Master Thesis

Design of a document retrieval system using Transformer-based models and a domain specific ontology

Emanuele Mottola

Academic Supervisors
Dr. Antonio Vetrò
Prof. Juan Carlos De Martin
Dr. Giuseppe Futia

Company Supervisors
M.Sc. Hans Ehm
M.Sc. Nour Ramzy

Final Report for the Thesis
Master in Software Engineering

Department of Automatic and Informatics (DAUIN)
Polytechnic University of Turin
Turin, Italy

23 October 2020

Design of a document retrieval system using Transformer-based models and a domain specific ontology

Author
Emanuele Mottola

Academic Supervisors
Dr. Antonio Vetrò
Prof. Juan Carlos De Martin
Dr. Giuseppe Futia

Company Supervisors
M.Sc. Hans Ehm
M.Sc. Nour Ramzy

Abstract

The scientific literature and internal research documents every institution produces is a key source of information for the members of the institution itself. To access this material effectively and to retrieve the information needed going beyond the keyword-based approach, a Transformer-based language model tailored on the semiconductor supply chain domain is employed together with the same domain ontology – the Digital Reference [1] – to build a document retrieval system over the pool of documents of the Infineon Corporate Supply Chain Innovation department. The further pre-training of the Bidirectional Encoder Representations from Transformers (BERT) model [2] on a text corpus based on the semiconductor supply chain literature is used to empower SentenceBERT [3] for sentence embeddings creation. Measuring the similarity score between the embedding representation of the query and the sentence embeddings related to the documents, the system is able to retrieve relevant documents to the query posed by the user. With the same mechanism, the classes of the Digital Reference are annotated, resulting in an ontology populated with documents that are shown to the user according to the match between query keywords and class names. The first results of the system are presented, where the F-measure reaches 0.58 and the mean Average Precision 0.45.

Diversamente dalle Leggi
che contraddistinguono la Natura,
in Spirito, non solo nulla si distrugge,
ma tutto va creandosi...
fino alla Rinascita e alla Rivoluzione.

Acknowledgment

I would like to thank Infineon Technologies AG, in particular Hans Ehm and Nour Ramzy for giving me the possibility to write my master thesis on such an interesting topic and for the support they gave me during the work.

I thank my university supervisors Dr. Giuseppe Futia and Dr. Antonio Vetrò for the advice and fundamental suggestions that helped me to address the path of this thesis.

Huge thanks go to the historical friends, the ones I met during high school and who have remained by my side for all these years. Thanks to Mattia, Marcella, Sara and Alessandro: there have been many evenings, afternoons, holidays spent growing together and, I hope, there will be many.

Heartfelt thanks to Roberto, for the genuineness of his friendship and the positivity he has always transmitted to me.

Special thanks go to Sabino. Thank you for our endless walks that gave the rhythm to our talks, and for never having stopped being my friend. To the always deep and sincere exchanges.

I want to thank my friends at UPGRADESTOC***O for the energy they have always been able to transmit to me. Thanks, in alphabetical order, to Andrea, Fabio, Roberto and Sofia, with whom I well understood the value of carefree.

Thanks to all those who have been part of that year out of any time which was the Erasmus. Thanks to Michele, Salvo, Luca, Diego, Alessio, Alessandro and Matteo, for making the experience in Karlsruhe unforgettable.

Thanks to Pellegrino Delfino, a fundamental friend throughout the journey made so far. His hunger for culture and his critical thinking are an inexhaustible source of inspiration for me. To the Knowledge.

A deep thanks to Naomi, for the invaluable support she has given me during these months. Passion and maturity are in her the magnificent traits of Beauty and Truth that help me to eliminate obstacles and move the limits further and further away. To the Beauty and the Truth.

Thanks to Umberto, for giving me the love of curiosity.

Finally, my heartfelt thanks go to my family. In particular to my brother Sergio and my sister Antonella who have always supported me, to Cristian and my nephews Giulio, Mattia and Andrea, and my mother, Eugenia, whose trust and understanding, combined with her sensitivity, are the best gift I could receive.

Ringraziamenti

Vorrei ringraziare Infineon Technologies AG, in particolare Hans Ehm e Nour Ramzy per avermi dato la possibilità di scrivere la mia tesi di laurea su un argomento così interessante e il supporto che mi hanno dato durante il lavoro.

Ringrazio i miei relatori universitari Dr. Giuseppe Futia e Dr. Antonio Vetrò per i consigli e suggerimenti fondamentali che mi hanno aiutato ad affrontare il percorso di questa tesi.

Un enorme grazie va agli amici storici, quelli che ho conosciuto durante il liceo e che sono rimasti al mio fianco per tutti questi anni. Grazie a Mattia, Marcella, Sara e Alessandro: tante son state le serate, i pomeriggi, le vacanze passate a crescere insieme e, spero, tante ne saranno.

Grazie di cuore a Roberto, per la genuinità della sua amicizia e la positività che mi ha sempre trasmesso.

Un ringraziamento speciale va a Sabino. Grazie per le nostre infinite camminate che davano il ritmo ai nostri discorsi, e per non aver mai smesso di essermi amico. Agli scambi sempre profondi e sinceri.

Voglio ringraziare i miei amici di UPGRADESTOC***O per l'energia che sono sempre stati in grado di trasmettermi. Grazie, in ordine alfabetico, ad Andrea, Fabio, Roberto e Sofia, con cui ho capito per bene il valore della spensieratezza.

Grazie a tutti coloro che hanno fatto parte di quell'anno fuori da qualsiasi tempo che è stato l'Erasmus. Grazie a Michele, Salvo, Luca, Diego, Alessio, Alessandro e Matteo, per aver reso indimenticabile l'esperienza a Karlsruhe.

Grazie a Pellegrino Delfino, un amico fondamentale per tutto il percorso fatto fin qui. La sua fame di cultura e il suo pensiero critico sono per me fonte inesauribile di ispirazione. Alla Conoscenza.

Un profondo grazie a Naomi, per l'inestimabile supporto che mi ha dato durante questi mesi. La passione e la maturità sono in lei i magnifici tratti di Bellezza e Verità che mi aiutano ad annullare gli ostacoli e spostare i limiti sempre più lontano. Alla Bellezza e alla Verità.

Grazie ad Umberto, per avermi trasmesso l'amore per la curiosità.

Infine, il grazie più sentito va alla mia famiglia. In particolare a mio fratello Sergio e mia sorella Antonella, che mi hanno sempre sostenuto, a Cristian e ai miei nipoti Giulio, Mattia e Andrea e a mia mamma, Eugenia, la cui fiducia e comprensione, uniti alla sua sensibilità, sono il più bel regalo che potessi ricevere.

Contents

1 Introduction 16

1.1 Motivation 16

1.2 Objective and Research Questions 16

1.3 Structure of the Thesis 17

2 Literature Review 19

2.1 Natural Language Processing 19

2.1.1 Neural Language Processing 21

2.1.2 Recurrent Neural Networks 23

2.1.3 The Transformer Architecture 28

2.1.4 BERT 31

2.2 Semantic Web 35

2.2.1 The Semantic Web stack 35

2.2.2 The Semantic Web at Infineon - the Digital Reference 36

2.2.3 Semantic Search 37

3 Methodology 38

3.1 Offline model preparation 38

3.1.1 BERT domain specific pre-training 40

3.1.2 Sentence-transformer 43

3.1.3 Annotation of Digital Reference's classes 44

3.2 Online architecture: the document retrieval system 47

3.2.1 Architecture design 47

3.2.2 The presentation tier 48

3.2.3 The application tier 49

3.2.4 The data tier 51

4 Evaluation 52

4.1 Evaluation 52

4.1.1 Metrics 52

4.1.2 Experiment 54

4.1.3 Results 54

4.1.4 Discussion 56

5 Conclusions 58

5.1 Summary 58

5.2 Outlook 59

A Appendix **60**
A.1 Sample of xml document 60
A.2 Server Response example 61

B Appendix **63**
B.1 Application tier 63

List of Figures

2.1 Tasks of NLP and NLU [12] 20
2.2 Word2Vec. CBOW and Skip-gram implementations. Redrawn based on [29] 23
2.3 Basic structure of a Recurrent Neural Network. Redrawn based on [10] 24
2.4 Illustration of Back Propagation Through Time (BPTT) 25
2.5 LSTM architecture. Redrawn by [50] based on [48] 26
2.6 Gated Recurrent Unit Architecture. Redrawn based on [49] 27
2.7 Illustration of Encoder and Decoder architectures. Re-drawn based on [10] 28
2.8 InferSent architecture [52] 29
2.9 The transformer architecture [55] 30
2.10 BERT, pre-training and fine-tuning [2] 32
2.11 SBERT architecture, with classification objective function [3] 34
2.12 The Semantic Web stack [67] 36

3.1 Pipeline for the conversion from PDF to JSON 39
3.2 Sentence tokenization of the content of the JSON files. 40
3.3 Representation of the training procedure on the specific domain. . . . 42
3.4 Sentence BERT empowered by the domain specific BERT. 44
3.5 Mapping of every sentence of the documents to the corresponding sentence embeddings using domain specific SentenceBERT. 44
3.6 Relations of the Organizational Unit class. 45
3.7 Triple representing a labeled class. 46
3.8 Architecture of the document retrieval system. 48
3.9 Example of the document retrieval system graphical interface. 49
3.10 Entity relationship model representing the database design. 51

List of Tables

3.1 Language model evaluation results. 42
3.2 Sentence-BERT: Training on Multi-NLI dataset, STSBenchmark evaluation 43
3.3 Sentence-BERT: Training on Multi-NLI dataset and STS dataset . . 43
3.4 Top-5 documents attached to the Organizational Unit class with rdfs:seeAlso 46

4.1 Queries defined together with the domain expert, according to the topics of interest of the department. 55
4.2 Golden standard defined together with the domain expert, according to the topics of interest of the department. 55
4.3 Results from the document retrieval system, according to the queries outlined in Table 4.1. 55
4.4 Results from the document retrieval system, according to the queries outlined in Table 4.1. 56
4.5 Summary of the results obtained running the queries described in Table 4.1. 56

Chapter 1

Introduction

1.1 Motivation

The amount of data available on the Internet and produced by private users, companies and public infrastructures is immense. It is estimated that in 2018 the volume of information produced was 33 Zettabytes and will grow to 175 Zettabytes in 2025 [4].

Besides the number of documents companies publish on the Web, most of the resources they produce remain private, since these might contain information of great interest for competitors or are simply regulated by privacy concerns. However, internal employees might be interested in browsing those resources, in order to access the information needed. In case the number of documents is high and not perfectly organized, the amount of time required to look for a single piece of information might increase, leading to a loss in the efficiency of usage of the available documents.

To speed up the process of retrievement of the needed information, and since the greatest part of the data available is unstructured and in the form of text, it is necessary to exploit a discipline – Natural Language Processing (NLP) – able to analyze and process the huge quantity of text that would be impossible for a human to examine in reasonable time [5]. For these reasons, we will design a document retrieval system that uses natural language processing techniques together with a domain specific ontology to retrieve documents relevant to a given query.

In the following section, the objectives and the research questions this thesis aims to reach and answer will be outlined.

1.2 Objective and Research Questions

Infineon Technologies AG is a semiconductor company whose products are addressed to be part of systems belonging to different industries, where the major challenges undertaken are related to energy efficiency, mobility and security [6]. In particular, the Corporate Supply Chain Innovation Department studies and researches over topics strictly related to the highly dynamic and volatile nature of semiconductor supply chains and supply chains containing semiconductors. In this context, a high number of papers, PhD's, Master's and Bachelor's thesis and confidential documents are produced, whose content is tailored to the domain of interest of the department and accessibility is a key point of interest. For these reasons, we would like to build a document retrieval system that uses NLP techniques customized to the specific domain we are dealing with. This brings us to the first research question:

- **RQ1: How can we tailor NLP techniques to a specific domain and use it to in the design of a document retrieval system?**

Moreover, the Semantic web technologies [7] are taking momentum in the last years, with a great interest from both academia and companies in the usage of this technologies to organize and structure the data they produce and gather. A further objective of this thesis is to include the domain specific ontology created within the Infineon Corporate Supply Chain Innovation department – the Digital Reference [1] – in the document retrieval system and see how the documents can be organized and accessed according to the data structure already existing. Hence, the second research question is:

- **RQ2: How can Natural Language Processing be used to annotate classes of a domain specific ontology with documents coming from a domain specific literature and use it in a document retrieval system?**

To sum up, in order to allow the users to retrieve the documents and the information they need as precise as possible, this thesis is focused on the design of a document retrieval system that combines the advantages of both the state of the art of Natural Language Processing techniques tailored to the specific domain and Semantic Web technologies using the domain specific ontology – the Digital Reference.

1.3 Structure of the Thesis

This chapter was mainly reserved to give an introduction to the purpose of the thesis.

Chapter 2 introduces the historical background of the Natural Language Processing techniques and their application. Moreover, we will introduce Neural Language processing and the architecture at the base of those that represent the current state of the art in many tasks – BERT [2]. In this Chapter, the Semantic Web stack [8] and the Digital Reference [1] will be presented as well.

Chapter 3 describes the implementation techniques adopted in order to build the document retrieval system. The architecture design is articulated into two phases: the *offline* and the *online*. The work carried out offline is needed to prepare the data and the system. Then, the online architecture is presented, with the description of the working principles that allow the document retrieval system to accept a query and return the documents that answer it.

Chapter 4 outlines the evaluation techniques and presents the results of the system compared to the golden standard defined by an expert of the Corporate Supply

Chain Innovation department.

Finally, Chapter 5 shows the conclusions and the future work, according to the most recent development in the field.

Chapter 2

Literature Review

2.1 Natural Language Processing

NLP is a speciality born from the intersection of Artificial Intelligence, Computer Science, and Linguistics which gathers together a wide range of techniques and theories to analyze and process of human language at different levels [9]. It is a broad research field exploring both written and spoken language, concerning its generation and understanding. Indeed, Natural Language Generation and Natural Language Understanding are the two main sub-fields of Natural Language Processing [9].

Natural Language Generation (NLG) is the branch of NLP that deals with the formation of human understandable content out of a concept representation [9, 10]. On the other hand, *Natural Language Understanding* (NLU) handles the creation of that concept representation to be used by NLG, using a text sequence as input. Many tasks like text summarization, question answering in chatbots and machine translation exploit this mechanism [10].

More in detail, NLU aims at the comprehension of human texts or speeches as a human being could do, finding for example the significance of figures of speech, solving ambiguities and polysemy, understanding the presence of synonyms and co-reference [11]. The core of Natural Language Understanding is *semantic analysis*, which is the process of analysis of text and speech in order to extract the correct sense and significance out of the input. To achieve it, the algorithm should be able to work at different levels [11]:

- *syntax* - the ability to understand the grammar.
- *semantics* - the ability to understand the literal meaning.
- *pragmantic* - the ability to understand the final purpose.

According to [9], a system capable of Natural Language Understanding would also be able to perform different tasks, like text paraphrase, question answering, and knowledge inference out of the processed text. Other tasks are shown in Figure 2.1

NLP has a wide range of use cases. It empowers the personal assistants Apple Siri, Amazon Alexa, IBM Watson and Google Assistant [13]. It is used to find out the opinions about a product - task called *sentiment analysis* - in order to derive marketing information out of them [14]. It can be also exploited to support financial traders, with systems able to mine information and produce forecast out

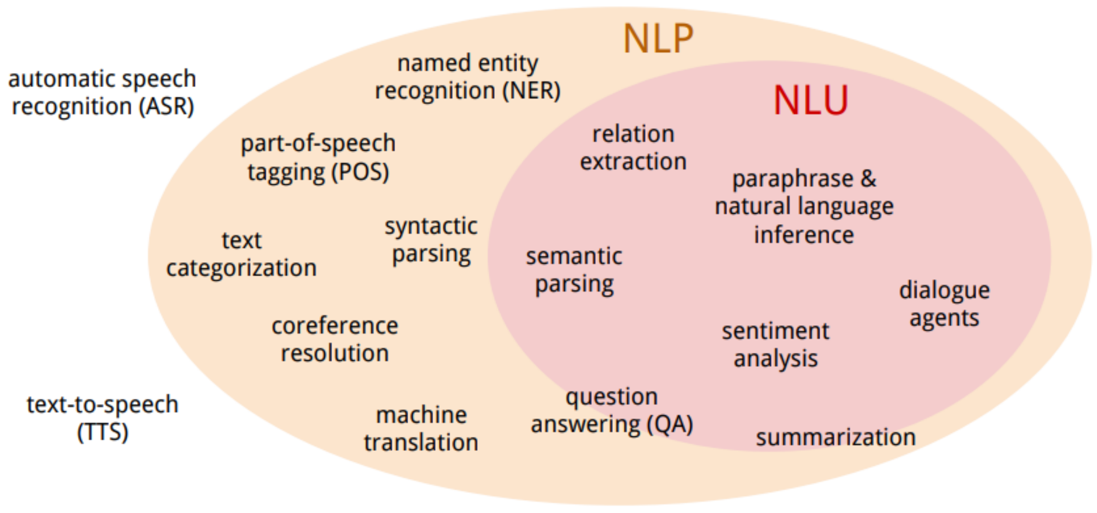


Figure 2.1: Tasks of NLP and NLU [12]

of the news [15]. To summarize, the use-case panorama is vast. In this thesis, we will apply NLP techniques – together with a domain specific ontology – to create a *document retrieval system*, whose aim is to find out relevant documents to a user's query [16].

In the following paragraph we will give a brief introduction to the history of NLP, in order to introduce the modern aspects of the discipline.

Brief history

NLP aroused in the 1950s from the intersection between Linguistics and Artificial Intelligence [17]. The former used to analyze the language to gain an accurate representation of its structure and formalism: the major investigation areas concerned the grammatical and syntactical aspects. On the other hand, the latter treated the study of the language as a communication problem between human and machine: many efforts were addressed for the creation of knowledge representation out of natural language corpora [11].
A significant contribution came from the field of formal languages, with the publication of *Syntactic Structures* by Chomsky in the 1957 [18]. This *symbolic* approach had generative grammars and parser at the core of its theory: given a sentence, the parser had to understand whether the sentence matched with the rules of the grammar. This work influenced the further creation of regular and context free grammars, which led - together with lexical analyzers (lexers) - to the development of the first compilers for programming languages. However, this approach could not live the complexity of human language up and solve ambiguity issues [17]. Consequently, other directions were investigated. In 1972, Winograd [19] built a system able to simulate the movement of some blocks according to natural language commands, while Wood [20] developed LUNAR, a system used to access a database with natural language. The two systems combined and integrated all the aspects of the language – syntax, semantics and inference [11, 21]. Even if the results were remarkable, these systems were not general purpose and accomplished a simple task. [21, 22].

At the beginning of 1980s, Linguistics and Artificial Intelligence began to converge. In particular, the creation of the knowledge representation took also into account the formalism described by linguistic and, at the same time, linguists taped into knowledge representations drawn by computer scientists. It is in this period that *Computational Linguistic* or *Natural Language Processing* actually came to life and got boosted [11]. At this stage, systems with hand-crafted rules - like the one proposed by Chomsky - were not considered prominent anymore.
Therefore, new approaches were introduced. IBM started applying *statistical methods* to process speech data and model the pronunciation of words [21]. The underlying idea is that the structure could be understood by investigating the distributional composition of the language elements [23]. Hence, given enough training data, the algorithm should learn based on the statistics embedded in it. A typical example of this approach is the determination of the etymology of names. As described in [23], the system calculates the probability that a name belongs to a specific language based on the pool of known names of that language. Then, the word is attached to the language for which the probability is maximized. These new concepts revolutionized the field of NLP and are still today at its core.
During the 1990s, statistical NLP gained a great popularity and got applied to Part-of-Speech Tagging, which is a task focused on labeling the different components of a sentence with the corresponding syntactical meaning. Here, the usage of Markov models led to satisfactory results for *Word Sense Disambiguation* [24] and *prepositional phrase attachment* [25], too. Further applications of Statistical NLP were machine translation, clustering and text classification [26]. Another important field of application of statistical NLP is the *Language Modeling* task, which consists in the prediction of the next word given the previous part of the sentence [26]. For this task, Bengio et al. [27] applied for the first time a neural network in 2003. This was the first milestone of the modern Neural Language Processing. In the next section, we will introduce its fundamentals, with the aim to present the different techniques used to create the a proper representation of word and sentences and how this is used to score the similarity between the user's query and the content of a document.

2.1.1 Neural Language Processing

The explosion of utilization of deep learning techniques began in 2010-2011 [22]. In these and the next years speech recognition reached a maturity that allowed the deployment of the first models in the industry, giving the propulsion for a broader usage of deep learning algorithms in other NLP areas. In 2016, Google and Microsoft announced the usage of neural networks for their machine translation systems. Facebook did the same almost one year later. To conclude, neural networks led to a real revolution in the field of NLP, with visible progress in many sub-disciplines [22].

Word and Sentence embeddings

The creation of a proper word representation is key to push machines as close as possible to human language understanding. Therefore, the development of algorithms able to map words to their vector representations – the *word embeddings* – gained a central role. According to [10], the neural network community approached the creation of word embeddings using the *distributed representation* conception. It consists in the creation of vectors whose values are assigned based on the context the word appears in. In agreement with this concept, the meaning, the similarity and the difference between two words emerge from the comparison among the dimensions of their vector representations [10].

In 2013, Mikolov and his colleagues [28, 29, 30] introduced a new algorithm for the creation of word embeddings: Word2Vec. It is a neural network based algorithm, declined in two fashions. Both cases use a shallow neural network originally trained for the creation of a neural language model then modified for efficiency purposes. As previously stated, language modelling was one of the first instances where neural networks were used in NLP. Bengio et al. [27] defines language modelling as the task of learning the distribution of words in a language in order to predict a word given the previous ones. It is consequently calculated according to this formula:

$$p(w_1^T) = \prod_{t=1}^{T} p(w_t|w_1^{t-1})$$

where T represents the number of words.
In the first architecture of Mikolov – the Continuous Bag Of Word model (CBOW) – the context C of a target word w_t is used to predict this last one. Since pure text cannot be analyzed by a neural network, the words are encoded with their *one-hot representation*. Hence, given a vocabulary $V = \{w_1, w_2, ..., w_t, ..., w_{|V|}\}$ each word is represented by a vector $1 \times |V|$ where $|V|$ is the cardinality of the vocabulary V and every different vector contains an element set to 1 and the others to 0. These rudimentary word embeddings are fed into the neural network. Here, the vector representations of the context are summed up in the hidden layer and then used to compute the probability distribution over V. The objective is to maximize the probability of w_t – the target word – over all the words of the vocabulary. Experimental results show that best performance was reached exploiting $C = \{w_{t+j}\}, -4 \leq j \leq +4, j \neq 0$, namely using the four words preceding and the four following the target word.
The second implementation – the Skip-Gram model – trains a neural network "to find word representations that are useful for predicting the surrounding words in a sentence or a document" [28]. Hence, the architecture is made of a log-linear classifier to predict words happening in a given context C. Both with the first and the second implementation, the embeddings are given by the rows of the weight matrix of the hidden layer, which are iteratively updated via *back-propagation* [31]. Both the implementations show to be scalable and resilient to unknown words, with reasonable memory and computational requirements [10].

Although Word2Vec was widely employed in many applications, like in the representation of biological sequences [32] and the vector representation of documents (Doc2Vec [33]), it was not the only framework for word embeddings creation. GloVe [34] - which stands for GlobalVectors - was developed at Standford in 2014. The model is able to seize the global statistics out of the text corpus analyzed and reflects them in the word embeddings. In fact, a co-occurrence matrix X is defined - where every element $X_{i,j}$ indicates the number of times the word w_j occurs into the context of the word w_i [34] - and then exploited to optimize an objective function. The objective function aims to make the dot product between every word embedding equal to the log probability of the words' co-occurrence. This lead to the following equation:

$$w_i \cdot w_j + b_i + b_j = logX_{ij}$$

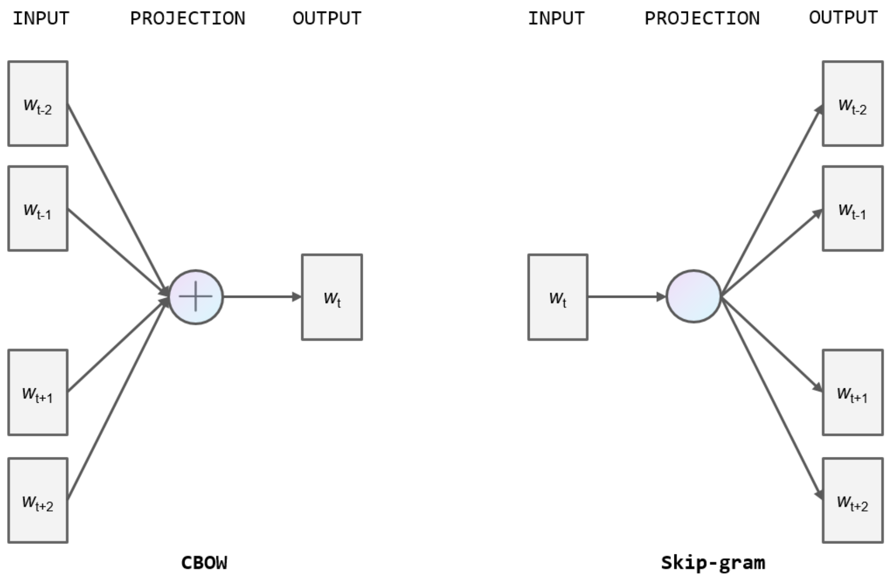


Figure 2.2: Word2Vec. CBOW and Skip-gram implementations. Redrawn based on [29]

where b_i and b_j are bias vectors. The resulting word embeddings are particularly suitable for tasks like semantic similarity and analogy, and used in many others, like intent detection [35] and recommendation systems [36].

Word2Vec and GloVe word representations were also applied to the computation of sentence embeddings: the SIF [37] model, for example, calculates the weighted average of the word embeddings and removes from each representation the component common to all the sentence embeddings. Another is GEM [38], which – considered the geometrical space created by the pool of word embeddings and analyzed its structure – infers novel orthogonal basis carrying the semantic meaning of the sentence. These systems led to quite promising results in semantic similarity for semantic search tasks, showing to be a good trade-off between performance and computational requirements. In the following paragraph, we introduce the Recurrent Neural Network architecture - able to capture the sequential relations between words composing a sentence - and the sentence representations they derive.

2.1.2 Recurrent Neural Networks

The Recurrent Neural Networks (RNNs) [39, 40] were introduced in the late 1980s to handle sequential input. Different applications like speech recognition, text summarization, text translation and document classification share a key characteristic: the input is presented as a sequence of symbols, where each word is dependent on the previous and conditions the following. According to [41], a sequence of vectors is given in the form $x = (x_1, ..., x_t, ..., x_{T_x})$ where $T_x \in \mathbb{N}^+$ and represents the length of the input sequence, and $0 \leq t < T_x$. Every element $x_t \in \mathbb{R}^{d_{in}}$, namely has dimension d_{in}. The RNN models a function that takes in input an element of the sequence x at every time-step and returns the vector $\hat{y}_t \in \mathbb{R}^{d_{out}}$, member of the output sequence $\hat{y} = (\hat{y}_1, ..., \hat{y}_t, ..., y_{T_x})$ [10]. Hence, as described in [42], the RNN is defined as a feed-forward neural network tailored to the sequential input, where the hidden state of each time-step is forwarded to the neural network of the following time-step through an edge. Therefore, at every time t, the input x_t is combined with the hidden state of the previous time step h_{t-1} to produce h_t, which is then used to output $\hat{y}_t$. Then, the current hidden state is passed to the hidden state at $t+1$ [41, 10]. To calculate h_t, the general formula is:

$$h_t = g(W_{hx}x_t + W_{hh}h_{t-1} + b_h)$$

where W_{hx} is the weight matrix of the input, W_{hh} is the weight matrix applied to the hidden state of the previous time *t - 1* and b_h is the bias vector. To compute the output, instead we use:

$$\hat{y} = f(W_{yh}h_t + b_h)$$

The pseudo-code of the algorithm is shown in the following code snippet [43]:

Algorithm 1 Forward pass RNN

```
1: for t from 1 to T do
2:     $u_t \leftarrow W_{hx}x_t + W_{hh}h_{t-1}$
3:     $h_t \leftarrow g(u_t)$
4:     $o_t \leftarrow W_{yh}h_t + b_h$
5:     $y^t \leftarrow f(o_t)$
6: end for
```

where $f(\cdot)$ and $g(\cdot)$ are non-linear functions.
The unrolled structure of the RNN is depicted in Figure 2.3.

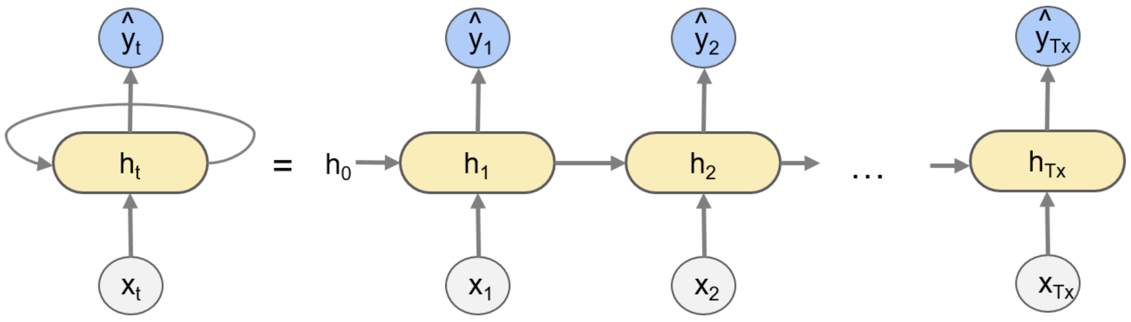


Figure 2.3: Basic structure of a Recurrent Neural Network. Redrawn based on [10]

Back Propagation Through Time (BPTT)

The mechanism adopted to train a RNN is slightly different from the one used for feed forward neural networks. Back Propagation Through Time was introduced by Rumelhart et al. [44] and Werbos et al. [45] and further deepened by subsequent researches [40, 39]. The objective of the training is to optimize the matrices W_{hx}, W_{hh} and W_{yh} in order to minimize the error function $E(y, \hat{y})$, where $y \in \mathbb{R}^{d_{out}}$ is the target value and $\hat{y} \in \mathbb{R}^{d_{out}}$ is the prediction. The error is computed as the sum of the of the errors in the time-step. Hence, the first step for the computation of the BPTT is the unrolling of the network and the consideration of only one time-step at the time, like in the following algorithm [43]:

This procedure allows a tuning of the values of the matrices involved as parameters of the network, leading to a reduction of the error function $E(y, \hat{y})$. In the next paragraph, the limits of this architecture are outlined.

Algorithm 2 Back Propagation Through Time

```
1: for t from T downto 1 do
2:     $do_t \leftarrow f'(o_t) \cdot dE(\hat{y}_t, y_t)/dy_t$
3:     $db_o \leftarrow db_o + do_t$
4:     $dW_{yh} \leftarrow dW_{yh} + do_t {h_t}^T$
5:     $dh_t \leftarrow dh_t + W_{yh}^T do_t$
6:     $dy_t \leftarrow g'(y_t) \cdot dh_t$
7:     $dW_{hx} \leftarrow dW_{hx} + dy_t {x_t}^T$
8:     $db_h \leftarrow db_h + dy_t$
9:     $dW_{hh} \leftarrow dW_{hh} + dyt {h_{t-1}}^T$
10:    $dh_{t-1} \leftarrow {W_{hh}}^T dy_t$
11: end for
12: return $[dW_{hx}, dW_{hh}, dW_{yh}, db_h, db_o, dh_0]$
```

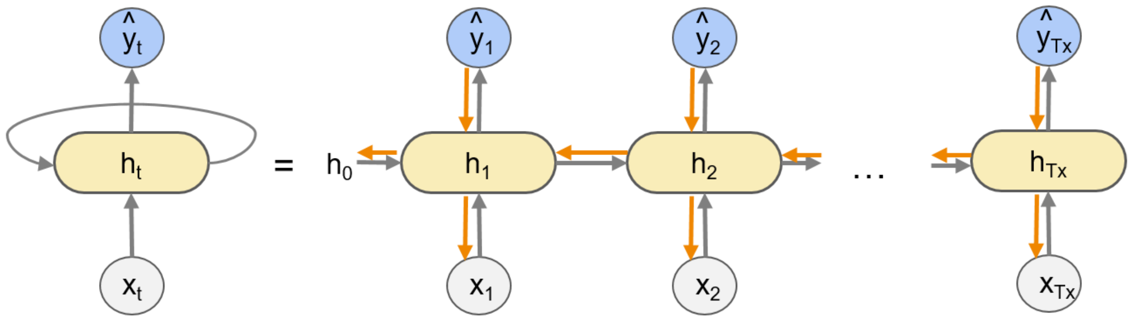


Figure 2.4: Illustration of Back Propagation Through Time (BPTT)

Problems training the RNN

The basic architecture of the RNNs – also called *vanilla RNN* – affected negatively the ability of the network to learn long distance dependencies between the elements of the sequence, as described by Bengio et al. [46] in 1994 and further studied by Pascanu et al. [47]. In particular, if the matrices involved in the calculation of the gradient have small values - less then 1 - the resulting gradients of the deeper layers will tend to 0, leading to the so called *vanishing gradient* problem. On the other hand, if the value of the matrices is greater then 1, the gradient will tend to infinite, also called *exploding gradient* problem.

In the following paragraph, we will introduce two architectures based on "memory" and "forget" cells to overcome this issue. In particular, Hochreiter et al. [48] handled the vanishing problem through the Long Short-Term Memory (LSTM) architecture, while Cho et al. [49] addressed the same issue with the Gated Recurrent Unit (GRU).

LSTM

Long Short-Term Memory [48] is an architecture based on the concept of gate and cell state. The intuition is that the cell state carries the important information down all the network - like a memory state - allowing the network to model dependencies between part of the input sequence which are distant, too. The forget and update gates remove or insert information in the cell state. The general structure of the

LSTM is shown in Figure 2.5.

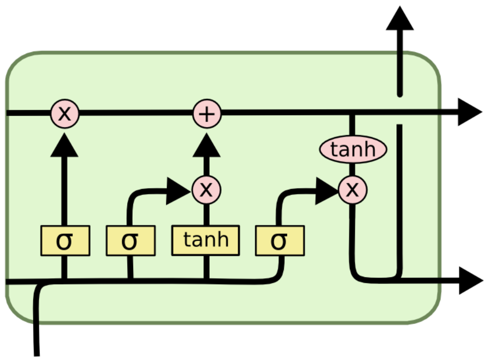


Figure 2.5: LSTM architecture. Redrawn by [50] based on [48]

The first operation is the concatenation of the hidden state h_{t-1} with the current input element x_t. Then the *forget gate* comes into play: a sigmoid (σ) function is applied, returning a value between 0 and 1. The closer to 0, the more the network will forget that information.

$$\Gamma_f = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

Next, the concatenation between h_{t-1} and x_t is further given in input both to the *tanh* and sigmoid function, the former used for the regularization of the network, the latter to decide which information to use to update the cell state. The two values are then multiplied. This is the so called *update gate*.

$$\Gamma_u = \sigma(W_u[h_{t-1}, x_t] + b_u)$$

$$\widetilde{c}_u = \sigma(W_c[h_{t-1}, x_t] + b_c)$$

Hence, the cell state is finally updated and the hidden state of the current time-step computed:

$$c_t = \Gamma_u \cdot \widetilde{c}_t + \Gamma_f \cdot c_{t-1}$$
$$h_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \cdot tanh(c_t)$$

GRU

As mentioned in the previous section, Gated Recurrent Unit (GRU) [49] is the other architecture able to propagate important information through the network. Like LSTMs, it is based on a system of gates, but it gets rid of the cell state, and propagates the information only using the hidden state. The architecture is shown in Figure 2.6.

The concatenation of the input and hidden state is fed into a sigmoid function, then multiplied for the hidden state. This acts like an update gate.

$$r_t = \sigma(W_u[h_{t-1}, x^{<t>}] + b_u)$$

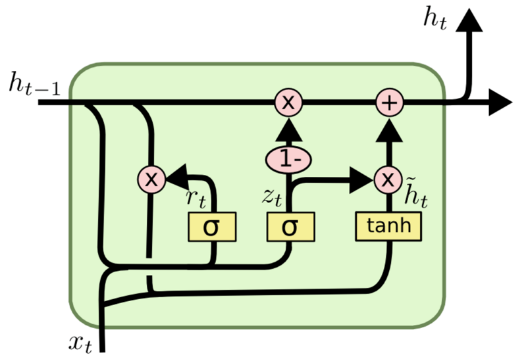


Figure 2.6: Gated Recurrent Unit Architecture. Redrawn based on [49]

Next, the concatenation of hidden and input state are processed by a sigmoid function and subtracted to 1. Instead, this part of the network has the role to forget information which is not considered important - it is called the *reset gate*. Hence, the matrix of weights used for this operation is different from the one previously presented in the definition of the update gate function.

$$z_t = \sigma(W_r[h_{t-1}, x_t] + b_r)$$

At this point, the reset gate comes into play and will actually influence the information from previous time-steps to be kept or not. Therefore the calculation of the current memory content is:

$$\widetilde{h}_t = tanh(W_c[r_t \cdot h_{t-1}, x_t] + b_c)$$

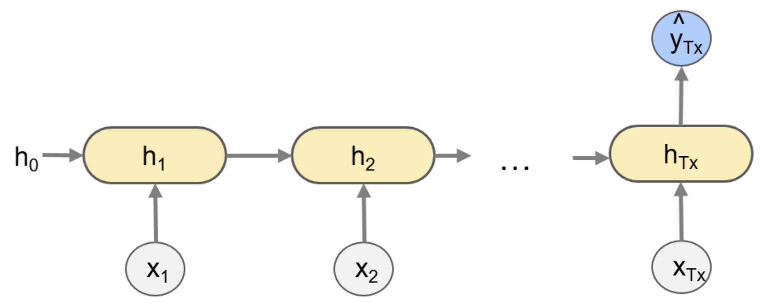
As last step, the final value of the hidden state at the current time-step is calculated as:

$$h_t = z_t \cdot \widetilde{h}_t + (1 - z_t) \cdot h_{t-1}$$

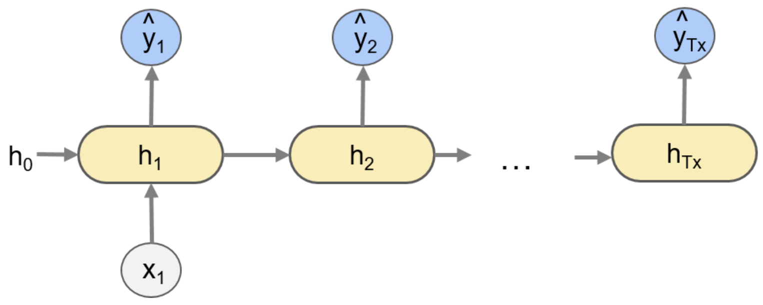
Parameterized Sentence Embeddings

The architectures presented in the previous section lead to notable improvements in the creation of sentence embeddings. In particular, Skip-Thought [51] allowed the creation of sentence embeddings that worked well also in practice. The approach used an encoder-decoder architecture, which provided good results in neural machine translation. An *encoder* is a many-to-one neural network able to condense the information coming from the words of a sentence in a vector representation. The core blocks of this neural network can be either the vanilla RNN, LSTM or GRU. Figure 2.7(a) is a visual representation of this concept. On the other hand, a *decoder* (Figure2.7(b)) is a one-to-many architecture that - given in input a vector representation - is able to output a meaningful sequence of words. Likewise the encoder, different architectures of the core blocks can be exploited.

The encoder-decoder architecture used by Skip-Thought aimed to predict the previous and following sentences given the current one. More precisely, it utilized a GRU encoder to build the vector representation of the current sentence, and two separate decoders to predict the previous and following sentences. The core blocks of the decoders were GRU units slightly modified, with the sentence vector used to bias the update gate, reset gate and hidden state [51].



(a) Encoder



(b) Decoder

Figure 2.7: Illustration of Encoder and Decoder architectures. Re-drawn based on [10]

InferSent [52] is the other parameterized approach that employes the Bidirectional Recurrent Neural Network with LSTM core blocks, where bidirectional indicates that the input sequence information propagates both from left to right and right to left. The architecture is of the type many-to-one - namely an encoder - where given in input a sentence, the vector representation is given in output. The model is trained for the sentence inference task, which consists in understanding whether a given couple of sentences is an entailment, a contradiction none of them. The architecture used is a Siamese network (Figure 2.8), where each sentence is encoded to the corresponding vector representation. Then, a feed-forward neural network is applied to these representations to output the relation between the sentences. The resulting embeddings showed to be competitive for semantic similarity tasks, scoring 75.8% in Pearson correlation [should put the definition of Pearson? Maybe create an Appendix?] on the STS benchmark dataset [53].

In the next section, we will introduce the Transformer architecture, a completely new structure that puts apart the sequential architecture of Recurrent Neural Networks.

2.1.3 The Transformer Architecture

The encoder-decoder architecture showed its limitations in machine translation task when seizing long range relationships in a sequence. According to Bahdanau et al. [54] this was mainly due to the usage of a fixed encoder vector when decoding to a target sentence, while the authors underlined the importance that the decoder focuses selectively on the part of the source sequence that is currently being translated.

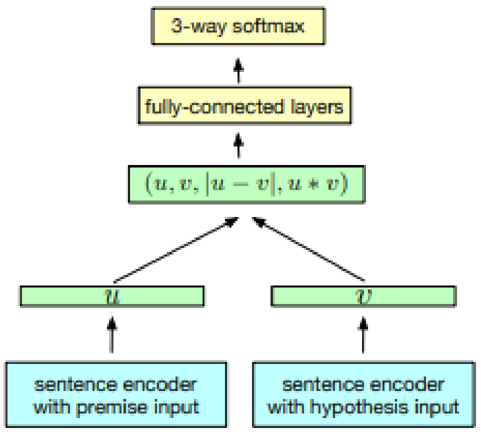

3-way softmax
fully-connected layers
$(u, v, |u-v|, u*v)$
u
v
sentence encoder with premise input
sentence encoder with hypothesis input

Figure 2.8: InferSent architecture [52]

The attention mechanism

The approach proposed by Bahdanau et al. is called *attention mechanism* and is implemented in an architecture that consists of a bidirectional encoder and a decoder that "jointly aligns and translates"[54]. The novelty of the architecture is composed by the decoder, indeed.

The main idea is to have a dynamic vector representation of the source sentence – called context vector c_i – which is a weighted sum of the hidden states of the encoder. Essentially, this reflects a selective focus over the source words in order to output the current target word. Therefore, during every time-step i, the decoder will use a context vector which is different from the context vector at time-step $t-1$ to output the target word. In other words, the decoder can search for part of the source sentence x that it thinks are relevant to predict the target work y_i. The context vector to produce the target word y_i is given by:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

where the weight of each encoder hidden state α_{ij} is computed as:

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})}$$

and

$$e_{ij} = a(h_{i-1}, h_j)$$

corresponds to the *alignment model*. Here, the hidden state of the decoder h_{i-1} is combined with every hidden state of the encoder h_j , in order to understand how much an input at position j is related to the output at position i. Consequently, α_{ij} represents the probability the output word y_j matches the input word x_i.This mechanism allowed the network to capture even long range dependencies, and to outperform the previous state-of-the-art in machine translation [54]. Furthermore, this approach was the founding idea of a novel architecture that represents a milestone in Natural Language Processing: the Transformer.

The Transformer

The Transformer [55] is an architecture that heavily exploits the attention mechanism described in the previous section. The algorithm puts the recurrent neural networks aside and comes to a parallelizable configuration that speeds up the training procedure, while also notably improving the quality of translations [55].

As for most of the architectures presented in the previous sections for machine translation, also the Transformer uses an *encoder-decoder* structure, with the encoder that maps the input sentence to a vector representation, and the decoder the reconstructs the sentence in another language, using the embedding produced by the encoder. This design is reported in Figure 2.9, with the encoder on the left and the decoder on the right.

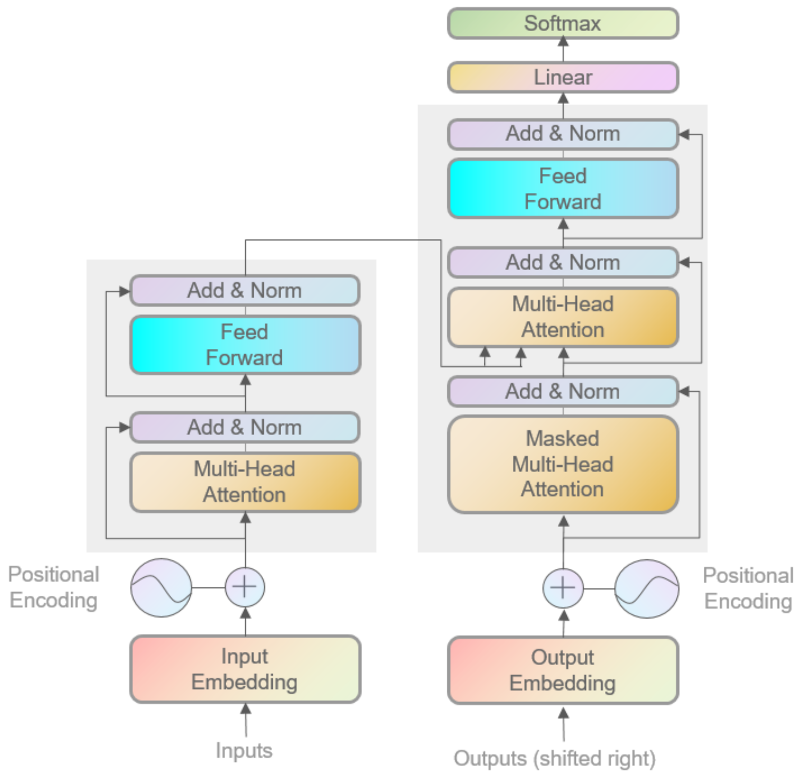


Figure 2.9: The transformer architecture [55]

According to the paper *Attention is all you need* [55], the *encoder* is made of six stacked indistinguishable layers. Each of these is made of two sub-layers:

- the *multi-head self-attention layer*
- a fully connected feed-forward layer

that we explain in the following paragraphs. Moreover, each sub-layer is followed by a residual and normalization layer, which normalizes the sum of the input of the sub-layer with the output of the sub-layer itself. On the other hand, in the six stacked twinned layers of the *decoder* each of them contains three sub-layers. In particular, in addiction to the couple of sub-layers identical to the ones in the encoder, there is a third one: it is a *masked multi-head self-attention* layer, which ensures that the predictions for a target word only depend on the previous words. Likewise the encoder, each decoder sub-layer is followed by a residual and normalization layer.

Multi-Head Self-Attention The novelty of the approach consists in the way the attention mechanism works. Every word embedding generates *key* K, *query* Q and *value* V vectors of d_k dimension, thanks to the multiplication of the embedding with three different matrices W^Q, W^K and W^V randomly initialized. The aim of the attention is the mapping of the *value* vector V to the output, using a combination of *keys* K and *query* Q to infer the amount of the weight. Hence, the attention is computed: the query is multiplied by all the keys, divided by $\sqrt{d_k}$ and then a *softmax* function is applied. In this way, the weight vector is computed. The multiplication of it for the *value* vector gives in output the *Scaled Dot-Product Attention* matrix. The operation is summarized as:

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

In order to gain the representation of the input sequence in different sub-spaces [55], the computation of the attention is raised $h = 8$ times, every time with different Q, V and K due to the random intialization of the matrices W^Q, W^K and W^V. This also allows the system to focus on different positions of the sequence, balancing the importance of words in the sentence. The output of each head is then concatenated with the others and projected once again.

$$MultiHead(Q,K,V) = Concat(head_1, ..., head_h)W^O$$

where

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

and $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_k}$ and $W^O \in \mathbb{R}^{hd_{model} \times d_k}$. The nature of this system – with its multi-heads – implies a high degree of parallelization, since each head can be computed independently from the others. Furthermore, this architecture is also able to seize long-range dependencies: in fact, the ability to perform well with this task depends on the length of the forward and backward paths the information has to cover in the network and the Transformer limits these lengths, also thanks to the parallelized architecture.

Among the variety of algorithms that exploit the Transformer architecture like ElMo [56] and OpenAI-GPT [57], in the next section we present BERT [2], the current state-of-the-art in NLU.

2.1.4 BERT

In the previous sections we have seen how the analysis of sequences with Recurrent Neural networks did not apprehend the long range dependencies between words in sequence, due to the long paths of the forward and backward pass of the signals [55]. The transformer is the architecture that overcame this issue, reducing the length of the pipeline and showing a greater ability to gather the information from the surrounding words to create a vector representation of a target word. Unfortunately, it did not show the flexibility necessary to address a wide range of NLP tasks. Together with a new training procedure, this is the great value added by the BERT model [2].

The Architecture

As described in [2], the implementation of the BERT system is essentially based on the encoder framework of the Transformer, with the same working principles, that for conciseness will not be reported here again. In particular, the authors created two main models, $BERT_{BASE}$ – with a number of stacked encoders $L = 12$, the hidden dimension $H = 768$, and the number of self-attention heads $A = 12$ – and $BERT_{LARGE}$ that has $L = 24$, $H = 1024$ and $A = 16$. The input sequence is initially mapped to a vector representation using the WordPiece tokenization [58] and its vocabulary. Every sequence is pre-poned with a [CLS] token and each sentence in a pair is separated by the [SEP] token. In fact, BERT is both able to handle sentence and word related tasks, as we will show in the next paragraph. The output of the [CLS] token is $C \in \mathbb{R}^H$, and the vector of the *i-th* word is represented by $T_i \in \mathbb{R}^H$ [2].The system is build around two main steps: *pre-training* and *fine-tuning*, like shown in Figure 2.10.

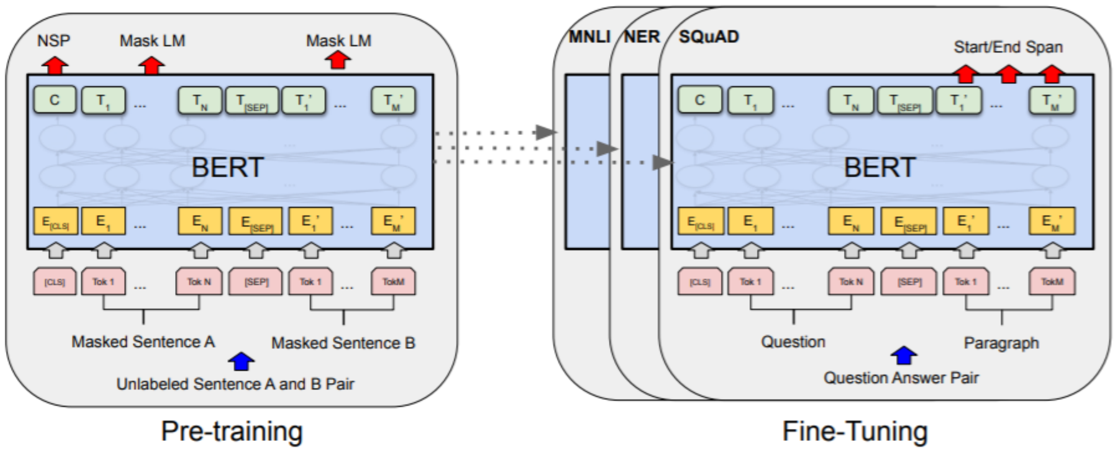


Figure 2.10: BERT, pre-training and fine-tuning [2]

Pre-training The pre-training procedure was already introduced by other architectures [56, 57], which scored the previously state-of-the-art in different NLP task. This procedure consists in training the network before the architecture is actually used in other objectives. It allows the system to learn the language model of the data it is trained on. Then, when fine-tuning for a given purpose, this knowledge is further exploited to initialize the system.

BERT pre-trains over two main NLP task: Masked Language Modelling and Next Sentence Prediction tasks. The Masked Language Modelling task was carried out exploiting a bi-directional training. In particular, this consists in randomly masking the 15% of the text corpus with the [MASK] token. In order to avoid inconsistencies with the fine-tuning procedure, where the mask token would not be used, given the 15% of the tokens to be replaced, only 80% where actually masked with [MASK], while 10% were replaced with a random token and the remaining 10% left unchanged. A cross-entropy function was used [reference in appendix], and the embedding T_i returned for the *i-th* token. The Next sentence prediction task aims at predicting whether a sentence is the sentence coming immediately after a given one. The dataset for this pre-training procedure can be built with any text corpus: given a sentence pair, 50% of the time the second sentence is the actual following sentence – labeled as *isNext* – the other 50% a random sentence from the corpus – labeled as *NotNext* [2]. This procedure is very effective for Question Answering and Natural Language Inference tasks.

Fine-tuning The fine-tuning procedure is a training procedure for one of the downstream tasks such as Question Answering, Natural Language Inference and Text Classification. Compared to the pre-training task, the fine-tuning is straightforward, since for every fine-tuning task it is only necessary to use inputs and outputs and tune the parameters [2]. An example of fine-tuning procedure is given by the sentence-pair completion task. The main dataset is the Situations With Adversarial Generations (SWAG) [59], that is composed of entries were for each sentence there are four options representing plausible continuations. The goal is to pick most suitable. To fine-tune on this dataset, four sentence-pairs are created, each made of the given sentence and one of the possible options. Then, they are fed into BERT, and the dot-product of the output token C with a task specific vector returns the likelihood that pair is the best sentence-pair. In this task, $BERT_{LARGE}$ outperforms ELMo and OpenAI-GPT by 27.1% and 8.3% respectively [2].

Domain Specific models

The BERT model was pre-trained on a general purpose text corpus, composed by the BooksCorpus [60] and the texts and paragraphs extracted from the English Wikipedia [2], that score in total 3.3 billion words. Hence, the model was not tuned for a specific domain. A step in that direction was taken by BioBERT [61], a BERT model trained on the PubMed abstracts and full-text articles. In fact, the authors state that the general model was not able to capture the complexity and the terminology used in the biomedical texts. The results indeed show an improvement – for example – in the question answering downstream task, scoring 12.24% better than BERT model [61]. A further study in the same direction was taken by SciBERT [62], that trained from scratch the BERT model on a text corpus made of 1.14 million papers picked randomly from Semantic Scholar [63], with the same procedure described in the BERT paper. Moreover, SciBERT used its own vocabulary, SCIVOCAB, that was built from scratch using the Sentencepiece library [64]. SciBERT achieved state-of-the-art results in different datasets [62].

Both these implementations show the importance of pre-training on specific text corpus when working within domain related environments.

Sentence Transformer: SBERT

The BERT architecture is suitable for a wide range of tasks like classification tasks, question answering and semantic sentence similarity, thanks to its structure that allows to plug any downstream network at its output. To work on semantic sentence similarity tasks, the pair of sentences whose similarity the system wants to score are fed in input preponed by the [CLS] and separated by the [SEP] token. On top of the system, a feed-forward fully connected layer can be attached in order to output the score.

According to [3], the drawbacks of this approach are mainly referred to the quality of the sentence embeddings produced averaging the word embeddings in output from BERT that result to be even worse than the ones coming out of GloVe.

Hence, the general idea presented by Reimers et al. [3] allows the creations of sentence embeddings to be used in tasks such as semantic similarity and semantic search. Sentence-BERT incorporates the knowledge already gained by BERT during the pre-training phase, and uses it to empower sentence embeddings construction. It uses a siamese network [52] – made of two identical BERT instances – that determines the fixed-length encodings: this approach derives independent outputs from the two sentences, namely u and v in Figure 2.11.

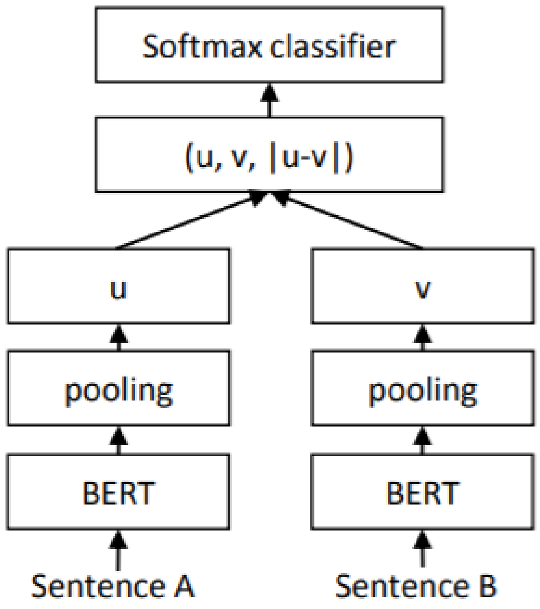


Figure 2.11: SBERT architecture, with classification objective function [3]

More in detail, SBERT uses a pooling layer on top of each single BERT instance. This is done in order to retrieve a fixed-length vector out of the set of word embeddings given in output from BERT. In particular, the pooling chooses either the [CLS] output token, the max value of all the word embeddings, or their average. The network is further composed of a layer that implements different objective functions, depending on the downstream task and on the dataset used for the training. Hence, being respectively u and v the sentence embeddings of the first and second sentence, the possible objective functions are:

- *Classification objective function*: it takes in input the concatenation of u, v and the element-wise difference $|u - v|$. Then, a softmax operation is applied, resulting in $o = sofmax(W_t(u, v, |u - v|))$
- *Regression objective function*: optimizes over the cosine-similarity measures of sentence embeddings u and v.
- *Triplet objective function*: given a sentence p, the network wants to maximize the difference between p and v when v is in contradiction with p and minimizes the difference with u when p and u are similar. Therefore, minimization of the function $o = max(||s_p - s_v|| - ||s_p - s_u|| + \epsilon, 0)$

The pool of data used for the training of this framework is made of the combination of the SNLI [65] and Multi-Genre NLI [66] datasets, tailored for the Natural

Language Inference task. It consists of predicting whether two sentences in a pair *contradict*, *entail* or are *neutral* to each other. Conneau et al. [52] showed that the Natural Language Inference task is suitable for learning sentence embeddings, since it forces the model to map the semantically meaningful information of the sentence into the vectors. SNLI and Multi-Genre NLI together make a corpus of around 1 million sentence pairs that cover a wide range of topics [3]. To tune the sentence embeddings, the *classification objective function* is used.

The resulting system was evaluated on two main downstream tasks: Semantic Textual Similarity (STS) task and SentEval. The first quantifies how much two sentences are similar in a scale from 0 to 5, using the sentence embeddings to compute the cosine-similarity and the Spearman [appendix] correlation to compare the result with the label. SBERT outperforms all the systems presented in the previous chapter, scoring the state-of-the-art on different STS datasets [3]. Likewise, SBERT scored the state-of-the-art in the SentEval toolkit of tasks used to evaluate the quality of sentence embeddings.

2.2 Semantic Web

The Semantic Web was introduced by Tim Berners-Lee in 2001 [7]. It is thought not as a different web, but as an extension of the existing one, where the information is given a well-defined meaning, linked and understandable by computers [8]. The fields where Semantic Web gained more and more popularity are diverse. It is exploited by organizations in knowledge management tasks, to insert, maintain and retrieve information in different branches. It is, for example, the technology that stays at the core of different search engines like Google and Baidu, to show to the users the information they are looking for. Semantic Web is also used in Business to Customer (B2C) companies, on the example of Amazon and Alibaba, which take advantage of the of the structure of data to retrieve to the customer the best product possible. In the following section, the semantic web stack is presented and analyzed.

2.2.1 The Semantic Web stack

The Semantic Web is based on the semantic web stack [67], where each layer relies on the underlying ones [68]. Unicode and URI are the fundamental layers of the stack, the first one addressing the set of characters accepted, the second – the Uniform Resource Identifier – is a unicode based string that uniquely identifies and locates the resource in the Web [69].

The second layer is made of the RDF/XML framework. The Resource Description Framework is the key component to instill structure in the data, since it allows the insertion of metadata [70] and its processing [71]. Moreover, it is particularly suitable to represent data to be processed by machines, since it is also understandable by them [68]. The idea behind the RDF is: every resource can be addressed by a URI, and a relationship between two entities is encoded in a triple. A triple is made of subject, predicate and object, all identified by URIs. This structure can encode any type of information, and can describe a resource via simple properties

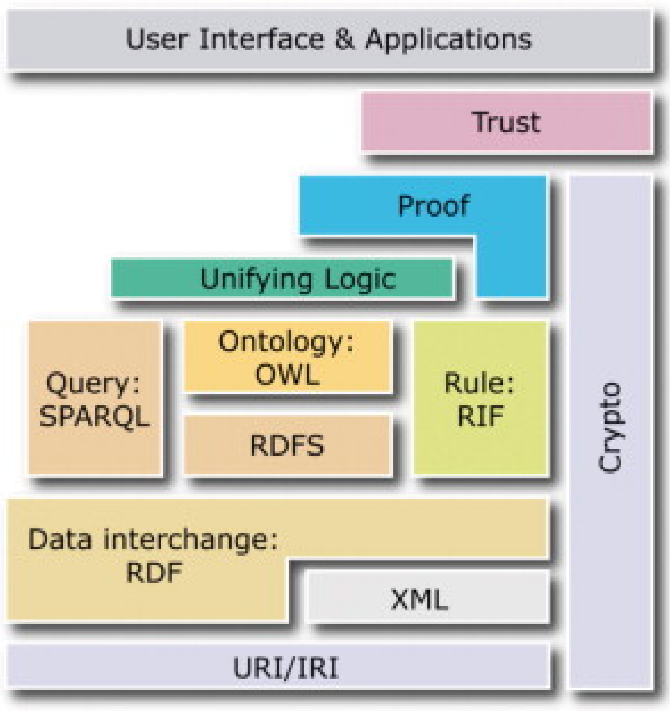


Figure 2.12: The Semantic Web stack [67]

or property values [70]. On top of it, there are the RDF Schema (RDFS) and the Web Ontology Language (OWL). The first creates a vocabulary over RDF and is a valid tool to create hierarchies between resource type – the *classes* – and among their properties [71]. OWL, instead, is a much more powerful language, since it allows more expressiveness and flexibility in ontology design [71]. An ontology is a description of a area of knowledge, formalized as a metadata schema that defines the vocabulary and incorporating the semantics [72]. The Semantic Web stack also includes a query language – SPARQL – which enables the pool of semantic data to be queried via specific point of presence on the Web – the SPARQL endpoint – using the HTTP protocol [8].

2.2.2 The Semantic Web at Infineon - the Digital Reference

In the context of digitization, Infineon Technologies AG joined the *Electronic Components and Systems for European Leadership Joint Undertaking (ECSEL JU)* [73], a program where the European Union funded both public and private projects to push the industry towards digitization [1]. In this context, the Productive 4.0 [74] project started with the aim to build a bridge between ”real and digital world by efficiently designing and integrating both the hardware and software of IoT” [1]. Here – in WP7 – the consortium defined a semantic web version of semiconductor supply chains and supply chains containing semiconductors [1] – the Digital Reference. It is made of different sub-ontologies, that describe sub fields of the semiconductor production and its supply chain as well as communication protocols and technologies, that have been merged in the past years [1]. In this thesis, this is the domain specific ontology we are going to exploit and annotate in the document retrieval system. Further details will be provided in the related description of Chapter 3.

2.2.3 Semantic Search

Semantic Search is the discipline to search and retrieve resources according to the topic of interest of the query. Either the resource is a web page or a document, the aim is to go beyond the pure matches of words and to exploit their meaning [75]. In fact, it is possible that the same word could refer to different concepts, which are not related to each other. Therefore, this is the case when *polysemy* occurs, and disambiguation techniques must be adopted to solve it. According to [76], one way to overcome this issue is the usage of ontologies to disambiguate the words. In fact, the schema depicted by the ontology clearly allows to distinguish different concepts, due to the structure that semantically describe each of them in a machine understandable fashion [8]. The tools created in order to implement semantic search are numerous, and they witness the importance of the usage of an ontology to structure and improve the results of searches.

GoNTogle According to [76], GoNTogle [77] is a system that allows to search over a pool of annotated documents, where the annotations are taken from the classes, properties and entities of an existing ontology. It implements a hybrid search, which takes into account the results given by a keyword based search and a search that uses the semantic annotations.

KIM In [78], Popov et al. describe *KIM - Semantic Annotation Platform*, an open source platform able to store, index and search over a pool of documents based on the ontology developed specifically for the tool – KIMO – and its specific knowledge base. The core of the system are the KIM APIs, which allow for a keyword and ontology-aware search as well as a semantic annotation system.

In the following chapter, we will present the methodology we adopted in order to combine ontology and semantic similarity measures over sentence embeddings in order to enhance the quality of document retrievement.

Chapter 3

Methodology

In the following Paragraphs the methodology to create the document retrieval system is outlined. It is articulated in two phases: in the first part, we show the description of the work done *offline* to prepare the main components of the online architecture. In particular, the complete pipeline describing the process to further pre-train the language model on the specific domain and the conversion of the sentences to the corresponding embeddings is carried out. Moreover, it also describes the procedure followed to annotate the Digital Reference. Then, in the second, we delineate the architecture of the *online* system the user will be interfaced with, other than the way in which it exploits the language model and the Digital Reference to retrieve relevant documents to the user.

3.1 Offline model preparation

In this Section we present the pipeline necessary in order to process the documents and create the vector representation out of their sentences. First, we will deal with the conversion from PDF to TXT, stored in JSON format. Then, the text contained in the documents is split into sentences with the SpaCy library [79].

PDF to JSON conversion

Most of the documents available for scientific purposes are usually in the PDF format. Since the PDF format does not allow to work directly with the content of the file due to the binary encoding, in order to be able to process these files, it is necessary to convert them to an encoding easily manageable by general purpose programming languages, such as the *utf-8* standard. There are many tools available on the market, either free or under paywall, desktop or online applications, like the Python library PyPDF2 [80], the desktop software Foxit PhantomPDF - PDF Editor [81] and Kofax PDF [82], and the online platforms like PDFX [83] and Smallpdf [84]. The quality of these tools is usually measured by the ability to precisely extract the text, reducing the noise and formatting related paragraphs together.

In the specific case of Infineon Technologies Corporate Supply Chain Innovation department, 663 files are present. They are scientific papers, Bachelor's, Master's and PhD's thesis in PDF format, concerning the topic of Semiconductor Supply Chain.

To extract the text out of these documents, we exploited the online tool PDFX

[83], a rule-based algorithm able to convert a PDF document in XML that is able to recognize font and layout of the articles, therefore maintaining the logical structure and assuring a good precision in the conversion. This allows us to identify the title and the chunks of text getting rid of the tables and the captions in the documents, that are usually source of noise in the conversion. Moreover, it is free of charge and allows batch conversion. The output of this system is an XML file, embedding in the XML tags the text, the tables and the figures of the document. A sample of the structure of the document can be found in Appendix A.1.

The following step consisted in the extraction of the text out of the XML files. In this phase, we did not consider tables and figures, as well as their captions. Analysing the content of the XML files, we could notice that most of the text is embedded in the *region* tags having the class *DoCO:TextChunk*. Hence, we considered those text chunks together with the corresponding headers – other than the paper title – to create a JSON file with two fields for each document: text and title.

Unfortunately, PDFX [83] was not able to convert all the documents, due to restrictions in number of pages and size of the document accepted. The tool, in fact, allows to submit PDF documents up to 5MB and until 100 pages. Moreover, it is not able to turn images into text. For this reason, of the 663 documents available to our department, only 499 were converted using that tool, while the others required the usage of another software. In particular, Infineon provides *Kofax Power PDF* [82], which is able to convert pdf documents of any size towards different formats, other than providing Optical Character Recognition functionalities, that allow the images containing text to be converted to text. The drawback of this tool is its inability to extract the text into a structured format – for example XML – that allows the individuation of noise, tables and pictures' captions in the document and the consequent removal using any general purpose programming language.

Likewise for the XML files, the TXT files are then saved to JSON file, in order to be able to further process them.

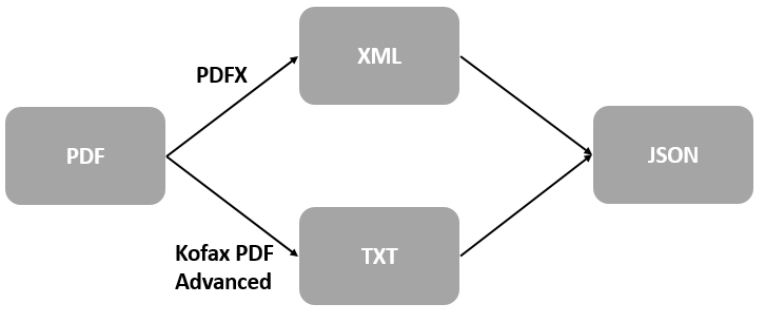


Figure 3.1: Pipeline for the conversion from PDF to JSON

Sentence tokenization As described in paragraph 2.1.4, to tailor the language model on a specific domain – as shown in [62, 61] – it is necessary to process the input documents in order to create the text corpus where BERT [2] is further pre-trained on.

SpaCy [79] is an open-source library for natural language processing in Python and Cython. It is an out-of-the-box tool which is widely used in the NLP community, and will be used here as well to tokenize the corpus into sentences. According to the documentation of the library, the sentence segmentation task is carried out using a dependency parser, which helps a statistical model to detect the sentence boundaries.

The final result of this step is a data structure made of a list of dictionary objects, each containing as fields:

- the title of the document
- the entire text corpus split into sentences

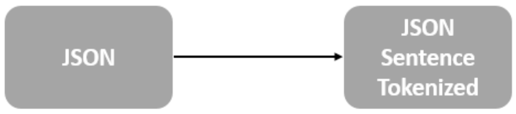


Figure 3.2: Sentence tokenization of the content of the JSON files.

In the next two Sections, the further pre-training of the BERT language model on the text corpus is described together with the way it is exploited to retrieve relevant documents to the user's query.

3.1.1 BERT domain specific pre-training

To further pre-train BERT on a domain corpus, the input must have a specific data format. According to the BERT official repository on GitHub [2], the input text has to be in the format of a unique text file made of one sentence per line. Moreover, each document is separated from the other by some blank lines.

Then, we proceeded with the creation of the dataset for the two training tasks described in paragraph 2.1.4, namely Masked Language Modelling and Next Sentence Prediction. For this purpose, the BERT repository provides a script called *create_pretraining_data.py* – implemented in Tensorflow v1.11.0 [85] – that takes in input the raw text and outputs binary files that will be later used to train the model. Additionally, the script concatenates the sentences up to the maximum length limit, in order to reduce the computation for padding elements. As shown in the repository, the command to launch the script is:

```
python create_pretraining_data.py
--input_file=<path/to/the/text/corpus>
--output_file=<path/to/the/tfrecords>
--vocab_file=<path/to/the/vocab/file>
--do_lower_case=True
--max_seq_length=128
--max_predictions_per_seq=20
--masked_lm_prob=0.15
--random_seed=12345
--dupe_factor=5
```

where there are different parameters. The *vocab_file* parameter indicates the path to the vocabulary file used in the by the BERT model. As stated in paragraph

2.1.4, it uses the WordPiece vocabulary [58]. The *masked_lm_prob* refers to the percentage of tokens to be masked, according to the procedure described in 2.1.4 it is the 15% of the total. The *dupe_factor* reflects the percentage to split in train and validation, having 5% of the corpus as validation set. According to the BERT repository [2], the pre-training should be splitted into two phases: the first phase – when the sequence length is 128 at maximum – helps the algorithm to seize the characteristics of the text corpus. Instead, in the second – when the length is 512 – longer sequence relations are captured. Since we trained the model twice with different sequence length, we created two binary files in accordance.

The number of global steps for which the model is trained in the first phase is 18000 steps, corresponding to 14 epochs. The system is further trained for 5000 steps when the maximum sequence length is 512, resulting in a training for 2 epochs more. To do the further pre-training, we used one Graphical Processing Unit of 24GB GDDR, and ran the script *run_pretraining.py* of the BERT repository [2] with the following parameters:

```
python run_pretraining.py
——input_file=<path/to/tfrecords>
——output_dir=<path/to/the/output/model>
——do_train=True
——do_eval=True
——bert_config_file=<path/to/config/file>
——init_checkpoint=<path/to/model/checkpoint>
——train_batch_size=56
——max_seq_length=128
——max_predictions_per_seq=20
——num_train_steps=18000
——num_warmup_steps=8000
——learning_rate=4e−5
```

Here the input corresponds to the binary files created with the previous script, while the second parameter refers to the output directory where the model will be saved. Since first we wanted to train and then evaluate the model, we set the *do_eval* and *do_train* flags to true. The next parameter is the *bert_config_file*, which requires a json file describing the architecture of the system used. In our case, we used the one described in section 2.1.4. Since we are further pre-training the original BERT model, we need to import the initial checkpoint containing the weights of the architecture trained by Google, and made publicly available in the repository [2]. Additionally, the batch size (*train_batch_size*) and the maximum sequence length (*max_seq_length*) are set. With a sequence length equal to 128, we could use a batch size of 56, while when moving to sequence length of 512, the batch size reduced to 8, due to high GPU memory requirements. The number of warm-up steps identifies the number of steps necessary to increase the learning rate from 0 to 4e-5. In the second training phase, with the *max_seq_length* parameter equal to 512, we decreased the learning rate to 2e-5, as suggested in the repository by the authors [2].

At the end of the training procedure, the language model is evaluated and the results shown in Table 3.1. The accuracy for the language modelling task is only 64.5%, while the accuracy for the next sentence prediction task is close to 99%.

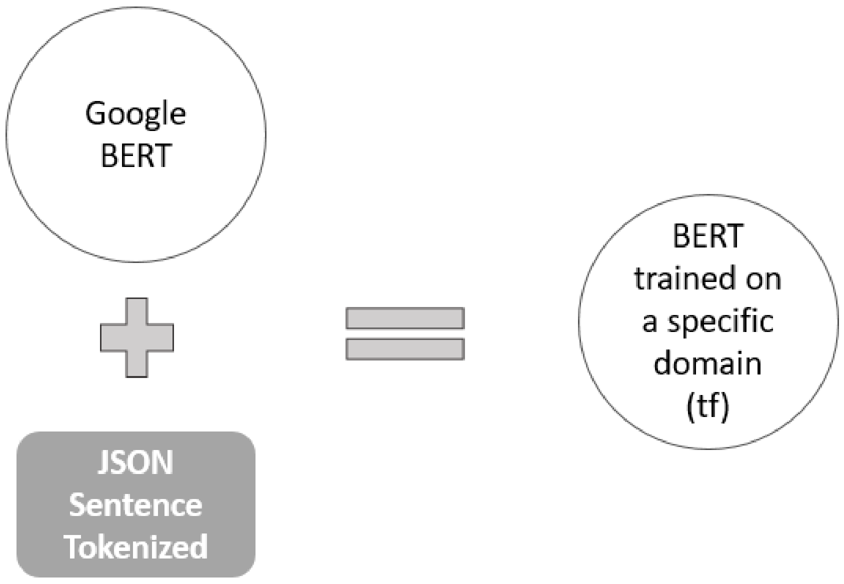


Figure 3.3: Representation of the training procedure on the specific domain.

There is not a clear metric to understand whether the results obtained are optimal, except testing in a downstream task [2].

Table 3.1: Language model evaluation results.

Metric	Value
global_step	23000
global_loss	1.8077
masked_lm_accuracy	0.6450
masked_lm_loss	1.7598
next_sentence_accuracy	0.9875
next_sentence_loss	0.0458

In the next Section, the model will be converted to the PyTorch v1.3.1 framework in order to be compatible with Sentence-BERT [3], the system used to finally produce sentence embeddings.

Conversion from Tensorflow to Pytorch checkpoint

In order to be able to train Sentence-BERT [3] on the language model further pre-trained as in the previous section, it is necessary to convert it to a PyTorch v1.6.0 [86] format checkpoint. The two frameworks, in fact, use a different representation of the computational graph underlying the architecture, being the Pytorch one dynamic and the Tensorflow static. Moreover, they use a different tensor representation, that makes the two models impossible to work together, except with the use of a proper script for the conversion.

For this task, the script *convert_bert_original_tf_checkpoint_to_pytorch.py* available in the public repository of HuggingFace [87] comes in handy. According to the documentation, it loads the weights from the Tensorflow checkpoint and saves them in the Pytorch model. The configuration file and the vocabulary are also necessary for this task, since they will be used by the PyTorch model as well.

3.1.2 Sentence-transformer

Once the BERT model compatible with the PyTorch framework has been created, it is possible to proceed with the encoding of the sentences. For this purpose, the repository sentence-transformers [3] is used. As described in section 2.1.4, this architecture allows the production of sentence embeddings exploiting a siamese network, where both of the two instances are BERT systems (Figure 3.5). For this purpose, we will use the BERT system that was further pre-trained as shown in the previous Section. On top of each BERT instance an average pooling layer is exploited to output the sentence embeddings of the proper dimension (*d=768*).

The next step implies the training of Sentence-BERT over the combination of SNLI and MultiNLI dataset and evaluation on the STS dataset, with the architecture made of the two siamese instances of the BERT model further pre-trained, average pooling enabled and soft-max function at the end to classify inference, contradiction or neutrality.

The results computed in term of Pearson and Spearman are shown in Table 3.2.

Table 3.2: Sentence-BERT: Training on Multi-NLI dataset, STSBenchmark evaluation

	Pearson	Spearman
Cosine-Similarity	0.7331	0.7633
Manhattan-Distance	0.7682	0.7658
Euclidean-Distance	0.7665	0.7655
Dot-Product-Similarity	0.7165	0.7225

To increase the amount of data the system is trained on, the STS dataset is also used to further train the model. The results are outlined in Table 3.3.

Table 3.3: Sentence-BERT: Training on Multi-NLI dataset and STS dataset

	Pearson	Spearman
Cosine-Similarity	0.8415	0.8490
Manhattan-Distance	0.8453	0.8483
Euclidean-Distance	0.8458	0.8488
Dot-Product-Similarity	0.8107	0.8117

As we can see from the Table 3.3, the results obtained are not so far from the state-of-the-art scored by Sentence-BERT trained on the generic dataset, which was of 0.8833 of Spearman correlation [3]. Our system scored 0.8412 on average, and we argue the gap is essentially due to the specific domain the BERT model was further pre-trained on.

Creation of the embeddings

Once Sentence-BERT is trained on the two main dataset, it is ready to create the embeddings of every sentence of every document. Therefore, every sentence

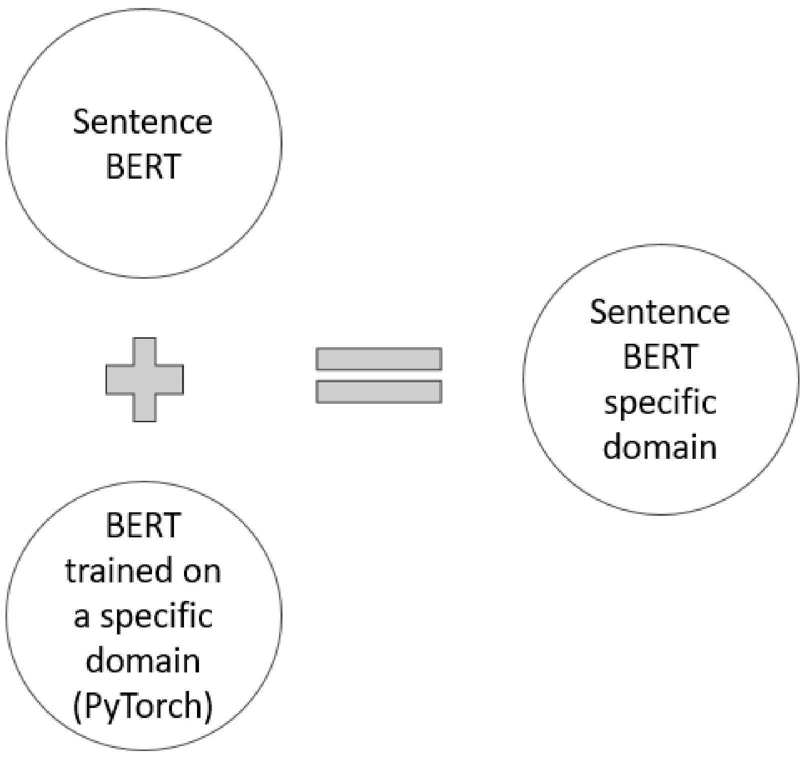

Sentence BERT
BERT trained on a specific domain (PyTorch)
Sentence BERT specific domain

Figure 3.4: Sentence BERT empowered by the domain specific BERT.

is processed by the model (Figure 3.5), the output embeddings saved in a database together with the sentence itself.

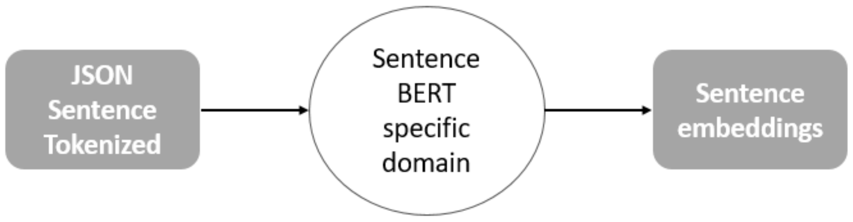


Figure 3.5: Mapping of every sentence of the documents to the corresponding sentence embeddings using domain specific SentenceBERT.

3.1.3 Annotation of Digital Reference's classes

The usage of ontology helps the description of a domain in a structured and consistent way, allowing all the players involved in its usage with different roles to "speak" the same language, precisely defining the relations between data and type [8]. In this paragraph we are going to exploit this structure to add information and resources to the domain specific ontology of interest.

When a domain ontology is used, it is likely to expect that all the classes are labeled with the corresponding natural language name, usually in English. This convention is fundamental to understand what the classes represent, since they are usually named after strings that do not have a meaningful sense. For instance – as shown in [88] – a class from the OBO library [89] ontology named *ex:BFO_0000006* has as label *"Spatial region"*, that allows us to understand it refers to the class representing a spatial region. According to the World Wide Web Consortium [90], the *rdfs:label* is an instance of the *rdf:Property* and is used with this purpose. Figure 3.6 shows this relation.

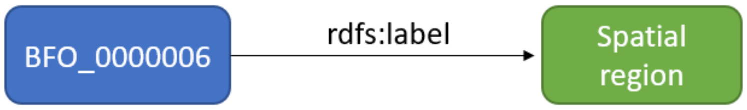


Figure 3.6: Relations of the Organizational Unit class.

The structure triples in the ontology have is suitable for the creation of sentences. In fact, each triple is composed of subject (the class we are considering), predicate and object. Taking advantage of this formalization, a set of sentences can be built up exploiting the graph structure and the set of triples each class establishes together with the predicates and the other classes that are directly connected to it. Therefore, the pool of sentences created in this way corresponds to the description of the relations the considered class has with the other classes of the ontology.

Then, these sentences are considered as queries and – for each of them – the corresponding sentence embedding is computed. With the computation of the cosine similarity score between each query and all the sentences of the literature text corpus, it is possible to retrieve – for each triple – the best sentences and consequently the best documents of the literature inherent to the class itself. The retrievement process will be further explained in detail in the next section, where the same approach is adopted to retrieve relevant documents to the user's query.

Accordingly, the top-k documents can be attached to the class using another relationship – *rdfs:seeAlso* – that, as defined in the W3C specification [90], allows to list the resources that can supply further information about the class.

This approach is the one we followed to populate the Digital Reference, the ontology for Semiconductor supply chains and supply chains containing semiconductors [1]. As an example, we can consider the class

http://www.w3.org/ns/org#OrganizationalUnit

that in the Digital Reference is used to describe a unit of an organization. The label of this class is "Organizational Unit". Figure 3.7 shows the relations between this class and the connected classes.

Then, we extracted the sentences out of these relationships, using the owlready2 library [91]. This allowed an easy access to properties of the classes of an ontology. Consequently, the concatenation of the labels led to the formulation of the proper sentences. Hence, the sentences:

- Organizational unit has sub unit organizational unit.
- Organizational unit sub unit of organizational unit.
- Organizational unit sub class of business line.
- Organizational unit is responsible of function object.

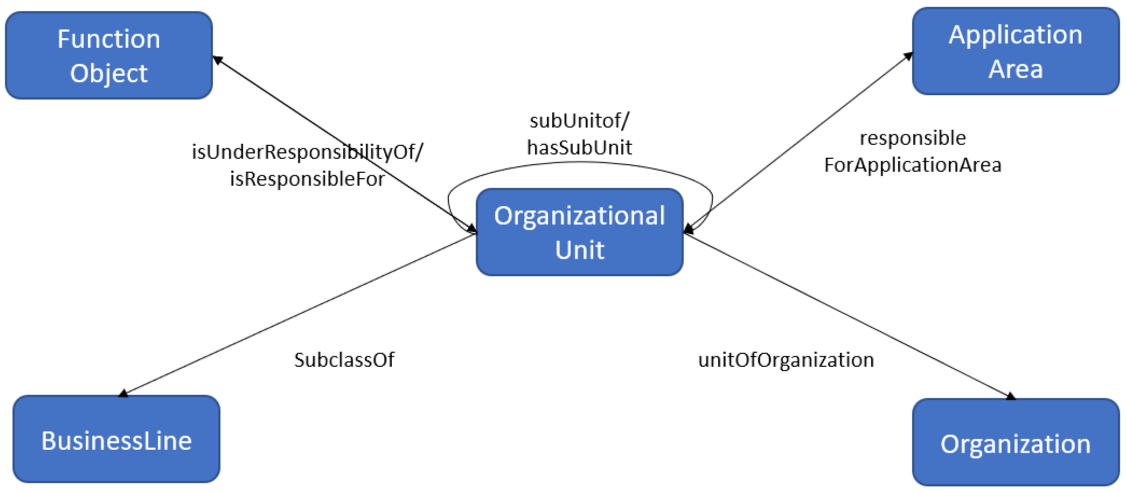


Figure 3.7: Triple representing a labeled class.

- Organizational unit responsible for application area application area.

These sentences are given in input to Sentence-BERT, producing the mapping to the vector representation. Through the computation of the cosine similarity between each query and the pool of sentences, the documents are consequently ranked. The number of documents we decided to attach to each class is equal to five. Therefore, the documents attached to the class *OrganizationalUnit* with the *rdfs:seeAlso* can be found in Table 3.4.

Table 3.4: Top-5 documents attached to the Organizational Unit class with rdfs:seeAlso

Document link	Cosine Similarity
Master_Thesis_final_Julian	0.8225
Paffenholz_Thesis	0.8154
winning with risk management	0.7698
MasterThesis_DavidJG	0.7607
Russland_diplomarbeit	0.7379

All the classes of the Digital Reference have been annotated with further resources in the same way as done for the Organizational Unit class.

However, the attached documents via *rdfs:seeAlso* and the label described in the *rdfs:label* relation are not the only meta information related to the class. Every class is further annotated with a comment that describes precisely the usage of the element referred to by the class. The *rdfs:comment* – described in [90] – is the predicate that links "human-readable description of a resource" [90] to the resource itself. This definition will be retrieved together with the five documents taken from the literature to better describe the elements of the ontology that match with the keywords of each query.

To conclude, the Digital Reference is exploited as data structure that contains useful resources for the classes that belong to the ontology – attached as links to documents – and as a vocabulary where the description of every class is easily accessible.

In the next Section, we will describe how the annotated Digital Reference will be embedded in the document retrieval system.

3.2 Online architecture: the document retrieval system

Concluded the *offline* design of the main elements, the *online* architecture can be delineated.

Since the number of documents the user can browser is 663 but potentially could be much more, it is convenient to design the system in order that it scales with the increasing number of documents. Moreover, the Sentence-BERT instance is not a lightweight model and requires a lot of computational power to be used effectively. For these two main reasons, the system was designed as a web application, with a client-server architecture, where both the documents and Sentence-BERT are stored on the server side together with the Digital Reference. In particular, the server is interfaced with a database where the sentence embeddings, the sentences and the relative PDF documents are kept. In order to effectively design the web-application, a design patter was considered. In the following Paragraph, a brief overview of the specific architecture design chosen is done.

3.2.1 Architecture design

Multiple components are involved in the design of the system, since – as we stated in the previous section – the Sentence-BERT model, the sentences and the corresponding embeddings, and the Digital Reference come into play. Each of these elements is used for different purposes. In particular:

- Sentence-BERT: used to convert the query into the corresponding vector embedding.
- The sentences and sentence embeddings: stored in a relational database together with the corresponding documents consist in the content to be retrieved to the user.
- The Digital Reference: it is used as an index to retrieve definitions and documents attached to the classes whose name appear as keywords in the user's query.

These elements have been prepared in an offline moment and are – in this phase – ready to be used. The overall architecture we adopted is a multi-tier architecture, with a logic built over the client-server one. Each tier represents a subsystem of the system itself and provides a specific functionality where the elements previously listed are harmonized to work together. The generic multi-tier architecture can be based on a variable number of tiers, depending on the complexity and the needs of the application [92].

The general framework expects three main tiers [92, 93]. The first in the presentation tier, that usually runs on the client and displays the Graphical User Interface. The second is the application layer, where the business logic is implemented and the latter is the database layer, where the data are stored. The general architecture of our document retrieval system follows this pattern, but with a small modification: the application layer is separated into two main parts, that are activated in different moments of the run. In the former, the application tier deals with the loading of Sentence-BERT, the sentence embeddings and the Digital Reference, while in the latter it uses this data to search for relevant documents to the query.

In Figure 3.8, the comprehensive architecture is depicted, showing the building blocks and how they are related to each other.

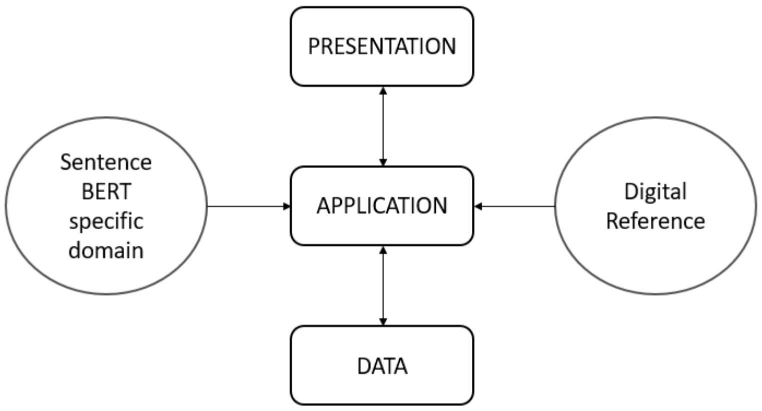


Figure 3.8: Architecture of the document retrieval system.

In the following Paragraphs, the detailed description of the functionalities of each of the elements is carried out.

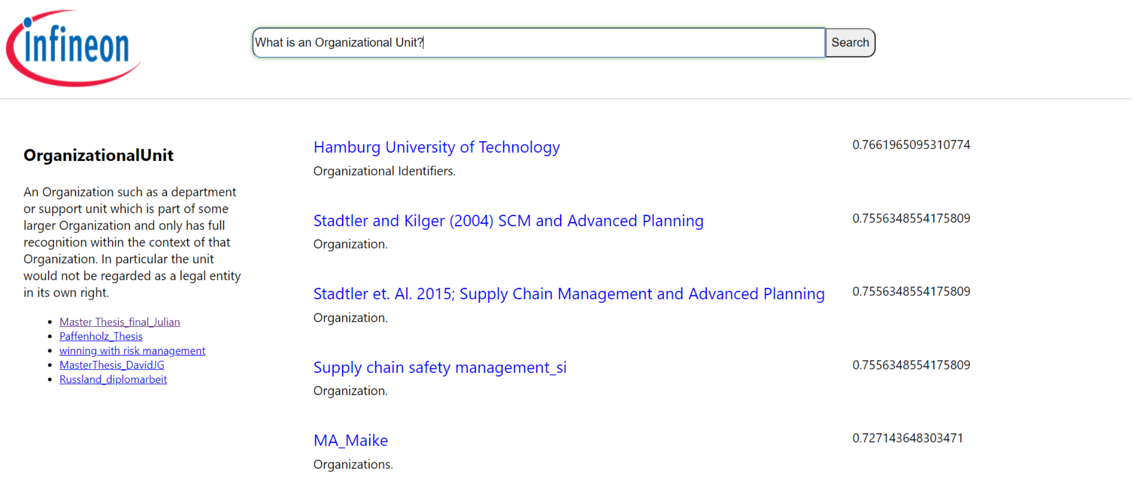
3.2.2 The presentation tier

The presentation tier (or the client) is responsible to take in input the query of the user in natural language and to forward it to the server. For this purpose a search bar is created – where the query can be typed – and a search button in charge of forwarding the string to the server with an HTTP request.

Once the server answers with the list of document titles, sentences and scores, documents and definitions coming from the Digital Reference, the client displays them. An example of response coming from the server can be seen in Appendix A.2.

In particular, for every document displayed to the user, the title is heading, immediately below it the sentences matching the query are reported, and on the right in respect of the title the score is shown. When the query contains keywords that match with the class names of the Digital Reference, the corresponding definition is retrieved and showed, together with the top-5 documents related to the class, on the left in respect to the list of documents. An example is represented in Figure 3.9.

When the user wants to open a certain document, the title offers an embedded link to the document. By clicking on it, another HTTP request to the server is carried out, with the title of the document embedded in the URL. In the end, the document received from the server will be opened in a new window of the browser.



OrganizationalUnit

An Organization such as a department or support unit which is part of some larger Organization and only has full recognition within the context of that Organization. In particular the unit would not be regarded as a legal entity in its own right.

- Master Thesis_final_Julian
- Paffenholz_Thesis
- winning with risk management
- MasterThesis_DavidJG
- Russland_diplomarbeit

Hamburg University of Technology 0.7661965095310774
Organizational Identifiers.

Stadtler and Kilger (2004) SCM and Advanced Planning 0.7556348554175809
Organization.

Stadtler et. Al. 2015; Supply Chain Management and Advanced Planning 0.7556348554175809
Organization.

Supply chain safety management_si 0.7556348554175809
Organization.

MA_Maike 0.727143648303471
Organizations.

Figure 3.9: Example of the document retrieval system graphical interface.

3.2.3 The application tier

The server is the most complex element of the building blocks, and performs different operations.

When it is started, it is in charge of three tasks:

- Loading the Sentence-BERT model.
- Loading the sentence embeddings and the sentence_id from the database.
- Loading the Digital Reference ontology.

Once the server has loaded the required elements, the system is ready to serve the requests of the user.

When the user submits the query, it is encoded by the Sentence-BERT encoder (SBERT in the algorithm formalization) to the corresponding sentence embedding. Then, the norm-2 of the vector is computed, and the dot product between all the sentence embeddings and the query embedding is used to rank all the sentences. To speed up the computation, the normalization of the sentence embeddings takes place when they are loaded from the database. Consequently, for every query the user submits to the system, it is only necessary to compute the norm of that sentence embedding and then to quantify the dot product of the query with each sentence embedding. This is the equivalent of computing the cosine similarity, and avoids to spend computation time while the user is waiting for a result.

$$Cosine\ similarity = \frac{\vec{a} \cdot \vec{b}}{||\vec{a}||||\vec{b}||} = \frac{\vec{a}}{||\vec{a}||} \cdot \frac{\vec{b}}{||\vec{b}||}$$

Next, the best fifteen results are taken in consideration. The *sentence_id* (see 3.2.4) is used to retrieve the corresponding sentence in natural language, the *document_id* (see 3.2.4) of the document the sentence belongs to and the title of the document. In Algorithm 3, the basic pipeline is outlined.

Algorithm 3 SBERT cosine similarity ranking

```
1: $query\_emb \leftarrow SBERT(query)$
2: $query\_emb\_norm \leftarrow ||query\_emb||_2$
3: for $(t = 0; t < NUM\_SENTENCES; t++)$ do
4:     $cosine\_similarity[t] \leftarrow query\_emb\_norm \cdot sentences\_norm[t]$
5:     $sentence\_id[t] \leftarrow t$
6: end for
7: $results \leftarrow sort\_based\_on\_cosine\_similarity(cosine\_similarity, sentence\_id)$
8: $sentence[], doc\_id[], title[] \leftarrow query\_database(top15(results))$
9: return $sentence[], doc\_id[], title[]$
```

Furthermore, if the query contains keywords that match with the class names of the Digital Reference, the corresponding class definition and documents are retrieved. In order to achieve this purpose, the SpaCy [79] library is used to identify the noun chunks, that according to the documentation represent the small sentences that refer and describe a noun and remove the stop words. Then, the keywords found so far are compared to the class names of the Digital Reference, using a fuzzy search algorithm [94]. Here, the Levenshtein distance [95] scores the distance between each keyword and the labels of the classes. It consists in the minimum amount of modifications a string must undergo to be equal to another one [96]. Therefore, all the class labels that score above above a certain threshold are used to access the class and retrieve the attached documents and related definition.

Algorithm 4 Digital Reference class-keyword fuzzy match

```
1: $matches \leftarrow list[]$
2: $key\_matches \leftarrow list[]$
3: $noun\_chunks \leftarrow identify\_noun\_chunks(query)$
4: $noun\_chunks\_without\_stop\_words \leftarrow remove\_stop\_words(noun\_chunks)$
5: for $keyword$ in $noun\_chunks\_without\_stop\_word$ do
6:     for $class$ in $Digital\_Reference$ do
7:         $match \leftarrow dictionary\{\}$
8:         $score \leftarrow Levenshtein(keyword, class.label)$
9:         if $score > 80$ then
10:            $match['keyword'] \leftarrow class.name$
11:            $match['definition'] \leftarrow class.comment$
12:            $match['related\_works'] \leftarrow class.seeAlso$
13:            $key\_matches.append(match)$
14:        end if
15:    end for
16:    $matches.append(key\_matches)$
17: end for
18: return $matches$
```

This set of information – together with the list of relevant documents found with the cosine similarity measure – is then sent to the client, in the format shown in Appendix A.2.

3.2.4 The data tier

In order to design the database properly, the entities that come into play are considered. The first pool of data concerns the sentence. It is necessary to have a sentence_id to identify the actual sentence, the related embedding and the document_id the sentence belongs to. The document_id is designed to be the foreign key of the other relevant entity, which is the document. Indeed, a document is identified by the document_id, its title and the file in PDF format, which is stored in the database. This structure is depicted in Figure 3.10.

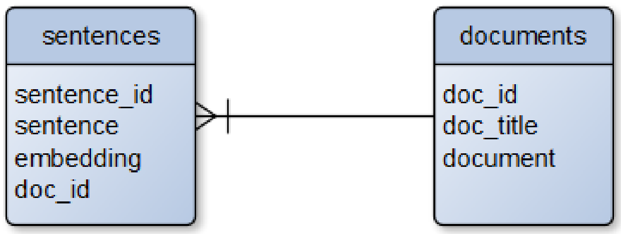


Figure 3.10: Entity relationship model representing the database design.

In the following Chapter, an evaluation of the system is built will be carried out, taking advantage of the knowledge of experts in the field of Semiconductor Supply Chain Innovation department.

Chapter 4

Evaluation

4.1 Evaluation

In order to state whether the system built so far is able to give an actual contribution to the company and to the user when this poses a specific query, we proceeded with the evaluation of the system. To the extent of our knowledge, there is no other tool previously developed on the same domain of semiconductor supply chain, there is no golden standard already available for the comparison. For this reason, we provided to the creation of a new reference for document retrieval systems over this domain exploiting the knowledge of a domain expert. In particular, given a set of queries, the person interviewed will be in charge of defining the ground truth that will be used to evaluate our system.

In the following Paragraphs, the identification of the metrics and their description will be followed by the overview of the experiment. In the end, the results are presented together with a brief discussion.

4.1.1 Metrics

The first step towards the evaluation of the system consists in the roughing out of the metrics to be adopted. The evaluation metrics normally used to evaluate information retrieval systems are divided into online and offline techniques [97]. While the formers are mainly oriented to the evaluation of the interaction between the user and the system in terms of user experience, the latter aims at stating whether the documents retrieved – and in general of the information – are relevant to the query submitted [98]: therefore, we will focus only on the offline assessment of the performance of the system, since the online is out of the scope of this thesis.

According to [99], the offline measures further divide into metrics for "unranked retrieval set" [99] and for "ranked retrieval set" [99]. As the names already say, the unranked metrics are used when the pool of results is treated as a pure set, where the order in which the items are presented does not play any role in the score of the metric.

Metrics without ranking

Among the measures that are often used to evaluate an information retrieval system when the order of the results is not central, we can find precision, recall and F-score [99]. In particular, precision in information retrieval is defined as the ratio between the number of relevant documents retrieved and the number of documents retrieved. Defined:

- the set of queries $Q = \{q_0, ..., q_i, ..., q_V\}$, with $V \in \mathbb{N}$
- the set of relevant documents to the query q_i retrieved by the system, $R(q_i) = \{r_0, ..., r_j, ..., r_M\}$, with $M \in \mathbb{N}$
- the set of documents retrieved by the document retrieval system according to the query q_i, $D(q_i) = \{d_0, ..., d_j, ..., d_T\}$, with $T \in \mathbb{N}$

the average precision among all the queries can be written as [99]:

$$Precision(Q) = \frac{\sum_{i=0}^{V} \frac{|R_{q_i}|}{|D_{q_i}|}}{V}$$

On the other hand, recall can be defined as the ratio between the number of documents retrieved by the system that are actually relevant to the query [99]. In order to calculate this metric, it is necessary to compute for every query which documents are relevant and which are not. Consequently, describing the set of all the documents relevant to a certain query q_i as $A(q_i) = \{a_0, ..., a_j, ..., a_O\}$ with $O \in \mathbb{N}$ and keeping the remaining notation the same as the one used to define the precision, we can define the average recall over all the queries as:

$$Recall(Q) = \frac{\sum_{i=0}^{V} \frac{|R_{q_i}|}{|A_{q_i}|}}{V}$$

According to [99], the presence of two measures to evaluate a system has some advantages. In fact, when the requirements of the systems do not imply it to return all the documents inherent to a query, but requires the document retrieved to be pertinent to the query, it is acceptable to have low recall and high precision. On the other side, some systems are particularly interested in getting all the information related to the query, trading it off for a low precision. In our case, no specific requirement on precision and recall is imposed. Hence the F-measure will be exploited, that balances them and will work as general indicator of the performance of the document retrieval system. It is defined as [99]:

$$F\text{-}measure = \frac{2 \cdot Precision \cdot Recall}{(Precision + Recall)}$$

Metrics with ranking

On the other hand, the ranked metrics are used when the order in the result is relevant to the the presentation of the documents related to the query. Among the metrics that can be used in order to establish whether the results returned are presented in the proper order, we can find Mean Average Precision (mAP) and Normalized Discounted Cumulative Gain (NDCG), that – according to Chen et al. [98] – are the metrics that are most widely used in the offline evaluation of ranked results.

As Sanderson et al. presented in [99], the Mean Average Precision is computed as the mean among all the queries of the average precision of every query. This, in turn, is calculated as the sum of the precision of a given set of k elements divided by the number of elements that are relevant for the query ($|A|$). More precisely, it can be seen as:

$$mAP = \frac{\sum_{i=0}^{V} AP(q_i)}{V}$$

where

$$AP(q_i) = \frac{\sum_{k=0}^{|R|}(Precision@k \cdot rel(k))}{|A|}$$

where Precision@k calculates the ratio between the number of relevant documents in the first k positions, and $rel(k)$ represents an integer equal to 1 in case the k^{th} element is relevant, 0 otherwise.

As it could be easily inferred, the mAP exploits a classification of the documents which is binary (e.g. *relevant*, *non-relevant*) to the query, while other metrics like NDCG require the documents to be classified according to a scale of relevancy, which is not as easy as the previous one to be stated. Moreover, the "good discrimination and stability" [99] of the mAP makes of this metrics the optimal candidate for our use-case.

The next Paragraph will describe how the experiment was conducted and will present the pool of queries chosen for the evaluation.

4.1.2 Experiment

Once the metrics to evaluate the system were defined, we proceeded with the individuation of the queries to run against the document retrieval system. Since the pool of documents to be handled has as specific domain the semiconductor supply chain, we considered the Supply Chain Operation Reference (SCOR) model [100] the semiconductor supply chain is based on and the main areas of interest of the department to individuate valuable queries. The internal organization of the department reflects the main topics addressed by the research, like digitization, simulation, green supply chain, human and artificial intelligence collaboration and semantic web. Therefore, the queries the expert decided to pose to the system – reported in Table 4.1 – inevitably inquire the system in these directions.

Consequently to the delineation of the queries, we asked the expert to identify the set of documents relative to every query according to a binary classification metric: for every document of the pool of documents, the expert had to label it with *relevant* or *non-relevant*. This allowed us to create a *golden standard*, whose content is outlined in Table 4.2.

Thus, all the elements for the evaluation are in place, and the results will be showed in the next paragraph.

4.1.3 Results

The queries previously defined are run against the document retrieval system and the results are presented in Table 4.3.

Table 4.1: Queries defined together with the domain expert, according to the topics of interest of the department.

ID	Query
Q1	What is the carbon footprint of Infineon Technologies?
Q2	What is an end-to-end supply chain?
Q3	What is the Semantic Web?
Q4	How is customer order behavior managed at Infineon Technologies?
Q5	Could artificial intelligence help identifying customer order behavior?
Q6	How can we optimize lot-size?
Q7	Which are the tools used in order management?
Q8	How does the order fulfillment work?
Q9	How do human and artificial intelligence collaborate together?
Q10	What is the Digital Reference?

Table 4.2: Golden standard defined together with the domain expert, according to the topics of interest of the department.

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
doc18	doc5	doc12	doc7	doc19	doc3	doc9	doc1	doc13	doc60
doc20	doc38	doc16	doc19	doc26	doc4	doc11	doc2	doc14	doc61
doc22	doc40	doc21	doc44	doc44	doc11	doc17	doc6	doc15	doc62
doc27	doc54	doc29	doc46	doc49	doc24	doc36	doc7	doc23	doc63
doc33	doc55	doc30	doc50		doc34	doc52	doc8	doc25	doc64
doc39	doc45	doc37	doc52		doc35	doc54	doc10	doc31	doc72
doc47		doc43	doc53		doc45	doc57	doc28	doc32	doc74
doc51		doc52	doc54		doc54				
doc59		doc73	doc55		doc55				
		doc74	doc79		doc56				
					doc58				

Table 4.3: Results from the document retrieval system, according to the queries outlined in Table 4.1.

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
doc39	doc54	doc29	doc44	doc76	doc54	doc82	doc55	doc25	doc63
doc51	doc69	doc12	doc76	doc81	doc24	doc17	doc6	doc23	doc25
doc59	doc70	doc72	doc54	doc44	doc56	doc7	doc84	doc32	doc91
doc33	doc40	doc30	doc77	doc49	doc82	doc15	doc10	doc14	doc92
doc54	doc71	doc63	doc78	doc26	doc4	doc11	doc2	doc89	doc60
doc22	doc55	doc48	doc19	doc82	doc83	doc54	doc86	doc13	doc72
doc67	doc17	doc43	doc79	doc79	doc45	doc85	doc87	doc90	doc62
doc27		doc73	doc80	doc6	doc55		doc1		doc74
doc65		doc74	doc49				doc88		
doc66		doc75	doc53						
doc18		doc16	doc55						
doc44									
doc68									

As next step, we proceeded to calculate the precision, recall, F-measure and mean Average precision according to Paragraph 4.1.1. The results are shown in Table 4.4.

Table 4.4: Results from the document retrieval system, according to the queries outlined in Table 4.1.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Precision	0.53	0.42	0.63	0.54	0.37	0.75	0.43	0.44	0.71	0.63
Recall	0.78	0.5	0.7	0.6	0.75	0.54	0.43	0.57	0.71	0.71
F-measure	0.64	0.46	0.67	0.57	0.5	0.63	0.43	0.5	0.71	0.67
AP	0.69	0.33	0.52	0.38	0.36	0.55	0.2	0.3	0.69	0.44

Consequently, we can proceed to the calculation of the mean of all the metrics per query, which is summed up in Table 4.5.

Table 4.5: Summary of the results obtained running the queries described in Table 4.1.

Precision	Recall	F-measure	Mean Average Precision
0.5486	0.6301	0.5777	0.4468

In the next paragraph, a short discussion of the results will be carried out.

4.1.4 Discussion

The results presented in the previous Paragraph outline different characteristics of the document retrieval system, that we will show according to the metrics used for the evaluation.

As a first instance, we can acknowledge that recall has – on average – values that fluctuate between 0.43 and 0.78, with an average of 0.63 . We argue that these results are due to the higher number of documents the system tends to consider as *relevant* compared to the number that actually are. Hence, among these last ones, it has more possibilities to individuates the correct ones, exactly like a system that retrieves all the documents for every query: it will have a recall equal to 1.

On the other hand, the precision shows that on average half of the documents retrieved by the engine are considered pertinent to the query posed. After a more accurate investigation, we inferred that the fact that the value of this metric is not excellent is due to the partial inability of the language model to precisely individuate the meaning of some expressions and the object they refer to. For example in Q10 – where the system was asked "*What is the Digital Reference?*" – some of the documents retrieved contained sentences with the word "digitization", hence representing the system not being able to understand the query exclusively referred to the supply chain semiconductor ontology. To overcome this issue, it is necessary to further pre-train the language model over a higher amount of documents specific to the semiconductor supply-chain domain, to allow the model to gain a better insight into the specific language used.

Moreover, the mean average precision shows that a good rate of relevant documents appear in the first positions of the results of the document retrieval system, therefore witnessing the effectiveness of the cosine similarity as score function.

To conclude, given the absence of another document retrieval system developed over the semiconductor supply chain domain we cannot compare the results to state the effectiveness of the overall engine. Anyway, we consider these first results encouraging for the further growth of the framework built so far. For the future work, we will evaluate the system on a publicly available dataset, to assess a comparison of the performance with other systems.

Chapter 5

Conclusions

5.1 Summary

The amount of data available on the Internet and produced by private users, companies and public infrastructures is immense. It is estimated that in 2018 the volume of information produced was 33 Zettabytes and will grow to 175 Zettabytes in 2025 [4].

The amount of public and restricted documents produced by institutions constitutes an invaluable source of knowledge for the members of the institution itself. Therefore the necessity to make these documents browsable, accessing them not only through the usage of a keyword based approach, but building a system able to understand the meaning of the content of the document.

To accomplish this goal, we created a document retrieval system with the usage of Transformer-based model and a domain specific ontology. In particular, the development of the system articulated in two parts: the offline and the online. In the first one – the *offline* – we adopted the Transformer-based model BERT [2], a bidirectional encoder pre-trained over a text corpus ready to be used in downstream tasks. To tailor the model to the specific domain we are dealing with – the semiconductor supply chain – we further pre-trained the model over the pool of papers, Ph.D. 's, Master's and Bachelor's thesis available to the Infineon Corporate Supply Chain Innovation Department. The result is a new language model that conserves the structure of the original one [2], and that furthermore includes the characteristics of the language model specific to the domain it is tailored for, consequently embedding the specific vocabulary and terms.

The new language model is then exploited to train a siamese network for the production of sentence embeddings, SentenceBERT [3]. SentenceBERT incorporates the knowledge BERT has stored during the training phase and uses it to train over the natural language inference task, that consists in understanding whether two sentences contradict, infer or are neutral to each other. The model obtained is consequently able to state whether two sentences are related to each other and – more specifically – understand whether they deal with the same topic. We will use this feature to understand to which extent each sentence of all the documents is related to the query the user submits to the system.

The mechanism just explained was used to annotate the Digital Reference [1] with the literature of the department, so to obtain a domain specific ontology where for every entity, some documents related to that class and those connected to it are attached as further annotation. This data structure will e exploited to show to the user the class related to the query submitted, when the query contains a keyword that fuzzy matches with an entity of the ontology.

The second phase of the architecture design – the online one – consisted in the development of the web architecture to make the framework accessible to the user. Based on the three-tier architecture, the document retrieval system contains a presentation tier responsible for the interactions with the user and the display of the information received by the application tier. The application tier, instead, is in charge of retrieving the documents whose sentences best match with the query posed by the user, other then browsing the Digital Reference to find a match between query keywords and entities. When the application tier identifies finds some relevant matches both within the ontology and in the set of sentences part of the literature, the application tier accesses the data tier to retrieve the relevant documents to the query.

The results gained comparing the documents retrieved for a set of ten queries with the golden standard provided by an expert are encouraging and allowed us to identify some points of action to improve the overall system.

5.2 Outlook

In this Paragraph we would like to address the future work that could lead to an improvement the overall performance of the document retrieval system built in this thesis.

In the first stages of the design of the offline architecture, we converted the PDF documents to pure text using different tools. After a further inspection of the results of the conversion, we acknowledge that some errors are present not only in the decoding of special characters, but also in the formatting. Hence, a tool able to improve the format conversion could involve a greater capability in the display of the results and in the further pre-training of the language models. To increase the performance of the language model, it is necessary to employ as much text as possible. Therefore, a future version should take into account this element as well.

The Transformer-based model we chose to use in this thesis the the *BERT base uncased*, whose characteristics are outlined in section 2.1.4. It could be interesting to see how bigger architecture – like *BERT large uncased* – and newer ones – like the Reformer [101] – would perform in this task.

Furthermore, similarity scores other then the cosine similarity can be explored to improve the ranking of the documents of the system.

To conclude, the visualization of the entities of the Digital Reference can be dynamically shown, in order to allow the user to gain a complete overview of the ontology and its structure.

Appendix A

Appendix

A.1 Sample of xml document

Sample of a xml document, given in output from the PDFX tool [83].

```
<?xml version='1.0' encoding='UTF-8'?>
<pdfx xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="http://pdfx.cs.man.ac.uk/
   static/article-schema.xsd">
  <meta>
    <job>098786967</job>
    <base_name>1x0i</base_name>
  </meta>
  <article>
    <front class="DoCO:FrontMatter">
      <outsider class="DoCO:TextBox" type="header" id="1">
          Title</outsider>
      <outsider class="DoCO:TextBox" type="outsider" id="2">
         Some text</outsider>
      <outsider class="DoCO:TextBox" type="outsider" id="3">
         Some Text</outsider>
    </front>
    <body class="DoCO:BodyMatter">
      <region class="DoCO:TextChunk" id="4" page="1" column=
         "1">Some text</region>
      <region class="DoCO:FigureBox" id="Fx5">
        <image class="DoCO:Figure" src="1x0i.page_001.
           image_01.png" thmb="1x0i.page_001.image_01-thumb.
           png"/>
      </region>
    </body>
  </article>
</pdfx>
```

A.2 Server Response example

Example of response from the server. Note: the title field is omitted, a generic identifier is used instead.

```
  {
"results": {
  "dr": [
    {
      "definition": [
        "An␣Organization␣such␣as␣a␣department␣or␣support␣
           unit␣which␣is␣part␣of␣some␣larger␣Organization␣
           and␣only␣has␣full␣recognition␣within␣the␣
           context␣of␣that␣Organization.␣In␣particular␣the
           ␣unit␣would␣not␣be␣regarded␣as␣a␣legal␣entity␣
           in␣its␣own␣right."
      ],
      "entity": null,
      "keyword": "OrganizationalUnit",
      "related_works": [
        "Master␣Thesis_final_Julian",
        "Paffenholz_Thesis",
        "winning␣with␣risk␣management",
        "MasterThesis_DavidJG",
        "Russland_diplomarbeit"
        ]
    }
  ],
  "sbert": [
    {
      "id": 373,
      "score": 0.7661965095310774,
      "text": "Organizational␣Identifiers.",
      "title": docX
    },
    {
      "id": 560,
      "score": 0.7556348554175809,
      "text": "Organization.",
      "title": docY
    },
    {
      "id": 561,
      "score": 0.7556348554175809,
      "text": "Organization.",
      "title": docZ
    },
    {
      "id": 584,
        "score": 0.7556348554175809,
        "text": "Organization.",
        "title": docX1
      },
      ...,
      {
        "id": 108,
        "score": 0.5750698906490561,
        "text": "Production␣and␣Operations␣Analysis.",
        "title": docY1
      }
    ]
  }
}
```

Appendix B

Appendix

The following sections present the code written to implement the

B.1 Application tier

Code of the application tier.

app.py

```
imports ...

app = Flask(__name__)
logging.basicConfig(level=logging.DEBUG)


@app.route("/")
def hello():
    return render_template("index.html")


@app.route("/query", methods=['POST'])
def collect_query():
    # text is retrieved from the form
    text = request.form['query']

    # the text is converted to vector
    render_data_SBERT = service.reply_with_sentence_BERT(infineonBERT,
    text, sentences_embeddings, id_vect)

    render_data_DR = service.reply_with_Digital_Reference(text,
    digital_reference_classes_and_names)


    # create a json to return it as application/json
    render_data = {'sbert': render_data_SBERT, 'dr': render_data_DR}

    result = {'results': render_data}

    return result


@app.route("/pdf/<filename>")
```

```
def get_pdf(filename):

    # file retrieved from the directory
    file = send_from_directory(app.config['UPLOAD_FOLDER'], filename +
   ".pdf")

    # response creation
    response = make_response(file)
    response.headers['Content-Type'] = 'application/pdf'
    response.headers['Content-Disposition'] = 'inline; filename=output.
   pdf'

    # response sent
    return response



if __name__ == "__main__":
    # load Sentence-BERT

    logging.debug("Sentence BERT loading..")
    infineonBERT = model.InfineonBERT("Infineon search engine")
    print("Sentence BERT loaded.")

    sentences_embeddings, id_vect = infineonBERT.
   files_sentence_embeddings

    # laod the ontology
    onto = searchWithDR.importOntology(searchWithDR.PATH_SAVED_ONTOLOGY
   )
    digital_reference_classes_and_names = annotateDR.
   get_tuple_class_name(onto)
    logging.debug("Ontology loaded.")

    app.config['UPLOAD_FOLDER'] = 'Folder/to/upload'
    app.run(debug=True, port=8080, use_reloader=False)
    logging.debug("Started server!")
```

service.py

```
NUM_THREADS = 4
NUM_DOCUMENTS = int(50 / NUM_THREADS)

# query to embedding conversion

def convert_query_to_embedding_Sentence_BERT(model, query):
    return model.encode(query, show_progress_bar=False)


# master thread management function to look into documents

def multiprocess_search(query_embeds, sentences_embeddings, id_vect):

    #logging.debug("Multiprocess search.")
    best_sent_documents = []
    results = {}
    threads = list()
    num_files = math.ceil(len(sentences_embeddings) / NUM_THREADS)
    lock = threading.Lock()

    for query_embed in query_embeds:

        query_embed = np.array(query_embed)
        query_embed = query_embed / np.linalg.norm(query_embed)

        for index in range(NUM_THREADS):

            index_left = index*num_files
            index_right = (index+1) * num_files

            if index_right > len(sentences_embeddings):

                index_right = len(sentences_embeddings)


            x = threading.Thread(target=search_in_doc, args=(
query_embed, sentences_embeddings[index_left:index_right],
best_sent_documents, id_vect[index_left:index_right], lock))

            threads.append(x)
            x.start()

        for thread in threads:
            thread.join()


        best_sent_documents = sorted(best_sent_documents, key=lambda x:
 x[0], reverse=True)

        doc_sent_title = database.retrieve_sentence_and_doctitle(
best_sent_documents)

        set_doc_id = set()

        for ((doc_id, sentence, title), (similarity, embedding)) in zip
(doc_sent_title, best_sent_documents):
            new_entry = {
                "id": doc_id,
                "title": title,
                "text": sentence,
                "score": similarity
            }
            if doc_id in results.keys():
                results[doc_id]["text"] += " " + new_entry["text"]
                results[doc_id]["score"] += new_entry["score"]
            else:
                results[doc_id] = new_entry

        results = sorted(results.values(), key=lambda x: x['score'],
reverse=True)

    return list(results)


# thread function to look into documents
```

```
def search_in_doc(query_embed, sentence_embeddings, best_doc, id_vect,
    lock):

    # make the transpose for the dot product
    start = time.time()
    sentence_embeddings = sentence_embeddings.transpose()
    check = time.time() - start

    # compute the distances using the dot product
    distances = np.dot(query_embed, sentence_embeddings)
    distances = np.squeeze(np.asarray(distances))

    # sort the sentences based on the similarity
    distances = zip(distances, id_vect)
    distances = sorted(distances, key=lambda x: x[0], reverse=True)

    # store in the data structure common to all the threads
    lock.acquire()
    print(NUM_DOCUMENTS)
    for i in range(50):
        best_doc.append(distances[i])
    lock.release()


# response creation with sentence BERT

def reply_with_sentence_BERT(infineonBERT, text, sentences_embedings,
    id_vect):

    list_query_embeds = convert_query_to_embedding_Sentence_BERT(
    infineonBERT.encoder, [text])

    render_data = multiprocess_search(list_query_embeds,
    sentences_embedings, id_vect)

    return render_data


# returns a list of dictionaries, where every dictionary represents the
      results of search keyword
def reply_with_Digital_Reference(text,
    digital_reference_classes_and_names):

    keywords = searchWithDR.extract_keywords_from_query(text)
    annotations = []

    for keyword in keywords:
        annotation = searchWithDR.search_class_annotations(keyword,
    digital_reference_classes_and_names)

        if annotation['entity'] != "":

            annotations.append(annotation)


    return annotations
```

model.py

```python

class InfineonBERT:
    def __init__(self, name):
        self.name = name
        self.PATH_TO_SENTENCE_TRANSFORMER = 'path/to/sentence/transformer'
        self.PATH_TO_PDFs = 'path/to/pdf'
        self.files_sentence_embeddings = self.load_embeddings()
        self.encoder = self.load_Sentence_Transformer()

    def load_embeddings(self):

        logging.debug("Loading embeddings.")

        count = database.count_sentences()
        conn = database.connect()
        cur = conn.cursor()

        sentences_embeddings = []
        id_vect = []

        for index in tqdm(range(1, count + 1)):
            id, vect = database.retrieve_embedding(cur, index)
            vect = np.array(vect)
            sentences_embeddings.append(vect/np.linalg.norm(vect))
            id_vect.append(id)

        sentences_embeddings = np.asmatrix(sentences_embeddings)
        return sentences_embeddings, id_vect



    def load_Sentence_Transformer(self):
        device = 'cuda' if torch.cuda.is_available() else 'cpu'
        model = SentenceTransformer(self.PATH_TO_SENTENCE_TRANSFORMER)
        model.to(device)
        return model
```

database.py

```python

imports ...

def connect():
    conn = psycopg2.connect(host="localhost", database="search_engine", user="postgres", password="postgres")
    return conn

def create_tables():
    conn = connect()
    try:

        cur = conn.cursor()
        commands = (
            """
                CREATE TABLE documents (
                doc_id SERIAL PRIMARY KEY,
                title VARCHAR(65535) NOT NULL,
                hook VARCHAR(65535) NOT NULL,
                document BYTEA
            )
            """,
            """
                CREATE TABLE sentences (
                sentence_id SERIAL PRIMARY KEY,
                sentence VARCHAR(65535) NOT NULL,
                embedding float[] NOT NULL,
                doc_id INTEGER NOT NULL,
                FOREIGN KEY (doc_id)
                        REFERENCES documents (doc_id)
                        ON UPDATE CASCADE ON DELETE CASCADE
            )
            """,
            """
                CREATE TABLE pickle (
                pickle_id INTEGER PRIMARY KEY,
                pickle_file BYTEA
            )
            """
        )

        for command in commands:
            cur.execute(command)

        conn.commit()

        logging.debug("Tables created.")

        cur.close()


    except (Exception, psycopg2.DatabaseError) as error:
        logging.error(error)


    finally:

        conn.close()


def insert_data(sentence_embeddings):
    conn = connect()
    try:


        cur = conn.cursor()

        sql_documents = """
                INSERT INTO documents (title, hook, document)
                VALUES (%s, %s, %s)
                RETURNING doc_id;
```

```
                    """

        sql_sentences = """
                    INSERT INTO sentences (sentence, embedding, doc_id)
                    VALUES (%s, %s, %s)
                    RETURNING sentence_id;
                    """

        sql_pickle = """
                    INSERT INTO pickle (pickle_id, pickle_file)
                    VALUES (%s, %s);
        """

        print("Inserting data..")

        for b in range(0, 10):
            with open("C:/Users/Mottola/Documents/InfineonSearchEngine/
    sentence_embeddings_" + str(b) + ".pickle", "rb") as pickle_file:
                p = pickle_file.read()
            cur.execute(sql_pickle, (b, p))

        for doc in sentence_embeddings:
            print("\t"+doc['title'])
            with open(PATH_TO_PDF + doc['hook'] + ".pdf", "rb") as f:
                pdf = f.read()
            cur.execute(sql_documents, (doc['title'], doc['hook'], pdf)
    )
            doc_id = cur.fetchone()[0]
            # logging.debug(doc_id)

            for (sent, emb) in zip(doc['sentences'], doc['embeddings'])
    :
                cur.execute(sql_sentences, (sent, emb.tolist(), doc_id)
    )
                sentence_id = cur.fetchone()[0]
                # logging.debug(sentence_id)

        conn.commit()

        logging.debug("Data inserted.")


    except (Exception, psycopg2.DatabaseError) as error:

        logging.error(error)


    finally:

        conn.close()



def retrieve_embeddings_from_pickle():

    try:

        conn = connect()

        cur = conn.cursor()

        query = """
            SELECT pickle_file
            FROM pickle
            WHERE pickle_id = %s;
        """

        list_embeddings = []
        for index in range(10):
            logging.debug("Executing query {} ...".format(index))

            cur.execute(query, str(index))

            logging.debug("Executed!")
            list_embeddings += pickle.loads(cur.fetchone()[0])

        return list_embeddings

    except (Exception, psycopg2.DatabaseError) as error:

        logging.error(error)


    finally:

        conn.close()


def retrieve_sentence_and_doctitle(N):
    try:
        conn = connect()
        cur = conn.cursor()

        result = []

        for sentence_id in N:
            query = """
                    SELECT sentences.doc_id, sentence, hook
                    FROM documents, sentences
                    WHERE sentence_id = {} AND
                        sentences.doc_id = documents.doc_id;
                """
            cur.execute(query.format(sentence_id[1]))
            result.append(cur.fetchone())

        return result

    except (Exception, psycopg2.DatabaseError) as error:

        logging.error(error)


    finally:

        conn.close()

```

```

def retrieve_embeddings_from_sentences():

    try:
        conn = connect()
        cur = conn.cursor()

        result = []

        count_query = """
            SELECT COUNT(*) FROM sentences;
        """
        cur.execute(count_query)
        count = cur.fetchone()[0]

        for index in tqdm(range(1,count+1)):
            query = """
                    SELECT sentence_id, embedding
                    FROM sentences
                    WHERE sentence_id = {};
                """

            cur.execute(query.format(index))
            result.append(cur.fetchone())

        return result

    except (Exception, psycopg2.DatabaseError) as error:

        logging.error(error)


    finally:

        conn.close()


def retrieve_embedding(cur, index):

    try:
        query = """
                SELECT sentence_id, embedding
                FROM sentences
                WHERE sentence_id = {};
            """
        cur.execute(query.format(index))

        return cur.fetchone()

    except (Exception, psycopg2.DatabaseError) as error:

        logging.error(error)



def count_sentences():

    try:
        conn = connect()
        cur = conn.cursor()

        result = []

        count_query = """
            SELECT COUNT(*) FROM sentences;
        """
        cur.execute(count_query)
        count = cur.fetchone()[0]

        return count

    except (Exception, psycopg2.DatabaseError) as error:

        logging.error(error)


    finally:

        conn.close()


if __name__ == "__main__":

    with open(PATH_TO_PICKLE_DATA, 'rb') as f:
        files_s_embeddings = pickle.load(f)

    create_tables()
    insert_data(files_s_embeddings)
```

searchWithDR.py

```

import requests
import pandas as pd
import re
from owlready2 import *
import spacy
import time
from fuzzywuzzy import fuzz
from fuzzywuzzy import process

PATH_TO_ONTOLOGY = "C:\\Users\\Mottola\\Documents\\ontologies\\
    DigitalReferenceUpdatedDescriptionNoIndividuals.owl"
PATH_SAVED_ONTOLOGY = "C:\\Users\\Mottola\\Documents\\ontologies\\
    AnnotatedDigitalReference_v2.owl"


# query the DR to retrieve all the triples with ObjectProperty
def queryDigitalReference():

    response = requests.post('http://localhost:3030/DigitalReference/
    sparql',
                             data = {
                                 'query': """prefix owl: <http://www.w3.org
    /2002/07/owl#>
```

```
                                prefix rdfs: <http://www.w3.org
    /2000/01/rdf-schema#>


                                SELECT DISTINCT ?domain ?dl ?dd ?
    predicate ?range ?rl ?rd
                                WHERE {
                                  ?predicate a owl:ObjectProperty .
                                  ?predicate rdfs:domain ?domain .
                                  ?predicate rdfs:range ?range .
                                  ?domain a owl:Class .
                                  ?range a owl:Class .
                                  FILTER (!isBlank(?predicate))
                                  FILTER (!isBlank(?domain))
                                  FILTER (!isBlank(?range))
                                  OPTIONAL { ?domain rdfs:label ?
    domlabel .
                                    BIND(STR(?domlabel) as ?dl)}
                                  OPTIONAL { ?domain rdfs:comment ?
    domdescription .
                                    BIND(STR(?domdescription) as ?dd)}
                                  OPTIONAL { ?range rdfs:label ?
    rangelabel .
                                    BIND(STR(?rangelabel) as ?rl)}
                                  OPTIONAL { ?range rdfs:comment ?
    rangedescription .
                                    BIND(STR(?rangedescription) as ?rd)
    }

                                }"""
                            })
    return response.json()


# retrieve the response and organizes it in a data frame
def create_data_frame(response_list):
    domain = [] # domain entity
    dl = [] # domain label
    dd = [] # domain description
    predicate = [] # predicate entity
    range = [] # range entity
    rl = [] # range label
    rd = [] # range description
    for x in response_list['results']['bindings']:
        #try:
        domain.append(x[response_list['head']['vars'][0]]['value'])
        #except KeyError:
        #    domain.append("")
        try:
            dl.append(x[response_list['head']['vars'][1]]['value'])
        except KeyError:
            dl.append("")
        try:
            dd.append(x[response_list['head']['vars'][2]]['value'])
        except KeyError:
            dd.append("")
        #try:
        predicate.append(x[response_list['head']['vars'][3]]['value'])
        #except:
        #     predicate.append("")
        #try:
        range.append(x[response_list['head']['vars'][4]]['value'])
        #except KeyError:
        #     range.append("")
        try:
            rl.append(x[response_list['head']['vars'][5]]['value'])
        except:
            rl.append("")
        try:
            rd.append(x[response_list['head']['vars'][6]]['value'])
        except:
            rd.append("")

    dict = {
        response_list['head']['vars'][0]: domain,
        response_list['head']['vars'][1]: dl,
        response_list['head']['vars'][2]: dd,
        response_list['head']['vars'][3]: predicate,
        response_list['head']['vars'][4]: range,
        response_list['head']['vars'][5]: rl,
        response_list['head']['vars'][6]: rd
    }
    return pd.DataFrame(dict)

# extracts the name of the entities, together with getSplittedText
def getClassName(classEntity):
    if len(classEntity.split("#")) == 2:
        return classEntity.split("#")[1]
    else:
        return "NOT A NAME"


def getSplittedText(name):
    # convert CamelCase and  syntax into normal strings
    splitted = re.sub('([A-Z][a-z]+)', r' \1', re.sub('([A-Z]+)', r' \1
    ', name)).split()
    splitted = [x.replace("_", " ") for x in splitted]
    return ' '.join(splitted)


# create the sentences out of triples
def create_string_from_triples(df):
    queries = []
    for index, row in df.iterrows():
        subject = getSplittedText(getClassName(row[0]))
        predicate = getSplittedText(getClassName(row[3]))
        obj = getSplittedText(getClassName(row[4]))
        sentence = subject + " " + predicate + " " + obj
        queries.append((owlready2.IRIS[row[0]], sentence))

    return queries



def importOntology(ontology):
    onto = owlready2.get_ontology(ontology).load()
```

```python
    return onto


def insert_annotations(onto, entity, annotations):
    #print(entity)
    titles = []
    for doc in annotations:
        titles.append(doc['title'])
    entity.seeAlso = titles
    #print(entity.seeAlso)


def extract_keywords_from_query(query):
    nlp = spacy.load("en_core_web_sm")
    stop_words = nlp.Defaults.stop_words
    stop_words.add('?')
    stop_words.add(',')
    stop_words.add('.')
    stop_words.add('!')
    start = time.time()
    # sentence = "which is the relation between the digital reference and semantic web?"
    # sentence1 = "how can heatmap and convolutional neural networks be related?"
    doc = nlp(query.lower())
    noun_phrases = [chunk.text for chunk in doc.noun_chunks]
    print("Noun phrases:", noun_phrases)
    remove_stop_words = [[word if str(word) not in stop_words else ',' for word in nlp(sent)] for sent in noun_phrases]
    remove_stop_words = [' '.join(str(x) for x in sent).split(',') for sent in remove_stop_words]
    final_list = []
    for a in remove_stop_words:
        final_list += [x.strip() for x in list(filter(lambda x: x != '', a))]
    print("Final result", final_list)
    print(time.time() - start)
    return final_list


# return a dictionary, were every dictionary represents the documents for one keyword
# every dictionary has 'entityname', 'definition' and a list of 'related_works', containing the list of documents
def search_class_annotations(keyword, digital_reference_classes_and_names):
    document = {
        'keyword': keyword,
        'entity': "",
        'definition': "",
        'related_works': []
    }
    for (entity, name) in digital_reference_classes_and_names:
        if fuzz.ratio(keyword, name) > 80:
            #print("Fuzzy match: DR " + name + "- Key: " + keyword)
            #print("\t\tMetadata of " + str(entity) + ":" + str(entity.
    seeAlso))
            document['keyword'] = entity.name
            document['entity'] = IRIS[entity]
            document['definition'] = entity.comment
            document['related_works'] = entity.seeAlso
    return document
```

annotateDR.py

```

imports ...

def importOntology():
    onto = owlready2.get_ontology(PATH_TO_ONTOLOGY).load()
    return onto


def getClasses(ontology):
    return list(ontology.classes())


def getProperties(ontology):
    return [x for x in list(ontology.properties())]


def getClassName(classes):
    # access the uris of the classes, and split them to get the
    classNames if they exist after the #
    return [(x, x.iri.split("#")[1]) for x in classes if len(x.iri.
    split("#")) == 2]


def getSplittedText(name):
    # convert CamelCase and  syntax into normal strings
    splitted = re.sub('([A-Z][a-z]+)', r' \1', re.sub('([A-Z]+)', r' \1
    ', name)).split()
    splitted = [x.replace("_", " ") for x in splitted]
    return ' '.join(splitted)

def get_tuple_class_name(ontology):
    digital_reference_classes_and_names = []
    for index, (entity, name) in enumerate(getClassName(getClasses(
    ontology))):
        digital_reference_classes_and_names.append((entity,
    getSplittedText(name)))
    return digital_reference_classes_and_names


def get_ontology_triples(ontology, classes):
    triples = list(ontology.get_triples(None, None, None))
    for triple in triples:
        for el in triple:
            print(ontology._unabbreviate(el))
    return triples

```

```
def attachDocumentsToClasses(infineonBERT, listOfTuples):

    for index, (classObj, className) in enumerate(listOfTuples):
        text = getSplittedText(className)
        print("{} - Working on class {}".format(index, text))
        list_query_embeds = service.
    convert_query_to_embedding_Sentence_BERT(infineonBERT.encoder, [
    text])
        documents = service.multiprocess_search(text, list_query_embeds
    , infineonBERT.files_sentence_embeddings)
        for doc in documents:
            try:
                #print("\t\t" + str(doc['title']) + "\t\t" + str(doc['
    score'])) # mettere a posto l'encoding
                classObj.comment.append(doc['title'])
            except UnicodeEncodeError:
                #print("\t\t" + "**Impossible to print the title**")
                pass


def extract_keywords_from_query(query):
    nlp = spacy.load("en_core_web_sm")
    stop_words = nlp.Defaults.stop_words
    stop_words.add('?')
    stop_words.add(',')
    stop_words.add('.')
    stop_words.add('!')
    start = time.time()
    # sentence = "which is the relation between the digital reference
    and semantic web?"
    # sentence1 = "how can heatmap and convolutional neural networks be
     related?"
    doc = nlp(query)
    noun_phrases = [chunk.text for chunk in doc.noun_chunks]
    print("Noun phrases:", noun_phrases)
    remove_stop_words = [[word if str(word) not in stop_words else ','
    for word in nlp(sent)] for sent in noun_phrases]
    remove_stop_words = [' '.join(str(x) for x in sent).split(',') for
    sent in remove_stop_words]
    final_list = []
    for a in remove_stop_words:
        final_list += [x.strip() for x in list(filter(lambda x: x != ''
    , a))]
    print("Final result", final_list)
    print(time.time() - start)
    return final_list


# import the Digital Reference
digitalReferenceOntology = importOntology()
print(digitalReferenceOntology.graph.dump())

# retrieve the classes of the digital reference
triples = digitalReferenceOntology.get_triples(None, None, None)
for triple in triples:
    for el in triple:
        digitalReferenceOntology._unabbreviate(el)
```

```
# get a list of tuples: (the class object, the names from the uri)
classTuple = getClassName(triples)

# instance of infineonBERT
infineonBERT = model.InfineonBERT("Infineon search engine")

start = time.time()
# attach documents to classes
attachDocumentsToClasses(infineonBERT, classTuple)


# print(lambda x: print(x.comment) for x in getClasses(
    digitalReferenceOntology))
print("Time needed to annotate all the classes: {}".format(time.time()
    - start))

# save the ontology to file with RDF/XML format
with open(PATH_SAVED_ONTOLOGY, "wb") as fp:
    digitalReferenceOntology.save(file=fp)
```

Bibliography

[1] Hans Ehm et al. “Digital Reference–A Semantic Web for Semiconductor Manufacturing and Supply Chains Containing Semiconductors”. In: *2019 Winter Simulation Conference (WSC)*. IEEE. 2019, pp. 2409–2418.

[2] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[3] Nils Reimers and Iryna Gurevych. “Sentence-bert: Sentence embeddings using siamese bert-networks”. In: *arXiv preprint arXiv:1908.10084* (2019).

[4] J. Rydning D. Reinsel J. Gantz. *White paper: The Digitization of the World - from Edge to Core. An IDC White Paper*. Tech. rep. IDC Corporate USA, 2018. URL: `https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf`.

[5] KV Kanimozhi and M Venkatesan. “Unstructured data analysis-a survey”. In: *International Journal of Advanced Research in Computer and Communication Engineering* 4.3 (2015), pp. 223–225.

[6] *Infineon Technologies AG*. https://www.infineon.com/cms/en/about-infineon/company/.

[7] Tim Berners-Lee, James Hendler, and Ora Lassila. “The semantic web”. In: *Scientific american* 284.5 (2001), pp. 34–43.

[8] Grigoris Antoniou and Frank Van Harmelen. *A semantic web primer*. MIT press, 2004.

[9] Elizabeth D Liddy. “Natural language processing”. In: (2001).

[10] Yoav Goldberg. “Neural network methods for natural language processing”. In: *Synthesis Lectures on Human Language Technologies* 10.1 (2017), pp. 1–309.

[11] Vineet Chaitanya, Rajeev Sangal, and Akshar Bharati. *Natural language processing: a Paninian perspective*. Prentice-Hall of India, 1996.

[12] *Difference between NLU and NLP*. `https://nlp.stanford.edu/~wcmac/papers/20140716-UNLU.pdf`. Accessed: 2020-06-15.

[13] Veton Kepuska and Gamal Bohouta. “Next-generation of virtual personal assistants (microsoft cortana, apple siri, amazon alexa and google home)”. In: *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE. 2018, pp. 99–103.

[14] Navdeep S Dhillon and Jisheng Liang. *NLP-based sentiment analysis*. US Patent 8,838,633. 2014.

[15] Craig Lewis and Steven Young. “Fad or future? Automated analysis of financial text and its implications for corporate reporting”. In: *Accounting and Business Research* 49.5 (2019), pp. 587–615.

[16] Peter Jackson and Isabelle Moulinier. *Natural language processing for online applications: Text retrieval, extraction and categorization.* Vol. 5. John Benjamins Publishing, 2007.

[17] Prakash M Nadkarni, Lucila Ohno-Machado, and Wendy W Chapman. “Natural language processing: an introduction”. In: *Journal of the American Medical Informatics Association* 18.5 (2011), pp. 544–551.

[18] Noam Chomsky and David W Lightfoot. *Syntactic structures.* Walter de Gruyter, 2002.

[19] Terry Winograd. “Understanding natural language”. In: *Cognitive psychology* 3.1 (1972), pp. 1–191.

[20] William A Woods. “Progress in natural language understanding: an application to lunar geology”. In: *Proceedings of the June 4-8, 1973, national computer conference and exposition.* 1973, pp. 441–450.

[21] Eric Brill and Raymond J Mooney. “An overview of empirical natural language processing”. In: *AI magazine* 18.4 (1997), pp. 13–13.

[22] Li Deng and Yang Liu. *Deep learning in natural language processing.* Springer, 2018.

[23] Mitchell Marcus. “New trends in natural language processing: statistical natural language processing”. In: *Proceedings of the National Academy of Sciences* 92.22 (1995), pp. 10052–10059.

[24] Luca Dini, Vittorio Di Tomaso, and Frédérique Segond. “Error driven word sense disambiguation”. In: *COLING 1998 Volume 1: The 17th International Conference on Computational Linguistics.* 1998.

[25] Eric Brill and Philip Resnik. “A rule-based approach to prepositional phrase attachment disambiguation”. In: *arXiv preprint cmp-lg/9410026* (1994).

[26] Christopher D Manning, Christopher D Manning, and Hinrich Schütze. *Foundations of statistical natural language processing.* MIT press, 1999.

[27] Yoshua Bengio et al. “A neural probabilistic language model”. In: *Journal of machine learning research* 3.Feb (2003), pp. 1137–1155.

[28] Tomas Mikolov et al. “Distributed representations of words and phrases and their compositionality”. In: *Advances in neural information processing systems.* 2013, pp. 3111–3119.

[29] Tomas Mikolov et al. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).

[30] Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. “Linguistic regularities in continuous space word representations”. In: *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies.* 2013, pp. 746–751.

[31] Xin Rong. “word2vec parameter learning explained”. In: *arXiv preprint arXiv:1411.2738* (2014).

[32] Ehsaneddin Asgari and Mohammad RK Mofrad. "Continuous distributed representation of biological sequences for deep proteomics and genomics". In: *PloS one* 10.11 (2015).

[33] Quoc Le and Tomas Mikolov. "Distributed representations of sentences and documents". In: *International conference on machine learning.* 2014, pp. 1188–1196.

[34] Jeffrey Pennington, Richard Socher, and Christopher D Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP).* 2014, pp. 1532–1543.

[35] Joo-Kyung Kim et al. "Intent detection using semantically enriched word embeddings". In: *2016 IEEE Spoken Language Technology Workshop (SLT).* IEEE. 2016, pp. 414–419.

[36] Asnat Greenstein-Messica, Lior Rokach, and Michael Friedman. "Session-based recommendations using item embedding". In: *Proceedings of the 22nd International Conference on Intelligent User Interfaces.* 2017, pp. 629–633.

[37] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. "A simple but tough-to-beat baseline for sentence embeddings". In: (2016).

[38] Ziyi Yang, Chenguang Zhu, and Weizhu Chen. "Parameter-free sentence embedding via orthogonal basis". In: *arXiv preprint arXiv:1810.00438* (2018).

[39] Michael I Jordan. "Attractor dynamics and parallelism in a connectionist sequential machine". In: *Artificial neural networks: concept learning.* 1990, pp. 112–127.

[40] Jeffrey L Elman. "Finding structure in time". In: *Cognitive science* 14.2 (1990), pp. 179–211.

[41] Zachary Chase Lipton. "A Critical Review of Recurrent Neural Networks for Sequence Learning". In: *CoRR* abs/1506.00019 (2015). arXiv: `1506.00019`. URL: `http://arxiv.org/abs/1506.00019`.

[42] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. "An empirical exploration of recurrent network architectures". In: *International conference on machine learning.* 2015, pp. 2342–2350.

[43] Ilya Sutskever. *Training recurrent neural networks.* University of Toronto Toronto, Ontario, Canada, 2013.

[44] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation.* Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[45] Paul J Werbos. "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.

[46] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[47] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *International conference on machine learning.* 2013, pp. 1310–1318.

[48] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[49] Kyunghyun Cho et al. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *arXiv preprint arXiv:1406.1078* (2014).

[50] MingxianLin.

[51] Ryan Kiros et al. “Skip-thought vectors”. In: *Advances in neural information processing systems*. 2015, pp. 3294–3302.

[52] Alexis Conneau et al. “Supervised learning of universal sentence representations from natural language inference data”. In: *arXiv preprint arXiv:1705.02364* (2017).

[53] Sebastian Ruder. *NLP-progress: Repository to track the progress in Natural Language Processing (NLP), including the datasets and the current state-of-the-art for the most common NLP tasks.* URL: `http://nlpprogress.com/`. (accessed: 22.06.2020).

[54] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473* (2014).

[55] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[56] ME Peters et al. “Deep contextualized word representations. arXiv 2018”. In: *arXiv preprint arXiv:1802.05365* (1802).

[57] Alec Radford et al. “Improving language understanding with unsupervised learning”. In: *Technical report, OpenAI* (2018).

[58] Yonghui Wu et al. “Google’s neural machine translation system: Bridging the gap between human and machine translation”. In: *arXiv preprint arXiv:1609.08144* (2016).

[59] Rowan Zellers et al. “Swag: A large-scale adversarial dataset for grounded commonsense inference”. In: *arXiv preprint arXiv:1808.05326* (2018).

[60] Yukun Zhu et al. “Aligning books and movies: Towards story-like visual explanations by watching movies and reading books”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 19–27.

[61] Jinhyuk Lee et al. “BioBERT: a pre-trained biomedical language representation model for biomedical text mining”. In: *Bioinformatics* 36.4 (2020), pp. 1234–1240.

[62] Iz Beltagy, Kyle Lo, and Arman Cohan. “SciBERT: A pretrained language model for scientific text”. In: *arXiv preprint arXiv:1903.10676* (2019).

[63] Waleed Ammar et al. “Construction of the literature graph in semantic scholar”. In: *arXiv preprint arXiv:1805.02262* (2018).

[64] Taku Kudo and John Richardson. “Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing”. In: *arXiv preprint arXiv:1808.06226* (2018).

[65] Samuel R Bowman et al. “The SNLI Corpus”. In: (2015).

[66] Adina Williams, Nikita Nangia, and Samuel R Bowman. "The multi-genre nli corpus". In: (2018).

[67] "Chapter Three - RDF and the Semantic Web Stack". In: *RDF Database Systems*. Ed. by Olivier Curé and Guillaume Blin. Boston: Morgan Kaufmann, 2015, pp. 41 –80. ISBN: 978-0-12-799957-9. DOI: `https://doi.org/10.1016/B978-0-12-799957-9.00003-1`. URL: `http://www.sciencedirect.com/science/article/pii/B9780127999579000031`.

[68] Archana Patel and Sarika Jain. "Present and future of semantic web technologies: a research statement". In: *International Journal of Computers and Applications* (2019), pp. 1–10.

[69] Tim Berners-Lee, Roy Fielding, Larry Masinter, et al. *Uniform resource identifiers (URI): Generic syntax*. 1998.

[70] Frank Manola, Eric Miller, Brian McBride, et al. "RDF primer". In: *W3C recommendation* 10.1-107 (2004), p. 6.

[71] Dan Brickley, Ramanathan V Guha, and Andrew Layman. "Resource description framework (RDF) schema specification". In: (1999).

[72] Alexander Maedche and Steffen Staab. "Ontology learning for the semantic web". In: *IEEE Intelligent systems* 16.2 (2001), pp. 72–79.

[73] *ECSEL Joint Undertaking. 2019. What We Do . . . and How.* https://www.ecsel.eu/what-we-do-and-how/.

[74] *Productive4.0 Consortium. 2019. Productive4.0 is a European Co-funded Innovation and Lighthouse Program.* https://productive40.eu/about/.

[75] Diana Maynard, Kalina Bontcheva, and Isabelle Augenstein. "Natural language processing for the semantic web". In: *Synthesis Lectures on the Semantic Web: Theory and Technology* 6.2 (2016), pp. 1–194.

[76] Kalina Bontcheva, Valentin Tablan, and Hamish Cunningham. "Semantic search over documents and ontologies". In: *PROMISE Winter School*. Springer. 2013, pp. 31–53.

[77] Giorgos Giannopoulos et al. "GoNTogle: a tool for semantic annotation and search". In: *Extended Semantic Web Conference*. Springer. 2010, pp. 376–380.

[78] Borislav Popov et al. "KIM–semantic annotation platform". In: *International Semantic Web Conference*. Springer. 2003, pp. 834–849.

[79] Matthew Honnibal and Ines Montani. "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing". To appear. 2017.

[80] *PyPDF2*. http://mstamy2.github.io/PyPDF2/.

[81] *Foxit Phantom PDF - PDF Editor*. https://www.foxitsoftware.com/pdf-editor/.

[82] *Kofax Power PDF*. https://www.kofax.com/. 2020.

[83] Alexandru Constantin, Steve Pettifer, and Andrei Voronkov. "PDFX: Fully-Automated PDF-to-XML Conversion of Scientific Literature". In: *Proceedings of the 2013 ACM Symposium on Document Engineering.* DocEng '13. Florence, Italy: Association for Computing Machinery, 2013, 177–180. ISBN: 9781450317894. DOI: 10.1145/2494266.2494271. URL: https://doi.org/10.1145/2494266.2494271.

[84] *Smallpdf.* https://smallpdf.com/pdf-converter.

[85] *Tensorflow v 1.11.0.* https://github.com/tensorflow/docs/tree/r1.11/site/en/api$_d$*ocs*.

[86] *PyTorch v1.6.0.* https://pytorch.org/docs/stable/index.html.

[87] Hugging Face. *The Big-&-Extending-Repository-of-Transformers: Pretrained PyTorch models for Google's BERT, OpenAI GPT & GPT-2, Google/CMU Transformer-XL.,(nd).*

[88] Guadalupe Aguado-de Cea et al. "Lexicalizing Ontologies: The issues behind the labels". In: *Procedia-Social and Behavioral Sciences* 212 (2015), pp. 151–158.

[89] Barry Smith et al. "The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration". In: *Nature biotechnology* 25.11 (2007), pp. 1251–1255.

[90] *World Wide Web Consortium, RDF Schema 1.1 W3C Recommendation 25 February 2014.* https://www.w3.org/TR/rdf-schema/ch$_l$*abel*.

[91] *Owlready2.* https://pythonhosted.org/Owlready2/.

[92] Channu Kambalyal. "3-tier architecture". In: *Retrieved On* 2 (2010), p. 34.

[93] Sumi Helal et al. "A three-tier architecture for ubiquitous data access". In: *Proceedings ACS/IEEE International Conference on Computer Systems and Applications.* IEEE. 2001, pp. 177–180.

[94] *FuzzyWuzzy.* https://github.com/seatgeek/fuzzywuzzy.

[95] Vladimir I Levenshtein. "Binary codes capable of correcting deletions, insertions, and reversals". In: *Soviet physics doklady.* Vol. 10. 8. 1966, pp. 707–710.

[96] Frederic P Miller, Agnes F Vandome, and John McBrewster. "Levenshtein distance: Information theory, computer science, string (computer science), string metric, damerau? Levenshtein distance, spell checker, hamming distance". In: (2009).

[97] Katja Hofmann, Lihong Li, and Filip Radlinski. "Online evaluation for information retrieval". In: *Foundations and Trends in Information Retrieval* 10.1 (2016), pp. 1–117.

[98] Ye Chen et al. "Meta-evaluation of online and offline web search evaluation metrics". In: *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval.* 2017, pp. 15–24.

[99] Mark Sanderson. "Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008. ISBN-13 978-0-521-86571-5, xxi+ 482 pages." In: *Natural Language Engineering* 16.1 (2010), pp. 100–103.

[100] *APICS - SCOR model.* http://www.apics.org/.

[101] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. “Reformer: The efficient transformer”. In: *arXiv preprint arXiv:2001.04451* (2020).