Learning Document Representations for Ranked Retrieval in the Legal Domain

Atle Oftedahl

Thesis submitted for the degree of
Master in Informatics:
Technical and Scientific Applications
(Language Technology Group)
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Autumn 2018

Learning Document Representations for Ranked Retrieval in the Legal Domain

Atle Oftedahl

Abstract

In this work I detail the compilation of a unique corpus of Norwegian court decisions. I utilize this corpus to train several different machine learning models to produce dense semantic vectors for both words and documents. I use the document vectors to perform ranked document retrieval, and evaluate and demonstrate the performance of the vectors for this task using a purposely-built ranked retrieval model utilizing the document references in the documents. Furthermore, I explore the interplay between pre-trained semantic word vectors and convolutional neural networks and conduct several hyperparameter optimization experiments using convolutional neural networks to produce document vectors.

Acknowledgements

First and foremost I want to express my sincere gratitude to my supervisors **Fredrik Jørgensen** and **Erik Velldal**, for not only guiding, motivating and inspiring me throughout this thesis, but for their continued dedication to me and my atypical progression from the first phone call until the end.

I would also like to extend my sincerest gratefulness to **Lovdata** and all my colleagues there for supporting me and allowing me the time, space and resources to undertake this project, as well as for giving me a chance and believing in a young, wide-eyed man at the start of his career.

My genuine thanks and love to Frida for her constant support and encouragement in all my endeavors. Every day I'm grateful for having you by my side for this journey, and for all to come!

Finally, I would like to thank my family for their decades of support and care, and for pushing and shaping me into the person I am today. I can truly never thank you enough!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the recent decades, natural language processing (NLP) have garnered much attention for continually expanding what we thought was possible when it comes to understanding natural language. Automatic translations, sentiment analysis of movie reviews and chatbots are just some of the things which have benefited greatly from the recent achievements in the field. Information retrieval and easier access to information is also undergoing a revolution. But while these new, and often gimmicky, ways for consumers to search for and retrieve information or interact with huge databases of knowledge seem to pop up every other week, many industrial applications of the same technology are lacking.

Law is often thought of as a very slow moving field, where laws and precedences are shaped over years and decades. As new, emergent technology makes its way into more and more of everyday life and professions, law is finally starting to see some change. Apps to automatically file complaints or claim insurance, predictive algorithms trying to guess the outcome of trials and lawyer chatbots giving you legal advice are just some of the new ideas attempting to be realized in the new and blooming field of Legal Tech.

Norway was on the forefront of the more law-focused field of legal informatics in the 1980's and the 1990's (Paliwala, 2010), but as the field got more traction in other parts of the world, Norway could not keep up, and today most the technological advancements are coming from abroad. The line between legal informatics and legal tech is hard to pin down, but it is clear that the increased utilization of NLP and big data is making it harder for Norway to keep up. Although there is a new resurgence in interest in NLP and Legal Tech in Norway, both in academic and professional arenas, the Norwegian language is still largely unexplored. By bringing new technology in the forms of NLP and machine learning together with the leading Norwegian publisher of legal documents and legal informatics pioneer, Lovdata, this thesis aims to bring a small part of the technological revolution to one of the most routinely time consuming tasks in law: document retrieval.

One area where this is especially relevant, is in the field of case law, one of the major pillars of Norwegian law. Previous court decisions define, affirm and affect how law is interpreted and understood by the majority of people in our society. For much of the technological area, digitally searching databases for court decisions has either required you to know exactly which document you were looking for or for you to have to use large and difficult search queries for metadata or keywords. Searching and retrieving legal documents is almost an art, and in the eyes of many in the legal profession, navigating cumbersome queries and rigid search restraints is not a problem to be solved, but a skill to be acquired. But the recent technological advancements have shown us the remarkable results possible with new and intuitive solutions for retrieving information. In this work I will take some of the first exploratory steps in the direction of integrating state-of-the-art technology with the Norwegian field of law and encourage others, both academics and seasoned lawyers alike, to explore and discover new solutions to old problems.

The first aim of this work is to attempt to establish a framework for evaluating ranked document retrieval and document similarity for Norwegian court decisions. This framework involves both compiling and preprocessing a unique document collection, as well as using it to define and evaluate a gold standard for ranked document retrieval. Through the authors affiliation with Lovdata, this project has unique access to the biggest collection of Norwegian court decisions in Norway. With over 130,000 court decisions, I will attempt to formalize a process for evaluating the effectiveness of several document retrieval systems without access to any preexisting gold standards for such a document collection. This gold standard process will need to be fully automatic and work on a case-by-case basis. Unfortunately, due to the sensitivity of personal information in many of the documents in the collection, the document collection and any models created in this work can not be made publicly available.

Document similarity is a rather subjective notion and our intent is not to create a perfect, all-purpose evaluation process, but one which will sufficiently enable us to complete the second aim. The second aim is to evaluate different popular frameworks for creating distributional semantic models used for document retrieval using the data set compiled for this thesis, as well as studying the effects of hyperparameter optimization, preprocessing and evaluation methods on both several popular frameworks as well as several purpose-built deep learning networks. The frameworks evaluated on the dataset in this project include, among others, standard Bag-of-Words models and more advanced representation learning using neural networks, both for words and entire documents. In addition, I will utilize Convolutional Neural Networks as both classifiers and incubators for document representations. With this in mind, it is not our intention to discover the most effective off-the-shelf framework for ranked document retrieval, nor building the most fine-tuned domain specific algorithm for retrieving documents from our collection. Our focus is somewhere in the middle; to explore the new possibilities brought by easy-to-use machine learning frameworks and the benefits of more precisely creating domain specific solutions. I will focus on the mathematical and technological aspects of this project and will try to avoid discussing or making decisions which require greater knowledge of the legal domain than the average computer scientist possess.

1.1 Outline

Chapter 2 gives a theoretical overview of the essential concepts discussed in this work. I present different measures for calculating the performance of document retrieval systems and some common ways of preprocessing documents. I also present the machine learning frameworks which will be used in this project, as well as some central concepts regarding standard neural networks, deep learning and convolutional neural networks. Moreover, I introduce and describe the unique document collection compiled for this thesis, as well as the evaluation process for different machine learning models.

Chapter 3 details the process of creating and performing several baselines experiments to establish a performance benchmark for the different experiments conducted in the rest of the project. The benchmarks will be used in several experiments to study the effects of hyperparameter tuning, initialization methods and preprocessing. Finally, this chapter includes a preliminary discussion on preprocessing and the evaluation process in light of the performed experiments.

Chapter 4 describes the process of estimating and evaluating several sets of word embeddings using the document collection compiled for this work. This includes utilizing a recently proposed set of benchmark data sets for evaluating Norwegian word embeddings. The word embeddings found in this chapter will be used in further experiments and revisited in chapter 6.

Chapter 5 gives a further details convolutional neural networks, as well as describing the preparation for the experiments in chapter 6. This preparation includes preprocessing the documents and constructing and analyzing the classification task used in training the neural networks in chapter 6, as well as the process of extracting document embeddings from a neural network.

Chapter 6 describes all the experiments conducted with the convolutional neural networks. The first part describes a baseline experiment to be used as a benchmark for the other experiments. The results of several different hyperparameter-tuning experiments are discussed, analyzed and compared. Finally, I perform a concluding experiment using the best hyperparameters found in the previous experiments, as well as summarize the results from all the experiments performed in the chapter.

Chapter 7 describes two exploratory experiments performed to help further describe the results from chapter 6. This includes studying the word embeddings from chapter 4 in the context of using them as trainable weights in the CNNs, and how further tuning them in a CNN effect them. I also explore the results of evaluating the document embeddings produced in chapter 6 on an additional ranked retrieval evaluation method which was discussed in chapter 3.

Chapter 8 concludes the work of this thesis and propose suggestions for future work.

Chapter 2

Background

In the first part of this thesis I will give an overview of the fundamental concepts and ideas which this thesis builds upon. I will reference back to this chapter throughout the thesis to avoid repetition and to connect topics together. First I will discuss information retrieval, more specifically document retrieval. Then I will cover some of the simplest ways for evaluating the performance of a document retrieval system. I will expand on these concepts further by discussing ranked retrieval and how we evaluate the performance of a ranked retrieval system. I will then explore the document collection used in this thesis and how I define the gold standard for the ranked retrieval task at the center of this project. Finally, I will discuss how we represent the documents from the document collection, both with sparse feature vectors and dense semantic embeddings. This will lay the ground work for the rest of the discussions.

2.1 Document retrieval

Information retrieval can be defined as (Manning, Raghavan, & Schütze, 2008)

> Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections.

We use the term *document retrieval* when we are only retrieving documents. In this thesis we are only retrieving documents, and thus I will use the term document retrieval instead of information retrieval, even though they can be used almost interchangeably.

Let us define a simple document retrieval system where a user gives the system a query containing a word, and the system retrieves all of the documents which contain the given word. A simple way of implementing this system is to go through each word in each document in the collection and return the documents which contain the query word. If we allow logic operators such as

AND, *OR* and *NOT* in the query, the complexity of our system dramatically increases. As does the potential usefulness of the system.

In this thesis I will build and test several document retrieval systems. We therefore need a way of quantifying *how good* the systems are at retrieving documents. The following sections are based on (Manning et al., 2008).

2.1.1 Accuracy, precision and recall

A simple information retrieval task consist of each document in the collection being labeled as either relevant or not relevant to a given query. After a system has retrieved a set of documents it thinks are relevant to the query, we give each document in the whole collection one of four labels. The documents that were retrieved by the system, and which are relevant to the query, are given the label true positive (TP), and the documents that were not retrieved and where not relevant to the query, are labeled true negative (TN). These are the documents which the system made the correct decision for; to retrieve and to not retrieve, respectively. The documents that were retrieved but where not relevant to the query are known as false positive (FP). The documents that were not retrieved but were relevant to the query are know as false negative (FN).

Obviously we want zero false positives and false negatives, but this is often difficult. When designing a document retrieval system one has to have these errors in mind and how many false positives and false negatives one can accept. These thresholds are dependent on the specific retrieval task. Maybe it is more important that the documents retrieved are mostly true positives, and thus one can accept a larger amount of false negatives errors, but are very strict on false positives.

The first measure is known as *accuracy* and is defined as the ratio between relevant documents returned and the total number of documents in the collection. This is expressed in equation 2.1, utilizing the four labels discussed earlier.

$$Accuracy = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{2.1}$$

For many document retrieval tasks in the real world, where we are only interested in a few documents relative to the size of the collection, accuracy is not a good measure of the performance of the system. The system could simply not return any documents, and according to equation 2.1 it would still achieve a fairly good accuracy, since for the majority of the documents it is correct to not return them. Such a system would be practically useless, even though it has a high accuracy. To get a more informative indication of the model's performance we can instead use the two measures *precision* and *recall*. I will first discuss the precision measure.

Precision is defined as the fraction of retrieved documents which are relevant to the query. See equation 2.2 on the next page for the definition of precision.

$$Precision = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.2}$$

For a moment, let us expand the requirements of the simple document retrieval system discussed earlier. Instead of only returning a list of the documents the system thinks are relevant, we now want this list to be sorted and ranked, with the top documents being the documents the system is most confident in being relevant. This is called *ranked retrieval*, where not only *what* is retrieved is important, but the order they are retrieved in. The further down the list, the less confident the system should be. To be clear, all the documents in the collection are still only considered relevant or not relevant to the query, but the model should treat some documents as *more* relevant than other relevant documents. If a system retrieves a document with a high confidence of it being relevant, and it is correct, we want to grade it better than if it retrieved a relevant document, but with a lower confidence.

To better evaluate the effectiveness of the system's capability to rank the retrieved results, we can calculate not just the precision, but the *average precision* of the returned results. Each of the documents in the retrieved results are either relevant or not relevant for the query, and we start by calculating the precision for every top-k subset of the results; the top document, the top two, the top three and so on. This means we define the subsets as starting at the top result and expanding the selection until the end, or until some position k in the results. We then average those precisions to get a final value, the average precision of the retrieval. By doing this multiple times for different queries and retrievals, and taking the mean of all of those average precisions, we get the final *mean average precision* (MAP) measure for the model. See equation 2.3 for the average precision, and equation 2.4 for the mean average precision.

$$\text{AveragePrecision} = \sum_{k=1}^{N} P(k)\Delta r(k) \tag{2.3}$$

where

$$\begin{aligned} N &= \text{size of the list,} \\ k &= \text{cutoff rank,} \\ P(k) &= \text{precision at a cutoff of k,} \\ \Delta r(k) &= \text{the change in recall between cutoff k-1 and cutoff k} \end{aligned}$$

$$\text{MAP} = \frac{1}{Q} \sum_{q=1}^{Q} \text{AveragePrecision(q)} \tag{2.4}$$

where

$$\begin{aligned} Q &= \text{the amount of different retrievals,} \\ \text{AveragePrecision(q)} &= \text{the average precision of a retrieval q} \end{aligned}$$

The second measure is recall. Recall is defined as the fraction of relevant documents which are retrieved for the query. See equation 2.5 for the definition of recall.

$$Recall = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.5}$$

However, on their own, precision and recall can easily be fooled. For example by returning all documents you will always get a perfect recall score of 1, and by retrieving only a single document, which is relevant, it would yield a perfect precision. In both these cases the other measure would act as a watchdog and expose the 'perfect' score. To avoid having to constantly check and compare the two measures, we combine the two measures into what is known as the F1-measure, as can be seen in equation 2.6.

$$F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{2.6}$$

2.2 Evaluating ranked retrieval

Since precision, recall and the F1-measure is based on the four TP, TN, FP, FN labels, they are not suited to evaluate ranked retrieval, as they do not take into consideration the order of the retrieved documents. We solved this earlier by expanding the precision-measure into the MAP-measure. Let us now expand the requirements for the document retrieval system examples discussed earlier. For each query, let us now give each document in the collection a relevance score instead of a binary relevant/not relevant label. We still want the results retrieved to be ordered by relevance, but now *every* document is suddenly relevant to the query, just some are more than others. To be able to evaluate and compare the ranking of the retrieved list with the 'true' ranking, we need some new measures. In this section I will cover some of the most used measures to evaluate the performance of ranked retrieval with graded relevance. I will start with the NDCG-measure, which is the measure I will primarily use in this work when evaluating ranked retrieval.

2.2.1 NDCG

Normalized Discounted Cumulative Gain (NDCG) is a measure of ranking quality. NDCG makes two assumptions about the data;

1. Relevant documents are worth more when appearing earlier in the list
2. Relevant documents are worth more than less relevant documents, which in turn are worth more than non-relevant documents.

These assumptions mean DCG, without normalization, uses a graded relevance scale for documents in a result list to measures the relevance, or gain, of a document based on its position in the list. The gain from each document is accumulated over the results, with a discounting factor being applied to documents the further down the list they appear. The DCG at a rank position p is given in equation 2.7

$$DCG_p = \sum_{i=1}^{p} \frac{gain_i}{log_2(i+1)} \tag{2.7}$$

When this measure is used in this thesis, the gain for a document is given by its position in a limited list containing the 'true' top 100 documents. This list is called the *gold standard*, and is considered the true ranking of the documents for a query. The gain is then a value between 1 and 100, with the top document in the gold standard list having a gain of 100, and the bottom document having a gain of only 1, with all other documents having a gain of 0, as they are considered not relevant to the query.

The DCG of a list of retrieved documents is calculated by first finding the gain for each document in the list. The gain for a document is based on the position of that document in the gold standard list. Once the gain is found, it is discounted by a function that takes into consideration how far from the top of the retrieved list the document appeared. The further down the retrieved list the relevant document appear, the more the gain is discounted. Then all the discounted gains are accumulated to produce the final DCG score.

The score produced by the DCG algorithm can not be directly compared with another DCG score if they do not use the same parameters. The relevance grading and size of the result set can be different for different queries, and these parameters directly affect the score. If you use the same parameters to produce difference DCG score, the scores can be directly compared. To be able to compare DCG score that use different parameters, it is possible to divide the DCG scores by their ideal DCG score (IDCG), which is the highest possible score for that set of parameters for that query. This will normalize the score to the range $[0, 1]$. This produces the final equation, equation 2.8.

$$NDCG_p = \frac{DCG_p}{IDCG_p} \tag{2.8}$$

It is now possible to compare DCG scores which have used different parameters, as each score is an indication of the performance of the model for its own query, which takes into consideration the limitations or benefits of that query. In this thesis I will only calculate DCG scores with the same parameters. It is therefore not necessary to normalize the score, as each DCG is calculated with the same parameters and the ideal DCG is always the same. However, normalizing the score makes it easier to read and put into perspective, as it is mapped to the familiar range $[0, 1]$, where 1 means a perfect result. For this reason I will use the normalized version of the DCG measure in this thesis.

2.2.2 Average Agreement and Rank-Biased Overlap

Another increasingly popular measure is the Rank-Biased Overlap (RBO), proposed in (Webber, 2010). Given two ranked lists, A and B, RBO defines the *agreement* at rank k, A_k, as the overlap, or the size of the intersection between the two lists at rank k, divided by the rank. See equation 2.9

$$A_k = \frac{|A \cap B|}{k} \tag{2.9}$$

The average agreement up to k, AA_k, is then:

$$AA_k = \frac{1}{K} \sum_{k=1}^{K} A_k \tag{2.10}$$

To get the RBO at a rank k, the agreement between the two lists are averaged for each rank down to k, and each agreement is weighted by a weight, w_k, that help put emphasis on the top documents. The weighting scheme used by RBO is a geometrically decaying weight.

$$RBO = (1 - p) \sum_{k=1}^{\infty} p^{k-1} \cdot \frac{|A \cap B|}{k} \tag{2.11}$$

However, the *average agreement* (AA) from equation 2.10 will only be used in this thesis, as even without weights it puts emphasis on the top documents. RBO was also specifically developed to function as a metric on possibly infinite lists, and as we do not operate on such lists in this thesis, we do not need geometrically decaying weights to keep the tail of the lists from outweighing the top documents.

2.3 Lovdata's document collection

We now have the necessary tools to evaluate the performance of document retrieval systems, both on a ranked and boolean document collection. In this section I will introduce and discuss the document collection we will be working with in this thesis; The *Lovdata Court Corpus* (LCC). But first I will give a quick overview of the different Norwegian courts to give the document collection some context.

The Norwegian justice system is divided into three instances (Boe, 2010). The first judicial instance is the District Courts, known as Tingretten in Norwegia. They divide the country into 63 judicial districts. For civil cases, there is a lower instance, the Conciliation Boards, which handles most cases. But the cases that are not resolved there are brought forward to the District Courts. All criminal cases start in the District Courts. Any party that disagrees with the results of a case can bring it forward to the second instance, the Courts of Appeal, known as Lagmannsretten in Norwegian. The Courts of Appeal divide the country into only 6 judicial circuits; Borgating, Eidsivating, Agder, Gulating, Frostating and Hålogaland Court of Appeal. After this, cases can be appealed to the final instance, The Supreme Court, known as Høyesteretten in Norwegian. This is a single court, and in total 20 judges are appointed to it. The verdicts are final and cannot be appealed to any higher Norwegian court. In addition to these three instances there are several special courts in Norway, which are briefly mentioned in the next paragraph.

The larger Lovdata document collection contains, among others, documents and rulings from the three judicial instances and eight special courts. The documents in this subset are sorted into 25 origin groups, based on where they originate from. The origin groups are given as such: The District Courts and The Supreme Court have the prefixes TR and HR, respectively. The six circuits in the Courts of Appeal; Agder, Borgarting, Eidsivating, Frostating, Gulating and Halogaland have the prefixes LA, LB, LE, LF, LG and LH, respectively. Each instance, and the courts therein, are further split into two: criminal cases and civil cases, and gain the suffix STR and SIV, respectively. These origin groups will mostly be combined into three super-groups for the three court instances, identified by the prefixes HR, L and TR. In addition, Lovdata has two groups for older Eidsivating-cases; LXSTR and LXSIV, and an extra group for The Supreme Court where appeals and other documents that are not rulings are collected, called HRU. This brings us up to 17 origin groups. In addition, there are some special courts in Norway: Arbeidsretten, Jordskifteoverretten, Jordskifteretten, Trygderetten, Høyfjellskommisjonen, Utmarksdomstolen for Finnmark, Kommisariske høyesterett 1941-1944 and Utmarkskommisjonen for Nordland og Troms. The origin groups for them are: ARD, JSO, JSR, TRR, HFK, UTMA, HKOM and UNT.

In the rest of this section I will compile the Lovdata Court Corpus. This corpus is a subset of the larger Lovdata document collection. The corpus was defined in November of 2017, and so no new documents were added to the corpus during the work on this thesis, even though new documents are added to the Lovdata document collection daily.

In the 25 origin groups mentioned above, there are 212,070 documents. The 8 special courts make up 18.2% of the documents, and are so domain specific that they will be excluded from the corpus. We are therefore left with 17 origin groups; the three legal instances. After disregarding the 8 special courts, the HRU base contains 21% of the documents in the corpus, but since this group mostly contains documents that are not court decisions, they are also omitted from the corpus. Thus, the final corpus contains 136,872 documents across 16 origin groups. This subset of the larger Lovdata document collection will from here on be known as the Lovdata Court Corpus (LCC).

Below, I will take a look at the three court instances in the corpus and present the total number of documents in each super-group, total number of words in each super-group and total number of references to other documents in each group.

Instance	Total documents	Total words	Total references
HR	40,246	68,787,157	589,656
L	77,676	213,756,900	1,710,390
TR	18,950	72,586,448	490,970

Table 2.1: Statistics for the three judicial instances.

From table 2.1 on the preceding page we can observe that 56% of the documents are from the Courts of Appeal, and 29% from The Supreme Court. This makes sense, as The Supreme Court only rules on cases that the Courts of Appeal have previously ruled on, which have then been appealed to The Supreme Court and the appeal accepted. Following this logic one would think that the District Courts subset would contain even more documents than the Courts of Appeal, yet it only contains a meager 15% of the documents. This is simply because Lovdata only possess a tiny fraction of all the documents from the District Courts.

To illustrate more easily the interplay between the words, references and documents, table 2.2 presents the average number of words per document, the average number of references per document and words-to-references-ratio, which will be called reference density, for each instance.

Instance	avg words per doc	avg refs per doc	avg words per ref
HR	1,709	14.6	116.6
L	2,752	22.0	124.9
TR	3,830	25.9	147.8

Table 2.2: Statistics for the three judicial instances.

We can observe that as we move up through the judicial instances, the decisions become shorter, but have a higher reference density, meaning there are fewer words between each reference. When appealing a decision, most often only a few parts of a decision is appealed, meaning that each higher instance has fewer legal questions to answer for each case (Robberstad, 2009), and thus the court decisions are shorter. In addition, the decisions from the District Courts often contain a lot of factual information about the case. These paragraphs do not contain any references as they do not raise any legal questions, they can for example be a simple account of the events when the purported crime took place. If the Court's decision is appealed to the District Courts, they will not retell those events. In the same manner, the Supreme Court will spend little time going over the facts of the case again. Thus, each higher instance will have shorter and more dense documents, with regards to references.

2.4 Defining the gold standard

As mentioned in section 2.2 on page 17, to be able to evaluate the retrieval of a system we need to know the correct answer. This information is called the *gold standard* and represents the 'truth'. For the LCC we do not possess such information, but we will try to come as close to the 'truth' as possible.

The focus of this thesis is using documents as queries for ranked retrieval. Given a query document, a retrieval system will try to find the most similar documents, and I postulate that there is an answer, or in this case, a list of similar documents that are objectively the most 'correct' and similar documents. One way of finding these documents is by manually going through all of the documents in the corpus and assessing and ranking the similarity to the query document by hand. With a corpus as large as ours, this is of course practically impossible. We need an automated process that can use the information we have available to generate this ranked list for us on a case-by-case basis using some easily quantifiable measure of document similarity. Creating an automated process that can assess and rank document similarities is difficult, especially in our situation where similarity is a rather subjective notion. We will work with what we have, and in this section I will try to formalize the process as best I can.

To begin with, let us think about the documents in our dataset, the most likely use of our future ranked retrieval system and most likely needs of our users; the dataset contains documents from the legal domain, specifically court decisions. Potential users of the system will most likely need to find documents that ask and/or answer the same legal questions asked or answered in the query document. Few of the meta information features in the documents in the corpus can be used to represent this. However, like in any scholarly text where arguments are backed by references to other scholarly texts, in legal documents legal arguments are backed by references to other legal documents. It can then be suggested that since users most likely need similar legal arguments, and legal arguments are tied to the references to legal documents, the references in a document can be used to represent the legal arguments contained therein. Thus the similarity of the references between documents can be a surrogate for the similarity between the documents themselves.

With this hypothesis I propose that the references to legal documents within a query document can act as the easily quantifiable measure used to discern document similarity. I have developed an automatic process which utilizes the references in documents to generate a gold standard on a case-by-case basis. This system will be known as the *Reference Vector System* (RVS). The Reference Vector System will represent the documents as simple Bag-of-Words vectors with the references acting as words. See section 2.5 on page 27 for the introduction to representing documents as vectors and Bag-of-Words.

One of the goals of this thesis is to in the end have built another system which can give equally correct answers on documents *with* references as on documents *without* references. This is possible with the assumption that while references can be used to represent the legal arguments, and thus the document to some extent, the actual text can also represent the legal arguments, and thus in the end the document. It is further assumed that the text is not different and fully represent the legal arguments in the same way whether the author have used explicit references or not.

2.4.1 An overview of legal references in the LCC

Producing a ranked list of similar documents by using the references in a query document naturally relies on the documents having enough references in them to be able to generate a nuanced profile of them. In the next sections I will explore whether using legal references to produce the gold standard list is possible for the LCC, and if it actually gives us a close approximation to the correct answers, as defined by manual effort.

References to legal documents within the full Lovdata document collection are marked in the xml of documents with the tag `ref` followed by the internal document ID, and if necessary, a specific part of the document, like a paragraph.[1] The tag can in very few instances contain a web-url, building code or other information that links to documents or information outside of the Lovdata collection. But at the moment it is not important where a reference leads, as it is not important what the content behind the references is, only the fact that it was referenced.

Unique references	140,761
Invalid references	564
Total references	2,791,016
Docs with no references	4,207

Table 2.3: Statistics for references in LCC. 'Unique references' refers to the reference vocabulary for the LCC.

In table 2.3 we can see that in total, in the entire LCC, there are 140,761 unique references being cited 2,791,016 times. 564 references are to documents or information outside of the Lovdata document collection, or not properly formated references to Lovdata documents. Only 4,207 documents in the court subset, or 3%, contain no references. It seems that references are abundant in documents and are not a rare property.

2.4.2 Evaluating the validity of the Reference Vector System

In this section we will take a closer look at the Reference Vector System (RVS) and evaluate how well it functions as a gold standard. I will do this by comparing it with user-made groupings of documents from Lovdata's websites.

On Lovdata's website, users have the ability to add documents to different 'favorites' lists. The use of this feature is widespread, but also widely different. Some users simply has a single list containing every document they find interesting or relevant so that they do not need to search for them again, while others create highly specific lists containing a handful of documents pertaining to for example a legal question, some piece of meta information or a case they are working on. No personal information was collected in the retrieval of this data.

We can use these lists to evaluate the performance of the RVS. The idea being that if a user has put a document in a favorites list, the user has probably thought the document somehow was similar or related to the other documents

[1]Example of tag-structure: <ref id="avgjorelse/hr-2005-845-u">and <ref id="lov/1915-08-13-6/§375">

in the list. To evaluate the RVS, I will define the subset of relevant documents for a query document to be the set of documents that are in the same favorites lists as the query document. See figure 2.1 for an illustration of this. The favorites lists can contain any documents from the larger Lovdata document collection, so documents that were not part of the LCC were removed from the set of relevant documents, as they could never be suggested as a neighbor by the RVS. The RVS will rank and retrieve the top k most similar documents to a query document, and we can measure how well the system performs by comparing the results against the set of relevant documents. Documents which appeared alongside less than 10 other documents were not used as query documents, as they would not give good indications of the actual performance of the system.

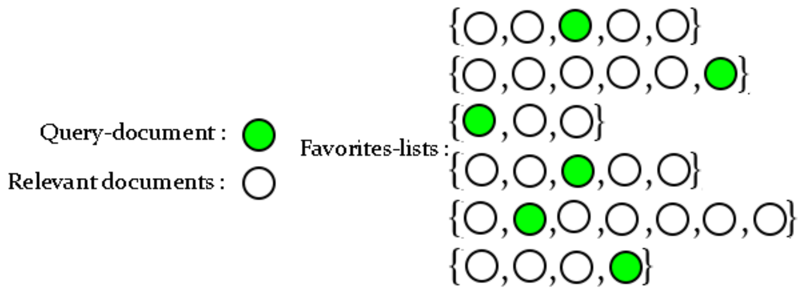

Figure 2.1: Illustration of how I define the set of relevant documents for a query document

An evaluation of the performance of the RVS was carried out using Gensim[2]. The reference-vectors created by the RVS for each document were treated as BoW vectors and a searchable TF-IDF-normalized index was built to be able to rank query results using cosine-distance. Cosine TF-IDF was chosen because it is the most commonly used measure in machine learning. The system was evaluated on 2000 randomly picked documents. See section 2.5 on page 27 for the introduction to representing documents as vectors, as well as the following sections for an introduction to length normalization and cosine distance. See table 2.4 for the results of the experiment.

Precision@100	0.187
MAP@100	0.297
Pearson's r	0.265

Table 2.4: Precision@100, Mean average precision@100 and Pearson's r for 2000 TF-IDF-normalized query documents using cosine-distance. @ referes to the rank at which we cut the lists, in this case I only consider the top 100 documents.

As we can see in table 2.4, the precision@100 for the system is slightly below one fifth, meaning one in five documents retrieved are correct. This is

[2]https://radimrehurek.com/gensim/

statistically very impressive. On average there were 450 correct answers for a query document, and to be able to do some probability calculations, let us use a precision@100 of 0.19. Now we can find the probability of randomly picking documents to retrieve and achieving the same precision. By using a probability density function on a hypergeometric distribution, given by equation 2.12, for an average query document, we can see that the probability of achieving a precision@100 of 0.19, that is correctly picking 19 correct documents from the entire corpus with only 100 tries, by randomly picking documents, is $4.62 \cdot 10^{-26}\%$, or basically 0.

$$P(X) = \frac{\binom{N}{k}\binom{K-N}{n-k}}{\binom{K}{n}} \tag{2.12}$$

$$k = \frac{nN}{K} \tag{2.13}$$

where

$K =$ size of the population, the LCC,
$N =$ number of success states in the population (size of the set of relevant documents),
$n =$ number of draws,
$k =$ number of observed successes

Another way of looking at it is to figure out what is the expected number of correct recommendations out of 100 if we were picking randomly. This is given by equation 2.13, and determines that if we were picking randomly, we would expect to get 0.33 correct recommendations out of 100, which is a precision@100 of 0.0033. We got a precision@100 of 0.187, which is 57 times better. This means that there is a signal which we are picking up, and the results are not only by chance. I do not know if there is an upper bounds on the precision and if human evaluation would give a perfect score, as it is out of scope for this project.

Measures taken at a fixed interval, like MAP@100 and Precision@100, will favor documents with larger sets of relevant documents, as they have a bigger pool of correct answers. To measure the impact of this on the validity of the experiment, the Pearson correlation coefficient for the cosine-TF-IDF results was calculated for the MAP score. The coefficient for the 2000 results was 0.26, suggesting a weak positive linear relationship between the size of the set of relevant documents and the MAP@100-score. (Evans, 1996) This means that the performance of the system in this test was somewhat tied to this aspect of the test, and thus a slightly less precise estimate of the true performance of the system.

2.4.3 Manual inspection

After this, a manual inspection was performed on the top 5 documents retrieved for four query documents.

For the first document, a Supreme Court appeal about insider trading, the top document was the same case from when it was in the Courts of Appeal, and the next two were the other Courts of Appeal cases of two of the other participants in the illegal trading. The final two results also pertained to insider trading, although the details were somewhat different.

The second case, a Courts of Appeal case about improper seizure of evidence, also showed promising results. The top document was a Courts of Appeal case about whether the judges in the query document were unfit to preside over the case. Two of the other retrieved documents also pertained to improper seizure of evidence, and the other two about insider trading. This was most likely because in the query document, the improper seizure of evidence was in an insider trading case. These results show some potential weaknesses of the system. Although all the top five results shared many of the same references, they did so for different reasons. In the query document, the references to insider trading pertained to the original case and to why and how the evidence was seized. This nuance was not understood by the system, and thus two of the results were simply about insider trading. Also in the query document, the improper seizure references were in the context of determining whether the seizures were improper, while in the top retrieved document, the question was whether the judges were fit to rule over such charges.

The third document, a drug case from the District Courts, also had similar documents in the top 5, most notably they all involved the suspect confessing. This is no surprise, as the fact that the suspect confessed, and this led to a milder punishment, meant that the regulations concerning confessions had to be referenced.

The fourth document, a Supreme Court appeal about an ill-tempered professor, highlighted some more of the possible limitations of the reference vector system. The top document was a ruling on whether a TV-broadcaster should be able to broadcast a political commercial. At first glance these two documents might not appear to be similar, but both revolve around the subject of freedom of speech, with references to the same laws regarding that subject. Furthermore, another three of the top 5 documents revolve around whether TV-broadcasters could broadcast certain media, with one of them being the Supreme Court ruling on the top retrieved document.

The results of the manual inspections show a greater promise than the automatic evaluation. The top documents retrieved seemed for the most part to have similar contents, and asked or answered mostly the same legal arguments. However, the cases of the improper seizure of evidence and the ill-tempered professor did show some of the weaknesses of the system. Even though the query documents shared many of the legal arguments with the top retrieved documents, it can be argued that the actual content of the case was not very similar. This problem arises precisely because the laws and court decisions are on purpose vague and can be applied to widely different situations, and are subject to interpretation.

2.4.4 Conclusion on the Reference Vector System evaluation

The ranked list of similar documents used to evaluate the RVS was defined by an automated process measuring the cosine-distance between TF-IDF-normalized reference-vectors. The validity of the RVS was evaluated by comparing the ranked top 100 results for each of 2000 generated ranked lists against user-made favorites lists. I assume the favorites lists can act as a reasonable stand-in for the correct answers for the evaluation, as I also lack a definite gold standard for this. The RVS achieved a MAP@100-score of 0.297 and precision@100 of 0.187 in this test, which would be hard to achieve unless there is a signal that we are picking up. The hypergeometric density showed that the probability of achieving the same precision@100 by randomly picking documents was almost 0, and the achieved precision@100 were 57 times better than the expected precision@100 if the system was picking randomly. At this point it becomes clear that we have no fool proof gold standard, and the RVS is probably the closest approximation I can get while still being able to explain it easily, and stay within the scope of this thesis. The manual inspections serve as a testament to this.

2.5 Representing documents as vectors

Now that we have defined what kind of documents we are working with and defined a system for generating gold standards we can used to evaluate any future document retrieval systems, we can delve into how we represent the documents in the LCC to allow us to calculate the similarity between documents. This section is based on (Manning et al., 2008) and (Singhal, 2001). To avoid too much jumping back and fourth between subjects, the previous sections have already mentioned and discussed some of the concepts introduced in this section. Regardless, the concepts will still be fully introduced even if we have encountered them before.

A document can be represented in a multitude of ways, and one of the most important steps towards measuring similarities between documents is finding a way to represent the documents such that we are able to use them as input to advanced algorithms and to perform different mathematical operations on them. At the same time we must be careful that the representation does not discard or overlook important features of the documents and the contents. Some simplifying assumptions will be made when deciding on a representation, and these assumptions can play a big part in how we compute the similarity and what kind of similarity we are measuring. In this section I will look at building a feature vector for a document and different ways of comparing vectors.

A document is typically represented by a vector of features, with each dimension in the vector corresponding to a feature. A feature is some information *in* or *about* the document that can represent it. A common option is to use the *Bag-of-Words* (BoW) approach, wherein the features are the frequency of use of each word in the document. The unique words in a document make up the documents *vocabulary*, and the BoW vector has a feature for each of the words in this vocabulary. This method can be expanded into using sequential sequences of words, called n-grams, for example two words at a time (bi-grams), as features in the document vector.

In this work I am interested in comparing the vector of one document to one or more other document vectors. We must therefore ensure that each vector has the same features, so as to be easier to compare. To make this process easier we can extend the basic BoW feature vector for a document to cover the entire vocabulary of the corpus, not just the document's own vocabulary. This means that each document vector will have a feature for all of the words in the corpus vocabulary, even the words which never appear in the actual document. This makes every vector directly comparable, as they all include the same features. As the corpus grows larger, so will the vocabulary, which leads to the vectors becoming *sparse*, meaning that only a small part of each vector is actually non-zero. In the next sections we will cover how we use these representations to measure document similarity.

2.5.1 Euclidean distance

One of the benefits of structuring the features as a vector is that there are many ways of comparing vectors, and they range from being rooted in a geometric understanding of the problem to a statistical way of looking at it. One of the most basic and easy to understand methods of doing this is by calculating the *Euclidean distance* between the two vectors, or the straight-line distance.

A vector simply describes a point in a space relative to the origin of that space. In our document vector space, where we have thousands of features, or dimensions, this point will be located in a high dimensional space. Another way of imagining a vector is as a line or arrow pointing from one point to another point. The length of this line, the magnitude of the vector, is given by the euclidean norm in eq. 2.14.

$$||A|| = \sqrt{\sum_{i=1}^{n} A_i^2} = \sqrt{A \cdot A} \tag{2.14}$$

If we have two document vectors, we essentially have two points, and we can imagine a vector stretching from one point to the other. The euclidean norm of this vector is the euclidean distance between the two feature vectors. The shorter the euclidean distance, the more closely related are the documents.

When we count the frequency of words in a document, it does not account for the fact that longer documents will naturally have not only more words, but each words will on average appear more often. This leads to documents appearing unrelated to each other by the algorithm, simply because they are longer. One way of circumventing this is to length-normalize the document vectors by dividing the vectors by their own length, which essentially removes the length of a document from the proverbial equation.

2.5.2 Length normalization and TF-IDF

To account for the length of a document we can simply scale the vector by the document length. This retains all the information about the relative *term frequencies* (TF) for each word and makes our vector of unit length, meaning the magnitude is 1. In this thesis we will use the terms 'term' and 'word' somewhat interchangeably. There is another keen observation about natural language that we can leverage to our benefit: some words appear a lot, no matter the context. These words will dominate our vectors and make our calculations less precise, as they are so common they give us no information to help us distinguish between documents. To combat this we turn to what is known as the *inverse document frequency*, or IDF for short.

IDF scales every word by the frequency of the same word appearing in *other* documents. This gives common, non-informative words such as 'the', less weight in the vector, making the much less frequent, but more informative words stand out more.

By combining, or multiplying, the two ways of scaling, TF and IDF, you get the TF-IDF weighting scheme. TF-IDF gives a high score to words with a high frequency within a document and a low overall document frequency.

$$\text{TF-IDF} = tf \cdot idf \tag{2.15}$$

2.5.3 Cosine similarity

Another measure of similarity is *Cosine Similarity*. This measures similarity between two vectors not by measuring the distance between them, but the angle. The smaller the angle between them, the more similar they are. This removes the magnitude from the problem entirely, meaning length normalization will not affect the similarity when using the Cosine method. Length normalization is in fact an integral part of deriving the cosine similarity, as one can observe in eq. 2.16

$$Similarity = cos(\theta) = \frac{A \cdot B}{||A||||B||} \tag{2.16}$$

2.5.4 Preprocessing

Before we are able to count the words in a document we have to first find all the words and distinguish them from for example punctuations. The first step is usually tokenization, which includes splitting sentences into words, removing punctuation, lower casing characters, and many more operations which will leave us with only the raw words, or tokens, in a document.

As discussed earlier, document vectors are often sparse and dealing with a lot of sparse vectors are cumbersome. We want to reduce their dimensionality further to make them easier to handle. Dimensionality reduction will erase some information from the vectors, but this has the added benefit of making the vectors more abstract, making them represent more of a concept rather than a precise piece of the puzzle.

The most straight forward way of making vectors less sparse is by creating a list of stop-words; words that we know does not help us in computing similarities, such as non-informative words like 'and', or words that we just don't want to consider. These words will then not be used in the vectors. The number of stop-words are usually in the hundreds, while there might be hundred thousands or millions of other words, so more often than not this will not save considerable space or computations. But there are other, better methods that have larger impacts on the system.

A popular method is lemmatization. When performing lemmatization we try to find the root of a word, the *lemma*, for example the root of 'are' is 'be'. This can be achieved using advanced morphological analysis of the term, dictionaries and the context of the term. Lemmatization can lead to saving more space and computations than stop-words, by reducing multiple terms to a single term. This will naturally lead to the representation of the document becoming more vague and general, as we will loose concepts like plurality and tense. But this can work in our favor, because we want our representations to be a more general representation of the document to more easily find similarities between them. A simpler, more brute force method is a stemming algorithm that slices of endings of words, turning 'walking' and 'walked' into 'walk'. Performing lemmatization on 'walking' will yield the same result as stemming, but while stemming might cut off '-ing' from every word, the lemmatization algorithm should be smart enough to know that some words should actually end with '-ing' and leave them intact.

2.6 Embeddings

Choosing a suitable representation for features, documents or words is an important step towards document similarity. Not only can the right representation increase computational performance, but it can improve the predictive performance of the model. The ways of constructing feature vectors for words we discussed earlier relied on us picking and choosing the features, but it is also possible for the computer to learn the representation on its own, without relying on fixed and static features. These learned representations can arise from complex relationships that we humans are unable to understand or find on our own. The downside is that they can be hard to decipher or explain. The past few years have produced significant improvements for learned representations, or embeddings, most notably Word2Vec (Mikolov, Chen, Corrado, & Dean, 2013) and Doc2Vec (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013). In the next sections we will discuss just what embeddings are and take an in-depth look at Word2Vec and how this framework creates word embeddings, as this is the best way to understand the inner workings of Doc2Vec, which is ultimately the most relevant algorithm for the challenges in this project. I will then look at FastText (Bojanowski, Grave, Joulin, & Mikolov, 2017) and GloVe (Pennington, Socher, & Manning, 2014) and see how they build upon and differ from Word2Vec. But first we need a gentle introduction to neural networks. We need to cover neural networks as Word2Vec, Doc2Vec and Fasttext all use neural networks to learn the embeddings. This section is based on (Marsland, 2014). Convolutional Neural Networks (CNN) will also be presented. Neural networks are also relevant for classification tasks further downstream, as neural networks have proven to be useful at generating input to other neural networks, such as in section 6.6 on page 68, were we use Word2Vec to generate word embeddings which are then used as weights in the input-layer of a CNN.

2.6.1 Neural networks

Neural networks try to mimic the way the brain works and how it learns. At the center of the model is the neuron, and the simplest neural network, the perceptron, simulates a single neuron. The perceptron consists of two numerical inputs which are channeled into a single node, or neuron. This neuron contains an activation function where the inputs are added, and if the sum of the inputs are high enough, the neuron fires and sends out another numerical value as output. The different parts of this setup are called layers; the input-layer, the output-layer, and in the middle the neuron forms the hidden layer. Between the layers are weights, which are multiplied with the signal coming through, changing it. This is the crucial part, as it is the weights that learn when we train the model. During training, when we get an output, we calculate an error between the output and the target using a loss function, and this error is then propagated backwards through the layers. Backpropagation can be very tricky to explain, but the essence of it is that the error reaches the weights between two layers, and the weights are adjusted using gradient decent so that if the signal was sent through again, the weights will change the signal to more closely resemble the correct target. Another error is then computed and propagated further back in the model where it changes another set of weights. This trains the model to output better predictions. But the perceptron is severely limited and cannot solve complex tasks, which is why we pair multiple perceptrons together in the hidden layer. This produces a network of neurons, with complex connections that are able to solve more difficult tasks. We can also stack multiple such layers to produce what is known as a deep neural network. There exists many variations on this idea and the steps involved, but for now it is sufficient to only use normal neural networks with a single hidden layer and two sets of weights; input and output.

Neural networks have a tendency to overfit, which is when it learns the patterns from the training data too well and performs worse on the development data we use to get an unbiased measure of the performance. We employ regularization to help prevent a network from overfitting. A simple method is to stop the training once we start seeing the performance on the development data drop, as this often indicates that the network is fitting to closely to the training data. Another regularization method is drop-out (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). In (Srivastava et al., 2014), the authors suggest temporarily removing nodes and incoming and outgoing connections from the network during training. For each pass, different nodes are dropped from the network according to a probability called drop-out rate. With a drop-out rate of 0.5, approximately half the nodes in the network are removed. This will force the network to not rely too much on any single node, as it might not be present in the network.

2.6.2 Normalization functions

Before exploring neural networks further it is useful to understand how and why the output of a neural network is normalized, and how a loss function works. In the next sections a few normalization methods and loss functions will be presented and explained. These sections are based on (Goldberg, 2015), (Goldberg & Hirst, 2017) and (Marsland, 2014).

To be able to learn, a neural network needs to know how the prediction it made compares to the actual true answer, also called the *ground truth*. Imagine we have a network that tries to classify what animals are present in a picture. The output of the network, the prediction, should be the probability given by the network that any given animal in a set of possible animals are present in the photo. This probability for each animal should naturally be in the range [0,1]. To make sure that the output of the network is in this range it can be squashed by a normalization function. The last layer of the network, the output layer, usually contains the normalization function. The activation from the previous layer for each output node, each animal, can be sent through the normalization function and be mapped to the range [0,1]. This means that the normalization function must be able to map any number into this range. The *Sigmoid function*, eq. 2.17, does this.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.17}$$

The Sigmoid function maps any value to the range [0,1], where large numbers are close to 1 and large negative numbers are close to 0. There are some drawbacks to the Sigmoid function, for example will the difference between two large numbers be much smaller than the difference between two smaller, close to 0, numbers. The rate of change at any point for the function, the gradient, will decrease the more positive or more negative the numbers become. The gradient is very import during the learning process of neural networks, as it helps the network decide how to tune the weights. When the gradient is very small, the network will adjust the weights less and learn less. Small gradients will effectively 'kill' the signal being sent through the network during backpropagation, and we want to avoid that. The Sigmoid function is usually used in *multi-label* classification problems; problems where the output prediction can include multiple non-exclusive classes, or labels, such as which animals are in a photo or which keywords belong to a document. This must not be confused with *multi-class* classification, which is any classification task where there are more than two possible classes. When there are only two classes the problem is a binary classification problem, for example predicting whether a specific animal is in a photo. The prediction can still be anywhere in the range [0,1].

For multi-class classification tasks where only a single class is the right answer, for example which animal is the subject of a photograph, we can use the *Softmax* function, eq. 2.18. The Softmax function turns the output of a network into a probability distribution. This means that not only is each element squeezed between [0,1], the elements are squeezed so that the sum of all the elements is equal to 1.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum\limits_{k=1}^{K} e^{z_k}}, j = 1, 2, ..., K \tag{2.18}$$

Let's say the network is trying to predict whether the photograph is of a lion, a giraffe or a whale, and the unnormalized output from the network is [5,2,4]. A simple max function would normalize this to [1,0,0], as it performs a 'hard' max-operation and singles out the 5, then turns this into a probability distribution, which means the probability of a lion being the subject of the photograph is 100%, as it had the highest activation. Softmax performs a 'soft' max-operation where it allows the other classes to retain some probability. Softmax would normalize the output to [0.7,0.04,0.26]. Although the activation for 'whale' is just 20% lower than for 'lion', Softmax gives it a probability that is 63% lower than the probability of 'lion'. The probability of 'giraffe' has almost disappeared, although it still retains a vary small probability, as softmax never discards a class.

It is worth noting that for a simple binary classification task, the Softmax function becomes the Sigmoid function and they will produce the same results.

2.6.3 Loss functions

After the predictions of the network have been normalized we compare them to the right answers. We use a loss function to quantify the discrepancy between the predictions and the ground truth. This discrepancy is called error or loss, and is sent back into the network to inform the weights of how much, and in which direction they need to change. There are many loss functions, and they provide different values for the loss. The most simple loss function works by measuring the absolute difference between the predictions and the ground truth, using eq 2.19.

$$\text{Absolute Error} = \sum |\text{prediction} - \text{truth}| \tag{2.19}$$

This is known as the *Absolute Error* function. By averaging the Absolute Error you get the *Mean Absolute Error*, MAE, which is also known as *L1 loss*. By squaring the difference in the L1 loss instead of using the absolute function, you get the *Mean Square Error*, MSE, which is also known as *L2 loss*.

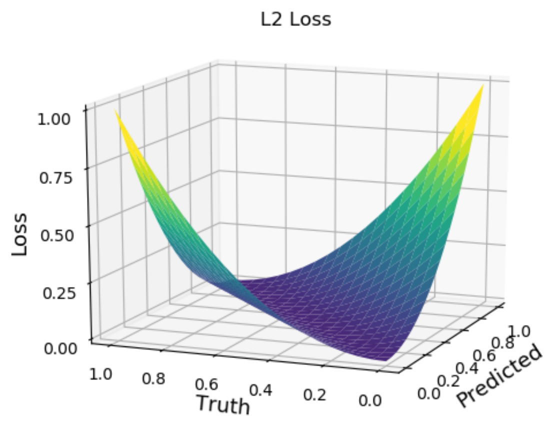

Figure 2.2: Illustration of L2 Loss for a binary classification in the range [0,1]. As the prediction and ground truth disagree more, the loss rises towards 1

The Log Loss for binary classification, or *Cross Entropy Loss*, eq. 2.20, can be difficult to understand by looking at the equation, but works relatively simply.

$$-(y\log(p) + (1 - y)\log(1 - p)) \tag{2.20}$$

Figure 2.3 on the following page illustrates the main concept of Cross Entropy Loss. When the predictions made by the network agree with the ground truth, the loss is low, but the more confident the network is in the prediction and the more wrong it is, the more it is logarithmically penalized.

Cross Entropy Loss is widely used as a loss function, and is often paired with a Softmax or Sigmoid activation function to normalized the input. Sometimes such a pairing is colloquially bunched together and called for example Softmax Cross Entropy Loss, or Sigmoid Loss, even though the activation function and loss function are two separate functions.

Before looking at Word2Vec and the other word embedding frameworks, I will give a quick overview of Convolutional Neural Networks, since they, as the name implies, are closely related to the neural networks explored in the previous sections. I will provide a more detailed and specific analysis of a CNN later in chapter 5 on page 51.

2.6.4 Convolutional neural networks

Convolutional Neural Networks (CNN) are a neural network architecture that is providing great results in the field of image recognition (Krizhevsky, Sutskever, & Hinton, 2012), as they not only draw inspiration from the brain, but more specifically the visual cortex. In the following years after (Krizhevsky

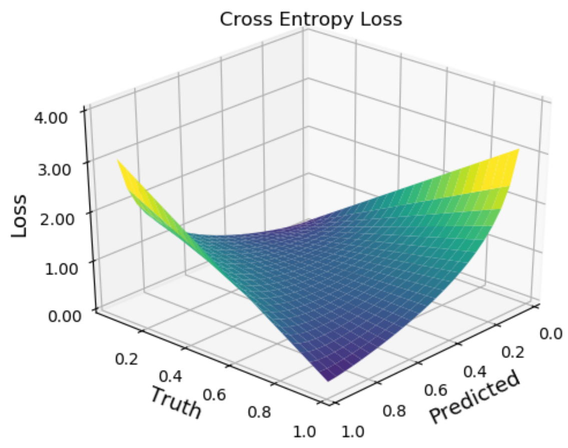

Figure 2.3: Illustration of Cross Entropy Loss for a binary classification in the range [0,1]. As the prediction and ground truth disagree more, the loss rises towards infinity

et al., 2012), CNNs also achieved good results in *natural language processing* (NLP)(Kim, 2014), as the ideas behind the models can lend themselves quite nicely to words and sentences. As the name suggests, CNNs incorporate convolution as a central part of the network. To better understand the concept of convolution, I will go through an example from the visual domain.

A black and white photo is essentially a group of pixels in a grid, or a multitude of values from 0 to 255, arranged in a matrix configuration. Convolution uses a filter, essentially another smaller matrix, to slide over the image matrix and perform some function to generate an output matrix. When the filter slides over a group of pixels, we multiply the filter matrix with the overlapping image matrix and sum the result to get a single final value. We then place this value in the output matrix and shift the filter to overlap a new part of the image matrix and start again. This means that for each output value, or new pixel in the output matrix, it will be a composition of all the pixels in an area of the image. For NLP tasks we substitute the image matrix with a matrix consisting of word vectors as rows, and the filter now slides over entire words; 3,5,7 etc. words at a time. This filter is essentially a matrix of weights, and these weights is one of the sets of weights the neural networks tunes while training. The convolutional layers of a CNN usually contain hundreds of different filters, and the filters often have one of only a handful of window sizes.

CNNs also use what is known as pooling layers. These layers subsample the input, for example with a simple max function. This function splits the input matrix into pieces and picks out the greatest value in each piece. This is one of the core advantages of CNNs, as they can pick up on local features which are good indicators for a class independent of the global position. It is also possible to use an averaging function which will compress the input by averaging the values. This method will also retain the information that the max function discards, but this comes at the cost of possibly retaining unwanted noise. In CNNs used for NLP purposes, each filter has it's own pooling layer, and the pooling layer can either pick out a single value from the entire filter matrix, or it can divide the filter matrix into pieces and pick out multiple values. If a filter matrix is for example divided into three pieces, the pooling layer will essentially pick a max value from the beginning, middle and end of a document for the same filter.

The values picked out by the pooling layers from each filter is at the end of the network assembled into a vector which is sent to the normalization and loss function. This vector can be extracted and used as an embedding for the input document. This embedding can be used to measure similarity between documents, either by measuring distance or angle between embeddings, or putting the embeddings in another network as input.

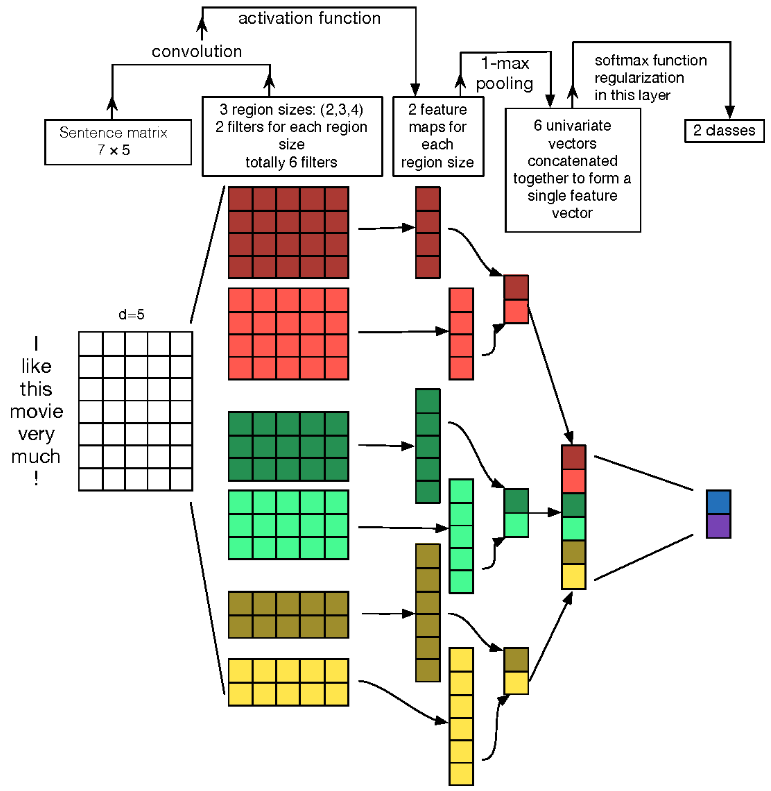
activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

Figure 2.4: Illustration of a CNN from (Zhang & Wallace, 2015). The figure illustrates three filter sets, with two filters each, extracting elements from the input sentence to make a colorful document embedding and making a classification.

A benefit that CNNs provide is speed. Since CNNs operate on images and matrices, something that we have developed GPUs specifically to do, they are able to do huge but quick calculations. The filters and pooling layers also help to pick up the most important features or summarize the content, which decreases the amount of computations needed in the later layers.

As CNNs build upon and expand regular neural networks by using things like convolutions and pooling, they need a prediction task and target to learn from. In this thesis we train the network on some prediction task that we have no direct interest in, as we only want the document representations the network produces as a byproduct of the classification. The prediction task might be to predict the meta data of a document. There are other methods, like Long Short-Term Memory networks (LSTM), which are neural networks where neurons can have loops, connecting back to themselves, essentially giving the network memory. But for now we settle for only using CNNs.

But before we start using CNNs we will first take a look at methods of generating word embeddings. The next sections will focus on some of the most popular word embedding frameworks.

2.6.5 Word2Vec - CBOW

A common theory in natural language processing is the idea that the meaning of a word is given by it's textual context. We used this line of thinking earlier when we discussed feature vectors for documents, where the words that appear in a document would form a representation of the topic or content of that document. We can apply this theory the same way for words: the other words that appear together with a certain word can form a representation of the meaning of that word. We could for example say that if a word appears in a sentence, then all the other words in that sentence is in the contextual window of the word, meaning we use them as features to represent the word. A feature vector for a word will then consist of all the other words it has occurred in a sentence together with. But for the most part we are interested in a window with a fixed size, like the two words on either sides of a center word. Picking the size of this window is an important part of the task. Following this line of reasoning you could say that if you could then predict the context a word would appear in, you would have understood the meaning of the word. This is the backbone of the Word2Vec algorithm.

Word2Vec can be separated into two different network structures: CBOW and Skip-gram. I will first give an overview of CBOW, then present Skip-gram.

Continuous Bag-of-Words (CBOW), aims at predicting the probability of a word given the context. It uses a neural network for this task, and the goal of the network is to build a vector for each word so that it excels at predicting other words appearing in a words context. However, in the end we are not interested in actually using the network or the output. We are interested in the learned weights, which are the word vectors. We feed the context of a center word to the input layer, and the output layer gives us a prediction for the center word. The input layer takes a CxV matrix as input, where C is the number of context words and V is the size of the vocabulary, the number of unique words in the corpus. The input vector of a single context word is a one hot encoding of the word, meaning it has a dimension for every word in the vocabulary, with a 1 in the spot that represents the context word and a 0 in the others. Likewise, the output layer delivers a 1xV one hot encoded vector representing the prediction for the center word.

The weights between the hidden layer and the output layer is what we use as the word vector representation of the word, or what we call the word embedding. One of the things that set embeddings apart from say a normal BoW approach is the fact that the vector representation is not based on some easily observable fact, like discrete word frequency, but a continuous and distributed representation of the context made by the network. Another is that the embedding is considerably denser and has lower dimensionality than the BoW vector representation. The reason for this is the size of the hidden layer. We set the size of the hidden layer, and thus the size of the embedding, quite low, usually in the hundreds. This forces the network to squeeze and compress the information before interpreting it. This has proven to enable the network to capture new and different concepts and relationships between words. As the input is multiplied by the input weights, or since the input is one hot encoded, we simply pick the corresponding rows from the weights, we get an activation with size CxN. We pass this to the hidden layer which is a mean linear activation which performs an element-wise average and end up with a final 1xN hidden activation that we pass to the output weights. The final output is produced and we get the negative log likelihood of a word given a set of context words. Finally we calculate the errors between the target and the output, and propagate this error backwards through the network.

2.6.6 Word2Vec - Skip-gram

Skip-gram works the same way as CBOW, we just flip the entire network around. This time we want to predict the context words given a center word. This means that the input is a single 1xV center word which is passed to the input-weights (VxN) and to the hidden layer (Nx1). The major difference this time is that the Nx1 hidden activation is sent to the output weights with size NxV and we get a CxV matrix as a result. Since we wanted to predict the context words we end up with C number of output vectors, which are all the same, and not just one. We calculate the error between the output and the target, and then we element-wise sum the errors for each vector to get one final error, which we then propagate through the network. For CBOW, the embedding was found in the output weights, and for skip-gram the input weights are the word embeddings.

2.6.7 Fasttext and GloVe

While Word2Vec and Doc2Vec, which we will look at in the next section, made quite the impact when they arrived, other methods have sprung up in the wake that provide better performance in certain areas. Facebook's AI Research team came up with the FastText library (Bojanowski et al., 2017), which can be used to produce word representations. They way it differs from Word2Vec is that while Word2Vec treats a word as the lowest, most basic building block, FastText splits words into n-grams, which together with the word itself, represent the word (Joulin, Grave, Bojanowski, & Mikolov, 2017). This can help with representing uncommon words, as some of the n-grams can be shared with common words, and this also applies to words that are not in the corpus, as they might also have n-grams that are shared with known words. This is something that Word2Vec and GloVe can not do. FastText also makes use of CBOW and Skip-gram, while GloVe does not.

GloVe (Pennington et al., 2014) runs over a corpus and fills out a co-occurrence matrix for each word and some context window. While Word2Vec optimizes the embeddings by performing a prediction task, for GloVe the embeddings are optimized using gradient decent so that the dot product of two embeddings equals the log of the number of times the two words will occur near each other, given by the co-occurence matrix. GloVe offer some performance gains for the computations and calculations, but whether GloVe performs better than Word2Vec or FastText depends on the task and data, especially the preprocessing of the data.

2.7 Document embeddings

We are interested in finding the embeddings for documents, not just words, but finding representations for structures larger than a single word can be difficult. There have been attempts to construct vector representations of n-grams and sentences, for example by averaging the embedding of each word in a sentence, but when the network tries to predict context words or the center word, it does not care about the actual order of the words, only that it is within the context window. There can be valuable information in the order of words. Simply averaging the word vectors will not produce a viable representation of a sentence, as two different sentences with the exact same words will share the same representation

2.7.1 Doc2Vec

Doc2Vec, or paragraph vectors, tries to use the same approach as Word2Vec, but preserving the information that stems from the word ordering. It does this by including a paragraph vector along side the context words in the Skip-gram model. When we train the network, the paragraph vector is trained as well, and in the end it holds an embedding of the paragraph. This model is called Distributed Memory version of Paragraph Vector (PV-DM) as it acts as a memory, remembering what is missing from the context. Another model is called Distributed Bag of Words version of Paragraph Vector (PV-DBOW), which resembles the CBOW method, only this time we use the paragraph vector as input instead.

Chapter 3

Preliminary baseline experiments

In the following chapters we will use different algorithms and approaches for recommending documents. It is therefore beneficial to first establish a benchmark performance measure of a baseline system. By establishing a baseline benchmark, we can easier see the performance gains or losses achieved by the different approaches relative to this references point. As baseline systems often are simple or common ways of solving a task, it can also become easier to intuit what contributed to the changes in performance of the other systems, as the baseline is easier to understand and more widely explored by literature. This will also help us to better understand the complexity and difficulty of our task, as we have more reference points for the baseline's performance. For this reason the baseline should also be robust and expected to perform well, and so we present multiple baselines and preprocessing methods to get a more informed overview of the task.

The first baseline algorithm for our task will be a Gensim TF-IDF normalized BoW model. A BoW model is often the standard method in solving a wide range of natural language processing tasks, and we will use it to create a benchmark which represents one of the most simple and easy to implement methods for solving this task. BoW also generally performs well and can be difficult to improve upon.

The second baseline algorithm for our task will be a Gensim Doc2Vec model. The Doc2Vec model have been used to easily achieve state-of-the-art results in recent years and we will use a Doc2Vec model to establish a benchmark to represent a recent state-of-the-art model.

3.1 preprocessing for the baseline

Before creating the models, the documents in the collection need to be preprocessed. The documents are structured according to the XML format, but we are only interested in the actual content of the document, not the surrounding XML structure, and so each document was stripped of the XML structure.

The documents, now free from XML structure, were preprocessed by UDPipe[1]. A pre-trained UDPipe model for Norwegian-Bokmål (Øvrelid & Hohle, 2016; Velldal, Øvrelid, & Hohle, 2017) was downloaded and applied, instead of training the UDPipe model ourselves, as this will give more consistent and replicable results. The documents were preprocessed in two ways for the baseline; for the first version of the baseline the documents were simply be tokenized, while for the second version they were lemmatized and lowercased. Using UDPipe, the sentences in the document were segmented and tokenized, with the resulting tokens being separated by whitespaces and with one sentence on each line in the final document. The same process was repeated for the second version of the document collection, and this time the tokens were also lemmatized and lowercased.

Many of the documents are written in Bokmål, the most popular of the two official written standards for the Norwegian language. The other documents are written in Nynorsk. Nynorsk and Bokmål share many similarities, but differ widely on others. Since only the UDPipe Bokmål model was used when preprocessing documents it is highly likely that documents written in Nynorsk were not ideally preprocessed. This also applies to any other language found in the documents, such as English, Swedish or German. It is out of the scope for this thesis to distinguish the two standards and different languages during preprocessing, as this would not only involve automatically detecting the language for each document, but also for each sentence or section in a document, as court decisions can include lengthy quotes or excerpts in different languages. It is not possible to rule out that this might affect the performances of any of the models tested in this thesis.

3.2 Splitting the LCC

The Lovdata Court Corpus (LCC) was then split into three groups; training, development and evaluation. 80% of the document mass was placed in the training set, and both the development and evaluation sets received 10% each. Since the documents in the LCC originate from several different courts and court instances, the split-ratio was preserved across the different origins. The documents from each origin were shuffled before dividing them, to prevent any bias arising from their ordering, as they are ordered alphabetically by their filename, which includes a publishing date. Naturally, as law is a field that is constantly evolving and adapting, there will be a discrepancy between old and new documents, which we wish not to be biased towards. This segmentation was separately performed on both the full form and lemmatized set of documents, with the same documents being placed in the same groups between the two.

[1]https://ufal.mff.cuni.cz/udpipe

Document Set	Total documents
Training	109,516
Development	13,677
Evaluation	13,679

Table 3.1: Total documents for the three sets used in this thesis

3.3 Setting up the baseline systems

The first baseline is a TF-IDF normalized BoW model created by Gensim using the training set. A searchable index was created by Gensim which allows us to send documents as queries and get a ranked list of all the training documents ranked according to their cosine-similarity to the query document.

The second baseline, a Doc2Vec model created by Gensim, was trained on the training set for 100 epochs, using the default hyperparameters of Gensim.

Both of the baselines were trained and evaluated on both the full form and lemmatized document collections. In the next sections we will first discuss the evaluation of the BoW model on the full form collection and the lemmatized collection, then the Doc2Vec models on the two collections, before we finally discuss the result of the evaluations of the two corpuses and models relative to each other.

3.4 Evaluating the baselines

To establish the benchmarks set by the baselines, the baselines were evaluated on a document-similarity task. The models were given the documents from the evaluation set as queries and asked to retrieve the top 100 most similar documents from the training set. The same documents were given to the Reference Vector System (RVS), which produced it's own top 100 list from the training set. This list was used as the gold standard for the two baseline models.

The evaluation set was intentionally not included in the training of either of the baselines, but was kept separate. For this reason, both the baselines and the RVS are given a document from the evaluation set as input and retrieve only documents from the training set. I chose this approach as it is more strict since documents are being held out from the creation or training of the systems. This is also because it more closely mimics the most likely use case for the final system, where new, unseen documents are given as input and the retrieved results come from a known, existing set of documents. However, another very close use case is a 'more like this'-feature for documents in a complete set. This use case would be closer to including the evaluation set in the creation or training of the system.

For each document, the two ranked lists produced by the baselines were compared against the gold standard and the performance of each baseline was measured using four measures: precision, MAP, NDCG and average agreement. From these scores we calculated the average score for each court instance, as well as for the entire evaluation set. See table 3.2, where the four measures are placed on the horizontal axis of the table, while the four baseline models are placed on the vertical axis.

	Model	Precision@100	MAP@100	NDCG@100	AA@100
HR	BoW full	**0.133**	**0.273**	**0.092**	**0.115**
	BoW lemma	0.131	0.266	0.091	0.112
	Doc2Vec full	0.026	0.077	0.020	0.020
	Doc2Vec lemma	0.020	0.063	0.015	0.017
L	BoW full	**0.140**	**0.251**	**0.103**	**0.107**
	BoW lemma	0.137	**0.251**	0.101	0.105
	Doc2Vec full	0.031	0.079	0.025	0.021
	Doc2Vec lemma	0.039	0.096	0.030	0.027
TR	BoW full	**0.137**	**0.281**	**0.093**	**0.120**
	BoW lemma	**0.137**	0.279	**0.093**	0.119
	Doc2Vec full	0.046	0.118	0.033	0.037
	Doc2Vec lemma	0.053	0.142	0.038	0.043
Overall	BoW full	**0.138**	**0.256**	**0.100**	**0.109**
	BoW lemma	0.136	0.255	0.099	0.107
	Doc2Vec full	0.032	0.083	0.025	0.023
	Doc2Vec lemma	0.038	0.097	0.029	0.027

Table 3.2: Precision, MAP, NDCG and AA for the four models and three court instances with a cutoff at rank 100.

3.5 Evaluation of the two baselines models

The overall group in table 3.2 show that there is a negligible difference between the BoW model on the full form and lemmatized documents. Both models retrieved on average 14% relevant documents, approximately one in seven. The District Courts (TR), which have longer documents according to table 2.2 on page 21, generally perform slightly better than the other two instances for the BoW models. This is most likely because longer documents have more words, and thus a higher probability of containing words which are good discriminators.

The overall group in table 3.2 show that there is a slightly larger performance-gap between the full form and lemmatized documents for the Doc2Vec model than for the BoW model. The Doc2Vec model trained on the lemmatized corpus perform about 20% better than the Doc2Vec model trained on the full form tokens. Showing the same tendency as the BoW model, the Doc2Vec model consistently perform far better on the longer documents from the District Courts.

3.6 Comparing the two baseline models

The overall group in Table 3.2 on the preceding page show that the BoW model performs on average four times better than the Doc2vec model on the corpus of full forms, and three times better on the lemmatized documents, even though Doc2Vec is considered more advanced. However, as stated in (Mikolov, Sutskever, et al., 2013), a BoW model tend to perform well on long documents, and is difficult to improve upon. In the paper this was a reference to the IMDB-dataset, which consists of movie reviews containing on average 267.9 tokens (Hong & Fang, 2015). Considering that we are using full documents that can span multiple pages and thousands of tokens, and that the Doc2Vec model uses the default hyperparameters suggested by Gensim, which are not tuned for this specific task, it is less surprising that the Doc2Vec model does not perform as well.

3.7 Comparing the metrics

Going forward, for simplicities sake we would like to only use a single metric for evaluating ranked retrieval instead of four. It is therefore important to observe in Table 3.2 on the previous page that for each of the three court instances, the four metrics consistently rank the four models in the same order relative to each other. For example does the BoW model for full form tokens always achieve the highest score for all of the measures, and the BoW model for lammas achieve the second highest. This shows that while the measures measure and express different aspects of the models, the relative relationships between the models stay the same. This means that the measure we use when evaluating and ranking the models is not a factor, and when deciding which measure to keep on using we will rather consider the aspects of the models it compares. Going forward I will use NDCG as the only measure, as it is the only measure that take into consideration the ranking of both the lists it compares, which is a property that that is integral to this project.

3.8 Conclusion on the baselines

In this section we have introduced four different baseline models, based on two different frameworks; Bag-of-Words and Doc2Vec. Two of the four models, one from each framework, were trained on the full form training subset of the LCC, while the two others were trained on the lemmatized training subset. I showed that the BoW models clearly outperformed the Doc2Vec models, and that the difference between using full forms or lemmas varies between the frameworks. The BoW models were not particularly impacted by the preprocessing, while the Doc2Vec models performed better when lemmatizing the text.

Chapter 4

Creating Norwegian Word2Vec models

Before commencing the experiments with convolutional neural networks (CNN), we will first have a look at estimating Norwegian word embeddings using the Word2Vec framework. The word embeddings from this section will later be used to initialize weights in a CNN as part of a set of experiments. Although not directly related to the ranked document retrieval studied in this thesis, word embeddings and their use is very fascinating, and so I will also devote a portion of this section to carry out experiments on the different Word2Vec models. While this is interesting in and of itself, it will help shed some contextual light on different parts of this thesis and provide supplementary information which will make some results more nuanced and clearer. After the experiments with the different CNN models we will revisit the results from this section, which will provide some more context to the results in both this section and chapter 5 on page 51.

The primary Word2Vec model will be trained on the Lovdata Court Corpus (LCC) used in this thesis. There does not exist any Norwegian corpus of court decisions as large as this one[1], or with the same structure and metadata. The corpus of court decisions used in this thesis is as such quite unique. There exists relatively few large-scale Norwegian text corpus resources, the most notable being The Norwegian Newspaper Corpus[2] (NNC), which contains over 1.6 billion tokens, in both Norwegian Bokmål and Norwegian Nynorsk, gathered from 24 Norwegian newspapers from 1998 to 2014. Another large Norwegian text corpus, The The Oslo Corpus of Tagged Norwegian Texts[3] from 1999, contains over 22 million tokens, in both Norwegian Bokmål and Norwegian Nynorsk, gathered from three genres: fiction, newspapers/magazines, and factual prose. The factual prose consists mainly of some NOU reports (Norwegian Official Reports) and Norwegian laws and regulations from between 1981 and 1995, but no court decisions. In

[1]Approximately 355 million tokens. See section 2.3 on page 19
[2]https://www.nb.no/sprakbanken/show?serial=sbr-4&lang=en
[3]http://www.tekstlab.uio.no/norsk/bokmaal/english.html

this section several Word2Vec models will be trained on both the LCC and the NNC, and a combination of both. The models will then be evaluated on two intrinsic evaluation tasks for Norwegian word embeddings.

The word embeddings produced by the Word2Vec models were evaluated on The Norwegian Analogy Test Set[4] and the The Norwegian Synonymy Test Set[5], the two only benchmark data sets for evaluating models of semantic word similarity for Norwegian. In (Stadsnes, 2018) and (Stadsnes, Øvrelid, & Velldal, 2018), the authors demonstrate the usefulness of the two resources by evaluating the performance of semantic vectors trained on the NNC by several different word embedding frameworks. I will compare the performance of the word vectors produced by the Word2Vec models trained in this work with the best- and worst-performing models from the paper: a Word2Vec CBOW model and a GloVe model, respectively. The models from (Stadsnes et al., 2018) use mostly the default hyperparameters for the respective frameworks, with word embedding size of 100, a minimum frequency cut-off of five and trained for five and 25 epochs, respectively. The models in this thesis also use the default hyperparameters of the Word2Vec implementation, but in this chapter the word embedding size is 50 and all words are included regardless of frequency. The models in this thesis was also trained for 100 epochs. For these reasons it is not very interesting only comparing our models against the Word2Vec CBOW and GloVe models.

Using the same lemmatized NNC as (Stadsnes et al., 2018), I trained two Word2Vec models with a word embedding size of 50. These models also have no frequency cutoff. In addition to training a model on the NNC, I also trained a model on the combined NNC and the LCC, to study the impact the judicial documents have on the performance of the model. These two models used the same hyperparameters as in table 4.1. When the models and results from (Stadsnes et al., 2018) are being discussed, they will be explicitly references, any other reference to models trained on the NNC will be in reference to the models trained in this thesis.

The first Word2Vec model was trained on the LCC, and word embeddings were estimated using the Gensim implementation of Word2Vec. The model used the CBOW variant of Word2Vec and was trained on the training subset of the lemmatized court corpus. See table 4.1 for the hyperparameters of the model.

Word2Vec variant	CBOW
Window size	five words
Embedding size	50
Epochs	100
Training data	Lemmatized training subset of the LCC

Table 4.1: Hyperparameters for the Word2Vec model trained on the LCC.

CBOW was chosen over Skip-Gram as this was also the case in (Kim, 2014),

[4]https://github.com/ltgoslo/norwegian-analogies
[5]https://github.com/ltgoslo/norwegian-synonyms

where they used vectors trained on the Google News Corpus[6] by a CBOW neural network. The window size for the Word2Vec models in this work was set to five as the vectors trained by Google also use this window size. The size of the word vectors were set to 50 since the standard baseline CNN model in section 6.2 on page 59 use word embeddings with a size of 50, and the vectors produced by the Word2Vec models are intended to be inserted into the word embedding layer of a CNN as part of the experiments in section 6.6 on page 68. The models were trained for 100 epochs. In figure 4.1 we can see that the reported loss during the training of the models stays almost constant during the 100 epoch training period. (Stadsnes et al., 2018) only train the models for five and 25 epochs, depending on the framework. It is uncertain if the training period greatly affects the usefulness of the word embeddings, but this was motivation for not training the models more than 100 epochs. The reason for all the models having a distinctly different loss is most likely because the Gensim implementation of Word2Vec does not normalize the loss, and so there are more words in the larger corpuses, and thus more predictions adding to the loss. The large spikes are also difficult to explain, but could be results of sudden overfitting and corrections.

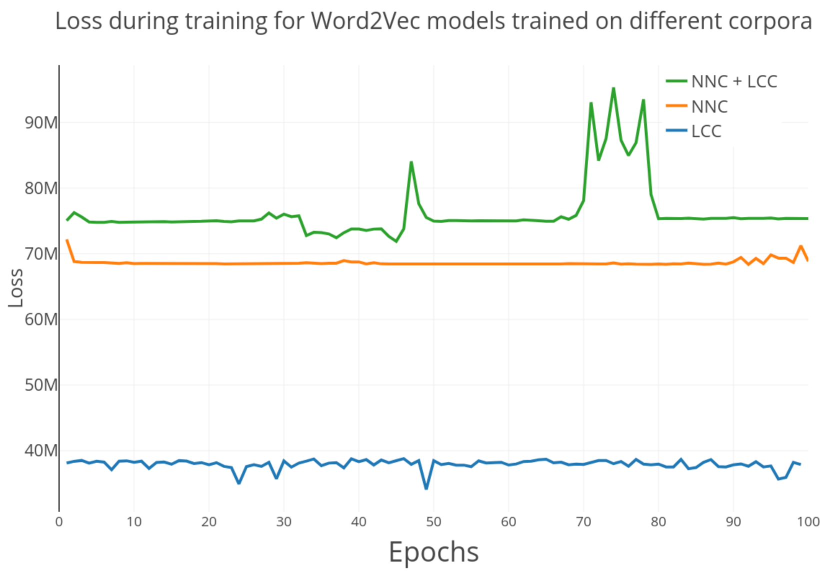

Figure 4.1: Training loss for the different Word2Vec models.

4.1 Evaluating the Word2Vec models

First we evaluate the different models on the synonym detection task. This task evaluates the model's ability to find synonyms for a target word by simply asking it to return the top ten most similar, or closest distance-wise, words, and comparing this with a list of known synonyms for the target word. For

[6]https://code.google.com/archive/p/word2vec/

all models the precision and recall among the 1, 5 and 10 nearest neighbors of the target word are reported. Like in (Stadsnes et al., 2018), only the 30K most frequent words in the vocabulary were considered. See table 4.2 for the results.

	k = 1		k = 5		k = 10	
Model	P	R	P	R	P	R
LCC	5.7	3.8	12.3	8.3	15.9	10.6
NNC	**8.4**	**7.2**	**17.7**	**15.2**	**22.5**	**19.3**
NNC+LCC	8.0	7.0	16.6	14.5	21.3	18.7
GloVe (Stadsnes et al., 2018)	8.4	7.2	18.8	16.1	23.7	20.3
Word2Vec (Stadsnes et al., 2018)	**10.3**	**8.8**	**21.3**	**18.2**	**26.5**	**22.7**

Table 4.2: Results for the synonym detection task.

As we can see in table 4.2, the model trained only on the LCC is outperformed by the two models which included the NNC in the training data, which is to be expected as the LCC is a much smaller corpus and has a more narrow domain. However, it is surprising that the model trained only on the NNC beats the model trained on both the NNC and the LCC. One would expect the larger combined corpus of words would lead to more nuanced and improved word embeddings. Perhaps the two domains are so different that including the LCC only confused the model by placing words in an entirely new context.

All of the models performed worse than the models from (Stadsnes et al., 2018), but this is to be expected as the word embedding size is smaller and thus able to capture less information.

Next we evaluate the word embeddings on the analogical reasoning task. This task is separated into several smaller evaluation tasks which focus on different analogies. These tasks cover either syntactical analogies or semantic analogies. The tasks are structured as questions on the form $\langle Man : King, Woman : ...\rangle$, which translates to 'Man is to king as woman is to...', where the right answer is 'queen'. The semantic questions are mostly about the relationship between countries and capitals, cities and counties and men and women. The syntactical questions are typically about verb tense or forms of adjectives, such as $\langle big : bigger, small : smaller\rangle$. Syntactical information is mostly removed for lemmatized texts, as lemmatization actively tries to conjugate verbs to a single form, such as the present infinitive form, or remove notions such as plurality. Because of this, the syntactical questions will not be used when evaluating models trained on a lemmatized corpus, which is also done in (Stadsnes et al., 2018). Again, only the 30K most frequent words in the vocabulary were considered. See table 4.3 on the next page for the results.

It is no surprise that the Word2Vec model trained on the LCC does not perform as well as the models trained on the entire NNC. The model trained only on the LCC achieves a total accuracy of only 12.6%, about half of the other models. Although the model has trained on documents from a very narrow domain, one would still think that many of the semantic questions are almost as applicable in this context. Norwegian cities, counties and the importance of

Task	Accuracy	Correct/total questions
Common capital city	20.8	15/72
All captial cities	9.5	4/42
Currency	0.0	0/28
City-in-county	7.7	167/2174
Man-Woman	56.7	136/240
Total accuracy	12.6	322/2556
GloVe total (Stadsnes et al., 2018)	**59.7**	
Word2Vec total (Stadsnes et al., 2018)	46.0	

Table 4.3: Results for the analogical reasoning task for the model trained on the LCC.

Task	LCC	NNC	NNC+LCC
Common capital city	20.8	**37.9**	36.2
All captial cities	9.5	38.7	**39.9**
Currency	0.0	35.0	**42.5**
City-in-county	7.7	16.0	**16.2**
Man–Woman	56.7	65.7	**66.3**
Total semantic accuracy	12.6	28.3	**28.8**
GloVe total (Stadsnes et al., 2018)		**59.7**	
Word2Vec total (Stadsnes et al., 2018)		46.0	

Table 4.4: Accuracy for the analogical reasoning task for different Word2Vec models.

distinguishing places should be familiar concepts in court decisions. Often when cities and places are mentioned, the county or jurisdiction is also mentioned, as court decisions should be precise and not ambiguous. However, as one can see in table 4.3, the model only achieves an accuracy of 7.7% on the City-in-county task. It is also curious that the model scored no points on the currency task. As the court decisions are strictly Norwegian, when a foreign currency is mentioned it is usually given the proper context, like 'American dollar', 'Japanese yen' and 'Swiss francs'. One would think that Word2Vec would pick this up.

Since the corpus contains a lot of domain specific terms and phrases, the lemmatization of the documents by the UDPipe model is far from perfect. While this might be problematic for the document similarity task or the CNN classification task, it should not greatly effect the outcome of the evaluation of the Word2Vec model on the analogies task, as the questions asked are about common words which will most likely have been lemmatized properly.

It is more understandable that the model does not perform well on the capital cities tasks, as the court decisions are Norwegian, and when other countries are mentioned it is most often in the context of someone being from that country, and the relationship between a capital city and the country is largely uninteresting. This is probably why the model achieves an accuracy of 25%

on the syntactic nationality adjective task, which is not included in the table. It performs even better on the man–woman semantic task. Male–female relationships are abundant in court decisions as many crimes establish clear and contextual differences between the genders, and things like pronouns make it especially easy to differentiate 'policeman' and 'policewoman' by the context alone. The same goes for parent–child relationships, which were a subset of questions in the man–woman semantic task, as there are many cases involving family members.

Focusing on the models trained on the NNC and the combined NNC and LCC, table 4.4 on the previous page show that unlike for the synonym task, the model which also trained on the LCC performs better. The improvement is slight, but still an improvement. Interestingly, the performance on the 'Common Capital City' task is slightly lower for the model also trained on the LCC. Another interesting behavior is the increase in performance on the 'Currency' task. The model trained only on the LCC failed every single currency-question it was asked. However, when a model was trained on the NNC *and* the LCC, the LCC somehow helped it perform better than when trained only on the NNC, even though the model trained on the LCC alone had not learned enough to answer correctly.

The model trained on the NNC and the LCC still did not perform as well as the models from (Stadsnes et al., 2018), but this is most likely due to differences such as the word embedding size. The results from (Stadsnes et al., 2018) are still included as reference points.

The results from this section will be revisited and discussed further in section 6.6 on page 68 and in section 7.1 on page 88.

Chapter 5

Convolutional neural networks

In this section I will take an in-depth look at the structure and set-up of a Convolutional Neural Network (CNN), and focus on the CNN configuration used in this thesis. This model is based on the model outlined in (Kim, 2014), and implemented in Python using TensorFlow[1]. Most of the code for the implementation in TensorFlow is based on a slightly simplified version of (Kim, 2014) made by Denny Britz[2]. The simplifications made are as follows (Britz, 2015):

1. The model does not enforce L2 norm constraints, a regularization method, on the weight vectors. (Zhang & Wallace, 2015), which conduct a sensitivity analysis of one-layer CNNs to explore the effect of architecture components on model performance, found that the constraints had little effect on the end result.
2. The original paper experiments with using both static and non-static word vectors at the same time. The model made by Denny Britz use only one channel.

CNNs have strong ties with computer vision and image recognition, and some of the terminology from these fields have been used in (Kim, 2014), and thus I will also employ some more image-related words, such as 'channel'. In (Kim, 2014) they experiment with having more than one input 'channel'. This is standard when dealing with image recognition, where images can be split into three dimensions, or 'channels'; red, green and blue. In (Kim, 2014) they experiment with using two channels; one channel with static word embeddings which are not trained any further, and another channel with word embeddings which are trained further. The model made by Denny Britz does not do this, it uses only a single channel which is randomly initialized and trained via backpropagation. For this thesis the code was modified so that the single input channel is also able to use pre-trained word embeddings,

[1]https://www.tensorflow.org/

[2]https://github.com/dennybritz/cnn-text-classification-tf

as explored further in section 6.6 on page 68. When using pre-trained word embeddings, the embeddings are also not kept static, but allowed to be further trained, or 'tuned'.

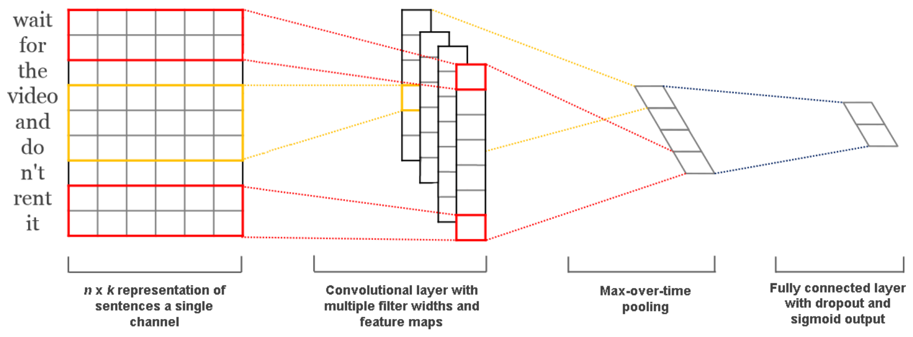
wait
for
the
video
and
do
n't
rent
it
n x k representation of sentences a single channel
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and sigmoid output

Figure 5.1: Simplified model architecture with a single channel for an example sentence. Illustration is based on (Kim, 2014), but altered according to our modifications.

5.1 Network structure

The model is comprised of three main parts; the word embeddings, the convolutional layer and the output layer. See section 2.6.4 on page 34 for a more general explanation of a typical CNN architecture. All documents will be made to fit the set document length, either by padding them or by cutting them short during preprocessing. The word embedding layer contains the weights that make up the word embeddings, acting as a look-up table for the word embeddings. These weights are either randomly initialized and then adjusted and learned during training, or initialized to some set of pre-trained weights. In this thesis, when using pre-trained word embedding weights they will be the word embeddings from the Word2Vec models from chapter 4 on page 45.

Next is another central part of the network, the convolutional layer. A convolutional operation involves a filter being applied to a window of W words. The filter produces a new feature from this window. The filter then slides over all possible windows in the documents. The combined features from each of these windows are put into a feature map. A max-pooling operation is then applied to the feature map, reducing the map to a single feature; the largest, and hopefully the most important, feature in the map. This is just a *single* filter, and the network has several in parallel. There are multiple window sizes, and each of these window sizes have multiple filters. Even though the filters for a specific window slide over the same groups of words, the filters themselves are different, and are tuned differently during training, so each filter hopefully picks up on something new. The filters perform narrow convolutions, meaning that they do not pad the edges of the documents. In the end, every feature produced by all the filters are concatenated into a single vector. This is the vector we extract and use as the embedding for a document. When talking about the parameters of a CNN I will mostly mention and discuss the size of the document embedding, not the number of filters used, as the size of the document embeddings are more near to the discussion in this thesis. The number of filters can also easily be inferred from the size of the document embedding and the number of different window sizes, as the number of filters for each window size is given by the simple relationship $\frac{\text{document embedding size}}{\text{number of window sizes}}$.

The final part of the network starts with a drop-out layer. Drop-out, as explained in section 2.6.4 on page 34, drops out a handful of nodes from a network to prevent the network from relying too much on any single node. Usually drop-out is applied to the hidden layers of a network, but it can also be used as a stand-alone layer. In this CNN the drop-out layer is a separate layer that the document embedding vector pass through, where a certain percentage of the nodes are dropped each pass, effectively denying parts of the embedding to be passed to the next layers. After the drop-out has been applied to the embedding we calculate the predictions the network has made, the error and the loss. The model made by Denny Britz uses a softmax cross-entropy loss function. With softmax, the sum of the probabilities all add up to 1, meaning if you increase the probability of the document belonging to one class, the probabilities for the other classes diminishes. Since the documents we are classifying can belong to multiple, independent classes we instead use a sigmoid cross-entropy loss function, which does not adhere to this constraint.

Unless stated otherwise, the 'performance' of a CNN model will refer to the performance of the document embeddings extracted from a network and used on the document similarity task, not the actual performance of the network on the classification task.

5.2 Data and preprocessing

As CNNs are based on the architecture of neural networks, the model is trained by updating the weights in the network using backpropagation. To be able to do this it needs to compute an error between the output of the network and some desired, target value. I intend to extract the document embeddings produced by the network after training and use them to compute document similarities. The document embeddings, or rather, the weights that produce the document embeddings, are adjusted during the backpropagation phase to minimize the loss. This means that even though I am not interested in the actual results from the network, it is important that the task it is training on will encourage the network to find document embeddings that are useful for the actual task I am using the embeddings for. In my case I am using the document embeddings as representations for documents to measure the similarity between them, and thus I need to force the network to learn to pick up on distinguishing features and attributes in the documents. I want to find out if a task such as predicting certain meta features of a document will provide the network enough constraints to be able to produce documents embeddings which can be used for similarity comparisons.

However, not all of the meta features are usable for this, as many of them will not require the network to learn useful document embeddings. In addition, the documents in the LCC contain a wide range of meta features, but they are sporadic and it is often either too random which documents have which meta features, or too predictable. The only features that form a sufficiently large and wide subset are for the most part not suitable as a classification task for this situation, such as title, the date it was published or author. These features are abundant, but tell us or the network little to nothing about the actual content of the document. The most fitting meta feature found in the data set, which also cover enough documents, is the legal area meta tag.

5.2.1 Legal area meta tag

The legal area metafield is used to tag documents as belonging to certain areas of law, for example fraud, marriage or renewable energy. Unlike many of the other metafields, this field is not populated by editors, but by an automatic process. Many laws, regulations, articles and other documents found in the larger Lovdata collection have a similar field, which is manually filled out by editors. The court decisions inherit the information found in the metafield of the documents they reference. This means that if a certain paragraph of a law have been tagged as belonging to a certain legal area, any document which reference that paragraph, also gets tagged with that legal area. Although the information is not filled out or checked by any person, I believe the tag can be trusted to act as a target for this classification task.

An interesting property of the legal area tag is the fact that it is structured like a tree. There are 35 tree-root legal areas, and each of those roots can have child nodes, and each of those nodes can have their own children. The trees are maximally three levels deep, but there can be leaf nodes on any level, including the root. The tags are integers or series of integers, corresponding to different areas and sub-areas. A document might be tagged with the root-tag *23*, which is 'criminal law', but also *23.03.04*, which is 'criminal law' $\rightarrow$ 'violence' $\rightarrow$ 'murder'. In the next paragraph, a single, unique 'tag' is referring to the tag given to a document, regardless of depth, for example *23.03*, and a document with only this tag is thought to have only one tag, even though the tag can be pruned to form the two tags *23.03* and the root *23*.

Of the 109,517 documents in the training-subset of the LCC, 71%, or 77,806 documents, have inherited one or more legal area tags. Focusing on only these documents, and disregarding duplicate tags in documents, there are in total 207,067 legal area tags, which mean each document have on average 2.7 tags, with a median of 2. The most tags for a single document is 21. There are 493 unique tags. When the tags were counted, only the complete tag was counted, not any of the sub-tags one could get by moving backwards through the branch.

If one were to count these sub-tags, there would be 526 unique tags, meaning that some 1-level and 2-level deep tags are never used in the actual data. The average depth of a tag was 2.1, with a median of 2. Going forwards I will decompose all tags into all possible pruned branches. If a document has a single tag *23.03.04* I will expand the tag so that the document is also tagged with *23.03* and *23*. Documents are not allowed to have duplicate tags. This will hopefully give the network more information during training, as it can give 'partially' right answers by predicting pruned tags.

Docs with legal area tag	77,806
Total legal area tags	207,067
Average tags per document	2.7
Most tags in single document	21
Possible unique tags	526
Unique tags used	493
Max tag depth	3
Average tag depth	2.1

Table 5.1: Statistics for the legal area tag.

5.2.2 Document set for CNN

For the training of the CNN I will use the same training and development subset of the Lovdata Court Corpus (LCC) as I used in section 3.2 on page 41, and the lemmatized variant. However, as not all the documents in the training and development subset have the required legal area tag I will reduce the training subset for the CNN to 77,806 of the documents in the training set, and the development subset to 9,745 documents.

For evaluating the document embeddings, both the evaluation and training subset is used. These subsets will be the same as in section 3.2 on page 41. I will use this evaluation document set and not remove any documents as the evaluation is not constrained by the metafields. When extracting the document embeddings, the documents do not need to have the meta tag to be able to be used as input to the CNN and get a document embedding out. I will also use the complete training subset of the LCC for the evaluation, as the documents omitted from the training can in a similar fashion also pass through the network without needing the meta tag.

The document vectors used as input to the CNN contain integers instead of tokens. Each integer corresponds to an index in a vocabulary and the corresponding word embedding in the word embedding layer. The integer 0 is reserved for tokens that does not appear in the vocabulary.

5.2.3 Input to CNN

The input documents to the CNN need to be of the same length, as the input architecture has a fixed size. Ideally the size to fit the documents into would be the size of the longest document in the training set, and all other, shorter documents would be zero-padded to fit. However, this leads to significantly more computations, as for essentially every document that needs to be padded the computer is spending computations on processing the extra padding. In the interest of time it is therefore beneficial to set the maximum document length to a size where the computer is not spending large amounts of time on processing padding, while at the same time it is not sacrificing information by cutting off longer documents. For the training subset of the corpus, the average length of a document is 2847.4 tokens, however using this as the maximum length of documents would mean that a large portion of the documents would be cut short. Instead, in order of document length, the top 0.5 percentile is at 17,000 tokens, meaning that for this cut-off point, half a percent of all the documents will be cut short. Using this cut-off point as the maximum document length ensures that for the overwhelming majority of documents there will not be any loss of information, while at the same time I keep computations low. The development and evaluation subset of the corpus will also be used as input to the network, and it is possible that basing the cut-off point only on the statistics from the training subset can have an impact on the performance of the network when using the development and evaluation subsets, if for example the documents in these subsets are on average much longer and would therefore result in more documents not being ideally represented as input. However, as the development and evaluation subsets share the same distribution of documents as the training subset, I think it is safe to base the cut-off point only on the data from the training set.

There are multiple ways to shorten the longer documents, the easiest being to simply cut of the excess tail. However, court decisions usually follow a similar format; an introductory part where the case and the different aspects of it are discussed, a middle part where the different legal questions are raised and discussed, and a conclusion which summarizes the discussions and gives a verdict, conclusion or otherwise concludes the document. It is easy to see that cutting off the end of a document might adversely effect the amount of information extracted from the document, as an essential, and highly condensed part of the document is presented at the very end. However, this also goes for the other parts, as the introduction gives context to the rest of the document, and the middle contains the essence of the document.

For the sake of simplicity, the length of documents will therefore be restricted by ignoring any words past the cut-off point, effectively cutting of the tail of longer documents. This might lead to some information loss for a very small part of the document collection, but this is a sacrifice I will make to keep computations relatively low.

Chapter 6

CNN experiments

A CNN contains a lot of hyperparameters which might be tuned to improve the network's performance, such as the size of the word embeddings, the number of window sizes, the number of filters, the pooling-function and so on. In this chapter, several experiments will be conducted to investigate the effects of changing some of these hyperparameters. Many of the hyperparameters in a CNN are most likely dependent on each other, meaning that two hyperparameters can be changed in separate experiments and give improved results, but the same is not guaranteed if they are both changed in the same experiment. The goal of this chapter is not to exhaustively map the interplay between the different hyperparameters using grid search or other hyperparameter optimization methods, but rather to study the effect of adjusting a few simple, but key, hyperparameters.

The performance of each CNN model will be recorded for two different tasks. The first task is the classification task the network is training to solve. The F1 score and loss during training will be discussed. The second task is a ranked document retrieval task which will be known as the *document similarity task*. This task is the most important of the two tasks and one of the core focuses of this project. It is therefore important to not only analyze a model's performance on the classification task, but also make note of how adjusting the hyperparameters effect the performance on the document similarity task.

6.1 CNN baseline

The document embeddings produced by a CNN model will be evaluated on the document similarity task used to evaluate the performance of the preliminary experiments in section 3.4 on page 42. This task essentially consists of comparing the top 100 most similar documents for a query documents with the topp 100 documents retrieved by the RVS for the same query document. To get a better understanding of the performance of the embeddings extracted from the different CNN models, I will establish a performance benchmark from a mathematical model. If a model does not find a signal in the data we expect the performance to be near to the performance of a model which randomly picks documents without following any pattern. The mathematical baseline will then simply be a model which randomly selects 100 documents to compare against the Reference Vector System (RVS). The mathematical formula to calculate the average NDCG performance of such a model is expressed in eq. 6.1.

$$NDCG(random) = \frac{\frac{a\frac{k^2}{n}}{\frac{1}{k}\sum_{i=1}^{k} log_2(i+1)}}{IDCG} \tag{6.1}$$

where

$$\begin{aligned}
k &= 100 = \text{number of documents to pick/relevant documents,} \\
a &= 50.5 = \text{average gain for all relevant documents} = \frac{1}{k}\sum_{i=1}^{k} i, \\
i &= \text{position in list,} \\
n &= 77,806 = \text{number of documents in pool, the training set,} \\
IDCG &= 1250.307 = \text{the ideal DCG score,}
\end{aligned}$$

In the next paragraphs the different parts of eq. 6.1 will be explained. First, $\frac{k^2}{n}$ is the average number of relevant documents in a subset of k random documents, in this case the number of relevant documents picked by the random model. With k relevant documents out of n total, the relationship between the amount of relevant documents and the total number of documents is $\frac{k}{n}$. Since the documents follow a uniform distribution, a subset of the total documents will have the same relationship between relevant and total number of documents. Thus, a subset of k number of documents will have $\frac{k^2}{n}$ number of relevant documents.

Each relevant document has an average gain of a, in this case 50.5, which means that the average cumulative gain for the documents picked by the model is $a\frac{k^2}{n}$. Next we need to figure in the discount. The documents are discounted logarithmically according to $log_2(i+1)$, which means that each document is on average discounted by $\frac{1}{\frac{1}{k}\sum_{i=1}^{k} log_2(i+1)}$. Applying the average discount to the average cumulative gain gives us the average DCG. The final step is to normalize the DCG by dividing it with the ideal DCG, which is the DCG of the perfect result, which is the list made by the RVS. The DCG is normalized to map it to the range [0,1]. This gives the final score, the average NDCG for a random model.

$$NDCG(random) = 0.000977 \tag{6.2}$$

The best performing Doc2Vec- and BoW model from section 3.4 on page 42 will also serve as a baseline for the evaluation performance against the RVS.

6.2 Experiment with standard parameters

The first experiment with a CNN was a simple out-of-the-box test run. The main goal was to set another baseline for the other CNN experiments, using either the hyperparameters suggested in (Kim, 2014) or (Zhang & Wallace, 2015), used in the code by Denny Britz or parameters that were decided on without any a priori knowledge. It is important to note that the networks evaluated in (Kim, 2014) and (Zhang & Wallace, 2015) are used for sentence classification, and thus the parameters and practices suggested in the papers might not be ideal for documents as large as ours. As stated in the introduction of chapter 6 on page 57, for inputs as large as documents I consider an exhaustive search for ideal hyperparameters out of scope for this thesis, and I will rather explore the effects of making a few adjustments to some key features. The features will mainly be the same features experimented on in (Zhang & Wallace, 2015), namely the window sizes, the number of window sizes, the total number of filters and regularization. Neither (Kim, 2014) nor (Zhang & Wallace, 2015) explicitly experiment with adjusting the size of the word embeddings, but I will explore this aspect in this work.

In the next sections the model using the initial hyperparameter settings and values will simply be known as the *standard model* and the hyperparameters used will be known as the *standard hyperparameters*. An overview of the hyperparameters for this network and the other models can be seen in table 6.1.

Parameters	Standard model	Other values experimented on
Size of training set	77,806	″
Max document length	17,000	″
Batch size	64	″
Word embedding size	50	25, 100, 300
Window sizes	(3, 4, 5)	4, 7, 10, 20, 30, (6,7,8), (7,7,7)
Total filters	200	100, 300
Doc embedding size	600	100, 300, 900
Drop-out rate	0.5	0, 0.25, 0.75
Nr. of training epochs	10	″
Word emb. initialization	Random	Pre-trained
Is corpus lemmatized?	Yes	No

Table 6.1: The parameters for the different experiments. The ditto mark denotes parameters which are always the same as the standard model

The max input document length was set to 17,000 tokens, based on the reasoning in section 5.2.3 on page 55. The batch size was 64, which was the default value in the code from Denny Britz, although (Kim, 2014) use a size of 50. The word embedding size (W) used in (Kim, 2014) and (Zhang & Wallace, 2015) is 300, however this was reduced to 50 to lower the amount of computations and allow for faster results for the exploratory experiments. The weights in the word embedding layer were randomly initialized and later adjusted during the backpropagation. The window sizes used are the same as in (Kim, 2014), but while (Kim, 2014) uses 100 filters for each window size, this experiment used 200. At 600, the document embedding length (D) for this model is then twice the length of the document embeddings produced by the model proposed by (Kim, 2014). The drop-out rate (p) was kept the same as in (Kim, 2014). The network trained for ten epochs, making ten passes over the training data. After every epoch the network was evaluated on the unseen development data.

The different models trained in this chapter all behave roughly the same during training. I will take a detailed look at the model trained using the standard hyperparameters and analyze the training and development log. To avoid repetition I will not analyze each model as thoroughly as the standard one and will rather discuss them in groups and with focus on their relative performance. During training, the performance of the models on each batch is recorded. In total, 12,160 measurements are recorded when training for ten epochs. These measurements are downsampled to 100 data points, ten per epoch, by averaging groups of around 121 measurements (10% of an epoch). These data points are then used to plot the figures in the next sections. This gives the figures a resolution of ten data points for each epoch. The reason for downsampling the measurements is to make it easier to plot and to interpret as there is a large variance in the measurements and this makes many of the graphs hard to read. The individual performance of each batch is also largely uninteresting, and so averaging them provides a more informative overview of the model.

After each epoch of training the model was evaluated on the classification task using the entire development set. The results of these evaluations are plotted in the figures to show the performance on the development set. Please observe that the evaluation is done on the classification task, and must not be confused with the evaluation of the models on the document similarity task. The evaluation will give a better estimate of the true performance of the models during training and help identify if and when overfitting occurs.

6.3 Analysis of the performance of the standard CNN model

All the weights in the network are randomly initialized and as such the network performs poorly in the beginning. After a full epoch, the loss is about 80 times lower than at the start, showing how the network rapidly tunes and improves the weights. The loss continues to decline and eventually nears a plateau. The final loss for the evaluation on the development data is about 157 times lower than at the first measurement .

The network predicts whether certain legal areas, as described in section 5.2.1 on page 54, are 'present' for a document. In the beginning, the network makes almost exactly the same amount of 'present' as 'not present' predictions. This shows that the network is truly randomly initialized. From table 5.1 on page 55 it is clear that on average only about 1% of the legal area tags are 'present' for any given document. The network learns quickly over a few steps

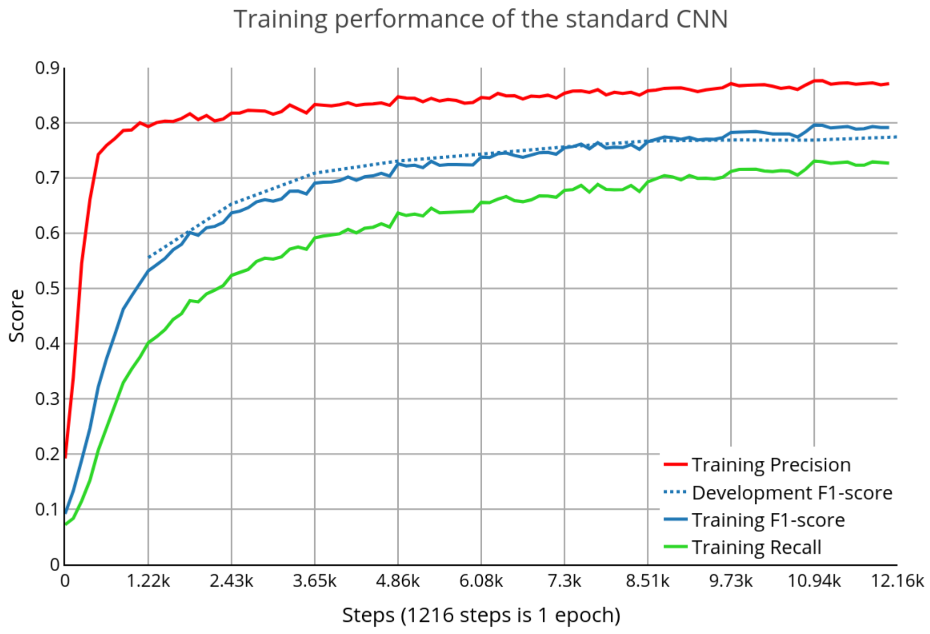

Figure 6.1: Training performance of a CNN model using the standard hyperparameters

that it will perform dramatically better by mostly answering 'not present'. It always makes some 'present' predictions, and after a few steps it has reached a precision over 0.1, meaning over 10% of the 'present' predictions are true. After one epoch the precision is already around 0.8. The precision starts plateauing after the first epoch, but keeps slowly rising for the rest of the training. The final precision measurement is about 96 times larger than the first.

As a result of the random initialization the recall starts at 0.55, which is not visible in figure 6.1 as the model quickly learns to mostly predict 'not present', and the recall drops to around 0.05 after a few steps. The model then slowly learns to make more predictions, and the recall rises, although not as quickly as the precision. After one epoch the recall is at 0.4, meaning 40% of the correct tags are being correctly predicted. Towards the end the recall stops rising and starts evening out.

The F1 score, a mix between the precision and the recall scores, show the trend for both the measures. The final F1 measurement is about 42 times larger than the first. After every epoch the F1 score is calculated for the development set, and in figure 6.1 one can see that the average F1 score for the development set closely follows the F1 score for the training set. In the next sections I will mostly feature two types of figures: a figure for the loss of the model, both recorded during training and when evaluated on the development data, and a figure for the F1 score for the model, both recorded during training and when evaluated on the development data. Although the F1 score and the loss for the most part describe the same behavior, there can appear some interesting discrepancies between the two measures which sometimes can paint two different pictures.

Unless there is some interesting behavior to highlight, precision and recall will not be included in any more figures, only the F1 score.

As the task is heavily skewed towards 'not present' predictions, as soon as the model learns to mostly answer 'not present', the accuracy jumps to 0.99. This reflects the 1% distribution of the present legal area tags. This also underscores why accuracy is a bad measure to use on heavily skewed data, as it does not consider the fact that predicting 'not present' tags is not the important part of the experiment, rather it is the small nuances of the 'present' tags which I am interested in. For this reason I will not feature the accuracy measure any more in this chapter.

Once the network had finished training, both the full training and evaluation sets were sent as input to the network and the resulting document embeddings were extracted. The performance of the model was then evaluated the same way as in section 2.4.2 on page 23; by picking the top 100 most similar document embeddings from the training set for each document in the evaluation set and compare the top list with the top list produced by the RVS. The similarities between the two top lists were measured using the NDCG measure and the results can be seen in table 6.2. The table also includes the NDCG of the random baseline.

Court instance	Model	NDCG@100	F1 score
HR	Standard	0.090	
L	Standard	0.132	
TR	Standard	0.129	
Overall	Standard	0.127	0.774
	BoW full	0.100	
	Doc2Vec lemma	0.029	
	Random	0.000977	

Table 6.2: NDCG for the standard CNN and the three court instances, and F1 score on development data after ten epochs

As we can see from table 6.2, the documents embeddings extracted from a CNN after ten epochs of training outperform all the other baselines. Although both Doc2Vec and a CNN are implementations of neural networks, the CNN produced document embeddings which performed over 330% better on the document similarity task.

6.4 Adjusting the word embeddings size

The goal of the first set of experiments was to see how changing the word embedding size would effect the network. As the word embeddings sit in the weights of the first layer, the complexity of the embeddings could possibly have a cascading effect through the network, as the filters in the convolutional layer will receive more or less information. Reducing the word embedding size is like shrinking an input image along an axis. This could possibly remove noise from the system, but the reduction in dimensionality might leave too much information out. It is possible that the network might still learn to discriminate the important features, regardless of the word embedding size, or within a reasonable window. The opposite might be true for increasing the word embedding size, allowing more granular information to be represented and in the end provide a more nuanced representation for different contexts.

The parameters for this set of experiments were the same as in the previous experiment with the standard parameters, with the only difference being the size of the word embeddings. See table 6.1 on page 59 for the hyperparameters, figure 6.2 and 6.3 on the next page for the performance on the classification task, and table 6.3 on page 65 for the performance on the document similarity task.

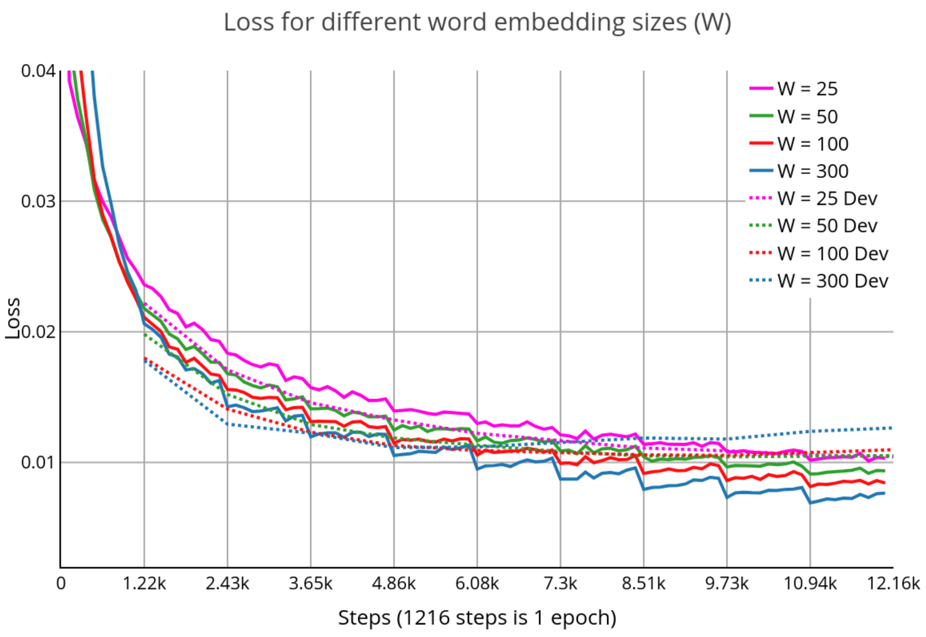

Figure 6.2: Loss for models with different word embedding sizes. The dotted line is the loss on the development data.

In figure 6.2 and 6.3 on the following page, we can see that the models are quite close in performance during training, and for the evaluation on the development set they are even closer. After ten epochs the performance on the development set is almost identical.

In table 6.3 on page 65 we can see that that the model with reduced word embedding size performs worst, with the overall NDCG score dropping by almost 7%. The model clearly still pick up on a signal and has no trouble outperforming the random baseline, even when the size of the word embeddings are one twelfth of the size used by (Kim, 2014).

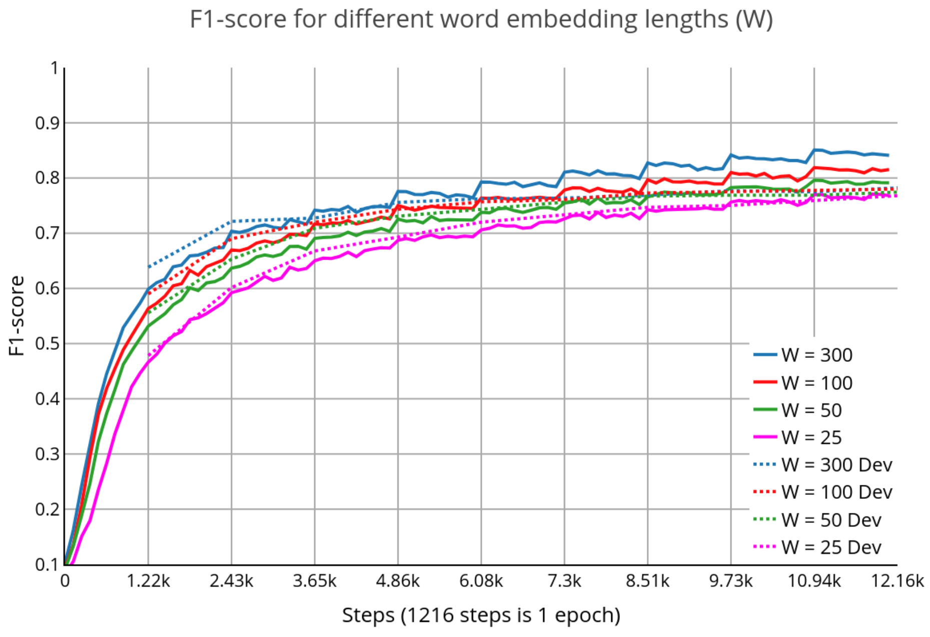

Figure 6.3: F1 score for models with different word embedding sizes. The dotted lines are the F1 score on the development data.

The final word embedding size experiment increased the size of the word embeddings to 300, the same size used in (Kim, 2014). From figure 6.3 one can see that even though 300 is three times larger than 100, the performance gain on the training data is not three times larger, and the performance on the development data is almost exactly the same as the model with size 100 from epoch six and out. In figure 6.2 on the previous page one can also see that the loss for this model starts to rise for the development data during the last two epoch, possibly indicating that the model is overfitting. Table 6.3 on the following page show that the performance of this model on the document similarity task is almost exactly the same as for the model with a word embedding size of 100.

Table 6.3 on the next page show that the models with double and six times the size of the standard word embeddings performed almost exactly the same as the standard model. It is clear that increasing the word embedding size positively impacted the performance, as the performance was higher for all three court instances, most notably for The Supreme Court, where the performance was just over 3% higher for the model with double the size, but the improvement is minimal compared to the increase in training time. There seemed to be an almost linear relationship between the size of the word embeddings and the training time, so the model with six times larger word embeddings took six times longer to train.

There seems to be a direct correlation between the word embedding size and the performance of the system. The largest word embedding size performed the best on both the classification task and the document similarity task.

	Model	NDCG@100	F1 score
HR	W = 300	0.091	
	W = 100	**0.093**	
	W = 50	0.090	
	W = 25	0.085	
L	W = 300	**0.134**	
	W = 100	**0.134**	
	W = 50	0.132	
	W = 25	0.123	
TR	W = 300	**0.129**	
	W = 100	**0.129**	
	W = 50	**0.129**	
	W = 25	0.119	
Overall	W = 300	**0.128**	**0.782**
	W = 100	**0.128**	0.780
	W = 50	0.127	0.774
	W = 25	0.118	0.768
	BoW full	0.1000	
	Doc2Vec lemma	0.029	
	Random	0.000977	

Table 6.3: NDCG for the word embedding size experiments and the three court instances, and F1 score on development data after ten epochs

Normally these promising results would lead to more experiments where the embedding size is increased until there is either a drop in performance or there are too small gains to warrant further experiments. However, due to time constraints no further experiments will be conducted. The relationship between the training time and the size of the word embeddings led to other experiments being prioritized over expanding the word embeddings further.

Each set of experiments will be based around the baseline performance of the standard model. Because of this the results from one set of experiments will not be carried forwards to the next experiments. The best performing word embedding size found in this section will thus not be used in the experiments in the next sections.

6.5 Adjusting document embedding size

The goal for the second set of experiments was to study the effect of adjusting the document embedding size. The motivation for this is the fact that the document embedding is the part of the network I am actually interested in using after the network has finished training. As I am using the cosine distance to calculate the similarity between document embeddings, having larger vectors might allow for more fine details to be conserved. This might not make a huge impact on the classifying task for the CNN, but it might play a greater role when evaluating document similarities later. For the experiments in this thesis the document embedding size is only dictated by the number of filters for each window size, and so when adjusting the document embedding size I am at the same time adjusting the total number of filters. I discuss this change as changing the document embedding size, since when I have extracted the embeddings and are using them for the document similarity task, the actual task I am interested in, the number of filters does not matter as I am not using the CNN anymore.

All the hyperparameters in the networks were kept the same as the standard experiment, with the exception of the document embedding size (D). The document embedding size was increased to 900, three times the size used in (Kim, 2014), and decreased to 300, the same size used in (Kim, 2014). See table 6.1 on page 59 for the hyperparameters, figure 6.4 and 6.5 on the following page for the performance on the classification task, and table 6.4 on the next page for the performance on the document similarity task.

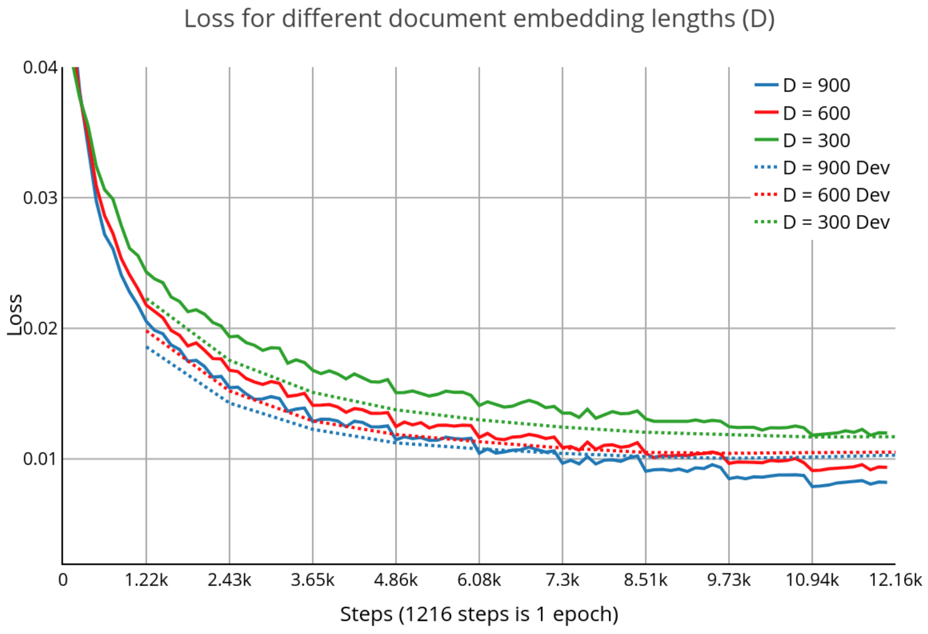

Figure 6.4: Loss for models with different document embedding sizes.

In figure 6.5 on the following page, we can see that increasing the document embedding size leads to a very small performance gain on the classification task, performing slightly better than the standard model. The performance on the document similarity task is also slightly better, performing better than the standard model by about 4%. By reducing the document embedding size to 300, half the size of the standard model, but the same size used in (Kim, 2014), the performance sees a larger drop, both on the classification and similarity task. The overall performance on the similarity task drops over 12%. As the documents embeddings are a central part of both the classification task

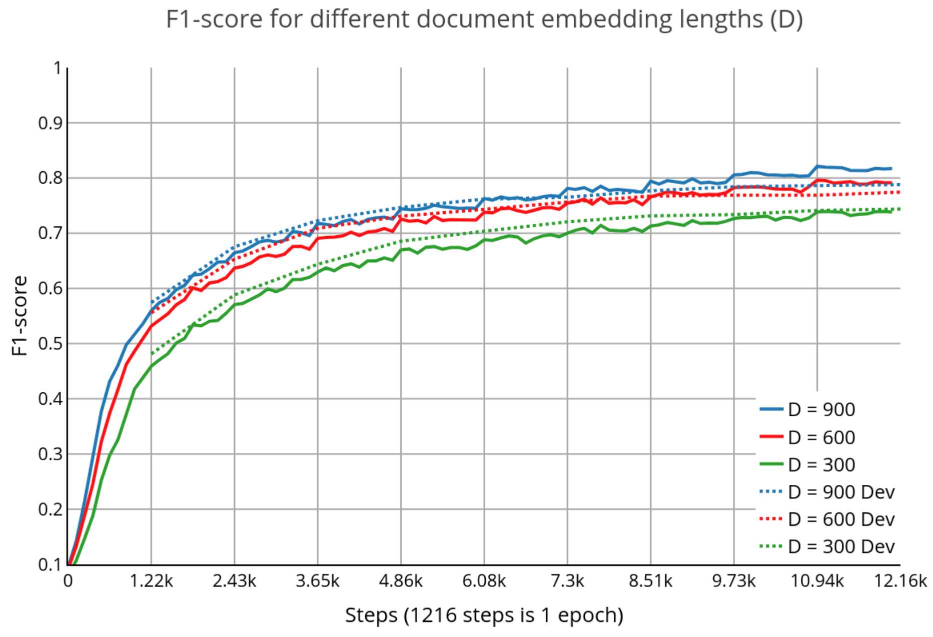

Figure 6.5: F1 score for models with different document embedding sizes.

	Model	NDCG@100	F1 score
HR	D = 900	**0.100**	
	D = 600	0.090	
	D = 300	0.083	
L	D = 900	**0.138**	
	D = 600	0.132	
	D = 300	0.115	
TR	D = 900	**0.130**	
	D = 600	0.129	
	D = 300	0.115	
Overall	D = 900	**0.132**	**0.788**
	D = 600	0.127	0.774
	D = 300	0.111	0.743
	BoW full	0.1000	
	Doc2Vec lemma	0.029	
	Random	0.000977	

Table 6.4: NDCG for the document embedding size experiments and the three court instances, and F1 score on development data after ten epochs

and document similarity task, it is not surprising that increasing their size and allowing them to hold more information leads to better results on the evaluation tasks.

It is worth noting that in figure 6.4 on page 66 and 6.5 on the preceding page one can see that training performance of the standard model and the model with an increased document embedding size starts outperforming the performance on the development set after around eight epochs, possibly indicating the start of overfitting. When the network is actually overfitting we expect the loss for the development set to start rising again. This does not happen after ten epochs, but the loss has started to heavily plateau. The relationship between the three models on the development data is close to the relationship between the models on the training data, and the development performances do not converge like in the previous experiments before training is over.

6.6 Using pre-trained word embeddings

The goal of this set of experiments was to study the impact of using pre-trained word embeddings to initialize the word embedding layer of the network. The motivation for this was the fact that there exists powerful tools and algorithms for making word embeddings, like Word2Vec. By utilizing the information already encapsulated in the embeddings, networks might get a jump start in performance by inserting pre-trained word embeddings into the word embeddings layer instead of learning the embeddings from the ground up while at the same time learning all the other weights of the network. These embeddings can be trained on the same data I am using to train the CNN, but I can also train them on other unlabeled data, such as the Norwegian News Corpus[1]. Word2Vec only need text to train on, so any Norwegian text can be used, regardless of domain. It is of course beneficial to use as much text from the relevant domain as possible to help the model learn useful embeddings. For this experiment the word embeddings in the word embedding layer will come from two different Word2Vec models: one Word2Vec model trained on the Lovdata Court Corpus (LCC) and another trained on the Norwegian News Corpus (NNC) *and* the LCC. See chapter 4 on page 45 for a deeper analysis of these models. In chapter 4 on page 45 it is shown that a Word2Vec model trained on only the NNC and a model trained on the combined NNC and LCC perform almost the same on the evaluation tasks used in that section. It was therefore decided to only use a single mode which had trained on the NNC, and thus I chose the one which had trained on both the NNC and the LCC, as it had trained on more data.

All the hyperparameters in the network stayed the same as in the standard experiment. The only difference was that the word embedding layer was initialized with pre-trained word embeddings. The word embedding layer was not kept static, but was allowed to be further trained. See table 6.1 on page 59 for the hyperparameters, figure 6.6 on the following page and 6.7 on the next page for the performance on the classification task, and table 6.5 on page 70 for the performance on the document similarity task.

As all the Word2Vec models used were trained on the training subset of the

[1]https://www.nb.no/sprakbanken/show?serial=sbr-4&lang=en

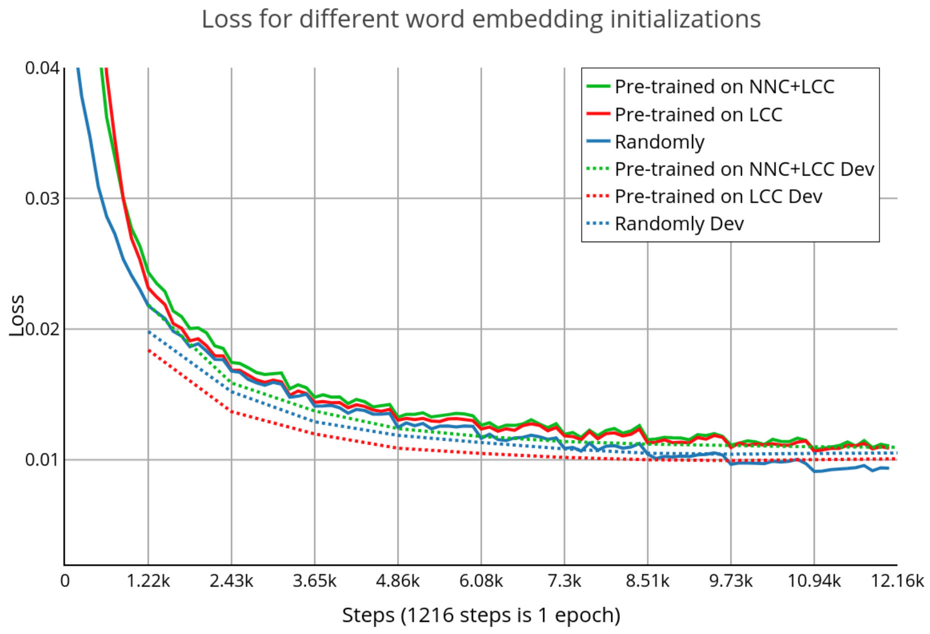

Figure 6.6: Loss for models with different word embedding initializations.

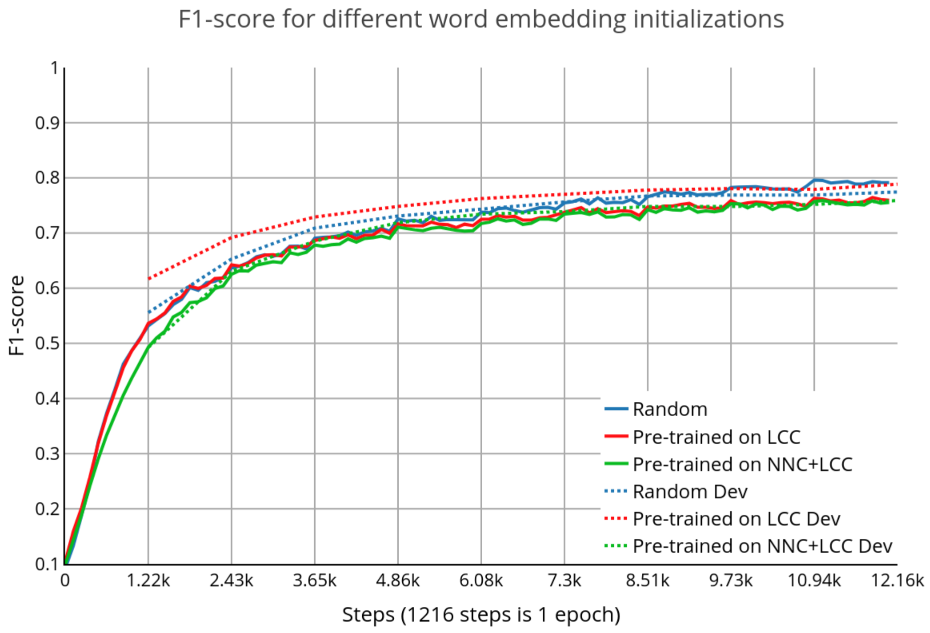

Figure 6.7: F1 score for models with different word embedding initializations.

LCC and thus had a vocabulary which included every term in the training subset, there is no out-of-vocabulary terms encountered during training. However, there are terms in the development and evaluation subsets which

	Model	NDCG@100	F1 score
HR	Pre-trained on LCC	0.085	
	Pre-trained on NNC + LCC	0.081	
	Randomly	**0.090**	
L	Pre-trained on LCC	0.124	
	Pre-trained on NNC + LCC	0.120	
	Randomly	**0.132**	
TR	Pre-trained on LCC	0.124	
	Pre-trained on NNC + LCC	0.123	
	Randomly	**0.129**	
Overall	Pre-trained on LCC	0.125	**0.788**
	Pre-trained on NNC + LCC	0.116	0.759
	Randomly	**0.127**	0.774
	BoW full	0.1000	
	Doc2Vec lemma	0.029	
	Random	0.000977	

Table 6.5: NDCG for the initialization experiments and the three court instances, and F1 score on development data after ten epochs

are not in this vocabulary, and these terms are mapped to the same ⟨UNK⟩-embedding, representing an unknown word.

From table 6.5 we can see that the model with word embeddings from the Word2Vec model which was only trained on the LCC performed almost the same as the standard model with randomly initialized word embeddings. In figure 6.6 on the preceding page and 6.7 on the previous page one can see that these two models performed almost exactly the same during the first part of training, but then the pre-trained model started dropping of and ends up performing almost exactly the same as the other pre-trained model. However, on the development data the pre-trained model performs much better, establishing a clear performance gap over the standard model. The model with pre-trained embeddings trained on both the NNC and the LCC performed worst, both on the classification task and the document similarity task. It is uncertain what caused the large difference in performance on the document similarity task for this model.

These results suggest that using pre-trained word embeddings from primarily the same domain can give a boost to the classification performance, but for the document similarity task it does not provide a positive gain. When using word embeddings trained on primarily out of domain documents, the results are much worse, possibly leading to greater confusion for the model. (Zhang & Wallace, 2015) also conclude that when there is enough training data, learning embeddings from scratch may be the best alternative for achieving a good score, which corroborate the document similarity scores achieved in this section, although the scores are for two completely different tasks.

6.7 Adjusting the drop-out rate

The goal for the next set of experiments was to study the effect of adjusting the drop-out in the network. The motivation for this was the fact that drop-out is used to regularize the network to prevent overfitting to the data, but it seems like the network either has enough diverse training data or does not have enough time to start overfitting heavily, so the drop-out might slow down the learning process and lead to unnecessary over-correcting. On the other hand, drop-out might be a crucial step in the process, as it is applied to the document embeddings during training and this might force the network to build extra strong and robust embedding weights.

All the hyperparameters in the network stayed the same as in the standard experiment. The only difference was that the drop-out rate. See table 6.1 for the hyperparameters, figure 6.8 and 6.9 on the next page for the performance on the classification task, and table 6.6 on the following page for the performance on the document similarity task.

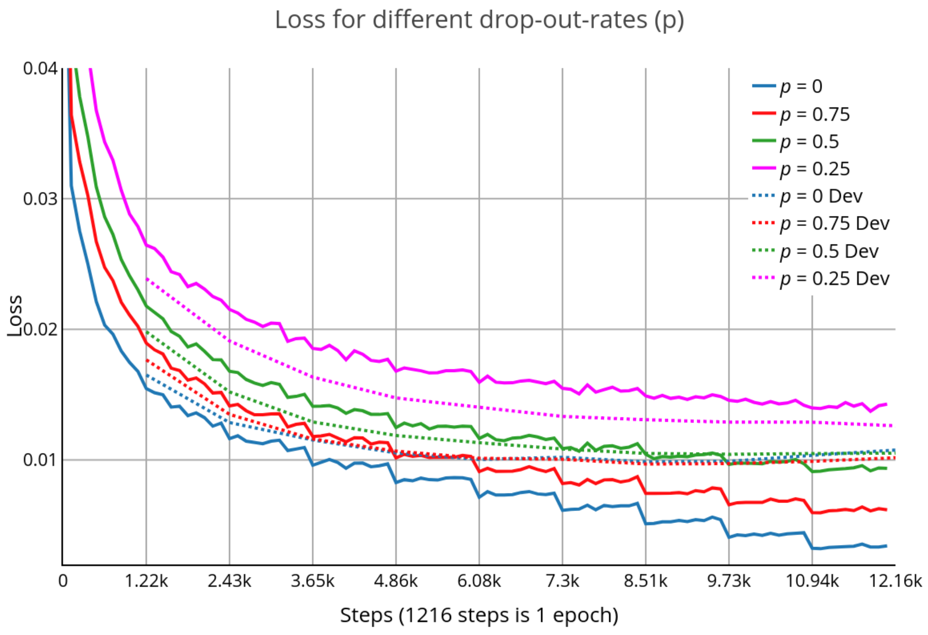

Figure 6.8: Loss for models with different drop-out rates.

In the first experiment the probability of drop-out was reduced from 50% to 0, meaning that no drop-out would be applied to the document embeddings during training. With no drop-out the models performance on the classification task on the training data dramatically improved, while the performance on the development data did not share the same increase. This is expected when regularization is removed, as this will more easily lead to the model overfitting, which will manifest itself as an increase in performance on the training data, but not on the development data.

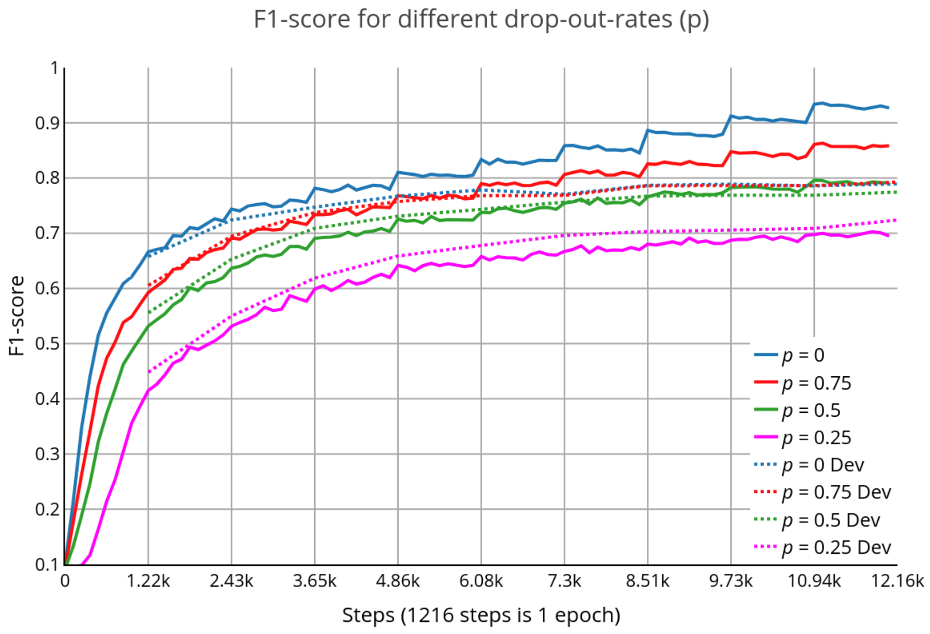

Figure 6.9: F1 score for models with different drop-out rates.

	Model	NDCG@100	F1 score
HR	p = 0.75	**0.091**	
	p = 0.5	0.090	
	p = 0	0.082	
	p = 0.25	0.080	
L	p = 0.75	**0.132**	
	p = 0.5	**0.132**	
	p = 0	0.118	
	p = 0.25	0.115	
TR	p = 0.75	0.125	
	p = 0.5	**0.129**	
	p = 0	0.113	
	p = 0.25	0.114	
Overall	p = 0.75	**0.127**	**0.792**
	p = 0.5	**0.127**	0.774
	p = 0	0.114	0.789
	p = 0.25	0.111	0.723
	BoW full	0.1000	
	Doc2Vec lemma	0.029	
	Random	0.000977	

Table 6.6: NDCG for the drop-out rate experiments and the three court instances, and F1 score on development data after ten epochs

The step-like increase in F1-performance at the start of each epoch is also more prominent than the other models. This can be an indication of overfitting as the 'benefit' of a training pass is not really visible until the model encounters the document again, at which point it remembers what it learned after the previous encounter without really using knowledge gained from the other documents. This behavior can also be observed in figure 6.8 on page 71. Also note the drop in performance as the epoch progresses before shooting up when the next epoch starts. This could indicate that the training is actually confusing the model as the epoch is progressing, but it still performs better than at the same point one epoch earlier because it is overfitting. In figure 6.9 on the preceding page, the difference between each data-point one epoch apart after around five epochs stays almost constant, gaining on average 0.05 each epoch. In figure 6.8 on page 71 it is also visible that the loss for the model with no drop-out starts lowest, but at the end of training, when overfitting is most likely occurring, the loss is rising and about to overtake the standard model. This further supports the theory that overfitting is indeed happening.

Another interesting behavior can be observed in figure 6.10. The model with no drop-out has an unusual jump in precision at the start, before dropping off, then slowly rising again. This happens because unlike the other models which always made a handful of 'present' predictions in the beginning, this model makes very few such predictions in the beginning. The model is more careful and makes a few, but good predictions, which leads to the precision rising. However, this also leads it to becoming the only model which at some point didn't make *any* 'present' predictions. However, after about an epoch the model makes the same amount of 'present' predictions as the other models.

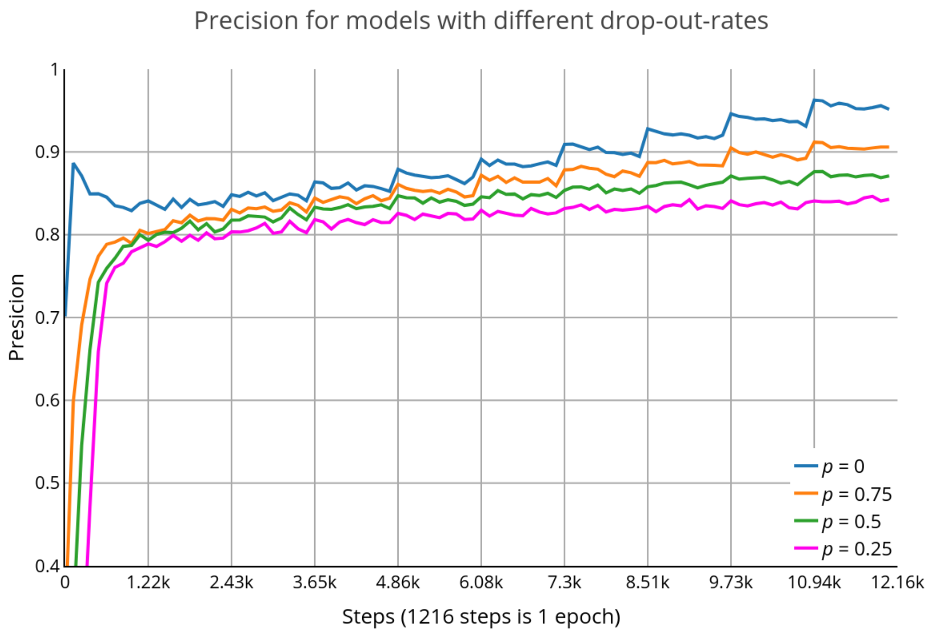

Figure 6.10: Precision for models with different drop-out rates.

What makes the large increase in performance on the classification task for the training data, and the quite normal performance on the development data, more interesting is the fact that the performance of the document embeddings were much worse. The model performed about 10% worse overall than the standard experiment with 50% drop-out. This could be the result of the model starting to overfit to the data, but it is still noteworthy that the performance on the development data is about the same as the other experiments, yet the performance of the embeddings are worse then the other experiments.

For the next experiment the drop-out probability was raised to 75%, meaning three out of four nodes were dropped out. This network also performed quite well on the classification task, beating the default parameters. Surprisingly both removing and increasing the drop-out rate led to improvements on the classification task. This fact might indicate that the model is not overfitting as severely as first thought, as increasing the regularization should combat overfitting. The 'steps' in performance might be explained by something else, but it is unknown what might cause them if not overfitting. But then again the difference in performance on the training data and development data strongly suggest overfitting is happening, and it might just be that the drop-out regularization is not effective enough. In contrast to the previous experiment, increasing the drop-out had little effect on the performance of the document embeddings, achieving a NDCG score on par with the standard model, as can be seen in table 6.11 on page 87.

Another interesting behavior in figure 6.9 on page 72 is the 'kink' in performance on the development data that happens after 7 epochs for both the models with no drop-out and increased drop-out. Both models converge at almost the same value from this point forward, even performing a synchronized sudden hop in performance, then plateauing.

Another experiment was conducted to study the effect of not entirely removing the drop-out, but lowering the drop-out probability to 0.25. This did also not give the results one would expect based on the previous experiments. The performance on both the training and development data was greatly reduced, although the relationship between the training and development performance was more stable than the other experiments. This could indicate that overfitting is not yet happening. The step-like behavior is also not present, which could corroborate this. It thus seems like *increasing* regularization increases overfitting, and *lowering* regularization decreases overfitting, while performing no regularization at all increases overfitting the most. From table 6.6 on page 72 one can also see that the loss in performance suffered by this model also means the performance on the document similarity task is lowered. From this set of experiments it is more difficult to draw any conclusions on the relationship between the performance on the classification task and the performance on the document similarity task, as there seem to be some deviation from the relationship seen in the previous experiments.

6.8 Adjusting the window sizes

(Zhang & Wallace, 2015) suggest performing a line-search over single window sizes to find the 'best' single window size, and then combining multiple different window sizes close to this ideal single window size. In the standard model the window sizes were a triplet of size (3,4,5), with 4 being the 'ideal' window size the triplet was centered around. In the next set of experiments five different single window sizes will be tested and compared against each other to find the best single window size among them, and then construct a window size triplet around this center window size. The five window sizes are: 4, 7, 10, 20, 30. 4 was selected as the first window size to be tested as it was the size the standard filter was centered around. In (Zhang & Wallace, 2015) they test single window sizes in the range from 1 to 30 and the four remaining window sizes were picked from this range. 7 was selected as the second window size to be tested as it was the size that gave best results in (Zhang & Wallace, 2015) for one of the datasets tested on. (Zhang & Wallace, 2015) suggest a range of 0 to 10 for datasets with short sentences, and a larger range for datasets with larger sentences. Since the dataset used in this thesis consist of large documents, window sizes of 10, 20 and 30 were also tested, as documents might need a much larger window size than sentences, and the sizes were also featured in the experiments performed in (Zhang & Wallace, 2015).

Almost all the hyperparameters in the network stayed the same as in the standard experiment. The only difference was that the window sizes were reduced to a single size and the number of filters for each window size was reduced from 200 to 100, as this was the size used in (Zhang & Wallace, 2015). This in turns means the document embedding size is lowered to 100. See table 6.1 on page 59 for the hyperparameters, figure 6.11 on the next page and 6.12 on the following page for the performance on the classification task, and table 6.7 on page 77 for the performance on the document similarity task. In figure 6.12 on the following page and 6.11 on the next page the models performances are normalized in relation to the model with a window size of 4 (f = 4), which will be used as the baseline for the single filter experiments, as it is the center size used in the standard model. This is done to make the figures easier to read, as there is little difference between the models and the relative performance between the models is more interesting than the relative performance to the other experiments. To avoid confusion with the word embedding size, the window sizes will be represented by a lower case 'f'.

Although (Zhang & Wallace, 2015) suggest that datasets with longer documents might benefit from larger window sizes, this does not seem to be the case for this dataset. Of course, this dataset and task is widely different than any of the sets used in (Zhang & Wallace, 2015) and it should come as no surprise that the recommendations made in the paper does not hold true for this data. Still, one would perhaps not expect that the ideal window size for single sentences would also be the ideal window size for entire documents. Both figure 6.11 on the following page, figure 6.12 on the next page and table 6.7 on page 77 all show that window sizes either side of 7 decrease performances. The relative close performance of the two models with window sizes 7 and 10 might sug-

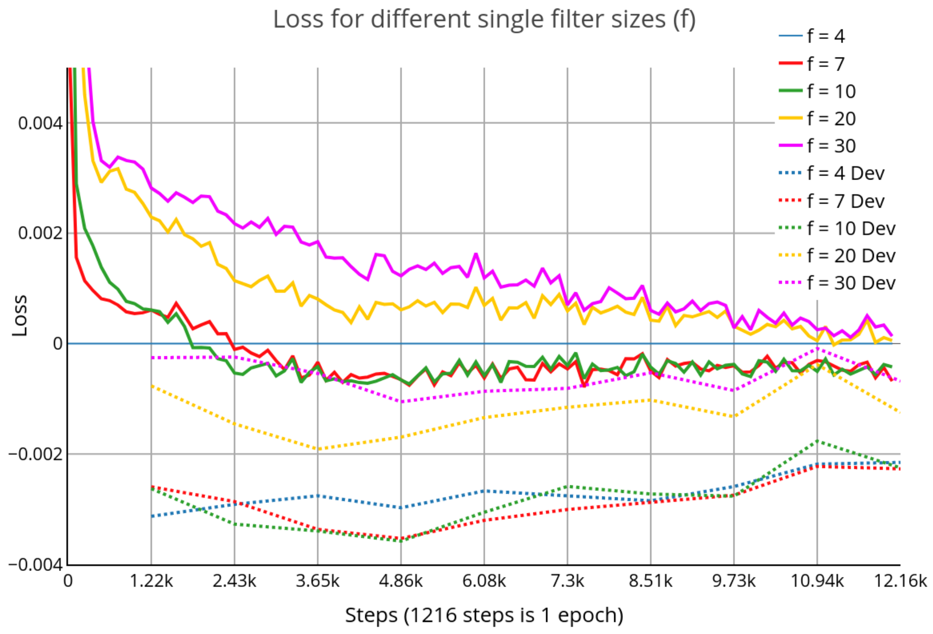

Figure 6.11: Loss for models with different single window sizes. The y-axis is normalized with relation to model ’f = 4’

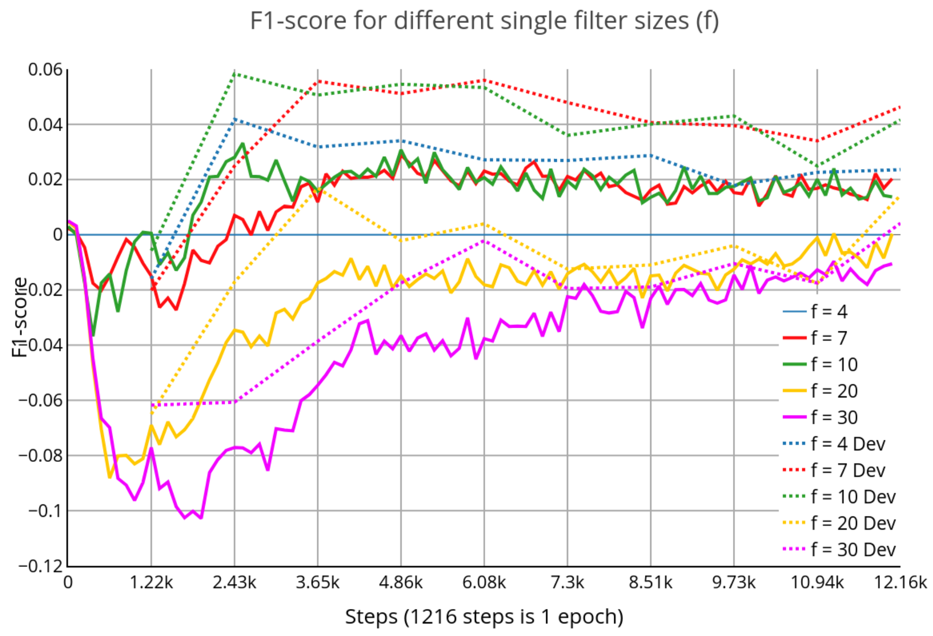

Figure 6.12: F1 score for models with different single window sizes. The y-axis is normalized with relation to model ’f = 4’

	Model	NDCG@100	F1 score
HR	f = 4	**0.056**	
	f = 7	**0.056**	
	f = 10	0.051	
	f = 20	0.047	
	f = 30	0.040	
	f = (3,4,5)	0.090	
L	f = 4	0.080	
	f = 7	**0.086**	
	f = 10	0.083	
	f = 20	0.075	
	f = 30	0.067	
	f = (3,4,5)	0.132	
TR	f = 4	0.075	
	f = 7	**0.080**	
	f = 10	0.078	
	f = 20	0.070	
	f = 30	0.062	
	f = (3,4,5)	0.127	
Overall	f = 4	0.077	0.63
	f = 7	**0.082**	**0.65**
	f = 10	0.079	0.64
	f = 20	0.071	0.62
	f = 30	0.063	0.61
	f = (3,4,5)	0.127	0.774
	BoW full	0.1000	
	Doc2Vec lemma	0.029	
	Random	0.000977	

Table 6.7: NDCG for the single window size experiments and the three court instances, and F1 score on development data after ten epochs

gest that the ideal window size lies somewhere in between them, either 8 or 9, but these window sizes were not tested. In (Zhang & Wallace, 2015), using the ideal single window size of 7 beats the performance of their baseline model, which has the same window sizes triplet as our standard model, but this does not happen in our dataset. The baseline model has three different filter-sizes, all with 200 feature maps each, making a final document vector of 600 features. The model with a single window size produces a document embedding only 100 features long, and this discrepancy in document embedding size probably accounts for most of the difference between the best single window size and the standard triplet. As can be seen in table 6.4 on page 67, reducing the size of the document embeddings severely impacts the performance, so I would not expect any model with only 100 total feature maps to beat the baseline, a model with 600 feature maps, regardless of window size or number of window sizes.

The goal of the next experiments was to test using multiple window sizes near the ideal single window size, as this would give results which were more easily comparable with the baseline model. Following the recommendation in (Zhang & Wallace, 2015), the window sizes tested were both combinations of different window sizes near the ideal size and copies of the ideal size. Two sets of window sizes were tested: (6,7,8), centering around the ideal size of 7, and (7,7,7) which was based on the recommendation in (Zhang & Wallace, 2015) that for some tasks simply increasing the number of filters for the ideal single window size was the best solution. For the rest of this section the (7,7,7) window size set will be reffered to as a 'set' of window sizes even though having three sets of 200 filters with window size 7 is the same as having a single window size of 7 and 600 filters. For these experiments the number of filters for each window size was increased back to 200, as this would produce a document embedding with the same amount of features as the standard model, which also uses three window sizes and 200 feature maps for each size.

Again, all the hyperparameters in the network stayed the same as in the standard experiment. The only difference was the multiple window sizes. See table 6.1 on page 59 for the hyperparameters, figure 6.13 and 6.14 on the following page for the performance on the classification task, and table 6.8 on the next page for the performance on the document similarity task. These figures are also normalized with regards to 'f = (3,4,5)'.

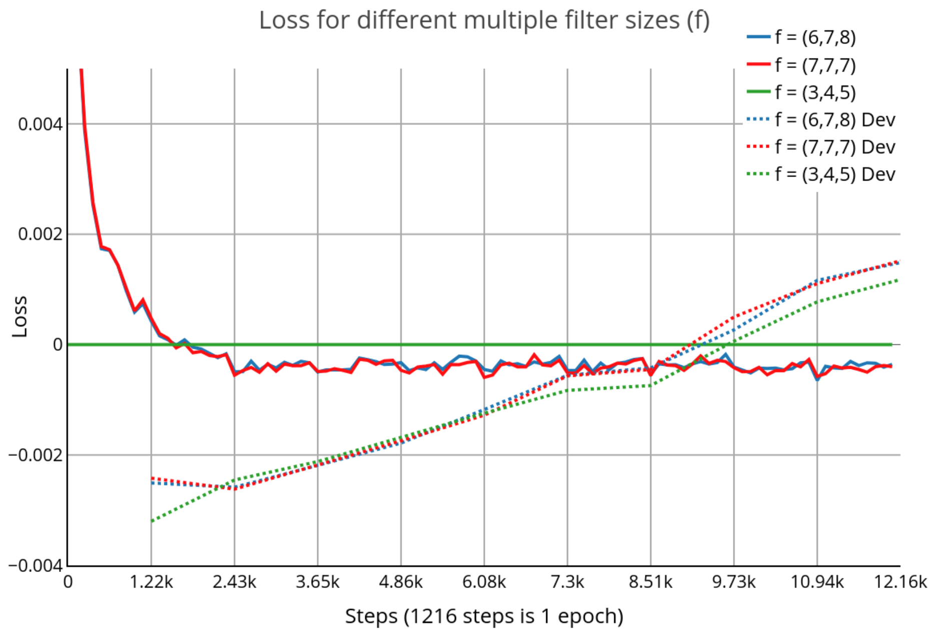

Figure 6.13: Loss for models with different multiple window sizes.

The first of the experiments moved the center value of the filter-size triplet to 7, the single window size which did the best in the previous experiments. This increased the performance on the training data, while the performance on the development data approximately followed the performance of the standard

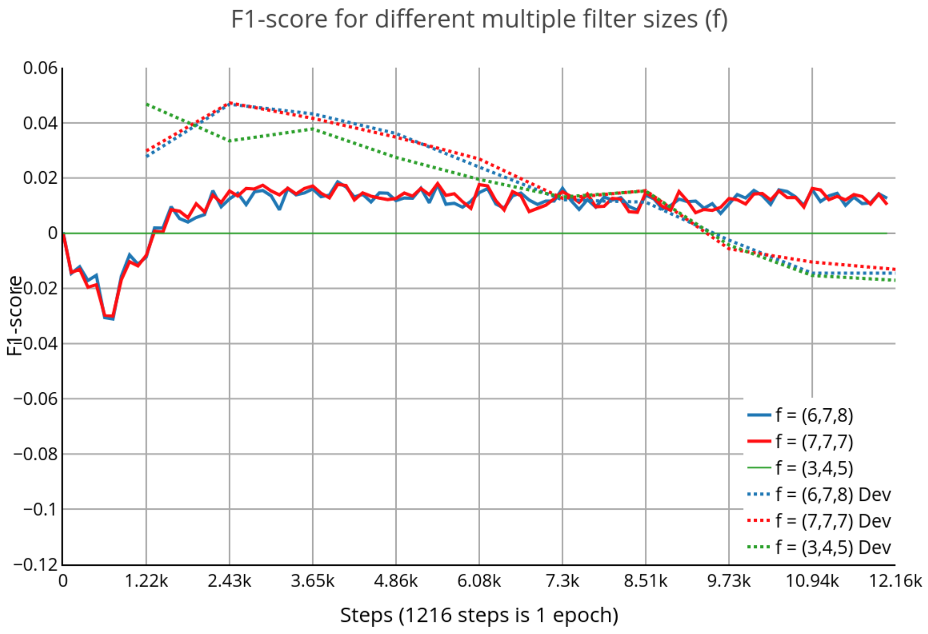

Figure 6.14: F1 score for models with different multiple window sizes.

	Model	NDCG@100	F1 score
HR	f = (6,7,8)	0.088	
	f = (7,7,7)	0.088	
	f = (3,4,5)	**0.090**	
L	f = (6,7,8)	**0.132**	
	f = (7,7,7)	**0.132**	
	f = (3,4,5)	**0.132**	
TR	f = (6,7,8)	**0.130**	
	f = (7,7,7)	**0.130**	
	f = (3,4,5)	0.127	
Overall	f = (6,7,8)	**0.127**	0.777
	f = (7,7,7)	**0.127**	**0.778**
	f = (3,4,5)	**0.127**	0.774
	BoW full	0.1000	
	Doc2Vec lemma	0.029	
	Random	0.000977	

Table 6.8: NDCG for the multiple window sizes experiments and the three court instances, and F1 score on development data after ten epochs

model on the development data. The performance on the document similarity task compared to the standard model did also not change. In (Zhang & Wallace, 2015) they suggest that simply using one window size and many filters could achieve top scores on some tasks. The next experiment then only used three sets of 200 filters with a size of 7, which as discussed earlier is the same as having a single window size of 7 and 600 filters. This model performed almost exactly the same as the previous experiment. It seems having one slightly smaller and one slightly larger window does not seem to have helped in picking out important features.

Table 6.8 on the preceding page show that centering the filter triplet around a higher value did not effect the final document similarity score. Although the previous experiments showed that a window size of 7 outperformed the lower window size of 4, which was the center size of the baseline model, this relationship did not continue when expanding these values into triplets or increasing the number of filters for a single window size.

The reason that higher window sizes does not seem to matter for this task might be a consequence of the training task. The network is attempting to predict the legal area tags for documents, and the tags are determined by the references in the documents. It is possible that the network is learning this relationship and adjusts the filters to pick out the references. These references would be a single word, or a couple of words long at most, and for the filters to pick them out they would not need to cover more than a few words at a time, and so the difference between covering 4 or 7 words is negligible. It is highly likely that some other classification task for this dataset would behave differently and possibly have a larger performance gap between the two window triplets.

In the single window size experiments the 7-centered model performed better than the 4-centered model derived from the standard model. In the multiple window size experiments, the 7-centered models did not perform better than the standard model. The difference between the single and multiple window size experiments with regard to the difference between the standard model and the 7-centered model performance can possibly be explained by the fact that the models with multiple window sizes have many more filters. For the single window size experiments, only 100 filters were used, and so the size of the filters might have a bigger impact, as 100 filters might not be enough to pick out all the important features. This might lead to larger window sizes having a small advantage as they cover more words and can 'see' more at once, which might help it make more informed decisions with few filters. When having multiple window sizes the total number of filters was 600, and so the models might have enough filters to go around to pick up the important features regardless of the window size and how much each filter 'sees', and thus a 4-centered and a 7-centered filter triplet performs the same.

6.9 Using full forms instead of lemmas

The final experiment will study the effect of not lemmatizing the corpus. The motivation for this comes from the experiments done in section 3.4 on page 42 where both the BoW model and the Doc2Vec model were tested on lemmatized text and full forms. The results from those experiments showed that there was little difference between the two preprocessing methods, but Doc2Vec performed slightly better using lemmas. As both CNNs and Doc2Vec are based on the neural network architecture, I expect CNNs to also perform better with lemmas, but as the classification task is closely related to the references in the documents, only a relatively few words might be important, and thus having nuanced full forms available might lead to better performances.

All the hyperparameters in the network stayed the same as in the standard experiment, but this experiment utilized a corpus of full forms. See table 6.1 on page 59 for the hyperparameters, figure 6.15 and 6.16 on the following page for the performance on the classification task, and table 6.9 for the performance on the document similarity task.

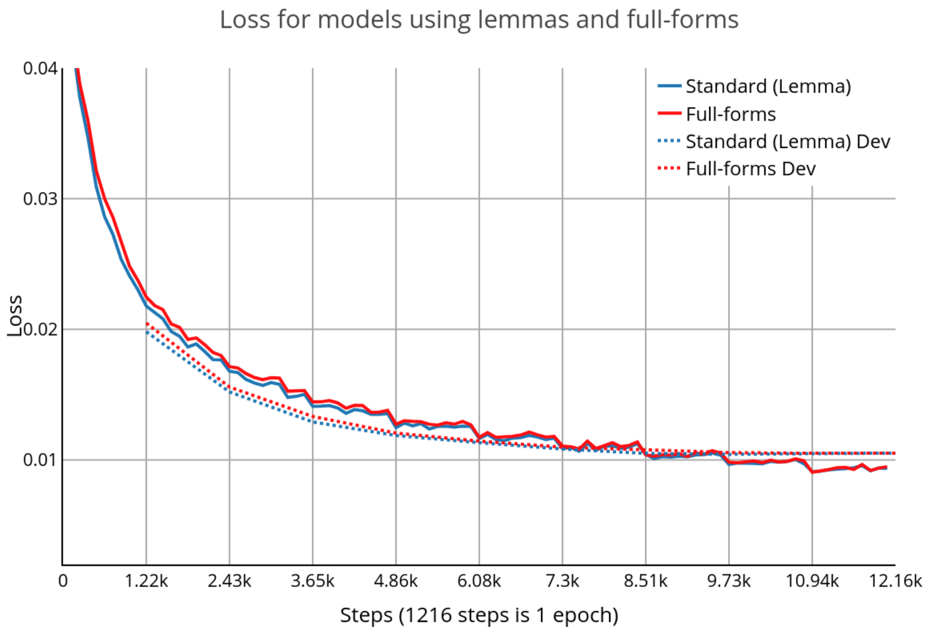

Figure 6.15: Loss for models using lemmas and full forms.

As we can see in figures 6.15 and 6.16 on the next page, both the performance on the training data and the development data is very close for both models during training. The lines almost completely overlap. The models performance on the document similarity task is slightly lower than the standard model, as can be seen in table 6.9 on the following page. Lemmatizing the corpus gave an 16% increase in performance for a Doc2Vec model, as shown in table 3.2 on page 43. For the document embeddings produced in this experiment, lemmatization of the corpus resulted in an increase in performance of about 2.4%. The reason for the performance on the classification task not dropping when using full forms might be because the classification task is closely related to the references in a document, and the network might learn to pick these few words out no matter the preprocessing.

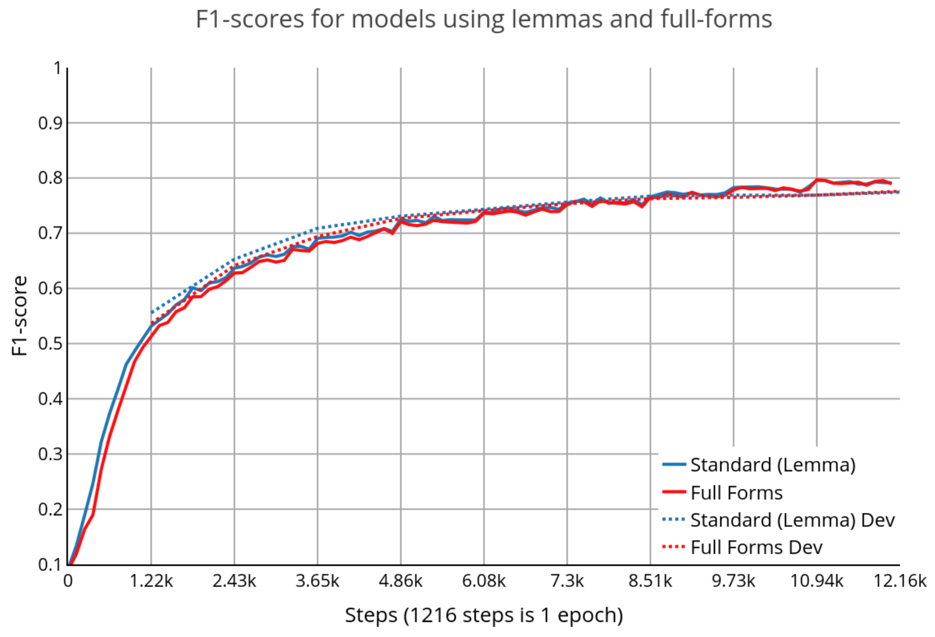

Figure 6.16: F1 score for models using lemmas and full forms.

	Model	NDCG@100	F1 score
HR	Standard	**0.090**	
	Full forms	0.087	
L	Standard	**0.132**	
	Full forms	0.129	
TR	Standard	**0.129**	
	Full forms	0.126	
Overall	Standard	**0.127**	0.774
	Full forms	0.124	**0.776**
	BoW full	0.1000	
	Doc2Vec lemma	0.029	
	Random	0.000977	

Table 6.9: NDCG for experiment where corpus was not lemmatized and F1 score on development data after ten epochs

6.10 Combining the best hyperparameters

After several experiments where some of the different hyperparameters of the network have been adjusted, a final experiment will be conducted to evaluate the performance of a network with a combination of the best hyperparameters from the other experiments. While one might think that this model should give the best performance on both the classification and the document similarity task, this is not the case. In reality the parameters are most likely dependent on each other, meaning two hyperparameters which on their own gave better results, does not necessarily give better results when they are combined. Due to time constraints, an exhaustive grid search over all possible combinations of hyperparameters, or random combinations of them, was not possible.

From section 6.4 on page 62 a word embedding size of 300 was selected, as it gave the highest score on both the classification and document similarity tasks. From section 6.5 on page 65 it was shown that a document embedding size of 900 was the best choice, as it also gave the best performance for both tasks in section 6.5 on page 65. From section 6.6 on page 68 it was decided to not use pre-trained word embeddings in the word embedding layer, as this did not improve the performance. From section 6.7 on page 71 a drop-out rate of 0.5 was selected as it performed the same as increasing the drop-out to 0.75 on the document similarity task. The filter-sets (3,4,5) and (6,7,8) performed as well as the (7,7,7) filter set, and so the (7,7,7) filter-set was chosen because section 6.8 on page 75 showed that for that task, using a single filter with window size 7 was a better choice than a window size of 4. With a document embedding size of 900, the number of filters for each of the three window sizes was 300. In addition to this model, another model with the same hyperparameters, except a word embedding size of 100, was also trained. Since the only difference between the two models is the word embedding size, they will be known as 'Tuned 300' and 'Tuned 100', respectively.

See figure 6.17 and 6.18 on the next page for the performance of the tuned models on the classification task, and table 6.10 on the following page for the performance on the document similarity task.

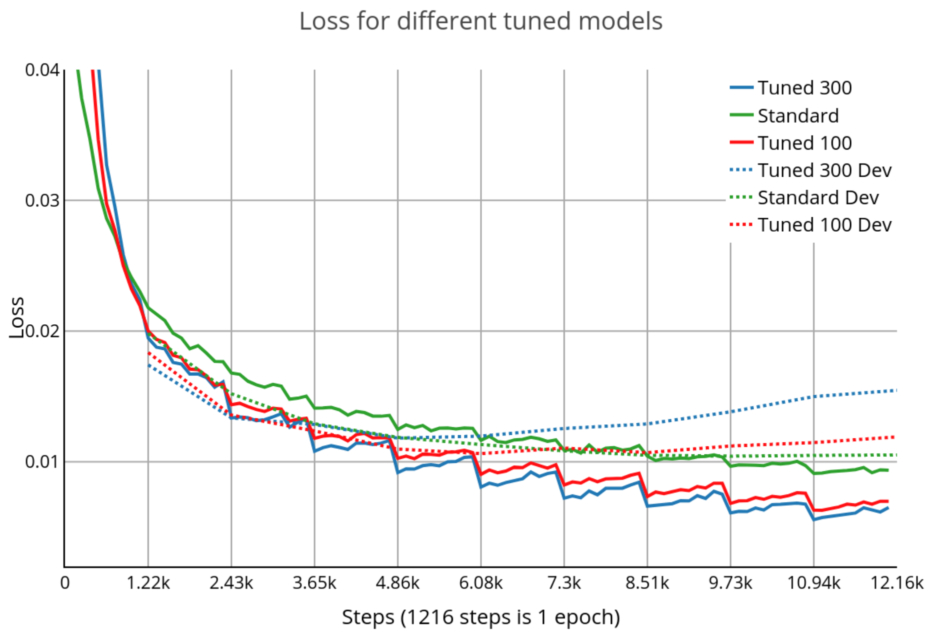

Figure 6.17: Loss for different tuned models.

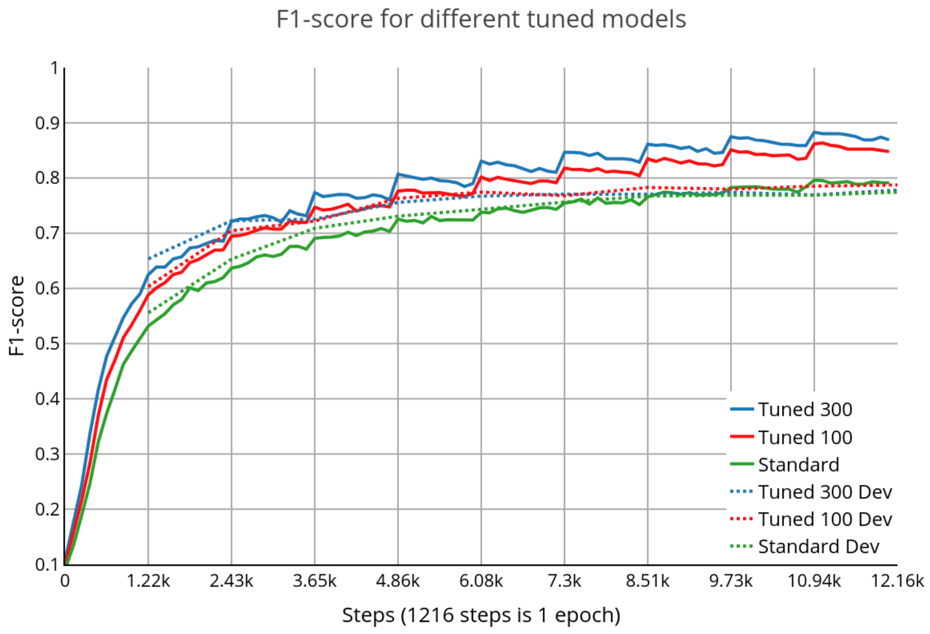

Figure 6.18: F1 score for different tuned models.

	Model	NDCG@100	F1 score
HR	Tuned 100	**0.091**	
	Standard	0.090	
	Tuned 300	0.052	
L	Tuned 100	**0.139**	
	Standard	0.132	
	Tuned 300	0.089	
TR	Tuned 100	**0.132**	
	Standard	0.129	
	Tuned 300	0.079	
Overall	Tuned 100	**0.133**	**0.787**
	Standard	0.127	0.774
	Tuned 300	0.083	0.778
	BoW full	0.1000	
	Doc2Vec lemma	0.029	
	Random	0.000977	

Table 6.10: NDCG for the tuned models and the three court instances, and F1 score on development data after ten epochs

As we can see from table 6.10, the tuned model with a word embedding size of 300 only achieved a score of 0.083 on the document similarity task, far below any of the other CNN experiments. As previously stated, the hyperparameters for the network are most likely dependent on each other, and simply using all the 'best' hyperparameters might not lead to the best model. In figure 6.17 on page 83 one can see that the loss for the model starts rising after about six epochs, earlier than any other model, and by a much larger amount, indicating much stronger overfitting. There is some evidence of this behavior also for the model with a word embedding size of 300 in figure 6.2 on page 63. Perhaps the model had too many weights, which allowed overfitting to more easily happen, and using a higher drop-out rate could have prevented some of this. After ten epochs the loss has reached a similar level as after one and a half epochs of training. This could mean the model could have 'regressed' in performance to a point which is equivalent of one to two epochs of training. With this in mind, the score on the document similarity task is not so strange. It should be noted however that the F1 score for the model does not fall as overfitting is happening, it only plateaus like most of the other models.

The other tuned model appears more successful. It seems that limiting the word embedding size to 100 prevented the model from overfitting to the same extent as the other tuned model, although figure 6.17 on page 83 show that the loss for this model also starts to rise towards the end. But it still manages to set the best score on the document similarity task for any model, achieving an NDCG@100-score of 0.133.

6.11 Summary of CNN hyperparameter experiments

In the previous sections we have seen the results of several different hyperparameter experiments. The hyperparameters of the CNN is most likely dependent on each other, and due to time constraints, a complete grid search over the hyperparameters to find the best combination was not feasible, and so several smaller experiments were conducted, altering a single hyperparameter at a time. Finally, the best hyperparameter configurations from the experiments were combined into two experiments where I studied the effect of combining the hyperparameters. One of the models from these two experiments achieved the highest scores on both the CNN classification task and the document similarity task. In this section the different experiments and their results will be quickly summarized. See table 6.11 on page 87 for a complete table of the results from every experiment. Due to time constraints, a manual inspection of some of the retrieved documents for the best model, like in section 2.4.3 on page 25, was not performed.

In both the experiments where the size of either the word embeddings, the document embeddings, the window size or the amount of drop-out were altered, increasing this size always led to a better score on both the classification task and the document similarity task. Lowering this size always led to worse performance. Most of the experiments consisted of only one experiment increasing the value of a hyperparameter and one decreasing the value, and so it is unclear to which extent one could increase the value and still see notable improvements on the evaluation tasks.

For the experiment on the single window size, the size was only increased.

This experiment showed that increasing the size beyond a certain point led to a decrease in performance. When experimenting with a trio of window sizes centered around a central value, the increase in performance when increasing the window size we had observed in the previous experiment was not present. This shows the hyperparameters dependency on each other. It is unknown which combination of hyperparameters would show the same relationship between a trio of window sizes and the performance on the evaluation task, if any.

Using pre-trained word vectors to initialize the word embedding layer did not have any positive effects on the performance. This will be explored and discussed further in section 7.1 on page 88.

Lemmatizing the corpus led to a small increase in performance over a model which used full forms. However, for the classification task, both full forms and lemmatization performed almost completely the same.

In almost every experiment with a CNN model, the performance on the document similarity task by the document embeddings was always best on documents from the Courts of Appeal and worst on the Supreme Court documents. In table 2.2 on page 21 we can see that the documents from the Courts of Appeal are neither the longest nor shortest, and they do not have the fewest or most references in them. It is therefore perplexing that these documents always led to the document embeddings performing best. One would expect the performance of the document embeddings was more strongly tied to the amount of words in a document, and as such the documents from the District Courts would have the best performance. Or if the CNNs are learning to only pick out the references in a document one would likewise expect the documents from the District Courts to have the best performance. Perhaps the slightly more condensed information in the Courts of Appeal documents led to better embeddings, and at the same time the Supreme Court documents were *too* condensed, which led to even worse embeddings than the documents from the District Courts.

The best hyperparameters from each experiment were combined into two models for the last set of experiments. One model used a word embedding size of 100 and the other a word embeddings size of 300. The latter model achieved a very low score on the document similarity task. The reason for this was most likely because of overfitting. The other model did not overfit to the same extent and achieved the best results on the document similarity task out of all the experiments. However, these results were only about 4.7% better than the standard model.

Pearson's r was calculated for the NDCG-scores and the F1 scores to determine the relationship between the CNN's performance on the document similarity task and the classification task. Ideally we would like this to be a strong relationship, because that means that when training the network, we would also be training the embeddings to perform better on the document similarity task, even though that is an entirely different task. When selecting a classification task for the CNN to train on I tried to find a task which I believed would have a strong relationship between these two tasks, however this was only speculations, as there was no results to give any indications of the

Model	NDCG@100	F1 score
W = 300	0.128	0.782
W = 100	0.128	0.780
W = 50	*0.127*	*0.774*
W = 25	0.118	0.768
D = 900	0.132	0.788
D = 600	*0.127*	*0.774*
D = 300	0.111	0.743
Randomly initialized	*0.127*	*0.774*
Initialized with LCC W2V	0.125	0.788
Initialized with NNC+LCC W2V	0.116	0.759
p = 0.75	0.127	**0.792**
p = 0.5	*0.127*	*0.774*
p = 0.25	0.111	0.723
p = 0	0.114	0.789
f = (7,7,7)	0.127	0.778
f = (6,7,8)	0.127	0.777
f = (3,4,5)	*0.127*	*0.774*
Lemmatized	*0.127*	*0.774*
Full forms	0.124	0.776
Tuned 100	**0.133**	0.787
Standard	0.127	0.774
Tuned 300	0.083	0.778
BoW full	0.100	
Doc2Vec lemma	0.029	
Random	0.000977	

Table 6.11: Overall NDCG for all the experiments, and F1 score on development data after ten epochs. The experiments with a single window-size are omitted due to them involving more than one hyperparameters being changed. Each group is sorted according to the NDCG-score, with the best scoring model on top. The models in cursive are the standard model.

relationship yet. After all the experiments I have more data to approximately determine the relationship between the two tasks. The Pearson's r of all the results is calculated to 0.248, indicating a weak positive relationship between the two tasks. I want the goals of the classification task to align with the goals of the document similarity task, but not too much, as this might indicate that the classification task is too similar to the document similarity task and the models are learning representations which might only be good for getting a high score on the chosen evaluation task, and not other ways of evaluating document similarity. On the other hand, the Pearson's r score of 0.248 might be a little bit too weak of a relationship.

Chapter 7

Further experiments in the context of CNNs

In this section I will perform some additional experiments pertaining to the CNNs and experiments performed in the previous chapter. I will take a close look at the word embedding layer of a few CNNs from the previous chapter and compare the word embeddings extracted from the CNNs *after* ten epochs of training to the embeddings from section 4 on page 45. In the last section I will evaluate the documents embeddings extracted from a few of the CNNs trained in the previous chapter on the retrieval task from section 2.4.2 on page 23. This was motivated by the fact that the retrieval task in section 2.4.2 was based on actual human made document groupings, and evaluating the document embeddings on this task might provide some more insight into the performance of the embeddings.

7.1 Evaluating word embeddings from a CNN

A convolutional neural network (CNN) is trained by updating the weights in the network. In the networks presented in this thesis, during training, the set of weights which make up the word embeddings are also trained and adjusted, even when pre-trained word embeddings are provided as an initialization for these weights. In chapter 4 on page 45, word embeddings were produced by different Word2Vec models and later used as initializations for the word embedding-layer of the the CNNs in section 6.6 on page 68. In this section I will work backwards and extract the word embeddings from a CNN which as been randomly initialized and trained, and evaluate them on the same evaluation tasks as the other Word2Vec models were in chapter 4 on page 45. The word embeddings were extracted from the 'tuned 100' model from section 6.10 on page 82, as it was the model which performed best on both the document similarity task and the classification task. In addition, the word embeddings from both networks with pre-trained word embeddings from section 6.6 on page 68 were extracted and tested. In this section, the process of further training a set of word embeddings in the word embedding layer of a CNN will be known as 'tuning', and the word embeddings extracted after tuning will be known as 'tuned' word embeddings.

See table 7.1 for the performances on the analogical reasoning task and table 7.2 for the performances on the synonym detection task. The 'tuned 100' model is omitted from these tables as it did not score any points. I will discuss this a little bit later.

	LCC		NNC+LCC	
Task	Before	After	Before	After
Common capital city	20.8	18.1	**36.2**	**36.2**
All captial cities	9.5	11.9	**39.9**	**39.9**
Currency	0.0	0.0	**42.5**	**42.5**
City-in-county	7.7	7.7	**16.2**	**16.2**
Man–Woman	56.7	56.7	66.3	**68.6**
Total semantic accuracy	12.6	12.6	28.8	**28.9**

Table 7.1: Accuracy for the analogical reasoning task for two Word2Vec models and the same word embeddings after being tuned in the word embedding layer of a CNN for ten epochs.

	k = 1		k = 5		k = 10	
Model	P	R	P	R	P	R
LCC	5.7	3.8	12.3	8.3	15.9	10.6
Tuned LCC	5.7	3.8	12.3	8.3	15.8	10.6
NNC+LCC	8.0	7.0	16.6	14.5	21.3	18.7
Tuned NNC+LCC	8.0	7.0	16.5	14.5	21.2	18.6

Table 7.2: Results for the synonym detection task for two Word2Vec models and the same word embeddings after being tuned in the word embedding layer of a CNN for ten epochs. k denotes how many neighbors are included.

Let us first focus on the word embeddings extracted from the model with word embeddings pre-trained on only the LCC. The performance of the word embeddings on the evaluation task ware largely uneventful as the word embeddings scored almost exactly the same points as prior to tuning. On the analogies task the only difference between the models was that after tuning the word embeddings were better at answering the questions about uncommon capital cities, while equally worse as answering the questions about *common* capital cities. It is almost as if it transfered the knowledge from one domain to another almost identical domain. In the end it achieved exactly the same overall score as prior to training. On the synonyms task the results were also exactly the same for both the first and the five first words, while looking at the top ten closest words the model had a slightly lower precision after training.

Next, the word embeddings from the network which was initialized with the pre-trained word embeddings from a Word2Vec model which was trained on the Norwegian News Corpus (NNC) and the LCC were extracted and tested. In chapter 4 on page 45, this Word2Vec model performed best on the analogies task, achieving a total semantic accuracy of 28.8. After tuning ten epochs in the word embedding layer of a CNN, the word embeddings performed slightly better, setting a new score of 28.9. It performed the same on all of the tasks, with the exception of the man–woman relationship task. On this task it had an increase in performance of about 3.5%. Out of all the tasks, this task had the best potential for an increase in performance from training on the LCC. The Word2Vec model trained only on the LCC show that the man–woman relationship is the strongest relationship picked up by the model, achieving a score of more than double that of the next highest score, the common capital cities. See section 4.1 on page 47 for a discussion of why this might be. For this reason it is likely that more training on the LCC would lead to learning this relationship better. On the synonym detection task the model performed slightly worse at both the five first and the ten first words.

The word embeddings extracted from the 'tuned 100' model, which was initialized with random word embeddings is omitted from both table 7.1 on the preceding page and 7.2 on the previous page as it scored no points on both tasks. This was perhaps expected, as the classification task was not aimed at learning good word embeddings, but good document embeddings. This seems to have made the word embeddings adapt to the task at hand and transformed them into representations which will benefit the classification of legal areas, not the task of learning the relationship between single words. The word embeddings are interpreted by the convolutional layer before being squeezed down into the document embeddings, meaning the word embeddings do not necessarily need to make sense to humans, as long as the other weights further into the network can decipher and interpret the embeddings into something ultimately useful for the classification task.

We have seen from the pre-trained models that the difference between the word embeddings before and after training is minimal, indicating that the word embeddings are not that important in the grand scheme of things, and therefore the randomly initialized word embeddings have probably only been altered very slightly to pick up the signal the network needs, and then it has let the other word embeddings be. As discussed in section 6.11 on page 85, it is possible the network has only learned to pick out the references or a few key words from the documents, and this would lead to many of the word embeddings being largely left alone. Unfortunately, the evaluation tasks are very general and would probably not pick up this very specific change in the embeddings, and I have not devised any tests of our own to explore whether this is happening.

The models in section 6.6 on page 68 share the same hyperparameters, the only difference is the initialization of the word embeddings. As we have seen, the word embeddings are only slightly altered, no matter the initialization method. This explains the close performance of the CNN models with pre-trained word embeddings, as it does not make a large impact how the word embeddings were initialized in the beginning and so the networks perform almost the same. In figures 6.6 and 6.7 on page 69 we can see that the models are so similar that the lines in the figure almost completely overlap.

7.2 A closer look at the word embeddings

To get some more insight into what ten epochs of tuning in the word embedding layer of a CNN have done to the three different sets of word embeddings, a simple experiment was set up and will be discussed in this section. This discussion is on the boundaries of the scope of this thesis, but is motivated by the fact that it might shed some light on the interplay between the Word2Vec models and the CNNs described in this work. With this in mind there will be limited discussion of the results in this section and I will try to focus the attention back towards the Word2Vec models and the CNNs rather than exploring further away from the scope of this thesis. Looking at the actual word embeddings before and after the tuning can perhaps tell us something about what has happened to them during the tuning, and what kind of influence the CNN had on them. Three words from three categories were selected for closer inspection. The three categories and words are domain specific words such as *forlik* (settlement), *rett* (court) and *straffelov* (penal code), normal content words such as *kjøleskap* (refrigerator), *snakke* (talk) and *mann* (man) and function words and numbers such as *og* (and), *i* (in) and lastly the number *23*.

The word *forlik* was selected because it is a non-legal word describing agreement or settling something, but is primarily used in a legal context. Even more divisive is *rett*, which in Norwegian has a plethora of widely different meanings, such as court, straight, dish and correct. It is therefore interesting to see what happens to this embedding with further tuning, as the word is mostly used in the context of a court in the Lovdata Court Corpus (LCC). *straffelov* was selected as it is the most referenced law in the entire corpus and the word is rooted firmly in the legal domain. The content words were selected largely with no particular intent, only to be normal words which could appear in a lot of different contexts. However, the content word *mann* was especially selected for this experiment as gender relationships were one of the analogies tasks Word2Vec models trained on the LCC excelled at, even though the word does not carry any extra meaning in a legal context. The function words were selected to study the change in the embeddings of words which are used a lot, no matter the context, as they are fundamental building blocks of language.

Let us first consider the amount of change the different embeddings experienced over the course of ten epochs of tuning. See table 7.3 on the next page for an overview of the change for the different words and different models. The amount of change was measured as the euclidean distance between the word embedding before tuning and the same word embedding after tuning, as well as the cosine similarity of the two embeddings. The euclidean distance will measure the amount of change an embedding has undergone in terms of distance, while the cosine similarity will measure the amount of change in semantic meaning. A cosine similarity of 0 indicates a 90 degree angle between two words, which mean they are semantically unrelated, while a cosine similarity of -1 represents an angle of 180 degrees, which mean the words are related again, but semantically the *opposite*. The mean for the entire vocabulary for each measure is also reported. In addition, the Pearson's r between each column and the frequency of the each word in the LCC is also reported. This was calculated to uncover the connection between the amount of training data for each word and the amount it changed.

initialization:		LCC		NNC+LCC		Randomly	
Word	Freq	Euc	Cosine	Euc	Cosine	Euc	Cosine
forlik	98918	2.14	0.993	0.84	0.999	8.25	0.082
rett	6258363	2.39	0.995	1.16	0.997	7.75	0.087
straffelov	453999	2.60	0.999	2.42	0.994	8.58	-0.122
kjøleskap	965	1.23	0.998	0.56	0.999	8.91	-0.163
snakke	27335	0.90	0.999	3.14	0.991	7.83	0.082
mann	2047381	2.02	0.992	2.11	0.994	8.02	0.017
og	9530594	4.41	0.900	5.06	0.903	8.53	-0.063
i	99525680	4.22	0.944	3.40	0.967	8.09	0.063
23	3714384	2.55	0.991	2.21	0.993	8.05	0.073
Mean	234	0.11	0.992	0.02	0.999	8.15	0
Pearson’s r		0.61	0.46	0.34	0.26	-0.14	-0.20

Table 7.3: Amount of change in word embeddings for different initializations after ten epochs of tuning in a CNN, measured in euclidean distance and cosine similarity between the embedding before tuning and after. Mean is reported for both measures. Pearson’s r is calculated between the column and the frequency column.

Let us first consider the word embeddings which have been pre-trained only on the LCC. In table 7.3 we can see that on average there is little change in the semantic meaning of the word embeddings, with an average cosine similarity of 0.992 between the words before and after tuning. We can see that the domain specific words and two of the content words are above this average, meaning those embeddings have changed less than average semantically. The more common and less informative content and function words have changed more. Perhaps the function words give the CNN little information for the classification task, and so the CNN changes them a lot to either align them with a pattern which helps the network, or to remove their influence as much as possible from the network. However, the Pearon’s r for the two columns tell us that the relationship between the frequency of the words in the training data and the amount of change has a medium positive link. So the greater amount of change for the function words and *mann* is somewhat linked with the fact that they appear more often in the training data.

For the word embeddings which have been pre-trained on *both* the NNC and the LCC, we can see in table 7.3 that they follow almost the same pattern as the two previous columns, but there are some differences. There is much less change in these embeddings, with an average cosine similarity of 0.999. In addition, there is a lower relationship between the word frequencies and the amount of change. Yet, the relative relationship between the words in this test is almost the same as the other word embeddings. The domain specific words mostly changing the least, and the function words change the most, but this time almost all of them change more than the average.

The randomly initialized word embeddings undergo a lot of change. As all the embeddings start out as random vectors, there is no pattern in how much they change, as the amount of tuning they need is tied to the state they are in before tuning, which is random. This also leads to a very weak negative relationship between the word frequencies and the amount of change. It is also interesting to note that the average amount of semantic change, as measured by the cosine similarity, is an almost exactly perfect 90 degrees, meaning the embeddings totally changed their semantic meaning. This is expected, as they started as random noise with *no* meaning. All the word embeddings sets had a very strong relationship between the amount they changed in euclidean distance and the change in semantic meaning, as measured by the cosine similarity. This indicates that when the embeddings changed it most often resulted in a change in semantic meaning, and not the embedding becoming 'stronger' or pointing *more* in the same semantic direction.

As the results for this section is based solely on nine out of 1,5 million words, they are not guaranteed to hold true for the rest of the vocabulary. But the results seem to follow somewhat expected patterns, and the average measurements tell us that there is little change for most of the words. While it is disappointing that the network seems to not leverage the large amounts of information encapsulated in the word embeddings, one of the greatest strengths of machine learning is the ability computers have to see patterns and meaning across huge amounts of data. While it seems the tuning of the word embeddings in the embeddings layer of a CNN had little impact on the embeddings, the networks were not training on a task which encouraged it to build stronger semantic relationships between the word embeddings. Most likely it made some small adjustments to allow a signal to flow more strongly to the other parts of the network, like the convolutional layer. The analysis performed in this section is already on the boundaries of the scope of this thesis and so while it is possible to investigate the word embeddings further, I considered any further analysis of the network and the flow of information as out of scope for this project.

7.3 Evaluating CNN models on favorites lists

In section 2.4.2 on page 23, to get a clearer picture of the validity of the Reference Vector System (RVS), it was evaluated on a simple task. The task consisted of predicting which documents appear alongside a target document in favorites lists made by users of Lovdata. The goal of the experiment was to see how well the RVS corroborated the grouping of documents made by users, which I considered one of the best approximations to a human-made document similarity gold standard. In this section I will evaluate the document embeddings made by a couple of the CNN models tested in chapter 6 on the same task in the same manner as in section 2.4.2 on page 23. This can give us some more insight into the performance of the CNN models as well as on how the three different ways of representing documents relate to each other.

The models tested in this section are the following: the standard model will again serve as a baseline for the other models, in addition, the best performing model, the 'tuned 100' model, will be tested, as well as one of the worst performing models, the model with a document embedding size of 300. The best performing models from some of the experiments will also be tested, specifically the best performing models from the word embedding size-, document embedding size-, drop-out rate- and the window-size experiments. See table 7.4 for a reminder of the hyperparameters of some of the CNN models tested in this section. The models were evaluated on the eligible documents from the entire evaluation subset, meaning all the query documents were documents from the evaluation subset and all the target documents were from the training subset. Like in section 2.4.2 on page 23, query documents which appeared alongside less than 10 other documents were not eligible and not used in the evaluation.

Parameters	Standard	Tuned 100	Worst
Size of training set	77,806	″	″
Max document length	17,000	″	″
Batch size	64	″	″
Word embedding size	50	100	″
Window sizes	$(3,4,5)$	$(7,7,7)$	″
Total filters	200	300	100
Doc embedding size	600	900	300
Drop-out rate	0.5	″	″
Nr. of training epochs	10	″	″
Word emb. initialization	random	″	″

Table 7.4: Hyperparameters for some of the CNN models evaluated on the favorites list task. See table 6.1 on page 59 for the rest.

	Against favorites lists		Against RVS	
	Precision@100	MAP@100	NDCG@100	F1
Reference Vectors	0.187	**0.297**		
p= 0.75	**0.202**	0.292	0.127	0.778
D = 900	0.201	0.288	0.128	0.78
Standard	0.197	0.282	0.127	**0.792**
f = (7,7,7)	0.187	0.270	0.114	0.789
W = 300	0.186	0.270	0.127	0.774
Tuned 100	0.185	0.268	**0.132**	0.788
D = 300	0.184	0.262	0.071	0.62
Randomly picking	0.003			

Table 7.5: Precision@100 and Mean Average Precision@100 for the different CNN models evaluated on the favorites list task, sorted according to the MAP@100 score. The NDCG@100 for the models when evaluated against the RVS is reported, as well as the F1 score for the model on the CNN classification task.

In table 7.5 we can see that most of the models perform almost as well as the RVS. This is interesting, as for example the standard CNN model only achieves an NDCG score of 0.127 when comparing directly against the RVS, while when comparing both of them against the favorites list task they perform almost as well. It should be noted that this is two entirely different tasks and measures used. The favorites list task does not involve the target document set being ranked and the sizes of the target set varies, while when comparing the RVS and the CNN models we are comparing two ranked lists limited to the top 100 documents, which is why I use the NDCG@100 measure and not the MAP@100 measure.

From table 7.5 on the previous page we can also see that the CNN model with document embedding size of 300, which performed most poorly on the document similarity task, also performs most poorly on the favorites lists task. Surprisingly, the CNN model which performed best on the document similarity task, the tuned model with a word embedding size of 100, has the second lowest performance against the favorites lists. Two other models, the model with 7-centered window sizes and the model with a word embedding size of 300, which both outperform the standard model on the document similarity task, albeit only just, are both beaten by the standard model on this task.

The CNN model with a drop-out rate of 0.75 narrowly beats the model with the increased document embedding size. As we can see, lowering the document embedding size leads to worse performance, and increasing it lead to better performance. This is to be expected, the task revolves around the document embeddings, and as such, larger embeddings are able to hold more information which can be utilized to make better decisions.

Pearson's r was calculated between two column-pairs in table 7.5 on the preceding page: The relationship between the models performance on the classification task and the performance of the document embeddings when compared against the favorites list, and the relationship between the models performance on the document similarity task and the performance of the document embeddings when compared against the favorites list. The Pearson's r for the first relationship was calculated to 0.637, indicating a medium positive relationship between the performance of the CNN on the classification task and the performance of the document embeddings later when compared against the favorites list. This was mentioned in section 6.11 on page 85 as a possible outcome of the very strong relationship between the F1 score and the NDCG-score. A very strong relationship between the F1- and the NDCG-score does not mean the document embeddings are necessarily bad, but they might struggle to perform *as well* on other tasks, as an evaluation task with strong ties to the classification task might be biased and give an inflated impression of the performance.

The Pearson's r for the second relationship was calculated to 0.467, also indicating a medium positive relationship between the performance of the CNN on the document similarity task and the performance of the document embeddings later when compared against the favorites list, although a somewhat weaker link than the first relationship. Again, this does not necessarily mean the document embeddings or the evaluation tasks are bad. One could argue that the document embeddings could be thought of as better if there was a strong relationship between the embedding's performance on the two evaluation tasks, meaning that when a model did well on one task it also did well on the other, and vica versa. But one could also argue that this indicates that the two tasks are so different that it could be unfair to compare them to each other so directly.

The most interesting aspect of this experiment is the triangular relationship between the three methods of estimating document embeddings: The favorites list-method, the RVS and the embeddings extracted from CNNs. This relationship is illustrated in figure 7.1. As stated above, NDCG and MAP are two entirely different measures, as are the two evaluation tasks. What this relationship show is perhaps the fragility of using the RVS as the gold standard. When both the CNN embeddings and the reference vectors achieve almost the same score when evaluated against the favorites list vectors, one would think when the CNN embeddings are evaluated against the reference vectors it would achieve a very good score, indicating that they are mostly the same. This does not happen, and it is uncertain what this could mean. When evaluating document vectors with the favorites list-method we look at the top 100 most similar documents to a target document according to the document vectors. We compare this top 100 against the content of the favorites lists, which can contain many more than 100 documents. It is thus possible for the CNN embeddings and the reference vectors to pick out *different* top 100 lists that both score roughly the same against the favorites lists, but against each other they match very little, as they have picket out different documents. This could explain why the favorites list-method gives the two document vectors roughly the same score, but when we pit them against each other the evaluation method tells us that the vectors have little in common. Even if the two top 100 lists were exactly the same, achieving the same score on the favorites list task, but opposites, where the top document in one is the bottom document in the other, they would still achieve a NDCG@100 of roughly 0.69 against each other. This is far away from the 0.127 the standard model achieves, meaning they must contain some number of different documents.

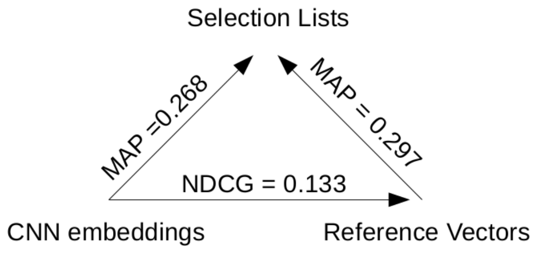

Figure 7.1: The triangular relationship between the three methods of estimating document embeddings for the ’tuned 100’ model.

Chapter 8

Conclusion

In this work I have detailed the compilation of a unique corpus of Norwegian court decisions. I have utilized this corpus to train several different machine learning models to produce semantic vectors for both words and documents. The document vectors were used for ranked document retrieval, and the performance of the vectors for this task was evaluated using a ranked retrieval model based on the document references in the documents, which was purposely built for this project. Finally, I have explored the interplay between pre-trained semantic word vectors and convolutional neural networks (CNN) and conducted several hyperparameter experiments using CNNs to produce document vectors.

In **chapter 2** I detailed the creation of the unique Lovdata Court Corpus (LCC), a document collection containing more than 130,000 Norwegian court decisions with extensive metadata. I provided an analysis of the document collection and the metadata, and in section 3.1 on page 40 I described the preprocessing which was applied. A lemmatized version of the corpus was primarily used in this thesis, but a corpus containing full forms was also included in some experiments to study the effect of lemmatizing the corpus. The LCC contains documents both in Norwegian Bokmål and Norwegian Nynorsk, but I preprocessed the documents using models specific to Bokmål documents, regardless of if the documents was in Nynorsk or any other language, such as Swedish or older versions of Norwegian. This is not ideal and will lead to poorer preprocessing for some documents, but I considered it out of scope for this project to detect the language used and act accordingly.

Furthermore, in section 2.4.2 on page 23 I created and assessed a ranked retrieval evaluation method for the LCC, the Reference Vector System (RVS). This resource did not exist for this corpus and was created solely to aid in the evaluation of different machine learning models for a ranked retrieval task. However, this evaluation method is not a perfect stand-in for a high-quality, human-made gold standard, but when the RVS was evaluated in section 2.4.2 on page 23 I showed that it did indeed pick up a signal and performed significantly better than randomly picking documents.

In **chapter 3** Both a standard Bag-of-Words model (BoW) and a Doc2Vec model was trained on both versions of the LCC and evaluated on the RVS. There was little difference between using a lemmatized or full form corpus, but BoW preferred full forms while Doc2Vec preferred lemmas. The BoW model trained on full forms performed best, achieving a NDCG@100 score of 0.1 and a Precision@100 of 0.138, meaning one out of seven documents were relevant to the query. The Doc2Vec models performed approximately three to four times worse than the BoW models. The performance of the models were disappointing, and due to the uncertainty around the quality of the RVS, I do not know if this was cased by shortcomings with the RVS, or the sheer difficulty of the ranked retrieval task.

In **chapter 6** I demonstrated the use of a CNN to produce document embeddings by training it on a metadata classification task. The performance on the ranked retrieval task for document embeddings were then evaluated using the RVS. An exhaustive search for the best combination of hyperparameters was out of the scope for this thesis, but several key hyperparameters were the subject of hyperparameter-tuning experiments which focused on adjusting only one hyperparameter. Finally, in section 6.10 on page 82, two combinations of the individually best performing hyperparameters were combined for a concluding experiment. One of these models, or rather, the document embeddings extracted from this model, had one of the worst performances compared to the other CNN experiments. This showed how volatile the combinations of hyperparameters are for a CNN for this task. It also demonstrated the onset of overfitting, which was present in many of the experiments, even when the amount of regularization in the network was the subject of experiments. The other model using a combination of the best hyperparameters set the best performance for the CNN experiments. It achieved a NDCG@100 of 0.133, which is a perfect 33% increase over the best BoW model. However, it is only a 4.7% increase over the baseline CNN experiment, which either used default hyperparameters or hyperparameters suggest in (Kim, 2014) and (Zhang & Wallace, 2015). I generally found the recommendations in (Kim, 2014) and (Zhang & Wallace, 2015) to also be valid for the experiments in this work, even though there is an enormous difference between the size of the documents used in this project and the size the recommendations were based on. Due to the closeness of the CNN classification task and the RVS, it is possible that the CNNs learned to detect and react to the references or a few local features in the documents. It is thus possible that a simple n-gram BoW model would perform equally good, but this was not tested.

In **chapter 4** I use the Word2Vec framework for estimating three different sets of semantic word embeddings. The sets were either trained on the LCC, the Norwegian News Corpus (NNC) or a combination of both. In section 4.1 on page 47, I evaluated these word embeddings on two benchmark data sets that enable intrinsic evaluation of distributional semantic models for Norwegian. I found that training on the NNC or the combination of the NNC and the LCC gave the best results. However, due to the generality of the tasks in the two benchmark data sets and the domain specific documents in the LCC, I found that the tasks were perhaps not fully able to quantify the added benefit of also including the LCC in the training. In **chapter 6**, I used two of the sets of word embeddings created as initializations for different CNNs in two separate experiments. While they did not improve on the performance of the baseline CNN, in **chapter 7**, I also performed the same intrinsic evaluation of the embeddings *after* the CNNs had finished training. This added tuning in the word embedding layer of a CNN did not lead to any significant improvements on the intrinsic evaluation tasks. A further analysis in section 7.2 on page 91 showed that the word embeddings had undergone little change while in the CNN.

8.1 Future work

In this final section I will reflect upon some of the potential use cases for the models created in this work and the results presented. I will also discuss some areas of the project where there is potential for improvement and different alternative approaches to some aspects of the work.

8.1.1 FastText and subword information

In section 2.6.7 on page 38 I briefly discussed FastText, a framework for learning word representations using subword information. In this subsection I will discuss how FastText can provide increased word embedding performance for Norwegian, and the potential gains from incorporating FastText into the systems from this thesis.

In section 2.6.7 on page 38 it was mentioned that FastText incorporates subword information into the Word2Vec CBOW or Skip-gram models discussed in section 2.6.5 on page 37 and 2.6.6 on page 38. The subword information is the character n-grams which make up the word. With this approach, words would be able to share representations, and out-of-vocabulary (OOV) words would be able to be represented either fully or partially by sharing n-grams with known words. This approach would not only reduce the impact of OOV words, but also strengthen the known word embeddings by more closely connecting them with similar words. In Norwegian, compound words are abundant and an integral part of our language. To further specify the meaning of a word we often add another word to it, like *sommerdag* and *vinterdag*, which is 'summer day' and 'winter day', respectively. Influence from, among others, the English language has increased the frequency of which compound words are wrongly split, and it has become so common for these mistakes to occur that we even have a word for it; *særskriving*, which literally mean "separate writing". The importance of not splitting Norwegian compound words is the fact that splitting the word can significantly alter its meaning. A good example of this is *røykfritt*, meaning 'free of smoke' or 'no smoking zone', and *røyk fritt*, meaning 'smoke freely' (Letnes, 2014).

There is thus great potential for FastText to take advantage of the subword information in Norwegian compound words to build stronger and better word representations. This does not only apply to 'normal' words, but also domain specific words like those found in legal texts. There are many compound words and concepts in the Lovdata Court Corpus (LCC) which could benefit from utilizing the subword information, for example *tingrett* (District Court), *lagmannsrett* (Court of Appeal) and *høyesterett* (Supreme Court) all end with *rett* (court), connecting them all as different court instances. Another example is *dommer* (judge), *lekdommer* (lay judge) and *høyesterettsdommer* (Supreme Court Judge), where the two latter judges are more specific variations of the first, unspecified judge.

(Stadsnes, 2018) showed that FastText produced Norwegian word embeddings which gave varied results, achieving a superior score to Word2Vec and GloVe on the analogies evaluation task, but inferior results on the synonym evaluation task. It is uncertain what kind of effect using FastText word embeddings to initialize the word embedding layer of a CNN would have, but seeing as the results of using pre-trained word embeddings as initialization did not provide better results than the standard model in 6.6 on page 68, it is doubtful that FastText would improve on this significantly. Like for many of the subjects in this work, there is not a lot of previous work using FastText on Norwegian texts, let alone legal documents, to build upon, and this is one of the main hurdles which led to FastText not being implemented in this project.

8.1.2 Triplet learning

One of the main issues faced when preparing the convolutional network to produce document embeddings was finding a classification task which would lead the network to learn useful document embeddings for our downstream document similarity task. Predicting a single or multiple types of metadata for a document seemed like one of the easiest and most straight forward tasks. However, much of the metadata for a document is probably not suited to be used for the classification task, as we want to 'trick' the network into making useful document embeddings as a by-product of this classification task. In addition, most of the useful and descriptive metadata was either rarely used or only used for a small subset of the documents. A subset of court decisions pertaining to different drug-related crimes have a great deal of relevant and descriptive metafields, which directly tie into the content of the document, but this subset was not large enough to be able to thoroughly train on, and if the network had been able to learn useful document representations, the representations would probably only be solid for documents from the same narrow subset of drug-related crimes, which was too small for the scope of this thesis.

During the work on this thesis, other approaches to the classification task were briefly experimented with, among them triplet learning (Wang et al., n.d.). Triplet learning, or triplet loss, is an intuitive proposal for a loss function. Instead of classifying pieces of metadata for a document, three separate documents, a triplet, are run through the network and the distance between the three document embeddings are computed. The idea is that one of the documents is an anchor document, another is a document related to the anchor document and the final document is unrelated to the anchor document. The goal for the network is to minimize the distance between the anchors and related documents, while maximizing the distance between the anchor documents and the unrelated documents, essentially learning to rank documents, albeit only two at a time. The loss is then calculated from how right or wrong the network is in it's prediction of which document is related and which is unrelated to the anchor. This loss is then used to train the network as per usual. One of the interesting properties of such a learning method is the fact that *we* must provide the network with two documents which we know for a fact are related and unrelated to the anchor, it is a supervised task after all. However, this task can be accomplished in many different ways and be different for every document. There is no need to keep to only a single piece of metadata for every document, we can mix and match them and combine them with other methods of ranking documents, as long as we end up with a quantifiable and consistent way of deciding on which document is related and unrelated. In fact, the more we combine and mix methods for finding the related and unrelated documents, the more nuanced the network will hopefully become. However, devising this ranking function in a consistent manner can be a challenge, and the right balance between related-ness and unrelated-ness of the two documents can be hard to find. If the unrelated document is too unrelated, the task becomes too easy and the network will learn to little, and the same will happen if the unrelated document is too close to the related document and the task becomes too difficult for the the network. But this factor of the learning process can be adjusted during the training by dynamically adjusting the difficulty of the triplet to always keep the difficulty of the task in the sweet spot or by requiring the network to distinguish between the documents beyond some margin which we can continually increase or decrease.

Due to time constraints it was decided not to pursue experimenting with triplet learning. On paper it offers some interesting benefits, but constructing a well functioning and consistent ranking function to draw the triplets in such a manner that the network will learn useful document representations were far too time consuming compared to the relatively easy classification of the legal area.

An easy implementation of a ranking function would be to use the Reference Vectors System (RVS) to pick a relevant and a non-relevant document. The downside to this approach could be that the CNN could possibly only learn to pick out the references in the documents to achieve a good performance during training, and when presented with documents not containing any references it would not be able to make a good prediction. If we continued to evaluate the document embeddings using the RVS we would probably not notice this, as the RVS requires there to be references in the documents. We would either need to find another way of evaluating the document embeddings which did not rely on references to be able to get an accurate idea of the actual performance, or get an unpleasant surprise when we tested the system on new data.

A weakness of the RVS in general is the fact that documents can reference the same laws or court decisions, but for different reasons, and as such the document embeddings from a network trained specifically to pick out references could be poor representations of the documents to use for document similarity tasks. This weakness also effects the reference vectors as a valid gold standard, but it was the lesser evil than some of the other methods of defining a gold standard. One might think the legal area tag is superior as a classification task in this way, as one can directly say that two documents which share a legal area tag must be somewhat similar. But alas, since the legal area is directly tied to the references in a document, it also does not differentiate between the context to as *why* the references were there.

8.1.3 Potential use of this work at Lovdata

This work was made possible by the author's access to Lovdata's document collection and systems. In this subsection I will explore some of the different uses within Lovdata for the systems developed in this project, as well as the groundwork it lays.

The most relevant system developed in this project for incorporating into Lovdata's production systems are of course a document retrieval system where users can use their own documents or paragraphs of text as search queries. When evaluated against the RVS the 'tuned 100' CNN model performed best, it is however unknown how the model would perform in a real life scenario with users that have complex information needs. One of the first steps in developing this system further is to expose it to real users and device a method for measuring the system's performance on real users. There are many methods of measuring how happy users are with a system, such as automatic logging of which documents of the retrieved documents are clicked on by the user or asking them to fill out a simple questionnaire. With the data gathered by this experiment we could get a better idea of the actual usefulness of the models, and depending on the extensiveness and duration of the experiment we could also devise a new and better gold standard than the RVS, which would in turn allow us to further evaluate new and existing models. The biggest room for improvements lies with performing many small improvements to get better data sets and evaluation measures, such as separating documents of different languages, inspecting and tweaking the preprocessing to remove tokens such as '321&nbsp;500' or treating references in the RVS as trees which can be pruned, like for the legal area tag.

The task of retrieving relevant documents could also be applicable in other areas, such as implementing a 'more like this'-function for court decisions. This would essentially be the exact same system and have the same functionality as the system proposed in the previous paragraph, but this system would not use unseen documents given by users as input, but rather documents from the existing LCC. The possible information gathered from interactions with such a system would enable us to for example establish a network linking documents together. This could in turn be useful for developing a better gold standard, like in the previous paragraph.

Another possible use for the document representations estimated in this work, and any future document representations, is predicting metainformation for documents using clustering techniques. Metainformation such as legal area, which is described in section 5.2.1 on page 54, is today either applied through an automated process or by manual effort. This could perhaps be applied automatically through clustering by using the most popular tag of the *n* nearest neighbors to a document. Studying these clusters along with their metainformation, such as legal area, keywords and publishing date, could also be beneficial for more academical or journalistic purposes, such as tracking the types of legal questions the Supreme Court has tackled over the years or if similar documents and cases end up with similar decisions.

Documents can also be classified as belonging to certain legal areas by using the CNNs trained in chapter 6. The classification task these networks trained on was classifying legal areas for documents, and as such we can easily incorporate them into the internal document workflow at Lovdata. We even have a solid gold standard for this task and can be confident in the reported performances in chapter 6. The best F1 score achieved in this chapter was 0.792, which is a good score. I have not experimented with weighting precision or recall, but it is likely that we could adjust the importance of precision or recall to suit our needs. We could either requiring a high precision so that the system would need less oversight and could be more or less autonomous, or a high recall so that the system has a larger coverage, but this would require more manual effort to double check the results.

The most important contribution from this project is the groundwork it lays for further work at Lovdata. Establishing relationships between documents is essential in many of the aspects of both the internal workflow and the functionalities offered to the users of the website, and the examples mentioned in the past paragraphs are just a few of the most prominent potential uses. The consequences of even implementing simple FastText word embeddings, which take a couple of hours to train, could bleed into several parts of the whole system, such as more efficient search using synonyms and subword information, keyword extraction, search query completion or named entity recognition. One of the goals of this thesis was to demonstrate the unexplored potential of machine learning and Norwegian legal documents. Although the structured documents used in this work are not possible to publicly release due to, among other things, their sensitivity, many of the concepts demonstrated in this work can be directly applied to a large law firm and their internal collection of legal documents. The examples explored in this section is just the tip of the iceberg of what is possible!

Bibliography

Boe, E. M. (2010). *Innføring i juss* (3rd ed.). Oslo: Universitetsforlaget.

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, *5*, 135–146.

Britz, D. (2015, december). *Implementing a cnn for text classification in tensorflow.* Retrieved 2018-10-29, from `http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/`

Evans, J. D. (1996). *Straightforward statistics for the behavioral sciences*. Pacific Grove: Brooks/Cole Pub. Co.

Goldberg, Y. (2015). A primer on neural network models for natural language processing. *CoRR*, *abs/1510.00726*.

Goldberg, Y., & Hirst, G. (2017). *Neural network methods in natural language processing*. San Rafael, California: Morgan & Claypool Publishers.

Hong, J., & Fang, M. (2015). *Sentiment analysis with deeply learned distributed representations of variable length texts* (Tech. Rep.). Technical report, Stanford University.

Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2017). Bag of tricks for efficient text classification. In *Proceedings of the 2017 conference of the european chapter of the association for computational linguistics.* Valencia, Spain.

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 conference on empirical methods in natural language processing, EMNLP.* Doha, Qatar.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th international conference on neural information processing systems - volume 1* (pp. 1097–1105). Lake Tahoe, Nevada.

Letnes, A. M. (2014). *"dessverre har jeg en slags problemmer med dysleksi" : the effects of dyslexia on the acquisition of norwegian as a second language in adults, as seen in written texts from test of norwegian – advanced level* (Master's thesis, NTNU, Trondheim). Retrieved from `https://brage.bibsys.no/xmlui/handle/11250/299551`

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval*. Cambridge, UK: Cambridge University Press.

Marsland, S. (2014). *Machine learning: An algorithmic perspective* (2nd ed.). London, UK: Chapman & Hall/CRC.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. In *Iclr workshop papers.* Scottsdale, Arizona.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111–3119). Lake Tahoe, Nevada.

Øvrelid, L., & Hohle, P. (2016). Universal dependencies for norwegian. In *Proceedings of the tenth international conference on language resources and evaluation LREC 2106* (pp. 1579–1585). Portorož, Slovenia.

Paliwala, A. (2010). *A history of legal informatics*. Zaragoza, Spain: Prensas de la Universidad de Zaragoza.

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (emnlp)* (pp. 1532–1543). Doha, Qatar.

Robberstad, A. (2009). *Sivilprosess* (2nd ed.). Fagbokforlaget.

Singhal, A. (2001). Modern information retrieval: a brief overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, *24*(4), 35–42.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(1), 1929-1958.

Stadsnes, C. (2018). *Evaluating semantic vectors for norwegian* (2018, University of Oslo, Oslo). Retrieved from `https://www.duo.uio.no/handle/10852/61756`

Stadsnes, C., Øvrelid, L., & Velldal, E. (2018). Evaluating semantic vectors for norwegian. In *Norsk informatikkonferanse 2018.* Longyearbyen, Svalbard.

Velldal, E., Øvrelid, L., & Hohle, P. (2017). Joint ud parsing of norwegian bokmål and nynorsk. In *Proceedings of the 21st nordic conference of computational linguistics* (pp. 1–10). Gothenburg, Sweden.

Wang, J., Song, Y., Leung, T., Rosenberg, C., Wang, J., Philbin, J., … Wu, Y. (n.d.). Learning fine-grained image similarity with deep ranking. In *Proceedings of the ieee conference on computer vision and pattern recognition.* Columbus, Ohio.

Webber, W. (2010). *Measurement in information retrieval evaluation* (Doctoral dissertation, University of Melbourne, Australia). Retrieved from `http://hdl.handle.net/11343/35779`

Zhang, Y., & Wallace, B. C. (2015). A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *CoRR, abs/1510.03820.*