Neural Question Generation

Daniel Do, Bilguunzaya Battogtokh

December 14, 2019

1 Introduction

Question generation attempts to create natural questions from a body of text. An important application of this technology is related to education: we can use question generation to provide readily-available reading comprehension material given any corpus of text. Successful question generation can also used to generate non-trivial annotated datasets for question answering given a piece of text. Furthermore, question generation can be used as a submodule in chatbot systems to prompt certain answers from users based on the responses that the user has given so far. This feature could be integrated in a wide-ranging variety of disciplines, such as in commercial chatbots or in healthcare-based chatbots.

2 Related Works

In the past, the rule-based approaches had been used for the task of question generation; these methods rely on well-crafted rules based on expert linguistic knowledge. However, these methods tend to lack robustness and semantic awareness while generating questions. On the other hand, deep learning methods utilizing encoder-decoder methods have shown remarkable success in this area. In the seminal paper of Du et al.[1], the authors propose an end-to-end trainable attention-based sequence model to generate questions from a corpus of text. Specifically, they use an encoder-decoder RNN model that pays attention to specific areas of the input text while generating the next word in the question sentence sequence.

Tang et al.[2] present an alternative approach where question answering and question generation are presented as dual tasks. The question answering (QA) model judges whether the question generated from the question generating (QG) model is relevant to the answer. Likewise, the QG model determines the probability of generating a specific question conditioned on an answer, which contributes to learning the QA model. Because both tasks in their paper are treated with equal priority, the authors state their approach is distinct from the generative adversarial network (GAN) model. Chan et al. also use a different approach for text generation. Instead of using an encoder-decoder structure, they use BERT, a pretrained language model, as the decoder itself[3]. In each iteration of decoding, Chan et al. feed as input the context sentence as well as the question words generated so far, take the last hidden state of BERT, and use an affine layer to map this last hidden state to their vocabulary in order to generate a new word.

3 Dataset

We are using SQuAD, the Stanford Question and Answering Dataset, which is a reading comprehension dataset consisting of over 100,000 crowdsourced questions from approximately 536 Wikipedia articles[4]. For any question in the dataset, the answer is an segment of text in the reading passage associated with the question. The questions and answers in this dataset are of relatively high quality since they are completely annotated by human crowdworkers.

4 BERT

Because both of our models utilize BERT, a pretrained language model, we will give a high-level overview of the purpose and function of BERT. BERT stands for Bidirectional Encoder Representations from Transformers, and is a bidirectional language model transformer that was pretrained on two datasets of text: BooksCorpus which contains 800 million words and EnglishWikipedia which contains 2500 million words.

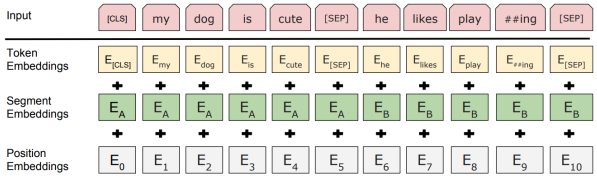
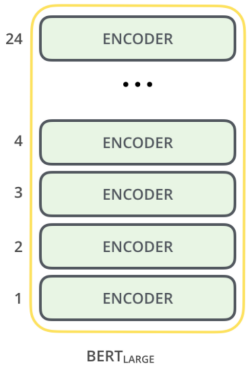
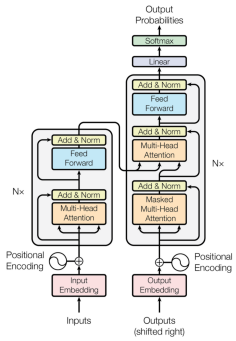


Figure 1: Each input word is replaced with the summation of three different embeddings.Image is from Devlin et al.

At it's core, BERT is mainly a transformer. The large version has 24 encoder stacks. First, each word in the input sentence is replaced with the summation of a WordPiece embedding, a learned segment embedding indicating whether the current word belongs to sentence A or sentence B in a pair of sentences, and a learned position embedding indicating the word position within the sentence. This resulting embedded sentence is then passed through 24 transformer blocks for the large version of BERT.



(a) The summed encoding is then passed through 24 transformer encoder blocks. Image is from http://jalammar.github.io/illustrated-bert/.



(b) This is the typical structure of a transformer encoder. This image is from https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04

Figure 2

BERT is trained with two methods: masked language prediction and next sentence prediction. While training with masked language prediction, BERT masks out 15% of the words in the sentence and uses the sentence context in order to predict the masked out words. In the second training task, BERT is provided with two sentences and the objective is to determine whether the two sentences are sequentially connected or not.

In this paper, we use DistilBERT instead of BERT_small or BERT_large for training and inference efficiency as well as to stay within our computational budgets. DistilBERT has only 66M parameters compare to 340M parameters in BERT. Additionally, DistilBERT retains 97% of the language understanding capabilities of BERT while being 60% faster[5].

5 Experiments

BERT Seq2Seq

We first trained the baseline model that Du et al. provided, and then ran the trained model on the development set in SQuAD. This model was a straightforward sequence2sequence model taking in the context sentence as the input to the encoder, and using the final hidden state of the encoder as the initialization of the decoder. They used 300-dimensional GloVe embeddings which were trained on 840B tokens to convert the words into tokens before feeding them into the network. In addition to the vanilla seq2seq, Du et al. also used additive attention while decoding words.

Our first approach was to replace the GloVe vectors with BERT word embeddings. In order to do this,

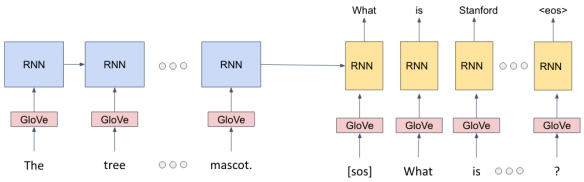


Figure 3: This is the baseline model from Du et al.

we pass the context sentence in, and use the output word embeddings as the input to the encoder. For each step in the decoding process, we fed in what we had generated so far into BERT and used the last word embedding as the input into the RNN and using the previous step's hidden state. We also used additive attention like in Du et al. For our RNN, we used GRU. Our encoder and decoder were both unidirectional and uni-layered and had a hidden dimension of 784. We had an affine layer mapping from 784 to the size of our vocabulary, which we used to generate words from the hidden state of the RNN at each step in the decoding process. We trained on the MLE objective with ADAM with learning rate $5e-5$, weight decay .0005, and mini-batch size 8. We used beam search for inference with a beam search size of 3.

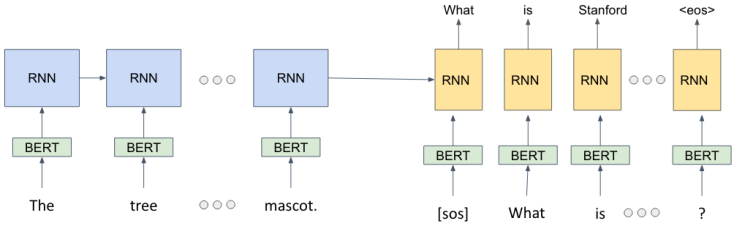


Figure 4: This is our first approach.

Bert Encoder

Our second model was replacing the encoder in the baseline model with BERT, and keeping the same decoder structure from Du et al. Since the baseline model was written in Lua, we needed to translate the codebase into PyTorch first. We used a pre-trained BERT model that produced a 1024-dimensional sentence embedding instead of individual word embeddings[6]. For each sentence that we input into the BERT model, we get an output vector of 1024 dimensions. For our decoder, we used a 2-layer LSTM where the hidden unit size was 1024. Like in approach number #1, we had an affine layer mapping from 1024 to the size of our vocabulary to produce words from the RNN at generation. Like in the baseline model, we used 300-dimensional GloVe vectors to encode our input into our LSTM. We trained our model on the MLE objective with the ADAM optimizer with learning rate .5 and mini-batches of size 64.

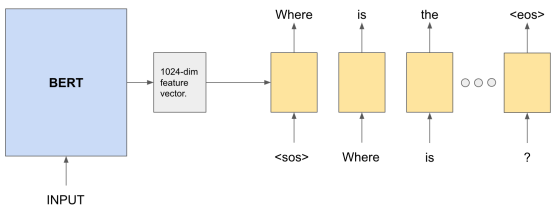


Figure 5: Our input, which is a list of words, is inputted into the pretrained BERT model, which produces a 1024-dimensional feature vector. We then use this feature vector as the initialization state in our LSTM. We first input the word vector for <sos> into our decoder with this initial hidden state, and then sample a word. We then feed the word vector of this next word into the RNN, and sample the word after that, and we repeat this process until we either reach a maximum length of 50 words or we produce an <eos> character.

Recurrent BERT

In this experiment, we attempt to follow a recently published paper by Chan and Fan[3] that differs from RNN based models which can suffer from processing long sequences. In particular, we attempt to follow their BERT-HLSQG model. This model tries to correct for answer phrase ambiguity since an answer phrase may appear multiple times in a long context, there is confusion as to which question we should generation for a particular answer phrase. For a given context, we mark the specific answer phrase by surrounding it with new `[hl]` tokens to eliminate this ambiguity. In BERT, the first token in a text sequence is always `[CLS]` and the last token is always `[SEP]`. The `[SEP]` token is used to separate two sentences for the next sentence prediction task and the `[MASK]` token is used in the masked language prediction task.

For a given context paragraph, $C = [c_1, \cdots c_{|C|}]$ and an answer phrase $[a_1, \cdots, a_{|A|}]$, we define the initial input to the Recurrent BERT model as $C' = [c_1, c_2, \cdots, \texttt{[hl]}, a_1, \cdots, a_{|A|}, \texttt{[hl]}, \cdots, c_{|C|}, \texttt{[MASK]}]$. At each step in generating the question $\hat{q}$, we take the final hidden state vector of the last token `[MASK]`, $h \in R^h$, and connect it an affine layer $W \in R^{h \times |V|}$. $|V|$ is the size of the vocab. We compute the $\hat{q}_i$ as follows:

$$\Pr(w|X_i) = \text{softmax}\left(\mathbf{h}_{[\text{MASK}]} \cdot \mathbf{W} + \mathbf{b}\right)$$
$$\hat{q}_i = \text{argmax}_w \Pr(w|X_i)$$

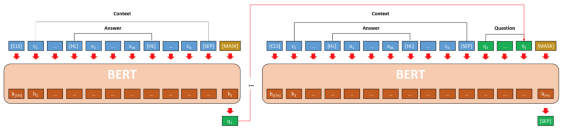


Figure 6: Recurrent BERT example .

6 Results

We use BLEU and METEOR to evaluate our model, which were the evaluation metrics utilized in [1]. Each of these metrics can be used as a measurement of the correlation between the generated questions and the reference questions from the text. BLEU, in particular, compares n-gram similarity between the generated questions and the reference questions. METEOR measures the harmonic mean of the unigram precision and recall of our generated questions.

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

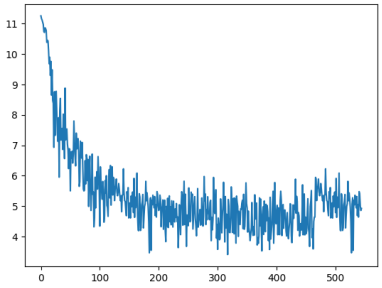
Figure 7: This is the calculation for the BLEU score. BP is a penalty for questions that are too short, w_n is the weight given to n-gram similarity, and p_n is the n-gram similarity. We used the NLTK package to calculate this metric.

$$F_{\text{mean}} = \frac{P \cdot R}{(\alpha \cdot P + (1 - \alpha) \cdot R)}$$

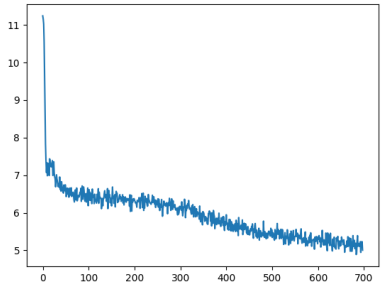
Figure 8: This is the main computation for the METEOR score. This is multiplied by a penalty for short generated questions to get the final METEOR score. We also used NLTK to calculate the METEOR score.

Outputs of each of the models:
Bert Seq2Seq:
Ground: in what year was the last known person sentenced to death in england for heresy ?
Prediction: what the the the the the the ? [pad] [pad] [pad] ..
Bert Encoder:

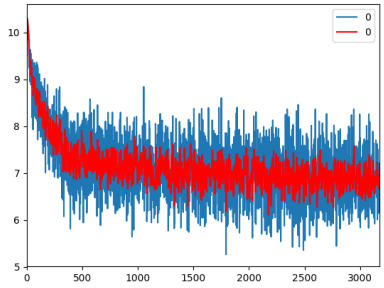
Ground: what did constantine the great and licinius to introduce the liberty of christianity in the roman empire ?
Prediction: what the the the the the the the ? ? [pad] [pad] [pad] .
Recurrent BERT:
Ground: what is the name of the name of the first of the united ? Prediction: what in name is in in ?



(a) This is the loss curve for BERT seq2seq.



(b) This is the loss curve for BERT encoder.



(c) This is the loss curve for Recurrent BERT.

7 Conclusion

Out of the three approaches, the Recurrent BERT model performed the best in terms of the BLEU and METEOR scores. However, we see that the questions generated by the Recurrent BERT model are still quite juvenile. The questions themselves contain no relevant information that is related to the input context. Unlike Chan and Fan, we were only able to run our model for approximately half an epoch rather than the five epochs they trained for. This is because fine-tuning a BERT model, even the DistilBERT, is still quite computationally expensive, and we were able to train for so long before we ran out of the proper resources. However, the questions generated by the Recurrent Bert model show promise in their diversity of length and word choice compared to the outputs generated by the other models.

The results from the BERT encoder produced outputs that contained multiple reptitions of "of the name of the name of the name". We hypothesize that this is because this model got stuck in a local optima which minimized the loss for questions that contained a high amount of the words "of" and "the". Likewise for the BERT Seq2Seq, we see that the model got stuck in a local optima of using "what" and "the" because most questions contain the interrogative word "what" and the glue word "the". One possible method of fixing this local optima would be to increase the loss for producing incorrect concrete words, such as words that directly pertain to the sentence at hand. For example, if a context sentence was "The tree is green", then we would penalize our model more for not producing concrete words like tree.

Future work could also include other methods of text generation distinct from the approaches we've outlined here. For example, a question generating model could utilize the copy mechanism that is typically used for dialogue generation. There has also been a resurgence of interest in using GANs to produce discrete outputs, so using the encoded output of a sentence as the latent space for a generator in a GAN could be another possible method of generating questions from a context sentence.

Please refer to our github repository for our code and methods: https://github.com/battogtokhb/cs229-nqg

References

[1] Xinya Du, Junru Shao, and Claire Cardie. Learning to ask: Neural question generation for reading comprehension. *arXiv preprint arXiv:1705.00106*, 2017.

[2] Duyu Tang, Nan Duan, Tao Qin, Zhao Yan, and Ming Zhou. Question answering and question generation as dual tasks. *arXiv preprint arXiv:1706.02027*, 2017.

[3] Ying-Hong Chan and Yao-Chung Fan. A recurrent BERT-based model for question generation. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, pages 154–162, Hong Kong, China, November 2019. Association for Computational Linguistics.

[4] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[5] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[6] Han Xiao. bert-as-service. `https://github.com/hanxiao/bert-as-service`, 2018.