Question Answering and Question Generation on the SQuAD Dataset (v. 1.1)

Natural Language Processing

Federico Battistella, Michele Iannello, Alessandro Pavesi, Andrea Polacchini

Alma Mater Studiorum - Università di Bologna

Master in Artificial Intelligence

A.Y. 2020/2021

Abstract

The main objective of this work is to address a Question Answering task, namely to develop an NLP system that, given a paragraph and a query about it, returns a single answer based on a segment of text from the paragraph. As an extension, we also address the task of Question Generation, consisting in the inverse problem of generating questions starting with paragraphs and answers.

We show that relatively simple approaches can nonetheless reach discrete results, while requiring less computational power with respect to large and well-established models. Specifically, we obtain an Exact Matching of 71.26% and an F1-score of 76.66 on Question Answering and a BLEU metric of 12.53 on Question Generation.

Introduction

The main goal of this paper is describe our approach to the task of Question Answering on the *Stanford Question Answering Dataset* (**SQuAD**) dataset (v.1.1) [Rajpurkar, Zhang, et al., 2016]. We include an additional chapter to describe an extension of the main work, addressing the task of Question Generation on the same dataset. Our work involves the use of external libraries and *task-agnostic* pre-trained models, to strive for consistent results and faster training.

Data

As outlined in the reference paper, SQuAD is "a [...] reading comprehension dataset consisting of 100,000+ questions posed by crowd workers on a set of Wikipedia articles, where the answer to each question is a segment of text from the corresponding reading passage." [ibid.].

SQuAD's reading sections are derived from high-quality Wikipedia entries and cover a broad range of topics, from music superstars to abstract concepts. A passage is a paragraph from an article that can be any length. SQuAD includes reading comprehension questions with each passage (also referred to as *context*). These questions are based on the substance of the passage and can be answered by reading it again. Finally, each question has one or more answers.

Chapter 1

Question Answering

Question Answering (QA) is a Natural Language Processing (**NLP**) area — specifically regarding Natural Language Understanding (**NLU**) and strictly related to Information Retrieval (**IR**) — that focuses on developing systems that automatically answer questions presented by humans in a natural language. A computer comprehension of natural language is defined as a program system's capacity to convert phrases into an internal representation in order for the system to create correct replies to queries posed by a user. QA systems may currently be found in search engines and phone conversational interfaces, and they're rather suitable at providing small information snippets. On more difficult queries, however, they usually merely provide a list of snippets that consumers must then read through to locate the answer.

1.1 Related Work

As reviewed in [Calijorne Soares and Parreiras, 2020], QA systems can be divided into four categories according to the paradigm they implement, namely:

- *Information Retrieval QA*: focuses on finding relevant documents and/or passages (e.g. using search engines), then applying filtering and ranking on the retrieved passages.
- *Natural Language Processing QA*: uses of machine learning methods to extract answers from retrieved (relevant) passages.
- *Knowledge Base QA*: relies on structured data sources (Knowledge Bases) to be queried after crafting semantic representations of the questions (e.g. using query languages).
- *Hybrid QA*: many high-performance industry systems combine several techniques together to exploit the respective advantages.

NLP approaches recently focused on Deep Learning techniques, in particular exploiting Recurrent Neural Networks (RNNs) and Memory Networks [Sharma and Gupta, 2018]. The later breakthrough success of Transformers [Vaswani et al., 2017] in pretty much all NLP fields paved the way to faster, lighter approaches to QA as well, raising the suspicion that slower and heavier structures such as RNNs could be outperformed by dropping the recurrent structure and focusing on self-attention, e.g. combined with convolutions [Yu et al., 2018].

1.2 Implementation

The goal of this work is the development of a model capable of extracting an answer, given a question and a relevant paragraph (*context*). Since SQuAD is a closed dataset, meaning that the response to a query is always a part of the context and covers a continuous span within it, the task can be formulated as that of finding the position of the answer within the provided context, as start and end position characters. This places the present work within the NLP approaces to QA (see Section 1.1).

1.2.1 Data preprocessing

Raw data was made available by the teaching professor of our Master NLP course, and consists of a JSON file providing data mentioned in the Introduction. The first task we addressed is the preprocessing of data, including data exploration, cleaning, transforming and formatting in order to achieve the best possible dataset to feed the model. To this purpose we deployed several preprocessing steps, in particular:

- **Question splitting**: since data is provided in the form of several questions for each context, we firstly had to duplicate the context as to equalize for the number of questions, so to have triples of context-question-answer for each initial answer. This step produced a considerable enlargement of the dataset size, which required us to save a compressed version of the file. We used Python library pandas [Reback et al., 2020] to load, inspect and preprocess data, and NumPy [Harris et al., 2020] to obtain compressed `.npz` files, achieving a 90% size reduction.
- **Tokenization**: We performed tokenization employing a pre-trained tokenizer in the HuggingFace implementation [Wolf et al., 2020]: in particular, our model structure allows for the joint tokenization of questions and contexts (see Section 1.2.2). Moreover, input samples need to be all of the same size in order to exploit Transformers: we accomplished this requirement by enforcing a maximum length in the tokenization process, and adding padding where needed. As an effect, tokenized data exceeding the maximum length needed to be truncated and splitted accordingly, thus producing several samples from a single one. We decided to overlap each context split by half their size, in order to limit the possibility that the answer span is cut within a context split to the highest possible extent. This approach enlarges the dataset but possibly reduces the errors of the model.
- **Data fixing**: As an effect of tokenization, the answers' positions provided as ground data needed to be shifted to account for the context splitting. In particular, we decided to accordingly rescale only samples whose answers spans fit entirely within the relative context split. Otherwise, namely if the answer is either partially found within the context split or totally absent, we chose to set both the answer start and end positions to 0, which corresponds to the token `[CLS]`, used by the tokenizer as a placeholder to indicate separations, thus not corresponding to any part of the tokenized sample. This feature enables us to manage unanswerable questions, a case not covered into the SQuAD dataset v.1.1, but supported in v.2.0 [Rajpurkar, Jia, and Liang, 2018]. Moreover, as tokenization is word-wise, a further adjustment is needed to translate answer positions, given as character-based indices, into word-based indices.

- **Training/Test split**: splitting the dataset into a training set and a test set was not straightforward, since the SQuAD dataset is firstly divided in titles (representing books or articles), with several passages extracted from each title. Hence, splitting at the level of titles and keeping all corresponding contexts within the same split achieves a more consistent splitting with respect to the naive one (i.e., simple random shuffling). We chose a test split ratio of 0.2, meaning that we employed ∼80000 samples for training and ∼20000 for evaluation purposes.

Issues SQuAD shows several peculiarities that turn out to be problematic for the tasks. A notable feature of the dataset is the possible presence of several different correct answers for the very same (question,context) pair, which introduces a multi-valued mapping problem. In particular, a conventional objective function such as the Categorical CrossEntropy is justified by the assumption of normally distributed samples — i.e., following an **univariate** Gaussian distribution. In the case of multiple possible outputs, though, this assumption no longer applies, since variations across targets probabilities are not to be read only through the presence of noise, rather they can be caused by the mixture of different distributions. The typical behaviour of a network trained on such objective function — namely to *regress to the mean*, i.e. to average across training samples — is no longer appropriate, since the ground truth is no longer a single value, thus the expected value cannot be simply matched with the mean of the distribution. Nevertheless, a step of data exploration resulted in noticing that a large majority of the cases of multiple answers actually boil down to little to no differences in the samples. In other words, quite often it is the case that answer text spans almost (if not fully) coincide, maybe differing for a few words. This justifies our choice to keep these samples for the purposes of training, since the target distributions of the samples nearly coincide, thus allowing for a good approximation still keeping the univariate distribution assumption.

1.2.2 Model

The model is a feed-forward Artificial Neural Network (ANN) and relies upon the pre-trained Transformers-based language model BERT [Devlin et al., 2019], with a Fully-Connected (FC, also *Dense*) layer after it. BERT is an architecture based on Transformers [Vaswani et al., 2017] specifically trained to learn language representations and is intended to be used as the primary architecture for NLP tasks. In this framework, we initially chose BERT as an encoding block and train the FC layer as a Linear Classifier. We also introduced Dropout in between the two layers in order to improve generalization [G. E. Hinton et al., 2012]: after some experiments, the dropout rate was set to 0.5.

Taking into account the computational power we could afford, and striving for efficiency and portability of our model, we ended up deciding to use a lighter version of BERT, called DistilBERT [Sanh et al., 2020], which allows for faster deployment with respect to BERT. In particular, DistilBERT was pre-trained on the same corpus as BERT in a self-supervised fashion, using the BERT base model as a teacher exploiting the approach that goes under the name of Knowledge Distillation [G. Hinton, Vinyals, and Dean, 2015]. According to the authors of the reference paper, it is able to reduce the size of a BERT model by 40%, while retaining 97% of its language understanding capabilities and being 60% faster [Sanh et al., 2020]. In particular, DistilBERT has 66 millions parameters, as opposed to 110 million

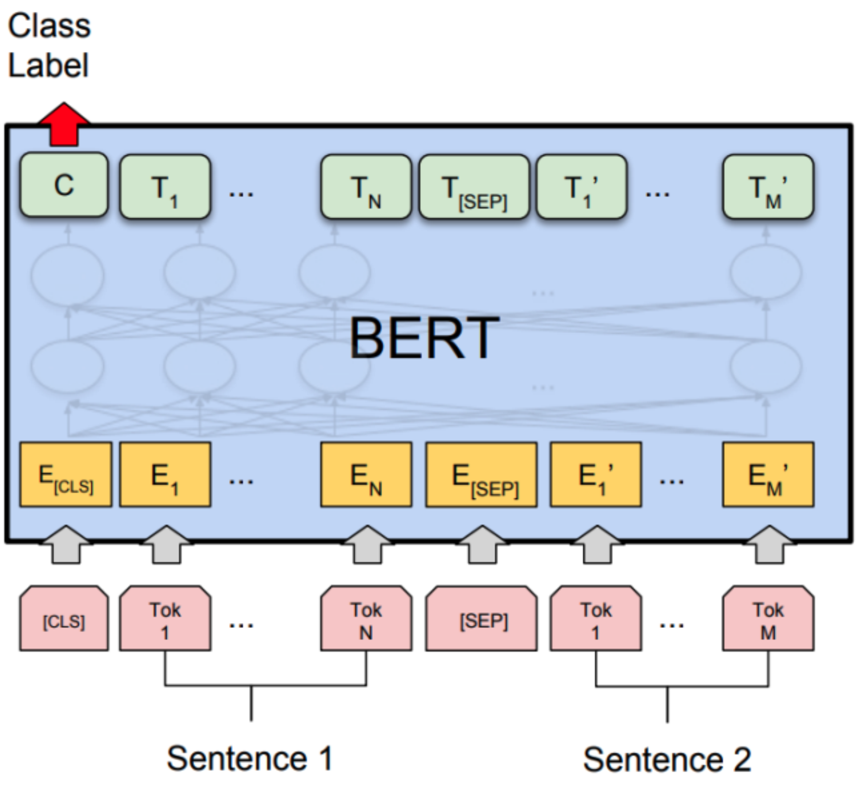

Figure 1.1: A graphical representation of the structure of the BERT architecture. DistilBERT differs from BERT only for the removal of token-type embeddings and the pooler, plus the halving of the number of layers.

parameters of the BERT architecture.

HuggingFace offers different versions of both DistilBERT and accordingly optimized tokenizers, already pretrained for multiple different tasks. We employed the general-purpose model fine-tuned on SQuAD v1.1 (`DistilBERTModel`) and used it along with the corresponding tokenizer (`DistilBertTokenizerFast`). In other words, tokenization and Transformers encoding are dataset-specific, but not task-specific.

As input, DistilBERT takes the tokenized samples (described in Section 1.2.1), whose shape is $(batch\_size, max\_length)$, where we used `batch_size=128` and `max_length=512`. Its output has shape $(batch\_size, max\_phrase\_length, hidden\_size)$, where we specified `hidden_size=768`.

The FC layer takes the output of the Transformers layer, after the dropout layer, and it is supposed to output a pair of values for each sample, namely the answer start and end positions. The shape of the output is thus $(batch\_size, max\_length, 2)$: the Classifier is basically expected to assign two values to each token referring to a sort of score of the answer start and end likelihood, which can be then normalized through a softmax activation to obtain valid probabilities.

In order to obtain the desired output, namely the answer text span, we firstly retrieve the answer start and end positions by taking the maximum of the probabilities over the second dimension (i.e., a pair for each sample). Next, we simply need to slice with these indices over the tokenized context, as to retrieve the list of tokens representing the answer, and apply decoding (*detokenization*) to finally obtain the actual answer as a piece of text.

1.2.3 Training

Since we framed the task as a multi-classification problem, we chose Categorical CrossEntropy as the objective function, as common practice. As already mentioned, training was performed on ∼80000 samples with a batch size of 128, thus requiring over 600 iterations per epoch. We run 100 epochs of training, saving a checkpoint of the model after 40 epochs, which implied a remarkable computational cost and required the deployment of GPUs to be bearable in terms of time and power consumption. We therefore relied on Google Colaboratory [*Google Colaboratory* 2017] to have quick and easy access to such resources, which allowed us to achieve an overall computation time of nearly 18 hours per run. We used the Adam optimizer with a learning rate of 10^{-3}.

1.2.4 Evaluation

To evaluate the performance of our model at *training time* we employed the **F1 score** (averaging on samples within batches) in addition to the loss function itself, as it is the main reference index for text evaluation. F1 combines Precision P $= \frac{TP}{TP+FP}$ and Recall R $= \frac{TP}{TP+FN}$ with a fair balance: is the harmonic mean of the two metrics, and it can be obtained as F1 $= 2 * \frac{P*R}{P+R} = \frac{TP}{TP+\frac{1}{2}(FP+FN)}$.

At the end of the training process we performed a global evaluation step to verify the performances of the model on the full test set all at once. We adopted several metrics to this purpose, both drawing from widely used metrics in the field of NPU and employing task-specific metrics:

- *Exact match*: a naive metric, it is simply the percentage of exact matches between target and corresponding predictions.
- *Precision*: the ratio of the number of words shared between prediction and target to the total number of words in the prediction.
- *Recall*: the ratio of the number of words shared between prediction and target to the total number of words in the target.
- *F1-score*: it is computed as the usual function of Precision and Recall, which take the particular above definitions when it comes to Question Answering tasks.
- *Jaccard Index*: This is computed as the intersection over union between predictions and targets, so we compute it as the length of the intersection divided by the length of the union of the two answers. This metric represents another kind of compromise with respect to the F1, because it focuses on penalizing the predictions containing way more context with respect to the target.

Monte Carlo Uncertainty Score We implemented a method to help us evaluate our approach and our predictions, based on an interpretation of Dropout as approximate Bayesian inference [Gal and Ghahramani, 2016]. As pointed out by the authors of the reference paper, class probabilities provided by the softmax output are sometimes misinterpreted as model confidence in classification problems. As shown in their work, though, a model's predictions might be imprecise even with a high softmax output. Since this method is based on *Monte Carlo* processes, and it provides a sort of the model's uncertainty score on its same predictions, we will refer to this metric as Monte Carlo Uncertainty (MCU). We implemented MCU in few simple steps: we set the model in evaluation mode while keeping the dropout layer active, and we performed the forward pass of the network feeding each input sample for a fixed number of times, eventually collecting all the different predictions. In the end we averaged the softmax scores of all the predictions for a single sample and computed the entropy value, which represents the uncertainty score of the network on that prediction.

1.3 Results

Our model shows good performance on our test split, exhibiting an improvement along the training. The evaluation on the test set shows that about the 73% of the total test set predictions (11557 out of 17653) reach a minimum Recall of 70%, meaning that predictions contain at least 70% of the actual answer. The network has an average Jaccard index of 0.696 on the test set.

Target	Generated	MCU
'Master of Divinity'	'Master of Divinity'	0.317
'Alliance for Catholic Education'	'Alliance for Catholic Education'	0.848
'1854 - '	'1854 – 1855 academic year'	1.447
'Department of Pre - Professional Studies'	'Department of Pre - Professional Studies'	0.702
'Joan B. Kroc Institute for International Peace Studies '	'Joan B. Kroc Institute for International Peace Studies '	0.964
'twice a year'	'[CLS]'	0.829

Table 1.1: **Examples**: we provide a few targets along with the corresponding model predictions. The [CLS] token represents an unanswerable question.

Observing the results we can notice that the model uncertainty does not always correlate well with the actual quality of the predictions. In particular, we witness the occurrence of both higher uncertainties in case of correct predictions, as well as lower uncertainties in correspondence of errors.

Model Name	F1 Score	Exact Match(%)
Our model (40 epochs)	75.53	70.43
Our model (100 epochs)	76.66	71.26
BERT	93.16	87.43
LUKE	95.71	90.62

Table 1.2: Performances: comparison of our model performance on the test set with some example from [PapersWithCode, 2021a]

All the models (apart from ours) are trained on SQuAD v.1.1. The state-of-the-art model included in this comparison is LUKE [Yamada et al., 2020], which is first pretrained using a new pretraining task based on the masked language model of BERT. We show also the performance of an ensemble BERT model trained for QA.

Comparing the performances we reached a reasonably good Exact Match score of 71.26%, meaning that the answers are quite often exactly the same as the targets, and a good F1 score of 76.66.

1.4 Limitations and Future Works

Although our approach reached a discrete level of performance, comparing it to others implementations that can be found online shows that there is much room for improvement. The motivations for this deficiency are firstly to be searched in the relatively simple setup we employed, namely an ANN composed of a Transformers architecture with a Classifier on top of it. Deeper networks could arguably reach more satisfactory results, even though they would require higher training times and more complex fine-tuning. Moreover, we were forced to a relatively small batch-size compared to the training set by the computational resource limits we deployed. Another possible source of limitation comes from the preprocessing step, which causes a personalized transformation of the original dataset following our particular setup. As an example, our elaboration of the dataset includes the concatenation of questions and contexts to produce the input to feed the network, which requires to truncate contexts at a determined length. Another example concerns how multiple answers were dealt with: as already mentioned in the paragraph *Issues* of section 1.2.1, we decided to keep them and to treat them as separate training samples, still in a univariate distribution assumption that exploits Categorical Crossentropy and its link with Negative Log-likelihood and the Least Squares Method. Following another strategy, it could be possible to define a different loss function according to a multivariate distribution, explicitly taking into account the possibility of a multi-valued output instead of forcing a regression to the mean. Moreover, the final dataset after the tokenization is cleaned to keep only the data useful for the training, leaving though some rows that have special characteristics, namely those involving context splits not including the provided answer, as a result of truncation. Those lines, presenting both the answer start and end set to 0, show a case that is not included into the SQuAD dataset v.1.1, namely the possibility of unanswerable questions. This particular situation is not studied by this report, mainly because of the negligible number of unanswerable questions, which causes an under-representation of the corresponding samples, likely causing poor learning and inference capabilities of the network when it comes to such samples, without specifically-oriented training.

A future work could explicitly include unanswerable questions in the dataset (which is envisaged by SQuAD v.2.0) and aim at training the network to also abstain from answering in such cases.

The biggest limitation of this project was the size of the employed network. To perform the encoding of the texts we decided to use a pre-trained version of DistilBERT, since training such a big network is out of our computational potentiality, moreover the computational power offered by Colaboratory (similar to the ones offered by the University clusters) isn't enough to train it within reasonable time periods. Besides, we had to limit the batch size to avoid memory leaks from the GPU. Among other things, this means that testing several different models was prohibitive, which greatly limited the possibility to fine-tune the hyperparameters. Hence, the availability of greater computational and graphical power could make training and fine-tuning our model more quick, easy and affordable.

Chapter 2

Question Generation

Question Generation (QG) concerns the field of Natural Language Generation (**NLG**), and can be defined as the task of "automatically generating questions from various inputs such as raw text, database, or semantic representation" [Rus, Cai, and Graesser, 2008] Education, dialogues/interactive question answering systems, search engines, and other sectors can all benefit from the automatic production of semantically well-formed questions from a given text. It is well-known as a difficult assignment that includes many of the same challenges as other natural language processing (NLP) operations.

2.1 Related Work

According to [Pan et al., 2019], QG can be mainly divided into two categories: *Answer-Aware* and *Answer-Unaware* (also Answer-Agnostic) Question Generation. Both names being self-explanatory, the former relies on data including the answers as labeled spans of text passages, while the second one does not. Focusing on Neural Question Generation (NQG), which aims at applying Deep Learning (DL) techniques to QG, the recent approaches include widely used NLP techniques such as Part-Of-Speech- (POS-) tagging and Named Entity Recognition (NER). Further distinctions can be made based on the text encoding methods, for instance *seq2seq* models (relying on RNNs) or Transformers (powered by self-attention layers).

2.2 Implementation

The goal of this work is the development of a model capable of generating a valid question, given a relevant paragraph (*context* and the corresponding answer (as a span of text within the paragraph). Since SQuAD provides carefully labelled answers as text spans within the respective context, we decided to exploit this information following an Answer-Aware approach.

2.2.1 Data preprocessing

To perform data preprocessing we mainly relied on the correspondent QA step (see Section 1.2.1). Still, some changes are needed, in particular:

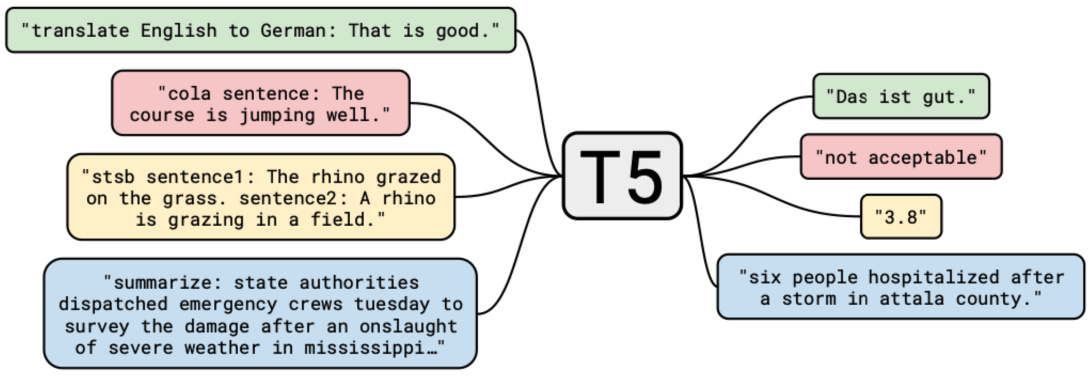
"translate English to German: That is good."
"cola sentence: The course is jumping well."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi…"
T5
"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."

Figure 2.1: A graphical representation of the T5 model, emphasizing its multi-task capabilities once fine-tuned on different downstream tasks.

- Tokenization was performed in a different fashion. In particular, we jointly tokenized (answer,context) pairs (which became our input data), and questions separately (our target data).
- We decided to keep only one answer for each (context,question) pair, in order to avoid confusing the model by providing different targets for the very same input sample.

2.2.2 Model

Following an Answer-Aware kind of approach, we implemented a ANN relying on the **Text-to-Text Transfer Transformer** (T5) model from Google [Raffel et al., 2020]. As outlined in the reference paper, the basic idea underlying the work is to apply Transfer Learning to several NLP tasks in a unified fashion, namely providing a task-agnostic pre-trained model to be then fine-tuned on downstream tasks. The approach involves treating all NLP tasks as "text-to-text" problems, i.e. taking text as input and producing new text as output, which allows for a direct application of the very same model, objective, training procedure, and decoding process to every considered task. According to the authors, this architecture relies on the original Transformers architecture [Vaswani et al., 2017], without significant deviations. Taking into account the computational power we could afford, we employed `t5-small`, which is a lighter version of the baseline implementation relying on 8-headed attention (instead of the original 12) and 6 layers each in the encoder and decoder architectures (as opposed to 12), for an overall size of 60 million parameters (with respect to 220). Following again the HuggingFace implementation [Wolf et al., 2020], we employed the model called `T5ForConditionalGeneration` along with the corresponding tokenizer (`T5TokenizerFast`).

It's an encoder decoder Transformer model, meaning that giving an input text it is supposed to learn to generate an output text, more precisely it will generate a question that is ideally suited, to the context and an answer.

The input of the T5 model is the output of the tokenizer, of dimension:

$$(batch\_size, max\_length)$$

while its output has dimension:

$$(batch\_size, max\_length, vocabulary\_size)$$

and represents — for each input sample — a 2D array with the chosen output dimension as row size, and each column representing a score of the corresponding position along the whole vocabulary. Basically, the result represents a sort of probability distribution ranging over the possible tokens for each index of the output. Hence, we take the maximum value over the third axis of the 3D output array, in order to extract a token prediction for each position (and for each sample): eventually, we transform the resulting sequence by mean of the tokenizer to decode (*detokenize*) it into the actual text span representing the question.

2.2.3 Training

Due to the native End-to-End structure of the T5 model, namely the direct production of (tokenized) text samples as model outputs, the Hugging Face implementation of the T5 model relies by default on the Categorical Crossentropy as a loss function — as common practice for Language Models — and directly produces the loss computation as result of the forward pass. Since the T5 model is quite large in memory size (∼230 MB), we set `batch_size=8` to avoid memory leakages on the GPU. However, this would lead to ∼10000 iteration steps, which would have been prohibitive in computational time. Hence, we decided to cut down the size of both the training and the test sets, which we achieved by keeping only one answer for each (context,question) pair. The approach lead to a reduction of ∼80% of both sets with respect to those used for QA, thus keeping the original test split ratio. We performed 6 epochs of training, saving a checkpoint of the model at 3 epochs in order to compare the relative performances and evaluate the quality of the learning process. We used the Adam optimizer with a learning rate of 10^{-3} for the first 3 epochs, reducing it to 10^{-4} for the last 3 ones.

2.2.4 Evaluation

The evaluation of the Question Generation results is more challenging with respect to Question Answering, mainly due to the generative nature of the former. In particular, a very well-known issue when it comes to evaluate NLG models is that many of such tasks are open-ended, meaning that the output of a model could be valid even if it completely differed from the original target used to train the model, despite using a supervised-learning approach. Therefore, human evaluation remains the gold standard for almost all NLG tasks to the present day [Celikyilmaz, Clark, and Gao, 2021]. Actually, regarding the particular task of QG we dare add that the capability of a model to formulate different — as long as valid — questions should be considered more as a strength than a weakness, since it would be an indicator of a more successful and complete learning. However, we decided to evaluate our results through the help of some common automatic metrics, mainly because of their ease of use and their popularity, which allowed us to draw a comparison between our model and other notable results. We performed quantitative analysis exploiting the following metrics, as they are widely used in many NLP fields, including QG [ibid.] :

- **BLEU** (BiLingual Evaluation Understudy) is an algorithm originally designed to evaluate the quality of machine translations from one natural language to another. According to the authors, it ”correlates highly with human evaluation, and that has little marginal cost per run” [Papineni et al., 2002], making it one of the reference metrics in the literature. However, it is worth pointing out that its algorithm doesn't take word ordering into account, and it rewards exact matches while penalizing every difference — albeit minor — across the output and the target. Moreover, since it considers neither semantic meaning nor sentence structure, BLEU has some drawbacks for NLG tasks where contextual understanding and reasoning is the key [Celikyilmaz, Clark, and Gao, 2021].

- **METEOR** (Metric for Evaluation of Translation with Explicit ORdering) is another automatic metric for machine translation evaluation based on a generalized concept of unigram matching between the machine-produced translation and human-produced reference translations [Banerjee and Lavie, 2005]. It was developed to address some of the issues found in BLEU, for instance introducing stemming and synonymy matching. Compared to BLEU, which only measures precision, METEOR is based on a weighted harmonic mean of the unigram precision and recall, in particular giving recall a greater relevance. According to the authors, METEOR reaches a higher correlation to human evaluation with respect to BLEU, where it should be emphasized that the METEOR evaluation is performed at a sentence- and segment-level, as opposed to the BLEU system-level evaluation [ibid.].

In order to assess the performance of our model, we established several models to carry out a comparison upon, namely: a baseline, a first improvement on the baseline and a state-of-the-art model. As baseline we chose the general-purpose T5 pre-trained model that we used as starting model, which is designed for conditional generation but not fine-tuned neither on the task of Question Generation nor on the SQuAD dataset: hence, we reasonably expect it to perform poorly on the given task. As a first improvement, we chose another T5-based approach, but specifically trained for QG on SQuAD (v.2.0) by the Allen Institute for AI [*Allen Institute for AI* 2021], and whose weights are available on the Hugging Face website. As state-of-the-art model we chose ERNIE-GENLARGE [Xiao et al., 2020], since it is known to reach top performance on several different tasks and datasets, and SQuAD v.1.1 in particular [PapersWithCode, 2021b].

2.3 Results

To generate the final questions we can tune some hyperparameters that affect the generation method, affecting the final predictions:

- `do_sample`: Whether to apply a stochastic process (e.g. random/top-K sampling) with respect to a deterministic one (Greedy search/Beam search). We decided to use Beam search, thus we set this parameter to `False`.
- `num_beams`: This parameter allows to choose the number of beams for Beam Search. Beam search reduces the risk of missing hidden high probability word sequences by keeping the most likely num_beams of hypotheses at each time step and eventually choosing the hypothesis that has the overall highest probability.

- `repetition_penalty`: A penalty used to decrease or increase the number of repeated generation [Keskar et al., 2019]. In particular, the proposed approach aims at balancing the trust in the model learned distribution through near-greedy sampling (as opposed to random sampling, which would highly under-weigh the model predictions), while preventing repetitions through a penalty.

Across our experiments, we obtained the best results by setting the generation hyperparameters `num_beams = 2` and `repetition_penalty = 10.5`.

2.3.1 Examples

We firstly show some example pairs of target questions along with those generated by our model.

`Target`	'What entity provides help with the management of time for new students at Notre Dame?'
`Generated`	'During the First Year of Studies program, what is one that provides time management and collaborative learning?'
`Answer`	'Learning Resource Center'

Table 2.1: **Example 1**: in this example the generated question is substantially different from the target one in terms of words, still being quite similar in terms of meaning

We can notice that generated questions are often distinguishable from real ones because of the not-so-natural language they use, even though the meaning is usually easily understandable.

`Target`	'How many BS level degrees are offered in the College of Engineering at Notre Dame?'
`Generated`	'A.S. degrees are offered in the College of Engineering?'
`Answer`	'eight'

Table 2.2: **Example 2**: in this case the question isn't well-posed for the corresponding answer, however it is relevant with respect to the context.

As already mentioned, the comparison between the generated and the target questions in these kind of task is sometimes not representative, especially because in many cases the generated question is different from the target in terms of words, though contextually relevant.

`Target`	'Where is the headquarters of the Congregation of the Holy Cross?'
`Generated`	'The official headquarters of the University is located in what city?'
`Answer`	Rome

Table 2.3: **Example 3**: this situation is quite interesting, since the generated question has a different meaning with respect to the target one, still representing a valid question not only with respect to the context, but also to the corresponding answer.

2.3.2 Performance

We now analyze the results based on the previously mentioned metrics, still recalling their well-established limitations in the evaluation of this kind of task.

Model Name	BLEU(%)	METEOR(%)
T5 not fine-tuned	1.06	45.8
AllenAI T5	3.97	54.1
Our fine-tuned T5 (3 epochs)	5.42	54.9
Our fine-tuned T5 (6 epochs)	12.53	60.3
ERNIE-GENLARGE	25.41	ND

Table 2.4: **Performance evaluation**: comparison of the various models performance on the test set employing the metrics BLEU and METEOR.

The state-of-the-art result is unattainable for us, at least with respect to the BLUE metric. According to the authors, it ”introduces a span-by-span generation flow that trains the model to predict semantically-complete spans consecutively rather than predicting word by word” [Xiao et al., 2020], which is likely its major strength. Nonetheless, our models show significant improvements with respect to the non-fine-tuned T5 model, comparing very well with the same architecture trained by the Allen Institute for AI. In particular, the first model (trained for 3 epochs) already reaches and slightly overcomes it, while the second one (trained for another 3 epochs) shows a further gain in performance, substantially improving on both metrics.

Our model is divided by number of training epochs, the first trial was done with 3 epochs of training and yet overtake the AllenAI model, moreover training it for 6 epochs make it be even better, with a BLEU score half the State-Of-The-Art result.

2.4 Limitations and Future Works

Limitations on our approach to Question Generation are similar to those for Question Answering. In particular, the evaluation of our results should take into account that we had to use the small version of T5, together with our necessity to constrain the training time by reducing the dataset size (see Section 2.2.3) as well as the batch size due to the memory limits of Colab's GPUs. Employing a larger model, a larger dataset and more powerful GPUs could arguably produce better results, along with allowing for several fine-tuning tests on different hyperparameters combinations.

Chapter 3

Conclusion

3.1 Conclusion

The use of a pretrained network to encode texts provides huge speedups for Natural Language Processing tasks. BERT is an important advancement for NLP, more in general Transformers are the game changer for these tasks both for their capability of learn from complex input texts and their ease of use within custom architectures.

Our results are encouraging but not optimal. As for the Question Answering task, we defined a quite simple model consisting of a Transformers architecture (DistilBERT) with a Fully-Connected layer on top of it: the first one was pretrained, while the second one was trained from scratch. Our model reached the following performances on our test set: F1-score of 76.66, Exact Match of 71.26, Jaccard index of 0.696. The state-of-the-art model outperforms our model, which reaches an F1-score of 95.71 and an Exact match of 90.62, but uses a lot more computational power and is bigger, which was impossible for us to replicate.

To address the Question Generation task we relied on an existing architecture namely the Text-to-Text Transfer Transformer (T5), which is a general-purpose pre-trained model to be fine-tuned on specific tasks. We reached very good performances on the chosen evaluation metrics, in particular we obtained a BLEU metric of 12.53 and a METEOR metric of 60.3, while the state-of-the-art model reaches a BLUE metric of 25.41.

References

Allen Institute for AI (2021). URL: https://allenai.org.

Banerjee, Satanjeev and Alon Lavie (June 2005). "METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments". In: *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization.* Ann Arbor, Michigan: Association for Computational Linguistics, pp. 65–72. URL: https://www.aclweb.org/anthology/W05-0909.

Calijorne Soares, Marco Antonio and Fernando Silva Parreiras (2020). "A literature review on question answering techniques, paradigms and systems". In: *Journal of King Saud University - Computer and Information Sciences* 32.6, pp. 635–646. ISSN: 1319-1578. DOI: https://doi.org/10.1016/j.jksuci.2018.08.005. URL: https://www.sciencedirect.com/science/article/pii/S131915781830082X.

Celikyilmaz, Asli, Elizabeth Clark, and Jianfeng Gao (2021). *Evaluation of Text Generation: A Survey.* arXiv: 2006.14799 [cs.CL].

Devlin, Jacob et al. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* arXiv: 1810.04805 [cs.CL].

Gal, Yarin and Zoubin Ghahramani (2016). *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning.* arXiv: 1506.02142 [stat.ML].

Google Colaboratory (2017). URL: https://colab.research.google.com/.

Harris, Charles R. et al. (Sept. 2020). "Array programming with NumPy". In: *Nature* 585.7825, pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean (2015). *Distilling the Knowledge in a Neural Network.* arXiv: 1503.02531 [stat.ML].

Hinton, Geoffrey E. et al. (2012). *Improving neural networks by preventing co-adaptation of feature detectors.* arXiv: 1207.0580 [cs.NE].

Keskar, Nitish Shirish et al. (2019). *CTRL: A Conditional Transformer Language Model for Controllable Generation.* arXiv: 1909.05858 [cs.CL].

Pan, Liangming et al. (2019). *Recent Advances in Neural Question Generation.* arXiv: 1905.08949 [cs.CL].

PapersWithCode (2021a). *Question Answering on SQuAD1.1.* URL: https://paperswithcode.com/sota/question-answering-on-squad11.

— (2021b). *Question Generation on SQuAD1.1.* URL: https://paperswithcode.com/sota/question-generation-on-squad11.

Papineni, Kishore et al. (2002). "BLEU: A Method for Automatic Evaluation of Machine Translation". In: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics.* ACL '02. Philadelphia, Pennsylvania: Association for Computational Linguistics, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://doi.org/10.3115/1073083.1073135.

Raffel, Colin et al. (2020). *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer.* arXiv: `1910.10683 [cs.LG]`.

Rajpurkar, Pranav, Robin Jia, and Percy Liang (2018). *Know What You Don't Know: Unanswerable Questions for SQuAD.* arXiv: `1806.03822 [cs.CL]`.

Rajpurkar, Pranav, Jian Zhang, et al. (2016). *SQuAD: 100,000+ Questions for Machine Comprehension of Text.* arXiv: `1606.05250 [cs.CL]`.

Reback, Jeff et al. (Feb. 2020). *pandas-dev/pandas: Pandas.* Version latest. DOI: `10.5281/zenodo.3509134`. URL: `https://doi.org/10.5281/zenodo.3509134`.

Rus, Vasile, Zhiqiang Cai, and Art Graesser (Jan. 2008). "Question Generation: Example of A Multi-year Evaluation Campaign". In.

Sanh, Victor et al. (2020). *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.* arXiv: `1910.01108 [cs.CL]`.

Sharma, Yashvardhan and Sahil Gupta (2018). "Deep Learning Approaches for Question Answering System". In: *Procedia Computer Science* 132. International Conference on Computational Intelligence and Data Science, pp. 785–794. ISSN: 1877-0509. DOI: `https://doi.org/10.1016/j.procs.2018.05.090`. URL: `https://www.sciencedirect.com/science/article/pii/S1877050918308226`.

Vaswani, Ashish et al. (2017). *Attention Is All You Need.* arXiv: `1706.03762 [cs.CL]`.

Wolf, Thomas et al. (2020). *HuggingFace's Transformers: State-of-the-art Natural Language Processing.* arXiv: `1910.03771 [cs.CL]`.

Xiao, Dongling et al. (2020). *ERNIE-GEN: An Enhanced Multi-Flow Pre-training and Fine-tuning Framework for Natural Language Generation.* arXiv: `2001.11314 [cs.CL]`.

Yamada, Ikuya et al. (2020). *LUKE: Deep Contextualized Entity Representations with Entity-aware Self-attention.* arXiv: `2010.01057 [cs.CL]`.

Yu, Adams Wei et al. (2018). *QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension.* arXiv: `1804.09541 [cs.CL]`.