Question Answering on Amazon Reviews

Zheyuan Hu, James Zhan, Kunru Lu, Issac Guo
New York University, New York, NY 10003
zh2095@nyu.edu, jz4665@nyu.edu, kl3743@nyu.edu, xg2084@nyu.edu

Abstract

E-commerce has become an essential part of many of our lives. But how does a customer know the products they see on Amazon are right for them? They can post questions but sometimes their questions do not get answered for a long time. An automated way to have these questions answered would be extremely helpful to both the customer and Amazon. According to the authors of AmazonQA, a large portion of questions can be directly answered from the reviews themselves. Using this information, we aim to build a system that accepts a query question for a product, scans through its reviews and outputs an answer if it can find one.

1 Introduction

One of the greatest challenges in natural language processing today is question answering. That is, given a question and some context can a machine learn to output not only relevant but also high quality answers? Not only would such a model need to comprehend the question and context but it must also perform reasoning in order to produce answers that make sense. SQuAD[1, 2] is one of the most well-known datasets for QA, but the context is only one document and the answers are a few words long that are contained within the context. In AmazonQA [3], such a QA model must analyze multiple reviews that are specific to each question before constructing free-form answers. In this paper, the authors implemented a seq2seq encoder decoder with an Uni-LSTM architecture. However beyond that there are no leading models for this task as of our knowledge. In this project, we will try to improve upon the current performances for this task. We will also compare how well our models built from scratch can hold against transfer learning from a pretrained transformer-based language model when applied to this dataset.

2 Dataset and Tools

We are be working with the AmazonQA dataset curated by the original paper [3]. This dataset consists of product related questions asked on Amazon with each question provided a list of the top upvoted answers from the community. Additionally, the context for each question is a list of user reviews about the product and relevant to the question, with relevancy determined and ranked by the metric, BM25 [4]. In total there are 923k questions, 3.6M answers and 14M reviews across 17 different product categories in this dataset. For our work, we focus on the Video Games category which has 7.5k questions, 28k answers and 75k reviews. We choose to split our data using a 80/10/10 split for train/val/test.

Our code is written in Python with PyTorch as our deep learning framework. For computational resources we work with Google Colab using their default GPU.

3 RNN-Based Encoder Decoder

3.1 Methodology

Our first methodology would still be focusing on the RNN-based encoder-decoder. We noticed that the original seq2seq pipeline is using a simple Unidirectional LSTM, so here we found the following four aspects where the model can be improved, and we adjusted the whole pipeline to incorporate these aspects into the original algorithm.

3.1.1 Pretrained Embedding

In the original seq2seq code, the author is using the default embedding module of the torch package, nn.embedding, to store word embeddings and retrieve them using indices. Knowing that nn.embedding is basically a simple look up table that takes in a list of indices, and output the corresponding word embeddings, we decided to replace it with some pretrained embedding as the initial weight of the embedding layer so that the meanings and semantic relationships can be captured before being fed into the RNN cells.

Our initial thought is to use ELMo, which is the state-of-the-art contextualized embedding. However, the problem is that ELMo needs to fit in Character CNN for its initial representation, so it expects sentences to be tokenized into characters instead of words while the AmazonQA dataloader is using word as token. Therefore, we decided to implement GloVe instead, which is an unsupervised learning algorithm to obtain vector representations for words, and we used the GloVe model pretrained on 27 billion tokens from Twitter with embedding size 200. The reason we selected the largest embedding size 200 is that we noticed that some of sequences in our task is relatively long so it might require a higher dimension to capture its full meaning.

3.1.2 Attention Mechanism

In the encoder-decoder architecture, all necessary information in the input sequences has been compressed into a fixed length vector. This poses problems in holding on to information at the beginning of the sequence and encoding long-range dependencies. Therefore, based on our knowledge, we believed that implement attention mechanism is going to alleviate this problem by focusing on the most relevant parts of the input sequence for each output. Besides, attention also helps with the vanishing gradient problem since it provides a direct path to the inputs. Knowing that the alignment model of attention, can be computed in various ways, here we decided to use the additive one to compute our alignment model:

Computing $e_1, \ldots, e_n \in \mathbb{R}$ from $k_1, \ldots, k_n \in \mathbb{R}^{d_1}$ and q $\in \mathbb{R}^{d_2}$

- Basic dot-product attention:

$$e_i = q^T k_i \in \mathbb{R}$$

where $d_1 = d_2$

- Multiplicative attention:

$$e_i = q^T W k_i \in \mathbb{R}$$

where $W \in \mathbb{R}^{d_2 x d_1}$ is a weight matrix

- Additive attention:

$$e_i = W_3{}^T \tanh(W_1 k_i + W_2 q) \in \mathbb{R}$$

where $W_1 \in \mathbb{R}^{d_3 x d_1}$, $W_2 \in \mathbb{R}^{d_3 x d_2}$ are weight matrices and $W_3 \in \mathbb{R}^{d_3}$ is a weight vector.

Fig 3.1. Different Ways to Compute Alignment

3.1.3 Bidirectional LSTM

Another important aspect that might enhance the model is that we can turn the unidirectional LSTM into a bidirectional LSTM. Bi-LSTM have two recurrent components, a forward recurrent component and a backward recurrent component. The forward component computes the hidden and cell states similar with a standard unidirectional LSTM whereas the backward component computes them by taking the input sequence in a reverse-chronological order. Therefore, unlike the Uni-LSTM who stores information only from the past, Bi-LSTM is able to preserve information from both the past and the future and understand the context better since it also takes the context information of the future into consideration. In our case, we set the RNN cell for both the encoder and the decoder to be bidirectional.

3.1.4 GRU

Finally, we decided to try another cell type of RNN, the Gated Recurrent Unit (GRU), as an alternative of the LSTM cell. The reason is that LSTM has three gates (input, output and forget gate) while GRU has only two gates (reset and update gate) by coupling the forget gate and the input gate together. With fewer gates, GRU use less training parameters and therefore use less memory, execute faster and train faster than LSTM. In addition, we are using a reduced dataset to train the model whose size is relatively small, and GRUs is proved by experiments to usually have a better performance than LSTMs on small size of the training data.

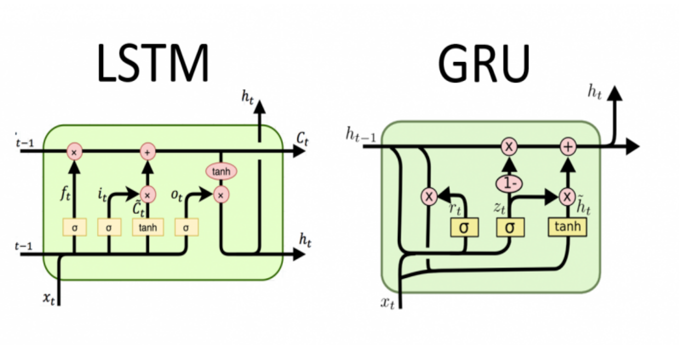


Fig 3.2. Cell Types of RNN

3.2 Models Selection

Based on the previous four aspects, we updated the whole pipeline and implemented the following five models for future analysis and comparison:

- Baseline:
  - Uni-LSTM
- Enhanced Models:

- Uni-LSTM with pretrained GloVe embedding (Twitter 27B, 200d)
- Uni-LSTM with encoder attention
- Bi-LSTM
- Uni-GRU with encoder attention

We also tested the Bi-LSTM with encoder attention, but the result was not as good as we expected, so we decided not to include it in our final selection of models.

3.3 Hyperparameters Tuning

For the hyperparameter tuning part, since there are too many hyperparameters for a RNN based sequence to sequence model and each set of configuration takes hours to train, it is not possible for us to tune on all the hyperparameters for it might take days or even weeks. Thus, we first divided all the hyperparameters into two groups, the fixed group and the tuned group, based on the extent to which these parameters affect the results. For those fixed parameters, we basically used the author's choice. While for those in the tuned group, we first selected some values to be the candidate parameters based on our knowledge and experience, and then performed a thorough grid search on them. The below table shows the hyperparameters tuning for the Uni-LSTM (baseline) model, and the red values are our final selections.

Fixed Hyperparameters	
Parameter Name	Value
Number of Epochs	10
Batch Size	32
Learning Rate Decay	0.5
Teacher Forcing Fatio	1.0
Global Norm Max	5

Tuned Hyperparameters	
Parameter Name	Value Tested
Learning Rate	1e-2, 1e-3, 1e-4
Hidden Dimension	128, 256, 512
Hidden Layers	1, 2
Embedding Dimension	128, 256, 512
Dropout Rate	0, 0.1, 0.2

Table 3.1. Hyperparameters Tuning for Uni-LSTM

3.4 Evaluation

Then after training the models, we decided to evaluate them based on their performance and computational cost. In terms of performance, we used BLEU and ROUGE_L to assess how well our model performs at providing high quality and relevant answers. The BLEU score measures precision by indicating how similar the candidate text is to the reference texts , while the ROUGE_L score measures recall by taking into account the sentence level structure similarity naturally and identifying longest co-occurring in sequence n-grams automatically.

As for computational cost, we simply used the time consumption in each epoch as an indication, so we recorded in all the 5 models how long they take to train one epoch and compare the results with each other. Besides the machine evaluation, we also evaluated the models by comparing the answers they generated in a human perspective, which will be discussed in the later section.

Model	Performance	
	BLEU	ROUGE_L
Uni-LSTM (baseline)	35.67	23.64
Uni-LSTM+GLoVe	41.39	23.76
Uni-LSTM+Attention	49.61	25.01
Bi-LSTM	68.07	64.39
GRU+Attention	34.41	21.43

Model	Time Comsumption in Each Epoch
Uni-LSTM (baseline)	4 mins 21 secs
Uni-LSTM+GLoVe	4 mins 13 secs
Uni-LSTM+Attention	4 mins 27 secs
Bi-LSTM	8 mins 43 secs
GRU+Attention	4 mins 4 secs

Table 3.2. Model Evaluation Results

We can see that compared to the baseline model, the Bi-LSTM has the greatest improvement, while the GloVe embedding and attention also helps improve the performance to some extent. Although the Bi-LSTM model takes about 9 minutes to train a single epoch, which is almost twice slower than the other 4 models, considering its significant improvement and with sufficient time and computing power, we still consider the Bi-LSTM to be our best model here.

4 DistilBERT-Based QA Model

4.1 Methodology

Besides RNN, another method we tried is to build a bert-based QA model, inspired by DrQA[5]. The main concept is as shown in Fig 4.1.

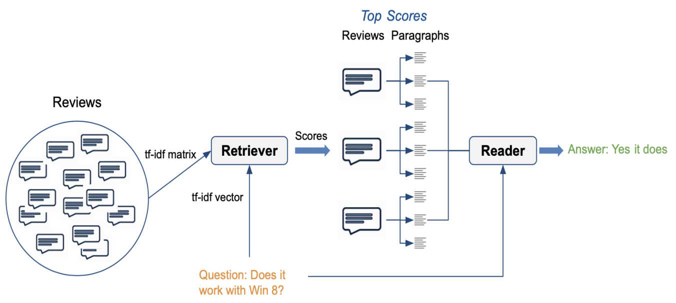


Fig 4.1. Mechanism of BERT QA Pipeline

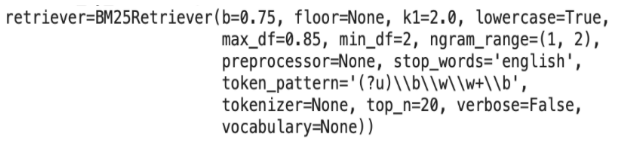
The model architecture consists of two main components: the Retriever and the Reader. In our case, firstly, the Retriever will find the most related reviews to the question, by creating TF-IDF features based on uni-grams and bi-grams and computes the cosine similarity between the question and each review of the dataset.

Then, the most probable reviews selected will be divided into paragraphs and sent to the Reader, together with the question. The Reader is basically a DistilBERT-based model pre-trained on SQuAD 1.1 dataset. It will then output the most probable answer it can find in each paragraph.

Finally, the answers will be compared based on an internal score function, and the most likely one will be output as the generated answer.

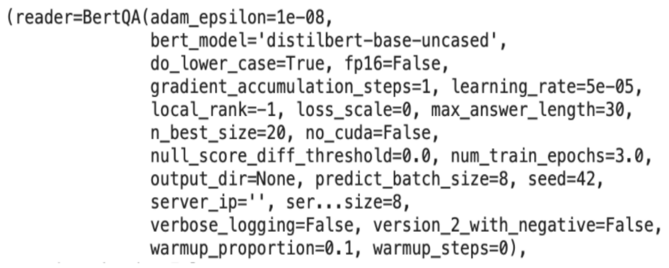
4.2 Fine-tuning

To improve the performance of the model, we fine-tuned the Retriever by the reviews of the AmazonQA dataset and tried to fine-tune the pre-trained Reader by the annotated custom dataset. The parameters of the Retriever and the Reader are as shown in Fig 4.2. and Fig 4.3. respectively.



```
retriever=BM25Retriever(b=0.75, floor=None, k1=2.0, lowercase=True,
                        max_df=0.85, min_df=2, ngram_range=(1, 2),
                        preprocessor=None, stop_words='english',
                        token_pattern='(?u)\\b\\w\\w+\\b',
                        tokenizer=None, top_n=20, verbose=False,
                        vocabulary=None))
```

Fig 4.2. Parameters of the Retriever



```
(reader=BertQA(adam_epsilon=1e-08,
               bert_model='distilbert-base-uncased',
               do_lower_case=True, fp16=False,
               gradient_accumulation_steps=1, learning_rate=5e-05,
               local_rank=-1, loss_scale=0, max_answer_length=30,
               n_best_size=20, no_cuda=False,
               null_score_diff_threshold=0.0, num_train_epochs=3.0,
               output_dir=None, predict_batch_size=8, seed=42,
               server_ip='', ser...size=8,
               verbose_logging=False, version_2_with_negative=False,
               warmup_proportion=0.1, warmup_steps=0),
```

Fig 4.3. Parameters of the Reader

4.3 Result

We initially evaluated the model by F1 score and got a result of 0.0480 when applied to the whole dataset. This rather low score is expected, since the answers of AmazonQA dataset are written by humans, while the generated answers are extracted from the reviews. Thus, in this case, the comparison metrics are not quite applicable. So, we compared the generated answers of the BERT model with that of other models and the actual answers in a human perspective in the following section.

5 Analysis

In this section we will analyze and evaluate our model from a human based perspective. We will aim to highlight and compare the pros and cons between our baseline model implemented by the original paper (Uni-LSTM), our best model (Bi-LSTM) and DistilBERT.

Here we provide some examples of our model predictions.

Question: Can it run on windows 8?

Uni-LSTM: ['yes , I works on with windows 7 . 1 , . . work work work work work work work work work are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are are']

Bi-LSTM: yes , it runs great on windows 8 .

DistilBERT: This simply does not work

Answer: 'I run it on Windows 8.1 pro with no issues!

Question: Does it work on the ps4?

Uni-LSTM: yes , you a same cable .

Bi-LSTM: yes if it's the screen version

DistilBERT: dose not work

Answer: Yes, works good on the ps4

From our generated predictions, we notice some clearly distinctions across the models.

For the Uni-LSTM, we see that the grammar is very poor in its predictions. Additionally, in the first highlighted prediction, and in many other predictions that weren't shown, the baseline model will tend to repeat words until it reaches the max prediction length. A possible explanation may be because the generated output sequence minimizes on perplexity, and perplexity is not good on penalizing repeating words and limited vocabulary.

However, a good aspect of the baseline model is that the predictions are for the most part correct.

For DistilBERT, although the predictions are grammatical, the logic of its predicted answers is poor. For both the examples, and many others, DistilBERT's predicted answers is not the same as the top answer and can be quite contradictory. This may be because DistilBERT is pretrained on the Wikipedia corpus, which follows a different than the information that is specific for a product. Additionally, the number of reviews for each question is not enough to effectively fine-tune DistilBERT to be able to answers these questions correctly since DistilBERT has many parameters.

For the Bi-LSTM, we see that the predictions are both grammatical and matches the logic of the top answers.

Conclusion

In our work, we were able to effectively answer questions asked on Amazon using user reviews. Furthermore , we were able to implement multiple models that can be shown to improve on the original paper's performances for metrics like BLEU and ROUGE_L and can be used as new leading performances for this task. Finally, we emphasize that our models are lightweight as it does not take long to train and can be trained using open access resources such as Google Colab.

References

[1] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.

[2] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad, 2018.

[3] Mansi Gupta, Nitish Kulkarni, Raghuveer Chanda, Anirudha Rayasam, and Zachary C Lipton. Amazonqa: A review-based question answering task, 2019.

[4] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, April 2009.

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

[6] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *In EMNLP*, 2014.

[7] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018.

[8] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2019.

[9] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2019.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[11] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

[12] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.