Generating Adversarial Questions for QA Systems

Mitesh Kumar Singh
SBU ID: 112078427
miksingh@

FNU Udbhav
SBU ID: 112006916
uudbhav@

Nikhil Siddhartha
SBU ID: 112126894
snikhil@

Abstract

Creation of training datasets for QA systems requires substantial manual effort. The process to create such large scale good quality datasets is cumbersome and time consuming. In this paper, we describe a framework for automatically generating QA datasets from free text. We also propose an adversarial question generation framework which generates questions on which a given QA system will fail. Using our framework, we generate a set of 2523 good quality questions involving paraphrasing and counting and having answers spanned across multiple sentences. Our adversarial framework removes easily answerable questions by classifying these questions as answerable or unanswerable (with an accuracy of 65%). We hope that training an existing QA model on adversarially generated questions will boost it's performance.

1 Introduction

Deep learning models are known to perform well when tested on data similar to their training data. When trained on large datasets, these models learn all the patterns in the data. For instance, QA systems, trained on SQuAD (Rajpurkar et al., 2016), often perform very well on factoid questions. Recently released models like BERT(Devlin et al., 2018a) and XLNet(Yang et al., 2019) have even surpassed human performance on such datasets.

However, most QA systems fail miserably when tested on datasets even slightly different than what they are trained on. In order to improve QA systems, we need to train them on diverse datasets. Creating such datasets manually is time taking and cumbersome. Thus, there's a need to automatically generate good quality questions for training QA systems.

There have been some efforts in the direction of automatic generation of questions to improve QA systems. (Duan et al., 2017) proposed a question generation system where they have adopted both retrieval based and generative question generation approaches to generate questions similar to frequently-asked questions from the given passages. Then these generated questions are used to improve a QA System. In a similar effort, (Yuan et al., 2017) proposed a recurrent neural model that generates natural-language questions from documents, conditioned on answers. Recently, (Dong et al., 2019) have proposed a UNIfied pre-trained Language Model (UNILM) which achieves the new state-of-the-art on five natural language generation datasets. Given a passage and a hint, it generates good quality questions of several different types. Table 5 shows the example questions generated by it.

As we observe, there have been multiple efforts present in the domain of automatic question generation, UNILM being the most recent and best performing of all of them. There have been efforts of domain specific QA dataset generation such as in (Su et al., 2016) and semi-automated QA dataset generation such as (Pampari et al., 2018).

The contribution of our work is two-fold. Firstly, we have built an end-to-end fully automated system to generate good quality questions that is agnostic of any domain knowledge. We demonstrate an instance of our QA dataset generation framework in generating a large-scale QA dataset by generating 4930 *(paragraph, question, answer)* triplets. Secondly, we have built an adversarial question-generation system that can understand the types of questions the QA system can answer and then specifically present the QA system with questions that are difficult for it to answer. We expect that a QA system should benefit by this additional set of questions as it can be trained with this additional questions in order to improve its overall performance.

We have tested our hypothesis of improving the performance of a QA system by training SAR (Chen et al., 2017) with our additionally generated questions on SQuAD 1.1 dataset. In one experiment, the additional set of difficult questions improved the SAR's exact match accuracy from 67.25 to 68.01 however we couldn't reproduce it consistently. This shows that our idea of training a QA system with difficult questions is helpful for improving its performance, but to completely validate our idea, more experiments with multiple QA systems such as BiDAF, BERT are needed.

The main outcomes of this project are:

1. An automated end-to-end, domain agnostic, grammatically accurate question generation system.

2. An adversarial system which can understand the types of scenarios where a QA system will fail.

3. We contribute a dataset of 2523 tough questions that is generated using SQuAD 1.1 paragraphs

2 Our Task

Given the training, validation data and results of a QA system (say X) on them, our goal is to build a framework that can automatically generate a set of new *(paragraph, question, answer)* triplets on which the QA system will fail. This involves solving 2 key challenges

- How to automatically generate a new dataset (say D) i.e. *(paragraph, question, answer)* triplets?

- How to identify QA pairs in D on which X will fail?

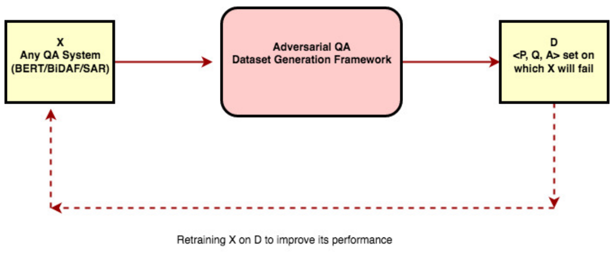
X
Any QA System
(BERT/BiDAF/SAR)
Adversarial QA
Dataset Generation Framework
D
<P, Q, A> set on which X will fail
Retraining X on D to improve its performance

Figure 1: QA Dataset Generation Framework

We solve the first problem by proposing a QA Dataset Generation Framework which generates good quality questions. To the best of our knowledge, there are no existing end to end automated QA dataset generation frameworks. We consider QA Dataset Generation Framework as the baseline system and Adversarial QA Dataset Generation framework as the improved version of it. Both of these frameworks are discussed in detail in later sections.

We solve the second problem by adding an adversarial classifier to the QA Dataset Generation Framework. The classifier is trained on the failed QA pairs of X and can thus predict QA pairs on which X will fail. The most challenging part of this project was to build and stabilize the high performing classifier that can understand the behaviour of a given QA System and predict which questions the QA System can and cannot answer.

3 Our Approach

3.1 QA Dataset Generation Framework

Inspired from the approach followed by (Zhang et al., 2018), we propose a pipelined approach to generate the dataset. We use SQuAD v1.1 dataset (Rajpurkar et al., 2016) as source of our initial paragraphs. SQuAD has 23,215 paragraphs obtained from 536 Wikipedia articles. We randomly sample 500 paragraphs from it to generate our final $< Paragraph, Question, Answer >$ triplets. We pass the sampled paragraphs through the 4 components described below.

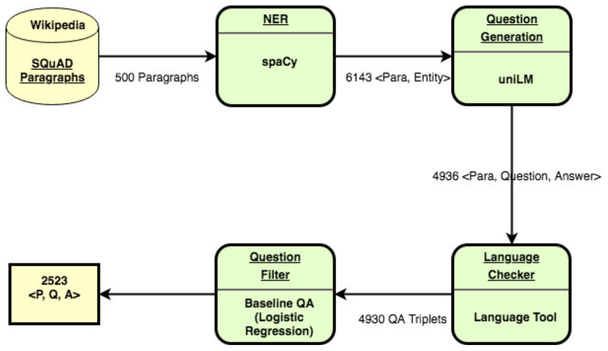

Figure 2: QA Dataset Generation Framework

3.1.1 Named Entity Recognition

We pass each paragraph through a named entity recognition model to retrieve all the named entities in a paragraph. We explored 2 NER models, Allen NLP [1] and spaCy [2]. Allen NLP provides a bi-LSTM CRF model to identify entities.

[1]https://demo.allennlp.org/named-entity-recognition

[2]https://spacy.io/usage/linguistic-features#named-entities

spaCy features a statistical entity recognition system, that assigns labels to contiguous spans of tokens. As we had to process large number of paragraphs, speed of named entity recognition was important. We found out that spaCy's NER model was extremely fast and more accurate than Allen NLP's model in our use case. The bi-LSTM based model failed to identify phrase like 'Santa Clara' as a single entity, thus returning very large number of entities.

Hence, we used spaCy's model over 500 paragraphs to obtain 6143 named entities. A passage may contain an entity multiple number of times at different locations in a paragraph but the question generation model generates the same question for that entity, not taking into account location of that entity. Therefore, we just keep a single instance of each entity in the passage, thus avoiding duplicate questions.

3.1.2 Answer Aware Question Generation Model

Given the passages and the named entities, we use Unified pretrained Language Model, a state of the art question generation model proposed by (Dong et al., 2019), to generate questions that ask for the named entities as answer. UniLM formulates the question generation task as a sequence to sequence generation problem. The first segment is the concatenation of input passage and answer, while the second segment is the generated question. It uses a multilayer transformer network pretrained on a sequence to sequence Language Modelling task and finetuned on SQuAD v1.1 dataset (Rajpurkar et al., 2016) to harvest a set of passage, question and answer triplets.

We generate 1 question per <passage, entity> pair, thus generating a total of 6143 questions. Sometimes the model generates same questions for 2 different entities in the paragraph. After removing the duplicate questions, we get a final set of 4936 questions.

We manually analyze a set of 400 generated questions and discuss our observations in section 4.4.

3.1.3 Language Checker

Neural sequence to sequence models have a tendency to repeat themselves (See et al., 2017). We observed that UniLM generates a few questions which either have repeating words or missing words, thus making them syntactically incorrect.

We use a proof reading software LanguageTool[3] to identify and remove such incorrect questions. In order to automate this, we use a python wrapper of the tool[4] to iteratively remove such incorrect questions. We also explored a popular proof reading tool Grammarly [5]. However, Grammarly does not have a public API.

These grammar checking softwares have curated a list of thousands of language specific errors as rules. Out of our 4936 questions, the language checker identified 6 questions which matched with their error rules. For example,

Generated Question	What was the name of On the Freedom of a?
Tool Message	Did you forget something after 'a'?
Tool Rule Id	THE_SENT_END

Table 1: Syntactically Erroneous Generated Question Example

We remove those 6 syntactically incorrect questions from our dataset. An important point to note is that sometimes such proof reading software can be too strict in identifying the errors. For example, they identify lowercasing of the first letter in a sentence as an error. To overcome this, we ran the Language Tool through the SQuAD development set and created a list of rules to be ignored. When running on our generated question dataset, we ignore all errors matching with these rules.

3.1.4 Baseline QA Model

Recent QA models like BERT and XL-NET work have achieved top positions on most QA leader boards. Such models are quite good at answering simple factoid questions, even if the sentences are paraphrased. We believe that generating such questions in the dataset is of no use. We thus incorporate a baseline QA system in our pipeline to filter out such common questions.

We use the Logistic Regression based QA system released with SQuAD to identify easily answerable questions. We pass the generated questions through the system and remove all those questions from our dataset which can be successfully answered by the QA system. The EM accuracy of the baseline QA system on SQuAD devel-

[3]https://www.languagetool.org/

[4]https://github.com/myint/language-check

[5]https://www.grammarly.com/

opment set is 40%. After filtering 4930 questions through the baseline QA system, we are left with a final set of 2523 questions.

We, thus, generate a set of 2523 qood quality questions using our QA agnostic dataset generation framework.

3.2 Adversarial QA Dataset Generation Framework

QA Dataset Generation Framework generates questions agnostic of a QA system. To maintain difficulty in the generated dataset, questions are filtered using a baseline QA system i.e. if the baseline QA answers the question then the question is not difficult enough to be included in the dataset.

On the other hand, with this framework, difficult questions will be generated for a given QA system. In this pipeline a classifier is used in place of the Baseline QA system. Classifier learns to model whether a question will be answered by the given QA system or not and filter out questions which are likely to be answered.

This framework acts as an Adversary for the QA system. It generates questions which are difficult to answer by the QA therefore, when QA is trained on these difficult questions it's performance will improve as indicated by our experiments. Adversary then learns to generates more difficult questions and this cycle continues for multiple iterations.

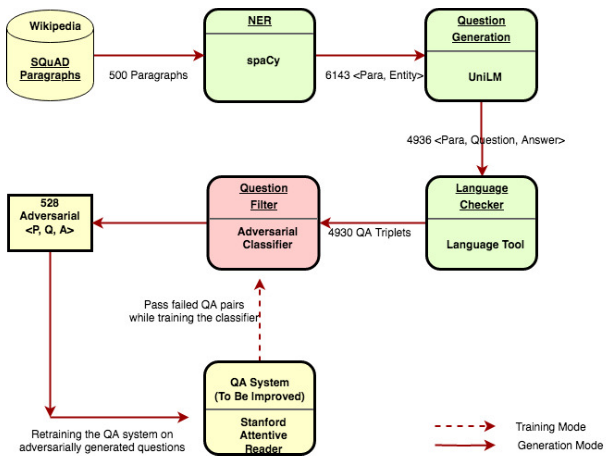
Wikipedia
SQuAD Paragraphs
500 Paragraphs
NER
spaCy
6143 <Para, Entity>
Question Generation
UniLM
4936 <Para, Question, Answer>
Language Checker
Language Tool
4930 QA Triplets
Question Filter
Adversarial Classifier
528 Adversarial <P, Q, A>
Pass failed QA pairs while training the classifier
QA System (To Be Improved)
Stanford Attentive Reader
Retraining the QA system on adversarially generated questions
Training Mode
Generation Mode

Figure 3: Adversarial QA Dataset Generator

3.2.1 Adversarial QA Classifier

To filter out bad quality Adversarial Questions for a given QA system a neural classifier is built which understands the success and failure cases of the QA System. Training Dataset for the classifier is formed by running a given QA system on SQuAD v1.1 (Rajpurkar et al., 2016). The success and failure cases of the QA system are identified using the Exact Match metric of ground truth and QA model output. Since, the classifier is trained on output of a specific QA system, it learns to model patterns specific to that QA system.

For training the classifier, the question and answer text are passed through Facebook's Infersent sentence encoder (Conneau et al., 2017) which provides a one-dimensional vector representation of both question and answer of a fixed size 4096. Then, dot product of the question and answer representation is taken to create a combined representation which is passed through a 2-layer neural classifier to get the prediction on whether a question will be answered by the QA system. Figure 2 shows this architecture.

Approaches tried to get the features for classifier

1. **Idea 1**: Provided only Question embeddings to the classifier.
   Classifier couldn't learn anything from the data and achieved validation accuracy of only 0.57 i.e. slightly better than the random.

2. **Idea 2**: Concatenated embeddings of Question and Sentence.
   This performed better as compared to the previous approach but max. validation accuracy we could achieve was only 0.61.

3. **Idea 3**: Take dot product of the question and answer embeddings.
   This approach works best, with max. validation accuracy achieved = 0.702. The intuition behind this is that dot product represents relationship between question and answer which helps classifier identify patterns.

4. **Idea 4**: Seeing success with previous approach, we tried giving input dot product of embeddings of Passage, Question and Answer but we couldn't train the model successfully as generating paragraph embeddings was taking too long on CPU.

3.3 Implementation Details

3.3.1 Question Generation using UniLM

We setup the environment as mentioned on the UniLM Github codebase. As UniLM is a multi-layer transformer model, it needs pretraining on

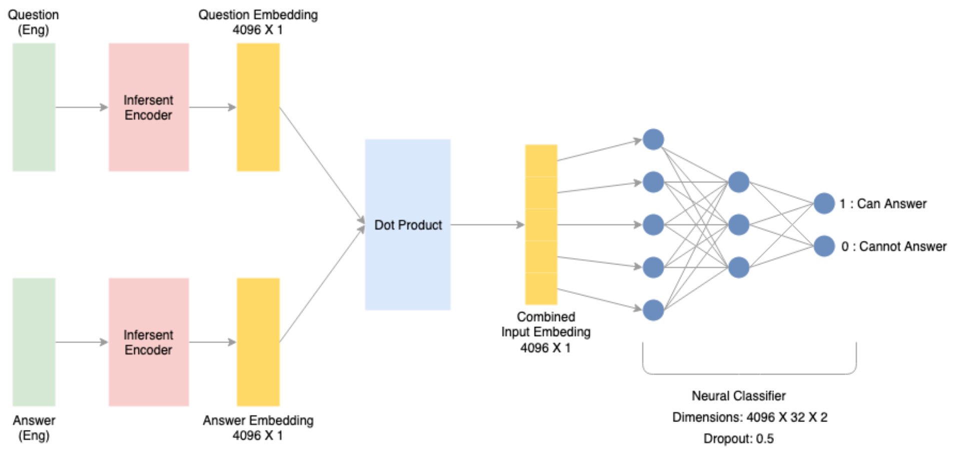

Figure 4: Classifier Architecture

Epochs	Batch Size	Hidden Layer	Weights Initializer	Dropout	LR	Train Acc	Val Acc	Finding
30	256	1(512)	$1/sqrt(\|W\|)$	0	0.01	1.0	0.591	Overfitting
30	256	1(64)	$1/sqrt(\|W\|)$	0	0.01	0.999	0.607	Overfitting
30	256	1(16)	$1/sqrt(\|W\|)$	0	0.01	0.858	0.611	Overfitting
30	256	0	$1/sqrt(\|W\|)$	0	0.01	0.74	0.634	Fitting
100	256	1(16)	Normal (sd=0.001)	0.5	0.001	0.712	0.662	Val Acc. Improved
100	512	1(16)	Normal (sd=0.001)	0.5	0.001	0.715	0.673	Val Acc. Improved
100	1024	1(16)	Normal (sd=0.001)	0.5	0.001	0.711	0.668	No gain
100	512	1(32)	Normal (sd=0.001)	0.5	0.001	0.723	0.702	Val Acc. Improved

Table 2: Classifier Experiments on BERT's predictions

2-4 GPUs. Due to resource constraint, we use a pretrained model and generate questions by running the inference on CPU. Generating 4930 questions through the model took almost 45 hours. We used 512 as maximum input sequence length, 48 as maximum target length, 16 as batch size and 1 as beam size while generating questions. We added a few scripts for data formatting while providing UniLM input and using UniLM output.

3.3.2 Adversarial QA Classifier

The classifier is a 2-layer neural network, *Linear Layer(32)* $\rightarrow$ *ReLU* $\rightarrow$ *Linear Layer(2)* $\rightarrow$ *Softmax* with dimensions $4096 * 32 * 2$ Figure 4. It is trained for 100 epochs with learning rate = 0.001 and batch size = 1024. Since, input dimension is huge, model quickly learns to overfit the training data as indicated in the Table 2. Regularization using dropout(0.5) on the input layer and normally initialized weights with standard deviation = 0.001 are employed to prevent overfitting.

Classifier is implemented in Pytorch v1.3 and is trained on the output of three different QA systems namely BERT (Devlin et al., 2018b), SAR (Chen et al., 2017) and BiDAF (Seo et al., 2016) on SQuAD v1.1 (Rajpurkar et al., 2016). Success and Failure cases (model was/wasn't able to answer the question) of these models are identified using the Exact Match metric (Rajpurkar et al., 2016). Success cases are labelled as 1 and consequently, failure cases are labelled as 0. These labelled <Question, Answer> pairs are used to train the classifier.

Since, QA systems used here have good exact match and F1 score on SQuAD, the number of Positive and Negative examples in the dataset is imbalanced. Therefore, while training the classifier positive examples are randomly sampled to maintain positive and negative examples ratio close to 1.

Training classifier for 100 epochs takes approx. 30 mins on CPU where most of the time is consumed in generating question and answer representation from the Infersent (Conneau et al., 2017). Infersent model is kept fixed during classifier training.

3.3.3 Training BiDAF

We trained BiDAF from scratch on one 12GB GPU on GCP as per the instructions mentioned in Allen AI Github Page but were unsuccessful in integrating it with our classifier to complete the pipeline. Therefore, Allen AI API of trained BiDAF model was used in the QA dataset generation pipeline.

3.3.4 Training BERT

BERT (Devlin et al., 2018b) model is finetuned on SQuAD v1.1 (Rajpurkar et al., 2016). It took 20-22hrs to train with Batch Size = 8, Learning Rate = 3e-5, max_seq_length = 384 and doc_stride = 128 for 2 epochs on one Tesla K40 12GB GPU. The implementation of BERT is done in Tensorflow.

3.3.5 Training Stanford Attentive Reader

SAR (Chen et al., 2017) model is also trained on SQuAD v1.1 (Rajpurkar et al., 2016) as per the instructions given in their Github page. It took 10hrs to train it with Batch Size = 32, Learning Rate = 0.1, number_layers = 3 and max_len = 15, random seed = 1013 for 40 epochs on one Tesla K40 12GB GPU. The implementation of SAR is done in Pytorch.

4 Evaluation

4.1 Dataset Details

- Paragraphs from SQuAD v1.1 (Rajpurkar et al., 2016) are used for generating questions through UniLM.

- All three QA models, BERT (Devlin et al., 2018b), BiDAF (Seo et al., 2016) and SAR (Chen et al., 2017) are trained on SQuAD v1.1 (Rajpurkar et al., 2016)as well.

- Finally, our classifier is also trained on <Question, Answer> pairs from SQuAD v1.1 (Rajpurkar et al., 2016).

4.2 Evaluation Measures

- **Adversarial Classifier**
  Classifier is evaluated using classification accuracy, Precision, Recall and F1 score.

- **QA System**
  As proposed by (Rajpurkar et al., 2016), we use **Exact Match** and **F1 Score** to evaluate the performance of a QA system.

- **QG Model**
  The UniLM Question Generation model is evaluated using the BLEU-4 score.

4.3 Results

QA Model	Val Acc.	Precision	Recall	F1
BERT	0.702	0.862	0.755	0.805
SAR	0.652	0.752	0.721	0.736
BiDAF	0.653	0.768	0.737	0.752

Table 3: Classifier Performance on QA Systems

Classifier's performance is evaluated on the three QA systems namely BERT (Devlin et al., 2018b), Stanford Attentive Reader (Chen et al., 2017) and Bidirectional Attention Flow for Machine Comprehension (Seo et al., 2016). The comparative results are shown in Table 3. Classifier achieves a validation accuracy of approx. 65% for different models. This indicates it is able to learn some patterns to identify which questions can be answered by the QA system.

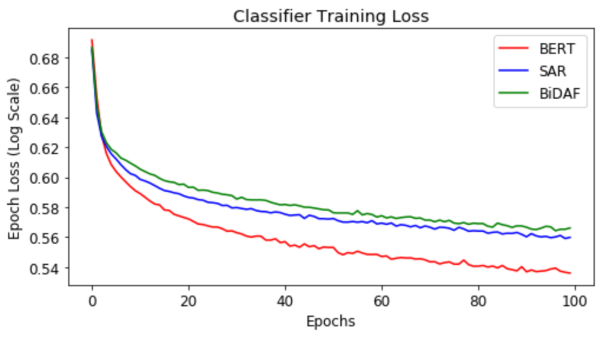

Figure 5: Training Loss of Classifier

The figures 6 and 7 show the comparative performance of the classifier on training and validation sets across multiple QA Systems.

Our classifier, when trained on failure cases of Stanford Attentive Reader (SAR) QA model,

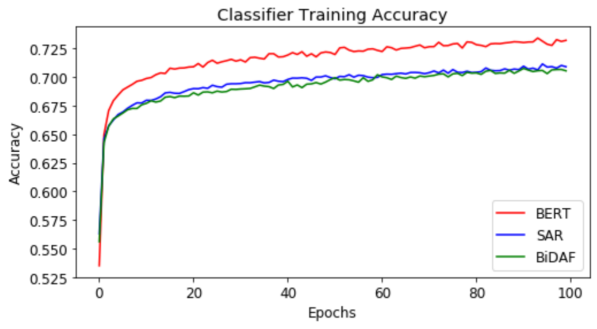

Figure 6: Training Accuracy of Classifier

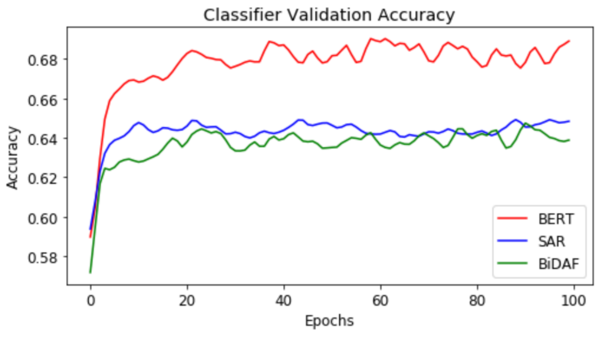

Figure 7: Validation Accuracy of Classifier

identifies 528 out of 4930 questions as unanswerable. We augment the SQuAD training data set by adding these questions and show that in one of the experiments, the EM accuracy of the SAR model increased from 67.25% to 68.01% when trained on the augmented data set.

4.4 Analysis

4.4.1 UniLM Examples

To analyze the quality of questions generated by UniLM, we manually analyze 400 QA pairs and filter out 133 diverse quality questions. We observe that the UniLM model can effectively generate questions involving paraphrasing, counting and having answers in multiple sentences. Table 5 shows types of question generated by the UniLM model.

In the first example, the question word **renamed** is paraphrased version of the phrase **suspending the tradition of naming each Super Bowl game with Roman numerals**.

In the second example, the answer **The Super Bowl 50** can be obtained by reasoning across the 2 sentences. In the paragraph, the **It** in the last sentence refers to **Super Bowl 50** mentioned in the first sentence.

In the third example, the question word **majority** represents the paragraph phrase **1,178,914 inhabitants 833,500**, thus confirming that the UniLM model can also count.

We found that few of the questions generated by UniLM are incorrect. For example:

Paragraph	BSkyB's direct-to-home satellite service became available in 10 million homes in 2010, Europe's first pay-TV platform in to achieve that milestone. Confirming it had reached its target, the broadcaster said its reach into 36% of households in the UK represented an audience of more than 25m people.
Question	What was the target audience of BSkyB ' s direct - to - home service?
Answer	more than 25

Table 4: Semantically Erroneous Generated Question Example

In the above question, UniLM used the word 'audience' as the cue for given named entity 'more than 25' and generated the above question. Clearly, the question generated is wrong.

Thus, our framework still needs to involve humans to check the correctness of generated questions.

4.4.2 Retraining QA systems

Due to time and resource constraints, we could only retrain the Stanford Attentive Reader on our adversarial 528 $< P, Q, A >$ triplets. In one of the experiments, we achieved an accuracy jump from 67.25% to 68.01%. However, we could not reproduce it consistently. In order to claim that our idea of adversarial classifier works well, we also need to retrain BERT, BiDAF and other QA systems on our adversarial system.

4.5 Code

All code, data, and experiments for this paper are available on our Github repository: https://github.com/miteshksingh/AdversarialQuestionGeneration

5 Conclusions

In this work, we describe a full automated QA dataset generation framework which can generate large amounts of *(paragraph, question, answer)* triplets. In our framework, we use spaCy's NER model, UniLM for question generation, Language Tool for grammar checking and SQuAD Logistic regression based QA for QA filtering. We generate a set of 2523 QA pairs. We further train an adversarial classifier, with an average accuracy of 65%, which can predict QA pairs on which a given QA system will fail.

Due to time and resource constraint, we could not retrain other QA systems (BERT and BiDAF) on our adversarial dataset. In future, we would like to generate large sets of questions and verify if the performance of multiple QA systems improve on our adversarially generated data. Involving human in the loop for verifying generated questions might be a good experiment to try as well. We would also like to model paragraph embeddings in the classifier. Lastly, we hope to have an iterative learning process for any QA system in which the QA system improves on each iteration by retraining on adversarially generated dataset.

References

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680, Copenhagen, Denmark. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018a. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018b. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019. Unified language model pre-training for natural language understanding and generation. *CoRR*, abs/1905.03197.

Nan Duan, Duyu Tang, Peng Chen, and Ming Zhou. 2017. Question generation for question answering. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 866–874, Copenhagen, Denmark. Association for Computational Linguistics.

Anusri Pampari, Preethi Raghavan, Jennifer Liang, and Jian Peng. 2018. emrQA: A large corpus for question answering on electronic medical records. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2357–2368, Brussels, Belgium. Association for Computational Linguistics.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. *CoRR*, abs/1704.04368.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2016. Bidirectional attention flow for machine comprehension.

Yu Su, Huan Sun, Brian Sadler, Mudhakar Srivatsa, Izzeddin Gür, Zenghui Yan, and Xifeng Yan. 2016. On generating characteristic-rich question sets for QA evaluation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 562–572, Austin, Texas. Association for Computational Linguistics.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237.

Xingdi Yuan, Tong Wang, Caglar Gulcehre, Alessandro Sordoni, Philip Bachman, Saizheng Zhang, Sandeep Subramanian, and Adam Trischler. 2017. Machine comprehension by text-to-text neural question generation. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 15–25, Vancouver, Canada. Association for Computational Linguistics.

Sheng Zhang, Xiaodong Liu, Jingjing Liu, Jianfeng Gao, Kevin Duh, and Benjamin Van Durme. 2018. Record: Bridging the gap between human and machine commonsense reading comprehension. *CoRR*, abs/1810.12885.

#	Paragraph	Question	Answer
1.	As this was the 50^{th} Super Bowl, the league emphasized the "golden anniversary" with various gold-themed initiatives, as well as temporarily suspending the tradition of naming each Super Bowl game with Roman numerals (under which the game would have been known as "Super Bowl L"), so that the logo could prominently feature the Arabic numerals 50.	What was the Super Bowl 50 called before it was renamed ?	Super Bowl L
2.	The Super Bowl 50 halftime show was headlined by the British rock group Coldplay with special guest performers Beyonce and Bruno Mars, who headlined the Super Bowl XLVII and Super Bowl XLVIII halftime shows, respectively. It was the third-most watched U.S. broadcast ever.	What was the third most watched broadcast ever ?	The Super Bowl 50
3.	In 1933, out of 1,178,914 inhabitants 833,500 were of Polish mother tongue. World War II changed the demographics of the city, and to this day there is much less ethnic diversity than in the previous 300 years of Warsaw's history.	What was the main language of the majority in 1933 ?	Polish

Table 5: Example Question Generated by UniLM