LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

CENTRUM FÜR INFORMATIONS- UND SPRACHVERARBEITUNG
STUDIENGANG COMPUTERLINGUISTIK

Bachelorarbeit

im Studiengang B.Sc. Computerlinguistik

an der Ludwig- Maximilians- Universität München

Fakultät für Sprach- und Literaturwissenschaften

Unsupervised Learning Of Joint Neural Embedding Spaces for Queries and Documents

vorgelegt von
Joseph M. Birkner

Aufgabensteller:	Dr. Maximilian Hadersbeck
Betreuer:	Prof. Dr. François Bry, M.Sc. Yingding Wang
Bearbeitungszeitraum:	27. September - 4. Dezember 2017

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig angefertigt, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

München, den 4. Dezember 2017

....

Joseph M. Birkner

"All sorts of computer errors are now turning up. You'd be surprised to know the number of doctors who claim they are treating pregnant men."

— Isaac Asimov

Contents

Abstract

Students of the 21st century should have ubiquitous access to educational resources which empower them to learn any skill as effectively as possible, and on-demand. Massive Open Online Courses (MOOCs) naturally contribute to this vision, but finding the right course often presents a challenge for prospective students. Therefore, this work aims to design, develop and deploy an intelligent MOOCs search and recommendation engine. The project is therefore anchored in the field of (Neural) Ad-hoc Information Retrieval (*Neu-IR*).

Essential to any good IR system is an optimum encoding function $\Phi(x)$ which generates useful representations $\Phi(D)/\Phi(Q)$ of user (student) queries Q and documents (MOOC summaries) D. Such representations can then be used to calculate the relevance $R(\Phi(d), \Phi(q))$ of a document d with respect to a user query q. Traditionally, the encoding function $\Phi(x)$ returns a weighted term frequency vector over all given terms in a document or query. The loss of term order and extreme dimensionality are inherent problems to the frequency-distribution derived representation.

As part of this bachelor thesis, a variety of neural-network derived encoding functions $\Phi_{NN}(x)$ are explored, implemented and evaluated. In particular, the Weighted Average Word Embedding Representation $\Phi_{\text{avgemb}}(x)$ is used as a baseline. This baseline serves to evaluate experimental, neural representation models which are derived from unsupervised Sequence-to-Sequence (*'Seq2Seq'*) Autoencoder-LSTMs, trained on a corpus of Wikipedia paragraphs. It is hypothesized that such neural representation generators are superior to weighted term frequency distributions, because they overcome the modeling shortcomings (Loss of word order and extreme dimensionality) of traditional IR models.

Furthermore, this work aims to develop *one* unsupervised neural-network derived encoding function $\Phi_{NN}(x)$ which can be used both for encoding queries and text documents. The relative step-wise RNN embedding-difference $\texttt{RNN}_{\Delta E}$ is proposed as a measure for the influence of a word w on the representation vector of a document D_w. It is shown that $\texttt{RNN}_{\Delta E}$ strongly correlates with IDF, and is more helpful than IDF for assessing the contextual relevance of tokens. To synthesize queries from documents for unsupervised training of the encoder function $\Phi_{NN}(x)$, this work proposes the *Token-Relevance-To-Queries (ToRQue)* algorithm.

The latent vector spaces from $\Phi_{NN}(x)$ and $\Phi_{\text{avgemb}}(x)$ are evaluated on *document triplet semantic similarity judgments* and *Normalized Discounted Cumulative Gain (NDCG)* over the IROM corpus of $(\texttt{title}, \texttt{topic}, \texttt{text})$ tuples of MOOC offers. On both benchmarks, negative results are reported with the models trained by this work. However, training evaluation indicates that these results are a consequence of the selected loss function (cosine distance prediction loss). Further investigation of unsupervised embedding learning with the *ToRQue* algorithm in conjunction with word or character-level cross-entropy loss is recommended. Surprisingly, it is found that $\texttt{RNN}_{\Delta E} \times log(TF) \times IDF$ as a word vector weight for the Average Word Embedding Representation outperforms both the $log(TF) \times IDF$ and $TF \times IDF$ baseline weights on the benchmark tasks.

Zusammenfassung

Schüler und Stundenten des 21. Jahrhunderts sollten allgegenwärtigen, wahlfreien Zugriff auf für sie maßgeschneiderte Bildungsangebote haben. So genannte *Massive Open Online Courses* (MOOCs), d.h. über das Internet angebotene Kurse, sind ein natürlicher Bestandteil dieser Vision. Genau *den* richtigen Kurs zu finden, kann jedoch für einen Kurssuchenden eine große Herausforderung darstellen. Ziel dieser Arbeit ist daher die Entwicklung und Bereitstellung eines Such- und Empfehlungssystems für Onlinekurse. Deshalb ist dieses Projekt im Feld des '(Neural) Ad-hoc Information Retrieval (*Neu-IR*)' verankert.

Ein essentieller Bestandteil jedes guten IR-systems ist eine optimale Abbildung $\Phi(x)$, die mächtige Repräsentationen $\Phi(D)/\Phi(Q)$ von benutzergenerierten Suchanfragen Q, sowie Dokumenten (MOOC-Textbeschreibungen) D generieren kann. Solche Repräsentationen können später benutzt werden, um den Relevanz-Score $R(\Phi(d), \Phi(q))$ eines Kurses d im Bezug auf eine Suchanfrage q zu berechnen. Traditionell ist $\Phi(x)$ eine Funktion, die eine gewichtete Frequenzverteilung über allen Termen in einem Dokument/einer Suchanfrage als Repräsentation generiert. Allerdings sind der Verlust jeglicher Wortreihenfolgen-Informationen, sowie extrem hohe Dimensionalität, inhärente Probleme von Termfrequenzverteilungs-abgeleiteten Repräsentationen.

Im Rahmen dieser Bachelorarbeit soll eine Auswahl an Neuronalen-Netzwerk-abgeleiteten $\Phi_{NN}(x)$ erklärt, implementiert und evaluiert werden. Insbesondere wird der gewichtete Word-Embedding Durchschnitt $\Phi_{\text{avgemb}}(x)$ als Vergleich (Baseline) herangezogen. Diese Baseline dient zur Einschätzung von unüberwachten Sequenz-zu-Sequenz (*'Seq2Seq'*) Autoencoder-LSTMs, die mit einem Korpus von Wikipedia-Absätzen trainiert werden. Es wird die Hypothese aufgestellt, dass solche neuronalen Repräsentationen den traditionellen gewichteten Termfrequenzverteilungen als Dokumentrepräsentationen überlegen sind, da sie deren inhärente Probleme (Verlust der Wortreihenfolgen-Informationen und zu hohe Dimensionalität) nicht teilen.

Weiterhin ist es Ziel dieser Arbeit, *eine* Repräsentationsfunktion $\Phi_{NN}(x)$ zu entwickeln, die sowohl zur Enkodierung von Suchanfragen, als auch zur Enkodierung von Textdokumenten verwendet werden kann. Hierfür wird die schrittweise relative RNN-Zustandsdifferenz $\texttt{RNN}_{\Delta E}$ als ein Maß zur Bestimmung des Einflusses eines Wortes w auf den Repräsentationsvektor des Dokumentes D_w prsëntiert. Es wird gezeigt, dass $\texttt{RNN}_{\Delta E}$ mit dem IDF-Maß stark korreliert, und dieses an Aussagefähigkeit teilweise übertrifft. Zur Synthetisierung von Suchanfragen aus Dokumenten für unüberwachtes Lernen der Enkodierungsfunktion $\Phi_{NN}(q)$ schlägt diese Arbeit den *Token-Relevance-To-Queries (ToRQue)* Algorithmus vor.

Die Vektorräume der Enkodierungsfunktionen $\Phi_{NN}(x)$ und $\Phi_{\text{avgemb}}(x)$ werden anhand von *Semantischen Dokument-Triplet Ähnlichkeitsentscheidungen* und *Normiertem Discounted Cumulative Gain (NDCG)* über dem IROM-Korpus von $(\texttt{Titel}, \texttt{Thema}, \texttt{Text})$-Tupeln von MOOC-Angeboten evaluiert. In beiden Maßen gelingt es den Seq2Seq Modellen aus dieser Arbeit nicht, die $\Phi_{\text{avgemb}}(x)$-Baseline zu übertreffen. Allerdings wird nach Analyse des Trainingsprozesses der Schluss gezogen, dass die Leistung der Modelle eine Konsequenz der gewählten Dekodier-Fehlerfunktion ist (Kosinusähnlichkeit zwischen den Goldstandard-Wortvektoren und den vorhergesagten Wortvektoren). Eine weitere Evaluierung des ToRQue-Algorithmus unter Hinzunahme alternativer Dekodier-Fehlerfunktionen (Cross-Entropie auf Wort- oder Zeichenbasis) wird empfohlen. Überraschenderweise wird festgestellt, dass $\texttt{RNN}_{\Delta E} \times log(TF) \times IDF$ als Wort-Vektor-Gewicht für den Gewichteten Word-Embedding Durchschnitt als Dokumentrepräsentation die Baseline-Gewichtungen $log(TF) \times IDF$ und $TF \times IDF$ in beiden Evaluationsmaßen übertrifft.

Acknowledgments

First, I would like to thank **Prof. Dr. François Bry** at the Institute of Informatics, Teaching and Research Unit of Programing- and Modelling Languages (PMS) for his invaluable academic guidance and advice, inexhaustibly encouraging me to think outside the box and broaden the scope of my observations.

The support from **Dr. Maximilian Hadersbeck**, program coordinator at the Center for Information- and Language Processing (CIS), was undoubtedly essential for the success of this work, as it enabled the inter-institutional cooperation that sparked this work's incubation.

Most importantly, **M.Sc. Yingding Wang** tirelessly accompanied me throughout my long fractal learning curve, from aspiring English poet to prospective scientific writer. During our numerous sessions of inspired brooding over Neural Network designs, I had a contemptuous amount of fun, which immeasurably motivated me to maintain my endeavoring stride.

"People who think they know everything are a great annoyance to those of us who do."

— Isaac Asimov

1. Introduction

1.1. Vision

Ad-hoc access to education, for everyone

Students of the 21st century should have ubiquitous access to educational resources which empower them to learn any skill as effectively as possible, and on-demand. Massive Open Online Courses (MOOCs) naturally contribute to this vision, but finding the right course often presents a challenge for prospective students.

This work is part of the project *IROM* which is an acronym for *Intelligent Recommendation of Massive Open Online Courses*. The project IROM envisions the design, development and deployment of an intelligent MOOCs search and recommendation engine. Such a system would have the ability to make course recommendations to users based on a deep understanding of course contents and user needs, both from explicit (query string) and implicit (user profile) contextual signals. The vision for such an IR system is to combine the quantitative performance of a software system with the introspective ingenuity of a human advisor.

Project IROM is generally anchored in the field of (Neural) Ad-hoc Information Retrieval (*Neu-IR*), a rather novel sub-field of Information Retrieval [1–3].

Contributing to a landscape of expert recommendation/search systems

Narrow, vertical search engines are designed to fit very specific characteristics of a certain type of retrieval unit, rather than an abstract, broad model of many different types of retrieval units simultaneously. The set of such vertical search engines is a commonly accepted alternative to abstract Web search engines for many types of retrieval units (such as audio-books, academic papers, videos, music, movies, etc.).

IROM as a search engine for MOOCs fits neatly into the landscape of vertical search engines, specifically optimized to calculate relevance scores with respect to the intrinsic characteristics of MOOCs as retrieval units. An illustration of this process can be seen in figure 1.1.

1.2. Motivation

As illustrated in figure 1.1, an optimum encoding function $\Phi(x)$ which generates useful representations $\Phi(D)/\Phi(Q)$ of user (student) queries Q and documents (MOOC textual descriptions) D, is essential to the functionality of any good IR system. Such representations can then be used to calculate the relevance $R(\Phi(d), \Phi(q))$ of a course d with respect to a user query q. Traditionally, the encoding function $\Phi_{tf}(x)$ returns a weighted term frequency vector over all given terms in a document or query (TF-IDF/Okapi BM25) [4, 5]. This carries some inherent problems with it. The following two sentences which vastly differ in their semantics, illustrate these problems:

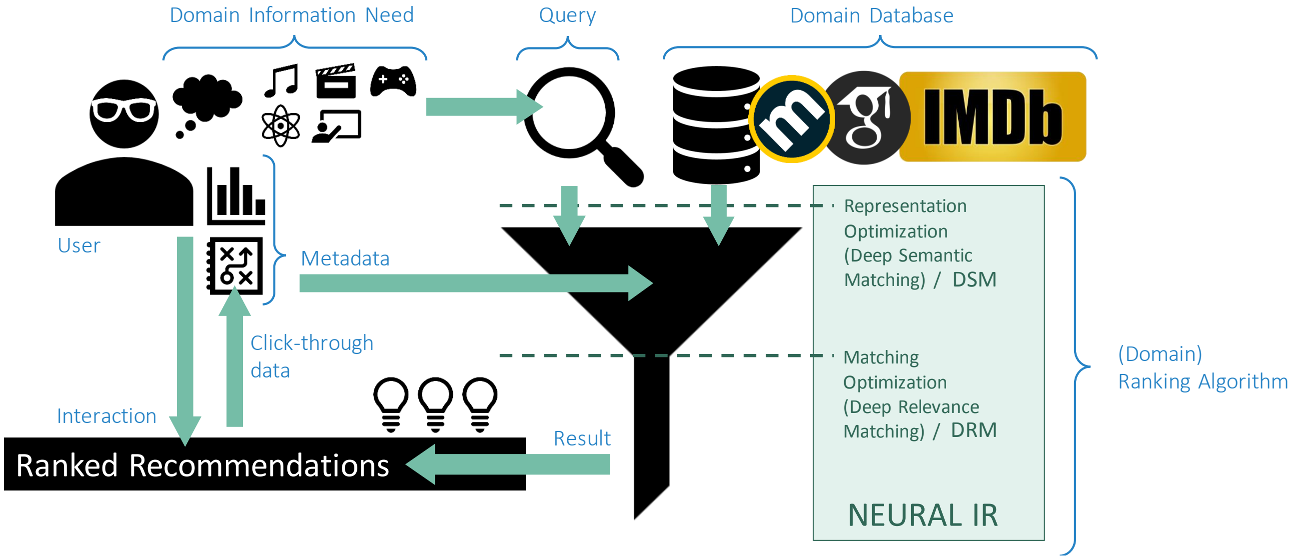


Figure 1.1.: Overview of a neural vertical search engine (Graphic produced by this work).

```
[Introduction Into Programming]
D1:  This course is a general introduction into programming.

[Medieval History]
D2:  We programmed this course as a general introduction.
```

After the usual normalization steps, these sentences would result in the same unordered set of terms, yielding exactly the same document vectors:

$$\Phi_{tf}(D_1) = \Phi_{tf}(D_2) = \{\text{course}, \text{general}, \text{introduction}, \text{program}\}$$

In conclusion, as representations of a document's content, term-frequency derived document vectors carry the following shortcomings:

1. Loss of word order: Any semantics based on word order are lost in the document vector. However, word order can dramatically alter the semantics of a sentence (as shown above).

2. Misleading term independence assumption: The term frequency vector assumes that each term models an orthogonal (independent) dimension of meaning. With respect to synonyms (two words share the same meaning) and homonyms (one word has multiple meanings), this assumption is obviously flawed.

These shortcomings harm the correctness of the relevance assessment between a given document-query pair. In the special case of the project IROM (and as per the example above), this might lead to the suggestion of a *'Medieval History'* course as an answer to a user's search query for *'programming'*. It is therefore desirable to obtain document representations for IR that entail a deeper understanding of a documents content.

1.3. Objective

To generate better document representations, this work assumes that an optimum representation $\Phi(D)$ of a document D should be useful to accurately predict D's textual content. Extracting features to achieve this representation can therefore be modeled as a machine learning task, more specifically a so-called auto-encoding task:

Definition 1 (Definition of an Autoencoder [6])**.** *Let $AE_t = \{\Phi_{AE_t}, \Phi^{-1}_{AE_t}\}$ be a model consisting of an encoder $\Phi_{AE_t} : D_i \to \mathbb{R}^k$ and a decoder $\Phi^{-1}_{AE_t} : \mathbb{R}^k \to D_{reconstructed}$ at optimization step t: At the next optimization step $t+1$, an optimization algorithm over a training corpus of documents $\{D_1, ..., D_n\}$ trains AE_{t+1} per gradient descent over a loss function $\mathscr{L} : D_i, AE_t \to \mathbb{R}$. This loss function determines the difference between the original document D_i and the reconstruction $D_{reconstructed} = \Phi^{-1}_{AE_t}(\Phi_{AE_t}(D_i))$ given by the decoder. If the cumulative loss $\sum_{i=1}^{n} \mathscr{L}(D_i, AE_t)$ reaches zero, the auto-encoder AE_t has learned an optimum encoding function Φ_{AE_t} for the documents $\{D_i\}$.*

The sequential, varying-length nature of textual data suggests the implementation of a recurrent neural network (RNN), or more specifically a Long Short-Term Memory (LSTM) for the encoder and decoder, respectively. Such encoder-decoder architectures for sequences are called *Seq2Seq* or *Sequence-to-Sequence* models [7–9].

In the relatively new field of Neural IR (*Neu-IR*), the *Seq2Seq* approach to generate document representations falls under *Deep Semantic Matching (DSM)*, one of two main categories [1–3]. The other category of approaches in Neu-IR is summarized under the term *Deep Relevance Matching (DRM)*. DSM models are optimizing the representation function $\Phi(D)$, whereas DRM models optimize the relevance function $R(\Phi(D), \Phi(Q))$.

This bachelor thesis applies *Seq2Seq* to extract feature vectors from MOOC textual descriptions, to overcome the shortcomings (Loss of word order and misleading term independence assumption) in the traditional IR models.

1.4. Tasks

First, a Sequence-to-Sequence-Autoencoder Φ_{AE} shall be trained in TensorFlow [10] to learn fixed-k-length feature vectors $\Phi(D), \Phi(Q)$ for document and query representations. Pre-trained Word2Vec [11] or GloVe [12] embeddings shall be used in place of one-hot encoded terms, to simplify the abstraction task of the neural network. A Variational Recurrent Autoencoder should be similarly trained, to potentially defeat the *Curse of Dimensionality* [13] that is inherent to high-dimensional embedding spaces. The embedding vector length k should be treated as a hyperparameter, and all embedding models should be trained with $k \in \{8, 64, 256\}$. The embedding models will be trained from a random selection of English Wikipedia paragraphs. Therefore, the model search space contains $|\{\texttt{RNN}, \texttt{VRNN}\}| * |\{k_i | 1 \leq i \leq 3\}| = 6$ models.

Second, technologies should be explored, designed and implemented to jointly embed queries and documents into a shared embedding space, in an unsupervised fashion. For this purpose, this work proposes a novel keyphrase-extraction process called *Token-Relevance-to-Queries (ToRQue)* which uses the relative step-wise RNN embedding-difference $\texttt{RNN}_{\Delta E}$ as a measure of phrase importance to create artificial search phrases. $\texttt{RNN}_{\Delta E}$ shall be evaluated as a measure of term importance, and compared against TF-IDF under the aspects of correlation and quality.

Third, given a collection of documents $C = \{D_1, ..., D_n\}$, and a user query Q, a ranking function shall be implemented by ordering the documents by their cosine similarities $cos(\Phi(D_i), \Phi(Q))$. The ranking/encoding modules should be built into a REST-service, such that they can be used as a back-end for the IROM search engine.

Fourth, all trained embedding models shall be evaluated against a TF-IDF and $\texttt{RNN}_{\Delta E}$ weighted average word embedding model [14], on the metrics of (1) ranked Ad-hoc retrieval *Normalized Discounted Cumulative Gain (NDCG)* [15–17] and (2) document triplet similarity accuracy [18]. Both measures will be evaluated over a selection of a corpus of title-labeled textual course descriptions crawled by project IROM, with cluster labels adapted from previous work in the project [19].

1.5. Related Work

Information Retrieval (IR) as a field has seen a lot of research since roughly the 1950s, when IR was primarily occupied with large-scale Boolean retrieval architectures [20–29]. With the dimensionality problem of one-hot term representations widely apparent, more sophisticated algebraic/probabilistic document/query representation/ranking models were developed: *Latent Semantic Analysis (LSA)* [30], *Latent Dirichlet Allocation (LDA)* [31], the *Probabilistic Ranking Principle (PRP)* [4, 5, 32], and finally *Probabilistic Language Models (LM)* [33–35]. All of these more traditional approaches towards modeling documents, queries and relevance scores are wonderfully described and summarized in the book *'Introduction to Information Retrieval'* [36]. For evaluation of a ranked list of retrieved documents with regard to a user query, this work uses the *Normalized Discounted Cumulative Gain (NDCG)* measure [15–17].

Then not yet with a use-case for Information Retrieval in mind, but nonetheless useful for it today, were the first encounters of Neural Networks with NLP, in particular word representation modeling. Building upon a legacy of research by Yann Le-Cun and Geoffrey Hinton into *Multi-Layer Perceptrons*, *Neural Networks* (NNs), *Backpropagation-of-Error* [37, 38], *Autoencoders* [6], and much earlier insights into distributional semantics in Natural Language Processing (NLP) [39], advances in computer hardware enabled the use of *Feed-Forward Neural Networks* (FFNNs) for rich word representation (embedding) modeling, like *Word2Vec* [11] or *GloVe* [12]. These word embeddings rapidly found their way into IR, in the form of the average word embedding representation for documents, or *Distributed Bag-of-Embedded-Words (DBoEW)* [14] which this work uses as a baseline.

Given the success of these neural word embedding models, a logical next step was to apply these approaches to sentences and documents. However FFNNs are not practicable as function approximators for this task, because they can only operate on fixed-length inputs, not on sequences. The type of NN required to process sequential textual data is a *Recurrent Neural Network (RNN)* [40, 41], more specifically *Long Short-Term Memory (LSTM)* networks [42, 43]. Mikolov's team built upon Word2Vec with the *Doc2Vec* system [44], while other teams developed competing sentence/paragraph/document representation systems: *Sequence-to-Sequence (Seq2Seq)* [7, 8], mainly for Neural Machine Translation, and Paragraph Vectors [18]. Furthermore, the *Curse of Dimensionality* [13] encouraged the combination of LSTMs with *Variational Autoencoders* [45], resulting in *Variational RNNs (VRNNs)* [46] which tend to fill their latent embedding space more smoothly [9, 47, 48].

As noted by Christopher Manning and implied by his SIGIR 2016 contribution [49, 50], neural representation models are now bringing IR and NLP much closer together than historically understood. The underlying idea is that an NLP-inspired semantically aware IR system might enable much more intelligent rankings than previously seen. This new, neural approach for IR is called *Neu-IR*, and it has attracted a figurative explosion of research with *Deep Structured Semantic Matching (DSSM)* [51], *Architecture-I/II (ARC-I/II)* [52], *MatchPyramid* [53], *Paragraph Vectors for IR* [54], the *Deep Relevance Matching Model (DRMM)* [55], *Deep Sentence Embeddings from RNNs* [56], *MatchTensor* [57], and the *Neural Vector Space model (NVSM)* [58], very recently. Luckily though, there are some excellent overview papers which summarize the state of the field [1–3, 59].

Unsupervised keyphrase extraction which this work uses for unsupervised training of an embedding generator for both queries and documents, is a research topic with established statistical approaches, most notably the TextRank algorithm [60]. However, keyphrase extraction with RNNs is less explored. Proposals include keyphrase generation with Autoencoders (CopyRNN) [61].

Standing on the shoulders of giants, this work would have severely suffered in quality, if it had not been for some deeper research into the properties, behavior and implementation of Neural Networks [10, 41, 62–68].

1.6. Notational Agreements

To simplify the mathematical formulation of matrices, vectors and tuples, this work uses some short-hand notations, as seen in table 1.1.

Notation	Meaning
$M \in \mathbb{R}^{m \times n}$	Declare a matrix M with m rows and n columns.
$\overrightarrow{c}_i \in M \in \mathbb{R}^{m \times n}$	The i-th row vector of matrix M.
$s_{ij} \in M \in \mathbb{R}^{m \times n}$	The scalar component in column j of row i of matrix M.
$s_i \in \overrightarrow{v} \in \mathbb{R}^n$	The i-th scalar component of a vector of rank n.
$\overrightarrow{v} = \langle f(i) \rangle_{i=1}^{n}$	Construct a vector of rank n, where each component is the value of a function $f(i)$ of the component index i.
$(x_i)_{i=1}^{n}$	Declare a tuple with n values

Table 1.1.: Shorthand notations used by this work.

2. Traditional Information Retrieval

2.1. Terminology

The term *Information Retrieval* was established through an article by Calvin Mooers in 1950 [20]. One of the main references for this chapter is the book *Introduction to Information Retrieval* by *Christopher D. Manning*, *Prabhakar Raghavan* and *Hinrich Schütze* in its first (2008) edition [36]. The book offers the following definition of *Information Retrieval*:

> *Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers). [36, p. 1]*

In accordance with this definition, this chapter focuses on Information Retrieval over text documents. Hereby, the text document assumes the role of the so-called *retrieval unit* within the IR system. Obviously, Information Retrieval systems support a wide range of retrieval units; for example images, videos or music. Due to the radically different data formats behind these different retrieval units, the underlying retrieval algorithms also differ quite radically. When retrieving text documents, the common understanding is that the process adheres to the principles of *Ad-hoc retrieval*: An *information need* by a (usually human) user is formalized as a *query* and posted to the Ad-hoc Retrieval system. Information is observed as part of a document D which itself is part of a collection or *"corpus"* $C = \{D_1, ..., D_n\}$. The predicted degree of overlap between the information contained in a document and the posted user query is also called *relevance score*. The relevance score S_i of a document $D_i \in C$ in the context of a user query Q is determined by a model $M : D_i, Q \to S_i$. The descending order of documents D_i by their relevance score $S_i = M(D_i, Q)$ in the context of a user query Q is also called *Ranking*.

Furthermore, it should be defined what a *Term* is in IR: Terms constitute a set of equivalence classes over token types. A *"token"* in a text may be approximately synonymous to a *"word"*, although it may also consist of multiple or partial words: *New York* as a common name is a single token, while *aren't* can be divided into *are* and *not*. A token is always a concrete occurrence of a string in a text (also called *incidence*). On the other hand, a *token type* is the set of all incidences of the same token string. The equivalence relation $t_1 R_{\text{TERM}} t_2$ between two token types t_1 and t_2 is differently defined across languages and IR systems, but most commonly boils down to the following formula:

Equation 1 (Equivalence of the term relation)**.**

$$t_1 R_{TERM} t_2 \leftrightarrow normalize(t_1) = normalize(t_2)$$

Therefore, t_1 and t_2 belong to the same term equivalence class, if they match in their normalized form. Normalization may contain steps such as spelling correction, lower case conversion, stoppword-filtering and lemmatization [36, p. 26-28].

In this work, it is assumed that all documents in a corpus occur as sequences of term references. For example, consider the following 'document' D_1:

$Text_{D_1}$ =*Noam Chomsky said that little green ideas sleep furiously in the green grass.*

Then a fictional retrieval system is expected to extract and normalize the tokens, create term equivalence classes $T_{1 \leq i \leq N}$, and succeedingly converts the document into a sequence like $[T_i]$. For example, the following equivalence classes could be extracted from the sample document D_1:

$$\begin{aligned} T_1 &= [\text{noam chomsky}] \\ T_2 &= [\text{say}] \\ T_3 &= [\text{little}] \\ T_4 &= [\text{green}] \\ T_5 &= [\text{idea}] \\ T_6 &= [\text{sleep}] \\ T_7 &= [\text{furious}] \\ T_8 &= [\text{grass}] \end{aligned}$$

The internal representation of D_1 in the aforementioned fictional retrieval system would then look as follows:

$$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_5 \rightarrow T_6 \rightarrow T_7 \rightarrow T_4 \rightarrow T_8$$

Another simplification now constitutes the foundation for many traditional and established retrieval models: The internal representation completely discards any information about specific word order, and models each document exclusively as a frequency distribution over all terms $T_{1 \leq i \leq N}$ that occurred in the document. If the domain of the per-term frequency is limited to $\{0,1\}$, the representation is also called the *Bag-Of-Words* representation [39]:

$$FD(D_1) = \langle 1,1,1,2,1,1,1,1 \rangle$$

This representation allows for very efficient and effective retrieval models.

2.2. The Boolean Retrieval Model

2.2.1. Modeling IR within a Boolean Algebra

In 1847 George Boole established the theoretical foundations for the *Boolean Algebra* [69] which operates over the domain $\mathbb{B} = \{\text{TRUE}, \text{FALSE}\}$. The value TRUE is commonly associated with 1, FALSE with 0. Let A and B be expressions in the Boolean Algebra $\mathscr{B}$, then the following expressions are also valid expressions in $\mathscr{B}$:

- **And** (Conjunction): $A \wedge B := \min(A,B)$
- **Or** (Disjunction): $A \vee B := \max(A,B)$
- **Not** (Negation): $\neg A := \begin{cases} 0 : A = 1 \\ 1 : A = 0 \end{cases}$

If $\mathscr{B}$ is extended by a function *contains* $: D,T \rightarrow \mathbb{B}$ which evaluates to TRUE if the term T is contained in document D and FALSE otherwise, then queries $Q_{\mathscr{B}}$ may be expressed in $\mathscr{B}$.

For example, a user may be looking for Asimov's *Foundation* within a corpus of science-fiction books. Unfortunately, neither does she remember the name of the author, nor the book. Maybe she would express her information need as follows:

$$Q_{\mathscr{B}}(D) = contains(D, [\texttt{knowledge}]) \wedge contains(D, [\texttt{galaxy}]) \wedge contains(D, [\texttt{earth}])$$

A Boolean retrieval model $M_{\mathscr{B}} : D, Q_{\mathscr{B}}(D) \rightarrow \mathbb{B}$ could determine documents that match with the above stated query, where I_D is a document identifier:

Definition 2 (Simple Boolean Retrieval Model over the Science Fiction Corpus)**.**

$$Result(Q_{\mathscr{B}}(D)) = \{(I_D, Q_{\mathscr{B}}(D)) | D \in ScienceFictionCorpus\}$$

The model generates a set of pairs, each consisting of a document identifier and the relevance score calculated by $M_{\mathscr{B}}$. The exact computations that are performed by the *contains* function are not yet defined though. At this point, it may be useful to have a second look at the *Bag-Of-Words* representation of documents that was defined in the previous section: This representation already holds the Boolean value of the *contains*-function for every term! The set of bag-of-words representations results in a matrix $M \in \mathbb{B}^{|C| \times N}$, where each column represents a term T_i, and every row a document D_j. The value of M_{ji} is accordingly TRUE if D_j contains T_i, otherwise FALSE. A transposed version of this matrix is called *Incidence Matrix*. A part of this matrix for the previously mentioned *ScienceFictionCorpus* could look as follows:

$$M^{\mathscr{T}}_{ScienceFictionCorpus} = \left\{ \begin{array}{rccccc} & \dots & D_{\text{Foundation}} & D_{\text{StarWars}} & \dots \\ \dots & \dots & \dots & \dots & \dots \\ [\text{galaxy}] & \dots & 1 & 1 & \dots \\ [\text{knowledge}] & \dots & 1 & 1 & \dots \\ [\text{earth}] & \dots & 1 & 0 & \dots \\ [\text{light saber}] & \dots & 0 & 1 & \dots \\ \dots & \dots & \dots & \dots & \dots \end{array} \right.$$

Figure 2.1.: Example of an incidence matrix

Two observations can be made:

1. This matrix should usually be very sparsely populated, such that most entries will be 0: Many terms are going to be unique in their works, especially considering names, dates and numbers. For example, a medium-sized corpus may contain 100,000 documents and 500,000 terms. The incidence matrix of this corpus would be $\frac{100{,}000 * 500{,}000}{8{,}000{,}000{,}000} = 6.25\text{GB}$ in size, most of which is wasted on storing 0-values.

2. Instead of iterating over all required terms for each document, it is possible to replace the operators $A \wedge B$, $A \vee B$ and $\neg A$ with the set operators $A \cap B$, $A \cup B$, and $\overline{A}$ in any Boolean search query, if at the same time the *contains* function is replaced with a function like $DocumentsWithTerm : M^{\mathscr{T}}, T_i \rightarrow Result, DocumentsWithTerm(M^{\mathscr{T}}, T_i) = \{D_j \in C | M^{\mathscr{T}}_{ij} = 1\}$. Therefore, the domain of $\mathscr{B}$ would change from $\{0, 1\}$ to $\mathscr{P}(C)$, meaning the power set of the corpus. The most important thing to note here is that within this alternate Boolean Algebra, $\emptyset \in \mathscr{P}(C)$ is equivalent to FALSE and $P \in \mathscr{P}(C), P \neq \emptyset$ is equivalent to TRUE.

The latter observation results in what is commonly called an *Inverted Index*: Instead of keeping a set of contained terms for every document, a set of containing documents is kept for every term.

2.2.2. Efficient evaluation of Boolean search queries

The previous observations (1) and (2) encourage an optimization of the incidence matrix [36, p. 4-11]. The sparsity problem can be solved, by storing document-references (also called *Postings*) as a singly-linked list per term which only contains references to such documents that contain the particular term. This list is also called the *Posting-List*. The 0-entries which would otherwise clutter the incidence matrix, are therefore not needed. Another optimization can be made, by sorting the document-ids in the posting-list ascendingly. Below is an example, of what a dictionary of posting-lists may look like, assuming the following sample document ids are assigned: $\{\mathrm{I}(D_{\text{StarWars}}) = 77, \mathrm{I}(D_{\text{Foundation}}) = 51, \mathrm{I}(D_{\text{Galactica}}) = 78\}$:

...
[cylon] $\rightarrow 78$
[empire] $\rightarrow \cdots \rightarrow 51 \rightarrow 77 \rightarrow 78 \rightarrow \ldots$
[earth] $\rightarrow \cdots \rightarrow 51 \rightarrow 78 \rightarrow \ldots$
[light saber] $\rightarrow 77$
...

Figure 2.2.: Exemplary posting-list dictionary snippet

If the retrieval algorithm uses set intersection as described above, by means of the pre-sorted posting-lists, it can compute $A \cap B$, and therefore $M_{\mathscr{B}}(D, Q_{\mathscr{B}})$, with linear run time complexity [28]. Set unification as a model of OR is equally intuitively computable in linear time. Set subtraction ($\neg T$) may be modeled as an intersection with the complement of $DocumentsWithTerm(T)$. Another run time performance gain can be made, by ordering the intersections by their restrictiveness: First intersect the smallest set with the second-smallest, the result with the third-smallest, and so on. That way it is assured that the intersection operation terminates in the smallest amount of time possible.

2.2.3. Extended functionality of a Boolean retrieval model

Due to the historic importance of the Boolean retrieval model, there exist a number of extensions for it which shall be mentioned. These extensions enable use-cases which are not covered by the *contains*-function [36, p. 15]:

1. A so-called *proximity operator*, by means of which a word can be searched *in proximity* of another word. This is particularly relevant for long documents, e.g. books. For example, given a corpus of chemistry books, a search for health effects of radiation might be formulated as $proximate(T_{\text{health}}, T_{\text{radiation}})$

2. The ability to search for phrases (immediately chained sequences of terms), such as "global warming".

3. The ability to search for part-tokens through wildcards ('*'). For example, a search engine user may want to match all mentions of the 1950's in a news corpus by searching for "195*".

4. Phonetic Search: Token normalization under consideration of pronunciation.

2.2.4. Non-binary relevance judgments in a Boolean retrieval model

An important drawback of $M_{\mathscr{B}}$ (as presented so far) is that its returned relevance score is only either TRUE or FALSE which makes a useful ranking practically impossible (an arbitrary set of TRUE-values is already "sorted"). Therefore, beyond an initial, randomly ordered set of potentially relevant documents, the user of the IR system is not assisted in satisfying her information need.

An approach towards resolving this drawback is offered by the insight that retrieval units usually don't just consist of a single piece of unstructured text. On the contrary, most objects have an intrinsic set of fields/properties (meta-data), only a couple of which will contain unstructured data (e.g. free text). For example, books usually have a title, an author, a date of publication, an edition, and last but not least their content.

An extended Boolean IR model enables queries over such additional properties of the retrieval unit: An inverted index is generated for every property. Two different kinds of indices are differentiated here: **Parametric Indices** for fields with structured values (numbers, dates etc.) and **Zone Indices** for fields with unstructured values (e.g. titles, free text), as handled so far [36, p. 101-102]. This allows for the important extension:

Documents are now modeled as n-tuples $D_{\mathscr{B}} = (d_i | 1 \leq i \leq k)$ (shorter noted as $(d_i)_{i=1}^k$), where k is the number of fields in each retrieval unit. The same holds for queries $Q_{\mathscr{B}} = q_1^k$. Further Boolean functions $f : *, d_k \to \mathbb{B}$ which may operate in the continuous domains of parametric indices (like a function $between$: date_after, date_before, $d_k \to \mathbb{B}$ to constrain a date/time property), may also be defined. Furthermore, let $w_{1 \leq i \leq k} \in \mathbb{R}^+$ be a user-defined series of weights, with the constraint that $\sum_{i=1}^k w_i = 1$. As such, the model $M_{\mathscr{B}}$ can be extended as follows:

Definition 3 (Weighted Zone Indexing Model)**.**

$$M_{\mathscr{B}}((d_i)_{i=1}^k, (q_i)_{i=1}^k, (w_i)_{i=1}^k) = \sum_{i=1}^{k} q_i(d_i) * w_i$$

The model from definition 3 is also called the *Weighted Zone Indexing* or *Ranked Boolean Retrieval* model [36, p. 102-107].

2.3. The Vector Space model

2.3.1. Modeling documents as vectors

A deeper look at the incidence matrix $M_{ji}^{\mathscr{T}}$ which was introduced in section 2.2.1, reveals that its columns are almost valid vector representations for the documents D_j of a corpus, if for each of its term frequency components $T_i \in \overrightarrow{D_j}$ the integer frequency of the term is stored instead of just the lower-information TRUE/FALSE value. Each column vector in M_{ji} would then correspond to a term-frequency distribution over all terms for a particular document.

To enable a simple yet powerful ranking model with these column vectors, a few observations must be made:

1. Most ad-hoc queries are logically limited to conjunctions (a loosely ordered set of terms which sketches the users information need).

2. If one of the query terms has nonzero frequency in multiple documents, the document for which the term has higher frequency should be ranked more highly.

3. A term which appears in *too many* documents -a stoppword, for example- should *not strongly* increase the documents relevance score.

In order to formalize these observations in the Vector Space model, a metric called *Inverse Document Frequency* (IDF) is needed. This metric is used as a factor for the frequency of a term within a document, to boost or suppress its importance for the relevance score. Let C be a corpus, T_i a term-equivalency class from C, $|C|$ the total number of documents and $tf(T_i, \overrightarrow{d}) = d_i$ the term frequency of the term T_i in document $D = \overrightarrow{d}$. Then $idf : T \to \mathbb{R}$ is calculated with the following formula:

Definition 4 (Definition of $idf(T)$)**.**

$$idf(T) = log \frac{|C|}{|\{D | D \in C \wedge tf(T, D) > 0\}|}$$

Notice the following properties of the (co-)domain of *IDF*:

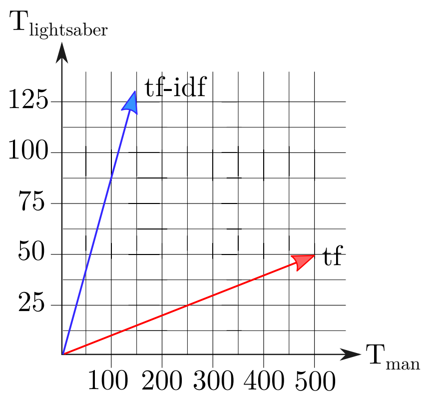
1. The higher the number of total documents in the corpus within which T_i appears, the lower is *IDF*. The minimum of the function is 0 ($log 1$), if T_i appears in each document.
2. The logarithm 'dampens' the curve in a 'sub-linear' fashion: It is still monotonously growing within its domain, but extremely rare terms (e.g. homonyms) are not valued proportionally higher [21].

With IDF in mind, it is now possible to introduce *TF-IDF* term weighting [36, p. 108-110]. In its basic form, it is calculated as described in definition 5.

Definition 5 (Definition of $tf\text{-}idf(D, T)$)**.**

$$tf\text{-}idf(D, T) = idf(T) * tf(T, D)$$

There are several alternate versions of the aforementioned formulae which, for example, also weaken the tf-factor sub-linearly [23]. The geometric impact of TF-IDF is illustrated in figure 2.3.



|C|=2e3

$T \in D_{StarWars}$	tf	df	idf	tf-idf
$T_{lightsaber}$	50	6	2.52	126
T_{man}	500	1e3	0.30	150

Figure 2.3.: Example of *TF-IDF* impact (Graphic produced by this work.)

A shortcoming of $\overrightarrow{vd}(\overrightarrow{d}) = \langle tf\text{-}idf(\overrightarrow{d}, T_i)\rangle_{i=1}^{N}$ as a document representation is that in addition to a documents semantics, it also encodes its size. For example, a short summary text of the book *Foundation* may be semantically very similar to the book itself, yet their vector representations $\overrightarrow{vd}_{\text{Summary}}$ and $\overrightarrow{vd}_{\text{Book}}$ will be far apart in euclidean space, because the coordinates of $\overrightarrow{vd}_{\text{Book}}$ will be larger by magnitudes. For this reason, document vectors are normalized:

Equation 2 (Calculation of the normalized document vector).

$$\overrightarrow{vdn}(\overrightarrow{d}) = \overrightarrow{vd}(\overrightarrow{d})/|\overrightarrow{vd}(\overrightarrow{d})|$$

Now, when picturing document vectors $\overrightarrow{vdn}$ as euclidean points, they all appear on the surface of the same unit hypersphere.

2.3.2. Ranking in the Vector Space model

Let $Q_V \subseteq \{T_i | 1 \leq i \leq N\}$ be a set of 'conjunct' terms, with which the user of an IR system has modeled her information need. Then $\overrightarrow{vq}(Q_V) = \langle|\{T_i\} \cap Q_V|\rangle_{i=1}^{N}$ is, similar to the document vector $\overrightarrow{vd}$, a vector which represents a query. The vector is also normalized, such that $\overrightarrow{vqn}(Q_V) = \frac{\overrightarrow{vq}(Q_V)}{|\overrightarrow{vq}(Q_V)|}$. The euclidean point $\overrightarrow{vqn}$ now appears on the surface of the same unit hypersphere as the set of normalized document vectors $\overrightarrow{vdn}$. As an intuitive metric for measuring the distance between two points on the surface of a sphere, the cosine may be used [29]. As such, the relevance score of the Vector Space model $M_V : Q_V, D \rightarrow \mathbb{R}$ is defined as follows:

Definition 6 (The Vector Space model $M_V : Q_V, D \rightarrow \mathbb{R}$).

$$M_V(Q_V, D) = \overrightarrow{vdn}(D) \cdot \overrightarrow{vqn}(Q_V)$$

2.4. Statistical Models

2.4.1. The Probabilistic Ranking Principle (PRP)

Probability theory offers a foundational basis for making decisions under fuzzy conditions [36, p. 201]. Let the classification of the random variable R be a so-called *1/0-loss* experiment, where R is classified binarily (e.g. *'relevant'/'irrelevant'*), depending on the prior variables d, q. Then it can be proven that $R = \begin{cases} 1 : P(R=1|d,q) > P(R=0|d,q) \\ 0 : otherwise \end{cases}$ is an optimal decision, if the prior probabilities $P(R)$ and $P(d,q)$, as well as the posterior probabilities $P(d,q|R)$ are exactly known (in reality they can only be estimated approximately though). This is also called the *Optimal Bayesian Decision Rule*. If this is the case (the priors and posteriors are known with sufficient accuracy), then $P(R=1|d,q)$ can be calculated with the following formula [70]:

Definition 7 (Bayes' Theorem).

$$P(R=1|d,q) = \frac{P(d,q|R=1) * P(R=1)}{P(d,q)}$$

Let R be the anticipated relevance score, given a document d and a query q. Then Bayes' Theorem above from definition 7 is also called the *Probabilistic Ranking Principle (PRP)* [24] [25, p. 113-114].

2.4.2. The Binary Independence Model (BIM)

Let $\overrightarrow{q}$ and $\overrightarrow{d}$ be term incidence vector, similar to $\overrightarrow{vqn}$ and $\overrightarrow{vdn}$ from chapter 2.3.2 which model a users information need, as well as a document from a corpus. But, as opposed to $\overrightarrow{vqn}$ and $\overrightarrow{vdn}$ the domain of q_i and d_i is binary (*Bag-of-Words Representation*: 1='*Term occurs*',0='*Term does not occur*'). Also, let R be a random variable which determines, whether a document is relevant in the context of a user query. Under application of Bayes' Theorem, it is now possible to define a probabilistic scoring model $M_{\text{PRP}} : \overrightarrow{q}, \overrightarrow{d} \rightarrow [0,1] \subset \mathbb{R}$ [24] [25]:

Definition 8 (The Binary Independence Model)**.**

$$M_{BI}(\overrightarrow{q}, \overrightarrow{d}) = \frac{P(R=1|\overrightarrow{q}, \overrightarrow{d})}{P(R=0|\overrightarrow{q}, \overrightarrow{d})}$$

This ratio of disjunctive probabilities is also called *Odds* (chance). The transformations of $P(R|\overrightarrow{q}, \overrightarrow{d})$ in equation 3 are needed to simplify M_{BI} [36, p. 204-207].

Equation 3 (Transformation of the BI-model under application of the chain rule)**.**

$$\begin{aligned}
P(R|\overrightarrow{q}, \overrightarrow{d}) &= \frac{P(\overrightarrow{q}, \overrightarrow{d}|R) * P(R)}{P(\overrightarrow{q}, \overrightarrow{d})} \\
&= \frac{P(\overrightarrow{q}, \overrightarrow{d}, R)}{P(\overrightarrow{d}|\overrightarrow{q}) * P(\overrightarrow{q})} \\
&= \frac{P(\overrightarrow{d}|R, \overrightarrow{q}) * P(R, \overrightarrow{q})}{P(\overrightarrow{d}|\overrightarrow{q}) * P(\overrightarrow{q})} \\
&= \frac{P(\overrightarrow{d}|R, \overrightarrow{q}) * P(R|\overrightarrow{q})}{P(\overrightarrow{d}|\overrightarrow{q})} \\
\rightarrow M_{BI}(\overrightarrow{q}, \overrightarrow{d}) &= \frac{P(R=1|\overrightarrow{q}, \overrightarrow{d})}{P(R=0|\overrightarrow{q}, \overrightarrow{d})} \\
&= \frac{\dfrac{P(\overrightarrow{d}|R=1, \overrightarrow{q}) * P(R=1|\overrightarrow{q})}{P(\overrightarrow{d}|\overrightarrow{q})}}{\dfrac{P(\overrightarrow{d}|R=0, \overrightarrow{q}) * P(R=0|\overrightarrow{q})}{P(\overrightarrow{d}|\overrightarrow{q})}} \\
&= \frac{P(\overrightarrow{d}|R=1, \overrightarrow{q}) * P(R=1|\overrightarrow{q})}{P(\overrightarrow{d}|R=0, \overrightarrow{q}) * P(R=0|\overrightarrow{q})}
\end{aligned}$$

Because the factors $P(R=1|\overrightarrow{q})$ and $P(R=1|\overrightarrow{q})$ are constant for the ranking of a set of documents for a single query, they can be removed without changing the ranking. Also, just as in the Vector Space model, the highly simplifying assumption is made that occurrences of different terms are always fully independent events. As such, M_{BI} can be simplified as in equation 4.

Equation 4 (Transformation of the BI-model with inter-term independence assumption).

$$
\begin{aligned}
M_{BI}(\vec{q}, \vec{d}) &= \frac{P(\vec{d}|R=1,\vec{q})}{P(\vec{d}|R=0,\vec{q})} \\
&= \prod_{i=1}^{N} \frac{P(d_i|R=1,\vec{q})}{P(d_i|R=0,\vec{q})} \\
&= \prod_{i=1:d_i=1}^{N} \frac{P(d_i=1|R=1,\vec{q})}{P(d_i=1|R=0,\vec{q})} * \prod_{i=1:d_i=0}^{N} \frac{P(d_i=0|R=1,\vec{q})}{P(d_i=0|R=0,\vec{q})}
\end{aligned}
$$

Let p_i be $P(d_i=1|R=1,\vec{q})$ and u_i be $P(d_i=1|R=1,\vec{q})$. Also, the assumption is made that the absence of a term in a query should not be considered an event that influences the ranking. Under those assumptions, further simplifications as in equation 5 can be made, because $\sum_{\alpha\in\{0,1\}} P(d_i=\alpha|R,\vec{q})=1$ [36, p. 206-207].

Equation 5 (Transformation of the BI-model focusing on $d_i : q_i = 1$).

$$
\begin{aligned}
M_{BI}(\vec{q}, \vec{d}) &= \prod_{i=1:d_i=1}^{N} \frac{P(d_i=1|R=1,\vec{q})}{P(d_i=1|R=0,\vec{q})} * \prod_{i=1:d_i=0}^{N} \frac{P(d_i=0|R=1,\vec{q})}{P(d_i=0|R=0,\vec{q})} \\
&\sim \prod_{i=1:d_i=q_i=1}^{N} \frac{p_i}{u_i} * \prod_{i=1:d_i=0,q_i=1}^{N} \frac{1-p_i}{1-u_i} \\
&= \prod_{i=1:d_i=q_i=1}^{N} \frac{p_i*(1-u_i)}{u_i*(1-p_i)} * \prod_{i=1:q_i=1}^{N} \frac{1-p_i}{1-u_i}
\end{aligned}
$$

Here, again, the left-most term is constant throughout all documents for a constant query, such that it may be ignored. By taking the logarithm of left remaining value, the so-called *Retrieval Status Value (RSV)* can be calculated:

Definition 9 (Transforming the BI-model into the RSV).

$$
\begin{aligned}
M_{BI}(\vec{q}, \vec{d}) &= RSV_{\vec{q},\vec{d}} \\
&= \log \prod_{i=1:d_i=q_i=1}^{N} \frac{p_i*(1-u_i)}{u_i*(1-p_i)} \\
&= \sum_{i=1:d_i=q_i=1}^{N} \log \frac{p_i*(1-u_i)}{u_i*(1-p_i)} \\
&= \sum_{i=1:d_i=q_i=1}^{N} \log \frac{p_i}{1-p_i} + \log \frac{1-u_i}{u_i}
\end{aligned}
$$

Now, the only remaining task is to find meaningful values for the probabilities p_i (the document containing the term T_i is relevant), and u_i (the document containing the term T_i is irrelevant). Interestingly enough, u_i may be approximated with $idf(T_i)$ (see definition 4, chapter 2.3).

The calculation of p_i presents a larger challenge, because the binary bag-of-words document representation allows no judgment over the relevance influence of a single term. That is why mostly, an initial value is assumed for p_i, e.g. $\frac{1}{2}$. Over the course the IR systems run time, per-term relevance information can be gained via a process called *Relevance Feedback*: When a user marks a document from a ranked list $\mathscr{L}$ of the best n retrieved documents $D = \overrightarrow{d} \in C$ for an information need modeled as a query Q as relevant, then the set of tuples $\{(T_i, \overrightarrow{d}) | \overrightarrow{d}_i = 1\}$ may be added to a set M of relevant term incidences. Subsequently, an updated prediction of p_j^{k+1} of p_j^k can be made via *Maximum Likelihood Estimate* (see definition 10).

Definition 10 (Relevance Feedback update of p_i).

$$p_j^{k+1} = \frac{|\{(T_j, D) \in M\}| + \alpha * p_j^k}{|M| + \alpha}$$

The hyperparameter α indicates the weight which the previous (or initial) estimate should have on the new one. Therefore, a complete probabilistic binary independence model could look like definition 11, where $p_i^{max(k)}$ is the "most recent" relevance feedback update of p_i.

Definition 11 (Binary Independence Model with Pseudo-IDF).

$$M_{BI}(\overrightarrow{q}, \overrightarrow{d}) = \sum_{i=1:d_i=q_i=1}^{N} \log \frac{p_i^{max(k)}}{1 - p_i^{max(k)}} + \log \frac{|C|}{|\{\overrightarrow{d} | \overrightarrow{d}_i^N \in C \wedge d_i = 1\}|}$$

2.5. Evaluation in Information Retrieval

To evaluate the quality of a ranking, this work uses the *Normalized Discounted Cumulative Gain (NDCG)* [15–17] algorithm. Let n be the total number of retrieved documents for a certain query Q, and $(r_i \in [0,1])_{i=1}^n$ the tuple of ground truth relevance values of the retrieved document d_i at position i in the ranking. Then the NDCG score for this sequence of relevance labels accumulates a quality score for the ranking, by assigning each result a score which depends on the ground truth relevance and the position of the result. Furthermore, the score is discounted by the position of the result: A relevant retrieved document at position ten will not contribute to the final score as highly as a relevant retrieved document at position one. Definition 12 shows the formula for calculating this basic discounted cumulative gain.

Definition 12 (Discounted Cumulative Gain [15–17]).

$$DCG((r_i)_{i=1}^n) = \sum_{i=1}^{n} \frac{2^{r_i} - 1}{log_2(i) + 1}$$

A slightly more interpretable version of this formula puts the DCG into perspective as a relative measure, by diving it with the *ideal* score that could be achieved by retrieving documents in the order of $\forall g_{2 \leq i \leq n} : g_i \leq g_{i-1}$, where g_i is the ground truth relevance of the document at position i. This relative form of the DCG is called the *Normalized* DCG, as defined in 13.

Definition 13 (Normalized Discounted Cumulative Gain [15–17]).

$$NDCG((r_i)_{i=1}^n, (g_i)_{i=1}^n) = \frac{DCG((r_i)_{i=1}^n)}{DCG((g_i)_{i=1}^n)}$$

2.6. Discussion

2.6.1. Discussion of the Boolean Model

Boolean models are characterized by their strongly deterministic/determined scoring algorithm, and the resulting performance and simplicity of the underlying IR system. They also offer the use case of being able to model very complex, very specific information needs via sufficiently complex Boolean expressions.

But during the evolution of the field of Information Retrieval, from Library Management systems towards Web Search Engines, the importance of sophisticated Boolean operators for the formulation of information needs has declined dramatically. Casual IR system users tend to express their information needs in overly restrictive Boolean queries, when asked to write them [27]. The Boolean model also has the disadvantage that queries, depending on the use of the AND or the OR-operator, tend to result in *either* a high ranking precision *or* a high recall, but rarely both at the same time. As such, the resulting F-measure is usually fairly low [26].

2.6.2. Discussion of the Vector-Space Model

In the spirit of the *Distributional Hypothesis* [39], the Vector Space model proposes an intuitive approach to calculate the relevance score as the maximum semantic overlap through the cosine measure which naturally returns non-binary values. Especially in the domain of purely conjunct ad-hoc queries, the Vector Space model is superior to the Boolean model. Intuitively though, the bijective relationship between orthogonal semantic concepts and terms (as presented so far) is unrealistic. Obvious counter-examples for this orthogonality are synonyms (n *Terms* $\leftrightarrow$ 1 *Meaning*) and homonyms (1 *Term* $\leftrightarrow$ n *Meanings*).

Since the Vector Space model is also the general model used in this thesis for relevance score calculation, a detailed discussion about achieving high-quality document vectors can be found in chapter 3.3.

2.6.3. Discussion of the Binary Independence Model

Probabilistic retrieval models offer a solid theoretical foundation for maximizing the quality of decisions in non-deterministic environments. As such, the Probabilistic Ranking Principle (PRP) describes an optimal retrieval system. Unfortunately, the BIM (Binary Independence Model), the presented implementation of the PRP, makes some very simplifying assumptions, evading the strict correctness requirements towards the posterior term-relevance probabilities. The following assumptions are made [36, p. 212]:

1. Binary Bag-of-Words vectors are sufficiently informative representations of documents and information needs.
2. A binary model of relevance is sufficient.
3. Term incidences of terms T are independent probabilistic events, such that $\forall T_i, T_j : i \neq j \rightarrow p(T_i, T_j) = p(T_i) * p(T_j)$, similar to the orthogonality assumption of term frequency dimensions in the simplest form of the Vector Space model.
4. Terms that do not appear in the query should not influence the ranking.

Some research which partially solves/disarms these assumptions has not been mentioned so far. Language models which approach relevance as the cross-entropy between a query and a document, offer the most promising non-neural approach towards resolving all of the assumptions above [32–34]. The *Okapi-BM25* model is a refined version of the BIM, with a non-binary notion of relevance [4, 5]. BM25-Models were surpassed by Language Models in 2005 for the first time [35].

3. Neural Information Retrieval

3.1. Introduction to Neural Networks

3.1.1. Machine Learning

Machine Learning (ML) techniques have a proven history of success in solving tasks which are fairly easily executed through human intuition, yet extremely hard to program as a deterministic set of rules. Such tasks include driving, recognition of handwriting, or, more to the core of the topic at hand, learning of high information and low dimensional document representations. The standard way of exercising ML is to design a function $m : i, \vec{w} \to o$ which by adjustment of a range of internal degrees of freedom $\vec{w}$ "learns" to map an input i (consisting of a number of scalar *feature* values) to an output o with maximum correctness (or minimum error). The adjustment of m's degrees of freedom or *weights* $\vec{w}$ is carried out by a learning function $l_f : \vec{w}^k, i_{\text{train}}, o_{\text{train}}, o_{\text{predict}} \to \vec{w}^{k+1}$, where $\vec{w}^{(k)}$ is the weight vector at the k-th training step. This is also called *supervised learning*, because the ability for the learning function to compare the predicted output o_{predict} with the correct output o_{train} is regarded as *supervision*. The voluminous set of $(i_{\text{train}}, o_{\text{train}})$-pairs is also called *training data*, and it is a strong requirement of supervised learning techniques.

The acquisition of training data is a laborious and expensive process, because it requires humans to *label* or *annotate* training input with training output. Due to this drawback, a set of so-called *unsupervised* learning functions has been developed which can incorporate *unlabeled data* into the training process. In contrast to *labeled data*, with the emergence of the World Wide Web [71], *unlabeled data* has become ubiquitous, especially unlabeled text corpora (e.g. Wikipedia). Therefore, *unsupervised* learning functions have high economic value if they can learn quality weights.

3.1.2. Neural Networks and Autoencoders

One very practical architecture for unsupervised ML is the Autoencoder (See definition 1) which this work uses for unsupervised learning of dense document, and query representations. The standard Autoencoder architecture envisions a simple *Multi-Layer Perceptron (MLP)*, or *Feed-Forward Neural Network (FFNN)*, as function approximators for both Φ_{AE_t} and $\Phi^{-1}_{AE_t}$ [6], as illustrated in figure 3.1. In figure 3.1, the input data is compressed from $\mathbb{R}^{36}$ to $\mathbb{R}^2$ by the encoder, and color intensity represents neuron output activation.

Definition 14 (Definition of a Neuron [37, 38, 72]). *A neuron $\mathcal{N} : (I \in \mathbb{R}^n, W \in \mathbb{R}^n, b \in \mathbb{R}, f_a : \mathbb{R} \to \mathbb{R}) \to o \in \mathbb{R}$ is a function which maps a series of inputs $I = (i_1, ..., i_n)$ to an output activation o via a series of weights $W = (w_1, ..., w_n)$, a bias b and a continuously differentiable activation function f_a. Given these inputs, o is calculated as $o = f_a(\sum_{k=1}^{n}(i_k w_k) + b)$.*

The property of continuous differentiability of the activation function allows for the computation of an *error gradient* of $\mathcal{N}$ which enables the use of *Gradient descent* and *Backpropagation* [37, 38, 72] as optimization algorithms for $W_{\mathcal{N}}$ and $b_{\mathcal{N}}$.

To compact the visualization of neural network architectures from now on, *compute graph* notation [10] is introduced (as illustrated in figure 3.2). Note, how multiple neurons which operate with the same inputs and activation function are bundled into *fully-*

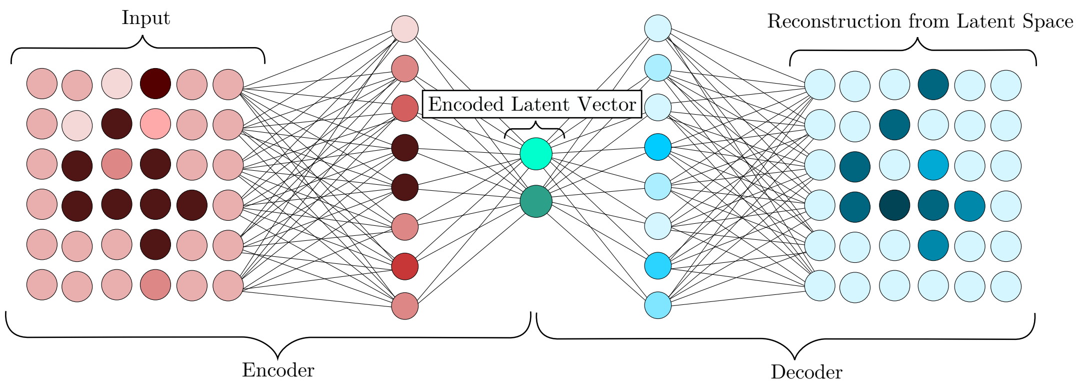


Figure 3.1.: Example of an Autoencoder Neural Network for images of numbers (Graphic produced by this work).

connected layers. They are called fully-connected, because each neuron sees all of the input data.

$\mathbb{R}^x$	Matrix with x elements.		Matrix duplication.
$\mathbb{R}^n$ $\mathbb{R}^{n+m}$ $\mathbb{R}^m$	Matrix concatenation.	$\mathbb{R}^n$ $\mathbb{R}^{n+m}$ $\mathbb{R}^m$	Matrix split.
	Neuron layer with sigmoid activation function $\sigma(x) \in [0,1] \in \mathbb{R}$ for "selecting" input.		Neuron layer with tanh activation function $tanh(x) \in [-1,1] \in \mathbb{R}$ for "spreading" input.
$\mathbb{R}^n$ $\mathbb{R}^n$ $\mathbb{R}^n$	Hadamard (element-wise) matrix multiplication	$\mathbb{R}^n$ $\mathbb{R}^n$ $\mathbb{R}^n$	Element-wise matrix addition.
	(Named) Subgraph.		

Figure 3.2.: Compute graph notation (Graphic produced by this work, inspired by TensorFlow Graph Visualization [10])

For example, the image Autoencoder from figure 3.1 would look like figure 3.3 when converted to compute graph notation.

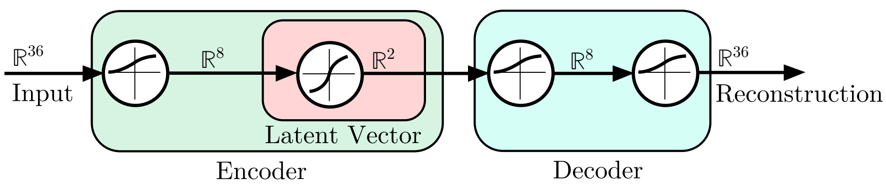


Figure 3.3.: Autoencoder Neural Network from figure 3.1 in Compute Graph notation (Graphic produced by this work).

The standard Autoencoder is extremely useful for fixed-length input, such as images or fixed-size bundles of one-hot encoded term vectors. However, sentences or text documents cannot be used as input, because they strongly vary in length. As such, for this type of *dynamically sequential* data, a different type of function approximator is needed, both for the encoder and decoder.

3.1.3. Recurrent Neural Networks (RNN)

To process sequential data with neural networks, the *Recurrent Neural Network (RNN) Cell* [40] $\texttt{RNN-CELL}_t : S_{t-1}, i_t \rightarrow S_t, o_t$ is introduced. Given a sequence of *input events* $(i_1, ..., i_n)$ the RNN computes the output o_t and the new state S_t at timestep t as a function of the previous state S_{t-1}, and the input i_t itself. Figure 3.4 show the RNN function in compute graph notation.

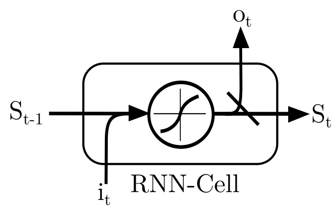


Figure 3.4.: Recurrent Neural Network Cell (Graphic produced by this work).

An input time series is processed, by *unrolling* the RNN cell for the length of that series, to form the RNN. For example, for a language consisting of the terms [`<s>`], [`</s>`], [`peter`], [`mary`], [`likes`], [`ice cream`], [`cake`] an unsupervised RNN language model [41] may be trained by feeding the RNN cell at timestep t the previous word as input, and requiring it to predict the correct current word. Both input and output terms are encoded as one-hot vectors of length 7. The error of the network is computed as the mean cross-entropy between the correct and the predicted term probability distribution for each timestep. An example of such an unrolled RNN language model can be seen in figure 3.5.

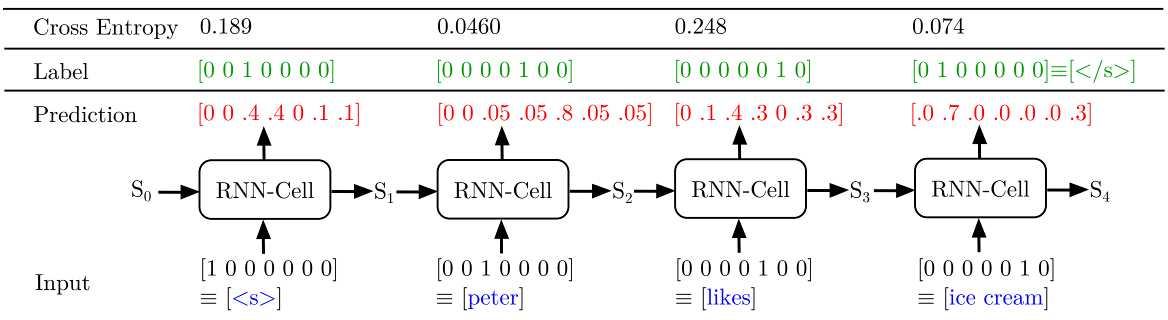


Figure 3.5.: Unrolled RNN language model (Graphic produced by this work).

Unfortunately, Recurrent Neural Networks in this basic form suffer from a problem called the *"diminishing gradient problem"* [42, 43]: Because the RNN essentially functions as a dynamic neural network with as many layers as the input has events, it is very hard for error signals distant from an early timestep to influence the "decisions" which the RNN made at this later timestep. The error gradient "diminishes" as it is back-propagated through time [40]. This inhibits standard RNNs from learning long-term dependencies between events in their input data.

3.1.4. The Long Short-Term Memory (LSTM)

To enable the use of RNNs for modeling sequences with long-term dependencies, a slightly more complex RNN architecture is introduced: The *Long Short-Term Memory (LSTM)* [42, 43]. As illustrated in figure 3.6, unlike the standard RNN cell, the LSTM cell is not forced to produce an entirely new state at every timestep, but instead wired to *add to* and *subtract from* its current state through an input gate and a forget gate. This way, changes in the state can be "blamed" throughout the time series by Backpropagation. The LSTM cell architecture proves to solve the problem of vanishing gradients [65].

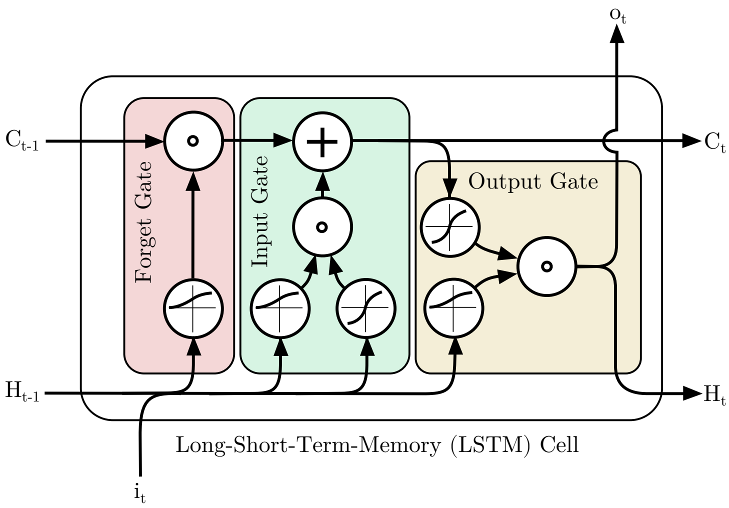


Figure 3.6.: Long Short-Term Memory Cell (Graphic produced by this work).

Note the following interesting properties of the LSTM cell:

1. Via the second state input argument H_{t-1}, the LSTM cell self-feeds its previous output o_{t-1}. This is essentially a bi-gram model built into the LSTM cell.

2. On two occasions, the LSTM cell uses the product of a sigmoid and a tanh layer. In both cases, the sigmoid layer is used to "select" information from the tanh layer via its $[0, 1] \in \mathbb{R}$ outputs. The tanh layer is used to "present" the information that is being selected on a $[-1, 1] \in \mathbb{R}$ scale.

Henceforth, the unrolled LSTM-RNN will be used in graph notation as defined in figure 3.7. In this notation the input sequence always enters the cell from the bottom, and the optional initial state from the left. The output sequence leaves the cell from the top, and the concatenated final state/output from the right.

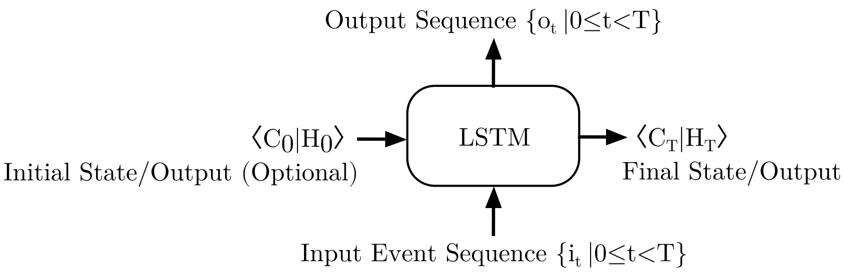


Figure 3.7.: LSTM in graph notation (Graphic produced by this work.)

3.1.5. Sequence Autoencoders

Recurrent Neural Networks enable using Autoencoders to compress sequential data into representation vectors of arbitrary fixed size. Such Sequence Autoencoders are a sub-type of *Sequence-to-Sequence* [8] models which were originally developed for Neural Machine Translation. In the special case of sequence Autoencoders, the *Sequence-to-Sequence* model can be used unsupervised, because it is only required to learn the identity function by "translating" the source sequence into itself via the embedding bottleneck. In its simplest form, such an architecture may look as illustrated in figure 3.8.

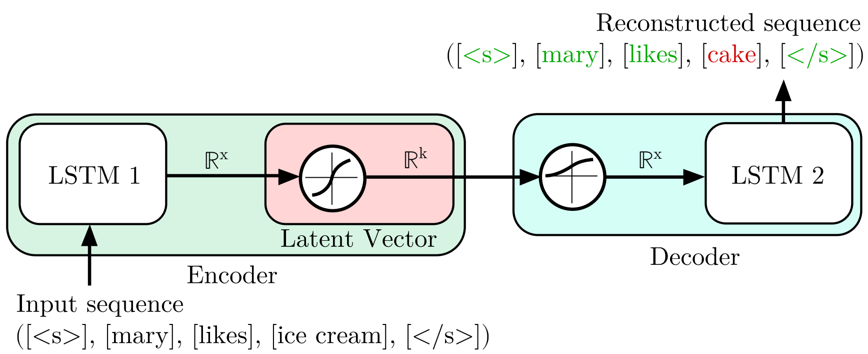


Figure 3.8.: Sequence Autoencoder with LSTMs (Graphic produced by this work).

The loss of the sequence Autoencoder is defined as either the mean cross-entropy, the mean squared error or the mean cosine similarity between the reconstructed sequence $(r_{nk}) \in \mathbb{R}^{n \times k}$ from the decoder with n timesteps and k features, and the "correct" encoder's input sequence $(s_{nk}) \in \mathbb{R}^{n \times k}$, as compared in table 3.1 [68].

	Mean Squared Error	**Mean Cosine Distance**	**Mean Cross-Entropy**
Formula	$\frac{\sum_{t=1}^{n}\sum_{i=1}^{k}(r_{ti} - s_{ti})^2}{n * k}$	$\frac{\sum_{t=1}^{n} \vec{r}_t \cdot \vec{s}_t}{n * \prod_{t=1}^{n} \lvert\vec{r}_t\rvert\lvert\vec{s}_t\rvert} + 1$	$\frac{\sum_{t=1}^{n} - \sum_{i=1}^{k} s_{ti} * log(r_{ti})}{n}$
Behavior	Simple euclidean interpretation, but exponential and may lead to chaotic/divergent learning.	Linear, therefore danger of non-convergence (momentum-based gradient never going "uphill" to "slow down" [66]).	Excellent convergence due to logarithm, but requires probability distributions as feature vectors, therefore not directly compatible with *Word2Vec*.

Table 3.1.: Comparison of sequence loss functions.

To increase the abstraction capabilities of the encoder and decoder, two architectural improvements are commonly applied in sequence Autoencoder models: The **first** such improvement is a so-called *Bidirectional RNN (Bi-LSTM)* [73]. Given an input sequence $(i_0, \ldots, i_n)$, the Bi-LSTM augments the input i_t at timestep t with information about all previous timesteps (Forward pass) $i_{[0,t]}$ and information about all succeeding timesteps (Backward pass) $i_{(t,n]}$. Such a Bi-LSTM is illustrated in figure 3.9. The Rev. operations on the input and output sequences of the Backward LSTM indicate the *reversal* of those sequences.

As also illustrated in figure 3.9, the **second** improvement is to stack [64] multiple LSTM layers to improve the abstraction capabilities of the encoder and decoder RNN.

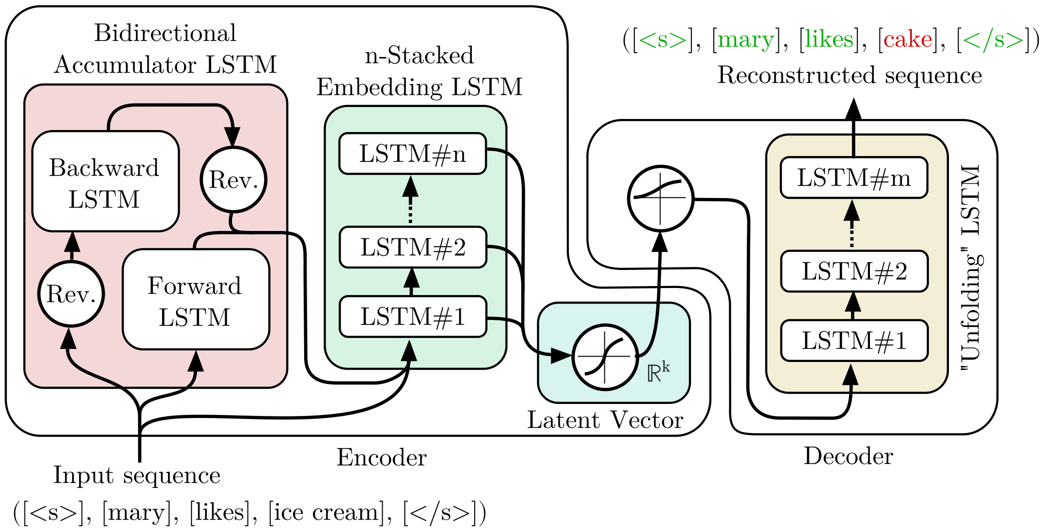


Figure 3.9.: Sequence Autoencoder with Bidirectional and Stacked LSTMs (Graphic produced by this work).

3.1.6. Variational Autoencoders

One problem of normal Autoencoders, especially sequence Autoencoders, is that they are not robust against smooth interpolation of their latent space [9, 48]. This means that Autoencoders might assign meaning to non-uniformly distributed sub-spaces of the latent space, because there is no training incentive to normalize “gaps” in the latent codes. If such a latent space is then used for Cosine-similarity based Information Retrieval, rankings are likely to contain very sharp score "cliffs" (e.g. similarity scores of $[1, 1, 0.99, 0.2]$ etc.). As a consequence, the model may not have learned global conceptual dimensions, only disjoint sub-spaces of similar documents.

To encourage the network to utilize its latent space more uniformly, insights from variational inference techniques are combined with Autoencoders to form so-called *Variational Recurrent Autoencoders (VRAE)* [46, 48]. Most importantly, a loss term is added to the encoder network which forces the generated latent vectors to roughly look like a unit Gaussian distribution $\mathcal{N}$. To achieve this, the encoder not only has to output a latent mean vector $\overrightarrow{\texttt{z\_mean}}$, but also a standard deviation vector $\overrightarrow{\texttt{z\_stddev}}$, from which the decoder input $\overrightarrow{\texttt{latent}}$ is sampled (See figure 3.10). The random normal sampling from the encoder distribution is also called the *“reparameterization trick”* [46, 48]. Therefore, the loss `vrae_loss` of the VRAE is defined as in def. 15.

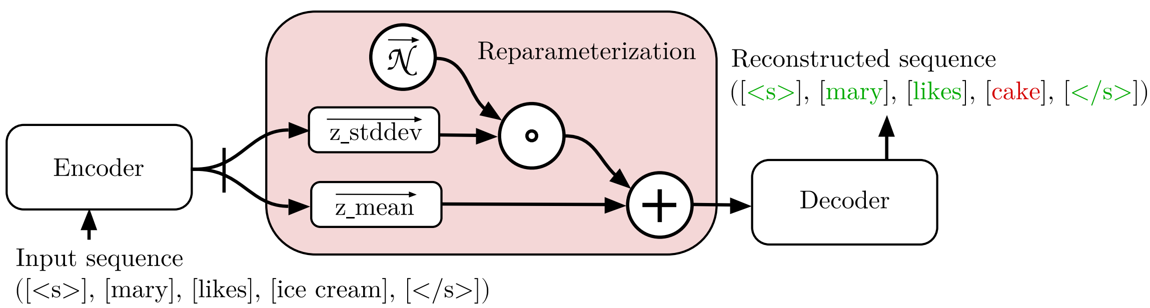


Figure 3.10.: Variational Sequence Autoencoder (Graphic produced by this work).

Definition 15 (Variational Recurrent Autoencoder Loss [46]).

$$\texttt{sample\_sequence} := (i_1 : [\textit{<seq>}], i_2 : [\textit{mary}], i_3 : [\textit{likes}], i_4 : [\textit{cake}], i_5 : [\textit{</seq>}])$$

$$\overrightarrow{\texttt{z\_mean}}, \overrightarrow{\texttt{z\_stddev}} := \Phi_{\texttt{VRAE}}(\texttt{sample\_sequence})$$

$$\boldsymbol{latent\_loss} := KLD(\overrightarrow{\texttt{z\_mean}}, \overrightarrow{\texttt{z\_stddev}}, \mathcal{N})$$

$$\overrightarrow{\texttt{latent}} := \overrightarrow{\mathcal{N}_{\texttt{random}}} * \overrightarrow{\texttt{z\_stddev}} + \overrightarrow{\texttt{z\_mean}} \quad \textit{// Reparameterization}$$

$$\texttt{reconstructed\_sequence} := \Phi^{-1}_{\texttt{VRAE}}(\overrightarrow{\texttt{latent}})$$

$$:= (j_1 : [\textit{<seq>}], j_2 : [\textit{mary}], j_3 : [\textit{likes}], j_4 : [\textit{ice cream}], j_5 : [\textit{</seq>}])$$

$$\boldsymbol{reconstruction\_loss} := \frac{1}{len(\texttt{sample\_sequence})} \sum_{t=1}^{len(\texttt{sample\_sequence})} (i_t - j_t)^2$$

$$\boldsymbol{vrae\_loss} := \boldsymbol{reconstruction\_loss} + \boldsymbol{latent\_loss}$$

Similar to the cross-entropy, the *Kullback Leibler Divergence (KLD)* [74] measures the similarity between two probability distributions, in this case the difference between the distribution given by $(\overrightarrow{\texttt{z\_mean}}, \overrightarrow{\texttt{z\_stddev}})$ and the Gaussian Normal Distribution $\mathcal{N}$. With those foundations explained, a more specific analysis of state-of-the-art Neu-IR models follows in sections 3.2 and 3.3.

3.2. Deep Relevance Matching

The sequence-to-sequence architectures from sections 3.1.5 and 3.1.6 are used by this work, but are by no means the foundation of every *Neu-IR* model. In the *Deep Relevance Matching* class of Neural IR models, IR is formalized as a text matching problem [52, 53, 55]: Simple lexical representations $\Phi_{lex}(D)$, $\Phi_{lex}(Q)$ for queries Q and documents D are evaluated by a complex neural matching function $S : \Phi_{lex}(D), \Phi_{lex}(Q) \rightarrow \mathbb{R}_{DRM}$ which maps $\Phi_{lex}(D)$ and $\Phi_{lex}(Q)$ via a deep Feed-Forward Neural Network to a a relevance score.

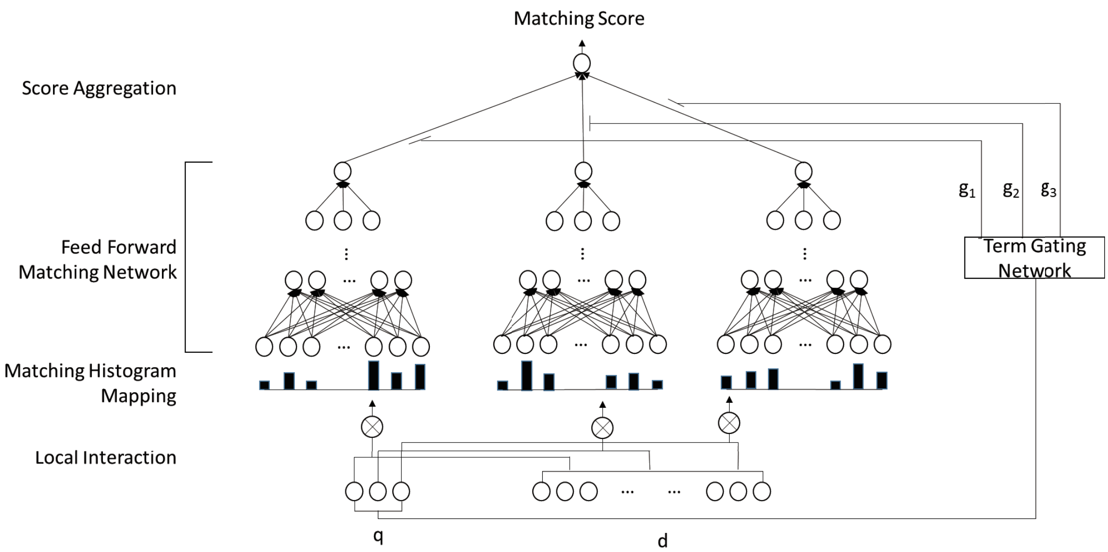


Figure 3.11.: Deep Relevance Matching Model [55]

The best such current model is the *Deep Relevance Matching Model (DRMM)* [55] (illustrated in figure 3.11). It models the relevance score calculation between a list of n query terms $t^Q_{0\le i<n}$ and m document terms $t^D_{0\le i<m}$ as a three-step process:

Definition 16 (Relevance Score calculation in the DRMM [55])**.**

1. *For each query term t^Q a so-called log-count interaction histogram (LCH) is accumulated over the list of $(t^Q \times t^D_{0\le i<m})$ pairs. The accumulated features per query token include k similarity-degree features $s_{0\le i<k}$ and one exact-match feature s_{exact}. The similarity features each cover a certain partition of the cosine similarity range $[-1,1]$. For example, if $k=4$, there would be 4 buckets covering $\{[-1,-.5),[-.5,0),\ [0,.5),[.5,1.0]\}$, respectively. Given (t^Q,t^D), s_i is incremented if $\frac{i}{k} \le \frac{cos(t^Q,t^D)+1}{2} < \frac{i+1}{k}$. The exact-match feature is incremented, if $t^Q = t^D$. Thus, a histogram $H_{t^Q} = \langle log(s_{0\le i<k})|log(s_{exact})\rangle$ with $k+1$ values is produced for each term. A smoothing logarithm is applied to each histogram value.*

2. *Each query term interaction histogram H_{t^Q} is mapped to a relevance score $r^{t^Q}_{0\le i<n} \in \mathbb{R}$ via a Feed-Forward Neural Network.*

3. *The individual query term scores $r^{t^Q}_{0\le i<n}$ are weighted with IDF (see definition 4, chapter 2.3) and accumulated to form the final relevance score $S = \sum \mathrm{IDF}(t^D_i) * r^{t^Q}_i$.*

Note the following interesting properties of the DRMM:

1. It is currently the first neural model for Ad-hoc IR that beats the established statistical models.

2. It does so without a notion of word order. Word order is neither considered for the query, nor for the document.

3. The calculation of a relevance score $S : \Phi_{lex}(D), \Phi_{lex}(Q) \to \mathbb{R}_{DRMM}$ for a query of length n and a document of length m has very unfavorable run-time complexity of $O(n*m)$.

4. Training of the DRMM is a supervised process which requires labeled corpora of $\{(\text{query}, \text{document})\}$ pairs.

3.3. Deep Semantic Matching

3.3.1. Motivation

The performance (both in execution speed and ranking quality) of a text retrieval system is highly dependent on the quality of the internal document, and query representations. Therefore, *Deep Semantic Matching* Models in IR emphasize the complexity of the encoding function $\Phi(x)$, whereas the scoring function $S(\Phi(D),\Phi(Q))$ is usually a straightforward cosine measure between the query and the document representation vector. Those representations are generated from the documents' and queries' text by a so-called embedding algorithm which "embeds" the documents as points into a space with a particular fixed dimensionality. This space is also called a latent semantic space, because its dimensions are potentially (latently) representing particular semantic concepts. This work identifies four main properties of the latent semantic space that may be produced by an embedding algorithm:

1. **Online (incremental) improvement capability**: If the document corpus that is encoded by the embedding model changes frequently, it is important that the model can be incrementally adapted, especially for some of the more traditional semantic approaches (in particular *Latent Semantic Analysis (LSA)*) the model is completely invalidated if the corpus changes, and therefore it could not be improved upon incrementally.

2. **Capability to capture information from word order**: Most models [4, 5, 20–35] assume that word order does not strongly influence the overarching semantics of a document. From an NLP perspective though, grammar is necessary for Natural Language Understanding. Therefore, models that ignore word order tend to capture only very superficial semantics, if any. This is indicative of the traditional divide between IR and NLP [49, 50].

3. **Sparsity of the latent coordinates**: An optimal semantic vector space would contain only dimensions that are completely perpendicular to each other. These dimensions are also called the "basis vectors" of the latent semantic space. If the space contains more dimensions than it has linearly independent basis vectors, the points in this space are going to be artificially "spread out". This problem usually expresses itself in a large amount of zero coordinates contained in the semantic vectors.

4. **Syntactic/Semantic character of latent dimensions**: Different search use-cases require different document representations. Two such use-cases that are highly divergent representation-wise are, for example, recommendation search and Boolean retrieval. The latter use-case requires the representation to contain terms discretely and deterministically, in order to be able to evaluate contains(document, term) Boolean queries. On the other hand, recommendations are (mostly) not guided by explicit conditions, and require a deeper, abstract understanding of the document to be encoded by the latent dimensions.

3.3.2. Revisiting Algorithms for Dense Document Embeddings

With the previous establishment of above mentioned four dimensions of comparison for latent embedding models, six such models are presented here.

Latent Semantic Analysis/Indexing (LSA/I) [30] implies the application of Gaussian elimination, also called "row reduction" which is a popular method to reduce a vector space to its basis spanning vectors . To apply row reduction to a document space, first all document term-frequency vectors (see chapter 2.3.1) are combined into a matrix, where each document vector is one column. LSA employs a slightly more sophisticated, fuzzy version of Gaussian elimination called *"Singular Value Decomposition" (SVD)*. This method gained a lot of traction for its mathematically motivated approach to create a densely populated latent semantic space.

The most important drawback of LSA is the extreme computational effort behind SVD. The computational complexity of this algorithm is $O(n^2 * k^3)$, where n is the number of documents plus the number of terms, and k is the number of embedded latent dimensions. This is especially unfortunate considering that any changes to n will trigger the need to re-execute SVD.

Another method for creating dense document representations is **Latent Dirichlet Allocation (LDA)** [31]. *Dirichlet* is a synonym for *topic*. It is essentially a probabilistic clustering method which tries to associate terms T with fixed-length sets of dirichlets (topics) Q by calculating the posterior probability $p(t|q)$ for each $t \in T$, $q \in Q$. The optimization goal is for each dirichlet to be as unique as possible in its posterior term probabilities.

The vector of posterior probabilities of a document d to have a particular topic q, $p(D|Q)$, can then be used to embed the document into a latent space. This space will be densely populated and encode reasonable semantic abstractions, such as *dog-related* and *cat-related*. However, due to the models focus on words, the abstractions stay in the realm of "topics" which are superficial by nature. Non-topical characteristics, such as attitude or style of writing which would be necessary for a truly insightful encoding, remain too subtle to be captured.

Diverging from previous statistical/count-based methods **(TF-IDF weighted) Distributed Bag-Of-Embedded-Words (DBoEW)** [14] emerged as a very viable document embedding model after Mikolov and his team presented a novel approach to encode terms: The Word2Vec system [11]. A Neural Network Autoencoder was put to the task of embedding terms into a latent space by predicting the term's context from the latent space (or vice versa). Starting with this work, embedded word vectors slowly started to replace one-hot vectors as the standard term representation.

The latent embedding space produced by the Word2Vec system tends to be of such high quality regarding its semantic encoding behavior and the density of the resulting vector space that the idea of using it to represent higher-order objects like sentences, paragraphs or even full documents seems reasonable.

This leads to a vector representation of documents as a TF-IDF weighted sum over the embedding vectors of all terms in the document. However, this approach still assumes that word order does not influence meaning in a considerable way. This assumption is obviously still flawed as much as it was in the 1950's. However, the simplicity and performance behind this approach makes weighted word embedding sums a respectable document embedding technique.

Following the impressive work that was done as part of the Word2Vec system, Mikolov's team turned their attention to the embedding of sentences, paragraphs and documents. They delivered in 2014 with the **Doc2Vec** system [44]. However, this system was not able to prove itself directly relevant for IR [1–3].

One of the most experimental, but highly promising [56] approaches for encoding text as a sequence of words lies in **(Variational) Sequence-To-Sequence Autoencoders** [46, 47]. RNN encoder-decoder models have mostly been researched in relation to Neural Machine Translation [8]: An encoding RNN outputs a so-called "thought vector" as its final state. This thought vector then serves as the initial input to a decoder RNN. Backpropagation of error is performed over the outputs of the decoder back into the encoder. The optimization requirement is for the encoder to produce a latent representation which is good enough for the decoder to reproduce the document (see chapter 3.1.5).

3.3.3. The Neural Vector Space model

Currently, the most successful Deep Semantic Matching Model is the *Neural Vector Space model (NVSM)* [58] which learns both query word and document representations by backpropagation, much like the *Doc2Vec* system. Previous models which also utilize a neural representation function include *Deep Structured Semantic Matching (DSSM)* [51], *Deep Sentence Embeddings from RNNs* [56] and *ARC-I* [53].

As illustrated in figure 3.12, the NVSM simultaneously learns two embedding spaces; one for queries and one for documents. The $\boxed{\text{emb.}}$ operation symbolizes a function $lookup(x)$ which looks up a trainable vector for the given word or document x. Query vectors are averaged, and translated into the document space via a neural tanh layer. Both query word vectors and document vectors are learned jointly by Backpropagation from the cosine similarity error signal by the following algorithm:

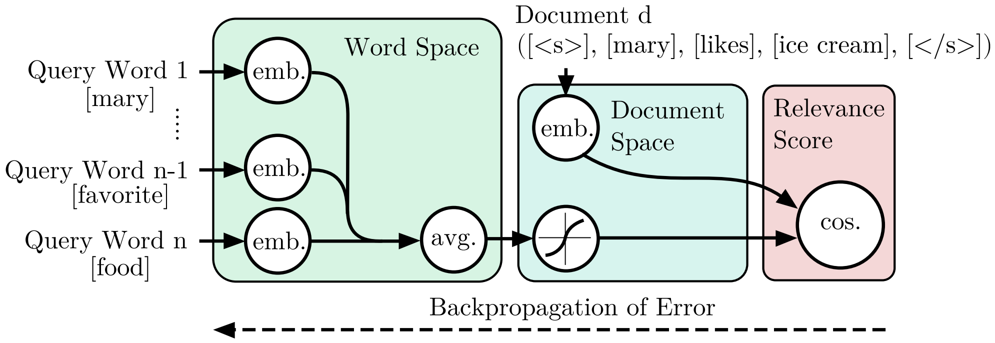


Figure 3.12.: Neural Vector Space model (Graphic produced by this work).

Definition 17 (Definition of the NVSM optimization algorithm [58]).

1. *At first, all document vectors* $\overrightarrow{d}_{0 \leq i < |D|} \in D$ *and query word vectors* $\overrightarrow{v}_{0 \leq i < |V|} \in V$ *are initialized randomly.*

2. *A random selection of* $(ngram \subset V, \overrightarrow{d_{sampled}} \in D, \lambda_{sim.} \in \mathbb{R})$ *triplets is sampled from the corpus. The similarity label* $\lambda_{sim.}$ *is estimated from the cosine similarity between* $\overrightarrow{d_{sampled}}$ *and the average vector of documents* $\{\overrightarrow{d_i} \mid \overrightarrow{d_i} \in D \wedge ngram \in ngrams(d_i)\}$.

3. *All query word vectors* $\overrightarrow{v} \in ngram$ *are averaged and translated into the document space via a tanh-activation neuron layer, such that query vector* $\overrightarrow{q}$ *is calculated as follows:*

$$\overrightarrow{q} = tanh(W \times (\frac{1}{|ngram|} \Sigma_{\overrightarrow{v} \in ngram} \overrightarrow{v}) + \overrightarrow{b}).$$

4. *A loss is calculated from the difference* $\lambda_{sim.} - cos(\overrightarrow{q}, \overrightarrow{d_{sampled}})$*. The gradient from this loss is back-propagated into the vectors* $\overrightarrow{d_{sampled}}, \overrightarrow{v} \in ngram$.

5. *Steps two through five are repeated, until the error gradient has converged sufficiently.*

Note the following interesting properties of the NVSM:

1. It is the first *unsupervised* neural IR model, and the first Deep Semantic Matching Model, to rival[1] traditional statistical IR models.

2. It learns both word and document vectors by Backpropagation during training time, *not* during inference. Therefore, it is very hard to add a new document to the corpus after training (The model would need to be re-trained).

3. It synthesizes queries by random selection of word n-grams from the documents in the corpus.

4. It discards word order information, but word co-location is still learned from the n-gram sampling.

A goal of this work is to create an unsupervised Deep Semantic Matching model that is able to generate query and document representations during *inference* time, with universal regard for word order.

[1]The DSSM partially surpassed traditional benchmarks, but fell short on others.

4. Query Embeddings from Token Relevance

4.1. Measuring Token Relevance as RNN Embedding Change

Unsupervised training of an inference-based neural document and query embedding model requires extraction of synthetic queries from the documents in the training corpus. Such synthetic queries should contain some of the most relevant phrases $P_i = \{p_1, ..., p_n\}$ from a document D_i. An optimization requirement for an unsupervised document encoder $\Phi(x)$ for Information Retrieval is then to maximize $\sum_{p \in P_i} cos(\Phi(p), \Phi(D_i))$.

The capability of RNN state to capture highly informative sentence, document and query representations has been shown previously [56]. As can be seen in figure 4.1, observation of LSTM state change has been used for keyword extraction, although for the overarching purpose of model verification, not unsupervised training [56].

KEY WORDS FOR DOCUMENT: "*shanghai hotels accommodation hotel in shanghai discount and reservation*"

	shanghai	*hotels*	*accommodation*	*hotel*	*in*	*shanghai*	*discount*	*and*	*reservation*
Number of assigned cells out of 10 Left to Right	-	**4**	3	**8**	1	**8**	**5**	3	**4**
Number of assigned cells out of 10 Right to Left	**4**	**6**	**5**	**4**	**5**	1	**7**	**5**	-

Figure 4.1.: Keywords extracted from LSTM states for sample query (Table II of [56])

One contribution of this work is definition 18 which clearly describes the step-wise RNN embedding-(state-)difference $\texttt{RNN}_{\Delta E}$ as a measure of token relevance.

Definition 18 ($\texttt{RNN}_{\Delta E}$ as a measure of token relevance)**.** *Given a sequence $(I_1, ..., I_n)$ of length n and its encoder's RNN-Cell $RNN_t : I_t, S_{t-1} \rightarrow E_t, S_t$ at timestep $1 < t < n$ with the initial state S_0, the previous state S_{t-1}, the current state S_t and the output embedding E_t at the current timestep t, the relevance of the event $I_{2<t<n}$ for the encoding E_n is measured as*

$$\texttt{RNN}_{\Delta E} = \frac{||E_t - E_{t-1}||_1}{\sum_{j=2}^{n} ||E_j - E_{j-1}||_1}$$

Note, that the token relevance is accumulated as the L1-Norm of the embedding difference. Any higher norm would be unnecessary for the use-case of this work, because the higher norm would not affect the final ordering of terms by their measured importance.

4.2. Comparing RNN Embedding Change with IDF

The most popular measure for determining the relevance of a token for a document in a corpus is IDF (see chapter 2.3.1). However, in order to extract keyphrases and not just keywords, it is insufficient to only measure the global specificity of individual tokens, specifically due to the following problems:

1. IDF will exaggerate the importance of rare synonyms. For example, a course description may contain the phrase *"trimensional bodies"* as a synonym for *"3D geometry"*. Subsequently, IDF would probably assign much more relevance to the word "trimensional", whereas the contextually more important noun "body" would be disregarded.

2. IDF has no built-in mechanism to observe relevance interactions between neighboring tokens. For example, one course from the IROM corpus called *"Understanding College and College Life"* contains the phrase "financing education". IDF disregards the importance of "financing" (frequent word), whereas $\texttt{RNN}_{\Delta E}$ realizes the unique combination. This difference is highlighted by the heatmaps in figure 4.2[1]. The heatmap shows the measured importance of tokens, with darker colors indicating greater measured importance. In this figure, it becomes clear how $\texttt{RNN}_{\Delta E}$ is more contextually aware, giving great importance to the phrases "provides overview" and "computational perspective", and spreading relevance judgments generally more evenly.

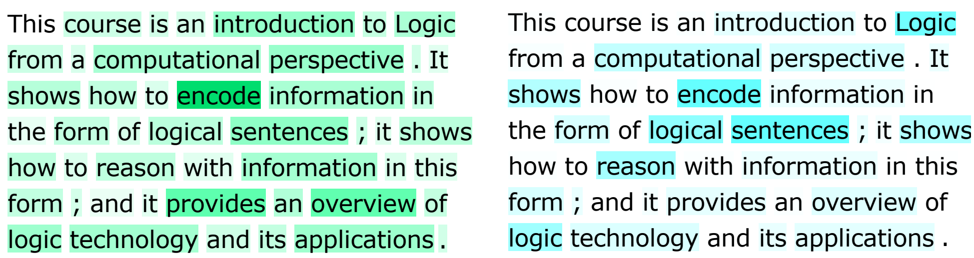


Figure 4.2.: Sample relevance heatmap comparison of $\texttt{RNN}_{\Delta E}$ (left) and IDF (right) (Graphic produced by this work).

An advantage of IDF is its linear computational uppper-bound run-time complexity. However, were it tasked with identifying keyphrases from the power-set $\mathscr{P}(T)$ of the set of tokens T, the computational complexity would become $O(|\mathscr{P}(T)|) = O(2^{|T|})$. As will be shown in the following section, $\texttt{RNN}_{\Delta E}$ in conjunction with a linear algorithm solves this problem more elegantly.

4.3. Unsupervised Query Synthesis from Token Relevance

Given a sequence of tokens `tokens` and an equally long sequence of per-token relevance measurements `relevance_per_token`, it is possible to extract keyphrases from `tokens`. For this purpose, this work proposes the *Token-Relevance-To-Queries (ToRQue)* algorithm, as described in *Python3*[2] code in appendix A.

The algorithm has linear upper-bound run-time complexity. It accumulates sequences of neighboring tokens at a maximum distance `MAX_GAP`, from the top fraction `PHRASES_FROM_TOP_FRACTION` of relevant tokens. Even if neighboring tokens are close enough to form a phrase, they will not be joined if any of the tokens in-between is contained in the set of punctuation characters `PUNCTUATION_MARKS`.

Values of `MAX_GAP=2` and `PHRASES_FROM_TOP_FRACTION=0.15` have been found to perform well. For `MAX_GAP=2`, the chosen value is a compromise between making it

[1]The heatmap generator was written as part of the Python ranking service from task three of this work.
[2]https://www.python.org

larger which could potentially cause long nonsensical phrases to be extracted, and making it smaller which might lead to phrases being broken up prematurely. The value of 0.15 for `PHRASES_FROM_TOP_FRACTION` was determined by trial and error, again as a compromise between a large value (diminishingly relevant phrases will be generated) and a smaller value (some relevant words will not be used for phrase generation). In tables 4.1 and 4.2, some sample queries which were generated with $\texttt{RNN}_{\Delta E}$ token relevance measurements from the `V-ToRQue-256` model (see chapter 6.3) with the *ToRQue* algorithm can be inspected.

One important problem becomes apparent in table 4.2: For the text of *Age of Jefferson*, the $\texttt{RNN}_{\Delta E}$ measure is consistently high for "Thomas", but then low for *"Jefferson"*, because *"Jefferson"* is not "suprising" enough for the RNN to change its output embedding too much:

Course Description Text	**Keyphrases**
(*Age of Jefferson*) This course provides an overview of Thomas Jefferson's work and perspectives presented by the University of Virginia in partnership with Thomas Jefferson's Monticello. Together, UVA and Monticello are recognized internationally as a UNESCO World Heritage Site.	Virginia partnership Thomas UNESCO Heritage Site overview Thomas
(*The Ancient Greeks*) This is a survey of ancient Greek history from the Bronze Age to the death of Socrates in 399 BCE . Along with studying the most important events and personalities , we will consider broader issues such as political and cultural values and methods of historical interpretation.	survey ancient Greek history Bronze Socrates methods studying cultural

Table 4.1.: Sample keyphrases (I) genrated by the *ToRQue* algorithm with $\texttt{RNN}_{\Delta E}$ relevance measurements from the `V-ToRQue-256` Autoencoder model.

Course Description Text	**Keyphrases**
(*America Through Foreign Eyes*) The United States has always been a source of fascination — both attraction and repulsion — for the people of France, Mexico, China, and African countries such as Nigeria, Ghana, and Sudan. "America Through Foreign Eyes" is a rich, interdisciplinary, international course that features Rice University faculty from a variety of disciplines and area studies. Focused on perceptions of America abroad, the course is a cross between World Cultures and American Studies. The course features four modules, each covering the perceptions and interactions of particular regions with America, Americans and Americanization.	perceptions America Cultures American African countries Americans Americanization Nigeria America Mexico France Rice fascination repulsion
(*Cryptography I*) Cryptography is an indispensable tool for protecting information in computer systems. In this course you will learn the inner workings of cryptographic systems and how to correctly use them in real-world applications. The course begins with a detailed discussion of how two parties who have a shared secret key can communicate securely when a powerful adversary eavesdrops and tampers with traffic. We will examine many deployed protocols and analyze mistakes in existing systems. The second half of the course discusses public-key techniques that let two parties generate a shared secret key. Throughout the course participants will be exposed to many exciting open problems in the field and work on fun (optional) programming projects. In a second course (Crypto II) we will cover more advanced cryptographic tasks such as zero-knowledge, privacy mechanisms, and other forms of encryption.	adversary eavesdrops tampers traffic inner workings cryptographic examine deployed protocols fun optional programming cryptographic tasks Crypto II forms encryption discusses techniques privacy mechanisms mistakes exciting generate begins information secret

Table 4.2.: Sample keyphrases (II) genrated by the *ToRQue* algorithm with $\texttt{RNN}_{\Delta E}$ relevance measurements from the `V-ToRQue-256` Autoencoder model.

5. Implementation

5.1. System Overview

All functionality for training, inference and benchmarking with the Deep Semantic Matching models created by this work is implemented in a single *Python3*[1] package (see UML diagram in figure 5.1). Supplemental functionality is drawn from the *Natural Language Toolkit (NLTK)*[75] for tokenization, the *Gensim*[2] library for loading pre-trained *Word2Vec* word vectors, the *WikipediaExtractor*[3] for extracting plain text paragraphs from XML-based Wikipedia dumps, and finally the *TensorFlow* [10] library for automated loss gradient computation and back-propagation of error in computation graphs.

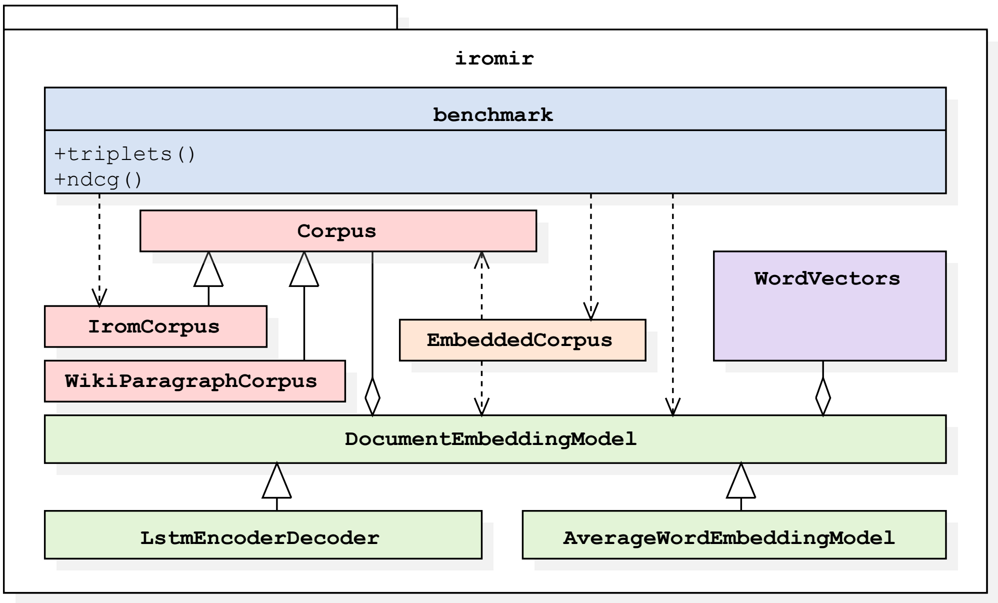


Figure 5.1.: UML diagram of `iromir` package (Graphic produced by this work).

The `iromir` package implements the `Corpus` class as an abstract collection of tokenized textual documents which can be used for unsupervised training of a `DocumentEmbeddingModel` class instance. The `EmbeddedCorpus` class represents a corpus not as a collection of documents, but as a collection of embedded document vectors. Therefore, it contains the functionality of a Vector Space Model for IR. For training of document embedding models, both the *LstmEncoderDecoder* model and the *AverageWordEmbeddingModel* require a mapping from tokens to pre-trained word vectors. The `WordVectors` class contains functionality both to load pre-trained *Word2Vec* and *GloVe* word vectors.

[1] https://www.python.org
[2] https://radimrehurek.com/gensim/
[3] https://github.com/attardi/wikiextractor

5.2. Training and Hyperparameter Search

As illustrated in figure 5.2, the hyperparameter search in this work focuses on permutations of the number of latent dimensions $k \in \{8, 64, 256\}$ (marked red) and the *Seq2Seq* flavor (direct, see chapter 3.1.5/variational, see chapter 3.1.6) (marked green). Fixed values are assumed for all other hyperparameters. These values are partly guided by insights from previous work [63–67], and partly by constraints imposed by the training hardware[4]. Most importantly, the learning rate is set to $3 * 10^{-3}$ [65], the gradient descent algorithm to *RMSProp* [62, 66] the learning rate decay factor to 0.6 per epoch, the batch size to 150 for documents and 300 for queries, the decoder input dropout to 50% [9], and KLD loss delayed by one epoch. The state size of the encoder forward and backward LSTM is set to 128, the combining LSTM's state size to 256. On the decoder side, the decoding is performed by a two-stacked LSTM with a respective state size 192. The cosine-distance loss was used on the decoder's predicted word embeddings.

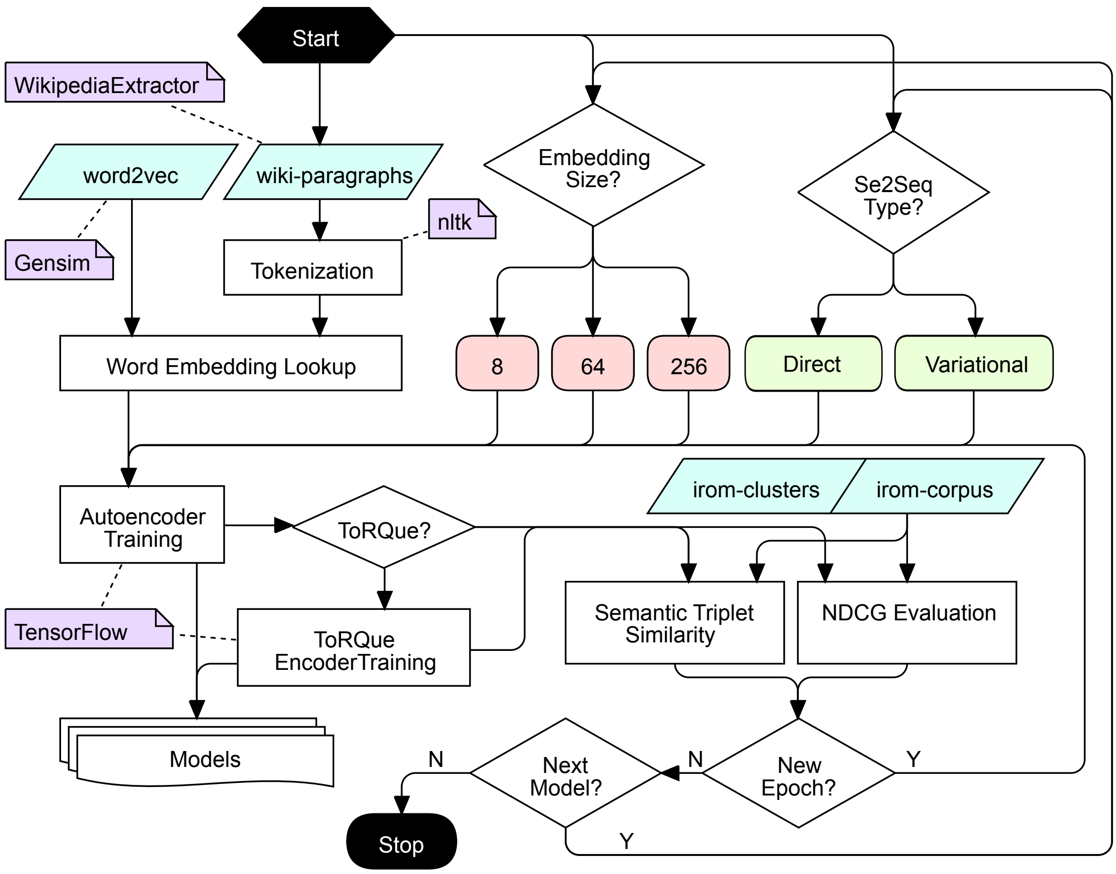


Figure 5.2.: Flowchart diagram of the ToRQue training process (Graphic produced by this work). Language resources are marked blue, third-party software violet, hyperparameters green and red.

The document and query embedding models are trained in seven epochs per model, whereas the first four epochs are used for pure Seq2Seq training, after-which the learning rate is reset and a ToRQue encoder training step succeeds each following epoch. The training is performed on a random selection of 425,286 English Wikipedia paragraphs with lengths between 1000 and 2000 characters. The paragraphs are tokenized using `nltk.word_tokenize`, and translated to pre-trained *Word2Vec* word vectors. Tests show a 75% coverage of word vectors for the words in the Wikipedia paragraphs. All other words are encoded with a special $\overrightarrow{\texttt{UNK}}$ vector.

[4]60-90 min./epoch @ 8GB GForce 1070 GPU, Intel i7 6700K CPU w/ 16GB RAM, 24 PCIe lanes

6. Evaluation

6.1. Test Setup

The trained models as described in chapter 5 were evaluated with the measures of document triplet similarity and Normalized Discounted Cumulative Gain (see chapter 2.5). As test data, a corpus of 1256 textual headline-description pairs was was used. These pairs were annotated with cluster labels from 64 categories, with an average of 20 courses per cluster. Both the corpus and cluster labels were developed by project IROM [19].

6.1.1. Semantic triplet similarity judgements

For the measure of semantic triplet similarity accuracy, 10,000 document triplets $\{(d_1, d_2, d_3)|1 \leq i \leq 10,000\}$ were randomly selected, with the criterion that d_1 and d_2 are from the same cluster, and therefore more likely to be semantically similar, while d_3 is from a different cluster. The encoder Φ_{AE} was then tasked to create latent vectors for both d_1, d_2 and d_3. Document similarity is measured as cosine distance. The result accuracy was calculated as $\frac{|\{(d_1,d_2,d_3)_i | cos(\Phi(d_1),\Phi(d_2)) < cos(\Phi(d_1),\Phi(d_3))\}|}{10,000}$. This measure mainly serves to evaluate the quality of the underlying embeddings as document encodings.

6.1.2. Normalized Discounted Cumulative Gain

On the other hand, the **Normalized Discounted Cumulative Gain** (see chapter 2.5) measure served to evaluate the quality of the neural vector spaces with respect to the ad-hoc Information Retrieval task. Ground truth query-ranking pairs were assembled as follows: A set of 1000 documents $\{d_i = (\mathtt{title}_i, \mathtt{text}_i, \mathtt{cluster}_i)|1 \leq i \leq 1000\}$ is randomly picked from the test corpus. For each selected triplet, the string $\mathtt{title}_i$ is used as a test query, for which the document d_i obviously has a ground truth relevance of 1. However, all other documents that are in the same cluster as d_i were assigned a ground truth relevance of 0.5.

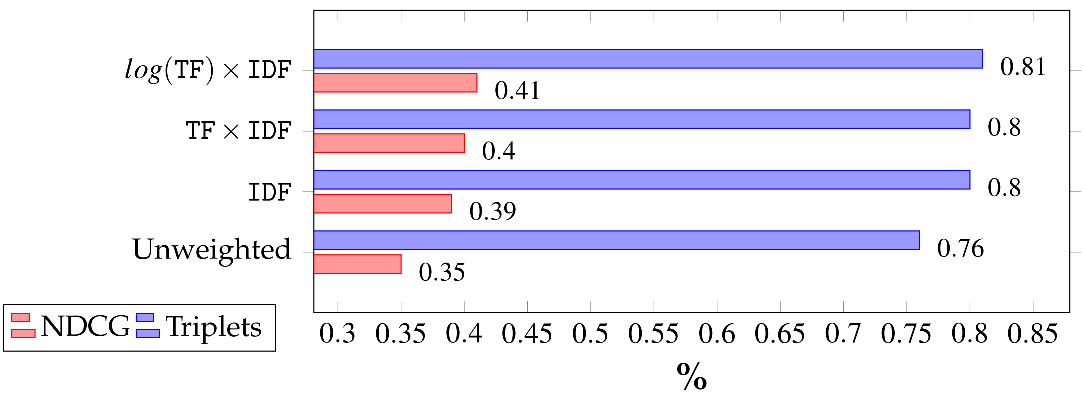


Figure 6.1.: Accuracies of the baseline Weighted Average Word Embedding Model on evaluation measures (NDCG and Triplet Similarity) with IROM data for various combinations of TF/log(TF)/IDF word vector weights.

6.2. Weighted Average Word Embedding Baseline

The baseline neural document vector space for Information Retrieval in this work is created by means of the Weighted Average Word Vector Model Φ_{avgemb} [14]. Like in the traditional Vector Space Model for Information Retrieval (see chapter 2.3), the per-word weight obviously has great influence on the final embedding. As can be seen in figure 6.1, several combinations of the term frequency $\texttt{TF}/log(\texttt{TF})$ and inverse document frequency $\texttt{IDF}$ were tried to find the optimum baseline. This baseline was finally determined to be $log(\texttt{TF}) \times \texttt{IDF}$, with a document triplet similarity judgment score of 81% and an NDCG score of 41%.

6.3. Quality of Latent Query- and Document Vectors

Figure 6.2 shows the performances of the experimental ToRQue Seq2Seq Autoencoder Models on the evaluation measures for different embedding sizes (8/64/256) and different Seq2Seq types (direct/variational, indicated by "V-" prefix). It is obvious from this figure, that the Autoencoder models in their current form strongly under-perform the baseline. However, the best of these models (V-ToRQue-256) was still useful to perform the $\texttt{RNN}_{\Delta E}$ measurements in chapter 4!

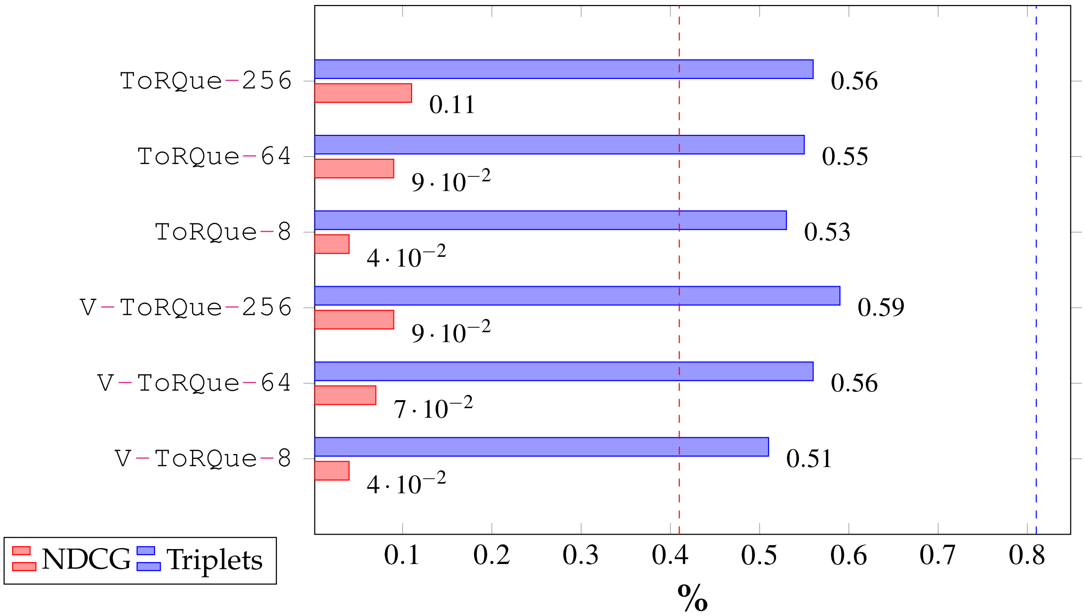


Figure 6.2.: Accuracies of the experimental ToRQue Seq2Seq Autoencoder Models on evaluation measures (NDG and Triplet Similarity) with IROM data for different embedding sizes (8/64/256) and different Seq2Seq types (direct/variational, indicated by "V-" prefix). Baselines indicated as dashed lines.

6.4. Diagnosing the Underperformance of Seq2Seq Models

After a thorough verification of the implementation, the architectural reason for the flawed benchmarks was found to be the choice of loss function: The noisy, linear loss generated by average cosine similarity between the ground truth word vectors and the reconstructed word vectors is insufficiently discrete, therefore no meaningful error gradients reach the encoder. This leads to a complete cutoff of the encoder from the decoder, where the decoder only learns the prior probability of the data, ignoring the latent posterior. The problem of ignored posterior encoder information by the decoder is also called *"posterior collapse"* [9, 46].

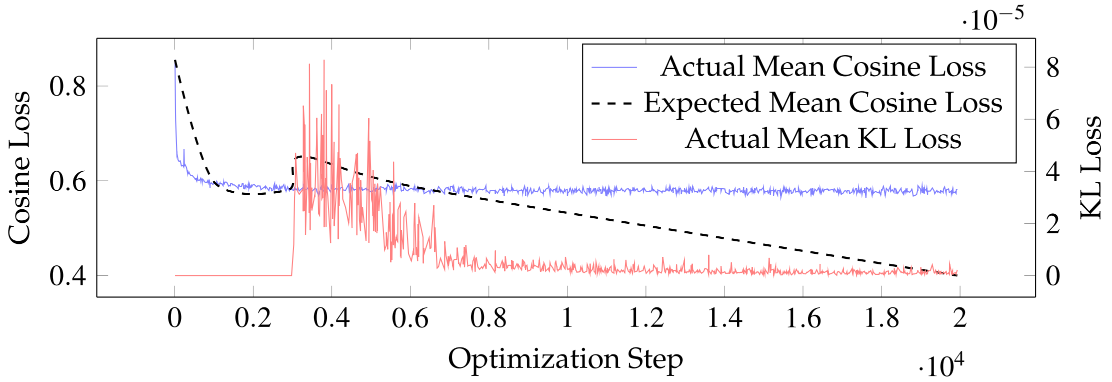


Figure 6.3.: Overlaid loss curves for KLD loss (right scale) and cosine reconstruction loss (left scale) of the V−ToRQue−256 model. Also overlaid is the expected reconstruction loss curve from figure 2 in [9]

.

This finding is supported by observations from the loss curves of the Variational Autoencoder models, as can be seen in the diagram in figure 6.3. The figure shows the KLD loss and reconstruction loss for the V−ToRQue−256. As can be observed in the figure, the KLD loss spikes at iteration 3,000. This is, because the KLD loss term is only introduced into the model after 3,000 iterations with increasing weight, to *prevent* posterior collapse (as suggested by previous work [9]). Normally, the expected behavior is for the encoder and decoder to reach an optimization point where the error gradient is insufficient to cut off the posterior, and then to introduce the noisy reparameterization. It is then expected for the reconstruction loss to briefly spike with the KLD loss. However, such a spike in reconstruction loss cannot be observed for the Autoencoder models from this work, indicating that the posterior collapse has already occurred *before* the KLD loss term is introduced, and therefore pointing to a problem with the word vector cosine similarity reconstruction loss term.

6.5. $\mathtt{RNN}_{\Delta E}$ as Word Vector Average Weight

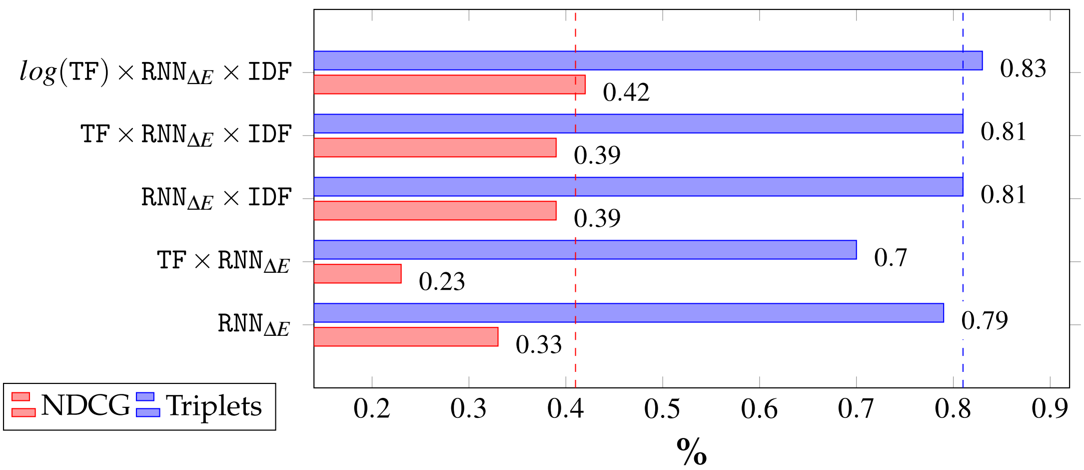


Figure 6.4.: Losses for evaluation measures (NDG and Triplet Similarity) on IROM data for various forms of $\mathtt{RNN}_{\Delta E}$ word vector weights. The baseline is indicated by the vertical dashed lines for each measure.

With two competing measures for word importance ($log(\texttt{TF}) \times \texttt{IDF}/\texttt{RNN}_{\Delta E}$), it may be interesting to compare them as weights for the Average Word Embedding Model. As can be seen in figure 6.4, such an evaluation was performed with an exhaustive list of combinations of $log(\texttt{TF})$, $\texttt{IDF}$ and $\texttt{RNN}_{\Delta E}$. Interestingly, $\texttt{RNN}_{\Delta E}$ on its own is insufficient to beat the $log(\texttt{TF}) \times \texttt{IDF}$-baseline, but performs remarkably well as a supplemental weight, such that $log(\texttt{TF}) \times \texttt{RNN}_{\Delta E} \times \texttt{IDF}$ finally beats the baseline on both measures.

Given the posterior collapse problem of the Seq2Seq models in their current form, it should be noted that the $\texttt{RNN}_{\Delta E}$ measurements from this work have probably not exhausted the full potential of the metric so far.

7. Conclusion

Using pre-trained word vectors as optimization labels for deep recurrent Autoencoder architectures is problematic, since it reliably leads to posterior collapse, even for non-variational architectures, as displayed by the findings mentioned in section 6.

However, the step-wise encoder state change $\mathtt{RNN}_{\Delta E}$ was found to be a useful source of information to extract keyphrases via the *ToRQue* algorithm, which could then be used for unsupervised training of an encoder for both queries and documents.

With regard to the Autoencoder architectures, it is strongly suggested that euclidean coordinates such as word vectors perform poorly as labels, because they induce noisy loss gradients. Therefore, a reconstruction loss term must be introduced which uses distinct class probability labels (possibly a Word2Vec/GloVe pos./neg. softmax, or one-hot term vectors).

Furthermore, it should be stressed that most hyperparameters were assumed constant in this work (learning rate, learning rate decay, loss function, batch size, dropout, encoder/decoder network size, gradient descent optimization method). Additionally, the artifical title-cluster-retrieval task on IROM data may not be optimal to measure the true performance of an IR system.

Due to these obviously unexplored directions for future work, no conclusive judgment can yet be made on the hypothesis that Seq2Seq Autoencoder architectures may have the capacity to generate latent spaces for queries and documents which solve the shortcomings (loss of word order information and extreme dimensionality) that are inherent to traditional IR models.

As a result of this work, a framework for both productive application and future research of Neural Autoencoder IR models has been implemented: The `iromir` module, in particular the `AverageDocumentEmbeddingModel`, serves as a search engine backend for the IROM search engine[1]. The `benchmark` module and the abstract `Corpus` and `DocumentEmbeddingModel` classes may ease the implementation and testing of models in future work.

"Optimism is a strategy for making a better future. Because unless you believe that the future can be better, you are unlikely to step up and take responsibility for making it so."

— Noam Chomsky

[1]https://irom.pms.ifi.lmu.de

8. Future Work

The first direction of future work concerns architectural shortcomings of the models which were implemented so far, in particular the problem of posterior collapse when using embedded word vectors as decoder labels. The problem might be resolved, by replacing the cosine loss with cross-entropy loss, and replacing the Word2Vec-based word embeddings with probability-distribution-based word feature vectors (like one-hot encoded term vectors, or concatenated softmaxes over positive and negative components in Word2Vec/GloVe vectors).

Furthermore, an exhaustive hyperparameter search should be conducted to optimize the Seq2Seq Autoencoder models, with development data from actual IR corpora such as *ClueWeb09* or *Robust04*. It should also be explored, whether domain adaption has any desirable effects on a Seq2Seq Autoencoder.

However, this would still require the IR system to be designed around a fixed set of top terms. A more radical approach may be to attempt a purely character-based Autoencoder system, which is more robust against potentially unknown terms. It should be explored, how the $\texttt{RNN}_{\Delta E}$ measure could be applied to sequences of characters instead of sequences of terms.

Another direction concerns the use of the recurrent LSTM Autoencoder architecture itself. In particular with respect to keyphrase-extraction, it may be beneficial to use convolutional networks [76] or a grammatically inspired *Recursive Tree-LSTM* [77], which is the representation model of the current state-of-the-art Sentiment Analysis architecture. In that case as well, the definition of $\texttt{RNN}_{\Delta E}$ would need to be adjusted to fit the changed neural architecture.

Generally, it should be explored how $\texttt{RNN}_{\Delta E}$ can be used to determine not just the relevance of words or characters, but also the relevance of larger textual units such as sentences or paragraphs. It should be noted, that $\texttt{RNN}_{\Delta E}$ can measure the importance of a sequencial unit for whatever the decoder is conditioned to reconstruct - that may not just be content, but could also be sentiment or style.

Additionally, the relevance heatmap algorithm from this work could be added to the `iromir` module to highlight the parts of a retrieved document which have high cosine similarity with a user query (such a visualization is also called a *"document-query interaction heatmap"*), to assist the user in assessing the true relevance of a document with respect to her query.

"Begin at the beginning," the King said gravely,
"and go on till you come to the end: then stop."
— Lewis Carroll, *Alice in Wonderland*

Appendix

A. The Token-Relevance-To-Queries (ToRQue) Algorithm

```
MAX_GAP=2                           # Maximum gap between tokens that
                                    # will be combined.
PUNCTUATION_MARKS=set(".!?,;:")     # Characters which break up phrases,
                                    # even if tokens are close enough.
PHRASES_FROM_TOP_FRACTION=0.15      # Fraction of most important tokens
                                    # that will be used for phrase generation.

def token_relevance_to_queries(relevance_per_token):
    global PUNCTUATION_MARKS, MAX_GAP, PHRASES_FROM_TOP_FRACTION
    relevance_per_token = enumerate(relevance_per_token)

    # The 'relevance_per_token' argument is now a list of 2-2-tuples like
    # (index, (token, relevance)). The following function returns the rel-
    # evance term from such a tuple, unless the token is punctuation.
    def fn_relevance_for_id_tok_rel_tuple(idx_tok_rel):
        index, (token, rel) = idx_tok_rel
        if token[-1] in PUNCTUATION_MARKS:
            return 0
        return rel

    # This algorithm extracts the *positions* of the most important
    # tokens, and stores them in a set for fast contains-lookup.
    top_n_token_pos = set(
        idx_tok_rel[0] for idx_tok_rel in sorted(
            relevance_per_token,
            key=fn_relevance_for_id_tok_rel_tuple,
            reverse=True
        )[:len(relevance_per_token)*PHRASES_FROM_TOP_FRACTION])

    result_phrases = defaultdict(float)
    prev_top_token_pos = 0
    current_phrase = []
    current_relevance = .0

    # Loop over all token positions and create phrases on-the-fly
    for t, (term, attention) in relevance_per_token:
        is_top_term = t in top_n_token_pos
        is_punctuation = term[-1] in PUNCTUATION_MARKS
        is_gap_too_large = t - prev_top_token_pos > MAX_GAP

        # Check whether current phrase should be terminated
        if ((is_top_term and is_gap_too_large) or is_punctuation) and \
            current_phrase:

            result_phrases[tuple(current_phrase)] = max(
                result_phrases[tuple(current_phrase)],
                current_relevance)
            current_phrase = []
            current_relevance = .0

        # Append current term to current phrase if it is important
        if is_top_term:
            current_phrase.append(term)
            current_relevance += attention
            prev_top_token_pos = t

    # Make sure that the last current phrase is added
    if current_phrase:
        result_phrases[tuple(current_phrase)] = max(
            result_phrases[tuple(current_phrase)],
            current_relevance)

    return result_phrases
```

Bibliography

[1] Y. Zhang, M. M. Rahman, A. Braylan, B. Dang, H. Chang, H. Kim, Q. McNamara, A. Angert, E. Banner, V. Khetan, T. McDonnell, A. T. Nguyen, D. Xu, B. C. Wallace, and M. Lease, "Neural Information Retrieval: A Literature Review," *CoRR*, vol. abs/1611.06792, 2016.

[2] K. D. Onal, I. S. Altingovde, P. Karagoz, and M. de Rijke, "Getting Started with Neural Models for Semantic Matching in Web Search," *CoRR*, vol. abs/1611.03305, 2016.

[3] B. Mitra and N. Craswell, "Neural Models for Information Retrieval," *CoRR*, vol. abs/1705.01509, 2017.

[4] K. S. Jones, S. Walker, and S. Robertson, "A probabilistic model of information retrieval: development and comparative experiments: Part 1," *Information Processing & Management*, vol. 36, no. 6, pp. 779 – 808, 2000.

[5] S. Robertson, "How Okapi came to TREC," *Voorhees and Harman (2005)*, pp. 287–299, 2005.

[6] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[7] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *CoRR*, vol. abs/1406.1078, 2014.

[8] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 3104–3112, Curran Associates, Inc., 2014.

[9] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Józefowicz, and S. Bengio, "Generating Sentences from a Continuous Space," *CoRR*, vol. abs/1511.06349, 2015.

[10] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," *CoRR*, vol. abs/1603.04467, 2016.

[11] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *CoRR*, vol. abs/1301.3781, 2013.

[12] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global Vectors for Word Representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.

[13] R. Bellman, "Combinatorial processes and dynamic programming," tech. rep., Feb. 1958.

[14] S. Clinchant and F. Perronnin, "Aggregating continuous word embeddings for information retrieval," in *Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality*, pp. 100–109, 2013.

[15] K. Järvelin and J. Kekäläinen, "Cumulated Gain-based Evaluation of IR Techniques," *ACM Transactions on Information Systems*, vol. 20, pp. 422–446, Oct. 2002.

[16] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hul-

lender, “Learning to Rank Using Gradient Descent,” in *Proceedings of the 22Nd International Conference on Machine Learning*, ICML ’05, (New York, NY, USA), pp. 89–96, ACM, 2005.

[17] Y. Wang, L. Wang, Y. Li, D. He, T. Liu, and W. Chen, “A Theoretical Analysis of NDCG Type Ranking Measures,” *CoRR*, vol. abs/1304.6480, 2013.

[18] A. M. Dai, C. Olah, and Q. V. Le, “Document Embedding with Paragraph Vectors,” *CoRR*, vol. abs/1507.07998, 2015.

[19] K.-H. Wilker, “Multidimensional Clustering of MOOC Offers – Applying unsupervised learning algorithms FCM and SOM to text descriptions.” Bachelorarbeit/Bachelor Thesis, 2016.

[20] C. S. Mooers, “Coding, Information Retrieval, and the Rapid Selector,” *Journal of the American Society for Information Science*, vol. 1, no. 4, p. 225, 1950.

[21] H. P. Luhn, “A Statistical Approach to Mechanized Encoding and Searching of Literary Information,” *IBM Journal of Research and Development*, vol. 1, pp. 309–317, Oct 1957.

[22] J. P. McMurray, “The bureau of ships rapid selector system,” *American Documentation*, vol. 13, no. 1, pp. 66–68, 1962.

[23] G. Salton, *The SMART Retrieval System—Experiments in Automatic Document Processing*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1971.

[24] S. E. Robertson and K. S. Jones, “Relevance weighting of search terms,” *Journal of the American Society for Information Science*, vol. 27, no. 3, pp. 129–146, 1976.

[25] C. J. V. Rijsbergen, *Information Retrieval*. Newton, MA, USA: Butterworth-Heinemann, 2nd ed., 1979.

[26] E. A. Fox, G. L. Nunn, and W. C. Lee, “Coefficients of Combining Concept Classes in a Collection,” in *Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR ’88, (New York, NY, USA), pp. 291–307, ACM, 1988.

[27] H. Turtle, “Natural Language vs. Boolean Query Evaluation: A Comparison of Retrieval Performance,” in *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR ’94, (New York, NY, USA), pp. 212–220, Springer-Verlag New York, Inc., 1994.

[28] A. Moffat and J. Zobel, “Self-indexing Inverted Files for Fast Text Retrieval,” *ACM Transactions on Information Systems*, vol. 14, pp. 349–379, Oct. 1996.

[29] J. Zobel and A. Moffat, “Inverted Files for Text Search Engines,” *ACM Comput. Surv.*, vol. 38, July 2006.

[30] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, “Indexing by latent semantic analysis,” *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE*, vol. 41, no. 6, pp. 391–407, 1990.

[31] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent dirichlet allocation,” *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[32] D. R. H. Miller, T. Leek, and R. M. Schwartz, “A Hidden Markov Model Information Retrieval System,” in *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR ’99, (New York, NY, USA), pp. 214–221, ACM, 1999.

[33] D. Hiemstra, *A Linguistically Motivated Probabilistic Model of Information Retrieval*, pp. 569–584. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998.

[34] F. Song and W. B. Croft, “A General Language Model for Information Retrieval,” in *Proceedings of the Eighth International Conference on Information and Knowledge Management*, CIKM ’99, (New York, NY, USA), pp. 316–321, ACM, 1999.

[35] D. Hiemstra and W. Kraaij, *A Language Modeling Approach to TREC*, pp. 373–396. Digital Libraries and Electronic Publishing, MIT Press, 9 2005. Imported from DB

PMS [db-utwente:inbo:0000003574].

[36] C. Manning, P. Rhagavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[37] G. E. Hinton, "Connectionist learning procedures," *Artificial Intelligence*, vol. 40, no. 1, pp. 185 – 234, 1989.

[38] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Neurocomputing: Foundations of Research," ch. Learning Representations by Back-propagating Errors, pp. 696–699, Cambridge, MA, USA: MIT Press, 1988.

[39] Z. S. Harris, "Distributional Structure," *WORD*, vol. 10, no. 2-3, pp. 146–162, 1954.

[40] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, pp. 1550–1560, Oct 1990.

[41] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur, "Recurrent neural network based language model," in *Proceedings of the 11th Annual Conference of the International Speech Communication Association, INTERSPEECH 2010*, vol. 2, pp. 1045–1048, 01 2010.

[42] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[43] Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies," 03 2003.

[44] Q. V. Le and T. Mikolov, "Distributed Representations of Sentences and Documents," *CoRR*, vol. abs/1405.4053, 2014.

[45] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes.," *CoRR*, vol. abs/1312.6114, 2013.

[46] O. Fabius and J. R. van Amersfoort, "Variational Recurrent Auto-Encoders," *ArXiv e-prints*, Dec. 2014.

[47] S. Semeniuta, A. Severyn, and E. Barth, "A Hybrid Convolutional Variational Autoencoder for Text Generation," *CoRR*, vol. abs/1702.02390, 2017.

[48] X. Zhu and E. Grefenstette, "Deep Learning for Semantic Composition," *Proceedings of ACL 2017, Tutorial Abstracts*, pp. 6–7, 2017.

[49] C. Manning, "Understanding Human Language: Can NLP and Deep Learning Help?," in *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, (New York, NY, USA), pp. 1–1, ACM, 2016.

[50] N. Craswell, W. B. Croft, J. Guo, B. Mitra, and M. de Rijke, "Report on the SIGIR 2016 Workshop on Neural Information Retrieval (Neu-IR)," *SIGIR Forum*, vol. 50, pp. 96–103, Feb. 2017.

[51] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck, "Learning Deep Structured Semantic Models for Web Search Using Clickthrough Data," in *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management*, CIKM '13, (New York, NY, USA), pp. 2333–2338, ACM, 2013.

[52] B. Hu, Z. Lu, H. Li, and Q. Chen, "Convolutional Neural Network Architectures for Matching Natural Language Sentences," *CoRR*, vol. abs/1503.03244, 2015.

[53] L. Pang, Y. Lan, J. Guo, J. Xu, and X. Cheng, "A Study of MatchPyramid Models on Ad-hoc Retrieval," *CoRR*, vol. abs/1606.04648, 2016.

[54] Q. Ai, L. Yang, J. Guo, and W. B. Croft, "Analysis of the Paragraph Vector Model for Information Retrieval," in *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval*, ICTIR '16, (New York, NY, USA), pp. 133–142, ACM, 2016.

[55] J. Guo, Y. Fan, Q. Ai, and W. B. Croft, "A Deep Relevance Matching Model for Ad-hoc Retrieval," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, (New York, NY, USA), pp. 55–64, ACM,

2016.

[56] H. Palangi, L. Deng, Y. Shen, J. Gao, X. He, J. Chen, X. Song, and R. Ward, "Deep Sentence Embedding Using Long Short-term Memory Networks: Analysis and Application to Information Retrieval," *IEEE/ACM Transactions on Audio, Speech and Language Proc.*, vol. 24, pp. 694–707, Apr. 2016.

[57] A. Jaech, H. Kamisetty, E. K. Ringger, and C. Clarke, "Match-Tensor: a Deep Relevance Model for Search," *CoRR*, vol. abs/1701.07795, 2017.

[58] C. Van Gysel, M. de Rijke, and E. Kanoulas, "Neural Vector Spaces for Unsupervised Information Retrieval," *ArXiv e-prints*, Aug. 2017.

[59] D. Cohen, Q. Ai, and W. B. Croft, "Adaptability of Neural Networks on Varying Granularity IR Tasks," *CoRR*, vol. abs/1606.07565, 2016.

[60] R. Mihalcea and P. Tarau, "TextRank: Bringing Order into Text," 2004.

[61] R. Meng, S. Zhao, S. Han, D. He, P. Brusilovsky, and Y. Chi, "Deep Keyphrase Generation," *CoRR*, vol. abs/1704.06879, 2017.

[62] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[63] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *CoRR*, vol. abs/1312.6199, 2013.

[64] M. Hermans and B. Schrauwen, "Training and Analysing Deep Recurrent Neural Networks," in *Advances in Neural Information Processing Systems 26* (C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds.), pp. 190–198, Curran Associates, Inc., 2013.

[65] A. Karpathy, J. Johnson, and F. Li, "Visualizing and Understanding Recurrent Networks," *CoRR*, vol. abs/1506.02078, 2015.

[66] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016.

[67] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, pp. 2222–2232, Oct 2017.

[68] K. Janocha and W. M. Czarnecki, "On Loss Functions for Deep Neural Networks in Classification," *CoRR*, vol. abs/1702.05659, 2017.

[69] G. Boole, *The mathematical analysis of logic*. Philosophical Library, 1847.

[70] T. Bayes, "An essay towards solving a problem in the doctrine of chances," 1763.

[71] T. Berners-Lee, D. Dimitroyannis, A. J. Mallinckrodt, S. McKay, *et al.*, "World Wide Web," *Computers in Physics*, vol. 8, no. 3, pp. 298–299, 1994.

[72] D. Kriesel, *A Brief Introduction to Neural Networks*. 2007.

[73] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural Networks*, vol. 18, no. 5, pp. 602 – 610, 2005. IJCNN 2005.

[74] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[75] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[76] N. T. Vu, H. Adel, P. Gupta, and H. Schütze, "Combining Recurrent and Convolutional Neural Networks for Relation Classification," *CoRR*, vol. abs/1605.07333, 2016.

[77] K. S. Tai, R. Socher, and C. D. Manning, "Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks," *CoRR*, vol. abs/1503.00075, 2015.

List of Figures

List of Definitions

List of Tables