FindZebra - using machine learning to aid diagnosis of rare diseases

Svenstrup, Dan Tito

Publication date:
2018

Document Version
Publisher's PDF, also known as Version of record

FindZebra - using machine learning to aid diagnosis of rare diseases

Dan Svenstrup

Kongens Lyngby 2017

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary (English)

FindZebra is a search engine for rare diseases intended to act as a diagnosis decision support system (DDSS) capable of assisting the user both during and after a search. Rare diseases are diseases that affect only a small part of the population (less than one in two thousand). Currently around seven thousand rare diseases are known and it is estimated that $6 - 8\%$ of the population will be affected by a rare disease during their lifetime. Due to their rarity and large number, diagnosis of rare diseases is difficult and often associated with year long delays and diagnostic errors. These difficulties with diagnosis have a profound human and societal cost. This means that even a small increase in success rate when using a tool such as FindZebra could potentially have a great impact on society. In this dissertation we explore four lines of research for improving FindZebra using machine learning methods.

The first line of research is on how to improve the retrieval performance of FindZebra. By using a combination of improved models, medical databases and corpus expansion we show that it is possible to obtain a substantial improvement in retrieval performance compared to current state-of-the-art document retrieval systems.

Improving retrieval performance is important, but is not the only way of improving the success rate of a DDSS such as FindZebra. Following an unsuccessful search, the search engine should assist the user by indicating what information is likely to be missing. This idea is called Information Completion (IC) and will be explored in the second line of research.

In order to represent words (and other discrete tokens) in a neural network it

is necessary to transform each word to a vector form. This is typically accomplished by using a *word embedding*, which is an essential component in any word based neural network. The third line of research is on how to improve this basic component.

Users of FindZebra who do not have English as their primary language often have difficulty expressing complex medical queries in English. Optimally, a user should be able to write a query in his or her native language and the search engine should then give a suggestion for a differential diagnosis based on all the information contained in a multilingual corpus, not only in the native corpus. Methods for performing multilingual search will be the fourth line of research explored in this dissertation.

Summary (Danish)

FindZebra er en søgemaskine til sjældne sygdomme. Den fungerer som et værktøj til diagnostisk beslutningsstøtte som kan assistere brugeren både under og efter en søgning. En sjælden sygdom er en sygdom som kun rammer en lille del af befolkningen (mindre end 1 ud af to tusinde indbyggere). I dag kender vi til ca. syv tusinde sjældne sygdomme og det estimeres at $6 - 8\%$ af befolkningen kommer til at lide af en sjælden sygdom på et tidspunkt i deres liv. Sjældne sygdomme er ofte svære at diagnosticere. Dette resulterer ofte i fejldiagnosticeringer, og der kan gå mange år før den korrekte diagnose stilles. Disse problemer med diagnosticering har en høj pris, både for det enkelte menneske og for samfundet. Dette betyder at selv en lille forbedring i antallet af korrekte diagnoser ved brug af et værktøj som FindZebra potentielt kan have en stor indvirkning på samfundet. I denne afhandling vil jeg undersøge fire forskellige områder hvor FindZebra kan forbedres ved hjælp af kunstig intelligens.

Det første område drejer sig om, hvordan man kan forbedre præcisionen af søgninger i FindZebra ved at benytte en kombination af forbedrede modeller, medicinske databaser og korpus udvidelser.

Forbedret præcision af søgninger er vigtigt, men det er ikke den eneste måde man kan forbedre diagnostisk effektivitet. Efter en fejlet søgning bør søgemaskinen f.eks. assistere brugeren ved at gøre opmærksom på hvilken information der kunne mangle. Denne type assistance kaldes Informations Fuldstændiggørelse og er det andet område, der vil blive undersøgt.

I et neuralt netværk er man nødt til at transformere ord (og andre diskrete elementer) til en vektor form. Dette gøres traditionelt ved at benytte en *ord*

indlejring, som er en essentiel komponent i ethvert ord-baseret neuralt netværk. Det tredie område jeg vil undersøge er, hvordan denne meget vigtige komponent kan forbedres.

Det er ofte svært for brugere af FindZebra at udtrykke komplekse medicinske søgestrenge på engelsk. I en optimal verden ville en bruger være i stand til at udtrykke en søgestreng på sit modersmål og søgemaskinen ville herefter give et forslag til en differential diagnose baseret på al informationen i det multisprogede korpus. Det fjerde område jeg vil undersøge er, hvordan man kan udvikle en metode til multisproget søgning.

Contributions

The thesis will be based on the main contributions listed below:

Main contributions

- Svenstrup, Dan, Jonas Meinertz Hansen, and Ole Winther. "Hash Embeddings for Efficient Word Representations." arXiv preprint arXiv:1709.03933 (2017). *Accepted at NIPS 2017.* [SHW17a].
- Svenstrup, Dan and Ole Winther. "Performance Optimization for Specialized Domain Information Retrieval.", *Submitted to Scientific Reports.* [SW17].
- Svenstrup, Dan, Jonas Meinertz Hansen, Mads Emil Matthiesen and Ole Winther. "Information Completion for Medical Search Assistance.", *Submitted to Artificial Intelligence in Medicine.* [SHMW17].
- Svenstrup, Dan, Jonas Meinertz Hansen, and Ole Winther. "Zero Shot Cross language Text Classification.", *Submitted to ICLR 2018.* [SHW17b].

Other contributions

We wrote one additional article that has not been included in this thesis. It is a review article where we investigate state-of-the-art medical retrieval. In the

article we compare FindZebra to other search engines such as Google, PubMed and we try to give an (approximate) comparison with IBM Watson.

- Svenstrup, Dan, Henrik L. Jørgensen, and Ole Winther. "Rare disease diagnosis: a review of web search, social media and large-scale data-mining approaches." Rare Diseases 3.1 (2015): e1083145. [SJW15].

Software

I wrote a Python library that is similar to Scikit-Learn. It has less build-in features compared to Scikit-Learn but is a bit more flexible.

- Svenstrup, Dan. "Flow framework for text classification". https://github.com/dsv77/flow, 2017. "Flow Framework for text classification." [Sve17]

Preface

This thesis was prepared at Department for Applied Mathematics and Computer Science at Technical University of Denmark in fulfillment of the requirements for acquiring a PhD in Engineering. The work was funded by the Lundbeck foundation. The PhD was conducted under guidance from the main supervisor professor Ole Winther, Department for Applied Mathematics and Computer Science, Technical University of Denmark. The work was carried out between September 2014 and October 2017.

Lyngby, 31-October-2017

Dan Svenstrup

Acknowledgements

My years as a PhD student has developed me both personally and academically. It has, however, not always been easy and has sometimes been difficult to get through. I would like to thank my supervisor professor Ole Winther for letting me take on this project, for letting me be a part of FindZebra and for some of the most challenging years of my life. You have encouraged me to look at things from different perspectives, and I appreciate your advice especially on how to write academic papers. And thank you for always taking time to give extremely valuable feedback, no matter how busy your schedule.

I would like to thank Thomas Terney, Jonas Meinertz Hansen, David Kofod Wind, Kaspar Kristensen and Jonatan Bording for some very interesting project collaborations and discussions on machine learning and entrepreneurship. Those discussions have significantly shaped the contents of this thesis.

I would like to thank my colleagues and staff at DTU Compute. Among others I would like to thank the research group around Ole Winther. Especially I would like to thank Marco Fraccaro and Camilla Falk Jensen, it was really a pleasure sharing an office with you. I would also like to thank Wanja Andersen.

I would also like to thank Mads Emil Matthiesen, Tobias Due Munk, Søren Anker Nielsen and Rudolfs Berzins for their work at FindZebra. Thank you for a very interesting journey.

I would also like to thank all the students who have written their master thesis, bachelor thesis and other projects at FindZebra. It has been a pleasure working with you.

I would also like to thank Andreas Jespensgaard, Peter Lucas and Christina Kildentoft at Hedia for some inspiring company at COBIS.

I would also like to thank my friends and family for supporting me through three years of considerable amounts of frustration and doubt.

Most importantly, I would like to thank Julie Rønnebæk Kongsbak for moral support and for proofreading the thesis. Without your support there would be no thesis. Thank you for being the co-author of my life.

Lastly, I would like to thank the Lundbeck Foundation for supporting this research project.

Contents

Chapter 1

Introduction

A rare disease is a diseases that affect only a small percentage of the population. There is no consensus on how small this percentage has to be for a disease to be classified as a rare disease. In the United States a rare disease is a disease that affects less than 1 in 1500, in the EU it is a disease that affect less than 1 in 2000. It is estimated that 6-8% of the European population will be affected by one of the known 7.000 rare diseases during their lifetime [Rod05]. Due to their rarity, diagnosis of rare diseases is often associated with yearlong diagnostic delays and errors [Rod05]. In other words, rare diseases (and their diagnosis) pose a huge societal problem.

The internet is increasingly being used for diagnosis by medical professionals. A survey from 2012 among 506 general practitioners (GPs) in the United States showed that the use of online web tools had become an integral part of the daily work of medical professionals [Res]. The study showed that doctors use of the web is dominated by general search engines such as Google, and portal websites intended for physicians (such as OMIM, PubMed or FindZebra).

Doctors are not the only ones using the internet for diagnostic purposes. A study from 2013 [Cen] showed that 35% of all American adults had used the internet specifically for diagnosis within the past year. 77% began the search for a diagnosis using a search engine such as Google or Yahoo. 13% began at a website specializing in health information, such as WebMD or FindZebra and

the remaining 10% started the exploration in other places such as Wikipedia or Facebook.

FindZebra is an online, publicly accessible web search engine for rare diseases. It is intended to serve as a diagnosis decision support system (DDSS) capable of assisting the user both during and after a search.

Due to the high number of persons suffering from a rare disease and due to the widespread use of online tools for diagnosis, even a modest increase in diagnostic performance of online tools such as FindZebra can have a huge impact. In this thesis I will explore three directions for improving the performance of FindZebra as a DDSS. The first is by improving the retrieval performance, the second is by assisting the user following an unsuccessful search and the third is by providing native language support. In addition to these three research directions I will explore a general method for improving the way words and discrete tokens are represented in neural models. The outline of the thesis is as follows:

Chapter 2 will contain a short introduction to the FindZebra search engine and the various forms of data used by the search engine.

Chapter 3 will contain a short introduction to the the most essential machine learning concepts used in the thesis.

Chapter 4 will give an introduction to the article *Performance Optimization for Specialized Domain Information Retrieval* that can be found in Appendix A. The article describes several methods for improving retrieval performance through corpus expansion, improved machine learning models, ensembles and the inclusion of structured data. Using a combination of these methods we were able to obtain a substantial increase in performance of more than 13% compared to our baseline model.

Chapter 5: describes the concept of *Information Completion* (IC) and gives an introduction to the article *Information Completion for Medical Search Assistance* that can be found in Appendix B. Medical IC is defined as the process where the search engine asks the user relevant questions with the express purpose of filling out missing or incomplete information that could be important for finding the correct diagnosis. The article describes a method for training IC models based on just unstructured medical data and the UMLS database. We also describe a possible improvement to the presented IC method that is not described in the article.

Chapter 6 gives a short summary of the article *Hash Embeddings for Efficient Word Representations* that can be found in Appendix C. In order to represent words (and other discrete tokens) in a neural network it is necessary to trans-

form each word to a vector form. This is typically accomplished by using an embedding. The hash embeddings presented in the article are an improvement to the standard embeddings normally used in NLP models. These embeddings have been used for word representation in all of the models described in this thesis (except for the information completion article, which pre-dates the hash embedding idea).

Chapter 7 gives a short description of the ideas and results from the article *Zero-Shot Cross language Text Classification* that can be found in Appendix D. Medical literature is almost completely dominated by literature in English and as a consequence, users of a medical search engine must also specify queries in English. This can be a problem for users not having English as their primary language. Optimally, a user should be able to write a query in his or her native language and the search engine should then give a suggestion for a differential diagnosis based on all the information contained in a multilingual corpus, not only in the native corpus. The article described in this chapter presents one possible way of accomplishing this.

Chapter 8 contains the closing remarks and directions for future work.

Chapter 2

Introduction to FindZebra

This chapter will give a brief introduction to the current FindZebra search engine, which is available online at www.findzebra.com/raredisease. The website does not yet include any of the improvements to the search engine described in this thesis. However, it is necessary to know how the website works in order to understand how to improve it.

2.1 The FindZebra search engine

FindZebra is (primarily) a search engine for rare diseases. It does, however, contain articles on most diseases including common diseases such as influenza. The user interface is very simple and can be seen in Figure 2.1. It is based on an Apache Solr [SPPM15] backend. Solr uses a scoring algorithm that computes the similarity between a query q and a document d. The scoring formula is given

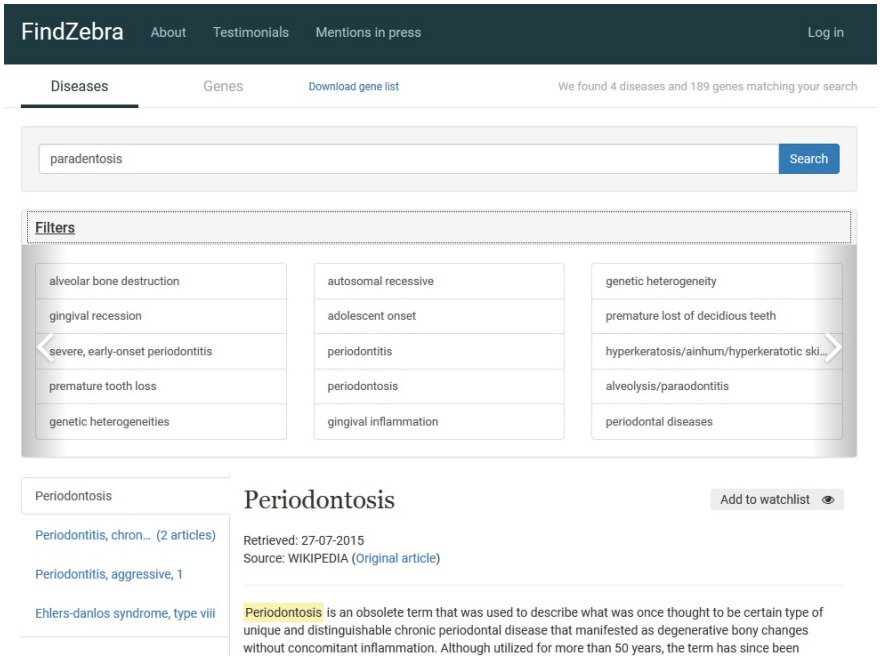


Figure 2.1: The FindZebra website. The figure shows the *disease view* of the search results. The middle section are the symptom filters.

by

$$
\begin{aligned}
\text{score}(q,d) &= L_d C_{q,d} \sum_{t \in q} \sqrt{\text{tf}(t,d)} \cdot \text{idf}(t)^2 B_t \\
t, d, q &= \text{term, document, query} \\
L_d &= \text{“higher score to shorter docs”} \\
C_{q,d} &= \text{fraction of q covered by d} \\
B_t &= \text{boost applied to term } t \\
\text{tf(t, d)} &= \#\text{occurrences of term t in d} \\
\text{idf(t)} &= 1 + \log \frac{\#\text{docs}}{1 + \#\text{docs containing t}}
\end{aligned}
$$

Even though Solr uses a similarity scoring that is (relatively) simple, it is still a very strong baseline model that often performs surprisingly well in practice. In [SJW15], for example, it is shown that FindZebra with a Solr backend is more than twice as likely to have the correct diagnosis in top 20 of the search results compared to other popular alternatives such as Google or PubMed.

FindZebra uses a corpus of medical texts as well as two sources of structured data. These will be described in the following section.

2.2 Structured datasources

2.2.1 The UMLS database

The UMLS database is an ontology of medical concepts. It is currently maintained by the National Library of Medicine[1]. Each word/phrase in the database is associated with a concept id (CUI). For example *influenza*, *influenzas*, *flu*, *human influenza* etc. all share the same CUI since they are all conceptually equal. Each concept has several important properties:

Semantic type. The semantic type describes what kind of concept it is, such as *Amino Acid Sequence* or *Congenital Abnormality*. This property can for example be used for identification of symptoms and genes.

Relationships to other concepts. Each concept has relations to other concepts. For example, a concept can be narrower/broader than other concepts.

Classification codes. The most important of these codes are the ICD-10 codes, which are used in all parts of the American health care system for classification of diagnosis. The ICD-10 codes are directly translatable to the Danish SKS codes which are used in the Danish health care system. This means that in order to able to link a disease to information from the Danish/American health care system (e.g. disease prevalence information), it is necessary to be able to map the disease to an ICD-10 code.

2.2.2 Disgenet

Disgenet[2] [PBQR+17] is a database consisting of 561k gene-disease associations between 17.1k genes and 20.4k diseases. These relationships have been extracted from curated sources such as UNIPROT[3] and text mined from sources such as GAD[4]. Each disease-gene relationship is given a heuristic score based on the number of curated sources in which the relationship is described, the number of

[1] See http://www.nlm.nih.gov/research/umls/
[2] http://www.disgenet.org
[3] http://www.uniprot.org/
[4] https://geneticassociationdb.nih.gov/

animal models describing the relationship and the number of literature publications supporting the relationship[5].

2.2.3 Use of the UMLS/Disgenet databases in FindZebra

The UMLS and Disgenet databaeses are used in several places of FindZebra:

- **Synonym extraction**. By using the semantic types *Sign or Symptom* and *Finding* we can identify symptoms in text. For each disease we have extracted all the symptoms in documents about that disease. This information is for example available through the FindZebra API[6].
- **Filters**. Following a search, a range of symptom suggestions are presented to the user (see middle part of Figure 2.1). The ordering of the symptom suggestions is based on a simple score for each symptom s. The score is calculated by

$$\text{score}(\text{s} \mid \text{query}) = \text{idf}(\text{s}) \sum_{\text{doc in corpus}} \text{doc}_{\text{score}} \mathbb{1}_{s \in \text{doc}}$$

  where $\mathbb{1}$ denotes an indicator function. Documents not in the search results will have a score of zero. These filters can be considered a very simple form of *information completion* which is the topic of chapter 5.
- **Symptom synonyms**. Using the semantic types we can identify symptoms and using the CUI of a symptom we can get all synonyms of each symptom. This relationship between symptoms can be incorporated into the Solr search engine. This is done by normalizing both query and corpus to the same normalized form where each symptom is replaced by a canonical form of the symptom. This means that a search for e.g. `paradentosis` will also return documents containing `periodontosis` (as illustrated in Figure 2.1).
- **Gene view**. The Disgenet database provides disease-gene relationships and a score for each such relationship. These gene-disease relationships can be turned into query-gene relationships by using the (heuristic) formula

$$\text{score}(\text{gene} \mid \text{query}) = \text{idf}(\text{gene}) \sum_{\text{doc in corpus}} \text{gda}(\text{gene}, \text{doc}) \text{doc}_{\text{score}}$$

[5] see http://www.disgenet.org/web/DisGeNET/menu/dbinfo#gdascore for a full description of the score.

[6] see http://www.findzebra.com/about for a description of the API.

where gda(gene, doc) is the gene-disease association score between the gene and the disease described by the document. The genes are then sorted by score and displayed in the *gene view* (see Figure 2.2).

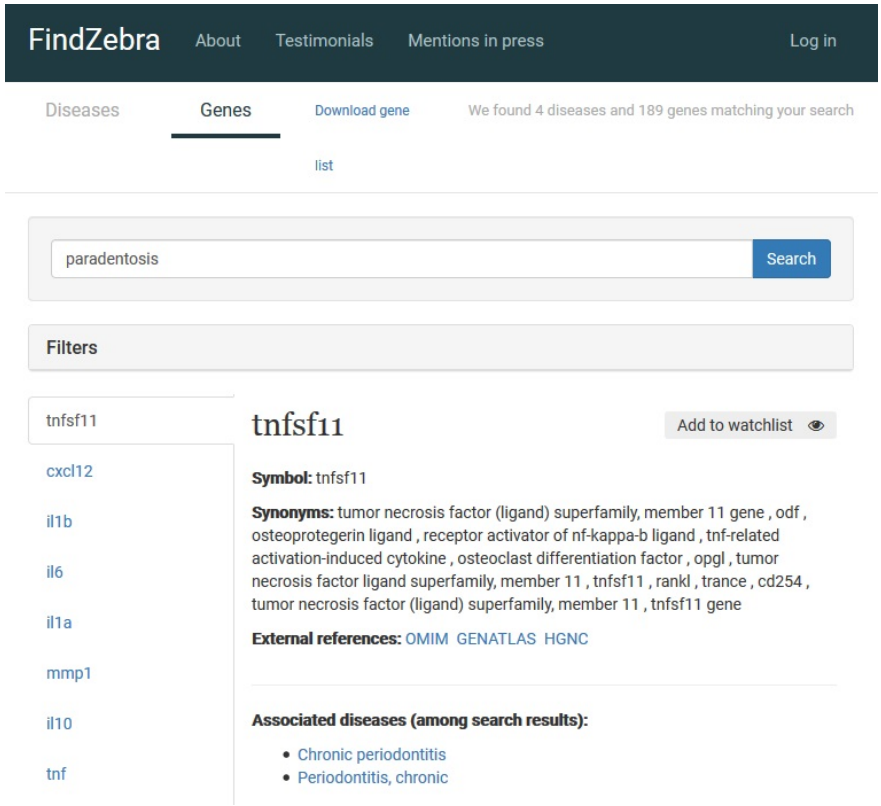


Figure 2.2: The FindZebra website. The figure shows the *gene view* of the search results

2.3 The FindZebra corpus

The FindZebra corpus was constructed by downloading 21.4k articles on diseases from the 6 datasources listed in Table 2.1. This resulted in 21.4k articles on 12.5k diseases. These articles are the ones used by the current version of the website. This dataset is called the *full FindZebra corpus*.

From the full FindZebra corpus we selected the 14.3k articles on 8.3k diseases that could be mapped to ICD-10 codes (not all concepts can be mapped to ICD-10 codes using the UMLS database). These articles constitute the *baseline corpus* which can be seen in Table 2.1. The baseline corpus is of a very high quality and is the part of the full corpus that we can map to other health system information (such as prevalence).

2.4 Validation and test sets

In order to estimate performance of different models in a realistic setting we use a dataset consisting of medical questions from a Jeopardy! like game called Doctor's dilemma featured by ACP[7]. The game is also known as Medical Jeopardy. The initial dataset from ACP consisted of 3000 questions. A large number of these questions were removed because they were either not related to diagnosis or required visual inspection of e.g. x-ray images. In addition to these Doctors Dilemma questions we used a set of 56 questions that had been constructed manually by a medical professional. In total we created 496 queries, of which half is used for validation and the rest is used for testing. The test and validation sets can be found on-line[8].

Source	Full corpus	Baseline corpus
GARD	1.917	1.34
Gene Reviews	638	321
GHR	1.075	738
OMIM	8.062	4.615
Orphanet	4.578	3.376
Wikipedia	5.154	3.870
Total	21.424	14.261

Table 2.1: Overview of the documents in the baseline/full FindZebra corpus. The full corpus cover 12.5k diseases in total and the baseline corpus cover 8.3k diseases.

[7] `https://www.acponline.org/membership/residents/competitions-awards/doctors-dilemmasm`

[8] Test set: `http://www.intellifind.dk/article/test_queries.csv`. Validation set: `http://www.intellifind.dk/article/valid_queries.csv`

2.5 Prevalence

Point prevalence is the perhaps most important type of disease metadata. It is measured as the percentage of the population suffering from a given disease at a given point in time. I.e. if we select a group of people at random, the point prevalence is the expected percentage of the group members suffering from the disease.

The importance of prevalence can be illustrated by an example. Let us assume that a patient is suffering from a set of symptoms S. He uses a search engine and is presented with two diseases D_1 and D_2. He can see that for both of the diseases his symptoms S are almost present (lets say that $P(\mathrm{S} \mid \mathrm{D}_1) = P(\mathrm{S} \mid \mathrm{D}_2) = 0.99$). He can also see that his symptoms are quite rare in general (lets say that $P(\mathrm{S} \mid \text{not } \mathrm{D}_1) = P(\mathrm{S} \mid \text{not } \mathrm{D}_2) = 0.01$)). Based on the symptom information alone, both diseases are equally probable. Without any further information we can assume that the prevalence of the disease is uniformly distributed between 0 and 1. By using Bayes formula and integrating out the prevalence we get that

$$\begin{aligned} P(\mathrm{D}_1 \mid \mathrm{S}) = P(\mathrm{D}_2 \mid \mathrm{S}) &= \frac{P(\mathrm{S} \mid \mathrm{D}_1)P(\mathrm{D}_1)}{P(\mathrm{S})} \\ &= \frac{P(\mathrm{S} \mid \mathrm{D}_1)P(\mathrm{D}_1)}{P(\mathrm{S} \mid \mathrm{D}_1)P(\mathrm{D}_1) + P(\mathrm{S} \mid \text{not } \mathrm{D}_1)P(\text{not } \mathrm{D}_1)} \\ &= \frac{P(\mathrm{S} \mid \mathrm{D}_1)\frac{1}{2}}{(P(\mathrm{S} \mid \mathrm{D}_1) + P(\mathrm{S} \mid \text{not } \mathrm{D}_1))\frac{1}{2}} \\ &= 0.99 \end{aligned}$$

This means that based on the symptom information alone, it is very probable that the patient is suffering from one or both of the diseases. However, let us furthermore assume that D_1 is rare, e.g. $P(\mathrm{D}_1) = 0.0001$. We wish to determine $P(\mathrm{D}_1 \mid \mathrm{S})$. According to Bayes formula we have that:

$$\begin{aligned} P(\mathrm{D}_1 \mid \mathrm{S}) &= \frac{P(\mathrm{S} \mid \mathrm{D}_1)P(\mathrm{D}_1)}{P(\mathrm{S})} \\ &= \frac{P(\mathrm{S} \mid \mathrm{D}_1)P(\mathrm{D}_1)}{P(\mathrm{S} \mid \mathrm{D}_1)P(\mathrm{D}_1) + P(\mathrm{S} \mid \text{not } \mathrm{D}_1)P(\text{not } \mathrm{D}_1)} \\ &= \frac{0.99 * 0.0001}{0.99 * 0.0001 + 0.001 * 0.9999} \\ &= 0.09 \end{aligned}$$

I.e. even considering the overwhelming amount of symptom "evidence", there is actually only a very small probability of 9% of the patient actually suffering from disease D_1. In Figure 2.3 we see a plot of what happens with the probability $P(D \mid S)$ as a function of prevalence. The point of the figure is to show that

the inclusion of prevalence information can completely change how probable we should consider a disease.

As mentioned in the introduction it is quite common for both professionals and non-professionals alike to use Google and other online tools for diagnosis. However, the results returned from such tools do not take into account the prevalence of a disease. This has the undesirable result that a user often finds a good symptom match with some rare disease and uses this for diagnosis, even though he has a very low probability of actually suffering from the disease.

Unfortunately, we have not been able to gain access to a suitable prevalence dataset. We did, however, spend a lot of time trying to obtain such a dataset, but as of yet our efforts has not been a complete success:

- We succeed in obtaining a dataset from Danmarks Statistik that gave the number of hospital admittance in a year grouped by SKS codes (the Danish equivalent of ICD10 codes). Unfortunately, this dataset was not exactly what we were looking for since it e.g. gave *influenza* a very low prevalence (since an influenza patient will typically not go to the hospital)
- We have tried obtaining prevalence data by making a partnerships with a large private hospital chain (work in progress).
- We entered into a research collaboration with Østerbroundersøgelsen[9] at Frederiksberg Hospital in order to gain access to a longitudinal study on diseases in the general population.

[9] Østerbrounderøsgelsen: https://www.frederiksberghospital.dk/afdelinger-og-klinikker/oesterbroundersoegelsen/om-undersoegelsen/Sider/default.aspx

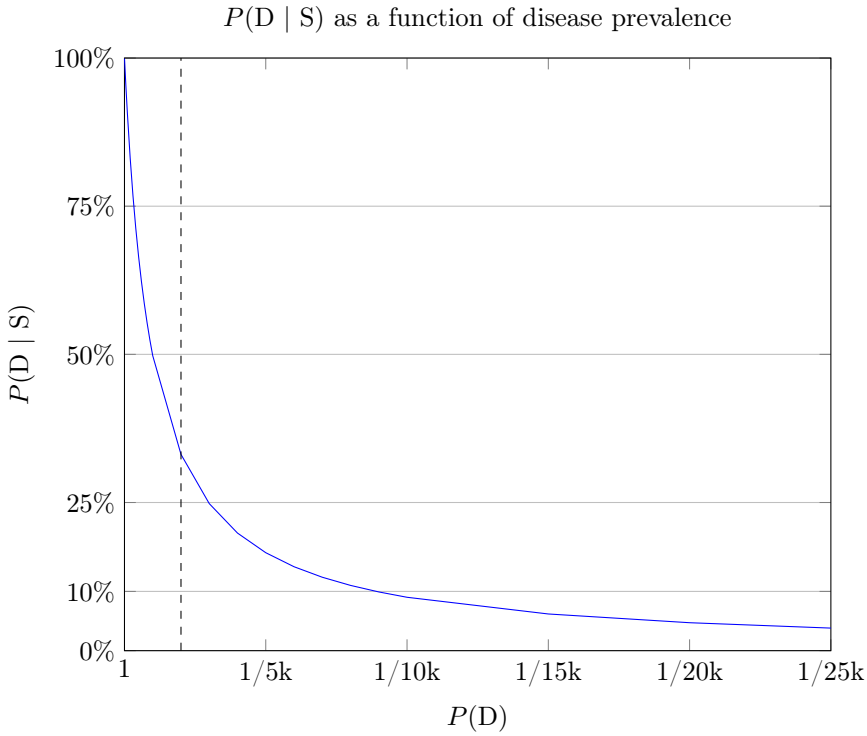


Figure 2.3: P(D | S) as a function of disease prevalence for the example in section 2.5. The prevalence to the right of the dashed line corresponds to rare diseases (prevalence less than 1/2000 in EU)

Chapter 3

Machine learning theory for NLP

In this section I will give a short introduction to the most important machine learning models and concepts used in the thesis. In section 3.1 I will describe the two types of neural networks used in the thesis and describe how the models can be trained. In section 3.2 I will describe what overfitting is and how it may be avoided. Note that the material presented in this section is by no means an exhaustive treatment of the topic, but it is sufficient for understanding the rest of the thesis.

3.1 Neural networks

There are many different types of neural networks. The two types used in this thesis are feed-forward networks and a recurrent neural networks (RNN).

3.1.1 Feed-forward network

An example of a simple feed-forward network is illustrated in Figure 3.1. It consists of three layers, an input layer, a hidden layer and an output layer. The

variables in the center of each node contain the output of the node, i.e. the output of the three layers is $x = [x_1, x_2, x_3, x_4]$, $h = [h_1, h_2, h_3]$ and $y = [y_1, y_2]$. Each of the hidden variables h_i is defined by an affine transformation of the outputs of the input layer, followed by a non-linear activation function. I.e. $h_i = f(b_i + xw_i)$ where w_i is a weight vector for node h_i and f is a non-linear transformation. b_i is called a bias. Typical examples of activation functions are hyperbolic tangent ($f(x) = \tanh(x)$)), sigmoid ($f(x) = 1/(1 + \exp(-x))$ and rectifier ($f(x) = \max(0, x)$).

We can of course write the calculation of h in vector notation as $h = f(b+xW)$. The exact same calculation is repeated between the hidden layer and the output. I.e.

$$\begin{aligned} y &= g(b_{\text{output}} + hW_{\text{output}}) \\ &= g(b_{\text{output}} + f(b + xW)W_{\text{output}}) \end{aligned}$$

We see that the output of one layer only depends on the previous layer output:

$$h_{\text{layer}} = g_{\text{layer}}(b_{\text{layer}} + h_{\text{layer-1}}W_{\text{layer}})$$

It is therefore straightforward to extend the simple example to a network with more than one hidden layer.

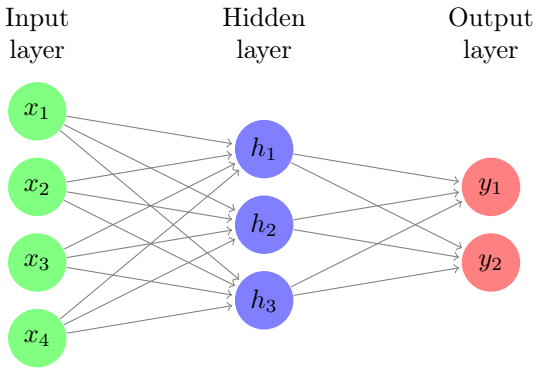


Figure 3.1: A simple feed-forward network.

3.1.2 Recurrent neural network

Recurrent neural networks (RNNs) are used for sequence data such as text. An example of a so-called unrolled visualization of a recurrent neural network is

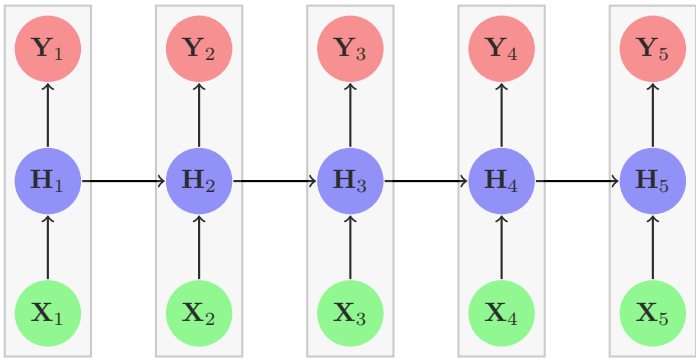


Figure 3.2: A simple recurrent neural network. Each of the filled rectangles correspond to a timestep.

shown in Figure 3.2. H_i is calculated by

$$H_i = f(X_i^T W + H_{i-1}^T U + b) \tag{3.1}$$

where f is an activation function, W, U are weight matrices and b is a bias term. I.e. H_i is almost calculated as for a feed-forward network, except that an extra term $H_{i-1}^T U$ has been added such that it depends on both the current input vector X_i and on the output H_{t-1} of the previous timestep. Note that parameters are shared across timesteps.

3.1.3 Training of neural networks

In order to train a neural network it is necessary to define a loss function. A loss function quantifies the cost associated with the output of the neural network, and the goal of training is to minimize this loss. The standard loss function for classification is the cross-entropy loss (CE loss) and the standard loss for regression is the mean squared loss (MSE loss):

$$CE(\theta) = -\frac{1}{|X|} \sum_{x \in X} p_x^t \log q_x \tag{3.2}$$

$$MSE(\theta) = \frac{1}{|X|} \sum_{x \in X} ||y_{\text{true}}^x - y_{\text{est}}^x||_2^2 \tag{3.3}$$

Here p_x, q_x denotes respectively the true and the estimated distribution for the training sample x. The distribution p is the distribution that puts all probability

mass on the correct label (i.e. a *one-hot representation*). y^x_{true} and y^x_{est} denotes the true and the estimated vector for the input x.

The training is typically performed using gradient descent where the gradient of the loss function w.r.t. the network parameters θ is found using *backpropagation*. Backppropagation sounds fancy, but it is actually just the chain rule of differentiation applied to the loss function. Typically we do not use the entire training set X for calculating the loss. Instead we only use a small batch of samples and use that to approximate the loss. This is called *stochastic gradient descent* (SGD).

The vanilla SGD algorithm updates the parameters by using the update rule

$$\theta \quad = \quad \theta - \eta \nabla L(\theta) \tag{3.4}$$

where η is called the *learning rate* and L is the loss function. There are several variants of the SGD algorithm. For training of the models described in this thesis I typically used Adaptive Moment Estimation Method (ADAM) [KB14]. When using ADAM updates an adaptive learning rate for each parameter is computed instead of using the same fixed learning rate for all parameters.

Simple recurrent neural networks often have problems with exploding and vanishing gradients. These problems are caused by the way gradients are calculated through backpropagation. In this thesis we avoid such problems by using Long Short Term Memory units (LSTM) [HS97] or Gated Recurrent Units (GRU) [CGCB14]. The difference between the simple recurrent networks and these networks is that the GRU/LSTM networks replace the calculation of the hidden nodes by a more complex calculation. This calculation is better suited to capture long-range dependencies and to avoid exploding/vanishing gradients. The details of how LSTM/GRU units are implemented will not be described here.

3.2 Regularization

Loosely speaking, overfitting is what happens when a model learns a multitude of specific details about the training data instead of some general patterns. Suppose, for example, that we want to train a network for text classification. Such a network might notice small details such that the exact phrase "and, but not exclusively" only occurs in 1 document. In each document we might have one or more of such unique phrases and the network can learn to recognize these if it has a sufficient number of parameters. However, such a network will generalize poorly to new examples. In order to prevent the model from overfitting we use

regularization. Two standard methods for regularization will be described in the following.

3.2.1 Dropout

Dropout [SHK+14] is a method for neural network regularization where units are dropped from the network with a certain probability. The method is very effective for preventing overfitting and has become a standard regularization method for neural networks. In Figure 3.3 we see an example where dropout has been applied to the hidden layer of a simple feed-forward network. We see that (for this batch) h_2 has been dropped out, effectively resulting in a network with only two hidden units.

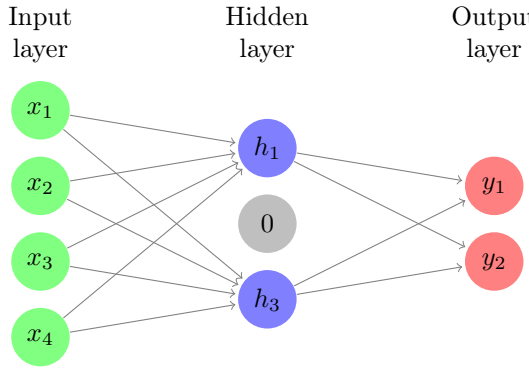


Figure 3.3: Example of using dropout during training. At each batch update we train a "new" network where we have randomly deleted some of the nodes. The figure shows the network in Figure 3.1 with dropout in the hidden layer.

3.2.2 L_p regularization

L_p regularization is a regularization method where there is a loss penalty for using large/many weights. It is quite easy to implement L_p regularization for subset W of the parameters θ. We simply modify the loss function such that

$$\hat{L}(\theta) \quad = \quad L(\theta) + \gamma ||W||_{L_p}^p \tag{3.5}$$

Typically, $p = 1$ or $p = 2$ are used. We can also use a combination of e.g. L_1 and L_2 regularization in which case the method is called *elastic net regularization*.

Chapter 4

Information retrieval optimization

The following chapter is based on the article *Performance Optimization for Specialized Domain Information Retrieval* in Appendix A. It contains a commented summary of the main results of the article.

4.1 Introduction

FindZebra is primarily a tool for information retrieval. A lot of my research has therefore focused on improving retrieval performance. A wide range of models and methods have been researched and the article gives a summary of the main findings. The article uses FindZebra as a case study, but the described methods are quite general and can be used for optimizing performance in other specialized domain search engines.

The retrieval optimization problem was approached from three directions:

1. **Improved retrieval models and ensembles**. We explored several different types of neural networks, both on word level and character level. We

furthermore investigated two methods for combining the trained models in an ensemble.

2. **Corpus expansion**. The corpus expansion was performed in two ways. The first was through static expansion and the second was by using a simple algorithm for inclusion of a collection of case studies.

3. **Use of structured data sources**. We tried improving performance by introducing synonym information into the models by using the UMLS database.

We compared our methods with Apache Solr, which is a state-of-the-art open software information retrieval system. We showed that using a combination of the methods above it is possible to obtain an absolute increase in performance of more than 13%.

Each of the directions for performance optimization will be described briefly in the following sections.

4.2 Improved retrieval models and ensembles

For each dataset we trained and evaluated performance for five models: a neural network bag-of-words model, a neural network bag-of-ngrams model, a word level recurrent neural net, a character recurrent neural net and a tf-idf based Apache Solr model. We also experimented with using a combination of models in an ensemble. The models were combined using two different ensemble methods: one using a soft voting scheme and one using "merge voting". By using a soft voting ensemble consisting of two neural BOW models where one is trained using synonym augmentation and one without we obtained a recall@20 performance of 81.85%. This is 4.4% better than the best single model and 13.3% better than the baseline model on the baseline corpus.

A medical query is often complex with a lot domain specific words. Such a query can be difficult to spell, even for a medical professional. Therefore it would be valuable to have a model that is robust towards spelling errors. Character based models have the advantage of being very robust to spelling errors since they have the ability to smooth over spelling mistakes. Figure 4.1 shows how misspellings affect the performance of a character based model compared to two word based models. We see that the character based model is much less affected by spelling errors compared to the word-based models. However, the general performance of the character based models is far below that of the neural BOW models and

therefore the spelling error robustness of the character based model does not give a real advantage in practice (unless we expect most of the users of FindZebra to suffer from dyslexia).

4.3 Corpus expansion

Most of the diseases in the baseline corpus are described by only a single document. The primary problem with having so few articles on each disease is that we have no way of determining word importance except by using word frequencies (i.e. a rare word is more discriminative than a common word). Even though a machine learning model will be able to extract some information from the baseline corpus, it is necessary to expand the corpus in order to be able to fully exploit the capabilities of machine learning methods.

The first corpus expansion method was by *static expansion*. We used the UMLS database in conjunction with the baseline corpus and a general web search engine to discover clusters of disease information on the internet. Then we manually selected the largest clusters and developed regular expressions that could extract disease information from different parts of each page in a cluster. Using this approach we were able to increase the number of documents from 14.3k to 143.7k articles and the number of diseases from 8.3k to 12.5k. As can be seen in Table 4.1, this additional information benefits all models substantially and it causes the performance of the neural BOW models to exceed the performance of the Solr model by several percent.

The second method for augmenting the corpus was by incorporating medical case studies. We extracted 51k case studies from the PubMed Central Open Access dataset[1]. In order for our search engine to be able to use these articles, they had to be labeled somehow. Unfortunately, these case studies are not labeled with information on the identity of the disease and, to make things worse, a case study does not even always only describe a single disease. For example, it is a bit unclear if the case study[2] with the title *Treatment of Ipilimumab Induced Graves' Disease in a Patient with Metastatic Melanoma* is about Graves' disease or about metastatic melanoma. This ambiguity problem is solved by including the same case study several times but with different labels (one for each of the possible diseases). E.g. in the example above we would include a copy of the case study with the label *Graves' disease* and a copy with the label *metastatic melanoma*. Even though this method is quite simple, it gave quite a large performance increase (see Table 4.1).

[1] https://www.ncbi.nlm.nih.gov/pmc/tools/openftlist/
[2] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4737013/

Note that it would actually have been quite easy to employ semi-supervised techniques such as label propagation [ZG02] or co-training [BM98] instead of using a non-parametric classifier as we did. However, the article already spanned quite a lot of topics and we decided to keep it simple and focus on the other aspects of the article.

4.4 Synonym injection

By using the UMLS database we compiled a list of 6200 symptom words corresponding to 2092 distinct symptoms. The list was used for teaching the models synonym relationships between symptoms. For the Solr model this is done by normalizing both the corpus and queries to the same canonical form. For the neural models we replace a symptom with one of its synonyms with a certain probability (during training).

Surprisingly, the use of synonyms only gave a very small performance increase of 0-1%. For the neural models, part of the explanation can be deduced from Figure 4.2. In the figure we see representations of synonymous words in a neural BOW model trained without synonym injection (top) and with synonym injection (bottom). It is quite clear that the model learns synonymous relations even without being explicitly told. Thus we can expect a neural model to benefit less from such forms of structured data. Note that our method for synonym injection actually forces the representation of words in our synonym list to have the exactly the same representation (even though e.g. “weak” and “weakness” cannot strictly be considered synonyms since one is a verb and the other is a noun).

4.5 Concluding remarks

The article mostly present what did eventually work. However, I also believe that it is quite interesting to know what did *not* work. First of all, we have experimented a lot with different neural models such as convolutional neural networks (both on character level and word level) and convolutional neural networks and recurrent neural networks in combination. However, the more sophisticated the model, the worse the performance seemed to become. However, I strongly believe that this is caused by the lack of data and that once the corpus has been expanded suitably, such methods will surpass the BOW models by a wide margin.

Table 4.1: Recall@20 [%] for the different models. The best results for each dataset is bolded.

	Baseline	Extended	Pubmed
Neural BOW (without synonyms)	62.90	70.97	**77.42**
Neural BOW (with synonyms)	63.31	**71.37**	**77.42**
Neural n-gram	48.79	60.48	64.52
Word LSTM	46.97	58.06	66.94
Char LSTM	46.37	64.11	63.71
Solr (with synonyms)	**68.55**	69.76	76.41
Solr (without synonyms)	**68.55**	68.75	75.81

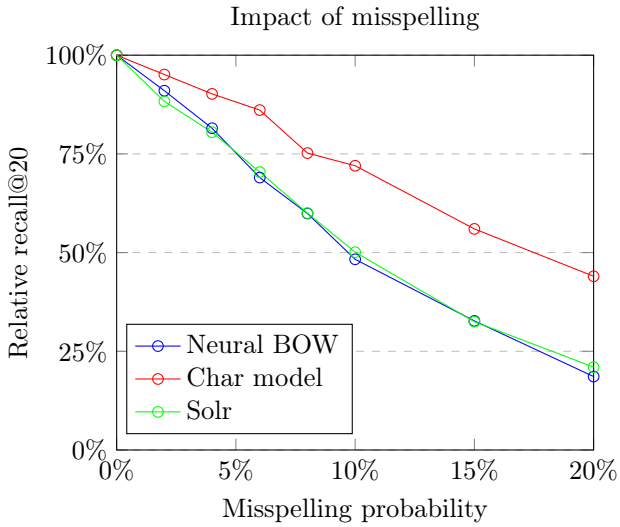


Figure 4.1: Impact of misspelling errors on the different models. The figure shows the recall@20 degradation as a function of misspelling probability for two word based models and one character based model. The models have been trained on the Pubmed dataset and the numbers are relative to model recall@20 when there are no misspellings.

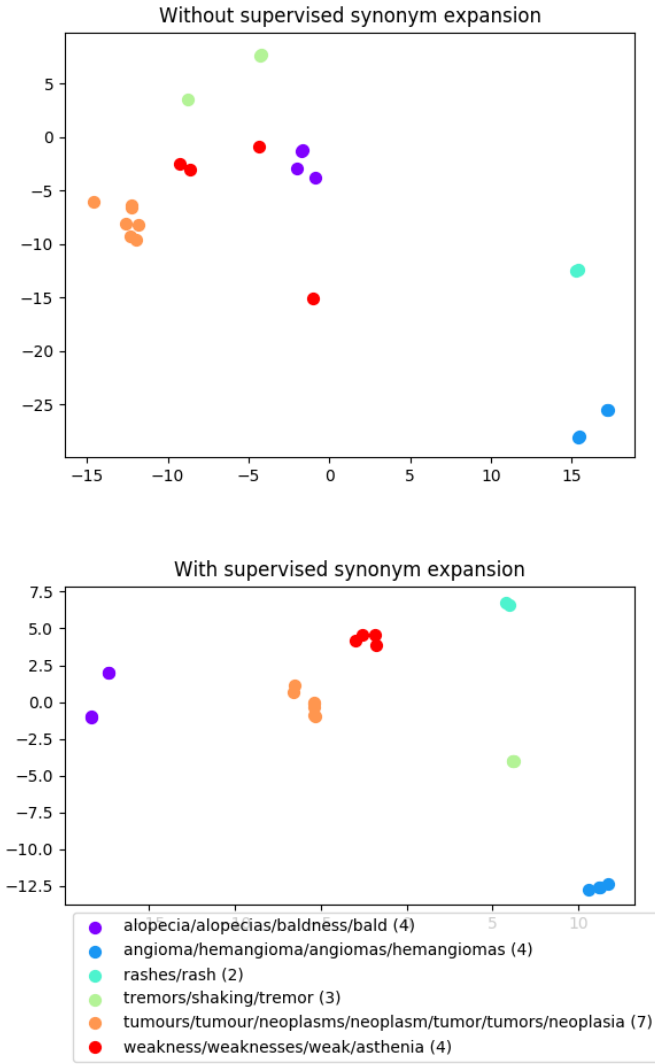


Figure 4.2: T-snee plots of representations of synonymous words. The upper plot shows the representation of words in a model without synonym injection, and the lower plot shows the representation of words in a model with synonym injection. The numbers after the word lists indicate the number of words in the list. The models used are BOW models trained on the PubMed dataset.

Chapter 5

Information completion

The following chapter is based on the article *Information completion for medical search assistance* in Appendix B. In section 5.1 I will give an introduction to the concept of information completion. In section 5.2 I will briefly describe how to train and measure performance of IC systems. In section 5.3 I will present two, perhaps better, methods of training IC systems that were not mentioned in the article. Finally, I will give some concluding remarks in section 5.4.

5.1 Introduction

An important property of a search engine is to be able to retrieve the most relevant documents based on a query. However, in order for the search engine to do this, it needs the query to be sufficiently informative. A query such as "abdominal pain" could for example correspond to hundreds of diseases, and based only on the query it is impossible for the search engine to know which ones are most relevant. I.e. some information seems to be missing from the query in order for the search engine to be able to retrieve relevant content. In a clinical setting a doctor would iteratively ask the patient questions based on a continuously changing differential diagnosis (see Figure 5.1). This process would continue until the doctor is confident enough to give a final diagnosis.

The purpose of Information Completion (IC) is to facilitate this dialogue following an unsuccessful search. More precisely, we define medical information completion as *the process of asking a user relevant questions with the express purpose of filling out missing or incomplete information that could be important for the diagnosis.* This additional information could be in the form of test results (e.g. blood pressure), other diseases that the user might have (e.g. congenital abnormalities), or symptoms that the user has not yet supplied to the search engine (e.g. nausea). In the following I will refer jointly to these different types of additional information as "symptoms"

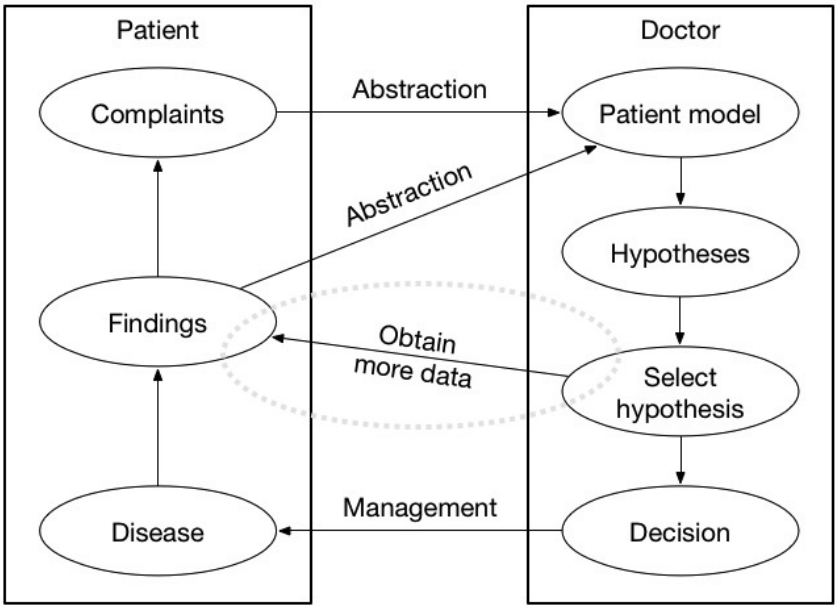


Figure 5.1: The diagnostic process, from [Wya91]. A disease causes the patient to exhibit some symptoms (findings). The patient tells the doctor about some or all of the symptoms and based on these complaints and the symptoms/findings of the patient, the doctor constructs a patient model. Based on the model, the doctor first compiles a differential hypothesis and then selects a hypothesis. This hypothesis will either be choosen as the diagnose or the doctor may query the patient for more information in the form of additional symptoms or run additional tests. Medical information completion corresponds to this part of the diagnostic process (within the ellipsis in the figure).

5.2 Article summary

The main obstacle for training an IC model is the lack of a suitable dataset. Ideally, we would have a data set consisting of a large number of pairs of the form `(search query, missing information)` where the missing information could be a symptom or a test result. Unfortunately such a data set does not exist. Instead we first defined a list of 2141 target symptoms (by using the UMLS database). Based on this list we constructed a dataset by extracting a large number of word sequences with at least 1 symptom in the target symptom list. From each of these sentences we deleted the target symptom and created a sample (`sentence minus symptom`, `symptom`) for each target symptom in the sentence. Using this approach we created a dataset consisting of 4.5M samples. A few examples of such sentences are listed in Table 5.1.

Removed symptom	**Training set sample**
neoplasms	,this has returned different genetic alterations (data not shown) cdc are generally considered to be aggressive ~~neoplasms~~
hypophosphatemia	of 11 years , at which time laboratory data revealed ~~hypophosphatemia~~ , elevated vitamin d levels , and hypercalciuria a
diarrhea	infancy two forms are recognized : early-onset mvid with ~~diarrhea~~ beginning in the neonatal period, and late-onset, with

Table 5.1: Some samples from the training set. The removed symptom is crossed out. Note that a "symptom" in this context can mean both an actual symptom, a test result or a disease.

As can be seen in Table 5.1, the samples in the training set has very little resemblance with what we would normally consider a medical query. A performance estimate based on such sentences would therefore not be very realistic. Instead we measured how well the IC system can predict withheld symptoms from real multi-symptom diagnosis queries (Doctors Dilemma questions). I.e. given a diagnosis query with a withheld target symptom we estimate the probability of having the withheld symptom in a list of n symptoms (recall@n). Note that this is a quite conservative measure because it ignores the possibility of the proposed symptoms being *related* to the exact match. For example, when withholding the symptom "high fever" from the query *a high fever, runny nose, sore throat, muscle pains, headache, coughing, and feeling tired*, we get the proposals [`fatigue, chills, dizziness, fever, nausea`]. Considering that the query is a query for influenza, all of these proposals are probably relevant, but none of them is the actual withheld symptom.

We trained several models such as a neural bag of words model, a neural n-gram model, a neural bag of UMLS terms and a recurrent neural network. The recall@n performance of the different models as a function of n is shown in Figure 5.2. We can loosely translate the results to a clinical setting: If a patient queries the neural BOW IC system and retrieves a list of 10 questions/symptoms, there is more than 35% probability of uncovering one or more pieces of missing information. We believe that such a level of performance is sufficiently high for the system to be usable in practice (especially considering that the performance estimate is actually only a loose lower bound of the actual performance, as discussed above).

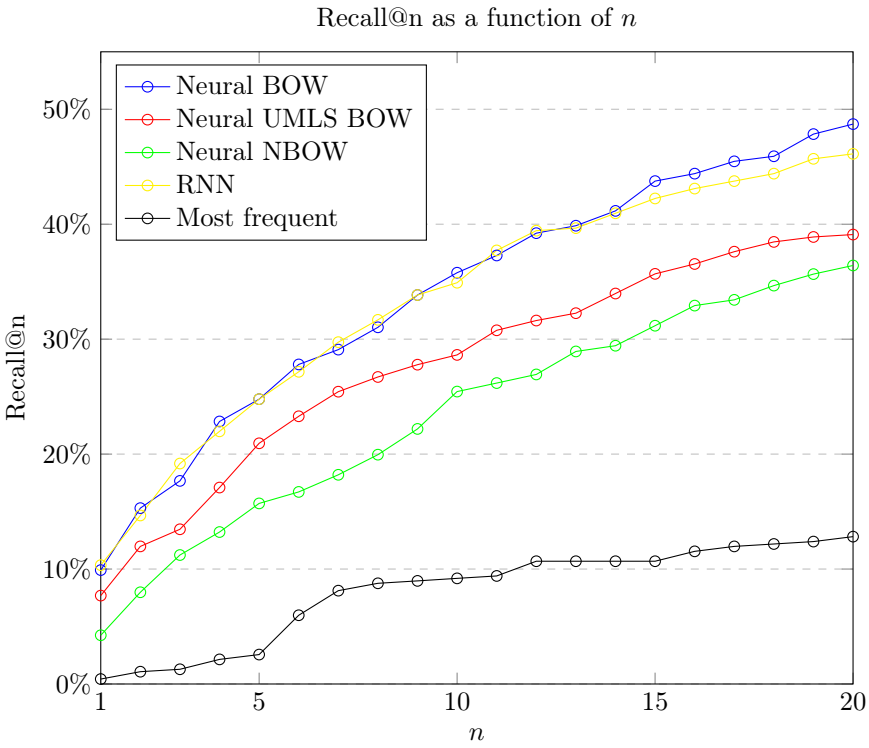


Figure 5.2: Recall@n for the different models as a function of n.

5.3 Alternate methods for training an IC system

In the article we described how to propose new symptoms that might be missing from a query. However, the effect of the proposed symptoms on the diagnostic

process was not taken into consideration. I.e. two different symptoms will not have the same diagnostic value, e.g. "runny nose" might be a more discriminative feature compared to "headache". A simple way of ordering the symptoms could be to multiply the probability of each symptom by a factor indicating how discriminative it is. For example, we could use the inverse document frequency

$$\text{idf(symptom)} = 1 + \log \frac{\#\text{docs}}{1 + \#\text{docs containing symptom}}$$

and order the symptoms according to $\text{idf(symptom)} \cdot p(\text{symptom}|\text{query})$, where p denotes the distribution from the IC system.

Another possibility for training an IC system is the following.

1. Train an IC system as described in the article.
2. Train a neural retrieval model.
3. Extract a set of random sentences V from a corpus of labeled articles. For each sentence $v \in V$
   (a) Determine the most plausible N "missing" symptoms $s_1, \ldots, s_N$ using the IC system.
   (b) For each symptom s_i determine the Δ cross-entropy:

   $$\Delta(s_i, v) = \max(H(p, q|v) - H(p, q|v + s_i), 0)$$

   Here $H(p, q|v)$ denotes the cross-entropy between the "true" (label) distribution p of diseases and the distribution estimate q from the trained retrieval model. $\Delta(s_i, v)$ for all other symptoms is set to e.g. -1.
4. Train a new IC system Q by using the loss function

   $$\begin{aligned} L &= \frac{1}{|V|} \sum_{v \in V} \mathbb{E}_{Q(s|v)} \Delta(s, v) \\ &= \frac{1}{|V|} \sum_{v \in V} Q(\cdot|v)^T \Delta(\cdot, v) \end{aligned}$$

   Here $Q(s|v)$ denotes the probability of symptom s given the query v.

Using such a method we can start to quantify the *diagnostic usefulness* of the IC system in terms of the expected cross-entropy reduction from a symptom proposal. This will allow us to define e.g. a cut-of point such that the user is never proposed a symptom if it is only of marginal importance to the diagnostic process.

5.4 Concluding remarks

In the article we explain how to train a system for medical information completion using just unstructured data (and a list of target symptoms). However, the same idea could be applied to *any* search engine in order to fill in missing information following an unsuccessful search. I believe that IC performance is almost as important as retrieval performance, especially in domains such as medicine where it can be very difficult to formulate a good query.

I believe that the performance of the IC system presented in the article is sufficiently high in order to be of practical use. Note that it would be quite easy to implement a user interface for the system by using e.g. *filters* (see chapter 2).

Chapter 6

Hash Embeddings

The following chapter is based on the article *Hash Embeddings for Efficient Word Representations* in Appendix C. I will try to explain the motivation behind hash embeddings and describe the advantages of using them. For a full description and analysis of hash embeddings, please see the full article text in the appendix.

6.1 Motivation behind Hash Embeddings

A neural network is almost always trained using some kind of gradient based method. This means that we need to transform all discrete inputs (such as words) into continuous vectors. This transformation is typically implemented by creating a mapping I from the set of discrete inputs $\mathcal{T}$ to the set $\{1, \ldots, B\}$ of integers, and subsequently use the mapping to lookup a (row) vector in a $B \times d$ matrix E. I.e. we represent a discrete token w by the continuous vector $E[I(w), :] \in \mathbb{R}^n$. Each entry in E is a model parameter that can be updated during training in order to give the best possible representation of each discrete token.

The index mapping I is typically created before training by enumerating the tokens in a vocabulary. We will call an embedding created in this way a *standard*

embedding and we call the enumeration a *dictionary*. In some situations, such as in online learning, the need for creating a dictionary can be a nuisance. In such cases we can use *feature hashing*. When using feature hashing we define I to be a random hash function that maps a token to a number (called a "bucket") in $\{1, \ldots, B\}$. This typically results in some tokens "colliding" with each other because they are assigned to the same bucket. The smaller we make B, the more collisions we will have. When multiple tokens $w_1, \ldots, w_n$ collide, they will get the same vector representation $E[I(w_1), :]$ which prevents the model from distinguishing between them. Even though some information is lost when tokens collide, the feature hashing method often works surprisingly well in practice [WDA$^+$09, JGBM16].

The initial idea behind hash embeddings was to improve the feature hashing idea. The goal was to create a hash function such that

1. all "unimportant" words were somehow assigned to a "zero bucket" if they happened to collide with an important word.
2. no "important" words should collide with each other.

What constitutes an important word depends on the purpose of the model. It is therefore a requirement that we are able to train the hash function jointly with the rest of the model. Unfortunately we cannot train a hash function using a gradient based method since it has values in a discrete set. Instead we introduce a new trainable parameter p_w for each word and represent w by $p_w E[I(w), :]$. This will allow us to (approximately) obtain property 1 above: If we have a collision between an important word w_i and unimportant word w_u, the model can set $p_i = 1, p_u = 0$. This will effectively assign w_u to a "zero bucket". This is the reason why we have given the p_w parameters the name *importance parameters*.

The second property above can be obtained by using more hash functions: Instead of using just one hash function I we can use e.g. two hash functions I_1, I_2 and represent w by $p^1_w E[I_1(w), :] + p^2_w E[I_2(w), :]$. If w_1 and w_2 are both important and $I_1(w_1) = I_1(w_2)$ we can set $p^2_{w_1} = p^1_{w_2} = 0$ and $p^1_{w_1} = p^2_{w_2} = 1$. This would mean that w_1 would be represented by $E[I_1(w_1), :]$ and w_2 would be represented by $E[I_2(w_1), :]$ which would avoid the collision. Of course, this can be extended to an arbitrary number of hash functions. However, our experiments have shown that typically nothing is gained by using more than two hash functions.

Note that we can use B (the number of rows in E) as a regularization parameter and that we can interpret B as the number of important/discriminative words

in the vocabulary. This interpretation is more intuitive than when using e.g. L_1/L_2 regularization. Also note that when using L_1/L_2 regularization, we first add "too many" parameters and then we set some of them to zero, which is a bit wasteful. This is avoided when using hash embeddings since the parameters do not have to be added in the first place.

In order to use hash embeddings (with two hash functions) we need $2|\mathcal{T}|+|E| = 2|\mathcal{T}|+Bd$ parameters. In a standard embedding we need $|\mathcal{T}|d$ parameters. Note, however, that we only need as many rows B in E as there are "discriminative words" in the vocabulary. This means that we can often choose $B \ll |\mathcal{T}|$ without affecting performance, resulting in a huge reduction in parameters.

From the description above we see that the computational overhead of using hash embeddings compared to a standard embedding is just a matrix multiplication of a 1×2 matrix (importance parameters) with a $2 \times d$ matrix (rows of E). In practice the overhead is not noticeable since the embedding layer is responsible for only a tiny fraction of the computational complexity of most models. Therefore using hash embeddings instead of regular embeddings does not make any real difference in terms of training time.

In the article we generalize the idea above slightly by introducing a two-layer hashing scheme where a token is first hashed to a large "vocabulary" space $1, \dots K$ by a hash function h_{layer1}. The number of collisions in this large space is typically negligible. Each of these "word" numbers is treated exactly as a regular word would be (i.e. each number has two associated importance parameters). The reason for this two layer approach is that it eliminates the need of having to create a dictionary. However, the price for this is $2T$ additional parameters. The first layer hash function has the property that $h_{\text{layer1}}(i) = i$ for $i < T$ which means that we can still use hash embeddings with a dictionary, if desired. This is done by simply letting $T = |\mathcal{T}|$ and by using the word dictionary indices as input to the hash embedding.

6.2 Construction of Hash Embedding vectors

A hash vector representation for a token $w \in \mathcal{T}$ is constructed in two steps. The steps are also illustrated in Figure 6.1:

1. Use k different functions $\mathcal{H}_1, \dots, \mathcal{H}_k$ to choose k *component vectors* for the token w from a predefined pool of B shared component vectors
2. Combine the chosen component vectors from step 1 as a weighted sum:

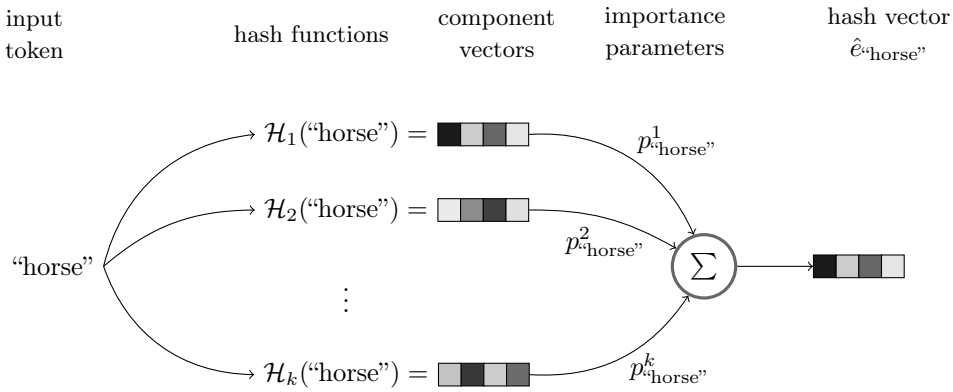


Figure 6.1: Illustration of how to build the hash vector for the word “horse”. The size of component vectors in the illustration is $d = 4$ and we use k hash functions.

$\hat{e}_w = \sum_{i=1}^{k} p_w^i \mathcal{H}_i(w)$. $p_w = (p_w^1, \ldots, p_w^k)^\top \in \mathbb{R}^k$ are called the *importance parameters* for w.

6.3 Summary of article results

In the article we evaluate hash embeddings on 7 different datasets for various text classification tasks including topic classification, sentiment analysis, and news categorization. We show how to use hash embeddings both with and without a dictionary, and compare hash embeddings to feature hashing. We also show that we can improve model performance (without increasing the number of parameters) if we use an ensemble of hash embedding models instead of a single model with standard embeddings.

One of our main article results is the comparison between feature hashing and hash embeddings. The feature hashing model uses 10^7 buckets and an embedding size of $d = 20$. The hash embedding also uses 10^7 buckets in the first layer and $1M$ component vectors of dimension $d = 20$. This means that the embedding layer used for feature hashing uses $200M$ parameters, while the hash embedding only uses $40M$ parameters. The results of the comparison can be seen in Table 6.1. The results shows that there is a clear advantage of using hash embeddings.

The classification models used in the experiments are chosen to be as simple as

possible: The document representation is obtained by simply adding the word vectors of the document. This representation is then passed through a single dense layer with a softmax activation.

Table 6.1: Test accuracy (in %) for the selected datasets

	Hash embedding	Feature hashing
AG	**92.4**	92.0
Amazon full	**60.0**	58.3
Dbpedia	98.5	**98.6**
Yahoo	**72.3**	**72.3**
Yelp full	**63.8**	62.6
Amazon pol	**94.4**	94.2
Yelp pol	**95.9**	95.5

6.4 Concluding remarks

Besides the application described in the article, hash embeddings have been used in various other settings with success. In the article *Zero-Shot Cross Language Text Classification* (discussed in the next chapter) for example, hash embeddings are used for representing the union of the vocabularies in English, German, French and Italian. In the article it is found that the best results are obtained using an embedding size of 1.500, which would have required more than a billion parameters using standard embeddings. However, when using hash embeddings with 25K buckets and two hash functions, we need less than 60M parameters. Also, in the article *Performance Optimization for Specialized Domain Information Retrieval* (discussed in chapter 4) hash embeddings are used in all word-based information retrieval models.

Finally we note that hash embeddings can be used to represent all kinds of discrete objects, not just words. I strongly believe that there may be other areas than NLP where hash embeddings will prove useful.

Chapter 7

Multilingual text classification

The following chapter is based on the article *Zero-Shot Cross language Text Classification* in Appendix D. I will explain the motivation behind the article in section 7.1. In section 7.2 I will give a summary of the main findings of the article. In 7.3 I will give some final remarks.

7.1 Motivation

Users not having English as their primary language often have difficulty expressing complex medical queries in English. Unfortunately, finding a suitable corpus on which to base a native language search engine is difficult since medical literature is almost completely dominated by literature in English.

Our goal with the model described in the article was to construct a search engine where a user could write a query in his or her native language. Based on this query the search engine should give a suggestion for a differential diagnosis based on *all* the information contained in a multilingual corpus. By adding all available data in English, French, Italian, Spanish, etc. we hoped to be able to even increase the performance of the original English search engine.

In the article we show how to create an effective native language classifier that can be trained without any labeled native language samples (i.e. a zero-shot classifier). However, when adding additional languages beyond the first, performance does not increase as we had initially hoped. I am, however, still quite confident that it can be done by changing the sampling method. Unfortunately I have not had time for looking more into this.

The intuition behind the idea was that instead of considering each language as separate, we can consider a language as a "dialect" of the same global language. I.e. we use an input vocabulary consisting of all words in all the included languages. We then first train an encoder to produce a language independent representation of a given text. This representation can in turn be used to train a single classifier using the entire multilingual corpus as training data.

Note that the approach presented in the article has several advantages compared to using a collection of "standard" native search engines. When using a collection of native search engines you would first have to determine the input language (either by estimation or by input from the user). Then the corresponding native search engine would have to be queried. Based on the query, *native* results can then be returned. When using the method presented in the article the query can be expressed in any language the encoder has been trained on (and the language does not have to be specified). Based on the query the search engine can return results in *all* languages (not only the native language). E.g. given a query in Danish, we can return matching medical case studies in English or French.

Of course, the problem of lack of a suitable native language dataset is not only relevant for medical literature. The model presented in the article is completely general, and therefore we used a typical classification task (classification of Wikipedia articles) as an example instead of specialized domain medical search.

7.2 Article summary

The cross language text classification model presented in the article consists of a universal encoder and a classification module. The purpose of the universal encoder is to transform a text into a language independent representation. This language independent representation can in turn be used by the classifier in order to create language independent classifier. The encoder is trained using the setup in the left side of Figure 7.1. I.e. we draw two random snippets of text from two different articles (and perhaps in different language) and train the encoder to be able to tell if the two snippets describe the same topic or not.

This distinction is made entirely based on the similarity between the encodings of the two snippets and this forces the model to give similar representations to texts about the same topic.

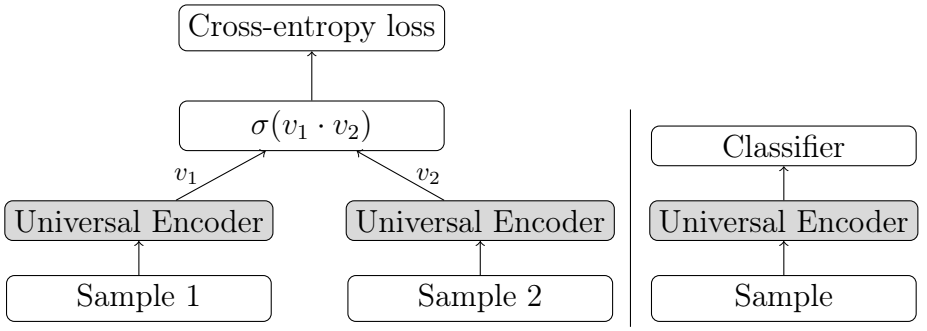


Figure 7.1: Illustration of how the *Universal Encoder* is trained (left) and how it is used when training and making predictions using the classifier (right). All *Universal Encoders* (gray background) are exactly identical by sharing architecture and weights

We compare three models:

1. A **pure native classifier** trained on Italian articles and tested on Italian articles. The performance of this model can be considered an upper bound for the performance.

2. A **machine translation model** trained on English Wikipedia articles and tested on Italian Wikipedia articles that have been translated to English using Google translate.

3. **Our model**. The *encoder* is trained on Italian plus a subset of English, German and French Wikipedia articles. The *classifier* is trained on a subset of English, German and French articles. I.e. the classifier has never seen an Italian article.

From the results of the experiments in the article we note several things. We see that the zero-shot classifier obtains very good results, especially in a bilingual setting where the largest classifier (embedding size 1500) obtains a very high accuracy of 78.5%. This is far better than the model based on machine translation (72.1%) and very close to the upper bound on accuracy of 80.4% (the native classifier). The performance unfortunately drops to 75.9% and 73.5% when using three and four languages, respectively. This is, however, still far better than the machine translation approach.

7.3 Concluding remarks

In the article we show how to create a language independent representation that subsequently can be used for e.g. classification. Note, however, that once we have trained such a language independent representation it can be used for any number of applications. For example, we could train a language independent representation on a corpus of medical texts and subsequently use the representation for training both an IC system and an information retrieval system.

Even though we obtain really good results in a bilingual setting, it is a bit unfortunate that the performance seems to decrease as we add more languages. I do, however, feel confident that this can be solved by using uniform sampling of language pairs, but I will have to leave this investigation to future work.

Two of our master thesis students, Nicolai Dahl and Christian Andersen, very recently finished compiling a multilingual corpus. I will leave the development of a multilingual medical search engine based on this corpus and the zero-shot classification model be a topic for future work.

Chapter 8

Discussion and conclusion

The work presented in this thesis is a selection of the primary findings and results from the past three years of my research. The purpose of the research was to improve FindZebra as a diagnosis decision support system. The results presented in this thesis describe substantial improvements of FindZebra in almost all parts of the diagnostic process.

However, for every success there have been many failures. Unfortunately, even though these failures can be very interesting for other researchers, they rarely end up in scientific publications. In this chapter I will elaborate on the context of each of the papers and briefly mention some of the things that did not work. I will also indicate where I think future research should focus.

Even though this thesis has been a start, there are still enough open research problems for at least one more thesis. It is my hope that others will finish what I have started.

Retrieval performance optimization

In the first contribution *Performance Optimization for Specialized Domain Information Retrieval* we showed that a performance improvement of more than

13% is possible through the use of improved retrieval models, corpus expansion, ensembles and inclusion of meta-data. This is a huge performance increase, but I believe that some improvements are still possible through corpus expansion and improved models.

Some queries are simply not informative enough for making a reliable diagnosis and therefore there is an upper limit to performance. Once this limit for performance has been reached, there is still room for improvement of performance through information completion. I.e. we could start to ask the question "how many questions must the information completion system pose to the patient on average in order for the retrieval system to have enough information for making the correct diagnosis?".

I believe that the most important improvement to search performance would be through the use of prevalence data. As we saw in Figure 2.3, prevalence information is extremely important. Also, in order to assess clinical performance, we should probably change the performance measure from recall@20 to a prevalence weighted recall@20.

Interestingly, this article was actually the last that that I finished, even though performance optimization has been a central research theme for all three years. During that time I tried a multitude of things such as

1. searching using different variants of Latent Dirichlet Allocation and other Bayesian Networks.
2. improving performance by including the UMLS hierarchy.
3. improving retrieval performance by using more complex neural networks both on a character level and on a word level. I tried a multitude of architectures that combined convolutions, recurrent networks, attention and other methods.

A lot of these ideas should probably be revisited once the corpus has been sufficiently expanded, since they have only been tested using the baseline corpus. Since the goal was to obtain better results than Solr, each of the methods above was discarded since the performance was always somewhat inferior to that of Solr. However, as described in the article, this performance superiority of Solr using the baseline corpus is hardly surprising since we typically only have 1-2 articles on each disease.

I had planned one more article about information retrieval optimization. In the article I would have shown how to include the case studies using more complex

semi-supervised methods. The same method would then have been used for including articles from the web. Hopefully, this expansion of the corpus would then result in a "critical mass" of information that would enable the use of more complex models (preferably character based RNNs).

Information completion

In the article *Information completion for medical search assistance* we showed that it is possible to train an information completion system using just an unlabeled corpus (and the UMLS database for defining symptom targets). We show that the performance of the trained system is sufficiently high to be usable in practice. However, I would have liked to implements and test the alternate IC model described in section 5.3 where only symptoms with the highest diagnostic value are proposed. But I leave this to future work.

I believe that IC systems will play an important role in the future health care system, both for reducing the risk of misdiagnosis and for saving time for medical professionals. In a hard pressed health care system the efforts of nurses and doctors should focus on caring for patients, not on asking standard questions and filling out questionnaires. In a stressed and hectic environment people sometimes make mistakes, even medical professionals. A patient complaining about for example fever and headache should always be asked about neck stiffness (which could be a sign of contagious meningitis). In case of neck stiffness the medical staff could also be made aware of the possibility of meningitis by using e.g. the information retrieval system described in this thesis. Such a system could perhaps have prevented several recent errors made by the Danish 1813 hotline (see for example https://www.regionh.dk/presse-og-nyt/pressemeddelelser-og-nyheder/Sider/Kritik-af-1813.aspx).

I also believe that IC systems will become much more common in general search engines. E.g. a search for "Tom Cruise" could for example result in the question "Are you looking for a film, an actor, or a person related to Tom Cruise?". For a huge search engine such as Google it would be trivial to define a couple of thousand categories and then train a classifier to predict the category of the selected page based on the query. This classifier could then be used in an IC system.

Hash Embeddings

In *Hash Embeddings for Efficient Word Representations* we showed that it is possible to improve one of the fundamental building blocks in NLP models, namely the representation of words. For almost two decades it has been standard practice to represent words and other discrete items by using a simple look-up table. Hash embeddings improve this method. They can be used wherever a standard embedding can be used, but provide an intuitive form of regularization and a reduction in the number of required parameters, especially for large vocabularies or large embedding sizes. The method is furthermore simple and easy to implement.

I consider hash embeddings to be the primary contribution of this thesis. Since I came up with the idea I have not used anything but hash embeddings in my own work. For example, the performance obtained in *Zero-Shot Cross language Text Classification* would not have been possible to achieve (at least not easily) without hash embeddings due to the the vast number of parameters required for word representations.

I feel confident that there are applications of hash embeddings besides representation of words. Astronomers, for example, might use hash embeddings to represent stars by giving similar representations to stars with similar properties. Even before releasing the final code for the hash embeddings we received numerous requests for access to the code by researches who did not work in the area of NLP. This makes me think that there probably is a lot of applications of hash embeddings that we did not think of and it will be interesting to see what these applications might be.

Multilingual search

In the article *Zero-Shot Cross language Text Classification* we show that it is possible to create an effective classifier for a task in a language L_t for which there exists no labeled dataset. The only pre-requisite is that there exists a *comparable* corpus in the languages L_t and L_s and that there exist a labeled dataset for the task in language L_s.

The idea for the article grew out of an actual demand. During the development of a chat system for a Danish private hospital chain we ran into problems caused by the lack of a suitable Danish medical corpus. The solution was to re-visit one of our old ideas for information retrieval where we pooled all languages and trained a retrieval model on top of this vocabulary union. Unfortunately,

this idea did not work since we just ended up with a “parallel” classifier for each language. The solution was to let the final part of the encoder be totally dependent on the similarity between representations and this gave us what we wanted, namely a language independent text representation.

Appendix A

Performance Optimization for Specialized Domain Information Retrieval

Performance Optimization for Specialized Domain Information Retrieval

Dan Svenstrup[1,*] **and Ole Winther**[1]

[1]DTU Compute, Danish Technical University, Lyngby, 2800, Denmark
[*]dsve@dtu.dk

ABSTRACT

In this article we explore several methods for improving retrieval performance for specialized domain search engines. As a case study we use a specialized search engine for rare diseases called FindZebra.
We approach the retrieval optimization problem from four directions. The first direction is by means of using improved retrieval models based on machine learning. The second direction is through the use of ensembles. The third direction is through corpus expansion. The fourth direction is by incorporating structured data sources.
Specialized domain search engines are often faced with the problem of a small amount of data. This aggravates the risk of over-fitting. We propose a regularization method that greatly reduces the risk of over-fitting. The method can be considered a very aggressive form of input dropout and enables use of complex models even when only a small amount of data is available.
We compare our methods with Apache Solr, which is a state-of-the-art open software information retrieval system. By using a combination of a neural word based model, corpus expansion, ensembles and synonym injection we show that an absolute performance increase of more than 13% can be obtained.

1 Introduction

Specialized domain search engines[1] are very common. For example, most companies and educational institutions have a custom search engine for their internal data collection. Each item in the data collection typically consists of a text and some meta-data which is often very specific to the domain. For example, a bookstore might have a description of each book along with its category, author, title and ISBN number. Or a medical search engine might have a unique identifier of the disease described by each document along with meta-data on common symptoms, relation to genes etc.

Open source software packages such as Apache Solr[1], Elastic Search[2] or Indri[3] offers easy access to creating specialized domain search engines. They are very fast, fault tolerant, easy to use, and typically provide good performance. However, they rely on very simple document retrieval models based on word frequencies and such retrieval models may not be optimal for all purposes. Consider, for example, the case where we have several documents describing aspects of the same object. In such a case there is information lost by treating each document as separate, as is e.g. done in a typical word based model such as the one used by Solr. To give a concrete example, we may have several articles describing the same disease in a medical corpus. Similarities between texts describing the same disease will give information about which features (e.g. symptoms) are considered important.

In this article we will explore several directions for improving retrieval performance for specialized domain search engines using the rare disease[2] search engine FindZebra[3] as a case study. Even though some of the methods used will be specific to FindZebra, the general ideas and findings will be applicable to other specialized search engines.

The rest of the article is organized as follows: We give an introduction to the FindZebra search engine in section 2. In section 3 we will describe the different models used in our proposed methods. In section 4 we will investigate two methods for corpus expansion, one by manual expansion rules and one by using a very simple semi-supervised algorithm for inclusion of case studies. In section 5 we will take a look at how to incorporate meta-data into the search engine. In section 6 we will present the results from using the described methods for optimization of retrieval performance. In section 7 we will take a look at some possible improvements and future directions for the methods. Finally, we conclude the article in section 8.

[1]Also called vertical search engines, see https://en.wikipedia.org/wiki/Vertical_search

[2]A rare disease is a disease that affect a small percentage of the population. There is no consensus on how small this percentage has to be for the disease to be considered rare. In the United States a rare disease is defined as a disease that affects less than 1 in 1500.

[3]See http://www.findzebra.com

2 The FindZebra search engine

FindZebra is primarily a search engine for rare diseases. It does, however, also contain articles on more common diseases such as influenza. It uses a Solr backend and the baseline corpus is based on a high quality corpus of medical texts consisting of 14.261 articles from the 6 sources shown in Table 1. Besides from this corpus, FindZebra uses a couple of databases on genes and disease-gene relationships. Theses databases will neither be described nor used in this article. The by far most important source of structured information used by FindZebra is the UMLS database.

The baseline FindZebra corpus	
Source	**#documents**
GARD	1.341
Gene Reviews	321
GHR	738
OMIM	4.615
Orphanet	3.376
Wikipedia	3.870
Total	14.261

Table 1. Overview of the documents in the baseline FindZebra corpus. The articles cover 8.280 diseases in total.

2.1 The UMLS database

The UMLS database is an ontology of medical concepts. It is currently maintained by the National Library of Medicine[4]. Each word/phrase in the database is associated with a concept id (CUI). For example *influenza*, *influenzas*, *flu*, *human influenza* etc. all share the same CUI since they are all conceptually equal. This relation between synonymous words/phrases is very useful and is used in several parts of FindZebra. In the following we describe how the UMLS database is used for symptom extraction and for mapping of articles to diseases IDs.

2.1.1 Mapping of symptoms using UMLS

Each concept in the UMLS database has an associated semantic type such as *Amino Acid Sequence* or *Congenital Abnormality*. This taxonomy makes it possible to identify symptom terms and synonyms of these. By extracting the concepts with the semantic types *Sign or Symptom* and *Finding* we compiled a list of 6200 symptom words corresponding to 2092 distinct symptoms. This symptom list is used various places in FindZebra. For example it is used for boosting symptom terms in queries, for symptom synonym detection in text and queries and for symptom extraction from articles. In this article though, we will only use the synonym list for teaching the models symptom synonym relationships (section 5).

2.1.2 Mapping of articles to disease IDs using UMLS

Each article in the corpus has a title consisting of the disease name. These disease names can be mapped to a CUI. However, just as a word such as "jaguar" can have different meanings depending on the context, so can UMLS concepts. In order to map an ambiguous article title, we use the concept with a semantic type of *Disease or Syndrome*. By using this mapping we can group articles describing the same disease in the search results and thereby change FindZebra from being a simple document search engine to a diagnostic hypothesis engine.

3 Models

3.1 Neural models

We compare three different types of neural model architecture. The architectures all consist of a text encoder module and a classification module. The classification module is the same for all three architectures and can be seen in Figure 1. It consists of three dense layers with 2000 units in each and relu activation. Between the layers we use batch normalization[4]. After the three layers we have a dense output layer with a softmax activation. The full architecture for for each of the models will be described separately in the following.

3.1.1 Neural BOW model

The architecture of the neural bag of words (BOW) model can be seen in Figure 2. It takes a set of words as input. These words are transformed to embedding vectors by using a hash embedding[5] with 35k buckets and two hash functions. The word vectors are subsequently summed in order to create a single vector representation of the text.

[4] See http://www.nlm.nih.gov/research/umls/

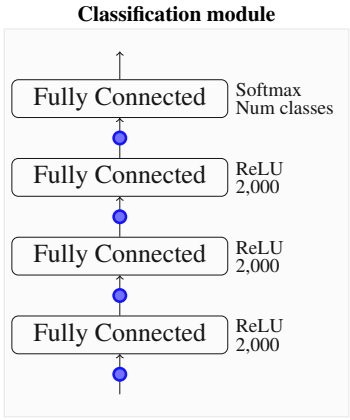


Figure 1. The architecture of the classification module (last four layers) of the neural models. The numbers to the right of each layer is the activation function of the layer (above) and the number of units (below). The blue dots represent batch normalization.

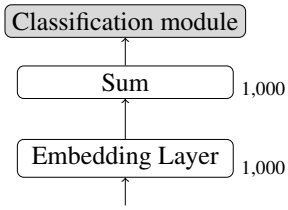


Figure 2. The architecture of the neural BOW network. The numbers to the right of each layer indicates the number of units in the layer.

A BOW model does not use the ordering of words in a sentence. This has the undesirable consequence that two sentences with a very different meaning such as

- *the patient suffered from a rash on the stomach and pain in the hand*
- *the patient suffered from pain in the stomach and a rash on the hand*

are considered as equal by a bag of words model. Note that not even very local sequence information is preserved, e.g. the BOW model does not consider "abdominal pain" to be a single concept but as two separate words.

It is, however, possible to incorporate *local* sequence information into a BOW model by using n-grams. In an n-gram model sequences of up to n words are used as input features instead of single words. For example, the first sentence above could be transformed to *the_patient suffered_from a_rash_on_the_stomach and pain_in_the_hand* in an n-gram model. In that way all essential sequence information can be preserved. However, it can be difficult to determine the best transformation of unigrams to n-grams. E.g. should the sentence *the patient suffered from* be transformed to *the_patient suffered_from* or to *the patient_suffered from*? There are two methods of transforming a sentence into a bag of n-grams. The first method is based on corpus statistics. It compares the frequency of e.g. the sequence *abdominal pain* with the frequencies of each of the words *abdominal* and *pain*. More precisely, it identifies n-grams by scoring each sequence of two (n-1)-grams by

$$\text{score}(w_i, w_j) = \frac{\#w_i w_j - \delta}{\#w_i \#w_j}$$

Here $\#w_i w_j$ denotes the number of occurrences of the bi-gram $w_i w_j$ and $\#w_i$, denotes the number of occurrences of the word w_i. If $\text{score}(w_i, w_j)$ is above a given threshold, we consider $w_i\_w_j$ to be an n-gram. The method can be applied iteratively in order to identify higher order n-grams (see[6] for details). The second method is simply to use all possible n-grams of a sentence as

input and let the model figure out which ones to use. We will use this second method for transforming a sentence into a bag of n-grams, for $n \leq 2$. The architecture of the neural bag of n-grams is the same as the architecture of the BOW model.

3.1.2 Word recurrent neural network

A recurrent neural network has the capability of understanding the sequence structure of a sentence and is able to relate different parts of the sentence to each other. A sentence such as *the patient denied that he had abdominal pain but not that he had a headache* would be impossible for a BOW/n-gram model to understand correctly, but it could be understood by a recurrent model. The architecture of the word based recurrent network is shown in Figure 3. The input layer is the same as for the BOW model, i.e. we use a hash embedding layer with 35k buckets and two hash functions. Following the embedding layer, the word vectors are feed through two bidirectional LSTM[7] layers. After each of the LSTM layers there is a linear transformation with a relu activation (applied at each timestep).

We tried several different variants of the architecture, including architectures with/without the time distributed layer, with one/two bidirectional layers, with/without activation after the sum layer and architectures with residual connections[8]. The architecture in fig 3 was, however, the architecture with the by far lowest validation error.

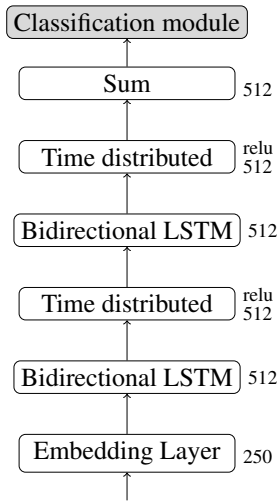


Figure 3. The architecture of the word based recurrent network. The numbers to the right of each layer denote the activation function of the layer (above) and the number of units (below). The outputs of the forward and backward LSTM layers are concatenated for each timestep and the number next to the LSTM layer indicate the total number of units in the two layers. The time distributed layer consists of a linear transformation applied a each timestep, followed by a relu activation.

3.2 Character recurrent neural network

Character level recurrent models are models that consider a text as a sequence of characters instead of a sequence of words. Character level models can be quite difficult and time consuming to train, but they also have several advantages:

1. they can easily handle all words (including for example proper names), and do not require a dictionary.
2. they can more easily learn similar representations for similarly spelled words (such as the same word in singularis and pluralis).
3. they are very robust to spelling errors since such models typically can smooth over spelling mistakes (see section 6.6).

A word based model, on the other hand, is very vulnerable to spelling errors since just a small spelling error will alter the information content of a word completely. This problem of word based models can be alleviated somewhat by using Levenshtein distance[9] to correct misspellings. Levenshtein distance is also sometimes called edit distance since the distance between two words in this metric is equal to the minimum number of single character edits required in order to turn one word into the other.

When encountering a word not in our dictionary, we can thus choose one of the closest words as a correction. However, such methods are far from perfect and typically the user will have to verify the correction.

The architecture of the character based model used in the experiments is show in Figure 4. It uses a standard embedding in the input layer. After the embedding layer we have two bidirectional LSTM layers with batch normalization in between. The characters used by the model consist of digits, letters, punctuation and whitespace letters.

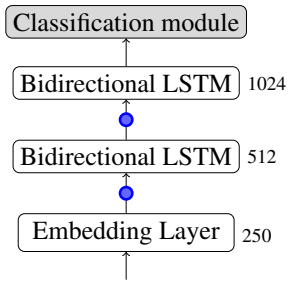


Figure 4. The architecture of the character based recurrent network. The blue dots represent batch normalization. The outputs of the forward and backward LSTM layers are concatenated for each time step. The number next to the LSTM layer indicate the total number of the units in the two layers. We use the output from the last time step as output for the final bidirectional LSTM layer.

3.3 Baseline retrieval model

The baseline retrieval model consists of an Apache Solr database with the default similarity scoring. This scoring algorithm computes the similarity between a query q and a document d. The scoring formula is given by

$$\begin{aligned}
\text{score}(q,d) &= L_d C_{q,d} \sum_{t \in q} \sqrt{\text{tf}(t,d)} \cdot \text{idf}(t)^2 B_t \\
t,d,q &= \text{term, document, query} \\
L_d &= \text{“higher score to shorter docs”} \\
C_{q,d} &= \text{fraction of q covered by d} \\
B_t &= \text{boost applied to term } t \\
\text{tf(t, d)} &= \#\text{occurrences of term t in d} \\
\text{idf(t)} &= 1 + \log \frac{\#\text{docs}}{1 + \#\text{docs containing t}}
\end{aligned}$$

Note that scores cannot be compared across queries. I.e. the fact that $\text{score}(q_1, d) > \text{score}(q_2, d)$ does not imply that q_1 matches d better than q_2. However, in order to be able to (approximately) compare scores between queries, Solr normalizes queries by a normalization factor. A more thorough description of how Solr ranks documents for a query can be found in[10].

Even though Solr uses a similarity scoring that is (relatively) simple, it is still a very strong baseline model that often performs surprisingly well in practice. In[11], for example, it is shown that FindZebra with a Solr backend is more than twice as likely to have the correct diagnosis in top 20 of the search results compared to other popular alternatives such as Google or PubMed.

3.4 Ensembles

The models described above are very different. There is especially a large difference between the frequency based Solr model and the neural models. Diversity is often considered key for obtaining performance improvements when using ensemble models[12]. We try two different types of ensembles:

1. **Majority voting (soft voting)** In soft voting each classifier C_i in the ensemble cast their vote in the form of a probability distribution $P_i = (p_1, \ldots, p_N)$ over the N target classes. We then use the element-wise average $P = \frac{1}{N} \sum_i P_i$ for ranking. Note that this method cannot be used with a Solr model in the ensemble since the ranking scores of Solr do not translate to probabilities in any meaningful way.

2. **Merge voting** When using this method each model iteratively proposes its most confident target prediction. I.e. we make a list by starting with the most confident prediction from each of the models, then adding the second most confident predictions, etc. When using this simple ensemble scheme, we can use any model that can be used for ranking (including a Solr model).

4 Corpus expansion

The baseline FindZebra corpus consists of 14.3k articles from the datasources listed in table 1. 62.7% of the 8280 diseases in the corpus are described by a single document, 20.5% by two documents, and only 16.9% are described by three documents or more. This means that the by far the largest part of this baseline corpus is described by only one or two documents. The primary problem with having so few articles on each disease is that we have no way of determining word importance except by using word frequencies (i.e. a rare word is more discriminative than a common word). However, a machine learning model will be able to use other kinds of important information contained in the corpus. For example, a machine learning model can learn information about synonyms. This is because the *meaning* of a word is determined by its context (the distributional hypothesis[13]), and such patterns are identifiable by a machine learning model. Such information can help increase generalization performance. We will investigate this property further in section 5. Thus even though our machine learning models will be able to extract some information from the baseline corpus, it is necessary to expand the corpus in order to be able to fully exploit the capabilities of machine learning methods.

In the following we will describe two methods for corpus expansion, one by static expansion and one using semi-supervised learning for inclusion of case studies.

4.1 Corpus bootstrapping by static expansion

This corpus expansion method bootstraps a new corpus by using the disease ids already present in the baseline corpus, a general search engine, the UMLS database and a collection of regular expressions. The approach was as follows:

1. Compile a list of diseases from the baseline FindZebra corpus and determine all known aliases for each disease name using the UMLS database.
2. For each of theses disease alias, search the internet using a general purpose search engine (we used DuckDuckGo) and save all search result *links*.
3. Sort the web domains by number of hits
4. Most of the domains in the top domains turned out to be disease databases such as Webmd or Disease Info Search, that are not in the baseline FindZebra corpus. We chose 12 of these disease databases.
5. For each domain, we created 2-4 regular expressions capable of extracting the core information from their disease pages. These regular expressions are not bulletproof and were constructed to include too much rather than too little.
6. For each of the saved links belonging to one of the selected domain, we then used one or more of the regular expressions to extract the relevant text. These new articles were then added to the FindZebra corpus.

Using this approach we were able to increase the number of documents from 14.3k to 136.6 articles and the number of diseases from 8.280 to 12.537. Even thought the quality of these new articles is not as good as those in the baseline corpus, they are still of very high quality. In addition to the new articles we also included 7.1k articles from the baseline sources that had previously been excluded from the baseline corpus because they could not be mapped to ICD-10 codes. The total number of articles in the expanded corpus is thus 143.7k articles.

The disadvantage of this static approach to corpus expansion is that it does not scale well and that it uses only a fraction of the information available. I.e. we retrieve 75 links for each disease alias, but we use only 12 of these at most.

4.1.1 Semi-supervised case study labeling

The PubMed Central Open Access dataset[5] is a publicly available dataset consisting of more than 1 million biomedical articles covering a wide range of subject areas. Most of these categories are irrelevant to diagnosis. However, one of the categories is *case studies* and articles under that category could have a high diagnostic value since they typically describe signs and symptoms not included in standard textbook material. We extracted 51k case studies from the total dataset. In order for our search engine to be able to use these articles, they must be labeled somehow. Unfortunately, these case studies are not labeled with information on the identity of the disease. A case study does furthermore not even always describe a single disease. For

[5] https://www.ncbi.nlm.nih.gov/pmc/tools/openftlist/

example, it is a bit unclear if the case study[6] with the title *Treatment of Ipilimumab Induced Graves' Disease in a Patient with Metastatic Melanoma* is about Graves' disease or about metastatic melanoma. Clearly, our labeling should take this ambiguous nature of case reports into account.

We use a non-parametric classifier c for labeling the case studies. The classifier gives equal probability to each disease that has an alias of the disease name occurring in the case study title. The ambiguity problem is solved by including the same case study several times but with different labels (one for each of the possible diseases). E.g. in the example above we would include a copy of the case study with the label *Graves' disease* and a copy with the label *metastatic melanoma*. Even though this method is quite simple, it gives very good results (as we will see in section 6.5).

5 Synonym injection

There are several potential ways to use the synonym list described in section 2.1.1. We could for example identify all symptoms in a text and then augment the text with a list of all known synonyms for these symptoms. However, the most natural way of including symptom synonyms is by using them where they occur. We do this a bit differently in the Solr model and in the neural models, since we have more flexibility when training the neural models.

1. **Solr model.** When indexing the Solr database we replace each symptom with a canonical form. The same is done with a query. I.e we normalize the corpus and queries to the same canonical form.

2. **Neural models** It is reasonable to hypothesize that the most natural form of a symptom is the one already occurring in the text. When creating a text sample for the neural models we therefore only select a synonym for replacement with 50% probability. In case of replacement, we choose a replacement between all known synonyms (including the one occurring in the text). I.e. the probability of replacing a symptom with one of its $n-1$ synonyms is $0.5\frac{n-1}{n}$. There are of course several potential variations of this method. For example we could use the corpus word distribution and use this to choose a synonym replacement (i.e. the probability of choosing a given synonym would be proportional to its corpus frequency).

6 Experiments

6.1 Choice of performance metric

FindZebra is intended to be a diagnosis decision support tool. Given a query, the search engine returns a list of possible diseases. We can view these returned search results as an automatically generated differential diagnosis. The probability of having the correct diagnosis in this automatically generated differential diagnosis is therefore a reasonable performance metric. This performance metric is called recall@k, and is an estimate of how frequent the correct diagnosis appear among the first k search results. In this paper we use recall@20 as a performance metric.

When training using the baseline corpus, there are 8.3k disease classes, but when training using the expanded corpus there are 12.5k disease classes. In order to be able to compare performance between the models, we therefore evaluate performance using only the classes in the baseline corpus. I.e. when evaluating performance for the models trained on the extended corpus, we filter out disease results that are not in the baseline corpus.

6.2 Training of machine learning models using the FindZebra corpus

The purpose of FindZebra is to rank diseases based on a medical query. We would therefore like to have a large dataset consisting of medical queries with a "known" diagnosis. Unfortunately, such a dataset does not exist. Instead we use each article as a proxy of a medical query and hope that the learned knowledge of *article text - disease relationships* can be transfered to *medical query - disease relationships*. Note that there are several differences between an article text and a medical query. First of all a typical query is rarely more than a list of symptoms, which is different from an article sentence (which is typically expressed in syntactically correct natural language). Also, a typical medical query only consists of a couple of sentences whereas an article is typically much longer. Using an article text as a proxy for a medical query poses another problem: If we use the entire text as input, a machine learning method will typically learn primarily to identify features such as the title of the document (i.e. the disease name), or to identify document id numbers. This has the consequence that the model will not generalize very well to new medical queries. In order to force a machine learning method to put emphasis on other features (such as symptoms), we will use a method that we call snippet sampling. Snippet sampling consists in using a small snippet of text sampled from a random part of the document as a sample instead of using the entire document. The method can be considered a very aggressive form of input dropout where we dropout everything but a small fraction of each text. Snippet sampling can therefore be considered to be a regularization method. As we will see, this regularization method will enable us to train complex machine learning models even in cases where we typically only have a single article on each disease.

[6] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4737013/

6.3 Validation and test sets

As mentioned in section 6.2 we use snippets of text from the documents in the corpus for training. However, such snippets are not representative of a typical search query. In order to estimate model performance in a realistic setting we use a dataset consisting of medical questions from a Jeopardy! like game called Doctor's dilemma featured by ACP[7]. The game is also known as Medical Jeopardy. The initial dataset from ACP consisted of 3000 questions. A large number of these questions were removed because they were ether not related to diagnosis or required visual inspection of e.g. x-ray images. In addition to these Doctors Dilemma questions we used a set of 56 questions that had been constructed manually by a medical professional. In total we created 496 queries, of which half is used for validation and the rest is used for testing. The test and validation sets can be found on-line[8].

6.4 Training details

All neural models were trained by minimizing the cross entropy error using the stochastic gradient descent-based Adam method[14] with default parameters. Besides using batch normalization, we did not use l2 or similar methods of general regularization.

The Keras[15] library (with a tensorflow[16] backend) was used for implementing and training the neural models. The models were trained on a GeForce GTX TITAN X with 12 GB of memory.

The training time for the models varied from 1-2 days for the word based models to about 2-4 days for the character based models.

6.5 Results

Table 2. Recall@20 [%] for the different models. The best results for each dataset is bolded.

	Baseline	Extended	Pubmed
Neural BOW (without synonyms)	62.90	70.97	**77.42**
Neural BOW (with synonyms)	63.31	**71.37**	**77.42**
Neural n-gram	48.79	60.48	64.52
Word LSTM	46.97	58.06	66.94
Char LSTM	46.37	64.11	63.71
Solr (with synonyms)	**68.55**	69.76	76.41
Solr (without synonyms)	**68.55**	68.75	75.81

6.5.1 Single model results

The results for the single model setups is shown in Table 2. We note several things from the results on each dataset:

Baseline dataset: We see that the neural models do not perform better than the Solr baseline model on this dataset. This is not surprising considering that we only have very few samples per disease in this corpus. The recurrent models and n-gram models do not perform very well using this dataset. However, considering that a typical disease is only represented by 1-2 articles, it is actually quite remarkable that a complex model such as a deep recurrent net can be trained at all without severe overfitting.

Extended dataset. Using the extended dataset for training gives a huge performance increase for all the neural models, and a minor performance increase for the Solr model. We also see that the inclusion of more examples pr. disease makes the neural BOW models perform better than the Solr baseline.

Pubmed dataset. We see that the performance of all models (except the character based model) increases substantially when the Pubmed case studies are added to the extended dataset.

6.5.2 Synonym injection results

We see that using synonym injection only gives small performance increase of about 0-1%. For the neural models this is not surprising since synonym information is actually learned in an unsupervised manner by these models. Figure 5 displays a t-SNE plot[17] of word representations (i.e. output of the embedding layer). The upper plot shows word representations for the neural BOW model trained on the Pubmed dataset. We see that synonymous words are represented with almost identical vectors. This means that the model does not discern between e.g "tumor", "neoplasms" or "tumors". In the lower half of Figure 5 we see

[7] https://www.acponline.org/membership/residents/competitions-awards/doctors-dilemmasm

[8] Test set: http://www.intellifind.dk/article/test_queries.csv. Validation set: http://www.intellifind.dk/article/valid_queries.csv

the same model trained with synonym injection. We see clearly that the model has learned to give identical representations to synonyms based on the synonym replacements. I.e. by using the synonym injection method we can force the model to give identical representations to words/phrases that we know are the same. The method can of course also be used to give identical representation to e.g. gene aliases.

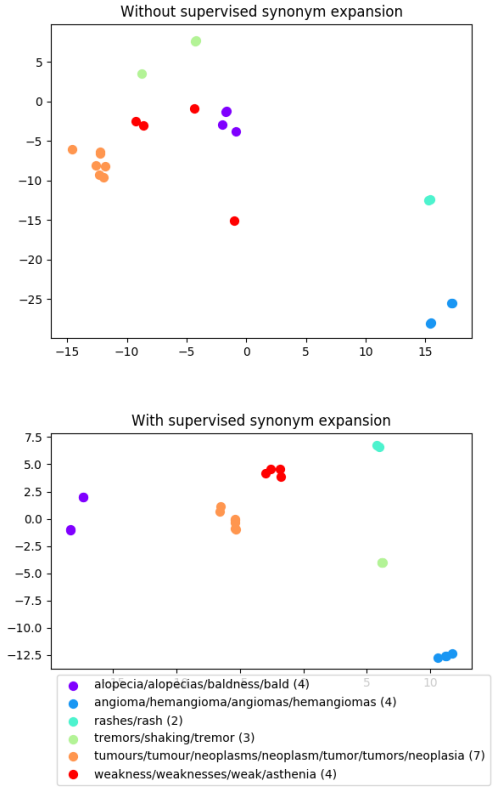


Figure 5. T-snee plots of representations of synonymous words. The upper plot shows the representation of words in a model without synonym injection, and the lower plot shows the representation of words in a model with synonym injection. The numbers after the word lists indicate the number of words in the list. The models used are BOW models trained on the PubMed dataset.

6.5.3 Ensemble results

We explore two kinds of ensemble:

- The first ensemble is a soft voting ensemble consisting of two neural BOW models trained on the Pubmed dataset. We use a model trained using synonym expansion and one trained without. Even though the models are quite similar, we obtain a recall@20 of **81.85%** which is much better than the best single model recall of 77.42%.
- The second ensemble is a merge voting ensemble consisting of a neural BOW model trained on the Pubmed dataset with synonym expansion and the Solr model. Using this ensemble we obtain a recall@20 of **81.05%**.

6.6 Misspellings

As noted in section 3.2, the character based models are much more robust towards spelling errors. In order to quantify this we introduce artificial spelling errors and measure the decrease in performance for both word based models and character

based models. We define a spelling error to be either a deletion, an insertion or an addition of a new character. A new query is created by iterating over the characters in a query and introducing a misspelling with a given probability at each character. For example, a 10% misspelling probability could turn the query "*ataxia, confusion, insomnia, death*" into the misspelled query "*atxia,RconfuHsion, intonia, death*". In Figure 6 we see the recall@20 performance degradation of three different models as a function of the misspelling probability. We see that the performance degradation of the two word based models are almost identical and quite severe. The character based model on the other hand is not affected nearly as much by misspellings.

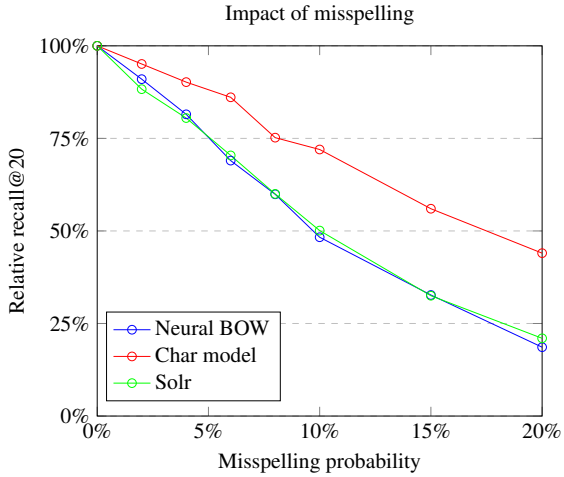


Figure 6. Impact of misspelling errors on the different models. The figure shows the recall@20 degradation as a function of misspelling probability for two word based models and one character based model. The models have been trained on the Pubmed dataset and the numbers are relative to model recall@20 when there are no misspellings.

7 Future Work

There are several possible directions for future work. We saw that the inclusion of additional data in the corpus gave a large increase in performance. Two interesting directions for future work could therefore be to:

- investigate methods for incorporating the Pubmed case studies by using more sophisticated semi-supervised methods such as label propagation[18] or co-training[19] instead of using a non-parametric classifier as we did.
- investigate methods for incorporating even more data. For example, a web crawler could easily collect a huge amount of data by using a web search engine and this data could be added in the same way as the case studies.

We saw that a character model is very resilient toward spelling errors. We believe that introduction of misspellings when training will make the model more or less immune to spelling errors and provide a degree of regularization that might even increase the overall performance.

It could also be interesting to investigate other methods for inclusion of structured meta-data. E.g. we could use structured data such as ICD-10 codes, gene synonyms or databases of gene-disease relationships.

Since each of the models described in this article is quite time-consuming to train, we have only performed a crude investigation of model hyper parameters in order to determine the best architectures. It will probably be possible to improve performance somewhat by fine-tuning the model hyper parameters.

The more complex recurrent models did not perform as well as we had expected. This is probably caused by insufficient training data. We might be able to obtain better results by pre-training the models in an unsupervised manner on unlabeled data.

8 Conclusion

In this article we have shown how to improve the performance of a specialized domain search engine using various methods such as corpus expansion, synonym injection, improved neural retrieval models and ensembles. The combination of these

methods yielded a substantial *absolute* increase in performance of **13.3%** compared with the baseline Solr model using the baseline corpus. The bulk of this performance increase seems to come from the corpus expansion and use of ensembles.

The results indicate that a certain amount of data is necessary in order to obtain better performance using a neural model compared to a frequency based model such as the one used by Solr. This seemed to be especially true for the more complex models such as the recurrent nets and *n*-gram models.

We showed that the snippet sampling regularization method allows us to train extremely complex models even when only very few samples of each class is available.

We saw that inclusion of meta-data such as symptoms has a very limited impact of performance, especially for the neural models since these models implicitly learn such relationships during training.

We believe that the findings in this article may prove useful in other specialized search engines.

Acknowledgments

The authors wish to thank the Lundbeck Foundation for supporting this research project and ACP for use of the Doctors dilemma questions.

References

1. Smiley, D., Pugh, E., Parisa, K. & Mitchell, M. *Apache Solr enterprise search server* (Packt Publishing Ltd, 2015).
2. Gormley, C. & Tong, Z. *Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine* (" O'Reilly Media, Inc.", 2015).
3. Strohman, T., Metzler, D., Turtle, H. & Croft, W. B. Indri: A language model-based search engine for complex queries. In *Proceedings of the International Conference on Intelligent Analysis*, vol. 2, 2–6 (Amherst, MA, USA, 2005).
4. Ioffe, S. & Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR* **abs/1502.03167** (2015). URL http://arxiv.org/abs/1502.03167.
5. Svenstrup, D., Hansen, J. M. & Winther, O. Hash embeddings for efficient word representations. *arXiv preprint arXiv:1709.03933* (2017).
6. Le, Q. V. & Mikolov, T. Distributed representations of sentences and documents. In *ICML*, vol. 14, 1188–1196 (2014).
7. Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural computation* **9**, 1735–1780 (1997).
8. He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778 (2016).
9. Levenshtein, V. I. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, vol. 10, 707–710 (1966).
10. Grainger, T., Potter, T. & Seeley, Y. *Solr in action* (Manning Cherry Hill, 2014).
11. Svenstrup, D., Jørgensen, H. L. & Winther, O. Rare disease diagnosis: a review of web search, social media and large-scale data-mining approaches. *Rare Dis.* **3**, e1083145 (2015).
12. Brown, G., Wyatt, J., Harris, R. & Yao, X. Diversity creation methods: a survey and categorisation. *Inf. Fusion* **6**, 5–20 (2005).
13. Harris, Z. S. Distributional structure. *Word* **10**, 146–162 (1954).
14. Kingma, D. & Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
15. Chollet, F. *et al.* Keras (2015).
16. Abadi, M. *et al.* Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).
17. Maaten, L. v. d. & Hinton, G. Visualizing data using t-sne. *J. Mach. Learn. Res.* **9**, 2579–2605 (2008).
18. Zhu, X. & Ghahramani, Z. Learning from labeled and unlabeled data with label propagation. (2002).
19. Blum, A. & Mitchell, T. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, 92–100 (ACM, 1998).

Appendix B

Information Completion for Medical Search Assistance

Information completion for medical search assistance

Dan Svenstrup, Jonas Meinertz Hansen, Mads Emil Matthiesen and Ole Winther

Abstract: We present a method for performing information completion (IC) in the context of medical search assistance. Medical IC is defined as the process where the search engine queries the user for additional information with the purpose of finding the correct diagnosis. This additional information could be in the form of test results (e.g. blood pressure), other diseases that the user might have (e.g. congenital abnormalities), or symptoms that the user has not yet supplied to the search engine (e.g. abdominal pain). We show that it is possible to train an IC system using only a corpus of unstructured medical texts and the UMLS database. The performance of the system is measured by its ability to predict withheld symptoms from multi-symptom diagnosis queries. This is a quite conservative measure because it ignores the possibility of the proposed symptoms being related to the exact match. Since we usually expect that the user will only have short time to solve the task, the IC system is limited to ask the user a very limited number of questions. Our main metric for evaluating performance is therefore recall@k where $k \leq 10$. Recall@k is defined as the probability of a relevant question being among the top k questions. In order to obtain a good baseline level of performance for future IC systems, we have trained several different deep neural models and have achieved a recall@10 of more than 35% using our best model on a test set of medical questions. This level of performance makes the IC system usable in practice, especially considering that this recall estimate is a very loose lower bound.

Keywords and phrases: Deep learning, information completion (IC) systems, information retrieval, medical search, symptom checker.

1. Introduction

Medical search is one of the major usages of the internet. Each year approximately 35% of all adult Americans use the internet for diagnosis, and most (77%) of all the searches began at a standard search engine such as Google or Bing [Pro13]. This is also reflected in the fact that a staggering 1% of all Google searches are health related [blo16]. In [Dra+13] it was shown that a specialized search engine for rare diseases can actually give more precise results compared to standard web search such as Google search (measured

by recall@k). This was shown to be caused by the nature of the ranking algorithm (a standard search engine is in a way weighing the document matches against the "popularity" of the document, instead of finding the best symptom match combined with prevalence information). On top of providing an improved recall, a specialized search engine can also assist the user in important ways both before and after the search:

- **Before the search**. At this stage the search engine can help the user construct a good search query, e.g. by providing a clickable avatar that can help the user specify the correct body part of an ailment, if applicable.
- **After the search**
  - If the search has been successful, the search engine should help the user by providing more information on the disease. This information could be in the form of links to case reports and other textual information about the disease, addresses to the nearest expert centers and so on.
  - If the search has not been successful, the search engine can still provide important feedback by e.g. asking relevant questions about other diseases/symptoms that the user might have, or suggest additional medical tests. We will refer to this interactive procedure following an unsuccessful search as **medical information completion** (IC). More precisely we will define medical IC as *the process of asking a user relevant questions with the express purpose of filling out missing or incomplete information that could be important for the diagnosis.*

In the rest of this article we will describe a quantitative method for evaluating the performance of medical IC systems. I.e. we will quantify how well the model "finds the missing information" and use the method to evaluate the performance of several deep neural network models trained for IC.

The rest of the article will be structured as follows: In section 2 we will take a look at some methods that are similar to the those proposed in this paper in the sense that they can be used to find information related to a given text string. In section 3 we will describe how the collection of possible IC questions (in the form of symptoms, test results and diseases) was selected. Once such a list of questions has been compiled, the next problem is how to create test and training/validation sets. This will be described in section 4. In section 5 we will describe the architectures of the different used IC models. In section 6 we will explain why the reported performance of the IC models

can only be considered a lower bound, and we will report the performance (measured by recall@k) of the different models. Finally, in section 7 we will discuss other, and perhaps better, ways of performing medical IC.

2. Related work

To our knowledge, no previous work has been done specifically on medical information completion. However, there are several closely related areas that have been treated in the literature.

Arriving at a diagnosis based on a series of queries may be considered a medical sequential decision problem that can be approached by reinforcement learning techniques [Poo03]. The approach taken in this paper may be seen as a greedy approximation to a full reinforcement learning approach. The main reason for this is that the data required for setting the problem up as reinforcement learning is not easily available, see the section 4.

Textual information completion in general can be thought of as *the process of finding information related to the semantic content of a string or group of strings, with the express purpose of finding the answer to a specific question.* One way of finding information related to the semantic content of a group of strings is by Latent Dirichlet Allocation (LDA) [BNJ03]. In LDA a document is generated by:

1. choosing a topic distribution ϕ for the entire document.
2. for each word we wish to generate, we first draw a word topic θ from ϕ, and then draw a word from the conditional word distribution $P(w \mid \theta)$.

Once such a model has been trained we can draw words related to the query *headache, abdominal pain* by first estimating the document topic distribution, draw a topic from this topic distribution and finally draw a word from the conditional word distribution. Using an LDA model in this way will generate words that are related to the query string in the sense that "in this kind of document we see these kinds of words". This could seem like a good idea, but the proposed words will not bring us closer to the answer that we seek. The reason for this is best illustrated by an example: suppose we wish to build an information completion system for movies (i.e. a system that, based on an input query, asks the user questions with the express purpose of finding the movie title that the user is looking for). This system would take as input some information about the movie e.g. "the lead actor was Tom Cruise and it was produced in the 80's". Based on this information the system should try asking the user questions like: "Was Val Kilmer also in

the movie?"[1]. However, if you give this information to an LDA model, it will typically just propose random male actors from the 80s. This is of course also related to the query string, but the proposed questions will not help us get closer to the name of the movie. Thus, LDA fails to meet the second requirement of an IC system.

Information completion is also related to auto-completion as found in e.g. web browser search fields. However, the purpose of auto-complete is finding a *syntactical* completion of a query sentence (and not a *semantic* completion as is the case with IC models).

Filters on web search pages is another form of information completion. A filter is a way of indicating the category of the search. For example, an internet store might have a general search field for the initial search and then allow the user to filter out everything except results about books, guitars, food etc. Something similar to information completion can also be found on many medical search sites. For example, WebMD uses an avatar (a clickable miniature body) that you can click on in order to narrow down the region of the possible input symptoms. This, however, is not IC since you click on the avatar before you enter the text query (i.e. you do not really complete anything).

All but one of the medical IC models that we are presenting in this paper estimates a target symptom from its context bag-of-words. Such a model can be considered an advanced form of correlation analysis, where we use a deep neural network to estimate how much a group of words "correlate" with a given symptom. The multiple correlation coefficient measures the Pearson correlation between a true target value and a target value predicted by a linear model. The quantification method used to evaluate performance of the models presented in this paper can thus be seen as an extension of the multiple correlation coefficient, where we use a deep neural network instead of a linear model, and recall@k instead of correlation.

3. Symptom selection and extraction

In the rest of the article, the word symptom will be used for both test results (e.g. low blood pressure), actual symptoms (e.g. abdominal pain) and diseases (e.g. skin cancer). A necessary pre-requisite of being able to propose or identify symptoms in text, is of course a list of symptoms. The list of symptoms was created in a multi-stage filtering process:

[1]The movie Top Gun featured Tom Cruise and Val Kilmer in the lead roles.

1. First a raw symptom list was created, consisting of all UMLS[2] terms with a semantic type of *Acquired Abnormality, Anatomical Abnormality, Cell or Molecular Dysfunction, Congenital Abnormality, Disease or Syndrome, Experimental Model of Disease, Finding, Injury or Poisoning, Mental or Behavioral Dysfunction, Sign or Symptom, Neoplastic Process*, or *Pathologic Function*. These semantic types cover all diseases, signs and symptoms.
2. Then all symptoms that we had no information about were removed, i.e. symptoms that do not occur in the corpus (see Section 4 for the contents of the corpus).
3. Then all but the most frequent 3000 symptoms were removed.
4. Each symptom on the symptom list was then reviewed and we manually removed 859 concepts that should not be posed as a question to a user, e.g. "room air", "administered intravenously", "very limited" or "male predominance". This left us with a final list of 2141 symptoms. The list can be found on-line[3].

85% of all symptom occurrences in the FindZebra corpus are symptoms that occur in our curated list. Furthermore, most of the symptoms in the FindZebra corpus that are *not* in the list seem to consist mainly of specialized versions of symptoms in our list e.g. *severe abdominal pain* instead of *abdominal pain*. The curated list of symptoms can therefore be considered to provide relatively good coverage of all symptoms in the UMLS database.

4. Data

4.1. Training set

Ideally, we would have liked to have a data set consisting of a large number of pairs of the form `(search query, missing information)` where the missing information could be a symptom or a test result. Unfortunately such a data set does not exist. Instead we constructed a data set by extracting a large number of word sequences with at least 1 symptom from our target symptom list. Each sequence consisted of up to 20 words. From each sentence we iteratively removed one target symptom and used the pair `(modified sentence, target symptom)` as a sample. The sequence length was chosen based on a rough estimate of the average sentence length.

[2] the UMLS database is a database describing the properties and relations between medical concepts. It is maintained by National Library of Medicine. See www.nlm.nih.gov/research/umls/

[3] http://www.intellifind.dk/article/symptom_list_curated.txt

The corpus used for extraction of sentences was an extended version of the corpus used by our rare disease search engine FindZebra (see Table 1). As mentioned above, we iteratively deleted one of the symptoms/test results from each of the extracted sequences. E.g. the sentence *The patient suffered from abdominal pain, fever and headache* was transformed into the 3 samples

- *The patient suffered from fever and headache.*
- *The patient suffered from abdominal pain and headache.*
- *The patient suffered from fever and headache.*

This resulted in a data set consisting of approximately 4.5 million samples. Of course, a data set created in this way will be quite noisy (see Table 2 for some samples from the data set). Often there is only a small resemblance between a typical query (which is often just a list of symptoms) and samples in the training set. Therefore there is no guarantee beforehand that a trained model will generalize well to typical queries. From Table 2 it is also worth noticing that the UMLS database is not quite complete. For example we should have matched *intra-abdominal carcinomatosis* instead of just *carcinomatosis* in the last sample.

The core FindZebra corpus		
Source	**Number of documents**	**Data quality**
GARD	1917	Very high
Gene Reviews	638	Very high
GHR	1075	Very high
Omim	8062	Very high
Orphanet	4578	Very high
Wikipedia	5154	Very high
Bootstrapped data set	122300	Good

TABLE 1

Overview of the number and quality of documents in the FindZebra corpus. The bootstrapped data set was created by searching the internet for articles about each of the diseases in the core FindZebra data set.

4.2. Validation and test sets

Normally the data set would be split into a training set, a validation set and a test set and the generalization performance of the different models would be measured on the test set. However, as mentioned above, many of the samples in the training set have little resemblance with a typical query. In order to test the model in a realistic setting we therefore used a data set consisting of actual medical questions/queries. The questions used were

Removed symptom	**Training set sample**
neoplasms	, this has returned different genetic alterations (data not shown) cdc are generally considered to be aggressive ~~neoplasms~~
fever	mds is discovered only incidentally on routine blood counts previous chemotherapy or radiation exposure is an important historic fact ~~fever~~
diarrhea	infancy two forms are recognized : early-onset mvid with ~~diarrhea~~ beginning in the neonatal period , and late-onset , with
hypophosphatemia	of 11 years , at which time laboratory data revealed ~~hypophosphatemia~~ , elevated vitamin d levels , and hypercalciuria a
hypercalciuria	of 11 years , at which time laboratory data revealed hypophosphatemia , elevated vitamin d levels , and ~~hypercalciuria~~ a
lesions	sekeres ma , theil ks , maciejewski jp chromosomal and uniparental ~~lesions~~ disomy detected by snp arrays in mds ,
ehlers-danlos syndrome	mitral valve prolapse , ~~ehlers-danlos syndrome~~ and other diseases that present with aortic aneurysm such as loeys-dietz syndrome (see
carcinomatosis	one with possible early stromal invasion two of the five patients who developed intra-abdominal ~~carcinomatosis~~ were among 78 patients in

TABLE 2

Some samples from the training set. The removed symptom is crossed out. Note that a "symptom" in this context can mean both an actual symptom, a test result or a disease.

from a Jeopardy! like game called Doctor's dilemma featured by ACP[4]. The game is also known as Medical Jeopardy, and is held once a year at the scientific Internal Medicine Meeting where up to 50 teams of residents from all over the world compete for the title of national champion. We obtained a data set from ACP consisting of 3000 questions. A large number of these questions were removed because:

1. The question was not related to diagnosis (i.e. it was not found in the FindZebra database). The FindZebra database both contains most of the known rare diseases and the more common diseases found on Wikipedia.
2. The question required visual inspection of e.g. x-ray images.
3. The question text did not contain a target symptom from our list of 2141 symptoms.

The test and validation sets can be found on-line[5]. From these data sets we then created samples in the same way as we created the training set: by iteratively removing a symptom from the question (see Table 3 for a few examples). This left us with a total of 477 validation samples and 473 test samples.

5. Description of models

In this section we will describe several models for performing information completion:

1. A model that always predicts the most frequent symptoms.
2. Deep bag of words model (Section 5.1): A basic network with three hidden layers and a softmax output, using words as input.
3. Deep bag of UMLS terms model (Section 5.2): The same as the words model above, except that it uses only symptom/gene UMLS terms as input tokens.
4. Deep bag of n-grams model (Section 5.3): The same model as above, except that it uses n-grams as features.
5. Deep recurrent model (Section 5.4): A deep, bidirectional recurrent network with GRU units, with words as input.

[4] https://www.acponline.org/membership/residents/competitions-awards/doctors-dilemmasm

[5] Test set: http://www.intellifind.dk/article/test_queries.csv.
Validation set: http://www.intellifind.dk/article/valid_queries.csv

5.1. Deep bag of words model

The deep bag of words model uses the most frequent 20.000 words in our corpus as features. The number of words was chosen by cross-validation. The model used embeddings of size 1000, followed by a sum layer over the time dimension. On top of the sum layer we placed three hidden layers with 2000 units each, and a softmax output layer. We used batch normalization [IS15] between each of the hidden layers.

As can be seen in table 4 this model actually achieves the best overall performance among all of the used models. Some examples of query completions proposed by this model can be seen in table 3.

As with all bag of words models, this model has two serious drawbacks:

1. **Word ordering**: a bag of words model does not take the word ordering into account. This means that two widely different sentences such as *the patient suffered from a rash on the stomach and pain in the left foot* and *the patient suffered from pain in the stomach and a rash on the left foot* are considered as equal by a bag of words model. Also, "abdominal pain" is not considered a single concept but as two separate words.
2. **Feature selection**: when using a bag of words model it is necessary to choose a subset of the words in the corpus as features, and this subset might not be optimal. E.g. in our case we choose the most frequent 20.000 words, and even though a lot of these words will be important, we probably also exclude some other important words, such as specialized medical terms.

In the following we will describe several models that in one way or another try solve the two problems above.

5.2. Deep bag of UMLS terms

The deep bag of UMLS terms model tries to solve the feature selection problem by focusing exclusively on medical (UMLS) terms. There are almost 50.000 symptom UMLS terms and based on cross-validation we choose the most frequent 25.000 of these. One of the largest advantages when using the UMLS database is that we can collapse n-grams such as "abdominal pain" into a single feature instead of two single word features, and that synonyms can be mapped to the same feature. That is, "stomach ache" is the same feature as "tommy ache" and "belly ache". However, even though there are advantages to using the UMLS database compared to using just raw word features, there is also a big problem due to the high variability

on how medical terms are written. For example, “abdominal pain” could be written as “pain in the stomach”, “stomach pains” etc. and even though the UMLS database is very detailed, it is not complete. Furthermore, there might also be a lot of information in non-symptom features (for example the word “eat” should probably increase the probability of symptoms related to food poisoning or diabetes). This non-symptom related information is of course lost when considering only UMLS terms.

The architecture of the deep bag of UMLS terms is identical to the bag of words model described above, except that it takes UMLS terms as input.

5.3. Deep bag of n-grams

The deep bag of n-grams uses n-grams instead of words as input features. It has the potential to overcome the incompleteness problem with the UMLS database and the problem that there might be important information in non-symptom words. The method identifies n-grams by giving each sequence of two (n-1)-grams a score given by

$$\text{score}(w_i, w_j) = \frac{\text{count}(w_i, w_j) - \delta}{\text{count}(w_i) * \text{count}(w_j)} \tag{1}$$

If the score is above a given threshold, the sequence of (n-1)-grams will be considered an n-gram. Thus, by iteratively calculating the score we can identify higher order n-grams (see [LM14] for details). However, in this article we only consider n-grams for n=1-2.

Even though the n-gram model is more nuanced compared to the basic UMLS model, we still have the problem that the choice of n-grams might be suboptimal, and we still have not solved the word ordering problem, except for small word sequences of length less than 3.

The architecture of the deep bag of n-grams is identical to the bag of words model described above, except that it takes n-grams as input.

5.4. Deep recurrent network

A *sequence* model such as a recurrent network has the potential to overcome both of the problems of a bag of words model. Since we are basically classifying sequences, such a model would normally be our model of choice. However, there are two potential problems when using a sequence based model for this particular task:

- when removing a symptom from a training sample we are disrupting the syntactical structure of the sentence.
- the syntactical form of a typical training set sample (natural language) is often different from the syntactical form of a typical query (typically not much more than a list of symptoms and test results).

Both of these problems occur because the syntactic structure of a sentence is (to some degree) learned by a sequence model and used when predicting the missing piece of information. I.e. the syntactic structure of a sentence is considered as a feature for such a model. Differences between the syntactical structure in the train and test set might therefore increase the generalization error.

We trained the recurrent net on top of a word embedding of size 1000, with vectors for the most frequent 10.000 words. We used two layers of bidirectional gated recurrent layers GRU[Chu+14]. The first layer had 256 units, the second had 512. The outputs of the two bi-directional layers were joined by a sum layer in order to get one representation of the entire sentence. The sentence representation was finally channeled through two hidden layers with 2000 units each, terminating in a softmax layer.

6. Results

6.1. Tightness of the performance bound

As described earlier, we test performance on a test set of 473 questions. It should be noted, however, that measuring the recall performance on the test set will only give us a very loose lower bound on the performance. This is due to the fact that the performance evaluation is automatized such that only prediction of the removed symptom counts as correct, but other symptoms might also be relevant. I.e. a deleted symptom is automatically relevant, but other symptoms might also be relevant. For example, when removing the symptom "high fever" from the query *a high fever, runny nose, sore throat, muscle pains, headache, coughing, and feeling tired*, we get the proposals [fatigue, chills, dizziness, fever, nausea]. Considering that the query is a query for influenza, all of these proposals are probably relevant, but none of them is the actual deleted symptom. Other examples of this can be found among many of the queries in table 3. Also, since we allow the missing information to be a disease, the correct diagnosis can be among the proposed symptoms, but even this will typically also count as an error.

When measuring performance we use all queries containing a target symptom. However, there are several arguments why it would also have been rea-

Diagnosis	Removed symptom	Test set sample	Rank	Proposals
Meniere's disease	hearing loss	syndrome characterized by tinnitus, ~~hearing loss~~, vertigo, and aural fullness	1	hearing loss, hearing, sensorineural deafness, deafness, dizziness
Lemierre Syndrome	sore throat	syndrome of ~~sore throat~~, fever, sepsis, and unilateral neck swelling	5	headache, chills, lymphadenopathy, cough, sore throat
Tuberculosis	hemoptysis	infection characterized by productive cough, ~~hemoptysis~~, and cavitary pulmonary lesion in a patient with pulmonary silicosis	3	dyspnea, pneumonia, nodules
Klinefelter syndrome	gynecomastia	syndrome of ~~gynecomastia~~, small testes, and hypergonadotropic hypogonadism	5	cryptorchidism, infertility, delayed puberty, primary amenorrhea, gynecomastia
Frontotemporal dementia	personality change	dementia associated with prominent disinhibition and ~~personality change~~	10	depression, memory, cognitive impairment, psychosis, apathy
Eosinophilic esophagitis	plaques	diagnosis for atopy, recurrent food impaction and esophageal rings, furrows, ~~plaques~~, or strictures	360	gastroesophageal reflux, foreign body, dysphagia, asthma, swallowing
Autoimmune hepatitis	type 2 diabetes	liver disease associated with ~~type 1 diabetes~~, thyroiditis, and celiac sprue	2	autoimmune diseases, type 1 diabetes, thyroid disease, rheumatoid arthritis, endocrinopathy
Metabolic syndrome	hypertriglyceridemia	diagnosis associated with abdominal obesity, ~~hypertriglyceridemia~~, low hdl, hypertension, and elevated glucose	5	diabetes, diabetes mellitus, dyslipidemia, hypercholesterolemia, hypertriglyceridemia
Asthma	wheezing	common syndrome of cough, reversible ~~wheezing~~ , and peripheral blood eosinophilia	2	dyspnea, wheezing, chest pain, asthma, pulmonary infiltrates

TABLE 3

Samples from the test set. The removed symptom is crossed out. The proposals in the last column are generated by the deep bag of words model described in section 5.1

sonable to include only those queries with 2 or more symptoms in the query text:

- The system is an information completion system. This means that it is intended to be used when the user has already entered a couple of symptoms.
- For some queries the target symptom is more or less the only information content in the string. Therefore it does not seem reasonable to expect a model to be able to predict the target symptom from the modified query. For example, the query *schober test is associated with this condition* would become a test sample with text *is associated with this condition* and target *schober test*.

Using only queries with 2 or more symptoms in the query text would lead to higher reported performance. However, we have chosen not to do this, since the main purpose of the performance measurement is to enable us to compare different IC models. In any case, no matter how we decided to choose the test set, the bound would still be very loose due to the other problems discussed above.

6.2. Recall@k performance

The recall@k performance for the different models is shown in Figure 1. In table 4 the recall performance for some selected n is given. The inferior performance of the UMLS model could indicate that either the UMLS database is not complete enough for this task or that non-symptom words are actually important. Using learned n-grams does not improve the results, and actually the opposite happens. This could be caused by the fact that even though the n-gram adds sequence information to the query, it also takes away information on the individual words that make up the query. Using n-grams also makes the query much more specific. For example, reducing *abdominal pain* to *abdominal_pain* is similar to replacing an OR operator with an AND operator in the query.

The recurrent model performs just as well as the deep bag of words model, but is much more complex. All in all it seems that the deep bag of words model is the best choice among the four models.

7. Discussion and further work

In this article we have described a framework for training and evaluating IC systems. That is, we have shown how we can find missing information in a

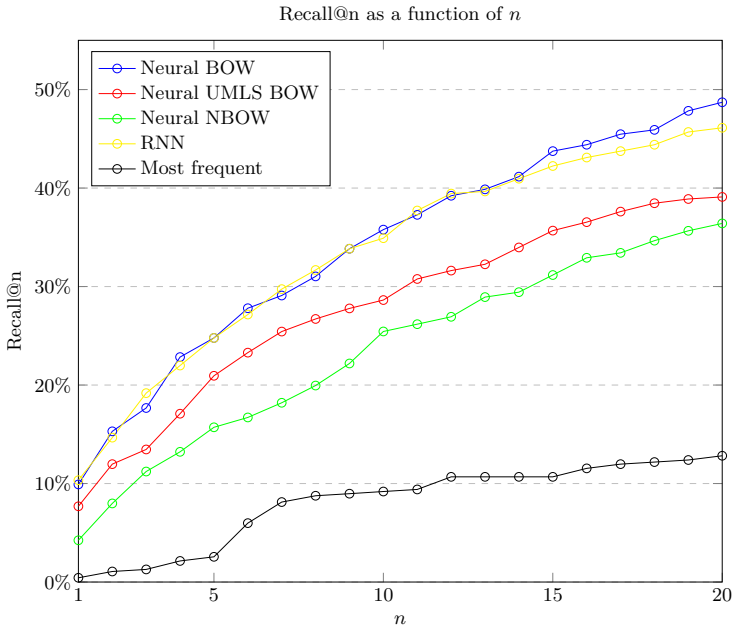


FIG 1. *Recall@n for the different models as a function of n.*

search query, how we can compare different IC systems, and how we can get an estimate of a lower bound on the performance of the system.

What we have actually not shown is how we can integrate this information in the best way possible into the diagnostic process. I.e. instead of just showing the user the symptoms that fits best in the current search query context, we should propose the symptoms that gives most value to the following diagnostic process. This could be done in several different ways. One method could be fitting a random forest to the list of possible diseases (found by standard search), using the top n symptom proposals plus the query as features. Based on the feature importance we would then obtain a ranking of the proposed symptoms based on their diagnostic usefulness. However, it is difficult to objectively measure the performance of such a system, and this task will be left for future work.

The information completion system described in this article is already being tested in two FindZebra applications. The first application is a pre-

Model	r@1	r@3	r@5	r@10	r@20
Most frequent symptoms	0.00	0.01	0.03	0.09	0.13
Deep bag of words	**0.10**	0.18	**0.25**	**0.36**	**0.49**
Deep bag of UMLS terms	0.08	0.13	0.21	0.29	0.39
Deep bag of n-gram	0.04	0.11	0.16	0.25	0.36
Deep recurrent net	**0.10**	**0.19**	**0.25**	0.35	0.46

TABLE 4
Recall performance for the different architectures.

consultation app (chat bot) that performs information completion using a mix of static and model generated questions. The purpose of the app is to gather information on a patient in a pre-consultation session before the actual consultation with medical professionals begins. The second application is on findzebra.com where the system will be used for improving the filters shown following a search.

In this article we have defined the missing information as either a symptom, a disease or a test result. But many other types of information could be useful. For example, assume that a patient is suffering from a known disease and that there exists a drug for the disease that has an adverse effect. If the patient is suffering from the adverse effect it would be valuable information to know if the patient is actually taking that drug.

8. Acknowledgements

The authors wish to thank the Lundbeck Foundation for supporting the FindZebra research project and ACP for use of the Doctors dilemma questions.

References

[BNJ03] David M Blei, Andrew Y Ng, and Michael I Jordan. "Latent dirichlet allocation". In: Journal of machine Learning research 3.Jan (2003), pp. 993–1022.

[Poo03] Radhika Poolla. "A reinforcement learning approach to obtain treatment strategies in sequential medical decision problems". PhD thesis. University of South Florida, 2003.

[Dra+13] Radu Dragusin et al. "FindZebra: A search engine for rare diseases". In: International journal of medical informatics 82.6 (2013), pp. 528–538.

[Pro13] Pew Internet Project. Health Online 2013. 2013. URL: http://www.pewinternet.org/~/media//Files/Reports/PIP_HealthOnline.pdf.

[Chu+14] Junyoung Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: arXiv preprint arXiv:1412.3555 (2014). URL: https://arxiv.org/abs/1412.3555.

[LM14] Quoc V Le and Tomas Mikolov. "Distributed Representations of Sentences and Documents." In: ICML. Vol. 14. 2014, pp. 1188–1196.

[IS15] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: CoRR abs/1502.03167 (2015). URL: http://arxiv.org/abs/1502.03167.

[blo16] Google official blog. I'm Feeling Yucky :(Searching for symptoms on Google. 2016. URL: https://blog.google/products/search/im-feeling-yucky-searching-for-symptoms/.

Appendix C

Hash Embeddings for Efficient Word Representations

Hash Embeddings for Efficient Word Representations

Dan Svenstrup
Department for Applied Mathematics and Computer Science
Technical University of Denmark (DTU)
2800 Lyngby, Denmark
dsve@dtu.dk

Jonas Meinertz Hansen
FindZebra
Copenhagen, Denmark
jonas@findzebra.com

Ole Winther
Department for Applied Mathematics and Computer Science
Technical University of Denmark (DTU)
2800 Lyngby, Denmark
olwi@dtu.dk

Abstract

We present hash embeddings, an efficient method for representing words in a continuous vector form. A hash embedding may be seen as an interpolation between a standard word embedding and a word embedding created using a random hash function (the hashing trick). In hash embeddings each token is represented by k d-dimensional embeddings vectors and one k dimensional weight vector. The final d dimensional representation of the token is the product of the two. Rather than fitting the embedding vectors for each token these are selected by the hashing trick from a shared pool of B embedding vectors. Our experiments show that hash embeddings can easily deal with huge vocabularies consisting of millions of tokens. When using a hash embedding there is no need to create a dictionary before training nor to perform any kind of vocabulary pruning after training. We show that models trained using hash embeddings exhibit at least the same level of performance as models trained using regular embeddings across a wide range of tasks. Furthermore, the number of parameters needed by such an embedding is only a fraction of what is required by a regular embedding. Since standard embeddings and embeddings constructed using the hashing trick are actually just special cases of a hash embedding, hash embeddings can be considered an extension and improvement over the existing regular embedding types.

1 Introduction

Contemporary neural networks rely on loss functions that are continuous in the model's parameters in order to be able to compute gradients for training. For this reason, any data that we wish to feed through the network, even data that is of a discrete nature in its original form will be translated into a continuous form. For textual input it often makes sense to represent each distinct word or phrase with a dense real-valued vector in $\mathbb{R}^n$. These word vectors are trained either jointly with the rest of the model, or pre-trained on a large corpus beforehand.

For large datasets the size of the vocabulary can easily be in the order of hundreds of thousands, adding millions or even billions of parameters to the model. This problem can be especially severe when n-grams are allowed as tokens in the vocabulary. For example, the pre-trained Word2Vec vectors from Google (Mihált, 2016) has a vocabulary consisting of 3 million words and phrases. This means that even though the embedding size is moderately small (300 dimensions), the total number of parameters is close to one billion.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

The embedding size problem caused by a large vocabulary can be solved in several ways. Each of the methods have some advantages and some drawbacks:

1. **Ignore infrequent words**. In many cases, the majority of a text is made up of a small subset of the vocabulary, and most words will only appear very few times (Zipf's law (Manning et al., 1999)).

   By ignoring anything but most frequent words, and sometimes stop words as well, it is possible to preserve most of the text while drastically reducing the number of embedding vectors and parameters. However, for any given task, there is a risk of removing too much or to little. Many frequent words (besides stop words) are unimportant and sometimes even stop words can be of value for a particular task (e.g. a typical stop word such as "and" when training a model on a corpus of texts about logic). Conversely, for some problems (e.g. specialized domains such as medical search) rare words might be very important.
2. **Remove non-discriminative tokens after training**. For some models it is possible to perform efficient feature pruning based on e.g. entropy (Stolcke, 2000) or by only retaining the K tokens with highest norm (Joulin et al., 2016a). This reduction in vocabulary size can lead to a decrease in performance, but in some cases it actually avoids some over-fitting and increases performance (Stolcke, 2000). For many models, however, such pruning is not possible (e.g. for on-line training algorithms).
3. **Compress the embedding vectors**. Lossy compression techniques can be employed to reduce the amount of memory needed to store embedding vectors. One such method is quantization, where each vector is replaced by an approximation which is constructed as a sum of vectors from a previously determined set of centroids (Joulin et al., 2016a; Jegou et al., 2011; Gray and Neuhoff, 1998).

For some problems, such as online learning, the need for creating a dictionary before training can be a nuisance. This is often solved with *feature hashing*, where a hash function is used to assign each token $w \in \mathcal{T}$ to one of a fixed set of "buckets" $\{1, 2, \dots B\}$, each of which has its own embedding vector. Since the goal of hashing is to reduce the dimensionality of the token space $\mathcal{T}$, we normally have that $B \ll |\mathcal{T}|$. This results in many tokens "colliding" with each other because they are assigned to the same bucket. When multiple tokens collide, they will get the same vector representation which prevents the model from distinguishing between the tokens. Even though some information is lost when tokens collide, the method often works surprisingly well in practice (Weinberger et al., 2009).

One obvious improvement to the feature hashing method described above would be to learn an optimal hash function where important tokens do not collide. However, since a hash function has a discrete codomain, it is not easy to optimize using e.g. gradient based methods used for training neural networks (Kulis and Darrell, 2009).

The method proposed in this article is an extension of feature hashing where we use k hash functions instead of a single hash function, and then use k trainable parameters for each word in order to choose the "best" hash function for the tokens (or actually the best combination of hash functions). We call the resulting embedding *hash embedding*. As we explain in section 3, embeddings constructed by both feature hashing and standard embeddings can be considered special cases of hash embeddings.

A hash embedding is an efficient hybrid between a standard embedding and an embedding created using feature hashing, i.e. a hash embedding has all of the advantages of the methods described above, but none of the disadvantages:

- When using hash embeddings there is no need for creating a dictionary beforehand and the method can handle a dynamically expanding vocabulary.
- A hash embedding has a mechanism capable of implicit vocabulary pruning.
- Hash embeddings are based on hashing but has a trainable mechanism that can handle problematic collisions.
- Hash embeddings perform something similar to product quantization. But instead of all of the tokens sharing a single small codebook, each token has access to a few elements in a very large codebook.

Using a hash embedding typically results in a reduction of parameters of several orders of magnitude. Since the bulk of the model parameters often resides in the embedding layer, this reduction of

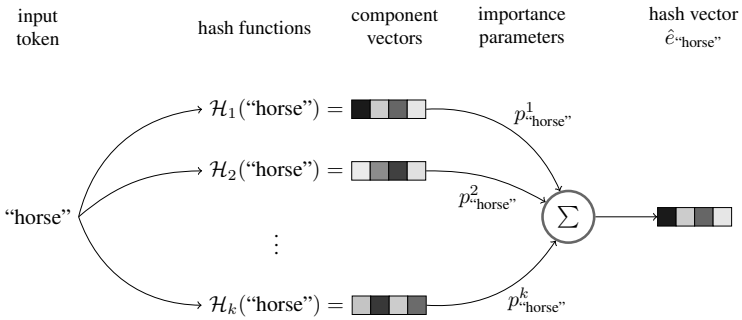


Figure 1: Illustration of how to build the hash vector for the word “horse”. The optional step of concatenating the vector of importance parameters to $\hat{e}_{\text{“horse”}}$ has been omitted. The size of component vectors in the illustration is $d = 4$.

parameters opens up for e.g. a wider use of e.g. ensemble methods or large dimensionality of word vectors.

2 Related Work

Argerich et al. (2016) proposed a type of embedding that is based on hashing and word co-occurrence and demonstrates that correlations between those embedding vectors correspond to the subjective judgement of word similarity by humans. Ultimately, it is a clever reduction in the embedding sizes of word co-occurrence based embeddings.

Reisinger and Mooney (2010) and since then Huang et al. (2012) have used multiple different word embeddings (prototypes) for the same words for representing different possible meanings of the same words. Conversely, Bai et al. (2009) have experimented with hashing and treating words that co-occur frequently as the same feature in order to reduce dimensionality.

Huang et al. (2013) have used bags of either bi-grams or tri-grams of letters of input words to create feature vectors that are somewhat robust to new words and minor spelling differences.

Another approach employed by Zhang et al. (2015); Xiao and Cho (2016); Conneau et al. (2016) is to use inputs that represent sub-word units such as syllables or individual characters rather than words. This generally moves the task of finding meaningful representations of the text from the input embeddings into the model itself and increases the computational cost of running the models (Johnson and Zhang, 2016). Johansen et al. (2016) used a hierarchical encoding technique to do machine translation with character inputs while keeping computational costs low.

3 Hash Embeddings

In the following we will go through the step by step construction of a vector representation for a token $w \in \mathcal{T}$ using hash embeddings. The following steps are also illustrated in fig. 1:

1. Use k different functions $\mathcal{H}_1, \ldots, \mathcal{H}_k$ to choose k *component vectors* for the token w from a predefined pool of B shared component vectors
2. Combine the chosen component vectors from step 1 as a weighted sum: $\hat{e}_w = \sum_{i=1}^{k} p_w^i \mathcal{H}_i(w)$. $p_w = (p_w^1, \ldots, p_w^k)^\top \in \mathbb{R}^k$ are called the *importance parameters* for w.
3. *Optional*: The vector of importance parameters for the token p_w can be concatenated with $\hat{e}_w$ in order to construct the final hash vector e_w.

The full translation of a token to a hash vector can be written in vector notation ($\oplus$ denotes the concatenation operator):

$$\begin{aligned} c_w &= (\mathcal{H}_1(w), \mathcal{H}_2(w), \ldots, \mathcal{H}_k(w))^\top \\ p_w &= (p_w^1, \ldots, p_w^k)^\top \\ \hat{e}_w &= p_w^\top c_w \\ e_w^\top &= \hat{e}_w^\top \oplus p_w^\top \text{(optional)} \end{aligned}$$

The token to component vector functions $\mathcal{H}_i$ are implemented by $\mathcal{H}_i(w) = E_{D_2(D_1(w))}$, where

- $D_1 : \mathcal{T} \to \{1, \ldots K\}$ is a token to id function.
- $D_2 : \{1, \ldots, K\} \to \{1, \ldots B\}$ is an id to bucket (hash) function.
- E is a $B \times d$ matrix.

If creating a dictionary beforehand is not a problem, we can use an enumeration (dictionary) of the tokens as D_1. If, on the other hand, it is inconvenient (or impossible) to use a dictionary because of the size of $\mathcal{T}$, we can simply use a hash function $D_1 : \mathcal{T} \to \{1, \ldots K\}$.

The importance parameter vectors p_w are represented as rows in a $K \times k$ matrix P, and the token to importance vector mapping is implemented by $w \to P_{\hat{D}(w)}$. $\hat{D}(w)$ can be either equal to D_1, or we can use a different hash function. In the rest of the article we will use $\hat{D} = D_1$, and leave the case where $\hat{D} \neq D_1$ to future work.

Based on the description above we see that the construction of hash embeddings requires the following:

1. A trainable embedding matrix E of size $B \times d$, where each of the B rows is a component vector of length d.
2. A trainable matrix P of importance parameters of size $K \times k$ where each of the K rows is a vector of k scalar importance parameters.
3. k different hash functions $\mathcal{H}_1, \ldots, \mathcal{H}_k$ that each uniformly assigns one of the B component vectors to each token $w \in \mathcal{T}$.

The total number of trainable parameters in a hash embedding is thus equal to $B \cdot d + K \cdot k$, which should be compared to a standard embedding where the number of trainable parameters is $K \cdot d$. The number of hash functions k and buckets B can typically be chosen quite small without degrading performance, and this is what can give a huge reduction in the number of parameters (we typically use $k = 2$ and choose K and B s.t. $K > 10 \cdot B$).

From the description above we also see that the computational overhead of using hash embeddings instead of standard embeddings is just a matrix multiplication of a $1 \times k$ matrix (importance parameters) with a $k \times d$ matrix (component vectors). When using small values of k, the computational overhead is therefore negligible. In our experiments, hash embeddings were actually marginally faster to train than standard embedding types for large vocabulary problems[1]. However, since the embedding layer is responsible for only a negligible fraction of the computational complexity of most models, using hash embeddings instead of regular embeddings should not make any difference for most models. Furthermore, when using hash embeddings it is not necessary to create a dictionary before training nor to perform vocabulary pruning after training. This can also reduce the total training time.

Note that in the special case where the number of hash functions is $k = 1$, and all importance parameters are fixed to $p_w^1 = 1$ for all tokens $w \in \mathcal{T}$, hash embeddings are equivalent to using the hashing trick. If furthermore the number of component vectors is set to $B = |\mathcal{T}|$ and the hash function $h_1(w)$ is the identity function, hash embeddings are equivalent to standard embeddings.

[1] the small performance difference was observed when using Keras with a Tensorflow backend on a GeForce GTX TITAN X with 12 GB of memory and a Nvidia GeForce GTX 660 with 2GB memory. The performance penalty when using standard embeddings for large vocabulary problems can possibly be avoided by using a custom embedding layer, but we have not pursued this further.

4 Hashing theory

Theorem 4.1. *Let $h : \mathcal{T} \to \{0, \ldots, K\}$ be a hash function. Then the probability p_{col} that $w_0 \in \mathcal{T}$ collides with one or more other tokens is given by*

$$p_{col} = 1 - (1 - 1/K)^{|\mathcal{T}|-1} . \tag{1}$$

For large K we have the approximation

$$p_{col} \approx 1 - e^{-\frac{|\mathcal{T}|}{K}} . \tag{2}$$

The expected number of tokens in collision C_{tot} is given by

$$C_{tot} = |\mathcal{T}| p_{col} . \tag{3}$$

Proof. This is a simple variation of the "birthday problem".

□

When using hashing for dimensionality reduction, collisions are unavoidable, which is the main disadvantage for feature hashing. This is counteracted by hash embeddings in two ways:

First of all, for choosing the component vectors for a token $w \in \mathcal{T}$, hash embeddings use k independent uniform hash functions $h_i : \mathcal{T} \to \{1, \ldots, B\}$ for $i = 1, \ldots, k$. The combination of multiple hash functions approximates a single hash function with much larger range $h : \mathcal{T} \to \{1, \ldots, B^k\}$, which drastically reduces the risk of total collisions. With a vocabulary of $|\mathcal{T}| = 100$M, $B = 1$M different component vectors and just $k = 2$ instead of 1, the chance of a given token colliding with at least one other token in the vocabulary is reduced from approximately $1 - \exp\left(-10^8/10^6\right) \approx 1$ to approximately $1 - \exp\left(-10^8/10^{12}\right) \approx 0.0001$. Using more hash functions will further reduce the number of collisions.

Second, only a small number of the tokens in the vocabulary are usually important for the task at hand. The purpose of the importance parameters is to implicitly prune unimportant words by setting their importance parameters close to 0. This would reduce the expected number of collisions to $|\mathcal{T}_{\text{imp}}| \cdot \exp\left(-\frac{|\mathcal{T}_{\text{imp}}|}{B}\right)$ where $\mathcal{T}_{\text{imp}} \subset \mathcal{T}$ is the set of important words for the given task. The weighting with the component vector will further be able to separate the colliding tokens in the k dimensional subspace spanned by their k d dimensional embedding vectors.

Note that hash embeddings consist of two layers of hashing. In the first layer each token is simply translated to an integer in $\{1, \ldots, K\}$ by a dictionary or a hash function D_1. If D_1 is a dictionary, there will of course not be any collisions in the first layer. If D_1 is a random hash function then the expected number of tokens in collision will be given by equation 3. These collisions cannot be avoided, and the expected number of collisions can only be decreased by increasing K. Increasing the vocabulary size by 1 introduces d parameters in standard embeddings and only k in hash embeddings. The typical d ranges from 10 to 300, and k is in the range 1-3. This means that even when the embedding size is kept small, the parameter savings can be huge. In (Joulin et al., 2016b) for example, the embedding size is chosen to be as small as 10. In order to go from a bi-gram model to a general n-gram model the number of buckets is increased from $K = 10^7$ to $K = 10^8$. This increase of buckets requires an additional 900 million parameters when using standard embeddings, but less than 200 million when using hash embeddings with the default of $k = 2$ hash functions. I.e. even when the embedding size is kept extremely small, the parameter savings can be huge.

5 Experiments

We benchmark hash embeddings with and without dictionaries on text classification tasks.

5.1 Data and preprocessing

We evaluate hash embeddings on 7 different datasets in the form introduced by Zhang et al. (2015) for various text classification tasks including topic classification, sentiment analysis, and news categorization. All of the datasets are balanced so the samples are distributed evenly among the

classes. An overview of the datasets can be seen in table 1. Significant previous results are listed in table 2. We use the same experimental protocol as in (Zhang et al., 2015).

We do not perform any preprocessing besides removing punctuation. The models are trained on snippets of text that are created by first converting each text to a sequence of n-grams, and from this list a training sample is created by randomly selecting between 4 and 100 consecutive n-grams as input. This may be seen as input drop-out and helps the model avoid overfitting. When testing we use the entire document as input. The snippet/document-level embedding is obtained by simply adding up the word-level embeddings.

Table 1: Datasets used in the experiments, See (Zhang et al., 2015) for a complete description.

	#Train	#Test	#Classes	Task
AG's news	120k	7.6k	4	English news categorization
DBPedia	450k	70k	14	Ontology classification
Yelp Review Polarity	560k	38k	2	Sentiment analysis
Yelp Review Full	560k	50k	5	Sentiment analysis
Yahoo! Answers	650k	60k	10	Topic classification
Amazon Review Full	3000k	650k	5	Sentiment analysis
Amazon Review Polarity	3600k	400k	2	Sentiment analysis

5.2 Training

All the models are trained by minimizing the cross entropy using the stochastic gradient descent-based *Adam* method (Kingma and Ba, 2014) with a learning rate set to $\alpha = 0.001$. We use early stopping with a patience of 10, and use 5% of the training data as validation data. All models were implemented using Keras with TensorFlow backend. The training was performed on a Nvidia GeForce GTX TITAN X with 12 GB of memory.

5.3 Hash embeddings without a dictionary

In this experiment we compare the use of a standard hashing trick embedding with a hash embedding. The hash embeddings use $K = 10$M different importance parameter vectors, $k = 2$ hash functions, and $B = 1$M component vectors of dimension $d = 20$. This adds up to 40M parameters for the hash embeddings. For the standard hashing trick embeddings, we use an architecture almost identical to the one used in (Joulin et al., 2016b). As in (Joulin et al., 2016b) we only consider bi-grams. We use one layer of hashing with 10M buckets and an embeddings size of 20. This requires 200M parameters. The document-level embedding input is passed through a single fully connected layer with softmax activation.

The performance of the model when using each of the two embedding types can be seen in the left side of table 2. We see that even though hash embeddings require 5 times less parameters compared to standard embeddings, they perform at least as well as standard embeddings across all of the datasets, except for DBPedia where standard embeddings perform a tiny bit better.

5.4 Hash embeddings using a dictionary

In this experiment we limit the vocabulary to the 1M most frequent n-grams for $n < 10$. Most of the tokens are uni-grams and bi-grams, but also many tokens of higher order are present in the vocabulary. We use embedding vectors of size $d = 200$. The hash embeddings use $k = 2$ hash functions and the bucket size B is chosen by cross-validation among [500, 10K, 50K, 100K, 150K]. The maximum number of words for the standard embeddings is chosen by cross-validation among [10K, 25K, 50K, 300K, 500K, 1M]. We use a more complex architecture than in the experiment above, consisting of an embedding layer (standard or hash) followed by three dense layers with 1000 hidden units and ReLU activations, ending in a softmax layer. We use batch normalization (Ioffe and Szegedy, 2015) as regularization between all of the layers.

The parameter savings for this problem are not as great as in the experiment without a dictionary, but the hash embeddings still use 3 times less parameters on average compared to a standard embedding.

As can be seen in table 2 the more complex models actually achieve a *worse* result than the simple model described above. This could be caused by either an insufficient number of words in the vocabulary or by overfitting. Note however, that the two models have access to the same vocabulary, and the vocabulary can therefore only explain the general drop in performance, not the performance difference between the two types of embedding. This seems to suggest that using hash embeddings have a regularizing effect on performance.

When using a dictionary in the first layer of hashing, each vector of importance parameters will correspond directly to a unique phrase. In table 4 we see the phrases corresponding to the largest/smallest (absolute) importance values. As we would expect, large absolute values of the importance parameters correspond to important phrases. Also note that some of the n-grams contain information that e.g. the bi-gram model above would not be able to capture. For example, the bi-gram model would not be able to tell whether 4 or 5 stars had been given on behalf of the sentence *"I gave it 4 stars instead of 5 stars"*, but the general n-gram model would.

5.5 Ensemble of hash embeddings

The number of buckets for a hash embedding can be chosen quite small without severely affecting performance. $B = 500 - 10.000$ buckets is typically sufficient in order to obtain a performance almost at par with the best results. In the experiments using a dictionary only about 3M parameters are required in the layers on top of the embedding, while $kK + Bd = 2M + B \times 200$ are required in the embedding itself. This means that we can choose to train an ensemble of models with small bucket sizes instead of a large model, while at the same time use the same amount of parameters (and the same training time since models can be trained in parallel). Using an ensemble is particularly useful for hash embeddings: even though collisions are handled effectively by the word importance parameters, there is still a possibility that a few of the important words have to use suboptimal embedding vectors. When using several models in an ensemble this can more or less be avoided since different hash functions can be chosen for each hash embedding in the ensemble.

We use an ensemble consisting of 10 models and combine the models using soft voting. Each model use $B = 50.000$ and $d = 200$. The architecture is the same as in the previous section except that models with one to three hidden layers are used instead of just ten models with three hidden layers. This was done in order to diversify the models. The total number of parameters in the ensemble is approximately 150M. This should be compared to both the standard embedding model in section 5.3 and the standard embedding model in section 5.4 (when using the full vocabulary), both of which require $\approx 200M$ parameters.

Table 2: Test accuracy (in %) for the selected datasets

	Without dictionary Shallow network (section 5.3)		**With dictionary** Deep network (section 5.4)		
	Hash emb.	Std emb	Hash emb.	Std. emb.	Ensemble
AG	**92.4**	92.0	91.5	91.7	92.0
Amazon full	60.0	58.3	59.4	58.5	**60.5**
Dbpedia	98.5	98.6	98.7	98.6	**98.8**
Yahoo	72.3	72.3	71.3	65.8	**72.9**
Yelp full	**63.8**	62.6	62.6	61.4	62.9
Amazon pol	94.4	94.2	94.7	93.6	**94.7**
Yelp pol	**95.9**	95.5	95.8	95.0	95.7

6 Future Work

Hash embeddings are complementary to other state-of-the-art methods as it addresses the problem of large vocabularies. An attractive possibility is to use hash-embeddings to create a word-level embedding to be used in a context sensitive model such as wordCNN.

As noted in section 3, we have used the same token to id function D_1 for both the component vectors and the importance parameters. This means that words that hash to the same bucket in the first layer get both identical component vectors and importance parameters. This effectively means that those words become indistinguishable to the model. If we instead use a different token to id function $\hat{D}$ for

Table 3: State-of-the-art test accuracy in %. The table is split between BOW embedding approaches (bottom) and more complex rnn/cnn approaches (top). The best result in each category for each dataset is bolded.

	AG	DBP	Yelp P	Yelp F	Yah A	Amz F	Amz P
char-CNN (Zhang et al., 2015)	87.2	98.3	94.7	62.0	71.2	59.5	94.5
char-CRNN (Xiao and Cho, 2016)	91.4	98.6	94.5	61.8	71.7	59.2	94.1
VDCNN (Conneau et al., 2016)	91.3	98.7	95.7	**64.7**	73.4	63.0	95.7
wordCNN (Johnson and Zhang, 2016)	**93.4**	99.2	**97.1**	67.6	**75.2**	**63.8**	**96.2**
Discr. LSTM (Yogatama et al., 2017)	92.1	98.7	92.6	59.6	73.7		
Virt. adv. net. (Miyato et al., 2016)		**99.2**					
fastText (Joulin et al., 2016b)	**92.5**	98.6	95.7	**63.9**	72.3	60.2	94.6
BoW (Zhang et al., 2015)	88.8	96.6	92.2	58.0	68.9	54.6	90.4
n-grams (Zhang et al., 2015)	92.0	98.6	95.6	56.3	68.5	54.3	92.0
n-grams TFIDF (Zhang et al., 2015)	92.4	98.7	95.4	54.8	68.5	52.4	91.5
Hash embeddings (no dict.)	92.4	98.5	**95.9**	63.8	72.3	60.0	94.4
Hash embeddings (dict.)	91.5	98.7	95.8	62.5	71.9	59.4	**94.7**
Hash embeddings (dict., ensemble)	92.0	**98.8**	95.7	62.9	**72.9**	**60.5**	**94.7**

Table 4: Words in the vocabulary with the highest/lowest importance parameters.

	Yelp polarity	Amazon full
Important tokens	What_a_joke, not_a_good_experience, Great_experience, wanted_to_love, and_lacking, Awful, by_far_the_worst,	gave_it_4, it_two_stars_because, 4_stars_instead_of_5, 4_stars, four_stars, gave_it_two_stars
Unimportant tokens	The_service_was, got_a_cinnamon, 15_you_can, while_touching, and_that_table, style_There_is	that_my_wife_and_I, the_state_I, power_back_on, years_and_though, you_want_a_real_good

the importance parameters, we severely reduce the chance of "total collisions". Our initial findings indicate that using a different hash function for the index of the importance parameters gives a small but consistent improvement compared to using the same hash function.

In this article we have represented word vector using a weighed sum of component vectors. However, other aggregation methods are possible. One such method is simply to concatenate the (weighed) component vectors. The resulting kd-dimensional vector is then equivalent to a weighed sum of orthogonal vectors in $\mathbb{R}^{kd}$.

Finally, it might be interesting to experiment with pre-training lean, high-quality hash vectors that could be distributed as an alternative to word2vec vectors, which require around 3.5 GB of space for almost a billion parameters.

7 Conclusion

We have described an extension and improvement to standard word embeddings and made an empirical comparisons between hash embeddings and standard embeddings across a wide range of classification tasks. Our experiments show that the performance of hash embeddings is always at par with using standard embeddings, and in most cases better.

We have shown that hash embeddings can easily deal with huge vocabularies, and we have shown that hash embeddings can be used both with and without a dictionary. This is particularly useful for problems such as online learning where a dictionary cannot be constructed before training.

Our experiments also suggest that hash embeddings have an inherent regularizing effect on performance. When using a standard method of regularization (such as L_1 or L_2 regularization), we start with the full parameter space and regularize parameters by pushing some of them closer to 0. This is in contrast to regularization using hash embeddings where the number of parameters (number of buckets) determines the degree of regularization. Thus parameters not needed by the model will not have to be added in the first place.

The hash embedding models used in this article achieve equal or better performance than previous bag-of-words models using standard embeddings. Furthermore, in 5 of 7 datasets, the performance of hash embeddings is in top 3 of state-of-the art.

References

Argerich, L., Zaffaroni, J. T., and Cano, M. J. (2016). Hash2vec, feature hashing for word embeddings. *CoRR*, abs/1608.08940.

Bai, B., Weston, J., Grangier, D., Collobert, R., Sadamasa, K., Qi, Y., Chapelle, O., and Weinberger, K. (2009). Supervised semantic indexing. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 187–196. ACM.

Conneau, A., Schwenk, H., Barrault, L., and LeCun, Y. (2016). Very deep convolutional networks for natural language processing. *CoRR*, abs/1606.01781.

Gray, R. M. and Neuhoff, D. L. (1998). Quantization. *IEEE Trans. Inf. Theor.*, 44(6):2325–2383.

Huang, E. H., Socher, R., Manning, C. D., and Ng, A. Y. (2012). Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL '12, pages 873–882, Stroudsburg, PA, USA. Association for Computational Linguistics.

Huang, P.-S., He, X., Gao, J., Deng, L., Acero, A., and Heck, L. (2013). Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management (CIKM)*, pages 2333–2338.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

Jegou, H., Douze, M., and Schmid, C. (2011). Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128.

Johansen, A. R., Hansen, J. M., Obeid, E. K., Sønderby, C. K., and Winther, O. (2016). Neural machine translation with characters and hierarchical encoding. *CoRR*, abs/1610.06550.

Johnson, R. and Zhang, T. (2016). Convolutional neural networks for text categorization: Shallow word-level vs. deep character-level. *CoRR*, abs/1609.00718.

Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., and Mikolov, T. (2016a). Fasttext.zip: Compressing text classification models. *CoRR*, abs/1612.03651.

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016b). Bag of tricks for efficient text classification. *CoRR*, abs/1607.01759.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Kulis, B. and Darrell, T. (2009). Learning to hash with binary reconstructive embeddings. In Bengio, Y., Schuurmans, D., Lafferty, J. D., Williams, C. K. I., and Culotta, A., editors, *Advances in Neural Information Processing Systems 22*, pages 1042–1050. Curran Associates, Inc.

Manning, C. D., Schütze, H., et al. (1999). *Foundations of statistical natural language processing*, volume 999. MIT Press.

Mihált, M. (2016). Google's trained word2vec model in python. `https://github.com/mmihaltz/word2vec-GoogleNews-vectors`. Accessed: 2017-02-08.

Miyato, T., Dai, A. M., and Goodfellow, I. (2016). Virtual adversarial training for semi-supervised text classification. *stat*, 1050:25.

Reisinger, J. and Mooney, R. J. (2010). Multi-prototype vector-space models of word meaning. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 109–117, Stroudsburg, PA, USA. Association for Computational Linguistics.

Stolcke, A. (2000). Entropy-based pruning of backoff language models. *CoRR*, cs.CL/0006025.

Weinberger, K. Q., Dasgupta, A., Attenberg, J., Langford, J., and Smola, A. J. (2009). Feature hashing for large scale multitask learning. *CoRR*, abs/0902.2206.

Xiao, Y. and Cho, K. (2016). Efficient character-level document classification by combining convolution and recurrent layers. *CoRR*, abs/1602.00367.

Yogatama, D., Dyer, C., Ling, W., and Blunsom, P. (2017). Generative and discriminative text classification with recurrent neural networks. *arXiv preprint arXiv:1703.01898*.

Zhang, X., Zhao, J. J., and LeCun, Y. (2015). Character-level convolutional networks for text classification. *CoRR*, abs/1509.01626.

Appendix D

Zero Shot Cross language Text Classification

ZERO-SHOT CROSS LANGUAGE TEXT CLASSIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Labeled text classification datasets are typically only available in a few select languages. In order to train a model for e.g news categorization in a language L_t without a suitable text classification dataset there are two options. The first option is to create a new labeled dataset by hand, and the second option is to transfer label information from an existing labeled dataset in a source language L_s to the target language L_t. In this paper we propose a method for sharing label information across languages by means of a language independent text encoder. The encoder will give almost identical representations to multilingual versions of the same text. This means that labeled data in one language can be used to train a classifier that works for the rest of the languages. The encoder is trained independently of any concrete classification task and can therefore subsequently be used for any classification task. We show that it is possible to obtain good performance even in the case where only a comparable corpus of texts is available.

1 INTRODUCTION

Automatic systems that can classify documents quickly and precisely are useful for a wide range of practical applications. For example, organizations may be interested in using *sentiment analysis* of opinion posts such as tweets that mention their products and services. By classifying the sentiment of each post (e.g. positive, neutral, or negative), the organization can for example learn which parts of a product should be improved.

Creating a suitable, large labeled dataset for training a classification model requires a lot of effort and available public datasets are typically only available in the most common languages. In order to train a classification model for a languages L_t without a suitable text classification dataset there are two options: The first option is of course to create a new labeled dataset from scratch, and the second option is to use the label information in existing labeled datasets in a language L_s and then transfer this label information to L_t. The first option usually requires a great amount of work and is typically not a viable solution. The second option is called cross-language text classification (CLTC) (Wan, 2009).

In this article we present a method for performing CLTC by means of a *universal encoder*. The method consists of two steps. In the first step, a *universal encoder* is trained to give similar representations to texts that describe the same topic, even if the texts are in different languages. In the second step, a classification module uses the language-independent representations from the universal encoder as inputs and is trained to predict which category each document belongs to. Compared to previous work, this method has several advantages:

1. The universal encoder can be trained using just a *comparable corpus*. A comparable corpus is a corpus where the multilingual versions of a document are not necessarily translations of each other, but merely about the same topic.
2. It enables zero-shot classification. I.e. if we have a comparable corpus in French-Spanish, we can build a classifier for Spanish by using a labeled dataset in English.
3. The universal encoder does not rely on single word translations, but rather on encoding entire contexts. This can help alleviate ambiguity problems caused by polysemy.
4. The input language does not have to be specified at test time.

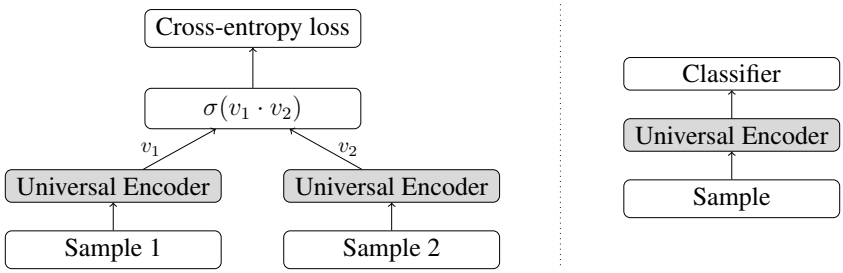


Figure 1: Illustration of how the *Universal Encoder* is trained (left) and how it is used when training and making predictions using the classifier (right). All *Universal Encoders* (gray background) are exactly identical by sharing architecture and weights

The method presented in this article is conceptually similar in spirit to Google's zero-shot machine *translation* model (Johnson et al., 2016), which is used in the Google translate API. That model also uses a shared vocabulary and a language independent encoder. It does, however, require a large corpus of aligned sentences for training. Additionally, translating a text is a much harder problem than merely extracting discriminative features since it requires encoding of e.g. syntactic information that is not necessary for text classification. Therefore such a model is much more complex than it needs to be, and a more parsimonious model is therefore preferable. We will compare our zero-shot *classification* model with an equivalent model based on the zero-shot *translation* model in section 3.

The rest of the article is organized as follows: We present our CLTC model in section 2. Experiments, data and results are presented in section 3. In section 4 we review previous approaches to cross-lingual text classification. In section 5 we will take a look at some possible improvements and future directions for the method. Finally, we conclude the article in section 6.

2 Alignment of document representations

The zero-shot classification model consists of two independent components, a universal encoder and a classifier module. The two components will be described in the following.

2.1 Universal encoder

The universal encoder provides a language independent encoding of all information useful in order to identify a text. If the language independent representation is very close for comparable texts, a classifier for representations in one language should be usable across all languages.

The universal encoder transforms the input text by using a function F_u. The mapping F_u could for example consist of a word embedding followed by a recurrent layer, or a sum of word embedding vectors. Or it could be the LSA representation of the input text followed by a dense layer.

To create a training sample for the encoder, a pair of texts are drawn from the corpus. Half of the pairs will consist of two texts on the same topic (but in different languages) and half the pairs will consist of two texts on different topics (and possibly different languages). The encoder is trained (using cross-entropy error) to be able to predict whether two texts s_1 and s_2 are comparable (i.e. about the same topic) based on the inner product $F_u(s_1) \cdot F_u(s_2)$ between their representations. The encoder training setup is illustrated on the left side of fig. 1.

2.2 Classifier

The output of the universal encoder is a language independent representation of a text. Training a classifier based on the universal encoder will give a universal classifier that can be used to classify text in any language that the encoder has been trained on. The universal classifier is depicted on the right hand side of fig. 1.

3 EXPERIMENTS

We compare our zero-shot classification model with two other models on the task of predicting the category of Italian Wikipedia articles.

- **A monolingual classifier**. The performance of this model can be considered an upper bound for the performance. However, in a realistic scenario we would only have a very limited number of native samples available. We therefore train the model several times using a varying number of native samples from as little as 5k to more than a hundred thousand. This will give the performance of the native classifier as a function of number of available native samples, and will enable us to tell how many native samples are required in order to obtain a performance equivalent to the that of the CLTC models.
- **A model based on machine trainslation.** As mentioned, Google's zero-shot translation model is conceptually similar to our model. It is therefore natural to compare a model based on that model with the zero-shot classification model presented in this paper.

3.1 DATA

We evaluate the universal encoder using Wikipedia article abstracts in Italian, German, French and English. Wikipedia inter-language links are used to relate articles in different languages about the same topic (such as "Tom Cruise"). We use DBPedia mapping-based properties[1] to assign each topic to a category (such as Person or City). For pages with multiple categories we select one at random. We restrict the number of classes to the 200 most frequent ones. The number of articles in each language can be seen in table 1. We don't do any pre-processing besides removing punctuation characters (such as `.,:|-_/"*'=`) and tokenizing the text.

Table 1: Number of articles in different languages in the dataset. There are on average 2.3 articles for each Wikipedia topic.

Italian	493k
German	675k
French	796k
English	1457k

The dataset is split evenly into an *encoder dataset* and a *classification dataset*. The encoder dataset is used to train the universal encoder and the classification dataset is used to train the different classification models. When training the encoder, we sample uniformly between pairs of articles with the same topic and pairs of articles with different topic. In the first column of table 2 we see the number of training pairs for the encoder for the different language sets. Column two shows the number of samples in the classifier training set for the different language pairs (i.e. the number of non-Italian articles in the classification dataset). Column three shows the number of samples in the classifier test set for the different language sets (i.e. the number of Italian articles in the classifier dataset). Topics with only one article are removed, and so are topics not belonging to one of the 200 target classes.

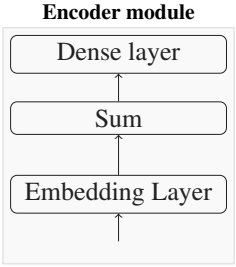
It is important to note that the classification training dataset for our classification model *does not* include any Italian samples.

Table 2: Number of samples for the different language sets. The number of training samples for the encoder denotes the number of pairs of articles about the same topic but in different languages.

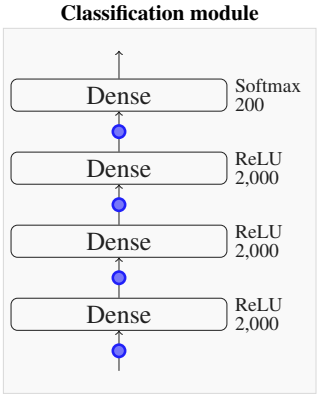
	Encoder train pairs	Classifier train samples	Classifier test samples
It-En	245k	163k	167k
It-En-Fr	813k	864k	167k
It-En-Fr-De	1500k	1305k	167k

The Wikipedia articles are by no means translations of each other. Typically the English articles are much longer compared to articles in other languages and the content is often quite different. We

[1] See http://wiki.dbpedia.org/services-resources/datasets/dbpedia-datasets#h434-10 for a description.



(a) The architecture of the encoder.



(b) The architecture of the classification module of the neural models. The numbers to the right of each layer is the activation function of the layer (above) and the number of units (below). The blue dots represent dropout layers with 50% dropout probability.

therefore use the heuristic that the *essence* of an article is contained in its beginning. As a consequence, the semantic content of the beginnings of the articles in different languages is hopefully more similar than the entire texts. For this reason we use only the first 200 words of each article.

We do not use the entire remaining article text as input samples. Instead we draw a random *snippet* from the text. A snippet from a text document is a small sequence of random length consisting of 3-200 tokens. Note that the snippet idea can be seen as an extreme form of input dropout (we dropout everything but a small fraction of the text). That is, the snippet sampling provides a degree of regularization.

3.2 MODELS

The architectures of the three models have been kept as similar as possible in order to provide the most fair comparison. We use Adam updates (Kingma & Ba, 2014) with default parameters (except for a learning rate set to 10^{-4}) for training of all of the models. The architectures of the different models will be described in the following.

3.2.1 ZERO-SHOT CLASSIFICATION MODEL

The architecture of the universal encoder can be seen in fig. 2a. The embedding layer is shared among all input languages and therefore the vocabulary can become quite large. As we will see in section 3.3, it is beneficial to choose the embedding vector size as large as possible. We therefore use a word based hash embedding in the input layer (Svenstrup et al., 2017). A hash embedding is a recently proposed improvement to a regular embedding that requires far less parameters for large vocabulary problems or problems with large embedding sizes. It is thus perfect for our purpose. We use a hash embedding with 25k buckets and two hash functions for all experiments. The embedding vector size is varied between 250 and 1500 in the experiments.

Following the embedding layer we have a sum layer where the coordinate-wise sum of all word vectors is computed. On top of the sum layer we use a dense layer with twice the number of units as the embedding vector size.

Once the encoder has been trained we freeze the parameters and use the universal encoder as input to the classification module (see right side of fig. 1). The architecture of the classification module is illustrated in fig. 2b. It is a neural network consisting of three dense layers with 2000 units with rectified linear units activation, followed by a dense layer with 200 units (number of classes) with

softmax activation. We use dropout (Srivastava et al., 2014) with a 50% dropout probability as regularization between all the layers.

3.2.2 NATIVE CLASSIFIER

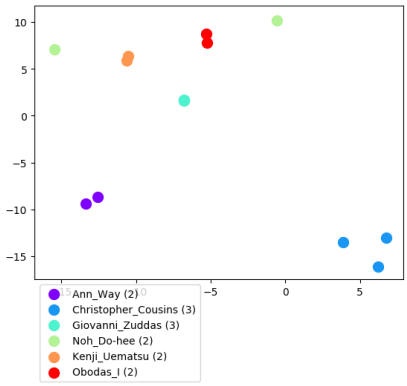
The architecture of the native classifier is identical to the architecture of the universal encoder + classifier described above. The differences are that we train the encoder and classifier simultaneously, and that the model is both trained and tested on Italian samples. It is trained on 3.5k, 17k, 34k, 68k, 137k and 158k samples. It is tested on 8k Italian samples.

3.2.3 CLASSIFIER BASED ON MACHINE TRANSLATION

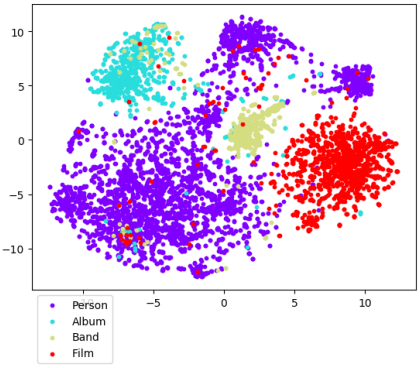
The architecture of this model is identical to the architecture of the native classifier. The differences are that we first train on all *English* samples in the classification dataset and then test on a testset of Italian samples translated to English by using Google translate.

3.3 RESULTS

3.3.1 THE UNIVERSAL ENCODER



(a) t-SNE plot of universal representations of articles about persons. The articles are from the classifier dataset so the encoder has not been trained on the articles.



(b) t-SNE plot of universal representations of multilingual articles of different categories. The articles are from the classifier dataset and the encoder has therefore never seen the the documents before. The plot shows good separation between the categories.

The purpose of the universal encoder is to provide representations for articles such that articles about *different* topics are distant from each other, while articles on the *same* topic are almost identical, even if the articles are in different languages. Figure 3a displays a t-SNE plot (Maaten & Hinton, 2008) of the universal representations of a few articles about persons. The plot shows good separation between articles on different people, and a low separation between articles about the same person in different languages.

The t-SNE plot in fig. 3b shows universal representations of articles from multiple different languages and categories. Even though the encoder wasn't ever explicitly trained for that purpose, it easy to see the articles from different categories are nicely separated even though some of the categories are very similar.

3.4 CLASSIFICATION RESULTS

We report the accuracy for the different models in table 3. We see that the zero-shot classifier obtains very good results, especially in a bilingual setting where the largest classifier (embedding size 1500)

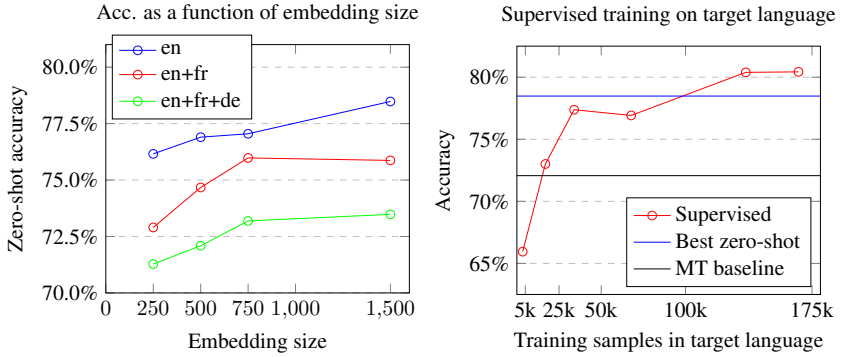
obtains a very high accuracy of 78.5%. This is far better than the model based on machine translation (72.1%) and very close to the upper bound on accuracy of 80.4% (the native classifier).

In fig. 4a we see accuracy as a function of the embedding size for the different language sets. We note that the accuracy increases monotonically with the embedding size for all language sets. However, this increase seems to be less pronounced for larger embedding sizes. We also note that the accuracy unfortunately decreases with the number of languages. The reason for this decrease in performance is less clear and does not seem to be caused by lack of capacity in the encoder. We believe that it may be caused by the sampling method used. We sample uniformly from topics and then select a random language pair. Since there are much more articles in English, French and German than in Italian (see table 1), there will be many more sampled pairs from these languages. This may cause the encoder to favour these languages. We have, however, not investigated this further.

Table 3: Test accuracy [%] of the different classification models. ZSC denotes our zero-shot classification model. All results are with an embedding size of 1500. Since the dataset is not balanced we have included a largest class classifier in the results.

Model	**Accuracy**
ZSC (It-En)	78.5
ZSC (It-En-Fr)	75.9
ZSC (It-En-Fr-De)	73.5
Machine translation	72.1
Native classifier	80.4
Largest class	9.0

A typical (realistic) scenario for use of CLTC methods is where a small amount of native, labeled data is available, but where a large comparable corpus is available. In that case our zero-shot classification model will have a higher performance compared to the monolingual native classifier. As we see in fig. 4b the native monolingual classifier needs about 100k native samples in order to obtain a higher performance compared to our zero-shot classification model.



(a) Accuracy as a function of embedding size for the different language sets.

(b) Accuracy as a function of training samples in target language (Italian) for a native classifier

4 Related Work

Several strategies for multilingual text classifications have previously been proposed. The different strategies can be grouped into three groups according to whether they require a corpus of aligned sentences (a parallel corpus), a comparable corpus or just a dictionary. The different approaches will be described in the following.

4.1 APPROACHES REQUIRING A PARALLEL CORPUS

The main problem with these methods is of course that an aligned corpus is typically only available for a handful of language pairs. One such corpus is the Europarl dataset for machine translation (Koehn, 2005), which is available in 21 European languages.

4.1.1 MULTILINGUAL WORD EMBEDDINGS

A word embedding is a mapping from a set of words to a set of dense real-valued vectors. Such representations of words have proven to be very useful in many monolingual natural language processing problems. By exploiting that words occurring in the same type of context have similar meaning, it is possible to perform unsupervised training of word embeddings such that words with the same meaning have similar vector representation (Bengio et al., 2003). In (Klementiev et al., 2012; Chandar et al., 2014), this property is extended to a multilingual setting such that words with the same meaning, but possibly in different languages, have similar representation. Klementiev et al. (2012) uses a multitask learning algorithm for training the embeddings, and Chandar et al. (2014) uses a method based on autoencoding.

Once created, the multilingual word embeddings can be used to perform cross-lingual text classification. Note that just as the universal encoder introduced in this article, the word embeddings are not trained with any specific classification task in mind. I.e. the word embeddings can be trained once and subsequently be used in a variety of classification tasks.

Multilingual word embeddings can also be trained using only a dictionary, see below.

4.1.2 APPROACHES BASED ON MACHINE TRANSLATION

A lot of work in cross-language text classification has relied on the availability of machine translation models. Most of the methods use a two step process where features are either extracted and then translated (Shi et al., 2010; Montalvo et al., 2007; Wei & Pal, 2010), or are extracted from the translated text (Wan, 2009; Ling et al., 2008; Rigutini et al., 2005).

Methods based on machine translation are attractive because they are typically intuitive and easy to understand. Unfortunately, the machine translation step introduces a lot of noise in the form of information loss, translation error and noise due to the discrepancy between data distributions of the different languages. Several methods have been proposed in order to reduce the performance penalty induced by the translation step. These methods include model translation based on the EM algorithm (Rigutini et al., 2005; Shi et al., 2010), and methods based on domain adaption (Wei & Pal, 2010; Blitzer et al., 2006)

4.2 APPROACHES REQUIRING A COMPARABLE CORPUS

A comparable corpus is typically much easier to obtain than a parallel corpus since it merely requires that the topic of a text is known across languages. For example, news articles are typically tagged with categories such as finance or sports and these categories can be used to create a comparable corpus for news classification. Wikipedia is another very good example of a comparable corpus.

4.2.1 LDA APPROACHES

Latent Dirichlet Allocation (LDA) is a Bayesian Network model where each document is assumed to have a latent topic distribution. For each topic there is a corresponding word distribution. In order to create a document of N words, we first draw a topic distribution T_d for the document. Each of the N words in the document is then generated by first drawing a word topic from T_d, and then use the word distribution corresponding to the topic to generate the word. This can easily be extended to a multilingual setting by letting documents with the same content (but in different languages) share the same topic distribution. The topic distribution of a document can then be estimated after training. Documents with similar topic distribution will tend to be semantically similar. This property can be used to train cross-lingual classifiers. Multilingual LDA approaches to cross-language text classification have been explored in (De Smet et al., 2011; Ni et al., 2011).

4.2.2 MULTI-VIEW LEARNING

In multi-view learning it is assumed that different language versions of a document describe the same data object. The representation of the views should therefore be similar. There are several variants of the multi-view learning method but typically the method consists in optimizing a set of monolingual classifiers subject to the constraint that the representation of the views should be similar (Amini & Goutte, 2010; Wan, 2009; Guo & Xiao, 2012). The different views are often constructed using machine translation, however.

4.3 APPROACHES REQUIRING ONLY A DICTIONARY

These approaches are attractive due to the low requirement on data alignment. However, such methods have an inherent problem with poly-synonymous words since they rely on single word translations that ignore the context.

4.3.1 STRUCTURAL CORRESPONDENCE LEARNING

In structural correspondence learning (SCL) (Blitzer et al., 2006; Prettenhofer & Stein, 2010) a set of discriminative words called pivots are identified in a source language and translated to a target language. Each pivot (and its translation) will induce a bisection of the union of the texts in target and source languages. For each pivot a simple linear classifier is trained to predict if a text (with all occurrences of the pivot deleted) contains the pivot. The information contained in the parameters of all of the classifiers are then used to create a bilingual classifier. SCL can be trained using just a corpus of labeled data in a source language, translations of the pivots and an unlabeled corpus for the target language. SCL has shown a performance equal to that of models based on machine translation (Blitzer et al., 2006).

4.3.2 MULTILINGUAL WORD EMBEDDINGS

Multilingual word embeddings can also be trained using only a dictionary and unlabeled text (Wick et al., 2016). This can be done by switching words with the same meaning in different languages. E.g. the sentence *The red hand* could be modified to *The rojo hand* using a dictionary. These artificially modified sentences can then be used to train multilingual word embeddings by using e.g. a CBOW (Mikolov et al., 2013) model. There are some challenges to using this kind of model, however. For example, the method relies on the fact that the meaning of a word is determined by its context (the distributional hypothesis). But in the example above readers familiar with Spanish grammar rules will know that rojo and hand would not belong to the same context (but roja and hand would).

5 FUTURE WORK

There are several interesting directions for future work. First of all, it would be interesting to experiment with more complex versions of F_u such as recurrent nets.

In the experiments presented in the article we only did a small amount of experimentation with e.g. the size of the embedding layer and the transformation to a language independent representation. We believe that even better results may be obtained by hyper-parameter optimization.

In this article we used Wikipedia both when training the encoder and the classifier. However, if the discriminative features in the encoder corpus is sufficiently close to the discriminative features in the classifier corpus it is perhaps possible to use an encoder corpus that is different from the classifier corpus. E.g. we could train the encoder on Wikipedia articles and then use the encoding to classify news articles.

Our results show that performance decreases with the number of languages. As mentioned in section 3.3, this might be caused by the sampling method. It would be interesting to test this hypothesis by super sampling the smaller languages.

Finally, hinge loss would probably have been a more natural loss function for the encoder instead of cross-entropy loss. It could be interesting to see if performance could be improved by changing the loss function.

6 CONCLUSION

In this article we have shown how to create a language independent representation using only a corpus of comparable texts. The language independent representation can subsequently be used for zero-shot classification.

We show that it is possible to obtain very good performance even when only a comparable corpus of texts is available.

The unsupervised classifier of course does not perform better than a supervised classifier trained on the same number of samples. It is, however, equal in performance to a native language supervised classifier trained on about hundred thousand samples. This means that if the number of native samples is limited and a large comparable corpus is available, the performance of our zero-shot classification can be better than that of a monolingual classifier.

Our results show that even though it is possible to obtain good results using several languages at once, the best performance is obtained by using only two languages. Our results furthermore show that it is necessary to use a very large embedding size in order to obtain the best possible performance.

REFERENCES

Massih-Reza Amini and Cyril Goutte. A co-classification approach to learning from multilingual corpora. *Machine learning*, 79(1-2):105–121, 2010.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 conference on empirical methods in natural language processing*, pp. 120–128. Association for Computational Linguistics, 2006.

A P Sarath Chandar, Stanislas Lauly, Hugo Larochelle, Mitesh M Khapra, Balaraman Ravindran, Vikas Raykar, and Amrita Saha. An autoencoder approach to learning bilingual word representations. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NIPS'14, pp. 1853–1861, Cambridge, MA, USA, 2014. MIT Press. URL http://dl.acm.org/citation.cfm?id=2969033.2969034.

Wim De Smet, Jie Tang, and Marie-Francine Moens. Knowledge transfer across multilingual corpora via latent topics. *Advances in Knowledge Discovery and Data Mining*, pp. 549–560, 2011.

Yuhong Guo and Min Xiao. Cross language text classification via subspace co-regularized multi-view learning. *arXiv preprint arXiv:1206.6481*, 2012.

Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al. Google's multilingual neural machine translation system: enabling zero-shot translation. *arXiv preprint arXiv:1611.04558*, 2016.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

Alexandre Klementiev, Ivan Titov, and Binod Bhattarai. Inducing crosslingual distributed representations of words. *Proceedings of COLING 2012*, pp. 1459–1474, 2012.

Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pp. 79–86, 2005.

Xiao Ling, Gui-Rong Xue, Wenyuan Dai, Yun Jiang, Qiang Yang, and Yong Yu. Can chinese web pages be classified with english data source? In *Proceedings of the 17th international conference on World Wide Web*, pp. 969–978. ACM, 2008.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Soto Montalvo, R Martínez, Arantza Casillas, and Víctor Fresno. Multilingual news clustering: Feature translation vs. identification of cognate named entities. *Pattern Recognition Letters*, 28 (16):2305–2311, 2007.

Xiaochuan Ni, Jian-Tao Sun, Jian Hu, and Zheng Chen. Cross lingual text classification by mining multilingual topics from wikipedia. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pp. 375–384. ACM, 2011.

Peter Prettenhofer and Benno Stein. Cross-language text classification using structural correspondence learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pp. 1118–1127. Association for Computational Linguistics, 2010.

Leonardo Rigutini, Marco Maggini, and Bing Liu. An em based training algorithm for cross-language text categorization. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, pp. 529–535. IEEE Computer Society, 2005.

Lei Shi, Rada Mihalcea, and Mingjun Tian. Cross language text classification by model translation and semi-supervised learning. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pp. 1057–1067. Association for Computational Linguistics, 2010.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Dan Svenstrup, Jonas Meinertz Hansen, and Ole Winther. Hash embeddings for efficient word representations. In *Proceedings of the Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

Xiaojun Wan. Co-training for cross-lingual sentiment classification. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-volume 1*, pp. 235–243. Association for Computational Linguistics, 2009.

Bin Wei and Christopher Pal. Cross lingual adaptation: an experiment on sentiment classifications. In *Proceedings of the ACL 2010 conference short papers*, pp. 258–262. Association for Computational Linguistics, 2010.

Michael Wick, Pallika Kanani, and Adam Pocock. Minimally-constrained multilingual embeddings via artificial code-switching. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pp. 2849–2855. AAAI Press, 2016. URL `http://dl.acm.org/citation.cfm?id=3016100.3016300`.

Bibliography

[BM98] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In Proceedings of the eleventh annual conference on Computational learning theory, pages 92–100. ACM, 1998.

[Cen] Pew Research Center. Health online 2013.

[CGCB14] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555, 2014.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Lstm can solve hard long time lag problems. In Advances in neural information processing systems, pages 473–479, 1997.

[JGBM16] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. CoRR, abs/1607.01759, 2016.

[KB14] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[PBQR+17] Janet Piñero, Àlex Bravo, Núria Queralt-Rosinach, Alba Gutiérrez-Sacristán, Jordi Deu-Pons, Emilio Centeno, Javier García-García, Ferran Sanz, and Laura I Furlong. Disgenet: a comprehensive platform integrating information on human disease-associated genes and variants. Nucleic acids research, 45(D1):D833–D839, 2017.

[Res] Google/Manhattan Research. The doctor's digital path to treatment.

[Rod05] Joachim Rode. Rare diseases: understanding this public health priority, 2005.

[SHK+14] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. Journal of machine learning research, 15(1):1929–1958, 2014.

[SHMW17] Dan Svenstrup, Jonas Meinertz Hansen, Mads Emil Matthiesen, and Ole Winther. Information completion for medical search assistance. Submitted to Journal of Artificial Intelligence, 2017.

[SHW17a] Dan Svenstrup, Jonas Meinertz Hansen, and Ole Winther. Hash embeddings for efficient word representations. arXiv preprint arXiv:1709.03933, 2017.

[SHW17b] Dan Svenstrup, Jonas Meinertz Hansen, and Ole Winther. Zero-shot cross language text classification. Submitted to ICLR 2018, 2017.

[SJW15] Dan Svenstrup, Henrik L Jørgensen, and Ole Winther. Rare disease diagnosis: a review of web search, social media and large-scale data-mining approaches. Rare Diseases, 3(1):e1083145, 2015.

[SPPM15] David Smiley, Eric Pugh, Kranti Parisa, and Matt Mitchell. Apache Solr enterprise search server. Packt Publishing Ltd, 2015.

[Sve17] Dan Svenstrup. Flow framework for text classification. `https://github.com/dsv77/flow`, 2017.

[SW17] Dan Svenstrup and Ole Winther. Performance optimization for specialized domain information retrieval. Submitted to Scientific Reports, 2017.

[WDA+09] Kilian Q. Weinberger, Anirban Dasgupta, Josh Attenberg, John Langford, and Alexander J. Smola. Feature hashing for large scale multitask learning. CoRR, abs/0902.2206, 2009.

[Wya91] Jeremy Wyatt. Information for clinicians: use and sources of medical knowledge. The Lancet, 338(8779):1368–1373, 1991.

[ZG02] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. 2002.