NLP Project

Niccolai Lorenzo
Pozzoli Davide
Vizzuso Simone

GitHub: https://github.com/treyvian/SQuAD-Question-Answering

Project report for
Natural Language Processing

Artificial Intelligence
University of Bologna
Italy

Contents

1 Abstract

Question Answering (QA) is a task which consists of generating textual answers given a question and a context, which is where the useful information for answering is. The task will be carried out on the Stanford Question Answering Dataset (SQuAD)[1] (version 1.1) which was provided to us with the material for this project. The project is divided in two main parts: data preprocessing and model experiments. After the preprocessing the dataset is used to compare the performance of different architectures.

2 Dataset

SQuAD dataset is a reading comprehension dataset, consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable.. The structure of the dataset json file has the following structure:

- Paragraph_1
  - Context_1
    - QAs_1
      - Answer text
      - Answer start index
      - ID
      - Question
    - QAs_2
    - ...
  - Context_2
  - ...
- Paragraph_2
- ...

The 1.1 version of the dataset contains 100,000+ question-answer pairs on 500+ articles.

3 Splitting

Since we have the data in the json format previously shown, the first step to obtain a usable collection of question-answer samples is splitting the data based on

title, as suggested. In particular we will create a dataframe of records composed as follows:

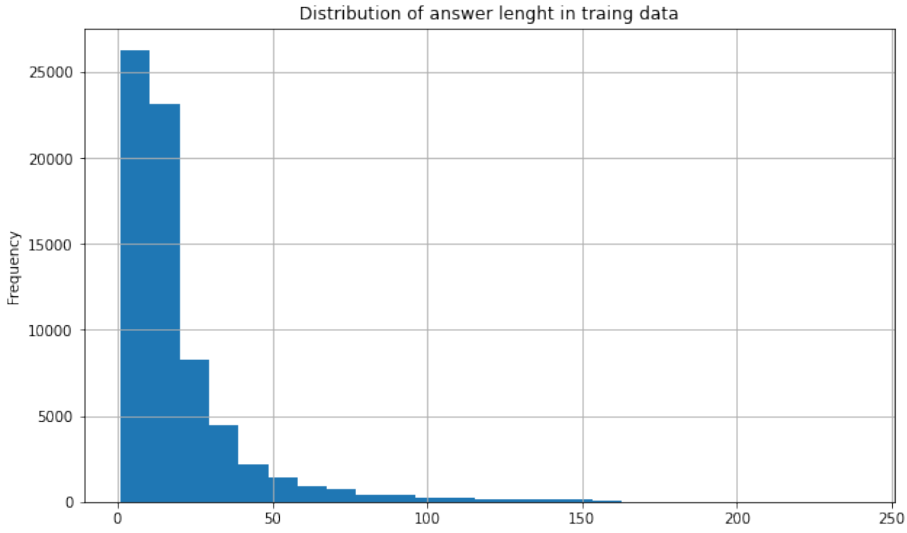
	title	context	question	id	answer_start	answer_text
0	University_of_Notre_Dame	Architecturally, the school has a Catholic cha...	To whom did the Virgin Mary allegedly appear i...	5733be284776f41900661182	515	Saint Bernadette Soubirous

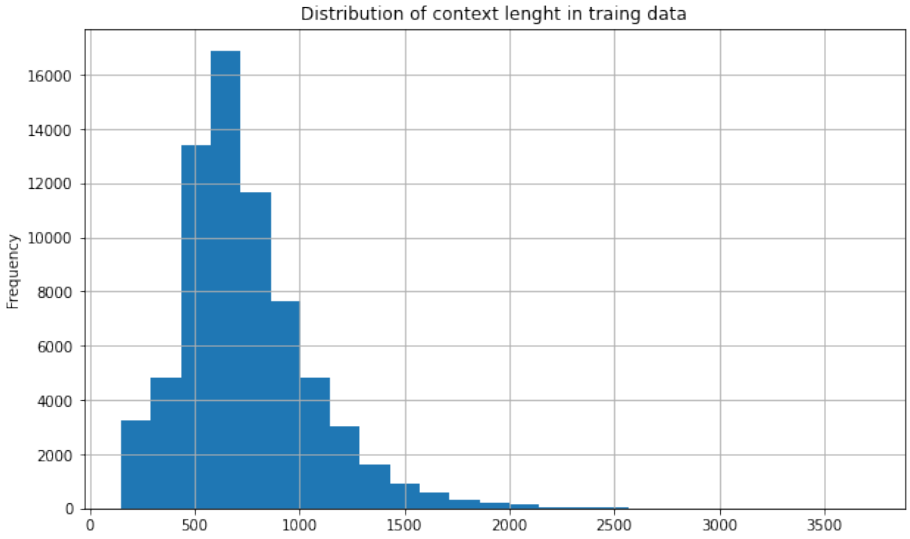
Through this process we create a dataframe made of 87599 samples, which we further split into training and validation using the suggested 80-20 proportions. Ending with 69392 samples for training and 18207 samples for validation.

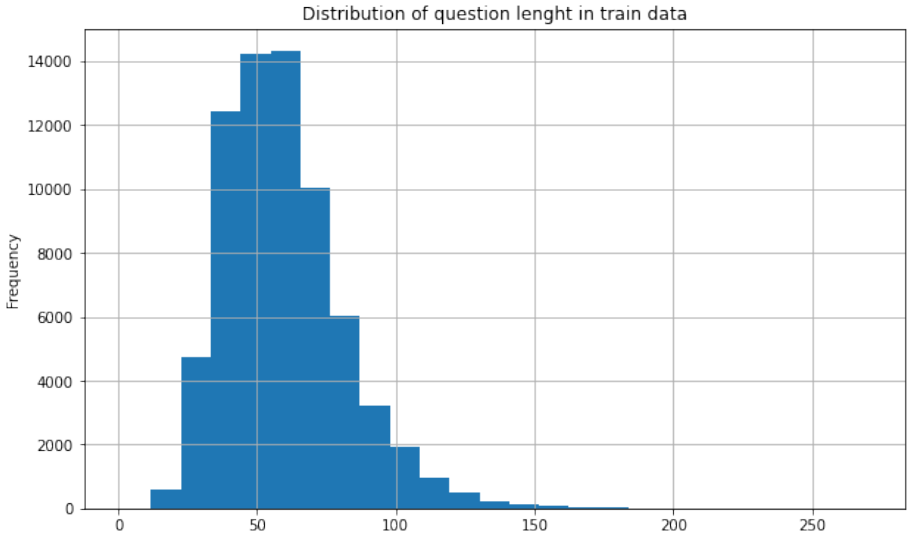
4 Data Exploration

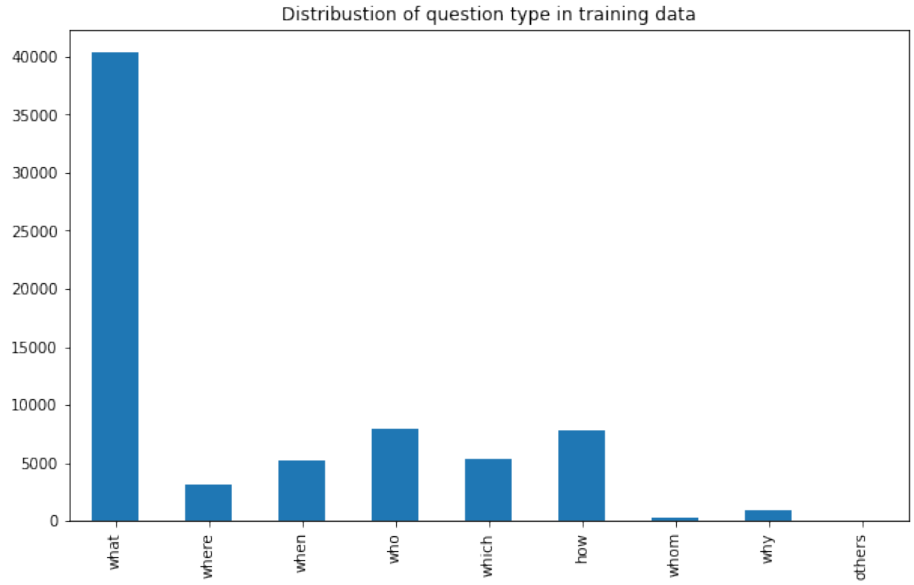
After the splitting we plotted some basic histograms to explore the training data and investigate the composition. In order:

- The distribution of the answer's length
- The distribution of the context's length
- The distribution of the question's length
- The distribution of the question's type (what, who, where,...)




Distribution of context lenght in traing data
Frequency
0
2000
4000
6000
8000
10000
12000
14000
16000
0
500
1000
1500
2000
2500
3000
3500


Distribution of question lenght in train data
Frequency
0
2000
4000
6000
8000
10000
12000
14000
0
50
100
150
200
250



5 Preprocessing

The data preprocessing is contained in the procedure `prepare_features`. The first operation performed is tokenization of the input sample, this is performed by a pre-trained tokenizer deployed through `AutoTokenizer`, a generic class which is part of the Hugging Face library.

The first 2 arguments correspond to the input text fed to the tokenizer is a span composed of the question and the related context, the code snippet can be found below.

```
inputs = tokenizer(examples["question"],
                   examples["context"],
                   truncation="only_second",
                   max_length = max_len,
                   stride=doc_stride,
                   return_offsets_mapping=True,
                   padding="max_length")
```

The other arguments passed to the tokenizer are handled by the `pad_on_right` boolean variable:

- `truncation`: since we want to truncate only the context and never the question.
- `max_length=384`: the maximum accepted length of the input span (question and context), if the span exceeds, it is going to be split. We set this parameter at 384.

- `stride=128`: since a sample having a question and a context could be split if it exceeds *max_length*, we have to prevent the context being truncated in the exact point where the answer is located. This is done using a stride, the parts resulting after the split will overlap by exactly the number of token passed here. We set the stride at 128.
- `return_offsets_mapping=True`: if set to `true` the offset mappings will be returned from the tokenizer. They are a map from token to character position in the original context. This will help to compute the start positions and end positions
- `padding="max_length"`: the outputs will be padded to *max_length*

The operations that happen after the tokenization are finalized to add 2 features to the input sample: the start position and end position of the answer in the context.

This function is applied both to the train and the validation split.

6 Models

We decided to tackle this task with different architectures, all of them are taken off the shelf, directly from Hugging Face library. The models were chosen based on the quantity of RAM used during training since we were limited by the Colab resources. We also tried larger model (BERT or alBERT) but they were too big to fit in the resources we had available.

In particular the models that we decided to train on SQuAD are:

- `DistilBERT`: a transformers model[2], smaller and faster than BERT, which was pretrained on the same corpus in a self-supervised fashion, using the BERT base model as a teacher. More precisely, it was pretrained with three objectives: distillation loss, Masked Lanuguage Modeling (MLM) and Cosine embedding loss. We used both the `cased` and the `uncased` versions.
- `DistilRoBERTa`: a distilled version of the RoBERTa-base model. It follows the same training procedure as DistilBERT. The code for the distillation process can be found here. This model is case-sensitive: it makes a difference between english and English. The model has 6 layers, 768 dimension and 12 heads, totalizing 82M parameters (compared to 125M parameters for RoBERTa-base). On average DistilRoBERTa is twice as fast as Roberta-base.

7 Tables and findings

In this section the results of the evaluation script ran on the models will be displayed. The metrics calculated from the script are 2:

1. **F1-Score:** it is calculated between the tokens of the computed answer and the ground truth (gold standard answer).
2. **Exact Matches (EM):** the ratio between the exact answers produced by the model and the total answers.

In the following table the reader can find the output metrics. The model[3] in the first row is selected as baseline for this task because of its simplicity.

	Exact Matches	F1-Score
Match-LSTM with Ans-Ptr (baseline)	54.50	67.74
DistilBERT-base-uncased	61.62	76.90
DistilBERT-base-cased	61.52	75.75
DistiRoBERTa-base	65.62	80.09

Table 1: The rows correspond to different model architectures, the columns correspond to different metrics.

7.1 Error Analysis

We would like to make a premise, the Exact Matches (EM) is a somewhat tricky metric to evaluate a Question Answering system, because as the name suggest it only counts the fully exact matches between computed answer and the gold standard.

For example we take in exam the following question (which is part of the test set):

```
Which city is the capital of Telangana?
```

The gold standard for this questions is:

```
Hyderabad is the capital of the southern Indian state of Telangana
```

While the answer produced as output by `distilbert-base-uncased` is:

```
Hyderabad
```

Of course this does not count as exact match, while for us humans is a pretty good answer, even if it omits some context which could be good for the questioner.

7.1.1 Discussing the F1 Score

One may argue that the F1-Score should be indicative of how good the model answers since it gives an estimate (taking into account precision and recall) of how similar the prediction and the gold standard are at a syntactical level. But for example the latter answer gives a F1-Score of 0.143.

After a broad estimate of how many samples can fall under this heading we came up with an upper bound of 15%. In particular we counted the computed answers which were not exact matches but where the gold standard text was contained in the prediction, or vice versa. Using the latter metric to count the possible good answers the output percentage is 85%.

7.1.2 Analyzing the "bad" answers

If the 85% can be considered "good", this leaves us with 15% of "bad" answers predicted. Now we are going to explore this portion of the results.

We tried to analyze how the bad answers were distributed by the related title (i.e. the context), to see if there is an imbalance in the model performance.

	Bad Answers	Size	Impact
Tucson, Arizona	14.05	3.08	43.27
Bacteria	28.70	1.80	51.66
Premier League	13.13	2.98	39.13
Roman Republic	18.11	3.27	59.22
Tuvalu	17.73	2.49	44.15
Immaculate Conception	53.85	0.87	46.85
United States Air Force	18.67	2.01	37.53
Qing dynasty	20.68	2.70	55.84
Religion in ancient Rome	23.46	3.38	79.29
The Bronx	18.82	2.26	42.53

Table 2: The rows correspond to different context titles, the columns correspond in order to: the percentage of bad answers relative to the total samples of the context in the test split, the percentage of how much of the test split accounts for the context and the overall impact on the model errors.

The Impact columns is simply the product of the first two columns, it gives an approximation of how much a context weighs on the model bad answers. In the previous table are listed the contexts which have an Impact over a given threshold. These particular contexts are 12, the total contexts contained in the test split are 58. This make us think that there is an imbalance in the model performance based on the input context.

7.2 Limitations and further improvements

The main limitations are related to the computational power to our use, we could not experiment with bigger architectures which could achieve better results on the task. So the main way to improve is to use models which are not distilled from a base model. The perk of using distilled architecture is the decreased complexity both in space and time, but of course there is a trade-off on performance.

References

[1] Pranav Rajpurkar et al. *SQuAD: 100,000+ Questions for Machine Comprehension of Text*. 2016. DOI: 10.48550/ARXIV.1606.05250. URL: https://arxiv.org/abs/1606.05250.

[2] Victor Sanh et al. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. 2019. DOI: 10.48550/ARXIV.1910.01108. URL: https://arxiv.org/abs/1910.01108.

[3] Shuohang Wang and Jing Jiang. *Machine Comprehension Using Match-LSTM and Answer Pointer*. 2016. DOI: 10.48550/ARXIV.1608.07905. URL: https://arxiv.org/abs/1608.07905.