NLP Project
Question Answering and Information Retrieval on SQuAD dataset

Domenichelli Daniele, Lavista Andrea, Rispoli Luca

June 4, 2021

Contents

Chapter 1

Summary

1.1 Objectives

The starting aim of the project was to implement a model that, given as input a question and a paragraph of text, selects from the input paragraph the exact sequence of text that is the answer to the question.
The project has then been extended with an **informational retrieval** component so that it is possible to autonomously select the relevant document and paragraph, hence requiring only the question and a set of document as input, as shown in picture 1.1.

1.2 Methodologies

The first step was a brief analysis of the **SQuAD dataset**, from which we decided to apply some pre-processing depending on the tasks we were performing.
Then we split as usual the dataset in three parts: training set (80%), validation set (10%) and test set (10%), according to the titles.
These sets were then used in both the proposed models, in order to tune the hyper-parameters and test the final evaluation. We also decided to use the same splits for each step of the pipeline because using different splits would positively affect the results of those models that were trained over a part of the test set.

1.3 General architecture

The architecture is composed by 3 main steps:

- *Pre-processing*
- *IR task*, composed by two step:
  - Document ranking
  - Paragraph ranking
- *Questions answering*

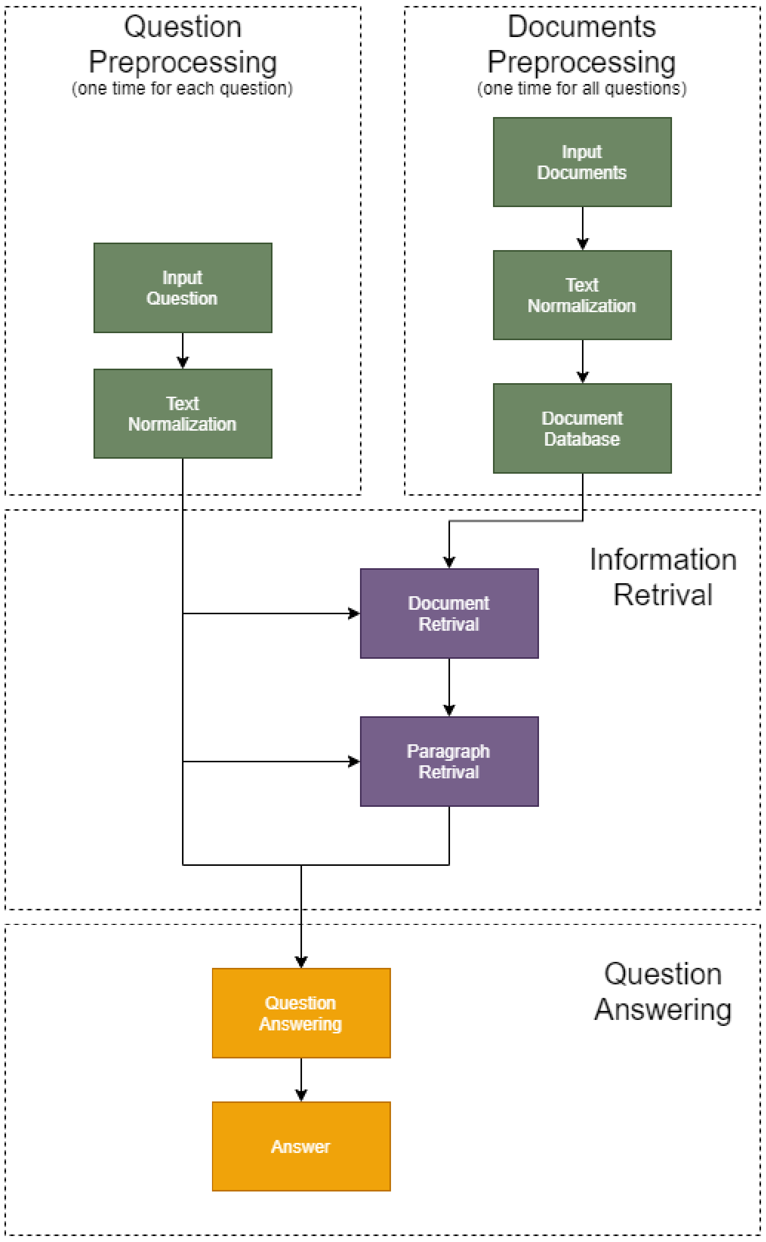

Figure 1.1: Pipeline

Chapter 2

Background

Before approaching these problems we did some research and studied existing solutions, trying to understand the most common and effective methods today used. In particular, for the question answering task we combined different features of existing solutions, which are:

- *Seq2seq-Attention Question Answering Model by W. Hou, Y. Nie* [1]: it consists in a seq2seq model, in which the encoder is composed by two bidirectonal LSTMs layers combined with the attention mechanism while the decoder is composed by two bidirectonal LSTMs and two softmax layers that perform the predictions respectively about the beginning and the end of the answer.
- *Stanford Attentive Reader*: it's composed by two bidirectional LSTM that receive respectively the questions and paragraphs embeddings; these layers are followed by a layer computing the similarity and a layer that makes the predictions. Moreover, the paragraphs' input consist not only in the embeddings of the words, but also other information are concatenated, which are: POS tag, NER tag, a flag that represents if each word is included also in the question and the output of the attention mechanism between paragraph and question.

[1]https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1174/reports/2761153.pdf

Chapter 3

System Description

3.1 SQUAD task

The first task we implemented was the question answering, on the SQuAD dataset.
After studying the papers described in the previous chapter, we developed our custom sequence to sequence architecture based on the notions we learned.
The first phase of the task consisted in extracting and classifying the dataset: we decided to organize it into tuples of (Paragraph,Question,Answer,Id,Title).
We then utilized the **spaCy** library in order to extract from the (lowerized) text several features like *POS tags*, *NER tags* and information about the token being a *stop-word* or a *number*.
The input then has been tokenized through the use of Keras' tokenizer module, and the tokens were used to generate the embedding matrix.
In order to save memory and increase performances, we decided to remove from the dataset all the paragraphs with a length greater than 400 tokens: this process has excluded about 0.25% of the 87599 tuples.
Then, each token in a given paragraph was flagged with a boolean variable stating whether or not the token was also present in the question, tuples in which the answer was not contained inside the related paragraph were discarded.
The last part of the pre-processing step was classifying the questions according to their starting word: if it is a wh-word, then they are flagged correspondingly, otherwise they are given a default token.

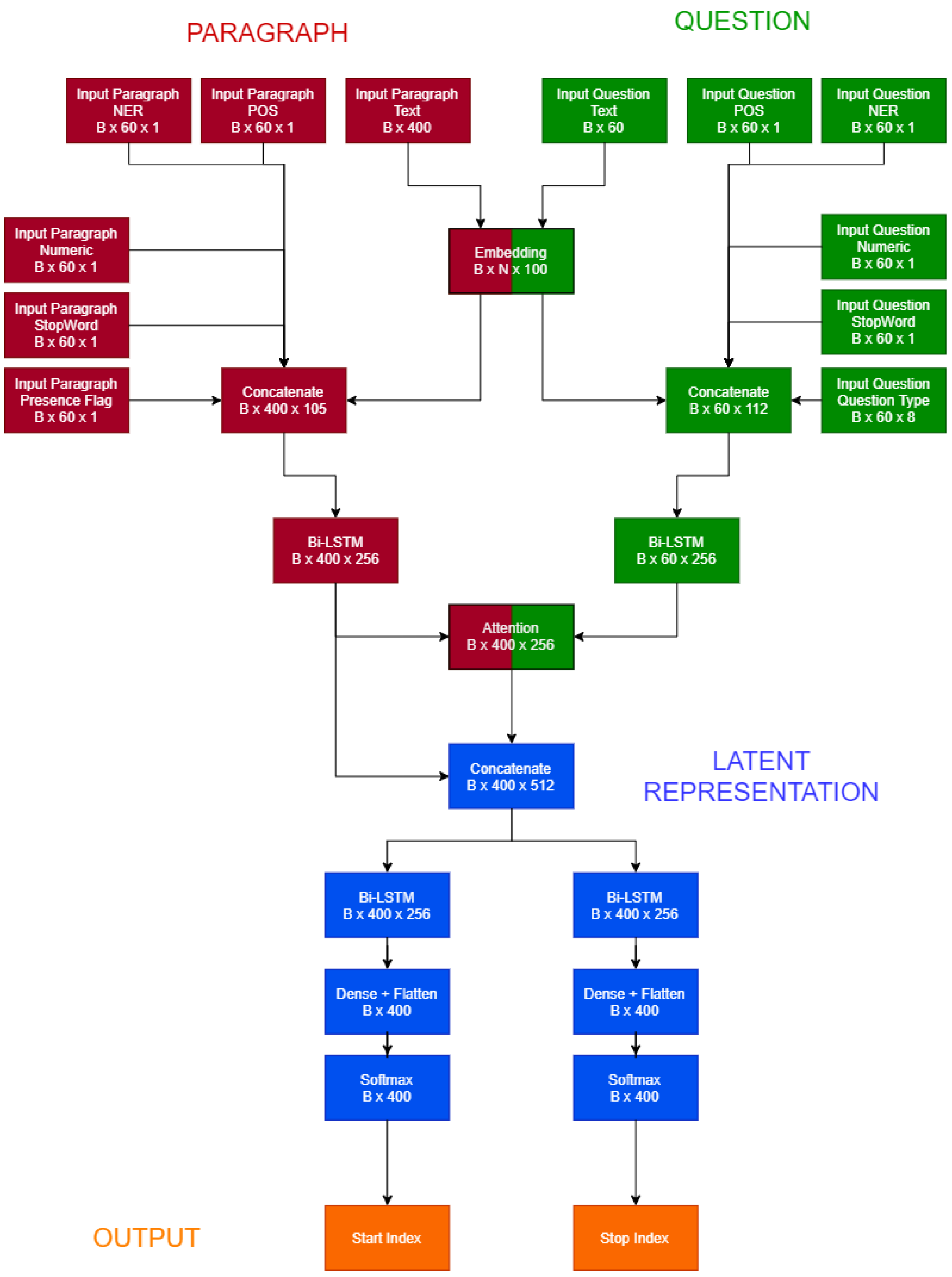
PARAGRAPH
QUESTION
Input Paragraph NER B x 60 x 1
Input Paragraph POS B x 60 x 1
Input Paragraph Text B x 400
Input Question Text B x 60
Input Question POS B x 60 x 1
Input Question NER B x 60 x 1
Input Paragraph Numeric B x 60 x 1
Input Paragraph StopWord B x 60 x 1
Input Paragraph Presence Flag B x 60 x 1
Embedding B x N x 100
Input Question Numeric B x 60 x 1
Input Question StopWord B x 60 x 1
Input Question Question Type B x 60 x 8
Concatenate B x 400 x 105
Concatenate B x 60 x 112
Bi-LSTM B x 400 x 256
Bi-LSTM B x 60 x 256
Attention B x 400 x 256
Concatenate B x 400 x 512
LATENT REPRESENTATION
Bi-LSTM B x 400 x 256
Bi-LSTM B x 400 x 256
Dense + Flatten B x 400
Dense + Flatten B x 400
Softmax B x 400
Softmax B x 400
OUTPUT
Start Index
Stop Index

Figure 3.1: Squad Task Model

As shown in the picture 3.1, this is a encoder-decoder model, in detail:

- The input question and paragraph are first encoded through an embedding layer, then the result is concatenated to the other input features like POS and NER tags.
  The concatenated tensor are then fed to the encoder of the model which is composed by two LSTM and an attention layer, which utilizes the question as query and the paragraph as key/value.

- the decoders (one for the predicting the start position of the answer, and one for the end) are composed by a bidirectional LSTM, a dense layer (to reduce the dimensionality) and at the end a softmax layer which computes the probabilities of each token to be either a starting or an ending index of the answer. We implemented the model with **Keras** library and trained it with the categorical cross entropy as loss function.

3.2 IR task

Starting from a question, the IR task's aim is to select the document and the paragraph in which the answer is contained. In this way, we decided to embed the question and each document/paragraph of the dataset into a latent vector representation, and then compare the question vector with each document/paragraph vector by using cosine similarity. By doing this, we computed a similarity score ranking in the range $[0 - 1]$, which informs us about how much each document is correlated to the answer.
In the following paragraphs we will explain in more details our implementations and the techniques we used.

3.2.1 LSA

Latent Semantic Analysis (LSA) is a technique used in NLP to represents documents in a dense vector space. LSA uses **Singular Value Decomposition** (SVD) to perform a dimensionality reduction from the term-document matrix, and it needs a parameter that indicates the dimension of the latent space.
We used the implementation of LSA provided by **Gensim** and searched for the best parameter, which is, in our case, 350.

3.2.2 Doc2Vec

Another well known model available from Gensim is **Doc2Vec**, a document embedder which allows to infer a vector from a sequence of words. At first, the model was trained over the whole training set of documents and questions, using a grid search in order to optimize the hyper-parameters. At each epoch we measured the evaluation of the model, trying to minimize the average position of the target document. We found out that after a certain amount of epochs (nearby 26 epochs), the model diverge and get worse in the score. We decided to stop the training the epoch before the model degenarates, testing the final results over the test set.

3.2.3 Implementation

Our implementation includes the two previous discussed models (LSA and Doc2Vec) used together, taking their average to generate the ranking of the documents.
Then, stated that there is a high probability that the answer contains at least

one (non-stop) word from the question, we evaluated to exploit this knowledge by adding a bonus score to those documents that shares a lexeme with the question (through exact matching).Furthermore the bonus is increased if the lexeme belongs to an entity name (has an associated NER tag). At the end the scores are normalized into the final ranking array.
So we have two bonuses:

- **EM_bonus**: associated with the presence of the question's tokens in the documents;
- **NER_EM_bonus**: associated with the presence of the question's NER tokens in the documents.

$$y_{i,j} = model_i(document_j)$$

$$Y_j = \frac{\sum_i y_{i,j}}{N} + \alpha \cdot EM\_flag_j + \beta \cdot NER\_EM\_flag_j$$

$$\hat{Y_j} = norm(Y) = \frac{Y_j - min(Y)}{max(Y) - min(Y)}$$

$$N = \text{number of models}$$

$$\alpha, \beta = \text{tuned bonus coefficients}$$

We also provide the possibility to normalize the score before the adding of the exact match bonuses, but in this way the performances were lower.

3.2.4 Evaluation Procedure

In order to understand if a model is improving or not, we decided a set of metrics. Here a brief description of each one, in order of importance:

1. *Average and Standard Deviation of Position*: the average position where the target document is placed. We are interested in extract just a set of plausible documents, so the lower is this position, the narrower can be the set. It is a proxy of the position precision of the model. We also computed the standard deviation that well represents the average worst case scenario of the model.
2. *Target doc as first result*: represents the percentage of times the target document occupies the first place of the ranking. It is an index of the model accuracy for the ranking.

3. *Target Position Quantiles and Median*: median and quantiles were used to measure how many times the target documents fall within the first X% of all the results. We used the median and the 75, 90, 98 percentiles in order to have another view about the model precision.

3.3 Failed approaches

During the first phase of the project, we made several tries with different technologies that did not turn very succesful in terms of effectiveness.
Here is a brief description of those attempts, which were then discarded either because of slow performances or low accuracy.

3.3.1 Paragraph selection through sentence embeddings

A common technique for question answering we found is to work on tokens representing sentences rather than single words. We used this approach to try to address the second step of the IR task (selecting the paragraph containing the question).
As such, we tried two different libraries to extract sentences from the original dataset:

- *TextBlob*
- *SpaCy*

At the end, TextBlob was the one that yielded the best results, turning each paragraph into a coherent set of sentences.
We then compared the results we obtained with different sentence embedding models, which are:

- *Infersent*
- *Sentence BERT*
- *Doc2Vec*

The embeddings were used to locate the correct paragraph inside a document in two different ways:

- The first try was to calculate the embedding of every paragraph as the mean of the embedding of the sentences that composed it, then we calculated the similarity of each question with all the paragraphs of its document, selecting the one that gave the maximum result.

- The second try was to use a neural network (similar to the Seq2Seq used in the SQUAD task) that took as input the embeddings of the question and the paragraphs, calculated as before, and gave as output the index of the paragraph that contains the answer.

Unfortunately both approaches didn't give satisfying results, so they were not pursued anymore and we used also for this step the same approach to retrieve documents.

Chapter 4

Experimental setup and results

4.1 SQUAD task's results

After several empirical tries, the best results have been obtained with the configuration:

- Number of Epochs: 8
- Batch Size: 128
- Latent Space Dimension: 128
- Embedding Dimension: 100

Exact(%)	F1 score(%)	Answered	Total
43,50	56,08	6635	6635

Table 4.1: Scores of the baseline SQuAD model

Table 4.1 shows the overall results of the model, out of 6635 samples, 2886 have an exact match.
Of the remaining 3749 samples:

- the mean of the similarity scores, computed by comparing the predicted answer with its ground truth, is **0.79** $\pm$ **0,10**
- **886** predicted answers were contained inside the ground truth
- **419** predicted answers contain the ground truth

4.2 IR task's results

If not specified, the following tests are performed using only the documents of the test set and the relative question. This means that for each question we define a ranking of the test set's documents and used the position of the target document (i.e. the one that contains the answer) to compute the metrics.

4.2.1 LSA model

We built the LSA model with a latent dimensions equal to 350, removing from the corpus of the documents all the *hapax legomena* (i.e. the tokens occuring only one time).

Average position	5,30
Standard devitation	7,33
Target doc as first result(%)	41,3%
Median	2
75%-90%-98%	6-15-31

Table 4.2: Results of LSA on IR task searching for the target document

4.2.2 Doc2Vec model

After doing a search for the best value for the hyperparameters we chose the following configuration:

- *Number of Epochs*: 26
- *Latent Space Dimension*: 100
- *Alpha*: 0,025 and *Min_alpha*: 0.00025 (alhpa is the initial learning rate; then it will linearly drop to min_alpha as training progresses)

Observing the previous results, we saw that the two models, LSA and Doc2Vec, had similar performance on the same task.
The main difference is that Doc2Vec more often put in the first position the target document, while LSA put in average the target document in a higher position.

Average position	7,42
Standard devitation	10,56
Target doc as first result(%)	44,8%
Median	2
75%-90%-98%	9-24-41

Table 4.3: Results of Doc2Vec on IR task searching for the target document

4.2.3 LSA + Doc2Vec ranking

Even if the model have similar performaces, they may probably make mistakes in differents cases, so we thought that combining the similarity scores of both the models could improve the search.
To combine them we performed, for each question-document pair, an average of the scores of the two model and as the following table shows, the results are really improved from before.

Average position	3,99
Standard devitation	6,78
Target doc as first result(%)	57,3%
Median	1
75%-90%-98%	3-11-30

Table 4.4: Results of LSA+Doc2Vec on IR task searching for the target document

4.2.4 LSA + Doc2Vec + Exact Match bonuses ranking

After testing on the validation set possible different values for the bonuses, we define the $EM\_bonus = 0.2$, and the $NER\_EM\_bonus = 0.2$, obtaining also in this case a good improvement on the results.

Average position	2,66
Standard devitation	4,40
Target doc as first result(%)	65,99%
Median	1
75%-90%-98%	2-6-18

Table 4.5: Results of LSA + Doc2Vec + EM bonuses on IR task searching for the target document

4.2.5 LSA + Doc2Vec + Exact Match bonuses ranking on paragraphs

Then we performed a test in order to understand how good this model is in detecting which is the paragraph that contain the question, knowing the target document.

Average position	1,57
Standard devitation	1,31
Target paragraph as first result(%)	73,8%
Median	1
75%-90%-98%	2-3-6

Table 4.6: Results of LSA+Doc2Vec on IR task searching for the target paragraph, knowing the target document

4.2.6 LSA + Doc2Vec + Exact Match bonuses ranking on all the documents

This test in the same performed in the paragraph 4.2.4, but in this case the documents are not just those of the test set, but of all the dataset.

Average position	18,04
Standard devitation	44,00
Target doc as first result(%)	25,1%
Median	3
75%-90%-98%	10-48-176

Table 4.7: Results of LSA + Doc2Vec + EM bonuses on IR task searching for the target document (on all the dataset's documents)

4.3 End-to-end

The final evaluation was performed as an end-to-end test, where all the models seen so far were involved: the questions were fed into the IR model in order to extract documents and then paragraphs, and the IR output was chained into the question answering model in order to compute the final answer. The experiments were conducted to find out if the whole chain is able to extrapolate the right paragraph and subsequently the right answer.
The chain follows these steps:

- *Document Extraction* - compute the document ranking for the question and then extracts the N most promising documents.
- *Paragraph Extraction* - take all the paragraph from the N most promising documents and compute the ranking. Then slice the M most promising paragraphs.
- *Question Answering* - for each paragraph out the M most promising, compute the start and end and a confidence score for both. Use the start and end to cut the answer from the paragraph and return it as output.

At first, looking at the IR scores, we decided to take in account the first 3 documents, as they were enough to have more then 75% of possibilities to retrieve the right document. For all of those, we extrapolated a paragraph ranking and extracted the most promising. At last, the paragraph was used as input, with the question, in the neural model and the answer was computed.
By analysing the outputs and the errors of the model, we found out that a relevant number of times the first paragraph doesn't contain the right answer: the errors of each step of the chain are subject of a cascade effect which amplify the error on the final result. For this reason we raised the number of

the promising paragraph and sort the relative potential answer by the sum of confidence scores (start and end confidences) given by the question answering model. This strategy improved the effectiveness of the model.
This process was replicated for each question of the test set, but for time reasons, we were unable to complete the test. As the number of documents and parameters growth the computational time grows too: the model can take up to 30 seconds to compute an answer. So we decided to limit the test set to 2000 samples, for any end-to-end test performed.
The test was repeated with different configurations, in particular a different amount of documents were used in the IR search database in order to highlight the impact of the database richness in the final scoring.

Documents	Paragraphs	Exact(%)	F1 score(%)	database size	#questions
3	1	4.05	6.70	1916	2000
3	10	5.30	7.21	1916	2000
20	1	1.30	2.83	18896	2000
20	10	1.74	3.12	18896	2000
20	40	2.30	3.35	18896	2000

Table 4.8: Scores for End-to-End

Chapter 5

Results analysis

5.1 SQUAD Error Analysis

After training the model and evaluating the performances in quantitative scores, we started analysing the quality of the predictions, checking if there is any recurring patterns when the model fails to predict the correct answer.

Semantic Ambiguities

There are some cases in which the model predicts an answer which is semantically equivalent to the ground truth, usually with very minor differences from it.

Example:

- **Context:** The Olmec hieroglyphs tablet has been indirectly dated from ceramic shards found in the same context to **approximately 900 BCE**
- **Question:** When has the Olmec tablet been dated to?
- **Predicted answer:** Approximately 900 BCE
- **True answer:** 900 BCE

Another ambiguity is when the text has multiple correct answers but the model picks one that doesn't match the one chosen as ground truth.

- **Context:** Tennessee is home to several Protestant denominations, such as the **National Baptist Convention** (headquartered in Nashville)

.... **The Southern Baptist Convention** maintains its general headquarters in Nashville.

- **Question:** Which Protestant denomination has its headquarters in Nashville ?
- **Predicted answer:** Southern Baptist Convention
- **True answer:** National Baptist Convention

Wrong focus

Sometimes the model correctly understands the type of answer it has to output (ex. a date, a name and so on), but fails to individuate the correct context inside the text

- **Context:** The earliest good evidence for oligochaetes occurs in the Tertiary period, which began **65 million years ago** ....Body fossils going back to the mid Ordovician, from **472 to 461 million years ago**, have been tentatively classified as oligochaetes.
- **Question:** When were the earliest annelid fossils found?
- **Predicted answer:** 65 million years ago
- **True answer:** 472 to 461 million years ago

5.2 IR task Error Analysis

Both the models used for IR task (LSA and Doc2Vec) are trained only on the SQUAD datasets; despite this dataset is quite big, it could be not enough to deeply understand semantics, as a pre-trained models could do. This limit is probably the main cause of errors.
Other difficulties can occur when the length of the question is particularly low, making more difficult the creation of a proper representation by the models.
Then, it's important to consider the negative effects of the presence of out-of-vocabulary tokens in the question (due to typos, foreign words or just absence in the dataset used for training). In particular, when a question doesn't contain any valid token, with LSA no vector is generated preventing the generation of the document ranking, while with Doc2Vec the related vector is basically randomized.

Chapter 6

Discussion

In the end the proposed models gave promising results, we had the occasion to try out many different approaches, compare them, and then decide which one performed better.

Possible improvements

There are some improvements that might be done to both models in order to improve the performances and have more accurate answers:

- Use Doc2Vec pre-trained models and use more text to train LSA model for IR task.
- Add another neural network layer to the IR model that, given as input a series of candidate answers from different paragraphs, returns the one that better answers the question.
- Add query-reformulation or query-expansion techniques to both models in order to make it easier to find the answers.
- Improve question taxonomy, in order to better classify the pairs question/answer types.

Future works

It should be possible to adapt the proposed models to other questioning answers dataset, such as WikiQA, NewsQA and so on, deploying the models with other datasets might help also to analyze more in-depth the errors made by the network.
Furthermore, it would be interesting to check if the weights learned by the model, just like BERT, could be easily adapted, through fine tuning, to other tasks such as sentence classification, next sentence prediction and so on.