Sentence Compression Report

Gavin Lynch, Beck Schemenauer, Jonah Ji

Introduction:

Sentence or text compression is the art of shortening text while maintaining its core meaning. Humans instinctively perform this task, effortlessly condensing long sentences or summarizing complex ideas. For example, we can easily reduce "the quick brown fox jumped over the lazy dog" to "the fox jumped over the dog."

While humans possess an innate understanding of language, computers lack such intuitive knowledge. However, computer-based sentence compression offers significant potential in various applications, including text summarization, machine translation, and information retrieval. This project aims to explore effective computational techniques for sentence compression. We will investigate three primary approaches: manual compression, a GRU-based encoder-decoder network, and a transformer-based compression model. By comparing these methods, we will identify the most promising approach for tackling sentence compression challenges, and evaluate its effectiveness.

Datasets, Tools, and Services:

We first explored the Google dataset that was provided with the project. The Google dataset had 200,000 examples in the training set with 10,000 in the set set. It mainly featured excerpts from various news articles gathered across the internet. The idea was to harvest news articles where the headline was similar to the first sentence and use the headlines as a basis for the compressed sentence (Katja and Yasemin). The reason was that often news headlines would preserve the core meaning of the article, removing any auxiliary details. In theory, this would make for an ideal dataset for sentence compression; however, in practice, it is not as effective as it seems.

Since the Google sentence compression dataset was based on news articles and their headlines as data, there were many instances where the sentences were excessively compressed and lost much of their original meaning. For example, a sentence like

“The USHL completed an expansion draft on Monday as 10 players who were on the rosters of USHL teams during the 2009-10 season were selected by the League's two newest entries, the Muskegon Lumberjacks and Dubuque Fighting Saints.”

becomes compressed to “USHL completes expansion draft.”

Here we can better illustrate the problems we were having with the dataset as there are many other similar examples in the training data like this one. First of all, the sentence that is the target for compressing is extremely long, and it could be split into 2 sentences. Second, the “compressed sentence” is far too short compared to the original sentence, it only contains 4 words and barely even qualifies as a sentence. In fact, it is more fitting to call it a phrase. So when we trained the AI on this dataset, naturally it learned the patterns of this data and started returning extremely short sentences when we tested it. After much deliberation, we decided that the Google dataset is unable to suit our needs and proceeded to search for a different one.

Our new dataset is a Microsoft text compression dataset. It has about 6,000 source texts, ranging from business letters and newswires to journals and technical documents sampled from the Open American National Corpus (OANC1). Each source text has up to 5 crowd-sourced rewrites which are constrained to a compression ratio, and these rewrites each include ratings done by human judges on the quality of the compressed text. One more unique aspect is that this dataset is the first to provide multi-sentence compression primarily with two-sentence paragraphs, and may yield insights into text compression at higher levels. Compared to the Google dataset, the Microsoft dataset is a major step up in quality, especially with large amounts of human involvement. To properly compare the quality in a simple manner, here is an example source text:

‘“Except for this small vocal minority, we have just not gotten a lot of groundswell against this from members,” says APA president Philip G. Zimbardo of Stanford University.’

This is compressed into 4 different shorter sentences:

- “‘Except for this small vocal minority, we have not gotten a lot of groundswell against this,’ says APA president Zimbardo.”
- “‘Except for a vocal minority, we haven't gotten much groundswell from members,’ says Philip G. Zimbardo of Stanford University.”
- “APA president of Stanford has stated that except for a vocal minority they have not gotten a lot of pushback from members.”
- “APA president Philip G. Zimbardo of Stanford says they have not had much opposition against this.”

As you can see, this dataset does a much better job with compressing sentences, and it is more realistic as well, as these may actually be how a real human would write compared to a simple phrase like the Google dataset. While it’s great that there are many things inside the dataset we can use, there is no need to use all parts of it. We don’t need to have 4-5 source text to compression pairs for every source text and we don’t really need the judge ratings either. For our purpose, we will only select the shortest compressed text in each text-to-compression group for training.

In our three main approaches, we used quite a few different tools and services to accomplish our compression task. For our manual compression, we used NLTK's tokenization methods (word_tokenize, TreebankWordDetokenizer). For grammar checking, we used language_tool_python the PyTorch with a BERT model (textattack/bert-base-uncased-CoLA) from Hugging Face's Transformers library. For building and handling probabilistic context-free grammars (PCFG), we relied on NLTK's PCFG functionality and treebank corpus.

We used pandas and PyTorch to train our GRU encoder-decoder model, primarily the PyTorch neural network library, and Pandas for data preprocessing and manipulation. Many specific features like special token insertion and training epoch evaluation had to be coded by hand.

For our transformer-based compression task, we used Meta's BART model, first published in July 2020 (Lewis and Liu). This model is imported from Hugging Face's Transformers package, allowing us to easily access and finetune the model. This model, although a couple of years old now, was a nice lightweight and easily trainable option for our task, while still performing very well. Besides Hugging Face and BART, we used Pandas and the datasets package for preprocessing, as well as data manipulation. With these tools, we were able to create our most accurate and well-rounded sentence compression model.

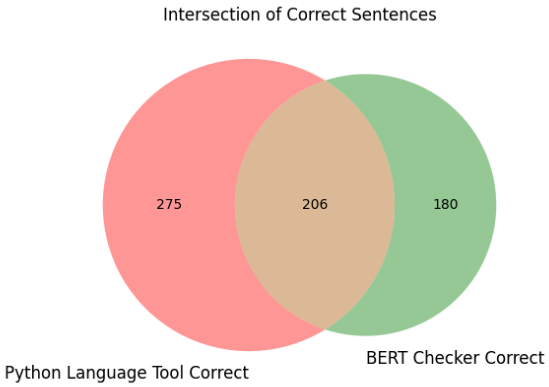
Manual Compression:

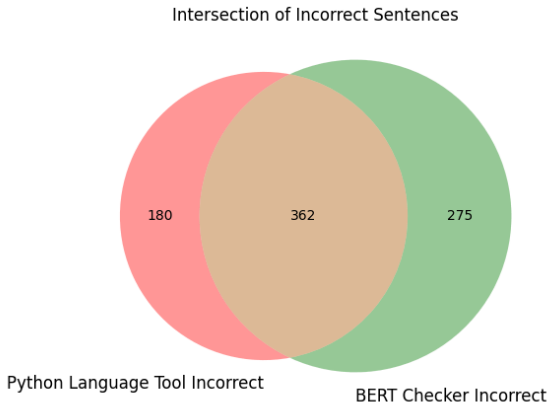
After attempting to implement a strict rule-based compression system, we decided to explore probability-based compression rules using PCFGs (Probabilistic Context-Free Grammars). To construct the PCFG, we used annotated parse trees from the NLTK Treebank corpus. We first extracted production rules, where each syntactic constituent (e.g., `NP` for noun phrase, `VP` for verb phrase) was explored to identify its child sequences, such as `NP -> DT NN`. Terminal nodes, which represent individual words, are treated as productions as well, such as `DT -> "the"`. However, in our grammar, we replace actual words with part-of-speech (POS) tags as terminals, meaning 'the' is replaced by `DT`. This allows us to understand sentences that include words outside of our lexicon. We then recorded the frequency of each production rule for a given parent, calculating probabilities by normalizing these counts so that the probabilities of all rules for a parent sum to 1. There were some special cases, such as symbols or non-standard characters (punctuation or terminals like `-NONE-`). These were replaced with valid syntax resolved to ensure compatibility with the grammar parser. The resulting rules, formatted as PCFG-compatible strings (e.g., `NP -> DT NN [0.8]`), allowed us to convert the PCFG dictionary into an NLTK PCFG object to hold our grammar.

The next technique involved in this manual approach involved the language_tool_python library and a BERT-based grammar checker ("textattack/bert-base-uncased-CoLA") to check if a given subset of a sentence was grammatically correct. Subsets were generated with the following logic:

"Bob is fast" -> ["Bob", "is", "fast", "Bob is", "Bob fast", "is fast", "Bob is fast"]

By combining this with our PCFG implementation, we could hypothetically extract probable, shorter but still grammatically correct sentences. This approach did not work, however, as most grammar-checking models used were unable to classify incorrect sentences as ungrammatical. As seen below, the results were not consistent enough to use.





Correct Sentences by Both:
The quick brown jumps over the dog
The quick fox lazy
The brown fox dog.
The fox jumps the
The fox jumps over the lazy

Incorrect Sentences by Both:
brown jumps over the lazy.
quick fox over
jumps over the.
quick brown lazy.
brown fox jumps the lazy

Ideally, the models would agree significantly more than they did. The next step would have been to not only rank sentences by probability, grammatical correctness, and length but also by semantic meaning. However, based on the time spent working on this approach, it was best to abandon it in favor of more promising techniques.

GRU-Based Encoder-Decoder Network:

In this approach, we decided to explore how a traditional encoder-decoder neural network would fare in doing sentence compression. Encoder-decoder networks are well-suited for this task due to their ability to process sequential data and generate output sequences. The encoder component processes the input sentence, capturing its semantic and syntactic information. The decoder component then leverages this information to produce a compressed output sentence.

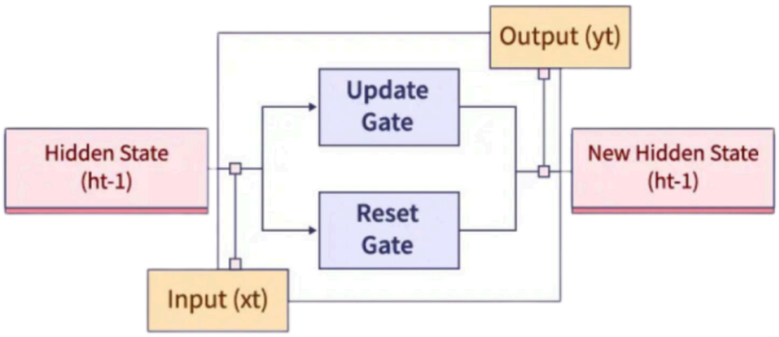


Figure - Architecture of a GRU

To enhance the performance of our encoder-decoder network, we incorporated Gated Recurrent Units (GRUs). GRUs are a type of Recurrent Neural Network (RNN) that was introduced in 2014, it is particularly effective for handling sequential data, such as sentences. Compared to Long Short-Term Memory (LSTM) networks, GRUs have a simpler architecture with fewer gates (update and reset gates), making them computationally efficient. The update gate controls information passed to the future, and the reset gate controls information that is forgotten. This streamlined design allows GRUs to process information more efficiently, leading to improved performance in sentence compression tasks.

Our model is a sequence-to-sequence architecture comprising an input layer, an embedding layer, an encoder GRU, a decoder GRU, a fully connected layer, and an output layer. The data is preprocessed by first performing some simple data cleaning, removing data with missing values, then selecting the shortest compressed sentence for each source text, tokenizing it, and inserting special tokens like <sos> <eos> to indicate the start and end of sentences, and <pad> for padding to work around the varying text lengths. A vocabulary is built from the tokenized words from the source and target data, which become the input and output layers where the size of the vocabulary corresponds to the size of the layers. Theoretically, this architecture is well-suited for sentence compression as it can process sequential data and generate compressed output sequences.

We trained the model using the PyTorch NN library, which provides various useful functions for neural network training. However, despite its theoretical promise, the model encountered several challenges:

1. Slow Training:

- Even with GPU acceleration, the model exhibited slow training speeds, requiring 20 minutes to 1 hour to train 10 epochs.
- This significantly hindered the experimentation process and limited the number of iterations we could perform.

2. Overfitting:

- The model tended to overfit the training data, as evidenced by the decreasing training loss and the relatively constant evaluation loss.
- The evaluation loss remained stubbornly high, hovering around 7.9-8.1, indicating a significant gap between the model's performance on the training and evaluation sets.
- The high entropy loss suggests that the model's predictions were highly uncertain and inaccurate, further indicating overfitting and a lack of generalization ability.

3. Limited Vocabulary:

- The model's vocabulary was constrained to the training dataset, which only has around 6,000 source texts, limiting its flexibility and effectiveness when encountering unseen words.
- This restricted the model's ability to handle diverse and real-world text.

4. Long-Range Dependency Issues:

- RNNs, including GRUs, struggle with capturing long-range dependencies within sentences, which can hinder their performance in sentence compression.
- This limitation can lead to the loss of important contextual information, especially in longer sentences.

Despite our efforts to optimize hyperparameters, such as tweaking dropout and learning rates, batch size, and layer quantity and complexity, the model's performance remained unsatisfactory. Therefore, we decided to explore more advanced techniques, such as transformer-based models, which have shown superior results in natural language processing tasks.

Transformer-Based Compression:

Our next model exploration involved pre-trained transformers, which we knew would likely be the most suitable for this task. Making use of a pre-trained system would allow us to skip most of the steps of creating a language model, and simply fine-tune it on the small dataset that we did have. Because of this, and the high-quality outputs that transformers are now able to give, we knew that this would likely be our most optimal approach.

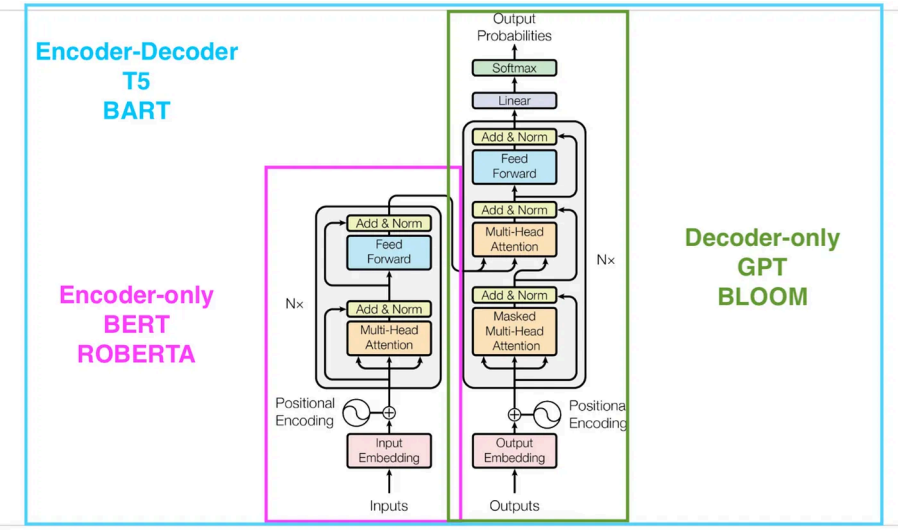


Figure - Architecture of BART

We started out looking for a relatively lightweight, and still highly capable open-source model that we could use to fine-tune on our dataset. We looked through most of Meta's open-source models such as Llama, and even some other models as well (such as GPT-2), but eventually settled on BART from Meta. BART is unique as it combines a bidirectional approach (like BERT), with a GPT-like approach (like Llama or GPT-2). This allows the model to bidirectionally analyze the input text, potentially capturing more dependencies between tokens than with a strictly unidirectional model like GPT-2 (Lewis and Yinhan). This model is also fairly lightweight, only coming out to about 600MB per trained model, much less than some other offerings by Meta. Although it is now a couple of years old, it was still able to produce great output and train fairly quickly on our lower-powered machines. Below is an example compression of a long sentence, taken straight from the output.

Input:

"As climate change continues to pose a significant threat to our planet, governments and industries are increasingly adopting renewable energy solutions to mitigate environmental damage and promote sustainable development."

Output (Compression Level 3):

"Governments are adopting renewable energy solutions to mitigate environmental damage."

A big benefit of this transformer model is its ability to repeatedly compress a sentence without losing readability or the grammatical correctness of a sentence. This combined with the lower compression ratios afforded to us by the MSR dataset allowed for multiple compression levels. These compression levels started out as an input to the transformer, but we quickly realized that the simplest way to create our desired levels is to run our model multiple times. The compression level directly corresponds to the number of runs for each sentence. For example, the sentence above had to be processed 3 times by our model to generate that level of compression. While not the most computationally efficient, this method gave the best and most consistent results we could find.

Evaluation:

Analyzing the efficacy of a sentence compression model is a complicated and often subjective task. It depends greatly on what your end goal with the model is. If you are looking for the most efficient and short compressions, you might want to optimize for compression ratio. If you are looking for a model that maintains the most information possible, you might want to optimize for a combination of compression ratio and semantic similarity. Most likely, you will land somewhere in between the two extremes, which makes it very hard to come up with an objective measure of how good a model is at compressing text.

For our purposes, we used four different comparison techniques for the different methods and models that we trained. First was a simple compression ratio, which we aimed for an average of around 0.75-0.85 (a compressed sentence is 75-85% of the original text length). Next was cosine similarity, which we used to determine how similar the two texts were to each other, and how much of the information was kept. We didn't have a set goal for this metric, just the higher the better. The third metric that we tested was a readability metric, measuring how well the output maintained grammatical correctness. To do this, we used perplexity, calculated from the loss of GPT-2. The higher the perplexity, the more "uncertain" the model was when predicting the next tokens in the compressed text, and therefore the less fluent or grammatical the text should be. Lastly, the most important metric for us was how we felt about the output of the model. It's one thing to try to quantify the results, but a simple analysis of the results by us was one of the most effective ways to see if it was performing better or worse than previous models.

Results of Tests:

While we tested out three main approaches, we decided to run our conclusive experiments on the transformer, as it showed the most promise through the training and early testing cycles. On the bottom, you can see the average cosine similarity and average compression ratio calculated over different compression levels. These results were averaged over a set of 20 sentences, and show how the sentences change as compression level increases. While we see that the cosine similarity between output and input goes down, this is expected as the sentences

necessarily lose information as you remove words. It is nonetheless a good result, as we were able to keep our cosine similarity over around 0.45, indicating that we still preserved a good amount of the meaning. The compression ratio also decreases steadily and starts to reach a minimum of around 0.35. This minimum and rate of descent all depends on the length and verbosity of the input text. Lengthy input will be able to decrease more than short input text, but this was an average result.

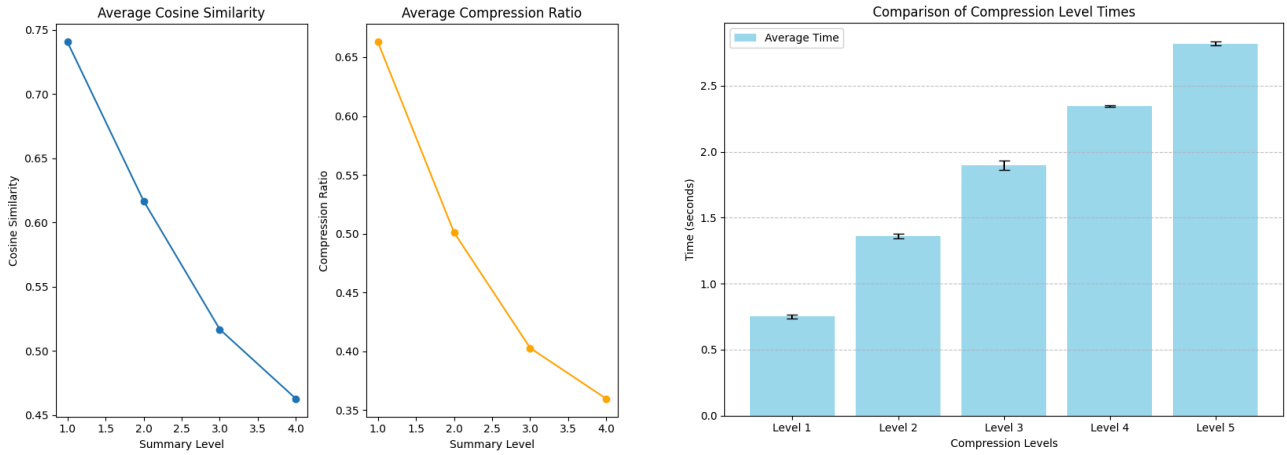


Figure - Compression metrics

Our last analysis involved sentence compression level runtimes. The benefit of using BART is the smaller model sizes and relatively quick inference time. You can see above the nearly linear relationship (as we would expect) between the compression level and runtime. The longest our compression will take to run on a standard laptop is around 3 seconds, although this can vary based on platform and specifications.

Shortcomings:

The Google Dataset had several issues that hindered effective sentence compression. Target sentences were often excessively compressed, losing critical details and reducing the outputs to short, phrase-like structures that barely retained the original meaning. Additionally, the dataset contained unrealistic targets, as the compressed results frequently resembled disjointed phrases rather than grammatically valid sentences, making it less useful for training meaningful compression models. This inherent bias led to models trained on this dataset producing outputs that were overly short and often uninformative.

For the manual compression approach, inconsistent grammar checking emerged as a major drawback. Grammar-checking models, such as those based on BERT, struggled to reliably identify ungrammatical sentences, resulting in unreliable classification. Moreover, the complexity of integrating factors like probability, grammatical correctness, and semantic meaning made sentence ranking computationally intensive and challenging to implement effectively. These limitations, compounded by the tools' shortcomings, led to the eventual abandonment of this approach in favor of more promising methods.

The GRU-based encoder-decoder network also faced significant limitations. Training the model was slow, even with GPU acceleration, taking between 20 minutes and an hour for every 10 epochs, which limited the ability to experiment with different configurations. The model exhibited noticeable overfitting, achieving high training accuracy but poor evaluation performance, with losses ranging between 7.9 and 8.1. Additionally, the small dataset size (6,000 texts) restricted the model's vocabulary, reducing its generalizability to diverse real-world text. GRUs also struggle with capturing long-range dependencies within sentences, negatively impacting their ability to comprehend and compress complex text effectively.

Conclusion and Future Work:

In our exploration of sentence compression techniques, we experimented with both manual and transformer-based methods, using two different datasets. The Microsoft text compression dataset was more suitable than the Google dataset due to having more realistic and varied data. Among the models we implemented, Meta's BART transformer was the most effective, allowing for both compression accuracy and semantic retention. We achieved an evaluation loss of 0.19, compared to a higher loss of the GRU model, and the inconclusive results of the grammar-based implementation.

In terms of future work and expansion, reinforcement learning could improve our model's adaptability, enabling it to optimize several factors such as length, semantic similarity, grammatical correctness, and sentence probability. Additionally, we could expand the scope of this work to include paragraph or document-level compression and summarization. These approaches could be tailored to fit different contexts, such as academic papers, novels, or transcripts.

References:

Katja Filippova and Yasemin Altun. 2013. Overcoming the Lack of Parallel Data in Sentence Compression. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1481–1491, Seattle, Washington, USA. Association for Computational Linguistics.

Microsoft. (n.d.). *Microsoft/msr_text_compression · datasets at hugging face*. microsoft/msr_text_compression · Datasets at Hugging Face. https://huggingface.co/datasets/microsoft/msr_text_compression

Lewis, Mike, and Yinhan Liu. "Bart: Denoising Sequence-to-Sequence Pre-Training for Natural Language Generation, Translation, and Comprehension." Facebook AI Research, ai.meta.com/research/publications/bart-denoising-sequence-to-sequence-pre-training-for-natural-language-generation-translation-and-comprehension/. Accessed 8 Dec. 2024.