, \ldots, 1, \ldots, 0, 0, 0)^\intercal \in \mathbb{R}^{|V|\times 1}$ as $w' = {M_{main}}^\intercal\, o' \in \mathbb{R}^{k\times 1}$.

Context embeddings M_c are of the same dimension as M_{main} but contain word embeddings particular to a certain user context c that, when combined with the main word embedding, produces final word embeddings for that context. Hence, the embedding of the word *boy* for male in the US is obtained

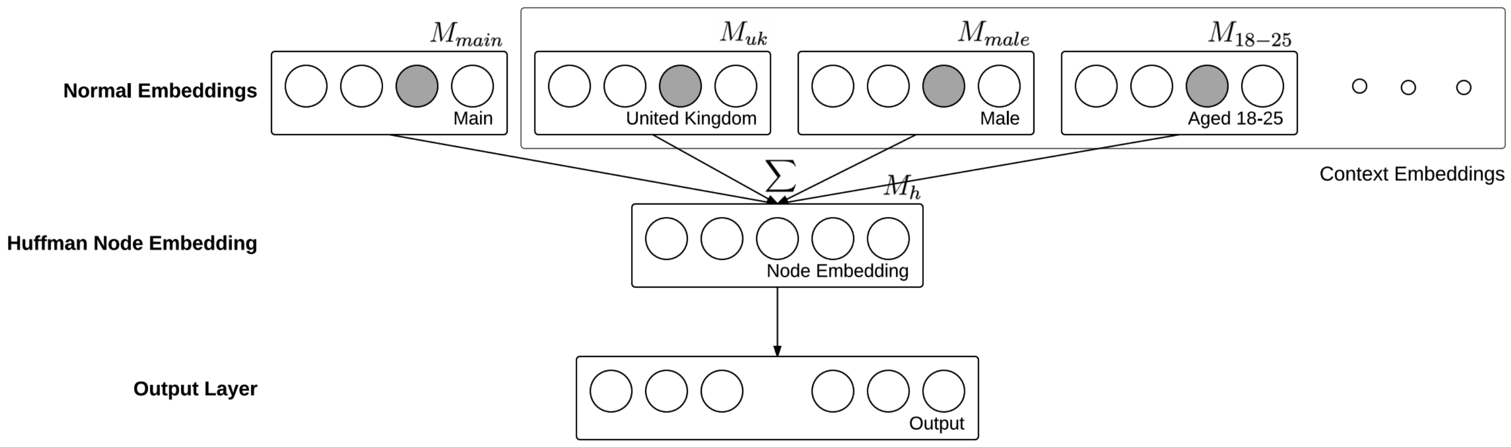


Figure 4.2: Skip-gram network architecture. Given an input words and its contexts, corresponding rows are accessed from the main embedding matrix M_{main} and appropriate context embeddings matrices M_c, and summed in the hidden layer.

as:

$$w'_{boy} = (M_{main} + M_{male} + M_{us})^{\intercal}\, o'_{boy}$$

The Huffman node embedding matrix $M_h \in \mathbb{R}^{Z \times k}$ contains k-dimensional embeddings for each inner node in the Huffman vocabulary tree. With input and output words w_I and w_O, and a given node level j of the output word, the network computes ${v'_{n(w_O,j)}}^{\intercal} v_{w_I}$ as output. Note that v_{w_I} is the embedding for word w_I in the main word embedding matrix M_{main}, whereas $v'_{n(w_O,j)}$ is the Huffman node embedding for node $n(w_O, j)$ in the node embedding matrix M_h.

Regularisation

Hinton *et al.* developed a biologically-inspired regularisation technique to avoid co-adaptation of weights in neural networks [21]. This technique, known as *Dropout*, randomly omits a hidden unit (sets to zero) from the network with some probability p. This is amounts to taking an average of an exponential number of different networks in a very efficient manner, as well as preventing a hidden unit from rely on the presence of any other.

Dropout was implemented in the CDSG model by randomly excluding some weights during computation of the dot product at the output layer.

4.2 Training

The model was trained using stochastic gradient ascent and backpropagation. Every training example was only fed into the network once and the weights were updated after each example. Only a small number of parameters are updated after the network sees each example: assuming the sentence has contexts *US* and *female*, only two rows of dimension 1×200 in matrices $M_{main}, M_{us}, M_{female}$ and M_h are updated for each (w_I, w_O) pair.

Objective function derivation

The training objective of the model is to find the set of embeddings that maximises the average log probability of the training data:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} \log p(w_{t+j}|w_t), \tag{4.4}$$

where T is the size of the training corpus. Recall that $\log p(w_{t+j}|w_t)$ in the context of skip-gram is given in Equation 4.3 and repeated below:

$$\prod_{j=1}^{L(w)-1} \sigma([\![n(w,j+1) = ch(n(w,j))]\!] \cdot {v'_{n(w,j)}}^{\intercal} v_{w_I}). \tag{4.5}$$

Let $sim = {v'_{n(w,j)}}^{\intercal} v_{w_I}$ and $code = n(w, j+1)$ (in bits, hence either 0 or 1; 0 represents the left child, while 1 represents the right). Then the expression inside the product can be re-written as:

$$\sigma_{obj}(code, sim) = \frac{\exp((1 - code)sim)}{1 + \exp(sim)}. \tag{4.6}$$

The above holds because in the case where $code = 0$ (*i.e.* $j + 1^{\text{th}}$ node is the left child of the j-th node):

$$\sigma_{obj}(0, sim) = \frac{\exp(sim)}{1 + \exp(sim)} = \frac{1}{1 + \exp(-sim)} = \sigma(sim).$$

The expression within $[\![\cdot]\!]$ in Equation 4.5 is true because we choose $ch(n)$ to be the left child of node n. Consequently, it evaluates to 1. If, on the other hand, $code = 1$:

$$\sigma_{obj}(1, sim) = \frac{1}{1 + \exp(sim)} = \sigma(-sim).$$

The expression $[\![\cdot]\!]$ in this case evaluates to -1. Note that $\sigma_{obj}(0, sim) + \sigma_{obj}(1, sim) = 1$, $\forall sim$.

Taking the logarithm of Equation 4.6, we get

$$\log \sigma_{obj}(code, sim) = (1 - code)sim - \log(1 + \exp(sim)).$$

If we take its derivative with respect to sim,

$$\begin{aligned} \frac{d(\log \sigma_{obj}(code, sim))}{dsim} &= 1 - code - \frac{\exp(sim)}{1 + \exp(sim)} \\ &= 1 - code - \sigma(sim). \end{aligned} \tag{4.7}$$

As $sim = {v'_{n(w,j)}}^\intercal v_{w_I}$, where $v'_{n(w,j)} = {M_h}^\intercal o'_{w_O}$ and $v_{w_I} = (M_{main} + \sum_c M_c)^\intercal o'_{w_I}$ (where o'_{w_I}, o'_{w_O} are one-hot vector to index into words w_I and w_O respectively), the derivative of sim with respect to each element of the weights are:

$$\frac{dsim}{dM_{main}(i,j)} = \sum_k M_h(k,j) \tag{4.8}$$

$$\frac{dsim}{dM_c(i,j)} = \sum_k M_h(k,j). \tag{4.9}$$

assuming i is the index of w_I and k ranges over the indexes of the nodes of w_O. $M_{main}(i,j)$ accesses the element of i-th row and j-th column in matrix M_{main}. Also,

$$\frac{dsim}{dM_h(k,j)} = (M_{main} + \sum_c M_c)(i,j). \tag{4.10}$$

Gradient ascent

Gradient ascent algorithm increments the parameters with the gradient of the objective function with respect to the parameter, multiplied by the learning rate. This updates the parameters in the direction of the steepest gradient.

$$\theta \leftarrow \theta + \eta \frac{\partial f_{obj}(\theta)}{\partial \theta}.$$

This can be rewritten into

$$\theta \leftarrow \theta + \eta \frac{\partial sim}{\partial \theta} \frac{\partial f_{obj}(sim)}{\partial sim}.$$

Note that the latter partial derivative is given in Equation 4.7. Hence, this gives

$$\theta \leftarrow \theta + \eta \frac{\partial sim}{\partial \theta} (1 - code - \sigma(sim)). \tag{4.11}$$

Letting $\theta = M_z(i,j)$ where $z \in \{main, c, h\}$, the appropriate gradient $\frac{\partial sim}{\partial M_z(i,j)}$ is given in eqs. (4.8) to (4.10).

Backpropagation

Standard backpropagation is performed to update the weights of the network.

In forward propagation, a single row in M_{main} and M_c interacts with multiple rows in M_h, as the output word is decomposed into several nodes. Hence, the gradient of M_{main} and M_c is accumulated over all nodes on the path to the output word and updated at the end. On the other hand, each row of M_h interacts only with one row of M_{main} and each of M_c, hence can be updated immediately.

Algorithm 2 Outline of training process.

```
For each inner node along to path to the output word $w_O$:
    Compute a forward pass through the network, obtain $1 - code - \sigma(sim)$.
    Update $M_h$ with the gradient of $M_h$: sum of relevant rows in $M_{main}$ and
      $M_c$.
    Accumulate gradients of $M_{main}$ and $M_c$.
Update $M_{main}$ and $M_c$ with the accumulated gradient.
```

Context-specific learning rate

Mikolov's implementation uses a single global learning rate that linearly decays from an initial value. All weights in M_{main} and M_h are updated with the same learning rate.

For the context-dependent skip-gram, given that examples with different contexts appear with different frequencies in the training data, it makes sense to have a separate learning rate for each context c. This enables the network to learn relatively little from a single example of frequently occurring contexts, and to pay more attention to examples with rare and informative contexts, thereby learning more from those occurrences.

The learning rate $\eta(c, t)$ for the context c at time t during training evolves as:

$$\eta(c, t) = \max(\eta_{min}, \eta_0 \times (1 - \frac{\delta_t(c)}{\delta_{total}(c)})).$$

η_{min}, η_0 are predefined values for initial and final learning rates (shared by all contexts). $\delta_t(c)$ is the number of words associated with context c seen so far up to time t during training, and $\delta_{total}(c)$ is the total number of words tagged with context c in the training corpus. Empirically, η_{min}, η_0 were chosen as 0.0001 and 0.1.

4.3 Implementation

Cython

Rehurek's implementation was optimised for selected hotspots by calling core C routines and BLAS functions within Cython code. As Cython's memory management is not thread-safe, implementing training and dropout in Cython involved careful use of global mutex called GIL (Global Interpreter Lock) to prevent several threads from accessing Python bytecodes simultaneously.

```
for a in range(0, size):
    syntemp[a] = syn0[indexes[j] * size + a]

# Summing context embeddings
for x in range(0, context_length):
    for a in range(0, size):
        syntemp[a] += syncon[ (contexts[x]*vocab_size*size)
                 + indexes[j] * size + a ]

# Dropout
for a in range(0, size):
    if (xor32() % 100) < dropout_param:
        syntemp[a] = 0
```

Listing 4.1: Cython code for dropout regularisation at the hidden layer.

Listing 4.1 shows a Cython code snippet to implement summation of main and context word embeddings at the input layer and drop hidden units out with some probability afterwards.

BLAS functions

BLAS (Basic Libear Algebra Subprograms) is a set of interfaces to perform linear algebra operations with heavy optimisations in C and Fortran. As several operations in the backpropagation routine involve computing the dot product of vectors, BLAS functions were used to optimise a loop of similar linear algebra operations. An example is the `dsdot(n,x,incx,y,incy)` function, which computes the dot product of vectors `x` and `y` of size `n`, with increment `incx` and `incy` to index each element of the vectors.

Precomputed sigmoid table[1]

As Equation 4.11 shows, computing the gradient involves obtaining $\sigma(sim)$. As evaluating the sigmoid function on every training example can lead to a significant performance hold-up, the sigmoid function was precomputed in discretised steps and stored in a table, effectively replacing explicit sigmoid evaluations in the core training routine with dictionary lookups.

```
# computing the sigmoid table.
for i in range(SIG_TABLE_SIZE):
 SIG_TABLE[i] = exp((i / SIG_TABLE_SIZE * 2 - 1) * MAX_SIG)
 SIG_TABLE[i] = (SIG_TABLE[i] / (SIG_TABLE[i] + 1))

# f is the vector similarity (dot product)
if f <= -MAX_SIG or f >= MAX_SIG:
 continue
f = SIG_TABLE[((f + MAX_SIG) * (SIG_TABLE_SIZE / MAX_SIG / 2))]
```

Listing 4.2: Cython code for precomputing and accessing sigmoid function.

[1]Optimisation used by Mikolov *et al.* in the C implementation: `https://code.google.com/p/word2vec/source/browse/trunk/word2vec.c`.

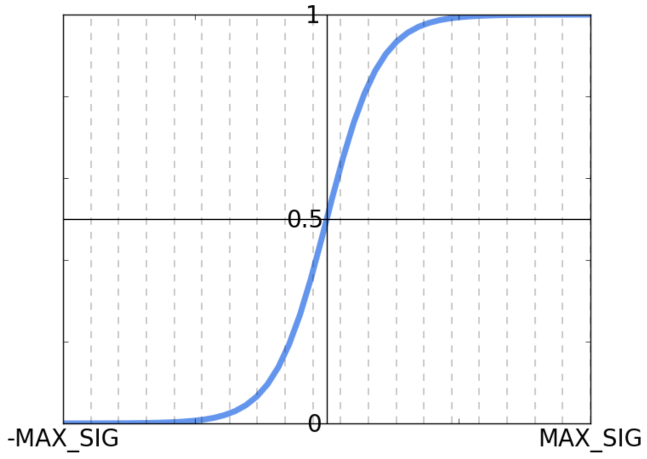


Figure 4.3: The sigmoid function is discretised into SIG_TABLE_SIZE pieces.

In Figure 4.3, note that $\sigma(x) \approx 1$ if x is large and $\sigma(x) \approx 0$ if x is small. Hence we define a boundary MAX_SIG such that we approximate the function as 0 and 1 if x is beyond either barrier. Within that range, we discretise the function into SIG_TABLE_SIZE pieces. Concretely, MAX_SIG and SIG_TABLE_SIZE were chosen to be 6 and 1000, respectively.

In Listing 4.2, SIG_TABLE is defined as:

$$\begin{aligned}\texttt{SIG\_TABLE[i]} &= \frac{\exp((\frac{2*i}{\texttt{SIG\_TABLE\_SIZE}} - 1) * \texttt{MAX\_SIG})}{\exp((\frac{2*i}{\texttt{SIG\_TABLE\_SIZE}} - 1) * \texttt{MAX\_SIG}) + 1} \\ &= \frac{1}{1 + \exp(-(\frac{2*\texttt{MAX\_SIG}}{\texttt{SIG\_TABLE\_SIZE}} * i - \texttt{MAX\_SIG}))}\end{aligned} \tag{4.12}$$

Then,

$$\texttt{SIG\_TABLE}[((f + \texttt{MAX\_SIG}) * \frac{\texttt{SIG\_TABLE\_SIZE}}{\texttt{MAX\_SIG} * 2})]$$

$$
\begin{aligned}
&= \frac{1}{1+\exp(-(\frac{2*\texttt{MAX\_SIG}}{\texttt{SIG\_TABLE\_SIZE}} * ((f+\texttt{MAX\_SIG}) * \frac{\texttt{SIG\_TABLE\_SIZE}}{\texttt{MAX\_SIG}*2}) - \texttt{MAX\_SIG}))} \\
&= \frac{1}{1+\exp(-((f+\texttt{MAX\_SIG}) - \texttt{MAX\_SIG}))} \\
&= \frac{1}{1+\exp(-f)} \\
&= \sigma(f)
\end{aligned}
\tag{4.13}
$$

Hence, this method correctly computes the value of the sigmoid function.

Shuffling tweets

As that the learning rate decreases linearly throughout training, the network learns the most from examples that appear early in the training corpus. Therefore, shuffling the order of the tweets causes huge perturbations on the evolution of the cost function of the network, thereby allowing the model to explore different regions of the cost surface throughout several iterations of training. When training the same model several times to obtain the performance bound, the examples were shuffled by random for each training session. This leads to a more robust evaluation of the model's performance independently of the order of samples.

Initialising embeddings

Mikolov's implementation of Skip-gram initialises each element of the general embedding matrix with a random draw from a uniform distribution $[-0.5, 0.5]$. This is the only source of randomness in the original C implementation. However, I chose to initialise the general embeddings with an external embedding matrix pre-learnt from a standard corpus such as Wikipedia. This will potentially lead to performance improvements because the model now has substantial knowledge about facts from Wikipedia. For example, it can now learn more nuanced changes in meaning of 'Mercedes' in different contexts as it already knows that Mercedes is a carmaker.

As such, I trained the standard Skip-gram model on the first billion characters from Wikipedia[2]. Then, I initialised the general embeddings of words in my model with the embeddings learnt from Wikipedia. The words in my model that do not occur in Wikipedia were simply initialised as before: uniformly distributed as $[-0.5, 0.5]$.

4.4 Measuring brand perception

Word association in the vector space model is commonly measured with *cosine similarity* between the two word vectors. Once the context-dependent model is trained, its word embeddings change for different contexts. I propose to measure brand perception as the cosine similarity between word embeddings that represent product category and brand names. For example, to see which of *smirnov, absolut* and *svedka* is most associated with *vodka*, I compute the following:

$$sim(x, vodka),\ x \in \{smirnov, absolut, svedka\}.$$

$sim(a, b)$ is the cosine similarity between $vec(a)$ and $vec(b)$. The item with the greatest cosine similarity with *vodka* is the one most associated with vodka according to the model's learnt embeddings.

On the other hand, to measure perception with contexts, I can instead compute:

$$sim_x(alcohol, beer),\ x \in \{male, female\},$$

[2]http://mattmahoney.net/dc/enwik9.zip

where $sim_c(a, b)$ denotes the cosine similarity between $vec(a)$ and $vec(b)$ in the context of c. Computing this first involves adding context embeddings for c to the main embeddings of a and b, *i.e.*

$$sim_c(a, b) = sim(vec_{main}(a) + vec_c(a), vec_{main}(b) + vec_c(b)).$$

Beer can be said to be more associated with alcohol for one gender if the similarity between $vec(beer)$ and $vec(alcohol)$ is greater in the context of that gender.

Chapter 5

Evaluation

5.1 Evaluation setting

This chapter evaluates the effectiveness of the context-dependent skip-gram model by comparatively assessing the quality of context embeddings against those learnt by a model insensitive to user contexts. Then, it presents notable observations of context embeddings' ability to capture various contextual variations of word meanings.

Models

In quantative evaluation schemes (see Section 5.2), three models were evaluated against each other:

- SGLM: Mikolov's vanilla Skip-gram model augmented with context embeddings.
- CDSG: SGLM model with context-specific learning rates.
- CDSGD: CDSG model with 10% of its hidden units randomly omitted with dropout. 10% was found to give best performance.

Parameter set

The list of model-generic parameters and the values used in practice are given below:

- vector_size : the dimension of the learned vector representations, chosen to be 200.
- ini_alpha : initial learning rate (both general and context-specific), chosen to be 0.1.
- min_alpha : minimum learning rate (both general and context-specific), chosen to be 0.0001.
- window : the width of the context-window from which to perform word prediction, chosen to be 5. This implies that all contexts within five words to the left and the right were used to predict the current word.
- min_count : the threshold for which words appearing in corpus with a smaller frequency are ommitted, chosen as 50.

5.2 Quantitative evaluation

As a quantitative measure of the model's ability to learn context-sensitive representations of language, I constructed three different evaluation schemes, which each test if the model captures different types of metadata well.

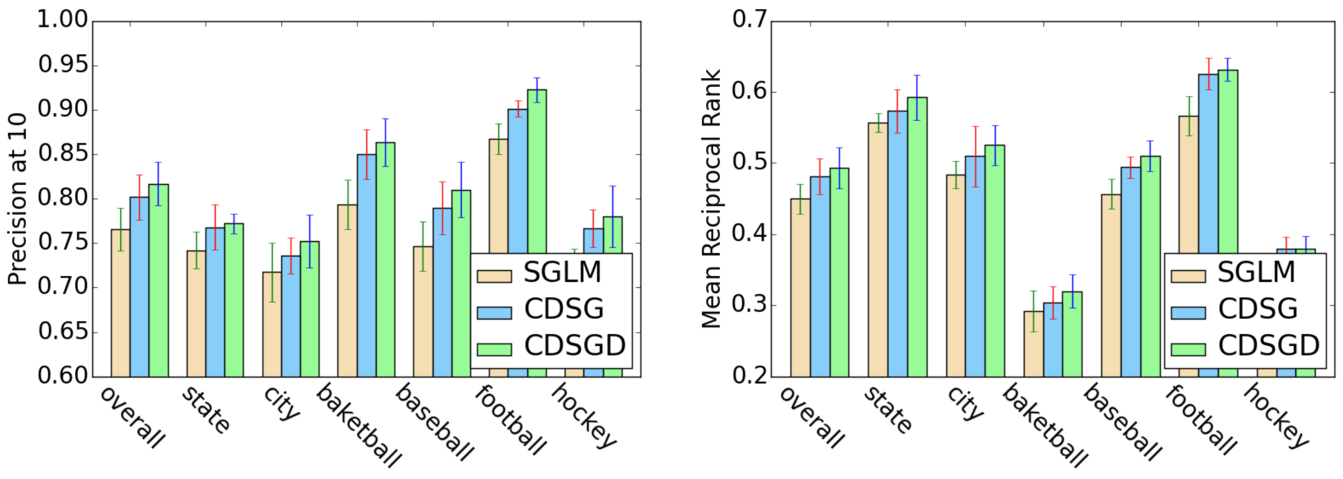
5.2.1 Location

Dataset and methodology

To evaluate the accuracy of geographic embeddings, I adopted the dataset used by Bamman *et al.* to evaluate distributed representations to learn geographically situated language [22]. The authors measure semantic similarity of pairs of words whose meaning has strong geographic correlations and report the mean similarity of geographically-related pairs of words. For example, in U.S., California, we expect the word *state* to be associated most with *california* than any other American state, such as *texas*. I take the rank of *california* from all states based on the similarity to vector *state* in the context of *california*, and proceed with other states analogously. Six categories of words with strong geographic connotation were constructed, as shown below. For each category, an example of the list of similarities to be computed is shown for each sports team/state, with the target similarity (the one we expect to rank highest) displayed in bold.

1. **state**: for each state, I compute the similarity between the word *state* and every US state, in the context of that state;
   e.g., $\mathbf{sim_{CA}(state, california)}, sim_{CA}(state, texas), sim_{CA}(state, arizona),$ $\cdots$ for the state California.
2. **city**: for each state, I compute thet similarity between the word *city* and the most populous city of each state, in the context of that state;
   e.g., $\mathbf{sim_{MA}(city, boston)}, sim_{MA}(city, newyork), sim_{MA}(city, houston),$ $\cdots$ for the state Massachusetts.
3. **basketball**: for each NBA team, I compute the similarity between $(vec(basketball) + vec(team))$ and every NBA team name, in the context of the state the team is based in;
   e.g. $\mathbf{sim_{IL}(basketball + team, bulls)}, sim_{IL}(basketball{+}team, knicks),$ $sim_{IL}(basketball + team, celtics), \cdots$ for the team Chicago Bulls
4. **baseball**: for each MLB team, I compute the similarity between $(vec(baseball) + vec(team))$ and every MLB team name, in the context of the state the team is based in;
   e.g. $\mathbf{sim_{MA}(baseball + team, red\ sox)}, sim_{MA}(baseball{+}team, yankees),$ $sim_{MA}(baseball + team, white\ sox), \cdots$ for the team Boston Red Sox.

5. **football**: for each NFL team, I compute the similarity between $(vec(football) + vec(team))$ and every NFL team name, in the context of the state the team is based in;
   e.g. $\mathbf{sim_{TX}(football + team, cowboys)}$, $sim_{TX}(football{+}team, giants)$, $sim_{TX}(football + team, bears), \cdots$ for the team Dallas Cowboys.
6. **hockey**: for each NHL team, I compute the similarity between $(vec(hockey) + vec(team))$ and every NHL team name, in the context of the state the team is based in;
   e.g. $\mathbf{sim_{NJ}(hockey + team, devils)}$, $sim_{NJ}(hockey{+}team, rangers)$, $sim_{NJ}(hockey + team, islanders), \cdots$ for the team New Jersey Devils.



(a) Precision at 10 for each category

(b) Mean reciprocal rank for each category.

Figure 5.1: Location evaluation results.

Three models (SGLM, CDSG, CDSGS) were each run five times with the same parameter set and the training data shuffled. In Figure 5.1, I report the average precision at 10 and mean reciprocal rank for each category, with standard deviation for each category as an error bar. I explain both evaluation metrics below.

- Precision at 10 : In California, we expect *california* (current word) to be more similar to *state* (target word) than any other US state names (competitor words). I report the ratio of current words that rank within the top 10 of all competitor words, based on the similarity between the target word. The precision at 10 of 0.8 implies that the expected word appears in the first 10 ranked by similarity, empirically with probability 0.8.

- Mean reciprocal rank : In Massachusetts, we expect *red sox* to be more similar to *baseball+team* then other MLB team names. I report the mean of the inverse of the rank of the current word (*red sox*) amongst all competitor words. An average MRR of 0.5 implies that the expected word appears in the first 2 names returned.

Results

Figure 5.1 presents the result of location evaluation with two metrics, with 95% confidence interval shown as error bars. It is clear that context-dependent models learn more accurate geographic embeddings than the vanilla skip-gram model in general. Specifically, CDSG with dropout attained 81.6% precision at 10 on average, while the context-independent model gave 76.6%. The difference between the two models is over 4% for mean reciprocal rank. This demonstrates that a model with a separate learning rate for each context is able to achieve local optimum in the cost space more often and obtain better quality context embeddings.

As a concrete example, after running the CDSG model with dropout, the word *texas* was the closest word in the vocabulary of 1,521,103 words to the word *state* in the context of Texas, with similarity 0.770. For the CDSG model, however, the cosine similarity was 0.596 (rank #10). For the skip-gram model, *texas* was the 18th most similar word to *state*, with similarity 0.512.

5.2.2 Gender

Dataset and methodology

As a quantative measure of the model's performance with gender-specific word representations, I compiled an evaluation dataset of 24 words, 12 for each gender, that are shown to be particularly better known to members of that gender than the other (Brysbaert, 2014).[1]

Table 5.1 shows top 12 recognisable words for each gender in column 'Word1'. The column 'Rate' shows the recognition rates: (a, b) signifies that a percentage of men could correctly recognise the meaning of the word, while b percentage of women could do the task. Then, column 'Word2' gives a single word that is a synonym of or a generally associated word with 'Word1' according to Oxford Dictionaries.[2]

Male Words			Female Words		
Word1	Rate	Word2	Word1	Rate	Word2
codec	(88, 48)	data	taffeta	(48, 87)	silk
solenoid	(87, 54)	coil	tresses	(61, 93)	hair
golem	(89, 56)	robot	bottlebrush	(58, 89)	brush
mach	(93, 63)	speed	flouncy	(55, 86)	decorative
humvee	(88, 58)	vehicle	mascarpone	(60, 90)	cheese
claymore	(87, 58)	sword	decoupage	(56, 86)	paper
scimitar	(86, 58)	sword	progesterone	(63, 92)	hormone
kevlar	(93, 65)	fiber	wisteria	(61, 89)	pea
paladin	(93, 66)	warrior	taupe	(66, 93)	grey
bolshevism	(85, 60)	communism	flouncing	(67, 94)	angry
biped	(86, 61)	two-footed	peony	(70, 96)	plant
dreadnought	(90, 66)	battleship	bodice	(71, 96)	dress

Table 5.1: Gender evaluation dataset.

For each row in the table, I computed $\sigma_m = sim_{male}(vec(word1), vec(word2))$

[1]`http://gnodevel.ugent.be/crr.ugent.be/archives/1628`
[2]`http://www.oxforddictionaries.com/`

and $\sigma_f = sim_{female}(vec(word1), vec(word2))$. Then, I computed the Spearman's rank correlation of the difference between male rates and female rates and $\sigma_m - \sigma_f$ for each row. Spearman's rank correlation, also known as Spearman's ρ, is a non-parametric metric of statistical dependence between two variables. It shows how well the relationship between the two variables can be represented using a monotonic function (not necessarily linear). A perfect Spearman correlation of +1 or -1 is achieved when the two variables are in a perfectly monotonic relationship, or its inverse. The intuition here is that for words with large recognition rate gap between genders, the similarity between 'Word1' and 'Word2' should also greatly differ between two genders.

For a sample size n, raw scores X_i, Y_i are converted to ranks x_i, y_i and ρ is computed as:

$$\rho = 1 - 6\frac{\sum {d_i}^2}{n(n^2-1)},$$

where $d_i = x_i - y_i$, *i.e.* the difference between ranks. To report the overall correlation from five independent trials, I first converted each ρ_i using Fisher's z-transformation:

$$z_i = \frac{1}{2}\ln(\frac{1+\rho_i}{1-\rho_i}) = \operatorname{arctanh}(\rho_i).$$

Then, I averaged z_i as $\bar{z} = \sum_{i=1}^{5} z_i/5$ and backtranform the averaged $\bar{z}$ into the ρ space:

$$\bar{\rho} = \frac{\exp(2\bar{z})-1}{\exp(2\bar{z})+1} = \tanh(\bar{z}).$$

The bottommost row in Table 5.2 shows the overall correlation $\bar{\rho}$ over five trials derived as above.

As a concrete implementation of Spearman's rho, I used the `spearmanr` function of package `scipy.stats`. The function returns the correlation coefficient and p-value as output, where the p-value describes the chance that random sampling would result in the correlation coefficient obtained when there is no correlation between two random variables.

As in location evaluation, three models (SGLM, CDSG, CDSGD) were trained five times each with the same set of parameters. Table 5.2 presents the correlation coefficient and p-value for each training trial as well as the overall correlation for each model. Rows with 'Word1' nonexistent in the vocabulary were dropped from computation.

Results

	SGLM		CDSG		CDSGD	
	Corr	P-val	Corr	P-val	Corr	P-val
Trial 1	0.405	6.842E-3	0.463	3.451E-2	0.575	6.139E-4
Trial 2	0.562	8.020E-3	0.539	1.166E-3	0.593	3.778E-4
Trial 3	0.415	6.141E-3	0.589	4.939E-3	0.603	3.815E-4
Trial 4	0.598	4.215E-2	0.622	2.584E-3	0.734	1.537E-4
Trial 5	0.609	3.403E-4	0.749	9.265E-5	0.514	1.702E-3
Overall	0.523		0.602		0.609	

Table 5.2: Gender evaluation results.

As Table 5.2 shows, the context sensitive models perform noticeably better than the skip-gram model, a clear indication that the gender-specific embeddings learnt by the last two models are of arguably better quality than those for the context-independent model. As a concrete example, the similarity between *scimitar* and *word* for male was 0.250 for male and 0.087 for female under the CDSGD model. However, for the skip-gram model, the similarities were 0.211 for male and 0.213 for female. Although the number of test cases is rather small (24), each p-value of individual correlation coefficient is reasonably small (<4%) to reject the null hypothesis that the purported correlation does not exist.

5.2.3 General Semantic Similarity

Dataset and methodology

In addition to context-specific embeddings, the accuracy of general word embeddings was tested with standard word similarity tasks, using corpora WordSimilarity-353 and SimLex-999. WordSimiliarity-353 dataset contains 353 word pairs tagged with the average of their similiarity scores assigned by 13-16 subjects [23]. The instruction to subjects was to measure the *relatedness* of the pair of words on a scale from 0 (totally unrelated words) to 10 (very similar words or synonyms). SimLex-999 is a more recent dataset that provides a gold standard for word *similiarity* rather than relation or association. Hill *et al.* demonstrate that SimLex-999 is challenging for computational language models to capture as they should learn word similarity independently of relatedness. Most language representation learning models infer relationship between words from their co-occurrences, which primarily reflects relatedness not similarity [24].

	SimLex-999	WordSim-353
coast-shore	9.00	9.10
clothes-closet	1.96	8.00

Table 5.3: Comparison of WordSim-353 and SimLex-999 evaluation datasets.

Results

The correlation coefficients obtained confirm that SimLex-999 is indeed more difficult for computational models to capture than WordSim-353. However, no one model was shown to learn better quality general word embeddings than the other two. On WordSim-353, CDSGD model gave the biggest correlation with the gold-standard, despite with marginal difference with the other models. On SimLex-999, however, it gave the worst performance, again with minor differences.

		SGLM		CDSG		CDSGD	
		Corr	P-val	Corr	P-val	Corr	P-val
WordSim-353	Trial 1	0.510	1.935E-23	0.439	3.775E-17	0.491	1.230E-21
	Trial 2	0.466	2.143E-19	0.447	9.341E-18	0.481	1.142E-20
	Trial 3	0.460	8.149E-19	0.474	4.483E-20	0.497	3.369E-22
	Trial 4	0.516	4.896E-24	0.452	3.599E-18	0.488	2.510E-21
	Trial 5	0.440	3.540E-17	0.465	2.702E-19	0.490	1.514E-21
	Overall	0.479		0.456		0.490	
SimLex-999	Trial 1	0.232	1.213E-13	0.178	1.477E-08	0.214	8.055E-12
	Trial 2	0.209	2.454E-11	0.216	6.231E-12	0.223	1.047E-12
	Trial 3	0.187	2.652E-09	0.232	1.083E-13	0.194	6.943E-10
	Trial 4	0.217	4.676E-12	0.200	1.940E-10	0.186	3.027E-09
	Trial 5	0.212	1.406E-11	0.195	5.874E-10	0.198	3.219E-10
	Overall	0.212		0.204		0.203	

Table 5.4: General semantic similarity results.

In terms of how models are designed, there is no reason to believe that one model should learn better general word representations than any other, as all three models use the same training algorithm to backpropagate errors to update the general embeddings. As a result, I conclude there is no clear winner for the two general word semantic similarity tasks.

5.3 Qualitative evaluation

Table 5.5 presents the closest terms to *cheers* in both UK and US after running the CDSGD model. It clearly demonstrates that the word cheers strongly evokes the sense of drinking in the US, as its close terms are either types of alcohol or related to the act of drinking. In the UK, however, the meaning of cheers seems to be more diverse, ranging from a parting wish (*e.g. xx*) or an expression of gratitude to someone close (co-occurs with *e.g. name, mate, friends, brother*).

Table 5.6 shows an interesting variation of the meaning of *work* based on

US		UK	
term	cosine	term	cosine
cheers	1.000	cheers	1.000
#beer	0.538	name	0.584
#craftbeer	0.515	well	0.573
kolsch	0.499	xx	0.568
brewery	0.497	pal	0.547
#happyhour	0.487	lads	0.546
ipa	0.448	mate	0.525
gose	0.445	friends	0.520
root	0.445	#beer	0.520
fest	0.443	brother	0.513

Table 5.5: Terms with the biggest cosine similarity to *cheers* for people in US/UK

speaker's age. The list of close terms for people aged 18 or less exhibits a strong correlation to studying, homework and school. For the next age group, the meaning gradually shifts to professional work, mirrored by terms *pay* and *money*. This becomes more pronounced in the last age group, for which the amount of training data is relatively small, a possible explanation for the small similarity values.

Given that a language model computes the sense of 'similarity' based on the co-occurrence of two words, some results are not directly similar to work but related, for example *hand* as in a student handing in work. The results are interesting because they not only reveal similar terms for each age group but also capture words that portray and mark each period of one's life.

5.4 Brand perception analysis

As a quantative analysis of brand perception of users, I used the results of a survey to measure global public brand perception in 2014 from over 2.5 mil-

< 18		$18 - 25$		> 25	
term	cosine	term	cosine	term	cosine
work	1.000	work	1.000	work	1.000
hard	0.722	stuff	0.791	make	0.457
class	0.719	home	0.779	car	0.435
probably	0.697	money	0.763	home	0.428
stuff	0.689	drive	0.754	pay	0.396
friends	0.688	life	0.741	budget	0.391
school	0.652	class	0.737	drive	0.388
hand	0.651	drinks	0.736	money	0.383
college	0.642	sleep	0.730	commute	0.333
questions	0.641	pay	0.724	kids	0.332

Table 5.6: Terms with the biggest cosine similarity to *work* for people in each age group.

lion interviews, known as Annual Global Brand Rankings from BrandIndex.[3] The results are in the form of rankings of companies for each country and product category (for US). In the survey, respondents were asked to evaluate companies based on 'buzz': whether if they have heard anything about the company in the last two weeks via advertisement, online marketing or word-of-mouth, and how positive/negative it is. The 'buzz score' is hence calculated for each company, ranging from -100 to 100 where 0 implies neutral impression and *e.g.* +40 signifies that 40% more people heard positive stories about the brand.

Although the results covers 15 countries, it only provides product category-specific rankings for US, hence only the results for US were used. As Table 5.7 shows, five brands are ranked for each product catgory based on buzz score.

I take this BrandIndex ranking as a 'gold standard' and compare it with the ranking derived by my model. For each product category, I compute the

[3] `http://www.brandindex.com/ranking/2014-annual`

[4] `http://www.brandindex.com/ranking/us/2014-annual/category/internet-search`

Rank	Brand	Score
1	Google	22.6
2	Yahoo!	10.3
3	Bing	5.8
4	Ask.com	2.8
5	Yahoo! Answers	2.8

Table 5.7: 2014 BrandIndex Annual Ranking for Internet Search.[4]

similarity between the brand names and the category name in the context of the US and rank five companies. Computing the Spearman's rank correlation between this ranking and the gold standard gave a correlation coefficient and p-value for each category, as shown in Table 5.8. Table **??** gives the full ranking of companies computed by the CDSGD model.

Category	Corr	P-val	Category	Corr	P-val	Category	Corr	P-val
Airlines	-0.3	0.624	Hotel	-0.4	0.505	Oil & Gas	0.7	0.188
Amusement Parks	0.5	0.391	Insurance	0.1	0.873	Restaurants	0.7	0.188
Footwear	0.5	0.391	Search	0.5	0.391	Fast Food	0.9	0.037
Automobile	0.6	0.285	Social Media	0.5	0.391	Clothing	0.9	0.037
Cable	0.7	0.188	Investment	-0.2	0.747	Discount	0.4	0.505
Car Rental	0.3	0.624	Networks	0.2	0.747	Travel Agents	0.3	0.624
Overall	0.455							

Table 5.8: Correlation coefficients and p-values for each category.

Overall correlation coefficient was again computed via averaging the z-transformed coefficients of 18 categories and backtransforming the result. Having only five items to rank against inevitably entails high p-values as there is a greater chance that the correlation was obtained as a result of random sampling. However, general correlation of 0.455 across 18 different categories indicates the model learns consumers' perception of brand to a reasonable extent.

Chapter 6

Conclusions and Future Work

6.1 Summary

This thesis provides an efficient and accurate method to extract user perception of brands via learning a neural network language model on a corpus mined from Twitter. I used a state-of-the-art neural language model known as the Skip-gram model and extended the network architecture to learn separate word representations for different contexts of users. The key contribution of this work is introducing separate learning rates for each context. This enables the network to learn each context embedding at its own pace, *e.g.* learning more from each occurrence of rare contexts than those of common contexts. As a result, the context-dependent skip-gram model compares favourably to Mikolov's skip-gram on several context-sensitive word similarity tasks.

From the brand analytics perspective, this work provides a novel method of analysing user sentiment of brands manifested by their online microblogs. This computational approach involves no questionnaires or surveys, which traditional methods to learning brand perception often require. As the entire data collection process is performed entirely automatically from Twitter, it has an additional advantage that mining new data from Twitter and updating the model with the recent user content is very efficient and straightforward.

This thesis comprises two main equally important portions. The data part includes mining a sizable corpus from Twitter and cleansing the dataset of noise inherent to online writing. Correctly inferring useful user metadata from available information was key to ensuring that model has enough training data to learn from. The next part trains the language model to learn word embeddings sensitive to user contexts. Both parts had to proceed hand-in-hand to ensure quick prototypes to be evaluated and improved throughout the development process.

In a quantative evaluation on the task of judging similarity of pairs of words with strong geographic and gender correlation, the proposed context-dependent language model outperforms the standard skip gram model that does not leverage situational information in the training data. The model's validity is also confirmed with a qualitative task of retrieving words in the vocabulary most similar to a certain word based on context-informed measure of similarity.

As a preliminary analysis of brand perception, I took a published ranking of brands for each market sector based on consumer perception in the US. I reranked the companies for each category by similarity with the category name itself and the best performing CDSGD model gave a ranking correlation of 0.455. The model can be used to reveal brand perception for more specific demographics in a straightforward manner.

6.2 Future work

Accommodating multiwords/phrases

Perhaps the biggest shortcoming of all three models considered is inability to represent multiword expressions and idiomatic phrases. For example, the meaning of 'San Francisco' is not adequately captured by a combination of vectors 'San' and 'Francisco'. An easy workaround to battle this problem is suggested by Mikolov *et al.* [25]. The authors identified salient phrases with a data-driven approach and simply treated them as a single word while preprocessing the corpus. The measure used to determine the saliency of a phrase is the following:

$$score(w_i, w_j) = \frac{count(w_i w_j) - \delta}{count(w_i) \times count(w_j)},$$

where δ is the discounting factor used to prevent phrases made up of very infrequent words to be formed. All bigrams of score greater than a predefined threshold are treated as unigrams. This process is run for a few iterations with progressively smaller threshold to identify phrases with more than two words.

Given that many company names are multiwords and it is impractical to manually identy them in the corpus, using this method is likely to increase the model's coverage of companies, organisations and names of people.

Battling polysemy

Distributional semantics inherently struggles with representing polysemic words; as every word is mapped to a point vector, several senses of a polysemic word need to be collapsed into a single representation. For example, learning the representation *vec(subway)* should be a compromise between at least two distinct senses: *vec(metro)* and *vec(sandwich)*, leading to sub-optimal representations. Applying Vilnis' approach towards distributional semantics, one should be able to better represent polysemic words as every word is now mapped to a Gaussian distribution over a latent embedding space, rather than a single point in vector space [16].

Integrating context embeddings

In this thesis, I take a summation of main and context embeddings to derive a representation of a word in specific contexts. There is a long line of research in combining word embeddings to obtain a composite meaning representation. The most notable work was conducted by Mitchell and Lapata, where the authors considered several composition models including additive, multiplicative, weighted sum and pointwise multiplicative models [26].

Alternatively, having a separate embedding matrix for every context combination might lead to performance improvements. Instead of computing $vec_{main}(word) + vec_{male}(word) + vec_{us}(word)$, we can learn a separate representation for the specific context combination $vec_{male+us}(word)$. With more training data, the model should be able to learn an accurate vector for each context combination, despite potential explosion of memory requirements for training the network.

User embedding

This thesis shows the usefulness of having context-dependent word representations, with which we shift point vectors in space to reflect change in meaning of words based on contexts in which they are uttered. As a speaker can be characterised by a number of contexts, an embedding matrix tailored for each person can be easily derived from appropriately combining main and context embedding matrices. Having a user-specific embedding matrix enables linguistic similarities between different users to be inferred.

Bibliography

[1] James Makens. Effect of brand preference upon consumers perceived taste of turkey meat. *Journal of Applied Psychology*, 49, 1965.

[2] Johan Almenberg, Anna Dreber, and Robin Goldstein. Hide the label, hide the difference? *American Association of Wine Economist*, 165, 2014.

[3] Antonia Mantonakis, Bryan Galiffi, Ummugulsum Aysan, and Randi Beckett. The effects of the metacognitive cue of fluency on evaluations about taste perception. *Psychology*, 4, 2013.

[4] Alfred Kuehn. Consumer brand choice–a learning process? In *Mathematical Models in Marketing*, volume 132 of *Lecture Notes in Economics and Mathematical Systems*. 1976.

[5] Sunil Gupta. Impact of sales promotions on when, what, and how much to buy. *Journal of Marketing Research*, 25, 1988.

[6] Michael Trusov, William Rand, and Yogesh V. Joshi. Improving prelaunch diffusion forecasts: Using synthetic networks as simulated priors. *Journal of Marketing Research*, 50, 2013.

[7] Seshadri Tirunillai and Gerard J. Tellis. Mining marketing meaning from online chatter: Strategic brand analysis of big data using latent dirichlet allocation. *Journal of Marketing Research*, 51, 2014.

[8] Seshadri Tirunillai and Gerard J. Tellis. Does chatter really matter? dynamics of user-generated content and stock performance. *Marketing Science*, 31, 2012.

[9] Matt Thomas, Bo Pang, and Lillian Lee. Get out the vote: Determining support or opposition from Congressional floor-debate transcripts. In *Proceedings of EMNLP*, pages 327–335, 2006.

[10] Vikas Ashok, Song Feng, and Yejin Choi. Success with Style: Using Writing Style to Predict the Success of Novels. In *Empirical Methods in Natural Language Processing (EMNLP 2013)*, 2013.

[11] Zellig Harris. Distributional structure. *Word*, 10, 1954.

[12] Yoshua Bengio, Rjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[13] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. 1988.

[14] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, 2013.

[15] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. *CoRR*, 2014.

[16] Luke Vilnis and Andrew McCallum. Word representations via gaussian embedding. *CoRR*, abs/1412.6623, 2014.

[17] Yi Yang and Jacob Eisenstein. A log-linear model for unsupervised text normalization, 2013.

[18] Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *AISTATS?05*, pages 246–252, 2005.

[19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[20] David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, Sept 1952.

[21] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[22] David Bamman, Chris Dyer, and Noah A. Smith. Distributed representations of geographically situated language. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2014.

[23] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing search in context: The concept revisited. *ACM Trans. Inf. Syst.*, 20(1), 2002.

[24] Felix Hill, Roi Reichart, and Anna Korhonen. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *CoRR*, abs/1408.3456, 2014.

[25] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119. 2013.

[26] Jeff Mitchell and Mirella Lapata. Composition in distributional models of semantics. *Cognitive Science*, 34(8):1388–1429, 2010.