Question Generation using Deep Neural Networks

Ângela Ferreira
angela.ferreira@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

May 2019

Abstract

Automatic question generation is the task of producing questions from a given text passage, with neural approaches currently achieving state-of-the-art results. In this work, the techniques applied to this challenge were studied, from the neural architectures to the different available datasets and evaluation metrics. Following the steps of previous related work, we developed an attention based, recurrent neural model that generates questions based on sentences. In addition, we also explored the Transformer, a novel neural network architecture whose implementation in Question Generation has not been reported to date. We experimented these models both with and without pre-trained word embeddings. Automatic evaluation shows that the usage of pre-trained word embeddings slightly reduced the performance of the solutions and that our models come very close to the state of the art. Human evaluators considered our questions correct, but not very relevant to the input sentences.
Keywords: Question generation, recurrent neural networks, attention, Transformer

1. Introduction

Questions can be used to request information with the intent of better understanding our surroundings and improving our knowledge. They can also serve as a way to check proficiency and test the comprehension of a given subject. The topic can become quite complex, especially if the intention is to apply it to machines.

To pose a question, the first important aspect of it is producing a correct natural language sequence that is in the interrogative form. Then, it should be related to a given topic. Whichever system is generating these questions needs to extract relevant information from a text and use it in the question. To ask interesting questions, they must be enhanced with new words or multiple pieces of information from different parts of the input sentence.

Question generation (QG) could benefit multiple areas of interest. For instance, it could be applied to interactive virtual agents with educational or entertainment goals [1, 8]. Frequently Asked Questions (FAQs) documentation could be created with more ease and less human input. QG could also support the task of Machine Comprehension, by synthetically developing extensive datasets with a considerable number of Question-Answer pairs that could be used as training data [17]. Alternatively, learning to ask good questions might be able to improve the quality of question answering models [23].

The QG task has achieved state of the art results with Deep Learning. Many researchers have been applying the neural techniques explored in Machine Translation to obtain an interrogative sequence from text segments. This has been possible thanks to the increasingly amount of Machine Comprehension data available with Sentence-Question pairs.

Appraising the quality of QG solutions is a hard task. The work executed in the area has not found a reliable evaluation method to check the quality and relevance of the resultant questions. In most of the literature, the results are reported with automatic metrics based on n-gram overlaps (such as BLEU). Human evaluation is sometimes used, but its subjective nature can also influence the results.

Our main objective with this work is to develop at least one QG solution that, given a set of natural language sequences, it returns correct natural language questions. The questions should relate to the source sequences. The QG system has to produce results that are close to or even surpass the state of the art and, they should be positively evaluated by humans.

To reach these goals, we will delve into deep learning models. We start by exploring previous similar methods, which are mostly based on Recurrent Neural Networks (RNN). We will also try a novel neural network architecture, the Transformer [21], which is known to achieve state of the art results in Machine Translation and that, as far as our knowledge goes,

has not been applied to the QG task yet.

The resultant questions will be later inspected so that the performance of the final solutions can be judged and compared against each other. We rely on automatic and human evaluation to achieve this comparison.

2. Related Work

Reading Comprehension (RC) can be described as the process of reading a text and acquiring enough information to be able to answer questions about it. Several specific **datasets** have been emerging to support this task, and some are briefly described in Table 1. It has been given special attention to those that had natural question-answer pairs ready to use.

The first free and publicly available RC dataset was the MCTest by Richardson et al. [15], that consisted of a set of 660 short stories and about 2,600 multiple choice questions. It was crowd sourced, and the writers were to write stories that young children could reason about. The main issue with this corpus is its size: too few stories of small size and too few questions.

One of the most relevant corpora used in RC is the Stanford Question-Answering Dataset (SQuAD) [13]. It is a crowd sourced dataset based on approximately 500 Wikipedia articles, generating more than 107,000 pairs of questions and answers. It presents a great variety of answers, with categories like Common Noun Phrases, Dates, Locations, Adjective Phrases, among others. This implies a high collection of questions in terms of syntactic structure and, consequentially, the amount of reasoning that it might take to produce the right answer.

Recently, a second version of SQuAD was released (SQuAD 2.0) [14], with the intent of overcoming certain weaknesses of the previous version. The improvement lies in the addition of more than 50,000 questions that do not have a correct answer in the paragraph but are closely related to its content. In these cases, it provides a list of plausible answers. This was expected to improve question-answering models by forcing them to understand that there is not any span of the text that can actually answer certain questions.

Microsoft Machine Reading Comprehension (MARCO) [11] was first released with 100,000 queries, which were real queries issued to the Bing search engine. The answers were generated by humans, based on the search engine's response. The second version of this dataset has 1,010,916 unique real queries, approximately 182,000 well-formed answers and about 35% of this dataset consists of questions without answer.

NewsQA [20] is based on more than 10.000 news articles from CNN. The questioners were to ask the questions based on the title of the article and its summary points. This dataset presents some questions that cannot be answered by the contents of the source article.

The NarrativeQA dataset [7] consists of 46,765 questions based on 1,567 fictional story summaries. Most question start with WH-. The answers to these questions might require several spans of the document, instead of focusing on just one sentence.

TriviaQA [6] is a corpus composed by more than 95,000 triples (Question, Answer, Evidence). The evidence consists in multiple text sequences that confirm the answer to the question, and it was extracted from a search engine: this means that it might not contain the exact accurate information needed to answer certain questions.

In terms of **QG research**, after the work of Rus et al. [16] which defined the text-to-question task, many approaches for the QG task have appeared over the past few years. Predominantly, rule-based approaches were used to transform declarative structures into an interrogative, but the performance of these models relies on the quality of those hand-crafted rules and templates, requiring expert linguistic skills.

The first to use neural networks to generate questions was Serban et al. [17], by producing a Question-Answer dataset with 30 million simple factoid questions, mapping the structured representations of the SimpleQuestions dataset, which consists of tuples with (subject, relationship, object), into natural language. The idea was to translate these tuples into a question composed by both the subject and relationship, while the object represented the answer.

Zhou et al. [25] uses SQuAD to conduct a preliminary study to generate a question from a text passage that contained the correspondent answer. It uses the sequence-to-sequence paradigm based on RNNs, improved with Attention mechanisms. The input was enriched with embeddings representing features, such as Part-of-Speech and Named-Entity-Recognition tags, as well as information about the answer position.

The model by Du et al. [3] uses similar neural techniques to generate natural questions, but it also explores the usage of an encoding of the full paragraph.

Yuan et al. [24] developed a model that had many similarities to the previous ones. It is notable for the introduction of Reinforcement Learning to the training process, in an attempt to tune the model by combining the negative log-likelihood with policy gradient optimization. There were multiple suggestions of possible rewards, such as BLEU.

Table 1: Comparison between Reading Comprehension datasets.

Corpus	Number of Questions	Text Source	Question Type
MCTest	2,640	Grade school level stories	Multiple choice
SQuAD	107,785	Wikipedia articles	Based on text segments
MS MARCO	1,010,916	Bing user queries	User queries and relevant passages
NewsQA	119,633	CNN news articles	Based on article title and summary
TriviaQA	95,956	Generated by trivia enthusiasts	Independent of text passages
NarrativeQA	46,765	Summaries of books and movies	Non-localized questions

3. Automatic Question Generation

Inspired by the increasingly usage of deep learning techniques for NLP, we built our question generation models using the same principles. As mentioned in previous chapters, the main goal for this project lies in developing a solution, making use of state-of-the-art neural techniques, that will allow the generation of questions based on a text sequence of arbitrary size.

Given as input a text sequence x, the model should output a question y, related to the content of the original text.

Both RNN and Transformer-based solutions were developed with Texar[1] [5], v0.1.0. The solutions described in the previous sections of this chapter were based on multiple modules available in this tool and, therefore, many of the hyperparameters available for each module were left with their default values.

The baseline was executed with the dependency parsing algorithm available in spaCy[2].

3.1. RNN-based Solutions

RNNs are widely used in sequence-to-sequence [19] prediction tasks, whose objective is, as the name suggests, to obtain a sequence of text based on another given as input, like in machine translation. These problems are usually modeled with the Encoder-Decoder architecture. It consists in having one or more RNNs working as the Encoder, encoding the input into a fixed-size vector representation. Then, the Decoder, which is also based on RNNs, uses these vectors to generate the output sequence. The gradients are propagated through the whole network. Encoder-Decoder resolves the issue of having input and output vectors of different domains with different dimensions.

The model receives a sentence and it is supposed to generate a question. The sentences passed to the model are tokenized and represented as a vector with the constant size equal to the longest sequence found in the data. Each word is replaced with an identification number and the rest of sentence is padded with zeros to keep the predefined size.

[1] https://texar.io
[2] https://spacy.io

The first layer constituting the Encoder of the network is the Word Embeddings layer. It is responsible for mapping the previously mentioned indexes into fixed sized embedding vectors. These vectors have a dimensionality of $d = 256$ and are randomly initialized. We later did another experiment where we later applied to this layer pre-trained word embeddings from GloVe. We chose the word embeddings based on the Common Crawl corpus, with 840 million tokens available, represented with vectors of 300 features.

Then, we have a Bidirectional Long Short-Term Memory (LSTM), so the input can be read both in forward and backward order. The Encoder receives as input the embedding vectors described above. It shares the same dimensionality. Both LSTMs have the same configurations. They have 128 layers and dropout was applied to the outputted hidden states of these layers. Together, the concatenation of both hidden states produces a vector.

With this setup, the Encoder collects information from the entire given sequence that will be propagated to the rest of the network. The Encoder produces a vector: a hidden state with elements from the input. The vector, which will be called the context vector from now on, encapsulates information from the original sentence. Attention makes the model focus on specific parts of the input when it is decoding, instead of looking at it as a whole.

The idea of alignment is crucial to understand attention. It consists in finding the parts of the input that are relevant to the current word in the output. Given the current hidden state of the decoder h_t and all the hidden states of the encoder h'_s, it is possible to obtain the normalized weights 1, which will influence the decoding process.

$$\alpha_t = \frac{\exp(\text{score}(h_t, h'_s))}{\sum_{s'=1}^{S} \exp(\text{score}(h_t, h'_s))} \tag{1}$$

The context vector c_t is computed at each decoding timestep, using the weighted sum of all the hidden states of the encoder h_s.

$$c_t = \sum \alpha_t h_s \tag{2}$$

Luong Attention [10] was used in this implementation. The decoder is initialized with the last hidden state of the encoder. Then, the output at the

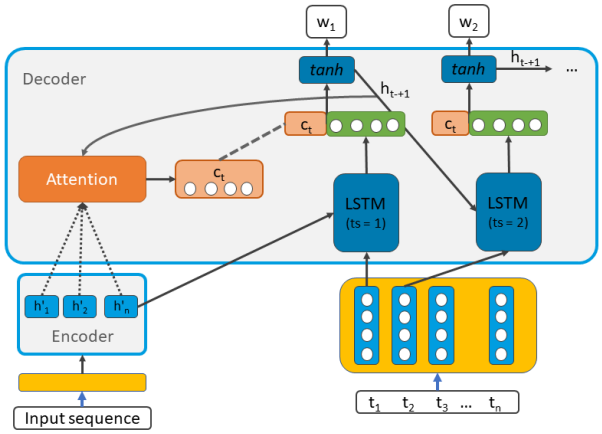

Figure 1: Representation of the RNN-based model.

current timestep is concatenated with the weighted context vector c_t. The result of this operation is then passed to a feed-forward neural network with a tanh activation 3. The neural network is trained with the rest of the parameters of the network. This will output a new hidden state and the word at each timestep.

$$h'_t = \tanh(W_c[c_t : h_t]) \tag{3}$$

We applied teacher forcing to our Decoder: instead of using the previous generated word, it uses at the target sequence in the train data at each timestep. The Decoder generates a state based on ground truth and the hidden state of the previous RNN layers. This allows the model to learn the correct sentence, instead of being penalized by its own mistakes. The result will be a vector with the size of the vocabulary and each position is filled with the probability of a certain word fitting in that part of the sentence.

The learning process of this model is divided in two phases: training and evaluation time. The dataset is divided in small batches of size $b = 10$, that are used to optimize the loss function and update the models weights. It computes the softmax cross entropy. It uses the Adam optimizer with default settings and learning rate of 0.001.

During the training part of the learning process, it uses a greedy strategy to select the output words, which means that it will choose the word with highest probability. At the end of each epoch, it enters in the evaluation phase. The data is switched to a validation set. Here, it uses beam search to produce the outputs. Given a certain beam size $bs = 10$, the model keeps track of that many best options. This allows the model to recover from any possible mistakes during word generation.

The produced questions will be compared with the target ones, using the automatic evaluation metric BLEU, which measures word overlaps. Throughout the learning process, the highest BLEU score is saved, and when one epoch manages to update the model in a way that this evaluation step returns a higher score, the model is updated with these new weights. Otherwise, the results obtained during this epoch are ignored. Either way, the training process repeats.

In our research, we used early stopping in our training. When the model stops learning for about 7 epochs, we stop the training and consider it over. The training process went on for 31 epochs, and the best BLEU score obtained during evaluation was 12.06. When we used pre-trained word embeddings, the model ran for 37 epochs, and the best BLEU score was 12.3.

3.2. Transformer Solutions

The Transformer model was proposed by Vaswani et al. [21]. It is a parallelized solution that focuses on attention and it is meant to speed up the training process of sequence-to-sequence tasks.

It also consists of an Encoder-Decoder paradigm, where these layers consist on 6 versions of the same sub-layers, stacked on top of each other, but without sharing weights. Both Encoder and Decoder are very similar to each other. They contain a self-attention layer, followed by a feed-forward neural network.

The model receives the input and output in the form of word embeddings. We will be exploring the case where they are initialized randomly and when they are initialized with the same pre-trained word embeddings described in the previous model (GloVe, 300 features).

To keep information about the location of each word in the input, it uses position encodings to do so. They are added to the input embeddings that will be passed to the first encoder and decoder layers. The embeddings are fed to the first Encoder layer. Each of the words in the input sentence is individually given to the self-attention layer. The encoding of each position will have information from the rest of the input. In this case, we used a dimensionality of 256 in these layers. It uses a Dropout rate of 10% on the input word and position embeddings.

Self-Attention is first introduced in this context and it is supposed to attend to important words in the input sequence while encoding a certain position in the sentence. The inputs of the Self-Attention layer are read "side-by-side", instead of being sequentially done like with RNNs. It makes use of three vectors, with dimensionality $d = 64$, for every word: Query, Key and Value. They are obtained by multiplying the embedding vectors with trainable matrices. These vectors are then used to calculate the score that will weight the input sequence.

The score is obtained with the dot product of the

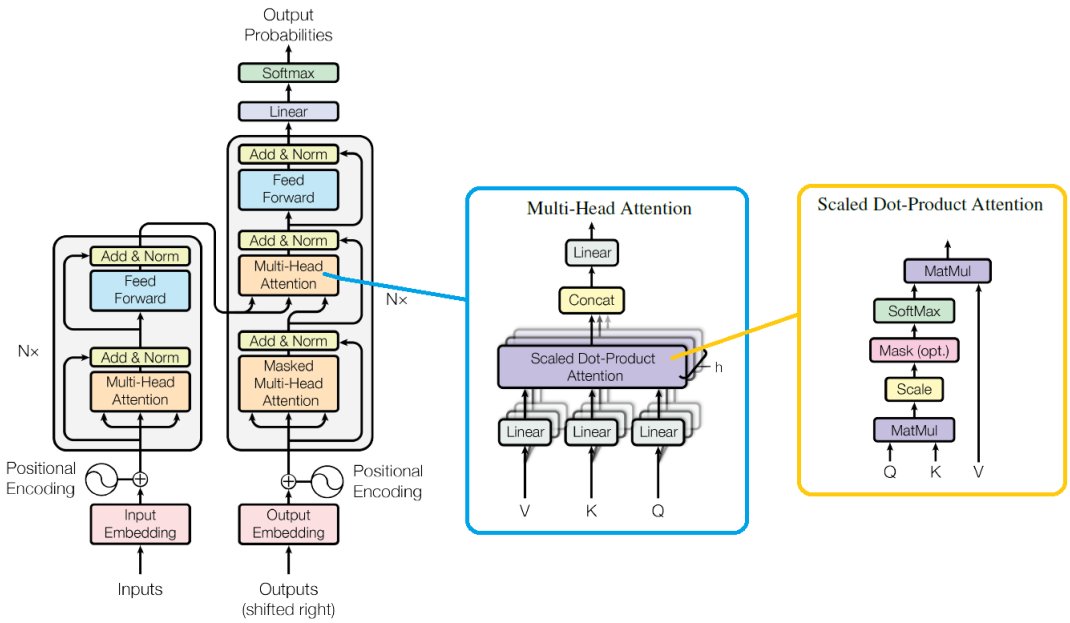

Figure 2: The Transformer model, as presented in [21]

Query and Key vectors and then normalized with softmax 4.

$$\text{attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \quad (4)$$

One important aspect of this architecture is what is called Multi-Headed Attention, where the Self-Attention sub-layer should be. This means that the model keeps track of more than one set of the matrices mentioned previously (in the original paper, $h = 8$ times the Keys, Values and Queries matrices). These are initialized randomly and then trained independently from each other. The Attention is calculated for all those "heads", who are then concatenated and scored to give origin to the vector that is given to the next layer. In equation 5, $head_i$ is computed with 4. W^O is also trained simultaneously with the network.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, ..., head_h)W^O \quad (5)$$

Finally, the Multi-Headed Self-Attention sub-layer returns the sum of the weighted value vectors, which is fed to the next sub-layer: a feed-forward neural network (FFN) with ReLU activation 6. Every position in the input goes individually through this FFN.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (6)$$

The output of each FFN is used on the next encoder layer, until it reaches the final one in the stack.

The Decoder has almost the same structure of the Encoder. Each layer possesses the Multi-Headed Attention and the FFN sub-layers. Then, it has an intermediate layer between these two, which serves as the Attention layer that allows the Decoder to focus on specific parts of the input when decoding. The Multi-Headed Attention sub-layer is masked, so that the Self-Attention cannot attend to future positions in the sequence, looking just at what was already outputted.

In the decoding phase, the output of the uppermost Encoder layer is given to this intermediate Attention sub-layer in every Decoder layer. The main difference between the bottom and mid sub-layers is that the one in the middle uses the values received from the Encoder in the Key and Value matrices. Bottom-up, the Decoder output is passed to the next layer, until it reaches the top and generates an output word. The Decoder never attends to words that appear after the current one.

The output of the final Decoder layer is passed to a Linear layer, which converts that outputted vector into another with the size of the considered vocabulary. With a softmax layer, it turns the values into probabilities of the most probable word for that location. We use beam search, with beam size equal to 5.

In Figure 2, it shows the full Transformer architecture, as shown in the original paper. Every sub-layer is followed by a layer normalization step.

In terms of the used Optimizer, we followed the original choice by Vaswani et al. [21]: Adam, with learning rate scheduling, to variate during the training. The learning rate will increase linearly for the first warm up steps and then decrease them.

The Transformer model, with and without pre-trained word embeddings, stopped training at the 20th epoch.

3.3. Baseline

We will use dependency parsing to generate natural language questions: it starts with analysing the grammatical structure of an input sequence, obtaining the relationships between the words composing a sentence. The idea is to obtain the root verb of a sequence and its respective subjects to fill in the blanks of predefined question structures:

- What **ROOT SUBJECT** ?
- What does **SUBJECT ROOT-LEMMA** ?

This solution is quite simple and it produces immediate results, by not requiring training. While it is guaranteed to obtain questions, it is easy to predict that it will come with a few issues on its own. For once, there is not a large variety of questions that this model can generate, as every question sequence will start with "*What*".

On the other hand, this model is not capable of generating questions based on broken, unreliable input. If a sentence is incomplete or even incorrect, it is not be able to parse the input in order to obtain dependencies. There is also an issue with complex sentences, where the automatic extraction of features might result in uninteresting or even confusing questions.

4. Experimental Evaluation

It is our intention to explore the questions that each model, described in section 3, generated and determine which provides the best ones. However, it is not easy to reach such conclusions because there is no right formula to evaluate the quality of questions.

It is important to define what constitutes a good question, especially in the context of this project. First, it is expected that the question is well-formed. A question can be classified as well-formed if it is grammatical, explicit and if it does not contain spelling errors [4]. We can follow these guidelines when evaluating our questions, but we are also interested in having them relate somehow to the sentences they were based on. Finally, we should take into consideration the original questions associated to each specific sentence and see how far from these are the ones obtained.

To inspect the quality of these solutions, we will apply automatic measures, widely used to evaluate various NLP tasks, and human evaluation.

Table 2: Percentage of generated questions, based on input.

System Name	Generated Questions
Dependency Parsing	49.89 %
RNN	100.00 %
RNN-GloVe	100.00 %
Transformer	84.42 %
Transformer-GloVe	88.35 %

4.1. Dataset

The SQuAD 2.0 was chosen for the development of our question generation models. For this specific job, however, we start by discarding the unanswerable questions, since we intend to create questions that are related to a given text.

The pre-processing of our corpus starts by separating paragraphs into sentences. This was executed using the Punkt Sentence Tokenizer for English sentences, from NLTK Data [3]. For this research we will consider all the resultant sentences, even if the automatic sentence tokenizer fails to capture a complete and correct sentence from the paragraph.

We produced sentence-question pairs by locating the sentence that had the answer. Then, punctuation is padded with whitespaces and all the words are lowercased. All text sequences in the set were surrounded by start and end of segment tokens.

Thus, the data used to train the question generation models has the following specifications:

A train set with 86,820 answered questions. The longest sentence is 325 tokens long, while the longest question has 42 tokens. The vocabulary on the target side (size = 36,732) is different from the source side (size = 69,243) and a total of 73,364 unique tokens. Out of these, 18,587 tokens do not appear in the GloVe pre-trained word embeddings. This is 25.33% of the tokens that had to be initialized randomly in the models that were enhanced with pre-trained word embeddings.

The test set has 5,927 answered questions, and its longest sentence is 163 words long, while the length of the longest question is 33.

4.2. Results Overview

Table 2 shows the percentage of questions generated, given the input sentences of the test set. Dependency Parsing, Transformer and Transformer-GloVe models did not generate the expected number of questions, with Dependency Parsing being the one producing the least number of questions.

In Table 3, the second column shows the value of unique sentences that appear more than once

[3] https://kite.com/python/docs/nltk.tokenize.punkt

Table 3: Number of questions and sentences repetitions in the data used and produced by the model.

System Names	Repeated Sentences	Sentence Repetitions	Repeated Questions	Question Repetitions
Dependency Parsing	21.95 %	30.71 %	21.75 %	40.45%
RNN	22.10 %	31.20 %	22.12 %	31.38 %
RNN-GloVe	22.10 %	31.20 %	22.10 %	31.40 %
Transformer	22.20 %	31.61 %	16.92 %	67.71 %
Transformer-GloVe	22.82 %	31.50 %	14.49 %	74.03 %

in data (and, in this case, they can be associated to different target questions). The fourth column shows questions that appear at least twice in the output of our models. The third and fifth column show the number of lines in the data (sentences and questions, respectively) that are repetitions. The sentences that did not generate questions are not considered in this table. The columns related to the sentences show quite consistent values between models, which makes sense because they are considering the same input sentences.

The Dependency Parsing and both RNN models show similar values in the "Repeated" columns. It is possible to infer that the model will behave consistently and produce the same question whenever it is fed with the same input sequence. This behaviour is further emphasised for RNN and RNN-GloVe in the "Repetitions" columns, as both columns show similar values. Here, the Dependency parser model reveals to have more repetitions. This happens because, sometimes, the parser chooses the word there as subject of the root and generates questions like What is there? for very different input sequences.

In the Transformer cases, we can assume that they are having a more erratic behaviour. In the "Repeated" columns of Table 3, the fact that we have a smaller value in the number of unique questions, in comparison to the number of sentences, seems to indicate that when a text is given to the model, it will not generate the same sentence every time. This is not necessarily a bad thing, as it could be an indication of variety. However, the "Question Repetitions" column shows that more than half of the results are repetitions of the same questions. Both Transformer models generated many questions that are a variation of what was the name of or what is the term for, sometimes followed by an apparently random term.

4.3. Automatic Evaluation

Automatic evaluation metrics are widely used in NLG tasks since these metrics are easy to compute and with a larger degree of availability. The most commonly used metrics are based on n-gram overlap, and they were originally meant to evaluate the performance of machine translation systems. However, they cannot capture semantic diversity.

The metrics reported in this section, Table 4, were obtained using the nlg-eval evaluation package by [18]. It includes BLEU [12], METEOR [2], ROUGE-L [9] and CIDER [22].

First, the BLEU score is reported from BLEU-1 to BLEU-4. It is quite visible that the baseline model is the one who returns the worst values, which are barely higher than 0.

All other models are very close to each other. The difference between RNN models is practically nonexistent, but it seems that the model suffers with the usage of pre-trained word embeddings.

The same goes for the Transformer models, but the distance between them is a little more noticeable. The model that does not use pre-trained word embeddings returns a slightly higher score.

It seems that the models based on RNNs perform better than when using the Transformer, showing a small decrease in the score. In line with these results, it seems that the RNN (no GloVe) is the best out of every question generation solution. It is important to emphasize that BLEU is not a precise measurement, and that this small differences in scores are inconclusive. .

The scores obtained for METEOR, ROUGE-L and CIDER show the baseline being the one coming way behind every solution, with values close to 5 for all the metrics, indication of its poor behaviour.

METEOR presents scores that resemble the BLEU results, with the RNN-based models having similar values, and the Transformer ones also being quite identical between them. In both cases, the best scores are obtained without using pre-trained word embeddings. Between the two options, it seems that the question generation process performs better when using the RNN models, but the discrepancy is not that relevant, as the difference is smaller than 1. We reach the same conclusions by analysing ROUGE-L scores. Here, the difference between the Transformer and RNN models is even smaller. In both metrics, the RNN has the best scores.

So far, the results seem consistent, but also very uncertain.s. Yet, when we look at CIDER, we see a significant drop from the RNN-based solutions to the Transformer ones as their scores are less than

Table 4: Automatic evaluation results obtained by the models, reporting BLEU [1-4], METEOR, ROUGE-L and CIDER.

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	METEOR	ROUGE-L	CIDER
Dependency Parse	0.8909	0.1384	0	0	4.5262	5.2213	5.1739
RNN	39.3362	25.268	13.1763	7.9029	29.3371	40.4695	43.0177
RNN-GloVe	39.1382	24.9798	13.0097	7.8027	29.2349	40.1844	41.9173
Transformer	38.8333	24.6561	12.0306	7.0544	28.5332	40.0584	18.7706
Transformer-GloVe	37.7911	23.829	11.3757	6.5936	28.1891	39.6555	15.3649

half of the first ones. This result is not surprising: CIDER is heavily based on TF-IDF, and it will favour uncommon n-grams that match the reference question, while reducing the importance of frequent ones. This makes sense, since the Transformers generate many questions that are the repetition of the same sequence (i.e. What was the term for). We had also seen that most of the output were repetitions of a certain set of questions. This could be influencing the weighing of n-grams in the data, which justify this variation.

4.4. Human Evaluation

Given the inconclusive results obtained by the usage of automatic metrics, we perform human evaluation on the generated questions. Given that we are trying to assert which of these models is more suitable to this task, we asked 5 fluent English speakers to observe our questions.

We chose 20 sentences from the test set to develop the survey. These sentences were selected so that every model had generated a question using it as input. The model questions (from RNN, RNN-GloVe, Transformer and Transformer-GloVe) and reference question were randomly shown to the user, so that the users could not find a relation between the order. The survey was executed on Google Forms.

The users are informed about the idea of correctness and relevance in the context of our research. For each question, the user must read and consider if it is correct or relevant, regarding the sentence. They must order every question by personal preference, where the first is the most suitable in the terms of this research.

In Figure 3, the second column presents the number of times the questions were evaluated as correct. The fourth shows the number of times a model was considered relevant.

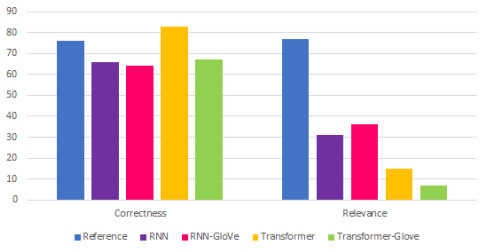

Figure 3: Percentage of correct and relevant answers, as selected by users.

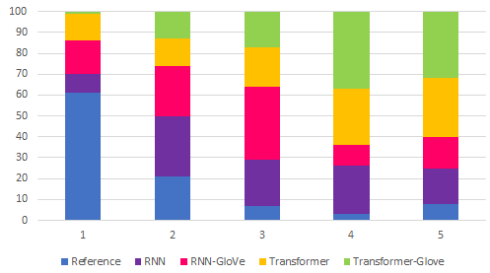

Figure 4: Amount of preferred questions, where 1is considered the best and 5 the worst

It is noticeable that the correctness of all the models is above 50%, which means that most of the time, users think these questions are correct. However, in terms of relevance, the results are very low. The human-generated questions achieved the highest relevance by a large margin value.

The RNN model has a slightly better correctness than RNN-GloVe, but once again the differences are just by a small margin. In respect to relevance, it achieves a better result than both Transformer models.

While RNN-GloVe has the lowest level of correctness, it achieves the highest value of relevance out of all the solutions.

The Transformer questions are the ones that are considered the most correct, surpassing even the Human questions. Relevance, on the other hand, drops to 15%, which reinforce the idea of good random questions that was already concluded in the previous section. The level of correctness in Transformer-GloVe drops to similar values of the RNN-based solutions. Relevance is very low.

This analysis proves that, while models seem very similar when passing through automatic evaluation, they show differences for the human eye. The questions produced by RNN-based solutions seem less correct than the Transformer, but more similar to the given sentences. The Transformer models generate correct, but irrelevant questions.

Regarding preference, in Figure 4, the gold question is the one that is preferred most of the time. It

is placed the least amount of times in the bottom three classification places (18%).

Out of the solutions, GloVe is the one that is classified most of the times with the best question. The RNN-based solutions are both selected approximately 40% of the time as the two top-most questions. The Transformers are selected as top 2 question 26%.

The model that is picked as the worst question most of the times is Transformer-GloVe, that is chosen for two bottom classification values 69%. There is only one instance where one user selected the Transformer-GloVe as the best question. Next comes the Transformer, showing more positive evaluations, but not that higher.

It appears that, even though users considered the Transformer model the most correct, they were the least preferred. Both RNN-based solutions scored similarly, in this case with RNN-GloVe being better classified most of the times: this could also be related with the weight that users put on relevance.

4.5. Discussion

After analysing the results, it is difficult to choose the best model. In terms of automatic evaluation, the results are very similar to each other. Even though they could not achieve the same level of correctness or relevance that the reference questions did, they were still positively evaluated by humans.

Both solutions saw a decrease of quality when using the pre-trained word embeddings by GloVe. This is something that goes against what would be expected. However, we saw that 25% of the tokens were initialized randomly, since there was no corresponding word embedding. One can conclude that such amount of unknown tokens can hurt the performance, and that the number of matches has to be higher for these embeddings to be useful.

Then, we concluded that the Transformer models were incapable of generating one question for every sentence in the data. They are also repetitive and the questions were not related with the source sentence. Even so, the questions were considered correct by human evaluators and the automatic evaluation placed this solution higher than the baseline.

5. Conclusions

In this work, neural techniques were used to generate natural language questions from a given text sequence as input. The studied models comprising RNNs with attention mechanisms, like the state-of-the-art of this specific task, and the Transformer architecture were able to generate distinct questions from the ones present in the training dataset.

After automatic and human evaluation, the direct comparison between RNNs and the Transformer is not conclusive given the large number of questions generated by both models that often did not share words with the reference questions, hence leading to low BLEU scores. Manual inspection of the results showed several problems in terms of textual coherence (e.g., generations with repeated words), but nonetheless many of the questions that were generated were correct, relevant and even interesting

Nonetheless, many of the questions that were generated were correct, relevant and even interesting. The automatic evaluations show results close to the state of the art in QG and human evaluators gave a positive score in terms of correctness and preference.

These evaluation results show that the Transformer architecture can be a promising alternative to the state-of-the-art models used in question generation and the further development of its implementation could lead to very accurate question generation tools in the future.

With further efforts over this introductory work on QG with neural networks, we believe that there is still great room for performance improvement of these models that could be achieved, for instance, by using other approaches for pre-trained word embeddings of unknown tokens. Instead of initializing them randomly, maybe using a similarity function can help achieve better results. Other possibility is experimenting other Word Embeddings, like BERT. Finally, we suggest tuning the hyperparameters that were left with the default values.

References

[1] S. Curto, A. Mendes, and L. Coheur. Question generation based on lexico-syntactic patterns learned from the web. 01 2015.

[2] M. Denkowski and A. Lavie. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the EACL 2014 Workshop on Statistical Machine Translation*, 2014.

[3] X. Du, J. Shao, and C. Cardie. Learning to ask: Neural question generation for reading comprehension. *CoRR*, abs/1705.00106, 2017. URL `http://arxiv.org/abs/1705.00106`.

[4] M. Faruqui and D. Das. Identifying well-formed natural language questions. *arXiv preprint arXiv:1808.09419*, 2018.

[5] Z. Hu, H. Shi, Z. Yang, B. Tan, T. Zhao, J. He, W. Wang, L. Qin, D. Wang, et al. Texar: A modularized, versatile, and extensible toolkit for text generation. *arXiv preprint arXiv:1809.00794*, 2018.

[6] M. Joshi, E. Choi, D. S. Weld, and L. Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading compre-

hension. *CoRR*, abs/1705.03551, 2017. URL http://arxiv.org/abs/1705.03551.

[7] T. Koisk, J. Schwarz, P. Blunsom, C. Dyer, K. M. Hermann, G. Melis, and E. Grefenstette. The NarrativeQA reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 2018. URL https://TBD.

[8] H. Kunichika, T. Katayama, T. Hirashima, and A. Takeuchi. Automated question generation methods for intelligent english learning systems and its evaluation. 01 2004.

[9] C.-Y. Lin. Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*, 2004.

[10] M. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015. URL http://arxiv.org/abs/1508.04025.

[11] T. Nguyen, M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, and L. Deng. MS MARCO: A human generated machine reading comprehension dataset. *CoRR*, abs/1611.09268, 2016. URL http://arxiv.org/abs/1611.09268.

[12] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL https://doi.org/10.3115/1073083.1073135.

[13] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. SQuAD: 100.000+ questions for maching comprehension of text. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016. URL https://aclweb.org/anthology/D16-1264.

[14] P. Rajpurkar, R. Jia, and P. Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018.

[15] M. Richardson, C. J. Burges, and E. Renshaw. MCTest: A challenge dataset for the open-domain machine comprehension of text. *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

[16] V. Rus, B. Wyse, P. Piwek, M. Lintean, S. Stoyanchev, and C. Moldovan. The first question generation shared task evaluation challenge. 2010.

[17] I. V. Serban, A. García-Durán, Ç. Gülçehre, S. Ahn, S. Chandar, A. C. Courville, and Y. Bengio. Generating factoid questions with recurrent neural networks: The 30m factoid question-answer corpus. *CoRR*, abs/1603.06807, 2016. URL http://arxiv.org/abs/1603.06807.

[18] S. Sharma, L. El Asri, H. Schulz, and J. Zumer. Relevance of unsupervised metrics in task-oriented dialogue for evaluating natural language generation. *CoRR*, abs/1706.09799, 2017. URL http://arxiv.org/abs/1706.09799.

[19] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014. URL http://arxiv.org/abs/1409.3215.

[20] A. Trischler, T. Wang, X. Yuan, J. Harris, A. Sordoni, P. Bachman, and K. Suleman. Newsqa: A machine comprehension dataset. *CoRR*, abs/1611.09830, 2016. URL http://arxiv.org/abs/1611.09830.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

[22] R. Vedantam, C. L. Zitnick, and D. Parikh. Cider: Consensus-based image description evaluation. *CoRR*, abs/1411.5726, 2014. URL http://arxiv.org/abs/1411.5726.

[23] T. Wang, X. Yuan, and A. Trischler. A joint model for question answering and question generation. *CoRR*, abs/1706.01450, 2017. URL http://arxiv.org/abs/1706.01450.

[24] X. Yuan, T. Wang, Ç. Gülçehre, A. Sordoni, P. Bachman, S. Subramanian, S. Zhang, and A. Trischler. Machine comprehension by text-to-text neural question generation. *CoRR*, abs/1705.02012, 2017. URL http://arxiv.org/abs/1705.02012.

[25] Q. Zhou, N. Yang, F. Wei, C. Tan, H. Bao, and M. Zhou. Neural question generation from text: A preliminary study. *CoRR*, abs/1704.01792, 2017. URL http://arxiv.org/abs/1704.01792.