Question Answering and Information Retrieval on SQuAD 1.1 Dataset

Using NLP techniques to extract answers from texts

Irene Rachele Lavopa[1], Lorenzo Loschi[2], and Alessandro Maggio[3]

[1]irenerachele.lavopa@studio.unibo.it
[2]lorenzo.loschi@studio.unibo.it
[3]alessandro.maggio5@studio.unibo.it

January 2022

Abstract– We deployed NLP techniques to extract meaningful features in order to feed a neural architecture based on BiDAF. Our aim was to produce a span of text representing an answer to a given question. We also built an information retrieval system based on tf-idf and similarity scores.

1 EXECUTIVE SUMMARY

The initial SQuAD dataset was structured in a JSON file which has been transformed in a .csv file in order to obtain a cleaner and more readable structure.
We applied a pre-processing step aiming to replace special characters and filter out uncommon symbols to reduce the length of the text and creating a more homogeneous dataset with less OoV terms. After this step we recomputed the initial and final index of the answer to make it match the new pre-processed text. Further informations were extracted from data, adding an exact match (1 if a context word occurs in the question, 0 otherwise) and a POS tagging. The obtained dataset was composed by these columns: question id, question text, context text, context id, start index of the answer, end index, exact match, POS.

To build the vocabulary we decided to use a Tokenizer based on the Keras one, using as input GloVe embeddings of dimension equal to 200, assigning to OoV terms a random value from a uniform distribution. We fitted the Tokenizer on the context column, exploited to convert tokens to indexes and to apply a padding based on the maximum length of both context and question. Regarding POS tagging, we created a naive tokenizer useful to map each tag to a unique discrete index. The last pre-processing step was to extend padding to exact match and POS tagging in order to obtain shapes that were coherent to the previous padded ones.

Concerning the models for the question answering task, we built two different models: DrQa (baseline) and a custom model based on BiDAF. The baseline model is inspired by DrQa, with small changes on the question encodings and in the alignment of question embeddings, which was used as comparison to our architecture. Our custom model applies more changes to the BiDAF backbone, especially in the attention layer, word embeddings and introducing new features as input. The evaluation of the performances of the model were based on precision, recall and F1score, all of them were calculated for each couple context-query and then averaged.

As Information Retrieval is concerned, firstly trough the pre-built tfidfVectorizer() we created a term – document bag of words matrix both for question and context. By means of a similarity measure (Cosine, Jensen-Shannon) we retrieved the top-5 most similar document with respect to each query. The overall system is an Hybrid Question Answering.

Our final architecture aims to recover promising documents given a query using our Information Retrieval that are going to be further processed by our Question Answering model in order to get a prediction only over the most related documents found. This is the final layout:

$$\text{Query} \rightarrow \text{IR} \rightarrow \text{QA} \rightarrow \text{chosen answer between candidates}$$

2 BACKGROUND

2.1 BiDAF[1]

Bi-Directional Attention Flow (BIDAF) network is an architecture designed to solve the tasks of machine comprehension (MC) and question answering (QA), reaching up to 68.0% of exact match and 77.3% F1 scores on the SQuAD test set, outperforming all the previous performances on that dataset. It was presented at ICLR 2017.

The model encodes both context and question at different levels of granularity by performing subsequently character-level, word-level and contextual embeddings, and uses bi-directional attention flow to obtain a query-aware context representation.
The Attention flow layer is responsible of linking and fusing information from the context and the query words, exploiting a bi-directional mechanism, query-to-context and context-to-query, which provides complimentary information to each other. Context-to-query (C2Q) attention explains which query words are most relevant to each context word producing a matrix containing the attended query vectors for the entire context. Therefore, Query-to-context (Q2C) attention explains which context words have the closest similarity to one of the query words and are hence critical for answering the query. Hence, this results in obtaining a context vector indicating the weighted sum of the most important words in the context with respect to the query. Attention vectors and contextual embedding are then merged either by concatenation or by a learnable function. Subsequently, the merged input pass trough the Modelling Layer (Bi-LSTM) with the purpose of capturing the interaction among the context words conditioned to the query. Finally, the output layer computes the probability distribution of the start index over the entire paragraph by a SoftMax of the weighted concatenation (learned weights) between the output of the modelling layer and the output of the attention layer. On the other hand, the probability distribution of the end index is computed in the same way with the only difference that the Modelling layer output is furtherly refined in a Bi-LSTM.

The loss is defined as the sum of the negative log probabilities of the true start and end indices by the predicted distributions, averaged over all examples

$$L(\theta) = -\frac{1}{N}\sum_{n=1}^{N} y_i \log(p^1_{y_i^1}) + \log(p^2_{y_i^2})$$

where θ is the set of all trainable weights in the model, N is the number of examples in the dataset, y_i^1 and y_i^2 are the true start and end indices of the i-th example, respectively, and p_k indicates the k-th value of the vector p. BiDAF exploits Adadelta optimizer.

2.2 DrQA[2]

The DrQA system consists of two components, the Document Retriever module for finding relevant articles and a machine comprehension model, Document Reader, for extracting answers from a single document or a small collection of documents.

Document Retriever exploits a non-machine learning approach to narrow the search

space to only promising documents. Documents and Questions are compared by means of tf-idf weighted bag of words, but taking also in account Bigrams, and the top 5 articles matching the queries are going to be the candidates to be further processed by the document Reader. The implementation of the Document Reader is taken as grounded, we only aim to stress how paragraph features are defined since we have took inspiration from this idea in our model. Indeed, beside pre-trained Glove embedding, each paragraph token is also expressed by the Exact match feature, the token feature and the aligned question embedding. The first one is just a dummy variable with value equal to 1 if that paragraph token occurs in the question and 0 otherwise, while the second one is a set of manual features that reflect some properties of token p_i in its context, including its part-of-speech (POS) and named entity recognition (NER) tags and its (normalized) term frequency (TF). Lastly, aligned question embedding exploits attentions scores to encode soft alignments between similar but not identical words. As shown in the DrQA paper in the Feature ablation analysis of the paragraph representations of our Document Reader, all these additional features proved to be beneficial, especially exact match and question alignment.

2.3 Jensen–Shannon divergence[3]

The Jensen–Shannon divergence is a method to measure the similarity between the probability distributions of two documents. It's defined as

$$JSD(P,Q) = \frac{1}{2}D(P||M) + \frac{1}{2}D(Q||M) \quad with\ M = \frac{1}{2}(P+Q)$$

where P,Q are the probability distributions of the two documents and $D(A||M)$ is the Kullback-Leibler divergence between A and M, where M is the average document distribution between the two documents. Therefore, we measure how much each of the documents P and Q is different from the average document M. The usage of this divergence corresponds to solving the statistical test of equivalent probability distribution of two documents Ho: p1 = q1, p2 = q2, ... , pn = qn with P = (p1 , p2, ..., pn) and Q = (q1, q2, ... qn) multinomial distribution of the words in document 1 and 2.The Jensen–Shannon divergence is symmetric and ranges between 0 and 1.

3 SYSTEM DESCRIPTION

The proposed architecture consists in two different parts: a Question Answering model to extract proposed answers for each query and an Information Retrieval system which, given a query, the most related documents are retrieved and then used to obtain answers with our deep neural network.

3.1 QA model

As Question Answering model, we defined two different approaches. The first one, which will be considered as our baseline model, is inspired from DrQA model applying some modification in the question alignment, question encoding and in the final similarity. The second one is the core model of this project, as already mentioned is inspired from BiDAF but with several differences.

3.1.1 Baseline Model

The backbone structure is the one described in DrQA paper, therefore we will just describe the main differences from that implementation. For the sake of description in the following $X_{T\times d}$ will be the context matrix where T is the total number of context word and d the embedding dimension and $Q_{J\times d}$ will be the question matrix where J is the total number of the question words.

Question alignment is obtained from the Multihead layer where $X_{T\times d}$ is the input matrix, while $Q_{J\times d}$ is the key-value matrix. Attention scores are computed in the same way as in DrQA, however in our model the function $\alpha()$ is not learned and it is considered as the identity function. To recover the loss of this information, we exploited multiple heads in our layer. The product of the attention scores and question embeddings is computed in the same manner as the multiplication occurring inside the Multihead layer, when attention matrix and value matrix are unified.

The difference in the computation of the weighted **question encoding** $\sum_{j=1}^{n} b_j q_j$ is how much importance is given to the vector $b_{1\times J}$. This is performed trough a 1D convolution network; indeed we have exploited a stack of 1D kernel to avoid heavy stemming and try to learn more complex relations (for the sake of description, it will be expressed as if it was a single 1D convolution).
Given the input question matrix $Q_{J\times 2d}$ a 1D convolution with kernel size equal to 1 and 1 output channel is slid along the question matrix with a SoftMax activation. In this way the learned filter will be a $1 \times 2d$ vector and, moreover, by applying a SoftMax activation to the final output we can obtain the importance vector where each element is computed with the same equation as the one expressed in DrQA.

The **final prediction** is computed in the same way for both start and end index. In our implementation we changed the way the matrix W is learned by applying over the encoded question a 1D convolution kernel with filter size equal to 1 and a Relu activation, with context embedding dimension as output channel. In doing so, the learned kernel can be seen as the learned weights matrix W. The result of the convolution is multiplied along with the context matrix and activated with a Relu function. The final output distribution is obtained by applying SoftMax over the just obtained matrix.

3.1.2 Core model

Our Question Answering model is a hierarchical multi-stage process and consists of six layers:

- Word Embedding layer: maps each word to a vector space using a pre-trained word embedding model;
- Context features layer: adds features information to further encode context sentence;
- Contextual Embedding: utilizes contextual cues from surrounding words to refine the embedding of the words;
- Attention Flow Layer;

- Modeling Layer: employs a Recurrent Neural Network to scan the context;
- Output Layer: provides an answer to the query.

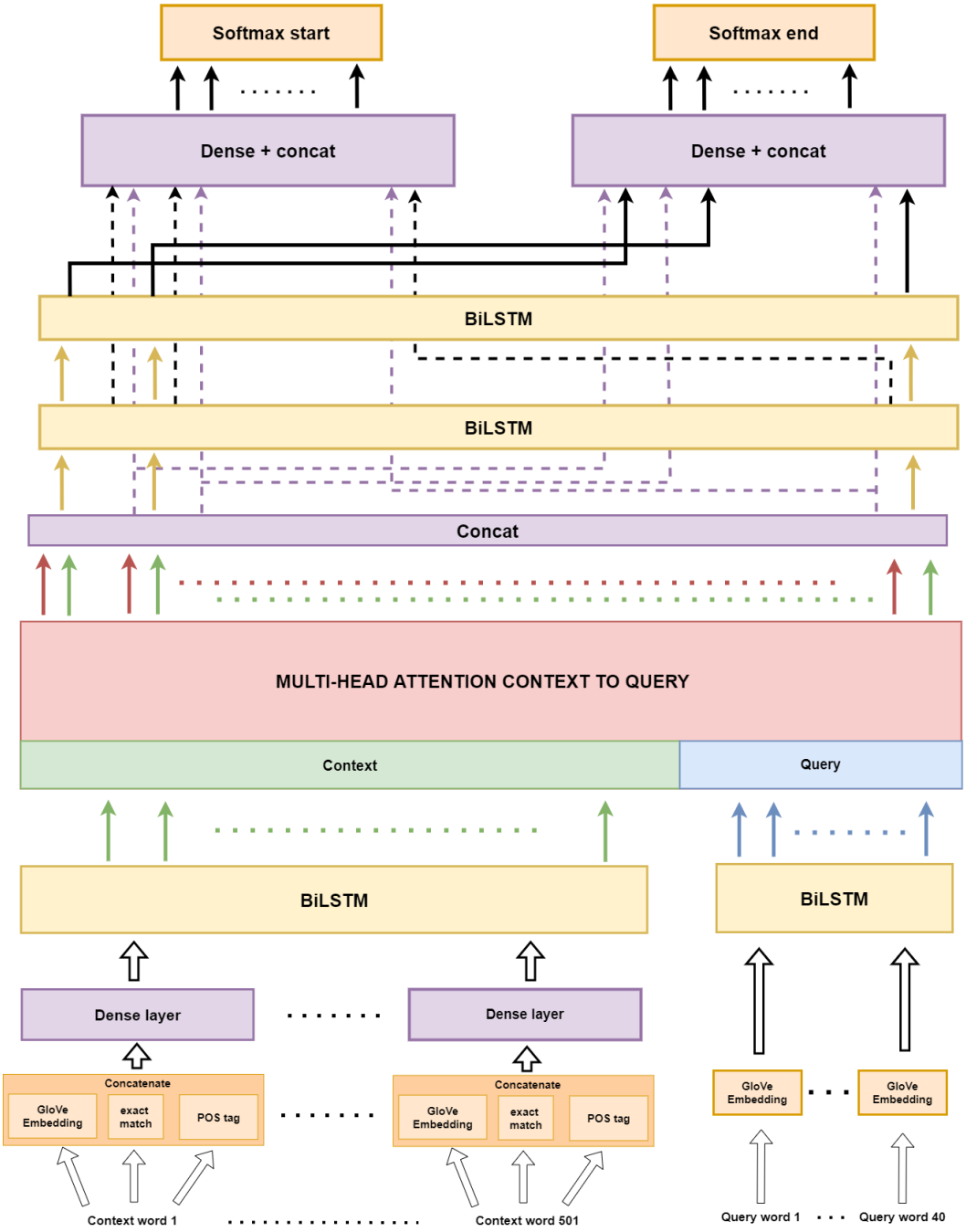

Figure 1: Architecture of the Neural Network.

Word Embedding Layer: By means of pre-trained word vectors, Glove 200, we obtained a word embedding for all tokens in both question and context. This layers returns as output two matrices: $X_{d\times T}$ for the context and $Q_{d\times J}$ for the query, where T is number of words in the context and J is the number of words in the query.

Context features layer: This layer adds to each context word external features that can be extracted with a context analysis. The Final context feature vector is obtained by concatenating its Glove embedding with the following features:

- $f_{ExactMatch}(x_i) = 1$ if and only if that word occurs also in the coupled query and 0 otherwise
- $f_{POS}(x_i)$ which is an array of dimension (1 x 38) containing the one-hot encoding of the POS tag of a context word obtained through nltk Tokenizer.

Therefore, the output of this layer is a matrix $\hat{X}_{k\times T}$ where $k = d + 38 + 1$

Contextual Embedding: Firstly, through a Dense layer, the contextual feature matrix is compressed from $d+39$ embedding dimension to the initial one equal to d. This step allowed us to obtain a more "structure aware" context embedding and, at same time, to make query and context dimensions matching again. Subsequently, we exploited a Bilinear LSTM on top of the previous embedding layer in order to model the temporal interactions between words. The results of both Long Short Term Memory layers have been concatenated together doubling the the first axis dimension (from d to 2d), hence, we obtain a matrix $H_{2d\times T}$ from context feature words and a matrix $U_{2d\times J}$ from query words.

Attention flow layer: This layer is responsible of linking together context and query informations. Unlike the BiDAF attention mechanism, which was based on a bi-directional attention mechanism $(C2Q–Q2C)$ we decided to exploit a Multihead attention layer[4]. Considering H as our query matrix and U as our key–value matrix, we obtained a matrix $H_{2d\times T}$. This design choice aimed to press which query words were the most relevant to each context word with respect to multiple dimensions. Finally, contextual embedding and attention vector are concatenated together to yield matrix $G_{4d\times T}$ where each column can be seen as a query aware representation of each context word $G = [H; \hat{H}]$.

Modeling Layer: Given the input of the layer G which encode query aware representation of the context word, we aimed to output a matrix encoding interaction between contextual word given the query. This is done by exploiting a bidirectional LSTM which return a matrix $M_{2d\times T}$ since each single LSTM has an output dimension equal to d and results are concatenated. This is different from what is done in the contextual embedding layer since previously we captured interaction that were independent from the query. Therefore, each column of M contains the representation of a given context word given the information inside in both query and in the whole context.

Output Layer: This final layer is responsible of returning the probability distribution of the start and end index over the context. The start index is obtained by $p^1 = SoftMax(w_{p^1}^T[G; M])$, where $w_{p^1}^T$ is a trainable weight vector. On the other output, the end index probability distribution is computed similarly as the previous one, as $p^2 = SoftMax(w_{p^2}^T[G; M^2])$ where M^2 is a matrix obtained passing M over an additional Bi-LSTM layer. As before $w_{p^2}^T$ is a trainable vector.

Loss function: We exploited the same loss function proposed in the BiDAF Layer. We implement it by reweighting the tf.nn.weighted_cross_entropy_with_logits adding the possibility to set a negative weight argument beside the positive one. Therefore, the negative weights have been set to 0 rather than the usual 1, in this way the weighted binary cross entropy boils down to the same loss function previously mentioned. The choice of BiDAF Loss function over the binary cross entropy was beneficial in constraining the network to improve performances by learning where to place the span rather than learning where not to place it.

3.2 Answer formulation

The predicted answer is computed by taking the top 4 (hyper parameter of the model, empirically calculated) initial and final indices found by our model, then we calculate every possible couple of them. We restricted the choice only for couples with an initial index value lower than the final index one. Finally, given a question, we calculate the vector similarity between that and every possible couple using cosine similarity. These two sentences are processed removing their stop words and are then vectorized using an average word embedding calculated with word2vec. This step is a sort of post-processing we added to regularize the neural model output, taking into to account the domain knowledge (end index can not be less than start index etc.).

3.3 Extended Model - Hybrid Question Answering System

The extended model combines the two different approaches by running the Information Retrieval architecture to find the most relevant documents related to a given query, and then computing the Question Answering network only on these documents. We extract an answer for each retrieved document-question pair, we clean both answer and query by means of a removal of stop words in order to compute a Jensen-Shannon similarity between every prediction and the initial question, the most similar answer found becomes our final prediction.
This kind of approach is inspired to Watson, an Hybrid Question Answering System where different candidate answers are compared with a scoring method. In our case this method is the Jensen Shannon distance applied on the tf-idf matrix computed on all the question-answers proposals, similarly to what is done among all questions-document pairs in the very first steps of the architecture (Document Retrieval).

3.3.1 Tf-Idf setting

Firstly, a tf-idf vectorizer is fitted on the train context (90% of the whole dataset context) to produce a term-document bag of word representation for both question and documents before and question-answers at the end. We decided to rely on this traditional NLP technique thanks to its huge number of successful applications in this field.
The same last 10% of the dataset is kept as test set for both QA and IR in order to preserve the consistency of our experiments.

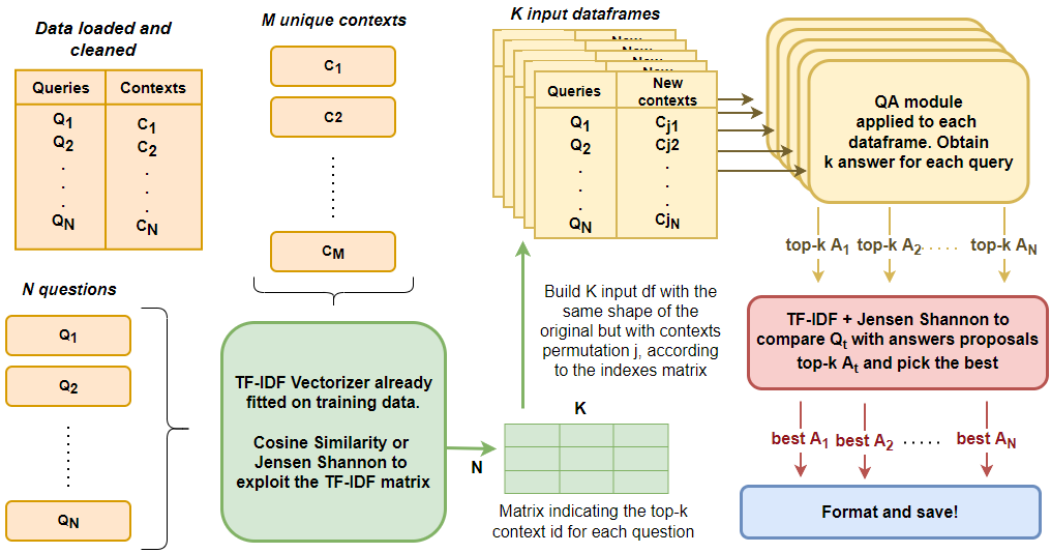

Figure 2: Structure of the extended model

For efficiency reason data are loaded as a standard dataframe and some cleaning/preprocessing operations are applied. Then the questions column and the contexts column are split in two different logical flow: questions are never permuted while contexts are identified by their id and duplicates are removed. The stopwords are removed only to perform the tf-idf operations but the QA module operates on the whole text.

A pre-trained TF-IDF Vectorizer returns two matrixes, one for the questions (N, features) and one for the contexts (M, features) where N is the number of questions, M is the number of unique documents, features depends on the pre-trained vectorizer. Then, according to the user choice, is computed a (N, M) matrix by using the Cosine Similarity or N by M Jensen-Shannon distances. In both cases are selected the TOP-K documents (maximum similarity or minimum distance) and the resulting product will be a matrix (N, K) of the K best context_id for all the N questions.

In the next step, K dataframe are built with the correct contexts permutation according to the previous matrix. Each of them is considered as a standard input dataframe to a QA problem. The overall outcome will be a set of K answers per query.

In order to obtain N outputs (and not N times K) a picking policy has to be defined. We decided to use again TF-IDF + Jensen Shannon (since K is small) so that the final function returns exactly what is expected from a standard QA task (the IR step is transparent to the user).

It is worth to highlight that every link between the question and the right context is lost, so there are no order relationships influencing the experiments. For this reason the exact_match function is performed after that the K distinct dataframes are built so that the question is compared with its new context.

4 EXPERIMENTAL SETUP AND RESULTS

4.1 Preprocessing

For Question Answering we set 60% as train size, 30% and 10% for validation and test size respectively. As far as QA is concerned, the dataset has been cleaned by eliminating multiple spaces, uncommon symbols or special character. We decided not to remove any stop words, because their encoding was an important information for the structure of the sentence, therefore they were crucial on the overall performances. Furthermore, we didn't apply a lowering on the words to perform better the POS tagging. Concerning the Information Retrieval system, the dataset has been preprocessed with the same cleaning as before, however performing also stop words elimination and lowercase transformation. Lowercase transformation has been performed in order to reduce the number of tokens, resulting in a more compact document/query description for following steps.

4.2 Training

Regarding the Question Answering model it's worth pointing out several implementations set up to understand how we achieved our results. In our first versions of the model it has been empirically observed a strong overfitting behavior after just few of epochs. To address this problem we have implemented **Dropout** over each bi-LSTM, Dense layer and over the final output layer with drop rate $= 0.3$; this resulted to be beneficial in dealing with severe overfitting after just few epochs. This hyper-parameter has been observed to be crucial for training performances, a lower value would restrict too much the model and a greater value would destroy at all the performances. We also tried to introduce L2 and L1 regularizers but they tend to slow down the learning phase, so we decided to not implement any kind of regularization but Dropout.

Increasing the **Embedding dimension** proved to be significant in pushing up the overall performances, we started from a very small dimension (equal to 50) and then we increased it to 200. We also tried to train our model with an embedding dimension equal to 300, however we could not seen any relevant difference w.r.t. the previous dimension within the number of epochs for training. This dimension could have found to be beneficial by training for more epochs, but due to hardware limitation it was impossible to dig deeper in the training. Perhaps a larger model might handle the 300-embedding without loss of information or curse of dimensionality.

As optimizer we exploited NADAM since it has shown better performances then ADAM, AdaGrad, SGD and AdaDelta optimizers. NADAM is an extension to ADAM using Nesterov Momentum, this helped in the speed of our training. In addition it does not require any further configuration setting, unlike other optimizers, that would have been difficult to be properly fine-tuned due to hardware limitations.

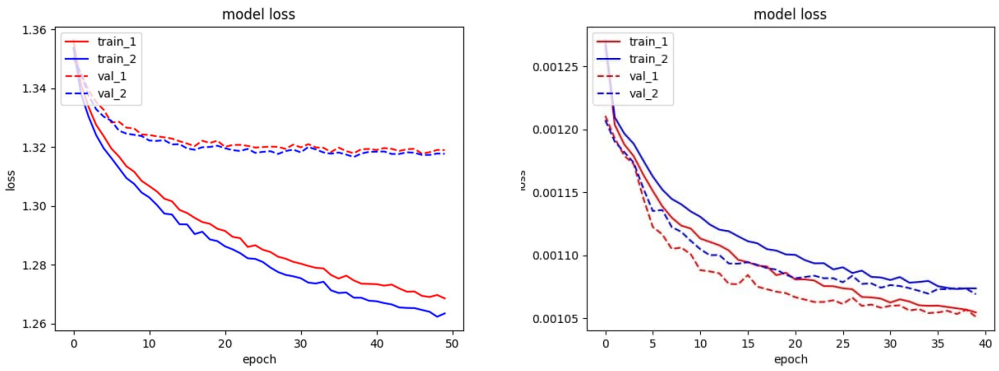

Figure 3: Loss before and after adding Dropout.

A meaningful intake to the overall performance of the QA network was given by the **contextual feature layer**, an idea inspired by the DrQA paragraph encoding layer. Differently from their work, we decided to not introduce any question alignment in the context feature encoding even if, as shown in the DrQA paper, it could provide a relevant impact in the overall performances when combined with the exact match feature. This choice was made because if we would have added the question alignment, the following attention layer would have been redundant (we use a Multi-Head Attention vs. the standard C2Q), as well as the Modelling Layer and the Contextual Layer, since in both of them we have captured the interaction of context word w.r.t. the query.

4.3 Custom Loss vs Binary Crossentropy

We decided to exploit the same loss function explained in the BiDAF paper (chapter 2.1). The binary crossentropy suffered from the hard unbalancing between classes. Indeed, given a context with 501 words, 500 of them are 0 (not the start/end index) while just one word is predicted as 1 (start/end index). As result, our model learned much more frequently where it should not place start/end index rather then actually understand where precisely it should occur. On the other hand, the BiDAF loss function does not decrease when we predict where start/end index do not appear, but it will only decrease if we predict correctly a larger number of exact positions of start/end indexes.

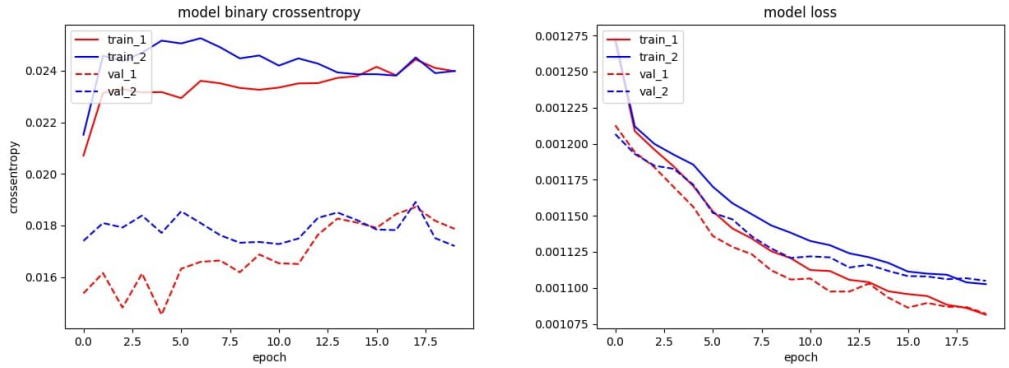

Figure 4: Difference between Binary Crossentropy and our custom loss

4.4 Information Retrieval

Regarding **Information (Document) Retrieval**, we decided to compare query and document with cosine similarity and Jensen-Shannon divergence. Indeed, in the case paper[5] is shown that cosine similarity is not optimal in defining inter-document similarity. Therefore, we added the Jensen-Shannon divergence (as pointed out in chapter 2.3). This similarity measure provided optimal solutions, with visible increasing performances w.r.t. the cosine one, however it is slowest to compute it, especially when dealing with a huge number of documents.

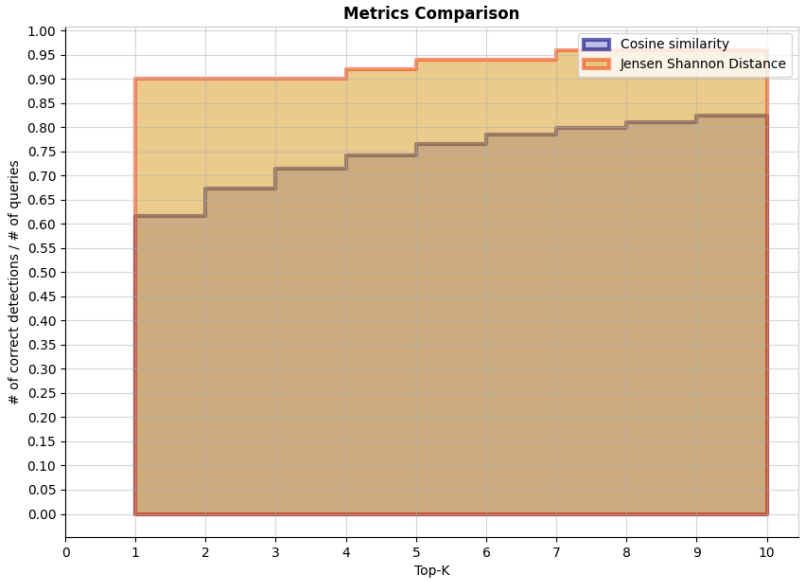

Figure 5: Difference of performances between cosine similarity and Jensen-Shannon

5 ANALYSIS OF RESULTS

These are the results of the training on 60 epochs with our final best core model chosen based only on the validation performances.

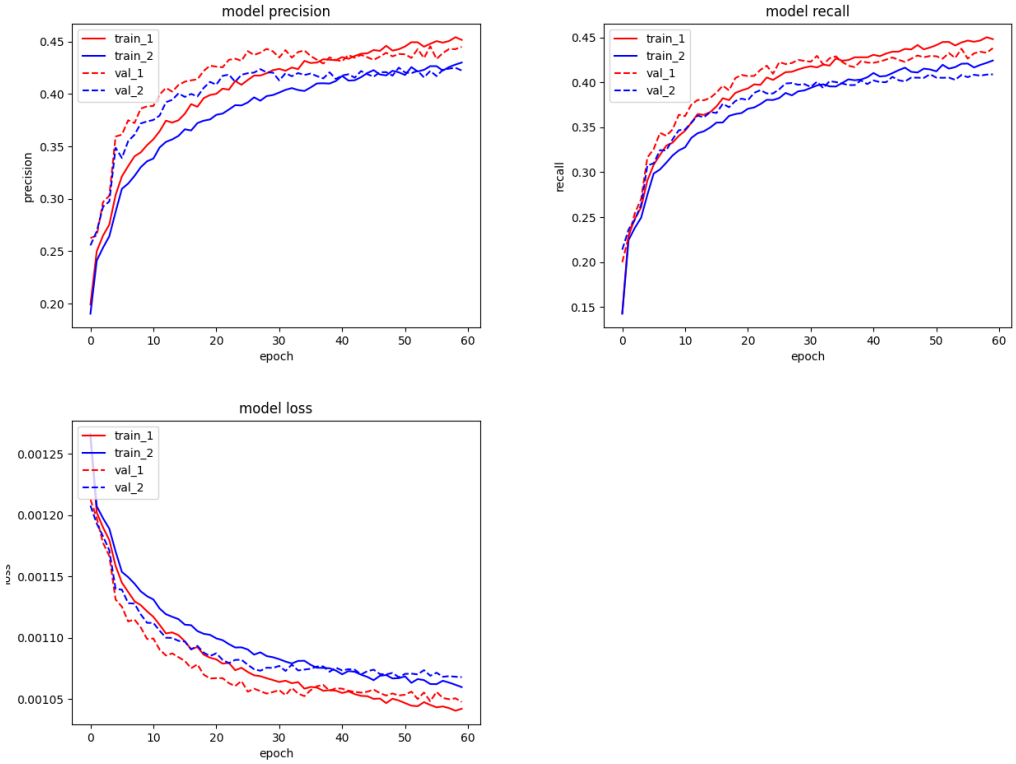

Figure 6: Precision, Recall and Loss of the final model

It is clear that the model does not present any kind of overfitting, even if our model is quite big with ≈ 29 millions of parameters whose ≈ 4 are trainable. Compared to the baseline model we obtained a significant improvement, since with the baseline we only achieved both precision and recall values lower than 0.35, training with the same number of epochs.

5.1 Metrics

To evaluate our Question Answering models we used the official SQuAD evaluation script and our custom script based on the top 4 answers.
These results are calculated according to the best model found based on the validation performances with the official script on the test set.

Core Model	
Metric	Value
Exact match	1.24%
F1score	23.5%
Total	8859
Has Answer exact	1.24%
Has Answer F1	23.5%
Has Answer total	8859

Table 1: Official SQuAD script evaluation

By observing the previous metrics we decided to apply a further analysis in order to evaluate the performances of our model basing on the best answer predicted over the top4 indexes found, as described in chapter 3.2 "Answer Formulation". We computed the Precision metric by defining as True Positive if the ground truth index is contained in the top 4 (out of 501) predictions, otherwise it is considered False Positive. Here are the results:

Core Model with top4	
Metric	Value
Start index Precision	0.52
End index Precision	0.57

Table 2: Top4 evaluation

We can clearly see by the exact match value that our core model is not able to correctly pick the right sentence from the context, for this reason we tried to understand if our model is actually able to find a correct index (start/end) over the top 4 predictions or if its behaviour is mostly due to a wrong learned pattern. As shown from the table 2 we can assert that more than in 50% of the cases we exactly guessed the right index in the 4 upmost predictions, this confirms that our model actually learned a valuable answering pattern. This pattern couldn't have been observable by only watching the official evaluation.

In case of extended model (so by activating the Document Retrieval) the prediction formulation is identical whenever the right context is selected. Otherwise it performs as its best with the wrong context. Using a different dataset, with more contexts suitable for the same question, we may see more interesting results.

```
SOME INTERESTING EXAMPLES

id = "57343a9f4776f41900661aaa"
Ground truth: "Museum District"
Predicted: "South of the Downtown"
comment: Wrong answer, but still a location.

id = "57343cffd058e614000b6b5f"
Ground truth: "generally cool"
Predicted: "subtropical climate"
comment: Again the right topic, but wrong answer.

id = "57343cffd058e614000b6b60"
Ground truth: "mountains"
Predicted: "The mountains to the west act as a partial barrier"
comment: As often, right answer but returns all the sentence.
```

Example 7: Some examples of predictions.

6 DISCUSSION AND FURTHER IMPROVEMENTS

As we can observe from our previous results we didn't likely reach an optimal solution for Question Answering, as the majority of our predictions are larger than what we actually expected. This means that our model is able to answer correctly to a query, but it does not take the straight answer. The previous metrics penalize the performances of our model since they only take into account if the indexes matched and not if the right answer is contained in the indexes found, even if it was semantically correct.

As it is shown in table 2, we can assume that if we would have used a different similarity measure to choose the final answer among the 16 possible couple of indexes (the one used in spacy is the cosine similarity) we would have probably reached better results and, as consequence, we would also have obtained an higher exact match in our custom evaluation. This could also be done with other NLP techniques, even neural networks, that may retrieve the correct answer in a fine grained way among the small top4 indexes.

A further technique that could be implemented to improve the capacity of the IR module is the lemmatization of words during the pre-processing phase. This could help in reducing a lot the term-document matrix space and enhancing the similarity among documents containing no more the same words, but the same lemmas. Furthermore, since POS applied to the context has been beneficial in boosting the behaviour of our QA model, we could also have added a POS tagging even for the queries that may help in understanding which patterns of the context are linked to the ones of the query.

Finally a quite obvious technique to increase the performances consists in building a larger and more complex models able to exploit an higher embedding space and more training epochs. These improvements would require high-performanc hardware.

References

[1] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, "Bidirectional attention flow for machine comprehensiondy," *ICLR*, vol. abs/1611.01603, 2017.

[2] D. Chen, A. Fisch, J. Weston, and A. Bordes, "Reading wikipedia to answer open-domain questions," vol. abs/1704.00051, 2017.

[3] "Jensen–shannon divergence." /https://en.wikipedia.org/wiki/Jensen%E2%80%93Shannon_divergence.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," vol. abs/1706.03762, 2017.

[5] J. S. Whissell and C. L. Clarke, "Effective measures for inter-document similarity," 2013.