Using Encoder-Decoder models for Short Text Simplification

Mingliang Wei
HEC Montreal
mingliang.wei@hec.ca

Yanhan Peng
HEC Montreal
yanhan.peng@hec.ca

Qihua Zhong
HEC Montreal
qihua.zhong@hec.ca

Zihui Deng
HEC Montreal
zihui.Deng@hec.ca

Abstract

Our goal of the project is to transform a given short text into an easier version for beginners (children or people who are learning English as a second language). In a such process, complex sentences will be broken down into simple ones, and difficult words will be replaced by easier ones. Current practice has heavily relied on human professional writers. In our project, we explored the possibility of performing this task with different deep NLP models. We experimented with the models on a dataset we crawled from newsinlevel.com, which contains 2588 news in 3 different levels(easiness to read). We utilized 3 different evaluation metrics(BLEU, SARI, and SAMSA on the results generated.

1 Introduction

1.1 Background

Text simplification is most similar to text summarization. There are two main types of text summarization approaches in NLP: Extraction-based summarization and Abstraction-based summarization. The former only extracts keywords from the original text without making changes to the text, while the latter paraphrases and shortens the source document. Our task in this project is more similar to the Abstraction-based summarization. Many neural network models, especially the ones incorporating attention mechanisms, have been proposed for abstraction-based text summarizations. [10] Given the success of attention-based models for text summarization, we believe that Transformer models are suitable for this task. Despite the similarities between text summarization and text simplification, text simplification has its own originalities and challenges. For text simplification, we are converting entire text to entire text, as oppose do traditional sentence to sentence translation, which means unimportant sentences can be ignored completely, and it's probable to adjust the output by specifying the level of simplicity. However, being such efficient means there is a high memory requirement for the model. Also, to effectuate the grammatical correctness of the output text, we face a wide variety of NLP problems such as natural language generation, semantic representation, and inference permutation.

1.2 Research process

We tried 2 different models – Bidirectional Encoder Representations from Transformers(BERT) and Generative Pretrained Transformer(GPT2) model from OpenAI. [4, 7]Our dataset was crawled from newsinlevel.com. The data contains 2588 news in 3 different levels(easiness to read). Our task is

Report Submitted to HEC Montreal for 80600A course project. Group J1-12

to convert the news from a higher level to a lower level. The results generated were then evaluated by 3 different evaluation metrics, which are the Bilingual Evaluation Understudy Score(BLEU), system output against references and against the input(SARI), and Simplification automatic evaluation measure through semantic annotation (SAMSA) [6, 12, 8]. Each metric focuses on different aspects of the generated text. While BLEU correlates with human judgement of adequacy and fluency, but not simplicity, SARI correlates with human judgement of simplicity, and SAMSA correlates to human judgement of fluency, adequacy and structural simplicity.

1.3 Report organization

In §2 we review the literatures of related works. In §3 we present the details of our methodology on how we trained the models, prepared the dataset, and implement evaluations on our generated results. §4 we will present our result of the evaluation, conclude our experiment and discuss the limitations and future directions.

2 LITERATURE REVIEW

2.1 Model training

2.1.1 Seq2seq

Seq2seq modeling has been the synonym of encoder-decoder structures based on RNN. [9] Firstly, the encoder RNN handles an input sequence $x = x_1, \ldots, x_m$, where m denotes the number of elements and then obtains the corresponding hidden state $z = (z_1, \ldots, z_m)$. The decoder RNN uses z as an input and generates the output $t = (t_1, \ldots, t_n)$ one by one from left to right. When generating the output ti+1, the decoder will produce a new hidden state hi+1 via the previous state h_i, along with a representation vector g_i of the previous target language word y_i and a conditional input c_i based on the decoder input z. Based on the above generic formulation, many seq2seq models try to have novelties on RNN type or conditional input.

If not considering adding the attention mechanism, models may merely utilize the final encoder state z_m by setting the conditional input c_i equal to z_m for each i or initializing the input to the first decoder as z_m.[3, 9] Those models with attention mechanism calculated ci as a sum of (z_1,...,z_m) with different weights at each time step. These weights of the sum are the so-called attention scores, which enable the decoder to focus on the different parts of the input sequence when generating the outputs. To compute attention scores, it needs to compare each encoder's output state z_j with a combination of the last prediction y_i and previous decoder state hi. In the end, the results will be normalized to a distribution over the input sequence.

Many seq2seq model choose LSTM and GRU as RNN models. [5, 3] Both networks extend Elman RNNs with a gating mechanism which enables the memorization of information of previous time steps so as to process long-term dependencies. Recently, bi-directional encoders are proposed to capture both future and past contexts.[2] In practice, models with many layers are often equipped with shortcuts or residual connections.

2.1.2 BERT

BERT is designed to pretrain deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers.[4] As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.[4]

There are two steps in the BERT framework: pre-training and fine-tuning.[13] During pre-training, the model is trained on unlabeled data over different pre-training tasks. For fine-tuning, the BERT model is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks. BERT's model architecture is a multi-layer bidirectional Transformer encoder based on the original implementation. [10]

the BERT Transformer uses bidirectional self-attention. [4] The attention-head view visualizes the attention patterns produced by one or more attention heads in a given layer, as shown in the Figure1 (BERT). [11]

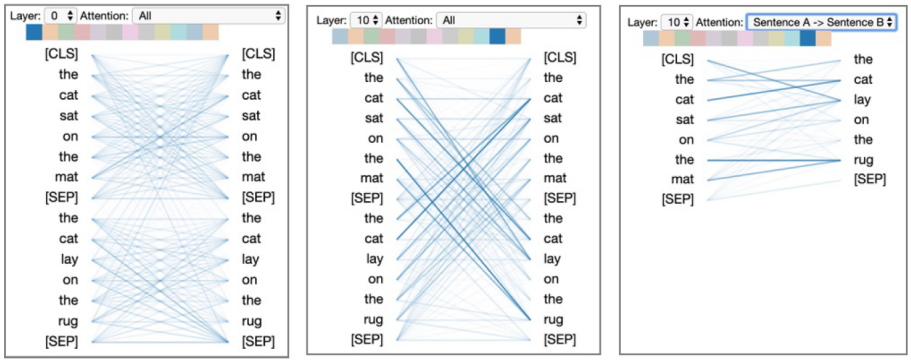

Figure 1: Attention-head view for BERT, for inputs the cat sat on the mat (Sentence A) and the cat lay on the rug (Sentence B). The left and central figures represent different layers / attention heads. The right figure depicts the same layer/head as the center figure, but with Sentence A to Sentence B filter selected

2.1.3 GPT2

GPT-2 is a large transformer-based language model with 1.5 billion parameters. GPT-2 is trained with a simple objective: predict the next word, given all of the previous words within some text. The diversity of the dataset causes this simple goal to contain naturally occurring demonstrations of many tasks across diverse domains.

GPT-2 is also a fully attention-based approach. The attention-head view visualizes the attention patterns produced by one or more attention heads in a given layer, as shown in the Figure2 (GPT-2).[11]

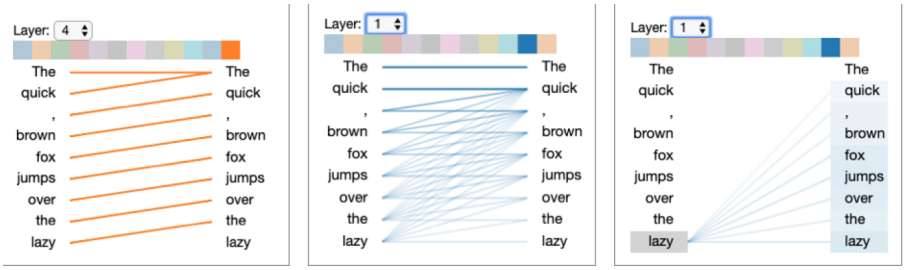

Figure 2: Attention-head view for GPT-2, for the input text The quick, brown fox jumps over the lazy. The left and center figures represent different layers / attention heads. The right figure depicts the same layer/head as the center figure, but with the token lazy selected

BERT and GPT-2 are the Transformer model, which uses a fully attention-based approach in contrast to traditional sequence models. By comparing the two models using in various papers. We think GPT-2 is better for text generation than BERT.

2.2 Evaluation metrics

Different from text classification, when the output of the task is text, the evaluation of the results becomes complicated. In this case, there's no simple answer about what metric is the best to evaluate your model. Therefore, we looked into several different metrics that focus on different aspects of the result. The metrics we looked into are BLEU, SARI, and SAMSA. Table 2.summarizes their major differences [1] .

Table 1: Major differences among different evaluation metrics

Metrics	Need reference	Correlation with human judgement	Coverage
BLEU	Yes	Fluency and adequacy	Comparing words
SARI	Yes	Simplicity	Comparing words
SAMSA	No	Fluency, adequacy, and structural simplicity	Comparing Semantic structure

2.2.1 BLEU

BLEU is a method for automatic evaluation of machine translation proposed by Papineni et al.[6] The score ranges from 0 to 1. The higher score indicates a higher precision in translation. This evaluation metric mainly measures the correctness of the text translation, and it's highly dependent on reference translations. The cornerstone of this metric is the "modified unigram precision" proposed by Papineni et al. [6] The modified n-gram precision scoring captures two aspects of translation: adequacy and fluency. [6] A translation using the same words (1-grams) as in the references tends to satisfy adequacy, and the longer n-gram matches would lead to higher fluency. [6] The basic unit of translation is sentence, and in order to compute a modified precision score, pn , for the entire test corpus, they use the following formula:

$$p_n = \frac{\sum_{C \in \{\text{Candidates}\}} \sum_{\text{n-gram} \in C} \text{Count}_{\text{clip}}(\text{n-gram})}{\sum_{C' \in \{\text{Candidates}\}} \sum_{\text{n-gram}' \in C'} \text{Count}(\text{n-gram}')}$$

In addition, a Sentence brevity penalty is applied in the metric. The details of the metrics is calculated as following [6]: First compute the geometric average of the modified n-gram precisions, pn, using n-grams up to length N and positive weights wn summing to one. Next, let c be the length of the candidate translation and r be the effective reference corpus length, thus the brevity penalty BP is computed by:

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases}$$

Then, the BLEU score is calculated by:

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

The ranking behavior is more immediately apparent in the log domain:

$$\log \text{BLEU} = \min(1 - \frac{r}{c}, 0) + \sum_{n=1}^{N} w_n \log p_n.$$

2.2.2 SARI

SARI is the first automatic metrics that are effective for tuning and evaluating simplification systems, it explicitly measures the goodness of words that are added, deleted and kept by the systems.[12] The metric compares the model's output to multiple simplification references and the original sentence. SARI has shown high correlation with human judgements of simplicity gain. For text simplification tasks, SARI is more suitable than BLEU, and it is the main metric currently used for evaluating sentence simplification models.

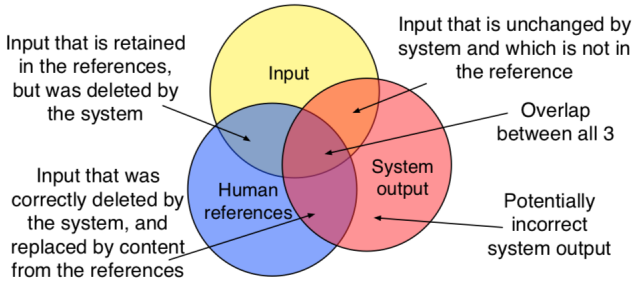

Figure 3: Metrics that evaluate the output of monolingual text-to-text generation systems can compare system output against references and against the input sentence, unlike in MT metrics which do not compare against the (foreign) input sentence. The different regions of this Venn diagram are treated differently with our SARI metric. [12]

2.2.3 SAMSA

SAMSA is the first measure to address structural aspects of text simplification (i.e. sentence splitting) proposed by Sulem et al. in 2018.[8] It doesn't require reference for the evaluation. SAMSA focuses on the core semantic components of the sentence, and is tolerant towards the deletion of other units, which means SAMSA will not penalize the model if the output is split into different sentences as long as the sentence components are correct.[8] SAMSA defines a structurally correct simplifs: (1) each sentence contains a single event from the input (UCCA Scene), (2) the main relation of each of the events and their participants are retained in the output.[8]The experiments performed by Sulem et al. show that SAlates well with human judgments when structural simplification is performed by the evaluated systems, in which case where strong metrics that assess lexical simplification quality like SARI failed. [8]

3 METHODOLOGY

3.1 Models

We implemented two lines of models for the task of text simplification. Both follow the general encoder-decoder framework.

The first model is a more "conventional" seq2seq model which utilizes RNN structure for both the encoder part and the decoder part. The second line of models attempts to utilize the available pre-trained GPT2 as the encoder and connects it to a customer decoder network. The implementation details are discussed in the following parts.

3.1.1 Seq2Seq using RNN from scratch

The first line of the models we use follows is a Seq2Seq model which obeying the encoder-decoder architecture, this special class of RNN architectures aims to solve complex language problems but not restricted to Machine Translation, Question Answering and Chat bot. Here we will deploy this model in solving the problem of text simplification.

Here we will use LSTM as our RNN model, the encoder will read the information from the input sequence through an embedding layer and LSTM mode, During this process, all the output Y_i, the predictions of LSTM model at each state will be discarded. Then encoder passes its last hidden state and cell state $[h_t, c_t]$ to the decoder.

In order to not missing any information from the training set, here we add 'START' and 'END' as two tokens to the start and end of the target sequence. The 'START' token will also help us to inform the decoder as a command of making predictions and 'END' token can be used as a stopping condition. Noted that there are two modes for the Decoder LSTM which are training mode and inference mode.

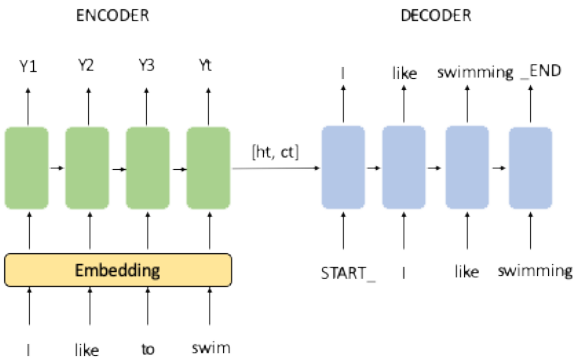

Figure 4: Encoder-Decoder architecture

Noted that there are two modes for the Decoder LSTM which are training mode and inference mode. During the training mode, initial state $[h_0, c_0]$ of the decoder are set to the final state of the encoder. Then the loss is calculated on the predicted outputs from each time step and the errors are backpropagated through time in order to update the parameters of the network. A technique called "Teacher Forcing" has also been used to make the training process faster and more efficient which will be explained later in next subsection. Whereas during the inference mode, we generate one word at a time, so the decoder LSTM will be called in a loop, initial state of the decoder will be set as the final states of the encoder and the initial input to the decoder will always be the token 'START'.

3.1.2 Pre-trained GPT2 encoder + Self-Attention Decoder

The second line of models we implemented still follows the general encoder-decoder framework. However, instead of training from scratch, we use a pre-trained GPT2 as an encoder. After the short text is encoded into context states (or more precisely, the attended word representations at all word positions in the short text), the encoded states are then passed to an attention-based decoder to generate simplified short text.

The entire encoder-decoder model is fitted to our dataset. Two different learning rates are applied to the encoder and the decoder parts of the model respectively. A smaller learning rate is used for GPT2 encoder so that it can be fine-tuned smoothly, while a larger learning is used for the decoder since the decoder is trained from scratch.

Attention Mechanism of the decoder network

The attention-based decoder is essentially a unigram language model that produces the probability of the next word, given the context states and the current word. Since the short text includes a large amount of information, the decoder needs to attend to the right positions very carefully. To that end, we use a *dot product* self-attention mechanism where the query is the concatenation of two vectors: 1) the embedding of the latest generated word and 2) the sum of embeddings of the previously generated words. The former helps the model to focus on the current words and finds what should come next. The latter provides context information about what has been said (generated) so far. This potentially has two effects. Firstly, it provides context for the decoder to finish the sentence. Secondly, it tells the decoder what information has been consumed (or summarized) already, and hopefully will help to reduce the problem of repeating generated information. Fig.5 illustrates the attention mechanism of the decoder.

Teacher forcing Teacher forcing is a method for quickly and efficiently training language models. It forces the decoder to take ground truth from a prior time step as input. It means that we discard the output from step t which should be the input to the next time step t+1. At the early stage of training, the decoder seldom predicts the right words. When the first few words are all wrongly predicted, the following prediction can hardly be correct and the error accumulates quickly.

Using teacher forcing can facilitate the training for the decoder quickly especially at the early stage. In our training schedule, the teacher forcing ration is 0.5, meaning that we alternate between using the ground truth and using the model prediction as the input for the next step with equal probability.

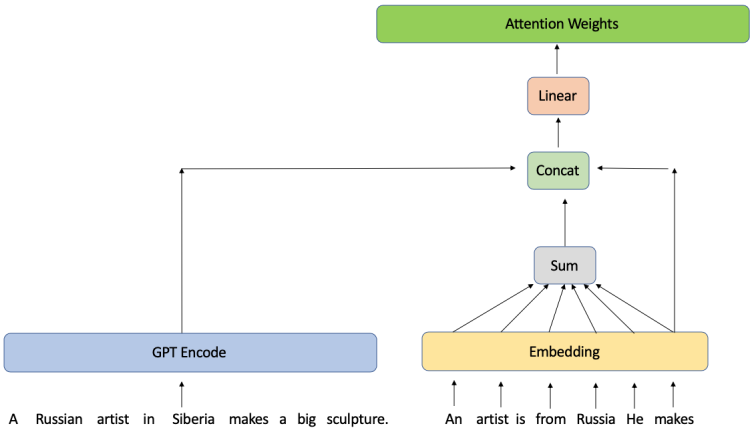

Figure 5: Self-attention mechanism of decoder

3.2 Dataset

Our dataset is scrapped from a news website[1] for beginner English learners and is not publicly available. The dataset contains triplets of short news, each triplet includes three versions of the same news written in different difficulty level by human editors.

In our models, we use the difficult version as input and aim to predict the easy version. The texts of medium difficulty are discarded. After prepossessing and cleanup, there are 2,588 valid pairs of short text. The data is further split into train and validation with a 90%/10% ratio.

3.3 Evaluation Metrics

All of the evaluation metrics will be deployed on EASSE [2](Easier Automatic Sentence Simplification Evaluation), which is a Python 3 package aiming to facilitate and standardise automatic evaluation and comparison of Sentence Simplification systems.

4 CONCLUSIONS AND FUTURE DIRECTIONS

4.1 Observations

We observe that none of the two experimented models provide satisfactory results on the evaluation metrics compared to state-of-art results reported on other datasets. Upon inspection of the predicted text, we found several issues.

Overfitting of Seq2Seq The Seq2Seq model seriously overfits on the training set before learning enough to make good predictions on the test set. There are likely two contributing factors: 1) The size of the dataset is too limited for language modeling; 2) The Seq2Seq model and its embedding is trained from scratch, so it overfits more easily to the vocabulary of the text we are using.

Gibberish Towards the End of the Text The generated text seems to be grammatically correct at the beginning of the text, but becomes gibberish towards the end. In hindsight, this is caused by the nature of the training process of language modeling. If the prediction beginning of the sentence or text is not correct, the error would accumulate and becomes worse at the end of the sentences.

Repeating Words A lot of the generated texts have words or short phrases that repeat themselves many times, usually some keywords of the text of high-frequency words in the English vocabulary. As a common problems for text generation this is also observed in texts generated by both of our models. It is especially sever for the GPT+Decoder model. To solve this problem, we employ a no repetition technique for the GPT2-Decoder model when generating text. This technique simply forces

[1]https://www.newsinlevels.com/
[2]https://github.com/feralvam/easse

the model the re-sample a different word if the word has appeared in the recent sampling. Table.2 shows that applying this rule improves the performance significantly.

Table 2: Major differences among different evaluation metrics

Metrics	RNN	GPT2+Decoder	GPT2+Decoder (no repeat)
BLEU	0.12	0.16	0.28
SARI	10.36	9.87	19.51
SAMSA	0.15	-0.98	0.23

4.2 Conclusions

We can notice that according to the BLEU metrics, GPT2+Decoder performs better than the RNN model. For SARI metrics, GPT2+Decoder (no repeat) performs the best whereas GPT2+Decoder performs worse than the RNN model. For SAMSA metrics, GPT2+Decoder (no repeat) also performs the best.

For the RNN model, since the encoder and decoder are all LSTM model, so when facing a long text task, the importance of the input at the beginning will be weakened or even forgotten by the model, that's one of the reasons why it didn't perform well. Also, when the dataset is not that large enough, the RNN model will overfit easily, we can even find out that some of the predicted sentences will keep exactly the same as the target sentence.

4.3 Limitations and Future Directions

The LSTM encoder and decoder may not be suitable for a long text simplification, also if the dataset is not that large, we need to fully use the advantage of pre-trained embedding and attention mechanism.

For GPT2+Decoder, although it does not overfit to the training set, the results are less than satisfactory. In addition to the limited size of the training data, there may be other reasons. Unlike the multi-head attention implemented in a typical transformer model, our custom-built decoder has only 1 attention head. Building a decoder with multiple attention heads likely improves the results significantly.

Since text simplification can be set as a lexical level and sentence level, so a two-stage simplification can also be taken into consideration in the future.

Acknowledgments

References

[1] Noof AlFear. Automatic text simplification evaluation aspects, 2019.

[2] K.and Bengio Y Bahdanau, D.and Cho. Neural machine translation by jointly learning to align and translate. 2014.

[3] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

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

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[6] Kishore Papineni, Salim Roukos, Todd Ward, and Wei Jing Zhu. Bleu: a method for automatic evaluation of machine translation. 10 2002.

[7] Irene Solaiman, Miles Brundage, Jack Clark, Amanda Askell, Ariel Herbert-Voss, Jeff Wu, Alec Radford, Gretchen Krueger, Jong Wook Kim, Sarah Kreps, Miles McCain, Alex Newhouse,

Jason Blazakis, Kris McGuffie, and Jasmine Wang. Release strategies and the social impacts of language models, 2019.

[8] Elior Sulem, Omri Abend, and Ari Rappoport. Semantic structural evaluation for text simplification. *CoRR*, abs/1810.05022, 2018.

[9] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[11] Jesse Vig. A multiscale visualization of attention in the transformer model. *CoRR*, abs/1906.05714, 2019.

[12] Wei Xu, Courtney Napoles, Ellie Pavlick, Quanze Chen, and Chris Callison-Burch. Optimizing statistical machine translation for text simplification. *Transactions of the Association for Computational Linguistics*, 4:401–415, 12 2016.

[13] Haoyu Zhang, Yeyun Gong, Yu Yan, Nan Duan, Jianjun Xu, Ji Wang, Ming Gong, and Ming Zhou. Pretraining-based natural language generation for text summarization. *CoRR*, abs/1902.09243, 2019.