Report SQuAD project

Francesco Farinola
Michele Vece

October 2021

1 Abstract/Introduction

Among Question Answering models, DrQa [1] and BiDAF [3] represent two well-known architectures. We decided to use them as baselines for building our model.

The BiDAF model is a hierarchical multi-stage architecture for modeling the representations of the context paragraph at different levels of granularity. It includes character-level, word-level, and contextual embeddings, and uses bi-directional attention flow to obtain a query-aware context representation.

On the other hand, DrQA system consists of two components (Document Retriever and Document Reader) which exploit different input features for paragraph and question encoding.

The proposed model can be viewed as a revised version of the BiDAF model. The architecture is very similar to it, but it is more lighweight. Also, it includes additional features as inputs, as suggested in the DrQA paper.

2 Background

Question Answering (QA) is a subfield of Machine Reading Comprehension (MRC) in which each question is given related context from which to infer the answer.

2.1 MRC tasks

It is possible to distinguish among four main MRC tasks [2]:

- **Cloze Tests:** also known as gap-filling tests, the system is asked to fill in the blank with the missing items;
- **Multiple Choice:** the system should select the correct answer to the question from candidates according to the provided context;
- **Span Extraction:** given the context and the question, the machine is required to extract a span of text from the corresponding context as the answer;

- **Free Answering:** the machine uses free-form natural language to answer questions. The answer may not be a subsequence in the original context.

Here, the proposed model is designed to perform Span Extraction tasks.

2.2 QA architectures

A typical Question Answering system contains four key modules [2]:

- **Embeddings:** encode input natural language words into fixed-length vectors. Word representation methods can be sorted into Conventional Word Representation and Pre-trained Contextualized Representation. To encode more abundant semantic and linguistic information, multiple granularity can be adopted;
- **Feature Extraction:** extract features of the context and question separately. Recurrent neural networks (RNNs), convolution neural networks (CNNs) and Transformer architectures are applied in this module;
- **Context-Query Interaction:** the attention mechanism plays a critical role in emphasizing which parts of the context are more important to answer the questions. The attention mechanism can be categorized into *unidirectional* or *bidirectional*, and in *one-hop* or *multi-hop* (according to the number of interactions between the context and the question);
- **Answer Prediction:** it gives answers to questions according to the original context. Its implementation is highly task-specific. In case of span extraction, it is possible to choose between:
  - the *sequence model*, which outputs the positions where answer tokens appear in the original context;
  - the *boundary model*, which only predicts the start and end positions of the answer.

Here, the proposed architecture is a boundary model based on RNNs with a one-hop bidirectional attention mechanism. Concerning the embedding, distribute word embedding (GloVe) is integrated with multiple granurality informations (Exact Match, POS, NER and Term Frequency).

3 System description

3.1 Dataset

3.1.1 Selection of the data

First of all, correspondences between answers and spans where they are extracted are analysed. Since some errors in the dataset have been signaled, it has been decided to:

- remove rows containing *multiple* instances of the answer in the context, because the start index may refer to the wrong span;
- remove rows where the answer is not located at the suggested start index (if any).

In this way, there should be no misalignment between the answers and their span start index.

In order not to completely eliminate those records, they are used as **test set**. However, since they may contain titles included in the training set, they are *not* used as reference to make decision about the generalization of the model.

3.1.2 Train/Validation split

Data are split in training and validation, according to the ratio 4/1. As suggested, the splitting is based on the `title` column, in order to maintain the questions/paragraphs regarding the same title in the same split.

Decisions concerning the performance of the model are taken **exclusively** according to the results on the **validation set**, since it does not include any title already present in the training set.

3.2 Text cleaning

Before feeding the data to the model, some preprocessing operations are applied. In details:

- **Contractions expansion:** contractions are expanded (e.g., *won't* → *will not*);
- **Tokenization:** the text is split in tokens. `Spacy` is used since it performs this operation in a more accurate way if compared to the `split` method of the class `string`;
- **Punctuation and symbols removal:** non-alphabetic characters are removed. In details:
  - some characters (such as hyphens, dashes, parenthesis, `:`, and `+`) are replaced by a space, since they may be used to join two different words;
  - other characters (such as `.` or `,`) are substituted by an empty string;
  - `£`, `$`, `%` are maintained since they are considered important, also because they appear in some answers;
- **Split of letters, digits and symbols:** words containing numbers or symbols are split in more tokens (e.g., *$5* → *$ 5*);
- **Spell correction:** mispelled words are corrected by `symspellpy`. In order not to compromise the meaning of the word:

  - numbers, short words (length<5) and words that start with an uppercase letter are ignored;
  - the edit distance is up to 2;
  - the original word is returned in case no correction within this edit distance is found;

- **Lemmatization:** each word is lemmatized with the lemmatizer provided in `nltk`, in this way different inflected forms of the same word are reduced into the same root form;
- **Lowercase:** uppercase characters are converted to lowercase in order to further normalize the text;
- **Strip text:** additional whitespaces are removed.

3.3 Features

As suggested in [1], Term Frequency, Exact Match, POS and NER tags are provided as input to the model. An ablation analysis in [1] on the feature vector of paragraph tokens has been conducted and showed that these features are similar but complementary and all contribute to the performances of the final model.

3.3.1 Term Frequency

The Term Frequency (TF) refers to the number of instances of the same word in the same sentence (in our case, in the same `context`).

It is achieved by using `CountVectorizer` from `scikit-learn`. The token pattern is `\S+`, meaning *everything but not a whitespace*.

An array of shape `1` containing the term frequencies is concatenated to the paragraph encoding feature vector.

3.3.2 Exact Match

The Exact Match (EM) consists in finding which terms in the context match with at least one term in the question. The aim is to verify the existence of words in the context that are relevant to the question.

Three types of exact match are computed as binary features:

- **Original match:** it compares the terms 'as they are'. Actually, both the context and the question are applied contractions expansion, tokenization, punctuation and symbols removal, alphanumeric and symbols splitting and text stripping. Furthermore, we decided to remove the stopwords from the question since they may cause noise and undermine the objective of the exact match;
- **Lowercase match:** as before, but also lowercase is applied before comparison;

- **Lemmatized match:** as before, but also lemmatization is applied before comparison.

As a result, an array of shape `3` of binary values is concatenated to the paragraph encoding feature vector.

3.3.3 POS and NER tagging

Also Part-Of-Speech and Named Entity Recognition tags are included as input.

Part-Of-Speech (POS) tags describe the characteristic structure of lexical terms within a sentence or text, therefore, we can use them for making assumptions about semantics.

Named entity recognition (NER), instead, seeks to locate and classify named entities in text into pre-defined categories (such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.).

Both of them are obtained from `spacy`, which identifies 54 POS tags and 20 NER categories. In addition to traditional tags:

- a `PAD` tag is included both in POS and NER, referring to artificially added tokens that are *not* part of the context;
- a `NONE` tag is included in NER, indicating terms that are part of the context, but no NER tag is assigned by the algorithm.

Before being fed to the system, tags and categories are converted to integers and then, one-hot encoded. As a result, an array of shape `54` for POS tags and an array of shape `20` for NER is concatenated to the paragraph encoding feature vector.

3.4 Embedding

3.4.1 Word Embedding

Both [1], [3] use GloVe for Word Embedding, with a dimension $d = 300$. In our case, due to resources contraints, GloVe is used with a reduced dimension, $d = 100$.
Our embedding matrix is built according to the following:

- terms in the GloVe vocabulary are assigned their GloVe vector representation;
- out-of-vocabulary terms, are assigned the average of the vector representation of the nearby in-vocabulary terms. In case no neighbour is in the dictionary, a random vector is assigned.

Then, during the validation/test phase:

- if the term is in the (training) word listing, then there exists a representation for the term in the embedding matrix and so the corresponding vector is assigned;

- otherwise, a zero vector is assigned.

3.4.2 Character Embedding

In [3], each terms is provided with its character embedding, obtained by applying CNNs followed by a max-pooling layer.

Here, for simplification, we use embeddings provided by `chars2vec`, with a dimension $d = 50$. It represents each sequence of symbols of arbitrary length with a fixed length vector. Given two or more words, the higher the similarity in words spelling, the higher the similarity between the corresponding vectors.

3.5 Further preprocessing

3.5.1 Padding

Inputs are padded before feeding them to the model since it requires data of the same length. A different maximum length is computed for contexts, questions and texts (namely, the answers).

In order to allow the system to process also data whose length is greater than that of the training set, the maximum length is increased of the 10%.

The choice of this percentage is due to resource constraints, but it could be further increased. In any case, if the maximum length is greater than the defined one, the system truncates the exceeding tokens at the end of the sequence.

3.5.2 Indices

Once sentences undergo text cleaning operations, original `answer_start` indices may result incorrect. As a consequence, new start/end indices need to be computed.

New indices are not character-based but token-based, namely refer to the position of a token w.r.t. other tokens (i.e., at the n-th token corresponds the index $n - 1$).

3.6 Span reconstruction

Given start/end indices of the suggested span, answers need to be reconstructed. In order to do so, an intermediate dataset is used.

It is obtained by applying some of the previous preprocessing operations (but not all) to the original dataset in such a way to guarantee the following properties:

- at each token in the preprocessed dataset corresponds a token in the new dataset, even if the token may be different due to normalization operations;
- new sentences contain words close to the original dataset (except for punctuation/symbols), since lemmatization and lowercase are not applied.

To reconstruct the original span, start and end indices are used to extract the answer in the intermediate dataset. Then, string matching is applied to find the corresponding span in the original dataset and the answer is returned.

3.7 Model implementation

Contexts and Questions

The first layers in the network are the `context` and `question` input layers. They contain the padded contexts and questions, respectively.

Word, Character and Contextual embedding

Context and question words are replaced by their GloVe word embedding and by their character embedding via two Embedding layers.

These layers are then concatenated and passed to a Dense[1] layer whose dimension is smaller than the sum of the two embedding dimensions. The aim is to obtain, for each word, a new shorter representation which synthesise the previous ones.

After that, the outputs are passed through a BiDirectional GRU[2] to obtain contextual representations for contexts (`H`) and questions (`U`).

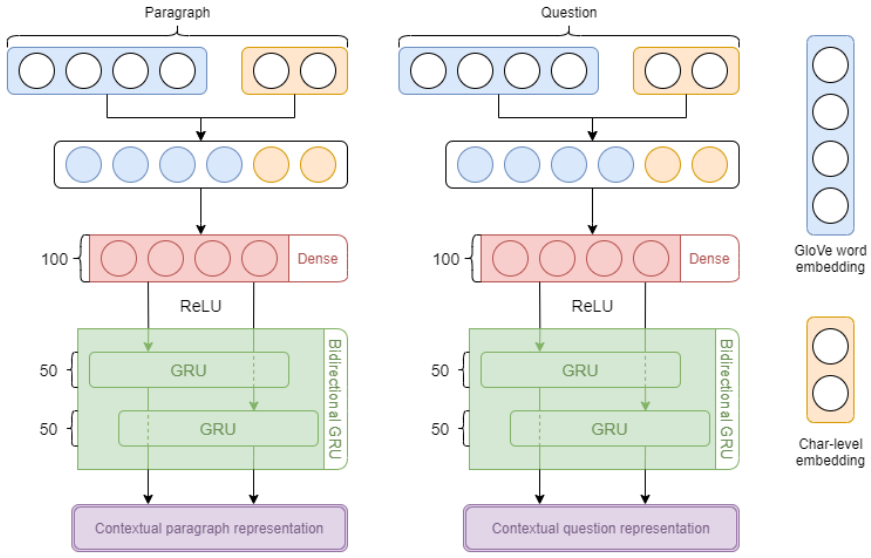

Figure 1: From context and question inputs to their contextual representation.

[1] In BiDAF, they use a two-layer Highway Network.
[2] In BiDAF, they use LSTMs. We opted for GRUs since they are more efficient.

Attention

Contextual representation are used to compute Attention.

First, the similarity matrix `S` is calculated. Each element s_{ij} is obtained as :

$$s_{ij} = w^\top [h; u; h \circ u]$$

with w trainable vector. In details, `H` and `U` are both tiled to obtain two matrices of dimension[3] `(context_size, question_size, embedding_size)`, so that we can easily compute the element-wise product $h \circ u$. Then, they are multiplied by `w` and squeezed on the last dimension to get the similarity matrix `S` of dimension `(context_size, question_size)`.

Second, we compute the Context-to-Query Attention. It measures which query words are most relevant to each context and it is defined as:

$$\tilde{U}_t = \sum a_{tj} U_j \quad \text{with} \quad a_t = \text{softmax}(S_t)$$

As indicated, we get `a` by applying softmax to `S`. Then, we add the context axis to `U` and the embedding axis to `a`, multiply them[4] and sum over the question axis. After squeezing, we end up with `C2Q` of dimension `(context_size, question_size)`.

Third, Query-to-Context is computed. It refers to which context words have the closest similarity to one of the query word and it is defined as:

$$\tilde{h} = \sum b_t H_j \quad \text{with} \quad b = \text{softmax}(\max_{col}(S))$$

The maximum is computed along the last axis, getting a tensor of dimension `(context_size)`, at which softmax is applied returning `b`. It is expanded on the last axis, so that can be multiplied to `H`. Then we sum over the context axis and tile, still on the same axis, to get `Q2C` of dimension `(context_size, question_size)`.

Input Features

Input features, referred as "multiple granularity", are passed through input layers. All of them have `context_size` as first dimension. The second dimension, instead, is `3` for Exact Match `em` and `1` for Term Frequency `tf`; `54` for Part-Of-Speech `POS` and `20` for Named Entities `NER` (after embedding).

Merge layer

In the original paper, contextual embedding and attention vectors are concatenated in the following way:

$$[h; \tilde{u}; h \circ \tilde{u}; h \circ \tilde{h}]$$

Here, we concatenate `H, C2Q, H*C2Q, H*Q2C` together with the additional inputs, obtaining `G`.

[3] For simplicity, the batch dimension is ignored.

[4] Broadcasting will make the dimensions match

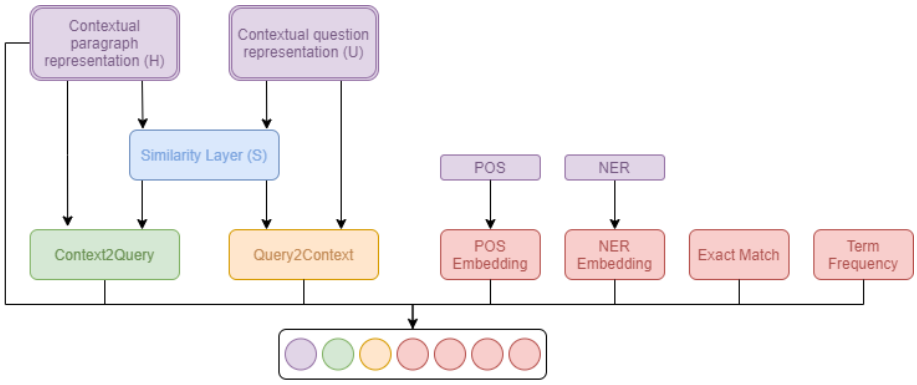

Figure 2: Attention, Input Features and Merge layer

Modeling layer and start/end probabilities

The output from the previous layer `G` is processed via GRUs (one for the start prediction, two for the end prediction), obtaining the modeling layers `M` and `M2`, respectively. They are concatenated to `G`, passed to `Dense` and `TimeDistributed` layers. After squeezing on the last axis, we get two tensors of dimension `(context_size)`, at which `softmax` is applied to return start and end probabilities for each context index.

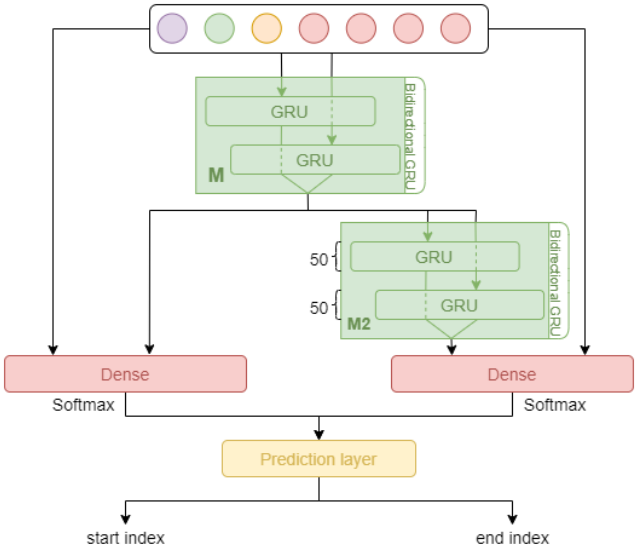

Figure 3: Modeling and prediction layer.

Prediction

As suggested in [1] and [3], it is forced the following:

$$i_{start} \leq i_{end} \leq i_{start} + k$$

where k is a fixed threshold. Here, this constraint is not applied after the training, but it is already included in the trained model.

In order to do so, start and end probabilities are expanded on different axes, such that their matrix multiplication gives a square tensor of dimension `(context_size, context_size)`. Now, `outer_band` is used to put everything outside a central band (of width k) to zero, such that the constraint is satisfied. After `reduce_max`, start and end indices are returned.

4 Experimental setup and results

4.1 Final model configuration

Table 1 resumes the final configuration of the model.

The system was trained with Nadam optimizer, batch size = 16 and early stopping with patience = 5 on validation loss. The training stopped after 12 epochs.

Table 1: Final model configuration

Inputs dimensions	
GloVe	100
Chars2Vec	50
POS	54
NER	20
Term Frequency	1
Exact Match	3
Units per layer	
Dense	100
GRU	50
Text maximum length	
Context	728
Question	44
Answer	47
Other	
k	15
GRU Dropout	0.2

4.2 Alternative configurations

Initially, other configurations with higher parameters were tried (e.g., word embedding up to 200, character embedding up to 100, dense units up to 200, GRU units up to 100).

Since overfitting manifested, higher dropout rate (i.e., 0.3, 0.5) was initially adopted. It mitigated the overfitting but, as side-effect, decreased the performance.

A valid alternative consisted in the reduction of the dimensions of the model. This increased the performance. That's the reason of this "lightweight" implementation.

Furthermore, also k was increased up to 47 (the maximum length of the answer), in order not to cut any prediction. However, changes in the performance were almost irrelevant.

5 Analysis of results

5.1 Final model results

Running the `evaluate` script on the whole dataset, the results in Table 2 are returned:

Table 2: F1, exact match on whole dataset

Data	Exact match	F1
training_set.json	52.27	67.44

Table 3, instead, resumes metrics separately computed on training, validation and test set:

Table 3: Metrics on training, validation and test set

Set	Start EM	End EM	Precision	Recall	F1
Training	0.6072	0.6514	0.7292	0.7113	0.6881
Validation	0.5158	0.5552	0.6211	0.6178	0.5918
Test	0.4836	0.4926	0.4947	0.5742	0.5117

Obviously, the model is more performant on data on which it is trained. However, still good results are obtained on unseen data, i.e., validation set. Low performance on exact match on test set is predictable, since these data may contain incorrect indices. Low F1, instead, should be further investigated.

5.2 Comparison to other models

Since the proposed model is inspired both from DrQA and BiDAF models, we decided to implement them. Then, we used these implementations as baselines.

In order to make a fair comparison, both architectures have configuration analogue to that shown in Table 1.

If we consider the validation loss as metric for comparison, it is possible to notice how all the models achieve, at the end of the day, similar results. However, our model not only has the lowest loss, but also achieves it in fewer steps than the other architectures.

Table 4: Comparison on validation loss

Model	Minimum loss	Epoch
DrQA	3.4203	14
BiDAF	3.4063	14
Our model	**3.3810**	**10**

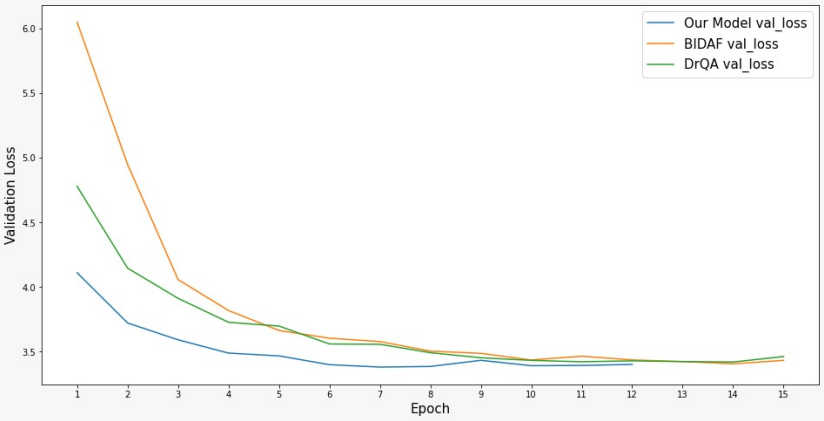

Figure 4: Validation losses over epochs

6 Discussion

Comparing the predictions on the validation set to the expected answers, it is possible to notice that 41.20% are correct while we need to make some considerations on the remaining 58.80% (10007 wrong predictions).

Partially correct predictions

Among the wrong predictions, 635 contain correct numeric values, i.e.:

`On which floor?`
Predicted: `61st` True: `61st floor`
`How many copies did the album sell?`
Predicted: `310,000` True: `310,000 copies`

Among those answers we can distinguish different types of predicted answers:

- there are some answers in which quantities are predicted correctly but the model generates, along with the quantity and without spacing, the correct unit measure, i.e.:

  True: `158.8 million` Predicted: `$158.8 million`

- in some cases, the answer predicted gives even more precise information than the one provided in the dataset, so the model captures quite well the context and gives enough importance to each token in the question, i.e.:

  `When did she appear on the cover of GQ?` (Beyoncé)
  True: `2013` Predicted: `January 2013`

Also, 2990 of the predicted wrong answers contain at least 2 words (with exact match or misspelled but still correct) that are also in the true corresponding answer. Table 5 resumes the situation.

Table 5: Correct or partially correct predictions

Classification	Quantity	% on validation set
Totally correct	7,011	41.20%
Correct numeric values	635	3.73%
Other partially correct	2,990	17.57%
Total entirely or partially correct	10,636	62.88%

All the previous kind of answers may be considered as correct predictions since semantically they provide similar, meaningful and still correct answers. Said so, we may conclude that F1 and EM are not perfect metrics to evaluate such models for this task.

Incorrect predictions

Among the remaining incorrect predictions (6382, 37.50% of the total), there are about 1295 (7.61%) which share at least one word[5] with the corresponding answer. Those answers have the chance of being correct but, by further investigations, most of them are not. So, we continue considering them wrong.

If we compute NER for the predicted answers and the true ones, we notice that among all the incorrect answers, only 2929 provide a Named Entity category and in 950 of those, the category is predicted correctly. So, even if the prediction is incorrect, the system is often able to understand the kind of answer required.

[5]Stopwords are not taken into consideration

Resume

Resuming (Figure 5):

- 62.50% of the predictions can be considered correct:
  - 41.20% have an exact match of the truth value;
  - 3.73% are answers in which the model predicts correctly the amount or the date but misspells the answer;
  - 17.57% are predictions which contain at least two words[6] that are also contained inside the true answer. In most of these cases, the answer contains the relevant information but does not match exactly due to stopwords, further or less details than the true one;
- 37.50% of the predictions can be considered wrong. As already said, 1295 of these have an exact match of at least one word but most of them are incorrect. These percentage can be further split:
  - 20.29% do not have a NE category available;
  - 5.61% match the NE category;
  - 11.60% do not match the NE category.

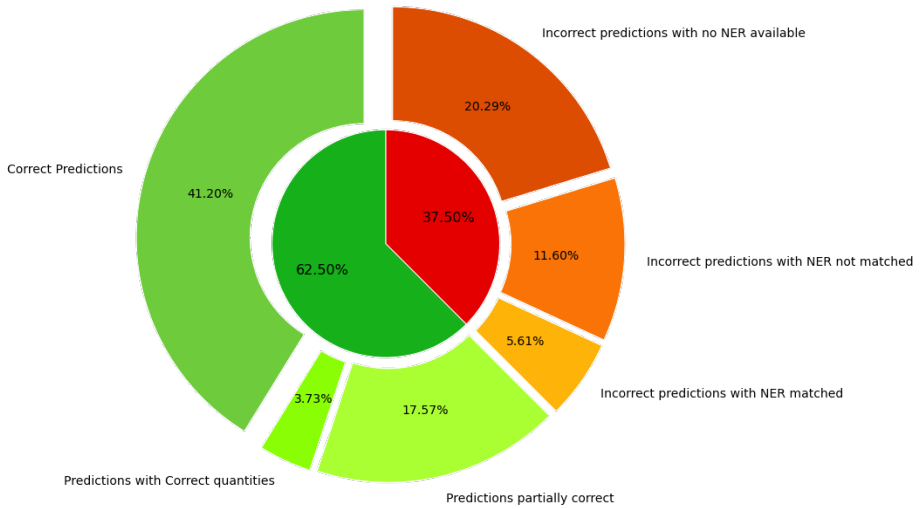

Figure 5: Predictions classification pie chart

[6]Here we did not consider words with one word match since it may include predictions by matching symbols but with wrong quantities, or symbols and cases where the answer may be 'Baltic Sea' and the predicted one is 'Dead sea'.

Further considerations

Finally, we can conclude that F1 and EM may not be the best metrics to evaluate a model for this task since a lot of particular cases need to be taken into consideration.

Also, such a model may provide some answers which can be correct and provide the relevant information but have no match at all with the truth value (synonyms).

Since we noticed that most of the incorrect answers given by the model do not have a Named Entity category associated, probably a more effective NER algorithm may improve the performance of the model.

References

[1] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading Wikipedia to Answer Open-Domain Questions. *arXiv:1704.00051v2*, 2017.

[2] Shanshan Liu, Xin Zhang, Sheng Zhang, Hui Wang, and Weiming Zhang. Neural Machine Reading Comprehension: Methods and Trends. *Applied Sciences*, 9(18):3698, 2019.

[3] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bi-Directional Attention Flow for Machine Comprehension. *arXiv:1611.01603v6*, 2018.