UNIVERSITAT DE BARCELONA

FUNDAMENTAL PRINCIPLES OF DATA SCIENCE MASTER'S THESIS

Identification of Negative Keywords in Search Marketing with Embedding Layers and Neural Networks

---

Author:
David LORIS

Supervisor:
Jordi VITRIA

A thesis submitted in partial fulfillment of the requirements for the degree of MSc in Fundamental Principles of Data Science

in the

Facultat de Matemàtiques i Informàtica

January 18, 2021

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc

Identification of Negative Keywords in Search Marketing with Embedding Layers and Neural Networks

by David LORIS

In this paper we introduce a model to help marketing specialists within the field of Search Advertising to limit spend on Google searches which have a low probability of leading to a revenue generating event. This is a topic which has not been widely addressed in scientific literature. For this study we obtained data from a company which spends a large amount on Google Ads, but relies on a subjective and time consuming approach to this problem. Our proposed model uses GloVe's pre-trained Embedding Layers and Neural Networks to speed up and improve accuracy of this process.

Acknowledgements

Special thanks to my adviser Jordi Vitria and to all the professors of the Masters. Studying Data Science at the University of Barcelona has been a wonderful experience and I am truly grateful for everything I have learned.

Chapter 1

Introduction

1.1 Objective

In this paper we focus on an important aspect of Search Marketing, namely the identification of negative keywords. This involves the identification of Searches performed by users on Google which have a low probability of generating revenue for an advertiser. Currently this is a manual process for most advertisers which has two major drawbacks:

- it is a **time consuming** process.
- it is often **subjective**.

Due to the fact that this is a boring and time consuming task it is often a neglected aspect of Search Marketing, which leads to wasted advertising spend. Our hypothesis is that using Embedding Layers and Deep Neural Networks we can can develop a classifier to help Marketeers to either automate or speed up this time consuming process. We hope to achieve a satisfactory level of accuracy of approximately 80%. Solving this problem would lead to significant time and cost savings to the advertiser. To build our model we have obtained a data set from a company (which has asked not to be cited). The data set allows us to see the the searches made by users on Google which resulted in a click on and advertisement. Subsequent to the click we are able to see if the user performed an action which resulted in the generation of Revenue.

1.2 Similar Work in Scientific Literature

After extensive research we were not able to find any Scientific papers addressing this topic. In general there seems to be a limited amount of research carried out in this field. We can speculate on the reasons:

- Companies tend not to share their data with researchers.

- Companies tackle these problems in-house and do not share their work publicly for fear of the competition adopting it. So researchers are not exposed to the need.
- The problem may not be easily solvable so researchers do not approach the topic.

Regardless of the reasons, it is our belief that we have identified an interesting topic which merits more in depth study. Fortunately, NLP techniques applied in other contexts such as spam detection lend themselves well to the identification of negative keywords.

1.3 The Analogy with Spam Detection

Prior to diving into the problem in depth it is useful to consider the analogy between spam detection and negative keyword identification. Models which detect spam are concerned with the identification of emails which are considered to be undesirable to the reader. Meanwhile, negative keyword identification seeks to identify *Search Terms* which are undesirable to an advertiser. In both cases model training is generally supervised, though finding patterns within documents or user searches can involve some unsupervised learning in order to mold unstructured data into a format conducive to classification analysis.

1.4 The Company providing the Data

The Company which has provided the data has requested not to be cited. However, we have permission to use the data for modeling purposes and to provide the context, at a higher level, on how revenue is generated on their websites. The Company operates comparison websites in the Health Insurance sector in the USA. They have several websites, where users can view a comparison of Health Insurance options available from third party Health Insurance providers. The company generates revenue when a user clicks on one of these third party insurance options and is redirected to their website.

The company's main objective is to generate revenue which exceeds the spend on marketing. The marketing team spends a significant amount on Search Engine Marketing (over 10 million US Dollars a year on Google and Bing) to drive traffic to their websites. An algorithm for optimization of marketing spend has been developed by an in-house Data Science team, but no solution has yet been developed for the identification of negative keywords.

1.5 Introduction to Search Marketing

Unless you have been living in cave for the last quarter of a century you will have heard of a Search Engine. Search Engines have become an every day tool that people simply cannot live without. A free service offered online, users type in a *Search Term* and receive a response with a list of web pages listed in order of relevance. Google is by far the most popular Search Engine with a very high market share. Founded in 1998 By Larry Page and Sergey Brin, they revolutionized searching the web with the introduction of the Page Rank algorithm which outperformed competing search engines of the time thanks to its ability to evaluate a sites' authority and influence based on the number of other authoritative and influential sites that linked to it. Since then they grew to having this near monopolistic control of the world wide search traffic. The verb "to Google" was included in the Oxford English Dictionary in on June 15, 2006. While Google does not share official data on their traffic the estimates are astounding. While there is a lot of variance in the numbers, depending on the source, the annual number of searches is most likely to be more than a trillion searches per year.

In order to keep the service free, Google sell advertising space. There are usually 3 or 4 listings above the normal search results and a similar number at the bottom of the page. In order for advertisers to be displayed an advertiser needs to write an *Ad copy* which targets a *Keyword*. The *Ad copy* is what is shown on the page. Figure 1.1 below shows an example of a typical advertisement.

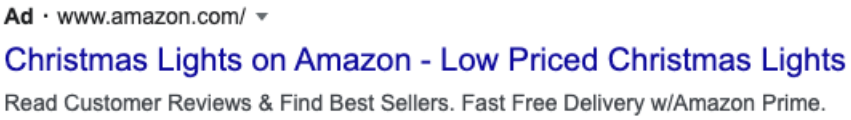

FIGURE 1.1: Example of ad copy seen by a user on Google

1.5.1 The Advertisement Auction

Prior to delving into the main background needed to understand the address the topic of this paper, we will briefly review the mechanism by which advertisements are placed on Google. While Google was founded in 1998, it was not until 2002 that Google introduced advertising based on a cost-per-click model (Bagshaw, 2015). Under this type of program, the advertiser only pays when a user actually clicks on an advertisement. This is in contrast to a cost-per-view (aka cost-per-impression) model where an advertiser is billed for every time a user views an ad. Initially Google partnered with Overture to have ads sourced on its platform, but in a later stage built its own advertising platform. When launching their own search advertising technology they also introduced the concept of ad relevance. This served to allow advertisers to compete on other criteria to price and to improve user experience. From available literature from Google data scientists and what information

is publicly available we can infer how the advertising mechanism works. It seems clear that Google uses a modified version of the Second Price Auction (Gagan Aggarwa, 2009).

Generalized Second Price Auction In this type of auction ever participant decides a *Bid*. The first place is assigned to the highest bidder, the second place to the next highest bidder and so on and so forth. Meanwhile, the price paid by the highest bidder corresponds to that of the bid of the second highest bidder. The second highest bidder then pays the amount bid by the third highest and so on (wikipedia, 2021).

Google Ad Rank Google's in-house advertising system was introduced in 2004 and is based on the *Ad Rank Algorithm*. The algorithm is a black box meaning that Google do not disclose almost anything about how it works. We can assume that, in parallel to many innovations introduced to Page Rank over the years, it has been updated over the years with state-of-the-art approaches. Many have tried to reverse engineer it without much success. What Google does divulge officially on Ad Rank is as follows (Google_Ads, 2021):

"A value that's used to determine your ad position (where ads are shown on a page relative to other ads) and whether your ads will show at all. Ad Rank is calculated using your bid amount, your auction-time ad quality (including expected click-through rate, ad relevance, and landing page experience), the Ad Rank thresholds, the competitiveness of an auction,the context of the person's search (for example, the person's location, device, time of search, the nature of the search terms, the other ads and search results that show on the page, and other user signals and attributes), and the expected impact of extensions and other ad formats."

1.5.2 A bit of Jargon

In order to help the reader quickly enter into the world of Search Marketing it is important to understand some definitions. These definitions are considered industry standard, but come from my experience working with the platform for over 15 years so as such specific references are not always provided. There may be slightly different definitions used depending on the practitioner. These discrepancies however are not influential to gaining an understanding of the field for the purpose of reading this paper.

Search Term A Search term is something a user types into a search engine, which generates a response with a list of websites and advertisements.

Keyword A keyword is a string which is associated to an advertisement by the advertiser. It allows the advertiser to target groups of search terms. Every keyword also has a *match type* which we will go into more detail in a dedicated sub-section.

Ad Rank Calculated for all eligible advertisements targeting a given search term. It ranks advertisements in order to decide the positioning in the results returned. While the precise formula is a closely guarded secret, we know that it is proportional to the *quality score* and to the *bid*, described below.

$$Ad\ Rank \propto Bid, Quality\ Score \tag{1.1}$$

Quality Score an indicator of the relevance of a search term to an advertisement. Its function is to encourage advertisers to create advertisement that are highly relevant to what a user is searching for. Quality Score is a very important component in the auction, but again, the exact value is not known to the advertiser. The higher the quality score, the more relevant the advertisement is deemed to be, thus the higher the ad rank. Quality score is a very interesting topic we could digress on for several pages, but, while interesting, is beyond the scope of this paper.

Bid the maximum amount an advertiser is willing to pay for a click. This is set at the keyword level, though can also be "modified" by device, user location, time of day and other factors. In this paper we are focusing our attention on a problem that does not require a deep understanding of the bidding process so we will not dwell on it further.

Impressions This is a statistic available in Google Ads reports which provides the number of times a Search Term triggered an advertisement. Impressions are proportional to Ad Rank thus having a *high quality score* and *bid* will lead to more impressions.

Clicks - This is a statistic for the number of times a user clicked on an advertisement. The number of clicks depends mainly on the number of impressions obtained and how high on the page the advertisement was shown for those impressions. The latter is decided by Ad Rank.

Click Through Rate (CTR) - The ratio of clicks to impressions, as shown in the following equation:

$$CTR = \frac{Clicks}{Impressions} \tag{1.2}$$

Cost the actual amount an advertiser will be billed for clicks generated.

Cost Per Click (CPC) the actual amount an advertiser will be billed for clicks generated divided by the number of clicks.

$$CTR = \frac{Cost}{Click} \tag{1.3}$$

Conversion - an event which occurs on the advertisers website that usually corresponds to the generation of revenue for the advertiser.

Conversion Rate - The number of conversions divided by the number of clicks.

$$CR = \frac{Conversions}{Clicks} \tag{1.4}$$

Cost Per Conversion The number of conversions divided by the total cost. Advertisers will typically have a cost per conversion target which represents the maximum amount they are willing to pay for a revenue generating event.

1.6 The Search Terms Report

Google provides a report which allows advertisers to analyze what Search Terms are triggering an advertisement. In table 1.1 we see an example of a typical Search Term report including only the statistics we are interested in. Using this data we are able understand how search terms perform.

TABLE 1.1: Example of a Search Term Report

Search Term	Clicks	Conversions
health benefits plans	1912	25
does nebraska have obamacare	1230	10
florida blue obamacare	45	0
healthcare texas	10	0

As part of our data cleaning and structuring we will process the search term report to create simplified report necessary as an input to a classifier. This will be described in more detail in the section on the model, but we can anticipate now the input to have the format as shown in table.

TABLE 1.2: Example of a cleaned Search Term Report

Search Term	Success
health benefits plans	1
does nebraska have obamacare	1
florida blue obamacare	0
healthcare texas	0

1.7 Negative Keywords

The way in which advertisers can prevent their ads from showing for a given Search Term is via the use of *Negative Keywords*. There are three types frequently used (Ad_Espresso, 2019). These are:

Exact Match Negative When this is specified the advertiser can block the advertisement from showing for this specific Search Term.

Phrase Match Negative When this is specified the advertiser can block the advertisement when the negative keyword is contained within the Search term.

Broad Match Negative When this is specified the advertiser can block the advertisement from showing on all Search Terms Google deems to be similar to the negative keyword. In practice this is rarely used due to the lack of visibility to the advertiser on what is actually being blocked.

In figure 1.2 we can see an example of how these negative keywords work to block traffic (Sparling, 2015).

Negative Match Type	Explanation	Example	Blocked Searches	Allowed Searches
Negative Exact	Ad will not show only if search query exactly matches the negative keyword.	[herbal cleanse]	herbal cleanse	herbal cleanses, herbal cleansing, buy herbal cleanse
Negative Phrase	Ad will not show if query contains this exact keyword, even with other terms before or after it.	"herbal cleanse"	herbal cleanse, buy herbal cleanse	herbal cleanses, cleanse herbal, herbal cleansing
Negative Broad	Ad will not show when a query contains these terms in any order.	herbal cleanse	herbal cleanse, cleanse herbal, best herbal cleanse	herbal cleanses, herbal cleansing

FIGURE 1.2: Example of ad copy seen by a user on Google

In this paper we will focus our attentions on identifying Search Terms to be implemented as Negative Exact. Potential follow up work could involve grouping these to find a more succinct lists to be added in Negative Phrase. This could be useful since the number of negative keywords an advertiser can implement is limited.

1.8 Our Chosen this Approach

Our objective is to build a classifier to predict whether a Search Term will lead to a conversion or not. Thus it is a classical success-failure classification model. We evaluated some other approaches prior to deciding to use Embedding Layers and Neural Networks. The best viable alternative is a more traditional approach using one-hot encoding and a tree based algorithms or SVM. While we did not actually build this alternative model for comparison of results due to time constraints, the choice of the approach with Embedding Layers and Neural Networks we believe will be superior approach since it ensures that:

- number of dimensions remains low compared to one-hot encoding.
- words that are similar can share information more effectively.
- we can take advantage of pre-trained embeddings to improve the model.

1.9 Determining Success

The objective of the modelling exercise is to achieve at least 80% accuracy prediction. This is considered to be sufficient level of accuracy for in order for the company to test this model in a real world setting.

1.10 The Current Process

Currently there is no algorithm in place to identify negative keywords. It is a manual process where the *Google Ads Specialist*, responsible for managing the account, will download a report with some filters in place. A common filter used would be Search Terms that have no conversions after 30 clicks. This provides a reduced list of Search Terms to review. The specialist then looks at each Search Term and subjectively decides whether or not it is relevant and has a high likelihood of success. Given the subjective nature of this approach it is hard to conceive of a way in which we can compare our approach against that taken by a specialist. Thus, we can consider that the method we are introducing can be considered a baseline model which future approaches will need to improve upon.

Chapter 2

Methods and Technologies

In this chapter we proceed to provide background on the methods and technologies we employ to create revenue per click predictions. We will review the modeling techniques, namely the use of embedding layers and neural networks for regression. We will also discuss how embeddings are trained, looking at state of the art methods. This offers us a comprehensive overview of everything used to achieve the results in this paper before moving to the next chapter where we describe in more detail the actual approach to the model.

2.1 Word Embeddings

Word Embeddings became mainstream in the last few years, enabled by growth in computing power, improvement in methods for training neural networks and the introduction of high performance pre-trained networks. Word Embeddings are essentially an algebraic representation of words and their relative meanings. Words are represented by dense vectors where a vector represents the projection of the word into a continuous vector space. The position of a word within the vector space (ie. the embedding) is learned based on the words that surround it in the training data. They require no annotation and can be trained entirely on unnotated corpora. They can be trained in one context then used in other projects and learned during the fitting of a neural network. They are considered to be a huge improvement over the bag-of-words model which would lead to a sparse representation of a word or document with mostly zero values and very high dimensions. Word Embeddings are considered to be one of the most successful applications of unsupervised learning to date.

2.1.1 The History of Word Embeddings

In this section we will provide a detailed review on the development of Word Embeddings over the last two decades. Since the 1990s distributional semantics have mas made use of vector space models (an algebraic model for representing text). Some novel approaches that were introduced, prior to the widespread adoption of

Word Embeddings, which pioneered the estimation of continuous representations of words including:

Latent Dirichlet Allocation (LDA) - this is a generative statistical model which enables discovery of natural groups within text documents in an unsupervised manner.

Latent Semantic Analysis (LSA) - another technique to analyze relationships within text, but by assuming words closer in meaning will occur in similar pieces of text.

Word Embeddings were introduced by Bengio and colleagues in 2003 who developed a neural language model and trained the embeddings as model parameters (Bengio, 2003). In 2008 the power of pre-trained word embeddings was demonstrated by Collebert and Weston in their paper on "A unified architecture for natural language processing" (Ronan Collobert, 2008). Their work also paved the way for a new neural network architecture which is still used at present. In 2013 Mikolov et al. made great strides forward with the introduction of word2vec, a framework for training and using pre-trained embeddings (Tomas Mikolov, 2013). Shortly later, in 2014, Pennington et al created Glove (Jeffrey Pennington, 2014), further pushing the envelope and bringing Word Embeddings into the mainstream.

2.1.2 Overview Word embedding models

Here we review the main types of Embedding models to provide a more in depth understanding of how they work and how they can be used in Natural Language Processing. In word embedding models a feed-forward neural network takes words from a vocabulary as input and embeds them in a lower dimension space as vectors. Then these are optimized via back-propagation to obtain the final word embeddings as the weights of the first layer, referred to as the *Embedding Layer*.

In particular we will review four types of models:

- Classic neural language model
- C&W model
- Word2Vec
- GloVe

2.1.3 Problem Definition

In order to compare different approaches and techniques we must first define the problem in mathematical terms, adopting notation standards as follows. Our training corpus has a sequence of T words $w_1, w_2, w_3, \cdots, w_T$ which belong to a vocabulary V with size $|V|$. The models have a context of n words. To each word an input

embedding of v_w is associated and has d dimensions. The output embedding for a given word is defined as v'_w. The model will optimize an objective function J_θ with parameters θ. The output of the model is a score $f_\theta(x)$ for every input value of x.

2.1.4 Classic Neural Language Model

Developed by Bengio et al. in 2003, the classic neural language model is a neural network with one hidden layer and uses feed forward to predict the next word in a sequence. See figure 2.1 for an overview of the model architecture.

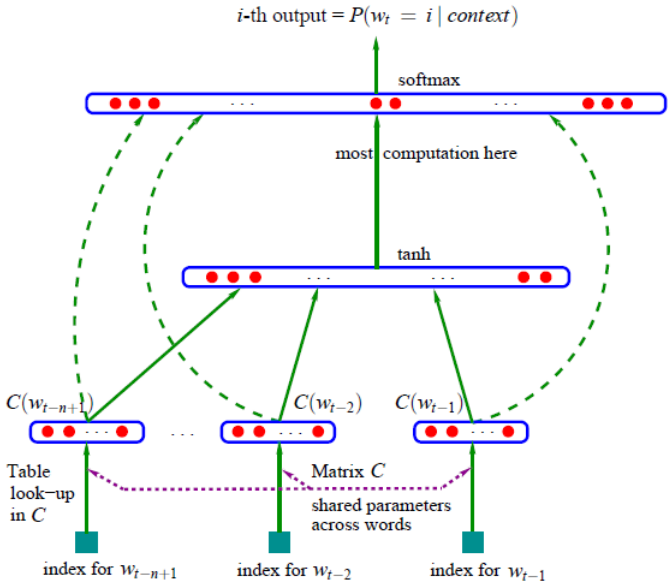

FIGURE 2.1: A neural language model (Bengio et al., 2003)

This model maximizes the standard neural language model objective shown in equation 2.2 where the regularization term has not been included for simplicity). :

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \log f(w_t, w_{t-1}, \cdots, w_{t-n+1}) \tag{2.1}$$

Where $f(w_t, w_{t-1}, \cdots, w_{t-n+1})$ is the output of the model. Another way to look at this is that is the probability $p(w_t \mid w_{t-1}, \cdots, w_{t-n+1})$ which is computed by the softmax and where n represents the number of prior words which where inputted to the model.

Since the introduction of the architecture proposed by Bengio et al in this landmark paper, the building blocks for modeling language with neural networks has remained largely unchanged. They include:

- An Embedding layer created by multiplying an index vector (usually hot encoding) with the word embedding matrix.

- One or more intermediate layers which serve to model the relationships between words such as a fully connected layer which models a non linear relationship between the embedding and those of previous words.
- A Softmax Layer which outputs the probability distribution over words in V.

In addition to laying the foundations for for modeling language with neural networks they also identified two issues which are still being addressed in the methods of today. Namely that the intermediate embedding layers could be replaced with an LSTM (which is the case in state-of-the-art models today) and that the softmax layer is the bottle neck from a computational perspective. The latter issue is still an important topic of focus today.

2.1.5 C&W model

Due to limited computation power at the time and thus the impossibility to manage large vocabularies, Bengio et al's work in neural language models did not gain much traction initially. Collobert and Weston (giving the name to the C&W model) in a paper of 2011 introduced a solution to avoid the bottle neck of the softmax layer (Ronan Collobert, 2011). Instead of using cross-entropy, as employed by Bengio et al., they introduced a pairwise ranking criteria as shown in equation :

$$J_\theta = \sum_{x \in X} \sum_{w \in V} \max\{0, 1 - f_\theta(x) + f_\theta(x^{(w)})\} \tag{2.2}$$

This served to train a network to output a higher score f_θ for a correct word sequence than for an incorrect one. This is achieved by sampling correct windows x containing n words from the set of all possible windows X. For each of these windows an incorrect version $x^{(w)}$ is also created, by substituting the center word of x with another word w from the vocabulary V. Then, when the objective function is minimized the model will naturally assign a higher score for the correct window. The architecture of this model (which does not include the ranking objective is shown in figure 2.2.

The outcome is a language model which is more computationally efficient than that of Benjo et al. due to the elimination of the softmax complexity. However, the intermediate fully connected layers still require intensive computation so while this model was a step forward it still cannot be considered state of the art. However, models trained with this architecture were already able to yield embeddings with similar properties to more modern approaches. For example, cities are clustered close together and synonyms occupy similar locations in the vector space.

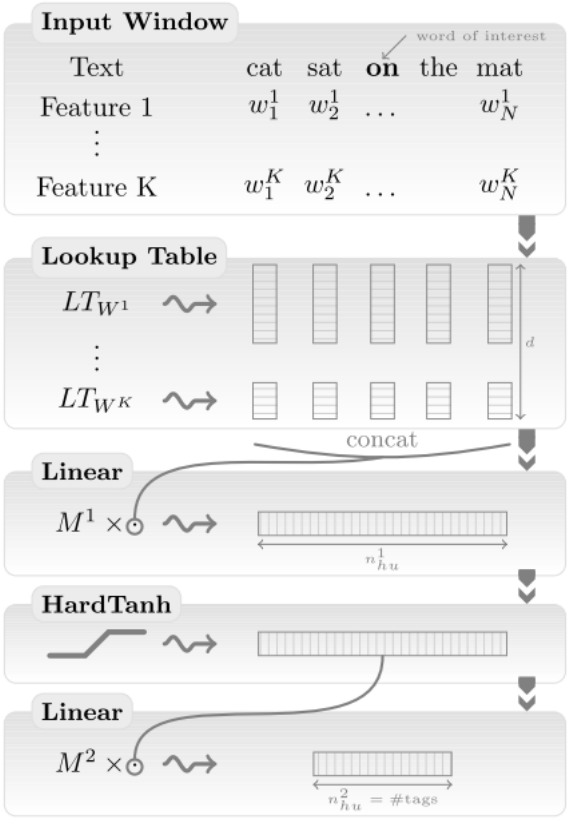

FIGURE 2.2: The C&W model without the ranking objective (Collobert et al., 2011)

2.1.6 Word2Vec

Probably the most influential word embedding model is Word2Vec. It led to significant innovation in deep learning models though technically it is not a deep learning model. Mikolov et al. published two landmark papers. In the first they proposed two new architectures, namely *continuous bag of words* and *skip-gram*, which were computationally more efficient than previous models, such as the Classic Neural Language and C&W models. In the second paper they improved further the strategies for reducing cost of computation. Overall, they outperformed these older due to:

- Having no hidden layer.
- They language model is able to take additional context into account.
- Improved strategies for training.

We now proceed to review the two strategies proposed by Mikolov et al:

Continuous-Bag-of-Words (CBOW) - The key innovation was to generate embeddings that use n words before and after the target wordw_t, rather than using previous words and evaluating their ability to predict the next word. It was

shown that this was sufficient to create accurate embeddings. The architecture is shown in figure 2.3. The objective function for the continuous-bag-of-

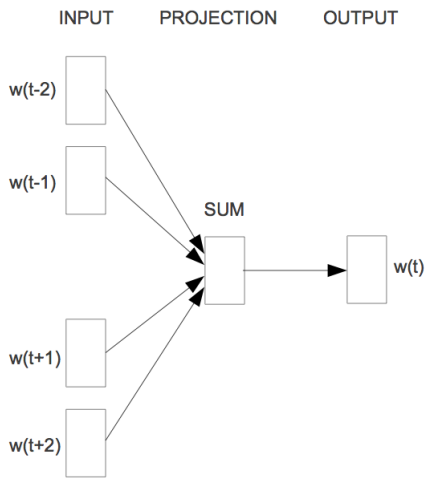

FIGURE 2.3: Continuous bag-of-words (Mikolov et al., 2013)

words model is as follows and as we can see is quite similar to that of the classic language model and is shown in equation 2.3. The only major difference is that, at each time step t, the model receives a window of n word around the target word, rather than n previous words.

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \log p(w_t \mid w_{t-n}, \cdots, w_{t-1}, w_{t+1}, \cdots, w_{t+n}) \tag{2.3}$$

Skip-Gram Model - The innovation introduced in the skip gram model was to invert the objective. While the continuous-bag-of-words model uses surrounding words to predict the central word, the skip-gram model does the opposite. It uses the central word to predict the surrounding ones.

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \sum_{-n\leq j\leq n, \neq 0} \log p(w_{t+j} \mid w_t) \tag{2.4}$$

In figure 2.4 this model is displayed visually.

2.1.7 GloVe

The last model we will review is GloVe. What can be considered a by product of the approach adopted in word2vec, GloVe makes explicit the encoding of vector offset in embedding space. The creators of GloVe showed that the ratio of co-occurrence probabilities of two words contains information and the aim of the model is to capture this information as vector differences. To do this a weighted least square

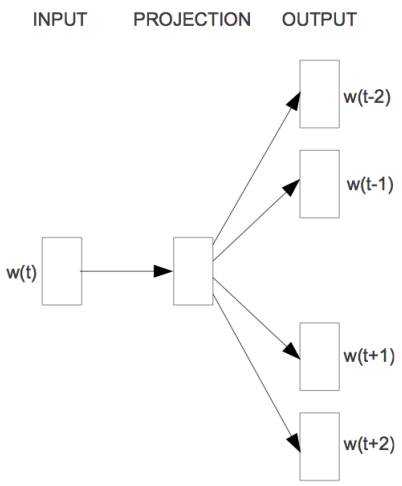

FIGURE 2.4: Figure 5: Skip-gram (Mikolov et al., 2013)

objective function is used (see equation 2.5 which serves to minimize the difference between the dot product of the vectors of two words and the log of their co-occurrence count.

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2 \tag{2.5}$$

To explain further the objective function, w_i is the word vector and b_i is the bias vector of word i. Meanwhile, w_j and b_j are the corresponding context vectors of word j.X_{ij} represents the number of times a word i occurs in the context of word j. Finally, f is a the weighting function.

2.2 Training Word Embeddings

In order to train word embeddings it is necessary to model words in a sequential manner. To derive accurate vector representations large amounts of text data is needed. For example, Word2Vec was trained on 100 billion words from Google News data with 300 dimensions but since then the size of the training data has continued to grow. The challenges of training embeddings on such large data sets can be summarised as follows:

- Expensive computation due to large size of data and number of dimensions.
- Difficulty in modeling long sequences of tokens.

Fortunately, libraries such as Word2Vec and GloVE come pre-trained, ready to use in the task at hand. However, it is interesting to understand how the introduction of LSTM was able to overcome some of the issues found when using RNN. We will

also review the state-of-the-art Attention models which use transformers in order run computation which offer many benefits over its predecessors.

2.2.1 RNN

They way humans absorb and interpret data is a good analogy for a RNN (Recurrent Neural Network). When we, for example, read a book we already have learnt how to read. We use this stored knowledge in order to process new information in a more effective manner. Another example would be predicting what happens next in a movie. In order to do this we need to see what happened previously, process this information then make a prediction on what comes next. Traditional neural networks are not able to do this in an effective manner which seems to be a major limitation. RNN's were designed to overcome this limitation as they are networks with loops in them which allow information to persist.

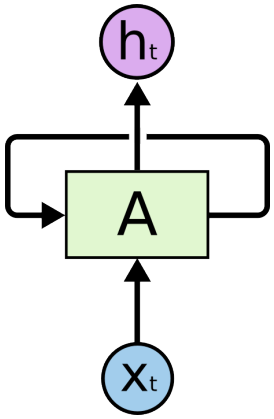

FIGURE 2.5: Example of RNN loops

In figure 2.5 we see a classic example of a RNN loop. Here we have a part of neural network A, which takes an input x_t and outputs the value h_t. This loop allows information to passed from one step of the network to the next. A RNN can be visualized as multiple copies of the same network, each gaining knowledge from the previous loop. If we unroll the loop we have a better visualization of this.

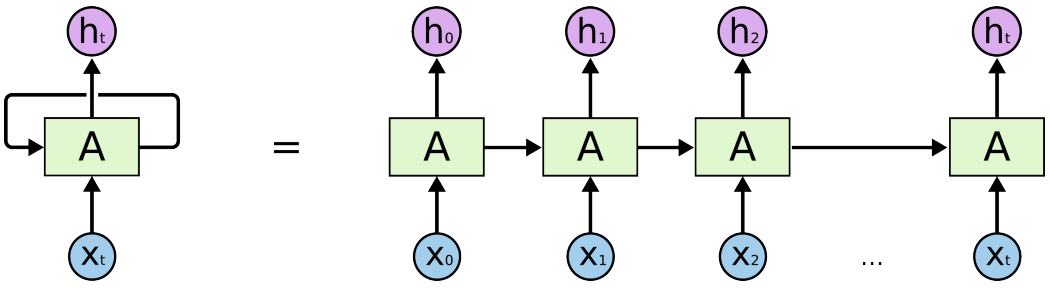

FIGURE 2.6: Example of unrolled RNN

This visualizations shows how related this architecture is to the modelling of sequences. This of course leads to their use in modeling of time series and similar types of problems where data is structured in a sequential manner.

However, RNN's in their natural state have some major problems when modeling long sequences. What we see is that with RNNs gradients propagate over many stages tend to either vanish (which happens most of the time) or to explode (which hinders the optimization process). Thus the distance between the relevant information and that where it is needed , the long term dependencies, presents a major challenge in using RNN's to model large amounts of unstructured text data. In addition to the difficulty in modeling long sequences, RNNs are also expensive from a computational perspective as they do not lend well to parallelization.

2.2.2 LSTM

A LSTM (Long Short Term Memory) network is a special type of RNN. They have the advantage over RNN's are able to learn long-term dependencies. They were initially introduced by Hochreiter & Schmidhuber in 1997 but over time they have been evolved and improved. In many contexts using LSTMs have allowed data scientists to overcome challenges where standard RNN's fell short and are still widely used today (Sepp Hochreiter, 2011).

All RNNs have repeating structures which for the standard version is very simple. An example can be seen in figure 2.7 where we have an extremely simple repeating structure comprised of a single *tanh* layer. With a large number of loops this will lead to problems due to the fact that the loop necessarily reprocesses all information.

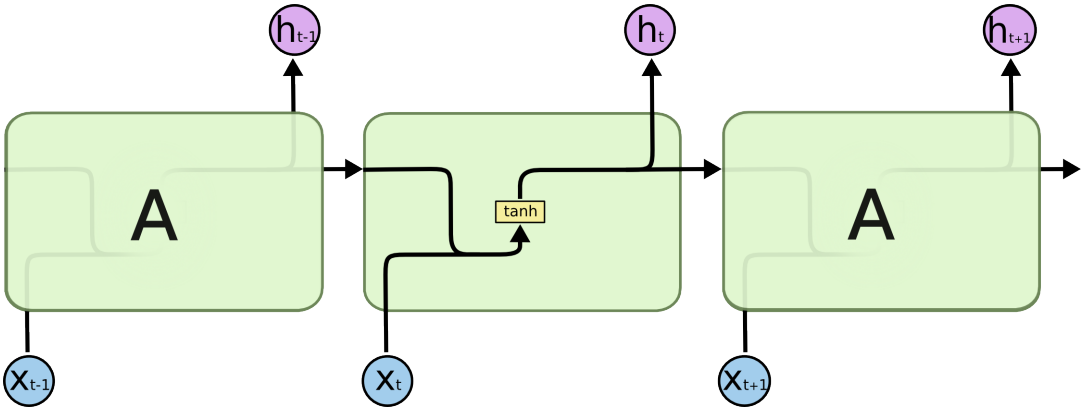

FIGURE 2.7: Simple repeating RNN module with single layer

The architecture of LSTM is designed to avoid the long-term dependency problem. They can easily remember information over long periods of time. They do this with a chain structure, but instead a single neural network layer, we have four that interact in way to allow information to better persist through loops. A visualization is shown in figure 2.8.

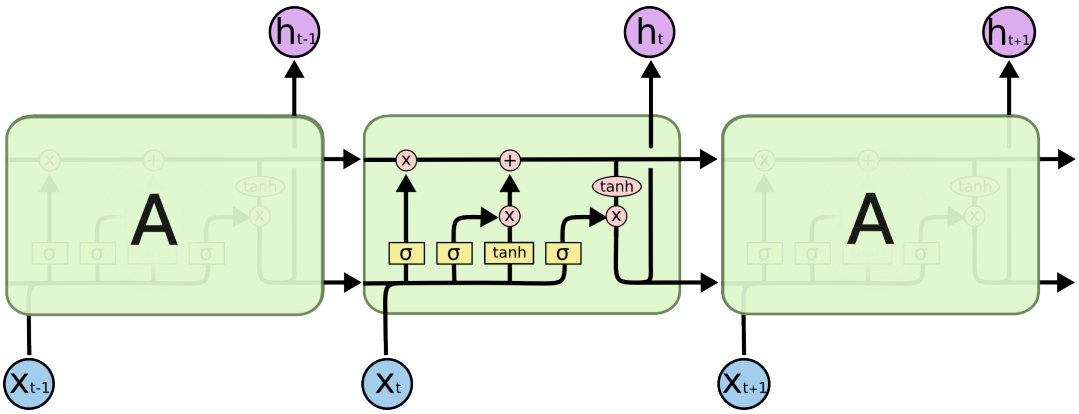

FIGURE 2.8: LSTM loop contains four interacting layers

Going into detail on how this architecture allows information to persist is beyond the scope of this paper, but it is useful to review at a higher level to gain intuition necessary to understand the problem at hand. The most important aspect of the LSTM is the cell state, which is the horizontal line in the top of the diagram also shown on its own in figure 2.9

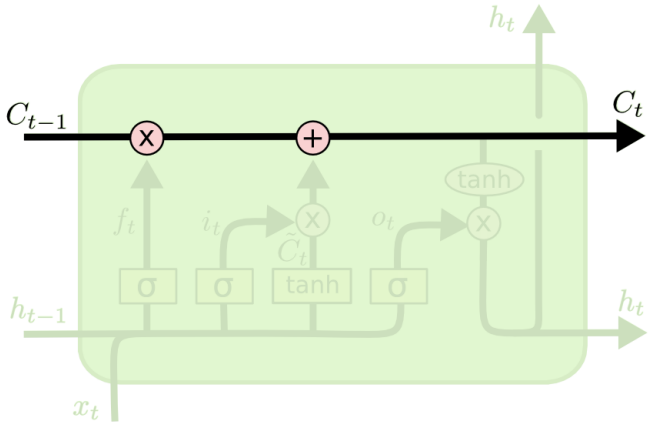

FIGURE 2.9: LSTM cell state

We can think of the cell state as a conveyor belt which runs down the entire chain. Information can run along it easily without being changed. The information which is then included in a given loop is regulated by a structure called a gate shown in figure 2.10.

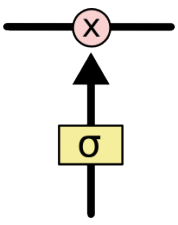

FIGURE 2.10: LSTM gate

These gates serve to optionally let information through and are composed of a sigmoid neural network layer and a point wise multiplication operation. The layer will output a zero to let no information through and a one to let information through. Through the use of multiple gates LSTM controls the cell state. This mechanism allows LSTM to selectively remember or forget things that are important and not so important, thus controlling the propagation of information along the chain. RNN's cannot do this since whenever it adds new information, it also transforms existing information completely, therby modifying all information regardless of whether or not it is important.

While LSTMs outperform RNNs in many contexts, they still have some important limitations. When sentences are too long LSTMs can also run into issues of keeping context from words which are distant from each other. Also, as was the case with RNNs, LSTM it is hard to parallelize processing since you necessarily need to model word to word relationships sequentially.

2.2.3 Transformers

The concept of Attention was first introduced in Computer Vision to allow classification algorithms. Larochelle and Hinton proposed a model to looking at different "glimpses" of an image simultaneously (Hugo Larochelle, 2010). The concept was later extended to NLP and paved the way for the development of the state-of-the-art language models of today. Using Attention we can look at all words at the same time and learn to "pay attention" to certain ones depending on the task at hand (Ashish Vaswani, 2017). However, Attention models still suffer from issues in parallelization when implemented with RNNs.

CNNs (Convolution Neural Networks) offered a potential alternative approach in order to parallelize computation. CNNs work in parallel, where each input word can be processed simultaneously, without dependence on the previous word. Furthermore, the "distance" from the output to the input is of order $log(N)$. Thus this architecture is less susceptible to vanishing and exploding gradients. However, CNNs do not do well at modeling dependencies within sentences so while computationally efficient do not lead to the development of high quality language models.

Transformers evolved to combine the language modelling benefits of Attention models with the parallelization power of CNN's. They have revolutionized NLP in particular machine translation. Since their introduction there has been a surge in NLP applications analogous to what occurred in Computer Vision following the ImageNet competition in 2012.

Today, Transformers are the state-of-the-art model for the sequence models and have superior performance to LSTM which working with text processing. In contrast to previous RNN and LSTM architectures there is no recurrent connection to previous states. Instead an entire sequence can be processed simultaneously. This

architecture lends itself particularly well to parallelization and modern day GPUs. They do require significant amount of memory during training but this can be circumvented by reducing precision.

Much literature is available on how transformers work. While interesting, we did not include an in detail discussion here since we did not actually use them in our experiments.

Chapter 3

The Model

In this chapter we go in detail the model which we built to solve the problem at hand. The first step taken was to introduce a way to convert the Google Search Term report into a format which can be inputted into a classifier. Following this the model we developed a model which firstly encodes search terms as word embeddings, then performs classification using a deep neural network.

3.1 Data Preparation

As described previously in chapter 1 the data needed is found in the Google Ads Search Term report.

3.1.1 Processing data with Binomial Distribution

In our data set we are interested in whether or not a Search Term can lead to a conversion event. The data set that we retrieve from the Search Terms Report, however, contains both clicks and conversions and is not yet ready for training of a classifier. We must find a way to convert the data into the desired format. For this we can model the probability of a conversion occurring using the binomial distribution. We can use the probability mass function of the binomial distribution to find the probability for a given number of trials then use this to derive the probability that at least one success occurs.

The formula for the Binomial Probability Mass Function is shown in equation 3.1

$$P(k) = P(X = k) = \binom{n}{k} p^k (1-p)^{n-k} \tag{3.1}$$

Where k is the exact number of successes in n independent Bernoulli trials and p is the probability of success.

Thus the formula we derive for the probability that at least one success is found using equation 3.2

$$P(k > 0) = 1 - (1 - p)^n \tag{3.2}$$

In our problem we can replace variables n number of trials and p probability of success as follows:

- p is substitute with the average conversion rate of the data set.
- we try different values of n until we have a probability of success of approximately 0.9. This number was agreed with the marketing team at the company.

In table 3.1 we see an example where $p = 0.15$. Accordingly we would choose a click threshold of 14 clicks.

TABLE 3.1: Probability of non zero conversions

Click Threshold	P(Conversions > 0)
0	0.00
2	0.28
4	0.48
6	0.62
8	0.73
10	0.80
12	0.86
14	0.90
16	0.93
18	0.95
20	0.96

3.1.2 Creating Success Label

Having obtained the n number of clicks necessary for a 0.9 probability that a conversion should have occurred we use this number to clean our data set. We do this by removing all records with less than the required click threshold n. Subsequently we label all remaining records as follows:

- if $conversions > 0$ success label is 1.
- if $conversions = 0$ success label is 0.

This is an appropriate data structure for purpose of running a classification algorithm either with a traditional approach or using Deep Neural Networks. An example is shown in table 3.2

3.1.3 Data Import

We import the data into Google Collab notebooks. It is similar to Jupyter notebook but hosted in the cloud and convenient for running small deep learning models. It offers hosted runtime which makes it a good choice as well as native installation of

TABLE 3.2: Example of a cleaned Search Term Report

Search Term	Success
health benefits plans	1
does nebraska have obamacare	1
florida blue obamacare	0
healthcare texas	0

tensor flow. This meant we could be up and running in a short time without the need to spend a lot of time installing packages and managing environments.

Since Google Ads data is very clean, minimal work in cleaning the data was needed. For example, there were no "NA" values to deal with or entries with missing data.

3.1.4 Remove Stop Words

Having imported the data, we proceed to clean stop words. Especially when dealing with classification problems, stop words do not offer much useful information and take up space. Common examples of stop words to be removed include “the”, “a”, “an” and “in”. We make use of NLTK package (Natural Language Toolkit) which provides stop words stored in 16 different languages. Since our data is in English we only apply the cleaning for the English language.

The code used for removing stop word is as follows:

```
---
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk

nltk.download('punkt')
nltk.download('stopwords')

stop_words = set(stopwords.words('english'))

def remove_stopwords(phrase):
  words = word_tokenize(phrase)
  return ' '.join([word for word in words if not word in stop_words])

df['query'] = df['query'].apply(remove_stopwords)
---
```

3.1.5 Dealing with Un-balanced Data

The next step is to balance the data. Since the data set is very unbalanced with 7367 successes and 424 failures, it will be difficult to fit a model to this data set. We tried

different approaches and then opted to randomly down sample the successes.

The code used for this step was as follows:

```
---
balanced_df = pd.concat([df.loc[df.sale == 0],
df.loc[df.sale == 1].sample(len(df.loc[df.sale == 0]))])
balanced_df = balanced_df.sample(frac=1, random_state=12)
---
```

3.1.6 Split Train and Test

We split the data into Train and test with the following proportions randomly sampled:

- Train = 80% of samples
- Test = 20% of samples

The code used for this step was as follows:

```
---
train_dataset = balanced_df.sample(frac=0.8, random_state=0)
test_dataset = balanced_df.drop(train_dataset.index)
---
```

3.2 The Classification Algorithm

In this section we propose the architecture for the classification algorithm to solve the problem at hand.

3.2.1 Prepare Embeddings

We used pre-trained embeddings

- Create a vocabulary index for all tokens in our data sample. We represent text with numbers using this index and to later map these numbers in the embedding layer.
- Download GloVE optionally keep only words of interest. We used 300-dimensional GloVe embeddings of 400k words computed on a 2014 dump of English Wikipedia.
- Create the embedding matrix with all words that could be seen in training to be used in the embedding layer

The code used for creating vocabulary index is as follows:

```
---
import tensorflow as tf
```

```
from tensorflow.keras.layers.experimental.preprocessing import TextVectorization

vectorizer = TextVectorization(max_tokens=1000, output_sequence_length=query_token_length)
text_ds = tf.data.Dataset.from_tensor_slices(train_dataset['query']).batch(64)
vectorizer.adapt(text_ds)

vocab = vectorizer.get_vocabulary()
word_index = dict(zip(vocab, range(len(vocab))))
---
```

The code used for Importing GloVe word embeddings is as follows:

```
---
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip -q glove.6B.zip
---
```

The code for creating the embedding matrix is as follows:

```
---
import numpy as np

embeddings_index = {}
with open('glove.6B.300d.txt') as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        embeddings_index[word] = coefs

print("Loaded %s word vectors." % len(embeddings_index))

num_tokens = len(vectorizer.get_vocabulary()) + 2
embedding_dim = 300
hits = 0
misses = 0

# Prepare embedding matrix
embedding_matrix = np.zeros((num_tokens, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # Words not found in embedding index will be all-zeros.
        # This includes the representation for "padding" and "OOV"
        embedding_matrix[i] = embedding_vector
```

```
        hits += 1
    else:
        misses += 1
print("Converted %d words (%d misses)" % (hits, misses))
---
```

3.2.2 The DNN Classifier

Keras was used to develop the model, due to it's ease of use and ample documentation for solving classification using embeddings. The architecture chosen is very simple, but could potentially be enhanced in future work. While we did try out more complex deeper networks they did not have much impact.

The architecture used can be seen in the figure 3.1.

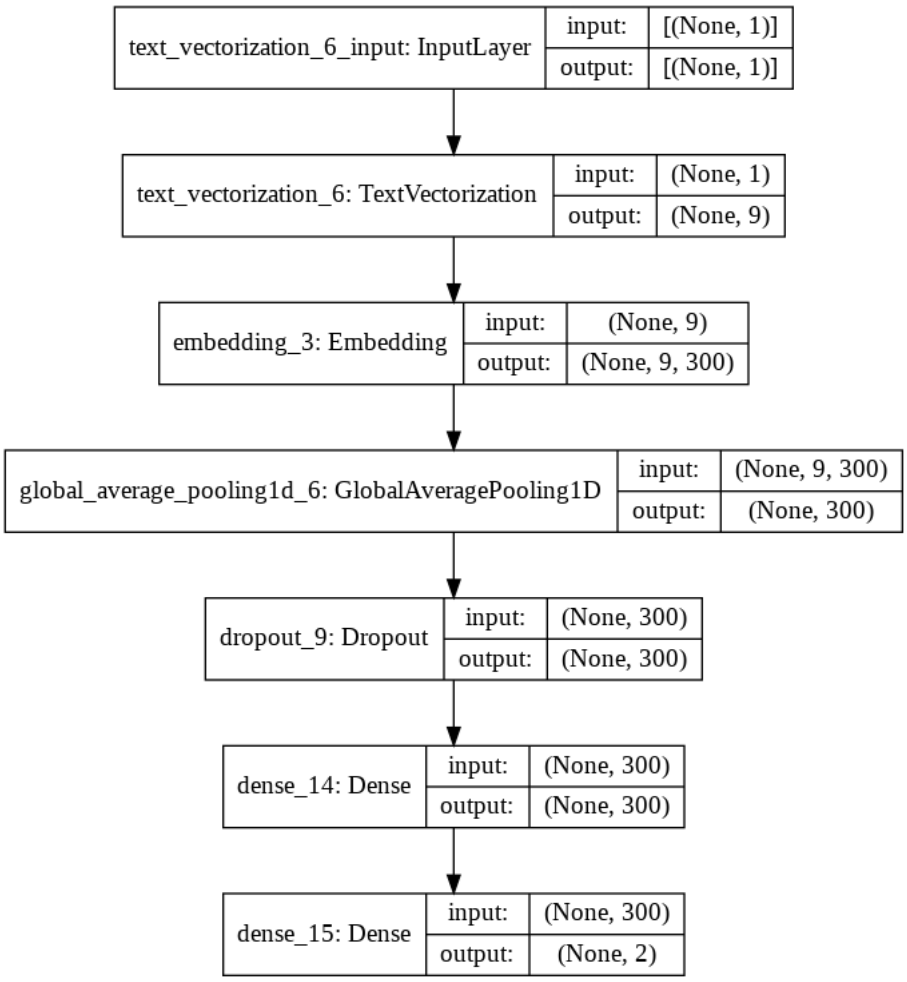

FIGURE 3.1: DNN Architecture of Classifier

In terms of the other model parameters we chose to use "adam" optimizer, "categorical_crossentropy" for loss and "accuracy" as the metric, which are frequently used in classification problems in the Keras documentation.

The code used to build the model is as follows:

```
---
from tensorflow.keras import layers

dnn_model = keras.Sequential([
      vectorizer,
      embedding_layer,
      layers.GlobalAveragePooling1D(),
      layers.Dropout(0.2),
      layers.Dense(300, activation='relu'),
      layers.Dense(2, activation='softmax'),
  ])

dnn_model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
---
```

The code for training the model is as follows:

```
---
%%time
history = dnn_model.fit(
    train_dataset['query'], tf.keras.utils.to_categorical(train_dataset['sale']),
    validation_split=0.2,
    verbose=0, epochs=20)

hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
hist.tail()
---
```

3.2.3 Full Code

Given that the company asked for data to remain confidential I am not publishing to Github. Rather I provide the following view link to a google collab notebook where the code is stored:

https://colab.research.google.com/drive/10E3YjOgE27ct$_aLQ6px7sEz4ewtZxheX?usp=sharing$

Chapter 4

Results and Conclusions

In this chapter we review results of the model we built, summarize our conclusions and discuss potential improvements to the model.

4.1 Results

4.1.1 Model Training

Training time was found to be very low with this architecture, consistently under 30 seconds using 20 Epochs. This number of Epochs was chosen, experimenting with different values, in order to limit the possibility of over-fitting but still getting good accuracy on training. We can see in figure 4.1

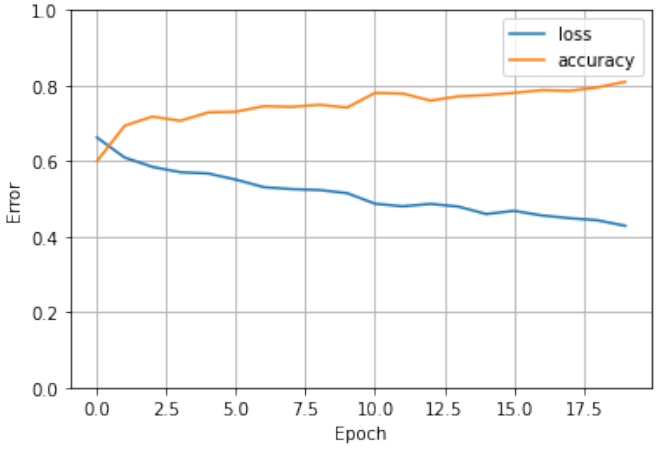

FIGURE 4.1: Training error by Epochs

4.1.2 Model Evaluation

The best result obtained was a 71.8% accuracy on the test data. This means that we can successfully predict that a keyword should be included as negative 71.8% of the time. While this was somewhat lower than what we set out to achieve there are several improvements which could be made in order to improve the score which could be interesting to explore in further work.

4.1.3 Model Variations

In order to try and improve the accuracy of the model we tried various experiments. These included:

- increasing the number of Epochs.
- up sampling instead of down sampling in data preparation.
- including more dense and dropout layers.
- different type of optimizer.

None of these had much impact on the accuracy of the model.

4.2 Conclusions

In this paper we introduced a rigorous approach to identifying Search Terms which should be blocked. We did this by converting the Search Term data set into a format more suitable for performing classification, creating embeddings for the words contained within, then performing classification with a DNN. The objective was to create an algorithm which could improve the productivity marketeer who currently spends a significant amount of time classifying Search Terms manually. To do this an accuracy of 80% was required while we obtained an accuracy just over 70%. Thus, the main objective was not achieved. However, we did apply a methodology never applied before in this context. There is a great potential to improve on the simple approaches introduced which we did not have time to explore in this paper.

4.3 Potential improvements of the model

While we did not manage to reach a satisfactory level of accuracy there is much room for improvements. Some ideas for development of this work to improve accuracy are as follows:

- Obtaining more data would help to improve the model. Because the advertiser has already optimized their account and has blocked undesirable Search terms over several years we do not have a lot of "bad" Search terms on which to train our model. It is likely that the model will perform quite differently with other advertisers.
- Potentially more work should be carried out to improve the management of the imbalanced data set. For example using a system such as SMOTE.
- A potential issue with our model is that search terms with multiple words will have multiple embeddings. To combine them we take the average. This could have the undesired effect of washing out information, especially if we

are dealing with long Search Terms. In future models this could be dealt with in a more elegant manner.

- We should develop a better system to identify Success/Failure in our data. We developed a simple system using an assumption of the true conversion rate in order to obtain a ground truth label, however, this assumption most likely flawed. This is because conversion rate is not always the same for different Search Terms and other factors such as device, user location and time of day come into play. Thus we need to develop a better system to obtain the ground truth label. Reinforcement learning could be very useful here as it would give the opportunity to integrate expert knowledge and the algorithm would work to enhance the productivity of the marketeer instead of attempting to replace him or her.
- Of particular interest would be to train our own domain specific word embeddings. The health insurance segment has a lot of jargon that is highly specific. It is likely that the use of certain words within this context have quite a different purpose. For example, a frequent word in our data set is "obama". GloVE most likely associates this word to other jobs or presidents, while in fact in our case it is most likely refering to a type of insurance "obama care".
- We should try using some more state-of-the-art technology to analyze text. Transformers now offer higher performance as compared to the approach we used in this paper. It would be interesting to apply these new methods in order to take things to the next level.

Bibliography

Ad_Espresso (2019). *Google Ads Keyword Match Type*.

Ashish Vaswani Noam Shazeer, Niki Parmar Jakob Uszkoreit Llion Jones Aidan N. Gomez Lukasz Kaiser Illia Polosukhin (2017). "Attention Is All You Need". In:

Bagshaw, Thomas (2015). *The Evolution of Google AdWords*.

Bengio Y., Ducharme R. Vincent P. Janvin C. (2003). "A Neural Probabilistic Language Model". In:

Gagan Aggarwa S. Muthukrishnan, Dávid Pál Martin Pál (2009). "General Auction Mechanism for Search Advertising". In:

Google_Ads (2021). *Ad Rank*.

Hugo Larochelle, Geoffrey E. Hinton (2010). "Learning to combine foveal glimpses with a third-order Boltzmann machine". In:

Jeffrey Pennington, Richard Socher (2014). "Glove Global Vectors for Word Representation". In:

Ronan Collobert, Jason Weston (2008). "A unified architecture for natural language processing: deep neural networks with multitask learning". In:

Ronan Collobert Jason Weston, Leon Bottou Michael Karlen Koray Kavukcuoglu Pavel Kuksa (2011). "Natural Language Processing (Almost) from Scratch". In:

Sepp Hochreiter, Jürgen Schmidhuber (2011). "Long Short-Term Memory". In:

Sparling, Evan (2015). *Negative Keyword Match Types – When to Use Broad Match*.

Tomas Mikolov Kai Chen, Greg Corrado Jeffrey Dean (2013). "Efficient Estimation of Word Representations in Vector Space". In:

wikipedia (2021). *Generalized second-price auction*.