Implementation of Text Simplification Models in the Domain of College Admissions

Completed for the Certificate in Scientific Computation and Data Sciences
Spring 2022

Maximus Chu
Bachelor of Science
Computer Science
College of Natural Science

Junyi Jessy Li
Assistant Professor
Department of Linguistics

Implementation of Text Simplification Models in the Domain of College Admissions

The field of natural language processing has seen remarkable growth and progress ever since its beginnings, but especially in the last decade, where new methodologies, techniques, and models are pushing the boundaries of what we can achieve within this field; this progress has been observed in all domains within this field, and this includes text simplification as well. However, even as all these breakthroughs and studies are being conducted, there is always room for improvement or new research to be explored in areas that previously haven't had much coverage or exposure. This trend can be seen within the domain of text simplification; traditionally, most of the work for text simplification has been conducted on training and improving the implementation of models using either news articles or Wikipedia articles, where training, testing, and application of these models have largely been used on these described articles. This works well for the most part in general for a lot of everyday tasks that can be more easily generalized, but quickly falls apart whenever encountered with any material that is more specialized. One of these is within the domain of college admissions, which traditionally has not been a huge field of focus in terms of research when compared to other topics. However, it certainly should be a bigger area of focus for research applications, for it is a largely untapped area of study while containing an abundance of more specialized material that millions of people encounter annually. The vast amount of paperwork one must process when applying for colleges is already a daunting task, and the complexity of these forms only further hinders one's progress. In order to potentially relieve the difficulties of this process, we investigate the application of text simplification within this domain, where simplifying these texts can lessen the dense jargon encountered. By reducing the complexity of sentences while still retaining its overall meaning, the application of text simplification within this field can be greatly beneficial, not only for potentially improving the process of college admissions itself, but also for the domain of text simplification in general as well.

Introduction

Although text simplification has made plenty of progress up to the present, there is still a lot left to be desired. As described before, although it can work moderately well for a lot of more simple tasks where easily generalized material is seen, the more complex and specialized the tasks and materials are, the harder it is for models to successfully simplify these materials, since these texts are not as easy to generalize. In order to improve the performance within these specialized fields, research and studies must be extensively and specifically be conducted on these specific fields in order to yield better results when these certain types of materials are encountered. College admission texts certainly falls under the category of being more specialized and difficult to generalize, where both the jargon and concepts encapsulated within these texts are not ones that one encounters every day. Because of how so many people encounter college admission texts annually, one might believe that much research and studies would have already been conducted within this field by now, but unfortunately this has not been the case, which has often led to subpar results whenever most current models need to generate the text simplification of any materials that fall within this category. In order to attempt to address this glaring issue of

the performance within this domain filled with untapped potential, we attempt to improve the implementation of text simplification models in the domain of college admission texts by focusing solely on working with materials that fall within this domain; all of our coding implementations in our study are done in Python.

Although one can easily identify and point out how performance is lacking in terms of text simplification within a certain area, improving the performance in college admission texts is easier said than done. College admission texts are inherently more complex than most regular texts that one would see every day, so it is already more difficult to simplify these texts based on the complexity alone, and together with the fact that there hasn't been many prior studies in terms of text simplification applied within this domain, it will almost certainly be difficult to yield performance that is leaps and bounds better than many of the current models out there. Nevertheless, we were confident that by focusing solely on materials from this field, we predicted that we would be able to yield significant and noticeable improvements for the performance of the simplification of these college admission texts. This prediction proved to be correct, where the difference between performance of the general models without any exposure to the college admission materials compared to that of those that were exposed and trained to the materials were significantly noticeable, which helps to illustrate not only how improvements in text simplification of these more specialized texts can be found with dedicated research towards college admissions texts, but also that this can be applied in general to any domains with more specialized texts, and that we should branch outside of focusing mainly on news and Wikipedia articles.

Materials and Methods

The first thing to do when attempting to conduct research to address a problem is to collect data, and for this study it was no different. Addressing the issue of performance of text simplification on college admission texts requires a large amount of texts that can be worked with in order to conduct any studies, so as a result these texts were the main requirement in terms of needed materials; since there hasn't been too many prior studies conducted in terms of application of text simplification in college admissions, there isn't too much else to actually build off of, so there weren't any other materials to be gathered since the study was mostly from scratch. College admission texts were gathered from online, from both public and private universities across the United States, all of which were from the 2019 application cycle. Over three hundred of these admission texts were gathered, all from different universities, and these texts were then manually simplified by hand as the basis of the research. Texts were simplified in a variety of ways, including deletion, substitution of more difficult vocabulary with easier ones, and combining multiple sentences into one through summarization, and the original and simplified texts were then separated into their respective folders, each assigned a unique number starting from one in ascending order, so that they are easily accessible and able to be modified later, both for data cleaning as well as for the training and experiments.

Data Filtering and Cleaning

Not all of the texts were suitable to be worked with within the context of this study of text simplification, where some texts were simplified too much, while other ones were simplified too

little. In order to only keep the texts that were suitable for training and analysis, each pair of original and simplified texts were compared to each other, where the ratio of each pair's simplified to original text length was computed in terms of character length, and only the ones that fell into a suitable chosen range were kept; the suitable chosen range of simplified to original text length chosen was between 0.3 to 0.7 inclusive, and after computing the ratios and filtering the texts, only a hundred plus pairs were left, or in other words two thirds of the initial data were eliminated.

data_filter.py

```
number_of_files = 341
lower_bound = 0.3
upper_bound = 0.7


# Filter out files that aren't within percentage range
def filter_by_percentage():
    files = []
    for current_file in range(1, number_of_files+1):
        f = open(
            f"Data/2019 ADM TXTS/{current_file} - ADM.txt", "r", encoding='utf-8')
        f_simple = NULL
        try:
            f_simple = open(
                f"Data/2019 ADM TXTS SIMPLE/{current_file} - ADM - ACCEPTED.txt", "r", encoding='utf-8')
        except OSError as e:
            continue
        current_percentage = len(f_simple.read())/len(f.read())
        if(current_percentage >= lower_bound and current_percentage <= upper_bound):
            files.append(current_file)
    return files
```

After filtering out the files, each file was then reformatted, so that each sentence was its own line within its document, for the models described later on all had sentence level simplification, so as a result putting each sentence on its own line allowed tasks later on such as aligning sentences to be much easier.

alignment.py

```
# Use for reformatting file so that one sentence per line
def split_sentences(f, location):
    file_format = open(
        f"Reformat/{location}/{file_number}.txt", "w", encoding='utf-8')
    for index_original, line_original in enumerate(f):
        if("." in line_original):
            sentences = nltk.tokenize.sent_tokenize(line_original)
            for current in sentences:
```

```
            file_format.write(str(current) + '\n')
        else:
            file_format.write(str(line_original))
```

Manual Sentence Alignment of Sentences in Original and Simplified Texts

After the texts had been filtered and cleaned, their sentences were then manually aligned, which is necessary both because they can be used as references later for comparisons in term of automatic alignment accuracy, as well as for testing the models' accuracies as well. Manual alignment means finding the corresponding sentence pairs that match up together in the original and simplified texts. Sentences can either have a one-to-one match, or multiple sentences can all match to a single sentence if the multiple sentences were summarized and compacted into a singular sentence in the simplified text. The previously discussed step of putting sentences into their own individual lines now becomes significant, for by putting each sentence in its own line it is much easier to track which sentences are aligned to which sentences, based on the index number that was assigned to each sentence using their line location within the document. Initially, we used a zero-based index that needed conversion in our heads from the one-based indexing in editors, so instead we just created new documents with the indexing above each sentence to ease the process.

index.py

```
for file_number in file_numbers:
    file_original = open(
        f"Simplified/{file_number}.txt", "r", encoding='utf-8')
    file_indexed = open(
        f"Indexed/Simplified/{file_number}.txt", "w", encoding='utf-8')
    for index, line in enumerate(file_original):
        file_indexed.write(str(index) + '\n')
        file_indexed.write(line)
```

With the text sentences indexed, we then aligned the sentences within the original and simplified document pairs, by writing down the pairs in the order of (original sentence, simplified sentence) using their indices in their respective documents, while the respective sentences themselves were also transcribed into the alignment documents as well. If a sentence did not have a corresponding sentence, the index location where an index would go for the missing sentence was simply marked with a N to indicate null, and the line where the sentence would be transcribed was also left blank as well; for sentences that matched to multiple sentences at once, the indices of the document with multiple sentences were marked together in parentheses, while the sentences themselves were combined and transcribed onto a single line. The texts were manually aligned in this manner, and they were aligned in both directions, meaning that we aligned the sentences from the original texts to the simplified texts and vice versa.

Template and example of aligned sentence pairs:

Template:
(Original sentence index, simplified sentence index)
Original sentence
Simplified Sentence

Example:
0, 0
Step 1: Apply for Admission (Line from the original file)
How to Apply: (Line from the simplified file)

Finally, after aligning the sentences, we also added a "<INFO>" tag to the end of all sentences that we deemed suitable to be classified as information, which included sentences composed of information such as addresses, emails addresses, dates, and the standardized testing identification codes of each respective university. By marking these sentences, we could later on easily filter out these sentences if so desired simply by searching for the "<INFO>" tag within each sentence, and because these sentences did not contribute much to the simplification efforts itself and could possibly skew results, we in fact did filter out these sentences later on.

N-Gram Analysis and Automatic Alignment

With the data now fully manually prepared, having been filtered, cleaned, and aligned by hand, we now proceeded to conduct experiments on the data. Before beginning, all texts were first simplified through the ACCESS model (Martin et al., 2019), which is a model that was trained on the WikiLarge (Zhang et al., 2017) test set that is encompassed of Wikipedia articles. Each original college admission text was used as input for the model, which then generated a simplified version of each text. Because the simplification for the ACCESS model is done on a sentence level basis, there is no manual alignment needed as preparation for analysis, since it is still a one-to-one correspondence for the sentence indices, and these sentences that were simplified were also added into the documents of the previously discussed aligned sentence pairs. After the text simplification was generated through the ACCESS model, we then conducted n-gram analysis on these sentence pairs, for the unigram, bigram, and trigram of each sentence pair.

Before the analysis could occur, the sentences had to be prepared so that it was suitable for being able to be properly analyzed. Steps included stripping any extra consecutive spaces that might have occurred, removing all punctuation, as well as filtering out all lines that are blank. After once again reformatting the data, we were then able to conduct n-gram analysis on the sentences, which we did by splitting each sentence into their individual words, then producing the n-gram depending on which value of n was desired, which for our experiments were values of one, two, and three. No matter the value of n, these values were then combined without spaces in order to produce the desired n-gram and returned as strings.

alignment.py

```
# Produce ngrams for desired sentence.
def n_gram(n, sentence):
```

```
    sentence = sentence.split()
    output = set()
    for i in range(len(sentence)-n+1):
        current = sentence[i:i+n]
        result = ""
        for word in current:
            result += word
        output.add(result)
    return output
```

With the n-grams generated, we want to be able to examine the n-gram overlap of aligned sentences. However, because one cannot always have access to manually aligned sentences pairs done by humans, so we also needed to devise a method of being able to automatically align sentences for the purpose of comparison. As a result, before conducting the n-gram analysis, we instead then formulated a method to automatically compute sentence alignments between an original and simplified text. The method we devised to do this computation was to check every sentence in one text to every sentence in the corresponding text within the pairs of original and simplified texts. For every sentence pair that was compared, the similarity between the two sentences was computed, and if it was above a certain threshold, then it would be considered an alignment; otherwise, it was ignored, and this continued until every sentence was compared to every sentence in the other document.

However, we quickly ran into some issues with this approach, where if we simply kept every sentence pair that had a similarity score above the chosen threshold, then it would still often yield too many automatic alignments. Therefore, to help limit this issue, for every current sentence that was being compared in the first text to all the other sentences in the other corresponding text, a maximum score and the sentence that produced this maximum similarity score in the second paired text was kept track of, and at the end only the sentence with the maximum score was kept; if no sentence pairs met the threshold, then no alignments were produced for that respective sentence.

For the computation of the similarity scores between two sentences themselves, at first, we attempted to use a simpler formula for computing similarity, which was the Jaccard Similarity Index. The Jaccard Similarity Index is computed by dividing the intersection of the words in the two sentences by the union of the words in the two sentences. For the score threshold, after some trial and error, it was found that a suitable threshold for the Jaccard Similarity Index would be 0.4. However, some of the alignments produced left more to be desired, so we then calculated the similarity score using Sentence-BERT (Reimers et al., 2019), which derives semantically meaningful sentence embeddings that can be compared using cosine-similarity, and as a result yielding better results due to the fact that it was designed specifically for comparing sentence embeddings. After again experimenting with the threshold level, this time for Sentence-BERT we found that 0.7 was a suitable threshold for the minimum score for sentence similarities.

alignment.py

```
for index_original, line_original in nonblank_lines(file_original):
        max_score = 0
        max_index = NULL
```

```
final_simple = NULL
multiple_lines = []

# Sentence-BERT
emb1 = model.encode(line_original)
for index_simple, line_simple in simple_lines:
    emb2 = model.encode(line_simple)
    current_score = util.cos_sim(emb1, emb2).item()
    if(choose_multiple_sentences == False):
        if(current_score > max_score):
            max_score = current_score
            max_index = index_simple
            final_simple = line_simple
```

Finally, we further refined our automatic alignment method in several ways, where all of these refinements could be toggled on or off via Boolean values, as well as being able to adjust the threshold values when applicable. As discussed before, we implemented the option for checking for the "<INFO>" tags in sentences that we had marked before if so desired, where for any sentence that contained the tag would automatically be ignored; by doing so, it helps to reduce potential skewing of results for sentences that either would not have been simplified or would not have been suitable for n-gram overlap analysis. Checking for the presence of these tags was quite simple, where one just needs to check if the current string contains the tag and ignore the sentence if the tag is detected.

Another improvement that was implemented was to have the option to filter out sentences by length, where if sentences didn't meet a required length in terms of number of words, it too would be ignored like any sentences with the "<INFO>" tag. Any sentence that wasn't above a certain threshold for length would be filtered, which we set as a length of five. For this implementation, the length of the sentence was checked by splitting the sentence into individual words, then checking the length of the array that contained these individual words that had been tokenized, to determine the total length of the sentence, where any sentences that failed to meet the threshold were ignored.

Finally, we also implemented a method of dynamically including more sentences depending on the context of the current sentence. Before, only the best potential sentence alignment with the highest score that exceeded the set threshold was kept, while all other potential alignments were eliminated automatically since only the current top score and sentence was kept in memory. However, this can eliminate many potential suitable alignments, especially in cases where one sentence would be aligned to multiple sentences if done manually. As a result, the option to dynamically include multiple potential sentences was added. Simply adding all sentences that exceeded the threshold proved to be problematic as previously discussed, so as a result dynamically adding sentences could help limit the potential overflow of too many alignments, while also still allowing for sentences to be aligned to multiple sentences. The dynamical addition of sentences was devised and implemented by checking for the difference in score between the last added highest score and the current potential candidate score. This requires first storing all potential alignments that exceed the threshold in an array, and after all potential alignments for the current sentence are calculated, the array is then sorted by highest to lowest score. The highest score is automatically added first while also being set to the current

highest score, and for any score after that, the difference between the current highest score and the score after the current highest score is calculated. If it is within a difference score, which we set to 0.1, then this next score is added as an alignment and set as the current highest score, and the process continues; otherwise, the process ends when the difference is too great, and all potential alignments afterwards are discarded. By doing this, this allows us to keep any potential alignments that are close in similarity to the highest scoring ones, while still filtering out ones that are significantly inferior in terms of similarity, and we also set an option of having an upper limit in the amount of total potential alignments that can be added per sentence, thus allowing us to having flexibility in tuning parameters to find the best setting for producing the highest accuracy alignments as possible.

alignment.py

```
count = 0
        result = sorted(multiple_lines, key=get_score)
        result.reverse()
        last_score = 0
        for current_line in result:
           if(count >= multiple_limit):
              break
           if(count == 0):
              alignments.append(current_line)
              last_score = current_line[4]
           else:
              current_score = current_line[4]
              if((last_score - current_score) < multiple_difference):
                 alignments.append(current_line)
                 last_score = current_score
              else:
                 break
           count = count + 1
```

With a satisfactory method of producing automatic sentence alignments, we then proceeded with the n-gram analysis, based on the sentence alignments from our previously manually aligned documents. Using the manual alignments as the input for the sentence pairs for the n-gram analysis, the unigram, bigram, and trigram overlap of all the manual sentence alignments from all the texts were calculated, to determine how much new vocabulary is introduced during the text simplification process compared to manual simplification, with the results of the averages across all files in the table below.

Table 1

Unigram	Bigram	Trigram
0.2805307354179571	0.4806695283405733	0.5925992379134858

From Table 1, it can be observed that the amount of new jargon introduced through simplification is enough to be significant; this indicates that simpler vocabulary is being introduced, and that the simplification is actually taking effect other than just deleting portions of

sentences, since the n-gram overlap is with respect to the simplified sentences. With the simplification process shown to be working as well as being able to generate automatic alignments for fine-tuning models, we proceed to measure the accuracy of the alignments themselves.

Measuring Accuracy

After analyzing the results of the n-gram overlap, we also wanted to determine how accurate the automatic alignments generated are compared to our manually alignments, to see if the alignments that were automatically generated would match those that were done by hand. To determine the accuracy of the alignments, we compared the automatic alignments generated for each document to our manual alignments, and measured the accuracy through computing the precision, recall, and F1 score of each text. Because we had previously noted the sentence indices of the sentence alignments when manually aligning the documents, we were able to determine if the manual alignments and automatic alignments had a matching alignment if two alignments had matching sentence indices for both the original and simplified sentence. Using these indices, we determined how many alignments the manual and automatic alignments shared for each document, by checking each alignment against every alignment in the other set. One-to-one sentence alignments were counted as a single correct alignment, while for alignments with one-to-many sentence alignments, they were weighted as fractional alignments based on how many sentences are in the sentence alignment, which is shown below.

alignment.py

```
for prediction in current_predicted_alignments:
        for actual in current_actual_alignments:
            if(len(actual) == 1):
                continue
            elif(len(actual) == 2):
                actual = tuple(actual)
                if(prediction == actual):
                    total_correct = total_correct + 1
            else:
                weight = 1/(len(actual)-1)
                for i in range(1, len(actual)):
                    current_actual = (actual[0], actual[i])
                    if(prediction == current_actual):
                        total_correct = total_correct + weight
```

After summing up the total amount of shared alignments in both sets, the precision is calculated by dividing the total shared amount by the number of alignments in the automatic alignments, while the recall is calculated by dividing the total shared amount by the number of alignments in the manual alignments; the F1 score is then calculated with the formula of two multiplied by both the precision and the recall as the numerator, while the denominator is set as the precision plus the recall, and these results are shown below.

Table 2

Precision	Recall	F1 Score
0.7276489482343556	0.23256051031019673	0.34178718532249014

Based off the results seen in Table 2, it can be observed that precision is noticeably better than both recall and the F1 Score; while it would have been ideal for all three to have been high scoring, because these automatic alignments for the most part serve to be used as training sets for fine tuning models, it is encouraging to see that precision is the highest number, for this indicates that most of the alignments being used as training input for models also match actual manual alignments, and thus for the most part will be quite representative of what manual simplification would look like and not lead to models learning unhelpful artifacts instead.

Training and Testing Models, Calculating Scores

With the data sufficiently processed and analyzed, while also producing the automatic alignments crucial to training, we then proceeded to training and testing a variety of different models, to see which we could improve on with the best yield in improvements. Due to the ACESSS model not being easily trained with custom input and fine-tuned, we just kept it as a baseline model for comparison purposes; we instead fine tuned the T5 model (Raffel et al., 2020), an encoder-decoder model that converts all NLP problems into a text-to-text format, including text simplification. We used two different T5 models, one which had been pretrained on WikiLarge (Zhang et al., 2017), while the other was pretrained on Newsela (Xu et al., 2015), a corpus of news articles.

Our data is then split into a fifty thirty twenty split: fifty percent of our texts were randomly chosen for training, twenty percent for validation, and thirty percent for testing. For our training set, the sentence pairs are generated from our method of creating automatic alignments, which is then packaged into a single csv file to be used as input for the two models. The validation set, while held out, was ultimately not used for our experiments, and the testing set is composed of our manual sentence alignments, to test how well the text is being simplified, and these sentence pairs are also compiled into a single csv file as input for the two models for testing.

With the datasets ready, the training set is inputted into both the pretrained WikiLarge T5 model and the pretrained Newsela T5 model, both of which are trained for three epochs. After fine-tuning these models, the test set is then used as input, to generate the simplified text that is then used to calculate the metrics of the model; the test set is also used as input into the two original pretrained models that have not been trained and fine-tuned, for the purpose of comparison. The scores of the results are then calculated through three metrics: Sentence-BERT (Reimers et al., 2019), BLEU (Papineni et al. 2002), a number between zero and one that measures the similarity of the machine-translated text to a set of high quality reference translations, as well as SARI (Xu et al. 2016), a lexical simplicity metric that measures “how good” are the words added, deleted and kept by a simplification model. The results of both models that have been trained on the college admission data, the two models again but untrained on the college admission data, as well as the results from the ACCESS model are measured and listed below, where these scores are calculated from the total average of all of the generated sentences compared to the gold label counterpart of each respective generated sentence.

Table 3

	ACCESS	Untrained Newsela T5	Trained Newsela T5	Untrained WikiLarge T5	Trained WikiLarge T5
Sentence-BERT	0.714859085	0.686019215	0.749319886	0.683655860	0.746820171
BLEU	0.104669810	0.131990040	0.212901339	0.171374629	0.210616656
SARI	0.270670255	0.226008842	0.210698337	0.169439821	0.193493668

Results

From the score calculations of Table 3, we can observe a few trends. All three metrics are based on a scale of zero to one, so higher scores are desired. All three models that were not fine-tuned, the ACCESS model, untrained Newsela T5, and untrained WikiLarge T5, in general yielded lower scores compared to their trained counterparts. When comparing the untrained models themselves, there is no clear superior model, for all three have the highest score for each metric when compared to the other two models, suggesting that training on either Wikipedia articles or news articles as a starting point is acceptable, since there is no significant winner between the models. After the models are fine-tuned through training on the college admission texts, we can see there is a noticeable improvement in both the Newsela T5 model and the WikiLarge T5 model as a whole. With increases in scores for virtually all metrics for the two trained models, this shows that improvements in the performance of the models have been made in terms of the generation of text simplifications of college admission texts.

Discussion

From our study, there are a number of conclusions we can derive from our results and observations. The most significant conclusion we can make is that by training and fine-tuning on already existing models, this allowed us to improve the performance of text simplification within the domain of college admissions, which is a field with more complex texts. This suggests that when utilizing text simplification for more specialized domains, it is better to pursue a specific model specifically for that domain, rather than an all-encompassing generalized model; otherwise, it is inevitable that performance will most likely suffer for the more specialized tasks with more complex sentences and difficult jargon. By focusing solely on a specific domain, it allows simplified texts to be produced that are more representative of ones that are manually simplified for that specific domain, where texts of this nature can have a specific model dedicated to it for certain uses when needed, because these texts serve a specific purpose and aren’t seen in everyday circumstances.

Another conclusion one can make is that the starting pretrained model does not greatly matter; this can be observed through our results from Table 3, where even though the two models started off being pretrained on different texts from different domains, in the end after fine-tuning and training on the college admission texts, the two models produced essentially the same performance metrics, suggesting that the input for training is more significant.

Finally, we believe that with more resources invested into application within this domain, results can be further improved. Even with being limited to admission texts from a single admission cycle, because of how the texts came from varying locations across different types of

institutions, both public and private, this variety allowed our models to produce substantial improvements in performance, even when tested with admission texts that varied greatly both in terms of length and context. As a result, we are positive that with the investment of more time and resources, there is certainly room for further improvement for the application of text simplification within this domain, whether it be through the collection of more texts for more training and texting, or improving the process of producing sentence alignments between original and simplified texts, both automatically and manually.

Acknowledgements

I would like to thank Professor Li, not only for being my supervisor for my project this semester, but also for giving me precious guidance and advice throughout the project; without her expertise, I would not have been able to achieve what I conducted throughout my research project. I would also like to thank Dr. Zach Taylor, for his original and simplified college admissions texts as the original source of data, his help in the manual alignment of the sentences in the text pairs, as well as knowledge on the domain of college admissions as a whole, where his insight was also greatly valuable in helping me more deeply understand the domain I was conducting research on.

Literature Cited

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. In *Journal of Machine Learning Research 21*, pages 1-67.

Louis Martin, Benoît Sagot, Éric de la Clergerie, Antoine Bordes. 2019. Controllable Sentence Simplification.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, Philadelphia, pages 311-318.

Nils Reimers, Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.

Wei Xu, Chris Callison-Burch, Courtney Napoles. 2015. Problems in Current Text Simplification Research: New Data Can Help. In *Transactions of the Association for Computational Linguistics 2015*.

Wei Xu, Courtney Napoles, Ellie Pavlick, Quanze Chen, Chris Callison-Burch. 2016. Optimizing Statistical Machine Translation for Text Simplification. In *Transactions of the Association for Computational Linguistics 2016*, volume 4, pages 401-415.

Xingxing Zhang, Mirella Lapata. 2017. Sentence Simplification with Deep Reinforcement Learning. In *EMNLP 2017*.

Appendix

All code that was discussed, including the code that was sampled within the paper, code and methods too long to be included directly within the paper, as well as the trained models, can be found at this GitHub repository: https://github.com/maximuschu/text-simplification-college-admission

Reflection

From doing this project, I have been able learn several different things through my personal experiences throughout the semester. The first thing I learned personally was how different conducting research is compared to that of a structured course, where everything was planned out by someone else. Here, I was able to have a large amount of freedom in planning out alongside with Professor Li what we should achieve and by when, where the only true deadlines I had were the weekly meetings and the deadline at the end of the semester; even then, the weekly deadlines were still quite flexible, so overall I really enjoyed the freedom of planning out everything for myself.

I also learned that conducting research can be quite rewarding, especially at the end when we achieved the goals that we had planned out; at first, I was somewhat intimidated by the prospect of having to do a semester long research project by myself, for I had not done anything similar before. This combined with the fact that I had to plan many things out with little to no similar prior studies before in this field made the task appear quite daunting to me, but as a result this in turn allowed every single goal accomplished to be that much more rewarding, knowing that I had achieved what I had set out to accomplish.

Finally, I learned both from through my experiences as well as all the conversations with Professor Li and Dr. Zach Taylor that there is so much more that can be done and accomplished; before doing this project, I had never thought of how one study can lead to another, but from my interactions with them, I have learned that there is always more that can be researched, more to be studied. There is always more to be learned and accomplished, and that is the most meaningful lesson that I learned from finishing this semester research project.