NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

MSc THESIS

QUESTION ANSWERING ENGINE OVER KNOWLEDGE GRAPH OF WIKIDATA

DIMITRIOS T. ORINOS

Supervisor: **Manolis Koubarakis,** Professor

Co-Supervisor: **Konstantinos Plas,** Postgraduate student

ATHENS

JUNE 2024

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΗΧΑΝΗ ΑΠΑΝΤΗΣΗΣ ΕΡΩΤΗΣΕΩΝ ΠΑΝΩ ΣΤΟ ΓΝΩΣΙΑΚΟ ΓΡΑΦΟ ΤΗΣ WIKIDATA

ΔΗΜΗΤΡΙΟΣ Θ. ΟΡΕΙΝΟΣ

Επιβλέπων: **Μανώλης Κουμπαράκης,** Καθηγητής

Συνεπιβλέπων: **Κωνσταντίνος Πλάς,** Μεταπτυχιακός Φοιτητής

ΑΘΗΝΑ

ΙΟΥΝΙΟΣ 2024

MSc THESIS

QUESTION ANSWERING ENGINE OVER KNOWLEDGE GRAPH OF WIKIDATA

DIMITRIOS T. ORINOS

S.N.: 7115132100006

SUPERVISOR: **Manolis Koubarakis,** Professor

COSUPERVISOR: **Konstantinos Plas,** Postgraduate student

THESIS COMMITTEE: **Manolis Koubarakis,** Professor
Alexandros Ntoulas, Professor
Stathes Hadjiefthymiades, Professor

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΗΧΑΝΗ ΑΠΑΝΤΗΣΗΣ ΕΡΩΤΗΣΕΩΝ ΠΑΝΩ ΣΤΟ ΓΝΩΣΙΑΚΟ ΓΡΑΦΟ ΤΗΣ WIKIDATA

ΔΗΜΗΤΡΙΟΣ Θ. ΟΡΕΙΝΟΣ
Α.Μ.: 7115132100006

ΕΠΙΒΛΕΠΩΝ: **Μανώλης Κουμπαράκης,** Καθηγητής

ΣΥΝΕΠΙΒΛΕΠΩΝ: **Κωνσταντίνος Πλάς,** Μεταπτυχιακός Φοιτητής

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: **Μανώλης Κουμπαράκης,** Καθηγητής
Αλέξανδρος Ντούλας, Καθηγητής
Στάθης Χατζηευθυμιάδης, Καθηγητής

ABSTRACT

The main goal of my thesis is the creation of a question answering engine that is able to answer simple questions from the dataset SimpleQuestions, which is relying on the Freebase and it is translated into Wikidata, where as a result each question of the dataset consists of an entity id and a relation id from this translation. In order to achieve this goal, firstly i find through SPARQL the entity label for each question by using the entity id which was resulted from the translation of the SimpleQuestions dataset to Wikidata. Secondly i create a relation vocabulary which consists of the unique relation ids that exists in the training and the validation dataset of the SimpleQuestions dataset. Later i find the index of each unique relation id of the relation vocabulary in it. Next, i create a neural network consisting of a bert transformer layer, a linear layer, an activation and an dropout layer with the output neurons equal to 129, as long as the length of the relation vocabulary. The relation indexes of the unique relation ids combined with this network conduce in order to predict the index for the relation id for each question of training and validation dataset of SimpleQuestions dataset. Also i create a similar neural network with two outputs one for the entity label start word index and one for entity label end word index with output neurons equal to 33 each, which is the biggest length is found for all SimpleQuestions dataset questions. In order to find the entity label start and entity label end index for each training and the validation dataset entity label, i split the training and the validation dataset questions into parts with length equal to its corresponding entity labels length and i create for each subset (training, validation) question by using the equal question part with the entity label, a list that consists of 0 and 1, where 1 is the case when the entity label word belongs to the entity label and the corresponding question and 0 if not. I also tried to find and later predict these indexes through text similarity either with cosine similarity by using spacy library combined with the Sentence Transformer model of the Transformers library or jaccard similarity metric by using only spacy library. These entity span indexes combined the entity span neural network that i created conduce in order to predict the start and the end index of the entity label in the corresponding question for each SimpleQuestions dataset question. Finally i create some versions of question answering engine by using spacy library, Sentence Transformer combined with the cosine similarity metric and jaccard similarity metric. Finally the best entity span evaluation scores are achieved for the methodology of finding entity span indexes by creating lists consisting of 0 and 1 only for SimpleQuestions training and validation dataset questions that their entity label consists a part of their wording. Also the biggest percentage of totally correct answered questions are observed for the QA engine with python library spacy for the methodology of finding entity span indexes (start,end) through cosine similarity metric and the methodology of finding entity span indexes by creating lists consisting of 0 and 1 in the case of training and validating all SimpleQuestions training and validation dataset questions respectively, regardless if their entity label consists a part of their wording.

SUBJECT AREA: Artificial Intelligence(AI)-Natural Language Processing(NLP)

KEYWORDS: Question Answering engine over Wikidata, entity span and relation index prediction, text similarity metrics, Entity span as sequence of 0 and 1, Sentence Transformers, BERT

ΠΕΡΙΛΗΨΗ

Ο κύριος στόχος της διπλωματικής μου εργασίας είναι η δημιουργία μιας μηχανής απάντησης ερωτήσεων που είναι σε θέση να απαντήσει σε απλές ερωτήσεις από το σύνολο δεδομένων SimpleQuestions, η οποία βασίζεται στη βάση γνώσεων Freebase και μεταφράζεται στη βάση γνώσεων της Wikidata, όπου ως αποτέλεσμα κάθε ερώτηση του συνόλου δεδομένων SimpleQuestions αποτελείται από ένα entity id και ένα relation id από αυτή τη μετάφραση. Για την επίτευξη αυτού του στόχου, αρχικά βρήκα μέσω της SPARQL το entity label για κάθε ερώτηση, χρησιμοποιώντας το entity id το οποίο προέκυψε από τη μετάφραση του συνόλου δεδομένων SimpleQuestions στη Wikidata. Δεύτερον δημιούργησα ένα relation vocabulary το οποίο αποτελείται από τα μοναδικά relation ids που υφίστανται στα σύνολα εκπαίδευσης και επαλήθευσης του συνόλου δεδομένων SimpleQuestions. Αργότερα βρήκα τη θέση (index) του κάθε μοναδικού relation id του relation vocabulary σε αυτό. Έπειτα δημιούργησα ένα νευρωνικό δίκτυο που αποτελείται από ένα επίπεδο bert transformer, ένα γραμμικό (linear) επίπεδο, ένα επίπεδο συνάρτησης ενεργοποίησης (activation) και ένα dropout επίπεδο αριθμό νευρώνων εξόδου ίσο με 129, όσο το μήκος του relation vocabulary. Οι σχεσιακοί (relation) δείκτες των μοναδικών relation ids σε συνδυασμό με αυτό το νευρωνικό δίκτυο συμβάλλουν στο να γίνει η πρόβλεψη το δείκτη για το relation id της κάθε ερώτησης του συνόλου εκπαίδευσης και επαλήθευσης του συνόλου δεδομένων (dataset) SimpleQuestions. Επίσης δημιούργησα ένα όμοιο νευρωνικό δίκτυο με δύο εξόδους, μία για τον δείκτη της αρχικής λέξης του entity label στην αντίστοιχη ερώτηση και μία για τον δείκτη της τελικής λέξης του entity label με αριθμό νευρώνων ίσο με 33 η κάθε έξοδος, που είναι το μεγαλύτερο μήκος ερώτησης από όλες τις ερωτήσεις του συνόλου δεδομένων SimpleQuestions. για να γίνει η πρόβλεψη των δεικτών της αρχικής και της τελικής λέξης του entity label μέσα στην ερώτηση για κάθε ερώτηση του συνόλου δεδομένων SimpleQuestions. Προκειμένου να βρω τον δείκτη της αρχικής και της τελικής λέξης για κάθε entity label του συνόλου εκπαίδευσης και επαλήθευσης, χώρισα τις ερωτήσεις των συνόλων εκπαίδευσης και επαλήθευσης σε τμήματα με μήκος ίσο με αυτό του αντίστοιχου τους entity label και δημιούργησα για κάθε ερώτηση αυτών των υποσυνόλων χρησιμοποιώντας, μια λίστα που αποτελείται από 0 και 1, όπου το 1 υπάρχει μόνο στις αντίστοιχες θέσεις της λίστας όπου το entity label ανήκει στην αντίστοιχη ερώτηση αν το entity label και η αντίστοιχη ερώτηση διασπαστούν σε λέξεις και 0 εαν δεν ανήκει. Επίσης προσπάθησα να βρω αυτούς τους δείκτες μέσω ομοιότητας κειμένου είτε με τη μετρική του jaccard similarity είτε με τη μετρική του cosine similarity χρησιμοποιώντας τη βιβλιοθήκη spacy συνδυαζόμενη με το μοντέλο του Sentence Transformer της βιβλιοθήκης των transformer είτε με τη μετρική του cosine similarity χρησιμοποιώντας μόνο τη βιβλιοθήκη spacy. Αυτοί οι entity span δείκτες σε συνδυασμό με το νευρωνικό δίκτυο για τη πρόβλεψη του entity span που δημιούργησα συμβάλλουν ώστε να γίνει η πρόβλεψη the start and the end index of the entity label in the corresponding question for each SimpleQuestions dataset question. Τελικά δημιούργησα ορισμένες εκδόχες της μηχανής απάντησης ερωτήσεων χρησιμοποιώντας τη βιβλιοθήκη spacy, τη βιβλιοθήκη του Sentence Transformer συνδυαζόμενη με τη μετρική του cosine similarity και τη μετρική του jaccard similarity. Τέλος, τα καλύτερα σκόρ αξιολόγησης για το entity span επιτυγχάνονται για τη μεθοδολογία εύρεσης των δεικτών του entity span με τη δημιουργία λιστών που αποτελούνται από 0 και 1 μόνο για τις ερωτήσεις του συνόλου δεδομένων εκπαίδευσης και επικύρωσης SimpleQuestions, των οποίων το entity label αποτελεί μέρος της διατύπωσής τους. Επίσης, το μεγαλύτερο ποσοστό απόλυτα σωστών απαντημένων ερωτήσεων παρατηρείται

για τη μηχανή QA με τη βιβλιοθήκη spacy της python για τη μεθοδολογία εύρεσης δεικτών για το entity span(αρχή, τέλος) μέσω της μετρικής του cosine similarity και τη μεθοδολογία εύρεσης δεικτών για το entity span με τη δημιουργία λιστών που αποτελούνται από 0 και 1 στην περίπτωση εκπαίδευσης και επικύρωσης όλων των ερωτήσεων του συνόλου δεδομένων εκπαίδευσης και επικύρωσης SimpleQuestions αντίστοιχα, ανεξάρτητα από το αν το entity label τους αποτελεί μέρος της διατύπωσής τους.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Τεχνητή Νοημοσύνη-Επεξεργασία Φυσικής Γλώσσας

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Μηχανή απάντησης ερωτήσεων στη Wikidata, Πρόβλεψη δεικτών σχέσης (relation), αρχής και τέλους του entity label, μετρικές ομοιότητας κειμένου, Entity span ως ακολουθία από 0 και 1, Sentence Transformers, BERT

Αφιερώνω αυτήν την εργασία στους δύο αγαπημένους μου γονείς Θοδωρή και Εύη που συνέβαλαν τα μέγιστα στην ανατροφή μου ώστε ότιδήποτε και αν κάνω να έιναι το καλύτερο δυνατό. Την αφιερώνω επίσης στη γιαγιά μου τη Μαίρη που δεν είναι πια εδώ, όπως επίσης και στους στενούς μου φίλους που μου στάθηκαν σε δύσκολες στιγμές.

ACKNOWLEDGEMENTS

Για τη διεκπεραίωση της παρούσας Πτυχιακής Εργασίας, θα ήθελα να ευχαριστήσω τον επιβλέποντα, καθ .Μανώλη Κουμπαράκη, για τη συνεργασία, την ουσιαστική επίβλεψη και την πολύτιμη συμβολή του στην ολοκλήρωση της. Όπως επίσης θα ήθελα να ευχαριστήσω τον μεταπτυχιακό φοιτητή του Μεταπτυχιακού Προγράμματος του τμήματος Πληροφορικής και Τηλεπικοινωνιών "Πληροφορική" και συνεργάτη του επιβλέποντα, καθ .Μανώλη Κουμπαράκη, Κωνσταντίνο Πλά για τις καθοριστικές επεμβάσεις και ουσιαστικές συμβουλές για την διεκπεραίωση αυτής της διπλωματικής εργασίας.

CONTENTS

1. INTRODUCTION **16**

1.1 Problem Statement 16

1.2 Project Scope 17

1.3 Aim and Objectives 18

1.4 Methodology 19

1.5 Thesis Layout 20

2. PRELIMINARIES **21**

2.1 RDF data 21

2.2 Linked data 22

2.3 Semantic Parsing 23

2.4 Knowledge Graphs 24

2.5 SPARQL 24

2.6 Question Answering over Knowledge Graphs 26

2.7 Transformers and BERT 26

3. RELATED WORK **30**

3.1 Question Answering over Knowledge Graphs (KGQA) 30

4. THE MAIN WORK-RELATION AND ENTITY SPAN PREDICTION **33**

4.1 Relation Prediction 33

4.2 Entity Span Prediction 34

5. THE MAIN WORK-THE QUESTION ANSWERING ENGINE MODEL **40**

5.1 Question Answering engine creation 40

6. EVALUATION **44**

6.1 Evaluation experiments 44

6.2 Evaluation results 44

6.3 Question Answering results 54

7. CONCLUSIONS AND FUTURE WORK **65**

ABBREVIATIONS - ACRONYMS **67**

APPENDICES **67**

A. FIRST APPENDIX **68**

B. SECOND APPENDIX **69**

C. THIRD APPENDIX **70**

REFERENCES **71**

LIST OF FIGURES

2.1 An RDF statement consists of three components, referred to as a triple: subject, predicate, object 22

6.1 Learning curve between relation training and validation loss- methodology of finding entity span indexes (start and end) through entity span, which is a list exclusively consisting of 0 and 1 for the SimpleQuestions dataset that includes all the questions regardless of whether questions are answerable over wikidata or not, where their entity label is a part of their wording. . . . 46

6.2 Learning curve between entity span training and validation loss- methodology of finding entity span indexes (start and end) through entity span, which is a list exclusively consisting of 0 and 1, for the SimpleQuestions dataset that includes all the questions regardless of whether questions are answerable over wikidata or not. The entity label of these questions is a part of their wording. 47

6.3 Learning curve between relation training and validation loss-methodology of finding entity span indexes (start and end) through entity span , which is a list consisting of 0 and 1 or only of zeros, for the SimpleQuestions dataset that includes all the questions that answerable over wikidata. The entity label of these questions is a part of their wording. 48

6.4 Learning curve between entity span training and validation loss-methodology of finding entity span indexes (start and end) through entity span, which is a list consisting of 0 and 1 or only of zeros, for the SimpleQuestions dataset that includes all the questions that answerable over wikidata. The entity label of these questions is a part of their wording). 49

6.5 Learning curve between relation training and validation loss for the SimpleQuestions dataset which contains all questions (answerable and non-answerable over wikidata). This learning curve is the same for the methodology of creating entity spans as lists consisting of 0 and 1 or only of 0, for the methodology of finding entity span start and end indexes through either spacy library or Sentence Transformers library combined with cosine similarity metric and for the methodology of finding entity span start and end indexes through through either spacy library or by using jaccard similarity metric (Described in chapter 1.4). 51

6.6 Learning curve between entity span training and validation loss- methodology of finding entity span indexes (start and end) through entity span, which is a list consisting of 0 and 1 for the SimpleQuestions dataset which includes all questions (answerable and non-answerable over wikidata. . . . 52

6.7 Learning curve between relation training and validation loss for the SimpleQuestions dataset which includes only questions which are answerable over wikidata. This learning curve is the same for the methodology of creating entity spans as lists consisting of 0 and 1 or only of 0, for the methodology of finding entity span start and end indexes through either spacy library or Sentence Transformers library combined with cosine similarity metric and for the methodology of finding entity span start and end indexes through through either spacy library or by using jaccard similarity metric (Described in chapter 1.4). 53

6.8 Learning curve between entity span training and validation loss -methodology of finding entity span indexes (start and end) through entity span, which is a list consisting of 0 and 1 for the SimpleQuestions dataset which includes only questions which are answerable over wikidata. 54
6.9 Learning curve between entity span training and validation loss-methodology of finding entity span indexes either with spacy library or Sentence Transformers + cosine similarity metric for the SimpleQuestions dataset (training and validation dataset) that includes all the questions (answerable, non-answerable over wikidata). 55
6.10 Learning curve between entity span training and validation loss-methodology of finding entity span indexes either with spacy library or Sentence Transformers + cosine similarity metric for the SimpleQuestions dataset (training and validation dataset) that includes only questions that are answerable over wikidata. 55
6.11 Learning curve between entity span training and validation loss-methodology of finding entity span indexes either with spacy library or jaccard similarity metric for the SimpleQuestions dataset which contains all questions (answerable, non-answerable over wikidata). 56
6.12 Learning curve between entity span training and validation loss-methodology of finding entity span indexes either with spacy library or jaccard similarity metric for the SimpleQuestions dataset which contains questions that are answerable over wikidata. 56

LIST OF TABLES

6.1 Best evaluation results about relation index, entity span start and entity span end index for the methodology of creating a list consisting of 0 and 1, depending on if the question includes its entity label in its wording, then there are ones in the list, else only 0. For these evaluation results is used the SimpleQuestions dataset which includes all the training and validation questions either they are answerable over wikidata or not. For these questions their entity label must exists in their wording. 45
6.2 Best evaluation results about relation index, entity span start and entity span end prediction for the methodology of creating a list consisting of 0 and 1, depending on if the entity label exists in question wording, then there are ones in the list, else 0. For these evaluation results is used the SimpleQuestions subset which includes only training and validation dataset questions that are answerable over wikidata. For these questions their entity label must exists in their wording. 45
6.3 Best evaluation results about relation index, entity span start and entity span end index for the methodology of creating a list consisting of 0 and 1, depending on if the question has its entity label as a part of its wording, else a list full of zeros. For these evaluation results is used the SimpleQuestions dataset that includes all the questions regardless of whether questions are answerable over wikidata or not. 50
6.4 Best evaluation results about relation index, entity span start and entity span end prediction for the methodology of creating a list consisting of 0 and 1, depending on if the question has its entity label as a part of its wording, else a list full of zeros. For these evaluation results is used the SimpleQuestions dataset that includes questions that are answerable over wikidata. 50
6.5 Best evaluation results about relation index, entity span start and entity span end index for the methodology of finding entity span indexes (start and end), by using sentence transformers with cosine similarity metric, or else through difflib.get_close_matches() for the SimpleQuestions dataset which contains all (training, validation, test) the questions (answerable, non-answerable over wikidata. 57
6.6 Best evaluation results about relation index, entity span start and entity span end index for the methodology of finding entity span indexes (start and end), sentence transformers with cosine similarity metric, or else through difflib.get_close_matches() for the SimpleQuestions dataset (training and validation dataset) that includes only questions that are answerable over wikidata. 57
6.7 Best evaluation results about relation index, entity span start and entity span end index for the methodology of finding entity span indexes (start and end), by using jaccard similarity metric, or else through difflib.get_close_matches() for the SimpleQuestions dataset (training, validation, test) which comprises all questions (either answerable over wikidata or not). 58

6.8 Best evaluation results about relation index, entity span start and entity span end index for the methodology of finding entity span indexes (start and end), by using jaccard similarity metric, or else through difflib.get_close_matches() for the SimpleQuestions dataset (training, validation, test) which comprises questions that are answerable over wikidata. 58
6.9 Percentage (%) of correctly answered questions (Questions that each QA engine finds the answer from the ”Questions_testing_with_answers” .txt file and some quite similar answers) for each one of the Question Answering engines and methodlogies that are implemented for the SimpleQuestions dataset (training, validation, test) which comprises all questions (answerable, non answerable over wikidata). 58
6.10 Percentage (%) of answered questions (Questions that each QA engine finds answers that at least most of them have either (very) small or no similarity with the answer from the ”Questions_testing_with_answers” .txt file. One of these answers could maybe be the answer of the .txt file itself) for each one of the Question Answering engines and methodlogies that are implemented for the SimpleQuestions dataset (training, validation, test) which comprises all questions (answerable, non answerable over wikidata). . . . 59
6.11 Percentage (%) of correctly answered questions (Questions that each QA engine finds the answer from the ”Questions_testing_with_answers” .txt file and some quite similar answers) for each one of the Question Answering engines and methodlogies that are implemented for the SimpleQuestions datasets (training, validation, test) which contain only answerable over wikidata questions. 59
6.12 Percentage (%) of answered questions (Questions that each QA engine finds answers that at least most of them have either (very) small or no similarity with the answer from the ”Questions_testing_with_answers” .txt file. One of these answers could maybe be the answer of the .txt file itself) for each one of the Question Answering engines and methodlogies that are implemented for the SimpleQuestions datasets (training, validation, test) which contain only answerable over wikidata questions. 60

1. INTRODUCTION

1.1 Problem Statement

One of the most important desired aims of “the AI” would be the ability for it to communicate naturally with humans in a human language. As opposed to formal languages that are used to precisely instruct machines to perform certain actions, human language expressions can be difficult to truly understand without an huge body of knowledge about the physical world and human perception of it. While this knowledge about the world can help resolve some ambiguity, or at least make certain interpretations more probable, natural language remains ambiguous, occasionally fooling even humans [1].

Developing systems that hold a certain degree of semantic understanding of human language is one of the most important problems in natural language processing (NLP) as well as information retrieval (IR), but in this thesis it is given more focus on NLP. Understanding human languages is especially interesting for the purpose of question answering, and as a result it could be more easier to compute a direct answer to a question given by the user. Question answering enables users to go through a simple keyword-based search that returns a ranked list of relevant pages (e.g. Google Search) and transition to a more focused and effective search enables the system to reason over the available knowledge and provide a direct answer. In fact, Google Search already embeds question answering in its search results when the answer is clear [1].

In this thesis, we focus on deep learning for semantic parsing and question answering over knowledge graphs (KGQA), which is a extensive research field. More specifically question answering can be performed using different sources of knowledge: relational databases, graph databases and knowledge graphs, text, images, other multimedia, and any combination of these. These different sources of knowledge can be categorized as structured and unstructured[1].

With structured sources, for example knowledge graphs and relational databases, we refer to those sources where the meaning of the stored information is unambiguously defined in a formal logical system. With unstructured sources, such as text and images, we refer to sources where the meaning of the stored information is not clearly indicated and where the data must be interpreted in their raw form in order to compute an answer[1].

Structured sources of knowledge usually provide some automatic way of querying the contained information, for example, relational databases typically support retrieval using SQL queries and RDF (Resource Description Framework) knowledge graphs can be queried using SPARQL [2,3,4,5]. The task of building a query given a question is a form of semantic parsing, for example text-to-SPARQL for question answering over knowledge graphs (KGQA) [1].

Generally the knowledge can be provided in several different ways, such as text, databases, knowledge graphs and even images. When structured data sources are used, such as relational databases or knowledge graphs, question answering is usually performed by relying on a semantic parser that translates the natural language question into a formal query in a compatible query language. This query can then be executed by a database to retrieve the answer to the question in the form of a narrow subset of its data (e.g. a set of entities) [1].

So semantic parsing is the task of translating a natural language expression (e.g question)

into a machine-understandable form that maintains the meaning of the original expression [1].

Also, it is worth mentioning that there has been an increased interest in neural network based methods from both academic and industrial communities. As with other application areas in NLP and beyond, deep learning automatically learns to extract relevant features, which are used by machine learning methods to describe examples numerically in order to be able to distinguish between them and make decisions. Thus the users could be able to avoid manual feature engineering, which was characteristic for earlier machine learning methods [1].

In general the larger size a dataset has the probabilities a model a neural network to be trained successfully are higher. However the difficulty and the cost of constructing datasets with large quantities of high-quality data is significant. This has as a consequence a lack of data, which can be moderated by relying on transfer learning from other tasks with more data. In transfer learning, (neural network) models are first trained on a different but related task, with the goal of capturing relevant knowledge in the pretrained model. Then, the pretrained model is finetuned on the target task, with the goal of reusing the knowledge captured in the pretraining phase to improve performance on the target task [6].

Recently proposed transfer learning methods [7,8,9,10,11,12] show that significant improvement on downstream natural language processing (NLP) tasks can be obtained by finetuning a neural network that has been trained for language modeling (LM) over a large corpus of text data without task-specific annotations. The use of this family of techniques is an emerging research topic in the NLP community, because these techniques are proved to be very useful [9].

SimpleQuestions dataset [13-18] is a very well-studied dataset that characterizes core challenges of KGQA. The large size of this dataset is particularly appealing for my thesis, because it enables me to investigate performance for a wider range of sizes of the data used for training [1,6]. These are the main reasons that i use this dataset for my thesis.

Based on all the aforementioned information, in this thesis the main problem that is stated is the creation of the best question answering (QA) engine over the knowledge graph of Wikidata [19] by training and validating SimpleQuestions dataset [13-18] and by using BERT [1,6,7,20] as the pretrained model for finetuning [21-26], in order to provide users directly answers to their natural language questions, by translating natural language (NL) inputs (e.g questions) to their logical forms (queries) through RDF query language SPARQL [2,3,4,5] (semantic parsing). From this problem occurs that knowledge graph question answering (KGQA) system has to understand the intent of the given question, formulate a query, and retrieve the answer by querying the underlying knowledge base [1,6].

1.2 Project Scope

The subject that this thesis is directly related is deep learning [1,6]. In the recent decades, and most notably in the recent years, incredible progress has been made in the field of deep learning. Compared to traditional machine learning methods (e.g. Support Vector Machines[27]), where features must be manually predefined and are then used in a classifier or regression model, deep learning methods aim to automatically learn to extract useful features. All the above combined with the rapid increase in computing power and

availability of data, resulted in the development of novel learning techniques for various applications [1].

One example of these techniques was the development of BERT [1,6,7,20-26] and other pre-trained language models in the field of NLP, which enabled transfer learning from more powerful pre-trained models. One of these pre-trained models is the recently released OpenAI GPT3 [28], which was impressive in its ability to generate coherent looking and eloquently written text and even other things (e.g. HTML code).

Semantic parsing and KGQA can similarly benefit from deep learning and it is important to investigate related questions. The ability to learn and pre-train representations rather than engineer features can make it easier to train and adapt semantic parsers for new domains. Also semantic parsing and KGQA have certain characteristics that make them a unique structured prediction problem, compared to the well-studied sequence generation methods used for machine translation. In particular, semantic parsing can often be formulated as a sequence-to-tree (or more generally sequence-to-graph) task, where the tree-structured output can inspire certain modeling choices[1].

1.3 Aim and Objectives

About this thesis the aim is to investigate how deep learning and semantic parsing could be used in order to create a question answering engine. More specifically the aim is to create a QA engine over the knowledge graph Wikidata [19] by using BERT [1,6,7,20] and also the RDF query language SPARQL [2,3,4,5].

In order to achieve this aim from this thesis the following list of objectives must be achieved:

1. The first objective is the creation of a simple SPARQL query [2-5,29-33] that uses the entity id of each one of the questions of the training and the validation dataset of the SimpleQuestions dataset [13-18] in order to find its entity label [34-39]. This simple query is the only involvement of semantic parsing with this thesis.

2. An important objective is to investigate how text similarity models can effect the performance of the neural networks that predict the relation id and the entity label start and end indexes [40-54].

   In fact we want to achieve the maximum score for all the evaluation metrics(F1, recall, precision and accuracy [55]) for the relation and entity span neural networks, by finding the entity label start and entity label end index using text similarity model of Sentence Transformers [56,57] combined either with cosine similarity metric [42,43, 58,59] or the jaccard similarity metric [44,45,58,59] or the spacy library [41-49,60].

   Also the achievement of the highest score for all the evaluation metrics [55] is an objective for the methodology of entity span to be a sequence of 0 and 1 (entity span encoded), where 1 if the word that is included in the entity label, is also included in a question part equal with the entity label, if it exists, and 0 if not [1,6,46,47,50,51] for each of the SimpleQuestions dataset questions, which now contains the entity labels [13-18,33-39].

3. Also we want to check for the methodology of entity span encoded with 0 and 1 [1,6,46,47,50, 51] through evaluation metrics if the performance of the neural networks for relation indexes and entity label start and end indexes [40-54], increases or decreases if we ignore questions that their entity label is not part of them or not.

This is an important objective for both the full SimpleQuestions dataset [13-18,34-39] and the corresponding SimpleQuestions subsets [14,16,34-37] that contain only questions that are answerable over wikidata [19].

4. Another objective is to check if the neural networks about relation, entity label start and entity label end index [40-54] in the corresponding questions of these indexes (mentioned in the objective 2),by using BERT transformer model [1,6,7,20-26] in order to create them [40-54] to predict the right relation id and the entity label start and entity label end index in the question correspondingly, could give good predictions which could lead to reliable answers both to other questions that are either different from the questions of the test subset of the SimpleQuestions dataset [17,18,38,39] or not.

5. Additionally we want to investigate if and how text similarity model of Sentence Transformers and similarity metrics such as cosine and jaccard [42-45,58,59] effect the performance of a question answering engine model [48,49,52,53]. More specifically:

    - I want to create the three QA engine models below: the QA engine model consisting of python library spacy [41-49,60], the corresponding model that include the Sentence Transformer similarity model combined with cosine similarity metric [52,56,57,58,59] and the QA engine model consisting jaccard similarity metric [53,58,59].
    - Also another objective is to check which of the above QA engine models are preferable in order to create a question answering engine that is able to give reliable answers to each question given by the user.

6. Also another objective is to investigate based on the correct predictions for entity id and relation id, when the answers given from QA engines [49,52,53] in a question are totally correct, at some point correct or totally incorrect.

1.4 Methodology

The main methodology that i applied in order to achieve the aim of my thesis is consisting of firstly finding the entity labels through SPARQL queries for each question of the SimpleQuestions dataset in order to include them in the initial dataset. After that i found the relation index for the relation id of the corresponding SimpleQuestions dataset, that includes the entity labels question through a list with the unique relation ids that exist in the SimpleQuestions dataset, i.e the relation vocabulary and also i found the entity span indexes (start, end) either through sequences of 0 and 1 or spacy library or jaccard similarity metric or Sentence Transformers library combined with cosine similarity metric. The next step is to predict with the greatest possible evaluation scores(F1, recall, precision, accuracy), the entity span indexes and the relation indexes.

So after all these steps, i create the QA engine models (QA engine including spacy library, QA engine including cosine similarity metric combined with Sentence Transformers similarity model, QA engine including jaccard similarity metric), where for each question given by a user the relation and the entity span (start and end) indexes are predicted through the aforementioned neural networks, that are also created in each QA engine, and through the suitable SPARQL queries the answer of this question, if this exists, is found.

1.5 Thesis Layout

My thesis consists of the following layers:

1. **Chapter 1: Introduction**

   This chapter states the basic problem of this thesis, explains the general scope of the thesis, presents the aim and the objectives of this thesis and describes in detail the main methodology that is followed in the thesis.

2. **Chapter 2: Preliminaries**

   This chapter provides the background knowledge necessary to understand the rest of the thesis. It refers mainly on question answering over knowledge graphs and deep learning and more briefly on semantic parsing (especially through SPARQL).

3. **Chapter 3: Related Work**

   In this chapter recent work that is related with the question answering over knowledge graphs and semantic parsing are presented and discussed briefly.

4. **Chapter 4: Methodology Explained** This chapter explains analytically the methodologies that are applied, i.e entity spans as sequence of 0 and 1, Spacy library or Sentence Transformers library combined with cosine similarity and Spacy library or Sentence Transformers library combined with jaccard similarity metric to have the best possible relation and entity span neural network parameters.

5. **Chapter 5: The Question Answering Engine Model**

   This chapter explains how the QA engines for this thesis are created. More specifically three types of QA engines are created: The first one including only spacy library, the second one including the Sentence Transformers similarity model combined with cosine similarity metric and the third including only jaccard similarity metric.

6. **Chapter 6: Evaluation** This chapter displays the experiments with the best evaluation results, i.e loss learning curve, training and validation losses values, as well as precision, accuracy, f1 and recall scores. Additionally for each one of these experiments, their results are interpreted and thus some conclusions are drawn.

7. **Chapter 7: Conclusions and future Work** Finally, we summarize our contributions and findings. Also based on our findings, the creation of a question answering engine for datasets with more complex questions(e.g ComplexQuestions) for is discussed as a direction for future research.

2. PRELIMINARIES

This thesis is referring to the following concepts: RDF data, linked data, Knowledge Graphs, RDF query language SPARQL [2,3,4,5], semantic parsing, Question Answering over Knowledge Graphs, Transformers and BERT.

Before i explain the above concepts, it is important to explain concisely some terminologies. The natural language question (NLQ), is also referred to as question or utterance. The meaning representation is also referred to as logical forms or formal queries.The execution results or answers for a formal query are also referred to as denotations.

2.1 RDF data

RDF stands for Resource Description Framework and is a standard for describing web resources and data interchange, developed and standardized with the World Wide Web Consortium (W3C). W3C includes the foundational concepts, semantics and specifications for different formats. Essentially RDF is a general method of describing and representing data by defining relationships between resources such as documents, physical objects, people, abstract concepts and data objects in a graph format [61,62,63].

The first syntax defined for RDF was based on the Extensible Markup Language (XML). Although the XML syntax was the only syntax option specified at first, standards for encoding RDF statements now include syntaxes such as Terse RDF Triple Language, JavaScript Object Notation for Linked Data (JSON-LD) and N-Triples (Turtle), a subset of the Turtle syntax [61,62,63].

RDF allows describing anything: persons, animals, objects, and concepts of any kind. They are considered resources. So RDF is a standard way to make statements about resources. Every resource is identified by a URI or IRI. Uniform Resource Identifier (URI) is a standardized format for identifying a resource, whether abstract or physical. IRI (Internationalized Resource Identifier) is a protocol standard that expands the Uniform Resource Identifier (URI) one. IRIs are similar and complementary to URIs,enabling the use of international character sets. URI standard only uses the US-ASCII character set. IRI allows containing characters from Unicode character set. IRI allows using Chinese, Japanese, Korean, and Cyrillic characters. IRIs could appear in all the positions of an RDF [61].

For example, the IRI for the Rosetta Stone is:

https://dbpedia.org/describe/url=http %3A%2F%2Fdbpedia.org%2Fresource%2FRosetta_stone&sid=4560

and the IRI for the British Museum is: https://dbpedia.org/describe/?url=http%3A%2F %2Fdbpedia.org %2Fresource%2FCategory%3ABritish_Museum&sid=4560.

RDF allows combining information from different datasets like Wikidata [19], DBpedia [64,65] etc.

RDF statements are also known as triples and consist of three components [61,62,63]:

1. Subject is a resource being described by the triple.
2. Predicate describes the relationship between the subject and the object.
3. Object is a resource that is related to the subject.

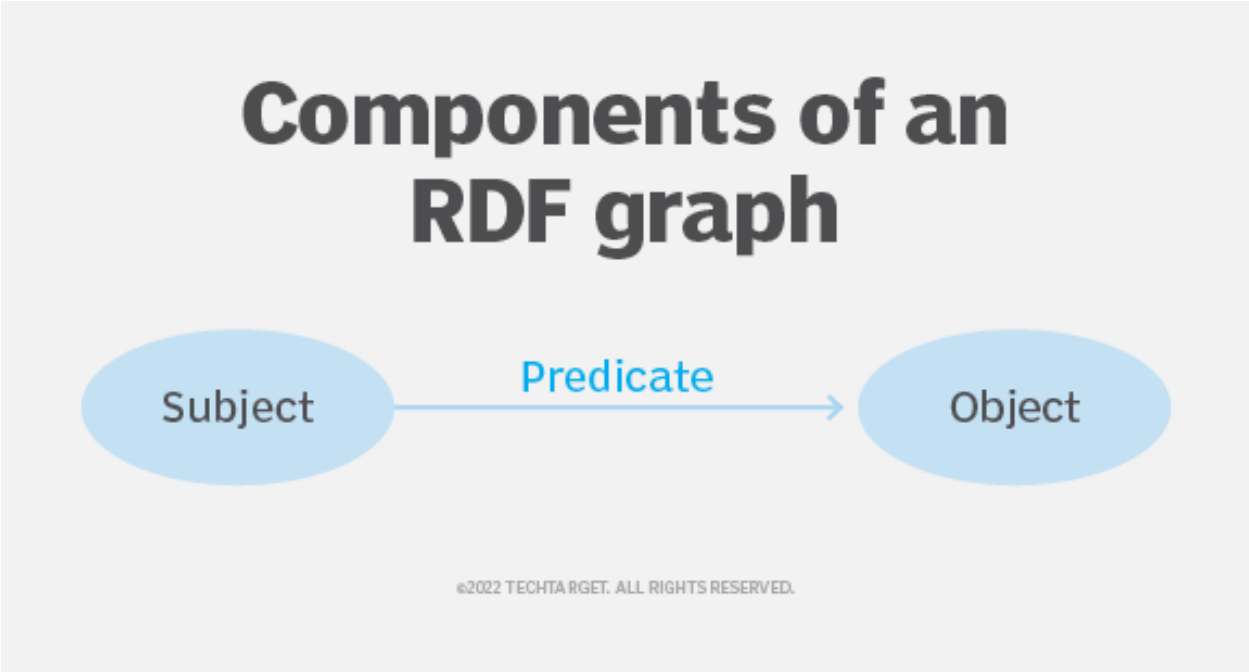

Figure 2.1: An RDF statement consists of three components, referred to as a triple: subject, predicate, object

The RDF standard provides for three different types of nodes [61,62]:

1. Uniform Resource Identifier (URI) (mentioned earlier): Uniform Resource Locator (URL) is a type of URI that is commonly used in RDF statements. When W3C updated the RDF specification to version 1.1 in 2014, it added Internationalized Resource Identifier (IRI) as a node type.
2. Literal is a specific data value and can be a string, a date or a numerical value. They could not appear in the subject or predicate positions, just in the object position. Literal values are expressed using the URI or IRI format.
3. Blank node identifier is also known as an anonymous resource or a *bnode*. It represents a subject about which nothing is known other than the relationship. Blank node identifiers use special syntax to identify them.

2.2 Linked data

The Linked data has the goal to publish structured data in such a way that it can be easily consumed and combined with other Linked Data. About Linked data in order to achieve their goal, these four basic principles must be respected [66,67] which are explained below:

1. **Use URIs as names for things**: Instead of using application-specific identifiers—database keys, UUIDs, incremental numbers, etc.—mapping names to a set of URIs [61]. Each identifier must map to one single URI [61]. Essentially With the help of URIs [61], it is more easier to distinguish between different things or to know that one thing from one dataset is the same as another in a different dataset.

2. **Use HTTP URIs so that people can look up those names**: Make URIs [61] roughly, accessible via HTTP protocol which provides a simple mechanism for retrieving resources. as we do for every human-readable Web page, i.e dereferenceable.

3. **When someone looks up a URI, provide useful information, using the standards such as RDF and SPARQL**: Have the web server reply with some structured data when invoked and model the data with RDF [61,62,63]. Here there is the need to perform a paradigm shift from a relational data model to a graph one. Essentially the use of RDF [61,62,63] and SPARQL [2,3,4,5] gives the ability for efficient use of URIs [61].

4. **Include links to other URIs so that they can discover more things**:Similarly to the hypertext web, links to other URIs [61] makes data interconnected and enables us to find different things. The main goal of this principle is to make explicit those links that were implicit before shifting to the Linked data approach.

Once these principles are respected, the Linked Data rules achieve their goal because people across the Web can start referencing and consuming the data in our rows easily [66]. It is easily understood that Linked Data is one of the core pillars of the Semantic Web. The Semantic Web is about making links between datasets that are understandable not only to humans, but also to machines, and Linked Data provides the best practices for making these links possible [67]. Essentially Linked Data is a set of design principles for sharing machine-readable interlinked data on the Web [63].

Linked data is not only about exposing data using Web technologies, but a Web of Data could be built through Linked data [67]. Imagine that there are hundreds of different datasets published on the Web according to the Linked Data principles: thousands different identifiers that people can rely to grab data about books, movies, actors, cities, or anything else. Consequently all these datasets form a giant Web-scale database which could be potentially embodied in many applications and the reference in this database whenever there is a need becomes more easier [67].

In other words instead of having links citations simply between books or Web pages, this allows links between anything to be followed for more information. For example if a single author has published in two different journals and both journals expose their catalogs as Linked Data, and the author's bio is on DBpedia [64,65], then the corresponding application can easily mash it all together with a simple query, automatically. As if all the data was in one database [67].

A benefit of Linked Data is that breaks down the information that exist between various formats and brings down the fences between various sources. Also it facilitates the extension of the data models and allows easy updates. So, data integration and browsing through complex data become easier and much more efficient [67].

2.3 Semantic Parsing

Semantic parsing is the task of translating a natural language (NL) question $x \in X$, $X \subset N$ into a machine-interpretable representation $q \in Q$ of its meaning in a given formal language (FL), such as SPARQL [2,3,4,5]. Here, X is a subset of the natural language N, which specifies the scope (domain) of some semantic parser, which is a concept that refers to the knowledge or competence area in which a particular semantic parser might

be specialized. N contains all possible questions that are considered to be part of a natural language (for example English) and the X, which is subset of it, may be restricted to those questions that are in the domain/scope of the semantic parser [1].

Thus the set X is the set of questions for which the system using the semantic parser should provide a response that is not an out-of-domain response. For example, if one can make a semantic parser to interpret commands for a basic autonomous vacuum cleaner, then only a small fraction of possible NL questions is meaningful, that is, should result in a change of movement or air flow. Other questions, for example asking about the weather, should be considered out-of-domain. [1].

The produced logical form *q* could be assumed as correct if it most accurately reflects the meaning of the NL question *x* under the restrictions of the scope of the parser and the used formal language. However, it is difficult to clearly express in a formal way the concept of meaning. One property of the correct logical form is that its execution in the environment *E* (which is denoted by *q(E)*) leads to the desired change of the environment [1].

For example, executing a query in a computer system to retrieve the results from a database yields the set of results. However, the correctness of execution results is not sufficient because multiple logical forms could give the expected results but not all of them correctly capture the meaning of the NL question. For example, translating “What is five plus 4” to $3*3$ yields the same result but is not the correct query [1].

2.4 Knowledge Graphs

A knowledge graph (KG) is a formal representation of facts pertaining to a particular domain (including the general domain). It consists of entities denoting the subjects of interest in the domain, and relations (The words predicate or property are often used interchangeably with relations) denoting the interactions between these entities. A relation can link two entities and also an entity to an data value of the entity (such as a date, a number, a string etc.) which is referred to as a literal. Several large-scale knowledge graphs have been developed, which contain general factual knowledge about the world. Some of the most prominently known examples are WikiData[19], Freebase [68] and DBpedia [64,65].

Formally, let E = $\{e_1, \ldots, e_{n_e}\}$ be the set of entities, L be the set of all literal values, and P = $\{p_1, \ldots, p_{n_e}\}$ be the set of relations connecting two entities, or an entity with a literal. A triple $t \in E \times L \times (E \cup L)$ is defined as a statement comprising of a subject entity, a relation, and an object entity or literal, describing a fact. Then, a KG K is a subset of $E \times L \times (E \cup L)$ of all possible triples representing facts that are assumed to hold [1].

2.5 SPARQL

SPARQL, a recursive acronym for “SPARQL Protocol and RDF Query Language”, is one of the most commonly used query languages for KGs and is supported by many publicly available KGs like Wikidata[19], Freebase [68] and DBpedia [64,65]. The central part of a SPARQL query [2,3,4,5] is a graph pattern, composed of resources from the KG (i.e. entities and relations) and variables to which multiple KG resources can be mapped. This graph patterns specifies conditions on variables, whereas the SELECT clause determines which variables are returned as the answer [2,3,4,5].

Essentially SPARQL is a query language that requests and retrieves data that use the RDF format (RDF data section). Thus, the database is a set of items with the format subject-predicate-object previously explained. Also SPARQL allows using query operations such as JOIN, SORT, AGGREGATE, along with others [58].

The following example permits obtaining the names and emails of all the persons in the FOAF (Friend Of A Friend) dataset [62]:

```
PREFIX foaf:
SELECT ?name
       ?email
WHERE
  {
    ?person  a          foaf:Person .
    ?person  foaf:name  ?name .
    ?person  foaf:mbox  ?email .
  }
```

Listing 2.1: Example in RDF language SPARQL which permits obtaining the names and emails of all the persons in the FOAF (Friend Of A Friend)vocabulary is found.

The PREFIX clause declares the label foaf representing the URI indicated between angle brackets. The SELECT clause joins together all the RDFs where the predicate a corresponds to a person according to the foaf dataset, and the person's name and mailbox [62]. The join result consists of a set of rows with the name and email of each person in the dataset. As a person may have multiple names and mailboxes, then the returned set of results could contain multiple rows for the same person [62].

It is worth mentioning one type of data that SPARQL used is the geospatial data. For example work [69] presents the dataset GEOQUESTIONS1089 for benchmarking geospatial question answering engines. GEOQUESTIONS1089 is the largest such dataset available presently and it contains 1089 questions, their corresponding GeoSPARQL or SPARQL queries and their answers over the geospatial knowledge graph YAGO2geo. The dataset GEOQUESTIONS1089 is used to evaluate the effectiveness and efficiency of geospatial question answering engines GeoQA2 (an extension of GeoQA developed by the AI team of the University of Athens (UoA)) and the system of Hamzei et al. (2021).

Also about question answering engine GeoQA2 is presented more analytically in [70]. Essentially GeoQA2 consist the most recent version of the engine GeoQA, originally proposed by Punjani et al and has been designed to work over the geospatial knowledge graph YAGO2geo. GeoQA2 improves GeoQA by being able to answer more complex geospatial questions (e.g., questions involving aggregates and arithmetic comparisons), having more effective individual components (e.g., for named entity recognition and disambiguation), supporting more templates for generating GeoSPARQL queries, and executing these queries more efficiently by using a technique based on materializing topological relations.

Large amounts of geospatial data [69,70,71] have been made available recently on the linked open data cloud and on the portals of many national cartographic agencies (e.g., OpenStreetMap data, administrative geographies of various countries, or land cover/land use data sets). These datasets use various geospatial vocabularies and can be queried using SPARQL or its OGC-standardized extension GeoSPARQL. [70] as a paper their authors go beyond these approaches to offer a question answering service on top of linked geospatial data sources. For the evaluation of the algorithms for the architecture that this paper follows is used a set of 201 natural language questions.

2.6 Question Answering over Knowledge Graphs

KGQA can be defined as the task of retrieving answers to NL questions or commands from a KG. While the expected answers could take on complex forms, such as ordered lists (e.g. for the command “Give me Roger Deakins movies ranked by date.”), or even a table (e.g. for the question “When was which Nolan film released?”), the vast majority of works on KGQA assume the answer to be of a simpler form: a set of entities and/or literals or booleans [1].

A more formal explanation it can be defined as [1]: Let G be a KG and let q be a natural language question. Then the set of all possible answers is specified A as the union of the power set $P(E \cup L)$ of entities E and literals L in G, the set of the numerical results of all possible aggregation functions $f : P(E \cup L) \mapsto \mathbb{R}$ (such as SUM or COUNT), and the set True,False of possible boolean results, which is needed for yes/no questions. The task of KGQA is to return the correct answer $a \in A$, for a given NL question q. The most common way in which KGQA approaches accomplish this is by creating a formal query representing the semantic structure of the question q, which when executed over G, returns A. Thus, KGQA can be assumed as an application area of semantic parsing.

2.7 Transformers and BERT

Transformer [1,6,7,15,16,17,72] networks have been recently proposed for NLP tasks and are fundamentally different from the common RNN and CNN architec tures. Especially compared to RNNs, which maintain a recurrent state, transformers use multi head self-attention to introduce conditioning on other timesteps. This allow for more parallel training, and offer better interpretability, unlike RNNs, which process the input sequence one time step at a time [1,6].

Essentially, transformers consists of several layers of multi-head self-attention with feed-forward layers and skip connections. Multi-head self-attention is an extension of the standard attention mechanism [73], with two major differences [6]:

1. Attention is applied only within the input sequence
2. Multiple attention heads enable one layer to attend to different places in the input sequence

In order to understand how transformers work, let the transformer consist of L layers, each ($l \in \{1, \ldots, L\}$) producing vectors $h_1^{l+1}, ..., h_N^{l+1}$, which are also used as inputs of the l+1-th transformer layer. The inputs $h_1^1, ..., h_N^1$ to the first transformer layer are the embeddings of the input tokens $h_1, ..., h_T$. Based on the above hypothesis we have about the transformer encoder layer and transformer decoder layer:

Transformer Encoder Layer: A single transformer layer for an encoder consists of two parts [1]:

1. a self-attention layer
2. a two-layer MLP.

The self-attention layer collects information from neighbouring tokens using the multi-head attention mechanism that attends from every position to every position. With $h_t^{(l-1)}$ being

the outputs of the previous transformer layer for position *t*, as referred before, the multi-head self-attention mechanism computes *M* different attention distributions (one for every head) over the *t* positions. For every head ($m \in \{1, \ldots, L\}$), self-attention score in each layer *l* for position t is computed through the following equations:

$$\begin{aligned} \boldsymbol{q}_t^{l,m} &= \boldsymbol{W}_Q^{l,m} \boldsymbol{h}_t^l \\ \boldsymbol{k}_t^{l,m} &= \boldsymbol{W}_K^{l,m} \boldsymbol{h}_t^l \\ \boldsymbol{v}_t^{l,m} &= \boldsymbol{W}_V^{l,m} \boldsymbol{h}_t^l \end{aligned} \tag{2.1}$$

$$\begin{aligned} \boldsymbol{a}_{t,j,l,m} &= \frac{(\boldsymbol{q}_t^{l,m})^T \boldsymbol{k}_j^{l,m}}{\sqrt{\boldsymbol{d}_k}} \\ \boldsymbol{a}_{t,j,l,m} &= \frac{e^{\boldsymbol{a}_{t,j,l,m}}}{\sum_{i=0}^{T} e^{\boldsymbol{a}_{t,i,l,m}}} \\ \boldsymbol{s}_t^l &= \sum_{i=0}^{T} \boldsymbol{a}_{t,j,l,m} \boldsymbol{v}_j \end{aligned} \tag{2.2}$$

Essentially the attention score is calculated through a softmax function [74-77] of dot products between the input vectors h_t of position t and h_j of position j in the layer l after multiplication with the input vectors h_t, h_j and W_Q, and W_K , which are trainable matrices for head *m* of layer *l*. Self-attention score is normalized by the square root of the dimension d_k of the $k_t^{l,m}$ vectors.

From the equation (2.2) $\boldsymbol{s}_t^l$ are defined as the intermediate representation vectors for each input position, which are computed as the concatenation of the *M* heads' summary vectors, each computed as a $a_{t,j}$ -weighted sum of input vectors $h_1^1, ..., h_N^1$, which are first projected using the matrix $W_V^{l,m}$

The summaries $\boldsymbol{s}_t^{(m)}$ for head *m*, as computed above, are concatenated and passed through a linear transformation to get the final self-attention output vector $\boldsymbol{u}_j$:

$\boldsymbol{u}_j = \boldsymbol{W}_O \oplus_{m=0}^{M} (\boldsymbol{s}_t^l)^{(k)}$

where $\boldsymbol{W}_O$ is a trainable matrix and $\oplus$ denotes concatenation.

The output of the *l*-th transformer layer (which is also the input to the *l*+1-th layer as referred before) is then given by applying a two-layer feed-forward network [78] with a ReLU activation function [79] on $\boldsymbol{h}_t^l$. Finally the whole transformer layer is shown through the following equations [1,6]:

$$\begin{aligned} \boldsymbol{q}_t^{l,m} &= \boldsymbol{W}_Q^{l,m} \boldsymbol{h}_t^l \\ \boldsymbol{k}_t^{l,m} &= \boldsymbol{W}_K^{l,m} \boldsymbol{h}_t^l \\ \boldsymbol{v}_t^{l,m} &= \boldsymbol{W}_V^{l,m} \boldsymbol{h}_t^l \end{aligned} \tag{2.3}$$

where W_A W_B are also trainable matrices [1,6].

Transformer Decoder Layer:

The decoder should take into account the previously decoded tokens, as well as the encoded input. Because of this the following two changes are made in the decoder, compared to the presented transformer encoder layer [1]:

1. The usage of a causal self-attention mask in self-attention

2. The addition of a cross-attention layer.

The causal self-attention mask is necessary to prevent the decoder from monitoring to future tokens, which are available during training but not during test. As such, this is more of a practical change. The cross-attention layer enables the decoder to monitor to the encoded input. Denoting the encoded input sequence vectors as $\boldsymbol{x}_i$, cross-attention is implemented in the same way as self-attention, except the vectors used for computing the key and value vectors are the $\boldsymbol{x}_i$ vectors. The decoder layer then becomes:

A more formal explanation it can be defined as: Let G be a KG and let q be a natural language question. Then the set of all possible answers is specified A as the union of the power set $P(E \cup L)$ of entities E and literals L in G, the set of the numerical results of all possible aggregation functions $f : P(E \cup L) \mapsto \mathbb{R}$ (such as SUM or COUNT), and the set True,False of possible boolean results, which is needed for yes/no questions. The task of KGQA is to return the correct answer $a \in A$, for a given NL question q. The most common way in which KGQA approaches accomplish this is by creating a formal query representing the semantic structure of the question q, which when executed over G, returns A. Thus, KGQA can be assumed as an application area of semantic parsing [1].

BERT consists a model that is created from Devlin et al. [7], who pretrained transformers on a large collection of unsupervised language data, following previous works on transfer learning from pretrained transformer based language models [11]. In particular BERT [1,6,7,15-17,20-26,72] is a transformer-based bidirectional model which builds on pretraining a masked language model (MLM).

The fact that BERT [1,6,7,15-17,20-26,72] has been pre-trained as a masked language model allows this model to use tokens both from the preceding (left) tokens as well as following (right) tokens to build representations of tokens. In contrast, a left-to-right language model of OpenAI-GPT [11] would only be allowed to use tokens to the left of the current position. In other words the left-to-right LM pretraining of OpenAI-GPT constrained the model to look only at the past [1,6].

More specifically BERT is pre-trained on a large corpus of text (BookCorpus (800M words) and English Wikipedia (2500M words)) using an objective that consists of two tasks:

1. **Masked LM task**

   The MLM task trains the model to predict words that have been omitted from the input. In order to do this, 15% of all tokens in a sentence are replaced either with a special [MASK] token in 80% of the cases, feeding this resulting partially masked sequence into the model and training the model to predict the words that have been masked out, given the other words or a random token (in 10% of the cases). In the remaining 10% of the cases, the selected token is not replaced. This is done to reduce the inconsistency between training and test, because during testing, no special mask tokens are used. The main objective of the masked LM task is to predict the (20%) masked words in the sentence [1,6].

2. **Next Sentence Prediction task**

   This task consists of predicting whether two sentences follow each other in the corpus (or generally in a text) or not. To train for this task, in 50% of the cases, two adjacent sentences are taken from the corpus, and in the other 50%, two random sentences are taken. The model is trained to distinguish between the two cases. The Next Sentence Prediction (NSP) task is useful for downstream tasks such as

single sentence classification and also entailment, which is formulated as classification of sentence pairs, but also for single sentence classification [1,6].

The fact that BERT is pre-trained as a masked language model enables BERT's feature vectors to include information both from the previous tokens as well as the following tokens [1,6]. Additionally BERT is pre-trained on a sentence pair classification task. Specifically, it is trained to predict whether one sentence follows another in a text [6].

The model used in BERT is simply a transformer encoder with learned positional embeddings rather than the original fixed sinusoid ones. Before the input (for example a sentence such as "What songs have Nobuo Uematsu produced") is fed to the model, it is first tokenized to (sub)word level using a WordPiece [89,90] vocabulary of 30000 tokens (["What", "songs", "have", "no", "#buo", "u", "#ema", "#tsu", "pro duced"], for the previous sentence). More common words are taken as words("What","songs","have"),while uncommon words are split into subword units ("nobuo" ["no", "#buo"]) [1,6].

This method significantly reduces vocabulary size and the amount of rare words without dramatically increasing sequence length. The input sequence is also padded with a special [CLS] token at the beginning and a special [SEP] token at the end[6]. Also if the input is a sentence pair, such as "The dog ate." and "Man bites dog", they are separated using another [SEP] token and they would be fed to the model as, "[CLS] The dog ate. [SEP] Man bites dog [SEP]" [1]. Also in addition to positional embeddings, a special segment embedding is added to the WordPiece embeddings. This segment embedding simply specifies whether the token is part of the first sentence or the second [1].

Finally the WordPiece [80,81] token sequence is then embedded into a sequence of vectors. The resulting embedding vectors are fed through the transformer, which uses several lay ers of multi-head self-attention [1,6] and feedfoward layers [6,78]. The output vectors for each token can be used for sequence tagging tasks, while the vector associated with the [CLS] token at the beginning of the sequence is assumed to capture relevant information about the input sequence as a whole, since BERT has been pre-trained for sentence pair classification [1].

3. RELATED WORK

Firstly i refer analytically in works that are related with the Question Answering over Knowledge Graphs (KGQA). All the related works that i refer about this thesis are related with the Question Answering over Knowledge Graphs (KGQA) and also with the SimpleQuestions dataset [13-18,30-35], because these are the main points the main points around which my thesis has been created.

3.1 Question Answering over Knowledge Graphs (KGQA)

Especially in the case of this thesis, where in the SimpleQuestions dataset [13-18,34-39] exists single fact based questions and consequently a single subject entity and a relation need to be predicted, the corresponding queries could be created by using SPARQL [2,3,4,5]. For questions such as, ”Where was Frank Herbert born?” and other examples of this type of questions, queries can be created following this logic: given a knowledge graph *G* that contains triples of the form (*s*, *p*, *o*), where *s* is a subject entity, *p* a predicate (also denoted as relation), and *o* an object entity. Therefore given a natural language question represented as a sequence of words $q = w_1,.....,w_T$, then through simple QA must be found the the set of triples $(\hat{s}, \hat{p}, \hat{o})$, for which right subject $\hat{s}$ and predicate $\hat{p}$ that question *q* refers to and which characterize the set of triples in G that contain the answer to *q*.

For fixed-structure prediction tasks like these described above, simple text classification and ranking methods could be used to predict the different parts of the target formal query given the natural language question as input [1]. Also for single-fact based prediction tasks, systems relying on standard classification models, can achieve state-of-the-art performance [82,83].

But for such approaches the target formal queries consist only of one subject entity and one relation and this means that they could be predicted using two separate classifiers, receiving the NLQ as input and producing an output distribution over all entities and all relations in the KG, respectively [1].

So, while the KG relation mentioned or implied in the question, could be predicted successfully with this approach, the entities could not be predicted with equal success, because large KGs like Freebase [68] contain a huge amount of entities and training datasets can only cover a small fraction of these and therefore many classes remain unseen after training. This makes it difficult to apply the approach described above for relation classification to entity prediction. Consequently this approach applied with success to the relation classification, but not the same is the case with entity classification [1].

Therefore, several works on SimpleQuestions [13-18,34-39] initially only perform relation classification and rely on a two-step approach for entity linking instead. In such an two-step approach, firstly an entity span detector, which does not need to learn representations for entities (like encoder networks used in classifiers), avoiding the need for data covering all entities, is applied to identify the entity mention in the NLQ and secondly simple text based retrieval methods can be used to find a limited number of suitable candidate entities [1].

The best fitting entity, given the question, can then be chosen from this candidate set based on simple string similarity and graph-based features. The entity span detector can

be implemented based on classifiers as well, either as a pair of classifiers (one for predicting the start position and one for predicting the end position of the span) or as a collection of independent classifiers, where in this case the entity span detector representing a sequence tagger (one for every position in the input, similar to a token classification network), where each classifier predicts whether the token at the corresponding position belongs to the entity span or not [1].

A simple concrete example of a full KGQA semantic parsing algorithm, which gives competitive performance on SimpleQuestions [13-18,34-39], relying on classification models, is the approach proposed by Mohammed et al. [1,82]. This QA system follows the following steps for predicting the entity-relation pair constituting the formal question:

1. Entity span detection: A bidirectional Long-Short Term Memory network (BiLSTM) [84] is used for sequence tagging, i.e. to identify the span *a* of the NLQ *q* that mentions the entity. In the example “Where was Frank Herbert born?”, the entity span could be “Frank Herbert”.

2. Entity candidate generation: Given the entity span, all KG entities are selected whose labels (almost) exactly match the predicted span.

3. Relation classification: A bidirectional gated recurrent unit (BiGRU) [85] encoder is used to encode *q*. The final state of the BiGRU [85] is used to compute probabilities over all relations using a softmax output layer, which is given from the equation [74-77]:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{3.1}$$

.

4. Logical form construction: The top-scoring entities and relations from previous steps are taken, and all their possible combinations are ranked based on their component scores and other heuristics. The top-scoring pair is then returned as the predicted logical form of the question.

For training the full system, the relations given in the formal queries of the dataset are used as correct output classes. For the span detector, we first need to extract “pseudo-gold” entity spans since they are not given as part of the dataset. This is done by automatically aligning NLQs with the entity label(s) of the correct entity provided in the training data, i.e. the part of the NLQ that best matches the entity label is taken as the correct span to train the sequence tagging BiLSTM [1].

Also another work that is related with my thesis, because in this work the entity span is identified, is the work by Mohammed et al. [82] and Petrochuk and Zettlemoyer [83] where the entity is disambiguated without using neural networks. Petrochuk and Zettlemoyer [83] employed a BiLSTM-CRF model for span prediction, which can no longer be considered a simple collection of independent classifiers and instead forms a structured prediction model that captures dependencies between different parts of the output formal query (in this case the binary labels for every position).

Also [82] investigates different RNN and CNN-based relation detectors and a BiLSTM and CRF-based entity mention detectors. Both [82] and [83] first identify the entity span, similarly to previous works, but disambiguate the entity without using neural networks.

Additionally Bordes et al. [1, 6, 86] propose a memory network (MemNN)-based solution to SimpleQuestions [13-18, 34-39]. They use bag-of-words representations for triples

and question and train a model that predicts the right triple by minimizing a margin-based ranking loss. They compute scores between questions and whole triples, including the triple objects. However, triple objects are answers and thus might not be present in the question, which may affect the performance adversely. The same work introduces the SimpleQuestions dataset [13-18, 34-39], consisting of approximately 100,000 question-answer pairs.

Referring in summary to other works on SimpleQuestions dataset [13-18, 34-39] typically predict the subject entity and predicate separately,unlike [87], which ranks whole triples. There are works, such as [87] which explore fully character level encoding of questions, entities and predicates, and use an attention-based decoder [73].

The model of He and Golub [87] consists of a character-level RNN-based question encoder and an attention-enhanced RNN decoder coupled with two separate character-level CNN-based entity label and predicate URI encoders. Given that their model works purely on character-level, which results in much longer input sequences, the effect of an attention mechanism could be more pronounced than with word-level models.

From the output size perspective, it is reasonable to assume that compared to machine translation an attention mechanism is of lesser importance in Simple QA where only two predictions need to be made per input sequence.

Additionally there are works that explore building question representations on both word- and character level [88]. Also there are works on SimpleQuestions dataset [13-18, 34-39] that have the aim to explore relation detection in-depth and propose a hierarchical word-level and symbol-level residual representation [89].

More specifically Yu et al. [89] employ CNNs and propose an attentive pooling approach. Firstly, the question is split into entity mention and relation pattern. Then, the entity mention is encoded using a character-level CNN and matched against a character-level encoded subject label. The relation pattern is encoded using a separate word-level CNN with careful max-pooling and matched against a word-level encoded predicate URI. Essentially in this approach an attention mechanism is used to obtain better matches for predicates instead of implicitly perform segmentation.

Both [88] and [90] works improve the performance of [89] by in corporating structural information such as entity type for entity linking. Also Dai et al. [90] investigate a word-level RNN-based approach,in which they encode the question on word-level only and train/use Freebase-specific predicate and entity representation.

[91] uses a BiLSTM+CRF tagger to improve the performance of [90]. The tagger is trained to predict which parts of the input correspond to the entity mention. The detected entity span is then used for filtering entity candidates. [83] estimates the upper-bound on accuracy for SimpleQuestions [13-18, 34-39], which is less than 83.4% due to unresolvable ambiguities (which is caused by the question lacking information to correctly disambiguate entities).

4. THE MAIN WORK-RELATION AND ENTITY SPAN PREDICTION

In my thesis the methodology that i followed consists of three main tasks that are executed: the first is the relation prediction, the second is the entity span prediction and the final task is the Question Answering model creation.

Regarding the initial dataset [13-18] , i.e all those .txt files, which either contains all questions or only the questions that could be answered over wikidata [19] or both of them, were used which contain the questions with entity ids and relation ids that have been obtained by annotating (let us say transforming) the questions from the Knowledge Base Freebase to the Knowledge Base Wikidata [88,89] (e.g. annotated_wd_data_train.txt, annotated_wd_data_valid.txt, annotated_wd_data_train_answearable.txt, annotated_wd_data _valid_answearable.txt [13-18]).

For these txt. files their entity labels are found through SPARQL queries and finally these txt. files are transformed [29-33] into .csv files [34-39]. So with these queries i achieve the first objective of my thesis (Section 1.3-Aim and Objectives), i.e the creation of a simple SPARQL query [2-5,29-33] that uses the entity id of each one of the questions of the training and the validation dataset of the SimpleQuestions dataset [13-18] in order to find its entity label [34-39].

Essentially i use these .csv files [34-39] in order to apply my methodologies and consequently the training and the validation process. Also when i refer the phrase ”full SimpleQuestions dataset” i mean that this dataset contains all the questions regardless if they are answerable over Wikidata [13-18,19] or not.

4.1 Relation Prediction

Firstly i create the relation vocabulary [41-54] and i found the index of each unique relation id of the relation vocabulary in this vocabulary through the functions relation_vocabulary [42-48,50-53] and find_relations_index [42-47,50,51] respectively.

The function relation_vocabulary [42-48,50-53], which takes as an argument the list of all relation ids of the training and validation dataset, creates and returns the relation vocabulary which is the list of unique relation ids present in the training and validation dataset of SimpleQuestions dataset [13-16,34-37]

Briefly through relation_vocabulary function[42-48,50-53], the relation vocabulary [41-54] is constructed by creating a list, by checking each time if a relation id is already in this list. If it is not already in the list (relation vocabulary [41-54]), then is added as a new unique relation id in the relation vocabulary [41-54], else the relation id is already in the list (relation vocabulary [41-54]) as a unique relation id.

Correspondingly through the function find_relation_index [42-47,50,51], which takes as an argument the relation vocabulary, returns for each relation id its position (index) in the relation vocabulary [41-54].

The length of the relation vocabulary [41-54] in the case of training and validating the full SimpleQuestions dataset [13,15,34,36] is equal to 129 and the length of the relation vocabulary for the case of training and validating the SimpleQuestions subset [14,16,35,37] that includes only questions that are answerable over wikidata[19] is equal to 125.

Finally i create a neural network model [40-54] through a class in python in order to predict

the relation index and thus to predict the relation id. This neural network includes the same layers as the entity span neural network, i.e a bert layer (bert-base-uncased model) [15,17,22-24], a linear layer [94] with 768 neurons for input and 129 neurons for output ,one layer consisting of an activation function [95], by a dropout layer[96], and softmax activation function layer[74-77], to pass the result obtained from the previous four layers and convert it into probability.

```
class BertRelationsClassifier(nn.Module):
  def __init__(self, activation, dropout_prob, rel_vocab_len):
    super(BertRelationsClassifier, self).__init__()
    activations = {
        "ELU" : nn.ELU,
        "LeakyReLU"    : nn.LeakyReLU,
        "Softplus"     : nn.Softplus,
        "Tanhshrink"     : nn.Tanhshrink
  }
    self.bert_model = BertModel.from_pretrained('bert-base-uncased')
    self.rel_vocab_len = rel_vocab_len
    self.fc = nn.Linear(768, self.rel_vocab_len)
    self.activation_type = activation
    self.dropout_probability = dropout_prob
    self.activation = activations[self.activation_type]()
    self.dropout = nn.Dropout(self.dropout_probability)
    self.softmax = nn.Softmax(dim=1)

 def forward(self, input_ids, attention_mask):
   out = self.bert_model(input_ids, attention_mask)
   out = self.fc(out[1])#CLS token-pooled output
   out = self.activation(out)
   out = self.dropout(out)
   out = self.softmax(out)
   return out
```

Listing 4.1: Code in Python for the neural network for relation prediction which is a python class named BertRelationsSpanClassifier.

The output of the linear layer [94], which is equal to 129, is the length of the corresponding relation vocabulary [41-54] created to group the unique relations and to find the relation index for each relation of the respective question.

4.2 Entity Span Prediction

The prediction of the entity start index and entity end index is concerned, could be done following the steps below:

- Initially each question, as well as the corresponding entity label, is splitted into words [41,49,54]. Then the parts of the question are created equal in length to that of their splitted entity label in words. These parts are stored in a list as well as the indexes of the initial word and final word of these question parts are stored in dictionaries with keys the question parts and values the indexes(positions) of the initial words and final words for each question. Everything mentioned above is done via the create_question_parts function [42-47, 50-51], shown below, which takes as arguments, the splitted question and the splitted entity label [41,49,54] and returns the list of question parts and dictionaries containing the indexes(positions) of the initial words and final words for each question part.

- After this first step it is more easier for me to find the entity start index and entity end indexes, because i can check for a question if its entity label is part of its wording [1,6]. About this indexes i can find them for each question of the training and the validation SimpleQuestions dataset [13-16, 34-37] through the following ways:

    1. The python spacy library and specifically the difflib.get_close_matches() [41-49,60] function could be used to find out for each entity label which part of its corresponding question are the same or very close to it either lexically (most times) or semantically. This method is applied between entity label and the parts of a question. If such parts are found that are identical or very close to the entity label, its position as entity start index and the position of the last word as entity end index are finally obtained from them.

       If no such question part is found, then via python's Sentence-transformers[56,57] the corresponding embeddings are created for both the entity label and each part of its question with a length equal to the length of the splitted entity label as mentioned above.

       More specifically, the pretrained model all-MiniLM-L6-v2 is used for the Sentence Transformer, which combines speed (5 times faster than all other pretrained models of the Sentence Transformer and (sentence) transformers in general) and good quality in terms of results [57].

       These embeddings are compared through the metric of cosine similarity [58,59] by creating a cosine similarity function [42,43,52,58,59],that takes as an argument the two encoded vectors, one for the entity label and one for a part of its corresponding question, and returns the cosine similarity score between the two vectors.

       This function give as a result the part of the question that has the highest possible similarity to the entity label. Therefore, it becomes very easy through the corresponding part of the question to define the index (position) of the first word of the question part as entity start index and the index (position) of the last word in this part of the question as entity end index.

       The function through which the embeddings are created via Sentence Transformer [56,57] and uses the cosine similarity function [42,43,52,58,59] which i mentioned earlier, best_match_with_cosine_similarity [42,43] which takes as arguments the entity label, the list of question parts, the dictionaries containing the indexes(positions) of the start words and end words for each question segment mentioned above, as well as the similarity model of Sentence Transformer [42,43,52,56,57] and returns the question part that has the highest similarity and the position of the initial and final word of that question part, otherwise if there is no such question part it returns None as the question part and as the index and for the initial and final word of the entity label the value -1.

       Finally, for the questions where no question part identical or similar to the corresponding entity label has been found through all the above methodology, which have been measured through the entity_span function [42,43] and are 17 for the training dataset of the full SimpleQuestions subset [34], 10 for the training dataset of the SimpleQuestions subset that include only answerable over Wikidata questions [19,35] and 2 for the validation dataset both for the full SimpleQuestions dataset [36] and the corresponding dataset including only answerable over Wikidata questions [19,37], are set as entity start index and entity end index the positions (indices) of the penultimate (question length-2) and the last word (question length-1) respectively. In this way the lengths of

the SimpleQuestions training and validation dataset [34-37] that are going to be used for training and validation process are kept constant.

The entity_span function [42,43] where the entity span start and entity span end indexes are found for each question, takes as arguments the list of all questions of SimpleQuestions training or validation dataset [13-16, 34-37], the list of all entity labels of SimpleQuestions training or validation dataset in question and the similarity model of Sentence Transformer [42,43,52,56,57].

This function calls the functions create_question_parts [42-47, 50-51] and best_ match_with _cosine_similarity [42,43] in order to create the parts of each question and find the question part that is equal or most similar to the entity label of the question with its start word index and end word index correspondingly and returns the list of indexes (positions) of the initial words of the entity labels within the question(entity span start indexes) and the list of indexes (positions) of the final words of the entity labels within the question (entity span end indexes).

2. The second way to find the entity span start and entity span end indexes is the same with the previously described methodology, with the only differences that instead of cosine similarity metric, i use jaccard similarity metric [58,59] in order to find the similarity score for a question between its entity label and a question part.

   The basic difference is that in order to calcuate the jaccard similarity metric, i do not use the Sentence Transformer library [56,57], but only the list of words for entity label and each question part respectively [44,45,53,56,57]. In this function [44,45] i create this list for each question part, while the corresponding list for entity label is created through entity_span function [44,45,53,56, 57], which is described earlier. Also the lengths of the SimpleQuestions training and validation dataset [34-37] that are going to be used for training and validation process are kept constant.

   Essentially the main changes are that instead of the functions cosine_similarity and best_match_with_cosine_similarity, i create the functions jaccard_similarity [44,45,53,58,59] and best_match_with_jaccard_similarity [44,45] respectively, where the entity_span function has changed accordingly [44,45].

3. An alternative way of finding the entity span start index and entity span end index is to create for each question an entity span of length equal to the length of the question (splitted in words), which will consist of 1 if the corresponding entity label words are present in the words of the question part which is equal to the entity label (i.e There is a part of the question to be equal with the entity label splitted in words) and 0 if it is not [1,6]. Essentially this approach focuses on the existence of the entity label itself exclusively within its corresponding question, rather than searching for some similar part of the question to the entity label.

   So, i use the function create_question_parts [42-47,50,51] mentioned in the first approach, the list with all parts of each question, the dictionaries with keys the parts for each question and values the indexes (positions) of the initial words and final words for each question.

   Then through the create_entity_span [46,47,50,51] function that takes as arguments the question splitted in words [41,49,54] and the indexes of the initial and final word of the part of the question that is equal to the corresponding entity label, the entity span for each question is created.

   In this methodology because the words that make up the entity label must be in consecutive positions within the question, it is always necessary that when

creating the entity span, the index specifying the position of the starting word of the equal question part with the corresponding entity label for the question must be less than or equal to the corresponding index of the ending word of that equal question part.

This indirectly ensures that the ones that enter the entity span are consecutive and equal in number to the length of the entity label splitted in words [41,49,54]. This function returns the entity span, as well as the number of consecutive ones in it for each question.

The find_start_and_end_index function [46,47] takes as arguments the entity span, the entity label, the list of all the parts of the question, the dictionaries with keys the parts for each question and values the indexes (positions) of the start words and end words for each question, the length of the question and the number of consecutive ones of the entity span which are counted in this function.

From the above function the question part that is equal with the entity label, the position of the initial word (index of the first one in the entity span) of the entity label and respectively the position of the final word (index of the last one in the entity span) could be found. If the entity span that is created for a question is not full of zeros, then a logical variable is used to confirm that all ones present are contiguous.

Thus, the index of the beginning of the entity label is considered to be the index of the first one that existing in the entity span and the index of the end of the entity label is considered to be the value of the index of the first one of the entity span to which the number of consecutive ones of the entity span is added and the value 1 is subtracted.

Essentially through this methodology the entity start index and the entity end index must be exactly the same with the indexes of the start word and the end word of the question part that is equal to the entity label, from the corresponding dictionaries including start and end indexes of question parts of a question which are returned from the function create_question_parts [42-47,50,51], respectively.

In the case, that there is no question part that is equal with the entity label, the entity span will consist only of zeros, so there is no word (i.e 1 in the entity span) to derive the corresponding indices for entity start and entity end.

To deal with this situation, initially from the list of question parts as obtained from the create_question_parts function [42-47,50,51] through spacy library [41-49,60], it is searched, which part is closest to the entity label and for that question part from the dictionaries containing the indexes(positions) of the initial words and the final words for it obtained from the same function, the corresponding positional indexes of its initial and final words are found.

Next, if no question part that is found by difflib.get_close_matches() [41-49,60] is even minimally similar to the entity label, then the positions (indices) of the penultimate (question length-2) and the last word (question length-1) are set as the entity start index and entity end index, respectively. In this way the lengths of the training and validation dataset that will be used for the training and validation process respectively are kept constant. The logical variable that confirms the existence of consecutive ones in the span is returned by this function, as well as the positional indices of the initial and final word of the entity label.

All of the above is implemented via the entity_span_encoded function [46,47],

which takes as arguments the list of all questions of either the full SimpleQuestions dataset (training or validation) [13,15,34,36] or the corresponding dataset with only answerable over wikidata questions [14,16,19,35,37], the list of all entity labels of the SimpleQuestions training or validation dataset (full or only with answerable over wikidata) [13-16,19,34-37] and returns the list of entity spans, as well as the lists of start and end position indices for each entity label for all questions. It is worth noting that this function counts the number of questions whose entity span consists of 0 and 1, where the ones are contiguous.

4. The last methodology that i created in order to find entity span start index and entity span end index is to create for each Simple Questions training and validation dataset question is the same as the previous methodology, with the only difference that i use for training and validation, only the questions that one of its parts is equal to their corresponding entity label. Thus the lengths of the training and of the validation dataset that will be used for the training and validation process respectively are smaller.

   So the first change compared to the previous methodology is that in the function find_start_and_end_index [50,51], there is no need to find the entity span indexes for the questions with no equal part with the entity label either through the spacy library [41-49,60] or by putting some specific values as the values of indexes. Another change is that the boolean value that the function find_start_and_end_index [50,51] returns is useful for the entity_span_encoded [50,51] function not only to count how many questions have their entity label equal to one of their parts, but also to insert into lists all these questions, their corresponding entity labels, entity spans, entity span start indexes, entity span end indexes and relation ids.

It is worth mentioning that all the words in entity labels and the questions have been converted to words with lowercase letters, regardless of whether they are capitalized or not, in order to have better comparison between question parts and entity labels.

- Finally i create the neural network in order to predict the entity start index and entity end index [40-54], through a class in python which includes two heads one entity start head and one entity end head. Each of these heads consists of a bert layer (bert-base-uncased model) [15,17,22-24], a linear layer [94] with 768 neurons for input and 33 neurons for output, which is the maximum question length that is observed in all the SimpleQuestions dataset [13-18, 34-39], one layer consisting of an activation function [95], by a dropout layer[96], where at the end the result obtained from the above four layers is passed through the softmax activation function [74-77], to convert the results into probabilities.

```
class BertEntitySpanClassifier(nn.Module):
  def __init__(self, activation, dropout_prob, max_quest_len):
    super(BertEntitySpanClassifier, self).__init__()
    self.bert_model = BertModel.from_pretrained('bert-base-uncased')

    activations = {
        "ELU" : nn.ELU,
        "LeakyReLU"    : nn.LeakyReLU,
        "Softplus"     : nn.Softplus,
        "Tanhshrink"     : nn.Tanhshrink
  }
    self.activation_type = activation
    self.activation = activations[self.activation_type]()
    self.max_length = max_quest_len
```

```
        self.dropout_probability = dropout_prob
        self.start_head = nn.Sequential(
              nn.Linear(768, self.max_length),
              self.activation,
              nn.Dropout(p=self.dropout_probability),
              nn.Flatten(),
              nn.Softmax(dim=1)
        )

        self.end_head = nn.Sequential(
              nn.Linear(768, self.max_length),
              self.activation,
              nn.Dropout(p=self.dropout_probability),
              nn.Flatten(),
              nn.Softmax(dim=1)
        )

    def forward(self, input_ids, att_mask):
        out = self.bert_model(input_ids, att_mask)
        out = out[0]
        out_start = self.start_head(out)
        out_end = self.end_head(out)
        return out_start , out_end
```

Listing 4.2: Code in python for the neural network for entity span prediction which is a python class named BertEntitySpanClassifier.

It is worth mentioning that the input of the two aforementioned neural networks for relation and entity span prediction [40-54], i.e the input ids and the attention masks are found through a custom dataset that is created [42-47,50,51,54]. This custom dataset has as input arguments the relation indexes of the unique relation ids of the relation vocabulary [41-54], the entity span start indexes, the entity span end indexes, the tokenizer BertTokenizerFast [25,26], the maximum question length that is found in the SimpleQuestions dataset [13-19, 34-39] (either full or only with answerable over wikidata questions), a pytorch generator [98] and a logical value shuffle with default value to be true.

The generator is reset by creating an instance (self.g) in order to shuffle differently but reproducibly every epoch of the relation or entity span prediction models [99]. This custom dataset class preprocesses the SimpleQuestions dataset questions (questions For preprocessing since the span start and end correspond to actual words, with the BERT tokenizer I need to locate the corresponding start and end tokens in the BERT tokenized question for each entity [41]. Finally this function returns the input_ids, attention_masks, entity span start and entity span end token positions, and relation IDs for each question as PyTorch tensors [41].

5. THE MAIN WORK-THE QUESTION ANSWERING ENGINE MODEL

The final and most important task of my thesis methodology is the creation of the Question Answering Engine. I have created three types of Question Answering Engines (QA Engine):

1. QA engine by using the Sentence Transformers library combined with the cosine similarity metric [42,43,52,56-59].
2. QA engine by using only the jaccard similarity metric [44,45,53,58,59].
3. QA engine by using only the spacy library [41-49,60].

It is worth mentioning i create an .txt file named ”Questions_testing” [100] that contains 60 questions with their corresponding entity id entity label relation id and answer id. In order to create this .txt file i use questions from SimpleQuestions dataset [13-18, 34-39] either the questions themselves or other questions similar with them, with regard to the wording. Also i create some questions on my own, for which of them i search for answer ids,answer label, relation id, entity id on wikidata [19].

This .txt file is going to be use for all question answering engines in order to see which QA engine answers either correctly or generally giving an answer to the most questions out of these 60 questions. Thus i will have a clearer view about which QA engine works better.

Also in order to create SPARQL queries that are needed for this work i download the SPARQLWrapper from python [29], with the piece of code from which the SPARQLWrapper [29] is downloaded could be found on Section 4-Dataset Generation.

5.1 Question Answering engine creation

More analytically i create the Question Answering Engine through the following steps:

1. The preprocessing of a question is done by calling the function preprocessing, which takes as argument the text (in this case the question) and performs accent removal, as well as punctuation mark removal (period, question mark, exclamation mark), s removal with apostrophe(’s) and comma(,) removal [26,34]. Finally this function returns the original text with the above mentioned processing [41-54].
2. I create the function question_encoding [48,52,53], which calls the the function preprocessing and takes as arguments the question and the corresponding tokenizer, in this case BertTokenizerFast [25]. Finally the function question_encoding encode the given question through the command tokenizer.encode_plus() [26]. The ids and masks resulting from the corresponding encoding are returned.
3. Creation of functions to predict entity span start index, entity span end index and relation index. The function for predicting the entity span start index and entity span end index takes as arguments the ids and the mask resulting from the encoding of the question via the question_encoding function, the aforementioned dictionaries Endict and entity_quests (explained below), the pretrained model of the Sentence Transformer similarity model [56,57], the entity_model_path, i.e. the path where the

best parameters for the best predictive model of entity span start index and entity span end index are stored, the activation function for the best predictive model of entity start index and entity end index, the dropout probability, dropout_prob and the SEED number which helps to have the same and reproducible answers. The predictions for entity span start index and entity span end index, are returned by this function.

Correspondingly the function for the prediction of relation index, takes as arguments the ids and mask resulting from the encoding of the question through the question_encoding function, the length of the relation vocabulary, where its creation is described in chapter 4, the relation_model_path, i.e. the path where the best parameters for the best prediction model of relation index are stored, the activation function of the best prediction model of relation index prediction,the dropout probability, dropout_prob and the SEED number. The prediction for the relation index gives through the relation vocabulary [41-54] which relation id is in the question. This relation index is returned.

Also in the case that for a QA engine i do not use the Sentence Transformers similarity model [56,57], i.e for the QA engine include either only spacy library [48,60] or jaccard similarity metric [53,58,59], the function entity_prediction does not have the dictionary quests and the Sentence Transformers similarity model [56,57] as arguments.

4. I create the get_entity_id function that takes an entity label as an argument, which involves creating a SPARQL [2,3,4,5] query that, given an entity label, finds and extracts its corresponding entity id [29,48,52,53,101].

5. The next step is the creation of two dictionaries, where one named Endict will have as keys the entity labels of the training, validation and test dataset and as values the corresponding entity ids of the training, validation and test dataset [48,52,53].

   The other dictionary named quests, which is needed when we create the QA engine by using Sentence Transformers library and cosine similarity metric [52,56-59], has as keys the entity labels of the training, validation and test dataset and as values the corresponding encoding vectors from the Sentence Transformer [56-57] for each entity label of the training, validation and test dataset [52].

   The steps of creating the above two dictionaries is extremely helpful when in the QA engine are entered questions of the SimpleQuestions dataset [34-39]

6. Creation of the function find_closest_match [48,52,53] that takes as arguments the entity label, the dictionary Endict with keys the entity labels and values the entity ids, a list of all keys of Endict, the dictionary entity_quests with keys the entity labels and values the encodings obtained for each entity label through the pretrained similarity model of Sentence Transformer [42-45,56,57].

   Through this function a way that the entity id for the entity of a question could be found is by using spacy library [41-49,60] between the entity label of a user question and the list of Endict keys [90], i.e the entity labels of the SimpleQuestions training and validation dataset [34-37] and the more similar entity label is returned, if the entity id for the entity of a query cannot be found through the get_entity_id function [29,48,52,53,101]. In this case the entity id of the entity that has the best cosine similarity score with the entity for which we are looking for the entity id is returned, if available.

The same function, if i want to use only the cosine similarity metric [42,43,52,58,59], takes the entity label, the dictionary Endict with keys the entity labels and values the entity ids, a list of all keys of Endict, the dictionary quests with keys the entity labels and values the encodings obtained for each entity label through the pretrained similarity model of Sentence Transformer [42,43,52,56-59].

Through this function the entity id of a question is found by the evaluation of the cosine similarity metric score between the pretrained similarity model encoding of this entity label for a question and each value of the dictionary quests [42,43,52,56-59]. The entity label with the best cosine similarity metric score [52,56-59] is kept and through Endict dictionary the corresponding relation id is found.

If i want to use only the jaccard similarity metric instead of cosine similarity metric [44,45,53,58,59], the find_closest_match [53] remains the same compared with the corresponding function [44,45,53,58,59] including cosine similarity metric [58,59] with the differences that a different similarity metric is used and Sentence Transformers library [56,57] is not used.

7. To extract the final answer or answers for each question, i create the function named answer which takes as arguments the question itself, the dictionary Endict the dictionary entity_quests if the Sentence Transformer similarity model [56,57] is used, the tokenizer BertTokenizerFast [25], the pretrained similarity model of the Sentence Transformer [56,57], if the cosine similarity metric [58.59] is used, the entity_model_path, i.e. the path where the best parameters for the each one of the models created (described in 1.4 chapter) of entity start index and entity end index are stored, the relation vocabulary [41-54], the relation_model _path, i.e. the path where the best parameters for the best prediction model of the relation index are stored, the activation function of the best prediction model (described in chapter 4) of the relation index, the activation functions of the best prediction models of the entity start index and entity end index (described in chapter 4) and the dropout probability, dropout_prob.

   Also the answer function calls the aforementioned function question_encoding in order to encode the question and get the ids and the mask that are needed for the functions entity_prediction and relation_prediction, which are also called from the function answer. This function returns a boolean value which shows if a question has an totally correct answer and also a boolean value which shows if a question has answer(s) either correct or incorrect or both of them, regardless if these answers include the expected answer of a question. Additionally the answers of this question, with the limitation of maximum 15 answers, are printed.

   Also entity_prediction function calls the function find_closest_match, when for the entity label occurred from prediction of the entity span indexes (start,end) there is no entity id from wikidata [19] that matches to it through the SPARQL query of the get_entity_id function, in order to find the most similar entity id for this question [29,48,52,53,101].

   More specifically if the QA engine contains only spacy library [41-49,60] the answer is totally correct if at least one of the relation id and entity id has predicted correctly and the list of predicted answers has one common element with the real answer according to the Questions_testing_with_answers.txt file [100].

   Also when at least one of the relation id and entity id has predicted correctly, while the question to be answered with either only correct or only incorrect answers or both

of them, but the expected answer is not one of these answers, then the questions is assumed as answered.

For these three QA engines that i create a question is not answered correctly if both of relation id and entity id is not predicted correctly. When i refer to the correct prediction of entity id and relation id, i mean the predictions to give the expected values for entity id and relation id, according to the ”Questions_testing_with_answers” .txt file [100]. Also For these three QA engines that i create a question has no answer if for the combination of the predicted relation id and entity the corresponding SPARQL query [2,3,4,5] gives an empty list with no answers.

If the QA engine contains either the combination of Sentence Transfromers library with the cosine similarity metric or only jaccard similarity metric [52,53,56-59] the answer is totally correct if at least one of the relation id and entity id has predicted correctly and either for a list with only one predicted answer, this answer has cosine similarity metric score with the real answer greater than 80% or also for a list with more than one predicted answers the maximum cosine or jaccard similarity metric score [58,59] between the real answer of a question according to the Questions_testing_with_answers.txt file [100] and one or more answers of the predicted relation id and entity id occurred from SPARQL [2,3,4,5] for the same question is greater than 80% and the corresponding minimum cosine or jaccard similarity metric score between the real answer and one of more predicted answers are lower than 5

For any other cosine similarity metric scores when at least one of the relation id and entity id has predicted correctly, while the question to be answered with either only correct or only incorrect answers or both of them, regardless if the expected answer is one of these answers, then the questions is assumed as answered.

8. To get the answer (or answers) the function answer calls the get_answer function which takes as arguments the entity and relation obtained by calling the entity_prediction and relation_prediction functions, through which the best parameters of the best prediction models for relation index, entity start index and entity end index are loaded. The get_answer function includes the corresponding SPARQL query [29,30,48,52,53,101] to find the answers, depending on whether the relation of the given question starts from R or from P. The answers of the question are returned.

   So all the above described process helps me to achieve the objective of creating the above QA engine models below(QA engine including spacy [41-49,60], QA engine including the Sentence Transformer similarity model combined with cosine similarity metric [52,56,57,58,59], QA engine including jaccard similarity metric [53,58,59]) (Section 1.3-Aim and Objectives).

   Finally i called the answer function to get the answer(s) for each question we put in it, thus calculating the percentage of questions that were answered(totally correct, totally incorrect, neither totally correct nor totally incorrect) [48,52,53].

6. EVALUATION

6.1 Evaluation experiments

In this section the best arithmetic results about the relation indexes, entity span start indexes and entity span end indexes for all the methodologies that i applied are shown. For the neural networks for relation and entity span prediction (described in chapter 1.4), i made experiments by using hyperparameters such as activation function [94], dropout probability [95], optimizer [102-105], learning rate [106,107], loss function [108,109,110] and also by using the scheduler get_linear_schedule_with_warmup [111,112].

More specifically in these experiments i use dropout probability values of 0.1, 0.2, 0.3 and 0.4 [95] and learning rate values of 2e-5 and 4e-5 [106,107]. Also i use the ELU, LeakyReLU, Tanhshrink and Softplus activation functions [94], the optimizer Adam [104,105] and the cross-entropy loss function [109]. The arithmetic results include the training loss, the validation loss, the accuracy score, the f1 score, the precision score and the recall score [55]. Also i plotted the learning curve between the training and the validation loss both on the relation prediction and on entity span prediction.

Additionally in order to improve the learning curve of the losses, but also the performance of the algorithms in general, i applied the method of early stopping, according to which if in one epoch no increase of the unique f1 value [20] for the prediction of the relation is observed for the validation dataset, but also the f1 values [55] for the entity start index and for the entity end index for predicting the corresponding indices for the entity span, then the training and validation process of the algorithm is stopped, because the performance of the model will not improve and there is a risk of overfitting [113,114,115].

From the procedure that i follow in order to find the entity label using SPARQL [2,3,4,5] (explained in Section 4), it is worth noting that there are cases that the entity label which is found from SPARQL [2,3,4,5] is not appearing in the question wording. For example a question of the train dataset containing answerable over wikidata [14,19,35] questions only is ”What a superhero movie that premiered on toonami” and the corresponding entity label for this question that occurred from SPARQL [2,3,4,5] is superhero film. This means that the entity label is not part of its question wording, although they have the same meaning.

This is an example of some similar cases for both the datasets with all the questions and the dataset with only the answerable questions over wikidata [13-19, 34-39]. Other examples in this datasets that this phenomenon appears is when the entity label could not be found from SPARQL [2,3,4,5] for a question and as entity label is assumed the corresponding entity id.

6.2 Evaluation results

The tables 6.1-6.4 show the best arithmetic results for the methodology of creating a list consisting of 0 and 1, depending on if the question includes its entity label, then there are ones in the list, else the list consists only of zeros. This list is the entity span for each SimpleQuestions training and validation dataset [13-16, 34-37].

More specifically for the results of the tables 6.1 and 6.2 there are trained only the questions of the SimpleQuestions training dataset that include their corresponding entity label [34,35,50,51], which is found from SPARQL (explained in Chapter 4). Respectively for the

validation process there are used only the questions of the SimpleQuestions validation dataset that include their entity label in their wording [36,37,50,51]. The best evaluation results for all the methodologies that i applied in this thesis are achieved for the hyper-parameter values of learning rate = 2e-5 [106,107] and dropout probability = 0.3 [96].

It is worth mentioning that when i refer in this section the word wording i mean the wording of a SimpleQuestions dataset question [13-19, 34-39]. Also i assume as the best evaluation results for each methodology, the results where the max evaluation metric score i.e accuracy,f1,precision and recall [55] occurred for the relation index prediction and at least for one of the entity span start index or entity span end index prediction.

Evaluation Metrics	Relation index	Entity span start index	Entity span end index
F1-score	0.95998	0.97032	0.96066
Recall-score	0.95998	0.97032	0.96066
Precision-score	0.95998	0.97032	0.96066
Accuracy-score	0.95998	0.97032	0.96066

Table 6.1: Best evaluation results about relation index, entity span start and entity span end index for the methodology of creating a list consisting of 0 and 1, depending on if the question includes its entity label in its wording, then there are ones in the list, else only 0. For these evaluation results is used the SimpleQuestions dataset which includes all the training and validation questions either they are answerable over wikidata or not. For these questions their entity label must exists in their wording.

Evaluation Metrics	Relation index	Entity span start index	Entity span end index
F1-score	0.94207	0.97180	0.95732
Recall-score	0.94207	0.97180	0.95732
Precision-score	0.94207	0.97180	0.95732
Accuracy-score	0.94207	0.97180	0.95732

Table 6.2: Best evaluation results about relation index, entity span start and entity span end prediction for the methodology of creating a list consisting of 0 and 1, depending on if the entity label exists in question wording, then there are ones in the list, else 0. For these evaluation results is used the SimpleQuestions subset which includes only training and validation dataset questions that are answerable over wikidata. For these questions their entity label must exists in their wording.

The figures 6.1 and 6.2 display the learning curves between the training loss and the validation loss for the relation index prediction and the prediction of entity span start and entity span end index respectively for the case of using the full SimpleQuestions dataset which includes all the training and validation dataset questions [34,36]. More especially this applies for questions that their entity label, which is found from SPARQL [2,3,4,5], is a part of their wording [50,51].

Correspondingly the figures 6.3 and 6.4 display the learning curves between the training loss and the validation loss for the relation index prediction and the prediction of entity span start and entity span end index respectively for the case of using the SimpleQuestions dataset which includes only the training and validation dataset questions that are answerable over wikidata [35,37]. This applies to these questions that their entity label is a part of their wording [50,51].

For the above methodology (Entity span as a list consisting of O and 1- All questions that their entity label, which is found from SPARQL [2,3,4,5] queries, exist in their wording) [50,51]:

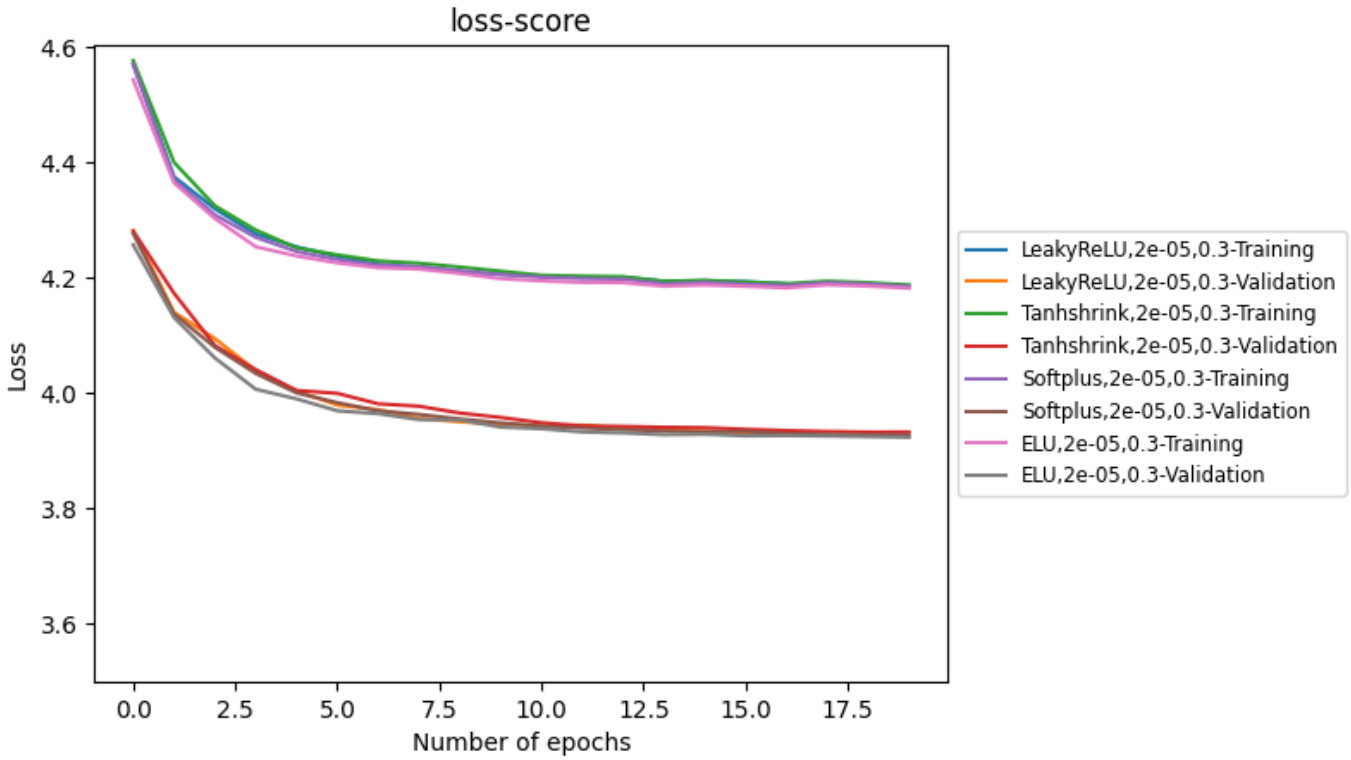

Figure 6.1: Learning curve between relation training and validation loss- methodology of finding entity span indexes (start and end) through entity span, which is a list exclusively consisting of 0 and 1 for the SimpleQuestions dataset that includes all the questions regardless of whether questions are answerable over wikidata or not, where their entity label is a part of their wording.

1. About entity span start and end index prediction, when i use the corresponding dataset (training and validation dataset), which includes only the questions that are answerable over wikidata [14,16,19,35,37], the best results are achieved for the activation function Softplus [95].

2. When i use the corresponding dataset (training and validation dataset) which includes all the questions (answerable and non answerable over wikidata [13,15,34,36]), the best results about entity span start and end index prediction are reached for the activation function ELU [95]. It is worth mentioning here that, i choose the experiment that includes ELU [95] as the best for this case because it has the highest possible value of entity span start and entity span end index prediction scores simultaneously ,which are equal to 0.97032 and 0.96066 respectively, from all other experiments (with activation functions LeakyReLU, Softplus and Tanhshrink [95]).

3. Through the activation function ELU [95] the best results for relation index prediction are achieved both in the case of using the dataset with all questions and in the case of using the dataset with only the answerable over wikidata questions [14,16,19,35,37].

The tables 6.3 and 6.4 show the best arithmetic results for the methodology of creating the entity span as a sequence 0 and 1, regardless of whether the question includes its entity label in its wording. Essentially i trained and validate the questions included in the SimpleQuestions training and validation dataset with all the questions (e.g annotated_wd_data_train.txt) [13,15,19,34,36] respectively either include in their wording their corresponding entity label, which is found from SPARQL [2,3,4,5](described in Chapter 4), or not.

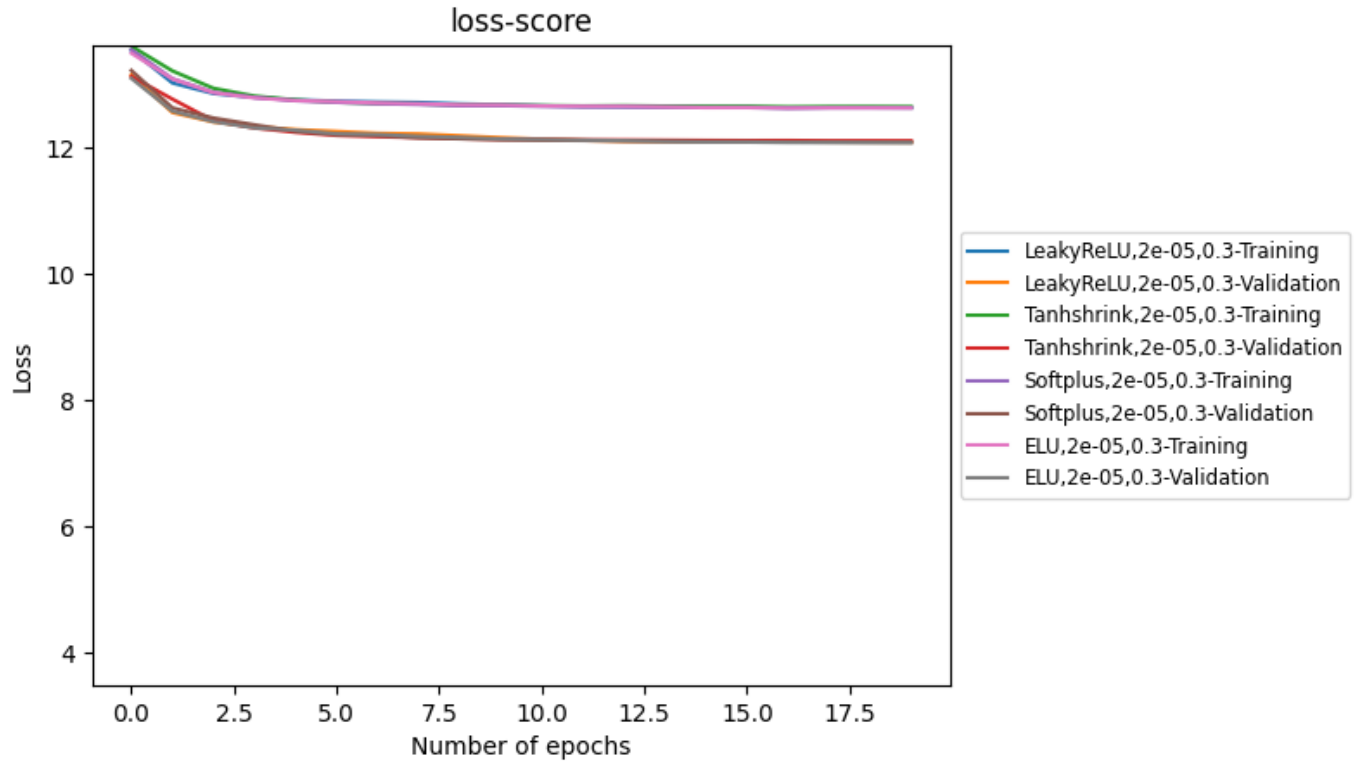

Figure 6.2: Learning curve between entity span training and validation loss- methodology of finding entity span indexes (start and end) through entity span, which is a list exclusively consisting of 0 and 1, for the SimpleQuestions dataset that includes all the questions regardless of whether questions are answerable over wikidata or not. The entity label of these questions is a part of their wording.

For the above methodology (Entity span as a sequence of O and 1- All questions either if their entity label exist in their wording or not) about entity span start and end index prediction, when i train and validate the corresponding dataset questions (training, validation, test), for the SimpleQuestions dataset which includes only the answerable over wikidata questions [14,16,19,35,37], the best results are achieved for the activation function ELU [95], while for the case of SimpleQuestions dataset which includes all the questions (answerable, non-answerable over wikidata [13,15,34,36]), the best results are achieved for the activation function Softplus[95].

About relation index prediction the best results are achieved for the activation function ELU [95], regardless if i train and validate all SimpleQuestions dataset questions (answerable and non-answerable over wikidata [13,15,34,36]) or only questions that are answerable over wikidata [14,16,19,35,37].

The figures 6.5 and 6.6 display the learning curves between the training loss and the validation loss for the relation index prediction and the prediction of entity span start and entity span end index respectively for the case of using the SimpleQuestions dataset with all the training and validation questions (answerable and non-answerable over wikidata [13,15,34,36]) regardless of whether their entity label is a part of their wording.

Also the figures 6.7 and 6.8 show the learning curves between the training loss and the validation loss for the relation index prediction and the prediction of entity span start and entity span end index respectively for the case of using SimpleQuestions dataset with training and validation questions that are answerable over wikidata [14,16,19,35,37], regardless of whether their entity label, which is found from SPARQL, is a part of their wording.

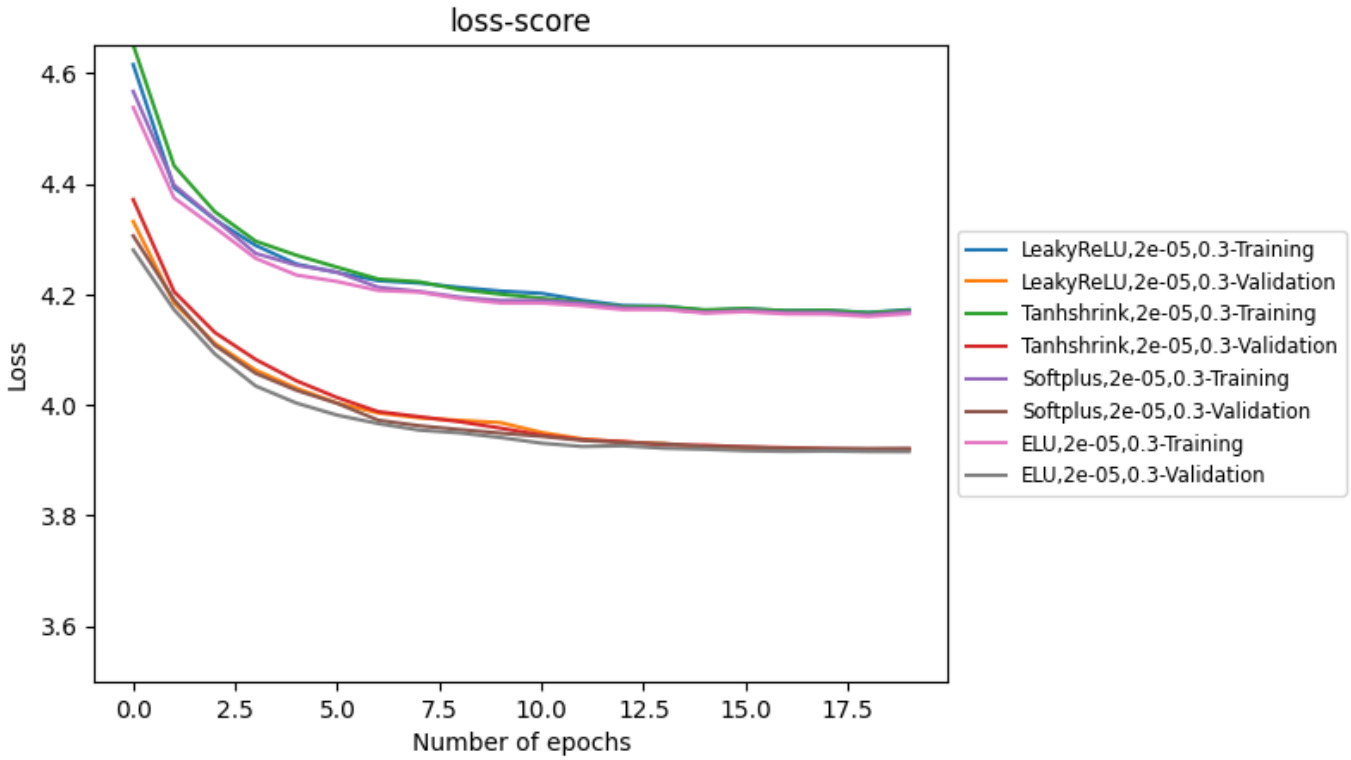

Figure 6.3: Learning curve between relation training and validation loss-methodology of finding entity span indexes (start and end) through entity span , which is a list consisting of 0 and 1 or only of zeros, for the SimpleQuestions dataset that includes all the questions that answerable over wikidata. The entity label of these questions is a part of their wording.

The tables 6.5 and 6.6 show the best arithmetic results for the methodology of finding the entity span start and entity span end index by using spacy library [41-49,60] or the Sentence Transformers library with the cosine similarity metric as the similarity metric [42,43,56-59]. This methodology is described analytically in chapter 1.4.

For this methodology of finding entity span start and entity span end index by using spacy library [41-49,60] or Sentence Transformers library combined with the cosine similarity metric [42,43,56-59]) about entity span start and end index prediction, when i train and validate the corresponding dataset questions (training, validation, test), for the SimpleQuestions dataset which includes only the answerable over wikidata questions [14,16,19,35,37], the best results are achieved for the activation function Softplus [95], while for the case of SimpleQuestions dataset which includes all the questions (answerable, non-answerable over wikidata [13,15,34,36]), the best results are achieved for the activation function ELU [95]. About relation index prediction the best results are achieved for the activation function ELU [95], both for the whole SimpleQuestions dataset and the dataset including only the answerable over wikidata questions [14,16,19,35,37].

The tables 6.7 and 6.8 show the best arithmetic results for the methodology of finding the entity span start and entity span end index by using spacy library [41-49,60] or the Sentence Transformers library with the jaccard similarity metric as the similarity metric [44,45,56-59]. This methodology is described analytically in chapter 1.4.

For this methodology (Finding entity span start and entity span end index by using spacy library jaccard similarity metric [41-49,58-60]):

1. About relation index prediction, the best results are achieved for the activation func-

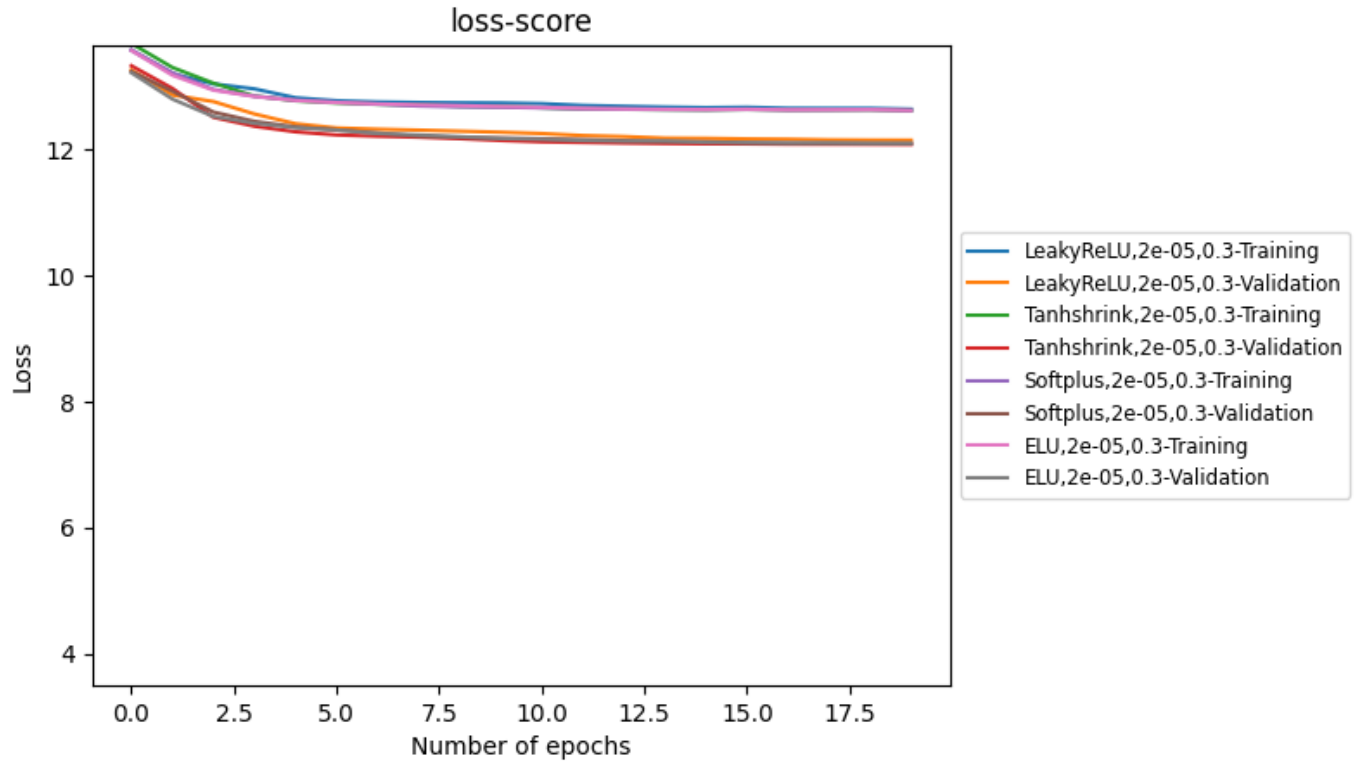

Figure 6.4: Learning curve between entity span training and validation loss-methodology of finding entity span indexes (start and end) through entity span, which is a list consisting of 0 and 1 or only of zeros, for the SimpleQuestions dataset that includes all the questions that answerable over wikidata. The entity label of these questions is a part of their wording).

tion ELU [95] either for full SimpleQuestions dataset [13,15,34,36], or with the subset contains only answerable over wikidata questions [14,16,19,35,37].

2. About entity span indexes (start and end) prediction, when i use the full SimpleQuestions dataset [13,15,19,34,36] and also when i use (train and validate) the corresponding SimpleQuestions dataset (training and validation), which contains only questions that are answerable over wikidata [14,16,19,35,37], the best results are achieved for the activation function Softplus[95].

So about entity span prediction the experiment that achieve the highest possible score for the entity span start prediction and the entity end prediction, i assume them as the most successful. For example in the methodology that i use either Sentence Transformer with cosine similarity metric [42,43,56,57] or spacy library [41-49,60] to find entity span start and end indexes for questions that are answerable over wikidata [14,16,19,35,37] in order to predict them layer, in entity span prediction the experiment that i use the activation function ELU [95], gives entity span start index prediction score for all types of scores (accuracy, f1, precision, recall [55]) equal to 0.93610 and entity span end index prediction score equal to 0.92065. The respective experiment that i use the activation function Softplus [95] give entity span start and end prediction scores equal to 0.93083 and 0.92591 correspondingly.

I finally kept the experiment with Softplus [95], because it has the max prediction score for entity span end compared with the corresponding score of experiment with Softplus [95], and also because although Softplus [95] has higher entity span start score, the difference between these entity span start prediction scores is small.

From all these experiments, the best evaluation relation or entity span model parameters

Evaluation Metrics	Relation index	Entity span start index	Entity span end index
F1-score	0.96058	0.93403	0.93648
Recall-score	0.96058	0.93403	0.93648
Precision-score	0.96058	0.93403	0.93648
Accuracy-score	0.96058	0.93403	0.93648

Table 6.3: Best evaluation results about relation index, entity span start and entity span end index for the methodology of creating a list consisting of 0 and 1, depending on if the question has its entity label as a part of its wording, else a list full of zeros. For these evaluation results is used the SimpleQuestions dataset that includes all the questions regardless of whether questions are answerable over wikidata or not.

Evaluation Metrics	Relation index	Entity span start index	Entity span end index
F1-score	0.94979	0.93153	0.92521
Recall-score	0.94979	0.93153	0.92521
Precision-score	0.94979	0.93153	0.92521
Accuracy-score	0.94979	0.93153	0.92521

Table 6.4: Best evaluation results about relation index, entity span start and entity span end prediction for the methodology of creating a list consisting of 0 and 1, depending on if the question has its entity label as a part of its wording, else a list full of zeros. For these evaluation results is used the SimpleQuestions dataset that includes questions that are answerable over wikidata.

and the best evaluation scores that are occurred, i observe that the relation prediction scores on the full SimpleQuestions dataset [13,15,34,36,55], regardless of the methodology that i followed in each case, gives very good results after 20 epochs, with the best values of all scores, i.e f1 score, recall score, precision score, accuracy score [55] ranging from 94.2% to 96.1%. For the dataset that contains only answerable questions over wikidata [14,16,19,35,37] the corresponding scores are a little bit lower in the range of 94.2% to 95% for all the methodologies that i applied. The best f1, accuracy, precision and recall scores are achieved with the activation function of ELU [95] for the SimpleQuestions dataset [13,15,34,36,55] that includes all the questions(table 6.3), for the methodology of finding the entity span indexes (start and end) by creating lists that consists of 0 and 1 or only zeros and assuming these lists as entity spans.

About the entity span and prediction the methodology that i achieve the best evaluation scores is the methodology that i create the lists that consists of 0 and 1 or only zeros and assuming these lists as entity spans only for the questions that have their entity label, which is found from SPARQL [2,3,4,5,29,30], as a part of their wording when when the full SimpleQuestions dataset [13,15,34,36] is trained and validated.

Essentially the best entity span start prediction scores for all the methodologies in the case of training and validating the full SimpleQuestions dataset (answerable and non-answerable over wikidata) questions [13,15,34,36,55] are ranging from 92.4% to 97% while in the case of training and validating the SimpleQuestions dataset including only questions that are answerable over wikidata [14,16,19,35,37,55] questions they are ranging from 92.9% to 97.2%. The corresponding ranges for the best entity span end prediction scores are 92.7% to 96.1% for the full SimpleQuestions dataset [13,15,34,36,55], 92.2% to 95.8% for the SimpleQuestions dataset including only answerable over wikidata [14,16,19,35,37,55].

More specifically i achieve the highest scores for the whole SimpleQuestions dataset [13-16, 34-37] with the ELU [95] activation function (table 6.1) and for the SimpleQues-

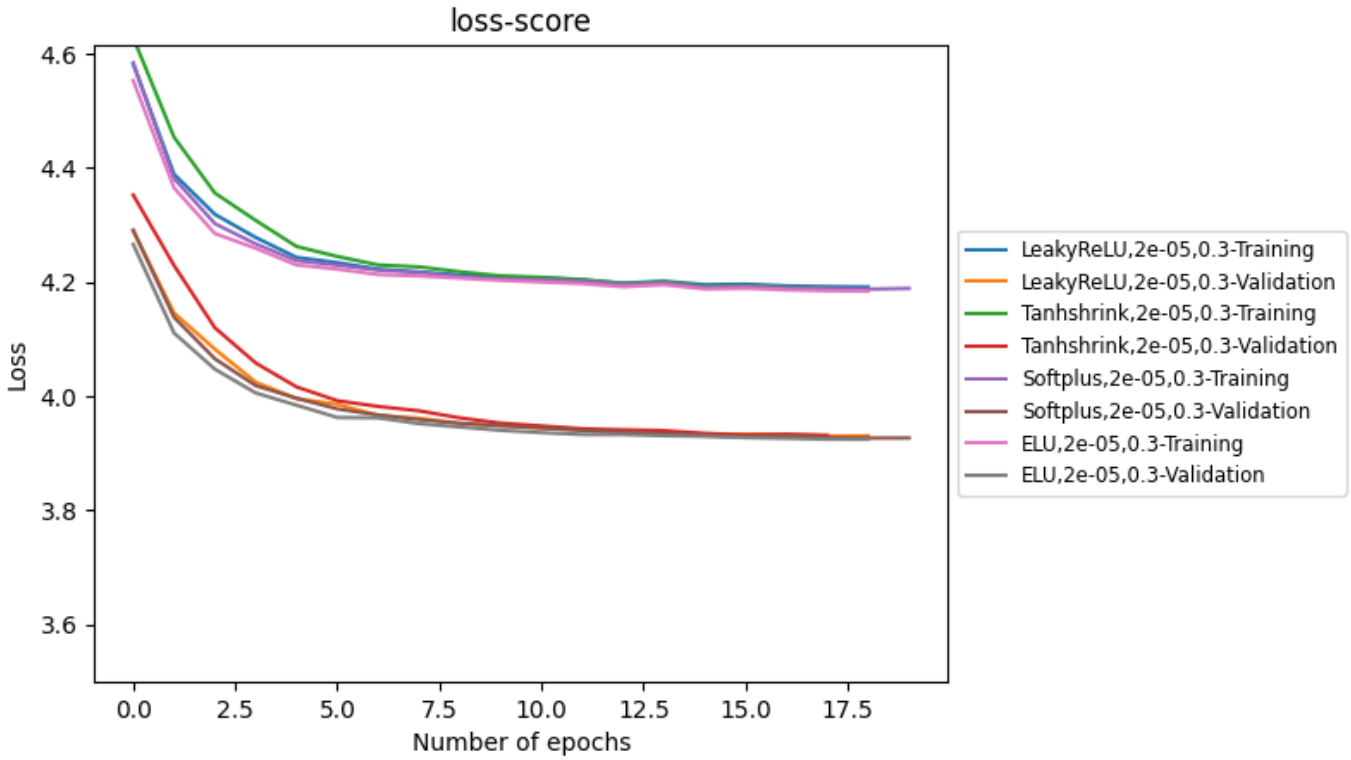

Figure 6.5: Learning curve between relation training and validation loss for the SimpleQuestions dataset which contains all questions (answerable and non-answerable over wikidata). This learning curve is the same for the methodology of creating entity spans as lists consisting of 0 and 1 or only of 0, for the methodology of finding entity span start and end indexes through either spacy library or Sentence Transformers library combined with cosine similarity metric and for the methodology of finding entity span start and end indexes through through either spacy library or by using jaccard similarity metric (Described in chapter 1.4).

tions subset that includes only the questions that are answerable over wikidata (table 6.2) [14,16,19,35,37] with the activation function Softplus [95].

Also the learning curves between the training losses and the validation losses for all the methodologies both for relation prediction and entity span prediction are very close to each other, i.e these curves converge a lot, as can be seen from figures 6.1-6.12. This fact and in combination with the very good evaluation results for relation index prediction and entity span indexes (start, end) prediction scores (f1,precision,recall,accuracy) [55], shows that the training and the validation of each relation prediction and entity span prediction model is made with great success.

For all the other methodologies of finding entity span start and end indexes(entity spans as a sequence of 0 and 1, regardless if their corresponding entity label is included in their wording, spacy library [41-49,60], Sentence Transformers combined with cosine similarity metric [42,43,56-59] and the methodology that uses only jaccard similarity metric[44,45,56-59]), the relation prediction scores are the same with each other for each SimpleQuestions dataset [13,15,19,34,36] which contains all questions and respectively are equal with each other SimpleQuestions subset containing only the answerable questions over wikidata [14,16,19,35,37]), but different between these two different datasets, because the process in order to find the relation index through the creation of a vocabulary with all unique relation ids from training and validation questions and make predictions for them through relation neural network (chapter 1.4) [41-54] is completely independent from the methodologies of finding entity span indexes (start and end) [42-47, 50,51]. In other words the

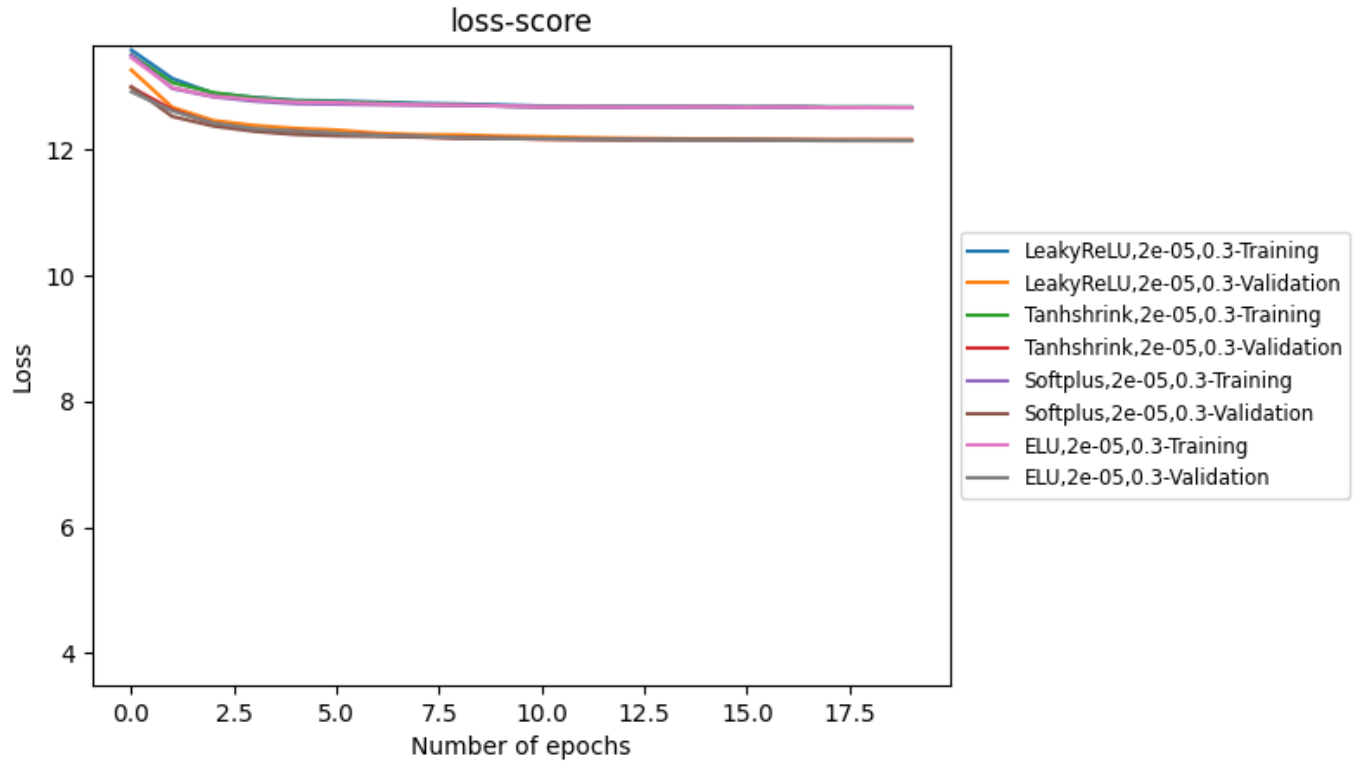

Figure 6.6: Learning curve between entity span training and validation loss- methodology of finding entity span indexes (start and end) through entity span, which is a list consisting of 0 and 1 for the SimpleQuestions dataset which includes all questions (answerable and non-answerable over wikidata.

relation index predictions are not affected at all from any methodology applied for entity span prediction.

All these other methodologies have very good results but a little bit lower compared to the methodology of creation of entity spans as sequences of 0 and 1 and specifically only for the questions that have their entity label as a part of their wording, because the methodology of finding entity span indexes with spacy library mixed either with the jaccard similarity metric [44,45,58-60] or with the use of Sentence Transformers library combined with cosine similarity metric [42,43,56-60] in a question tries to find not only the equality between entity label and a question part with equal length with it through spacy library [41-49,60], but, if the equality is not achievable, i tried to find similar question parts through spacy library mixed either with jaccard similarity metric [44,45,58-60] or with the combination of Sentence Transformers library with cosine similarity metric [42,43,56-60].

The finding of similar question parts with the corresponding entity label is a work that it is not only focused in the equality but also on the similarity between an entity label and a question part. So for a percentage of the questions have finally entity span indexes which are corresponding to question phrases that are lexically (most of times) or semantically (a few times) similar to their entity labels, where this similarity in several cases is not so great and in fact it could be quite small through cosine or jaccard similarity metric [42-45,58,59]. Finally this could lead to a small decrease on the entity span neural network performance for this methodology and as a sequence on the entity span prediction scores compared to the methodology of entity spans to be sequences of 0 and 1 only for the questions that have their entity label as a part of their wording. Thus the predictions could lose a bit of their credibility. Thus cosine similarity metric in general leads to better model parameters,

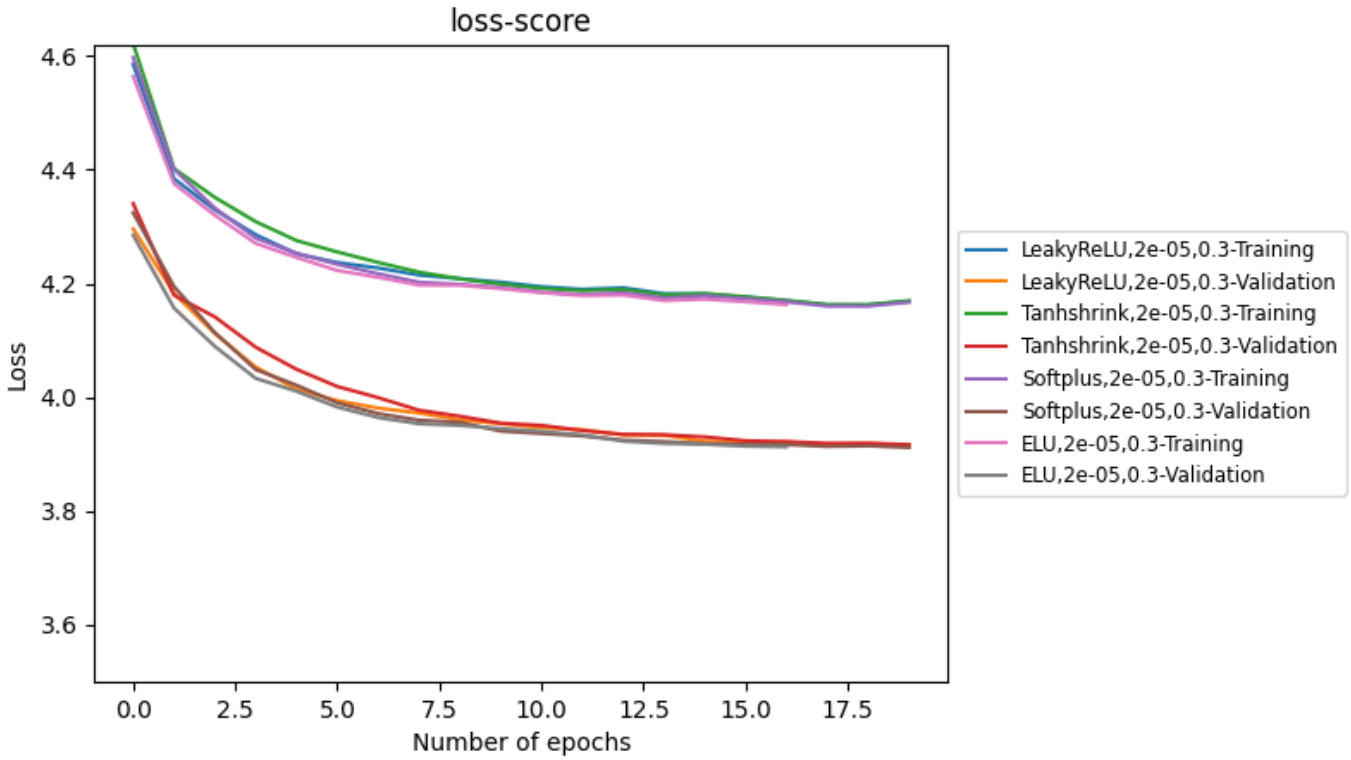

Figure 6.7: Learning curve between relation training and validation loss for the SimpleQuestions dataset which includes only questions which are answerable over wikidata. This learning curve is the same for the methodology of creating entity spans as lists consisting of 0 and 1 or only of 0, for the methodology of finding entity span start and end indexes through either spacy library or Sentence Transformers library combined with cosine similarity metric and for the methodology of finding entity span start and end indexes through through either spacy library or by using jaccard similarity metric (Described in chapter 1.4).

so and evaluation results than the jaccard similarity metric for this thesis [42-45,58,59].

Also about the methodology of creating entity spans as a sequence of 0 and 1 for SimpleQuestions dataset [13-16, 34-37], regardless if the corresponding entity label of each question is included in their wording, as a methodology has as aim to find the entity label itself, not something similar with it in the question. Based on the previous paragraph that i want to find the equality or the similarity of the entity label with an question part with the same length as its own, now there are the same amount questions that their entity label is not part of their question wording, so their entity spans are full of zeros.

This is something that is expected to cause some decrease on the entity span neural network performance for this methodology compared with the same methodology that is applied only for questions that their entity label is a part of their wording and as a result on the entity span prediction scores both for the SimpleQuestions dataset containing all questions [13,15,34,36] and the corresponding SimpleQuestions dataset containing only answerable questions over wikidata [14,16,19,35,37].

It is worth mentioning that through tables 6.1-6.4 the objective of achieving the maximum score for all the evaluation metrics(F1, recall, precision and accuracy [55]) for the relation and entity span neural networks, for the methodology of finding the entity label start and entity label end index by creating a sequence of 0 and 1 (entity span encoded), where 1 if the word that is included in the entity label, is also included in a question part equal with the entity label, if it exists, and 0 if not [1,6,46,47,50,51] for each SimpleQuestions dataset question [13-16, 34-37] is accomplished (Section 1.3-Aim and Objectives).

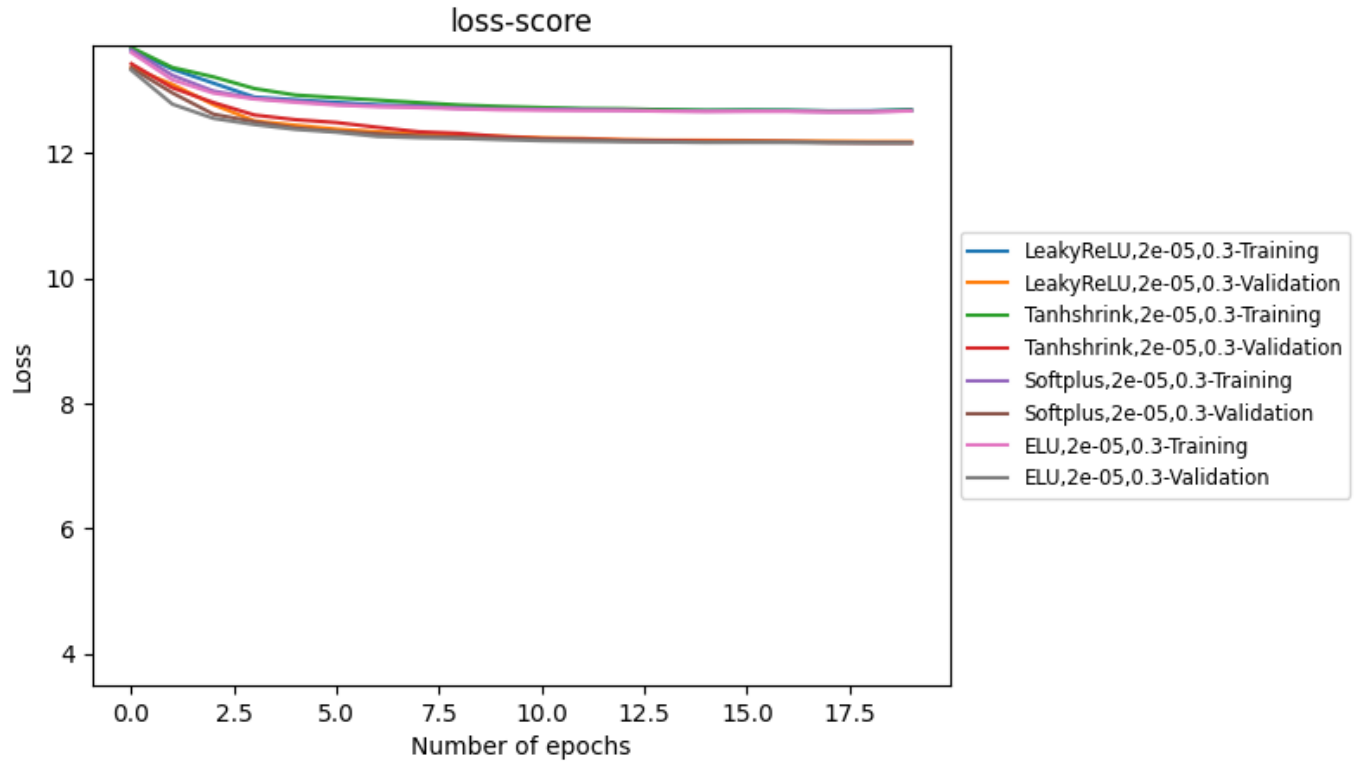

Figure 6.8: Learning curve between entity span training and validation loss -methodology of finding entity span indexes (start and end) through entity span, which is a list consisting of 0 and 1 for the SimpleQuestions dataset which includes only questions which are answerable over wikidata.

Also from these tables it is proved that the performance of the neural networks for relation indexes and entity label start and end indexes [40-54], increases if i train and validate only SimpleQuestions dataset questions [13-16, 34-37] that their entity label are part of them [1,6,46,47,50,51] (Section 1.3-Aim and Objectives).

Additionally through tables 6.5-6.8 the objective of achieving the maximum score for all the evaluation metrics(F1, recall, precision and accuracy [55]) for the relation and entity span neural networks, for the methodology of finding the entity label start and entity label end index using the spacy library [41-49,60] text or the combination of similarity model of Sentence Transformers [56,57] either with cosine similarity metric [42,43,58,59] or with the jaccard similarity metric [44,45,58,59] is accomplished (Section 1.3-Aim and Objectives).

6.3 Question Answering results

All the above best evaluation results, i.e the best parameters that lead the neural network to achieve these results, from each one of the methodologies that i implemented, help me in order to investigate how many questions can the question answering engines that i created (Section 5 - The Sentence Transformer Question Answering Model) and the degree of reliability for these answers. More specifically for all the methodologies for the question answering engines i have created, the percentage (%) of the answered questions in an indicative number of 60 questions that i insert in a chatbox is shown below.

These 60 questions i have inserted them in a .txt file named ”Questions_testing_with_ answers” [100] and i read them as a .csv file consisting of the question, relation id, entity

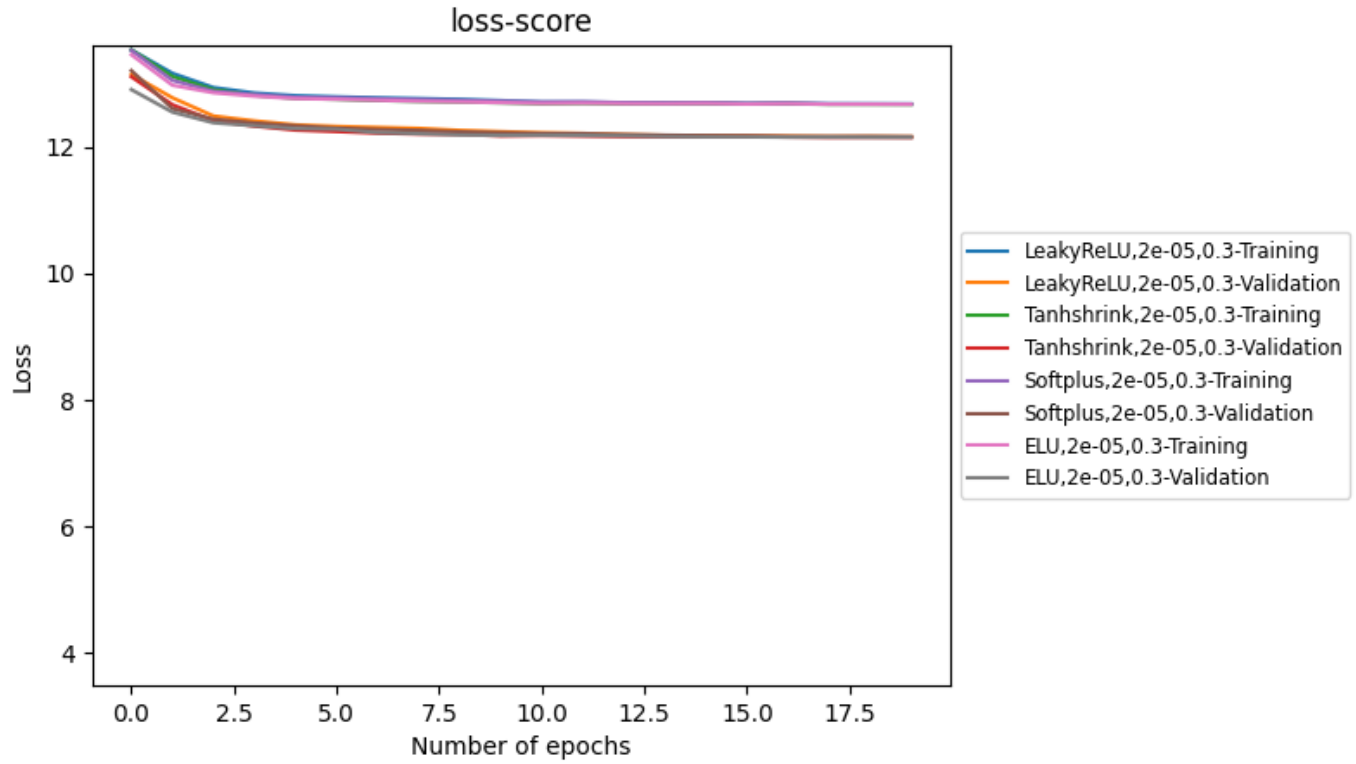

Figure 6.9: Learning curve between entity span training and validation loss-methodology of finding entity span indexes either with spacy library or Sentence Transformers + cosine similarity metric for the SimpleQuestions dataset (training and validation dataset) that includes all the questions (answerable, non-answerable over wikidata).

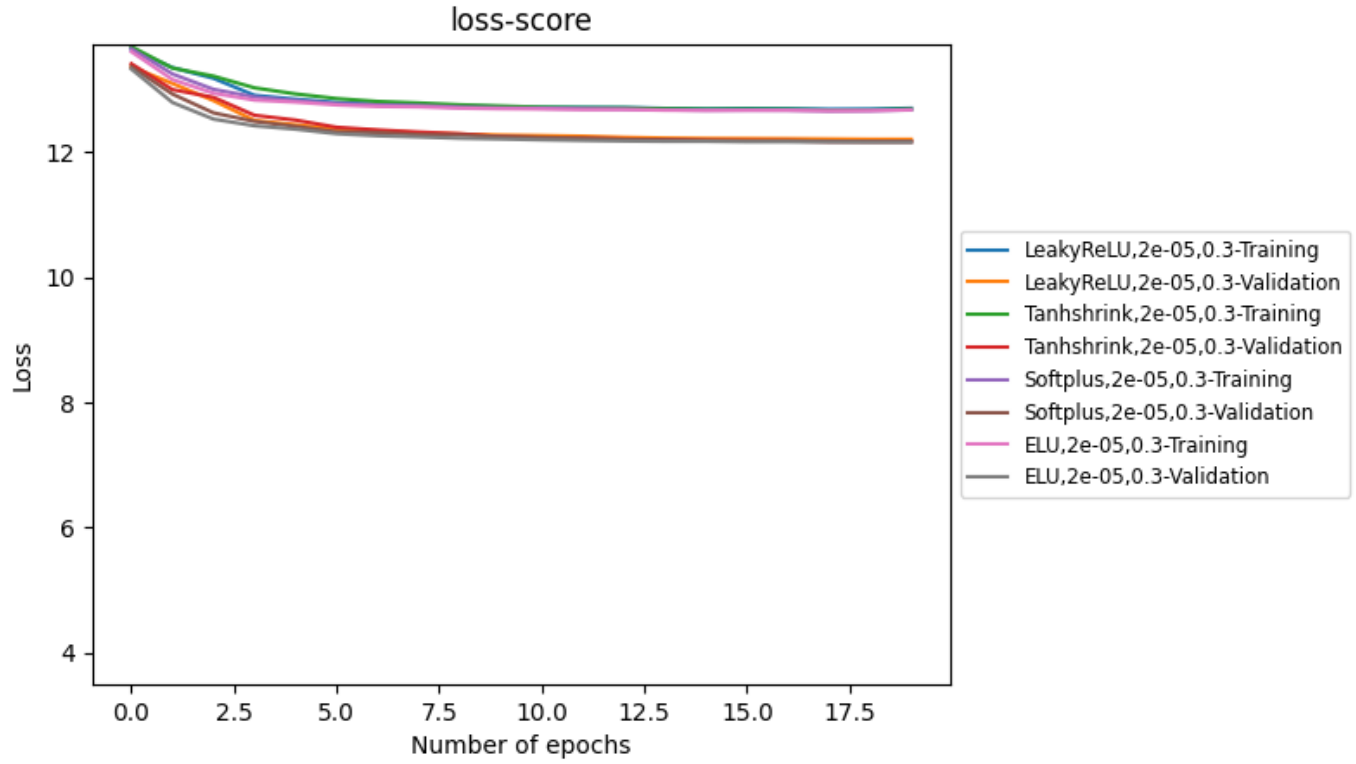

Figure 6.10: Learning curve between entity span training and validation loss-methodology of finding entity span indexes either with spacy library or Sentence Transformers + cosine similarity metric for the SimpleQuestions dataset (training and validation dataset) that includes only questions that are answerable over wikidata.

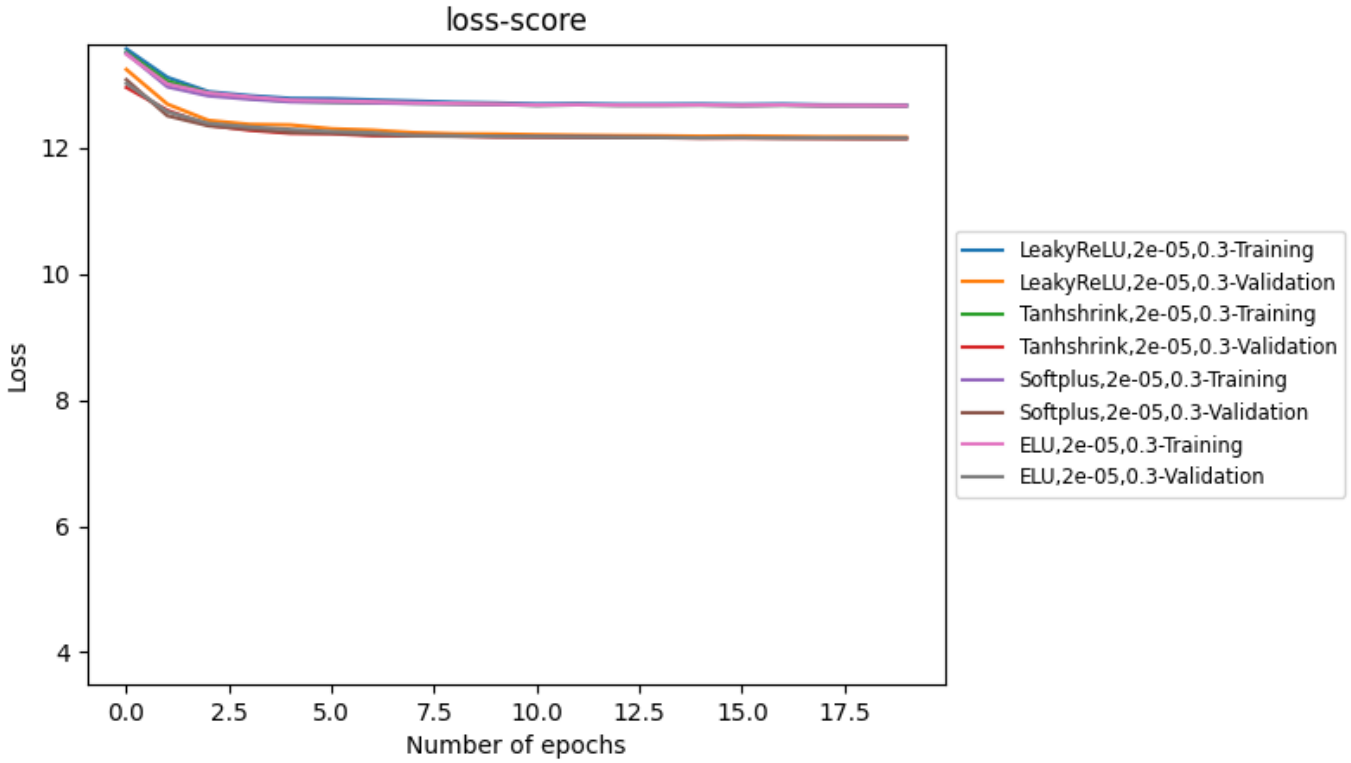

Figure 6.11: Learning curve between entity span training and validation loss-methodology of finding entity span indexes either with spacy library or jaccard similarity metric for the SimpleQuestions dataset which contains all questions (answerable, non-answerable over wikidata).

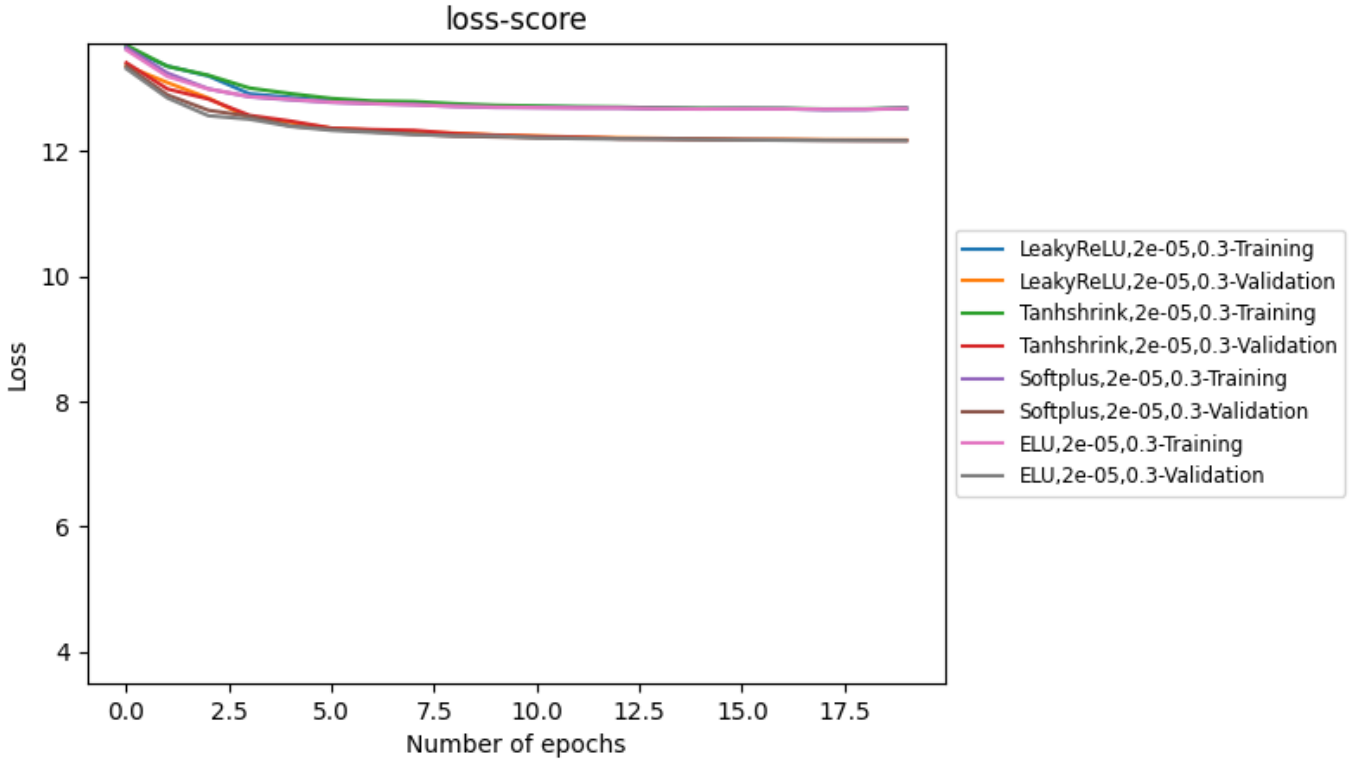

Figure 6.12: Learning curve between entity span training and validation loss-methodology of finding entity span indexes either with spacy library or jaccard similarity metric for the SimpleQuestions dataset which contains questions that are answerable over wikidata.

Evaluation Metrics	Relation index	Entity span start index	Entity span end index
F1-score	0.96058	0.92688	0.92872
Recall-score	0.96058	0.92688	0.92872
Precision-score	0.96058	0.92688	0.92872
Accuracy-score	0.96058	0.92688	0.92872

Table 6.5: Best evaluation results about relation index, entity span start and entity span end index for the methodology of finding entity span indexes (start and end), by using sentence transformers with cosine similarity metric, or else through difflib.get_close_matches() for the SimpleQuestions dataset which contains all (training, validation, test) the questions (answerable, non-answerable over wikidata.

Evaluation Metrics	Relation index	Entity span start index	Entity span end index
F1-score	0.94979	0.93083	0.92591
Recall-score	0.94979	0.93083	0.92591
Precision-score	0.94979	0.93083	0.92591
Accuracy-score	0.94979	0.93083	0.92591

Table 6.6: Best evaluation results about relation index, entity span start and entity span end index for the methodology of finding entity span indexes (start and end), sentence transformers with cosine similarity metric, or else through difflib.get_close_matches() for the SimpleQuestions dataset (training and validation dataset) that includes only questions that are answerable over wikidata.

id, answer id and answer label.

For the tables below: QA Engine 1: Question answering engine by using spacy library [41-49,60]

QA Engine 2: Question answering engine by using Sentence Transformers combined with cosine similarity metric [52,58,59]

QA Engine 3: Question answering engine by using jaccard similarity metric [53,58,59]

Also in order to understand better how the above QA engines work, i assume that a question is answered correctly if the QA engine finds the correct answer that exists in the .txt file ”Questions_testing_with_answers” [100] for each question and answers that have the highest possible simialrity score with this answer [58,59]. So i assume these questions as correctly answered questions.

Additionally i assume if for a question a QA engine finds answers that are different and even a little bit from the answer of this question included in the ”Questions_testing_with _answers” .txt file [100]. The similarity between the answer of the .txt file and an answer that is found through a QA engine for these questions is lower than the corresponding similarity between the answer of the .txt file and an answer that is calcuated for the correctly answered questions as i mentioned earlier.

Methodology 1: Creation of list consisting exclusively of 0 and 1 (entity span) for questions that their entity label, which is found from SPARQL [2,3,4,5], is a part of their wording [50,51].

Methodology 2: Creation of list consisting exclusively of 0 and 1 or only of 0 (entity span) for all questions regardless if their entity label, which is found from SPARQL, is a part of their wording [46,47].

Methodology 3: Finding entity span start and entity span end index by using spacy library

Evaluation Metrics	Relation index	Entity span start index	Entity span end index
F1-score	0.96058	0.92443	0.92729
Recall-score	0.96058	0.92443	0.92729
Precision-score	0.96058	0.92443	0.92729
Accuracy-score	0.96058	0.92443	0.92729

Table 6.7: Best evaluation results about relation index, entity span start and entity span end index for the methodology of finding entity span indexes (start and end), by using jaccard similarity metric, or else through difflib.get_close_matches() for the SimpleQuestions dataset (training, validation, test) which comprises all questions (either answerable over wikidata or not).

Evaluation Metrics	Relation index	Entity span start index	Entity span end index
F1-score	0.94979	0.92907	0.92205
Recall-score	0.94979	0.92907	0.92205
Precision-score	0.94979	0.92907	0.92205
Accuracy-score	0.94979	0.92907	0.92205

Table 6.8: Best evaluation results about relation index, entity span start and entity span end index for the methodology of finding entity span indexes (start and end), by using jaccard similarity metric, or else through difflib.get_close_matches() for the SimpleQuestions dataset (training, validation, test) which comprises questions that are answerable over wikidata.

or Sentence Transformers library combined with the cosine similarity metric [42,43,56-60].

Methodology 4: Finding entity span start and entity span end index by using spacy library [25,26] or jaccard similarity metric [44,45,58-60].

Correctly Answered questions(%)	QA engine 1	QA engine 2	QA engine 3
Methodology 1	68.33	65	50
Methodology 2	70	66.67	53.33
Methodology 3	70	65	53.33
Methodology 4	68.33	68.33	53.33

Table 6.9: Percentage (%) of correctly answered questions (Questions that each QA engine finds the answer from the "Questions_testing_with_answers" .txt file and some quite similar answers) for each one of the Question Answering engines and methodlogies that are implemented for the SimpleQuestions dataset (training, validation, test) which comprises all questions (answerable, non answerable over wikidata).

Generally for all the methodologies the percentage (%) of totally correct answered questions is ranging from 48.33% to 70%. Additionally the percentage (%) of questions that are neither totally correct nor totally incorrect answered is ranging from 8.33% to 33.33%.

From all the previous evaluation scores (f1, precision, recall, accuracy [20]-tables 6.1-6.8), it is easy to understand that both in case of answerable SimpleQuestions dataset questions over wikidata [14,16,19,35,37] and in case of all SimpleQuestions dataset questions [13,15,34,36], the combination of spacy library with the Sentence Transformers library either with cosine similarity metric or jaccard similarity metric [41-45,56-60], leads to very good neural network either relation or entity span performance [40-54] and as a result to very good evaluation results [55].

Nevertheless, from these evaluation results it cannot be made clear the impact of the spacy and Sentence Transformers libraries [42,43,56-60] and respectively the impact of the cosine and jaccard similarity metric [42-47, 58,59] separately. So in order to investigate this effect for each one of these libraries and similarity metrics i create the three

Answered questions(%)	QA engine 1	QA engine 2	QA engine 3
Methodology 1	15	16.67	30
Methodology 2	15	20	26.67
Methodology 3	15	20	28.33
Methodology 4	16.67	18.33	28.33

Table 6.10: Percentage (%) of answered questions (Questions that each QA engine finds answers that at least most of them have either (very) small or no similarity with the answer from the "Questions_testing_with_answers" .txt file. One of these answers could maybe be the answer of the .txt file itself) for each one of the Question Answering engines and methodlogies that are implemented for the SimpleQuestions dataset (training, validation, test) which comprises all questions (answerable, non answerable over wikidata).

Correctly Answered questions(%)	QA engine 1	QA engine 2	QA engine 3
Methodology 1	65	61.67	48.33
Methodology 2	66.67	65	50
Methodology 3	65	66.67	50
Methodology 4	68.33	68.33	51.67

Table 6.11: Percentage (%) of correctly answered questions (Questions that each QA engine finds the answer from the "Questions_testing_with_answers" .txt file and some quite similar answers) for each one of the Question Answering engines and methodlogies that are implemented for the SimpleQuestions datasets (training, validation, test) which contain only answerable over wikidata questions.

types of answering engines that are described in Section 5-The Question Answering Model [48,52,53].

From these percentages (%) of answered questions (tables 6.9-6.12) for the 60 questions of "Questions_testing_with_answers" .txt file [100], i observe that for all the methodologies either for all or for answerable questions over wikidata [13-17, 34-37] the best percentage of totally correct answered questions, which is equal to 70%, is achieved by the QA engine that includes spacy library [48] for the best parameters of the methodology including Sentence Transformers similarity model and cosine similarity metric and for the best parameters of the methodology of creating entity spans consisting of 0 and 1, regardless if an entity label of a question consists part of its wording [42,46] for the full SimpleQuestions dataset [13-16,34-37]. The most remarkable is that for this multitude of questions for these two aforementioned methodologies, the percentages of totally correct answered questions is exactly the same.

It is worth mentioning that from the tables 6.9-6.12 it is proved that the best parameters of the neural networks(relation, entity span) by using BERT transformer model [1,6,7,20-26] by using BERT transformer model [1,6,7,20-26] give good predictions which could lead to generally reliable answers both to other questions that are either different from the questions of the test subset of the SimpleQuestions dataset [17,18,38,39] or not. So another objective is accomplished (Section 1.3-Aim and Objectives).

This observation is strengthen more by the percentages of questions that are neither answered totally correct nor answered totally incorrect that is achieved for these two aforementioned methodologies, which is equal to 15% for both of these methodologies.

The QA engine including only spacy library [48] for all the methodologies achieves very good percentages(%) of totally correct answered questions, which ranges from 65% to 70% and also not insignificant, but much more lower than percentages(%) of totally correct

Answered questions(%)	QA engine 1	QA engine 2	QA engine 3
Methodology 1	8.33	11.67	23.33
Methodology 2	18.33	23.33	31.67
Methodology 3	18.33	21.67	33.33
Methodology 4	16.67	18.33	26.67

Table 6.12: Percentage (%) of answered questions (Questions that each QA engine finds answers that at least most of them have either (very) small or no similarity with the answer from the "Questions_testing_with_answers" .txt file. One of these answers could maybe be the answer of the .txt file itself) for each one of the Question Answering engines and methodlogies that are implemented for the SimpleQuestions datasets (training, validation, test) which contain only answerable over wikidata questions.

answered questions, percentages of only answered questions with answers neither totally correct nor totally incorrect, which ranges from 8.33% to 18.33%.

The QA engine including the Sentence Transformer similarity model combined with the cosine similarity metric [52] for all the methodologies attain very good percentages(%) of totally correct answered questions, which ranges from 61.67% to 68.33%, which consists lower percentages than the percentages(%) that QA engine including only spacy library [48] performs. Also the percentages(%) of answered questions with answers neither totally correct nor totally incorrect for this QA engine, which ranges from 11.67% to 23.33%, are a little bit higher than the corresponding percentages(%) that QA engine including only spacy library [48] performs.

From all these percentages is also shown that the jaccard similarity metric [44,45,53,58,59] is not the best similarity metric compared with cosine similarity metric [42,43,52,58,59] and also compared spacy library 41-49,60]. The percentages(%) of totally correct answered questions which ranges from 48.33% to 53.33% are not very high and also the percentages(%) of answered questions with answers neither totally correct nor totally incorrect for this QA engine, which ranges from 23.33% to 33.33% are the highest that are observed across the three QA engines [48,52,53].

Essentially the QA engine that includes spacy library [48],with a slight difference from the combination of Sentence Transformer similarity model with the cosine similarity metric [52], is the best QA engine model because the difference between the ranges of the percentages(%) of totally correct answered questions and the ranges of the percentages(%) of answered questions with answers neither totally correct nor totally incorrect are more farer than all the other QA engines. This shows that spacy library [41-49,60], is able not only to find the correct (real) answer to the question with success and also answers that have an common property (described through relation id) with the correct answer, for example the birthplace, the type of film etc. Also these percentages(%) shows that answers with an common property with the real answer are found from spacy library [41-49,60] more frequently than answers that are neither totally right nor totally wrong, but even a little close lexically (or semantically) to the real answer.

In order to explain the important difference in the performance of the QA engine models that include cosine and jaccard similarity metric [52,53], consider the sentences below: "The bottle is empty", "There is nothing in the bottle", "The bottle is empty empty", "David Beckham was playing at Manchester United", "Rolling Stones is a very-well known group playing rock music".

More specifically for the sentences "The bottle is empty" and "There is nothing in the bottle" by using the cosine similarity score function [26,27] that is described in Section

1.4-Methodology and in Section 5-The Question Answering model with the help of the Sentence Transformer similarity model [42-45,52], the cosine similarity score between these two sentences is equal to 0.66. Also by using the jaccard similarity score function [44,45,53,58,59] that is described in Section 1.4-Methodology and in Section 5-The Question Answering model, the jaccard similarity score [44,45,53,58,59] between these two sentences is equal to 0.43 [116].

As before for the sentences ”There is nothing in the bottle” and ”The bottle is empty empty”, the cosine similarity metric is equal to 0.64, while the jaccard similarity remains equal to 0.43. Also for the sentences ”David Beckham was playing at Manchester United” and ”Rolling Stones is a very-well known group playing rock music”, the cosine similarity metric is equal to 0.19, while the jaccard similarity remains equal to 0.06 [116].

Thus from all the above examples it is shown that in general cosine similarity metric [42,43,52,58,59] gives much better similarity scores than the jaccard similarity metric [44,45, 53,58,59]. So cosine similarity metric [42,43,52,58,59] could find better more similar answer to the real answer of an question and works better with text data and with text embeddings than jaccard similarity metric [44,45,53,58,59]. Generally jaccard similarity metric is rarely used with text data and it does not work with text embeddings. This is the reason why in many questions testing for each methodology in the QA engine including only jaccard similarity metric [53], the similarity score between an answer from the QA engine and the real answer for each question is equal to 0.

Essentially cosine similarity metric is not affected by the magnitude/length of the feature vectors, in this case the lengths of the questions and jaccard similarity metric takes into account only the set of unique words for each text document [58]. The difference between the first ”The bottle is empty” and the third sentence ”The bottle is empty empty” is the word ”empty” that exists twice in the third sentence. Based on what has been referred in this paragraph it is worth noting that jaccard similarity metric score [44,45,53,58,59], for both comparing the sentence ”The bottle is empty” with the sentence ”There is nothing in the bottle” and comparing the sentence ”The bottle is empty” with the sentence ”There is nothing in the bottle” remains unchanged, while the cosine similarity metric scores although they are higher would drop a little bit and this means that cosine similarity metric [42,43,52,58,59] is less sensitive to a difference in lengths [58].

Previously i observe that for the methodology of entity spans to be lists consisting of 0 and 1, in the case that i train and validate only the questions that their entity label, which is found from SPARQL queries [2,3,4,5,29,30], is part of their wording, the evaluation scores (f1,accuracy,recall,precision [55]), are a little bit better than the case of the same methodology and when i train and validate all the SimpleQuestions dataset questions [13-16, 34-37], regardless if their entity label is part of their wording.

Instead from all these percentages (%) of totally correct answered questions and questions answered neither entirely correct nor entirely incorrect(tables 6.9-6.12) it is shown that when i train and validate all the SimpleQuestions dataset questions, regardless if their entity label is part of their wording [13-16,34-37], these percentages are generally higher than the case of train and validate only the questions that their entity label is part of their wording for the same methodology.

Finally it is occurred that when i train and validate the SimpleQuestions dataset (training, validation datasets) that contain only answerable over Wikidata questions [14,16,19,35,37] the percentages (%) of totally correct answered questions are lower and the percentages (%) of questions that are answered neither totally correct nor totally incorrect are generally higher than in the case that i train and validate the full SimpleQuestions dataset (training,

validation datasets) [13,15,34,36].

All these mentioned in the last two paragraphs could be explained by the fact that when i train and validate the full SimpleQuestions dataset [13,15,34,36], although there are questions in the dataset that could not be answered through Wikidata [19], the bigger length of the dataset helps the neural network models to cope with other types of questions (with no answer over Wikidata or generally a knowledge graph for example) and thus makes the QA engines [48,52,53] to give answers to questions different , but maybe even slightly identical from the SimpleQuestions dataset (especially test dataset) [13-18,34-39]. Thus through SimpleQuestions dataset that includes only answerable questions over Wikidata [14,16,35,37], due to its lower length, all the aforementioned are impracticable to be carried out.

In addition all these show that the number of the 60 questions is a very indicative number in order to check if the QA engine models work well [100].

Based on the correct predictions or not relation id and entity id are predicted correctly and thus on the answers that are given for each question from QA engine, there are observed these types of questions:

1. Questions that are totally correct answered, with the entity id and the relation id are predicted correctly. An example of these questions is the question ”which company produced hot enough for june” which has answer ”Rank Organization”. This answer is the real and is proved to be the correct predicted for all the QA engines and all the methodologies. The real (and the correct predicted) relation ids and entity ids are P272 and Q12124859 correspondingly. These entity id and the relation id predictions for this question are achieved either from the QA engine with spacy library or the QA engine with cosine similarity metric or the QA engine with jaccard similarity metric [48,52,53] for all the methodologies [42-47,50,51].

   This shows that it is very important when a user puts a question to the QA engine box, the wording of the question should be as simple as possible and have as little and concise information as possible. When this happens it is more possible to have entirely correct answers.

2. Questions that are totally correct answered, with one of the entity id and relation id to be predicted correctly. An example of these questions is the question ”which was the country of citizehship of christopher robinson” ,which has answer ”Canada”, for the QA engine that include jaccard similarity metric [53] through the best parameters of the methodology of finding entity span indexes (start, end) through cosine similarity metric [42] for the activation function ELU [95] in the case of training and validating the full training and validation SimpleQuestions training and validation dataset [13-16, 34-37]. The real relation ids and entity ids are P27 and Q2966640 correspondingly. The real answer ”Canada” and the relation id P27 are predicted correctly . The prediction of this entity id from the aformentioned QA engine and methodology is not the correct one compared with the real entity id Q2966640, because this prediction gives Q370264 as entity id [42,53].

3. Questions that have no answers, regardless if even one of the entity id and relation id were predicted correctly. An example is the question ”who is the composer of the song liar?” which for all the QA engines and all the methodologies has no answer. The real entity id, relation id and answer label are Q1049271,P86,”Freddie Mercury”.

   More specifically for the QA engine including the Sentence Transformer similarity

model and the cosine similarity metric [52] through the best parameters of the methodology of finding entity span indexes (start, end) where the entity spans are lists consisting of 0 and 1 for all questions (regardless if the question has its entity label as part of its wording) for the activation function ELU [95] in the case of training and validating the training and validation SimpleQuestions training and validation subset only with answerable over wikidata questions [14,16,35,37], gives an incorrect prediction for the entity id: Q124707535 [46,52]. Also the same QA engine through the aforementioned model parameters predict correct the relation id:P86 [47,52]. The combination of this correctly predicted relation id and of the incorrectly predicted entity id gives no answers from the corresponding SPARQL query [2,3,4,5,29,30,48,52,53].

4. Questions that have answers that are assumed as totally wrong, because these answers have either small or no similarity with the real answer of their question, with none of the entity id and relation id to be predicted correctly.

   An example is the question ”what alternative rock band from Chicago is the author of of the blue colour of the sky” where for the QA engine including the Sentence Transformer similarity model and the cosine similarity metric through the best parameters of the methodology of finding entity span indexes (start, end) where the entity spans are lists consisting of 0 and 1 for all questions (regardless if the question has its entity label as part of its wording) for the activation function Softplus [95] in the case of training and validating the full training and validation SimpleQuestions training and validation dataset [13] the answers that are given from the corresponding SPARQL query [2,3,4,5,29,30,48,52,53] (for example Linkin Park with cosine similairty score metric equal to 0.21), are assumed as wrong, because the entity id and the relation id are predicted incorrectly (Q11366 instead of Q2071360 and R136 instead of P175) and as a sequence the predicted entity label is not the expected (alternative rock instead of alternative rock band from Chicago) [47,52].

5. Questions that have answers that are assumed as neither totally correct not totally incorrect, where these answers have either significant or small similarity with the real answer of their question, regardless if the real answer is a part of the predicted answers and if at least one of the entity id and relation id to be predicted correctly.

   An example is the question ”An example is the question ”Which home is an example of italianate architecture?” where for the QA engine including the Sentence Transformer similarity model and the cosine similarity metric through the best parameters of the methodology of finding entity span indexes (start, end) where the entity spans are lists consisting of 0 and 1 for questions that have their entity label, which are found from SPARQL query [2,3,4,5,29,30,48,52,53], as part of their wording for the activation function ELU [95] in the case of training and validating the full training and validation SimpleQuestions training and validation dataset [13-16,34-37] one of the predicted entity id, relation id and answer are Q615196,R149,”Fairfield House” respectively with cosine similarity score with the expected answer equal to 0.46 [51,52]. The real entity entity id, relation id and answer are Q615196,R149,John and Maria Adams House correspondingly, where this answer is not one of the predicted answers in this case. But also there are some other significant or a little similar answers with the real answer such as ”Criterion Hotel” ,with cosine similarity score with the expected answer equal to 0.2 [51,52].

   Another example for the QA engine including spacy library and the same methodology is ”Who was born in dakar?” where from the corresponding SPARQL query

[2,3,4,5,29,30,48,52,53], the answers that are occurred do not include the real answer, i.e ”Djibril Diawara”, although the predicted relation id and entity id are the same with the real ones, i.e R19 and Q3718 respectively. Another answer that is similar due to the common property of the town Dakar to be the birthplace (R19), is the answer ”Henri Saivet” [48,51].

Additionally it is worth mentioning that from the tables 6.9-6.12 it is proved that the best parameters of the neural networks(relation, entity span) by using BERT transformer model [1,6,7,20-26] give good predictions which could lead to generally reliable answers both to other questions that are either different from the questions of the test subset of the SimpleQuestions dataset [17,18,38,39] or not. So another objective is accomplished (Section 1.3-Aim and Objectives).

It is also shown that the QA engine model including spacy library [41-49,60] is preferable in order to create a question answering engine that is able to give reliable answers to each question given by the user (Section 1.3-Aim and Objectives).

An important observation is that, based on the correct predictions for entity id and relation id, it is absolutely clear when the answers given from QA engines [49,52,53] in a question are totally correct, at some point correct or totally incorrect (Section 1.3-Aim and Objectives).

Also i want to mention that when i referred to an answer or relation id or entity id of a question as real, i mean that i assumed the answers, relation ids and entity ids that are included for this question of ”Questions_testing_with_answers” .txt file [100] as real.

For all these methodologies, although the equality or the similarity of the entity label with an part of their corresponding question, the prediction of the relation ids or the entity span indexes is based on indexes and not in a phrase or sentence embedding and as a result the entity span indexes mainly are not predicted always correctly. This fact, although the results are importantly reliable, could be a further obstacle to improving results of QA engines.

It is finally occurred that due to the fact that i have a different neural network for relation prediction and different for entity span prediction, although that the prediction scores are very high as number, it is possible when the relation id to be predicted right, the entity id is not predicted right and vice versa. This is a reason that some questions either are not answered or answered not right.

7. CONCLUSIONS AND FUTURE WORK

In this thesis i create three question answering engines the one with cosine similarity metric and Sentence Transformer, one with spacy library and the other with jaccard similarity metric. I create before two neural networks one for the relation prediction and one for the entity span prediction, which give very good predictions. These predictions helps the QA engines to predict generally satisfyingly the relation ids and the entity span indexes and as a result the entity labels for all the best parameters of the neural networks for each one of the methodologies either for SimpleQuestions dataset questions that are answerable over wikidata or for all SimpleQuestions dataset questions (entity span creation as a sequence of 0 and 1 either for all questions or only for questions having their entity label as part of their wording, using of spacy library or Sentence Transformers library combined with similarity metrics (cosine, jaccard)).

More specifically the best evaluation results (scores) for entity span prediction are achieved for the methodology of finding entity span indexes (start,end) by creating lists consisting of 0 and 1 only for SimpleQuestions training and validation dataset (for both the full dataset and the corresponding dataset that inlcudes only answerable over wikidata questions) questions that their entity label consists a part of their wording.

Correspondingly for relation prediction the best evaluation results (scores) are achieved for the methodology of finding entity span indexes (start,end) by creating lists consisting of 0 and 1 only for all SimpleQuestions training and validation dataset (for both the full dataset and the corresponding dataset that inlcudes only answerable over wikidata questions) questions regardless if their entity label consists a part of their wording.

Also the biggest percentage of totally correct answered questions are observed for the QA engine with spacy library for the methodology of finding entity span indexes (start,end) through cosine similarity metric and the methodology of finding entity span indexes by creating lists consisting of 0 and 1 in the case of training and validating all SimpleQuestions training and validation dataset questions respectively, regardless if their entity label consists a part of their wording.

These results, although that are generally satisfactory, are based on indexes, i.e on numbers and not on sentence or phrase embeddings and it is proved a little bit difficult to have a much more bigger amount of correct predictions for relation indexes, but mainly for entity span indexes, than it is achieved in this thesis. This fact is a good change for the future to neural network models make predictions through phrases, sentences, embeddings of them and not through numbers(indexes in this thesis). The next step of this work is to create QA engines that predict the questions answers using such networks. This work could help QA engines to understand the semantics of an entity or a sentence much more better.

About neural networks another future work to be not only to accept as input phrase or sentence embeddings, but also to create a neural network that predict both the relation index and as a result the relation id and the entity span indexes and so the entity labels in such a way that the predictions of such networks to be very reliable for relation and entity id concurrently. The main reason for this future work is that in this thesis i create two neural networks for relation and entity span prediction respectively and in some questions in QA engine the relation id could be predicted correctly, but not the entity label and vice-versa. This is a reason for some questions the QA engine gives wrong or no answers. Essentially this future work could be an effort for simultaneous correct relation id and entity

id predictions and finally for fewer wrong or no answers.

ABBREVIATIONS - ACRONYMS

RDF	Resource Description Framework
SPARQL	SPARQL Protocol and RDF Query Language
OWL	Web Ontology Language
OGC	Open Geospatial Consortium

APPENDIX A. FIRST APPENDIX

APPENDIX B. SECOND APPENDIX

APPENDIX C. THIRD APPENDIX

BIBLIOGRAPHY

1. https://bonndoc.ulb.uni-bonn.de/xmlui/bitstream/handle /20.500.11811/9810/6670.pdf ?sequence=5isAllowed=y
2. https://en.wikipedia.org/wiki/SPARQL
3. https://medium.com/wallscope/constructing-sparql-queries-ca63b8b9ac02
4. https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial
5. https://query.wikidata.org/
6. https://arxiv.org/pdf/2001.11985.pdf
7. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
8. Howard, J., Ruder, S.: Universal language model fine-tuning for text classification. In: Pro ceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Vol ume 1: Long Papers). pp. 328–339 (2018)
9. Liu, X., He, P., Chen, W., Gao, J.: Multi-task deep neural networks for natural language understanding. arXiv preprint arXiv:1901.11504 (2019)
10. Peters, M.E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., Zettlemoyer, L.: Deep contextualized word representations. In: Proc. of NAACL (2018)
11. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I.: Improving language understanding by generative pre-training transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
12. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners. Tech. rep.
13. https://github.com/askplatypus/wikidata-simplequestions/blob /master /annotated_wd _data_train.txt
14. https://github.com/askplatypus/wikidata-simplequestions/blob/master/annotated_wd _data_train_answerable.txt
15. https://github.com/askplatypus/wikidata-simplequestions/blob /master /annotated_wd _data_valid.txt
16. https://github.com/askplatypus/wikidata-simplequestions/blob/master/annotated_wd _data_valid_answerable.txt
17. https://github.com/askplatypus/wikidata-simplequestions/blob /master /annotated_wd _data_test.txt
18. https://github.com/askplatypus/wikidata-simplequestions/blob /master /annotated_wd _data_test_answerable.txt

19. D.Vrandecic and M. Krötzsch, Wikidata: a free collaborative knowledgebase, Commun. ACM57(2014) 78, url: https://doi.org/10.1145/2629489.

20. J. Devlin et al., “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), 2019 4171

21. https://huggingface.co/docs/transformers/index

22. https://huggingface.co/transformers/v3.0.2/model_doc/bert.htmloverview

23. https://huggingface.co/google-bert/bert-base-uncased

24. https://huggingface.co/docs/transformers/model_doc/berttransformers.BertModel

25. https://huggingface.co/docs/transformers/model_doc/berttransformers.BertTokenizerFast

26. https://huggingface.co/transformers/v4.8.2/internal/tokenization_utils.htmltransformers.tokenization_utils_base.PreTrainedTokenizerBase.encode_plus

27. C. Cortes and V. Vapnik, Support-vector networks, Machine learning 20 (1995) 273.

28. T. B. Brown et al., Language models are few-shot learners, arXiv preprint arXiv:2005.14165 (2020).

29. https://sparqlwrapper.readthedocs.io/en/latest/SPARQLWrapper.Wrapper.html

30. https://people.wikimedia.org/ bearloga/notes/wdqs-python.html

31. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Dataset%20Preprocessing/Dataset_Preprocessing.ipynb

32. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Dataset%20Preprocessing/Answerable_Questions_Dataset_Preprocessing.ipynb

33. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Dataset%20Preprocessing/Test_dataset.ipynb

34. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Dataset%20Preprocessing/train_dataset.csv

35. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Dataset%20Preprocessing/train_answerable_questions_dataset.csv

36. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Dataset%20Preprocessing/valid_dataset.csv

37. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Dataset%20Preprocessing/valid_answerable_questions_dataset.csv

38. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Dataset%20Preprocessing/test_dataset.csv

39. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Dataset%20Preprocessing/test_answerable_dataset.csv

40. https://pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html

41. https://github.com/Themiscodes/Question-Answering-Transformers /blob/main/notebooks/01_dataset.ipynb

42. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Sentence%20Transformers %20%2B%20Cosine%20Similarity%20Metric%20Methodology/Thesis_Text_similarity_with_cosine_similarity_(with_SentenceTransformer_and_difflib).ipynb

43. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Sentence%20Transformers %20%2B%20Cosine%20Similarity%20Metric%20Methodology/Thesis_Text_similarity_with _cosine_similarity_for _answerable_questions_ (with_SentenceTransformer_and_difflib).ipynb

44. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Sentence%20Transformers%20%2B%20Jaccard%20Similarity%20Metric%20Methodology/Thesis_Text_similarity_with_jaccard_similarity_with_difflib.ipynb

45. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Sentence%20Transformers%20%2B%20Jaccard%20Similarity%20Metric%20Methodology/Thesis_Text_similarity_with_jaccard_similarity_with_difflib_for_answerable_questions.ipynb

46. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Thesis_Entity_span_creation_by_encoding_with_0_and_1_ (for_answerable_questions) _Not _ignoring_Questions_with_no_similarity _or_equality_with_the_entity_label.ipynb

47. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Thesis_Entity_span_creation_by_encoding_with_0_and_1_Not_ignoring_Questions_with_no_similarity_or_equality_with_the_entity_label.ipynb

48. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Question%20Answering%20Engines/QA_engine_with_difflib_for_SimpleQuestions.ipynb

49. https://github.com/Themiscodes/Question-Answering-Transformers /blob/main/notebooks/04_engine.ipynb

50. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Thesis_Entity_span_creation_by_encoding_with_0_and_1_(for_answerable_questions) _Ignoring_Questions_with_no_equality_with_the_entity_label.ipynb

51. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Thesis_Entity_span_creation_by_encoding_with_0_and_1_ Ignoring_Questions_with_no _equality_with_the_entity_label.ipynb

52. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Question%20Answering%20Engines/QA_engine_with_Sentence_Transformers _and _cosine_similarity_for_SimpleQuestions.ipynb

53. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Question%20Answering%20Engines/QA_engine _with_jaccard_similarity_for _SimpleQuestions.ipynb

54. https://github.com/Themiscodes/Question-Answering-Transformers /blob/main/notebooks/03_models.ipynb

55. https://scikit-learn.org/stable /modules/classes.htmlmodule-sklearn.metrics

56. SentenceTransformers Documentation — Sentence-Transformers documentation

57. Pretrained Models — Sentence-Transformers documentation

58. Top 7 Ways To Implement Text Similarity In Python

59. Ultimate Guide To Text Similarity With Python - NewsCatcher

60. Python | Find all close matches of input string from a list - GeeksforGeeks

61. https://www.techtarget.com/searchapparchitecture/definition/Resource-Description-Framework-RDF

62. https://www.baeldung.com/cs/rdf-intro

63. https://www.ontotext.com/knowledgehub/fundamentals/what-is-rdf/

64. P. N. Mendes et al., “DBpedia spotlight: shedding light on the web of documents,” Proceedings of the 7th international conference on semantic systems, ACM, 2011 1.

65. J. Lehmann et al., DBpedia- A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia, Semantic Web Journal 6 (2015) 167, Outstanding Paper Award (Best 2014 SWJ Paper).

66. https://cambridgesemantics.com/blog/semantic-university/intro-semantic-web/intro-linked-data/

67. https://www.ontotext.com/knowledgehub/fundamentals/linked-data-linked-open-data/

68. K.Bollacker et al., “Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge,” Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD’08, ACM, 2008 1247, isbn: 978-1-60558-102-6, url: http://doi.acm.org/10.1145/1376616.1376746.

69. https://cgi.di.uoa.gr/ koubarak/publications/2023/GeoKG___GeoAI_2023_GeoQA2-3.pdf

70. https://dl.acm.org/doi/10.1145/3281354.3281362

71. https://link.springer.com/chapter/10.1007/978-3-031-47243-5_15

72. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems. pp. 5998–6008 (2017)

73. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473 (2014)

74. https://pytorch.org/docs/stable/generated/torch.nn.Softmax.html

75. https://developers.google.com/ machine-learning/ crash-course/multi-class-neural-networks/softmax

76. https://deepai.org/machine-learning-glossary-and-terms/softmax-layer

77. https://pytorch.org/docs/stable/generated/torch.nn.Softmax.html

78. https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network

79. https://builtin.com/machine-learning/relu-activation-function

80. M.Schuster and K. Nakajima, “Japanese and korean voice search,” 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2012 5149

81. Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al.: Google’s neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144 (2016)

82. S. Mohammed et al., “Strong Baselines for Simple Question Answering over Knowledge Graphs with and without Neural Networks,” Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers), vol. 2, 2018 291.

83. M. Petrochuk and L. Zettlemoyer, SimpleQuestions Nearly Solved: A New Upperbound and Baseline Approach, arXiv preprint arXiv:1804.08798 (2018)

84. S. Hochreiter and J. Schmidhuber, Long short-term memory, Neural computation 9 (1997) 1735

85. K. Cho et al., “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation,” Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL, 2014 1724, url: http://aclweb.org/anthology/D/D14/D14-1179.pdf

86. A. Bordes et al., Large-scale simple question answering with memory networks, arXiv preprint arXiv:1506.02075 (2015)

87. He, X., Golub, D.: Character-level question answering with attention. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. pp. 1598–1607 (2016)

88. Lukovnikov, D., Fischer, A., Lehmann, J., Auer, S.: Neural network-based question answer ing over knowledge graphs on word and character level. In: Proceedings of the 26th inter national conference on World Wide Web. pp. 1211–1220. International World Wide Web Conferences Steering Committee (2017)

89. Yu, M., Yin, W., Hasan, K.S., dos Santos, C., Xiang, B., Zhou, B.: Improved neural relation detection for knowledge base question answering. In: Proceedings of the 55th Annual Meet ing of the Association for Computational Linguistics (Volume 1: Long Papers). vol. 1, pp. 571–581 (2017)

90. Dai, Z., Li, L., Xu, W.: Cfo: Conditional focused neural question answering with large scale knowledge bases,arXiv preprint arXiv:1606.01994. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). vol. 1, pp. 800–810 (2016)

91. Yin, W., Yu, M., Xiang, B., Zhou, B., Sch¨ utze, H.: Simple question answering by atten tive convolutional neural network. In: Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers. pp. 1746–1756 (2016)

92. https://ceur-ws.org/Vol-1963/paper555.pdf

93. Pellissier Tanon, T., Vrandecic, D., Schaffert, S., Steiner, T., Pintscher, L.: From Freebase to Wikidata: The great migration. In: Proc. of WWW. pp. 14191428 (2016)

94. https://pytorch.org/docs/stable/generated/torch.nn.Linear.html

95. https://pytorch.org/docs/stable/nn.functional.htmlnon-linear-activation-functions

96. https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html

97. https://pytorch.org/tutorials/beginner/data_loading_tutorial.html

98. https://pytorch.org/docs/stable/generated/torch.Generator.html

99. https://darinabal.medium.com /deep-learning-reproducible-results -using-pytorch -42034da5ad7

100. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Questions%20testing.txt

101. https://opendata.stackexchange.com/questions/9685/get-qid-from-wikidata-label-name-via-sparql

102. https://machinelearningknowledge.ai/pytorch-optimizers-complete-guide-for-beginner/

103. https://pytorch.org/docs/stable/optim.html

104. https://pytorch.org/docs/stable/generated/torch.optim.Adam.html

105. https://pythonguides.com/adam-optimizer-pytorch/

106. https://towardsdatascience.com/https-medium-com-dashingaditya-rakhecha-understanding-learning-rate-dd5da26bb6de

107. https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

108. https://neptune.ai/blog/pytorch-loss-functions

109. https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

110. https://www.datacamp.com/tutorial/the-cross-entropy-loss-function-in-machine-learning

111. https://huggingface.co/docs/transformers /main_classes/ optimizer_schedulestransformers .SchedulerType

112. https://neptune.ai/blog/how-to-choose-a-learning-rate-scheduler

113. https://pythonguides.com/pytorch-early-stopping/

114. https://debuggercafe.com/using-learning-rate-scheduler-and-early-stopping-with-pytorch/

115. https://clay-atlas.com/us/blog/2021/08/25/pytorch-en-early-stopping/

116. https://github.com/DimOriCoding/Question-Answering-Engine-with-Transformers/blob/main/Question%20Answering%20Engines /Examples_with_cosine_and_jaccard_similarity_metric.ipynb